Rensselaer

Towards Efficient Methods for Approximate Weighted Matching Applications

Michael Mandulak, Sayan Ghosh, S M Ferdous, Mahantesh Halappanavar, George M. Slota

Rensselaer Polytechnic Institute, Pacific Northwest National Laboratory

2025 SIAM Applied and Computational Discrete Algorithms: Parallel Algorithms

August 1, 2025

Overview

- Maximum Weight Matching Background
- Prior Works in Parallel Matching: Timeline
- Limitations of Parallel Matching + Case Studies
- Parallel Direction: 2D Matching
- Application Direction: Fuzzy Set Similarity Join

Maximum Weight Matching Problem

Matching: A matching *M* is a subset of edges such that no two edges in *M* are incident on the same vertex

- Common use cases:
  - Load balancing
  - Sparse Matrix Computations
- Applications:
  - Sparse linear solvers
  - Network switching
  - Graph partitioners
  - Coarsening
- **Problem:** Optimal matching runtimes

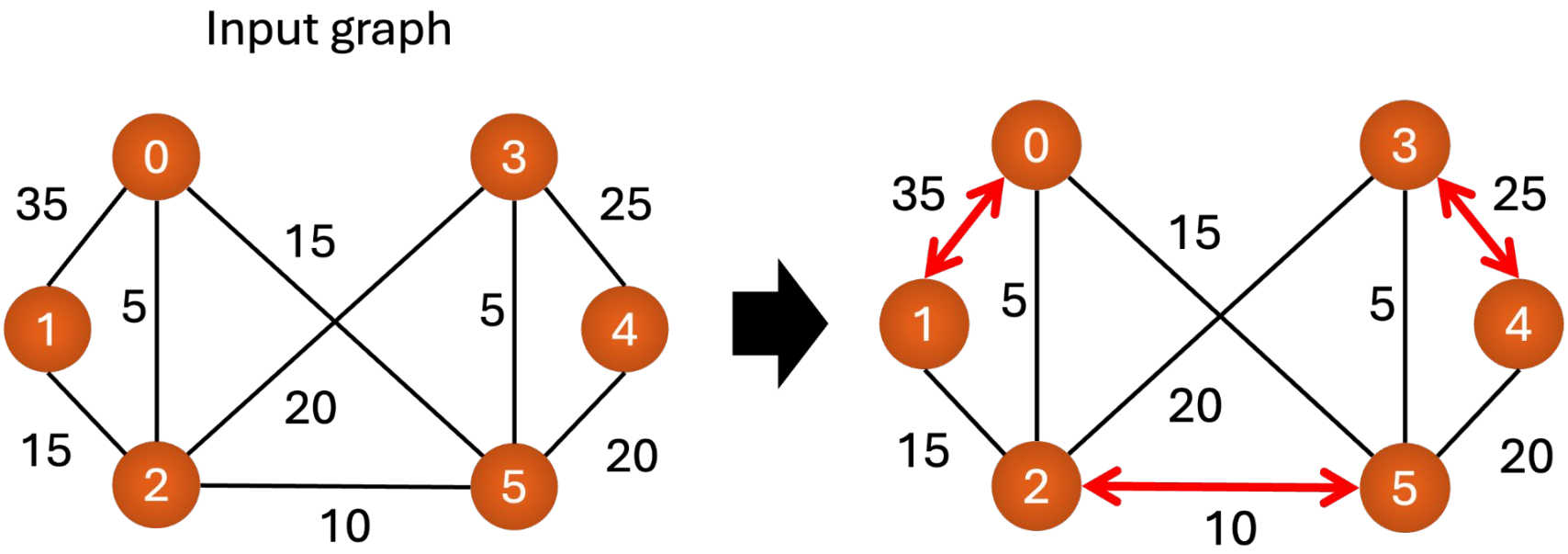
Our focus:

Approximation methods for maximum weighted matching!

- Modern implementations are very fast!
- Performance gains in application?

Target: Apply approx. algorithms with provable bounds within problem domains

Maximum Weight Matching Problem (MWM)

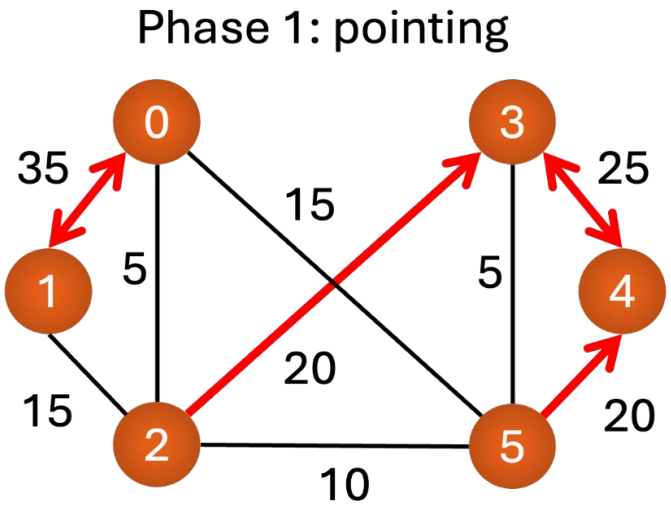


Find a matching set of edges to maximize the total weight

Traditional Half-approx Methods

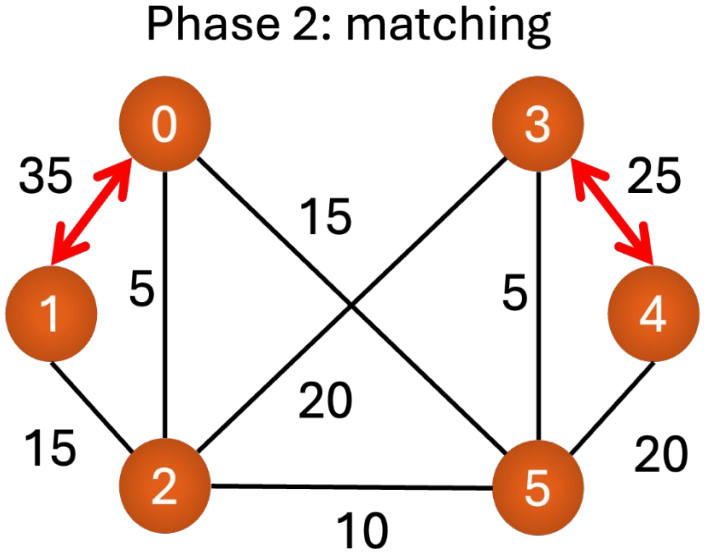
Suitor: proposal + consideration-based
Locally-Dominant (LD): choose best neighbor → commit mutuals

Locally-Dominant (LD) MWM Method



Vertex-independent choice of highest weight available neighbor

Tentative selection of matching partner



Commit mutually pointing matching pairings

Reset pointers for singleton vertices and repeat

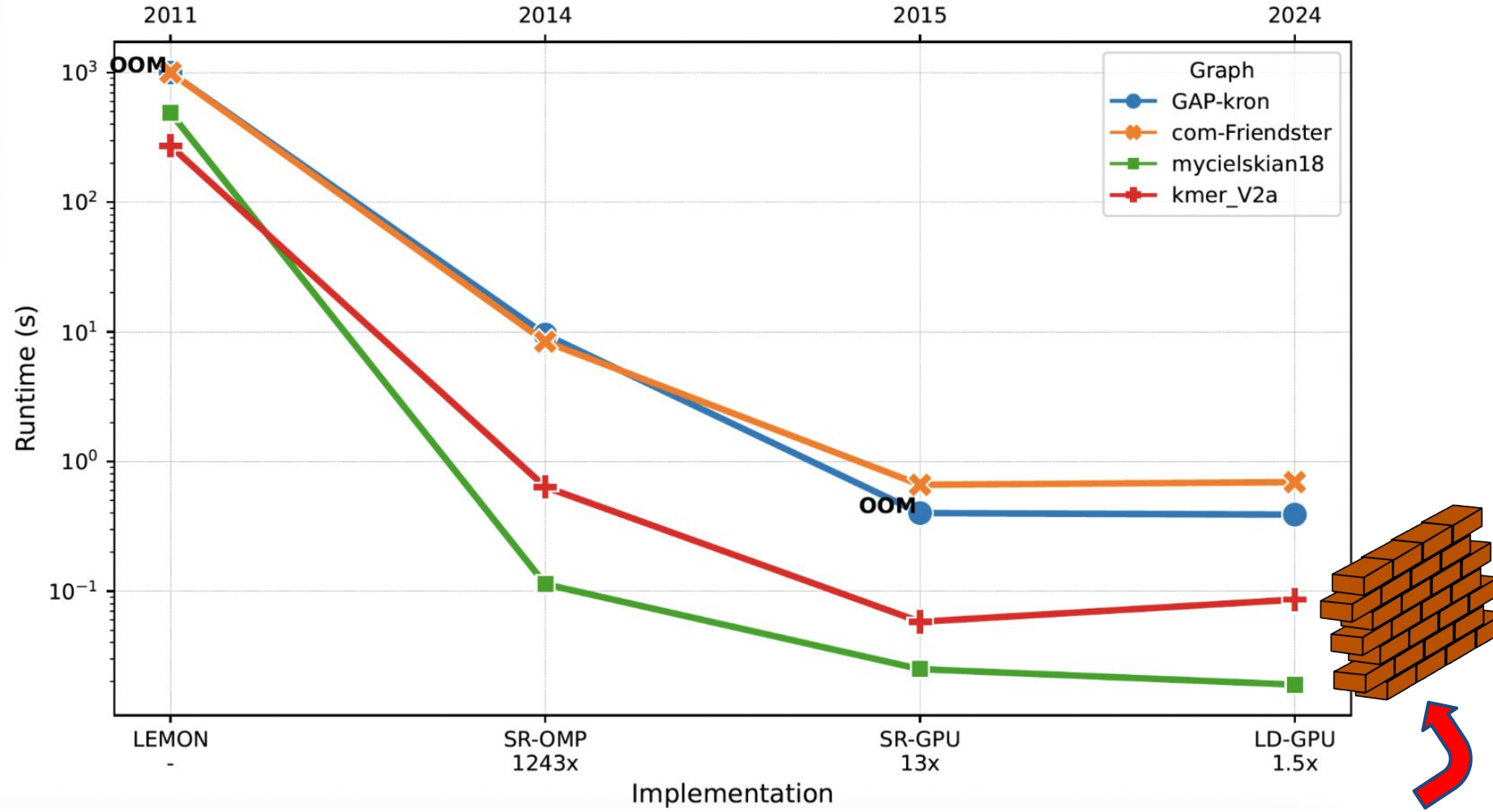
Matching Implementations Timeline

- **LEMON**: Sequential optimal - Blossom **(2011)**
- **SR-OMP**: OpenMP Suitor **(2014)**
- **SR-GPU**: Single GPU Suitor **(2015)**
- **LD-GPU**: Multi-GPU Locally-Dominant **(2024)**

Method	Algorithm	Parallel Mode	Approximation Ratio
LEMON	Blossom	Serial	Max. Weight
SR-OMP	Suitor	OpenMP	1/2-approx.
SR-GPU	Suitor	Single GPU	1/2-approx.
cuGraph	Locally-Dominant	Multi-GPU	1/2-approx.
LD-GPU	Locally-Dominant	Multi-GPU	1/2-approx.

1. https://lemon.cs.elte.hu/pub/doc/latest-svn/index.html
2. F. Manne and M. Halappanavar, "New Effective Multithreaded Matching Algorithms," *2014 IPDPS*
3. M. Naim, F. Manne, M. Halappanavar, A. Tumeo and J. Langguth, "Optimizing Approximate Weighted Matching on Nvidia Kepler K40," 2015 HiPC
4. M. Mandulak, S. Ghosh, S M Ferdous, M. Halappanavar, and G. Slota, “Efficient Weighted Graph Matching on GPUs,” SC '24

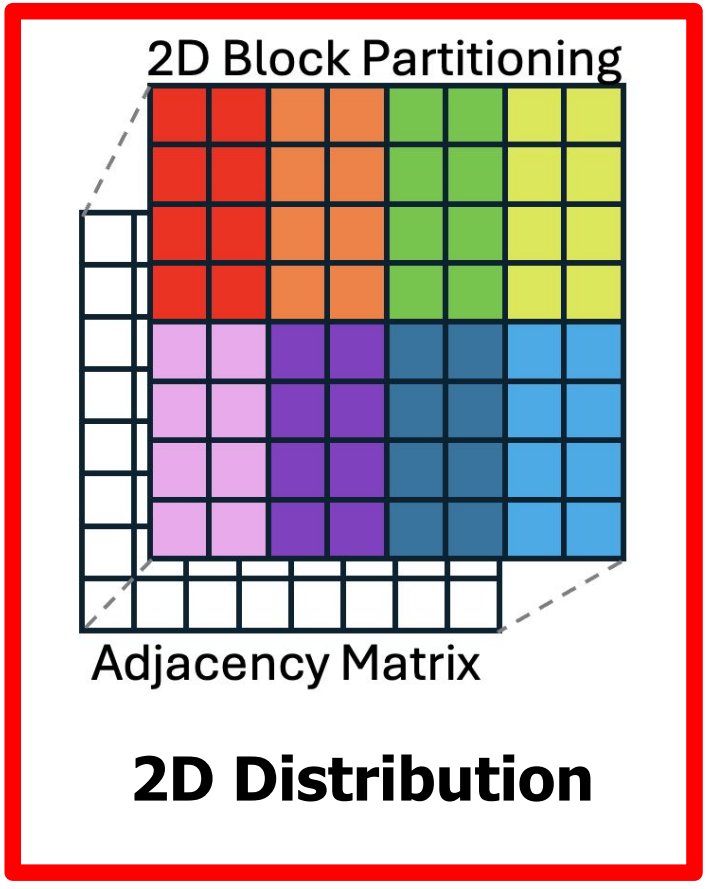
Limitations

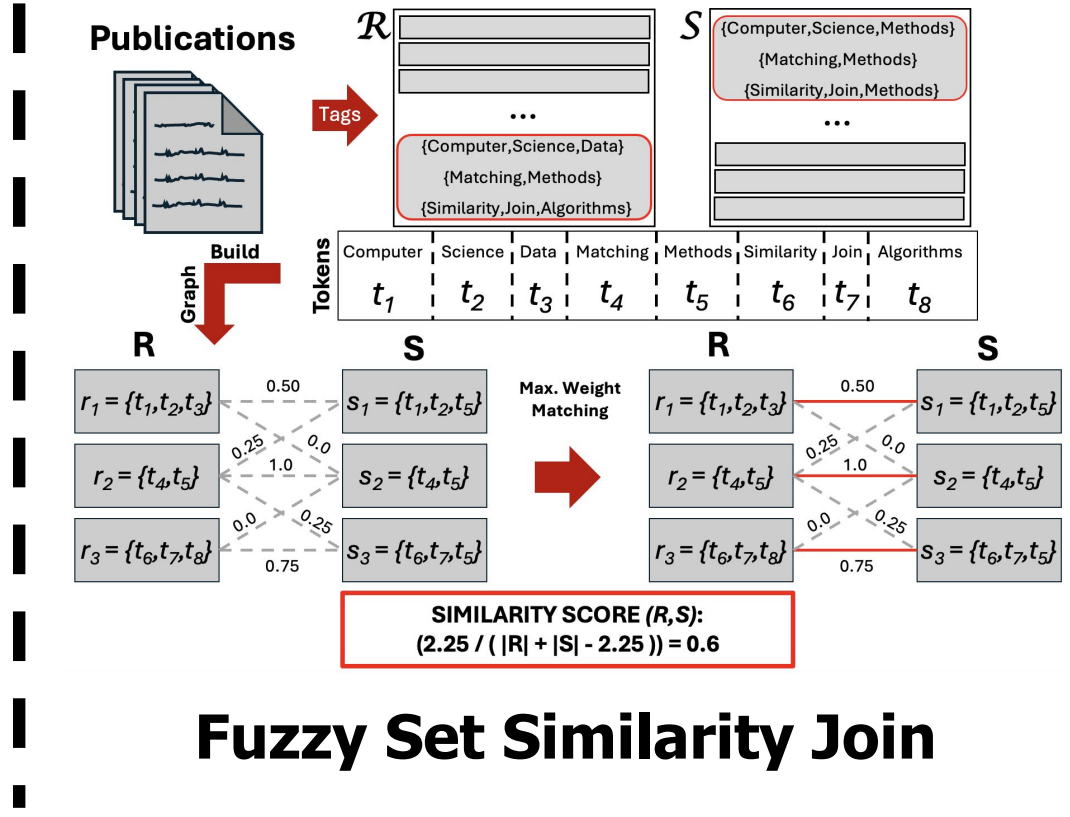


Can we explore alternative algorithm/implementation optimizations?

"Performance Wall" for traditional half-approx methods made increasingly parallel

Algorithm Design to Application: Two Cases



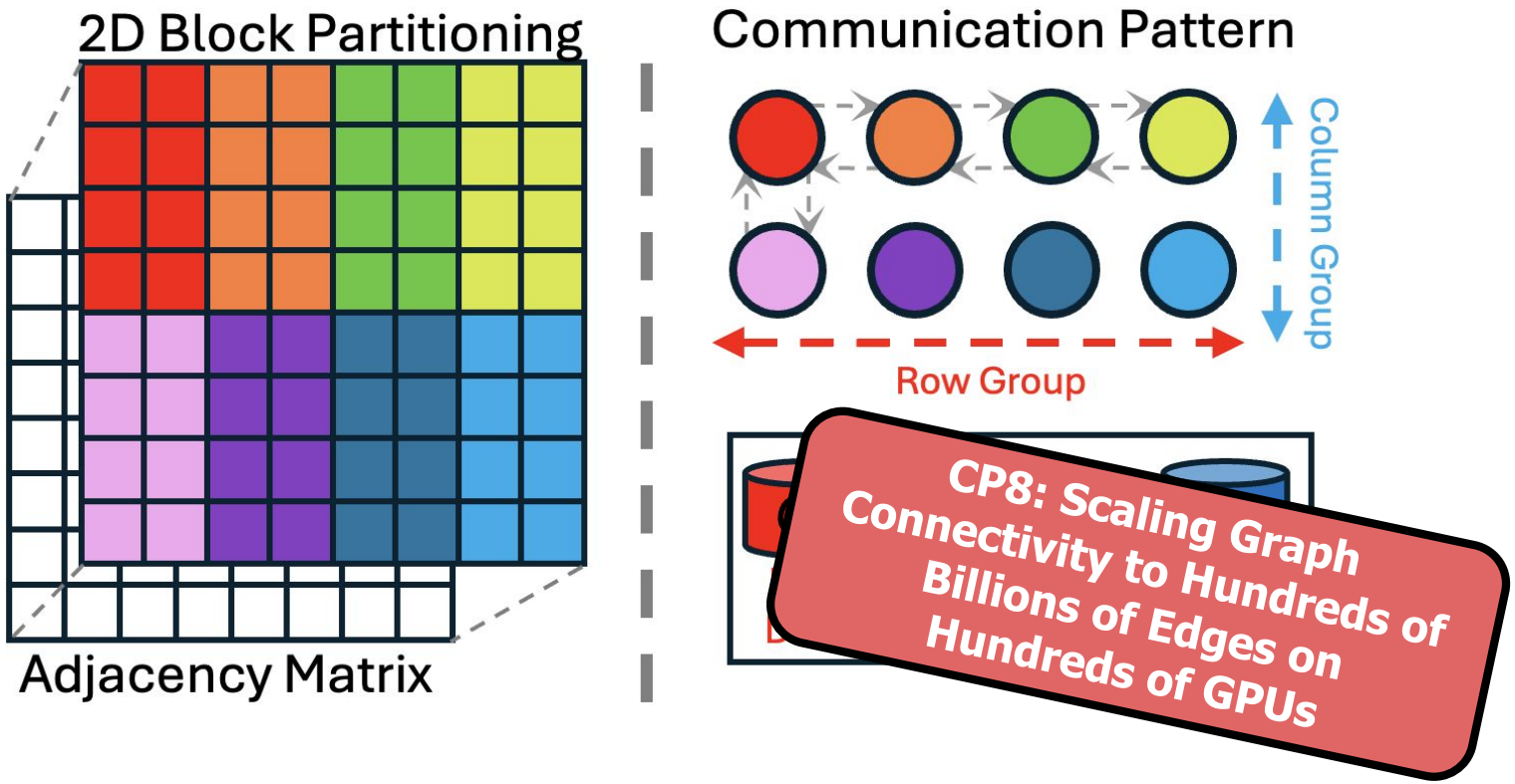


2D Distribution

Idea: Target bottlenecks of previous implementations

LD-GPU bottleneck: Communication costs!

- Synchronized pointers and matching each round



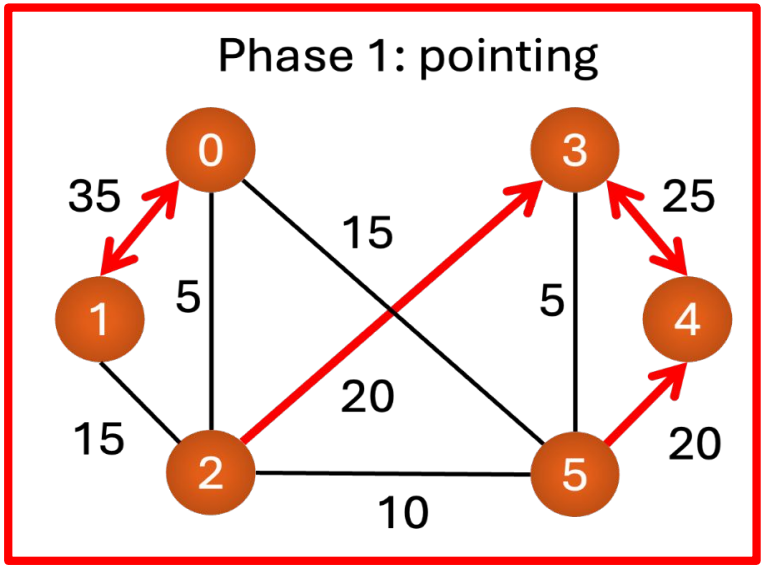
Original Work: Erik G. Boman, Karen D. Devine, and Sivasankaran Rajamanickam. Scalable matrix computations on large scale-free graphs using 2d graph partitioning. SC'13

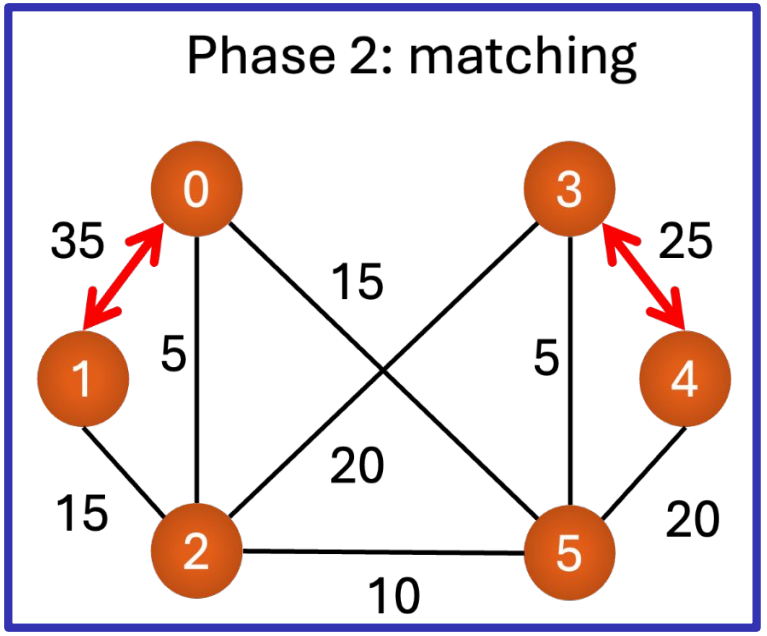
LD Sparse - Algorithm

Algorithm 1 2D LD Matching

Input: Graph: $G(V, E)$
Output: Matching M

1: $Q \leftarrow$ `InitQueue`(V)
2: **while** $\exists$ matching edges **do**
3: $ptrs \leftarrow$ `LocalMC`(Q, q_{in})
4: $sbuf \leftarrow$ `BuildQueue`$(G, q_{in}, ptrs)$
5: $rbuf \leftarrow$`Allgatherv`$(sbuf$,`COL_COMM`$)$
6: $Q \leftarrow$ `ReduceQueue`$(G, rbuf)$
7: $ptrs \leftarrow$ `UpdatePtrs`$(G, rbuf)$
8: $sbuf \leftarrow$ `BuildQueue`$(G, q_{in}, ptrs)$
9: $rbuf \leftarrow$`Broadcast`$(sbuf$,`ROW_COMM`$)$
10: $M \leftarrow$ `MutualCheck`$(ptrs)$
11: $Q \leftarrow$ `ReduceQueue`$(G, rbuf)$





LD Sparse - 2D Communication Framework

Algorithm 1 2D LD Matching

Input: Graph: $G(V, E)$
Output: Matching M

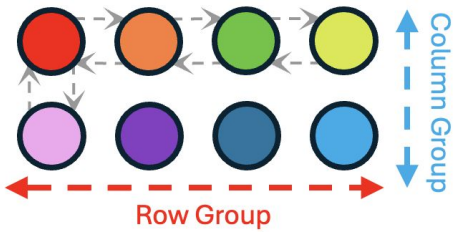
1: $Q \leftarrow$ `InitQueue`(V)
2: **while** $\exists$ matching edges **do**
3: $ptrs \leftarrow$ `LocalMC`(Q, q_{in})
4: $sbuf \leftarrow$ `BuildQueue`$(G, q_{in}, ptrs)$
5: $rbuf \leftarrow$`Allgatherv`$(sbuf$,`COL_COMM`$)$
6: $Q \leftarrow$ `ReduceQueue`$(G, rbuf)$
7: $ptrs \leftarrow$ `UpdatePtrs`$(G, rbuf)$
8: $sbuf \leftarrow$ `BuildQueue`$(G, q_{in}, ptrs)$
9: $rbuf \leftarrow$`Broadcast`$(sbuf$,`ROW_COMM`$)$
10: $M \leftarrow$ `MutualCheck`$(ptrs)$
11: $Q \leftarrow$ `ReduceQueue`$(G, rbuf)$

Communication Pattern:

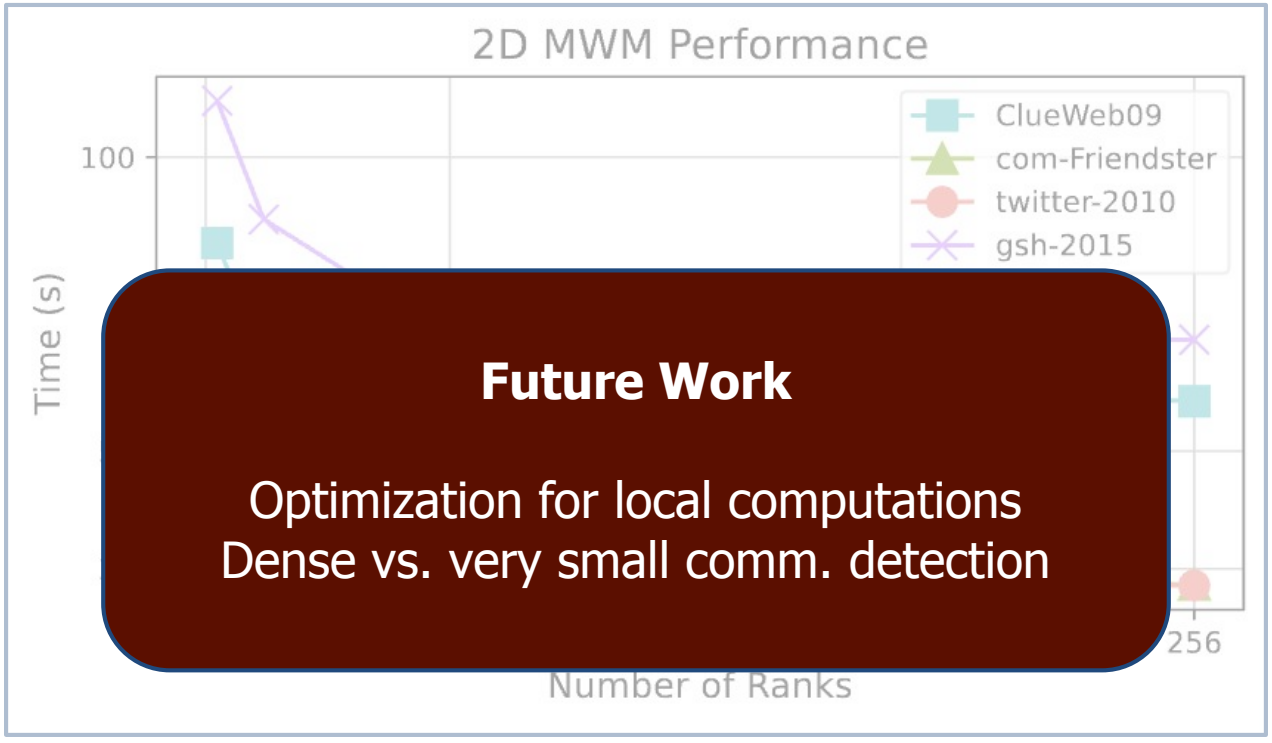
1. Computation
2. Compile relevant data (active vertices among group - sparse)
3. Communicate among row/column group
4. Reduce relevant data to global data queue

Can repeat this block per row/column or as needed

Communication Pattern



LD Results - Scaling



Future Work

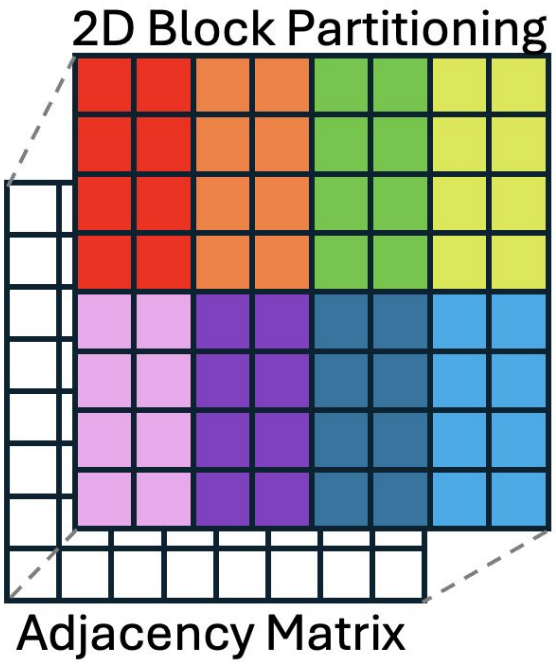
Optimization for local computations
Dense vs. very small comm. detection

Higher initial runtimes: computation

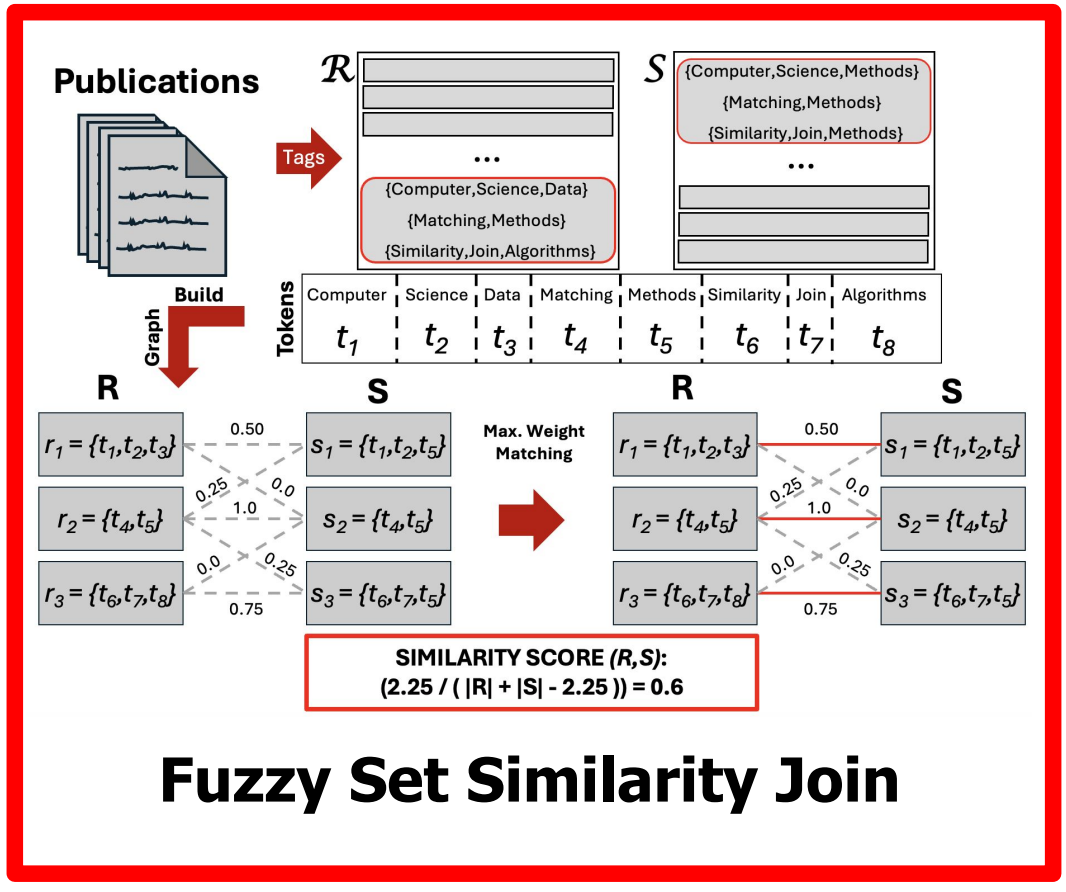
Strong scaling in most cases of general graphs

Plateaus at 64 ranks: problem complexity + synch.

Algorithm Design to Application: Two Cases



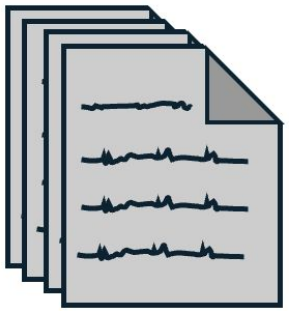
2D Distribution



Fuzzy Set Similarity Join

Fuzzy Set Similarity Join

Publications



How similar are each of the text data entries?

Depends on what is "similar"

Tags

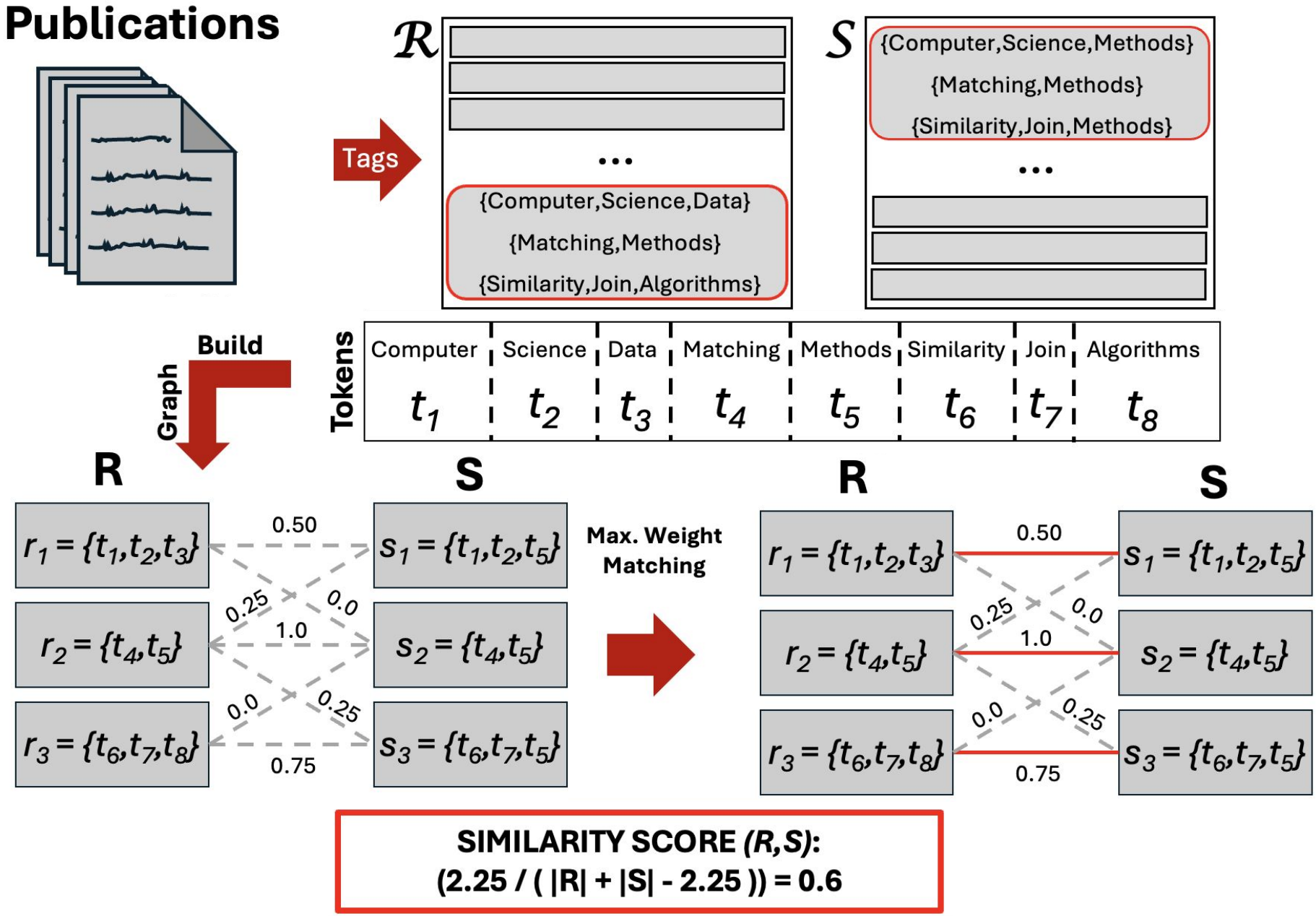
Computer Science Data	↔	Computer Science Methods
Matching Methods	↔	Matching Methods
Similarity Join Algorithms	↔	Similarity Join Methods

Exact comparison – not very similar: 1/3

Solution: Fuzzy Set Similarity

Basis Work: A. Zeakis, D. Skoutas, D. Sacharidis, O. Papapetrou, and M. Koubarakis. "TokenJoin: Efficient Filtering for Set Similarity Join with Maximum Weighted Bipartite Matching" VLDB 2022

Fuzzy Set Similarity with Bipartite Matching



Fuzzy Set Similarity Join Focus

Prior Works Focus: **Filter**-Verify framework

- Verification expensive → optimal matching calculation
- Better eliminate pairings in filtering → less work later

Our Focus: Filter-**Verify** framework

- Approx. matching is faster → quicker verification

	Variants	Approx.	Time	Memory
TokenJoin (TJPJ)	Hungarian (TJPJ–HG)	1	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
	Efficient Verification (TJPJ–EV)			
ApproxJoin (AJ)	Locally Dominant (AJ–LD)	0.5	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n^2)$
	Greedy (AJ–GD)			
	Paz Schwartzman (AJ–PS)	0.5	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$

A. Zeakis, D. Skoutas, D. Sacharidis, O. Papapetrou, and M. Koubarakis. "TokenJoin: Efficient Filtering for Set Similarity Join with Maximum Weighted Bipartite Matching" VLDB 2022
D. Deng, A. Kim, S.l Madden, and M.l Stonebraker. "SilkMoth: an efficient method for finding related sets with maximum matching constraints." VLDB 2017

Approx. Matching Integration

Algorithm D.1 Set Verification

Input: Candidate Sets: R, S, Threshold: θ_R
Output: Score: $sim_\phi(R, S)$

1: $\text{ov} = |R \cap S|$ ▷ *Phase 1: Deduplication*
2: $R_d, S_d = \texttt{deduplication}(R, S)$
3: **if** $R_d = \emptyset$
4: **return** $\frac{\text{ov}}{(|R|+|S|-\text{ov})}$
5: $G(V, E, w) = \emptyset, \text{UB} = |R|$ ▷ *Phase 2: Graph Building*
6: **for all** $r \in R_d$ **do**
7: $max_s = 0$
8: **for all** $s \in S_d$ **do**
9: $score = \texttt{sim}(r, s)$
10: $max_s = \texttt{max}(max_s, score)$
11: $V = V \cup r \cup s$
12: $E = E \cup \{r, s, score\}$ ▷ *Add edge to graph*
13: $\text{UB} = \text{UB} - (1 - max_s)$
14: **if** $\theta_R > \text{UB}$
15: **return** $\frac{\text{UB}}{(|R|+|S|-\text{UB})}$
16: $W_M = \text{ov} + \texttt{matching}(G)$ ▷ *Phase 3: Bipartite Matching*
17: **return** $\frac{W_M}{(|R|+|S|-W_M)}$

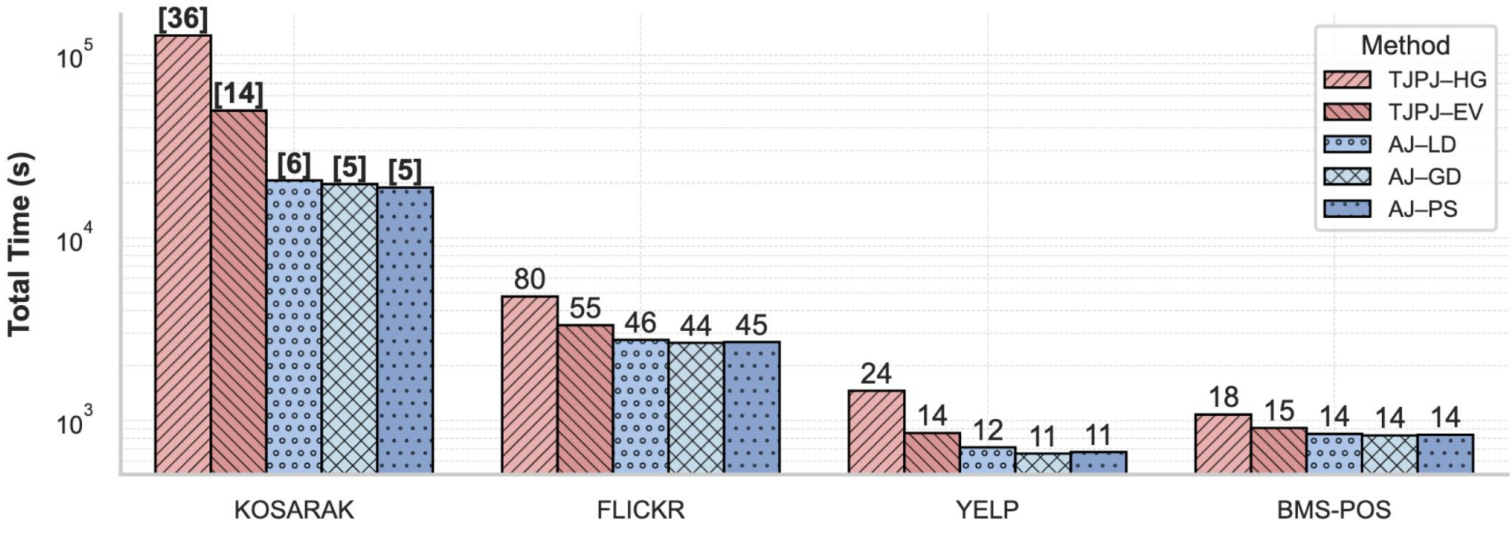
Expensive!!

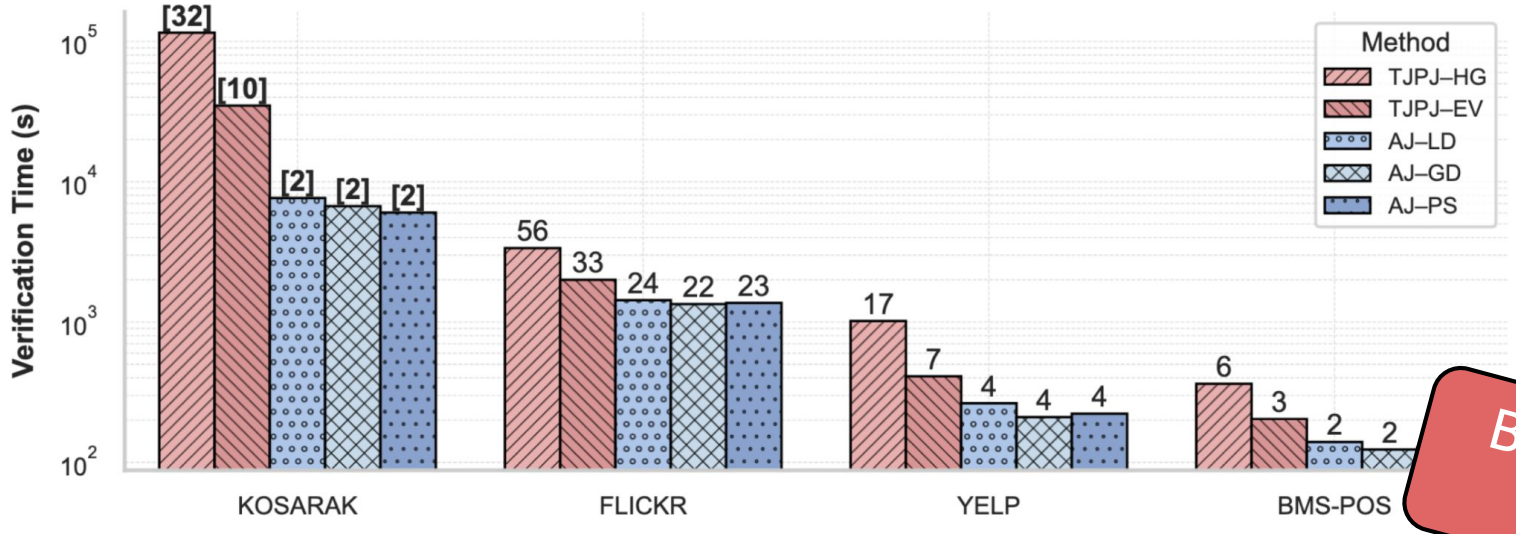
Algorithm 3.1 Verification with PS Matching

Input: Candidate Sets: R, S, Threshold θ_R, constant ϵ
Output: Score: $sim_\phi(R, S)$

1: $R_d, S_d, \text{ov} = \texttt{deduplication}(R, S)$
2: $\text{UB} = |R|; M \leftarrow \emptyset; \text{Stk} \leftarrow \emptyset; \forall v \in V : \phi(v) = 0$
3: **for all** $r \in R_d$ **do**
4: $max_s = 0$
5: **for all** $s \in S_d$ **do**
6: $w(e) = \texttt{sim}(r, s)$ ▷ $e = \{r, s\}$
7: $max_s = \texttt{max}(max_s, w(e))$
8: **if** $w(e) > (1 + \epsilon)(\phi(r) + \phi(s))$ ▷ *Streaming Process*
9: $w'(e) = w(e) - (\phi(r) + \phi(s))$
10: $\phi(r) = \phi(r) + w'(e); \phi(s) = \phi(s) + w'(e)$
11: Stk.push(e)
12: $\text{UB} = \text{UB} - (1 - max_s)$
13: **if** $\theta_R > \text{UB}$
14: **return** $\frac{\text{UB}}{(|R|+|S|-\text{UB})}$
15: **while** Stk $\neq \emptyset$ **do** ▷ *Post Processing*
16: $e(r, s) \leftarrow \text{Stk.pop}()$
17: **if** $(V(M) \cap \{r, s\}) = \emptyset$ ▷ *$V(M)$: vertex set covered by M.*
18: $M \leftarrow M \cup \{e\}$
19: **return** $\frac{w(M)+\text{ov}}{(|R|+|S|-(w(M)+\text{ov}))}$

ApproxJoin Results – Performance





[#]: hours
other: minutes

Up to 19x improvement with AJ-PS

Avg. 2x-4x improvement of verify timings

But what about quality?

ApproxJoin Results – Quality

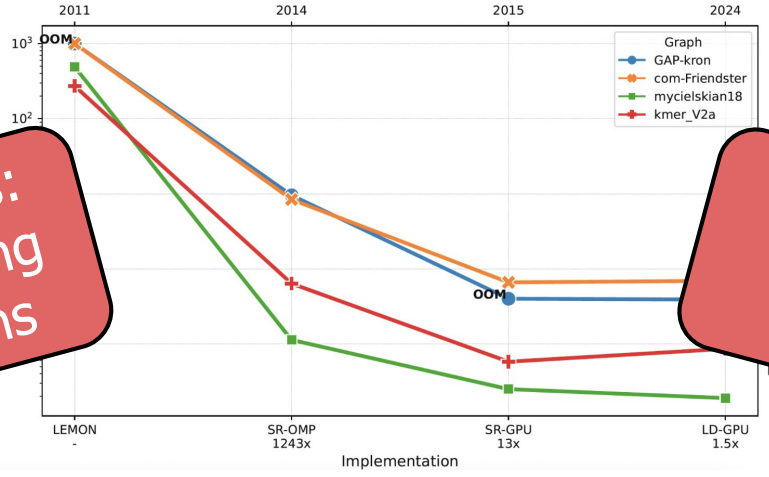
Dataset	Matching Weight based (ApproxJoin)					
	Δ#Sets			Recall		
	GD	LD	PS	GD	LD	PS
LIVEJ	0	0	0	1.0000	1.0000	1.0000
AOL	0	0	0	1.0000	1.0000	1.0000
KOSARAK	-1	-7	-1	0.9999	0.9999	0.9999
ENRON	-4515	-4637	-593	0.9293	0.9274	0.9757
DBLP	0	0	0	1.0000	1.0000	1.0000
FLICKR	-115	-115	-91	0.9994	0.9994	0.9995
GDELT	-937	-937	-481	0.9992	0.9992	0.9996
BMS-POS	0	0	0	1.0000	1.0000	1.0000
YELP	-47	-47	-80	0.9999	0.9999	0.9998
MIND	-38	-38	-8	0.9995	0.9995	0.9999

High recall relative to TJPJ (0.99)

ENRON-exception, data dense (133 elements per set)

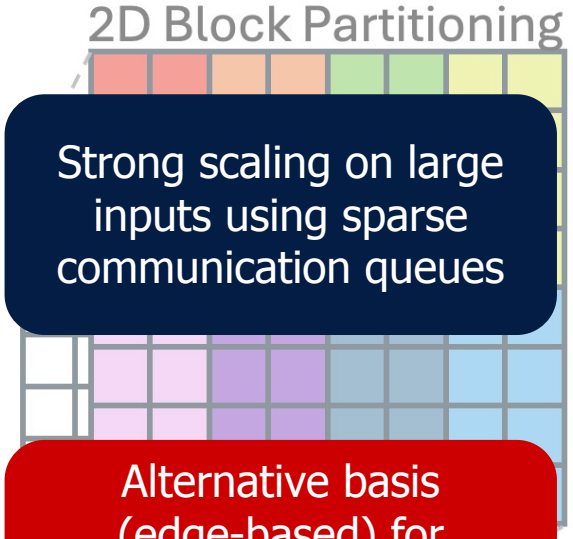
Alternative test of shifting bounds – lower precision but 1.0 recall

Conclusions



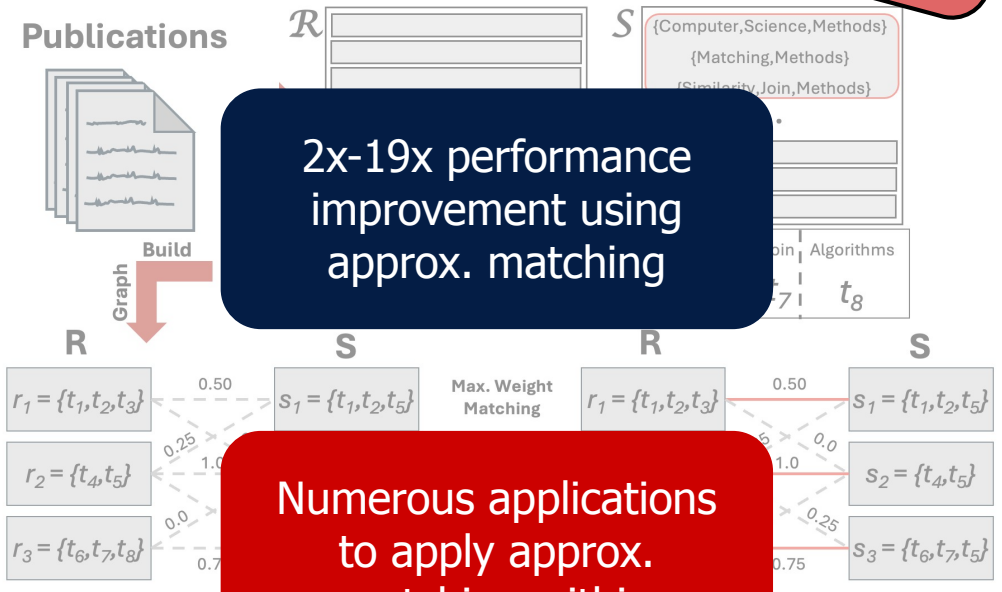
Current works towards: Higher-quality matching algorithm applications

Alternative optimizations for parallel matching



Strong scaling on large inputs using sparse communication queues

Alternative basis (edge-based) for performance improvements



2x-19x performance improvement using approx. matching

Numerous applications to apply approx. matching within

Acknowledgements & Contact

Contact

- Michael Mandulak: mandum@rpi.edu
- George Slota: slotag@rpi.edu

Acknowledgement

This work is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) Program through the FASTMath Institute under Contract No. DE-SC0021285 at Rensselaer Polytechnic Institute, Troy NY, by the National Science Foundation under Grant No. 2047821 and This research is in parts supported by the National Science Foundation under Grant No. 2047821, the U.S. DOE ExaGraph project; Data-Model Convergence Initiative (DMC) and the LDRD initiative at the Pacific Northwest National Laboratory (PNNL). PNNL is operated by Battelle Memorial Institute under Contract DE-AC06-76RL01830.

Collaborators:

- Sayan Ghosh (PNNL)
- S M Ferdous (PNNL)
- Mahantesh Halappanavar (PNNL)

Datasets - ApproxJoin

Datasets	Elements ∈ Set	#Sets	Ele/Set	
			Avg	Max
LIVEJ [30][4]	interests ∈ user	3.1M	36	300
AOL [34][5]	keywords ∈ search	1.8M	3	73
KOSARAK [5][6]	links ∈ user	610K	12	2.5K
ENRON [49][7]	words ∈ email	518K	134	3.2K
DBLP [49] [7]	authors ∈ work	500K	13	189
FLICKR [49] [7]	words ∈ photo	500K	9	361
GDELT [49] [7]	topics ∈ article	500K	19	396
BMS-POS [23][8]	items ∈ sale	320K	6	164
YELP [49] [7]	types ∈ business	160K	6	47
MIND [49] [7]	words ∈ article	123K	32	357