Artemii Kyrianov, Heorhii Loutskii,
Oleksandr Chaikovskyi

TRIANGULATION OF MOBILE PHONE LOCATION BY BASE STATIONS

The article discusses the method of triangulation in radar location in the cellular network. The issue of the task of changing the structure in which the triangulation is presented is considered, which may arise, for example, when building a greedy or optimal triangulation. Algorithms for their construction operate only with edges and nodes, and therefore they are forced to use data structures of the type "Nodes with neighbors" or "Nodes and edges". On the other hand, the purpose of triangulation may be to model a surface, which requires a data structure such as Nodes and Triangles. That is why the task of transition from one data structure to another arises. Data comparisons were made, the main gains, losses and prospects were identified.

Key words: fault tolerance, excess code, Latin square

Fig.: 11. Tabl.: 1. Bibl.: 4.

Structures to represent triangulation. As practice shows, the choice of a structure for representing a triangulation has a significant impact on the theoretical complexity of the algorithms, as well as on the speed of a specific implementation. In addition, the choice of structure may depend on the purpose of further use of triangulation. In triangulation, 3 main types of objects can be distinguished: nodes (points, vertices), edges (segments) and triangles. In the work of many existing algorithms for constructing the Delaunay triangulation and algorithms for its analysis, the following operations often occur with triangulation objects:

1. Triangle → nodes: get for the given triangle the coordinates of the nodes forming it.

2. Triangle → edges: getting a list for a given triangle

the edges that form it.

3. Triangle → Triangles: Retrieve a list of neighboring triangles for a given triangle.

4. Edge → nodes: get for this edge the coordinates of the nodes forming it.

5. Edge → Triangles: Retrieve a list of adjacent triangles for a given edge.

6. Node → edges: get a list of adjacent edges for a given node.

7. Node → Triangles: Get a list of adjacent triangles for a given node.

In some algorithms, some of these operations may not be used. In other algorithms, edge operations may occur infrequently, so the edges can be represented indirectly as one of the sides of some triangle. Consider the most common structures.

Data structure "Nodes with neighbors". In the structure "Nodes with neighbors" for each triangulation node, its coordinates on the plane and a list of numbers (or pointers) of adjacent (neighbors with which there are common edges) nodes are stored (Fig. 1):

Node = record
X: number; ← X coordinate
Y: number; ← Y-coordinate
Count: integer; ← number of adjacent nodes
Nodes: array [1..Count] of NodeNumber; ← list of adjacent nodes
end;

The order of adjacent nodes in the list is usually not important, but in some tasks it is sometimes required that this list of nodes be sorted clockwise or counterclockwise (Figure 1).

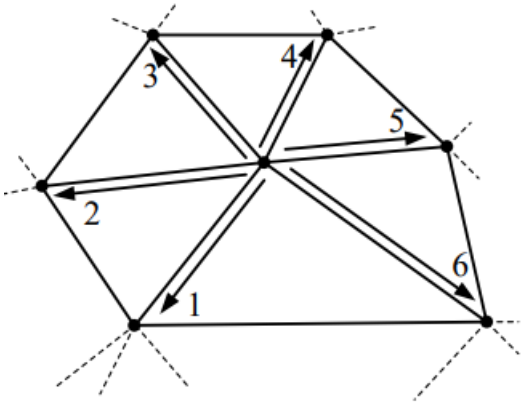

Fig. 1. Connections of nodes of the structure "Nodes with neighbors"

Essentially, the list of neighbors implicitly defines the triangulation edges. Triangles are not represented at all, which is usually a significant obstacle to the further use of triangulation. In addition, the disadvantage is the variable size of the node structure, which often leads to wasteful memory consumption when building a triangulation. The average number of adjacent nodes in a Delaunay triangulation is 6 (this is proved by induction or from Euler's planar graph theorem), so with an 8-byte coordinate representation, 4-byte integers, and 4-byte pointers, the total amount of memory occupied by this triangulation structure is 44 * N bytes.

Nodes and Edges Data Structure. In the "Nodes and edges" structure, nodes and edges are explicitly specified. There are no triangles in the structure. For each edge, pointers to two end nodes are stored. For triangles, pointers to the three edges forming the triangle are stored (Figure 2):

Node = record
X: number; ← X coordinate
Y: number; ← Y-coordinate
end;
Edge = record
Nodes: array [1..2] of NodeNumber; ← list of end nodes
end;

This structure is often used in cases where it is required to explicitly represent the edges of a triangulation, but there is no need to work with triangles. In particular, this structure is best suited for constructing greedy and optimal triangulations. This structure consumes quite a bit of memory: with an 8-byte representation of coordinates and 4-byte pointers, there are about 40 * N bytes

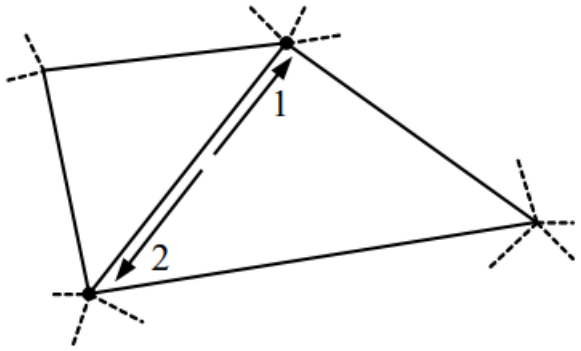

Fig. 2. Connections of edges of the structure "Knots and edges"

Data Structure "Double Edges". In the "Double Edges" structure, the basis of triangulation is a list of directed edges. In this case, each edge enters the triangulation structure twice, but directed in opposite directions:

Node = record
X: number; ← X coordinate
Y: number; ← Y-coordinate
end;
Edge = record
Node: Node_number; ← end node of the rib

Next: Edge_number; ← next clockwise in the triangle on the right

Twin: Edge_number; ← twin edge pointing the other way

Triangle: Triangle_number; ← pointer to right triangle

end;

Triangle = record ← There are no required fields in the record

end;

The following pointers are stored for each edge (Figure 3):

1) on the end node of the rib;

2) to the next clockwise edge in the triangle to the right of this edge;

3) to the "twin edge", connecting the same triangulation nodes as the given one, but directed in the opposite direction;

4) to the triangle located to the right of the edge.

The last pointer is not needed to build a triangulation, and therefore its presence should be determined depending on the purpose of the further application of triangulation.

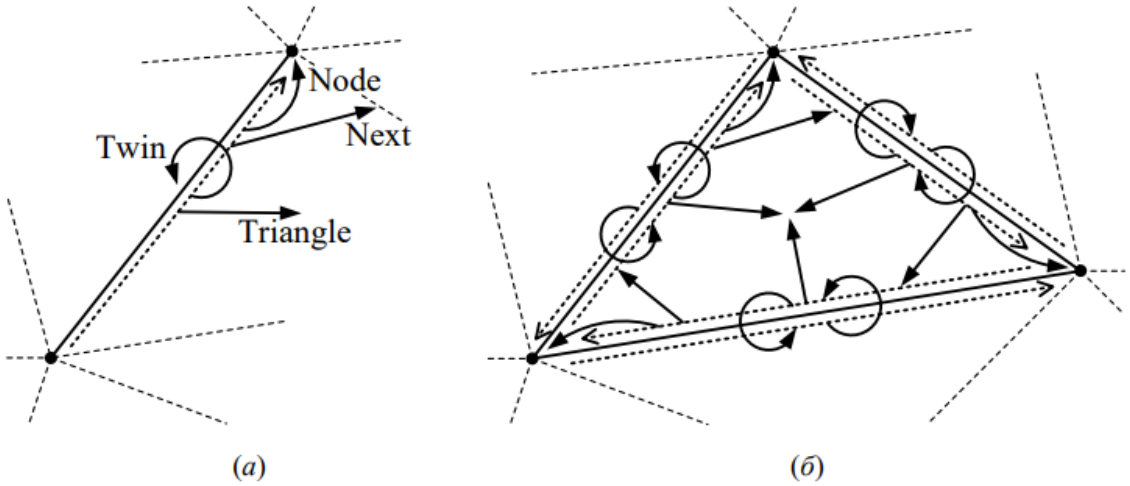

Fig. 3. Edge connections (a) and implicit definition of triangles (b) in the “Double edges” structure

The disadvantages of this structure are the representation of triangles in an implicit form, as well as a large memory consumption, which, with an 8-byte representation of coordinates and 4-byte pointers, is at least 64 * N bytes (not taking into account the memory consumption for representing additional data in triangles).

Nodes and Triangles Data Structure. In the "Nodes and Triangles" structure, for each triangle, three pointers to the nodes forming it and three pointers to adjacent

triangles are stored (Figure 4):

Node = record
X: number; ← X coordinate
Y: number; ← Y-coordinate
end;
Triangle = record
Nodes: array [1..3] of NodeNumber; ← generating nodes
Triangles: array [1..3] of TriangleNumber; ← neighboring triangles
end;

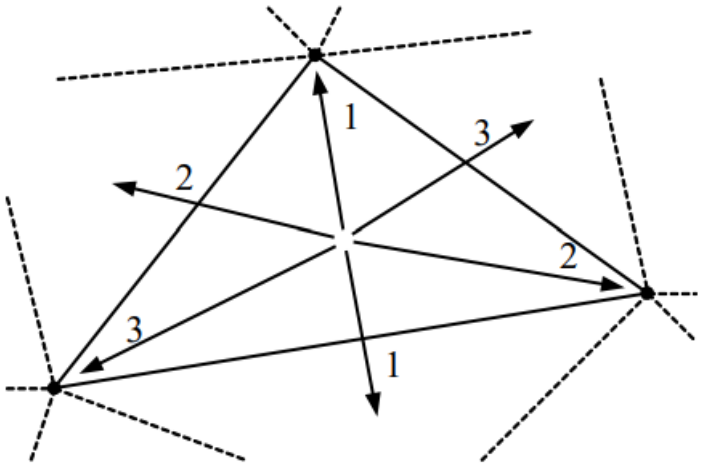

Fig. 4. Connections of triangles of the "Knots and Triangles" structure

Points and neighboring triangles are numbered in clockwise order, while opposite the point with the number i{1, 2, 3} there is an edge corresponding to the neighboring triangle with the same number (Figure 4). The edges in this triangulation are not explicitly stored. If necessary, they are usually represented as a pointer to a triangle and the number of an edge inside it. With an 8-byte coordinate representation and 4-byte pointers, this triangulation structure requires approximately 64 * N bytes.∈ Despite the fact that this structure is inferior to Nodes with Neighbors, it is most often used in practice due to its relative simplicity and ease of programming algorithms based on it.

Nodes, Edges and Triangles Data Structure. In the "Nodes, edges and triangles" structure, all triangulation objects are explicitly specified: nodes, edges and triangles. For each edge, pointers to two end nodes and two neighboring triangles are stored. For triangles, pointers to the three edges forming the triangle are stored (Figure 5):

Node = record

X: number; ← X coordinate
Y: number; ← Y-coordinate
end;
Edge = record
Nodes: array [1..2] of NodeNumber; ← list of end nodes
Triangles: array [1..2] of TriangleNumber; ← neighboring triangles
end;
Triangle = record
Ribs: array [1..3] of RibNumber; ← generating edges
end;

Points and neighboring triangles are numbered in clockwise order, while opposite the point with the number i{1, 2, 3} there is an edge corresponding to the neighboring triangle with the same number (Figure 5). This structure is often used in practice, especially in problems where it is required to explicitly represent triangulation edges.∈

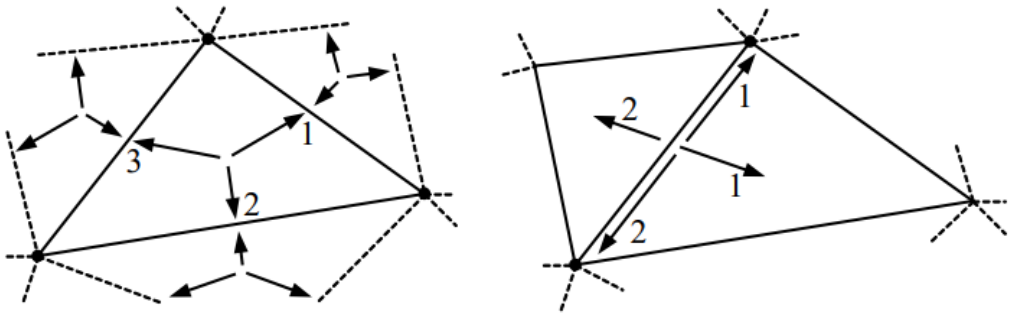

Fig. 5. Connections of triangles (left) and edges (right)
of the structure "Nodes, edges and triangles"

The disadvantage of this structure is a large memory consumption, amounting to approximately 88 * N bytes with an 8-byte representation of coordinates and 4-byte pointers.

Nodes, simple edges and triangles data structure. In the "Knots, simple edges and triangles" structure, all triangulation objects are explicitly specified: nodes, edges and triangles. For each edge, pointers to two end nodes and two neighboring triangles are stored. There is no special information for edges. For triangles, pointers are stored to the three nodes and three edges forming the triangle, as well as pointers to three adjacent triangles (Figure 6):

Node = record

X: number; ← X coordinate

Y: number; ← Y-coordinate

end;

Edge = record ← There are no required fields in the record

end;

Triangle = record

Nodes: array [1..3] of NodeNumber; ← generating nodes

Triangles: array [1..3] of TriangleNumber; ← neighboring triangles

Ribs: array [1..3] of RibNumber; ← generating edges

end;

This structure is often used in practice, especially in problems where it is required to explicitly represent triangulation edges.

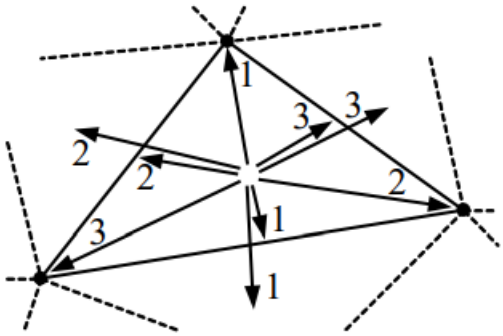

Fig. 6. Triangle relationships in the Nodes, Simple Edges and Triangles data structure

The disadvantage of this structure is a relatively large memory consumption, amounting to about 88 * N bytes with an 8-byte representation of coordinates and 4-byte pointers. To conclude this section, Table 1 summarizes the characteristics of the given data structures, including the memory cost and the degree of representation of the various elements of the triangulation ("–" - the element is absent, "+" - present, "±" - present, but no links to other elements triangulation). In general, it can be noted that the Nodes with Neighbors structure is less convenient than the others, since it does not explicitly represent edges and triangles. Among the rest, the Nodes and Triangles structure is quite convenient in programming. However, some triangulation algorithms require the explicit representation of edges, so the Nodes, Edges, and Triangles structure can be recommended there.

Table 1. **Main characteristics of data structures**

Data structure name	Memory	Knots	ribs	triangles
"Nodes with Neighbors"	44*N	+	-	-
"Knots and Edges"	40*N	±	+	-
"Double Ribs"	64*N	±	+	±
"Knots and Triangles"	64*N	±	-	+
"Knots, Edges and Triangles"	88*N	±	+	+
"Knots, simple edges and triangles"	88*N	±	±	+

Converting Data Structures. The problem of changing the structure in which the triangulation is represented can arise, for example, when constructing a greedy or optimal triangulation. The algorithms for their construction operate only with edges and nodes, and therefore they are forced to use data structures like "Nodes with neighbors" or "Nodes and edges". On the other hand, the purpose of building a triangulation may be surface modeling, which requires a data structure, such as "Knots and Triangles". That is why the problem of transition from one data structure to another arises. First, let's consider a fairly simple algorithm for moving from the "Nodes and Edges" structure to the "Nodes with Neighbors" structure. The main goal of this algorithm is to calculate edges adjacent to nodes. Algorithm for converting the data structure "Nodes and edges" into the structure "Nodes with neighbors" Data structures. The initial structures are represented by arrays of Nodes and Ribs. As a result, we should get an array of NewNodes. The algorithm will require a temporary array R of length N to count the number of edges adjacent to the corresponding nodes.

Step 1. Calculate the number of adjacent edges R[i] included in each i-th triangulation node. To do this, we first assign i: R[i]:=0. Then, in a loop over i, we look through all the edges and for each edge Ribs[i] connecting nodes with numbers a=Ribs[i].Nodes[1] and b=Ribs[i].Nodes[2], we increase the counters of edges in nodes : R[a]++, R[b]++.∀

Step 2. Allocate memory for each node of the Nodes with Neighbors structure, using R[i] as the number of nodes in the node's Nodes array. As a result, we get new entries NewNodes[i]. In NewNodes[i] we copy the fields with X,Y coordinates from the corresponding fields of Nodes[i] of the originaldata structures "Nodes and edges". Set other fields to zero for now: NewNodes[i].Count:=0, j: NewNodes[i].Nodes[j]:=0.∀

Step 3. In the loop over i, we look through all the edges and for each edge R[i] connecting nodes with numbers a and b, add to the nodes links to the node adjacent through this edge and increase the counters of adjacent nodes in new nodes:

NewNodes[a].Nodes[NewNodes[a].Count]:=b; NewNodes[a].Count++;

NewNodes[b].Nodes[NewNodes[b].Count]:=b; NewNodes[b].Count++.

End of the algorithm. The complexity of the described algorithm is linear in the number of triangulation nodes. The next algorithm for moving from the "Knots and Edges" structure to the "Knots and Triangles" structure is more complicated. It requires the creation of interconnected triangle structures. The first 3 steps of this algorithm almost coincide with the previous algorithm: in it, a special temporary data structure is created for each node. Algorithm for converting the "Nodes and Edges" data structure into the "Nodes and Triangles" structure. Data structures. The initial structures are represented by arrays of Nodes and Ribs. During the operation of the algorithm, a temporary array R of length N is required to count the number of edges adjacent to the nodes. In addition, we will need a temporary modified data structure "Nodes with neighbors" (extended with a list of adjacent edges and triangles, but without coordinates, since we can get them from the original Nodes array), which will be represented in the TmpNodes array. This temporary structure should have

the following view:

TempNode = record

Count: integer; ← number of adjacent nodes

Nodes: array [1..Count] of NodeNumber; ← list of adjacent nodes

Ribs: array [1..Count] of RibNumber; ← list of adjacent edges

Trns: array [1..Count] of TriangleNumber; ← adjacent triangles

end;

In this structure, the edge TmpNodes[i].Ribs[j] must connect to the node TmpNodes[i].Nodes[j], and the triangle TmpNodes[i].Trns[j] must lie between the edges adjacent to the node, defined by the edges TmpNodes[i].Ribs[j] and TmpNodes[i].Ribs[j mod TmpNodes[i].Count+1]. The algorithm will need another temporary structure that introduces additional fields for each triangulation edge and which will be provided for the edges in the TmpRibs array:

Temporary Edge = record

IndexInNode: array [1..2] of integer; ← edge numbers in the Ribs lists of edge nodes

Trns: array [1..2] of TriangleNumber; ← adjacent triangles

end;

In this data structure, the triangle TmpRibs[i].Trns[1] must lie on the right side of the vector formed by the nodes Ribs[i].Nodes[1] and Ribs[i].Nodes[2] (Figure 7). The temporary array IndexInNode is designed to quickly determine the next and previous edge in the triangle containing this edge.

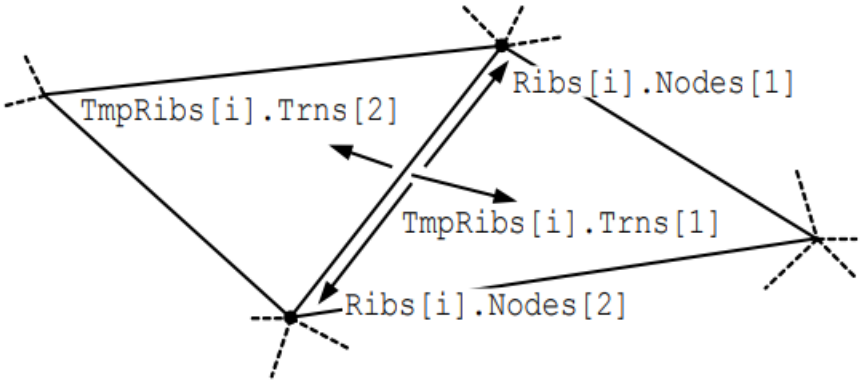

Fig. 7. Temporary connections of edges in the structure transformation algorithm data "Knots and edges" into the structure "Knots and triangles"

As a result of the algorithm, we should get an array of NewNodes.

Step 1. Calculate the number of adjacent edges R[i] included in each i-th triangulation node. To do this, we first assign i: R[i]:=0. Then, in a loop over i, we look through all the edges and for each edge Ribs[i] connecting nodes with numbers a=Ribs[i].Nodes[1] and b=Ribs[i].Nodes[2], we increase the counters of edges in nodes : R[a]++, R[b]++.∀

Step 2. Create temporary TmpNodes[i] entries for each node, using R[i] as the lengths of the Nodes, Ribs, and Trns arrays. Other fields are filled with zeros and empty references for now: TmpNodes[i].Count:=0, j: TmpNodes[i].Nodes[j]:=0, j: TmpNodes[i].Ribs[j]:=0, j : TmpNodes[i].Trns[j]:=0.∀∀∀

Step 3. We look through all the edges and for each edge R connecting nodes with numbers a and b, add to the nodes links to R and the node adjacent through this edge R, and increase the counters of adjacent nodes in the new nodes:

TmpNodes[a].Nodes[TmpNodes[a].Count]:=b;

TmpNodes[a].Ribs[TmpNodes[a].Count]:=R; TmpNodes[a].Count++;

TmpNodes[b].Nodes[TmpNodes[b].Count]:=b;

TmpNodes[b].Ribs[TmpNodes[b].Count]:=R; TmpNodes[b].Count++.

Step 4. At each node in the temporary structure TmpNodes, sort the adjacent nodes and edges clockwise (simultaneously in the TmpNodes[i].Nodes and TmpNodes[i].Ribs arrays). For all triangulation edges array

TmpRibs[i].Trns are filled with empty links (zero numbers of triangles): i,j: TmpRibs[i].Trns[j]:=0.∀

Step 5. In the loop over i, for each node, we make a nested loop over j over all adjacent edges. For each edge, determine the number k of the node in the edge and set TmpRibs[TmpNodes[i].Ribs[j]].IndexInNode[k]:=j.

Step 6. In the i loop for each edge, we do a nested j loop through two triangles adjacent to the edge and try to create a new triangle from TmpRibs[i].Trns[j] if this triangle has not yet been created (if TmpRibs[i] .Trns[j]=0). This triangle will connect the end nodes of the current edge (Ribs[j].Nodes[j], Ribs[j].Nodes[3-j]) and another node, which can be determined using the list of adjacent nodes in node Ribs[j] .Nodes[j] using TmpRibs[i].IndexInNode[j]. In addition, you need to define 3 edges that form a triangle and expose links from these edges to a new triangle.

Step 7. We establish mutual links of triangles to each other. To do this, we make a cycle along i along each internal edge of the triangulation (along all i-th edges with j: TmpRibs[i].Trns[j]≠0) and for it we set mutual links of adjacent triangles TmpRibs[i] .Trns[0] and TmpRibs[i].Trns[1].∀

End of the algorithm. Provided that at step 4 sorting with linear complexity is used (for example, digital), then the complexity of this algorithm is linear with respect to the total number of nodes in the triangulation. In general, even if you use a non-linear sort, the complexity of the algorithm on average will also be linear - O (N). This follows from the fact that the average number of edges adjacent to a node in a triangulation is a constant independent of N, and hence sorting will run O(1) time on average. Conclusion: The above algorithm can be easily modified to obtain other data structures in which triangles are explicitly represented.

Triangulation of mobile phone location by base stations. There is a common misconception that the geographic location of any GSM phone can be determined with sufficient accuracy by triangulating over three base stations. This is usually described as follows: for example, if it is possible to determine the distance from the base station to the phone by standard means, then by the distances from three base stations you can get the exact coordinates of the device, and by the distance from two base stations - two points, one of which will be located desired phone. As a rule, popular rumor endows criminal elements

or law enforcement agencies with the ability to use such technology to find the people they need. Part of this statement is true. Standard means can sometimes determine the distance from the phone to one base station. This, perhaps, explains the tenacity of the belief that triangulation is possible. Actually it is not. Before starting a detailed analysis of the triangulation case, it is worth making a significant reservation. It is necessary to draw a clear line between the delusion that the base stations of any GSM network always triangulate the location of the specified phone (variant - all phones in the coverage area) and the possibility of detecting the location of the phone by other means within a single network.

Location Based Services. To provide location-based services (LBS), there are many ways based on the availability of additional software and hardware at all base stations of a particular network, and sometimes also in the subscriber's SIM card / phone. An example of such services: show a subscriber a map of the city and his place on it, tell the address of the nearest restaurant or store, tell the location of another subscriber, get directions to a given point. For a number of applications, it is sufficient to approximately know one base station, in the coverage area of which the subscriber is located. This can be done on any GSM network. Result: a circle with a radius of up to 32 km with a center at the installation site of some base station. In urban areas, the radius can be reduced, since the coverage areas of base stations are usually small. It is worth mentioning that information about the “current” base station is updated with every call / SMS or about once an hour, therefore, to improve the accuracy of detection, an SMS is sent to the subscriber immediately before the “measurement” or the subscriber himself is encouraged to send an SMS with a request like “ where am I/where is the nearest restaurant/hotel/subway/...”. This result can be improved with a technique called "time of arrival". All base stations in the network need to be upgraded. Result: a circle with a radius of 100-500 meters centered on the base station installation site. The use of even more advanced methods (their description can be found on the web using the keywords "angle of arrival", "uplink time difference of arrival", "GPS", "assisted GPS") allows you to further reduce the radius of the circle or move its center to real location of the subscriber. The presence of any complex subscriber location detection system in the operator's network is very easy to determine - the operator will sell the relevant services, no one will invest in the creation of the necessary infrastructure just like that. Often a cursory glance at a service's promotional material is sufficient to determine the type of technology an operator is using, simply based on detection accuracy data.

On this excursion into the technology of determining the location of the subscriber can be considered complete and return to the original topic:

1. Is it possible to triangulate the location of a phone in the GSM network by three (four, ...) base stations?
2. Who can carry out this triangulation: the subscriber, the operator, or both parties?

Triangulation. Let's start with an analogy. Consider the following statement: "using the ping utility, you can determine the transit time of TCP packets from one computer to another, and therefore estimate the distance between them. Then, by the distances from three computers, knowing their coordinates, you can get the coordinates of the desired computer." If we take four computers, we will connect them with network cables to each other directly, without using intermediate networks, and we will lay the wires strictly in a straight line. Will we be able to determine the coordinates of the central computer in this case, knowing the coordinates of the peripheral ones and using only ping? We can. Does this mean that this method can always be used? Certainly not. Firstly, wires rarely connect two computers directly and strictly in a straight line, and secondly, we, as a rule, do not know the exact coordinates of the “reference” computers. It will be easy to continue this list. Now back to the original statement. Is it possible to determine the distance from the base station to the phone using standard GSM network tools? The short and self-explanatory answer is "you can". Let's ask additional questions:

1. Who is doing the measurements - the base station or the phone?
2. Is such a measurement always possible?
3. Will the shortest distance between them be measured?
4. How accurate will the measurement be?

To understand who can make such a measurement, you need to figure out what the phone and the base station know about each other. It is worth dividing the description into two cases: the phone is in standby mode and the phone is in active mode (they are talking on it, receiving SMS, ...).

Phone on standby

Base stations regularly broadcast signals over the air so that phones can know if they are in coverage area. Phones, on the other hand, most of the time do not transmit anything, only receive, in order to save battery power. It is easy to check this in practice by placing the phone next to computer audio speakers and observing the

disturbances induced by the phone, or by buying a simple GSM signal detector key fob. It follows from this that it is impossible to determine the location of a conventional GSM telephone in a conventional GSM network at an arbitrary moment of time simply because the telephone is silent and does not "tell" anyone where it is and where it is being carried. Periodically, the phone notifies the network of where it is in order to facilitate the delivery of incoming calls. This happens:

1. when registering on the network;
2. when a subscriber moves from the coverage area of one group of base stations to another (a group may include several hundred base stations, there may be only a dozen such groups in a city of a million people);
3. periodically - once every half an hour or an hour, depending on the network settings.

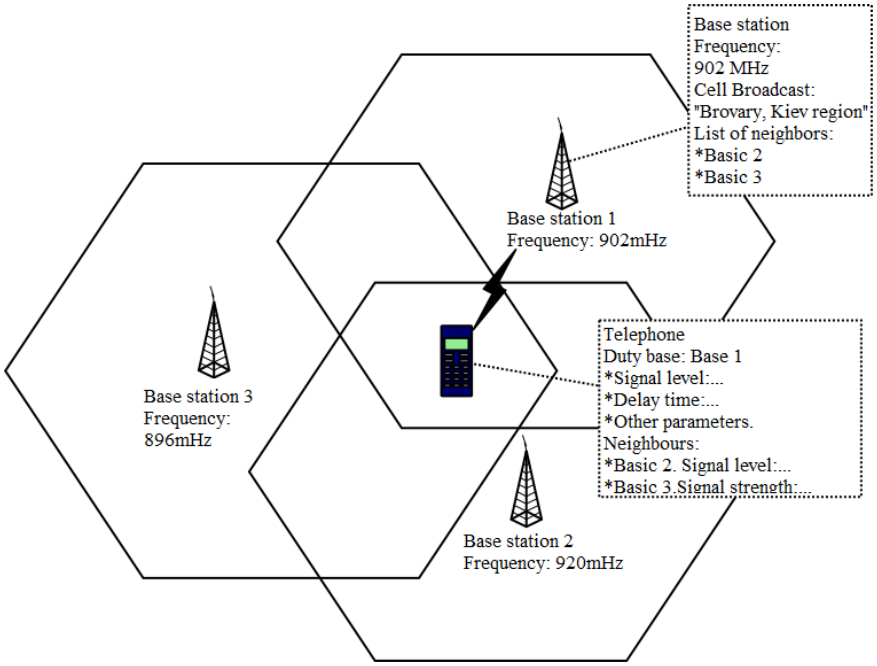

Fig. 8. Base stations

In this case, the phone tells the network only about which base station it “hears” best, without any details like signal strength. Base stations do not keep track of which phones are in their coverage area, this is pointless and technically unfeasible. Accordingly, most of the time, the mobile network has only a very rough idea of where the phone currently lives. Firstly, it is not clear from which base station to measure - since the last update of the location information, the phone could have been carried away for a considerable distance. Secondly, it is unclear what and how to

measure. The base station is not a radar, and if the phone is “silent”, then it does not exist for it. So, in standby mode, a standard phone in a standard GSM network is completely invisible to the mobile network and cannot be "triangulated" by it. The phone itself is in a more advantageous position. The fact is that each base station broadcasts information about its “neighbors”, indicating the frequencies on which the nearest base stations of the same network operate. The phone in standby mode constantly measures the signal level (but not attenuation) from each of the “neighboring” base stations and, if necessary, selects the one from which the signal is “better heard” as the standby base station. If the phone has some information about where (at what coordinates) the base stations are located, then it can try to calculate the zone in which the hypothetical coverage areas of all "neighboring" base stations intersect. Somewhere within this area there will be a telephone. The more accurately the phone knows (or estimates) the boundaries of coverage areas, the more accurately this method will work. According to available information, that's how the Google Latitude app works. If there is no data on the location of the base stations, then the phone will not have any opportunity to “triangulate” its position.

Phone in active mode

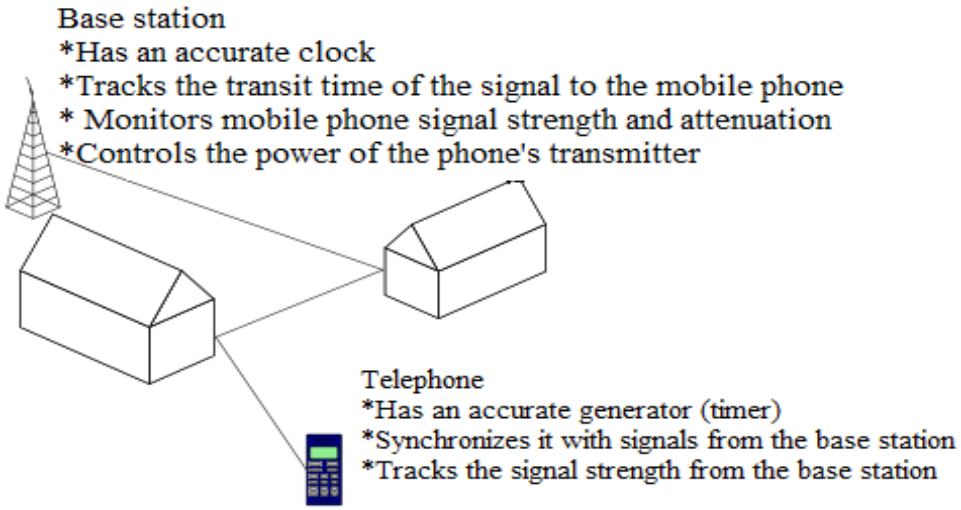

Fig. 9. Active Mode Schematic

In active mode, the phone sends signals to a single base station and receives response signals from it. Everyone is familiar with the fact that GSM networks can operate at frequencies of 900, 1800 and (rarely) 1900 mHz. In fact, we are talking about frequency ranges: 890-960, 1710-1880 and 1850-1990 mHz, respectively. Each base station

broadcasts only on one specific frequency from this range. Neighboring base stations, regardless of which operator they belong to, are always configured to create minimal interference with each other. In particular, neighboring base stations will never operate on the same frequency. The base station in the process of servicing the conversation performs control and regulatory functions. She is engaged in the calculation of the values of the so-called time shift (timing advance) and transmits them to the phone. The phone uses them to adjust its timer so that its and the base station's “clocks” are synchronized and the signals sent by the phone reach the base station within the “broadcast window” assigned to the phone. In order to correctly calculate the time shift, the base station measures the time it takes the signal to travel from itself to the phone, but it absolutely does not matter how many times the signal bounces off buildings and other obstacles along the way. The base station also evaluates the signal strength and attenuation of the phone and makes recommendations to the phone on the required transmission power. The phone during the conversation also has information about the time shift, the signal level from the base station and the power of its transmitter. It turns out that both the base station and the telephone can, in principle, somehow estimate the distance to each other along the path of the radio signal, but they cannot take into account all possible refractions and deviations. The accuracy of distance measurement by time shift is about 500 m.

Conclusions If we are not talking about a specific operator, but talking about GSM as a technology in general, then we can argue that:

1. The standard capabilities of the GSM network allow the construction of systems for determining the location of a subscriber based on measuring the parameters of the radio signal, but there is no GSM Phase 2+ standard for such systems / technologies.

2. If such a technology is not implemented in the operator's network, then by means of the network itself it is possible to determine only the last known location of the subscriber with an accuracy of the base station that serviced his call or registration in the network. You can send an SMS to your phone or call and update this information.

3. On the basis of a standard GSM telephone, it is possible to build a system for determining its location, but only if data on the coordinates of the installation of base stations are available.

4. The method of determining the location of the phone by means of the network using triangulation (in the form in which it is presented at the beginning of the article) is nothing more than a common fiction.

References

- GSM Frequency Ranges
- Comparison of the accuracy of the work of different methods of localization of subscribers
- GSM Standard 03.22 "Functions related to Mobile Station (MS) in idle mode and group receive mode»
- GSM Standard 03.30 "Radio Network Planning Aspects»

AUTHORS

Kyrianov Artemii – PhD student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Heorhii Loutskii – Professor, Doctor of Technical Sciences. Head of the specialized academic council, National Technical University of Ukraine "Ihor Sikorskyi Kyiv Polytechnic Institute".

Oleksandr Chaikovskyi – PhD student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.