ProbGraph: High-Performance and High-Accuracy Graph Mining with Probabilistic Set Representations

Paper Presented by
Alex Mcneilly

Authors

Maciej Besta	ETH Zurich
Kacper Janda	University of Geneva
Grzegorz Kwaśniewski	Free University of Bozen-Bolzano
Onur Mutlu	University of Copenhagen
Torsten Hoefler	University of Illinois
Cesare Miglioli	AGH University of Science and Technology
Paolo Sylos Labini	Warsaw University of Technology
Jakub Tetek	TCL Research Europe (phones)
Patrick Iff	University of Trento
Raghavendra Kanakagiri	
Saleh Ashkboos	
Michał Podstawski	
Niels Gleinig	
Flavio Vella	

Why graph mining?

- Graph mining = discovering patterns/structures in graphs
- Used across social networks, ML, scientific computing
- Standard tasks: Triangle/Clique counting, Clustering, Link prediction, Vertex similarity
- A common primitive shows up everywhere: intersection of neighborhoods
- Motivation: hard to parallelize on real graphs
  - irregular
  - locality

$$|N_u \cap N_v|$$

$$TC = \frac{1}{3} \sum_{(u,v) \in E} |N_u \cap N_v|$$

$$Jaccard(u,v) = \frac{|N_u \cap N_v|}{|N_u \cup N_v|}$$

The bottleneck: set intersections

- Exact CSR intersections dominate compute time
  - TC: computing common neighborhoods is >90% of time
- Two exact methods on sorted neighbor lists:
  - Merge (sorted lists): good when sizes are similar
    - Walks both lists in lockstep
  - Galloping (binary search): good for big size mismatch
    - Probes each item from small list into big one
- Skew & load imbalance: degrees vary wildly → some intersections huge, some time, hard to balance work
- Work-depth view
  - Fixed-size operations beat degree-sized operations
  - Paper turns each intersection into a fixed-size operation, so every thread does the same amount of work

$$\text{Merge: } O(|N_u| + |N_v|)$$

$$\text{Galloping: } O(\min\{|N_u|, |N_v|\} \log \max\{|N_u|, |N_v|\})$$

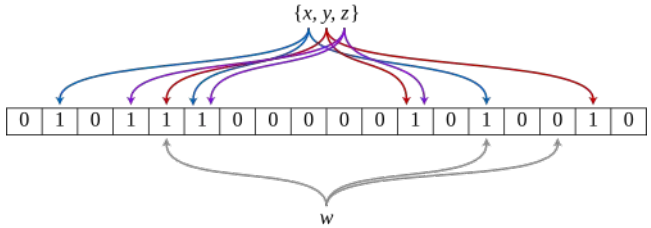
Probabilistic set representations

- Core idea: swap exact neighbor sets for small sketches of each neighborhood
- For any pair, it estimates the overlap from those sketches
- Two types of sketches
  - Bloom Filters (bit vectors)
  - MinHash (keep only the k smallest hashes)
- Benefits
  - Variable, degree-sized intersection → constant-sized, highly vectorizable computation
  - Accuracy vs. speed

$$|X \cap Y|$$

Bloom-filter estimators (BF)

- Bit vector + b hash functions
- False positives, no false negatives
- Intersection two neighborhoods by bitwise AND of the bit vectors → count the 1s
- Standard correction that accounts for hash collisions
- Vectorized bit operations on fixed-size array



$$|\widehat{X \cap Y}| \approx \frac{\text{popcount}(B_X \wedge B_Y)}{b}$$

$$(|\widehat{X \cap Y}| = -\frac{B}{b} \log\left(1 - \frac{B_1}{B}\right) \qquad B_1 = \text{ones after AND}$$

MinHash estimators (k-Hash & 1-Hash)

- MinHash shrinks each neighborhood to the k smallest hashed items from the set
- The fraction of the shared items between two sketches is an unbiased estimate of Jaccard similarity
- k-Hash (k independent hashes)
- 1-Hash (one hash, cheaper)
- Both concentrate tightly
  - k-Hash gives a maximum likelihood estimator (MLE)
    - Picks parameter value that makes the observed sketch most probable
    - As data grows, it achieves the smallest possible variance

$$\widehat{J} = \frac{|M_X \cap M_Y|}{k}$$

$$\widehat{|X \cap Y|} = \frac{\widehat{J}}{1+\widehat{J}}(|X| + |Y|)$$

Statistical guarantees + parallel work-depth

- Bloom filter estimators have polynomial concentration (errors drop in a controlled way as sketches grow)
- MinHash has exponential concentration (errors shrink exponentially)
- k-Hash is max likelihood estimator → consistent and best possible variance among comparable estimators
- All PG estimators are consistent and their biases approaches zero as the sample size becomes large
- Work/Depth analysis:
  - CSR merge: $\text{Work } O(d_u + d_v); \text{Depth } O(log(d_u + d_v))$
  - CSR galloping: $\text{Work } O(d_u \cdot logd_v); \text{Depth } O(log(d_u + d_v))$
  - Bloom filter (bitwise AND + popcount): $\text{Work } O(B/W); \text{Depth } O(log(B/W))$
  - MinHash (k hashes): $\text{Work } O(k); \text{Depth } O(logk)$
  - Fixed size → vectorizes well and load-balances

Triangle Counting with ProbGraph

- Classic TC = sum of common neighbors over edges / 3
- PG → algorithm doesn't change, just swap the inner operation with an estimator
- Same loops, same parallel structure, better work/depth
- MinHash-based TC gets exponential deviation bounds
- k-Hash TC is also a max likelihood estimator

$$TC = (1/3)\Sigma_{(u,v)\in E}|N_u \cap N_v|$$

$$\hat{TC} = (1/3)\Sigma|\widehat{N_u \cap N_v}|$$

```
1 /* Input: A graph G. Output: Triangle count tc ∈ ℕ. */
2 //Derive a vertex order R s.t. if R(v) < R(u) then d_v ≤ d_u:
3 for v ∈ V [in par] do: N_v^+ = {u ∈ N_v | R(v) < R(u)}
4 tc = 0; //Init tc; for all neighbor pairs, increase tc:
5 //Now, derive the actual count of triangles:
6 v ∈ V [in par] do:
7   for u ∈ N_v^+ [in par] do: tc += | N_v^+ ∩ N_u^+ |
```

Listing 1: Triangle Counting (Node Iterator).

4-Clique and Clustering with ProbGraph

- 4-Clique: find 3-cliques, then intersect again to close 4-cliques, inner steps are still intersections
- Swap intersections with PG estimates
- Vertex similarity is the basic building block of Clustering/Link Prediction
- For clustering (Jarvis-Patrick), the decision is based on similarity scores like Jaccard which we can estimate from our PG sketches
- Fixed-size intersections → vectorizable intersections, even when degrees are wildly uneven

```
1 /* Input: A graph G. Output: Number of 4-cliques ck ∈ ℕ. */
2 /Derive a vertex order R s.t. if R(v) < R(u) then d_v ≤ d_u:
3 for v ∈ V [in par] do: N_v^+ = {u ∈ N_v | R(v) < R(u)}
4 ck = 0;
5 for u ∈ V [in par] do:
6   for v ∈ N_u^+ [in par] do:
7     C_3 = N_u^+ ∩ N_v^+ //Find 3-cliques
8     for w ∈ C_3 do: //For each 3-clique...
9       ck += |N_w^+ ∩ C_3| //Find 4-cliques
```

Listing 2: Reformulated 4-Clique Counting.

```
1 /* Input: A graph G. Output: Similarity S ∈ ℝ of sets A, B.
2 * Most often, A and B are neighborhoods N_u and N_v
3 * of vertices u and v. */
4 //Jaccard similarity:
5 S_J(A,B) = |A ∩ B| / |A ∪ B| = |A ∩ B| / (|A| + |B| - |A ∩ B|)
6 //Overlap similarity:
7 S_O(A,B) = |A ∩ B| / min(|A|, |B|)
8 //Certain measures are only defined for neighborhoods:
9 S_A(v,u) = ∑_w (1/ log|N_w|) //where w ∈ N_v ∩ N_u ; Adamic Adar
10 S_R(v,u) = ∑_w (1/|N_w|) //where w ∈ N_v ∩ N_u ; Resource Alloc.
11 S_C(v,u) = |N_v ∩ N_u| //Common Neighbors
12 S_T(v,u) = |N_v ∪ N_u| = |N_v| + |N_u| - |N_v ∩ N_u| //Total Neighbors
```

Listing 3: Example vertex similarity measures [71].

```
1 /* Input: A graph G = (V, E). Output: Clustering C ⊆ E
2  * of a given prediction scheme. */
3 //Use a similarity S_C(v,u) = |N_v ∩ N_u| (see Listing 3).
4 for e = (v,u) ∈ E [in par] do: //τ is a user-defined threshold
5   if |N_v ∩ N_u| > τ: C ∪= {e}
6 //Other clustering schemes use other similarity measures.
```

Listing 4: Jarvis-Patrick clustering.

Implementation

- We keep the graph input in CSR
- For each vertex, build a sketch:
  - BF: bit-vector plus b hashes
  - MinHash: k-smallest hashed neighbors
- Core loop: replace exact intersections with either bitwise AND + popcount (BF) or k-sized intersections (MH)
- Parallelize the outer loops with OpenMP
- BF ops are SIMD-friendly and use popcnt
- MinHash stores small integer arrays
- Choose a storage budget s (up to ~33% extra) and translate

```
1 //Input: Graph G, two vertices u and v
2 //Create a standard CSR graph with G as the input graph
3 CSRGraph g = CSRGraph(G);
4 //Create a ProbGraph representation of G based on Bloom filters
5 ProbGraph pg = ProbGraph(g, BF, 0.25); //Use the 25% storage budget
6
7 //Derive the exact intersection cardinality |N_u ∩ N_v|
8 int interEX = pg.int_card(g.N(u), g.N(v));
9 //Derive the estimator |N_u ∩ N_v|_AND
10 int interBF = pg.int_BF_AND(pg.N(u), pg.N(v));
11
12 //Compute the exact Jaccard coefficient between u and v
13 double jacEX = interEX / (g.N(u).size() + g.N(v).size() - interEX)
14 //Compute the approximate Jaccard coefficient based on BF
15 double jacBF = interBF / (g.N(u).size() + g.N(v).size() - interPG)
```

Listing 5: Obtaining exact and approximate Jaccard (see Listing 3)

Empirical results

- Up to ~50x on 32 cores while staying above 90% accuracy
- Not uniform, variance exists
- Figure 3's boxplots for $|X \cap Y|$ show low medians with visible outliers, no single winner
- BF tends to be more accurate on many cases; MH often faster/smaller but can undercount clusters
- Denser graphs can hurt BF-AND, while MH can over-prune edges, inflating cluster counts

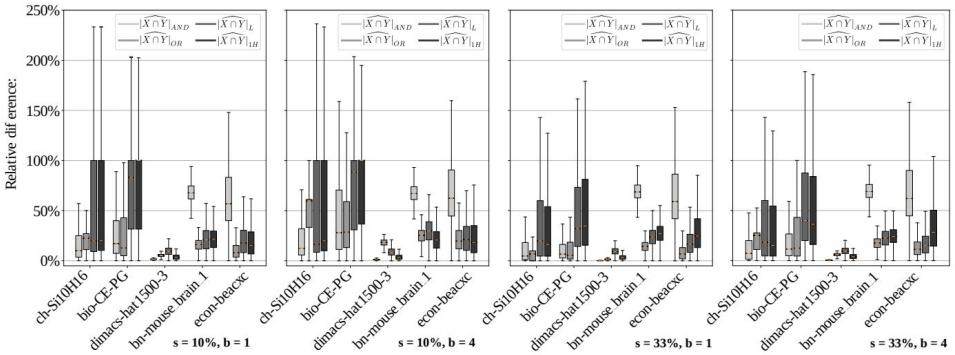


Fig. 3: Analysis of the accuracy of PG estimators of $|X \cap Y|$.

Empirical results

- End-to-end plots for 4-cliques show points near a relative count of 1.0 with large speedups, but not every point
- Big performance that we can tune for

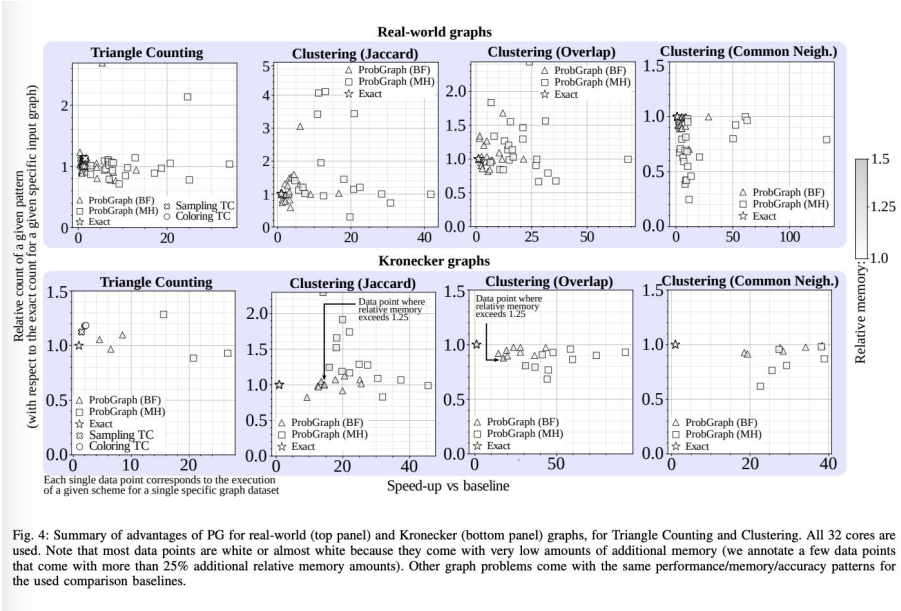


Fig. 4: Summary of advantages of PG for real-world (top panel) and Kronecker (bottom panel) graphs, for Triangle Counting and Clustering. All 32 cores are used. Note that most data points are white or almost white because they come with very low amounts of additional memory (we annotate a few data points that come with more than 25% additional relative memory amounts). Other graph problems come with the same performance/memory/accuracy patterns for the used comparison baselines.

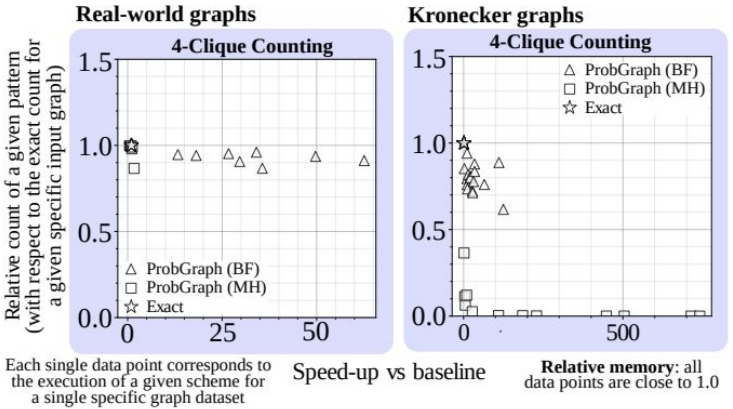


Fig. 5: Summary of advantages of PG for 4-clique counting for real-world (left panel) and Kronecker (right panel) graphs. All 32 cores are used.

Related work

- Prior work either samples edges (e.g., Doulion), performs clever combinatorial pruning (“colorful”), or builds task-specific sketches/sketching frameworks, ASAP/GAP, SlimGraph
- PG’s contribution: general drop-in set-intersection primitive with strong bounds across tasks
- MinHash TC = exponential bounds + max likelihood estimator (only one)
- TC comparison table:

	Reference	Constr. time	Memory used	Estimation time	AU	CN	ML	IN	AE	B
Past results	Doulion [46]	$O(m)$	$O(pm)$	$O(T(pm))$	👍	👍	✖	✖	✖	✖
	Colorful [47]	$O(m)$	$O(pm)$	$O(T(pm))$	👍	👍	✖	✖	✖	👍 (P)
	Sketching [48]	$O(km)$	$O(kn)$	$O(T(k^2n))$	👍	👍	✖	✖	✖	✖
	ASAP [49]	n/a	$O(n+m)$	$O(1)$ / sample	✖	✖	✖	✖	✖	✖
	GAP [50]	$O(m)$†	$O(m')$†	$O(T(m'))$†	✖	✖	✖	✖	✖	✖
	Slim Gr. [51]	$O(m)$	$O(pm)$	$O(T(pm))$	👍	👍	✖	✖	✖	✖
	Eden et al. [52]	n/a	$O\left(\frac{n}{TC^{1/3}}\right)$	$O\left(\frac{n}{TC^{1/3}}+\frac{m^{3/2}}{TC}\right)$	👍	👍	✖	✖	✖	👍
	Assadi et al. [53]	n/a	$O(1)$	$O(m^{3/2}/TC)$	👍	👍	✖	✖	✖	👍
	Tětek [54]	n/a	$\left(\frac{m^{1.41}}{TC^{0.82}}\right)$	$\left(\frac{m^{1.41}}{TC^{0.82}}\right)$	👍	👍	✖	✖	✖	👍
PG	$\widehat{TC}_{AND}$ (BF)	$O(bm)$	$O(n+m)$	$O(mB/W)$	👍	👍	✖	✖	✖	👍 (P)
	$\widehat{TC}_{kH}$ (MH)	$O(km)$	$O(n+m)$	$O(km)$	👍	👍	👍	👍	👍	👍 (E)
	$\widehat{TC}_{1H}$ (MH)	$O(km)$	$O(n+m)$	$O(km)$	👍	👍	✖	✖	✖	👍 (E)

AU = asymptotically unbiased
CN = consistent
ML = maximum likelihood estimator
IN = invariance
AE = asymptotically efficient
B = scheme is supported with concentration bounds
P = polynomial; E = exponential

Strengths & weaknesses

- Strengths
  - General, swappable primitive with theoretical guarantees and real speed
  - Simple integration
  - Clean math and straightforward (concentration, MLE, SIMD, load balance)
- Weaknesses
  - Accuracy is graph-dependent
  - BF-AND can degrade on very dense graphs
  - MinHash needs careful k-tuning and can amplify cluster counts by dropping edges
  - The accuracy that they state in the abstract (>90% accuracy) is not necessary guaranteed or universal, it is very dataset- and parameter dependent
  - Evaluated on CPUs; GPUs could change the trade-offs

Questions/discussion

- Where do BF false positives help vs hurt when performing routines?
- Can we adapt the sketch size per-edge or per-degree to improve accuracy?
- What if we tried a hybrid approach using BF and then MinHash to refine dense parts of the graph?
- Is there some way to auto-tune k or even pick which estimator is best to apply to a graph?
- How can the principles from this paper be applied to Iterative Closest Point?