Constant-Distortion Embeddings of Road Networks

James McClain

Abstract

There are a wide variety of clustering and optimization algorithms which work in vector spaces, fewer that work on graphs. An algorithm to embed road network metrics into normed vector spaces is presented in this article, which makes a number of existing clustering and optimization algorithms available for GIS applications. Here, we modify and extend the Graham-Winkler graph factoring algorithm to enable the construction of ℓ_1 embeddings. While similar algorithms have been known in much more limited contexts, this work provides the first analysis and experimental results to show that this approach is applicable to road networks.

1 Introduction

In this article we address the question of how to embed road network graphs into vector spaces. Our definition of what a "road network graph" is can be found in Section 3, but informally it can be thought of as an undirected graph on n nodes that has edges of unit length, occupies space proportional to n when drawn on paper, and the circumference and diameter of the drawing are both proportional to $\sqrt{n}$.

We choose undirected graphs here because our goal is to embed road network graphs into vector spaces, where the distance function is reflexive. The choice of undirected graphs is not too unrealistic, as typically the ratio of the distance from a to b is within a constant factor of a the distance from b to a, even when one-way streets are present. The rest of the description is meant to be broadly evocative of a graph drawn inside of a circle with the vertices approximately evenly spaced from one-another.

The question of how to embed graphs is a well-studied one. In Section 2 we present previous work that has been done on embedding various classes of graphs, as well as techniques for quickly computing distances in graphs and finding shortest paths. In that section we also present material related to graph factoring which we then build our contributions on top of.

Our interest in embedding graphs motivated by the fact that use of incremental optimization algorithms is indicated for many common and useful optimization problems, such as k-clustering [32]. That problem is known to be NP-Hard even in the Euclidean plane [33], and is therefore at least as difficult in a planar graph, making direct computation of answers infeasible. However, having the ability to embed a graph into a vector space with low distortion opens the possibility of using well-known tools like Lloyd's Algorithm [31] to solve these types problems.

Main Contribution

The main contribution that we present in this work is an algorithm for producing embeddings of road networks graphs into $\ell_1^{\mathcal{O}(\sqrt{n})}$[1]. The embeddings that we produce have the following property: for a large-enough road network graph, the probability that the distance in the graph metric between two vertices and their distance in the metric given by the embedding differs by more than a constant factor is nil. More formally: there exists a fixed constant c such that if $G(V, E)$ is a randomly-selected road network graph on n nodes, $a, b \in V$ are randomly selected, $\mathrm{d_G}(a, b)$ is the distance between a and b in the graph, and $\mathrm{d_e}(a, b)$ is the distance between a and b according to the embedding, then

$$\frac{1}{c} \leq \frac{\mathrm{d_e}(a, b)}{\mathrm{d_G}(a, b)} \leq \frac{c}{1}$$

is true with probability 1 as n goes to infinity. The constant c does not depend on n.

A low-distortion embedding algorithm like the one presented in Section 4 is a necessary ingredient for the types solutions to clustering and optimization algorithms that we have in mind, but more is required. Also required is the ability to take solutions computed in the embedding space and bring them back into the context of the graph, that is the subject of Section 5. Another concern is the "meaningfulness" of bringing computations done in the embedding space back into the graph: that question has been previously addressed in another work [34], where it is shown that (for example) if one computes the k-means of a collection of graph vertices in a constant-distortion embedding space, using a constant-factor approximation algorithm, then associating every solution point with the closest vertex in the graph gives a constant-factor approximation.

Motivation

So far we have already motivated the idea of using incremental optimization algorithms for finding solutions to graph problems, but we can also offer some additional examples for why solving such problems are interesting at all.

Examples of GIS applications for k-clustering on road networks can be found in [34], but we provide a few here, as well. The k-centers problem can be used to find the k locations in a city that minimize the maximum distance from a query vertex to one of the k locations. This might be useful for something like deciding where to locate k new hospitals if the goal is to minimize the maximum travel time to a hospital from any given residence. The k-medians problem might be used to minimize the total distance from k warehouses (the solution set) to a bunch of retail locations (the query points), while the k-mean problem could be used to minimize the average distance from the warehouses to the stores.

Organization and Notation

We have already mentioned that previous work is presented in Section 2, that our road network graph definition is presented in Section 3, that the main embedding algorithm is

[1] The space ℓ_p^d is a d-dimensional space in which the norm of a vector $x := (x_1, \cdots, x_d)$ is defined to be $\|x\|_p := \sum_{i=1}^{d} |x_i|$.

presented in Section 4, and that Section 5 details the algorithm for computing combinations in the embedding space and bringing them back into the graph. In Section 6 we evaluate the performance of the algorithms for computing embeddings and finding linear combinations. Some final thoughts are offered and the article is concluded in Section 7.

Symbol	Meaning
G	A graph representing a road network.
$\mathrm{EMB}(a)$	The point in EMB that corresponds to $a \in V$.
$\mathrm{EMB}(G)$	The image of the graph in the embedding, a vector space.
$\mathrm{d_G}(a, b)$	The distance between a and b in the graph metric.
$\mathrm{d_e}(a, b)$	The distance between $\mathrm{EMB}(a)$ and $\mathrm{EMB}(b)$ in the image space.

Table 1: Notation used in this article.

The notation used in this article is presented in Table 1. Definitions for terms are either given in context or presented in footnotes.

2 Previous and Related Work

This section summarizes and discusses prior work relevant to the subjects of defining what a road network graph is and computing embeddings of road networks into vector spaces, and the graph factorization work on which our main algorithm is based is also presented.

In Section 2.1 we discuss the notion of *highway dimension* which provides the most important existing road network definition, and we contrast it to the definition given in Section 3 of this article. In Section 2.2 we present previous work related to computing distances in graphs. Work related to embedding road network graphs as well as other classes of graphs is presented in Section 2.3. The prior work most directly-related to the present work, the Graham-Winkler graph factoring algorithm, is presented in Section 2.4.

2.1 Highway Dimension

For any vertex $a \in V$, let $\mathcal{B}(a, 4r)$ be defined as the set of vertices which are of distance $\leq 4r$ from a, for some r. The *highway dimension* [2] of a graph is the smallest number h such that there exists a subset $S \subset \mathcal{B}(a, 4r)$, with $|S| \leq h$, such that any shortest path of length at least r contained entirely in $\mathcal{B}(a, 4r)$ coincides with S in at least one vertex.

In plainer language, h is the size of the smallest set that touches every shortest-path of length r in a radius $4r$ ball around a; this statement is quantified over all vertices a and all radii r. Many of the current-best shortest-paths algorithms for road networks have their theoretical basis in analysis that uses this idea. The assumption is made in those works that the highway dimension h is somewhat small for a graph representing a road network.

Two highway dimension examples are given in [2]. An example of a graph with a highway dimension of one is the star graph on n vertices: one vertex with degree $n - 1$ and the rest of degree one, with all edges having unit length. All shortest paths meet at the vertex of degree $n-1$, and can therefore be covered by a set consisting of only that vertex.

The second example given is a grid-graph. Such a graph, with unit-length edges, has highway dimension $\Theta(\sqrt{n})$. A ball of radius $4r$ in a grid can be separated by a constant

number of vertex cuts of length (roughly) $8r$ in such a way that there are no shortest-paths of length r or more in any of the newly-created components. That implies that a collection of vertices of size proportional to the radius of the ball can be chosen so that any shortest-path of length r will touch one of those vertices. A ball of radius $4r$ would therefore need a set S of size $|S| \approx r$, but the ball would contain roughly r^2 vertices.

The class of graphs with low or moderate highway dimension is viewed by the authors of [2], [1], [19], [7], and others, as a definition of a road network.

2.2 Distance Queries

Computing the distance between two points is one of the operations inherently supported by an embedding, so it is necessary to talk about previous work that has been done on computing distances.

A *distance oracle* is a data structure to facilitate responses to distance queries, questions about the distance between two vertices in a graph. The name *distance labels* has been given to a sub-type of distance oracles, those that associate some particular data with the individual vertex that engendered it, rather than creating a data structure in which the data are stored in a more at-large manner.

2.2.1 Distance Oracles

A popular approach to the construction of distance oracles is the use of *graph separators*. If $G = (V, E)$ is a connected graph, a separator is a subset of vertices of the graph whose removal causes the graph to become disconnected. When one separates the graph into two (or more) pieces, the hope is that it will fall apart into pieces of roughly the same size. The repeated and recursive separations of graphs and subgraphs in this way enables a divide-and-conquer strategy.

The typical preprocessing procedure for all algorithms of this style is to build a *separator decomposition tree*. Each node in the tree represents one separation of the graph (or subgraph), each node contains the list of vertices in the (sub)graph being divided, the list of vertices in the separator, and the distance between every vertex in the separator and every vertex in the (sub)graph. In planar graphs, it takes $\mathcal{O}(n\sqrt{n})$ operations to construct each node, $\mathcal{O}(n)$ time to find the separator, and $\mathcal{O}(n\sqrt{n})$ time to find each of the $\mathcal{O}(\sqrt{n})$ shortest-path trees rooted at each respective vertex in the separator. This is a tree-division process, so the total construction time required is $\mathcal{O}(n\sqrt{n})$ and similar storage space is needed.

An instance of this idea is given in [3]. That work shows how to construct distance oracles for planar graphs that need $\mathcal{O}(\sqrt{n})$ operations per exact query and $\mathcal{O}(\log n)$ operations per 2-approximate query. It works in the following way: If $a, b \in V$ are two vertices of the graph, then an exact query of the distance between them is done by first finding the lowest node on the separator tree that contains both vertices, the node at which a and b are separated from one another by the separator given in that node. It is shown in that work that the shortest path between those two vertices must either pass through the separator stored in the lowest common ancestor, or must pass through the separator of one of its ancestors going upward toward the root of the tree. Consequently, the distance between a and b can be found by computing $\min_{c \in \mathrm{X}} \{\mathrm{d_G}(a, c) + \mathrm{d_G}(c, b)\}$ where X is the set of graph vertices

in the just-mentioned sequence of separator sets, starting at the one in the lowest common ancestor node of the separator tree and going up. In other words, if $T_1, \cdots, T_i$ is the sequence of tree nodes from the root to the separating node and if $S(T_j)$ is the separator set associated with T_j, then $X := \cup_{j=1}^{i} S(T_j)$. The total number of vertices in X is $\mathcal{O}(\sqrt{n})$, which is the query time.

A 2-approximate query can be done in fashion similar to that of the exact query. Again, the first step is to find the lowest common ancestor on the separator tree that contains both a and b. If we imagine that $u, v \in V$ are (respectively) the closest vertices to a and b in the separator set of some node of the separator tree, then it must be the case that if the shortest path between a and b goes through the separator then the length of the shortest path can be no less than half $\min\{\mathrm{d_G}(a,u) + \mathrm{d_G}(u,b), \mathrm{d_G}(b,v) + \mathrm{d_G}(v,a)\}$. With that, the observation that was made previously relative to shortest paths, and separator sets going up the separator tree, can be recycled; the shortest path must go through the separator set of either the lowest common ancestor or one of its ancestors. Since the separator tree has $\mathcal{O}(\log n)$ height and since we are now only required to check four distances for every separator tree node touched, the result is a $\mathcal{O}(\log n)$-time query algorithm.

2.2.2 Distance Labeling

Distance labeling schemes are similar to distance oracles, except that they pursue the association of specific data with specific graph vertices as a specific objective. The same data $f(u)$ are used for the vertex u to perform the distance query $\mathrm{d}(u,v)$ as are used when $\mathrm{d}(u,w)$ is to be computed. Another way to say that is: a distance label is a tuple-valued function $f(\cdot)$ such that for each $u, v \in V$ the distance in G between u and v can be computed solely from $f(u)$ and $f(v)$ without any reference to or knowledge about the graph G. This can be contrasted to a general distance oracle which cannot be so-described.

According to the survey [25], most the present interest in distance labels was begun by the paper [22]. The property of distance labels which attracted attention then is the same property that brings them into the conversation here: some distance labels can be viewed as embeddings into normed spaces. In fact, that work contains a famous conjecture: that any planar graph can be embedded into ℓ_1 with constant distortion. A function f which takes its vertex inputs to tuples of real numbers such that $\sum_{i=1}^{d} |f(u)_i - f(v)_i| \approx \mathrm{d_G}(u,v)$ for $u, v \in V$ is an approximates embedding of the graph G into ℓ_1^d. Conversely any embedding is also a distance label.

There are a number of results known about embeddings of graphs into normed spaces via the distance labeling concept. It is know from [11] that graphs can be embedded into ℓ_p with distortion proportional to $\mathcal{O}(\log n)$ and that was later improved in [37] to $\mathcal{O}\left(\sqrt{\log n}\right)$. It was also shown in [18] that any exact, optimal labeling scheme on a planar graph requires at least $\Omega\left(n^{\frac{4}{3}}\right)$ bits and at most $\mathcal{O}\left(n^{\frac{3}{2}} \log n\right)$ bits of space.

The results just mentioned are mostly existence proofs based on the idea of separators, there are comparatively few constructive results in this area. A good example of this tendency is [18].

Although the arguments in [18] are fairly formidable in their level of detail, the general idea behind the proofs of most of the bounds found in that paper is fairly simple. The strategy used is to count the number of graphs in a given family and then observe that if

there are that many members in the family, then the total number of bits needed to label the average individual member must be sufficiently large to hold the integer which is the size of the family. Examples of families of graphs are things like "all planar graphs on n vertices" or "all trees on n vertices of degree five".

If there are 144000 members in a particular family then one can never get away with fewer than $\lceil \log 144000 \rceil = 18$ bits to represent (to label) the average member of that family. Were that not the case, there would have to be two non-isomorphic graphs with the same label. The argument even applies to approximate distance labels: allowing an approximation factor essentially "mods out" some members of the graph family, it reduces the effective size of the family by some amount commensurate with the approximation factor because graphs with similar-enough metrics become indistinguishable modulo the approximation factor.

The article [17] is one of the few works that presents an actual distance labeling scheme that approaches optimality. A 3-factor labeling for planar graphs that requires $\mathcal{O}\left(n^{\frac{4}{3}} \log n\right)$ total bits for all of the labels is presented. Unfortunately, the labeling given is not one of the ones that can be interpreted as an embedding. The labels are complex and require the evaluation of numerous "if-statements" predicated on the label data to compute the approximate distance, rather than some simple aggregation operation like summation, as would be the case for a nice embedding.

The mechanics of the algorithm are essentially the same as those of the distance labeling schemes discussed above, except that the graph is split into regions in one step rather than through a recursive process. The graph G is divided into $n^{\frac{1}{3}}$ regions, then each of those regions is again split into $n^{\frac{1}{3}}$ subregions. The label of each vertex in the graph is comprised of the following data: the vertex's unique identification number, the region that it belongs to, the subregion within that region that it belongs to, the distance to every vertex in its subregion, the identifier of and distance to the closest boundary vertex in every region, the same for every subregion, and the distance to every vertex in the boundary of its region. Each of the regions of vertices mentioned above is of size no more than $n^{\frac{1}{3}}$ so that each vertex can be encoded in $\mathcal{O}\left(n^{\frac{1}{3}} \log n\right)$ bits of space.

To find the distance between $a, b \in V$, one notes whether the two vertices are in the same subregion, the same region, or different regions. If the two vertices are in the same subregion, then the distance between them is stored and can be read-out. If the two vertices are in the same region but different subregions, then the distance between them is the sum of the distance between a and the closest vertex in subregion holding b (which is on the boundary of the subregion) and the distance between that same vertex and b. Those two values are stored, so they can be read-out and summed. Finally, if the two vertices are in different regions, one consults the labels to find the distance between a and the closest vertex on the boundary of the region of b and the distance between that boundary vertex and b, then those are summed.

2.3 Embeddings

In [30] it is shown that embeddings of expander graphs into ℓ_2 necessarily have $\Theta(\sqrt{n})$ distortion, and it is proven in [35] and [37] that planar metrics cannot be embedded into ℓ_2 with constant distortion. In spite of the fact that the class of road network graphs and

the class of planar graphs are distinct (and neither is a subset of the other), they are similar enough for this to disqualify ℓ_2 as a target space for our embeddings. However, the conjecture is made in [22], [35], and [26] that any planar graph can be embedded into ℓ_1 with constant distortion. This puts ℓ_1 on the table for consideration. One of the prime candidates for fulfilling the conjecture is the algorithm which comes out of Bourgain's Theorem.

Bourgain's Theorem provides a method for embedding graphs into ℓ_p with $\mathcal{O}(\log n)$ distortion for $p < \infty$. In [16], a version of Bourgain's Theorem particularized to ℓ_1 embeddings is given. The work in [16] is actually focused on embeddings into tree metrics, but trees can be embedded into ℓ_1 with no distortion [6] and [29] are two important works on probabilistic tree embeddings that antedate [16].

The ℓ_1 version of Bourgain's Theorem, as presented in [16] works in the following way. For every value of i between one and the logarithm of the diameter of the graph, a uniform random number r_i is chosen from the range $[2^i, 2^{i+1}]$. For each random number chosen in this way, a random order is placed on the vertices in the graph. Following that order iteratively, groupings are established wherein all not-previously-grouped vertices within r_i units of the current vertex are placed into a group. Some of these groupings can be empty and a vertex is not necessarily in the grouping that it engenders.

Once the $\log n$ sets of groupings – each set representing a different length scale – have been created, they are then separately turned into an ℓ_1 embedding of the graph at the scale that they represent. The procedure for doing that is easy: if two vertices in the graph are in the same grouping at a particular scale, they are given a distance of 0 from one another, whereas if they are in different groupings, they are given the distance 2^i. The embeddings for each of the scales are then combined together to form the embedding of the graph.

The proof that this provides an $\mathcal{O}(\log n)$-distortion embedding runs roughly as follows. If two vertices a and b are a certain distance apart, then they are separated in all of the sets of groupings at scales less than the distance between a and b. This automatically implies that the distance between a and b in the embedding will be at least a fixed fraction of the actual distance. On the other hand, it can be proven that the probability that two vertices which are closer than r_i will be grouped separately, thereby creating an overestimate of the distance, is proportional to $\frac{1}{r_i} \log n$. This gives an expected overestimate of $\mathcal{O}(\log n)$.

There are a few things about the ℓ_1 version of Bourgain's Theorem that make it unsuitable for the goals that we have in mind. The first issue is that the best bound on the method's distortion is $\mathcal{O}(\log n)$, as stated before. (The distortion has been conjectured to be constant on planar graphs [26], but this has not been proven, and road networks are not planar.) The second issue is the running time of the algorithm. A straight-forward implementation of the algorithm takes at least $\Omega\big(n^2 \log n\big)$ steps, with a derandomized version needing $\omega(n^2 \log n)$ operations. These preprocessing times put this approach out of reach for large maps which have millions of vertices.

There has been some work done on embedding road network graphs into normed spaces of fixed dimension. One such work is [12] in which the road network for Tallahassee was embedded into six-dimensional space. That work views the embedding process as an optimization problem, attempting to find the lowest-energy state of spring model. The problem with this approach is that it is impossible to provide guarantees about the quality of the embedding: any scheme that embeds an input graph into a constant number of dimensions cannot guarantee low distortion.

There has also been some work on embedding graphs into spaces with non-numeric

coordinate values. The *Squashed Cube Conjecture* [20, 39] gives a methodology for embedding any graph isometrically into what amounts to an ℓ_1 space with coordinate values of 0, 1, and $*$. The "$*$" is defined to have zero distance from a 0, from a 1, and from another "$*$".

The Squashed Cube Conjecture posits the use of the * to help to represent hyperfaces which have been squashed together. For example "$00*$" represents an edge which has been squashed into a single vertex and "$0***11$" represents a cube that has been squashed into a single vertex. That this "squashing" of faces can be used to embed any graph was proven in [39]. In spite of this, the word "conjecture" continues to be part of its name.

A couple of definitions are needed before the Squashed Cube Conjecture embedding algorithm can be described. The first object to be defined is a breadth-first tree, called T, that is rooted at some random vertex in the graph. The discrepancy function is defined to be $c(i,j) := \mathrm{d_T}(i,j) - \mathrm{d_G}(i,j)$, the difference between the metrics provided (respectively) by the tree and the graph. The value of i' is defined to be the highest-indexed vertex on the path from the root to i, excluding i, according to the breadth-first order.

In the squashed cube isometric embedding technique, a vertex a is associated with an n-dimensional vector by giving $v_a(b)$ a value of 0, 1, or $*$. Here, b is another vertex in the graph.

The technique says that $v_a(b) := 1$ if b precedes a in the breadth-first order.

The coordinate $v_a(b)$ is given a value of $*$ if one of three conditions is true. The first condition is that $c(a,b) - c(a,b') = 2$, that is if the predecessor in the tree-order of b is one unit further from a than is b itself. The second is if $c(a,b) - c(a,b') = 1$, $a < b$, and $c(a,b) = 2k$. The third is if $c(a,b) - c(a,b') = 1$, $b < a$ and $c(a,b) = 2k+1$. The last of the two cases deal with the situation where $\mathrm{d_G}(a,b) = \mathrm{d_G}(a,b')$, that is b and its predecessor b' are the same distance from a in the graph, but $\mathrm{d_T}(a,b) \neq \mathrm{d_T}(a,b')$, due to the tree order.

If none of the above conditions are true, then $v_a(b) = 0$.

This is an ℓ_1 embedding of the tree metric T with modifications made where needed. In the places where an isometric embedding of T into ℓ_1 would provide the correct answer, $v_a(b)$ is simply given the value 0 or 1 as appropriate. The $*$ is used to shorten the distance in places where the tree metric would have caused the distance to be too long (unless the original graph is a tree, there must be some edges that are present in the graph but absent in the tree). The division of cases is to ensure that the distances are not shortened too much, but instead by just the right amount.

The $*$ facilitates isometric embedding by allowing the creation of separate subspaces within the embedding space which can house different, non-interfering, separately-embedded parts of a graph. The non-numeric nature of the $*$ value disqualifies the embeddings generated by this technique from being directly used in many applications for which a purely numerical ℓ_1 embedding would have been suitable. It also creates embedding spaces of $\mathcal{O}(n)$ dimensions.

A summary of the vital statistics of the embedding algorithms presented in this section, as well as for the yet-to-be-presented embedding algorithm presented in this article, can be found in Table 2. The first column of the table is for Algorithm 1. The second column is for the derandomized ℓ_1-version of Bourgain's Theorem presented in [16] which is hard to scale to large maps and produces a high-dimensional embedding. The third column summarizes the Squashed Cube Conjecture [39] which has perfect fidelity, but is not

	Alg. 1	Bourgain [16]	Squashed [39]	Springs [12]
Type	ℓ_1	ℓ_1	ℓ_1	ℓ_2
Preprocessing	$\mathcal{O}(n^2)$	$\omega(n^2 \log n)$	$\mathcal{O}(n^2)$	incremental
Dimensions	$\mathcal{O}(\sqrt{n})$	$\Omega(n)$	n	6
Distortion	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	1	∞
vs. Alg. 1		preprocessing time, distortion, more dimensions	non-numeric "$*$", more dimensions	limited scale

Table 2: Comparison of embedding algorithms.

usable because of its non-numeric coordinate values. The fourth column is for the spring-optimization model [12].

2.4 Graham-Winkler Graph Factoring

The embedding algorithm presented in this article is based on Graham and Winkler's algorithm for isometric embeddings of graphs into Cartesian products of graphs, which is discussed in [21]. Graham and Winkler's algorithm is described below, but first a few definitions are needed. The definitions are all relative to some graph $G(V, E)$.

If the edge $e \in E$ has $a, b \in V$ as its endpoints, then the vertices of the graph can be partitioned into three sets. Those vertices which are closer to a than to b, according to the graph distance metric $\mathrm{d_G}(\cdot, \cdot)$, are called G_a. The collection G_b contains those vertices which are closer to b than to a, and $G_=$ contains the vertices equidistant from the two endpoints of the edge.

The symmetric, and reflexive, but non-transitive relation θ can be defined on the edges of a graph $G(V, E)$ by saying:

$$\theta(e) := \{(x, y) \in E \mid x \in G_? \wedge y \notin G_?\}$$

where $G_?$ is either G_a or G_b or $G_=$, but only one of them at once. Worded differently, this non-transitive relation says that two edges $e = (a, b)$ and $e' = (x, y)$ are related if the partition of vertices defined by e separates the two endpoints of the edge e'.

The transitive *Djoković-Winkler relation* is defined as the transitive closure of θ and is rendered symbolically as $\hat{\theta}$. An example of θ being computed is found in Figure 1 and the corresponding $\hat{\theta}$, obtained by taking the transitive closure of the items seen in the earlier figure, is displayed in Figure 2. In the first figure mentioned, each subfigure shows a class $\theta(e)$ where each edge e is shown with a solid red line and the other members of $\theta(e)$ have dotted red lines.

The Graham-Winkler factorization algorithm consists of the following steps. Given a connected graph G as input, the first step is to compute the relation $\hat{\theta}$. We can say that $\hat{\theta} := \cup_i^k E_i$ where the sets of edges E_i are the equivalence classes of $\hat{\theta}$.

For each of the k classes, a corresponding factor graph G_i is created. The factor graph G_i is created by first subtracting the edges of E_i from G. Each of the connected components that results from the subtraction is then contracted to a vertex. An edge is put between

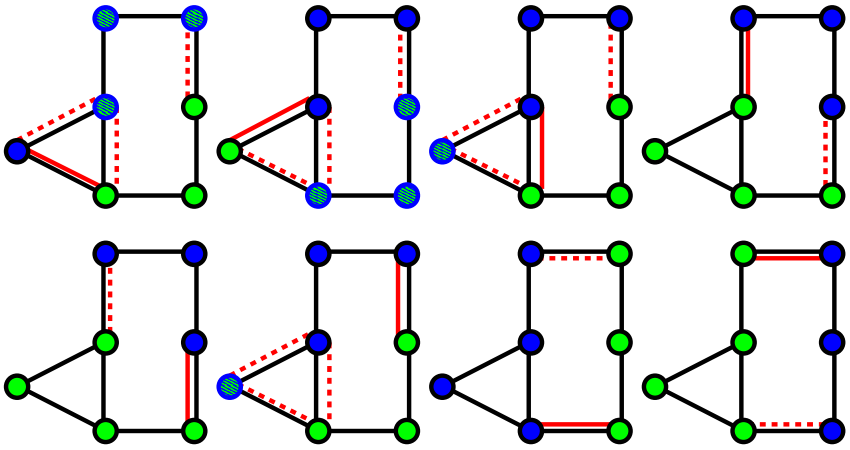


Figure 1: An example of computing the non-transitive relation θ. In each, the solid red line is e and the dashed red lines are the other members of $\theta(e)$.

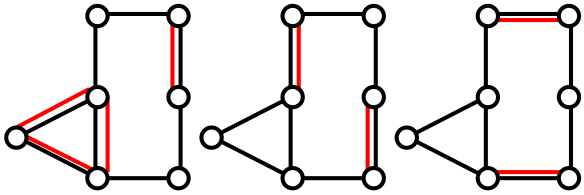


Figure 2: The three transitively-closed equivalence classes of $\hat{\theta}$.

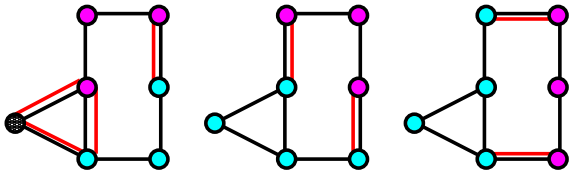
two of the resulting vertices of G_i if the vertices of the corresponding components share an edge of E_i in the original graph G.

The resulting graphs G_i are the unique prime factors[2] of G with respect to the standard graph Cartesian product. This fact is proven in [21]. An illustrated example is given in Figure 3.

The graph G is said to embed isometrically into the Cartesian product of its factors G_i because the distance between $a, b \in V$ is equal to the sums of the distances between the points corresponding to a and b in each of the respective factors. Please see Figure 4. In that illustration, the distance between the red vertex and the blue vertex in the original graph (the left-most one) is equal to the sum of the distances between the corresponding vertices in the factors.

This factoring algorithm was preceded and broadly prefigured by [38]. The ideas in that earlier work are generally the same, but the algorithm presented could only take a subset of the universe of undirected graphs (the class of hypercube embeddable graphs) as input. The Graham-Winkler algorithm was later made faster and more efficient in works

[2] A graph is a *prime factor* in this context if the factorization algorithm outputs it unchanged after receiving it as input.



(a) The equivalence classes of $\hat{\theta}$ with color-coded components.

(b) The resulting factors with color-coded vertices.

Figure 3: The three factors of the example graph from Figure 1. In each case, the magenta vertices are in G_a, the cyan in G_b, and the gray in $G_=$. The lower figure shows the results of contracting vertices of the same color together.

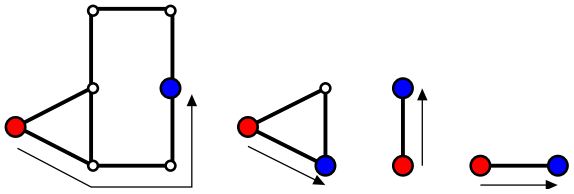


Figure 4: A distance measurement.

such as [24] and [4]. There is also a very thorough discussion in [14].

The prime factorization idea has also found expression in algorithms not directly related to the Graham-Winkler algorithm, as in [5] which runs faster than the algorithms just cited.

These factorization algorithms are relevant to a discussion about building embeddings, because if G has prime factors G_i, then the direct product of the individual embeddings of the graphs G_i is an embedding of G. The implication is that a factorization, if there are enough factors and they are small enough, can begin to be thought of as an embedding into a space where the coordinates are graphs.

The algorithms just referenced would be usable as subroutines in an isometric embedding algorithm if the factors were all small enough to be embedded by some method. Unfortunately, in realistic scenarios this is never the case. In fact, even graphs as everyday as loops of odd length are prime [14], so these isometric factorization algorithms do not tend to produce results which are helpful in-and-of themselves from an embedding perspective. If something could be done to guarantee smaller factors, then there would be hope for adapting factoring algorithms to our goals.

In [22] exactly that is done, and constant-distortion embeddings of particular classes of graphs into ℓ_1 are demonstrated. Regrettably, the types of graphs on which the approach provably works, series-parallel and outerplanar graphs, are not reasonable models of road

networks.

In Section 4 we successfully modify the Graham-Winkler algorithm for use with road networks.

3 Road Network Definition

A few ad hoc definitions and several standard ones are required in order to describe the road network model used in this article. The standard definitions are given in context where possible, otherwise they can be found in footnotes in this section. The new definitions are given in the next two paragraphs.

We say that a subgraph T is *tree-like* if T is a rooted tree, none of its edges participate in simple cycles in G, and $G - T$[3] and T only share the vertex which is the root of T.

A *nested loop* is an isometric subspace[4] whose vertices can be given the labels $u_0, \cdots, u_{m-1}$ and $v_0, \cdots, v_{m-1}$ so that v_i is adjacent to $v_{(i+1 \mod m)}$, u_i is adjacent to $u_{(i+1 \mod m)}$, and u_i is adjacent to v_j only if $i = j$.

Let $\mathcal{T}$ be a collection of tree-like subgraphs, let $\mathcal{N}$ be collection of nested loops, let B be the edges which occur exactly once in some fixed *minimum weight isometric cycle basis*[5] of G, and let X a minimal set of edges whose removal is sufficient to eliminate all *edge crossings*[6] in some fixed drawing of G (the drawing is described below in Condition 3).

The definition of a *road network* used in this work is that of a graph G, with n vertices, drawn uniformly randomly from a particular family of random graphs, the family of graphs is called $\mathcal{G}(n)$ and its members are:

1. (a) Connected;
   (b) Of bounded degree;
   (c) Bipartite;
   (d) Of diameter $\Theta(\sqrt{n})$ with $|B| = \Theta(\sqrt{n})$;
2. (a) Such that there exist collections $\mathcal{T}$ and $\mathcal{N}$ where

$$G' := G - \mathcal{T} - \mathcal{N}$$

   so that G' is free of nested loops and tree-like subgraphs;

[3] The *difference of two graphs*, say $G_1 - G_2$, is the graph that results from subtracting the adjacency matrix of G_2 from that of G_1. The difference of a graph and a collection of graphs, say $G - \mathcal{C}$ is the result of subtracting every member of $\mathcal{C}$ from G.

[4] A subspace in which the distances between the vertices are the same in the subspace as they are in the main space.

[5] A cycle is *isometric* if the distances between the vertices of the cycle are the same when paths are restricted to the edges of the cycle as they are when all of the edges in the graph can be used. A *cycle basis* of G is a collection of cycles that is sufficient to generate any cycle in the graph by taking the symmetric difference of one or more members of the basis. A minimum weight isometric cycle basis is a cycle basis consisting only of isometric cycles that contains the minimum number of edges among all possible isometric cycle bases.

[6] In a drawing of a (presumably non-planar) graph in two dimensions, an *edge crossing* is the circumstance of an edge in the drawing coinciding with another at a place other than at one of its endpoints. A *crossing edge* is an edge that participates in an edge crossing.

(b) Such that $\sum_{T\in\mathcal{T}}|T|, \sum_{N\in\mathcal{N}}|N| = \mathcal{O}(\sqrt{n})$;

3. Able to be rendered as a two-dimensional drawing where

   (a) All edges have unit length;

   (b) Such that $|X| = \mathcal{O}(\sqrt{n})$;

   (c) The interior of every isometric cycle of circumference m has m vertices in its interior.

The graphs are assumed to be connected in Condition 1a it is possible to drive from any place to any other place in the network via its roads. The boundedness of the degree of the graphs in Condition 1b is a physical constraint, the degree is the maximum number of roads that meet at any one intersection. The requirement in Condition 1c that every graph in $\mathcal{G}(n)$ is bipartite is not too onerous: If an input graph that one wishes to work on is not initially bipartite, it can be made so by subdividing (halving) each of its edges. That is true because one definition of a *bipartite graph* is one in which all cycles are of even length.

Making the graphs bipartite simplifies the arguments given in Section 4.2. In the discussion of the Graham-Winkler graph factoring algorithm in Section 2.4, we discussed the three sets G_a, G_b, and $G_=$ which help to define the relation θ. In a bipartite graph, $G_=$ is empty for every edge e because no vertex is equally distant from both endpoints of the edge. When $G_=$ is empty, $\theta(e)$ separates the graph into two pieces [14, Chapter 18], which simplifies the discussion.

The diameter and boundary are discussed in Condition 1d. The collection B is the set of edges that bounds G', both internally and externally. This set is imagined to represent the boundary of the city, county, or continent. The requirements that the length of the boundary is $\Theta(\sqrt{n})$ and that the *diameter*[7] of the graph is $\mathcal{O}(\sqrt{n})$ jointly suggest the "flatness" of the graph. By analogy to a circular figure drawn on a flat surface, the circumference $|B|$ and diameter of the road network are both roughly proportional to the square-root of the area.

The tree-like graphs represent roads ending in cul-de-sacs, but because of the fact that, $\sum_{T\in\mathcal{T}}|T| = \mathcal{O}(\sqrt{n})$ there are not too many of them. These subgraphs are discussed in Condition 2b largely for completeness: their presence does not play much of a role either positive or negative in the analysis or the performance of the algorithms. They are mentioned and given a name here just so they can be explicitly ignored in Section 4.

The limitation of $\sum_{N\in\mathcal{N}}|N| = \mathcal{O}(\sqrt{n})$ on the collection of nested loops, given in Condition 2b, is a reasonable assumption for a road network. As described before, these subspaces are formed when there is one loop of streets contained in another loop of exactly the same length with an additional bunch of streets – all of the same length – connecting the two loops at identical intervals. Because the inner and outer loops must be precisely the same length, these subspaces tend not to occur in real road networks. Much like the tree-like subspaces just discussed, nested loops are mentioned for the sake of completeness and given a name so that they can be explicitly ignored.

After removing the tree-like subgraphs and the nested loops, the graph $G' := G - \mathcal{T} - \mathcal{N}$ given in Condition 2a is the object that we are really interested in analyzing.

[7] The diameter of a graph is the maximum distance between any two vertices in the graph: $\max_{a,b\in V} \mathrm{d_G}(a,b)$.

One of the goals expressed in Section 1 is create embeddings which capture every point on the road network, not just the intersections. The existence of a rendering of the graph with unit-length edges given in Condition 3a makes this possible.

The graphs are not planar, but the number of edge crossings is limited to $\mathcal{O}(\sqrt{n})$ in Condition 3b. That condition is backed by an empirical study of road networks in the United States [15]. This condition primarily finds its use in Lemma 1 where there limitation on the non-planarity of road networks is used to put an upper bound on the sizes the sets $\theta(e)$.

Please see Figure 5 for picture of an imagined road network with the various features highlighted. Figure 5(a) shows the original network. Figure 5(b) shows the network with vertices added to make the graph bipartite and to make all edges roughly the same length. Figure 5(c) shows two tree-like subgraphs which are members of $\mathcal{T}$. Figure 5(d) shows G'. Figure 5(e) shows the boundary B. G' and the members of $\mathcal{T}$ can be taken together to reconstitute Figure 5(b).

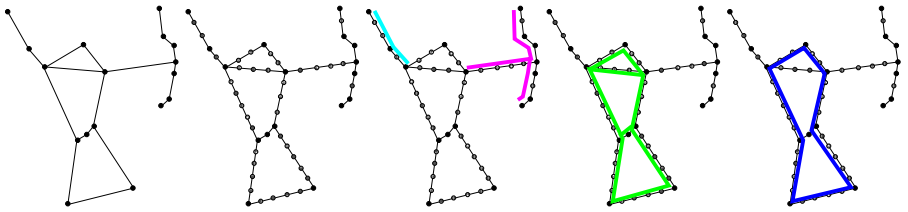


Figure 5: An example road network. Moving from left to right: The original network G is shown, it is then subdivided, the tree-like portions are identified, G' is found, then finally the boundary of G' is found.

4 Algorithms for Embeddings

The main embedding algorithm is developed in Section 4.1. There, it is shown that the embedding algorithm can a take a road network (as defined in Section 3) as input and produce an ℓ_1-embedding as output. We show in Section 4.2 that the embedding has the property that for *almost every pair*[8] of vertices $a, b \in V$, the distortion of the embedding distance relative to the real distance, $\frac{\mathrm{d_e}(a,b)}{\mathrm{d_G}(a,b)}$, is bounded by a constant factor. In Section 4.3, we provide a heuristic embedding-construction algorithm that requires less time than the algorithm.

[8]Throughout this section and the next, we make repeated use of this and similar phrases. Here we are using the term in its standard asymptotic-probabilistic sense: the probability that two randomly chosen vertices (in a randomly chosen road network $G \in \mathcal{G}(n)$) violate the property goes to zero as $n \to \infty$. We provide footnotes throughout this section clarifying the usage of these terms.

4.1 The Embedding-Construction Algorithm

The embedding algorithm that is described in this section is based on the Graham-Winkler graph factorization algorithm discussed in Section 2.4.

As mentioned in that section, the Graham-Winkler factorization algorithm suffers from the deficiency of not being able to fully digest most road networks. Because the prime graphs that result from the factorization tend to be too large to trivially embed, the factoring algorithm cannot be directly applied to embedding problems. The solution that we present is simple to state but requires some analysis to prove: instead of forming the factor graphs G_i by subtracting the equivalence classes $\hat{\theta}(e_i)$ of the Djoković-Winkler relation $\hat{\theta}$, we subtract a subset of the classes of the non-transitive relation θ.

Recall from Section 2.4 that the relation $\hat{\theta}$ is computed by taking the transitive closure of θ. The reason that the transitive closure is used in the unmodified Graham-Winkler algorithm is because doing that ensures that each edge in the graph belongs to exactly one equivalence class in the relation.

A full proof that Graham-Winkler provides an isometric factorization can be seen in [21] and an informal sketch is given in Section 2.4, but we will return briefly to that topic to help illuminate the design of our constant-distortion embeddings.

If the edge (a, b) does not appear in any relation, then a and b are in the same connected component in $G - E_i$ for all i. That implies a zero distance between the respective projections of a and b into every factor G_i, which implies a distance of zero in Cartesian product of the factors.

On the other hand, if an edge belongs to more than one class, then the distance between the two endpoints of that edge in the factorization will be strictly more than one. That is true because a and b will be in different connected components in more than one $G - E_i$ so that the respective projections of a and b will have non-zero distance in more than one G_i. The only way that a factorization can be isometric is if each edge belongs to one and only one class, which implies that the relation $\hat{\theta}$ must be transitive.

Based on this, it can be understood that building an embedding based on the non-transitive relation θ instead of the transitive relation $\hat{\theta}$ guarantees that in general we will *not* get an isometric embedding.

The payoff for using θ and surrendering the hope of a non-distorting embedding is that the individual factor graphs G_i are much smaller. In fact, because the input graph is required to be bipartite, the set $G_=$ is empty for every edge $(a, b) \in E$, so that the contraction of each $G_i := G - \theta(e_i)$ is as small as it can be: each is just a K_2 graph. This is the rationale for requiring that the graphs in $\mathcal{G}(n)$ are bipartite.

If graphs were allowed to be non-bipartite then the subtraction and contraction process might result in a K_2, K_3, or P_3 graph, instead of always a K_2. K_2 graphs are preferable because they are easy to embed: graph vertices that project to one vertex of the K_2 receive a value of 0 and graph vertices that project to the other vertex of the factor receive a value of 1. The embedding of G is then just the direct product of the embeddings of the numerous K_2 graphs, which results in an approximate embedding of the road network into ℓ_1.

The embedding algorithm is the following. First, the algorithm takes as input some graph $G := G(V, E)$, which belongs to the family of graphs $\mathcal{G}(n)$ as defined in Section 3. Next, the classes $\theta(e)$ of the non-transitive relation θ are computed on each edge $e \in E$, and this collection of these classes is given the name S. The set S is a collection of $|E|$

Algorithm 1 The Embedding Algorithm.

Require: $G(V, E)$

$S \leftarrow \emptyset$
for all $e \in E$ **do**
    $S \leftarrow S \cup \theta(e)$ ▷ Please see page 9 for discussion about $\theta(e)$.
end for

$S' \leftarrow \text{Cover}(E, S)$ ▷ Repeatedly use the BMCP problem from [28].

for all $\theta(e) \in S'$ **do**
    $G' \leftarrow G - \theta(e)$
    $G_a \cup G_b := G'$ ▷ G' consists of two components
    ▷ $\cdots$ called G_a and G_b.
    for all $v \in V$ **do**
        if $v \in G_a$ **then**
            ▷ If above the cut, then concatenate a zero.
            $\text{EMB}(v) \leftarrow \text{EMB}(v) \mid 0$
        else
            ▷ Otherwise, concatenate a one.
            $\text{EMB}(v) \leftarrow \text{EMB}(v) \mid 1$
        end if
    end for
end for

return $\text{EMB}(G)$ ▷ Return the embedding.

subsets of the edge set E. After S is computed, a subset of S called S' is computed that has the property that it covers all of the edges in E at least once while not covering individual edges more than $\mathcal{O}(1)$ times on average. (The fact that this can be done is proven in Lemma 1.)

The next step is to compute the sets G_a and G_b associated with each edge set $\theta(e_i)$ in the cover S'. (Remember that G_a is the set of $v \in V$ that are closer to a than to b according to the graph's distance metric.) Each of the sets $\theta(e_i)$ contributes one dimension to the embedding space: $v \in V$ gets a value of 0 or 1 in the dimension associated with a particular $\theta(e_i)$ depending on whether it is in the associated G_a or in G_b. A pseudo-code description of this process can be found in Algorithm 1.

With an embedding into ℓ_1 in hand, the estimated distance between $a, b \in V$ is $\|\text{EMB}(a) - \text{EMB}(b)\|_1$ where the norm is taken by summing entries of the difference vector. In our specific context, where the coefficients are bits, the distance is $\|\text{EMB}(a) \texttt{ xor } \text{EMB}(b)\|_1$, because taking the exclusive-or of the bit vectors is the equivalent of taking the difference. For bit vectors, the norm can be taken by finding the Hamming weight or "popcount" of the register holding the exclusive-or of the two operands.

4.2 Analysis

We will show that Algorithm 1 provides a constant-distortion embedding of road networks into ℓ_1, at least between *almost every* pair of vertices.

In the following arguments we will assume that the tree-like subgraphs mentioned in the road network model given in Section 3 are not present. Because trees are convex with respect to the graph metric, the ℓ_1-embedding of the graph $G \in \mathcal{G}(n)$ is simply the direct product of the ℓ_1-embeddings of $G - \mathcal{T}$ and all $T \in \mathcal{T}$. Trees can be embedded into ℓ_1 with no distortion [26], so appending the subspaces for the trees onto the embedding space for $G - \mathcal{T}$ will cause no distortion.

Because of these facts, imagining that our road networks are free of tree-like subgraphs will not invalidate any of the results, but it will simplify the discussion somewhat.

Lemma 1 *Each edge $e \in E$ of $G(V, E)$ is covered at least once and an average of $\mathcal{O}(1)$ times by members of the set S' from Algorithm 1.*

Proof: Two observations are needed:

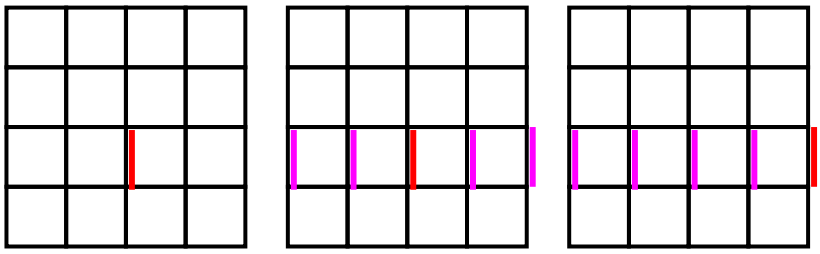
i The **first observation** is that the optimal S' is of size $\mathcal{O}(\sqrt{n})$.

It is shown in [14, Chapter 18] that for every $e \in E$, the set of edges $\theta(e)$ is an edge cut that divides G into two pieces.

Let us initially assume that no members of S' contain an edge that is also contained in a nested loop (recall that nested loops were previously defined).

Under that assumption, each member of S' contains at least one edge that is in the boundary set B. Were that not the case, G would not be disconnected by the edge set $\theta(e)$. Any edge that appears in the cycle basis more than once must be enclosed in some larger loop, so cutting through at least one edge in the boundary set is required for disconnecting the graph.

The reflexivity of θ implies that the set $S' = \{\theta(e) \mid e \in B\}$ is sufficient to cover the edges E. The road network definition requires that $|B| = \mathcal{O}(\sqrt{n})$, which means that no more than that many sets of edges are required to cover all of the edges of the graph. Please see Figure 6 for an illustration of the use of reflexivity to cover non-boundary edges with sets originating at the boundary.



(a) An edge e to cover. (b) The set $\theta(e)$. (c) Use the boundary.

Figure 6: Using reflexivity to cover an edge with a set from the boundary.

Now considering nested loops: the set of edge sets $\{\theta(e) \mid e \in \mathcal{N}\}$ sufficient to cover all edges participating in nested loops and it is given that $\sum_{N \in \mathcal{N}} |N| = \mathcal{O}(\sqrt{n})$.

Combining the various facts, we see that $S' = \{\theta(e) \mid e \in B \vee e \in \mathcal{N}\}$ is sufficient to cover all edges $e \in E$.

ii The **second observation** is that the $\mathbb{E}[|\theta(e)|] = \mathcal{O}(\sqrt{n})$.

Recall that the set X is defined on page 12 to be a fixed minimal collection of crossing edges whose removal will make G planar: let us temporarily assume that G is free of nested loops and that $X = \emptyset$.

Let $e \in E$ be any edge and let $e' \in \theta(e)$. Also let $e := (a, b)$ and $e' := (a', b')$ with $a' \in G_a$ and $b' \in G_b$ (a' closer to a than to b, b' closer to b than to a). The edges e and e' are at opposite ends of the cycle $C := (a, b, \cdots b', a', \cdots, a)$, where the ellipses respectively represent shortest paths from b to b' and from a' to a.

C can be written as the symmetric difference of a collections of isometric cycles, and because the isometric cycles are oriented randomly with respect to the motion of $\theta(e)$, each contributes an expected number of edges to C which is proportional to its own circumference. C can be written as the symmetric difference of a collection of isometric cycles, where the sum of the circumferences of those cycles has expected value $\Theta(\mathrm{d_G}(a, a'))$. An illustration of how $\theta(e)$ moves across the graph is found in Figure 7.

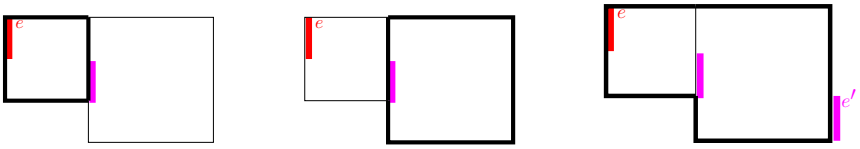


Figure 7: The procession of $\theta(e)$ across the graph: the red line is e and the magenta lines are other edges in $\theta(e)$. In the left-most figure, an isometric that includes e is shown. In the next figure, a neighboring isometric cycle is shown. Finally, the symmetric difference of the cycles, C, is shown with e and e' on either end of it.

For every $e \in E$, there are two edges in $\theta(e) \cap B$ such that the symmetric difference of the respective cycles between e and each of them, call them C_1 and C_2, contains all of the edges in $\theta(e)$ on its boundary or in its interior (here the notions of boundary and interior are with respect to the fixed rendering of G). The diameter of the graph is $\mathcal{O}(\sqrt{n})$, so the respective circumferences of C_1 and C_2 are $\mathcal{O}(\sqrt{n})$, and so is that of their symmetric difference. The sum of the circumferences of the isometric cycles that generate the two cycles is $\mathcal{O}(\sqrt{n})$, so by Condition 3c from Section 3 the number of edges inside of those cycles, and therefore in $\theta(e)$, is $\mathcal{O}(\sqrt{n})$.

Now let $|X| = \mathcal{O}(\sqrt{n})$ instead of $|X| = 0$. We have just shown that exclusive of crossing edges, $|\theta(e)| = \mathcal{O}(\sqrt{n})$. In the worst case, the length of the shortest walk from an encounter with X to the boundary is once again $\mathcal{O}(\sqrt{n})$ because of the diameter of the network. The probability of any edge being crossed by a member of

X, including any edge in $\theta(e)$ or on the isometric cycles whose interiors contain $\theta(e)$, is proportional to $\frac{1}{\sqrt{n}}$.

Therefore, the expected size of any one of these cut sets can be expressed by the following recurrence:

$$\begin{aligned} T(n) &:= \mathcal{O}\big(\sqrt{n}\big) + \mathcal{O}\bigg(\frac{1}{\sqrt{n}}\bigg)T(n) \\ &= \mathcal{O}\big(\sqrt{n}\big) + \mathcal{O}(1) + \mathcal{O}\bigg(\frac{1}{n}\bigg)T(n) + \cdots \\ &= \mathcal{O}\big(\sqrt{n}\big) \end{aligned}$$

from which we conclude

$$\mathbb{E}[|\theta(e)|] = \mathcal{O}\big(\sqrt{n}\big)$$

when $e \notin X$. When $e \in X$, then this expectation is larger, but only by a $T(n)$ term.

We can now readmit nested loops: we observe that $\sum_{N \in \mathcal{N}} |N| = \mathcal{O}(\sqrt{n})$, so that no more than that many edges can eventuate due to nested loops.

Taken together, observations i and ii prove that an optimal covering of E by the sets $\theta(e)$ has no more than $\mathcal{O}(\sqrt{n})\mathcal{O}(\sqrt{n}) = \mathcal{O}(n)$ edges, on average, when edges that are present in more than one class are multiply-counted.

In order to show that the covering S' computed in Algorithm 1 has a similar number of edges, it is sufficient to show that the covering produced is within a constant factor of optimal in terms of size. This can be done by couching the problem of computing the covering set in terms of the *budgeted maximum coverage problem* ("BMCP"). That problem asks for the best cover of a set (in our case E) using subsets of varying costs (in our case, each $\theta(e)$ having cost $|\theta(e)|$) with the restriction that the total cost of the set of covering sets (in our case $\sum_{\theta(e) \in S'} |\theta(e)|$) cannot exceed a given budget (in our case some multiple of $|E|$).

The optimization problem in the algorithm can be done if a solver for the BMCP is available as a subroutine. Each $\theta(e)$ is given a cost equal to the number of edges that it contains, while the budget starts at $|E|$ and is repeatedly doubled, and the subroutine re-invoked, until a set of edge sets is found that completely covers E. An algorithm that will solve the BMCP with a cost (total number edges) that is within a constant factor of optimal is given in [28]. □

Lemma 1 shows that it is possible to cover each edge of the graph at least once and an average of a constant number of times with edge sets which come out of the relation θ. There are three useful, additional facts which come out of the proof of the first lemma.

Corollary 1 *Algorithm 1 creates an embedding that is $\mathcal{O}(n\sqrt{n})$ bits in size.*

Proof: Observation i from Lemma 1 says that $|S'| = \mathcal{O}(\sqrt{n})$. The number of dimensions of the embedding is equal to $|S'|$, so each of the n vertices requires that many bits in the embedding. □

Looking at the size of an optimal set $|S'|$ and noting that the algorithm gets within a constant factor of this value gives a bound of $\mathcal{O}(n\sqrt{n})$ on the number of bits consumed by an embedding. In all probability, this bound is quite loose because most of the vectors representing each graph vertex are quite sparse. In theory they can be held to fifty percent average occupancy (in terms of non-zeros), but in practice they are quite a bit smaller than that.[9]

Corollary 2 *The worst-case overestimate of* $\mathrm{d}_G(a,b)$ *by* $\mathrm{d}_e(a,b)$ *is* $\mathcal{O}\Big(\frac{\sqrt{n}}{\mathrm{d}_G(a,b)}\Big)$.

Proof: Each $\theta(e) \in S'$ is associated with one dimension in the embedding ℓ_1. Since there are $\mathcal{O}(\sqrt{n})$ dimensions in the embedding, the measured distance between any two points cannot be more than $\mathcal{O}(\sqrt{n})$. □

Corollary 2 is another fairly straight-forward outflow from Lemma 1. Because the number of bits (or dimensions) comprising each vertex is bounded above by $\mathcal{O}(\sqrt{n})$, the embedding distance between two vertices can never be more than $\mathcal{O}(\sqrt{n})$. The worst-possible overestimate of the distance that the embedding can commit is the quotient of the largest measurable distance and the true distance.

Corollary 3 *Algorithm 1 runs in* $\mathcal{O}\big(n^2\big)$ *time.*

Proof: The computation of θ requires the computation of no more than $\mathcal{O}(n)$ shortest path trees, which can be done in $\mathcal{O}\big(n^2\big)$ total time.

An optimization procedure must be done to extract the cover set S' from the set of all sets $S = \{\theta(e) \mid e \in E\}$. The procedure given for doing that is to repeatedly call the $\mathcal{O}\big(n^2\big)$ subroutine for solving the BMCP, with the budget starting at $|E|$ and increasing by a factor of two until a cover is found. Each invocation of the solver requires $\mathcal{O}\big(n^2\big)$ time because there is a greedy selection step which computes the cost of $|S| = \mathcal{O}(n)$ sets, each of size $\mathcal{O}(\sqrt{n})$, and it iterates $\mathcal{O}(\sqrt{n})$ times. It was shown above that the optimal cover contains $\mathcal{O}(n)$ edges, and since the solver returns an answer within a constant factor of optimal, the solver will be called no more than a $\mathcal{O}\big(\log \mathcal{O}\big(\frac{n}{n}\big)\big) = \mathcal{O}(1)$ times.

□

Corollary 3 says that after having computed the set S in polynomial time, a close-to-optimal set S' can be found in polynomial time using the cited optimization algorithm.

Lemma 2 shows that *almost no*[10] pairs $a, b \in V$ have their distance overestimated in the embedding by more than a constant factor. That is done by looking at the mean and variance of the upward distortion. It is found that the average per-unit-length upward distortion (that is, $\frac{\mathrm{d_e}(a,b)}{\mathrm{d_G}(a,b)}$) is bounded above by some constant and the variance of that quantity due to any particular edge along a shortest path is constant. As the proportion of sufficiently long paths increases, as is the case when the map size goes to infinity, the Central Limit Theorem dictates that the variance of the upward distortion of *almost every* path goes to zero.

[9] Each cut $\theta(e)$ corresponds to a dimension in the embedding, and no more than half of the vertices will be "above" the cut (and thereby assume a non-zero value in that dimension). It is possible to count the number of vertices on either side of the cut and redefine "above" and "below" so that the majority of vertices assume values of zero in a particular dimension.

[10] The probability of randomly choosing $a, b \in V$ (in a randomly chosen road network $G \in \mathcal{G}$) such that the pair of vertices exemplify the property goes to zero as $n \to \infty$.

Lemma 2 *For almost every pair of vertices $a, b \in V$, the distance provided by the embedding given by Algorithm 1, $\mathrm{d}_e(a,b)$, overestimates the true distance, $\mathrm{d}_G(a,b)$, by at most a constant factor.*

Proof: This discussion takes place in the context of a randomly chosen graph $G(V,E)$ which is a member of $\mathcal{G}(n)$.

Let $f(n) \neq \mathcal{O}(1)$ be a function that assumes only positive values, such that as $n \to \infty$, *almost every* randomly chosen pair of vertices $a, b \in V$ has the property that $\mathrm{d_G}(a,b) > f(n)$. This can always be done: for example, let $f(n) \in o(\sqrt{n})$. For such an $f(n)$, *almost every* shortest path in a graph $G \in \mathcal{G}(n)$ is of length $f(n)$ or greater. The purpose of this function $f(n)$ in the argument is facilitate the statistical aspect of the argument: because the proportion of paths whose lengths are less than $f(n)$ goes to zero as $n \to \infty$, when we later prove facts about paths of length $f(n)$, we will be able to claim that those facts are true for *almost every* path. (The following might be a useful intuitive aide: if $f(n)$ is as described, if n as the area of a roughly circular flat shape, and if b is any point within that shape, then the proportion of the shape's area that is within $f(n)$ units of b will go to zero as n goes to infinity.)

Let us consider the random quantity $\max\left(1, \frac{\mathrm{d_e}(a,b)}{\mathrm{d_G}(a,b)}\right)$. We will compute the expected value of that quantity (that is, the expected distance-overestimate of the embedding), as well as the variance of that quantity. The expected value and variance will then be used to prove the statement in the context of a path of length $f(n)$ or more.

i We know from Lemma 1 that the average edge occurs $\mathcal{O}(1)$ times in the covering S' computed in Algorithm 1. That means that the average overestimate of the length of a randomly chosen edge is bounded above by $\mathcal{O}(1)$. That means that

$$\mathbb{E}\left[\max\left(1, \frac{\mathrm{d_e}(a,b)}{\mathrm{d_G}(a,b)}\right)\right] = \mathcal{O}(1)$$

so that the average overestimated distance between any two points is within a constant factor of the true distance.

ii Since $|\theta(e)| = \mathcal{O}(\sqrt{n})$, we can estimate the probability of a particular edge belonging to an edge set $\theta(e) \in S'$ as

$$p = \mathcal{O}\left(\frac{\sqrt{n}}{n}\right) \leq \frac{c_0}{\sqrt{n}}$$

where c_0 is some fixed constant. (In the expression above, the approximation symbol is meant to communicate the fact that the probability p is defined in terms of some function in $\mathcal{O}(1)$, we are giving an indication of the magnitude of the quantity p.) To estimate the magnitude of p, we are using the fact that $|E| = \mathcal{O}(n)$ and $\mathbb{E}\left[|\theta(e)|\right] = \mathcal{O}(\sqrt{n})$ so that for a randomly chosen graph, a randomly chosen $\theta(e)$, and randomly chosen edge, the probability of the edge being in $\theta(e)$ is $\leq \frac{c_0}{\sqrt{n}}$.

Given this bound on p, we can estimate the variance of the overestimate of the length of a single edge in the embedding by using the binomial distribution. We can say that the number of trials n' is

$$n' = \mathcal{O}\big(\sqrt{n}\big) \leq c_1\sqrt{n}$$

because there are only that many classes in S', as shown in the proof of Lemma 1. That gives a variance of

$$\begin{aligned}\text{Var}\left[\max\left(1, \frac{\text{d}_\text{e}(a', b')}{\text{d}_\text{G}(a', b')}\right)\right] &= n' * p * (1 - p) \\ &\leq c_1\sqrt{n} * \frac{c_0}{\sqrt{n}} * \left(1 - \frac{1}{\sqrt{n}}\right) \\ &\leq c_0 c_1 \left(1 - \frac{1}{\sqrt{n}}\right)\end{aligned}$$

where $(a', b') \in E$ is some edge and $\text{d}_\text{G}(a', b') := 1$. It can be seen that the variance of the overestimate of the length of an edge is well-defined and goes to a constant as $n \to \infty$.

Since observation ii shows us that the per-edge variance of the overestimate goes to a constant in the large n limit, the Central Limit Theorem implies that that

$$\text{Var}\left[\max\left(1, \frac{\text{d}_\text{e}(a, b)}{\text{d}_\text{G}(a, b)}\right)\right] \xrightarrow[n\to\infty]{} 0$$

where a and b are the two endpoints of a randomly chosen path of length $f(n)$ or more. Since the variance of this random variable goes to zero, observation i allows us to conclude that *almost every* path of length $f(n)$ is overestimated by only a constant factor. *Almost every* path is of length $f(n)$ or greater, so the statement is proven. □

Lemma 3 is the mirror half of Lemma 2. It shows that the per-unit-length underestimate of the distance given by the embedding (that is, $\frac{\text{d}_\text{G}(a,b)}{\text{d}_\text{e}(a,b)}$) is bounded above by a constant for almost every pair $a, b \in V$. The strategy employed in the proof is to diagnose when an underestimate of the distance can occur and model those occurrences using random graphs. Results in random graph theory are then invoked to show that those circumstances almost never occur.

Lemma 3 *For almost every pair of vertices $a, b \in V$, $\text{d}_e(a, b)$ underestimates $\text{d}_G(a, b)$ by no more than a constant factor.*

Proof: As in the proof of Lemma 2, the argument will be made in the context of a randomly chosen road network $G(V, E) \in \mathcal{G}(n)$.

Since every edge $e \in E$ is covered by S', an underestimate of the length of a path can occur only if some cut $\theta(e)$ is non-convex with respect to the distance metric of G. Such a non-convex cut allows a shortest path, say between a and b to cross the cut twice or more. That means that both a and b could have the same value, either 0 or 1, in the dimension of the embedding associated with $\theta(e)$. The result is that when the distance between a and b in the ℓ_1-embedding is computed, it is possible that none of the edges in the intersection of a shortest path and the cut $\theta(e)$ contribute anything to the distance in the embedding.

The strategy used in this proof is to show that the odds of that happening to more than a constant fraction of the edges in a randomly chosen shortest path are small for a large-enough road network.

In order to proceed, we need two additional definitions from graph theory. The standard model of a *random graph*, as in [9], is a collection of m vertices with each of the $\binom{m}{2}$ possible undirected edges between those vertices present with probability p and not present with probability $1-p$. This family of graphs is parametrized by m and p so that $G(m,p)$ is some random graph fitting the description just given. The second term that needs to be introduced, *clique*, refers to a maximal complete subgraph.

It is possible to use random graphs to bound the embedding's underestimate of the true distance between two randomly chosen points in a randomly chosen road network.

Each edge set in S' is of size $|\theta(e)| = \mathcal{O}(\sqrt{n})$, so we can say that the probability of two randomly chosen edges in a shortest path between a and b both belonging to the same set $\theta(e)$ is

$$p = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right) \leq \frac{c_2}{\sqrt{n}}$$

where c_2 is a constant greater than the average number of times each edge on the shortest path is covered by the set of edge sets S'.

If the length of the path under consideration is $m := \mathrm{d_G}(a,b)$, then there is a corresponding random graph $G(m,p)$ where and the probability p is as stated above. Each vertex in the random graph represents an edge in the shortest path, and the presence of an edge between two vertices in the random graph corresponds to the circumstance where the two associated edges in the path belong to the same class $\theta(e)$. A clique of size k represents a group of that many edges in the path that all belong to the same $\theta(e)$[11]. Please see Figure 8 for an example.

Let $g(k)$ be the average reduction of the measured distance due to each clique of size k. With that, we have

$$c_2\mathrm{d_G}(a,b) - \sum_{k=2}^{X_m} g(k)Y_k \leq \mathrm{d_e}(a,b)$$

where c_2 is as previously defined, and where the two random variables

$$Y_k := \{\text{The number of cliques of size } k \text{ in } G(m,p)\}$$
$$X_m := \{\text{The size of the largest clique in } G(m,p)\}$$

[11] It might be objected that this random graph model is not appropriate in this case because of the requirement that the edge probabilities must be independent and identically distributed.

The question of dependence is addressed with the following observation. Let x, y, z be three vertices of the random graph. The set S' is chosen to cover the edge set E while minimizing the total weight of S'. For a randomly chosen road network and a randomly chosen shortest path, the existence of an edge xy in the graph militates against the existence of another edge yz in the graph because that would imply that the edge in the road network that corresponds to x is multiply-covered by S', something which is explicitly discouraged by the method of its construction. Of course, the absence of an edge xy does not increase the probability of an edge yz beyond $p \leq \frac{c_2}{\sqrt{n}}$. That means that assuming independence of edge probabilities in the random graph actually overestimates the magnitude of the distortion.

The question of identical distribution is answered by choosing a constant c_2 so that each edge probability is bounded above by $\frac{c_2}{\sqrt{n}}$. That allows us to pessimistically assume that the edge probabilities are identical. Doing this will once again give us an overestimate of the magnitude of the embedding's distortion.

Therefore, we get an overestimate of the magnitude of the distortion by modeling it with a random graph with i.i.d. edge probabilities, and because we are trying to put an upper bound on the magnitude of the distortion this is permissible.

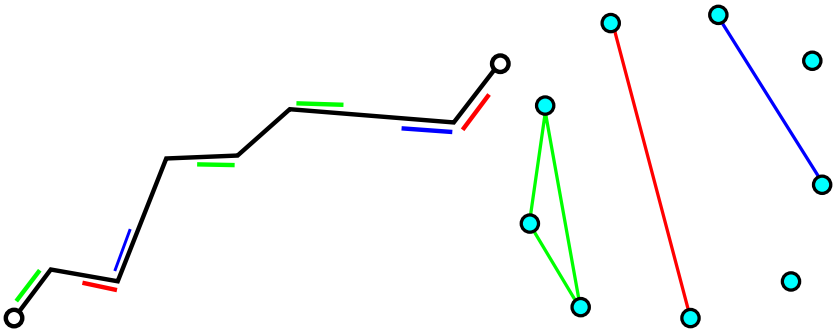


Figure 8: Modeling edge interactions with a random graph. The three green edges on the shortest path on the left all belong to some class $\theta(e_0)$, so the graph on the right has edges between the three corresponding vertices. Similarly, the two blue edges belong to another class $\theta(e_1)$ and the two red edges belong to yet another class $\theta(e_2)$.

are defined in [10].

It is shown that

$$X_m \xrightarrow[m\to\infty]{} 2\frac{\log m}{\log \frac{1}{p}}$$

in the same work, and the right-hand side is equivalent to

$$2\frac{\log m}{\log \frac{\sqrt{n}}{c_2}}$$

with $m = \mathcal{O}(\sqrt{n})$. That implies that when n is large enough, the size of the largest clique is (*in probability*[12]) no more than two. That further implies that when $m := \mathrm{d_G}(a, b)$ is non-constant vis-à-vis n, the statement

$$c_2 m - g(2)Y_2 \leq \mathrm{d_e}(a, b)$$

is true *in probability*.

We can get a bound on the difference $c_2 m - g(2)Y_2$ by examining the number of isolated vertices in the random graph. A vertex is isolated in the random graph if and only if the corresponding edge in the shortest path is not in the same class as any other edge in the shortest path.

In [9] it is shown that if x_m is the number of isolated vertices in a random graph with m vertices, then $\mathbb{E}[x_m] = m(1-p)^{m-1}$. Given the bound on the probability p stated above, this implies that $\mathbb{E}[x_m] \geq \frac{m}{\exp(c_2)}$ as n and m go to infinity.

We then have

$$c_2\left(1 - \frac{1}{\exp(c_2)}\right)m \leq c_2 m - g(2)Y_2 \leq \mathrm{d_e}(a, b)$$

[12]The probability of truth goes to 1 as $n \to \infty$.

in expectation for large n and m.

Now looking at the variance, we have

$$\text{Var}[Y_2] = \binom{m}{2} \frac{c_2}{\sqrt{n}} \left(1 - \frac{c_2}{\sqrt{n}}\right) = \mathcal{O}(m)$$

as $n \to \infty$, for $m = \mathcal{O}(\sqrt{n})$ and m non-constant. Because the variance is of size $\mathcal{O}(m)$, the standard deviation of Y_2 is in $\mathcal{O}(\sqrt{m})$. Since $g(2) \leq 2$, the standard deviation of $g(2)Y_2$ is also in $\mathcal{O}(\sqrt{m})$.

Because $\mathrm{d_e}(a, b)$ is bounded below in expectation by a positive fraction of $c_2 \mathrm{d_G}(a, b)$, and because the variance of the difference between $c_2 \mathrm{d_G}(a, b) := c_2 m$ and $\mathrm{d_e}(a, b)$ is bounded above by $\mathcal{O}(\sqrt{m})$, the probability of making up the order-m gap between the expected lower bound on $\mathrm{d_e}(a, b)$ and $o(m)$ is nil as n and m go to infinity. That means that the asymptotic probability of the underestimate $\frac{\mathrm{d_G}(a,b)}{\mathrm{d_e}(a,b)}$ being more than a constant is nil. □

Distortion	worst-case	almost every
Over-estimate	$\mathcal{O}\left(\frac{\sqrt{n}}{\mathrm{d_G}(a,b)}\right)$	$\mathcal{O}(1)$
Under-estimate	∞	$\mathcal{O}(1)$

Table 3: A summary of the worst-case and almost-certain distortions of distance measurements in embeddings produced by Algorithm 1.

The results of Lemmas 1, 2, and 3 and Corollary 2 are summarized in Table 3, and brought together in the shape of Theorem 1, which can be stated without further proof.

Theorem 1 *For a randomly chosen graph $G(V, E) \in \mathcal{G}(n)$ and a randomly chosen pair of vertices $a, b \in V$, the distance between the two vertices in the embedding $\mathrm{d}_e(a, b)$ distorts the true distance $\mathrm{d}_G(a, b)$ by no more than a constant factor, in probability.*

4.3 Faster Embedding Construction

The main embedding-construction algorithm, Algorithm 1, is for computing embeddings with good guarantees. It was shown in Corollary 3 that the construction time for those embeddings is polynomial in n. However, if translated directly from the pseudo-code, that algorithm would require quadratic time.

Algorithm 2 gives a heuristic algorithm for computing the embedding in roughly $\mathcal{O}(n\sqrt{n})$ time. Since that is the size of the embedding in bits, and since the embedding is created one bit at a time, that is probably as good as can be expected.

The embedding heuristic differs from the embedding algorithm in the way that it computes the set of classes S from which the covering $S' \subset S$ is found. Algorithm 1 computes $\theta(e)$ for every edge $e \in E$, whereas the heuristic Algorithm 2 attempts to do so for only $\mathcal{O}(\sqrt{n})$ edges. This is motivated by the fact that an optimal S' is of size $\mathcal{O}(\sqrt{n})$, as was shown in Lemma 1, which means that it is possible in principle to get away with an S of only that size.

Algorithm 2 The Embedding Heuristic.

Require: $G(V, E)$

1: $S, S' \leftarrow \emptyset$

2: **while** S' does not cover E **do**
3: $E' \leftarrow \{\text{A random subset of } E \text{ of size } \Theta(\sqrt{n})\}$
4: $S \leftarrow S \cup (\cup_{e \in E'} \theta(e))$
5: $S' \leftarrow \text{Cover}(E, S)$ ▷ Or, say $S' \leftarrow S$.
6: **end while**

⋮ ▷ Proceed as if at line 6 of Algorithm 1.

Without any special knowledge of how to pick the correct S, the approach taken in the heuristic is to randomly choose a set of edges that is (hopefully) of size $\mathcal{O}(\sqrt{n})$. The covering S' can then be extracted from S using either the BMCP-based approach discussed before, or by simply by choosing to let $S' = S$. Section 6.1 contains an experimental evaluation of Algorithm 2's execution time and output quality, which are both very good.

5 Algorithms for Linear Combinations

The actual vertices of the road network tend to be sparse in the embedding space: if the embedding is of size $\mathcal{O}(n\sqrt{n})$ bits, then the total number of points in the embedding space is more than exponential in the number of vertices in the graph. That suggests that merely being able compute combinations is not enough, a method is needed for projecting points in the embedding space back into vertices in the graph. Without such a method, there is no way to connect answers computed in the embedding space to the graph, which is crucial for many applications.

At first sight, a nearest-neighbor algorithm would seem to be the right choice for this goal, but the high dimensionality of the embeddings would strip most standard nearest-neighbors algorithms of their efficiency ("The Curse of Dimensionality").

A possible idea is to reduce the dimensionality of the embeddings using the Johnson-Lindenstrauss lemma [27, 13] or one of its ancestors or derivatives. The problem with this idea is the fact that most Johnson-Lindenstrauss type approaches tend to theoretically require Euclidean vectors as input. Because the embeddings presented in this article are ℓ_1 spaces, there is an incompatibility.

A more serious problem is the fact that Johnson-Lindenstrauss also requires preprocessing space and time that is exponential in $\frac{1}{\varepsilon^2} \log n$ where $(1 \pm \varepsilon)$ is the distortion of the lower-dimensional embedding versus the input. That means that if one wishes to have distortion that stays within ten percent of input embedding, preprocessing space and time that is one hundred times greater than the input size may be needed. For large maps this may be impractical.

Another issue to consider is the fact that we are working with ℓ_1 embeddings. Users

might be interested in finding types of "linear combinations" which are not directly accessible in ℓ_1. The mean of several points is an example of this.

The mean of k points is the point that minimizes the sum of the squared distances to each of the k points. In an ℓ_2 (Euclidean) embedding space, that point can be found by averaging the k points. In ℓ_1 that is not the case. Conversion between different types of spaces is a possibility, but that is costly in space and time and may incur too much distortion.

In Section 5.1, we present a method for computing linear combinations that avoids the problems just mentioned. The correctness of the algorithm is analyzed in Section 5.2 and its speed is discussed in Section 5.3. A faster heuristic for computing combinations and projecting them back into the graph is given in Section 5.4.

5.1 Combined Combination/Projection Algorithm

Algorithm 3 takes as input a graph $G(V, E)$, a starting vertex $s \in V$, an objective function estimator called Estimator, and a function SearchRadius which returns the search radius around the current vertex to be used at every step. The algorithm *almost certainly*[13] returns the vertex t_{G} in the graph closest to the abstract point t_{e} in the embedding that minimizes the objective value. The point t_{e} may or may not exist as a vertex in the graph G, but of course exists in the embedding space.

Algorithm 3 The Steepest-Descent Algorithm.

Require: $G = (V, E)$, s, Estimator, SearchRadius

1: $N_v \leftarrow \{s\}$

2: **while** $N_v \neq \emptyset$ **do**
3: $\quad v \leftarrow u \in N_v$ with smallest Estimator(u)
4: $\quad m \leftarrow$ The estimated distance to the target
5: $\quad N'_v \leftarrow N_v \cup \{$All vertices within SearchRadius(m) hops of $v\}$
6: $\quad N_v \leftarrow \{v' \in N'_v \mid \text{Estimator}(v') < \text{Estimator}(v)\}$
7: **end while**
return v

The algorithm is similar in spirit to the A^* algorithm [23]. This algorithm and A^* do essentially the same thing: they start from a given source vertex and the target vertex is searched-for by traversing the graph. At every step, a choice must be made about which branch to take (which edge out of the present vertex in the case of A^*, which vertex within the search radius in the case of Algorithm 3), and both algorithms use a heuristic distance estimator to determine which path is likely to be shortest.

The main difference is that Algorithm 3 looks at all of the vertices within some search radius and chooses the one most likely to be closest to the target, whereas A^* uses a search

[13] The probability of it not doing so in a randomly chosen $G \in \mathcal{G}(n)$ for a randomly chosen target vertex goes to zero as $n \to \infty$.

radius of one and looks only at the immediate neighbors of the present vertex and its predecessors. The other main difference is that A^* preserves the history of its traversal in its priority queue whereas Algorithm 3 is memoryless.

5.2 Correctness Analysis

This subsection is organized as follows: We will start by arguing that Algorithm 3 *almost certainly* returns the point $t_{\mathrm{G}} \in V$ closest to $t_{\mathrm{e}} \in \mathrm{EMB}(G)$ (some arbitrary point in the embedding) when the objective function estimator is defined to be $\mathrm{Estimator}(v) := \mathrm{d_e}(v, t_{\mathrm{e}})$ and the search radius function SearchRadius is as stipulated in the hypothesis of Lemma 4.

After that, we will argue that various types of linear combinations (in particular the one associated with Euclidean spaces, the mean) can be found by reducing the problem of computing various objective functions to the problem of computing the distance to some (unknown) point t_{e}. With that, it will be possible for us to conclude that the point $t_{\mathrm{G}} \in V$ that is closest to any *mean*, *median*, or *center* will *almost certainly* be found by the algorithm.

It is not immediately clear what relation t_{e} has to the vertex in the graph that actually minimizes the objective value on the graph. That issue is explored in [34], where it is discovered that the objective value of t_{G} is within a constant factor of optimal for medians and centers and is optimal for means.

The first part of the analysis is to show that the algorithm can find a target vertex given an estimator which estimates the distance between a given vertex and the target vertex.

Lemma 4 *Let t_e be some point in the embedding space, let t_G be the vertex in the graph whose corresponding point in the embedding is closest to t_e, and let $m := \mathrm{d}_G(v, t_G)$. There exists a function* $\mathrm{SearchRadius} = \mathcal{O}\Big(\sqrt{m \log^* m}\Big)$*, such that Algorithm 3 almost certainly gives the vertex t_G as its output when* $\mathrm{Estimator}(v) := \mathrm{d}_e(v, t_e)$.

Proof: The arguments given in the proofs of Lemmas 2 and 3 imply that the distribution of the per-unit-length distortion, that is $\frac{\mathrm{d_e}(a,b)}{\mathrm{d_G}(a,b)}$, can be pessimistically estimated with normal distribution.

In the case of overestimation, it was shown in the proof of Lemma 2 that the average per-edge distortion is a constant, that the per-edge variance is roughly $\mathcal{O}\big(\frac{1}{m}\big)$, and the distortion due to overestimation is independent at each edge of the shortest path. Each unit-length edge has an average measured-length of $\mu \leq c_0$ (recall that constant from the earlier proof) and the variance of that length is once again $\sigma^2 = \mathcal{O}\big(\frac{1}{m}\big)$.

For distortion due to underestimation at each edge, it was shown in the proof of Lemma 3 that one can overestimate that quantity by saying that independent probability of each vertex in the random graph being isolated, corresponding to an edge in the shortest path not being in the same class $\theta(e)$ as any other, is $\Theta\Big(\big(1 - \frac{1}{m}\big)^{m-1}\Big)$. That implies that when only considering underestimation, the average embedded length of each edge on the shortest path $\mu > \frac{1}{\exp(1)}$, and the variance is $\sigma^2 < \frac{1}{\exp(1)} - \frac{1}{\exp(2)}$.

Using the facts in the two preceding paragraphs, we observe that the distortion at every edge of a shortest path is independent, μ is a constant, and the variance is $\sigma^2 = \mathcal{O}(1)$.

Since the analysis in this proof is asymptotic, at this point we will make the simplifying assumptions that $\mu = 1$ and $\sigma^2 = 1$.

Using that setting, the Central Limit Theorem implies that the average distortion of a shortest path of length m is a normally-distributed random variable with $u = 1$ and $\sigma^2 = \frac{1}{m}$.

The cumulative distribution function of a normally-distributed random variable is

$$F(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right)\right]$$

where

$$\text{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\text{d}t$$

is the error function.

In this discussion, we will find it more convenient to work with absolute distortion, $\text{d}_\text{e}(a,b) - \text{d}_\text{G}(a,b)$, rather than relative distortion, $\frac{\text{d}_\text{e}(a,b)}{\text{d}_\text{G}(a,b)}$. If the relative distortion of a path of length m is x, then the absolute distortion is $y := m(x-1)$. For example, if a path has true length 1459 but is thought to be of length 1461 then it has relative distortion of 1.0013 and its absolute distortion is 2.

We now have

$$F(y) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{y/m}{\sqrt{2\sigma^2}}\right)\right]$$

for a path of length m.

For shorthand, we will let $r := \text{SearchRadius}(m)$ at any particular iteration of the algorithm. If v_current is the vertex on which the algorithm is currently sitting, then we will define v_best be the vertex in the search radius around v_current that is closest to the target t_e. The search radius will either contain t_G so that $v_\text{best} = t_\text{G}$, or else the two vertices will be distinct and $\text{d}_\text{G}(v_\text{best}, t_\text{G}) > m - r$. If the search radius contains t_G then the algorithm will find it and success is assured. Therefore, the focus of the discussion will be to see what happens when the search area does not contain it.

We must reference Figure 9 to start the discussion. The blue shaded area represents the portion of the graph that is within $m - r\frac{2}{3}$ units of t_G. The region with black north-west to south-east stripes represents the area of the graph that is within $m - r\frac{1}{3}$ units of t_G, but not within $m - r\frac{2}{3}$ units. The area with black south-west to north-east stripes represents the part of the graph that is within r units of v_current.

We will define the set Good to be the collection of vertices that are within $m - r\frac{2}{3}$ units of the t_G and are also in the search radius. We will define the set Fair to be those vertices that are within $m - r\frac{1}{3}$ of t_G, also in the search radius, but not in Good. Finally, we will say that Bad is the group of vertices that are in the search radius, but not in either of the two previously-mentioned groups.

In Figure 9, the blue vertices are in Good, the cyan vertices are in Fair, and the green vertices are in Bad. The important vertices v_current, v_best, and t_G are labeled.

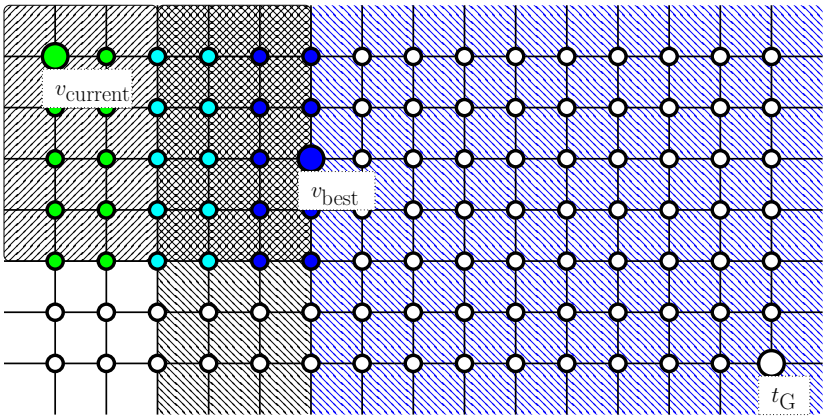


Figure 9: Labeled Steepest-descent example.

Plugging in σ^2 gives

$$F_{\text{Best}}(y) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{y}{\sqrt{2(m-r)}}\right)\right]$$
$$F_{\text{Bad}}(y) \leq \frac{1}{2}\left[1 + \text{erf}\left(\frac{y}{\sqrt{2m}}\right)\right]$$

where the first is the CDF for the absolute distortion of the length of a shortest-path from v_{best} going to the target, and the second CDF a bound on the same for some $v \in$ Bad. In the former case, what we had been calling m is at equal $m - r$ because v_{best} is no further than $m - r$ from t_{G} (because there exists a path from v_{current} to t_{G}).

The rest of the argument is composed of two pieces. The first part is to measure the probability of a vertex $v \in$ Bad having embedding distance $\text{d}_{\text{e}}(v, t_{\text{e}}) < m - r\frac{2}{3}$. The second part is to measure the probability of all vertices $v \in$ Good having embedding distance $\text{d}_{\text{e}}(v, t_{\text{e}}) > m - r\frac{1}{3}$. If both of these probabilities are small, then v_{next} must be in either Good or Fair because it is chosen according to shortest distance to the target in the embedding.

i The CDF of the greatest downward distortion exhibited by any point in the Bad group is

$$F_{\text{WorstBad}}(y) \leq 1 - (1 - F_{\text{Bad}}(y))^n$$

which is a very loose upper bound because the number of vertices in Bad is surely less than n.

We are interested in finding an r such that a negative deviation of magnitude $r\frac{1}{3}$ is sufficiently rare, that is $F_{\text{WorstBad}}(-r\frac{1}{3})$ is sufficiently small. An r such that

$$\frac{-r\frac{1}{3}}{\sqrt{2m}} \leq -|f(n)|$$

for $f(n) = \Omega\left(\sqrt{\log^* m}\right)$ would be sufficient to ensure that $F_{\text{WorstBad}}(-r\frac{1}{3}) \leq \varepsilon$ as $n, m \to \infty$. We can have that by letting $r = \sqrt{m \log^* m}$.

ii Using the same r just computed ensures that $F_{\text{Best}}(r\frac{1}{3}) \geq (1 - \varepsilon)$. Here we simply compute the probability of v_{best} suffering distortion such that its distance from the target appears to be greater than or equal to $m - r\frac{2}{3}$ (which certainly underestimates the probability of all members of Good appearing to be $\geq m - r\frac{2}{3}$ from the target).

Based on point i and ii, we can see that the odds of a Bad vertex appearing to be within $m - r\frac{1}{3}$ units of the target and/or all Good vertices appearing be at distance $m - r\frac{2}{3}$ or more is bounded above by $\varepsilon + \varepsilon = 2\varepsilon$. With the chance of that eventuality vanishingly small, we can conclude that $v_{\text{next}} \in (\text{Good} \cup \text{Fair})$, *in probability*, so that $\mathrm{d_G}(v_{\text{next}}, t_{\text{G}}) \leq m - r\frac{1}{3}$. □

This lemma says that the steepest-descent algorithm can be used to find the closest "real" vertex in the graph to some given vertex in the embedding space. During every iteration of the algorithm, there is a vertex v_{current} at which the algorithm is presently situated. The function SearchRadius controls the radius of the search around v_{current} in which the algorithm will search for the next v, v_{next}.

For purposes of execution speed, one would want the search radius to be as small as possible. However, the argument given in the proof shows that there must be some floor on the size of the search radius in order for the algorithm to have a reasonable chance of success. The above proof shows that roughly $\sqrt{m \log^* m}$, or any function that gives positive values and grows faster than $\sqrt{m}$, is long enough. This argument is quite loose though, so there is quite a bit of room for improvement of this bound.

Now that the capability to project points out of the embedding space back into the graph has been demonstrated, the next task is to relate this capability back to the problem of minimizing objective functions (which in turn will later be related to the problem of computing various types of linear combinations). We are particularly interested in *means*, *centers*, and *medians*. The following three definitions are needed:

$$\begin{aligned}
\text{OBJ}_2(v) &:= \sum_{i=1}^{k} \mathrm{d}(v, x_i)^2 \\
\text{OBJ}_\infty(v) &:= \max_{i=1}^{k} \mathrm{d}(v, x_i) \\
\text{OBJ}_1(v) &:= \mathrm{d}(v, \tilde{x})
\end{aligned}$$

where the first is the objective function for the 1-mean problem, the second is for the 1-median problem, and the objective functions are being computed over the points x_i for $i = 1 \cdots k$. The third is not the standard objective function for the 1-median problem, but it is the distance from the point v to the median of the collection of points $\tilde{x}$, which can be directly-computed in ℓ_1 by finding the median of each dimension. All of these objective functions can be minimized using the steepest-descent algorithm, Algorithm 3, by recasting them in terms of the distance between two points.

In the case of the 1-median, it is clear that this can be done because it is already written in that form. For the 1-mean it can be noted that $\text{OBJ}_2(v) = k\mathrm{d}(v, \bar{x})^2 + \text{OBJ}_2(\bar{x})$,

where $\bar{x}$ is the point that minimizes $\text{OBJ}_2(v)$. The fixed $OBJ_2(\bar{x})$ term and the factor of k are irrelevant as far as the comparison on line 6 of Algorithm 3 is concerned, and all non-zero distances between vertices in the graph are at least one so the squaring of the distance does not matter. All of that means that finding the mean of a collection of points is equivalent to minimizing the distance from the present point to some point $\bar{x}$ in the embedding. It is important to emphasize the following: the fact that $\bar{x}$ may be unknown at the time that the algorithm is evaluated does not impact our understanding of the correctness of the algorithm vis-á-vis Lemma 4. What is crucial is that $\bar{x}$ exists, is well-defined, and minimizing the 1-mean objective function is equivalent to minimizing the distance to this point.

Similarly, minimizing $\text{OBJ}_\infty(v)$ is equivalent to minimizing the distance between v and some other vertex whose identity may change over the course of the algorithm's evaluation. The fact that the exact vertex that determines the objective value can be different for different $v \in V$ does not matter. The Algorithm 3 is memoryless, so it is unaware of and unconcerned by the fact that the exact vertex that it is moving toward might change during the course of execution. As long as it continues to get closer and closer to the target, it will eventually terminate and return the correct answer.

Based on Lemma 4 and the above discussion, we can now state the following corollary without further proof.

Corollary 4 *If the function* SearchRadius *is as in Lemma 4, then Algorithm 3 almost certainly returns the graph vertex closest to the embedding mean (respectively, the median or the center) of a collection of points if the function* Estimator *is set equal to the appropriate* OBJ_2 *(respectively,* OBJ_1 *or* OBJ_∞*).*

This gives tidy tool for finding linear combinations of points as if we had an ℓ_2-embedding of the road network, which is helpful because many existing algorithms that we might wish to run on top of our infrastructure rely on this. As mentioned before, it was proven in [34] that the graph vertex closest to the mean in the embedding is also the optimal mean in the graph. Medians and centers appear less frequently as subroutines, but they are of interest in their own rights, and it was shown in that just-mentioned work that Algorithm 3 returns vertices that are within a constant-factor of optimal in those use-cases.

5.3 Speed Analysis

It was shown above that the steepest-descent algorithm shown in Algorithm 3 is capable of returning the vertex in the graph closest to the point in the embedding that minimizes various objective functions. The subject of this subsection is an analysis of cost of doing these queries. The first step that we will take in doing this will be to explicate some additional information found in the proof of Lemma 4.

5.3.1 Iterations

Within the proof of that lemma, it is shown that during the course of the algorithm's execution, if the present point v is at distance m from the target vertex t_e, then the next v will be at least $\Theta\Big(\sqrt{m \log^* m}\Big)$ units closer to the target if SearchRadius $= \Theta\Big(\sqrt{m \log^* m}\Big)$.

That means that the main loop of the steepest-descent algorithm executes no more than $\mathcal{O}(\sqrt{m})$ times per query. Since $m = \mathcal{O}(\sqrt{n})$, we can say that the algorithm iterates no more than $\mathcal{O}\left(n^{\frac{1}{4}}\right)$ times. That observation is presented as the following corollary, which can be stated without further proof.

Corollary 5 *Algorithm 3 iterates no more than* $\mathcal{O}\left(n^{\frac{1}{4}}\right)$ *times for an* Estimator *and* SearchRadius *as provided for in the hypothesis of Lemma 4.*

5.3.2 Cost Per Iteration

The next task is to establish the cost of each iteration. It is proven in Lemma 1 that each edge in G is covered $\mathcal{O}(1)$ times on average by the embedding algorithm. That fact makes it possible to arrange the computation of many interesting objective functions (distances, means, medians, centers, et cetera) in such a way that if $(a, b) \in E$ is an edge and OBJ(a) is known, then OBJ(b) can be computed with only $\mathcal{O}(k)$ additional cost where k is the number of vertices needed to define the objective function.

Lemma 5 *If* OBJ$(v) := f(\mathrm{d}_e(v, x_1), \cdots, \mathrm{d}_e(v, x_k))$ *and* $(a, b) \in E$*, then* OBJ(b) *can be computed from* OBJ(a) *with* $\mathcal{O}(k)$ *operations (on average), plus the cost of evaluating* $f(\cdot)$*.*

Proof: Let us first consider the objective function which simply measures the distance in the embedding space between a point t and an input point v.

$$\begin{aligned}\text{OBJ}(v) &:= \mathrm{d_e}(v, t) \\ &= \|\text{EMB}(v) \texttt{ xor } \text{EMB}(t)\|_1\end{aligned}$$

where the output of EMB$(\cdot)$ is the vector in the embedding space corresponding to a vertex in the graph. The distance function $\mathrm{d_e}(\cdot, \cdot)$ is therefore the Hamming weight of the difference of the two vectors (and the difference is the exclusive-or of the two binary vectors). With that, the reverse triangle inequality can be used to say

$$|\text{OBJ}(b) - \text{OBJ}(a)| \leq \|\text{EMB}(b) \texttt{ xor } \text{EMB}(a)\|_1$$

where we take EMB(t) to be the zero of the coordinate system.[14]

Let us consider each individual bit of the various vectors to see how they influence the objective value OBJ(b). For convenience, we will give bit i of EMB(a) the name w_i, so that $w_i :=$ EMB$(a)_i$ for the rest of this proof. Similarly, let $x_i :=$ EMB$(b)_i$, let $y_i :=$ (EMB(a) `xor` EMB$(t))_i$, and let $z_i :=$ (EMB(b) `xor` EMB$(t))_i$.

If $w_i = x_i$ then $y_i = z_i$, so bit i does not contribute anything to OBJ(b). If $w_i = 0$, $x_i = 1$, and $y_i = 0$, then $z_i = 1$ so that bit i contributes a value of $+1$ to OBJ(b) relative to OBJ(a), and so on, and so forth. The complete truth table is given in Table 4. In the table, * entries indicate "don't-care" bits.

[14] The reverse triangle inequality says that $||x| - |y|| \leq |x - y|$. Here, OBJ(a) is the distance from a to the zero EMB(t), and similarly for OBJ(b). The right-hand side $\|\text{EMB}(b) \texttt{ xor } \text{EMB}(a)\|_1$ corresponds to $|x - y|$. Since the objective function is a norm, the reverse triangle inequality applies.

w_i	x_i	y_i	z_i	change
0	0	*	*	0
1	1	*	*	0
0	1	0	1	+1
0	1	1	0	-1
1	0	0	1	+1
1	0	1	0	-1

Table 4: Lemma 5 truth table.

That means that OBJ(b) can be found from OBJ(a) using no more than

$$\mathbb{E}\left[\|\text{EMB}(a) \texttt{ xor } \text{EMB}(b)\|_1\right] = \mathcal{O}(1)$$

additions or subtractions of 1 from OBJ(a). It is known from Lemma 1 that $w_i = x_i$ for (on average) all but a constant number of indices i, for two vertices a and b at either end of an edge $(a, b) \in E$. If each edge is marked with the indices i for which w_i and x_i differ at the two endpoints, then because y_i is known as the algorithm proceeds, z_i and the change in the objective value can both be calculated with only $\mathcal{O}(1)$ additional cost. When the objective value of each point in the search radius is computed on line 6, the total cost of executing that line is expected to be bounded above by a constant times the number of vertices in the radius.

The above argument implies that if OBJ$(v) := f(\mathrm{d_e}(v, x_1), \cdots, \mathrm{d_e}(v, x_k))$ and $(a, b) \in E$, then OBJ(b) can be computed with OBJ(a) with an average of $\mathcal{O}(k)$ operations to compute the parameters to $f(\cdot)$, plus whatever the cost is of evaluating $f(\cdot)$ itself. □

The road networks that we are concerned with in this article are bounded-degree planar graphs with an additional $\mathcal{O}(\sqrt{n})$ crossing edges. Because of that, the number of vertices that are within r units of a vertex v is bounded above by $\mathcal{O}\left(r^2\right)$. In the context of Algorithm 3, that implies that the number of vertices in the search radius during each iteration of the algorithm is of size $\mathcal{O}\left(\left(\sqrt{m \log^* m}\right)^2\right) = \mathcal{O}(m \log^* m)$.

As stated in Corollary 5, the algorithm iterates $\mathcal{O}(\sqrt{m})$ times; as stated in the previous paragraph, each iteration touches $\mathcal{O}(m \log^* m)$ vertices; as stated in Lemma 5 the incremental cost of computing the objective values of each vertex is $\mathcal{O}(k)$ where k is as defined in that lemma. All of the just-mentioned facts can be amalgamated into following theorem concerning the cost of computing linear combinations of points in the embedding, as well as other points which minimize other admissible objective functions.

Theorem 2 *For an estimator function* Estimator *as provided for in Corollary 4, a search radius function* SearchRadius *as provided for in the hypothesis of Lemma 4, and a graph where the number of vertices within r hops of a given vertex is $\mathcal{O}\left(r^2\right)$, Algorithm 3 requires $\mathcal{O}\left(kn^{\frac{3}{4}} \log^* n\right)$ operations.*

Proof: There are $\mathcal{O}(\sqrt{m})$ iterations, each iteration touches $\mathcal{O}(m \log^* m)$ vertices, and

each vertex costs k to touch. The total cost is $\mathcal{O}\Big(km^{\frac{3}{2}} \log^* m\Big)$, which is equivalent to that stated in the hypothesis. □

If we are just going to compute estimated distances to the target for each vertex in some radius around the current vertex, then it is natural to question whether the embedding-based distance estimation brings anything to the table that other distance labels/oracles do not.

We can answer that by comparing the time bound stated in Theorem 2 to the bounds that we would get if we used other schemes instead of the ℓ_1 distance of the embedding. The best distance oracle that we discussed is the planar separator work [3] on page 4. That is a distance oracle, so it does not enjoy the ability to quickly compute the estimated distance from b if the distance at a is known and (a, b) is an edge. This is a perfect estimator, but it needs $\mathcal{O}(\sqrt{n})$ per exact query, so if the starting vertex and the eventual ending vertex are $\Theta(\sqrt{n})$ units distant from one another, then computing a linear combination would take $\mathcal{O}(n)$ time.

The best distance label that we discussed is [17], which is talked about on page 6. That distance labeling scheme provides a 3-approximate distance and needs $\mathcal{O}(\log n)$ operations per query. If each step across one edge were guaranteed in expectation to get closer to the target by some fixed (fractional) number of units, then this would be faster than the approach discussed above. The problem is that the distances are only 3-approximate and are discontinuous. This is further compounded by the fact that the regions and subregions that this scheme divides the map into are not in general convex with respect to the distance metric, so it is theoretically incompatible with Algorithm 3 which assumes a fairly smooth gradient.

5.4 Faster Combinations

If one is willing surrender theoretical guarantees, then there are faster ways to compute linear combinations, for example a modified version of the A^* [23] algorithm.

Briefly recapitulating A^*: it attempts to find the shortest path between two points s and t by summing two values $g(v)$ and $h(v)$, the former is the length of the best known path from s to v and the latter is the estimated distance from v to t, associated v with the summed values in a priority queue, and choosing the best vertex at every iteration.

One can repurpose A^* for use in finding linear combinations by making the function $g(v) := 0$ and by making the function $h(v)$ equal to the objective function. Doing that will cause the algorithm to disregard the distance from the starting vertex to v (which is irrelevant) and attempt to minimize the distance to the vertex that minimizes the objective function. The advantage that that has over Algorithm 3 is that it empirically touches only a small constant times the number of vertices on the path between the starting vertex and the target vertex. By comparison, Algorithm 3 touches about $n^{\frac{1}{4}}$ vertices for each vertex on that path.

Because it may not be clear when the optimal vertex has been reached, a termination rule is needed for the heuristic. For that purpose, we put a limit of 3 on the number of consecutive vertices that can be drawn from the priority queue without finding one that reduces the estimated objective function. If that limit is reached, then the heuristic is terminated, preventing it from loitering near the target for an inordinate amount of time.

Algorithm 4 The Steepest-Descent Heuristic.

Require: $G = (V, E)$, s, Estimator

```
1:  N_v ← {s}
2:  while N_v ≠ ∅ and c ≤ 3 do
3:      v' ← v
4:      v ← u ∈ N_v with smallest Estimator(u)
5:      if Estimator(v) < Estimator(v') then
6:          c ← 0
7:      else
8:          c ← c + 1
9:      end if
10:     N_v ← N_v ∪ {All neighbors of v}
11: end while
    return v
```

The heuristic is shown in pseudo-code form in Algorithm 4 and an experimental validation of it appears in Section 6.2.

6 Experiments

In this section, we report the results of various experiments. Since we are interested in supporting vector-like operations on graphs, and because the main operations associated with vectors spaces are distance computations and linear combinations, we investigate those two operations. The first of those is covered in Section 6.1 and the second in Section 6.2.

The embedding-construction technique used for the experiments in this section is the heuristic shown in Algorithm 2. Seven road maps of varying sizes, all obtained from the OpenStreetMap project [36], were used for these experiments. Six of the maps represent cities: Los Angeles, California; Compton, California; Las Vegas, Nevada; Newark, New Jersey; and Tallahassee, Florida; and the seventh is the road network map of Rhode Island. The smallest of these maps has around ten thousands vertices, and the largest has over one million. The number of vertices in each road network graph is given in Table 5.

The creation of the embedding for the smallest road network took less than seventeen seconds, and no map took more than two hours, on our experimental setup. All of the experiments were performed on a virtual machine with 16 cores (Xeon E5-2670 CPUs clocked at 2.6 gigahertz), 16 gigabytes of RAM, running the SmartOS variant of Solaris. The code was written in version 1.6 of Clojure, the JVM-hosted Lisp. Table 5 has the respective lengths of time (in seconds) required to construct each embedding.

The numbers of dimensions of the embedding spaces range from about 1,300 on the low end up to nearly 40,000. As hinted by the analysis given in Section 4.1, the representation of a typical vertex consists of mostly zero coordinates. That is important because the number of non-zero coordinates bears directly on the storage cost of the embedded

vertices, as well as the computational efficiency of using them. The average number of non-zero coordinates per vertex is shown in Table 5. The smallest of these averages is around 335 and the largest is less than 4,000, all modest multiples of $\sqrt{n}$.

	LA	CPT	Vegas	Denver	Tall'ee	Newark	RI
Nodes	299704	10737	123986	470946	223733	756855	1072435
Build Time	3400.02	16.36	992.61	3873.31	732.33	4060.25	6636.85
$\times 10^5 \div n\sqrt{n}$	2.07	1.47	2.27	1.20	0.69	0.62	0.60
Dimensions	17567	1289	14531	20640	14137	21637	38435
$\div\sqrt{n}$	32.08	12.44	41.28	30.08	29.89	24.87	37.11
Non-zeros	3397.45	334.77	1010.43	2907.55	900.95	3347.58	1902.11
$\div\sqrt{n}$	6.21	3.23	2.87	4.24	1.90	3.85	1.84
$\frac{\sigma}{\mu}$	0.044	0.062	0.058	0.0590	0.063	0.0547	0.041

Table 5: Numbers of nodes, construction times, embedding space dimensions, average numbers of non-zero coordinates, and embedding precisions for the various road maps.

6.1 Distance Measurements

For each road network map, ten thousand random pairs of vertices were chosen, and the distortion $\frac{\mathrm{d_e}(a,b)}{\mathrm{d_G}(a,b)}$ was computed for each pair. The average distortion, μ, and the standard deviation of the same, σ, were computed for each embedding. The results are shown in the last row of Table 5. The quantities $\frac{\sigma}{\mu}$ seen there are the "precisions" of the embeddings (or more formally, the precisions [8] of the distortions of the respective embeddings under repeated measurement).

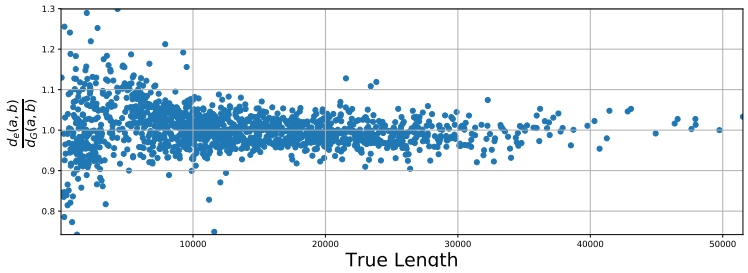


Figure 10: $\frac{\mathrm{d_e}(a,b)}{\mathrm{d_G}(a,b)}$ as a function of the true path length (in meters) on the Tallahassee data set. The embedding has been scaled so that $\mu = 1$. This figure represents 1,459 measurements.

As predicted by Theorem 1, the road maps with the fewest vertices generally yielded the least precise embeddings, and the embeddings for the largest maps were shown to be the most precise. The least precision was found on Tallahassee, which had a $\frac{\sigma}{\mu}$ value of 0.063, meaning that some seventy percent of distances in the embedding space are within seven percent of the true distance when the embedding is scaled so that $\mu = 1$. A scatter plot of the observed $\frac{\mathrm{d_e}(a,b)}{\mathrm{d_G}(a,b)}$ values given by the Tallahassee embedding can be found in

Figure 10. We measured a correlation[15] of -0.1133 between the true path length and the observed distortion.

6.2 Linear Combinations

The experiments reported in this subsection concern the execution cost and accuracy of the two point-projection techniques that have been presented in this article: the steepest-descent algorithm, Algorithm 3, and the steepest-descent heuristic, Algorithm 4. The purpose of both of these is to project an abstract point in the embedding space into the closest "real" vertex in the graph.

For instance, the projection of $\frac{\text{EMB}(s)+\text{EMB}(t)}{2}$ would be the midpoint of some shortest path between $s, t \in V$.

We plugged the following objective function estimator

$$\text{Estimator}(v) := \sum_{i}^{k} \text{d}_\text{e}(v, q_i)^2$$

into both the steepest-descent algorithm (Algorithm 3) and the steepest-descent heuristic (Algorithm 4)for the experiments in this subsection. This function estimates the objective function at any vertex v by making use of the distance metric provided by the embedding.

The means of groups of vertices ranging in size from two to thirty-three were taken on the Tallahassee map, the map with the least precise embedding.

The approximation ratios (the ratios of the objective values of the respective returned-vertices to that of the respective optimal vertices) and the numbers of milliseconds required to complete the computation were captured for the steepest-descent algorithm and the steepest-descent heuristic during 1,459 queries distributed evenly across the 32 values of k.

The observed approximation ratios of those experiments are shown in Figure 11, where the x-axes represent the optimal objective ratios divided by respective numbers of vertices over-which each mean was computed, $\frac{\text{OPT}}{k}$, while the y-axes represent the approximation ratios.

The steepest-descent algorithm had an average approximation ratio of 1.039 with a standard deviation of 0.1098, while the heuristic had an average approximation ratio of 1.290 with a standard deviation of 0.3235.

We measured almost no correlation between k and the approximation ratio for either the algorithm or the heuristic. The observed correlation was found to be -0.0660 in the former case and -0.0204 in the latter. There was even less correlation between $\frac{\text{OPT}}{k}$ and the approximation ratio: -0.0200 for the algorithm and -0.0164 for the heuristic. The small negative correlations show a slight tendency toward greater accuracy as k and $\frac{\text{OPT}}{k}$ respectively increase.

Our implementations of Algorithm 3 (the steepest-descent algorithm) and Algorithm 4 (the steepest-descent heuristic) are both single-threaded – that is they make use of only one processor core at a time – and are implemented in the Clojure language. Because the implementation language is one whose main strength is not single-threaded performance,

[15] Here we are using the standard definition, $\text{Corr}(\text{X}, \text{Y}) := \frac{\text{Cov}(\text{X},\text{Y})}{\sigma_\text{X}\sigma_\text{Y}}$.

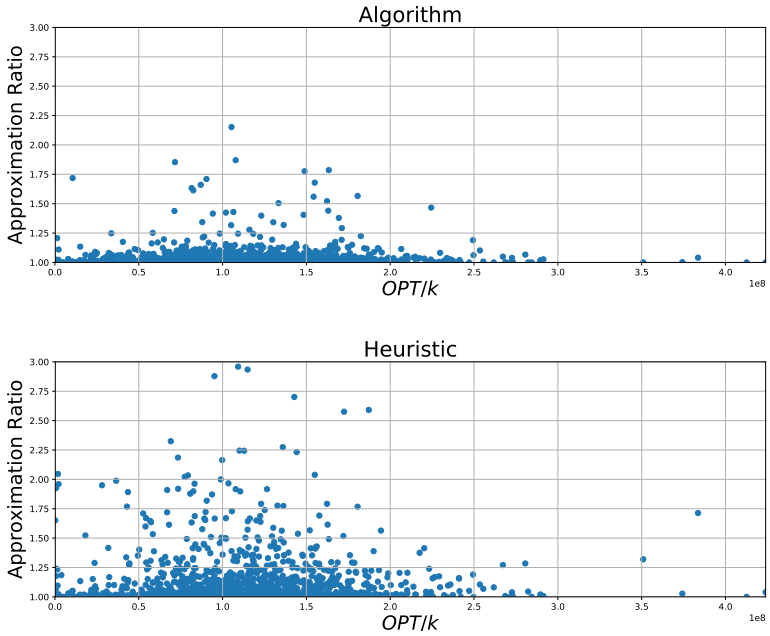


Figure 11: Approximation ratios for mean computations on the Tallahassee graph. One outlier with a ratio of 3.56 is not shown in the upper plot. Six outliers with ratios between 3.33 and 6.48 are not shown in the lower plot. Each figure represents 1,459 measurements.

the timing results given in this subsection should be regarded as lower-bounds on performance rather than indicators of how a production-ready implementation would behave. With those caveats in place, we present timing results in Figure 12: that figure shows the numbers of milliseconds required by the algorithm and the heuristic during the computation of 1,459 approximate means.

Each point in the scatter plot represents a mean computation over the same input, where the x-axis gives the time needed by the algorithm and the y-axis gives the same number for the heuristic.

We found very little correlation between k and the number of nodes touched[16]. The correlations found were 0.05940 and 0.06471 for the algorithm and the heuristic, respectively. Surprisingly, we also found little correlation between the number of milliseconds required and the number of nodes touched, with those correlations 0.05462 for the algorithm and 0.04458 the heuristic. However, we found strong correlations between k and the time requirements: a correlation of 0.6760 for the algorithm and 0.5798 for the heuristic. The relationship between computation time and k is illustrated in Figure 13.

In that figure the bold lines represent average milliseconds required per invocation at each value of k, and the shaded areas are one standard deviation above and below the average. For $k = 2$, the average number of milliseconds per invocation of the steepest-

[16]A node is considered to have been touched if it was enqueued in the priority queue during the course of the computation.

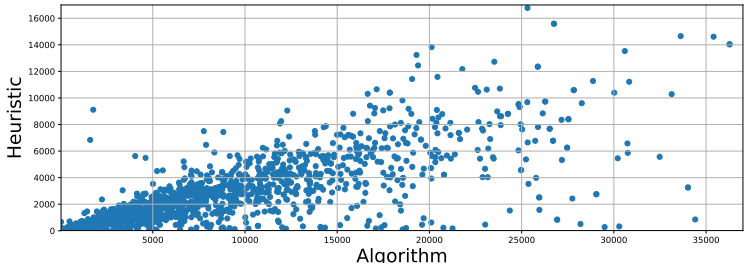


Figure 12: The milliseconds required to complete 1,459 mean computations. One outlier for which the algorithm required 54 seconds and the heuristic 12 seconds is not shown.

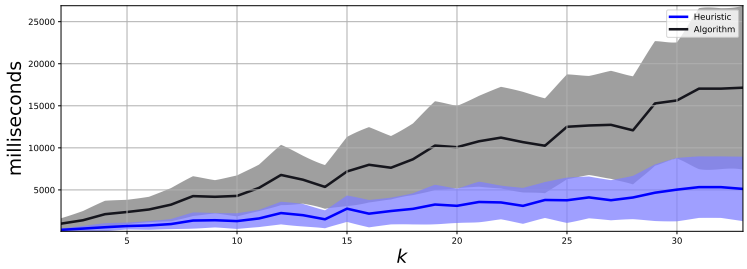


Figure 13: The milliseconds required by the algorithm and the heuristic as a function of k.

descent algorithm was found to be 980.0 milliseconds, while the average for the heuristic was 252.8 milliseconds. For $k = 33$, the averages were 17150.7 and 5119.5 milliseconds, respectively.

7 Conclusions

In this article we presented algorithms for embedding road network graphs into ℓ_1 with low distortion and for bringing computations done in the embedding space back into the graph.

Experimentally, we found that the method works very well. However, one area were improvement could be made is to the speed of the implementations of Algorithm 3 and Algorithm 4 (the linear combination algorithms). The timing measurements reported in this article were done on implementations whose primary design goal was not single-threaded speed. We believe that a more performance-centric implementation would probably improve single-threaded performance by a factor of ten, so producing such an implementation is an area of possible future work.

The speed of the embedding-construction algorithm is another area for possible improvement. Our implementation of Algorithm 2 does run fairly efficiently on the 16-core computer on which it was tested, but the algorithm is very parallel, with little need for

inter-thread communication, so a GPU-based implementation might be interesting.

Another avenue for future work is to consider dynamic road networks where the costs of traversing edges is allowed to change. The static networks that we consider in this work do not take into account the effects of predictable changes to traversal costs, such as traffic, and unpredictable changes, such as need for maintenance. This is a challenging proposition, but one possible method for efficiently dealing with predictable changes is to introduce a time parameter that interpolates between embeddings.

Acknowledgements

I would like to thank the anonymous reviewers for giving many helpful suggestions and comments. I would also like to thank Mark Pugner for many interesting conversations.

References

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In P. M. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer Berlin Heidelberg, 2011. URL: http://dx.doi.org/10.1007/978-3-642-20662-7_20, doi:10.1007/978-3-642-20662-7_20.

[2] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. *Highway Dimension, Shortest Paths, and Provably Efficient Algorithms*, chapter 64, pages 782–793. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611973075.64, doi:10.1137/1.9781611973075.64.

[3] S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In J. Diaz and M. Serna, editors, *Algorithms — ESA '96*, volume 1136 of *Lecture Notes in Computer Science*, pages 514–528. Springer Berlin Heidelberg, 1996. URL: http://dx.doi.org/10.1007/3-540-61680-2_79, doi:10.1007/3-540-61680-2_79.

[4] F. Aurenhammer and J. Hagauer. Computing equivalence classes among the edges of a graph with applications. *Discrete Mathematics*, 109(1–3):3 – 12, 1992. URL: http://www.sciencedirect.com/science/article/pii/0012365X92902740, doi:http://dx.doi.org/10.1016/0012-365X(92)90274-J.

[5] F. Aurenhammer, J. Hagauer, and W. Imrich. Cartesian graph factorization at logarithmic cost per edge. *computational complexity*, 2(4):331–349, 1992. URL: http://dx.doi.org/10.1007/BF01200428, doi:10.1007/BF01200428.

[6] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 184–193, Oct 1996. doi:10.1109/SFCS.1996.548477.

[7] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. *In Transit to Constant Time Shortest-Path Queries in Road Networks*, chapter 4, pages 46–59. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611972870.5, doi:10.1137/1.9781611972870.5.

[8] I. BIPM, I. IFCC, I. ISO, and O. IUPAP. The international vocabulary of metrology—basic and general concepts and associated terms (vim), jcgm 200: 2008, 2008.

[9] B. Bollobás. *Random graphs*, volume 73. Cambridge university press, 2001.

[10] B. Bollobas and P. Erdös. Cliques in random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 80:419–427, 11 1976. URL: http://journals.cambridge.org/article_S0305004100053056, doi:10.1017/S0305004100053056.

[11] J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985. URL: http://dx.doi.org/10.1007/BF02776078, doi:10.1007/BF02776078.

[12] S. Chatterjee, B. Neff, and P. Kumar. Instant approximate 1-center on road networks via embeddings. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 369–372, New York, NY, USA, 2011. ACM. URL: http://doi.acm.org/10.1145/2093973.2094025, doi:10.1145/2093973.2094025.

[13] S. Dasgupta and A. Gupta. An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report*, pages 99–006, 1999.

[14] M. M. Deza and M. Laurent. *Geometry of cuts and metrics*, volume 15. Springer, 1997.

[15] D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '08, pages 16:1–16:10, New York, NY, USA, 2008. ACM. URL: http://doi.acm.org/10.1145/1463434.1463455, doi:10.1145/1463434.1463455.

[16] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 448–455, New York, NY, USA, 2003. ACM. URL: http://doi.acm.org/10.1145/780542.780608, doi:10.1145/780542.780608.

[17] C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In F. M. auf der Heide, editor, *Algorithms — ESA 2001*, volume 2161 of *Lecture Notes in Computer Science*, pages 476–487. Springer Berlin Heidelberg, 2001. URL: http://dx.doi.org/10.1007/3-540-44676-1_40, doi:10.1007/3-540-44676-1_40.

[18] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 210–219, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=365411.365447.

[19] A. V. Goldberg, H. Kaplan, and R. F. Werneck. *Reach for A∗: Efficient Point-to-Point Shortest Path Algorithms*, chapter 12, pages 129–143. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611972863.13, doi:10.1137/1.9781611972863.13.

[20] R. Graham and H. Pollak. On embedding graphs in squashed cubes. In Y. Alavi, D. Lick, and A. White, editors, *Graph Theory and Applications*, volume 303 of *Lecture Notes in Mathematics*, pages 99–110. Springer Berlin Heidelberg, 1972. URL: http://dx.doi.org/10.1007/BFb0067362, doi:10.1007/BFb0067362.

[21] R. Graham and P. Winkler. On isometric embeddings of graphs. *Transactions of the American mathematical Society*, 288(2):527–536, 1985. URL: http://dx.doi.org/10.1090/S0002-9947-1985-0776391-5, doi:10.1090/S0002-9947-1985-0776391-5.

[22] A. Gupta, I. Newman, Y. Rabinovich, and A. Sinclair. Cuts, trees and ℓ_1-embeddings of graphs. *Combinatorica*, 24(2):233–269, 2004. URL: http://dx.doi.org/10.1007/s00493-004-0015-x, doi:10.1007/s00493-004-0015-x.

[23] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968. doi:10.1109/TSSC.1968.300136.

[24] B. Hochstrasser. A note on winkler's algorithm for factoring a connected graph. *Discrete Mathematics*, 109(1–3):127 – 132, 1992. URL: http://www.sciencedirect.com/science/article/pii/0012365X9290283L, doi:http://dx.doi.org/10.1016/0012-365X(92)90283-L.

[25] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 10–33, Oct 2001. doi:10.1109/SFCS.2001.959878.

[26] P. Indyk and J. Matoušek. Handbook of discrete and computational geometry. chapter 8, pages 177–196. CRC Press, 2nd edition, 2010.

[27] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. conference in modern analysis and probability (New Haven, Conn., 1982), 189–206. *Contemp. Math*, 26, 1984.

[28] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39 – 45, 1999. URL: http://www.sciencedirect.com/science/article/pii/S0020019099000319, doi:http://dx.doi.org/10.1016/S0020-0190(99)00031-9.

[29] G. Konjevod, R. Ravi, and F. Salman. On approximating planar metrics by tree metrics. *Information Processing Letters*, 80(4):213 – 219, 2001. URL: http://www.sciencedirect.com/science/article/pii/S0020019001001612, doi:http://dx.doi.org/10.1016/S0020-0190(01)00161-2.

[30] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. URL: http://dx.doi.org/10.1007/BF01200757, doi:10.1007/BF01200757.

[31] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, Mar 1982. doi:10.1109/TIT.1982.1056489.

[32] J. MacQueen. Some methods for classification and analysis of multivariate observations. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66, 1, 281-297 (1967)., 1967.

[33] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442(0):13 – 21, 2012. Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009). URL: http://www.sciencedirect.com/science/article/pii/S0304397510003269, doi:http://dx.doi.org/10.1016/j.tcs.2010.05.034.

[34] J. McClain and P. Kumar. Fast k-clustering queries on embeddings of road networks. In *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications*, COM.Geo '12, pages 11:1–11:9, New York, NY, USA, 2012. ACM. URL: http://doi.acm.org/10.1145/2345316.2345331, doi:10.1145/2345316.2345331.

[35] I. Newman and Y. Rabinovich. A lower bound on the distortion of embedding planar metrics into euclidean space. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, SCG '02, pages 94–96, New York, NY, USA, 2002. ACM. URL: http://doi.acm.org/10.1145/513400.513412, doi:10.1145/513400.513412.

[36] OpenStreetMap Wiki. Using OpenStreetMap — OpenStreetMap Wiki, 2013. [Online; accessed 22-January-2014]. URL: http://wiki.openstreetmap.org/w/index.php?title=Using_OpenStreetMap&oldid=959769.

[37] S. Rao. Small distortion and volume preserving embeddings for planar and euclidean metrics. In *Proceedings of the Fifteenth Annual Symposium on Computational Geometry*, SCG '99, pages 300–306, New York, NY, USA, 1999. ACM. URL: http://doi.acm.org/10.1145/304893.304983, doi:10.1145/304893.304983.

[38] D. Ž. Djoković. Distance-preserving subgraphs of hypercubes. *Journal of Combinatorial Theory, Series B*, 14(3):263 – 267, 1973. URL: http://www.sciencedirect.com/science/article/pii/0095895673900105, doi:http://dx.doi.org/10.1016/0095-8956(73)90010-5.

[39] P. M. Winkler. Proof of the squashed cube conjecture. *Combinatorica*, 3(1):135–139, 1983. URL: http://dx.doi.org/10.1007/BF02579350, doi:10.1007/BF02579350.