A Heuristic for The Optimal Network Design Problem with Steiner Points

Siddhartha Lewis-Hayre

April 2022

1 Introduction

The builder of a road network has a lot of flexibility. A road can travel from almost any place to almost any other place and in doing so, can take almost any route. If a road must travel from a one city to another city, however, usually the fastest route for the road will be about a straight shot from the first city to the second city depending on terrain considerations. On the other hand, if we have many roads connecting many different cities it may be too expensive to build roads directly linking every pair of cities. Therefore, in the best road network, the best route between one city and another city may go through a third city, or it may merge and fork with other routes along its journey. Wherever roads can be built, roads might merge and fork. In this way, building a road network is a problem of choosing optimal paths where paths can pass through almost any point on the map that they desire. Our goal is to find optimal road networks by considering a road network as a set of curves without restrictions and then choosing the curves to make a road network that does not use too much road but also allows quick travel between any pair of cities.

Past research has shown how to solve or how to approximate many different problems about minimum cost road networks. One problem that closely resembles our problem is the Optimal Network Design Problem. This problem is to find the graph that minimizes the sum of the lengths of all pairs of shortest paths given a budget for the total length of edges. Dionne and Florian (1979) survey the different optimal and approximate algorithms and compare their performances.[1] They give examples of good approximations that run quickly.

The shortcoming in this analysis, is that all roads in these road networks begin and end at a point in the set of points that the roads are trying to connect. Roads are merely an edge in a graph with a predetermined vertex set. Therefore if the predetermined vertex set is a set of cities, then all roads must begin and end in cities. When considering traffic within the city and how to build roads in urban areas to minimize congestion, this assumption makes sense. If we want to connect cities, however, we need to consider road networks that allow roads to merge and diverge wherever they want. In other words we want to allow our inputted set of vertices to be a subset of all the vertices in our final graph.

Once we give roads this flexibility, our problem begins to look like the Steiner Tree Problem. The Steiner Tree problem is to find the minimum spanning tree that spams an inputted set of vertices but allows for the graph to have a vertex set with vertices other than the inputted set of vertices. These vertices that are added to the graph in solving the problem are called Steiner Points. The Steiner Tree problem has many approximations. One of the first good approximations, Kuo et al. (1981) managed to approximate the optimal Steiner Tree within a factor of 2, and could guarantee better results for smaller graphs.[2] The idea of their algorithm is to build up the Steiner tree starting with a minimum spanning tree. The Steiner Tree Problem is important for deciding how to build internet networks where the optimal network uses as little cable as possible to connect a set of ports.

Our problem is a generalization of the Steiner Tree Problem. Instead of just worrying about the length of the network, we also want the the shortest path between any two cities to be short. Therefore we also care about the time it takes to travel in the network. If we value going from place to place quickly, we are no longer guaranteed that our optimal network is going to be a tree; therefore, our resulting network can be much more complicated.

More precisely, we want to find a drawing of a graph whose vertex set contains our original cities but can also contain Steiner points. We want this drawing of a graph to minimize the weighted sum of the length of the graph plus the sum of all pairs of shortest paths between cities. We weight the sum by a constant that represents how much it costs to build roads against the value of travellers' time.

To attack this problem we get away from considering our network as a set of points and lines that we add to our original vertex set to make a graph. Instead of using graph theory, we transform the problem into a problem of finding an optimal set of continuous curves in the plane. To do this, we treat each path between a pair of cities as a curve. We see since building roads has a cost, paths pull on one another with something akin to a gravitational force in attempts to share their route with another path. By pulling on one another, the paths can reduce the total length of the road network at the cost of making the travel time longer. Letting the roads pull on each other with the correct force of gravity gives us a heuristic to approximate the minimum cost road network with Steiner points.

2 The Problem

Firstly we represent our set of cities as a finite set in the plane. We call this set of cities, $P \subseteq \mathbb{R}^2$, where P is finite. Given this set of cities we want to define the set of drawings of graphs over which we want to minimize. We call this set of drawings the feasible set which we denote $\mathcal{A}$. The feasible set of drawings are all drawings in the plane of graphs that are connected and have a vertex set that contains P. Formally given that G is a drawing of a graph in the plane we have that:

$$\mathcal{A} = \{G = (V, E) : P \subseteq V \subseteq \mathbb{R}^2, G \text{ is connected}\}$$

Since $V \subseteq \mathbb{R}^2$, we can pin down our drawing G just by the vertex set V and the edge set E. Given such a drawing of a graph in the plane, $G \in \mathcal{A}$, define $d : V \times V \to \mathbb{R}$. Given $v, v' \in V$ let $d(v, v')$ be the euclidean distance between vertex v and vertex v', along the drawing of the graph, G. This allows us to define the time cost of the drawing, G, as the sum of the lengths of paths between pairs of nodes in P.

Definition 1: The *time cost* of the drawing $G = (V, E)$ is:

$$T(G) = \sum_{p \neq p' \in P} d(p, p')$$

Next we can define the road cost of the drawing, G, as the sum of edge lengths.

Definition 2: The *road cost* of the drawing $G = (V, E)$ is:

$$L(G) = \sum_{(v,v') \in E} \|v - v'\|$$

Now we see for each drawing G, we can define the total cost of each drawing as the sum of the time cost and the road cost of the graph, weighted by some parameter $\alpha > 0$.

Definition 3: The *total cost* of the drawing $G = (V, E)$ is:

$$V(G) = T(G) + \alpha L(G)$$

The problem is as follows. Given an input of a set of finite points $P \subseteq \mathbb{R}^2$ and a parameter determining the relative cost of time and roads, $\alpha > 0$, determine the feasible drawing that minimizes this total cost. More precisely we want to find the drawing $G \in \mathcal{A}$ that minimizes $V(G)$.

3 Existence

Now we want a way to classify different feasible drawings. We do this by fixing a drawing $G = (V, E)$ as above. We define $X = V \setminus P$. We see that $X \subseteq \mathbb{R}^2$, but for our classification we are only worried about which vertices each path goes through and not the actual location of vertices in the plane. Therefore define $\tilde{X}$ where each element of $\tilde{X}$ corresponds to a point in X, but we do not require $\tilde{X} \subseteq \mathbb{R}^2$. In this way $\tilde{X}$ represents a subset of a vertex set of a graph not a subset of a vertex set of a drawing of a graph in the plane.

Furthermore, for each pair of distinct points $p_i, p_j \in P$ we fix a shortest path between them. Let $\mathcal{P}$ be the set of all paths between pairs of points in P, where a path in $\mathcal{P}$ is a sequence $\{p_i, v_1, ..., v_n, p_j\}$ where $v_i \in \tilde{X} \cup P$ for $i = 1, ..., n$.

We see that given P if we fix $(\tilde{X}, \mathcal{P})$ then we refer to a set of drawings in the plane where our vertex set can lie anywhere in the plane, but our paths between points in P have a specific sequence of vertices they must pass through. We define a topology as a tuple $(\tilde{X}, \mathcal{P})$.

Definition 4: Given a finite set of points $P \subseteq \mathbb{R}^2$ a *topology* is a tuple $(\tilde{X}, \mathcal{P})$ where $\tilde{X}$ is a finite set of nodes and $\mathcal{P}$ specifies a path for each pair of points in P.

We note that for some drawings of $(\tilde{X}, \mathcal{P})$, the paths specified by $\mathcal{P}$ may not correspond to the shortest paths in our drawing. In these cases, a different topology will do better to describe this drawing. In this way, each drawing of a graph does have a unique topology associated with it, but as shown above, we can always find a topology to characterize the set of shortest paths of a given drawing.

We see that fixing a topology can induce an edge set from $\tilde{X}$. The set of edges E is all the pairs of nodes that appear next to each other in some path between points in P. If we then fix a set $X \subseteq \mathbb{R}^2$ that corresponds to $\tilde{X}$, then we have a drawing $G = (V, E)$ where $V = X \cup P$.

The purpose of categorizing drawings by topology is that we can show that there are a finite number of topologies and that given a topology, we can always find an optimal vertex set $X^* \subseteq \mathbb{R}^2$ for that topology. In other words we can find a vertex set $X^* \subseteq \mathbb{R}^2$ that induces a drawing with total cost less than any other drawing with our fixed topology.

Proposition 1: Among drawings G with topology $(\tilde{X}, \mathcal{P})$, there is a unique drawing G^* that minimizes total cost.

Proof:

As we saw above, if we fix $X \subseteq \mathbb{R}^2$ where each element in X corresponds to an element in $\tilde{X}$, then we can set $V = P \cup X$. Furthermore we can find our edge set E to be such that $(v, v') \in E$ if and only if there is some path in $\mathcal{P}$, such that v and v' are adjacent. Now we define the frequency of each edge by the following function. Let $f : E \to \mathbb{N}$, be such that given $(v, v') \in E$ we have $f((v, v'))$ is the number of paths in $\mathcal{P}$ in which v, v' are adjacent.

We note that without loss of generality we can assume that v appears only once in each path as if v appeared more than once, no matter our set X, it would be faster to skip to the second time v appears when we first arrive at v.

Now we can write the total cost of the drawing $G = (V, E)$ as:

$$V(G) = T(G) + \alpha L(G) = \sum_{p \neq p' \in P} d(p, p') + \alpha \sum_{(v,v') \in E} \|v - v'\|$$

$$= \sum_{(v,v') \in E} f((v, v') \cdot \|v - v'\| + \alpha \sum_{(v,v') \in E} \|v - v'\|$$

$$= \sum_{(v,v')\in E} (f((v,v') + \alpha)\|v - v'\|$$

We see since $\|\cdot\|$ is a norm and in the Euclidean case it is a norm with a uniformly convex unit ball, then we have that $V(G)$ is strictly convex in X. Therefore we have that there exists a unique drawing G^* with vertex set $V = P \cup X^*$ that minimizes total cost among drawings with topology $(\tilde{X}, \mathcal{P})\square$

Now we argue that the number of topologies is finite. Let $K = \binom{|P|}{2}$, the number of paths between points in P. Firstly we know that $|X| \leq \binom{K}{2}$ because each node in X must be the confluence or fork of at least two paths and without loss of generality, paths cannot fork or merge at more than one point as otherwise we could let each path take the shorter of the two routes between the two locations at which they meet. This is a very loose upper bound on the size of X. Furthermore, given X, we have a finite number of ways to choose paths $\mathcal{P}$.

Together these observations mean that a drawing minimizing total cost exists because we can limit our search to the best drawing in each topology and then take the best drawing out of the finite number of topologies. Furthermore we have discovered that if we can determine the optimal topology $(\tilde{X}, \mathcal{P})$ then we can find the drawing G^* that minimizes total cost. Determining that optimal topology is very difficult. We will do our best to approximate the best topology. The rest of the paper deals describes a heuristic to guess the best topology.

4 Restating the Problem

The first goal is to transform the problem from a problem about graphs to a problem about continuous functions. To do this, think of each path between points as a continuous function. Given points $p, p' \in P$ we can define $x : [0,1] \to \mathbb{R}^2$ to be the path between p and p' in G. We see this function is continuous as a path of a graph drawn in the plane is a sequence of connected curves in the plane which are connected by vertices.

If our set of points is $|P| = N$, then we have $\binom{N}{2}$ paths between points in P, so we can describe the drawing of our graph by $K \coloneqq \binom{N}{2}$ continuous functions. Therefore let $x_1, ..., x_K$ be the continuous functions representing the paths in our drawing G. Given $k \in \{1, 2, ...K\}$ define $A_{>k}$ to be all points in paths labeled greater than k. More precisely,

$$A_{>k} = \{x_i(t) : t \in [0,1], i \in \{k+1, k+2, ..., K\}\}$$

Note we have that $A_K = \emptyset$. With this definition we can consider the following proposition.

Proposition 2:

$$T(G) = \sum_{k=1}^{K} \int_{[0,1]} ||\dot{x}_k(t)|| \, dt$$

$$L(G) = \sum_{k=1}^{K} \int_{[0,1]} ||\dot{x}_k(t)|| \cdot \chi_k(x_k(t)) \, dt$$

where we define $\chi_k : \mathbb{R}^2 \to \mathbb{R}$ as:

$$\chi_k(z) = \begin{cases} 1 & \rho(z, A_{>k}) > 0 \\ 0 & \rho(z, A_{>k}) = 0 \end{cases}$$

given $z \in \mathbb{R}^2$ where ρ is the Euclidean distance from a point to a set and $p(z, \emptyset) \coloneqq \infty$.

Proof:

Firstly consider our proposed value for $T(G)$. We have that $x_k(t)$ is differentiable when $x_k(t) \notin V$ as edges are straight lines in the plane. In other words x_k is differentiable at all but finitely many points. Therefore the integral

$$\int_{[0,1]} ||\dot{x}_k(t)|| \, dt$$

is the length of path x_k. We conclude that summing over paths gives $T(G)$.

Now consider our proposed value for $L(G)$. Fix $e = (v, v') \in E$. Let $x_k^{-1}(e)$ be the set of t such that the path k is on edge e. More precisely:

$$x_k^{-1}(e) = \{t : \exists \lambda \in (0,1) \text{ s.t. } x_k(t) = \lambda v + (1-\lambda)v'\}$$

We want to prove that

$$\sum_{k=1}^{K} \int_{x_k^{-1}(e)} ||\dot{x}_k(t)|| \cdot \chi_k(x_k(t)) \, dt = ||v - v'||$$

Let $K_e \subseteq K$ be the set of paths that use edge e. In other words:

$$K_e = \{k \in \{1, 2, ..., K\} : x_k^{-1}(e) \neq \emptyset\}$$

Now we see if $\overline{k} = \max K_e$ then

$$\chi_{\overline{k}}(x_{\overline{k}}(t)) > 0$$

for all $t \in x_{\overline{k}}^{-1}(e)$. But if $k \in K_e$ and $k \neq \overline{k}$ then we have:

$$\chi_{\overline{k}}(x_{\overline{k}}(t)) = 0$$

for all $t \in x_k^{-1}(e)$. Finally if $k \notin K_e$ then we have $t \in x_k^{-1}(e) = \emptyset$. Combining all these facts we have that:

$$\sum_{k=1}^{K} \int_{x_k^{-1}(e)} ||\dot{x}_k(t)|| \cdot \chi_k(x_k(t))\, dt$$

$$= \sum_{k \in K_e} \int_{x_k^{-1}(e)} ||\dot{x}_k(t)|| \cdot \chi_k(x_k(t))\, dt$$

$$= \int_{x_{\overline{k}}^{-1}(e)} ||\dot{x}_{\overline{k}}(t)|| \cdot \chi_{\overline{k}}(x_{\overline{k}}(t))$$

$$= ||v - v'||$$

We conclude by summing over edges that

$$L(G) = \sum_{(v,v') \in E} ||v - v'|| = \sum_{k=1}^{K} \int_{[0,1]} ||\dot{x}_k(t)|| \cdot \chi_k(x_k(t))\, dt \; \square$$

From this proposition we can write:

$$V(G) = T(G) + \alpha L(G) = \sum_{k=1}^{K} \int_{[0,1]} ||\dot{x}_k(t)|| \left[1 + \alpha \chi_k(x_k(t))\right]\, dt$$

5 Restriction to Paths

Now that we have restated our problem we can more easily describe some properties of our paths $x_1, ..., x_K$. Firstly we have a condition that bounds how wild each path can be.

Proposition 3: Let x be a path in an optimal drawing G. Given $t_0, t_f \in [0,1]$ then we have that:

$$\int_{t_0}^{t_f} \|\dot{x}(t)\|\, dt \leq (1+\alpha)\|x(t_f) - x(t_0)\|$$

Proof:

Assume to the contrary that

$$\int_{t_0}^{t_f} \|\dot{x}(t)\|\, dt > (1+\alpha)\|x(t_f) - x(t_0)\|$$

Then define x' so that $x'(t) = x(t)$ if $t \in [0, t_0] \cup [t_f, 1]$ and if $t \in [t_0, t_f]$ then define:

$$x'(t) = x(t_0) + \frac{t - t_0}{t_f - t_0} \cdot (x(t_f) - x(t_0))$$

Then we see that if we replace x with x' to get drawing G' we have that:

$$V(G') \leq V(G) - \int_{t_0}^{t_f} \|\dot{x}(t)\|\, dt + (1+\alpha)\|x'(t_f) - x'(t_0)\| < V(G)$$

This contradicts that G is an optimal drawing $\square$

This proposition in turn allows us to prove that a path x_k must lie in a region surrounding its start and end points p^0 and p^f. Given x_k we define the ball of x to be:

$$B_k = \{p^0 + t_1 p + t_2 p^{\perp} : t_1 \in \text{supp}(b(\cdot, \|p\|, \alpha)),\ |t_2| \leq b(t_1, \|p\|, \alpha)\}$$

where we let $b : \mathbb{R}^3 \to \mathbb{R}$ be:

$$b(t, d, \alpha) = \frac{1}{2} d[(1+\alpha)^2 - (1 - 2t + 2t^2 + |t||1-t|)]^{\frac{1}{2}}$$

and where $p = p^f - p^0$ and $p^{\perp}$ is unit vector perpendicular to p that is rotated 90 degrees counterclockwise from p (although the direction does not matter). More precisely, $p^{\perp} = \Pi(\frac{p}{||p||})$ where Π is the rotation in the plane by 90 degrees counterclockwise. We want to show that the x_k lies within B_k.

Proposition 4: Let x_k be a path in an optimal drawing G. For all $t \in [0,1]$ we have that $x_k(t) \in B_k$.

Proof:

Fix $t_1, t_2 \in \mathbb{R}$. Then we see that we must have if there exists $t \in [0,1]$ such that $x_k(t) = p_0 + t_1 p + t_2 p^{\perp}$ then:

$$((\|p\|(1-t_1))^2 + t_2^2)^{\frac{1}{2}} + (\|p\|^2 t_1^2 + t_2^2)^{\frac{1}{2}} \leq \|p\|(1+\alpha)$$

This is because the left side is the minimum length of a path that goes through $x(t)$ and the right hand side is the maximum the path can cost. So by applying proposition 2 to $t_0 = 0$ and $t_f = 1$ we, get this inequality. Since both sides of the equality are non-negative we square both sides of the inequality to get:

$$(\|p\|^2(1 - 2t_1 + 2t_1^2) + 2t_2^2 + [\|p\|^2(1-t_1)^2 + t_2^2)(\|p\|^2 t_1^2 + t_2^2)]^{\frac{1}{2}} \leq (\|p\|(1+\alpha))^2$$

Now we see we can drop terms with t_2 within the square root and only make the expression on the left smaller. Therefore we get:

$$(\|p\|^2(1 - 2t_1 + 2t_1^2) + 2t_2^2 + \|p\||1-t_1||t_1| \leq (\|p\|(1+\alpha))^2$$

Finally we solve for t_2 and see that:

$$|t_2| \leq \frac{1}{2}\|p\|[(1+\alpha)^2 - (1 - 2t_1 + 2t_1^2 + |t_1||1-t_1|)]^{\frac{1}{2}}$$

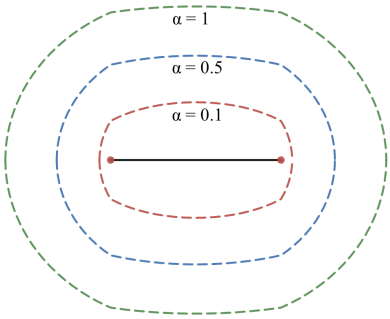

Figure 1: Balls around a path for different α

In other words $x_k(t) \in B_k$ for all $t \in [0, 1]\square$

We have seen that the each path is constrained to a region nearby its start and end coordinates. From this, we see that our sets that consider nearby roads $A_{>k}$ are actually larger than they need to be. In particular we can restrict $A_{>k}$ to only consider paths that can cut through B_k. In other words we can drop paths from $A_{>k}$ whose balls do not overlap with B_k. This is because in an optimal graph ignoring these paths will not change χ_k since these paths do cannot contribute to the minimum distance from our path k to $A_{>k}$.

Now we want to consider the interaction between two separate paths and consider when two paths can merge and diverge by applying these results. As we already argued, we can assume that a pair of paths do not merge or diverge from each other more than once in the optimal drawing. This is because if they were to merge or diverge more than once, between the points where the two paths share a road, we could weakly improve our drawing by letting each path take the shorter of the two routes. In other words the set on which any two paths overlap is connected.

In particular this greatly restricts how paths can behave if they share endpoints. We see that all the paths that originate at one point can be drawn as a tree where the root of the tree is the origin node and all other leaf nodes are the other points in P. Furthermore the paths to these other points in P have to be ordered by the angle of the vector between each destination and the origin point.

Furthermore if we have any two arbitrary paths say, x_k that has endpoints p_k^0 and p_k^f and x_l that has endpoints p_l^0 and p_l^f, then we can say the two paths cannot cross unless the vectors $p_k^f - p_k^0$ and $p_l^f - p_l^0$ cross. By paths crossing we mean that on either side of the connected component on which the two paths overlap, the paths switch what side of the other path they are on.

Furthermore, we have that if $p_l^f - p_l^0$ intersects with B_k but is not contained in B_k, then we can restrict the area which path x_k can travel as it must lie to

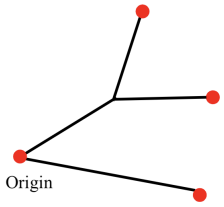

Figure 2: Example of tree formed by paths from an origin point

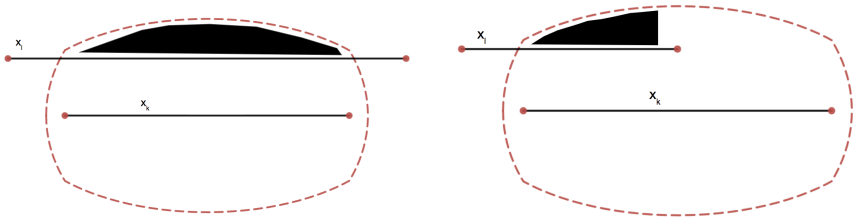

Figure 3: Examples of paths restricting one another. In particular path k cannot enter black region

one side of x_l. This is because otherwise x_k would be forced to cross x_l twice which is not possible in an optimal drawing. This is represented in Figure 3.

We see these restrictions make it much easier to determine which paths might lie next to which other paths. For example, we see if a path is surrounded by other paths that are sufficiently long on all sides, then we see that our path must lie within the bounds of these surrounding paths. Once we propose our heuristic, we will see that these restrictions narrow the space of possibilities for optimal drawings thereby making our heuristic look more reasonable.

6 Discrete Approximation

Now we want to approximate the total cost by a convex function. There are three difficulties in this. Firstly we see that that χ_k is an indicator function that depends on $\rho(z, A_{>k})$ which is a minimum of distances. This creates a very bumpy function that is not convex. All other parts of the problem can be approximated by convex functions, but this part of the problem seemingly cannot be approximated by convex functions. Although the other two forthcoming problems can be overcome, there is nothing we can do here. So instead we have to settle for only doing what is locally best. As a result our minimization does not minimize over all topologies but just some. Our simplification is that we

replace the minimum distance with the distance to a part of another path that we guess approximately minimizes distance. In other words we approximate $\rho(z, A_{>k})$ by replacing it with an element of $A_{>k}$.

The second problem is that χ_k cannot be well approximated by a convex function on its entire domain since the function takes on values of zero and has a limit of one as its argument grows. We will deal with this problem by repeatedly minimizing our approximation to the indicator function. Each new minimization problem differs from the last in that we make our approximating convex function look more like the indicator function, while ensure the function is still convex on the domain over which we minimize.

The final problem is that to mimic our objective function we want to multiply the arclength of our curve at any given point by this indicator function. Doing this will lose convexity. Therefore we approximate arclength by a fixed parameter.

Now we proceed to try and construct this function. To begin we introduce some notation that will help us discretized the continuous function we described in the previous section.

We approximate paths x_k by a piecewise function. To do this we represent each path by a set of vectors in the plane. Connecting adjacent vectors gives us a discretized path. More formally, fix $I \in \mathbb{N}$. For each path $k \in \{1, 2, ..., K\}$ define:

$$\hat{x}_k = \{x_k^1, x_k^2, ..., x_k^I\} \text{ s.t. } x_k^i \in \mathbb{R}^2$$

Now we restrict the points in this vector so that they lie perpendicularly above or below the line segment that connects the start and the end of path k. By start and end of a path, we just mean the two endpoints, which is the start and which is the end is not important. Furthermore, we require the points be evenly spaced. Formally we require that if path k starts at point p_k^0 and ends at point p_k^f then there exists $t_k^i \in \mathbb{R}^I$ such that:

$$x_k^i = p_k^0 + t_k^i \cdot p^\perp + \frac{i}{I+1} \cdot p, \ i = 1, 2, ..., I$$

where $p = p_k^f - p_k^0$, $p^\perp = \Pi(\frac{p}{||p||})$, and Π is the rotation in the plane by 90 degrees counterclockwise (although direction does not matter much as it only introduces a minus sign in t_i). If we connect adjacent points in $\hat{x}_k$ by a line we get a piecewise function that connects the start and end points of path k. Given the form we require of x_k^i we can equivalently represent this path with I real numbers, t_k^i, which represents how far away from the line segment between p_k^0 and p_k^f the point x_k^i lies. Figure 4 shows this process.

We see we have now restricted our paths so that each point in the path can be represented by one variable instead of two. In this way, we can represent each discretized path, which is a set of points in the plane, as a set of points in the real line. Each value in the real line represents how far away the corresponding point in the plane is from the line segment connecting the start and end of a path. Now we establish notation to easily transform between this vector of

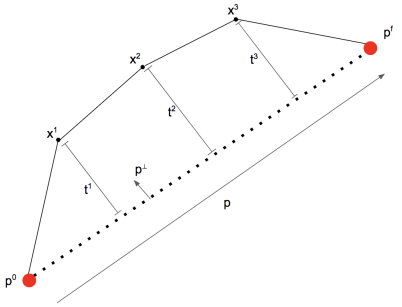

Figure 4: Discrete approximation of a path

points in the plane and this vector of points in the real line. Firstly we define T to be the map that lets us transform between the I variables on the real line that represent the height of our function and the points in the plane that make up our discretized function. Let $T_k^i : \mathbb{R} \to \mathbb{R}^2$ be such that:

$$T_k^i(t_k^i) = x_k^i = p_k^0 + t_k^i \cdot p^\perp + \frac{i}{I+1} \cdot p$$

Then define:

$$t_k = \{t_k^1, t_k^2, ..., t_k^I\}$$

$$\hat{x} = \{\hat{x}_1, \hat{x}_2, ..., \hat{x}_K\}$$

$$t = \{t_1, t_2, ..., t_K\}$$

So we can define $T_k : \mathbb{R}^I \to \mathbb{R}^{2I}$ to be such that:

$$T_k(t_k) = \hat{x}_k$$

and we can define $T : \mathbb{R}^{IK} \to \mathbb{R}^{2IK}$ to be such that:

$$T(t) = \hat{x}$$

From now on we write x_k^i to refer to $T_k^i(t_k^i)$. Also we let $x_k^0 = p_k^0$ and $x_k^{I+1} = p_k^f$. This establishes more compact notation to represent transforming between the two representations of our discrete function.

Next we want to come up with a discrete version of $A_{>k}$ for each term x_k^i. To do so we define for each $k \in \{1, 2, ..., K\}$ and each $I \in \{1, 2, ..., I\}$:

$$\hat{A}_k^i(\hat{x}) = \{x_{k'}^{i'} : k' > k, i' \in \{1, 2, ..., I\}\}$$

Again note we can have $\hat{A}_k^i(\hat{x}) = \emptyset$ in which case we define the distance $\rho(x_k^i, \hat{A}_k^i(\hat{x}))$ to be infinity.

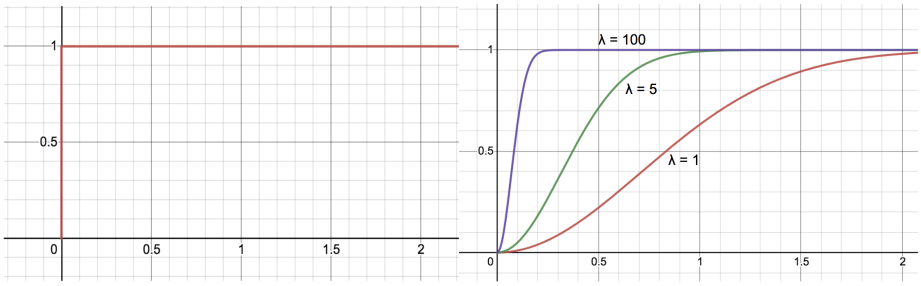

Figure 5: Left graph shows indicator function. Right graph shows exponential approximation to indicator function

Next fix $\theta > 0$. This will represent an approximation of the length of each segment of our path. As described above, we make this approximation to preserve convexity. Finally fix $\lambda \in \mathbb{R}^{IK}$. Now we can define $\tilde{V} : \mathbb{R}^{IK} \to \mathbb{R}$ to be:

$$\tilde{V}_{\lambda,\theta}(t) = \sum_{k=1}^{K} \sum_{i=1}^{I+1} ||x_k^i - x_k^{i-1}|| \cdot (1+\alpha)$$

$$- \sum_{k=1}^{K} \sum_{i=1}^{I} (1 + \frac{1}{2} \mathbb{1}_{\{i=1, i=I\}}) \cdot \alpha \cdot \theta \cdot exp(-\lambda_k^i \cdot \rho(x_k^i, \hat{A}_k^i(\hat{x}))^2)$$

We see that $\tilde{V}$ approximates the cost of our network. Given a drawing of a graph, if we convert its paths into discreteized functions and plug their values into this function, we should get an output value of $\tilde{V}$ that is approximately the total cost of our drawing. This is because we see the first term approximates the total time of the graph plus the cost of all the roads if no two road were to overlap. The second term reduces the cost by giving a credit to roads that overlap. In this way our function looks much like the continuous version of the cost of a graph established by proposition 2.

The two main choices in this approximation are that we approximated χ_k using the function $e^{-\lambda x^2}$ and we replaced $||x_k^i - x_k^{i-1}||$ with θ. As far as our approximation of the indicator function, we see as the coefficient λ in the exponent grows, the exponential function begins to look more like an indicator function. This is shown in figure 5. The left graph plots our indicator function χ_k in terms of $\rho(x_k^i, \hat{A}_k^i(\hat{x}))$, the right graph plots our exponential $exp(-\lambda \rho(x_k^i, \hat{A}_k^i(\hat{x}))^2)$ as a function of $\rho(x_k^i, \hat{A}_k^i(\hat{x}))$ for different values of λ.

As discussed before we see that $\rho(x_k^i, \hat{A}_k^i(\hat{x}))$ is not convex so this function is not convex. To deal with this problem we will have to replace $\hat{A}_k^i(\hat{x})$ with one of its elements, as we will do in the next section. The next section outlines a heuristic to find a low cost network and will deal this convexity problem.

7 Approximation Heuristic

The heuristic proposed in this section captures that paths get a benefit for sharing the same road. Paths are willing to make themselves slightly longer if it means they can share their route with another path, thereby saving road cost. Therefore, as we saw in our discrete approximation, we let the roads pull on each other.

More concretely, we start with an initial guess which is the drawing where all paths travel straight from their origin to their destination. We then minimize a convex function that pulls the paths towards each other. We then reapply the convex function to the result of the first optimization but increase the coefficient λ in the exponential. We repeat this process while increasing λ. Additionally, for each new minimization problem, we change θ to reflect the length of the paths in the result of the previous minimization problem. When points stop moving too much we stop the process. Then we have something that looks like a optimal drawing. Then we try to turn this set of paths that looks like an optimal drawing into a drawing with straight edges between nodes. This gives us a topology that we fix as our final topology. Our final graph is the best graph with this topology.

Now we give the specifics of this process. First we define the convex function we want to minimize. Given we are at step n of the algorithm, the input to this function is the previous optimal solution denoted $t_{n-1} \in \mathbb{R}^{IK}$ as well as λ and θ used at the previous step of the algorithm. We denote this previous exponent as λ_{n-1} and this previous slope as θ_{n-1}. When the algorithm starts we set $t_0 = 0$, in other words we make all paths straight. From this input we calculate the following parameters. We calculate $\lambda_n \in \mathbb{R}^{IK}$, $\theta_n \in \mathbb{R}^K$, and a function $f : [K] \times [I] \to [K] \times [I]$ where $(f_1(i,k), f_2(i,k))$ gives the indices of the closest element of $\hat{A}^i_k(\hat{x}))$ to x^j_i in the initial guess t_{n-1}. In other words we look for the closest element to x^i_k in $\hat{A}^i_k(\hat{x})$ when $\hat{x} = T(t_{n-1})$. We call $x^{f_2(i,k)}_{f_1(i,k)}$ the neighbor of x^i_k at step n.

Now we give more specificity as to how to determine λ_n, θ_n, and a function f from t_{n-1}, λ_{n-1}, and θ_{n-1}. If we are on the first repetition of the algorithm we set:

$$t_0 = 0$$

$$(\theta_0)_k = 0$$

and given that we have $\hat{x} = T(t_0)$, we set:

$$(\lambda_0)^i_k = \frac{s_1}{2 \cdot \|x^i_k - x^{f_2(k,i)}_{f_1(k,i)}\|^2}$$

where $0 < s_1 \leq 1$ is a parameter to the algorithm. Otherwise we have t_{n-1}, λ_{n-1}, and θ_{n-1} inputted. Set $\hat{x} = T(t_{n-1})$. Given these inputs we compute:

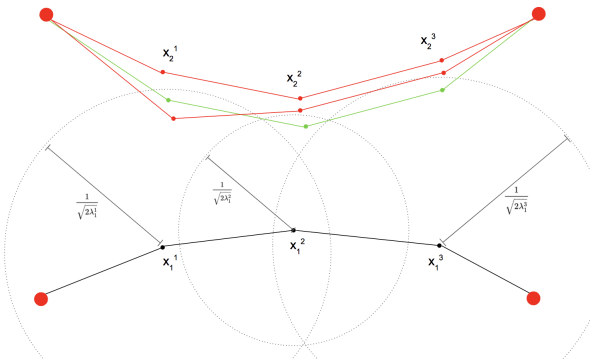

Figure 6: B_λ with 2 paths

$$(\theta_n)_k = \max\{(\theta_{n-1})_k, \sum_{i=1}^{I+1} ||x_k^i - x_k^{i-1}||\}$$

$$f(i,k) = (i',j') \text{ f.s. } x_{k'}^{i'} \in \text{argmin}_{x_{k'}^{i'} \in \hat{A}_k^i(\hat{x}))} \|x_k^i - x_{k'}^{i'}\|$$

$$(\lambda_n)_k^i = \min\{(\lambda_{n-1})_k^i + s_2, \frac{s_1}{2 \cdot \|x_k^i - x_{f_1(k,i)}^{f_2(k,i)}\|^2}\}$$

Where s_2 is some non-negative parameter determined as a maximum step size. Given λ, θ, f, we define our convex function $\hat{V} : \mathbb{R}^{IK} \to \mathbb{R}$ to be:

$$\hat{V}_{\lambda,\theta}(t) = \sum_{k=1}^{K} \sum_{i=1}^{I+1} ||x_k^i - x_k^{i-1}|| \cdot (1+\alpha)$$

$$-\sum_{k=1}^{K} \sum_{i=1}^{I} (1 + \frac{1}{2}\mathbb{1}_{\{i=1, i=I\}}) \cdot \alpha \cdot \theta_k^i \cdot exp(-\lambda_k^i \cdot \|x_k^i - x_{f_1(k,i)}^{f_2(k,i)}\|^2)$$

As we will see, this function is convex on the following convex set which we call a λ ball:

$$B_\lambda = \{t \in \mathbb{R}^{RI} : \hat{x} = T(t), \forall k \in [K], i \in [I],\ \|x_k^i - x_{f_1(k,i)}^{f_2(k,i)}\| \leq \frac{1}{\sqrt{2\lambda_k^i}}\}$$

At each step of the algorithm we restrict ourselves to finding the minimum on this domain. Figure 6 helps visualize this domain. Given a fixed path one, we consider how path two can look while keeping the network in B_λ. We see each point and its neighbor have to be close enough together. In figure 6, the green paths are options for path two that lie in B_λ. The red paths yield networks that do not lie in B_λ.

Now we can summarize this process in the following way:

1) Set $n = 1$, $t_0 = 0$, and set λ_0 and θ_0 as specified above

2) Repeat the following until $T(t_n) - T(t_{n-1})$ is small enough

a) Compute λ_n, θ_n, and f from t_{n-1} and λ_{n-1}. Then solve for

$$t_n = \text{argmin}_{t \in B_{\lambda_n}} \hat{V}_{\lambda_n, \theta_n}(t)$$

b) set index to be $n + 1$

Once this algorithm terminates we have something that looks visually like our ultimate solution. The problem is that two paths may travel very near each other but still use slightly different roads. This is because we cannot always get λ to grow as large as we want. Note in the limit that λ gets very large our exponential function $e^{-\lambda x^2}$ starts to look like an indicator function. To deal with the fact that two paths may run near each other without sharing the same road we do the following. For each point in our path, we find a little neighborhood. If this neighborhood includes other paths, then we count these paths as using the same road at that location. Furthermore, we look for parts of our paths where two paths diverge from one another and become farther apart than the radius of this neighborhood. At these locations we create a node. We make a list of nodes that each path goes through. This induces a topology $(\tilde{X}, \mathcal{P})$. Finally, given this topology, we find the drawing of this topology that minimizes cost.

8 Explanatory Example

Now that we have defined our algorithm let us consider running the algorithm on two paths that start and end at arbitrary points. For this example we fix those arbitrary endpoints as shown in step 0 in Figure 16. We see in this example our set P would have three nodes, the endpoints of the two paths. This would induce a network with three total paths but, in this example, we ignore the third path for simplicity.

We see that in the figures, red dots represent points in P. The smaller black dots represent points in $\hat{x}$. We start with a network with all straight lines. Then we solve an optimization problem and recalculate inputs λ, θ and f. We repeat this process until the graph does not change too much. Then we try and approximate our curves with a drawing of a graph with straight edges in the plane. The network we get from convex minimization seems to have one spot where edges merge, so we add a Steiner point there. Now we have our topology so we finish by finding the best drawing of this topology. We see that the minimization to find the best drawing of our fixed topology barely changes the drawing which suggests our algorithm did well to find curves to approximate the best network.

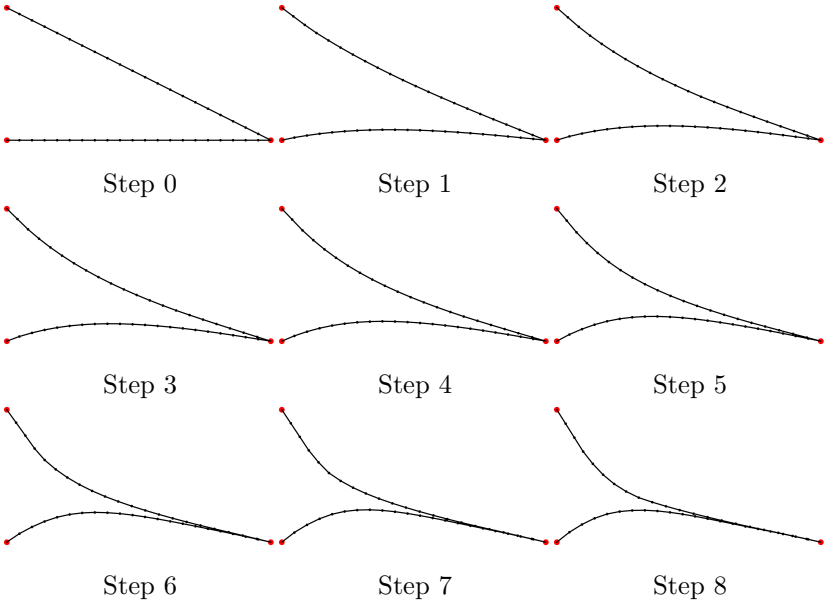

Figure 16: Networks at different steps of convex minimization process

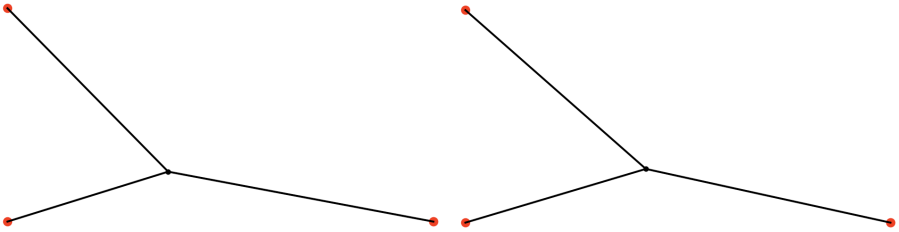

Figure 17: Drawing approximating curved network on left. Best network of this topology on right.

9 Discussion of Heuristic

The main choice we have made in our heuristic is to replace $\hat{A}_k^i(\hat{x})$ by a single variable that we determine from the initial guess at each repetition of the algorithm. This makes our function convex but limits our search to what would be a local minimum of our old function $\tilde{V}$. In effect this heuristic attempts to guess for each x_k^i which other variable $x_{k'}^{i'}$ we should be near in an optimal solution. If the heuristic guesses each variable x_k^i's neighbor correctly, the result of our heuristic should be close to the true optimal solution. We will use the next section to prove two results that shed light on this process.

Right now, this algorithm only lets the variable closest to x_k^i pull on it, but we could replace $x_{k'}^{i'}$ with some weighted sum of nearby variables. The closer the nearby variable at the start of the minimizing the higher weight it would get. This would allow the pull on x_k^i to not just come from the closest point but also from nearby areas.

As described above, the heuristic sets each point's neighbor to be the point closest to it. This might lead two points next to each other on the same line to have the same neighbor. Given that two points next to each other are constrained to live on a line perpendicular to the line segment connecting the origin and destination of the points' path, both of the points cannot both collapse onto their shared neighbor. This poses a problem as we might have points that choose neighbors which pull our points in directions that they cannot move. In order to fix this problem we could consider the following. Instead of choosing neighbors as the closest point, we want to incentive points to choose neighbors that lie close in the direction perpendicular to the line segment connecting the points origin and destination. Given path k with a line segment connecting its endpoints denoted p, given a perpendicular vector denoted $p^\perp$, and given a fixed $c > 1$, we want to determine our neighbors by the following minimization problem.

$$f(i,k) = (i',j') \text{ f.s. } x_{k'}^{i'} \in \text{argmin}_{x_{k'}^{i'} \in \hat{A}_k^i(\hat{x}))} \|x_k^i - x_{k'}^{i'}\|_c$$

where $\|\cdot\|_c$ is a norm on $\mathbb{R}^2$ defined by:

$$\|x\|_c = (x \cdot p)^2 + c(x \cdot p^\perp)^2$$

where the dot, $\cdot$, here refers to the dot product. In this way, now the unit ball is an oval that has its semi-major axis along $p^\perp$. Figure 18 shows how this norm is better for finding neighbors as if we use the normal Euclidean norm we will have that x_1^1's neighbor will be x_2^2.

Apart from choosing how to determine neighbors, we made choices on how to increase λ and θ. The algorithm forces λ to be non-decreasing. As we will see in the next section this means the sum of θ_k, which can be interpreted as the total length of paths, is non-decreasing. As such the algorithm is in effect testing longer and longer paths to see how long the paths should become. Therefore, to ensure smoothness of the algorithm we require each coordinate of θ_k to be increasing.

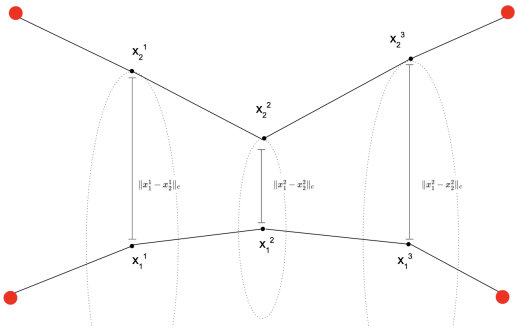

Figure 18: Finding neighbors using $\|\cdot\|_c$

Furthermore, we want λ to grow slowly so that points do not move too far so we can ensure that the point x_k^i's neighbor does not change too much during our minimization. Therefore we start with λ relatively low, as determined by s_1, and restrict how quickly it can grow with a maximum step size s_2. The problem is that if we set a maximum amount that λ can grow at each step then we might not be able to ultimately achieve a λ that is as large as if we did not limit its growth. Therefore there is a trade off in setting s_1, s_2. If they are large then the algorithm will do better to estimate the minimum at each step but it will not progress as far. This will be described more fully in the next section.

In the next section we will give results to defend the heuristic. One proves the convexity of our function the others show that increasing λ corresponds to some kind of progress.

10 Rationalizing the Heuristic

In this section we give four results to defend our heuristic. To do this we propose a similar heuristic that is slightly different from our real heuristic. Then we prove some characteristics about that heuristic. Then we argue that our heuristic is an improvement from this other heuristic.

Now we define this similar heuristic and call it the approximate heuristic. It has the same steps as the our final heuristic but some of the inputs are different. If we are on the first repetition of the algorithm we set $t_0 = 0$. Now let $\hat{x} = T(t_0)$ and define f to be:

$$f(i,k) = (i',j') \text{ f.s. } x_{k'}^{i'} \in \arg\min_{x_{k'}^{i'} \in \hat{A}_k^i(\hat{x}))} \|x_k^i - x_{k'}^{i'}\|$$

Unlike in the real heuristic, in the approximate heuristic f does not change. It stays this function that we just defined throughout the algorithm. As before, we initialize $\theta_0 = 0$. But now we also set $\lambda_0 = 0$. We will see that in this version of the algorithm our initialization of θ_0 does not actually matter.

In each subsequent round we also calculate λ and θ differently. In the approximate heuristic given t_{n-1} we calculate $\hat{x} = T(t_{n-1})$ and let:

$$(\lambda_n)_k^i = \frac{1}{2 \cdot \|x_k^i - x_{f_1(k,i)}^{f_2(k,i)}\|^2}$$

$$(\theta_n)_k = \sum_{i=1}^{I+1} ||x_k^i - x_k^{i-1}||$$

Now given this approximate heuristic we show four proprieties. Firstly we see that our minimization problem has a unique solution. Next, we have λ does not decrease between rounds. Thirdly we show that the sum of the components of θ does not decrease between rounds. Finally we show that each minimum we find of our objective function is a minimum on a larger set than just our minimization domain for that objective function. Together these results shows that each step of our algorithm makes progress.

First we prove convexity of our function on its constrained domain.

Proposition 5: $\hat{V}_{\lambda,\theta}(t)$ is convex in t on B_λ.

Proof:

Firstly fix $t \in \mathbb{R}^{IK}$, $s \in \mathbb{R}^{IK}$ and $c \in (0,1)$. We have that T is affine so for all $k \in [K]$, $i \in [I]$ we have that:

$$T_k^i(ct_k^i + (1-c)s_k^i) = cT_k^i(t_k^i) + (1-c)T_k^i(s_k^i)$$

Therefore by the triangle inequality of $\|\cdot\|$ we have that given $k, k' \in [K]$ and $i, i' \in [I]$ that:

$$\|T_k^i(ct_k^i + (1-c)s_k^i) - T_{k'}^{i'}(ct_{k'}^{i'} + (1-c)s_{k'}^{i'})\|$$

$$= \|cT_k^i(t_k^i) + (1-c)T_k^i(s_k^i) - (cT_{k'}^{i'}(t_{k'}^{i'}) + (1-c)T_k^i(s_{k'}^{i'}))\|$$

$$< c\|T_k^i(t_k^i) - T_{k'}^{i'}(t_{k'}^{i'})\| + (1-c)\|T_k^i(s_k^i) - T_k^i(s_{k'}^{i'})\|$$

We also used the homogeneity of the norm $\|\cdot\|$. Therefore we have that:

$$\|x_k^i - x_{k'}^{i'}\| = \|T_k^i(t_k^i) - T_{k'}^{i'}(t_{k'}^{i'})\|$$

is strictly convex in t.

Now we want to deal with the exponential. We see that by taking the second derivative and setting it to be greater than zero we see that the function $e^{-\lambda x^2}$ is convex if and only if

$$x \leq \frac{1}{\sqrt{2\lambda}}$$

Furthermore, it is an increasing function when $x > 0$. We conclude that since we already saw that $\|x_k^i - x_{k'}^{i'}\|$ is convex and non-negative on its entire domain, then we have that

$$exp(-\lambda_k^i \cdot \|x_k^i - x_{f_1(k,i)}^{f_2(k,i)}\|^2)$$

is convex for all $k \in [K]$ and $i \in [I]$ if $t \in B_\lambda$.

Finally we see that $\hat{V}$ on B_λ is a sum of scalar multiples of convex functions. We conclude that $\hat{V}$ on B_λ is convex. $\square$

Since B_λ is a compact set and $\hat{V}$ is continuous then we know a minimum exists. Furthermore, $\hat{V}$ is convex on its domain and the domain itself, B_λ, is convex, so any local minimum is also a global minimum. Although the minimum may not be unique, at least we can find a value that minimizes the function on this domain. This is the only one of our four properties which also applies to the real heuristic and not just our approximate heuristic.

Secondly we see that by definition of our minimization problem as being constrained to B_{λ_n} at step n, we cannot have that λ decreases. This is because for λ to decrease we need the following to increase:

$$\|x_k^i - x_{f_1(k,i)}^{f_2(k,i)}\|$$

We constrain each minimization problem so that this does not increase. This is because we defined λ_n such that t_{n-1} is on the boundary of B_{λ_n}. This shows that along with the fact that $\lambda_n \geq \lambda_{n-1}$ we also have that $B_{\lambda_n} \subseteq B_{\lambda_{n-1}}$.

The higher λ, the more our exponential function $e^{-\lambda x^2}$ looks like the indicator function with support $\{0\}$. Therefore it makes intuitive sense that increasing λ gives us a network that looks more like the optimal drawing. The problem is that in increasing λ we may be forcing paths to collapse onto one another even if it is not lowering the total cost of our final drawing. Soon we will show that if we fix each node's neighbor throughout the algorithm then for any λ we input into our algorithm, the resulting minimum will be a minimum on a domain larger than just the domain of minimization. To do this we first need to show that it is a reasonable assumption that each path's length increases throughout the algorithm.

Now we want to show that at each step of the approximate heuristic the total length of our network specified by t_n does not decrease as n increases. The following proposition is equivalent to this by our definition of θ.

Proposition 6: At each step n of the approximate heuristic,

$$\sum_{k=1}^{K} (\theta_{n+1})_k \geq \sum_{k=1}^{K} (\theta_n)_k$$

Proof:

Firstly we see when $n = 1$, we have that if p_0 and p_f are the endpoints of path k then:

$$(\theta_1)_k = \|p_f - p_0\|$$

We see that since $(\theta_2)_k$ is the length of a path between p_0 and p_f that

$$(\theta_2)_k \geq \|p_f - p_0\| = (\theta_1)_k$$

This concludes the case when $n = 1$.

Now fix $n \in \mathbb{N}$. Assume to the contrary that $\sum_{k=1}^{K}(\theta_n)_k < \sum_{k=1}^{K}(\theta_{n-1})_k$. Then we want to evaluate:

$$\hat{V}_{\lambda_{n-1},\theta_{n-1}}(t_n)$$

We see that if we set $\hat{x} = t_n$ then:

$$\hat{V}_{\lambda_{n-1},\theta_{n-1}}(t_n) = \sum_{k=1}(\theta_n)_k \cdot (1+\alpha)$$

$$-\sum_{k=1}^{K}\sum_{i=1}^{I}(1+\frac{1}{2}\mathbb{1}_{\{i=1,i=I\}}) \cdot \alpha \cdot (\theta_{n-1})_k \cdot exp(-(\lambda_{n-1})_k^i \cdot \|x_k^i - x_{f_1(k,i)}^{f_2(k,i)}\|^2)$$

Now we see that since $t_n \in B_{\lambda_n}$ and t_{n-1} lies on the boundary of B_{λ_n} by definition of B_{λ_n}, then we have that

$$\hat{V}_{\lambda_{n-1},\theta_{n-1}}(t_{n-1}) \leq \sum_{k=1}(\theta_n)_k \cdot (1+\alpha)$$

$$-\sum_{k=1}^{K}\sum_{i=1}^{I}(1+\frac{1}{2}\mathbb{1}_{\{i=1,i=I\}})\cdot\alpha\cdot(\theta_{n-1})_k\cdot exp(-(\lambda_{n-1})_k^i\cdot\|T_k^i((t_{n-1})_k^i)-T_k^i((t_{n-1})_{f_1(k,i)}^{f_2(k,i)})\|^2)$$

Now given our assumption by way of contradiction that $\sum_{k=1}^{K}(\theta_n)_k < \sum_{k=1}^{K}(\theta_{n-1})_k$, we can replace the first term with $\sum_{k=1}^{K}(\theta_{n-1})_k$ and make our value strictly smaller. Then we see our expression is just $\hat{V}_{\lambda_{n-1},\theta_{n-1}}(t_{n-1})$. We conclude

$$\hat{V}_{\lambda_{n-1},\theta_{n-1}}(t_n) < \hat{V}_{\lambda_{n-1},\theta_{n-1}}(t_{n-1})$$

This contradicts that t_{n-1} is optimal. Therefore we have that

$$\sum_{k=1}^{K}(\theta_n)_k \geq \sum_{k=1}^{K}(\theta_{n-1})_k \square$$

Note that the case when $n = 1$ could be lumped into the general case by viewing the first step of the algorithm as the minimization problem when $\lambda = 0$. We wrote the proof this way to emphasize the reason we choose $t_0 = 0$, which implies all paths are straight lines between their origins and destinations. The idea is that the algorithm keeps making our paths longer so it is natural to start

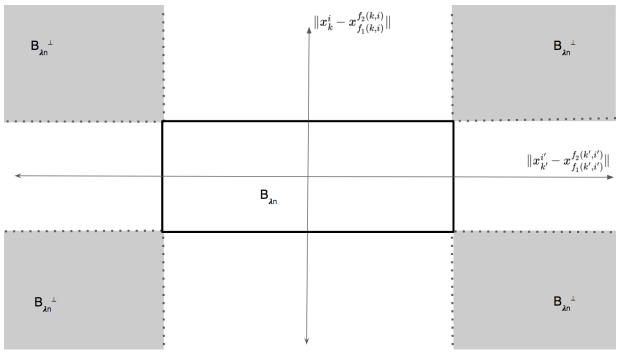

Figure 19: Drawing of $B_\lambda^\perp$

with the shortest possible path between each pair of endpoints. Furthermore, we get the stronger conclusion that in the first step of the algorithm all paths increase in length.

Given the previous proposition, we strengthen its claim and assume that $(\theta_n)_k$ is increasing for each k. Then we see that when we find a minimum on B_{λ_n} it is actually a minimum on a larger domain. To describe this larger domain define:

$$B_\lambda^\perp = \{t \in \mathbb{R}^{RI} : \hat{x} = T(t), \forall k \in [K], i \in [I],\ \|x_k^i - x_{f_1(k,i)}^{f_2(k,i)}\| > \frac{1}{\sqrt{2\lambda_k^i}}\}$$

We see this defines a set of networks where each point and its neighbor are further apart from one another than $\frac{1}{\sqrt{2\lambda}}$ for the λ that corresponds to a given point. Figure 19 depicts this region when considering two points x_k^i and $x_{k'}^{i'}$ and their neighbors. The axis are the distance between these two points and their respective neighbor. The black square is B_λ and the gray regions are $B_\lambda^\perp$. We see it extends infinitely.

With this definition we have the following theorem.

Proposition 7: At each step n of the approximate heuristic, if we have $(\theta_n)_k \geq (\theta_{n-1})_k$ for all $k \in [K]$ then

$$\min_{t \in B_{\lambda_n}} \hat{V}_{\lambda_n, \theta_n}(t) \leq \min_{t \in B_{\lambda_n}^\perp \cap B_{\lambda_{n-1}}} \hat{V}_{\lambda_n, \theta_n}(t)$$

Proof:

Fix $n \in \mathbb{N}$. Firstly we see that $t_{n-1} \in B_{\lambda_n}$. Therefore we have that:

$$\min_{t \in B_{\lambda_n}} \hat{V}_{\lambda_n, \theta_n}(t) \leq \hat{V}_{\lambda_n, \theta_n}(t_{n-1})$$

Now we fix $t \in B_{\lambda_n}^\perp \cap B_{\lambda_{n-1}}$. We want to show that:

$$\hat{V}_{\lambda_n,\theta_n}(t_{n-1}) \leq \hat{V}_{\lambda_n,\theta_n}(t)$$

Combining our first two inequalities we see this would prove the theorem. We see another way of stating this second inequality is as follows:

$$\hat{V}_{\lambda_n,\theta_n}(t) - \hat{V}_{\lambda_n,\theta_n}(t_{n-1}) \geq \hat{V}_{\lambda_{n-1},\theta_{n-1}}(t) - \hat{V}_{\lambda_{n-1},\theta_{n-1}}(t_{n-1})$$

This is because we know since t_{n-1} is a minimum on $B_{\lambda_{n-1}}$ then we have the right hand of this equality is greater than 0. Therefore proving this inequality above would imply that:

$$\hat{V}_{\lambda_n,\theta_n}(t) \geq \hat{V}_{\lambda_n,\theta_n}(t_{n-1}) \geq \min_{t \in B_{\lambda_n}} \hat{V}_{\lambda_n,\theta_n}(t) = \hat{V}_{\lambda_n,\theta_n}(t_n)$$

so this would conclude the proof. Now letting $t' = t_{n-1}$, we set off to show that:

$$\hat{V}_{\lambda_n,\theta_n}(t) - \hat{V}_{\lambda_n,\theta_n}(t') \geq \hat{V}_{\lambda_{n-1},\theta_{n-1}}(t) - \hat{V}_{\lambda_{n-1},\theta_{n-1}}(t')$$

We can rearrange this expression to get:

$$\hat{V}_{\lambda_n,\theta_n}(t) - \hat{V}_{\lambda_{n-1},\theta_{n-1}}(t) \geq \hat{V}_{\lambda_n,\theta_n}(t') - \hat{V}_{\lambda_{n-1},\theta_{n-1}}(t')$$

Using the definition of $\hat{V}$ we see no matter what the parameters λ and θ, the first part of our expression is a sum of terms $||x_k^i - x_k^{i-1}||$. We evaluate each side of the inequality at a single value, t on the left hand side, and t' on the right hand side. Therefore the first part of this expression which is a sum of terms $||x_k^i - x_k^{i-1}||$, cancels out on both sides of the inequality. We are just left with the exponential terms. We see by definition of $t \in B_{\lambda_n}^{\perp} \cap B_{\lambda_{n-1}}$, for all $k \in [K]$ and for all $i \in [I]$ we have:

$$\frac{1}{\sqrt{2(\lambda_{n-1})_k^i}} \geq \|T_k^i(t_k^i) - T_k^i(t_{f_1(k,i)}^{f_2(k,i)})\| \geq \frac{1}{\sqrt{2(\lambda_n)_k^i}} = \|T_k^i(t'^i_k) - T_k^i(t'^{f_2(k,i)}_{f_1(k,i)})\|$$

Furthermore we see that:

$$(\lambda_{n-1})_k^i \leq (\lambda_n)_k^i$$

$$(\theta_{n-1})_k \leq (\theta_n)_k$$

by our assumption on θ_n and the restriction of λ_n by the heuristic. Therefore our claim reduces to the following. Given:

$$\lambda \geq \lambda' \geq 0$$

$$\theta \geq \theta' \geq 0$$

$$\frac{1}{\sqrt{2\lambda}} = x \leq x' \leq \frac{1}{\sqrt{2\lambda'}}$$

Then we want:

$$\theta \cdot exp(-\lambda x^2) - \theta' \cdot exp(-\lambda' x^2) \leq \theta \cdot exp(-\lambda x'^2) - \theta' \cdot exp(-\lambda' x'^2)$$

Or equivalently:

$$exp(-\lambda x^2) - exp(-\lambda x'^2) \leq \frac{\theta'}{\theta}(exp(-\lambda' x^2) - exp(-\lambda' x'^2))$$

We see both sides are positive so it is sufficient to prove:

$$exp(-\lambda x^2) - exp(-\lambda x'^2) \leq exp(-\lambda' x^2) - exp(-\lambda' x'^2)$$

Now we plug in $x = \frac{1}{\sqrt{2\lambda}}$ to get:

$$exp(-\frac{1}{2}) - exp(-\frac{\lambda'}{2\lambda}) \leq exp(-\lambda x'^2) - exp(-\lambda' x'^2)$$

Now by taking derivatives we see that as λ' increases the difference between the two sides decreases. Therefore we only have to consider large λ'. By letting λ' go to infinity we get that to prove the previous inequality it is sufficient to show that:

$$exp(-\lambda x'^2) \geq exp(-\frac{1}{2})$$

Equivalently that $x' \leq \frac{1}{\sqrt{2\lambda'}}$, which is true by hypothesis. This proves our theorem as we have reduced our problem to this inequality that holds by assumption $\square$.

We now give an image of the domain over which t_n is a minimum as shown in the proposition. Figure 20 depicts $B^{\perp}_{\lambda_n} \cap B_{\lambda_{n-1}}$ by the green region. The smaller black rectangle is B_{λ_n}, and the larger black rectangle is $B_{\lambda_{n-1}}$. The proposition shows that t_n is a minimum for the entire green region.

We have managed to show that each step of the algorithm, our previous minimization problem teaches us that our next minimization problem is a minimum not just on its own domain but also on part of the domain that the previous minimization problem ruled out. This shows that each step of our process does not completely delete the work of the previous round of minimization. That being said, we would like the previous steps of our algorithm to rule at more than just a part of the domain outside of our convex ball. What we really wanted to show is that:

$$\min_{t \in B_{\lambda_n}} \hat{V}_{\lambda_n, \theta_n}(t) \approx \min \hat{V}_{\lambda_n, \theta_n}(t)$$

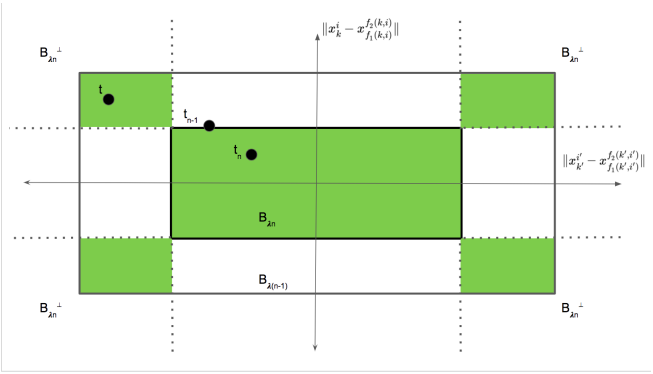

Figure 20: Drawing of $B_{\lambda_n}^{\perp} \cap B_{\lambda_{n-1}}$ in green

but we have come short of this. This would mean that our process of continually increasing λ helps us optimize a function even over regions that are not convex. To validate this process we need further analysis of what happens as λ increases and whether we do stay near the true global minimum

Together these proofs show that our approximate heuristic uses repeated convex optimization to keep searching for a better and better network. At each step of optimization, the length of the network keeps growing. The algorithm checks longer and longer paths until it finds that it is not longer beneficial to keep making the paths longer. As the paths get longer, they also get closer to one another. We see at each step of optimization we have the paths pull on each other with a stronger but more concentrated force. By the end of the algorithm we have that the paths, given their constraints, create the best network where paths have very concentrated and forceful pull on one another. This resembles how the actual problem is set up.

We adjust this approximate heuristic into our real heuristic because fixing each node's neighbor from the beginning is limiting. Instead we want each node's neighbor to always be as close as possible so at each stage in the algorithm we re-define each node's neighbor. As a result, it serves to prevent λ from growing too quickly and to start λ smaller than we have to in order to preserve convexity. If we let λ get large then points will move farther in each step and we may not register the closest node at each function call during convex optimization. On the other hand, restricting how fast λ can grow limits how large λ will eventually grow.

11 Applying Heuristic to Road Networks

First we need to determine the set of cities we want to connect. The goal is to represent a network of highways that connect cities. Therefore we treat each city as merely a point in a graph and do not consider the distribution of the population within city limits. In particular, we make no attempt to model

connecting one side of a city to another. For this reason, the model makes most sense for a set of cities that are far apart where the city population is relatively dense compared with the distance between cities.

Given these assumptions, we see the main consideration when determining our set of cities is how to split up population into points even though population seems to be continuously distributed. We must decide a cutoff of population density to determine if an area qualifies as a city or not. If we are trying to find a road network to connect the United States, for example, we see that we can let our set of cities be the 10, 20 or 50 largest cities and then our resulting road network will well-approximate how highways should be built to connect regions but will not be able to describe how highways should look in densely populated areas like the New York Metropolitan Area. For the following examples, when building a road network to connect a certain region we choose a fixed number of largest cities in the region. We then project these cities onto a plane, not considering the curvature of the Earth. This gives our set P.

Once we determine our set of cities, we must determine α. We want to find an α that will give a total cost of our network that represents the cost of roads and the cost of travelling. Therefore we want to know how much each mile of road costs to build, how many hours are spent travelling each on each mile of road during its lifetime, and how much each hour of a traveller's time is worth. Then we will compare how much a mile of road costs to how many dollars worth of time are spent on each mile of road over the course of its lifetime. This will give us α.

For highways in the United States we give the following estimates for these numbers. We estimate that a mile of four lane highway costs 10 million dollars to build.[3] Furthermore we see that in 2011 there were 47,000 miles of interstate in the US, and drivers drove a total of 720,000 million miles.[4] Since the average road lasts 25 years, assuming people drive on average at 60 miles per hour, we have that about 6 million hours were spent driving on each mile of highway in the US. Estimating drivers' time at 5 dollars per hour,[5] this means about 30 million dollars worth of time was spent on each mile of highway. Combining the 30 million dollar cost of time with the 10 million dollar cost to build a mile of road, we set $\alpha = 1/3$. This gives a rough idea of how to estimate the relative cost of time versus roads.

The final step input to our model is the demand for each path. So far we have not discussed demand, but for this application we assume a fixed number of drivers want to travel from each destination to each other destination over an arbitrary time period. Therefore we define $D : P \times P \to \mathbb{R}$ where $D(p, p')$ is proportional to the number of people who want to travel between $p \in P$ and $p' \in P$. We assume the demand is reflexive so $D(p', p) = D(p, p')$. Given this demand vector we weight each path by its demand so our time cost for a given route is proportional to its demand. Note that demand is fixed and does not depend on the distance between the two cities in the final network.

For our applications we will assume that our demand takes a certain form. We will employ the gravitational model of trade first proposed in Isard (1954).[6] This empirical rule says the amount of trade between two countries is approx-

imately proportional to the product of their sizes divided by the distance between them. We will apply this approximation to our cities. We assume if points $p, p' \in P$ represent cities with GDP's of $Y, Y' \in \mathbb{R}$ respectively, then we fix:

$$D(p, p') = \frac{Y \cdot Y'}{\|p - p'\|}$$

We then normalize so that the average demand is one. This enforces that on the average path, the cost of building a road is α times the cost of travellers' time. We note that no matter how large or far apart the cities in our input, the average path will have the same ratio of time cost to road cost. This assumption means that, in the next section, when we apply our model to different regions of the world, we are assuming that each region has equal average demand for highways. This is not a good assumption in general, but for the regions we chose it is reasonable. Alternatively, we could increase α for regions with higher demand and vice versa.

We have discussed how to specify our set of cities, the relative cost of time and road construction, and the demand for each route. These three inputs are all we need to run our heuristic.

12 Examples

In this section we will compare the heuristic to the optimal network without Steiner points. In other words we solve the problem for drawings of $G = (V, E)$ where $V = P$ and compare these results to our heuristic's results.

Firstly we consider the state of Florida and try to find the best way to connect its four largest cities: Miami, Tampa, Jacksonville and Orlando. We use FRED to find the GDP per capita of each city's surrounding region in order to compute demands.

Our first figure is the map Florida with our four cities labeled. Then we show the paths between these cities at different stages of our algorithm. Then we show the result of converting our final set of curved paths into a graph with straight lines and Steiner points. This gives us the topology that our algorithm has found. We then adjust our drawing to find the best drawing of this topology. Finally we show the best graph if we are not allowed to use Steiner points.

The graph given by our heuristic gives a total network cost of 1.37% less than the best network without Steiner points. We see this is because the two roads leaving Miami share the same road near Miami. This saves more in road expenditure than it costs in extra driving time.

Secondly, we consider the example of the Southeast of Australia. We include Sydney, Melborne, Brisbane, Adelaide, and Canberra. Although we now look at Australia, we keep our estimate of $\alpha = 1/3$. This time we use estimates from SGS of the GDP of regions surrounding these cities to determine demands.[7] We give the same figures as in the previous example. It turns out that our optimal network ends up being identical to the optimal network without Steiner points.

This example demonstrates how paths can be quite flexible to travel in awkward directions.

Thirdly, we consider the example of the Northeast of the United States. We include New York, Boston, Providence, Hartford, New Haven, and Springfield as our cities. Again we use FRED's GDP estimates for the regions surrounding these cities to determine demands. We give the same figures as in the previous examples.

In this example we see our network has 1.09% lower cost when compared to the network without Steiner points. That being said, when calculating the optimal network without Steiner points, we do not allow paths to switch edges at places where edges intersect. In this example we see the paths from Hartford and Springfield to Boston would benefit from switching to the edge from New Haven to Boston when their paths cross that road. If we were to add a Steiner point there, then the road network would have a lower cost. We see even if we add a Steiner point at this intersection, our optimal network is still 1.01% better.

Furthermore, we see in our final approximation with continuous paths that two paths leaving Springfield want to pass through Hartford but our restriction that paths cannot go backwards makes this impossible. In the end we turn this continuous set of paths into a graph that has these two paths originating in Springfield pass through Hartford, so, in this case, our restriction on paths does not end up creating a problem. This is a limitation of the fact that we represent each path as a function function with domain on the line segment between the path's origin and destination. In other examples a path may want to travel perpendicular to the line connecting the path's origin and destination. This is also not possible.

Another consideration in this case is how to prevent the roads from going through water. In this case we can avoid constructing roads in the ocean by merely paving the road from New York to New Haven so that it does not go into water. This leads to a larger question, however, of how to deal with terrain costs. One technique we can use is that if we do not want a path to go in a certain region we can add a big spike in the cost of the road network if any point on the path goes near that area. This spike can be made to preserve convexity of the overall function. Dealing with more general terrain costs is difficult.

This example suggests that it is bad social planning that in order to travel from New York to Boston you cannot take a straight shot but instead have to travel North within Connecticut and pass near Hartford. The amount of demand of between New York and Boston warrants a more direct route. The lack of a straight shot from New York to Boston might arise as our model does not consider traffic to Albany but this seems like it should not have a huge effect on the path from New York to Boston.

Finally we give two more examples of larger graphs that we were not able to compare to the optimal drawing without Steiner points. This is because it is difficult to find software to solve the Optimal Network Design Problem. In the first example we add two cities, Perth and Darwin, to our previous Australia example. For the second example we consider the 15 largest cities in

the United States. For each example we give the network resulting from convex minimization. Then we give our approximation of this curved-path network by a graph with straight edges. This determines the topology. Then we find the best drawing in this topology. Finally we superimpose this network onto the map of our two countries.

We see the Australia example leaves us without any Steiner points. Furthermore we notice two paths from Brisbane try to pass through Sydney but they cannot as we forbid paths from travelling backwards. To deal with this, we allow the paths to travel through Sydney in our graph with straight edges. We see one road passes through water but as described before this can be fixed by adding convex costs for paths going into areas of water.

In the US example, we see the large number of paths creates errors in paths finding neighbors but we can still make out the general structure of the network. Some Steiner points have degree two because some edges are not shown as no path uses these edges once we convert our network to a graph with straight edges. Again we run into problems with roads travelling through the water.

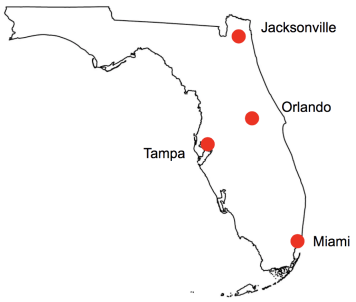

Figure 21: Map of Florida with Cities

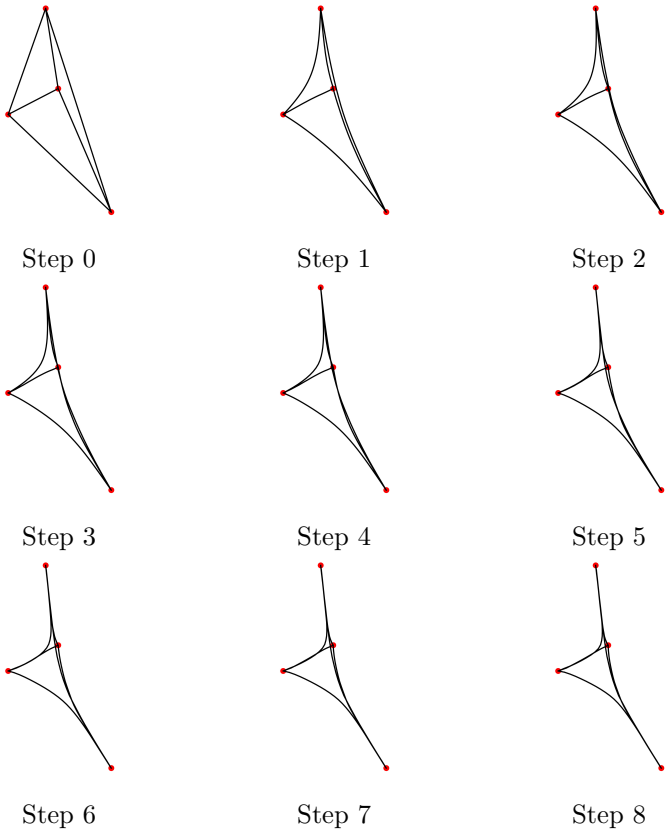

Figure 31: Network at different stages of algorithm

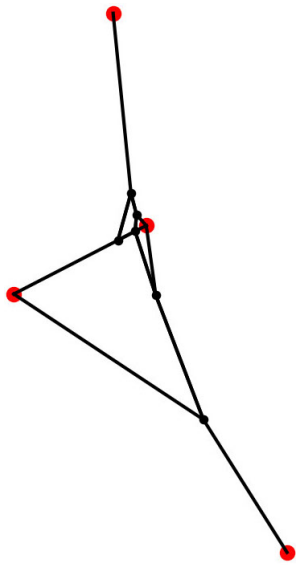

Figure 32: Network turned to graph

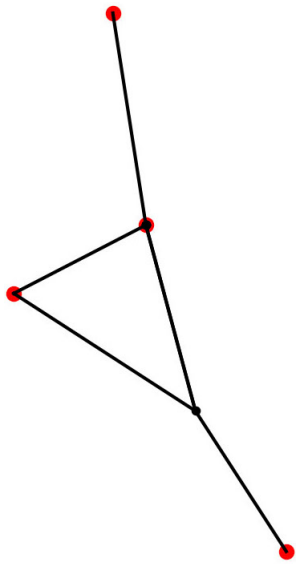

Figure 33: Heuristic Final Drawing

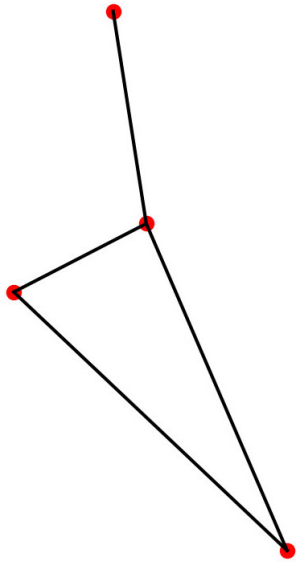

Figure 34: Best Drawing without Steiner Points

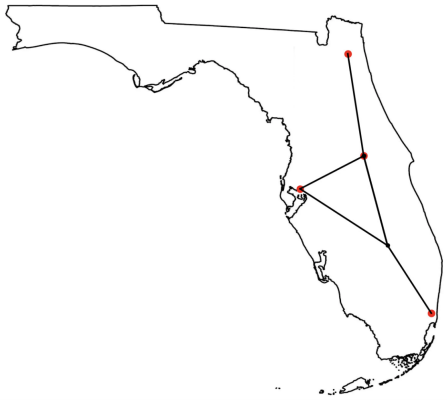

Figure 35: Our Highway Network

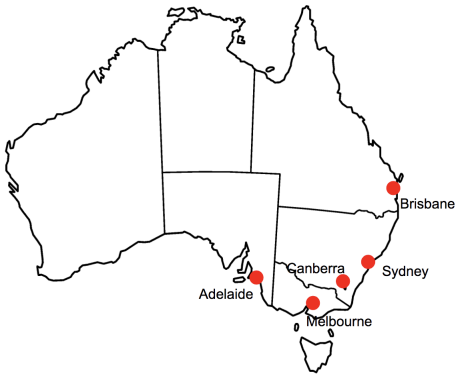

Figure 36: Map of Australia with Cities

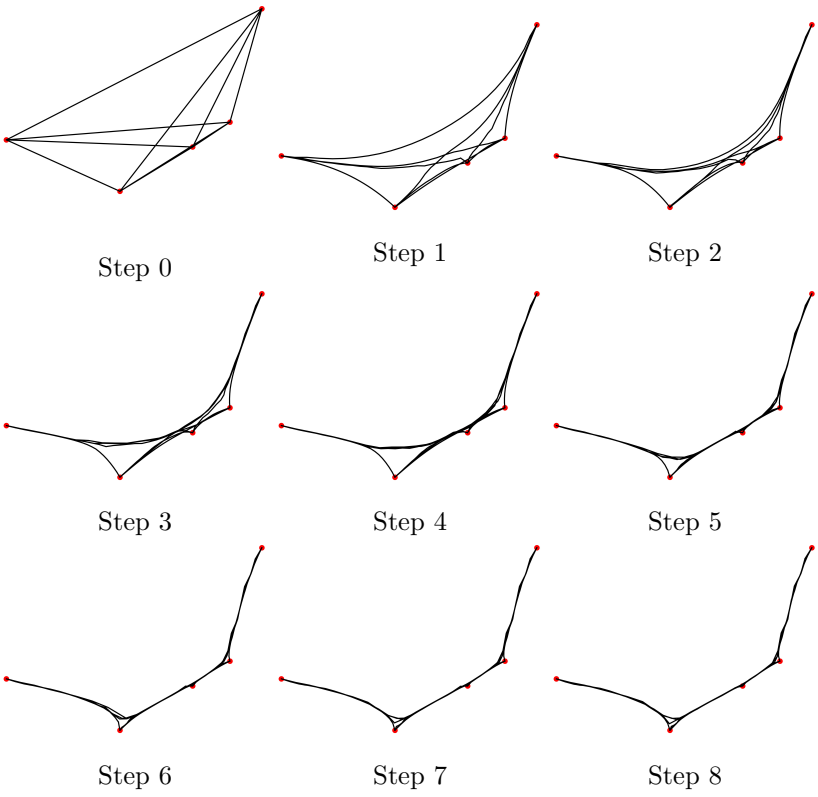

Figure 46: Network at different stages of algorithm

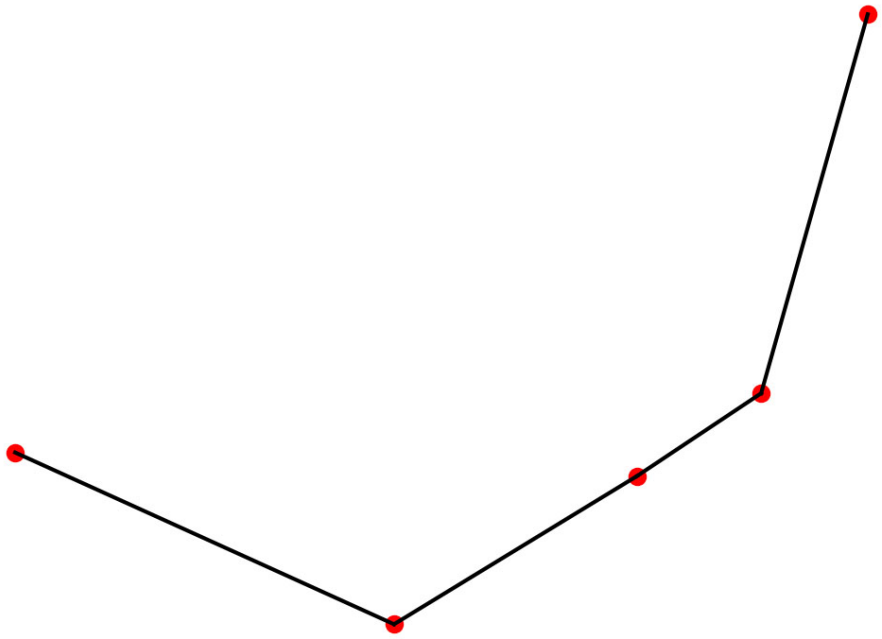

Figure 47: Network turned to graph

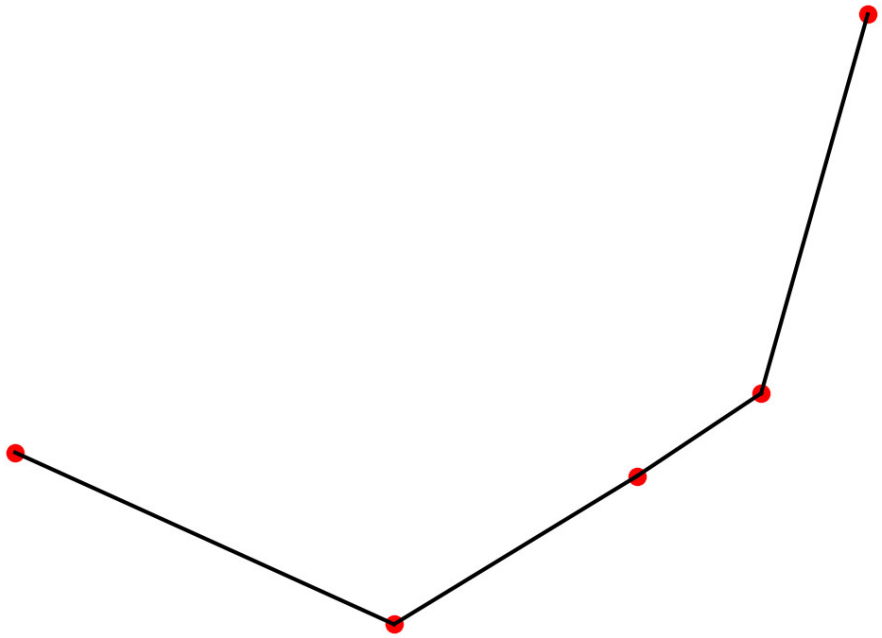

Figure 48: Heuristic Final Drawing

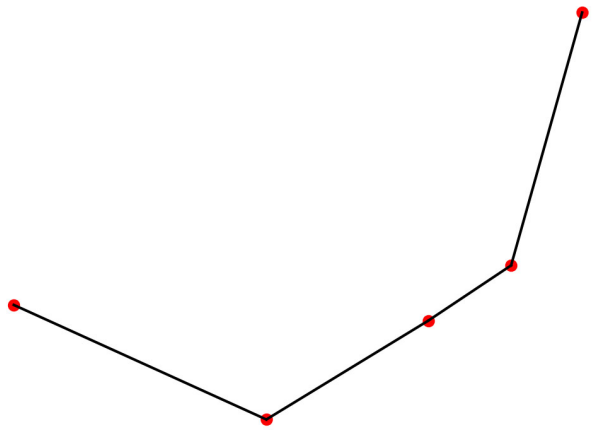

Figure 49: Best Drawing without Steiner Points

Figure 50: Our Highway Network

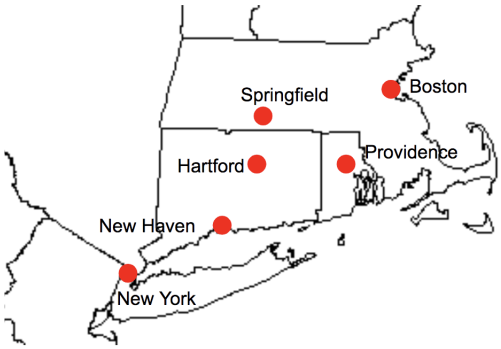

Figure 51: Map of Northeast with Cities

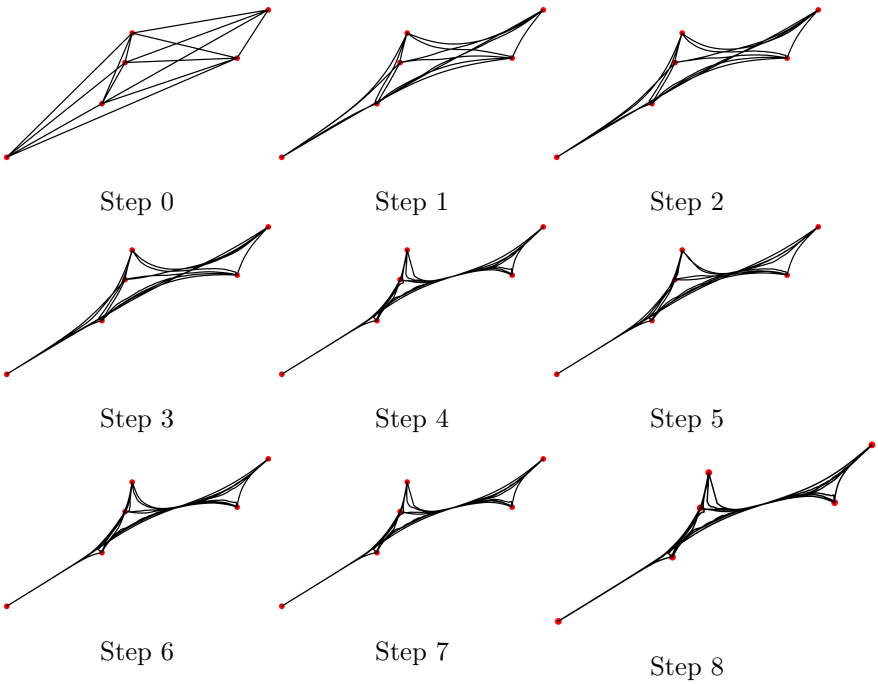

Figure 61: Network at different stages of algorithm

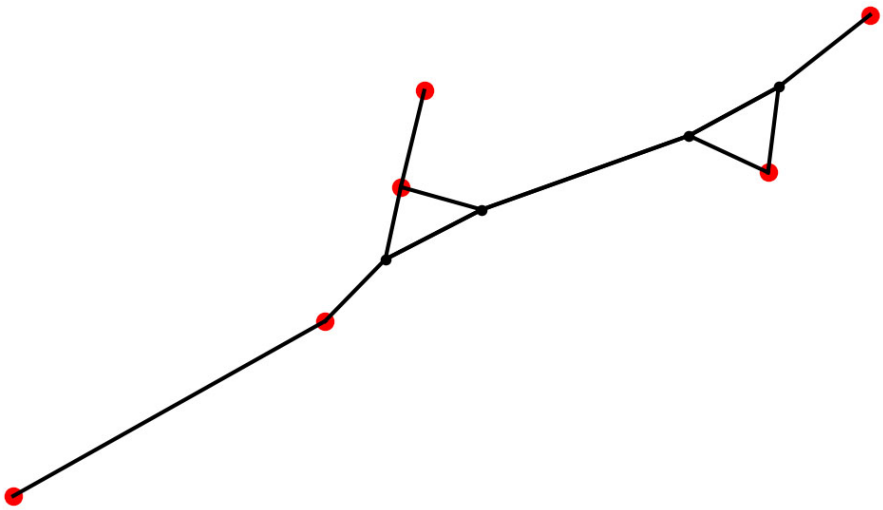

Figure 62: Network turned to graph

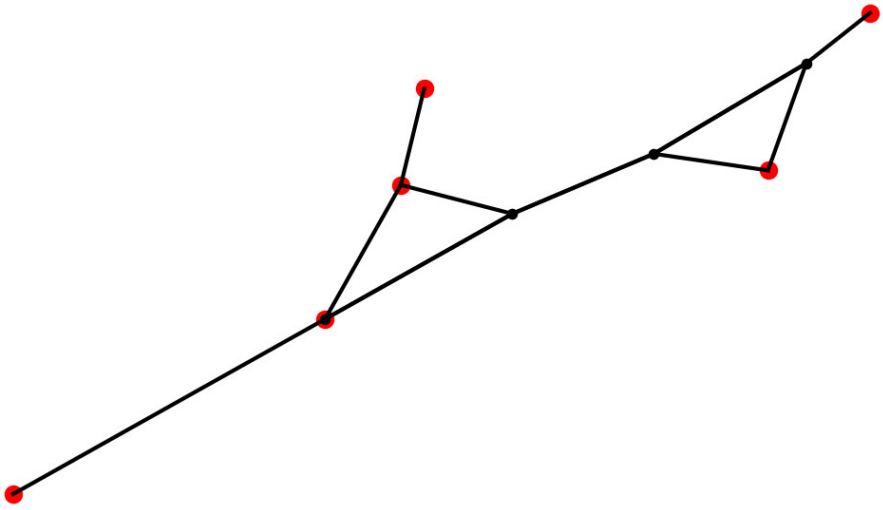

Figure 63: Heuristic Final Drawing

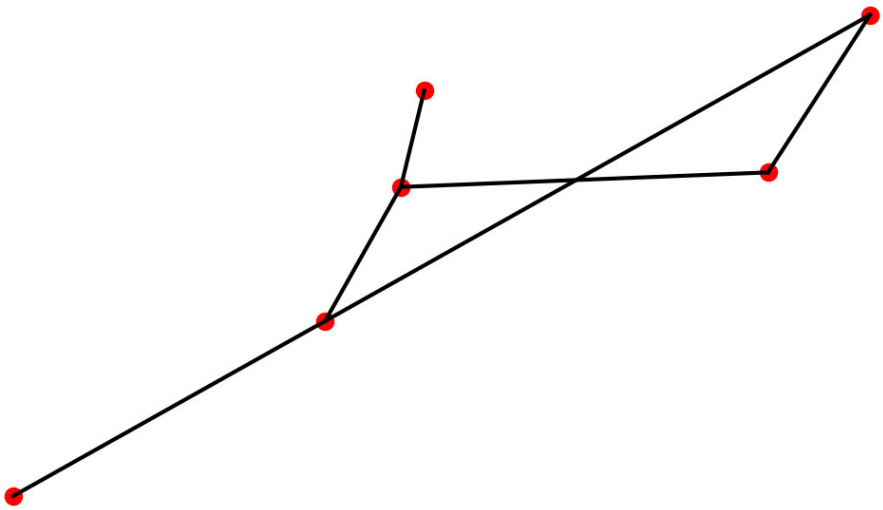

Figure 64: Best Drawing without Steiner Points

Figure 65: Our Highway Network

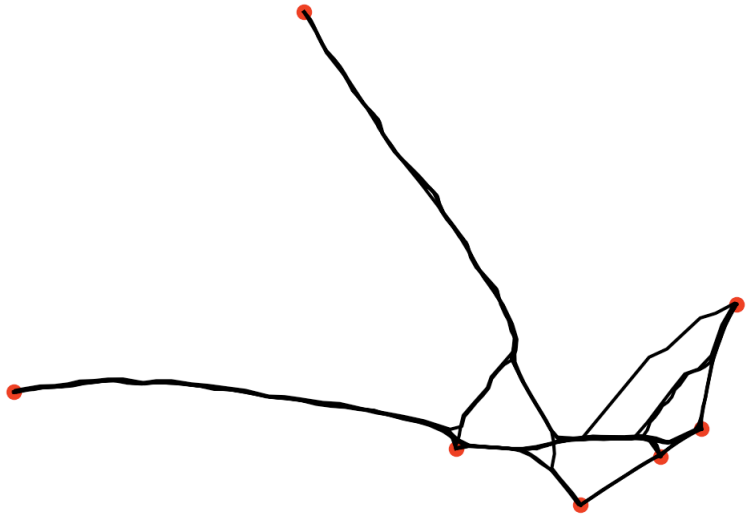

Figure 66: Network after Convex Optimization

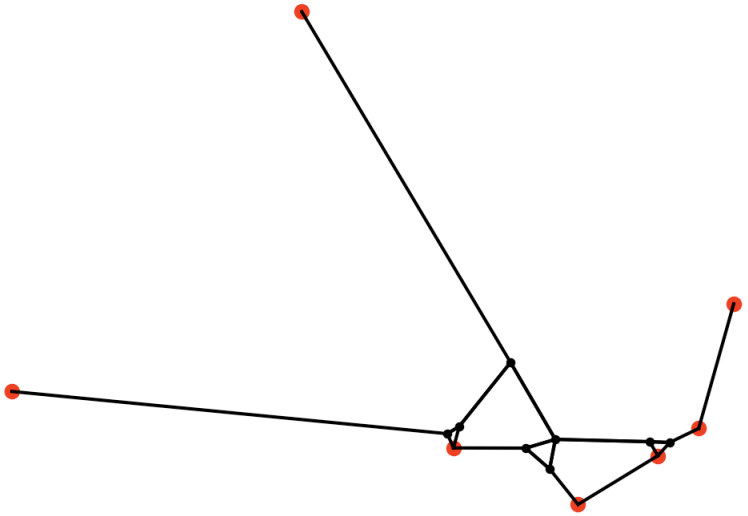

Figure 67: Network turned to graph

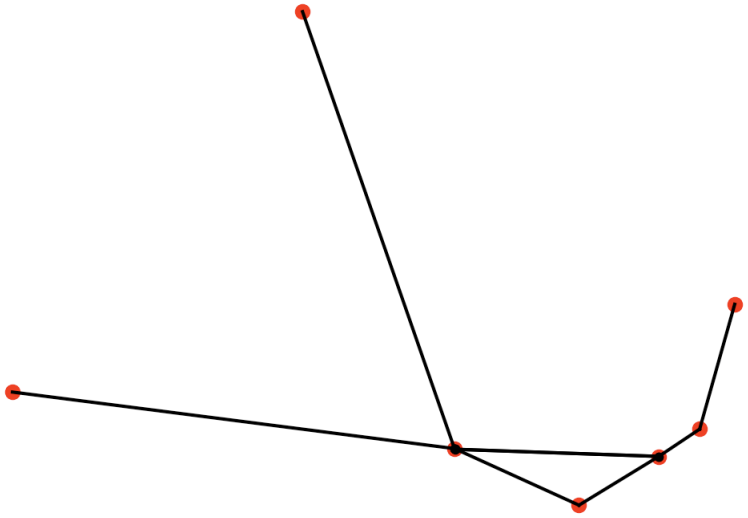

Figure 68: Heuristic Final Drawing

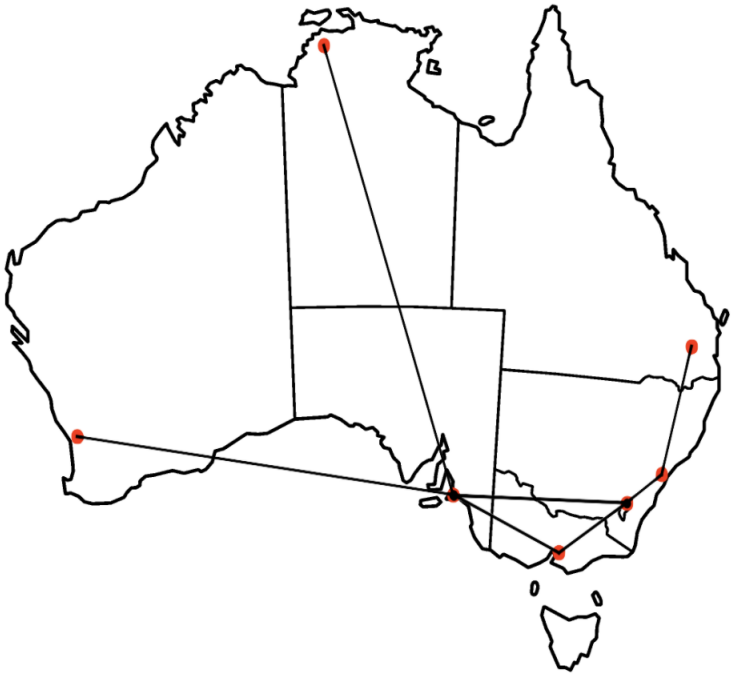

Figure 69: Heuristic Final Drawing Embedded on Map

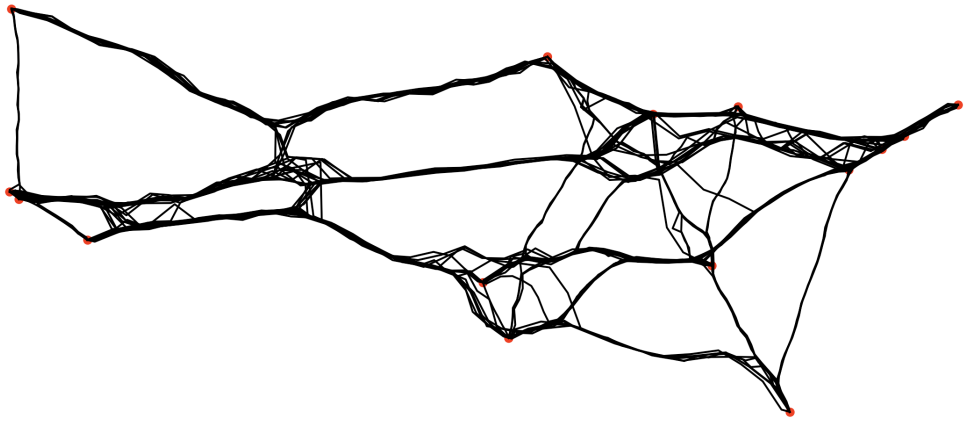

Figure 70: Network after Convex Optimization

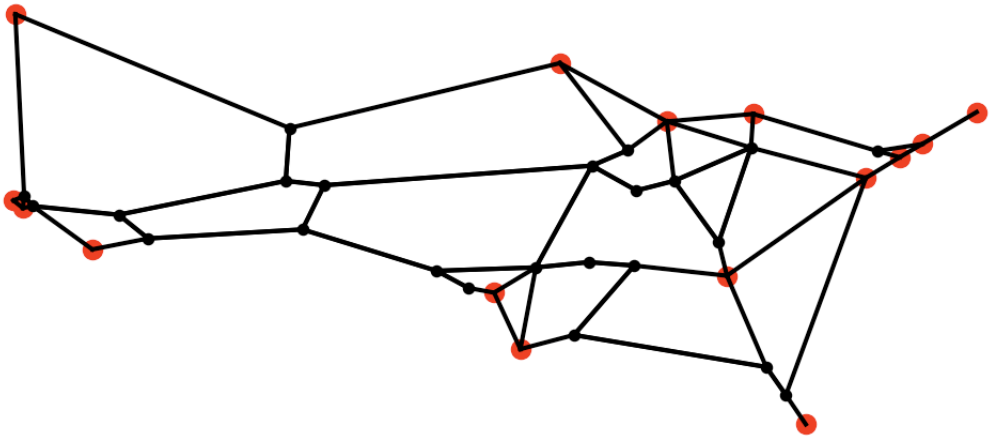

Figure 71: Network turned to graph

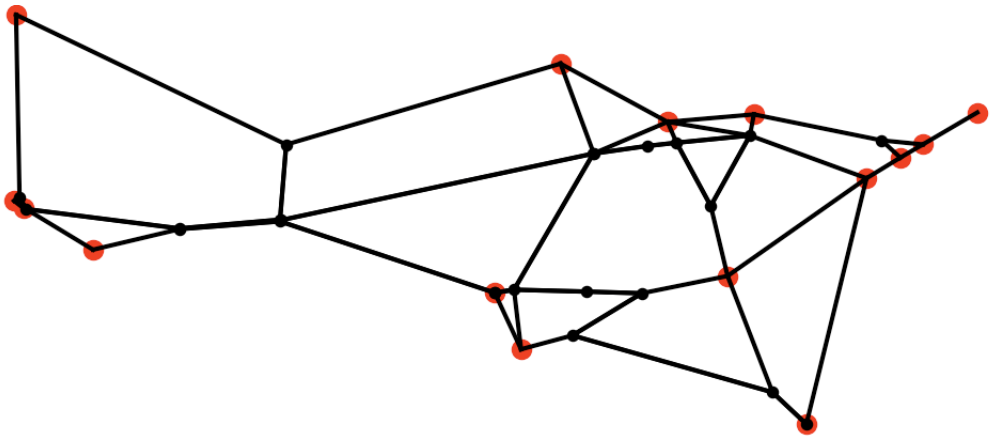

Figure 72: Heuristic Final Drawing

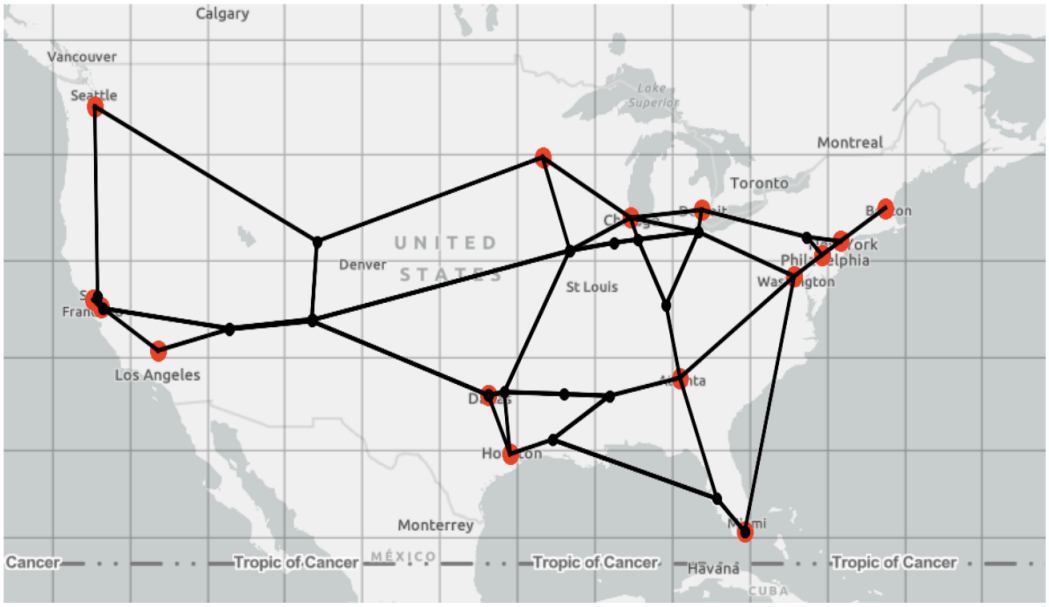

Figure 73: Heuristic Final Drawing Embedded on Map

13 Conclusion

The trade off between travel time and road length boils down to roads pulling on each other with a force of gravity. The thought of paths sharing a route pulls them together, but the cost of making the path longer keeps paths from falling onto one another completely. Suddenly, our problem is no longer discrete question. Instead we have curves continuously pulling at one another.

Considering gravity between paths, we can almost manage to represent the cost of our network with a convex function. The problem is that two paths struggle to pull on one another if there is a third path between them. In this way, each path serves as a hill preventing others paths from crossing, ruining our convex valley. Although this may seem devastating, actually, paths are constrained in that they cannot be too long and in that they cannot cross other paths too many times. This allows our local minima to be reasonable solutions.

We saw this method works reasonably well for small problems. In two examples, we could squeeze out a lower cost network by adding Steiner points. Although the sample size is very small, this research suggests that, at least for very small graphs, adding Steiner points makes a reasonable difference. We notice in the Florida example we are 20% off of the most trivial of lower bounds that $\alpha = 0$ and we have a complete graph. In the Australia example we are only 13% off of the same trivial lower bound. Given we are reasonably close to a network as cheap as a network where roads are free, a 1% improvement is significant. Another way to think about this is that the total cost of travel could be reduced by 1% by adding Steiner points. This is significant. It would be an important improvement if the magnitude of gains for adding Steiner points in larger graphs is similar.

For larger problems, the number of variables in the optimization problem grows like n^2 where $|P| = n$. This is quite good. The problem is that as n grows it seems like the algorithm needs to run more times. A further problem with large inputs is that there are so many paths pulling on each other in such a small area that two paths going the same direction do not necessarily use the same road but instead jump to different neighbors. Adding that points can pull on multiple points near them helped reduce this problem but not well enough.

One way we might go about fixing this problem of paths choosing their neighbors poorly is at each step we could combine paths that are already bound by the minimization problem to share a route. In other words we could take sections where paths share a route and convert that region into its own path. In this way, we might be able to break the problem up into different pieces and solve each piece well.

As the heuristic currently stands, it struggles to find good solutions for large problems. That being said, the method of considering paths as curves and attempting a convex minimization problem has the promise of performing well at large sizes as the number of variables in the minimization problem only grows quadratically with problem size.

14 References

1. Dionne, René, and Michael Florian. "Exact and approximate algorithms for optimal network design." Networks 9.1 (1979): 37-59.

2. Kou, L., et al. "A Fast Algorithm for Steiner Trees." Acta Informatica, vol. 15, no. 2, 1981, pp. 141–145., https://doi.org/10.1007/bf00288961.

3. "Highway Construction Costs." Washington State Department of Transport, July 2004.

4. "Summary of the Extent, Usage, and Condition of the U.S. Interstate System: By State and Interstate Route Number." U.S. Department of Transportation/Federal Highway Administration, https://www.fhwa.dot.gov/interstatebrief2011/.

5. "Transportation Cost and Benefit Analysis II – Travel Time Costs." Victoria Transport Policy Institute, 20 Mar. 2020.

6. Isard, Walter. "Location Theory and Trade Theory: Short-Run Analysis." Quarterly Journal of Economics 68 (1954): 305-320.

7. "Economic Performance of Australia's Cities and Regions." SGS Economics and Planning, Dec. 2018.