CDO: Extremely High-Throughput Road Distance Computations on City Road Networks *

Shangfu Peng Hanan Samet

Center for Automation Research
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742 USA
{shangfu, hjs}@cs.umd.edu

ABSTRACT

Some analytic queries on road networks, usually concentrating in a local area spanning several cities, need a high-throughput solution such as performing millions of shortest distance computations per second. However, most existing solutions achieve less than $5,000$ shortest distance computations per second per machine even with multi-threads. We demonstrate a solution, termed *City Distance Oracles* (CDO), using our previously developed ϵ-distance oracle to achieve as many as 7 million shortest distance computations per second per commodity machine on a city road network, i.e., $10K \times 10K$ origin-distance (OD) matrix can be finished in 14 seconds.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*; H.2.4 [**Database Management**]: Systems—*Query processing*

Keywords

high-throughput, spatial analytic query, distance oracle, city road network

1. INTRODUCTION

Browsing of spatial data is becoming increasingly important [8, 13, 21, 22]. During the spatial analyst's exploration, some types of spatial queries, termed *spatial analytic queries*, can potentially involve thousands to millions of road network distance computations. Examples of such analyses include complex scenarios such as how to assign and deliver $10,000$ packages for UPS in a city, how much traffic congestion could be reduced if build a new bridge, where to locate the next supermarket among a number of potential locations taking into account a variety of factors like demography, distance to a warehouse, etc., identifying bottlenecks in a road network for evacuation planning, or distance join queries on road networks [27].

Figure 1: The work flow of demo CDO: First extract any city road network such as New York City from OpenStreetMap [2] and TAREEG [3]; Then precompute the ϵ-DO [24]; Finally load the results in memory and implement multi-thread version to process query workload.

Our focus here is on *throughput* which is how to compute a spatial analytic query as quickly as possible. Note that although decreasing the latency time for a single shortest distance query results in reducing the total response time for a spatial analytic query, it is far from enough since these latency methods don't take into account considerations such as multi-users, multi-threads, reused results, and query optimization [17]. Our recent work [18] discussed how to obtain high throughput performance using ϵ-distance oracle (ϵ-DO) [24, 25] in a distributed key-value store such as Apache Spark for spatial analysis on the continental road networks such as the entire USA. However, the reaction of a number of companies that make use of such queries was that typical queries are concentrated in a small local region rather than the whole continental region, termed the *spatial concentration* property. As an example of such a use-case is a delivery company that needs to plan the delivery route for each truck every day, where the route of each truck must be restricted into a local region, i.e., the region near to the package warehouse. In particular, each such warehouse handles $1,000$ to $10,000$ packages per day, and each truck can deliver a maximum of 150 packages per route. In order to efficiently assign the packages to trucks and plan routes, the delivery company computes a distance matrix that captures the distance between every pair of destination locations of the packages, This is a common spatial analytic query which makes between 1 million and 100 million distance computations. Here, the *spatial concentration* property means that in the general case, all destination locations of packages must be in proximity to the warehouse, say within 100KM.

Now we demonstrate an extremely efficient solution to solve spatial analytic queries where the spatial concentration property holds. In particular, we will show how one can compute large origin-distance (OD) matrix of size $10K \times 10K$ in a few seconds. The

*This work was supported in part by the AWS research grant and the NSF under Grants IIS-12-19023, IIS-13-20791, and IIP-1634753.

SIGSPATIAL'16 October 31 - November 03, 2016, Burlingame, CA, USA

DOI: http://dx.doi.org/10.1145/2996913.2996921.

main work flow is shown in Figure 1. We extend our prior work on ϵ-DO [24, 25], which precomputes and stores approximations of the shortest distances between all pairs of vertices in a road network. The resulting representation takes $O(\frac{n}{\epsilon^2})$ space, where n is the number of vertices in the road network and ϵ is an approximation error bound on the result. Our contributions in this demo are:

1. An efficient implementation of using ϵ-DO in memory instead of in a database [24, 26] with multi-threads and query optimization illustrated in [17]. As a result, we achieve 7 million distance computations per second on the Bay Area New road network in latitude/longitude region $[37.173, 38.019] \times [-122.678, -121.571]$ with $755K$ vertices.

2. The design of a new key representation of the ϵ-DO which enables doing a binary search to retrieve the road network distance on the ϵ-DO without requiring any special indices. It greatly speeds up the query time.

3. An application of CDO is illustrated, and an evaluation of time performance is provided for CDO, HLDB [5], and CH [14].

In addition, we set up the CDO demo for the Bay Area and New York City [1] and provide some use cases in our blog site [2].

2. RELATED WORK

The methods for computing shortest distances fall into two main categories: *latency* methods and *throughput* methods. However, most shortest distance methods do no consider the spatial concentration property of the spatial analytic queries.

Latency approaches are designed to answer a single or a small number of shortest path or network distance queries on road networks. The original road network or a processed representation of it is stored in memory and queries perform operations on this in-memory representation. The most common latency approach is Dijkstra's algorithm [12]. Other latency methods [4, 6, 7, 10, 11, 14, 15, 16, 19, 23, 31] are based on the observation that some vertices in a spatial network are more important than others in answering shortest path queries. These methods offer different trade-offs between pre-processing time, storage, and query time.

A characteristic of *throughput* methods, [5, 18, 24, 25, 28], is that the shortest paths and distances are precomputed so that the query process only requires a lookup as opposed to any real computation on the fly. These methods are good for obtaining a high throughput as multiple lookups can be batched to take place at the same time thereby increasing the number of queries that can be answered at the same time. The simplest way is precomputing all-pairs shortest distances, but the storage for a city road network with $500K$ vertices is more than $500K \times 500K \times 12bytes \approx 2800GB$, no to mention the continental road networks with more than $30M$ vertices. Thus, the throughput methods could be considered as some compression algorithms for all-pairs shortest distances. Even though there are some efficient compression representations, the representations for the continental road networks are still large so that they need to be stored on disk/database thereby affecting both latency and throughput. We found that storing memory is only possible for city road networks, which will decrease the latency time a lot for the throughput methods.

Wu et al. [30] evaluate several state-of-the-art methods (i.e., [7, 14, 24, 28]) for computing road network distance in the same environment. Even though they do not make the distinction between latency and throughput methods, and only compare all the methods from a latency perspective, there are some valuable lessons to be learned from this evaluation. First, they show that TNR [7] and CH [14] have fast preprocessing, low space overhead, support for real time queries, and the ability to easily handle continental road networks with tens of millions of vertices. They also point out that although ϵ-DO [24, 25] and PCPD [28] are better for answering queries, they are not practical for continental road networks because they they are too expensive to precompute. The demonstration of this paper shows that ϵ-DO is very usable on city road networks, which can ignore the precomputation cost.

[1] http://sametnginx.umiacs.umd.edu/
[2] http://roadsindb.com/

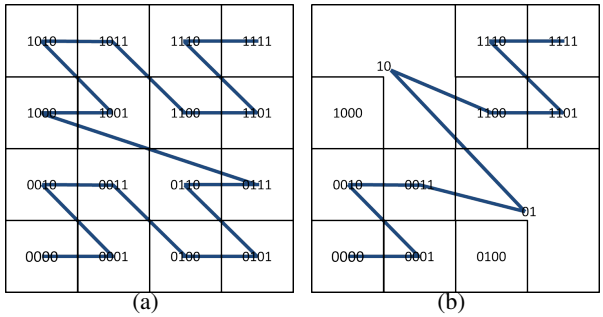

Figure 2: (a) Morton code and ordering in a 4×4 space. (b) Example to illustrate the key representation of distance oracle.

3. PRELIMINARIES

A road network G is modeled as a weighted directed graph denoted by $G(V, E, w, p)$, where V is a set of nodes or vertices, $n = |V|$, $E \subset V \times V$ is the set of edges, $m = |E|$, and w is a weight function that maps each edge $e \in E$ to a positive real number $w(e)$, e.g., distance or time. For each node v, $p(v)$ denotes the spatial position of v with respect to a spatial domain S, which is also referred to as an embedding space (e.g., a reference coordinate system in terms of latitude and longitude). We define the graph distance $d_G(u, v)$ to be the shortest path from u to v in the spatial graph, while $d_E(u, v)$ to be the Euclidean distance from u to v.

We use the Morton (Z) order space-filling curve [20] that provides a mapping, $\mathbb{Z}^2 \rightarrow \mathbb{Z}$, of a multidimensional object (e.g., a vertex or a quadtree block) in a 2-dimensional embedding space to a positive number. Given an object o, let $mc(o)$ be the mapping function that produces the Morton representation of o by interleaving the binary representations of its coordinate values.

Given a spatial domain S, the Morton order of blocks in S can be obtained by subdividing the space into $2^L \times 2^L$ equal sized blocks named *unit blocks*, where L is a positive integer named the maximal decomposition level. Each unit block i is referenced by a unique Morton code $mc(i)$. Figure 2(a) shows how a Morton order of quadtree blocks in a two dimensional space with $L = 2$. A spatial graph $G(V, E, w, p)$ on the domain S can also be divided into $2^L \times 2^L$ unit blocks. Given vertex v in the unit block i, v is assigned a Morton code $mc(v)$ as $mc(i)$,. All vertices located in the same blocks have the same Morton code. Besides the unit blocks, every larger block b has a unique Morton code, which is the longest common prefix of all unit blocks contained in b, e.g., the Morton code of the top left quadrant $(1000, 1001, 1010, 1011)$ is 10. In this demo, given blocks A and B, we define the relation $A \prec B$, if and only if block A is contained in block B, and thus $mc(B)$ is a prefix of $mc(A)$. Once the data are sorted using this ordering, the resulting blocks can be stored using any one-dimensional data structure such as, but not limited to, a B-tree.

Formally, ϵ-DO [24, 25] describes a well-separated pair decomposition (WSPD) [9] of a road network in order to produce well-separated pairs (e.g., (A, B) with particular network distance prop-

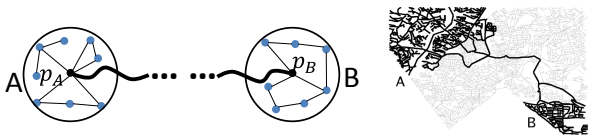

Figure 3: A well-separated pair example in form and in Silver Spring, MD, showing representative vertices p_a, p_b and radius, the representative shortest distance is $d_G(p_A, p_B)$.

erties). Two sets of vertices A and B are said to be well-separated as in Figure 3 if the minimum distance between any two vertices in A and B is at least $s \cdot r$, where $s > 0$ is a separation factor and r is the larger diameter of the two sets. The pair (A, B) is termed a well-separated pair (WSP), which satisfies the property that for any pair of vertices (s, t), $s \in A$ and $t \in B$, we can find the approximate distance $d_\epsilon(A, B)$, where $\epsilon = \frac{2}{s}$, providing an approximate network distance such that it satisfies the condition

$$(1 - \epsilon) \cdot d_\epsilon(A, B) \leq d_G(s, t) \leq (1 + \epsilon) \cdot d_\epsilon(A, B) \qquad (1)$$

4. METHOD

CDO is based on ϵ-DO on a city road network, e.g., the number of vertices n is less than one million. Note that on a commodity machine, immediately loading $O(n^2)$ distance results in memory is infeasible when n is larger than 50 thousand.

As a result, ϵ-DO generates $O(\frac{n}{\epsilon^2})$ well-separated pairs, denoted as $(A, B, d_\epsilon(A, B))$. Both A and B are a pair of PR quadtree blocks [20] at the same depth. In order to make a well-separated pair easy to embed in a database as a key-value pair, ϵ-DO uses the Morton (Z) order space-filling curve [20] to map a quadtree block in a 2-dimensional embedding space to a positive number. Thus, each well-separated pair $(A, B, d_\epsilon(A, B))$ is considered as a key-value pair $(mc(mc(A), mc(B)), d_\epsilon(A, B))$, where the value is the distance and the key is $mc(mc(A), mc(B))$.

4.1 Storing and Querying CDO

Here we illustrates how to obtain the network distance in CDO, given a source location $p_1 = (lat_1, lng_1)$ and a destination location $p_2 = (lat_2, lng_2)$. Once ϵ-DO has been computed, CDO loads all well-separated pairs, which the schema is (code, d), in memory as an array sorted by code, where code is a succinct representation of the well-separated pair and d is the approximate network distance. Although such a schema is similar to the one proposed in ϵ-DO [24], our method just uses the default integer comparator instead of redefining the string comparator operators (i.e., $<$ and $=$) while doing binary search. This is important because the default integer comparator saves much time in contrast to the redefined string comparator.

To illustrate our method for packing the code, we first start with a simpler two-dimensional example (i.e., Z_2) and we then describe how to encode a well-separated pair as a four-dimensional Morton block. Suppose that we have a number of various length Morton codes in two-dimensions, which means that the corresponding quadtree blocks are at different depths. The simpler problem we want to solve is that we are given a point p, and we need to efficiently find a unique quadtree block A containing p. Here we assume that the uniqueness property from the property of WSP [9] is also true in this simpler example. The uniqueness property here means that there is exactly one quadtree block containing p such as in Figure 2. This search problem is equivalent to finding the unique $mc(A)$ such that $p \prec A$.

Our approach is to make all the Morton codes have the same length by padding them with enough zeros, so that all Morton codes are always the same length, i.e., $2 \cdot L$ bits long in two-dimensions. For any Morton code $mc(A)$, padding with enough zeros is equivalent to choosing a unit-sized block that is a descendant of A in the quadtree that has the smallest Morton code. This needs to be done carefully as we illustrate with the following example. Suppose that our two-dimensional oracles has ten quadtree blocks as in Figure 2 whose Morton codes are 0000, 0001, 0010, 0011, 01, 10, 1100, 1101, 1110, and 1111. Only two Morton codes 01 and 10 are not 4 digits long. Thus, consider the quadtree blocks 01 and 10 in Figure 2, which we convert to 0100 and 1000 respectively by padding zeros to the right hand side. The codes of our oracle become: 0000, 0001, 0010, 0011, 0100, 1000, 1100, 1101, 1110, and 1111 in order. Given a query point $p = 0111$ that is contained by a unique quadtree block A. To find A, we need to find a quadtree block in the B-tree such that it is the largest value that is less than or equal to p, which in this case is 0100 (i.e., quadtree block 01, which is the correct answer).

Algorithm 1: GETDIST(lat_1, lng_1, lat_2, lng_2)

Data: A: Sorted array containing all well separated pairs
Result: $Result$: Network distances between (lat_1, lng_1) and (lat_2, lng_2)

```
code ← Z_4^0( Z_2((lat_1, lng_1)), Z_2((lat_2, lng_2)) );
left ← 0;
right ← A.size();
while left ≤ right do
    mid ← (left + right)/2;
    if A[mid].code ≤ code then
        left ← mid + 1;
    else
        right ← mid - 1;
return A[left − 1].d;
```

Now going back to CDO, we obtain a four dimensional Morton code by interleaving $mc(A)$ and $mc(B)$ two digits at a time. This packing is given by the function $Z_4(A, B)$. Next, we define function $Z_4^0(A, B)$ by padding $Z_4(A, B)$ with zeros to the right side. For example for the blocks in Figure 2, Z_4 and Z_4^0 should be

$$Z_4(01, 10) = 0110 \qquad Z_4^0(01, 10) = 01100000$$
$$Z_4(0000, 1111) = Z_4^0(0000, 1111) = 00110011$$

This packing Z_4^0 produces a Morton code of $4 \cdot L$ bits length. This forms the *code* attribute. At this point, given a source location p_1 and a destination location p_2, the approximate network distance query first calculates $key = Z_4^0(mc(p_1), mc(p_2))$ in $O(1)$ time using bitwise operations and then issues the following binary search function, Algorithm 1, to obtain the network distance.

The reason this scheme works is because of the uniqueness property of WSP. For any two points in the domain S, there is exactly one WSP containing them.

4.2 Multi-threads

As most memory resources in Algorithm 1 are only processed by read-only operations, parallel processing should increase the throughput a lot. Without loss of generality, we show how to obtain the network distances of a batch of source-target pairs with multi-threads in Algorithm 2.

5. DEMO SCENARIO

We extracted and prepared a CDO of the Bay Area road network with $781K$ vertices and one for the New York City road network

Algorithm 2: GETBATCHDIST(Q)

Data: Q: Query array where each element is a source-target pair; $nthread$: Number of threads
Result: $Result$: Network distances for Q

$len \leftarrow Q.size()/nthread$;
$ans \leftarrow$ initial a float array with $Q.size()$ elements;
for $i \leftarrow 0$ **to** $nthread$ **do**
    $thread[i] \leftarrow$ initial thread i, process $Q[j], j \in [i \cdot len, (i+1) \cdot len)$ by Algorithm 1 in thread i, and store distance results in corresponding $ans[j]$;
for $i \leftarrow 0$ **to** $nthread$ **do**
    $thread[i].join()$;
return ans;

with $407K$ vertices from OpenStreetMap [2]. The demo is set up at `http://sametnginx.umiacs.umd.edu/`.

We implemented three methods, our solution *CDO* where $\epsilon = 0.05$, HLDB [5], and CH [14], in C++, where all of them are processing *in memory* with multi-threads, and in the same environment, a Macbook Pro 15-inch, 2.8 GHz Quad-core Intel Core i7, 16 GB memory.

Figure 4 shows the time performance of the three methods on the Bay Area road network. CDO is the fastest one, which achieves 0.16 seconds with 8 threads for 1 million distance computations.

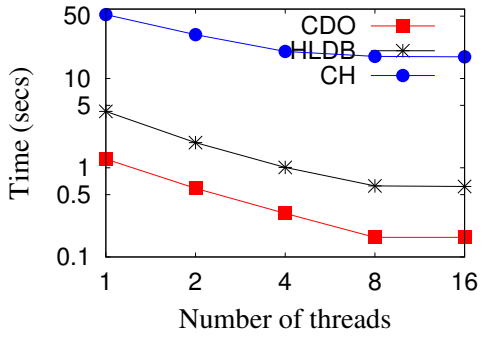

Figure 4: Time consumption for 1 million random source-target pairs on the Bay Area road network, varying with number of threads.

In addition, here we provide an application that can be efficiently solved by our demo. It is to measure the accessibility of jobs, i.e., how many job opportunities exist nearby each census block. We use the LEHD dataset [1] to obtain the job locations around the Bay Area. This workload shown in Figure 5 contains 120 million distance computations, where CDO only needs 18 seconds. Obviously, based on our solution, many analytic queries could be solved and visualized in a much quicker way, such as showing the influence of building a bridge from two arbitrary Bay Area locations. Future work involves investigating the use of distributed data structures in the application (e.g., [29]).

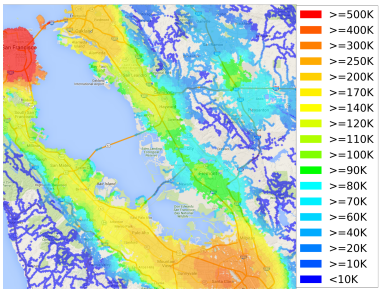

Figure 5: Nearby job opportunities (e.g., within 10 kms) for each census block in the Bay Area, requiring 120 million distance computations, where CDO finished it within 18 seconds.

6. REFERENCES

[1] LODES. `http://lehd.ces.census.gov/data/`.
[2] OpenStreetMap. `http://www.openstreetmap.org/`.
[3] TAREEG. `http://tareeg.org/`.
[4] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*, pages 230–241, Kolimpari Chania, Greece, May 2011.
[5] I. Abraham, D. Delling, A. Fiat, A. Goldberg, and R. Werneck. HLDB: Location-based services in databases. In *ACM GIS*, pages 339–348, Redondo Beach, CA, Nov. 2012.
[6] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *ESA*, pages 24–35, Ljubljana, Slovenia, Sep 2012.
[7] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In transit to constant time shortest-path queries in road networks. In *ALENEX*, pages 46–59, New Orleans, LA, Jan 2007.
[8] F. Brabec and H. Samet. Client-based spatial browsing on the world wide web. *IEEE Internet Computing*, 11(1):52–59, Jan/Feb 2007.
[9] P. B. Callahan. *Dealing with higher dimensions: The well-separated pair decomposition and its applications*. PhD thesis, The Johns Hopkins University, Baltimore, MD, Sep 1995.
[10] L. Chang, J. X. Yu, L. Qin, H. Cheng, and M. Qiao. The exact distance to destination in undirected world. *VLDB J.*, 21(6):869–888, 2012.
[11] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *SEA*, pages 376–387, Kolimpari Chania, Greece, May 2011.
[12] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
[13] C. Esperança and H. Samet. Experience with SAND/Tcl: a scripting tool for spatial databases. *JVLC*, 13(2):229–255, Apr. 2002.
[14] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333, Cape Cod, MA, May 2008.
[15] A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for A*: Efficient point-to-point shortest path algorithms. In *ALENEX*, pages 129–143, Miami, FL, Jan 2006.
[16] S. Ma, K. Feng, H. Wang, J. Li, and J. Huai. Distance landmarks revisited for road graphs. *CoRR*, abs/1401.2690, 2014.
[17] S. Peng and H. Samet. Analytical queries on road networks: An experimental evaluation of two system architectures. In *ACM GIS*, Seattle, WA, Nov 2015.
[18] S. Peng, J. Sankaranarayanan, and H. Samet. SPDO: High-throughput road distance computations on spark using distance oracles. In *ICDE*, pages 1239–1250, Helsinki, Finland, May 2016.
[19] M. Qiao, H. Cheng, L. Chang, and J. X. Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. *TKDE*, 26(1):55–68, 2014.
[20] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, CA, 2006.
[21] H. Samet, A. Rosenfeld, C. A. Shaffer, and R. E. Webber. A geographic information system using quadtrees. *Pattern Recognition*, 17(6):647–656, Nov 1984.
[22] H. Samet, H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND spatial browser for digital government applications. *Commun. ACM*, 46(1):63–66, Jan. 2003.
[23] P. Sanders and D. Schultes. Engineering highway hierarchies. In *ESA*, pages 804–816, Zurich, Switzerland, Sep 2006.
[24] J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. In *ICDE*, pages 652–663, Shanghai, China, Apr. 2009.
[25] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *TKDE*, 22(8):1158–1175, Aug. 2010.
[26] J. Sankaranarayanan and H. Samet. Roads belong in databases. *IEEE Data Eng. Bull.*, 33(2):4–11, June 2010.
[27] J. Sankaranarayanan, H. Alborzi, and H. Samet. Distance join queries on spatial networks. In *ACM GIS*, pages 211–218, Arlington, VA, Nov. 2006.
[28] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. *PVLDB*, 2(1):1210–1221, 2009.
[29] E. Tanin, A. Harwood, and H. Samet. A distributed quadtree index for peer-to-peer settings. In *ICDE*, pages 254–255, Tokyo, Japan, Apr. 2005.
[30] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *PVLDB*, 5(5): 406–417, 2012.
[31] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest path and distance queries on road networks: Towards bridging theory and practice. In *SIGMOD*, pages 857–868, New York, NY, Jun 2013.