Ego-net Sketching for Streaming Graph Analytics

Bortik Bandyopadhyay, David Fuhry, Aniket Chakrabarti and Srinivasan Parthasarathy
Department of Computer Science and Engineering
The Ohio State University
{bandyopadhyay.14, fuhry.4, chakrabarti.14, parthasarathy.2}@osu.edu

ABSTRACT

We propose a novel, scalable, and principled graph sketching technique based on min-wise local neighborhood sampling. For an n-node graph with e-edges, we incrementally maintain an in-memory min-wise neighbor sampled sub-graph, bounded by a user configurable memory limit. This sketch representation, capable of handling real-time edge streaming rate, lowers the memory requirement to $O(n)$ instead of $O(e)$, making it particularly useful for streaming graphs commonly with $e \gg n$, with both n and e possibly unknown apriori. Symmetrization and similarity-based techniques can recover from these data structures a significant portion of the original graph. With bounded memory, the quality of results using the sketch representation is competitive against baselines which use the full graph, and the computational performance is often significantly better. Our framework is flexible and configurable to be leveraged by numerous other graph analytics algorithms.

1. OUR FRAMEWORK

Minwise independent permutation based hash functions have seen ubiquitous use in graph and network problems, in the context of graph sparsification [10], community detection [8, 9], dense subgraph detection [4], link prediction [11] and computing various measures of interest like local triangle count [1]. In this paper, we use minhash in a manner orthogonal to its traditional usage. To the best of our knowledge, it's use has not been suggested as a fixed size sketch for an edge-streamed graph with low memory footprint. We additionally provide theoretical insights on the type of information retained by this representation.

Figure 1 shows a toy example of the min-wise neighborhood sampling, graph construction, and edge recovery of our graph sketching framework. Each row of M_{k} is initialized with self-loop and C with zero. The edges of source graph G are processed iteratively by Algorithm 1 to construct count vector (C) and sketch matrix (M_{k}) using k different linear min-wise independent hash functions (h_m) [3, 2]. Each node i in graph G is represented by row i in M_{k}, which is a min-wise sample of i's egonet. Next, unique neighbors of each node (row) in M_{k} form directed graph G^*, which is symmetrized to generate G_m. Additionally, using M_{k} and C, G_m is augmented with similarity induced edges thereby generating G_s, which might be useful for scenarios where a substantial portion of the original graph is lost due to sampling (like Twitter data with its power-law degree distribution). Additionally, the user can run a myriad of existing algorithms directly on G_m and G_s.

2. METHODOLOGY

2.1 Sketch Creation and Updating

Algorithm 1 Update Sketch Matrix

Parameter: Sketch Matrix M_{k}
Parameter: Count Vector C
Parameter: new edge (i,j)
for $m = 1$ to k **do**
  if $h_m(j) < h_m(M_{\mathrm{k}}[i,m])$ **then**
    $M_{\mathrm{k}}[i,m] = j$
  end if
  if $h_m(i) < h_m(M_{\mathrm{k}}[j,m])$ **then**
    $M_{\mathrm{k}}[j,m] = i$
  end if
end for
$C[i]++;\ C[j]++;$

2.2 Key Theoretical insights

We analyze the retention probability per edge due to minwise sampling and then use it to construct an unbiased estimator of the total number of edges to be retained in G_m. The proofs have been omitted due to lack of space.

LEMMA 2.1. *For any node i with degree d_i, the probability of losing any edge (i,j) of G in G^* with k hashes is $(1-\frac{1}{d_i})^k$.*

LEMMA 2.2. *The inclusion probability p_{ij} of any edge (i,j) of G in G_m is*

$$p_{ij} = 1 - [(1 - \frac{1}{d_i}) \times (1 - \frac{1}{d_j})]^k$$

LEMMA 2.3. *From G_m, an unbiased estimator of the total number of edges of G using edges E_m of G_m is*

$$\sum_{\{(i,j):E_m \in G_m\}} \frac{1}{(1 - [(1 - \frac{1}{d_i}) \times (1 - \frac{1}{d_j})]^k)}$$

3. EXPERIMENTS

We implemented all code in C++ and ran the experiment on a 3.40GHz Intel(R) Core(TM) i7-2600 machine with 256

[1] In this example let the randomized h_1 permutation be $1, 5, 2, 4, 3$, that is, $h_1(1) < h_1(5) < \ldots < h_1(3)$, and h_2 permutation be $4, 1, 5, 2, 3$.

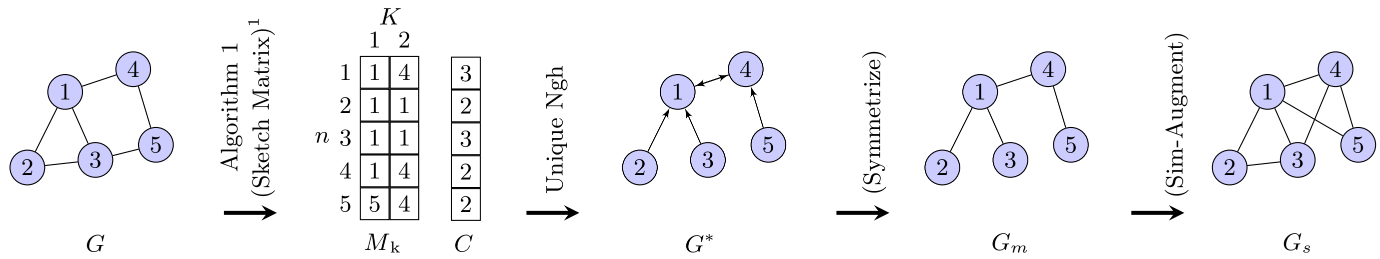

Figure 1: Toy example for Min-wise Neighborhood Graph Sketching framework

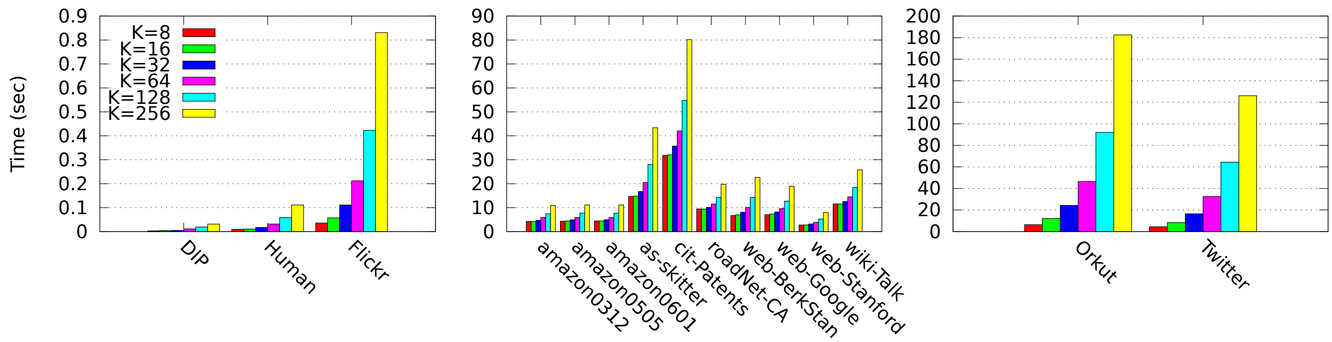

Figure 2: **Sketch matrix construction (Algorithm 1) time varying k.** Datasets grouped by size for scale.

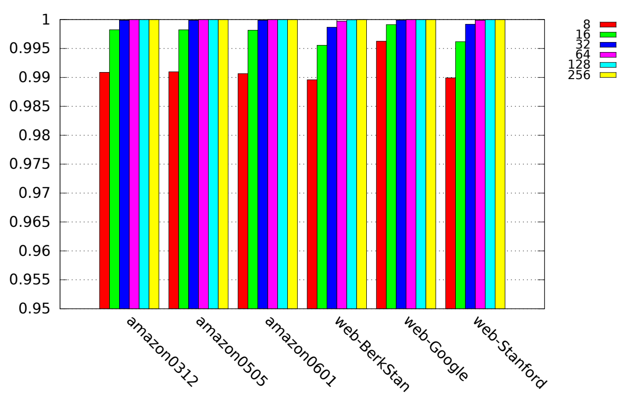

Figure 3: **NDCG score for Page Rank on G_m compared with G, for varying k.** Even for a small $k = 8$, NDCG score for all datasets is around 0.99, provided that their average degree is greater than 8.

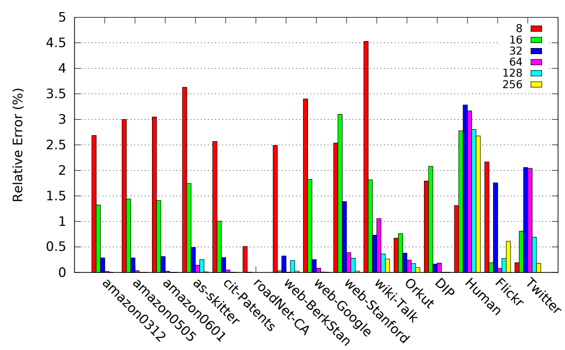

Figure 4: **Edge Estimator Performance:** The estimated value of number of edge converges to observed value of edges in G_m with increasing k.

KB L1, 1 MB L2, and 8 MB L3 cache and 16 GB memory. The datasets have been obtained from [6] and [10]. Bloom filter has been used to pre-process the input graph on-the-fly. The memory used by the sketch matrix representation is $[(2*k+n*k+n)*4]$ bytes, where the first term is for hash function parameters, the second term is for M_{k}, and the third term is for C, making memory footprint $O(n*k)$, significantly smaller than the $O(e)$ size of G.

Sketch construction is very fast as observed in Figure 2; even for the largest value of $k = 256$, on cit-Patents (using streaming edgelist format) processing speed is 205,980 edges/sec. For context, about 500 million tweets are generated per day on the Twitter social network, for an average of 5,800 tweets/sec, well within our processing capacity, especially considering that only 30% of tweets involve an edge-inducing user interaction (retweet or response) [7]. While G^* retains some percentage of edges proportional to k, G_m has a lot more edges recovered and graph properties like Page Rank are estimated very accurately.

4. FUTURE WORK

Using this sketch, we can generate multiple bootstrapped [5] variants(G_s) of the original graph (G) and estimate the properties of G from these samples. Parallel versions of our framework can be realized in multi-core CPU, GPU and MIC architectures for further scalability.

5. REFERENCES

[1] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. KDD '08.

[2] Tom Bohman, Colin Cooper, and Alan Frieze. Min-wise independent linear permutations. *Electronic Journal of Combinatorics*, 7:R26, 2000.

[3] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *JCSS*, 60:327–336, 1998.

[4] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 721–732. VLDB Endowment, 2005.

[5] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I. Jordan. The Big Data Bootstrap. In *ICML*, 2012.

[6] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[7] Replies and retweets on twitter. `http://sysomos.com/insidetwitter/engagement/`.

[8] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy. Efficient community detection in large networks using content and links. In *Proc. of eedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1089–1098, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[9] Yiye Ruan and Srinivasan Parthasarathy. Simultaneous detection of communities and roles from large networks. In *Proceedings of the Second ACM Conference on Online Social Networks*, COSN '14, pages 203–214, New York, NY, USA, 2014. ACM.

[10] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. SIGMOD '11.

[11] Jaroslaw Zola. Constructing similarity graphs from large-scale biological sequence collections. In *Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, IPDPSW '14, pages 500–507, Washington, DC, USA, 2014. IEEE Computer Society.