Efficient Distance Sensitivity Oracles for Real-World Graph Data (Extended Abstract)

Jong-Ryul Lee*, Chin-Wan Chung*†
* School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea
† Chongqing Liangjiang KAIST International Program, Chongqing University of Technology (CQUT), China
hellcodes.kaist@gmail.com, chungcw@kaist.edu

***Abstract*—A distance sensitivity oracle is a data structure answering queries that ask the shortest distance from a node to another in a network expecting node/edge failures. It has been mainly studied in theory literature, but all the existing oracles for a directed graph suffer from prohibitive preprocessing time and space. Motivated by this, we develop two practical distance sensitivity oracles for directed graphs as variants of Transit Node Routing, and effective speed-up techniques with a slight loss of accuracy. Extensive experiments demonstrate that our oracles greatly outperform all of competitors in most cases. To the best of our knowledge, our oracles are the first distance sensitivity oracles that handle real-world graph data with million-level nodes.**

***Index Terms*—Graph algorithms, Path and circuit problems, Graphs and networks, Distance sensitivity oracle, Distance query, Shortest distance, Shortest path algorithms**

I. Introduction

Given a graph $\mathcal{G} = (V, E)$ where V is the set of nodes and E is the set of edges, the distance sensitivity problem is to answer queries that ask to compute the distance of the shortest path from a node to another avoiding the failed part of the network. A distance sensitivity oracle is a data structure which is designed to answer such queries. The distance sensitivity oracle is useful for a network where a small number of recoverable failures (node or edge) can simultaneously and frequently occur. This is because a traditional dynamic distance oracle cannot answer any query while it is being updated for such failures and even a single failure can cause expensive update cost. Depite such usefulness, all existing distance sensitivity oracles for a directed graph with real-valued edge weights are not applicable to real-world networks, because they suffer from prohibitive costs for space and for preprocessing time.

In this work, we propose two efficient distance sensitivity oracles for a variable number of edge failures in directed graphs, which can answer distance queries without any stalling. This feature of them is so critical, because it is necessary for stable latency and enables them to handle multiple queries in parallel, each of which is processed with a separate thread on the same index structure. We also propose two effective speed-up techniques and maintenance strategies for permanent network changes. Finally, we conduct extensive experiments to show the superiority of our distance oracles.

II. Problem Definition

To represent a general network, we consider a directed graph $\mathcal{G} = (V, E)$, where V is the set of nodes and E is the set of edges, as the input graph. For any edge (x, y) in a graph $\mathcal{G}$, (x, y) is associated with a real-valued weight.

Definition II.1 (Distance Sensitivity Problem)**.** *Given a directed graph $\mathcal{G} = (V, E)$, the distance sensitivity problem is a query (s, t, F), where s is the start node, t is the destination node, and F is a set of at most f failed edges, asking to compute the shortest distance from s to t in the graph $(V, E \setminus F)$.*

III. Summary of Our Approach

All the technical details and explanations are included in the full version of this paper [1].

Our first oracle consists of a novel fault-tolerant index structure, which is used to construct a solution path and to detect and localize the impact of network failures, and an efficient query algorithm for it. Our second oracle is made by applying the A* heuristics to the first oracle.

Our oracles are variants of Transit Node Routing (TNR) [2]. The generic version of this technique consists of the following items [3]:

- **Transit Node Set**: A set of nodes $T \subseteq V$ that are supposed to participate in many shortest paths.
- **Distance Table**: A table (or function) $d_T : T \times T \to \mathbb{R}_0^+$, in which $\mathbb{R}_0^+$ is the set of non-negative real numbers, returning the shortest distance between transit nodes.
- **Out-access (in-access) Node Mapping**: A mapping A_{out} (A_{in}) from a node v in V to the set of all transit nodes, called out-access (in-access) nodes, each of which can be the first (last) transit node on a shortest path from (to) v.
- **Locality Filter**: A boolean function that returns true for any two nodes in V if there is no transit node on the shortest path between them, and otherwise returns false.

In *TNR*, for any two nodes s and t, if the locality filter returns false for s and t, the shortest distance from s to t, denoted as $d(s, t)$, is computed as [3]:

$$d(s,t) = \min_{(u,v)\in A(s,t)} \hat{d}(s,u) + d_T(u,v) + \hat{d}(v,t), \quad (1)$$

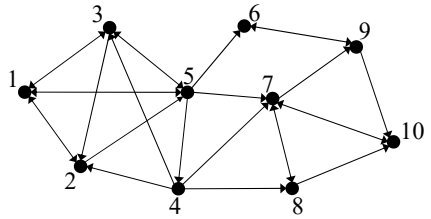
where $A(s, t)$ denotes $A_{out}(s) \times A_{in}(t)$ and $\hat{d}(x, y)$ denotes the distance of the shortest path from x to y which does not pass through any other transit node except x and y. Otherwise, an alternative algorithm is used to compute $d(s, t)$.

Adaptation. Let us explain how we adapt *TNR*. In our oracles, computing the access node mapping is simply done by a modified version of the Dijkstra's algorithm, called the

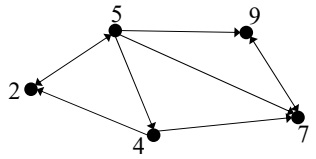
2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00233

bounded Dijkstra's algorithm. The algorithm is designed to avoid traversing beyond transit nodes except the source node. It is easy to see that the set of transit nodes visited by the bounded Dijkstra's algorithm from s is a superset of $A_{out}(s)$, which is denoted by $A^*_{out}(s)$. In addition, the reported distance from s to each node u in the superset is exactly the same as $\hat{d}(s,u)$. A superset of $A_{in}(s)$, which is denoted by $A^*_{in}(s)$, is similarly computed by the bounded Dijkstra's algorithm.

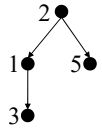
The locality filter is also handled by the bounded Dijkstra's algorithm. If the locality filter returns true, t must be visited by the bounded Dijkstra's algorithm from s and the reported distance from s to t must be the right query answer. Based on this fact, without computing the locality filter explicitly, if the reported distance is smaller than the distance based on (1), our oracles return the reported distance as the query answer.



(a) An input graph



(b) The distance graph with the transit node set $\{2, 4, 5, 7, 9\}$



(c) The path tree $\mathcal{G}_2$

Fig. 1. Examples of an input graph and a two-level index structure

Meanwhile, the distance table cannot be a $|T| \times |T|$ table for the distance sensitivity problem, because the distances stored in it may change by failures. Instead, we devise a novel fault-tolerant index structure, which consists of two levels. The first-level is a small overlay graph $\mathcal{D} = (T, E_{\mathcal{D}})$, called the distance graph, where $E_{\mathcal{D}} \subseteq T \times T$. For any pair $(u, v) \in T \times T$, (u, v) is included in $E_{\mathcal{D}}$, if there exists a path from u to v in $\mathcal{G}$ which does not pass through any other node in T. The weight of the edge (u, v) is the distance of the shortest path from u to v in the graph $(V, E \setminus F)$, which does not include any transit node as an intermediate node. Note that the edge weights in $\mathcal{D}$ are computed in preprocessing with $F = \emptyset$ and some of them are recomputed on demand in query processing. The second-level consists of trees of small sizes, called bounded shortest path trees. The bounded shortest path tree is a path tree given by the bounded Dijkstra's algorithm. An example of our index structure is depicted in Figure 1.

Our First Oracle. The query algorithm of our first oracle is a Dijkstra-like algorithm over $\mathcal{D}$. It first computes $A^*_{out}(s)$ and $A^*_{in}(t)$, and then computes the query answer like the Dijkstra's algorithm on $\mathcal{D}$. If it visits a node u whose bounded shortest path tree contains a failed edge, the weights of the edges from u in $\mathcal{D}$ are recomputed. The second-level of the index structure is utilized for efficient recomputation.

More Sparse Distance Graph. Since the query algorithm traverses $\mathcal{D}$ like the Dijkstra's algorithm, the density of $\mathcal{D}$ is an important factor to query efficiency. In order to construct a good distance graph, we borrow a decent concept called a k-path cover from [4]. Beyond the work in [4], we propose a novel way of selecting a better k-path cover so that a resulting distance graph is more sparse based on the concept of the independent set from the graph theory.

Our Second Oracle. We devise the second oracle by combining the query algorithm of our first oracle with the A* heuristics. This combination is achieved by a novel method of jointly computing a solution path on the distance graph and recomputing some part of the distance graph. This method effectively reduces the search space for the recomputation.

Efficient Speed-up Techniques. We propose two speed-up techniques to make our approach faster for specific classes of networks. The first speed-up technique is called partial detouring, which efficiently provides a detour of a sub-path of the original shortest path. The other technique is distance graph sparsification, which effectively removes out unnecessary edges from the distance graph. The partial detouring is effective for bounded-degree networks, while the distance graph sparsification is effective for scale-free networks.

IV. Evaluation and Results

We evaluate our oracles with real-world networks and various competitors. The real-world networks are three road network datasets and three social network datasets which are the representatives of bounded-degree networks and scale-free networks, respectively.

Our distance sensitivity oracles outperform the competitors in terms of query time in most cases. It is notable that our first oracle has mostly better query performance than even the A* search algorithm with better preprocessing time and space. Our second oracle is the most efficient exact method for the road networks in terms of query time with comparable preprocessing time and space. In addition, we can see that the partial detouring and the distance graph sparsification effectively make our oracles much faster.

Acknowledgements

This work was supported in part by 2019 Seed Money Project of Chongqing Liangjiang KAIST International Program, Chongqing University of Technology, and in part by Chongqing Research Program of Basic Research and Frontier Technology (No. cstc2017jcyjAX0089).

References

[1] J. Lee and C. Chung, "Efficient distance sensitivity oracles for real-world graph data," *IEEE Transactions on Knowledge and Data Engineering*, to be published.

[2] H. Bast, S. Funke, P. Sanders, and D. Schultes, "Fast routing in road networks with transit nodes," *Science*, vol. 316, no. 5824, pp. 566–566, 2007.

[3] J. Arz, D. Luxen, and P. Sanders, "Transit node routing reconsidered," in *Experimental Algorithms*, V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, Eds., 2013, pp. 55–66.

[4] S. Funke, A. Nusser, and S. Storandt, "On k-path covers and their applications," *PVLDB*, vol. 7, no. 10, pp. 893–902, 2014.