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

Author: Shi Wenlan

**Abstract:**

We study the hop constraint s-t path enumeration problem: 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. The state-of-the-art provides an effective solution to generate the index of the subgraph, but enumerating paths on the subgraph is still time consuming. And we develop a parallel algorithm to accelerate path enumeration on the subgraph.

**Acknowledgement:**

This project is supported by the grant “Undergraduate Research Awards” awarded by Undergraduate Research Awards Selection Committee in 2021.

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 and Formulation**

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.

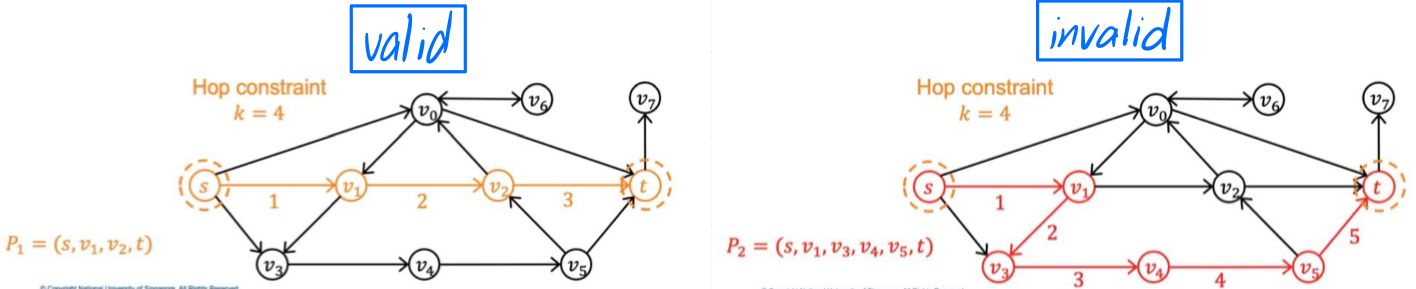


Figure : path sample

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. **Main Results**

My parallelization idea is to split the Depth First Search (DFS) on the subgraph into a one-step Breadth First Search (BFS) plus multiple DFS parallelisms. In detail, my parallel version is to perform a BFS at source vertex *s* first to obtain the out neighbors of *s*. Then, perform multiple DFSs with the out neighbors of *s* as the start vertex, target vertex *t* as the target vertex, within the depth limit (k-1) and without going through *s* (e.g., if *s* has three out neighbors *a*, *b* and *c*, then we need three DFSs to enumerate paths excluding *s* from *a* to *t*, from *b* to *t*, and from *c* to *t*). And excluding *s* can be implemented by marking *s* in the *visited* array. The tasks of multiple DFSs should be assigned to different threads for simultaneous execution.

Pseudocodes are as following:

Node{

uint32\_t parent;

uint32\_t element;

int step;

};

CycleEnumerator::parallel\_dfs(a) {

count = 0

visited = array of bool type with (number of vertices of the graph) \* false

visited[s] = true

visited[a] = true

sta = stack of Node type to store the vertices to explore next

sta.push({s, a, 1})

res = array of length (k + 1) to store the path from s to t, initiating as (k + 1) \* (-1)

res[0] = s

parent\_layer\_pointer = 0

while (sta is not empty){

step = sta.back().step;

v = sta.back().ele;

par = sta.back().parent;

if (step > parent\_layer\_pointer +1){

while (parent\_layer\_pointer < step-1){

vis[par] = true

res[++parent\_layer\_pointer] = par

}

}

else if (step < parent\_layer\_pointer + 1){

for (;parent\_layer\_pointer > step - 1; parent\_layer\_pointer --){

vis[res[parent\_layer\_pointer]] = false;

res[parent\_layer\_pointer] = -1;

}

}

sta.pop();

if (step >= k) continue;

budget = k - step - 1;

start = start neighbor index of v

end = end neighbor index of v within budget hops

for (i = start; i < end; ++i) {

w = neighbor vertex of index i

if (w == t) {

count += 1;

}

else if (step == k - 2 && visited[w] == false) {

count += 1;

}

else if (visited[w] == false) {

sta.push\_back({v ,w, step + 1})

}

}

}

return count

}

CycleEnumerator::dfs\_on\_subgraph(s, k) {

budget = k - step - 1;

start = start neighbor index of v

end = end neighbor index of v within budget hops

neigh = vector to store the neighbors of s using (degree, neighbor) pair

int j = end - start

for (i = start; i < end; ++i) {

v = neighbor vertex of index i

degree = number of neighbors of v that within budget

neigh.push\_back({degree,v});

}

sort the neigh according to degree from to low

omp\_set\_num\_threads(NumThreads);

#pragma omp parallel for reduction(+:total\_count) schedule(dynamic)

for (int i = 0; i < j; ++i) {

v = the i-th neighbor vertex in neigh

if (v == t) {

total\_count += 1

}

else if (k == k - 2) {

total\_count += 1

}

else {

total\_count += parallel\_dfs(v)

}

}

}

We are still writing the paper and will submit in the next several months. We will acknowledge the URA in the paper.

1. **Simulation or Experiment Results (if necessary)**

**Data Set**s are from <http://snap.stanford.edu/data/> and <http://networkrepository.com/networks.php>

**Edges** / **Nodes** means number of edges / nodes of the data set.

**k** is the hops number constraint.

**Threads** is the number of threads used by parallel algorithm, and due to the number of neighbors of the source nodes might be less than the number of threads, the utilization ratio might be less than 100%.

**Without Parallel** is average query time (unit: second) of path-enume algorithm on 1000 queries from hot-point (the top 10% vertices of degree size) to hot-point.

**With Parallel** is average query time (unit: second) of my parallel version algorithm on the same 1000 queries for path-enume algorithm.

**Acceleration Ratio** = Without Parallel / With Parallel.

Following is the table of some representative experiment data. It comes out that the denser the graph, the higher the acceleration ratio at the given k and number of threads.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data Set | Edges | Nodes | k | Threads | Without Parallel (s) | With Parallel (s) | Acceleration Ratio |
| soc-Epinions1 | 508837 | 75879 | 7 | 8 | 58.6607 | 8.3900 | 6.9917 |
| soc-Epinions1 | 508837 | 75879 | 7 | 20 | 58.6607 | 5.5541 | 10.5615 |
| web-google | 5105039 | 875713 | 10 | 20 | 9.6629 | 1.5895 | 6.0788 |
| wiki-topcats | 28511807 | 1791489 | 7 | 8 | 6.7417 | 1.5827 | 4.2595 |
| SocLivejournal1 | 69M | 5M | 7 | 8 | 93.7358 | 12.2595 | 7.6459 |
| SocLivejournal1 | 69M | 5M | 7 | 20 | 93.7358 | 8.4033 | 11.1545 |
| SocSlashdot0922 | 948K | 82K | 7 | 8 | 210.8135 | 30.2977 | 6.9580 |
| SocSlashdot0922 | 948K | 82K | 7 | 8 | 210.8135 | 22.1968 | 9.4974 |
| WebStanford | 2.3M | 282K | 7 | 8 | 0.2190 | 0.0249 | 8.7794 |
| WebStanford | 2.3M | 282K | 7 | 20 | 0.2190 | 0.0157 | 13.9195 |
| Web-uk-2005 | 334K | 121K | 7 | 8 | 483.0557 | 55.1189 | 8.7638 |
| Web-uk-2005 | 334K | 121K | 7 | 20 | 483.0557 | 22.8500 | 21.1402 |
| Web-uk-2005 | 334K | 121K | 7 | 30 | 483.0557 | 20.1451 | 23.9787 |
| Web-uk-2005 | 334K | 121K | 7 | 40 | 483.0557 | 19.3940 | 24.9073 |
| WikiTalk | 5M | 2M | 7 | 8 | 497.5287 | 107.0648 | 4.6469 |
| WikiTalk | 5M | 2M | 7 | 20 | 497.5287 | 90.2006 | 5.5157 |

1. **Appendix (if necessary)**

(Supporting documents can be put here)

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).