**Research on Efficient Hop-Constrained s-t Simple Path Enumeration**

Author: Shi Wenlan

1. **Introduction**

The graph is a data structure used in various areas. It represents information of different entities(vertices) and their relationships(edges). To evaluate the relationship between two entities, we need to enumerate simple paths from one entity (vertex s) to another entity (vertex t). Intuitively, the longer the path, the weaker the relationship between two entities, and enumerating long paths often requires a lot of calculation. So, it is natural to impose a hop constraint k to s-t path enumeration.

To enumerate all simple paths from s to t with the number of hops less than or equal to k in polynomial delay, Peng et al. (2019) proposed Barrier-based constrained DFS (BC-DFS). However, barrier update incurs high overhead, so it is still not fast enough. To meet the rigid time constraint in real-world applications, Sun et al. (2021) proposed the PathEnum algorithm. This algorithm proposed new research ideas such as a lightweight index and join optimizer, and there is still a lot of room for optimization.

If this problem can be solved, this technology can be used in many areas such as detecting money laundering and build knowledge networks, and have good development prospects.

1. **Problem Statement**

Hop constraint s-t path enumeration: Given a graph G, enumerate all simple paths p from a source vertex s to a target vertex t with the number of hops not larger than k.

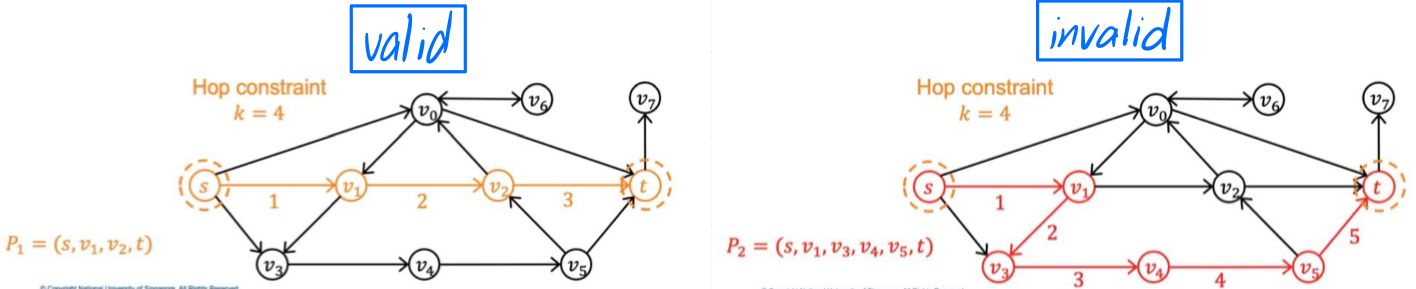


Figure 1: path sample

Figure 1 shows the sample of path that we need.

In this example, for given k = 4, the path we need to enumerate are: s-v0-t, s-v1-v2-t, s-v1-v2-v0-t, s-v3-v4-v5-t and s-v0-v1-v2-t.

And we want to enumerate these paths as fast as possible.

1. **Research Plan & Expected outcome**

Month 1-2: Collect and study related papers, determine breakthrough points for optimization, and implement algorithms.

Direction 1: build an index for hot vertices which are vertices with high degree.

1、identify hot vertices:

(1) Option 1: Use the value obtained by multiplying the total number of vertices by a percentage as the dividing line of “high”. For example, for a graph with 10 vertices, we specify the vertices with degree greater than 3 as hot vertices, and for a graph with 100 vertices, we specify the vertices with degree greater than 30 as hot vertices. This method can get the hot vertices by traversing the vertices in the graph only once. However, if the hyperparameter (percentage) is not set properly, it is likely that there are few hot vertices found, thus failing to achieve the purpose of acceleration.

(2) Option 2: Sort all vertices in descending order according to their degrees, and take the top 30% (or other percentage) of vertices as hot vertices. This method will be slower than option 1, but it can guarantee to find some hot vertices. However, the quality of the hot vertices found may not be very good. Maybe the original graph has a few hot vertices. This will mistake many vertices with very low degrees as hot vertices.

(3) Option 3: Take the 30% (or other percentage) value of the degree of the vertex with the highest degree as the dividing line of “high”. For example, for a graph with the highest degree 10, we specify the vertices with degree greater than 3 as hot vertices.

The pros and cons and optimization of the method still need follow-up research to determine.

2、index data structures for hot vertices

Option: matrix + hash table

3、index building methods: optimize time& space

Option: Breadth-first search, and cut unrelated edges

4、index-based query algorithms

Option: BC-DFS and join oriented approach

Direction 2: graph partitioning, and then build the index on the partitions

1. graph partitioning

Option: divide graph into subgraphs such as the left side of the hot vertex, the right side of the hot vertex, the subgraph without hot vertices and their related edges.

2、index data structures for subgraphs

Option: matrix + hash table

3、index building methods: optimize time& space

Option: Breadth-first search, and cut unrelated edges

4、index-based query algorithms

Option: BC-DFS and join oriented approach

Direction 3: as Sun et al. (2021) discussed, “improve join optimizer by searching the optimal plan in a larger plan space and considering more metrics such as the cost of materializing partial results.” (p.1769)

Direction 4: build a global index in an offline preprocessing step to reduce the cost of construing the query-dependent index.

Month 3-5: Experimental evaluation and algorithm optimization.

Dataset plan to use: US Patents, DBpedia, Web-google, Web-standford, Twitter-social, Baidu-baike, Wiki-trust, Soc-Epinsion1, Web-uk-2005, WikiTalk, Soc-Slashdot0922, LiveJournal, Rec-dating, Bio-grid-yeast and Twitter-mpi.

Month 6: Algorithm optimization and paper writing.

1. **References**

Peng, Y., Zhang, Y., Lin, X., Zhang, W., Qin, L., & Zhou, J. (2019). Hop-constrained st Simple Path Enumeration: Towards Bridging Theory and Practice. Proc. VLDB Endow., 13(4), 463-476.

Sun, S., Chen, Y., He, B., & Hooi, B. (2021, June). PathEnum: Towards Real-Time Hop-Constrained st Path Enumeration. In Proceedings of the 2021 International Conference on Management of Data (pp. 1758-1770).