Small-World in Social network

CS6240 Final Project, Group 1

David Aron Xing Du Srujan Sai Koduru Yun Peng

Apr 23rd, 2019

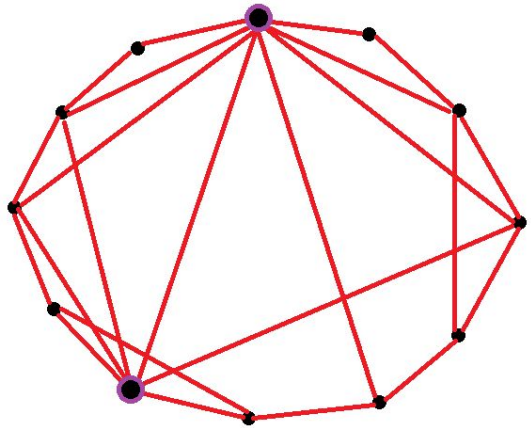
Khoury College of Computer and Information Science

Problem of interest

Does the small-world property holds on Twitter Dataset?

A **small-world network** is a type of mathematical graph

- most nodes are not neighbors of one another,
- the neighbors of any given node are likely to be neighbors of each other
- most nodes can be reached from every other node by a **small** number of hops/steps.
- the mean shortest path between nodes increases sufficiently slowly as a function of total nodes



https://en.wikipedia.org/wiki/Small-world_network

Project task

- Measure the distances between all pairs of Twitter Users.
  We first tried on a small graph (max filter = 10,000)
- Single Source Shortest path (BFS, min-heap Dijkstra)
- All Pair Shortest path (map-side join)
- A simple random sample might not be enough,
- we tried other graph sampling algorithms.
- Simple Random Walk (SRW)
- Random Walk with Fly back (RWF)
- Total induction edge sampling (TIES)

Single Source Shortest Path with BFS

- start at the source node S
- put this in current level bucket
- while we have not ran out of graph (next level bucket is not empty) and not found our target node
  - take a node from current level bucket – call it C
  - for each node reachable from current node C – call it N
    - (we can also check to see if it is visited node and skip it to avoid the cycles)
    - mark the predecessor of N as C
    - Check if N is our target node (that is have we reached the target node)
      - If YES, then exit
  - put this N in next level bucket
  - mark next level bucket as current level

Time complexity: O(V + E) for single source

Speed-up

- The program show poor parallelization

SSSP(BFS)	Time Taken
K = 1000, 5 workers	38 Minutes
K = 1000, 10 workers	38 Minutes

Scalability

- The program took long to compute subgraph k = 10,000

SSSP(BFS)	Time Taken
K = 10000, 10 workers	17.3hr

Improvement: use min-heap

- Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and distance value of the vertex.
- Initialize Min Heap with source vertex as root (the distance value assigned to source vertex is 0). The distance value assigned to all other vertices is INF (infinite).
- While Min Heap is not empty, do following
  - Extract the vertex with minimum distance value node from Min Heap. Let the extracted vertex be u.
  - For every adjacent vertex v of u, check if v is in Min Heap. If v is in Min Heap and distance value is more than weight of u-v plus distance value of u, then update the distance value of v.

Time complexity: O(Log(V)) for single source

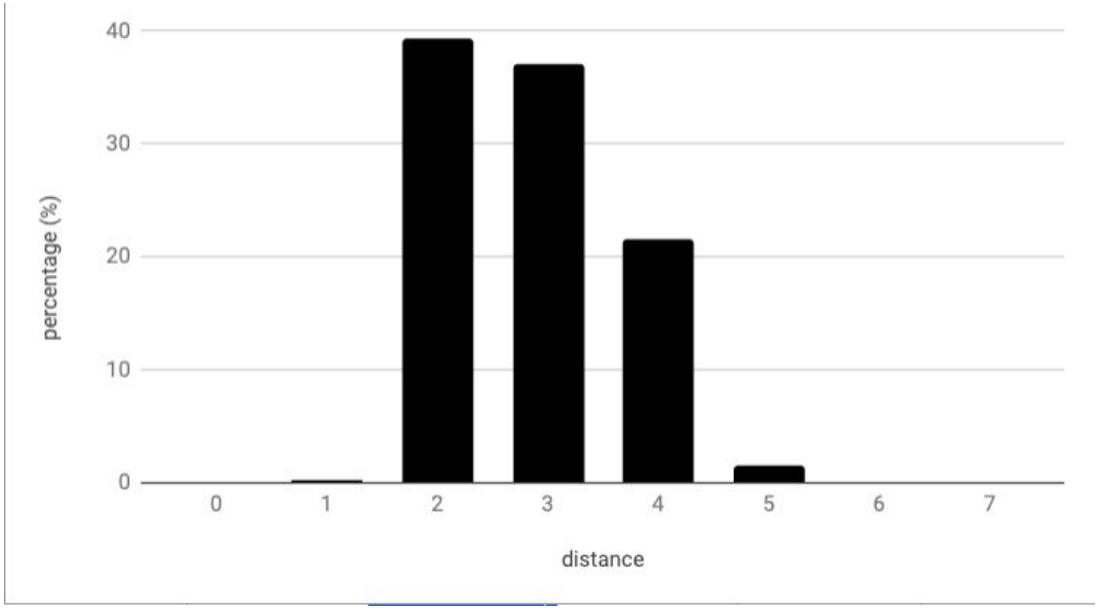
Speed-up & Scalability

- The program show poor parallelization
- Took shorter time to compute on subgraph

SSSP(min-heap)	Time Taken
K = 10000, 5 workers	1.9 hr
K = 10000, 10 workers	1.9 hr
SSSP(BFS)	Time Taken
K = 10000, 10 workers	17.3hr

Distribution of Shortest Path in Subgraph

max filter k = 10,000



distance	percentage (%)
0	0.01222086712
1	0.1579422079
2	39.41431717
3	37.12849426
4	21.589699
5	1.58806727
6	0.1090391802
7	2.20E-04

Peak at 2
76.7% within 3 hops
98.3% within 4 hops

Challenges

- We tested for inputs with k = 10000, but if run on the entire dataset the program would take weeks (or maybe longer) to run.
- No speedup is generated when run on multiple systems though, because neither BFS nor Dijsktra can be easily parallelized
- We need something not based on computing single source shortest path

Improvement: map-side join

- To try and speed up the performance of the program we decided to try and find the next “layer” of shortest paths in each iteration in parallel.
- Each iteration looks to see what nodes each node can reach and updates their distances accordingly.
- The program is then a single job as opposed to multiple single source shortest path jobs
- This is done by using a Map side join which joins on each node in the adjacency list of each node. No reduce phase is needed

Pseudo Code

```
Mapper
    adjacencyListMap

    Setup(){
        For each file in the cache:
            add each line to to adjacencyListMap
    }

    map(node n(with adjacency list of reachable nodes)){

        list = []

        for item in node's adjacency list:
            lookup value in adjacencyListMap:
                add all paths possible from adjacencyListMap to list if they are a new shortest path
        n.add(list)

        emit(null, n)
    }
```

Results

Job details	Time taken
5 Workers, k = 5000	37 Minutes, 38 seconds
10 workers k = 10000	21 Minutes, 17 Seconds

- In terms of scalability, the program scales well until a certain value of k is reached (around 10k) . The program is no longer feasible at this point.

Challenges

- The program works effectively in parallel for inputs of k < 10000
- When input gets too large the program is not able to handle the amount of text in a single line, and the amount of text broadcasted to all mappers. This causes a significant slowdown to occur
- Once the size of the adjacency list gets too large MR will no longer be able to keep the line in memory. This puts a limit on how much data can be processed.

Graph Sampling: node exploration

- Simple Random Walk Sampling (SRW)
- Uniformly at random pick a starting node
- then simulate a random walk(select neighboring node uniformly and randomly) on the graph.
- Random walk is continued until we reach the required sample size.
- Random Walk Sampling with Fly Back Probability (RWF)
- The Fly-back probability *(p)* is used to sample more than one neighboring node at any stage of already sampled node.
- This ensures that the neighborhood of a selected node could be sufficiently explored.
- *RWF* picks a node uniformly at random as start point and begins a sequence. At each step,
    - with *p* probability, we will fly back to the starting point.
    - with *1-p* probability it selects one node among neighbors of the current node with equal probability and moves to that node
        - If the neighboring node or the corresponding edge does not exist in the sample graph, they will be added to the graph

Graph Sampling: node exploration

- Problem of SRW and RWF
- *SRW* and *RWF* fundamentally biases the structure of the sampled subgraph, as at every step we choose only one neighbor uniformly and randomly of the node we sampled at the previous iteration.
- When a node is selected for inclusion in the sample, it is unlikely that all of its neighbors will be included in the sampled subgraph, and thus, sampled degrees of nodes tend to be smaller than original degrees.
- As random walk moves in the linear fashion, the connectivity in the sampled subgraph was also quite sparse due to under-sampling of edges.
- This under-sampling of edges caused overestimation of shortest path lengths in sampled subgraphs.

Graph Sampling: node exploration

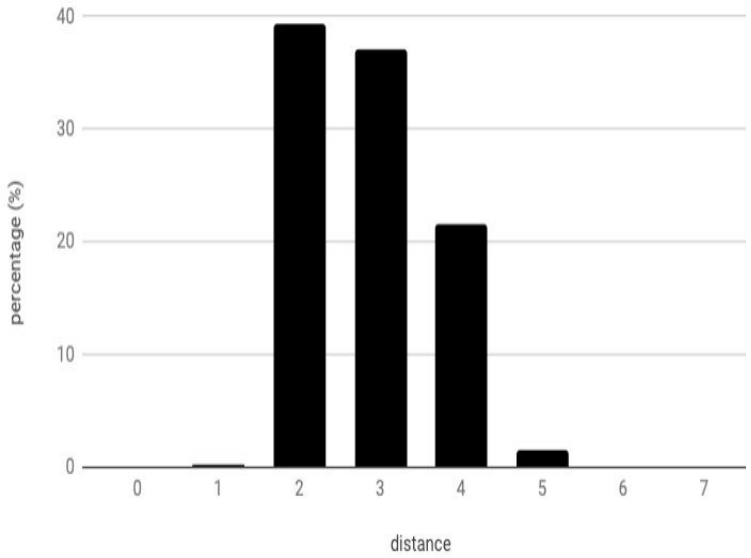
- Induced Subgraph Random Walk Sampling (ISRW)
- *Induced Subgraph Random Walk Sampling (ISRW)*, which tries to overcome the problem of undersampling of edges in *SRW*.
- We applied graph induction step to *SRW* to select additional edges between sampled nodes with the aim to restore connectivity and bring the structure closer to that of the original graph.

Graph Sampling: edge sampling

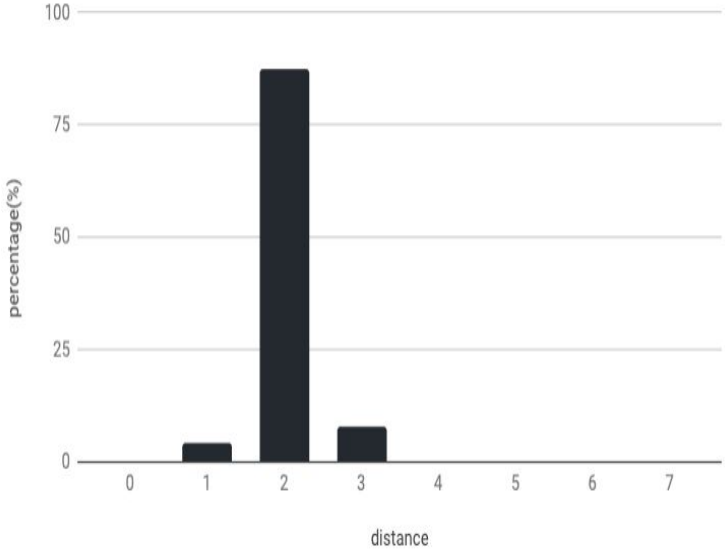
- Total Induction Edge Sampling (TIES)
- The algorithm runs in an iterative fashion, picking an edge at random from the original graph and adding both the nodes to the sampled node set in each iteration as in the classic edge sampling approach.
- It stops adding nodes once a target fraction φ of nodes are collected.
- After this, the algorithm proceeds to the graph induction step where it walks through all the edges in the graph and forms the induced graph by adding all edges which have both end-points already in the sampled node set.

Test Sampling algorithm

max filter k = 10,000



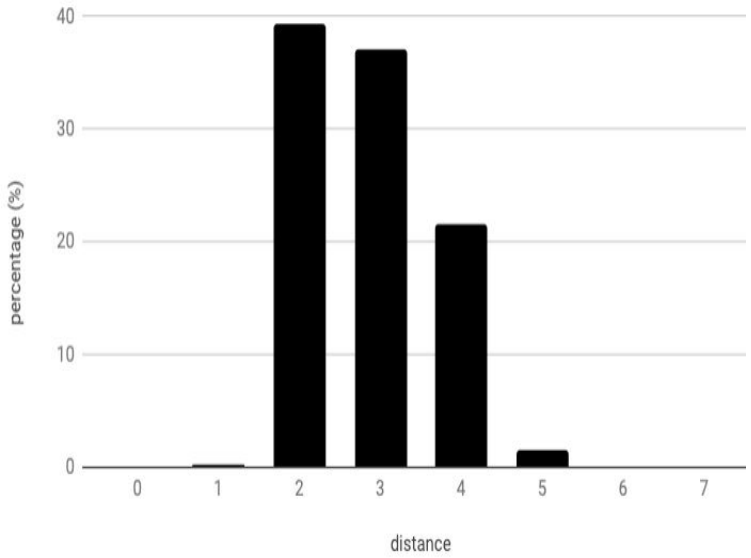
ISRW, sample 1000 nodes



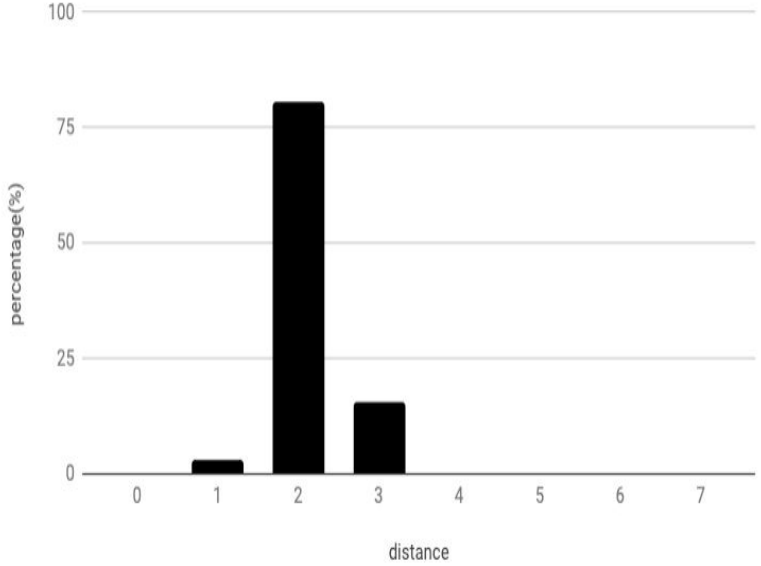
cosine similarity: 0.732

Test Sampling algorithm

max filter k = 10,000



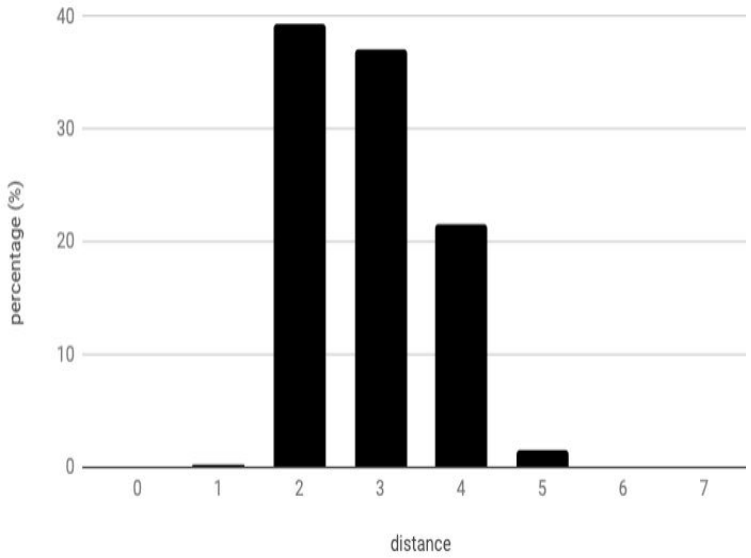
IRWF, sample 1000 nodes



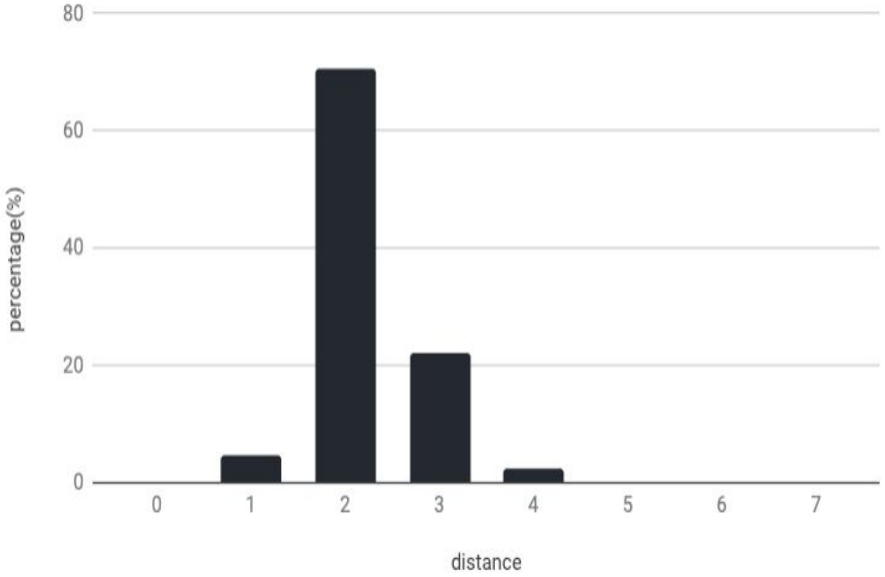
cosine similarity: 0.786

Test Sampling algorithm

max filter k = 10,000



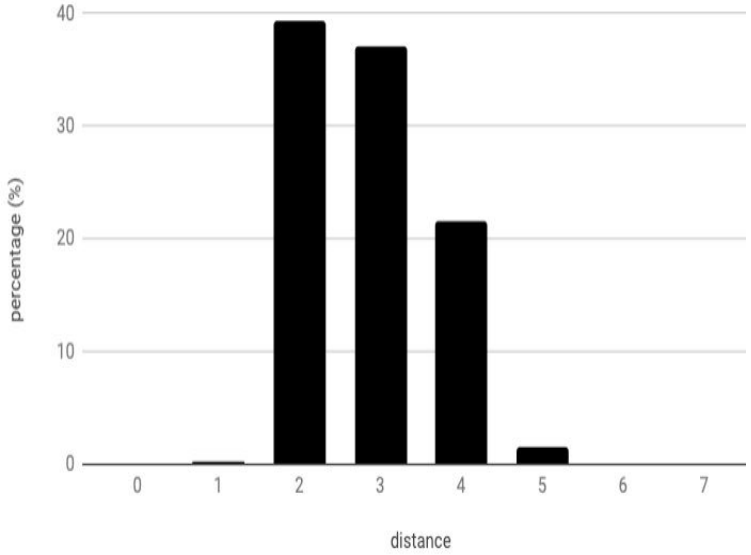
TIES, sample 1000 nodes



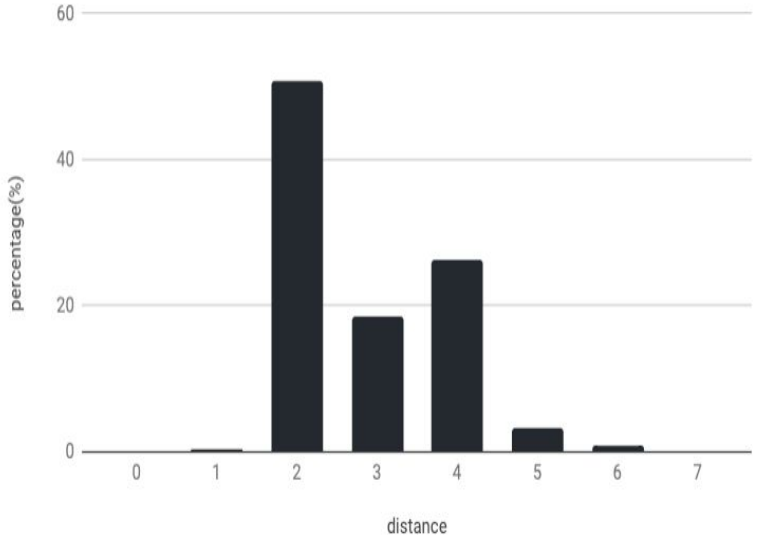
cosine similarity: 0.845

Test Sampling algorithm

max filter k = 10,000



max filter k = 1000



cosine similarity: 0.929

Conclusions

- We were interested to find that the number of hops between each person in the twitter dataset was not very big
- This was surprising because there are large groups of people that don’t know each other
- This shows that we are more connected as a society than one might expect
- Graph sampling is tricky, random sampling might be good enough in some cases