

Instant Approximate 1-Center on Road Networks Via Embeddings*

Samidh Chatterjee

Bradley Neff

Piyush Kumar

Department of Computer Science
Florida State University
Tallahassee, FL 32306

{chatterj,neff,piyush}@cs.fsu.edu

ABSTRACT

We study the 1-center problem on road networks, an important problem in GIS. Using Euclidean embeddings, and reduction to fast nearest neighbor search, we devise an approximation algorithm for this problem. Our initial experiments on real world data sets indicate fast computation of constant factor approximate solutions for query sets much larger than previously computable using exact techniques.

Categories and Subject Descriptors: H.2.8 [Database Applications]: Spatial Data and GIS

General Terms: Algorithms, Theory

Keywords: Road Network, Shortest Path, Embedding, Nearest Neighbor, Minimum Enclosing Ball

1. Introduction

We study the 1-center problem on road networks: Compute a ball of minimum radius enclosing a given set of locations (latitude, longitude) on the map. The measurements on the network are done using the shortest path metric. The 1-center problem on road networks has many important applications including location-based services and facility location. Important questions like: where should we construct a fire-station/hospital to cater to a set of people? can be answered by solving the 1-center problem on road networks. Another motivation to study the problem is: Given locations

*This research was partially supported by National Science Foundation through CAREER Grant CCF-0643593, Florida State University Committee on Faculty Research Support (COFRS) Summer Award and the Air Force Young Investigator Program.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SIGSPATIAL GIS '11, November 1-4, 2011, Chicago, IL, USA
Copyright ©2011 ACM ISBN 978-1-4503-1031-4/11/11...\$10.00

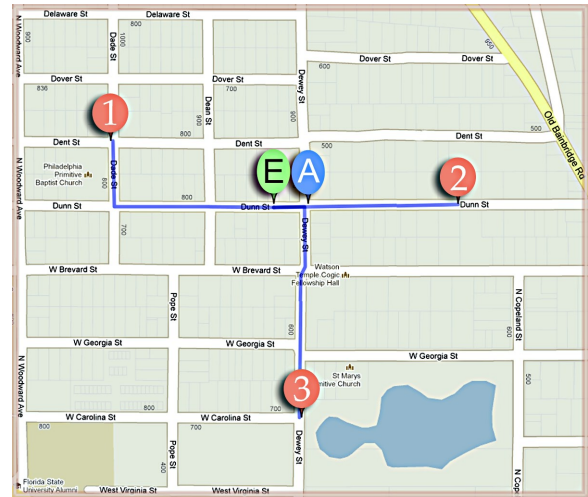


Figure 1. Example of a road network with exact (“E”) and approximate (“A”) 1-center computed. The shortest path from the locations to the exact center are also shown. The approximate answer is the one computed using the algorithm presented in this paper.

of people who want to meet, but want to keep the maximum distance traveled by anyone to a minimum. Figure 1 shows an exact solution to the 1-center problem.

We need to distinguish between problems in which the center can be located anywhere on the road network and problems in which centers can only be located on the nodes of the network. The former category of problems are known as *absolute center problems*, whereas the later are known as *vertex center problems*. In this paper, we implement an exact absolute 1-center as well as an exact vertex algorithm and compare each of them with an approximation algorithm that produces a vertex center as solution. Note that using Steiner points [6] on the edges, vertex centers can be used to approximate the absolute center solution easily.

The problem of 1-center computation has a long history and was solved exactly by Kariv and Hakimi [7] in 1979 using $O(n^3)$ time for a road network with n nodes (intersections). Henceforth, we will refer to this algorithm as the KH algorithm. One large motivation for trying to solve the

problem instantly (even though approximately), is that this could be put on the web for people to compute their results interactively.

An interesting generalization of the 1-center problem is the *aggregate nearest neighbor* (ANN) search problem [13]. Given a set P of interesting objects, a set Q of query points, and an aggregate function f (e.g., sum, max) an aggregate nearest neighbor (ANN) query retrieves the object p in P , such that $f\{d(p, q_i), \forall q_i \in Q\}$ is minimized. The 1-center problem is a special case of the ANN problem. In our setting, the set P is continuous (i.e. it is the whole road network instead of some interesting set of discrete points) and $f = \max$. Below we summarize the contributions of our paper:

- We propose a fast and approximate solution to the 1-center problem on the road network. To our knowledge, there does not exist any practical method for solving this problem instantly in the literature.
- We use embedding techniques to map the road network into low dimensional Euclidean space and empirically show that the Euclidean metric in the embedded space is a good approximation of the road network.
- Preliminary experiments show that embedding gives us fast and good approximate results.

The paper is organized as follows. In the remainder of this section we define our problem formally and introduce the notations we use. Section 2 gives an overview of our algorithm. Section 3 introduces our choice of optimization criterion for multidimensional scaling. Section 4 describes the tools we use in the query phase. In sections 5 and 6 we discuss our experimental setup and show our experimental results. Section 7 concludes the paper with future work.

Problem Definition and Notations : A network is an *undirected* graph $G = (V, E, w)$ where V is the set of vertices (or nodes) and E is the set of edges, and $w : E \rightarrow \mathbb{R}^+$ associates each edge to a positive real number. This number is the weight or cost of the edge. If $V = \{v_1, v_2, \dots, v_n\}$ and there is an edge $e \in E$ such that v_1 and v_2 are its two endpoints, then we represent e as (v_1, v_2) . Let Q be a set of query points $\{q_1, q_2, \dots, q_n\}$ (e.g., users) on the network. In this paper, using the longitude and latitude of each vertex, the Euclidean length of each edge is computed. We use this distance as the weight of each edge. The 1-center problem asks for a point p in the graph, such that the maximum distance of p to all points in Q is minimized, when traveling along the edges. The shortest path distance between two vertices v_i and v_j is denoted by d_{ij} , $\forall i, j = 1, 2, \dots, n$. The road network is assumed to be undirected, so that it can be embedded in a Euclidean space. We denote the Euclidean space by \mathbb{R}^d , where d is the dimension of the space. The Euclidean distance corresponding to d_{ij} is denoted by d'_{ij} .

2. Our Algorithm

We present a brief overview of our algorithm which is extremely simple and has two phases, namely the preprocessing phase and the query phase. The preprocessing phase, in turn is divided into two parts, the ‘‘Graph Generation Phase’’ and the ‘‘Embedding Phase’’.

Graph Generation Phase: We obtain our datasets from OpenStreetMap website(<http://cloudmade.com>). The next

task is to parse the map files to generate the road network graph, followed by calculation of the pairwise shortest path distances from each vertex. The distance matrix thus generated is fed into the embedding algorithm for mapping the points into an Euclidean space of a chosen dimension.

However, an actual query might come from anywhere on an edge of the graph. The particular point on the edge from where the query is coming from has no representation in the embedding. This is because that point is not a vertex in the graph and thus was not a member of the distance matrix. This problem can be fixed by using the following technique. Using the longitude and latitude of each vertex, its 2-dimensional coordinates are calculated. Each edge is divided into shorter, unit length edges using Steiner points [6]. We calculate the map coordinates (latitude, longitude) of a Steiner point by interpolation from the coordinates of original vertices, and also store the fraction (λ , say) that the Steiner point is along the edge. Finally, we store all vertices and Steiner points in a 2-d nearest neighbor (NN) [5] query structure. We denote this set of 2-d points, consisting of the vertices and the Steiner points, by V_2 . To handle a query, we use the NN structure to find the closest vertex or Steiner point, and then obtain the corresponding location in the embedded space. For a Steiner point, one can use λ to interpolate between the embedded positions of the endpoints of its edge.

In our actual implementation, we remove the nodes with degree 2 to compute the distance matrix for the embedding. Removing these nodes does not affect the shortest path calculations but reduces the data size. We keep the degree-2 nodes for constructing the 2-d NN structure. The number of these nodes on each edge are sufficient enough to act as our Steiner points.

Embedding Phase: In the embedding phase, we embed the graph in a low dimensional Euclidean space using Multidimensional Scaling (MDS) [4]. We denote the set of points in the embedded space by V_{d_e} , where d_e is the dimension of the embedded space. After the embedding is done, we compute another NN structure, this time using the points in V_{d_e} .

Query Phase: The input to the query phase are two NN structures (one from V_2 and the other from V_{d_e}), the query points and a set Q_{d_e} . Initially, $Q_{d_e} = \Phi$. The algorithm executes the following steps in this phase.

- Step 1 :** Find the NN of each query point (pairs of longitude, latitude) from V_2 using the 2-d NN data structure.
- Step 2 :** For each query point, if the NN in V_2 has a representation $q \in V_{d_e}$, add q to Q_{d_e} . If the NN is a Steiner point, interpolate its representation in \mathbb{R}^{d_e} and add the result to Q_{d_e} .
- Step 3 :** Compute the minimum enclosing hypersphere of the points in Q_{d_e} .
- Step 4 :** Find the nearest neighbor of the center of the minimum enclosing hypersphere from the points in V_{d_e} . Report the nearest neighbor’s corresponding vertex in the original graph as the answer.

In the next two sections we elaborate on the embedding phase and the tools used in the query phase.

3. Sammon Optimization Criterion for MDS

We used the “sammon projection” function [12] for the embedding purpose. The function is given by

$$S = \frac{1}{\sum_{i < j} d_{ij}} \sum_{i < j} \frac{(d_{ij} - d'_{ij})^2}{d_{ij}} \quad (1)$$

The Sammon cost function differs from the classical MDS in the sense that it puts more emphasis on retaining distances that were originally small. The minimization of the stress function can be performed using various methods, such as the eigendecomposition of a pairwise distance matrix, the conjugate gradient method, or a pseudo-Newton method [11]. We use the conjugate gradient method implemented in MATLAB. Our embedding dimension is fixed at 6.

The embedding obtained using Sammon function was compared with classical MDS, other optimization criteria available in MATLAB and the Place-Center algorithm which is a very recent MDS technique proposed by Agarwal et al. [2]. In all the cases, Sammon was the clear winner. Using semidefinite programming (SDP), we obtained optimal embeddings for road networks of small sizes (number of nodes ≤ 100). Although Sammon performed worse than SDP based embeddings, it was quite close to the optimal answer. We could not use SDP for large road networks because general purpose SDP solvers are both memory and CPU intensive.

4. Query Phase

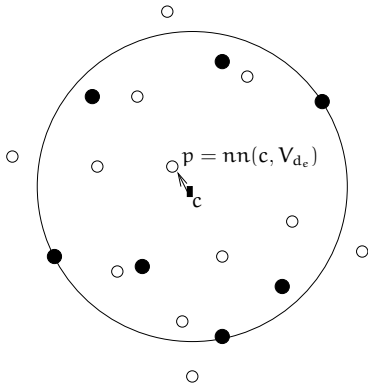


Figure 2. A GEQ example. The solid points are in Q_{d_e} and the hollow ones are in V_{d_e} . Center c of $\text{MEB}(c, Q_{d_e})$ is snapped to its nearest neighbor (denoted by $\text{nn}(c, V_{d_e})$) in V_{d_e} .

We explain the steps in the query phase in this section. Steps 1 and 4 involve NN computation. One of the most successful methods for Euclidean NN-search is to partition the entire point set using a tree type data structure. Queries are conducted by dropping a search ball on the tree and searching those nodes which intersect the ball. Tree based nearest neighbor searches have been found to work very well in practice in low dimensions; expected running time can be found to be $O(\log n)$ for $(1 + \epsilon)$ -approximate solutions [3]. We used a z-order based quadtree nearest neighbor search for implementing our algorithm [5].

In Step 3 we compute the minimum enclosing ball of the query points. We use $\text{MEB}(Q_{d_e})$ to denote the minimum enclosing ball for Q_{d_e} . Let $B(c, r)$ denote a ball centered at c with radius r . $\text{MEB}(Q_{d_e})$ is the smallest (measured by r) ball $B(c, r)$ such that $Q_{d_e} \subseteq B(c, r)$. In this context, the operator \subseteq refers to the geometric containment, i.e., every point in Q_{d_e} is fully contained by the ball defined by $B(c, r)$. If the center is fixed at p , $\text{MEB}(p, Q_{d_e})$ denotes the smallest ball centered at p that encloses all points in Q_{d_e} . Efficient algorithms for computing both exact and approximate MEBs are available in literature [8, 10].

The combination of Steps 3 and 4 gives an approximate solution to the *Group Enclosing Query* problem (GEQ), which is nothing but the ANN problem in the Euclidean setting [9]. See Figure 2. Li et al. [9] showed that if the exact center of the query points can be computed and the exact nearest neighbor of the center of the MEB is found, then the solution to the GEQ produced by the combination of Steps 3 and 4 is $\sqrt{2}$ -approximate. If one computes the $(1 + \epsilon)$ -approximate solutions to both the MEB and nearest neighbor, the approximation factor of the GEQ answer is $(1 + \epsilon)\sqrt{2}$. We compute the exact MEB and $(1 + \epsilon)$ -approximate NN in Steps 3 and 4 of the query phase. In Step 1, we compute the exact nearest neighbor.

5. Experimental Setup

Our algorithm was implemented in C++ and MATLAB. The implementation of the exact algorithm, the graph generation phase and the query phase part of our algorithm was written in C++ and compiled with g++ 4.1.2 with -O3 optimization. For constructing the graphs from the road networks, the BOOST (<http://www.boost.org>) library was used. The CGAL [1] and STANN [5] libraries were used for implementing the KH algorithm, computing the Minimum Enclosing Ball and the NN-search in the Euclidean space. The embedding phase was implemented using MATLAB. The machine used has 8 Quad-Core AMD Opteron(tm) Processor 8378 with hyperthreading enabled. Each core has a L1 cache size of 512 KB, L2 of 2MB and L3 of 6MB with 128 GB total memory. The operating system was CentOS 5.3. All data was generated and stored as 64 bit C++ doubles. Currently, we use 24 cores of this machine for preprocessing and one core for the queries. In the next section we present our results from the experiments on the road networks of Tallahassee (1597 nodes and 2425 edges) and Florida (1001982 nodes and 1327731 edges). The number of nodes and edges given are obtained after removal of degree 2 nodes.

6. Experimental Results

Once we embedded the shortest path metric into \mathbb{R}^6 , we compared the exact radius computed using KH algorithm and the radius computed by our algorithm. This is shown in Figure 3. There are two comparisons done in these graphs. One is with the “absolute answer” computed by the KH algorithm, shown by “Absolute Distortion” in the graph. The second is with the exact “vertex answer”, shown by “Vertex Distortion”. Let R_{ab} and R_v denote the true absolute and vertex radii and R_{approx} denote the approximate radius. Each radius is the shortest path distance between the 1-center and the furthest query point from it. Then

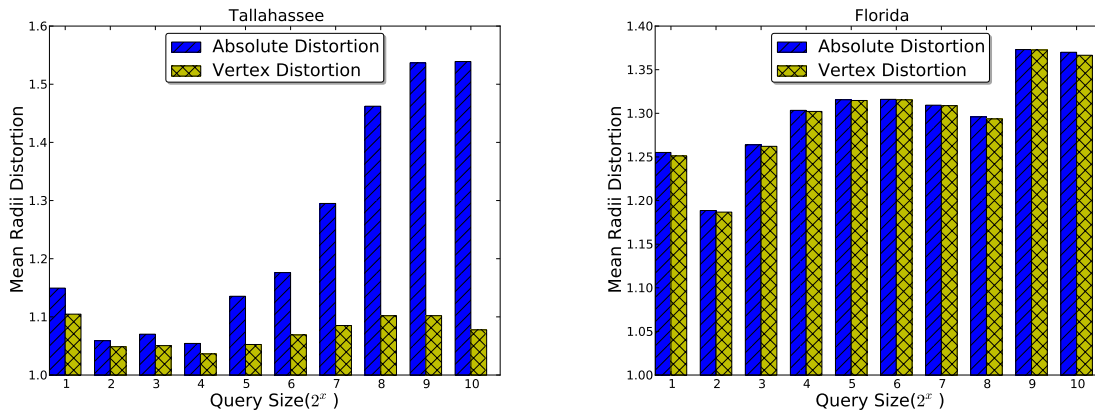


Figure 3. Error in approximation of the 1-center answer computed by our algorithm.

Table 1. Time chart in seconds for computing the 1-center for query sizes ranging from 2 to 2^{10}

		2	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Tallahassee	Vertex	0.002	0.003	0.005	0.01	0.021	0.04	0.08	0.162	0.336	0.710
	Absolute	0.006	0.013	0.031	0.053	0.132	0.306	0.809	2.052	4.874	11.238
	Approximate	10^{-7}	10^{-7}	10^{-6}	10^{-6}	10^{-6}	10^{-5}	10^{-5}	10^{-4}	1.3×10^{-3}	1.3×10^{-3}
Florida	Vertex	4.88	9.855	19.95	39.92	79.44	159.44	321.94	656.38	1309.9	2611.26
	Absolute	7.190	17.007	35.276	68.288	138.7	235.54	434.16	818.44	1467.98	2869.947
	Approximate	0.023	0.024	0.022	0.023	0.027	0.024	0.025	0.037	0.048	0.053

the figures in the y-axes of the graphs is the average from 50 runs of $R_{\text{approx}}/R_{\text{ab}}$ for computing the Absolute Distortion and $R_{\text{approx}}/R_{\text{v}}$ for computing the Vertex Distortion. We use the term “Mean Radii Distortion” for the measuring the error in approximation of our answer with the true radius. We see that we are more closer to the vertex answer than the absolute answer. This is because R_{approx} is greater than both R_{ab} and R_{v} , and $R_{\text{v}} \geq R_{\text{ab}}$. Hence $R_{\text{approx}}/R_{\text{v}} \leq R_{\text{approx}}/R_{\text{ab}}$. The increase in distortion for both the answers occur when the queries are fed from the points having large distortions when embedded. The timings for the exact and approximate algorithms are shown in Table 1. As stated before, a cubic time algorithm is no match to our algorithm.

7. Conclusion and Future work

This work opens up many new and interesting avenues of research. One immediate open problem is how to reduce the error in approximation of the 1-center answers for large data sets. Can we solve k-center/k-means problems for points in motion on road networks now? We already have done preliminary experiments with k-means on our embeddings and seen encouraging results. We are in the process of building a website (<http://maps.compgeom.com>) that will allow people to solve 1-center problem approximately but instantly. We hope that someday this method is used to save fuel and reduce pollution in the real world.

Acknowledgement:The authors would like to thank Tathagata Mukherjee for his valuable suggestions and the anonymous referees for their constructive criticisms.

References

- [1] CGAL, Computational Geometry Algorithms Library. <http://www.cgal.org>.

- [2] Arvind Agarwal, Jeff M. Phillips, and Suresh Venkatasubramanian. Universal multi-dimensional scaling. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '10*, 2010.
- [3] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45:891–923, November 1998.
- [4] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer-Verlag New York, 2005.
- [5] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):599–608, 2010. <http://compgeom.com/~stann>.
- [6] Mark de Berg, Otfried Cheong, Mark van Kreveld, and Mark Overmars. *Computational Geometry Algorithms and Applications*. Springer-Verlag, Berlin Heidelberg, 3 edition, 2008.
- [7] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):pp. 513–538, 1979.
- [8] P. Kumar, J. S. B. Mitchell, and A. Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *The ACM Journal of Experimental Algorithmics*, 8, 2003.
- [9] Feifei Li, Bin Yao, and Piyush Kumar. Group enclosing queries. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints), 2010.
- [10] Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31:114–127, January 1984.
- [11] L. J. P. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality Reduction: A Comparative Review. 2007.
- [12] Jarkko Venna and Samuel Kaski. Local multidimensional scaling. *Neural Netw.*, 19:889–899, July 2006.
- [13] Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Trans. on Knowl. and Data Eng.*, 17:820–833, June 2005.