Parallel and Space-Efficient Graph Estimators

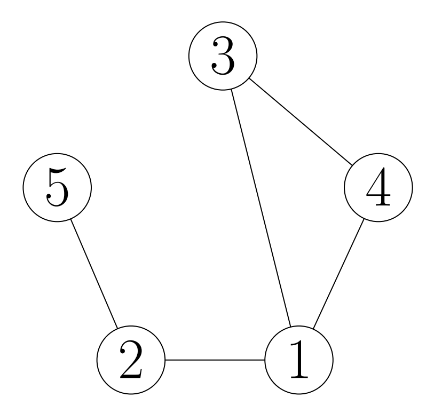
Steven Lu, Supervised by Guy E. Blelloch

Carnegie Mellon University, Computer Science Department

Introduction and Background

We present a data structure for estimating certain graph properties in unweighted graphs of low diameter. This data structure is both highly parallel and space-efficient, allowing for it's use for very large graphs. The data structure is built upon a combination of least-elements lists (le-lists) [3] and FRT trees [4].

Formally, the le-list for for vertex v, denoted $L(v)$, is a sequence of vertices u_i. We associate each vertex v with a random priority $p(v)$ and let $d(u,v)$ denote the distance between vertex u and vertex v. For each sequence $L(v)$, we have that $d(v, u_i) > d(v, u_{i+1})$ and $p(u_i) > p(u_{i+1})$. In practice, we typically store the distance $d(v, u_i)$ along with the vertex u_i in the list.



$L(1) = 5, 4, 1$ $\quad L(2) = 5, 2$ $\quad L(3) = 5, 4, 3$

$L(4) = 5, 4$ $\quad L(5) = 5$

Figure 1: The le-lists for the shown graph, using $p(v) = v$

Key Observations

There are two main observations regarding le-lists:

1. The highest priority vertex will be at the start of each le-list.
2. Each vertex will be at the end of its own le-list.

One way to think of an le-list of a particular vertex is to start with a list of vertices sorted by distance, and keep only the vertices with higher priority than everything that came before it.

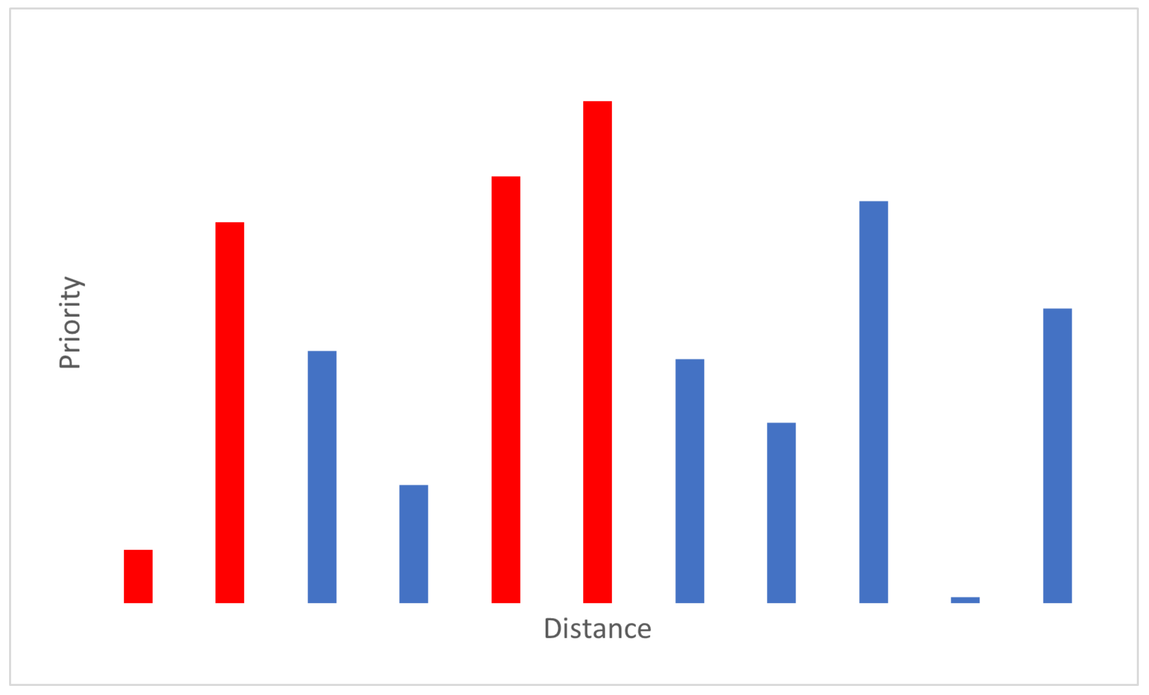


Figure 2: Visualization of vertices and their priority sorted by distance to source, with vertices appearing in the source's le-list highlighted in red.

Computing Le-lists in Parallel

We implement a parallel algorithm to construct le-lists for undirected graphs, based on the method described in [1]. We focus in particular on graphs with low diameter graphs, which provides the greatest amount of parallelism. Social media graphs are real world graphs with such properties.

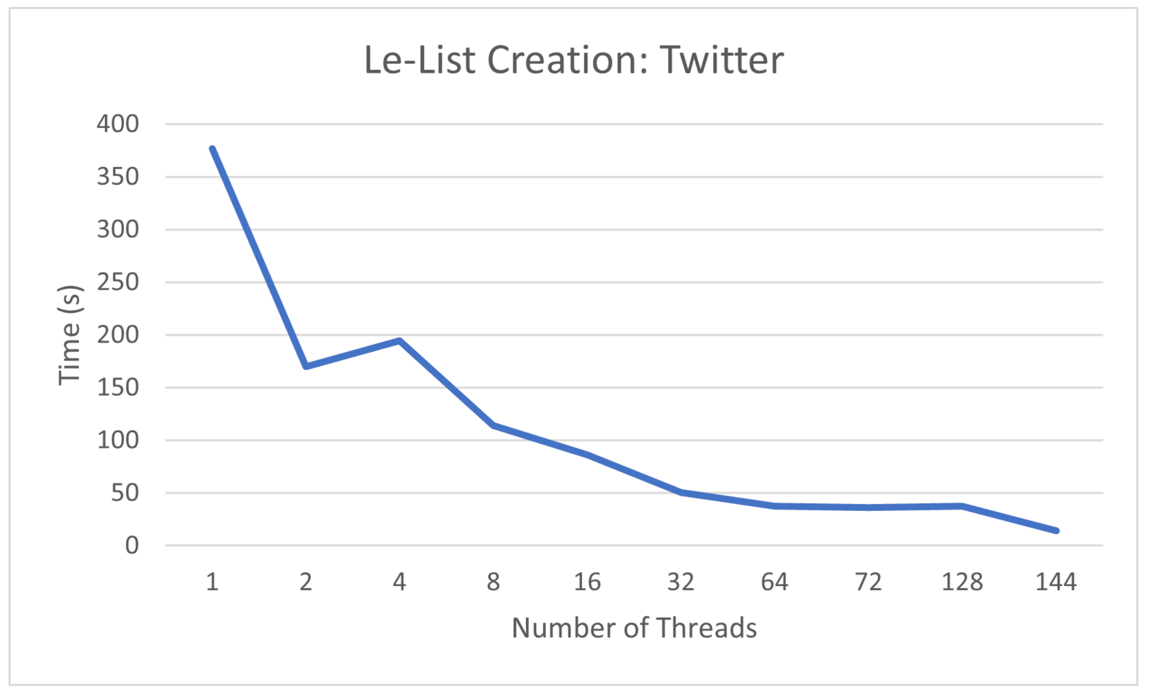


Figure 3: The performance of the parallel algorithm on a graph with 41652230 vertices and 1468364884 edges. With 144 threads, we achieved a self-speedup of 27x.

Convert Lists to Trees

The prior observations about le-lists allow us to structure them as a tree. Using a method described in [2], we can treat each le-list as a "string" of vertices, and construct a prefix trie. Notice that since the highest priority vertex will be at the start of every le-list, all vertices will have the same root. Additionally, since each vertex is at the end of it's own le-list, each leaf in the trie corresponds to a vertex. Converting the list to this structure allows us to save space by not storing repeated "prefixes", but still retain the original information in the list by storing the distances to all ancestors in the tree.

Neighborhood Size Estimation

One estimate our data structure supports is neighborhood size estimation. Formally, the d-neighborhood of a vertex v is the set of all vertices u which are within distance d of v.

Distance Estimation

The second query supported by our data structure is estimating the distance between two vertices. This can be used to create an approximate distance oracle (ADO) which allows fast queries of the distance between two vertices.

Results

We ran benchmarks of our data structure on various social network graphs. The parameter k refers to the number of trials used in each estimation, and can be thought of as the number of "samples", with higher k leading to better accuracy.

Our results show good accuracy in the estimates of neighborhood sizes, with results comparable to the accuracy of ANF, another method of estimating graph neighborhood sizes described in [5].

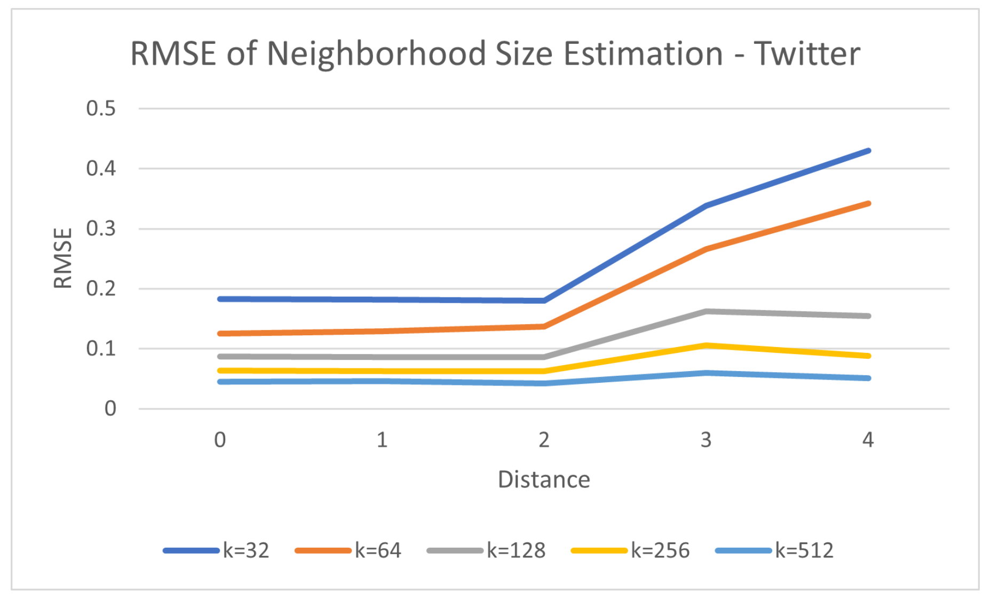


Figure 4: Accuracy of neighborhood-size estimates for different values of d and k.

Our estimates for distances is also quite good, especially for pairs of vertices which were closer together.

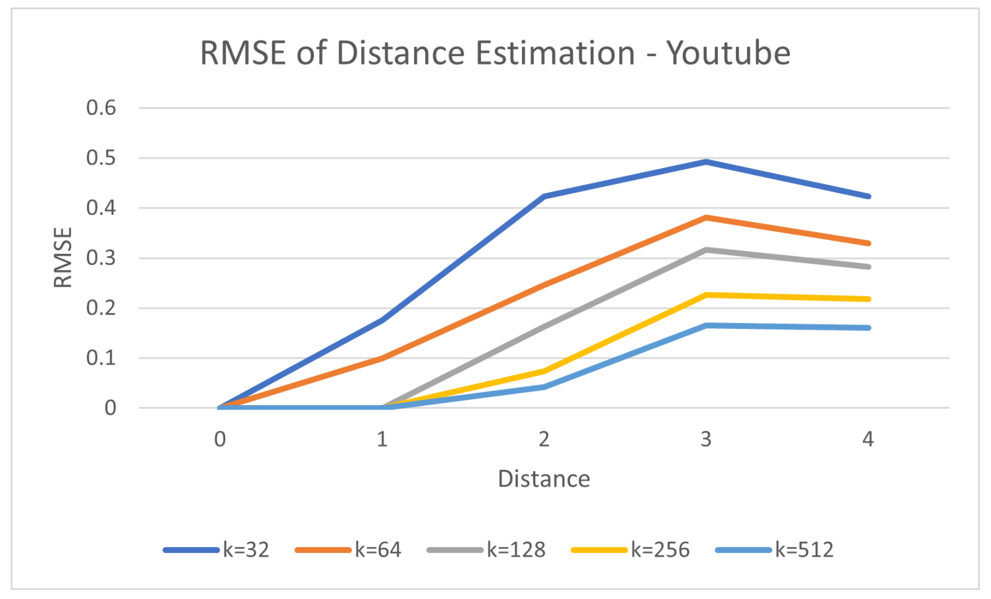


Figure 5: Accuracy of distance estimates for different values of d and k

Our results showed that our data structure provided good accuracy, comparable with previous work [5] This, along with the benefits of parallelism in the data structure creation allowed us to run benchmarks on larger graphs with more vertices.

References

[1] G. E. Blelloch, Y. Gu, J. Shun, and Y. Sun.
Parallelism in randomized incremental algorithms.
ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2016.

[2] G. E. Blelloch, A. Gupta, and K. Tangwongsan.
Parallel probabilistic tree embeddings, k-median, and buy-at-bulk network design.
ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2012.

[3] E. Cohen.
Size-estimation framkework with applications to transitive closure and reachability.
Journal of Computer and System Sciences, 1997.

[4] J. Fakcharoenphol, S. Rao, and K. Talwar.
A tight bound on approximating arbitrary metrics by tree metrics.
J. Comput. System Sci., 2004.

[5] C. R. Palmer, P. B. Gibbons, and C. Faloutsos.
ANF: A fast and scalable tool for data mining in massive graphs.
KDD, 2002.