SIGMOD Programming Contest 2014

Large Social Network Analysis

Team: **blxlrsmb**
Presented by Yuxin Wu

Department of Computer Science and Technology
Tsinghua University

June 24, 2014

Overview

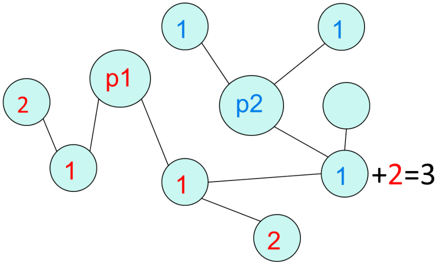
Task

Implement a social network analysis system.

Given a huge social network as an undirected graph, we should implement an analysis system which reads dataset & 4 types of queries and produce the results.

Query 1

Given x and vertex p_1, p_2 , calculate the length of shortest path from p_1 to p_2 s.t. each edge e on the path has weight $e_w \geq x$



Solution: Execute BFS from both p_1 and p_2. Use a bit-array maintaining all the visited vertices, to see if they have met in the middle.

Query 2

Given k and d, find top-k tags with largest range. Here the range of a tag T in graph $G(V, E)$ is defined as the size of the largest subgraph $G(V', E')$ s.t. $\forall v \in V, v_d > d, T \in v_t$.

Solution: First sort all the queries and vertices by d in descending order. Build an empty graph G_0 and incrementally insert sorted vertices and their corresponding edges to G_0 . During the insertion, we maintain the top-k largest connected components for each tag using Union-Find Set. A query q would finish as soon as all vertices v with $v_d > q_d$ are inserted.

Query 3

Given k , h , p , where p is used to define a subset V of V. Find the top-k pairs of vertex (u, v) ordered by $|u_t \cap v_t|$, also satisfying: $u, v \in V, dist(u, v) \leq h$.

Solution: Build an inverted list for every tag T: $L(T) =$ a list of vertex $v \in V$ where $T \in v_t$. This way, we can quickly find vertices in V sharing tags with a given v by counting # of occurrences in inverted lists.

Query 4

Given k , t , where t is used to define a subgraph G of G, find the top-k centralized vertex in G . Here the centrality for a vertex v in a graph $G(V, E)$ is defined as:

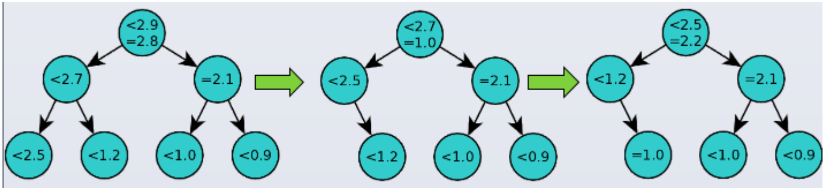
$$C(v) = \frac{r(v)^2}{(|V| - 1)S(v)},$$

where $r(v)$ is $\#$ of vertex reachable from v (exclusive), $S(v) = \sum_{u \in V} dist(u, v)$ is very expensive to calculate.

Framework

General framework to obtain top-k centrality:

1. Estimate an upper bound of $C(v)$ for each v and maintain them with a max-heap.
2. Iteratively pop the top element from the heap and calculate its actual value. If the actual value is still the largest in heap, then it's certainly the largest among all actual values of remaining element. Otherwise we insert this actual value back to the heap.



Lower Bound of $S(v)$

$$S(v) \geq \sum_{dist(u,v) \leq \ell} dist(u,v) + \sum_{dist(u,v) > \ell} (\ell + 1)$$

The first term is calculated by a BFS from v with search depth limited to ℓ. The second term is obtained by counting.

To choose a proper ℓ, we compare our current estimation of each $S(v)$ to an approximation of $S(v)$ and increase ℓ when the mean- square error is large. Since social networks have **small diameter(!)**, a small ℓ (3 or 4) is sufficient to guarantee a good lower bound estimation of $S(v)$.

Approximation of $S(v)$

Random sample a subset V_1 of V, and run a thorough BFS on each of them, giving $dist(u, v)$ for each $u \in V_1$ and $v \in V$. Then we have:

$$\forall v \in V, S(v) \approx \sum_{u \in V_1} dist(u, v) \frac{|V|}{|V_1|}$$

This is a highly accurate approximation, which gave an average 1∼2% error sampled with 0.1% of V in our experiments.

This approximation can also help with pruning: by calculating actual centrality of the vertices with top-$(3k)$ approximated centrality, we get $3k$ actual centrality that is very likely to contain the actual top-k centralities. Then the kth centrality among the $3k$ results is a very good lower bound of the top-k centrality.

Other tricks

1. Use mmap() to read data files faster.
2. Implement a thread pool to utilize multi-core processors.
3. Google's tcmalloc is faster to allocate memory.
4. Another way of BFS in estimating lower bound in query 4 is to use a bitset $B_\ell(v)$ of length $|V|$ whose uth bit is set iff. $dist(u, v) = \ell$. Then we have:

   $$B_{\ell+1}(v) = \neg B_\ell(v) \wedge (\vee_{(u,v)\in E} B_\ell(u))$$

   This search can be implemented using SIMD instructions.

Thanks!