Fast and Space-Efficient Parallel Algorithms for Influence Maximization

UC RIVERSIDE

Full Version

Letong Wang Xiangyun Ding

Yan Gu Yihan Sun

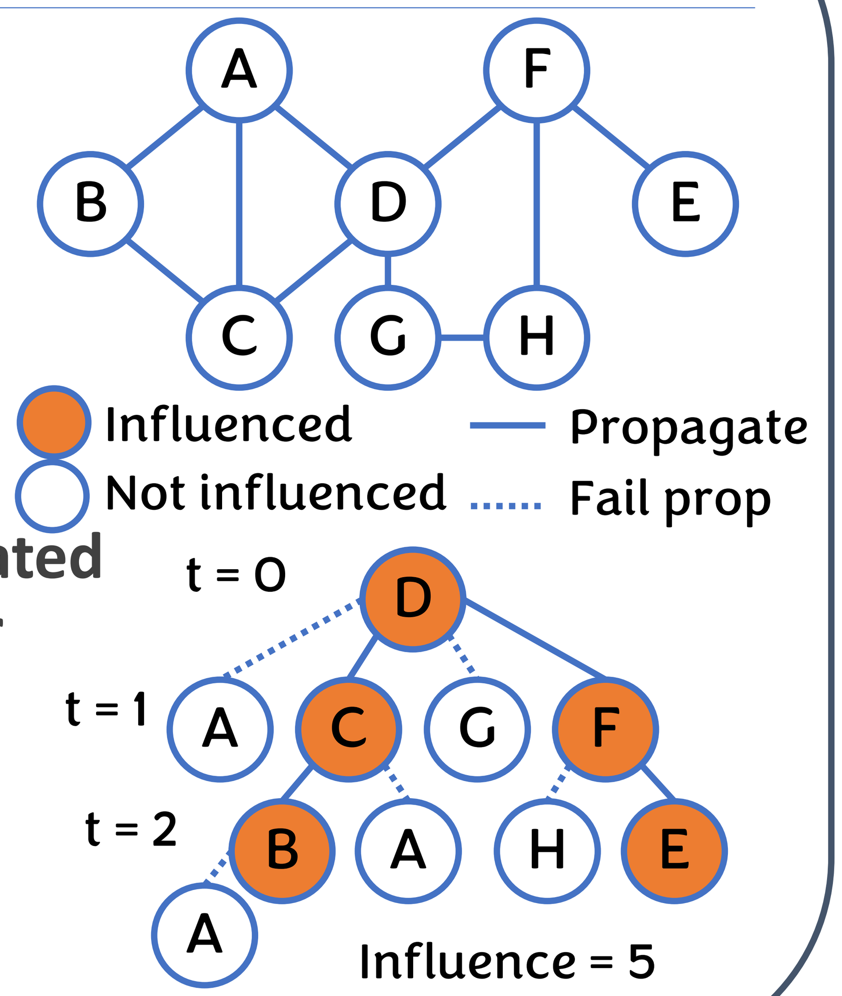
Code

VLDB2024 GUANGZHOU

Artifacts Available acm V1.1

Problem Definitions

- **Influence propagation: a node in a network may propagate its influence to its neighbors (in a cascading way)… The "word-of-mouth" effect**
- **Influence propagation model (Independent Cascading, IC): In each timestamp, a newly-activated vertex will activate its neighbor with probability p**
- **Influence maximization (IM) [KKT'03]: find k "seeds" such that the expectation of their total influence is maximized**



General Greedy Algorithm

- **IM is NP-hard under the IC model [4]**
- **If the objective is also submodular and monotone: a simple greedy algorithm is (1-1/e)-approximation [4]**

A Greedy Algorithm:

- Start from empty seed set S
- Repeatedly adding the vertex with the highest **marginal gain** to S.

```
marginal(S, v) = influence(S ∪ {v}) - influence(S)
S = { }
for i = 1 to k {
   v* = arg max_{v∈V} marginal(S,v)
   S = S ∪ {v}
}
```

How to compute the influence $\sigma(S)$?

- **Simulation [4,5]: simulate the influence propagation for R rounds and use the average**
  - **Very expensive in time**

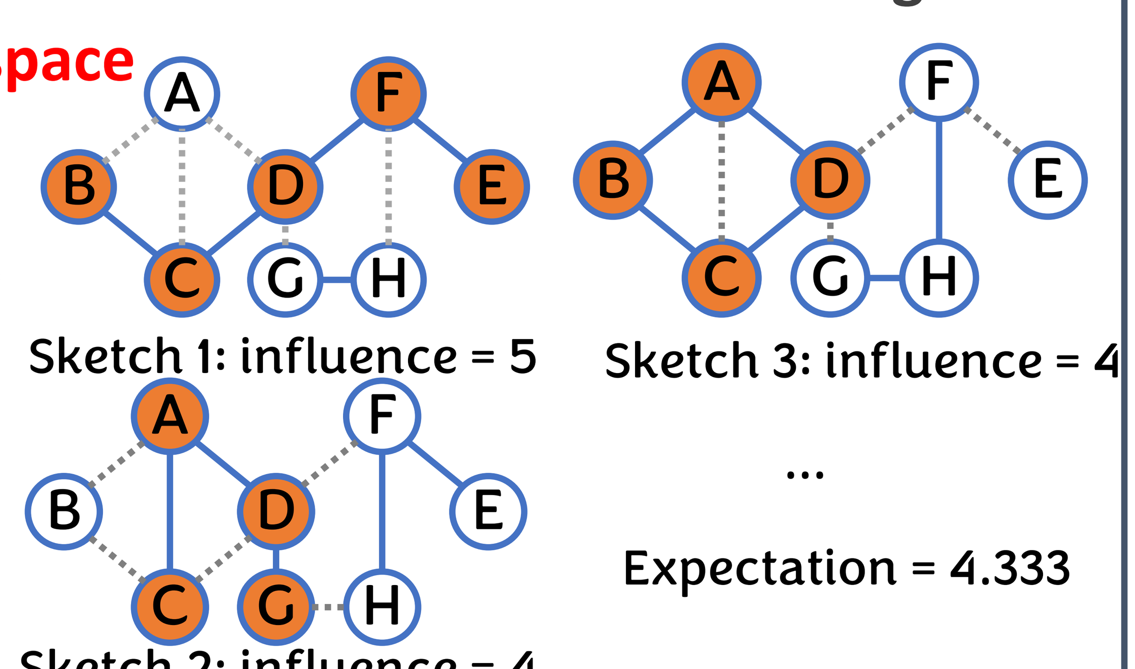
Influence({D}) = ?

Simulation 1: influence = 5
Simulation 2: influence = 4
Simulation 3: influence = 4
…
Expectation = 4.333

- **Memoization [2]: memorizing the simulation results of the influence propagation for R rounds and use the average**
  - **Very expensive in space**

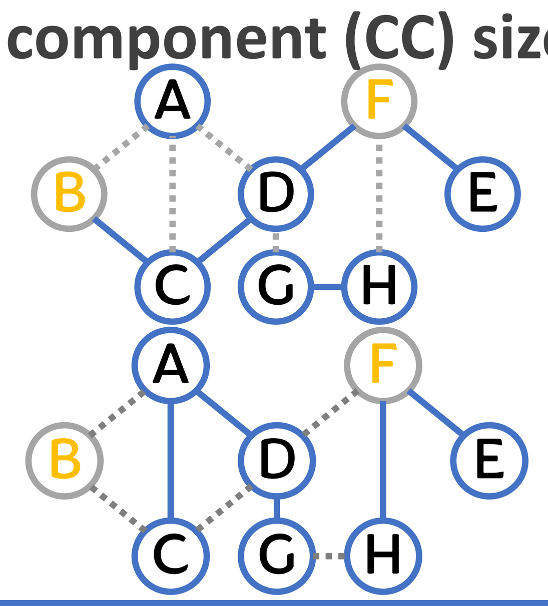
A sketch is a subgraph:

- where edges represent successful propagations
- The connected component (CC) size is the influence on that experiment



Our solution: compressible sketch

- **Pre-select αn centers and only memorize connectivity component (CC) sizes for the centers!**



Sketch 1: influence = CC(F) = 5

Case 1: encounter a center during BFS. Use the CC size of the center

- **If CC(v) is large, it's likely to be connected to a center, and the cost is cheap!**

Sketch 2: influence = BFS(D) = 4

Case 2: No center encountered. Use #vertices visited in BFS

- **If CC(v) is small, even finish the BFS is cheap!**

	Ours with αn centers	Simulation [4,5]	Memoization [2]
space	$O((1+\alpha)n)$	$O(n)$	$O(Rn)$
work	$P(R \cdot \min(\frac{1}{\alpha}, T))$	$O(RT)$	$O(R)$
notes	T is the avg. influence of v	$\alpha = 0$	$\alpha = 1$

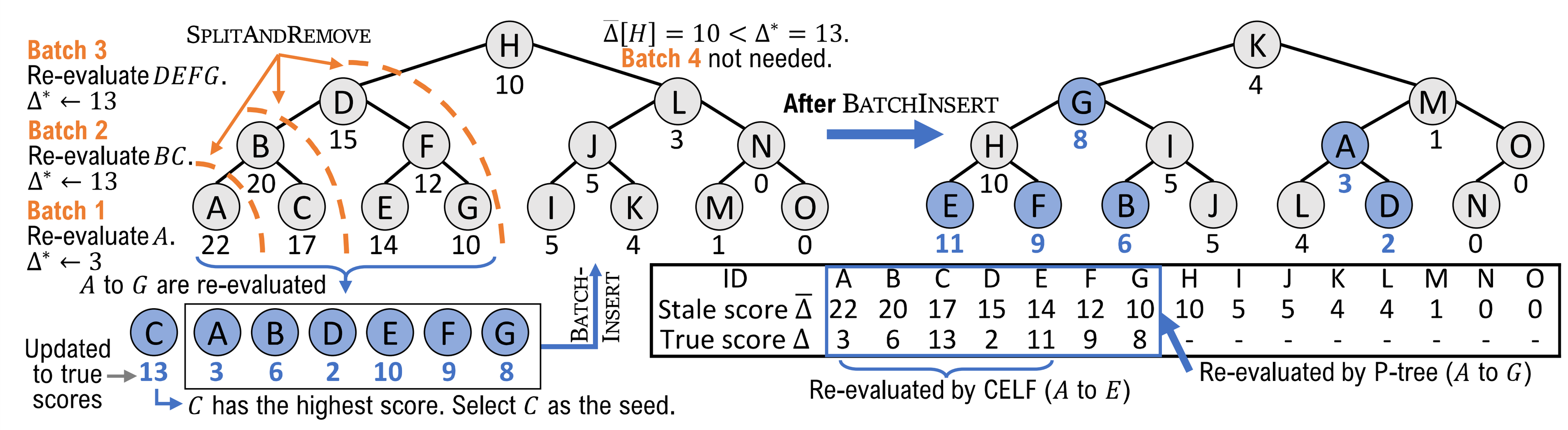
How to efficiently select seeds?

- **The Greedy algorithm itself is still expensive!**
  - **Selecting 1 seed needs to re-evaluate n vertices**
  - **$\Omega(k \cdot Rn)$ cost even not considering compression**
- **The CELF [5] Optimization re-evaluates much fewer vertices**
  - **For the same vertex v, marginal(v, S) is non-increasing as S grows larger**
  - **CELF is iterative and hard to be parallelized**

A Greedy Algorithm based on CELF: Lazily update scores

```
v* = Q.pop();                        ← Best (stale) score
score[v*] = marginal(v*, S)          ← Re-evaluate the true score
if v* is better than Q.top then {
  S = S + {v*}; break;               ← If still the top, select & go to next
} else insert v* with back to Q;     ← Otherwise, insert it back & repeat
```

Our solutions: P-tree or Win-tree based CELF

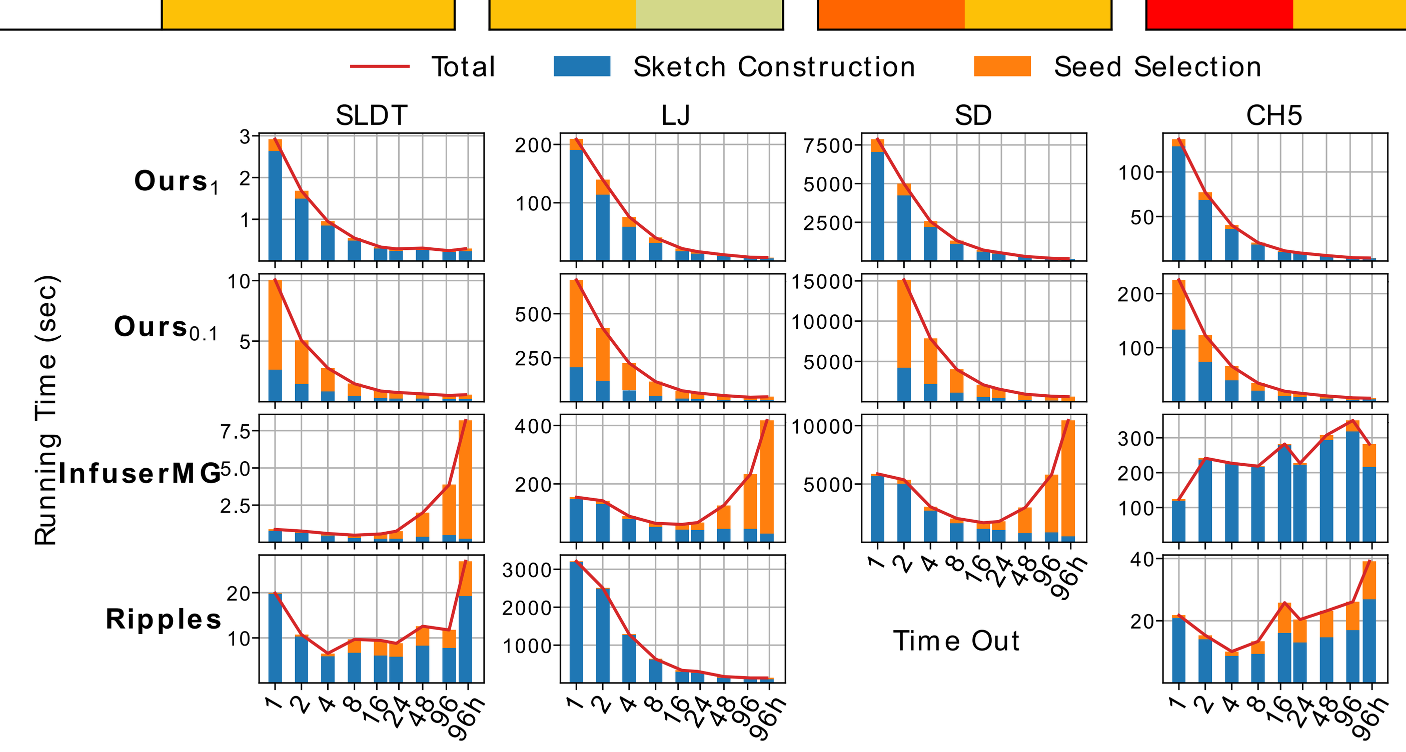


- **Solution based on Parallel Trees (P-trees)[1]**
  - **P-tree:** binary search trees support:
    - `split, batch_insert, construction`
  - **"prefix-doubling":** evaluate batches of size 1, 2, 4, 8, … until the best new score is better than all the stale scores!
  - **Theorem:** The total number of evaluations for P-tree is at most twice as that of CELF
- **(Another) Solution based on Winning Trees**
  - Slightly better in practice, but no worst-case guarantee

Experimental Results

- **Setup**
  - **Machine:** 96 cores, 1.5TB memory
  - **Baselines:** Ours_1=ours with no compression, $\text{Ours}_{0.1}$=ours with $\alpha = 0.1$, InfuserMG [3] and Ripples [6] are parallel baseline systems

	Ours_1 Time	Ours_1 Space	$\text{Ours}_{0.1}$ Time	$\text{Ours}_{0.1}$ Space	InfuserMG Time	InfuserMG Space	Ripples Time	Ripples Space
SLDT	1.00	1.00	1.76	0.38	1.60	1.17	22.8	1.65
LJ	1.00	1.00	3.42	0.27	10.2	1.76	21.7	3.41
SD	1.00	1.00	4.18	0.36	11.2	1.69	-	-
CH5	1.00	1.00	1.51	0.22	62.2	1.80	2.91	0.34
MEAN	1.00	1.00	1.76	0.26	5.66	1.56	18.3	1.34



References

[1] Guy Blelloch, Daniel Ferizovic, and Yihan Sun. 2022. Joinable Parallel Balanced Binary Trees. ACM Transactions on Parallel Computing (TOPC) 9, 2 (2022), 1–41.

[2] Suqi Cheng, Huawei Shen, Junming Huang, Guoqing Zhang, and Xueqi Cheng.2013. Staticgreedy: solving the scalability-accuracy dilemma in influence maximization. In Proceedings of the 22nd ACM international conference on Information& Knowledge Management. 509–518.

[3] Gökhan Göktürk and Kamer Kaya. 2020. Boosting parallel influence-maximization kernels for undirected networks with fusing and vectorization. IEEE Transactions on Parallel and Distributed Systems 32, 5 (2020), 1001–1013.

[4] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD). 137–146.

[5] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne Van Briesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. 420–429.

[6] Marco Minutoli, Mahantesh Halappanavar, Ananth Kalyanaraman, Arun Sathanur, Ryan Mcclure, and Jason McDermott. 2019. Fast and scalable implementations of influence maximization algorithms. In 2019 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 1–12.