Distance Oracle in Billion-Node Graph

Baochen QIAO  Junxue ZHANG
Outline

In today’s presentation, we are trying to introduce:

1. What is distance oracle? Why do we need it?

2. A feasible solution proposed by Qi et al. in VLDB 2013.

Note. To satisfy the course requirements, we will skip most of the proof and focus on delivering interesting problems and intelligent solutions. If you are super interested in those theoretical proof, please refer the paper.
Data as Graph

Social Network

Citation Network
Shortest Distance

If you are a direct friend of me, we are close with each other.
Shortest Distance

If you are a direct friend of me, we are close with each other.

If you are a friend of my direct friend, we are just-so-so friend.
Shortest Distance

If you are a direct friend of me, we are close with each other.

If you are a friend of my direct friend, we are just-so-so friend.

The relationship can be further complicated when the edges connected with two vertices have different weights.
Shortest Distance

To answer the shortest distance between any two vertices is important & meaningful concerning mining knowledge on graphs.

If you are a direct friend of me, we are close with each other.

If you are a friend of my direct friend, we are just-so-so friend.

The relationship can be further complicated when the edges connected with two vertices have different weights.
Traditional Methods

There are many research works proposed to solve the shortest distance during decades.

1. Dijkstra

2. Bellman-ford
Challenge and Dilemma

If we are dealing with graphs with billions of vertices, can we still directly apply those traditional algorithms to calculate the shortest distance between two vertices?
Pregel

In dealing with graph with billions node, Google makes their first step by proposing Pregel in 2010

1. Pregel is a synchronized distributed system
2. Algorithm is executed in multiple iterations (called superstep), between each superstep, the worker has to synchronized with the master
3. Pregel adopts “think as a vertex idea”
   a. Receive messages from other vertices
   b. Process the messages locally
   c. Send messages to other vertices
   d. Vertex can become inactive or active and algorithm terminates when all vertices are inactive
   e. Master can also do some global calculation (such as finding maximum value)
Partition the graph

Machine 1
- 0
- 10 to 0

Machine 2
- ∞
- 1 to ∞

Machine 3
- ∞
- 2 to ∞
Shortest Distance using online BFS

When received *messages* from other vertex
Let $m$ be the smallest value with *messages*
If (the value i owned is smaller than $m$):
  Become inactive
Else:
  Become active
  Update the value i owned with $m$
  For any neighboring vertex $v$ :
    Let $e$ be the edge connecting to $v$
    Send $\text{weight}(e) + m$ to vertex $v$
End
End
Shortest Distance using online BFS
Why not use online BFS directly

If you would like your users wait for a very long time

Distribute system usually involves multi-stage communication and synchronization, leading to long processing time
Why not stores the results in a file

Yes we can but the space complexity is $O(n^2)$

Suppose we have 1 billion vertices, the distance is an integer and takes 4 bytes to store under 32-bit OS

We have to consume $1 \times 10^9 \times 10^9 = 10^{18}$ bytes $= 10^9$ GB

How can we store such a big file and query efficiently from this file?

It is impossible at least nowadays
Distance Oracle

Distance Oracle is a **precomputed data structure** to **approximate** the shortest distance.

It should be:

1. At most linear space complexity (have no money to support $10^9$ GB storage)
2. Low time complexity calculation (we are working on billions of nodes)
3. Answer distance in constant time (users are always impatient)
4. Good approximation (we cannot just guess)
Toward a Distance Oracle for Billion-Node Graphs
Figure 1: Architecture of a coordinate-based distance oracle.
Some Background Knowledges

I want ignore it but cannot

Sketch-based distance oracle:

Distance oracles can be done by selecting some landmarks.

Landmarks are like the lighthouse in the sea

You can check the map to know the exact position of every lighthouse

You can always find one lighthouse guide you through to your destination

Then you know your distance to your destination
Shortest-distance-based sketches

The accuracy depends on the selection of landmark, the landmark should be some point where lots of shortest paths pass through.
Coordinate based sketches

What if we can first determine coordinates for some vertices and then the rest of them -> Be easy
Landmark based Coordinate Distance Oracles

Landmarks based + Coordinate-based

Given: Landmarks in Graph Embedding Space (every vertex have coordinate)

So:

\[ |d(u, L_1) - d(v, L_1)| \leq d(u, v) \leq d(u, L_1) + d(v, L_1) \]
S1-- Landmark selection--(MCL)

Shortest-distance-based sketches: **Landmarks**
A good set of landmarks: MCL (minimum complete landmarks)

1. Complete: If all vertex pairs (shortest distance) are covered by the landmark set \( L \), then \( L \) is complete.

2. Minimal: a landmark set \( L \) is minimal if there exists no \( L' \subseteq L \) such that \( L' \) can cover the vertex pairs covered by \( L \).
Shortest-distance-based sketches: Landmarks

MCL-s (minimum complete landmarks size constraint)

A landmark set that is complete is usually very large

Greedily select the most promising vertex as a landmark until the upper limit is reached.
S1--Landmark selection--How to find the MCL-s

1. Calculating the exact betweenness is costly. $O(|V| |E|)$
2. It may not be necessary to find the exact betweenness.
S1--Landmark selection-- Local to Global

We present a parallel algorithm to find the approximate betweenness. The idea is quite simple. On each machine $i$, for each vertex $v \in V_i$, we count the number of shortest paths in the local graph $G'_{i}$ that pass through $v$. This number is the approximate betweenness of $v$.

After determining the local betweenness, each worker sends the betweenness to the master.

Master node uses a sorting algorithm to find the top-$k$ ($k = 100$ here) nodes as the landmark.

Master sends the landmark back to all workers and now each worker has the landmark set.
S1--Landmark selection-- why it works?

1. Lightweight

\[ E(\lvert V'_i \rvert) = \frac{|V|}{k} (1 + \frac{\langle d \rangle (k - 1)}{k}) \]

\[ E(\lvert E'_i \rvert) = |E|(1 - p) = \frac{|E|(2k - 1)}{|k^2|} \]
S1--Landmark selection-- why it works?

2. Correctness

QUALITY OF LOCAL SHORTEST PATHS

- **Case 1:** \( u, v \in V_i \),
  \[ \Delta_i(u, v) \leq \max\{0, d_i(u, v) - 3\} \]

- **Case 2:** \( u \in V_i \) and \( v \in V_i' - V_i \) (or vice versa),
  \[ \Delta_i(u, v) \leq \max\{0, d_i(u, v) - 2\} \]

- **Case 3:** \( u, v \in V_i' - V_i \),
  \[ \Delta_i(u, v) \leq \max\{0, d_i(u, v) - 1\} \]
S2: Calculating shortest distances from vertex to all landmarks

After the landmarks are determined

In order to address the distributed ability we use Online Synchronized BFS to calculate the shortest distances between each landmark and each vertex

1. Each vertex $v$ maintains an $|L|$-dimensional vector $\langle d_1, ..., d_{|L|} \rangle$ requires $O(|L| \cdot |V|)$ space

2. In this work $|L| = 100$. 
S3: Graph embedding - Deriving the coordinates of landmarks.

Learn the coordinates of landmarks by minimizing the difference between the exact distances (calculated in S2) and the coordinate-based distances.

$$\arg\min_{\{c(v_i)| v_i \in L\}} \sqrt{\sum_{1 \leq i \neq j \leq |L|} [\bar{d}(c(v_i), c(v_j)) - d(v_i, v_j)]^2}$$

Then we get the Coordinate of all landmarks $C(L)$
S4: Graph embedding - Deriving the coordinates of all vertices.

We learn the coordinates of each vertex based on their distances to landmarks (calculated in S2) and the landmarks’ coordinates.

\[ \arg \min_{c(u)} \sqrt{\sum_{1 \leq i \leq l} \left[ \overline{d}(c(u), c(v_i)) - d(u, v_i) \right]^2} \]

Notes: we randomly pick one of Landmarks vertex to learn the coordinate of all vertex

S5: Finally Indexing

Each vertex is associated with coordinates \(<x_1, ..., x_c>\) and a distance vector \(<d_1, ..., d_{|L|}>\). We index the data so that we can support fast retrieval of such information by vertex ID.

\(<x_1, ..., x_c>\)
\(<d_1, ..., d_{|L|}>\) \rightarrow \text{space } O( |V| \times |L| )
Given a landmark set L, for any two vertices u, v

Landmark distance

\[ l(u, v) = \max\{|d(u, l) - d(v, l)| : l \in L\} \quad r(u, v) = \min\{d(u, l) + d(v, l) : l \in L\} \]

\[ l(u, v) \leq d(u, v) \leq r(u, v) \]

Coordinate distance

\[ \tilde{d}(x, y) = \arccosh \left( \sqrt{(1 + \sum_{i=1}^{c} x_i^2)(1 + \sum_{i=1}^{c} y_i^2) - \sum_{i=1}^{c} x_i y_i} \right) \times |\delta| \]

Final distance

\[ d^*(u, v) = \begin{cases} 
\tilde{d}(u, v) & \text{if } l(u, v) \leq \tilde{d}_{u,v} \leq r(u, v); \\
l(u, v) & \text{if } \tilde{d}_{u,v} < l(u, v); \\
r(u, v) & \text{if } \tilde{d}_{u,v} > r(u, v); 
\end{cases} \]
Inspirations from those works

Although distributed systems are useful in dealing with big data problems, we still sometimes need intelligent solution to break the tradeoff between space and time complexity

Approximation gives some directions for us to deal with big data problems. We can sacrifice some accuracy for speed and space efficiency
Reference


Backup Slides
Sketch-based Distance Oracles

Shortest-distance-based sketches: Landmarks

Denotes: $d(u,v) \rightarrow$ minimum distance between $u$ and $v$

$L$ -> Landmarks Set

$V$ -> Vertex

If Known: $\{(w, d(u,w)) | w \in L, u \in V\}$

Then: $d(u,v) = \min\{d(u,w) + d(v,w) | w \in L\}$