Mesh Network Topology Triangulation Algorithm Patent Extension

Bartlomiej Slupik

October 2, 2024

Contents

1 Introduction

In the following paper I present various techniques, improvements and ideas which can enhance the capabilities of the triangulation algorithm described in the previous patent application. The complex nature of the algorithm gives room for a large number of optimisations, although it is difficult to test and compare all the options. Therefore, it is possible, that in certain scenarios the process may work in an unexpected way or lose accuracy.

Brief description of the algorithm

The goal is to get approximate (x, y, z) positions of nodes in a network, assuming it is possible to measure the distance between the devices. Only the nodes, which are in direct reach of each other, can conduct this calculation. In order to compute the distance between nodes positioned further away, the following procedure is applied:

1. Three nodes G_1, G_2, G_3 are chosen, such that:

   They are all neighboring each other.

   They are all neighboring the origin O.

   They are all neighboring the target T.

   Such configuration is called a *gate*.

2. All edge lengths of a three-dimensional solid $OG_1G_2G_3T$, but one - $OT = x$ are known.

3. As all the points lie in the same three-dimensional space, the *content* of the 4-dimensional solid, which they form, is equal to zero. This might be explained in a simpler way by considering 4 points in a 3D space - they lie on the same 2D plane if and only if the volume of the 3D solid is zero.

4. The *content* of an n-dimensional solid can be calculated given all the edge lengths, using the Cayley-Menger Determinant. In this case, we don't know a single length, but we know the configuration is correct only, if its 4D volume is zero. Therefore, we get a matrix equation for x, which simplifies easily to a quadratic.

5. The equation produces two solutions, and in fact - both are theoretically possible. The possible measures, that can be applied to pick the correct one, are described later.

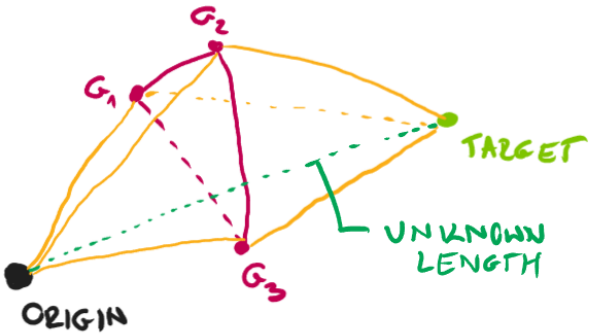

Figure 1: Configuration of nodes which enables calculating the missing length.

Once a node knows the distances to at least 4 anchor nodes (with known x, y, z), it can compute its own position using linear least squares minimization.

2 Interpreting the raw measurement data

In this section I present some revealations, that can improve the processing of repeated measurement data.

2.1 Dealing with reflections off walls

The characteristic of the Bluetooth radio makes the waves bounce off most walls. The resulting measurement errors are gigantic in comparison to the desired accuracy. In case someone desires a rather imprecise positioning algorithm, one could just naively measure the distances along the shortest hop path. Therefore, in order to really witness accuracy gains given by the described triangulation algorithm, we have to take every possible effort to remove all reflected measurements.

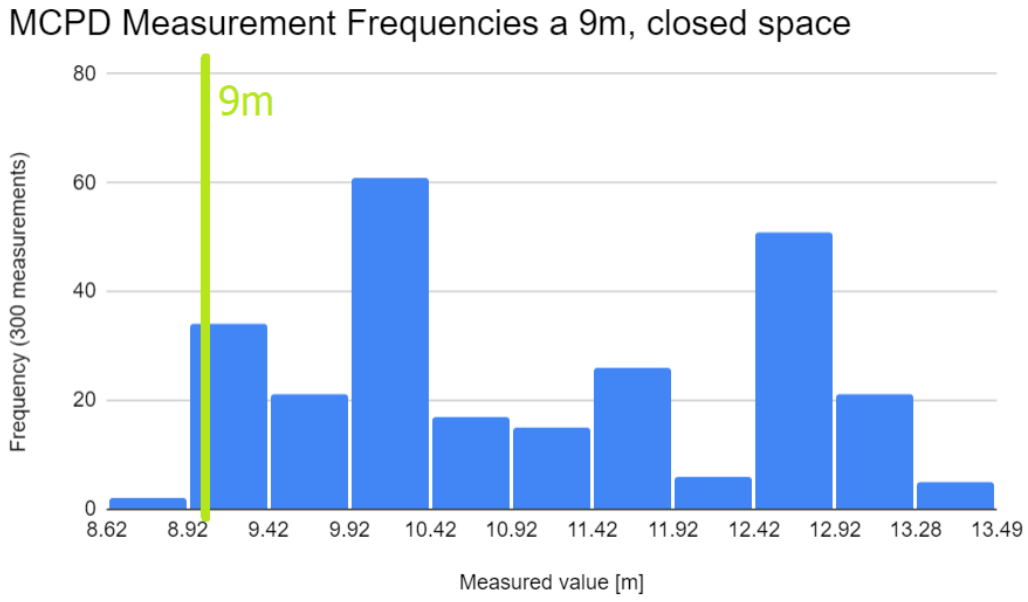

Figure 2: A typical distribution of measurements around the true value.

In Figure 2, one can see typical distance measurement results using the Multi-Carrier Phase Difference method in a semi-closed environment (hallway, narrow room). The most frequent values are distributed around the possible reflection path lengths. The spike with the smallest value is therefore very likely to corespond to the true value - the path without reflections.

A decent method of extracting the best value would be to discard all infrequent value buckets and settle on the smallest of the remaining.

2.2 Measurements through walls

The test data shows, that no accurate distance measurement between nodes separated by a wall is possible. It is best to entirely avoid them, as described in Section 5.

2.3 Short range measurements

The wall reflections have another interesting consequence. When the devices are close to each other, the reflections have an enormous impact on the observed data. The strength of the signal is high enough to suffer from multiple reflections. Moreover, there are many available reflection paths. In consequence, the measurements with real distance below around 3-4 meters give results with high deviations.

This error decreases quickly as the distance between the devices increases. At 5 meters, the results are decently accurate again, as the reflection paths are closer to the straight path.

The experimental data also proves, that the absolute measurement error is near constant and independent of the real distance. This means, that the best way to gather good measurements is to evade including data from the 0-5 meter range, while giving no priority to the shortest distances in the 5-20m range.

3 Calculation of a missing length

In this section we peek at the details of the process of dealing with multiple solutions for the same edge.

3.1 Discarding trivially wrong solutions

The range criterion

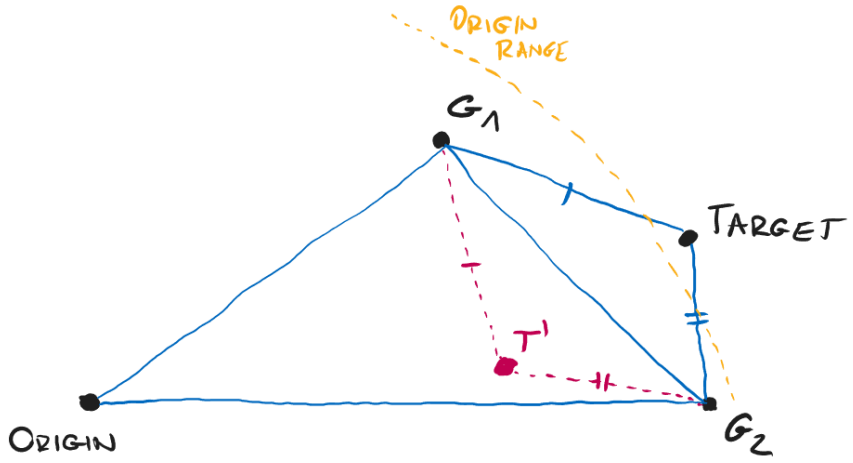

Figure 3: A calculation of distance between origin and target produces two valid solutions. Sometimes however, one of them is within the range of the origin node.

The two solutions given by the equation, TARGET and T', are shown on Figure 3. The other solution always has the property of $|G_iT| = |G_iT'|$ for all G_i nodes comprising the gate.

Most of the time however, the potential solution T' would be so close to the origin, that the nodes could communicate directly. Therefore, no such calculation should take place, which makes the configuration impossible. It is up to the user to determine the cutoff boundary, as the range of most radios is not a perfect sphere.

In my simulations I discarded the solution if it was less than $\alpha_{cutoff} = 0.7$ times the estimated maximum range.

The repetition criterion

In networks with high measurement precision, there is another way to distinguish between the false and correct solutions. The method, depicted on Figure 4, utilizes measurements taken over multiple Gates.

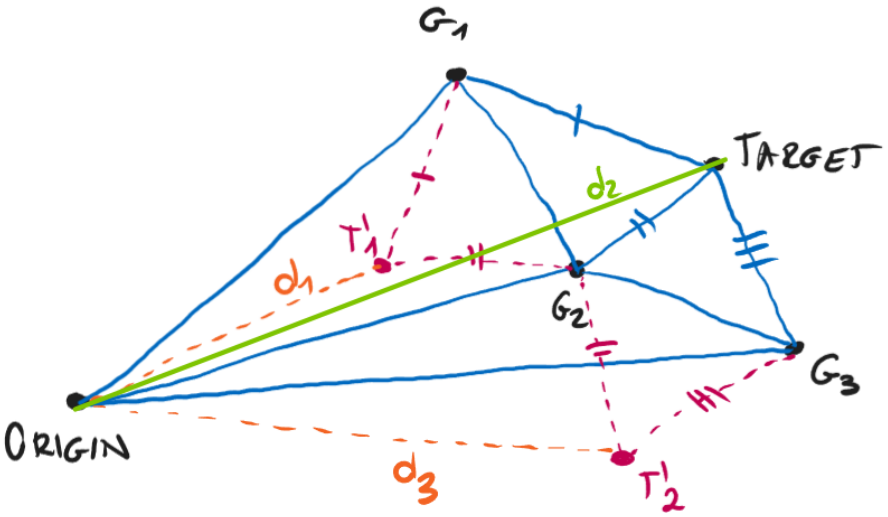

Figure 4: A configuration where a solution was picked because it was the result of multiple independent calculations.

In the configuration above, the following actions have been taken:

1. A measurement was conducted over Gate G_1G_2, producing solutions d_1 and d_2
2. A measurement was conducted over Gate G_2G_3, producing solutions d_2 and d_3

The solution d_2 was repeated, therefore it is the correct one and the others can be discarded.

This method works fine when the measurement data is very close to the real values - otherwise it is tough to consider a solution to have been repeated. This is especially true if the wrong solutions d_1 and d_3 are close in value to the correct one.

3.2 The Solution Set

An improvement of the previous technique is possible, when the data has larger variance. It involves keeping a set of all solutions for a given pair of nodes until a good value can be picked with enough certainty.

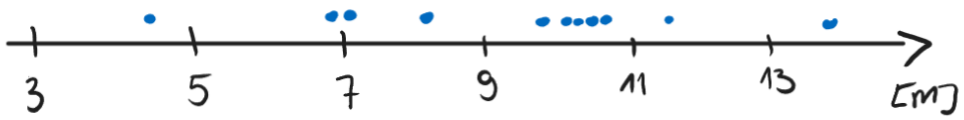

Figure 5: A Solution Set, from which a probable solution can be drawn.

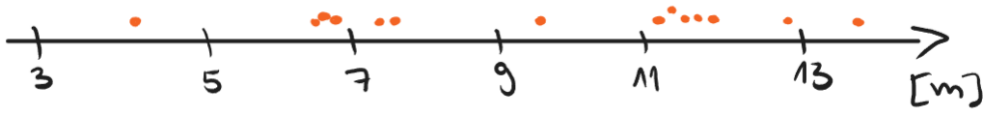

Figure 6: A Solution Set, which has multiple islands.

In Figure 5 one can see a Solution Set with 10 solutions applied. The human eye can draw an easy conclusion - the correct, repeating solution is around 10. On the other hand, in Figure 6, no conclusion can be drawn as there are two 'islands' of solutions. The problem is to find an algorithmic method that would give the same result.

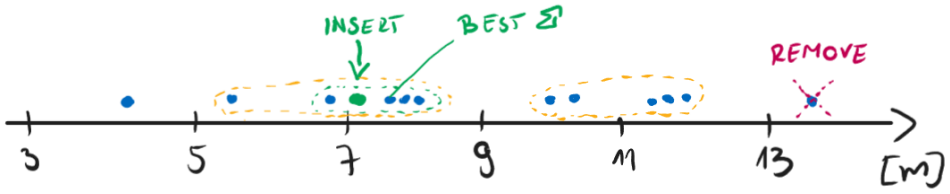

Figure 7: The process of finding the best solution.

Above, in Figure 7, the deciding insertion step is shown. Before, two orange groups have competed with each other and no conclusion could be made. After the insertion of a new solution, the left group became much smaller. Now it can be said with certainty, that the solution is around 7.5.

Below, a procedure is outlined that achieves this goal. Any other algorithm can be used at the user's discretion.

Solution Set Insert Procedure

Parameter	Value	Description
α_{reach}	4	Estimated range of a node
σ_{reach}	0.7	Range criterion discard constant
σ_{min}	10	Minimum length of Set to try to pick a solution
σ_{max}	20	Maximum Set size before it starts to discard solutions
σ_{filter}	5	Derivative filter size including the solution in the middle
σ_{T1}	1.7	Maximal Σ value for a solution
σ_{T2}	0.5	Minimal difference between two highest Σ's

1. If the solution x is smaller than $\sigma_{reach} \cdot \alpha_{reach}$, discard it and return. Otherwise, insert it to the set.

2. If the Set is populated with less than σ_{min} solutions, return.

3. Calculate the derivative of the Solution Set - take the differences between the neighboring solutions

4. For each solution, take the sum of differences (Σ) to $(\sigma_{filter} - 1)/2 = 2$ nodes to its left and right. For example, for a set of $[2, 5, 6, 9, 10]$ this would produce $(6 - 2) + (6 - 5) + (9 - 6) + (10 - 6) = 12$.

5. This value is smaller as the solution is closer to other solutions. Therefore, the solution with minimal 'closeness' Σ is the best.

6. If the set has reached its maximum capacity σ_{max}, discard the smallest or the biggest solution, whichever has the higher Σ.

7. If the smallest Σ is greater than the threshold σ_{T1}, return (pick no solution).

8. If the difference between two best Σ's is less than σ_{T2}, return (pick no solution)

9. Determine the solution with smallest Σ as the correct one.

3.3 Solution Badness

The Solution Set has a major flaw - the *quality* of the inserted solutions is not taken into account. It can be dealt with by marking each solution with a badness value. The greater the value, the less confident should the algorithm be to treat it as a valid one.

Badness originating from measurement data

The measurements given by the nodes can be sometimes provided by lower layers along with a ”likeliness” value. This parameter describes how fitting was the low-level radio data among multiple measurements or frequencies. It has not been tested in practice or simulation. However, it can provide a base Badness value for an edge in the node graph. For now, it is assumed that the Badnesses of all directly measured edges are equal to 1.

Badness originating from calculations

In order for the algorithm to work for large networks, the calculated distances must be taken as ’known’ in subsequent calculations. Because of that, the error can recursively propagate itself. To combat that, the badness should propagate and increase as well. I used the following principle:

The badness of a newly created edge is the maximum of all badnesses of edges used to calculate it, plus one.

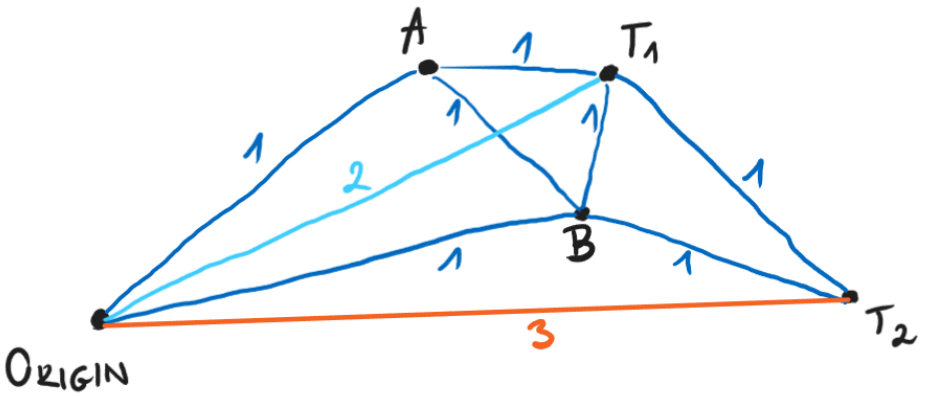

Figure 8: Recursive increase of badness.

In Figure 8, the following actions were taken:

1. The ORIGIN node has calculated the distance to T_1 using Gate (A, B). All given edges were measured directly (Base Badness = 1). The new edge has badness $2 = \max(1, 1, 1, 1, 1) + 1$.

2. The ORIGIN node has calculated the distance to T_2 using Gate (T_1, B). Three edges OB, T_1B, T_2B were measured directly, while the edge OT_1 was calculated before and has badness 2. The new edge has badness $3 = \max(2, 1, 1, 1, 1) + 1$.

Embedding Badness into the Solution Set

In order to let the Solution Set make certain decisions based on the Badness, we need to modify the Insert procedure. Let us make some observations first:

- The minimal badness of an entry of a Set is 2. This is because no Sets are needed for the directly measured edges.

- When a Solution Set contains an entry with badness 2, that means, that the distance between the nodes could be computed using only direct measurements and one Simplex Computation. Hence, the nodes are very close to each other.

  Heuristically, in networks with balanced distribution of nodes, it should be possible to determine the correct solution by only using others with badness 2 or 3. Simmilarly, if the minimal badness is 3, discarding all solutions with badness 5+ should suffice.

- Let us picture a Solution Set with average badness 2.5. If there are two solution islands, but one of them averages badness 2.2, while the other 2.8, we should discard the other one.

On the following page, an updated procedure for picking a solution is presented.

Solution Set Insert Procedure with Badness

Parameter	Value	Description
α_{reach}	4	Estimated range of a node
σ_{reach}	0.7	Range criterion discard constant
σ_{min}	10	Minimum length of Set to try to pick a solution
σ_{max}	20	Maximum Set size before it starts to discard solutions
σ_{filter}	5	Derivative filter size including the solution in the middle
σ_{T1}	1.7	Maximum Σ value for a solution
σ_{T2}	0.5	Minimum difference between two highest Σ's
β_{diff}	1	Maximum difference between the smallest and highest badness
β_{cutoff}	1.1	Multiple of average badness of the Set as a cutoff threshold

1. If the solution is smaller than $\sigma_{reach} \cdot \alpha_{reach}$, discard it and return.
2. If the solution's badness is greater than the minimum badness of the Set plus β_{diff}, discard it and return.
3. Insert the solution into the Set
4. If the Set is populated with less than σ_{min} solutions, return.
5. For each solution, take the sum of differences (Σ) to $(\sigma_{filter} - 1)/2 = 2$ nodes to its left and right. Also, compute the maximum and the average badness of these neighbors and the solution (B_{max}, B_{avg}).
6. The Σ value is smaller as the solution is closer to other solutions. Therefore, the solution with minimal 'closeness' Σ is the best.
7. If the set has reached its maximum capacity σ_{max}, discard the smallest or the biggest solution, whichever has the higher Σ.
8. If the smallest Σ is greater than the threshold σ_{T1}, return (pick no solution).
9. Mark all the solutions with B_{avg} greater than the average badness of the set times β_{cutoff} as invalid for the current iteration.
10. If the difference between two smallest valid Σ's is less than σ_{T2}, return (pick no solution).
11. Determine the solution with smallest valid Σ as the correct one. Mark it with corresponding badness B_{max}. Notice, that the badness of the result remains an integer as it is a maximum of integers.

3.4 Possible further improvements

The algorithm can still be improved in the following areas:

- Incorporating a better solution picking algorithm, which consideres the badness values more like a human would do.
- Taking into account the 'likeliness' provided with the measurements as a predecessor for badness
- Finding a way to bind the parameters between each other - with the current set, when one of them is changed, for example the node reach to 8, the algorithm would not work at all and all others would need to be manually adjusted.

4 Computation Strategies For a Macro Perspective

In this chapter I will outline the main concepts regarding the macro perspective of the positioning process. We have already discussed methods of calculating a single distance based on finding the diagonal of a solid. What follows is the strategy of intelligent ordering of these operations, to ensure a quick propagation of new measurements across the network.

It is to be noted here, that, due to the distributed nature of the algorithm, a node posesses only some limited information about the state of the network. It would be nearly impossible to both store all newly constructed relations between nodes, and update them across the whole network. Therefore I decided to make a practical assumption, that a node only remembers a list of other nodes, to which the distance is already known (KNOWN SET, K_{node}). It also has a partially complete list of unknown node addresses, which it can extract from network traffic.

So, how should the node decide, what should be its next target, and which gate to pick?

4.1 Random Gate Strategy

The first idea that comes to mind is involving randomness in the process. Due to the fact, that a node has no information regarding the position of its target, there is no way of picking the right gates just by looking at the network plan "from above". The first strategy I tried looked as follows:

Random Gate Strategy Procedure for Two Dimensions

1. Pick a random node from the Known Set, K_{origin}, call it G_1.
2. Ask that node for its Known Set, K_{G_1}.
3. Take the intersection $G = K_{origin} \cap K_{G_1}$. This is the set of potential second gate nodes.
4. Pick a random node G_2 from G. The nodes G_1 and G_2 form a valid gate.
5. Ask G_2 for its Known Set, K_{G_2}.
6. The Target Set is a set of nodes, which are neighbors of both G_1 and G_2, but are not already known by the origin. Therfore, take

$$T = (K_{G_1} \cap K_{G_2}) \setminus K_{origin}$$

7. For each member of T, conduct a distance calculation as specified before.

In the three-dimensional case, an analogous process can be followed that involves finding a third gate node.

The major flaw of this strategy is that in most cases, a gate will be drawn which cannot provide any new results. There is no way to store all „used" gates, as due to new known lengths appearing in other segments of the network, one cannot determine if the gate is useless before computing the target set. In consequence, a great waste of computation time, which results in little progress, may happen. It is however a valid strategy and it can ultimately calculate all the distances in the network, which are findable under the assumed parameters of the Solution Set.

4.2 Random Target Strategy

In order to reduce the wasted random draws, an alternative strategy can be followed. Instead of randomizing the gate and calculating distance to all targets available through it, one can randomize a target and strike it over all possible gates.

This reduces the „wasted randomness", as each time we are guaranteed to attempt a calculation of distance towards an unknown node. The procedure looks as follows:

Random Target Strategy Procedure for Two Dimensions

1. Draw a random node TARGET from the Unknown Set (defined as $\Omega \setminus K_{origin}$, where Ω is the set of all addresses which the node has registered while passing on network traffic).
2. Ask TARGET for its Known Set, K_{target}.
3. Take the intersection $G = K_{origin} \cap K_{target}$. This is the set of valid gates between the origin and target.
4. For each pair $(G_1, G_2) \in G$, conduct a distance calculation as specified before. In the 3D case, take triplets.

While this strategy seems superior to the Random Gate Strategy, it has a well-disguised drawback. The random target choice can provide targets that are too far away from the origin, which results in the Gate Set G being frequently empty. The „wasted randomness" has therfore not been entirely eliminated.

Nevertheless, the results of simulations hint, that the Random Target Strategy is much faster than randomizing the gate.

4.3 Hop Level

The inefficiences of the Random Target Strategy can be easily dealt with. The goal would be to roughly order the target nodes by distance away from the origin. Having done that, the random target would be chosen exclusively from the closest ones, thus decreasing the possibility of the event, that no gate can be formed between the origin and the target.

Such ordering can be achieved by taking the minimal number of network hops from the origin to the target. This can be easily measured in the network and even if the number is sometimes wrong due to environmental conditions, it will have no effect on the result.

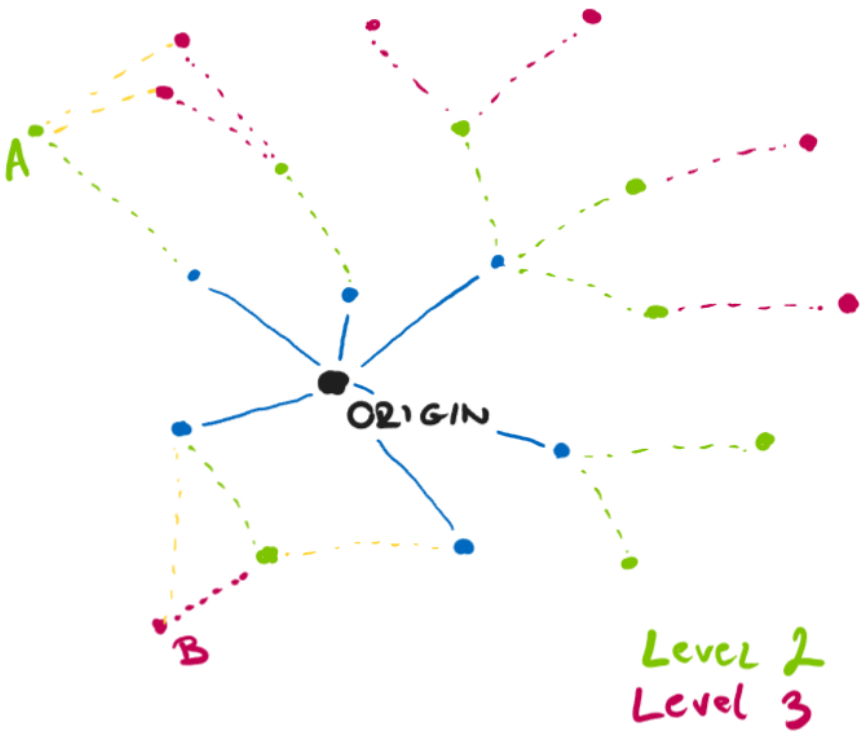

Figure 9: Hop distances to other nodes

The principle of this improvement is that the origin node can only draw targets, which are at or below the origin node's *hop level* Λ. This variable depends on the past successful measurements. The naive approach would be, that having established the distances to all nodes of the previous level, the node moves on to the next. There exist situations though, where the distance to a node two hops away can only be measured using other nodes further away - an example (Node A) can be seen at Figure 9. Another problem is posed by nodes, that have little number of connections to the main network body, making it impossible to calculate distance toward them (Node B).

To combat this, let us define Λ_x as the set of all nodes being *at most* x hops away from the origin, and λ_t as a threshold parameter chosen by the user. If a node has calculated distance to at least $\lambda_t \cdot |\Lambda_x|$ nodes at level x, it advances to level $x + 1$. In my simulations, the value $\lambda_t = 0.8$ proved to work well.

The implementation of the Hop Level strategy speeds up the process by the factor of around 10. This is because the nodes are starting with targets near them, which gives them more immediate gate choices for further nodes.

5 Accuracy Improvements

5.1 Reducing the Problem to Two Dimensions

The presented algorithm works both in two and three dimensional configurations. The 3D case is slightly more complex due to the following factors:

- The Gate comprises three instead of two nodes. In consequence, there are less possible gates in areas of the network which are scarcely populated by nodes. This results in the alegorithm not being able to access certain nodes and leaving their location unsolved.
- More nodes have to be chosen and more set memberships determined in each iteration of the algorithm. Also, the Cayley-Menger equation itself is more complex, as it involves computing a determinant of sixth order. This increases the computational cost significantly.
- More anchors are needed to provide sufficient information to position the nodes.
- More degrees of freedom in node positioning can result in higher errors in the final solution.

In many real applications the nodes are already positioned in a common 2D plane, for example the lamps on the ceiling of a building floor. Therefore it would be profitable to run the positioning task with the nodes grouped by 2D planes in advance. There are several ways to reduce the problem to the two-dimensional case:

- **Barometric altitude measurement.** Many nodes that are being used in real buildings include barometric sensors. They are accurate enough to measure relative height difference with accuracy of around 3m. This is enough to distinguish between the floors of a typical building.
- **By floorplan.** If the pressure sensors are not available, one can assign the nodes to the floor groups semi-manually while they are being installed.

5.2 Avoiding Measurements Over Walls

As mentioned before, walls have a devastating impact on the measurements. Therefore, it would be best to entirely avoid conducting measurements between devices separated by a wall. I describe several techniques of dealing with this problem below.

- **Increasing Edge Badness based on data variance.** The test data has shown clearly, that the measurements through walls are not only providing inaccurate data (around 1m too high on average), but they also fluctuate more. Increasing badness of those edges would disincentivise the algorithm to use them to calculate other distances.

  It is worth mentioning, that progress has been made on low level measurement algorithms recently. Now that the data contains an additional „likeliness" parameter for each measurement, it may already be related to variance between different channels. Adjusting badness based on this parameter has not been tested yet.

- **Using built-in lightness sensors.** Many devices that comprise Bluetooth Mesh networks are lamps that have lightness sensors on board. They can be used to detect walls between the devices. To do this, one node turns on its lamp (at nighttime), and its network neighbours measure the lightness. If it's below certain threshold, there must be a wall blocking off the light at least partially. Measurements between such devices should have increased badness or even be discarded in case of complete lack of received light.

5.3 Correct Anchor Placement

Another configuration characteristic that influences the final result is the placement and dencity of the anchor nodes. The figures on the following page depict two different configurations of anchors (orange nodes on left diagrams) and the results of the algorithm (right diagrams), where the red dots are the calculated positions of nodes, while the black lines point to the real locations.

When the anchors are grouped, the results can still be calculated, but the final positions have larger errors. It is best to position the anchors on the circumference of the network, while avoiding placing many of them in a line.

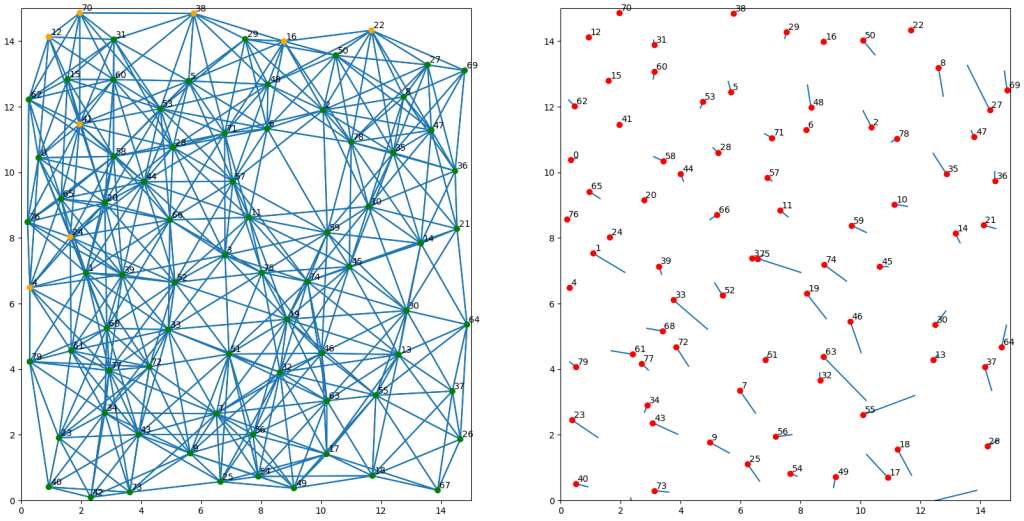

Figure 10: Badly placed anchors (orange) in the top-left corner

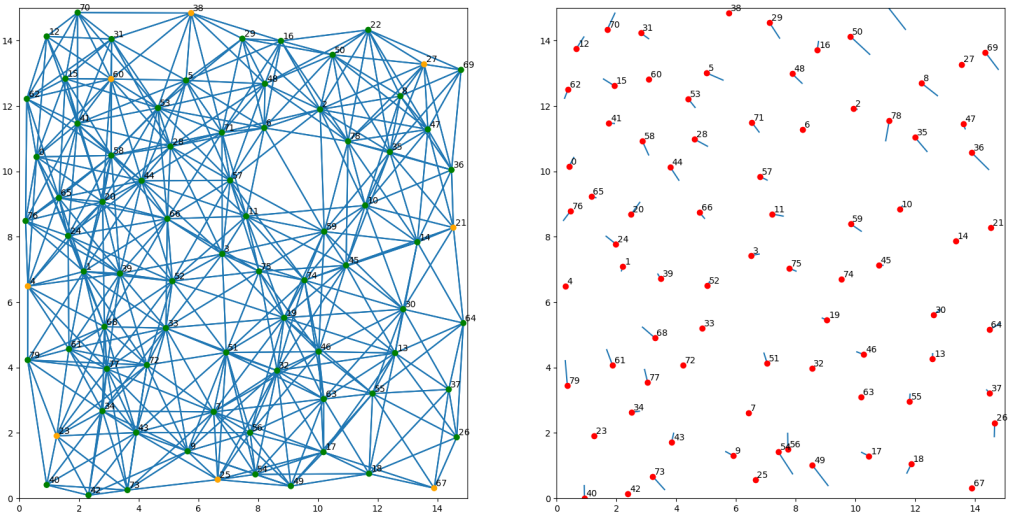

Figure 11: Optimally placed anchors (orange) in a circle around the network.

6 Computation Efficiency Improvements

6.1 Simplification of the Cayley-Menger Equation

Calculating a 6x6 (3D) or 5x5 (2D) matrix determinant with a symbolic variable is a relatively large workload if it has to be executed $O(n^4)$ times. The matrix has some interesting properties though, that enable certain simplifications. The determinant to be calculated looks as follows:

$$\det M = \begin{vmatrix} 0 & A & B & x & 1 \\ A & 0 & C & D & 1 \\ B & C & 0 & E & 1 \\ x & D & E & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix} = 0 \tag{1}$$

One can easily see, that this simplifies to a quadratic equation. Hence, the goal is to find the coefficients a, b, c of the equation $ax^2 + bx + c = 0$.

By using Laplace Expansion on the first row, we get:

$$\begin{aligned} \det M &= 0 - A \cdot \det(M_1) + B \cdot \det(M_2) - x \cdot \det(M_3) + 1 \cdot \det(M_4) \\ &= \begin{vmatrix} 0 & A & B & \mathbf{0} & 1 \\ A & 0 & C & D & 1 \\ B & C & 0 & E & 1 \\ x & D & E & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix} - x \cdot \begin{vmatrix} A & 0 & C & 1 \\ B & C & 0 & 1 \\ x & D & E & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix} \end{aligned} \tag{2}$$

Writing (2) as $\det(L) - x \det(R)$, we can further expand both of those matrices (Left, Right) in respect to the first column:

$$\det L = \begin{vmatrix} 0 & A & B & 0 & 1 \\ A & 0 & C & D & 1 \\ B & C & 0 & E & 1 \\ \mathbf{0} & D & E & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix} - x \cdot \begin{vmatrix} A & B & 0 & 1 \\ 0 & C & D & 1 \\ C & 0 & E & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix} \tag{3}$$

$$\det R = \begin{vmatrix} A & 0 & C & 1 \\ B & C & 0 & 1 \\ \mathbf{0} & D & E & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix} + x \cdot \begin{vmatrix} 0 & C & 1 \\ C & 0 & 1 \\ 1 & 1 & 0 \end{vmatrix} \tag{4}$$

Now we can easily read the coefficients a, b, c, as $\det M = \det L - x \det R$:

$$\begin{aligned}\det M = &-\begin{vmatrix}0 & C & 1\\ C & 0 & 1\\ 1 & 1 & 0\end{vmatrix} x^2 \\ &-\left(\begin{vmatrix}A & B & 0 & 1\\ 0 & C & D & 1\\ C & 0 & E & 1\\ 1 & 1 & 1 & 0\end{vmatrix} + \begin{vmatrix}A & 0 & C & 1\\ B & C & 0 & 1\\ 0 & D & E & 1\\ 1 & 1 & 1 & 0\end{vmatrix}\right) x \\ &+\begin{vmatrix}0 & A & B & 0 & 1\\ A & 0 & C & D & 1\\ B & C & 0 & E & 1\\ 0 & D & E & 0 & 1\\ 1 & 1 & 1 & 1 & 0\end{vmatrix}\end{aligned} \tag{5}$$

This can be simplified further. The first determinant can be calculated "by hand", while the x coefficient consists of determinants of a matrix and its transpose, which are equal. Therefore, we get:

$$\det M = -2Cx^2 - 2\begin{vmatrix}A & B & 0 & 1\\ 0 & C & D & 1\\ C & 0 & E & 1\\ 1 & 1 & 1 & 0\end{vmatrix} x + \begin{vmatrix}0 & A & B & 0 & 1\\ A & 0 & C & D & 1\\ B & C & 0 & E & 1\\ 0 & D & E & 0 & 1\\ 1 & 1 & 1 & 1 & 0\end{vmatrix} \tag{6}$$

These two matrices contain a lot of zeros. Their determinants simplify to expressions of 9 and 12 terms respectively. I leave finding them as an excercise for the reader. In case you really don't feel like dealing with a 5x5 matrix, you can use the incredible tool Matrixcalc.org to find them and embed them into your code.

The simplification for three dimensions can be derived in a simmilar way.

6.2 Distributing the Algorithm

All the enhancements presented in this paper were made to preserve the main advantage of the algorithm - its ability to be ran concurrently on every node. There are, however, some issues that need attention.

- **Concurrent initiation of a measurement.** The networking protocol must provide a solution to a situation, where two nodes decide to become initiators of the measurement between them at the same time. Also, a measurement should not interfere with other measurements done at the same time in different areas of the network.

- **Sharing the Solution Set.** The Solution Set for an edge must always be kept on both nodes of the edge in the same state. An update to one end must be immediately shared to the other end.

- **Solution Set update queue** A node might receive a lot of updates for its Sets at the same time from other nodes even if it is currently not conducting any calculations or measurements. No incoming data can be lost, as this would mean, that the Set at the other side of the edge would have more entries than the updated one.

6.3 Limitations

Finally its time to discuss some hurdles, that an implementation must overcome. While good-looking on paper, the algorithm will most likely be ran on small chips with little available memory.

- **Memory complexity of the Solution Set approach.** Each Solution Set contains over a dozen of floating-point solutions and their integer badnesses. For the algorithm to fully operate, a node must keep a Solution Set for each outgoing edge. This means that for networks of 1000 nodes at least 50kB of memory must be allocated for storing the solutions alone. Even that number requires a creative approach.

  A partial solution could be to delete the Solution Sets, once they have established a true solution. This means however, that a result cannot be updated if solutions with lower Badness become available.

  Another option is to prioritize storing solutions toward nodes, which lie closer in number of hops to an anchor, while discarding others. The ultimate goal is to find the distances to the anchors, not all nodes of the network.

  The Solution Set for an edge could also be stored on only one device, which would slightly decrease performance and network traffic.

- **Sharing data in networks with extremely low throughput.** A single packet of a Bluetooth Mesh network can contain only 9 bytes of payload. While this is far enough to send op-codes and simple values, the algorithm would require frequent sharing of addresses. The extra traffic is not huge, but it could make an impact. One way to reduce it (and also save some memory) could be to identify the nodes by parts of their adresses. The probability of a collision would still be low enough for the algorithm to function properly.