Making road networks that lower risk of traffic accidents

Lowe Hjerth

Kandidatuppsats 2025:20
Matematisk statistik
Juni 2025

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2025:20**
http://www.math.su.se

Making road networks that lower risk of traffic accidents

Lowe Hjerth*

June 2025

Abstract

In this thesis we compare different algorithms for constructing a road network from a set of points. The aim is to minimize the risk of traffic accidents by spreading out traffic across the network. We focus on the structure of the road network, rather than traffic signs and sight-lines. This is done with a simulation study using a simplified model where the road network is represented by a euclidean graph, and each point has a population of cars traveling to the other points. The risk of a point is the square of the number of cars traveling trough it, and the aim is to lower the total risk per capita with efficient use of road space.

A set of algorithms are tested that work off a base of the minimal spanning tree of a Delaunay triangulation, iteratively adding back edges from the triangulation based on some criteria. The notable front- runners are the algorithms many paths and crowded, which prior- itize adding back edges between points with many routes through it, and edges between high population points, respectively. We also find that the algorithms that add back edges with high or low difference in population between the connected points perform no better than each other, suggesting that the difference in population between connected areas is an unimportant factor when designing a road network.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: lowe.hjerth@gmail.com. Supervisor: Maria Deijfen, Daniel Ahlberg.

Contents

1 Acknowledgments

A big thank you to my two supervisors Maria Deijfen and Daniel Ahlberg. They were a huge help every step of the way and this would not have been possible without them.

ChatGPT was used only for spit-balling ideas and bug-fixing code.

2 Introduction

When designing a road network for cars, an important consideration is how likely a driver is to experience an accident. This risk can be mitigated with speed limits, improving sight-lines, better traffic signs, et cetera. These examples are only responses to the risks that are inherent to the underlying road network; responding to the symptoms, so to speak. The question then is, how can a road network be designed/improved by changing the structure of the network only? In other words, by only changing what places have roads between them.

In this thesis, we create and analyze a model depicting a road network with a simulation study. The model represents a road network with a euclidean graph with locations represented by vertices/points and roads represented by edges. It is a simplified model where cars are not individually simulated, and roads are always straight lines that do not cross. The aim is to generate a set of points in 2-dimensional space, and comparing algorithms for connecting them with roads to minimize the risk of accidents while not using too much road space. Theoretically, if all points had direct paths to all others, there would be no intersections and risk of accidents would be minimal, but this is a clear waste of road space, so it is important we take the amount of road space used into account when analyzing these algorithms.

Since the amount of possible edges for a set of vertices scales quickly, the algorithms will only consider a subset of these edges. Specifically we will use the Delaunay triangulation of the point set we generated, to act as a base for the algorithms to work off of. The Delaunay triangulation is also a planar graph, so there are no crossing edges which simplifies our model. The algorithms will start with the minimal spanning tree of the Delaunay triangulation - the minimal possible network we could use - and iteratively adding back edges from the full triangulation based on some criteria.

3 Theory

3.1 Graph Theory

To analyze a road network, the tools of graph theory are very useful.

A (simple, undirected) *graph* is defined as a set of *vertices* V and a set of *edges* E, where the elements of E are unordered pairs $\{v, w\}$ where v and w are in V. This is often written $G = (V, E)$. A graph is called *connected* if all vertices are reachable from the others by following a sequence of edges in E. If all possible edges between vertices is in the set of edges, the graph is called *complete*.

A *weighted* graph is a graph where each edge in E has a (usually positive) number associated with it. We might call this number the *weight* or *length* of an edge. If these weights refer to the euclidean distance between the two points, we call it a *euclidean graph*.

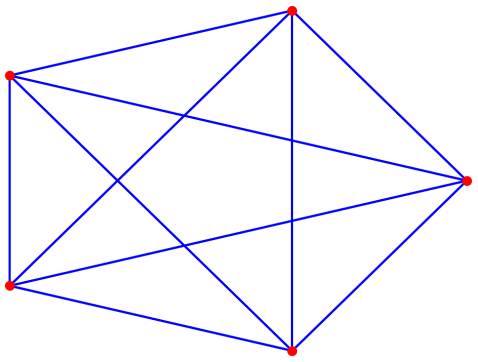
Figure 1: Example graph of 5 vertices

3.1.1 Minimal Spanning Trees

A graph is a *tree* if there is only one path (a sequence of unique connected edges) between any two vertices.

Given a graph G, a *spanning tree* of this graph is a subgraph (a graph using a subset of G's vertices and edges) that is both a tree, and contains all vertices in G. This subtree will only exist if G is a connected graph.

Given a **weighted** graph G, a spanning tree is *minimal* if it has the lowest total edge weight possible.

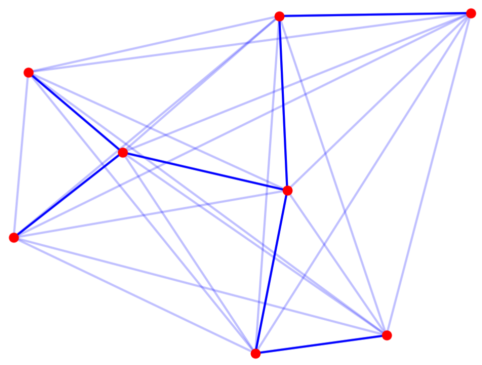

Figure 2: Complete euclidean graph of 8 vertices and its minimal spanning tree (dark blue)

3.2 Delaunay Triangulation

Delaunay triangulations are introduced in *Delaunay Mesh Generation*, Cheng, Siu-Wing, Dey, T. K., & Shewchuk, J. (2013) pages 31-33. Voronoi diagrams are defined in page 154.

A *triangulation* of a finite set of 2-dimensional points P, is a maximal set of non-overlapping edges between the points in P. The stipulation that it needs to be maximal makes it so that the edges only form triangles of points in P, and the outer edges form a convex polygon.

There are many triangulations you can make from a set of points, but an especially useful kind is the **Delaunay triangulation**. It have an additional stipulation that for any triangle in the triangulation, its circumcircle contains no additional points. A triangle's circumcircle is the unique circle that passes through all points in the triangle. Delaunay triangulations maximize the minimum angle among the triangles, which minimizes the amount of ”sliver triangles” in the triangulation.

Delaunay triangulations are also closely tied no Voronoi diagrams. For a finite set of 2-dimensional points P, the Voronoi diagram is the partition of the plane into regions, so that all points closest to some point p in P are in the same region. By drawing edges between points in P if their respective regions border each other, we get the Delaunay triangulation of those points.

3.3 Algorithms

A few well known algorithms are used in this thesis, they are described below.

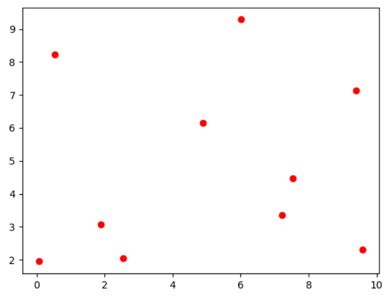

Figure 3: Ten random points

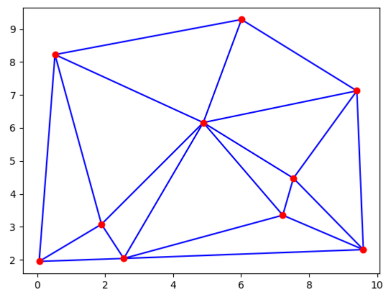

Figure 4: Delaunay triangulation

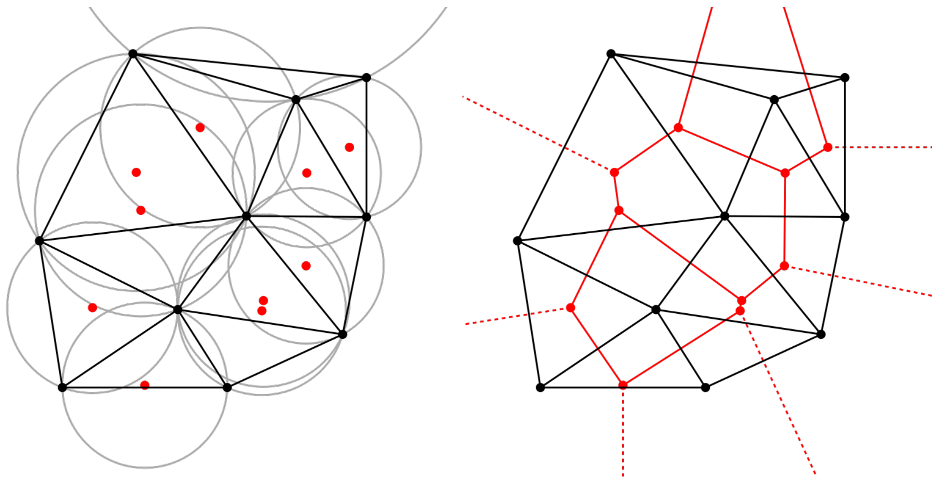

Figure 5: Delaunay triangulation with circumcircles (left) and the corresponding Voronoi diagram (red borders, right). Source: Wikipedia, *Delaunay triangulation* (2025)

3.3.1 Bowyer-Watson algorithm

This algorithm is detailed in *Delaunay Mesh Generation*, Cheng, Siu-Wing, Dey, T. K., & Shewchuk, J. (2013) pages 59-61.

The Bowyer-Watson algorithm computes a Delaunay triangulation for a set of 2-dimensional points P. It works by incrementally adding the points to a valid Delaunay triangulation and replacing any triangles that have circumcircles containing the new point.

It starts by placing three new points q_1, q_2, q_3 not in P, so that the triangle made by q_1, q_2, q_3 contains all points in P, which we call the super triangle. Then we add one point p_1 from P, and remove all triangles whose circumcircles contain p (for the first iteration it is only the super triangle), creating a hole in the triangulation. Then we add all triangles with two points from triangles that were removed, and p. This fills the hole and gives us a Delaunay triangulation for the points q_1, q_2, q_3, p. Continue this process of adding points until all elements of P have been added, and lastly remove all triangles made using q_1, q_2 or q_3. Then we have the Delaunay triangulation of P.

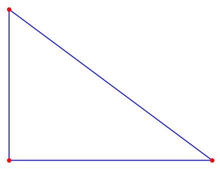
Figure 6: Super triangle

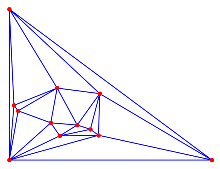
Figure 7: After adding 9 points

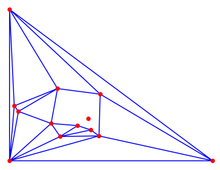
Figure 8: Adding tenth point, removing some triangles

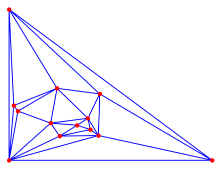
Figure 9: Adding new triangles to fill hole

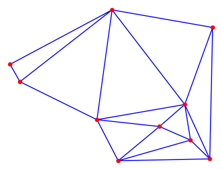
Figure 10: Final triangulation after removing $q1, q2, q3$

3.3.2 Prim's algorithm

This algorithm is detailed in *Discrete and Combinatorial Mathematics - An Applied Introduction* by R. Grimaldi (2004) page 639.

Prim's algorithm finds a minimal spanning tree of the connected graph $G = (V, E)$.

First, choose an arbitrary vertex in V and add it to the tree. For each iteration, choose the *shortest* edge in E connected to the tree, that doesn't connect two vertices already in the tree, and add the edge and its other vertex to the tree. This way, we always add a new vertex every iteration. Once all vertices in G are present in the tree, we have a minimal spanning tree of G.

3.3.3 Dijkstra's shortest path algorithm

This algorithm is detailed in *Discrete and Combinatorial Mathematics - An Applied Introduction* R. Grimaldi (2004) page 631

Dijkstra's algorithm is an algorithm for finding the shortest path from a vertex v to another vertex u in a connected weighted graph G.

First, list all edges connected to v and order them by length. These are all paths of one edge. Look at the shortest path (the path with the lowest sum of their edge lengths) l, and add to the list all paths starting with l with one more edge added. For the first iteration, that would be all paths of two edges where the first edge in it is the shortest edge from v. Then remove l from the list. Now sort the list again, look at the shortest path, extend it by one edge and add those to the list, removing the original. Continue this until the shortest path ends at the vertex u. this is the/a shortest path from v to u.

4 Model

Our model will consist of vertices and edges connecting pairs of these vertices. The vertices representing abstract locations - buildings, cities, et cetera - and the edges represent roads between them. To simulate the flow of cars, each vertex will have a population, and each person will have a location to travel to by following a path. The more people travel through a specific vertex, the higher the risk of accidents will be in that vertex. This gives us six decisions for how the model is constructed:

- How are the points generated?
- How will the edges, and thus road network, be generated?
- How are the point's the population generated?
- How will cars decide where to travel?
- How will cars decide what path to take to a destination?
- How will the risk in a point be determined based on people traveling through?

4.1 Points

To generate the points to use for our network, we must decide how they will be distributed in the x-y-plane.

We will limit the points to be generated in a 10 by 10 square, with a uniform distribution. We could use a distribution that generated points in the entirety of the plane, but this would mean some networks would likely have outlier points far away, skewing our data. By limiting the points to a square, we normalize our data to focus on our network creation algorithms, rather than having to worry about how the generation affects the data.

4.2 Edges

How we connect the points in our model is the central focus of this study. To make a road network we require it to be connected, so that anyone in one point can make their way to another. A natural way of doing this is with a triangulation. This would mean there are no overlapping edges, which would represent crossroads. In this study, we will consider the points themselves to be crossroads, so if two edges crossed, we would simply turn that crossing into a point. Thus, the lack of crossing edges in a triangulation is of minimal importance for the purposes of this study.

A specific type of triangulation that fulfills our needs well are Delaunay triangulations. They maximize the minimum angle among the triangles present. In other words, they minimize the occurrence of *sliver triangles*; triangles with one very small angle. A simplification of our model is that there is no maximum amount of cars in a road, so a sliver triangle means that two of the edges are nearly the same road. This is a waste of road space, so their minimal occurrence in Delaunay triangulations make them a good fit for our model.

Another reason Delaunay triangulations fit our needs is their connection to Voronoi diagrams. Voronoi diagrams have two regions bordering if the points on the border are equally close to the points from the set inside these regions. These borders, when turned into edges, create the Delaunay triangulation. This shows Delaunay triangulations have a geometric property. Traversing a Delaunay triangulation's edges is the same as traversing a Voronoi diagram's borders.

While a Delaunay triangulation connects our points into a reasonable road network, it might have too many unnecessary roads. By the nature of a triangulation, it creates a convex shape. If there is low demand for paths to and from a point on the end of this shape, that road space might be better used elsewhere. We can consider the Delaunay triangulation as the "maximum" network for a set of points, so it is natural to consider what the "minimal" network would be. The minimal spanning tree of the triangulation is a good candidate since it is the network with the smallest possible road length. It just so happens that the Delaunay triangulation's minimal spanning tree is the same as the euclidean minimal spanning tree, see M. I. Shamos. 1978. In other words, it is the absolute smallest network that connects all our points.

Now that we have a maximal network and a minimum network for our points, one being a subgraph of the other, we will use iterative methods of adding edges from the triangulation to the minimal spanning tree. The comparison of these methods will be the focus of this study. The methods are described in the section 5.

4.3 Population

Each point will have a population assigned to it that equals the number of cars that will travel from it. Each point will be independently assigned a population from a uniform distribution, for this study the range $[200, 1000]$ was chosen. This does mean that the total population will vary across networks, but by looking at the risk per capita we account for this variance.

4.4 Destinations

Each car will have a destination to travel to. We want this to be deterministic, since our study iterates on the structure of a road network. The only random thing should be the network itself.

The cars should choose some destinations more than others. Otherwise, The amount of cars traveling to each point will be nearly the same. Instead, we can reasonably assume that a person is more likely to want to travel to points with higher population. In real life this would be analogous to traveling to cities more often than villages, whether that be for work or vacations, et cetera.

For each point, a proportion if its population will go to each other point proportional to the population at that other point. Algebraically, if we have a network with the points A, B, and C, with populations 4, 2, and 3 respectively, we take A's proportion of the remaining population $\frac{A}{A+B}$ and multiply it by C's population. This gives us $\frac{C \cdot A}{A+B} = 12/6 = 2$ people traveling from C to A. Similarly we get 1 person traveling from C to B.

These resulting proportions will likely not be whole numbers if we use large populations. To remedy this, we want to round the proportions so their sum equals the population we are drawing from (C in the example above), so all cars leave the point and no cars go to multiple points. To that end we sort the proportions in descending order by their fractional part, and then take their floor. The difference between the sum of the floored proportions and the population it should equal is now some positive integer n, so we add 1 to the top n floored proportions, which were the closest ones to the next highest number by our previous orderings. These rounded proportions will be the amount of cars sent to each destination.

4.5 Paths

When having a car decide which path to take to get to their destination, similarly to the destinations themselves, we want it to be deterministic. An easy assumption we can make is that if someone wants to make their way to a destination, they want to take the shortest path to their destination. In the modern day, route planning services take into account traffic while calculating the fastest route, but that is only as a response to the effects of the network already in place, the focus of this study. Thus, we will be simplifying away this factor, and assuming that the time it takes to follow a path is simply the length of the constituent roads.

Every car will take the shortest path to their destination, which will we uniquely determined outside of degenerate cases in our network. Prim's algorithm will be used to find these paths.

4.6 Risk

For this study we will be working under the simple assumption that more cars $\rightarrow$ more accidents. We will be assuming accidents only occur in points, not on roads. The motivation for this is that intersections necessarily have cars meet and turn, while stretches of road such as motorways do not.

Risk will determined at each point by a function $f(x)$, where x is the amount of cars that drive through it. This function should be nonlinear, since otherwise the distribution of cars in the network would not change the total risk, only the amount of points they travel through. It should also be an increasing function because of our assumption that more cars $\rightarrow$ more accidents. Specifically this function should have the property $f(x) > x$ for all $x > 1$, so risk increases faster than x increases. The simplest choice for such a function is $f(x) = x^2$. While other functions are possible, it would likely only serve to complicate the model. The effect of a different risk function may be examined by future studies.

5 Simulation

Our model is a road network represented by a euclidean graph with populations assigned to each point, and risk calculated at each point based on the number of people traveling through that point. We also have a procedure for testing this model by generating a set of points and finding its Delaunay triangulation, and then iteratively adding back edges to its minimal spanning tree according som some algorithms. Now we have to create these algorithms and apply them. The algorithms we use will be based on some attribute of the points and edges.

5.1 Algorithms for Extending Tree

The simplest algorithms would be ones that look at the length of each edge to be added, the two extremes of this kind being the following algorithms:

- **short** - add back edges in *ascending* order of length
- **long** - add back edges in *descending* order of length

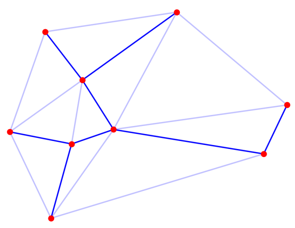
Figure 11: Initial network

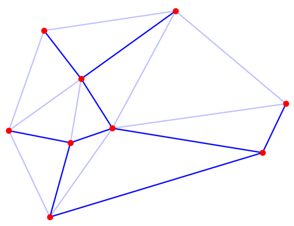
Figure 12: iteration 1 of **long**

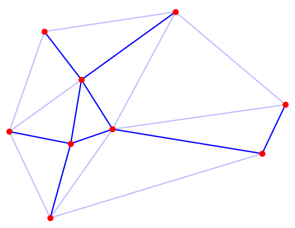
Figure 13: iteration 1 of **short**

Another attribute of a road network in our model is the population of a point. Since each edge connects two different points, we have two populations to take into count. The simplest ways to use these populations is to look at their sum and their difference. Taking the two extremes of these we get the following methods:

- **crowded** - each iteration, adds edge with the *highest* population summed between its connected points
- **alone** - each iteration, adds edge with the *lowest* population summed between its connected points
- **high population difference** - each iteration, adds the edge with the *largest* difference in population between its two connected points
- **low population difference** - each iteration, adds the edge with the *smallest* difference in population between its two connected points

We can also inform the algorithm with the paths cars travel through.

- **many paths** - each iteration, checks the total amount of fastest paths (not cars) that travel through the connected points, and adds the edge one with the *most*.

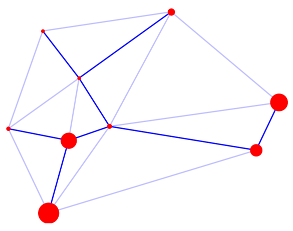
Figure 14: Initial network (populations represented by size)

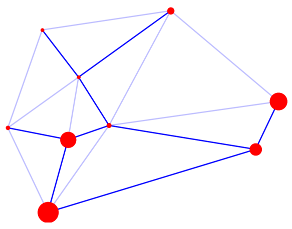
Figure 15: iteration 1 of **crowded**

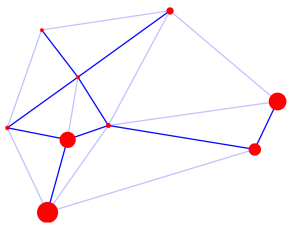
Figure 16: iteration 1 of **alone**

- **few paths** - each iteration, checks the total amount of fastest paths (not cars) that travel through the connected points, and adds the edge one with the *least*.

5.2 Code

The code used to build the model and run simulation was written in Python. We used the libraries `math` and `numpy` for a few mathematical operations and data organization, and the libraries `random` and `pandas` were used for simulating random points and saving data. The library `time` was used to time how long simulation took.

The code for computing Delaunay triangulations using the Bowyer-Watson algorithm is based on work by Viter, Antonii (2024). They made a demonstration of python code for the Bowyer-Watson algorithm, which was instrumental in understanding the algorithm and how to implement it.

The two datasets that were made took approximately 5 hours to generate.

6 Results

The amount of points to start with for a simulation is not self evident. We chose to make sets of points with size 20, 25, and 30. For each of these sizes we generated 100 sets, each with an accompanying Delaunay triangulation and minimal spanning tree. Each method for extending the tree was applied for nine iterations on each minimal spanning tree. Figures 17 - 20 contain plots of the average risk per capita for each algorithm and the amount of edges added. The first three use the 100 point sets sized 20, 25, and 30, and the last one uses all 300 sets. The plots are made with the number of edges along the x-axis.

We could try to plot with the total road length added as the x-axis, but because our stopping criteria for the algorithms is the number of edges added, the method **short** will have lacking data for longer total road lengths, and **long** will be the only one with data for especially long road lengths. Thus the resulting plots would be unhelpful.

Notably, the **short** method was by far the least effective at reducing risk

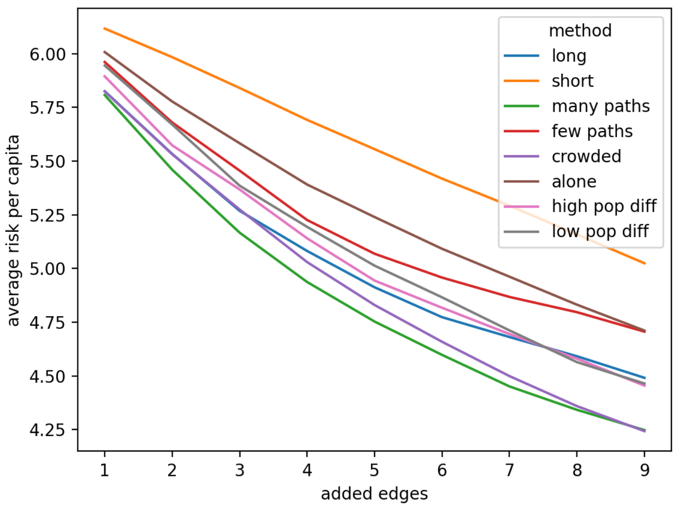

Figure 17: Average risk per capita for graph size 20

in these plots. This is likely a result of plotting based on the number of edges added rather than road length. Since this doesn't reflect the actual amount of road space used, and because this data set is unfit for plotting along total road length added, a new set of data was generated. The difference being that the stopping criteria now was the total length of the added. The stopping point was chosen to be halfway from the total length of the minimal spanning tree and the Delaunay triangulation.

This new set of data no longer has the problem of some algorithms not fitting well when plotting with road length added as the x-axis. Now for any given base graph, all algorithms reach roughly the same total road length added at the stopping point. However We still cannot simply plot along the amount of added road length. These values are not integers, so we cannot take averages of risk per capita, as there is likely only one data point with each value in the dataset. We also cannot simply plot all the points directly, as the exact values for added road length and risk per capita do not follow a neat line. Some datapoints have high added road length and low risk per capita, and vice versa, even for the same algorithm. Instead, to plot the

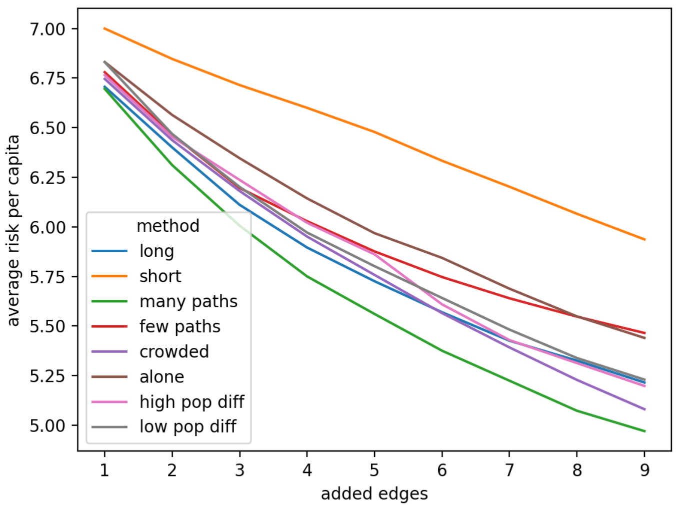

Figure 18: Average risk per capita for graph size 25

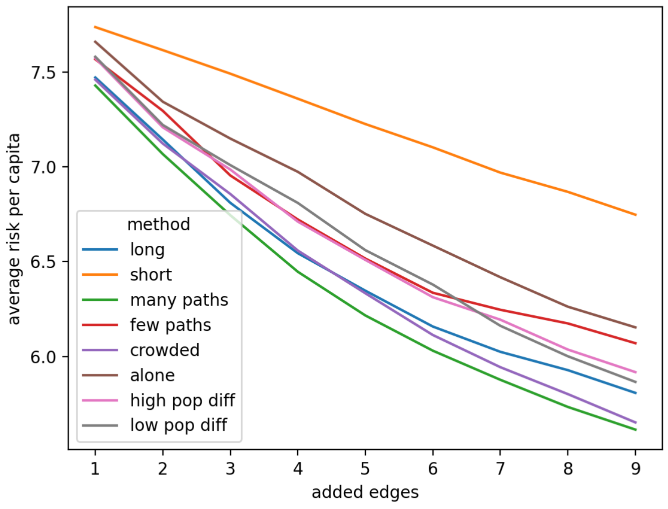

Figure 19: Average risk per capita for graph size 25

data we will estimate the data with a curve, using least-squares fitting to find the best parameters for a curve to fit the data. Looking at the plots for the first set of data, we see risk per capita lowering slower once more edges are added. An exponential curve $f(x) = ae^{-bx} + c$ has this property (where $b > 0$). This class of curves can have this slight bend upwards, making it a good choice. Fitting the parameters a, b, and c to the data for each algo-

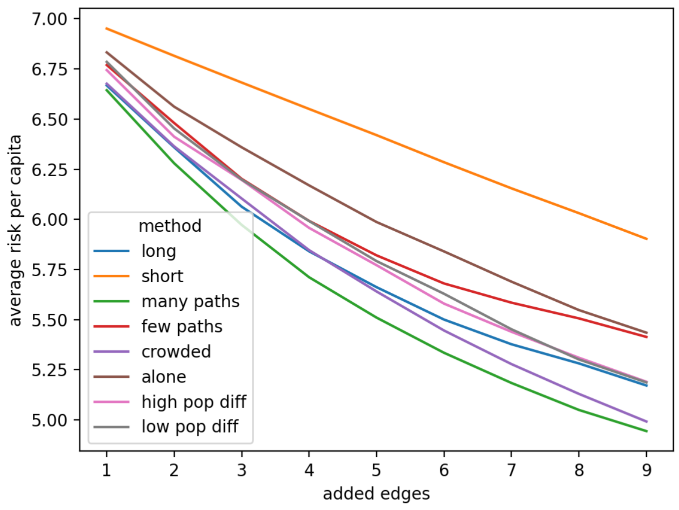

Figure 20: Average risk per capita for graph sizes 20-30

rithm for the graph sizes 20, 25, and 30 gives us the plots 21 - 23. A plot was also made using the data for all sizes in plot 24.

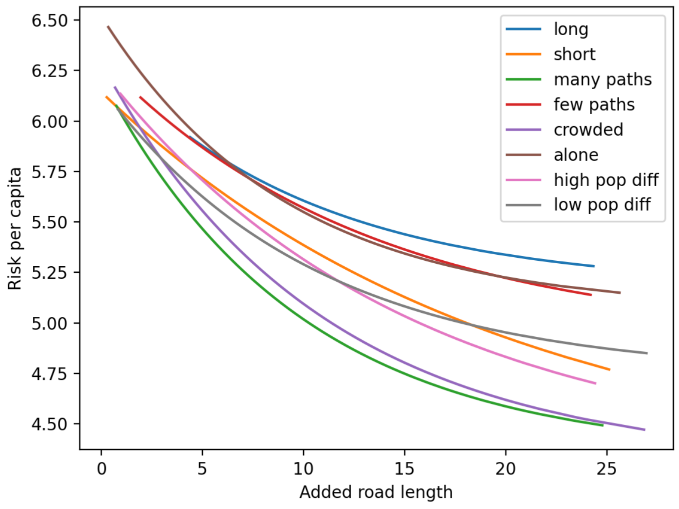

Figure 21: Estimated risk per capita for graph size 20

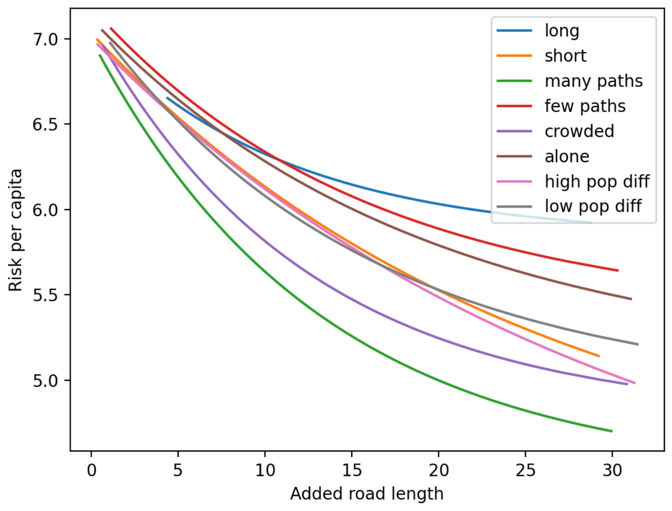

Figure 22: estimated risk per capita for graph size 25

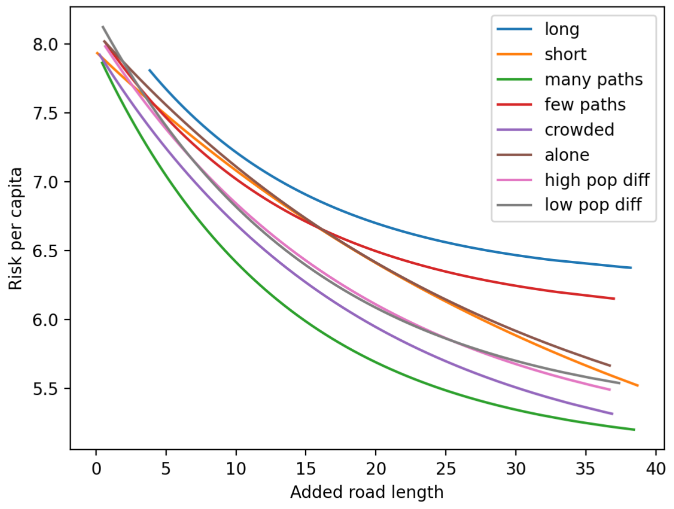

Figure 23: Estimated risk per capita for graph size 30

7 Discussion

We have two sets of data, one where the network building algorithms were limited by the number of edges that could be added, and one where they were limited by the total road length that could be added. The latter is more accurate for our purposes of determining algorithms that are efficient

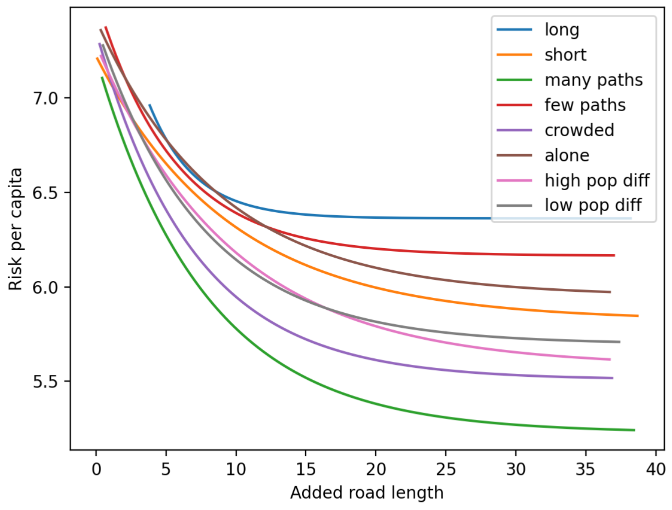

Figure 24: Estimated risk per capita for graph size 20-30

for the road space used, but it is still worth examining the results of the first set.

7.1 Data set 1

In plots 17-20 we see a clear outlier in our methods: **short**, the method that prioritizes adding short roads to our network. It is markedly worse at lowering risk, no matter how many edges are added. Conversely, the **long** method is third or fourth best in the plots. This implies that long roads lower risk more than short ones. It can be rationalized that short roads only alter an existing path slightly, while a long one can let some cars completely bypass their previous route for a more direct path. However this does not necessarily mean that long roads are more *efficient* than short ones, since the total road length added for a fixed number of edges is as different as they could be with these two methods.

For the other methods we can see that **many paths** is better than **few paths**, which makes since if you consider that having few paths cross a point means few people will change their path if a new road is added there. For the methods **crowded** and **alone** we see that connecting high population areas lowers risk better than connecting low population areas. Since high population areas with have high travel between them it makes sense that letting those people have a more direct route lowers risk. Lastly for **high population difference** and **low population difference** there is almost no difference between the methods, even though they will be adding completely different edges. This seems to imply that the difference in population between two areas does not matter compared to the total population.

There are two clear winning methods in this set of data and it is **crowded** and **many paths**. They are lower risk than all of the others after 6 edges added, with **many paths** staying lowest risk the whole way through in all plots. Notably, these two methods are independent of each other. Given a set of points, the Delaunay triangulation and its minimal spanning tree is uniquely determined (barring degenerate cases), and the values for the **many paths** algorithm are set, with no input from the population values of the points. Given that same set of points, the populations are independently generated, and is the only input for the **crowded** algorithm, with no input from the positions of the points.

7.2 Data set 2

The second set of data is more useful, since we can plot it by road length added rather than edges added. We have to approximate our data with a curve, and this produces curves in plots 21-23 that looks similar to the plots for our other set of data, but not for 24 that plots for all graph sizes.

The results in this set are relatively similar to the previous. The standout difference is **short** and **long**. In the other set, **long** was markedly better, but here, it is the opposite. **long** is by far the worst performing method. This time it is likely for the opposite reason from before; adding long roads adds the most length to the network, likely connecting points far from the center, which makes the road unlikely to be traveled. **Short** performs better than only **few paths** and **alone**, which were bad methods in the other set as well.

The methods **high population difference** and **low population difference** have similar results in figure 23, but in figure 21 and 22 **high population difference** performs better after much road length is added. That this difference is only once some edges have been added suggests that a strategy that adds edges of middling population difference might be better than picking from either extremes.

The best performing method by far is **many paths**, even more so than in the other set of data. We can think of this method as connecting choke points to each other, which after a few roads are added, is likely to create paths around them. It is also possible that this algorithm is connecting two points next to the choke point, bypassing the choke point for any cars going through.

Plot 24 has a noticeably more extreme curve than the others. When generating data for larger graph sizes, the more road length each algorithm must add before terminating, we see that in plot 21, the x-axis stretches from 0 to 25, while 23 goes all the way to 40. This means we have more data points for these lower values, making 24 have an extreme curve that tries to fit the early values at the expense of not fitting later ones as well. This makes plot 24 a misrepresentative estimation, but the most important quality of these plots - the ordering of the algorithms - is still intact.

7.3 Conclusion and improvements

Throughout all our data, we see two standout algorithms, **many paths** and **crowded**. **Many paths** gives the lowest risk network both for early iterations and late ones. They both lower the risk per capita in a network in different ways. **Many paths** lowers it by connecting points that are central to the network, ones that many cars have to travel through, and in turn make journeys that travel through them faster, lowering the amount of points visited by each car. **Crowded** solves it by connecting multiple high population areas, which will have many cars wanting to travel between them. These algorithms being at the top may not be very surprising, but lends credence to the underlying model being sound.

This makes the other results all the more important. The algorithms **high population difference** and **low population difference** had very similar results to each other, implying that the difference in population of two places matters less when connecting them than the total population compared to surrounding places.

An assumption made in our model is that expanding a road does little to lower the risk of accidents. That is why we don't allow multiple edges between points, and why roads have no carrying capacity. If a traffic official was to decide on what roadwork to order, and the assumption that expanding a road does not noticeably lower risk of accidents, the best choice is to go for short roads that circumvent central and high population areas in the network. This gives drivers options for their journeys, so choke points are less prevalent.

An improvement that could be made for this model optimizing the code. With faster code we could likely generate larger graphs and get more data to analyze. As is the code is not very optimized, and better implementations of the different algorithms and data management could go a long way. Another thing that could be done is using more granular algorithms, instead of only taking the most or least populous edges, including algorithms that take the $25\%, 50\%$, and 75% quantile edge could give us better data. We could also use an iterative approach for determining the algorithm itself, by making it a genetic algorithm instead.

8 References

Cheng, Siu-Wing, Dey, T. K., & Shewchuk, J. (2013). *Delaunay Mesh Generation*. p. 31-33, 59-61

M. I. Shamos (1978). *Computational Geometry*. Yale University.

R. Grimaldi. (2004). *Discrete and Combinatorial Mathematics - An Applied Introduction*. 5th ed., Pearson. p. 639, 631.

Viter, Antonii (2024). *Triangulation Algorithm in Python*.
https://github.com/AntoniiViter/Delaunay-Triangulation

Wikipedia (2025). *Delaunay triangulation*.
https://en.wikipedia.org/wiki/Delaunay_triangulation