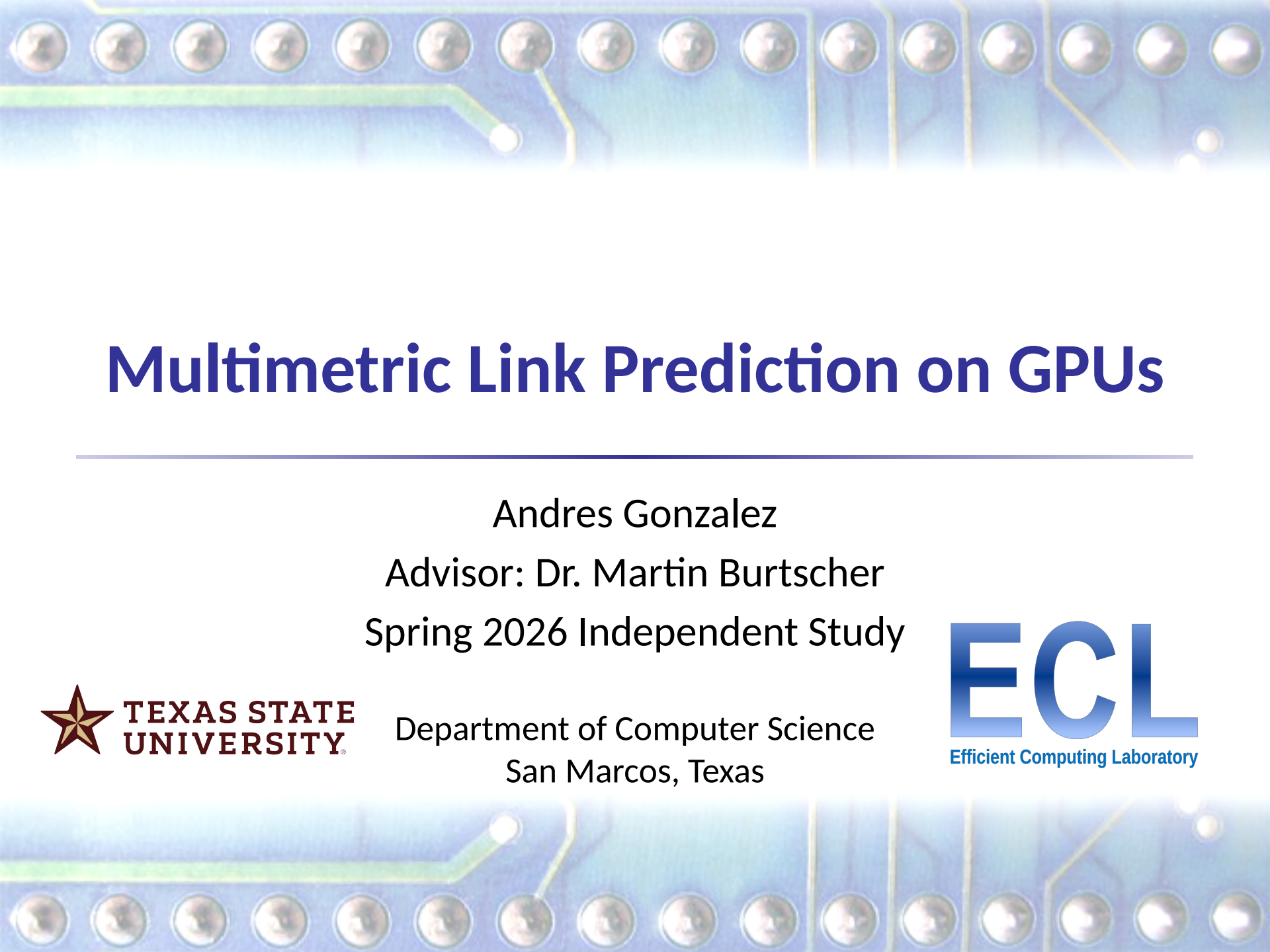




Multimetric Link Prediction on GPUs

Andres Gonzalez

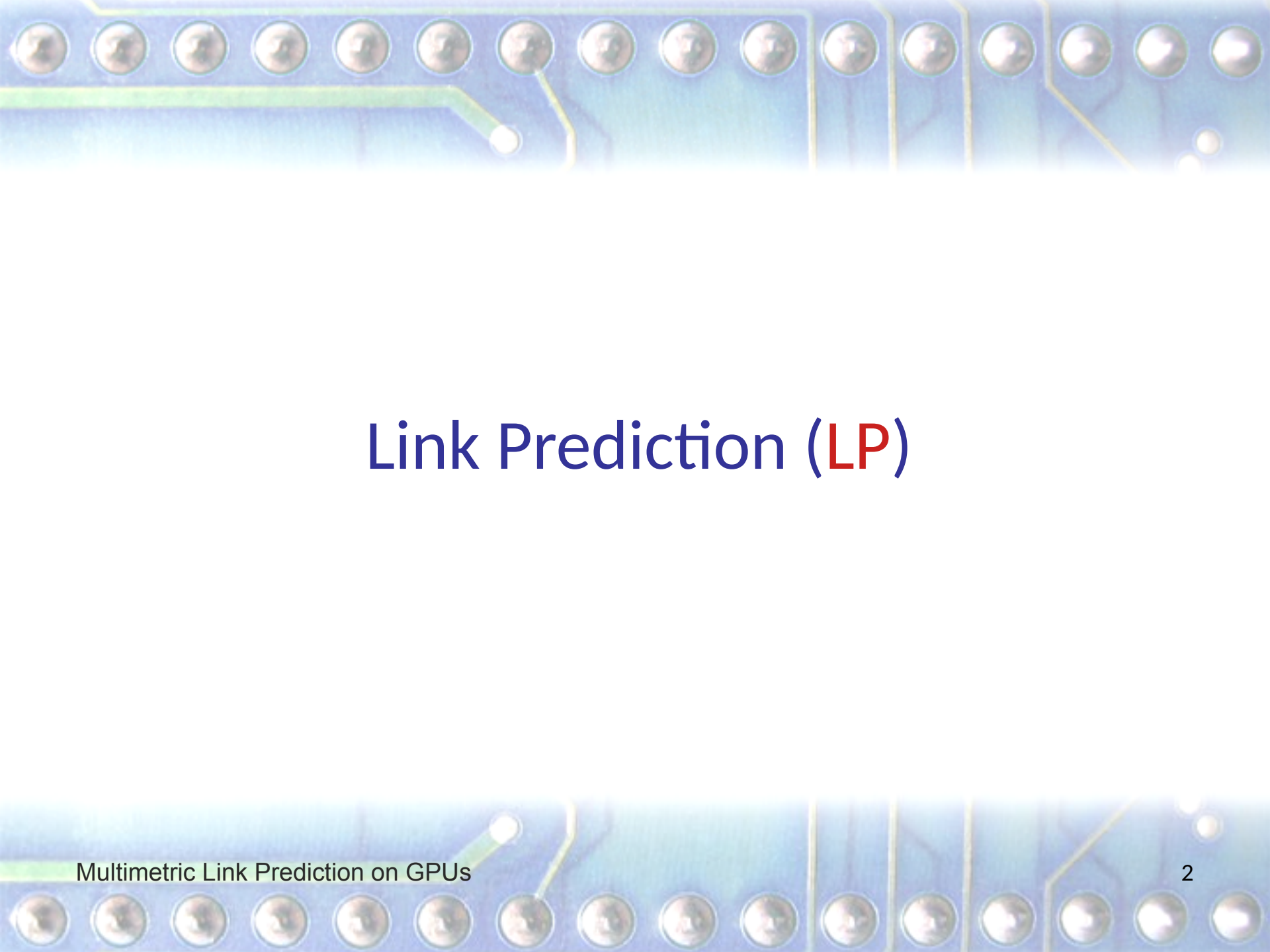
Advisor: Dr. Martin Burtscher

Spring 2026 Independent Study



Department of Computer Science
San Marcos, Texas





Link Prediction (LP)

Link Prediction (LP)

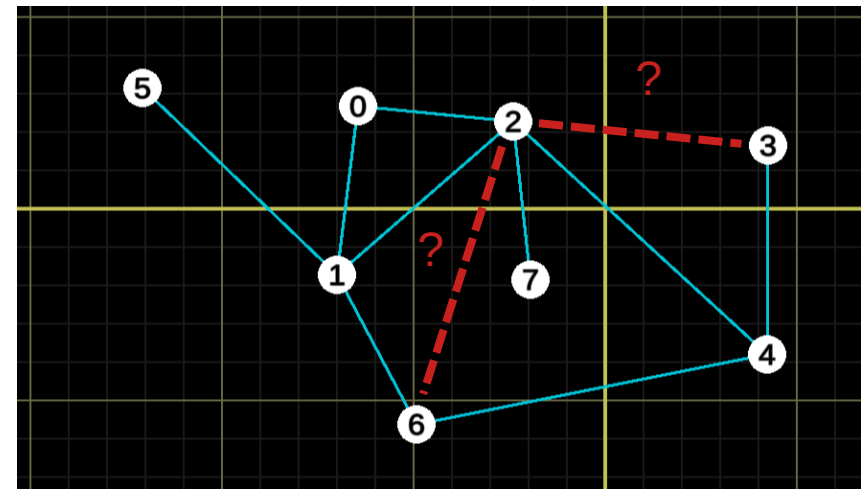
- A classic social-network analysis problem
- Can be generalized to a graph analytics problem
 - Undirected Graph G , Vertices V , Edges E , $G = (V, E)$
 - Predict how likely an edge is to exist between pairs of vertices that do not currently have an edge
 - Network intrinsic properties?

Link Prediction (LP) - Example

- “Who is likely to know who given the already-established links?”

- Nodes are **individuals**
- Edges are established relationships (“**links**”)

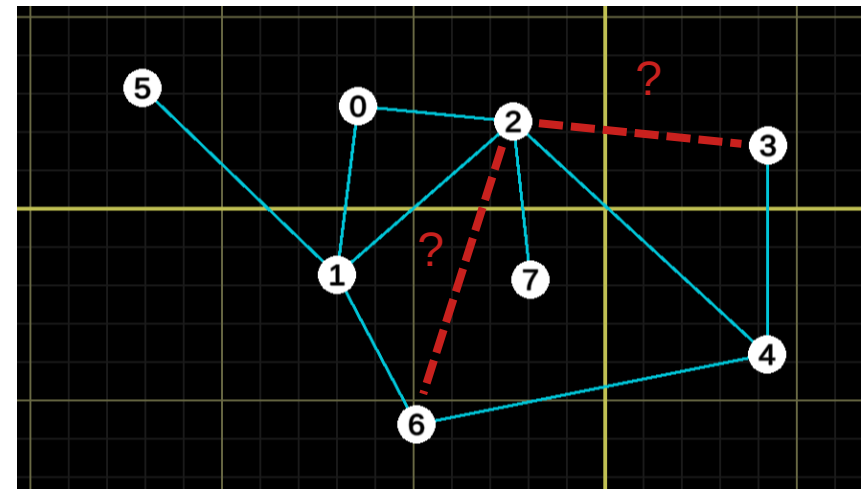
Some Social Network



Link Prediction (LP) - Example

- “2” and “3” both link with “4”
- “2” and “3” linked?
- “2” and “6”?

Some Social Network



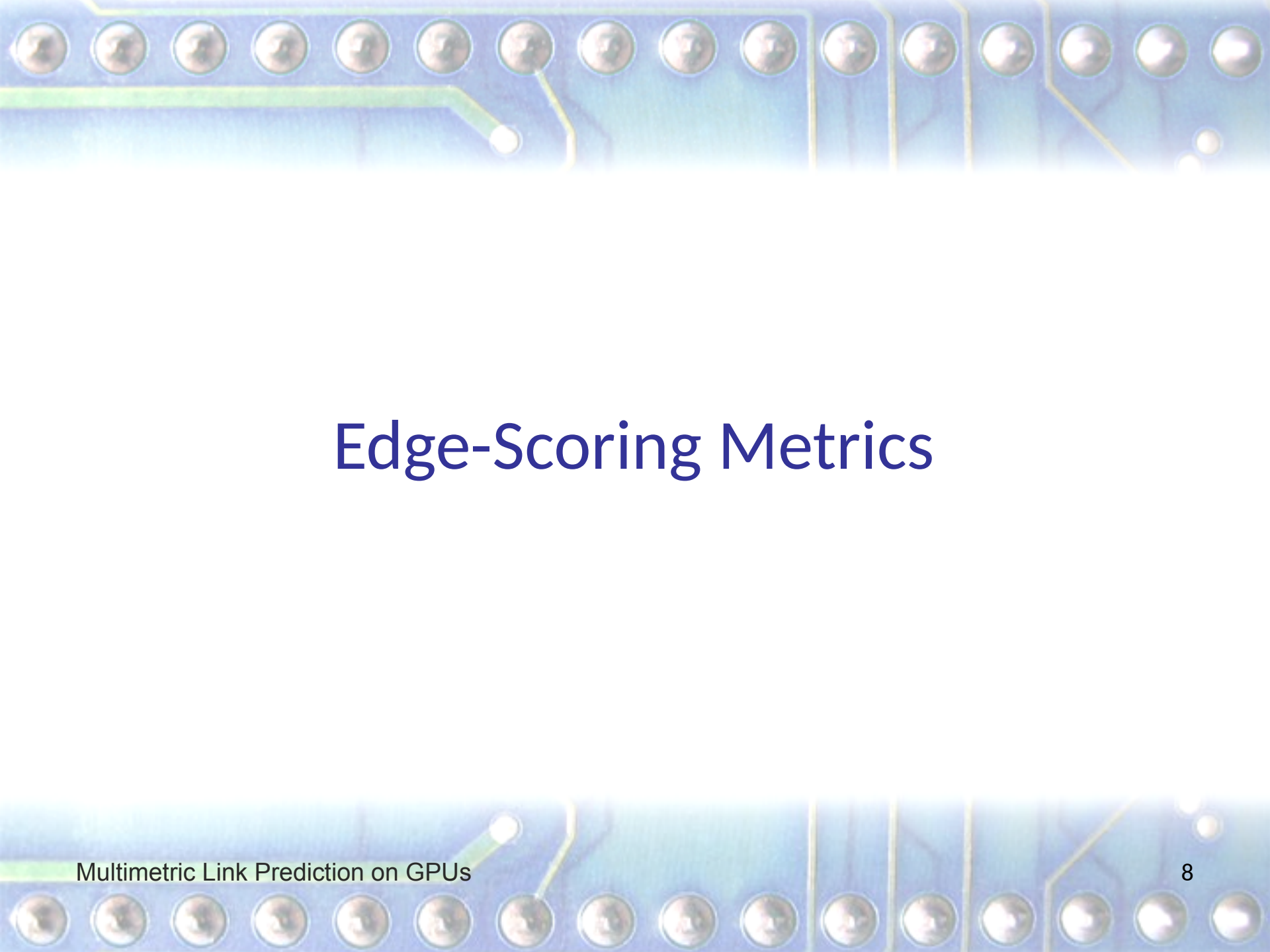
Social Network Representations

- Scientists that have **coauthored** papers
 - Who is likely to work with who?
- Employees in a company who have previously **collaborated** on a project
 - Who can we put together on a project?
- Monitoring criminal **association** networks
 - Who might be working together even though we have not seen any evidence of their communication?

LP Approaches

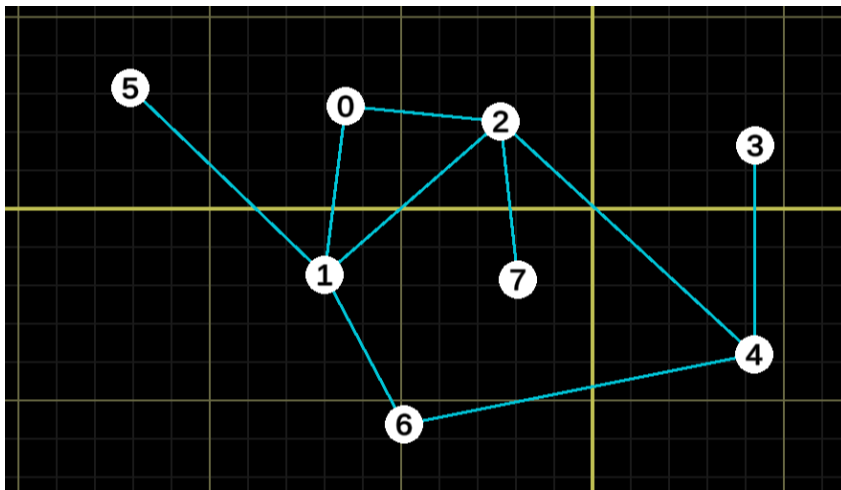
- Similarity- or **Metric-Scoring**^[1]
 - **Local** – Common Neighbors
 - Global – Katz Index
 - Quasi-local – Shortest Paths
- Matrix-Factorization
- ML GNNs
- Strategies are virtually-endless!

[1] <https://cran.r-project.org/web/packages/linkprediction/vignettes/proxfun.html>



Edge-Scoring Metrics

Scoring Metrics – Common Neighbors



$$\text{CN}(a, c) = |\Gamma_a \cap \Gamma_c|$$

$$\Gamma_a = \text{Neighbors}(a) = N(a)$$

$$N(1) = \{0, 2, 5, 6\}$$

$$N(4) = \{2, 3, 6\}$$

$$\text{CN}(1,4) = |\{2, 6\}|$$

$$\text{CN}(1,4) = 2$$

Scoring Metrics

$$\text{CN}(a, c) = |\Gamma_a \cap \Gamma_c|$$

$$\text{JC}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a \cup \Gamma_c|}$$

$$\text{SI}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a| + |\Gamma_c|}$$

$$\text{SC}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\sqrt{|\Gamma_a| \cdot |\Gamma_c|}}$$

$$\text{LHN}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a| \cdot |\Gamma_c|}$$

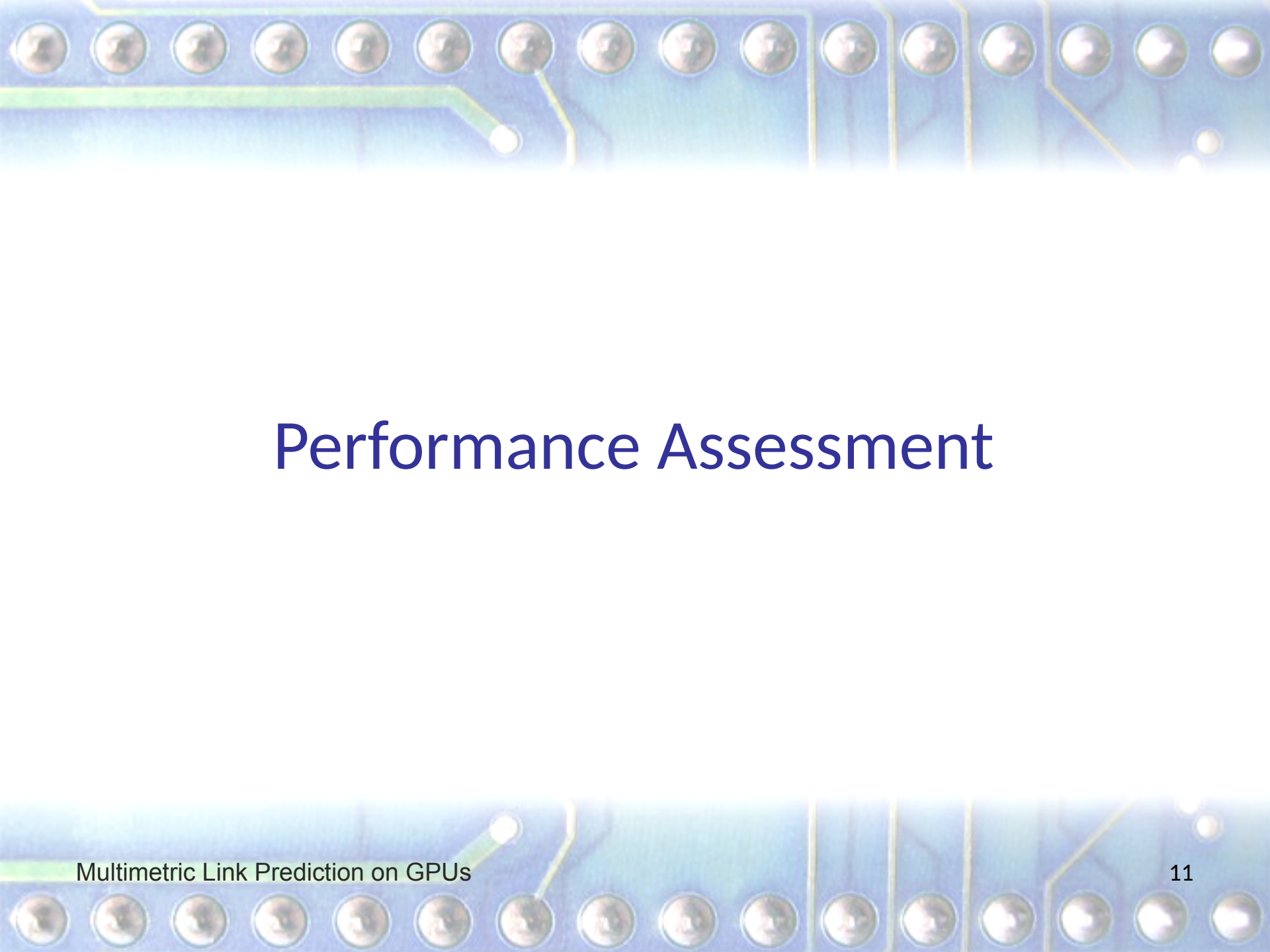
$$\text{HP}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\min(|\Gamma_a|, |\Gamma_c|)}$$

$$\text{HD}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\max(|\Gamma_a|, |\Gamma_c|)}$$

$$\text{AA}(a, c) = \sum_{b \in \Gamma_a \cup \Gamma_c} \frac{1}{\log |\Gamma_b|}$$

$$\text{RA}(a, c) = \sum_{b \in \Gamma_a \cup \Gamma_c} \frac{1}{|\Gamma_b|}$$

$$\text{PA}_{xy} = k_a * k_c$$



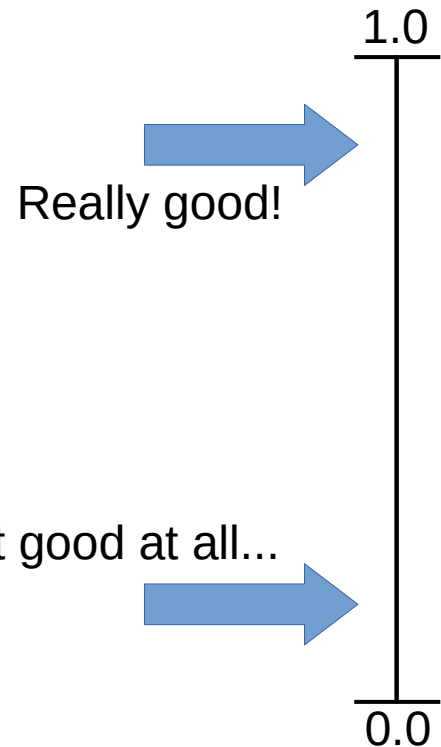
Performance Assessment

Performance Assessment

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1 score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

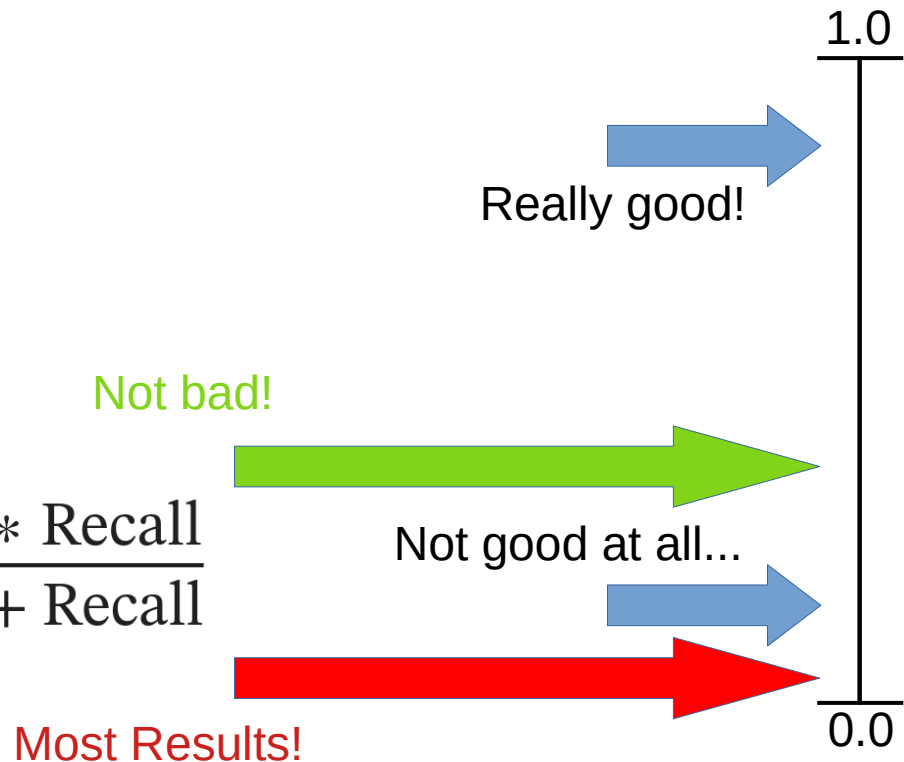


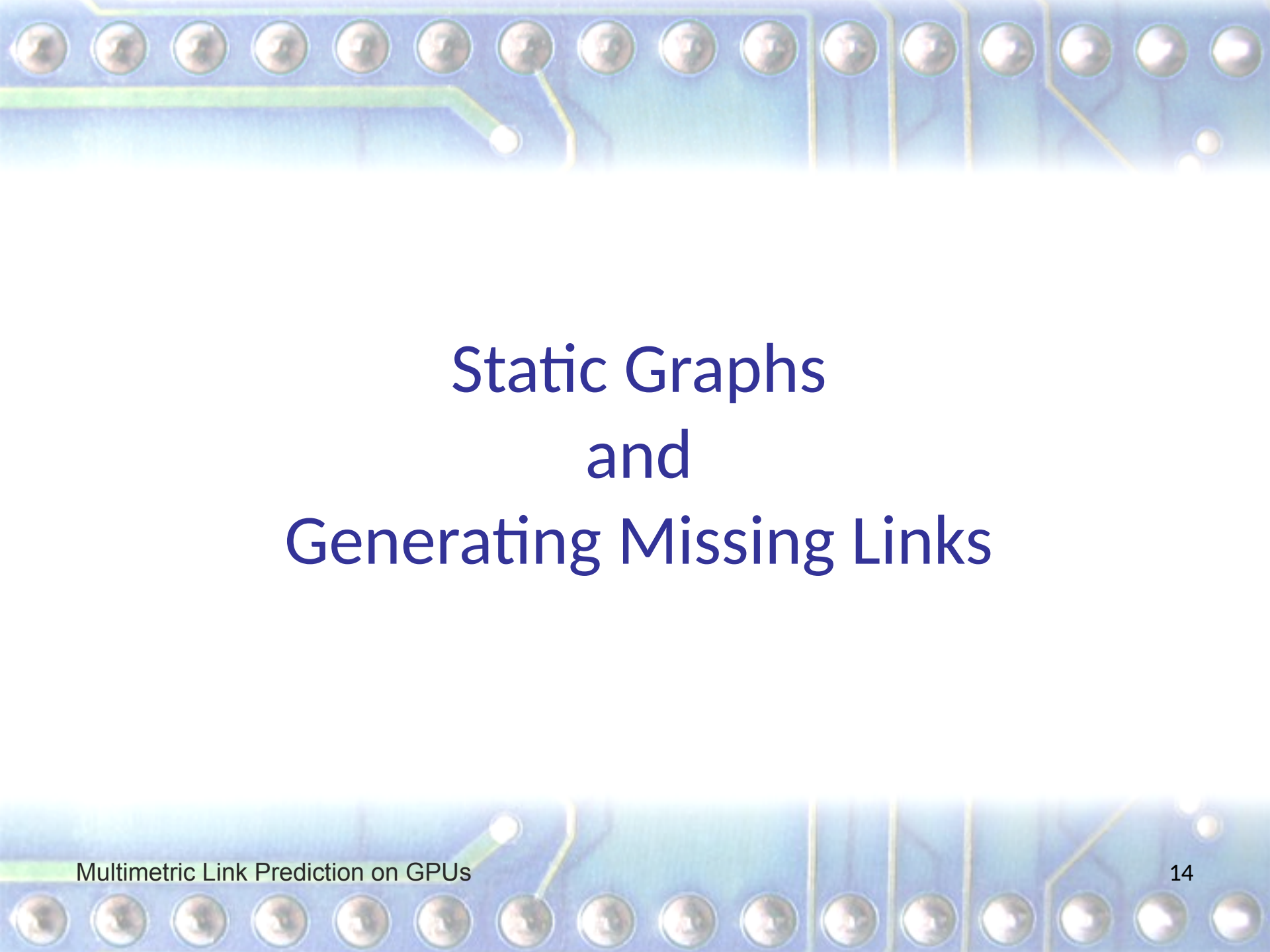
Performance Assessment

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1 score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$





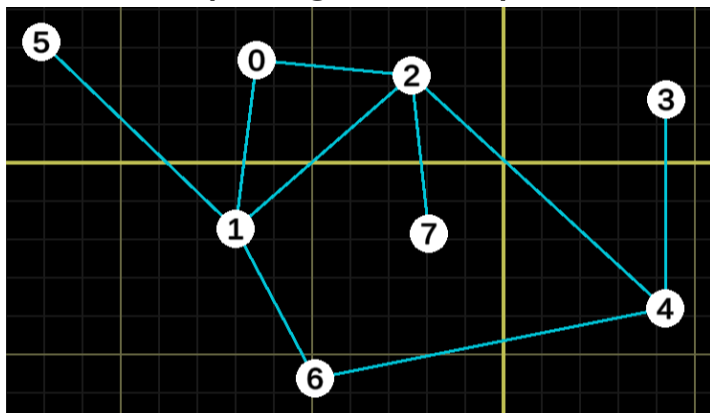
Static Graphs and Generating Missing Links

Generating Inputs

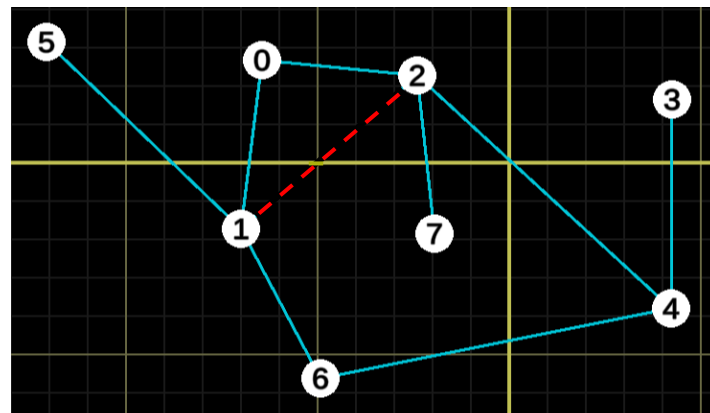
- Social networks take snapshots at points in time
 - Predict edges on 2016 snapshot
 - Compare to actual edges in 2024
- Static graphs with randomly-removed edges
 - 10% edge-removal keeps structure of network intact
 - Goal is to “re-predict” removed edges

Static Graph LP Workflow

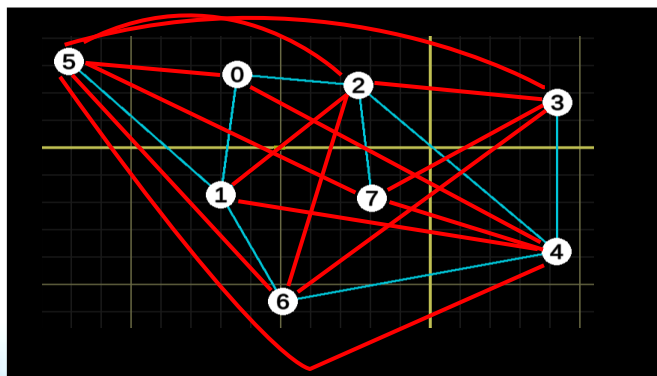
1) Original Graph



2) 10% Edges Removed Graph



3) Predict which potential edges are the true positives



Adjacency Matrix

1	1	0	0	0	0	0	0
0	0	0	1	1	1	0	0
0	0	1	0	0	1	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Predict From

Adjacency Matrix

1	1	0	0	0	0	0	0
1	0	0	1	1	1	0	0
0	0	1	0	0	1	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Compare To

ECL Graphs

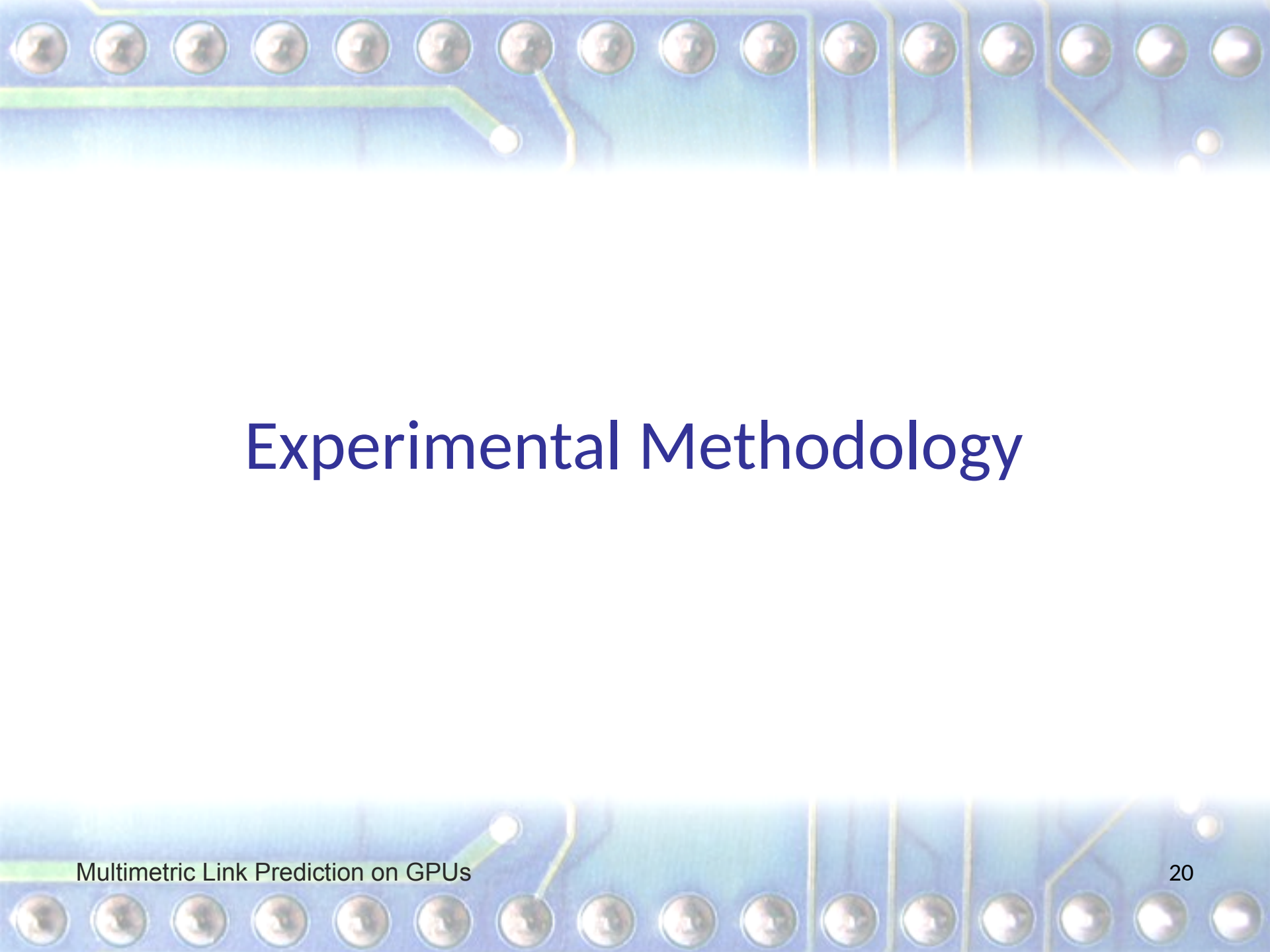
name	nodes	edges	min_deg	ave_deg	max_deg
internet	124,651	387,240	0	3.1	151
USA-road-d.NY	264,346	730,100	0	2.7	8
citationCiteseer	268,495	2,313,294	0	8.6	1,318
rmat16.sym	65,536	967,866	0	14.7	569
amazon0601	403,394	4,886,816	0	12.1	2,752
2d-2e20.sym	1,048,576	4,190,208	0	3.9	4
as-skitter	1,696,415	22,190,596	0	13.0	35,455
in-2004	1,382,908	27,182,946	0	19.6	21,869
delaunay_n24	16,777,216	100,663,202	0	5.9	26
rmat22.sym	4,194,304	65,660,814	0	15.6	3,687
USA-road-d.USA	23,947,347	57,708,624	0	2.4	9
cit-Patents	3,774,768	33,037,894	0	8.7	793
r4-2e23.sym	8,388,608	67,108,846	0	8.0	26
kron_g500-logn21	2,097,152	182,081,864	0	86.8	213,904

3rd Party Graphs 1

name	nodes	edges	min_deg	ave_deg	max_deg
arabic-2005	22,744,080	1,107,806,146	0	48.7	575,628
asia_osm	11,950,757	25,423,206	0	2.1	9
com-LiveJournal	3,997,962	69,362,378	0	17.3	14,815
com-Orkut	3,072,441	234,370,166	0	76.2	33,313
europa_osm	50,912,018	108,109,320	0	2.1	13
indochina-2004	7,414,866	301,969,638	0	40.7	256,425
kmer_A2a	170,728,175	360,585,172	0	2.1	40
kmer_V1r	214,005,017	465,410,904	0	2.1	8
uk-2002	18,520,486	523,574,516	0	28.2	194,955
uk-2005	39,459,925	1,566,054,250	0	39.6	1,776,858
webbase-2001	118,142,155	1,709,619,522	0	14.4	816,127

3rd Party Graphs 2

name	nodes	edges	min_deg	ave_deg	max_deg
coAuthorsCiteseer	227,320	1,628,268	0	7.1	1,372
coAuthorsDBLP	299,067	1,955,352	0	6.5	336
coPapersCiteseer	434,102	32,073,440	0	73.8	1,188
coPapersDBLP	540,486	30,491,458	0	56.4	3,299
germany_osm	11,548,845	24,738,362	0	2.1	13
great-britain_osm	7,733,822	16,313,034	0	2.1	8
italy_osm	6,686,493	14,027,956	0	2.0	9
soc-Epinions1	75,888	811,480	0	10.6	3,044
soc-LiveJournal1	4,847,571	85,702,474	0	17.6	20,333
soc-Slashdot0811	77,360	938,360	0	12.1	2,539
soc-Slashdot0902	82,168	1,008,460	0	12.2	2,552
web-BerkStan	685,230	13,298,940	0	19.4	84,230
web-Google	916,428	8,644,102	0	9.4	6,332
web-NotreDame	325,729	2,180,216	0	6.6	10,721
web-Stanford	281,903	3,985,272	0	14.1	38,625

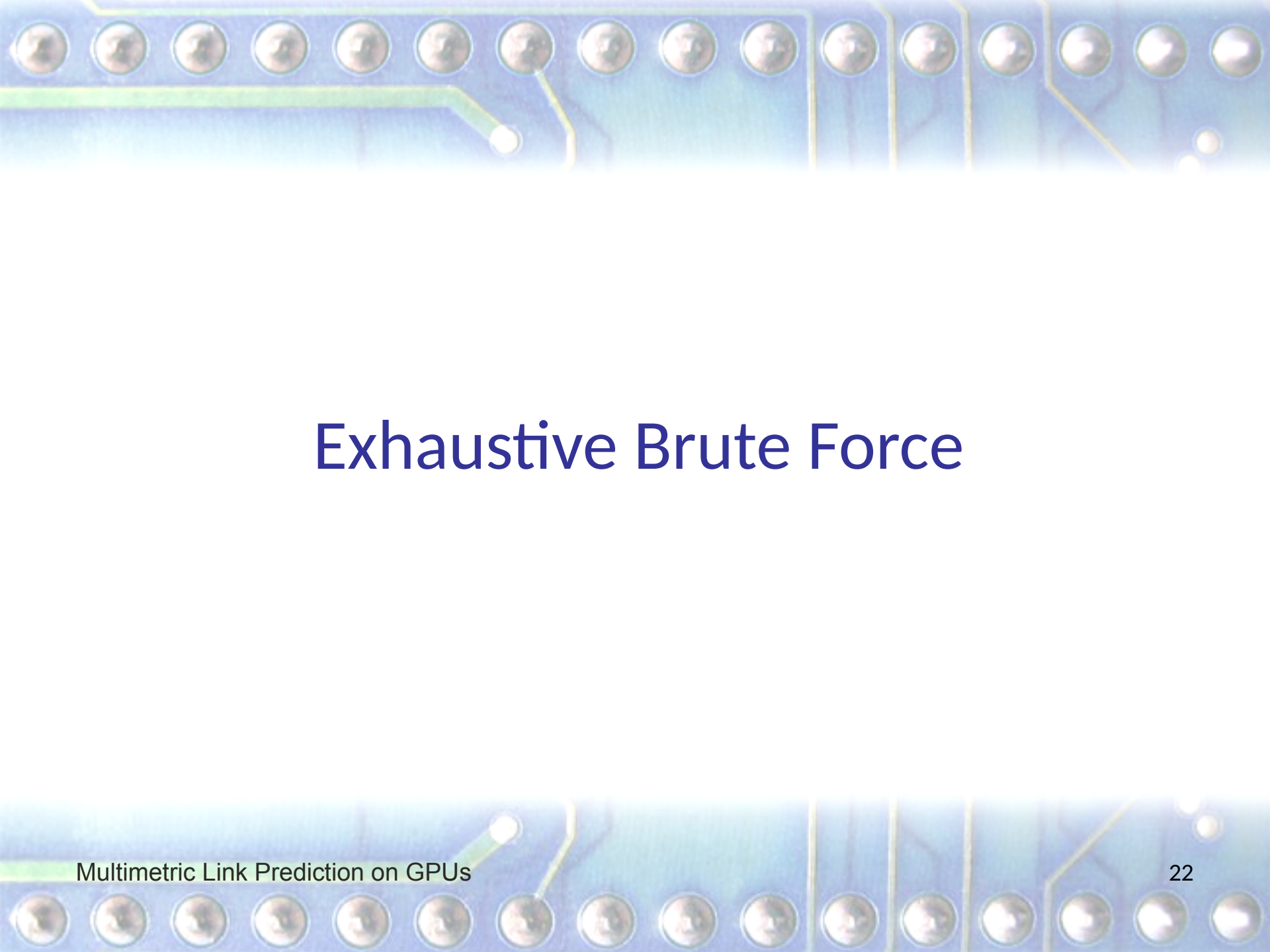


Experimental Methodology

Experimental Methodology

- Intel Xeon Gold **6226R**, 16 cores per socket at 2 threads per core, 3.5GHz, gcc 13.3

	PE	SM	PE/SM	Mem (GB)	NVCC
RTX 4090	16384	128	128	24	12.6
RTX 3090	10496	82	128	24	12.0
A100	6912	108	64	40	12.0
GTX 1650	1024	16	64	4	12.6



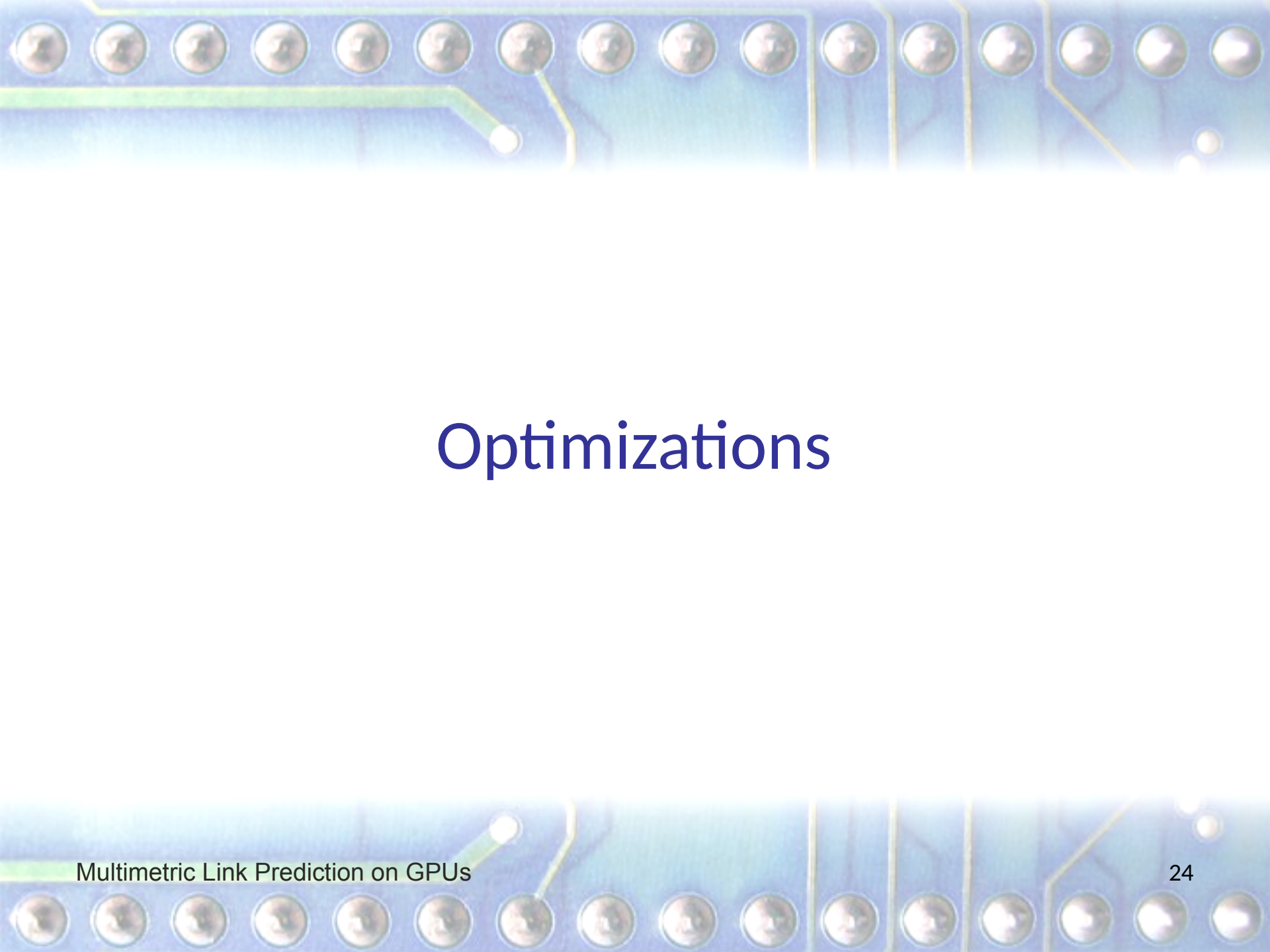
Exhaustive Brute Force

Exhaustive Brute Force Baseline

- Algorithm Complexity is $O(N^2)$
- Brute-force quickly becomes untenable as graph size grows
- Optimizations are necessary

name	nodes	edges	runtime(s)
internet	124,651	387,240	102
rmat16.sym	65,536	967,866	190
USA-road-d.NY	264,346	730,100	204
USA-road-d.BAY	321,270	794,830	292
...			
...			
USA-road-d.E	3,598,623	8,708,058	59,137
USA-road-d.W	6,262,104	15,119,284	142,949
delaunay_n22	4,194,304	25,165,738	169,017

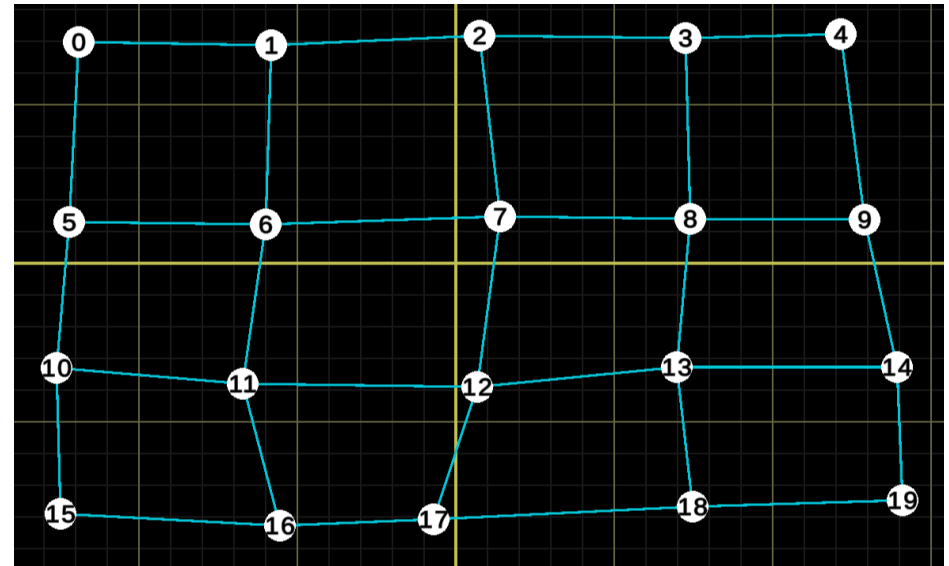
Almost 47 hours!



Optimizations

2-Hop DFS

- Metric scores **derived** from **CN** metric
- Limit to edges that are 2-hop DFS
- E.g., vertices 0 and 19 **do not have common neighbors** anyways...



Original_Possible_Edges(0):

$\{0, 2\}, \{0, 3\}, \{0, 4\}, \{0, 6\}, \{0, 7\}, \{0, 8\}, \{0, 9\}, \{0, 10\}, \{0, 11\}, \{0, 12\}, \{0, 13\}...$

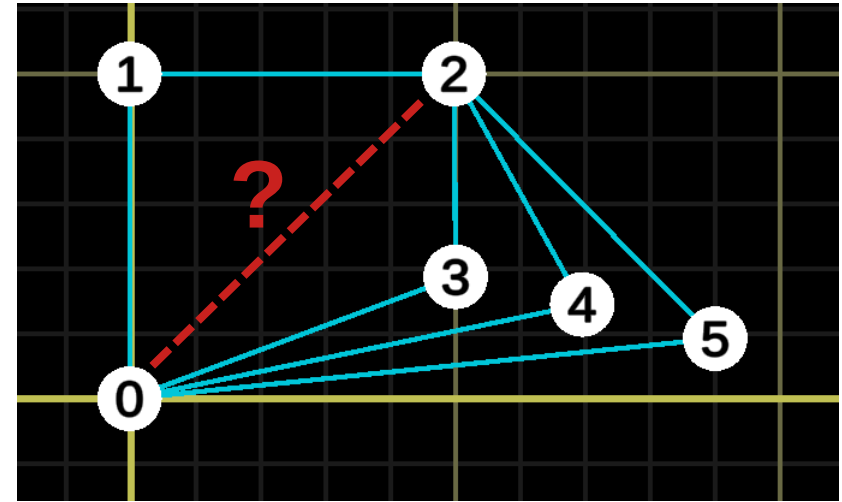
Optimized_Possible_Edges(0):

$\{0, 2\}, \{0, 6\}, \{0, 10\}$

Eliminating Duplicate Computations

- Edge “0” and “2” is computed several times:

- $\{0, 1, 2\}$
- $\{0, 3, 2\}$ - discarded
- $\{0, 4, 2\}$ - discarded
- $\{0, 5, 2\}$ - discarded



- Check if wedge's central vertex is **first CN**:
 - $N(0) = \{1, 3, 4, 5\}$
 - $N(2) = \{1, 3, 4, 5\}$

Chunk with Minimum Score

- **Chunked** edge processing
(~1B on A100)
- Keep only edges whose **score is above** the current TopK's **minimum**

name	10%	Edges To Compute
internet	19,362	1,226,077
USA-road-d.NY	36,505	623,590
soc-Epinions1	40,574	60,068,520
soc-Slashdot0811	46,918	55,520,197
rmat16.sym	48,393	19,203,255
soc-Slashdot0902	50,423	60,790,517
...		
...		
indochina-2004	15,098,482	464,156,535,310
kmer_A2a	18,029,260	184,438,542
kmer_V1r	23,270,546	248,919,342
uk-2002	26,178,726	163,463,396,167
arabic-2005	55,390,312	2,899,304,448,421
webbase-2001	85,480,984	1,015,944,856,564

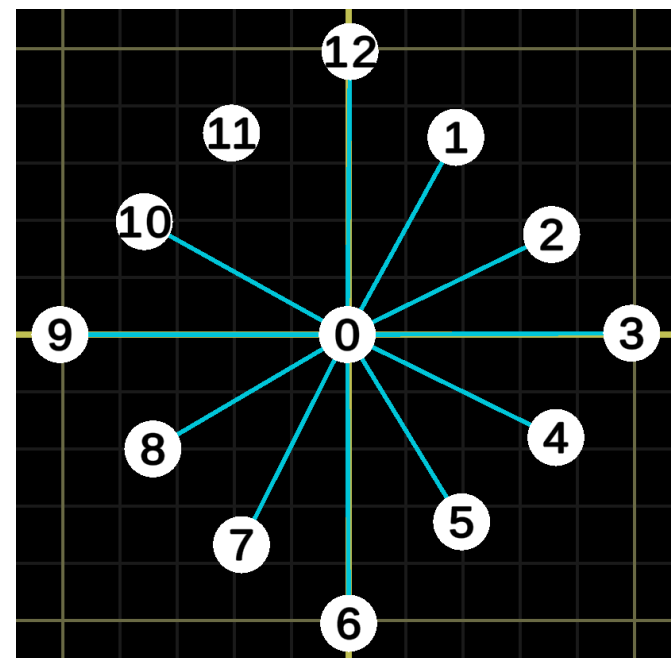
Simply Ignore Some Vertices!

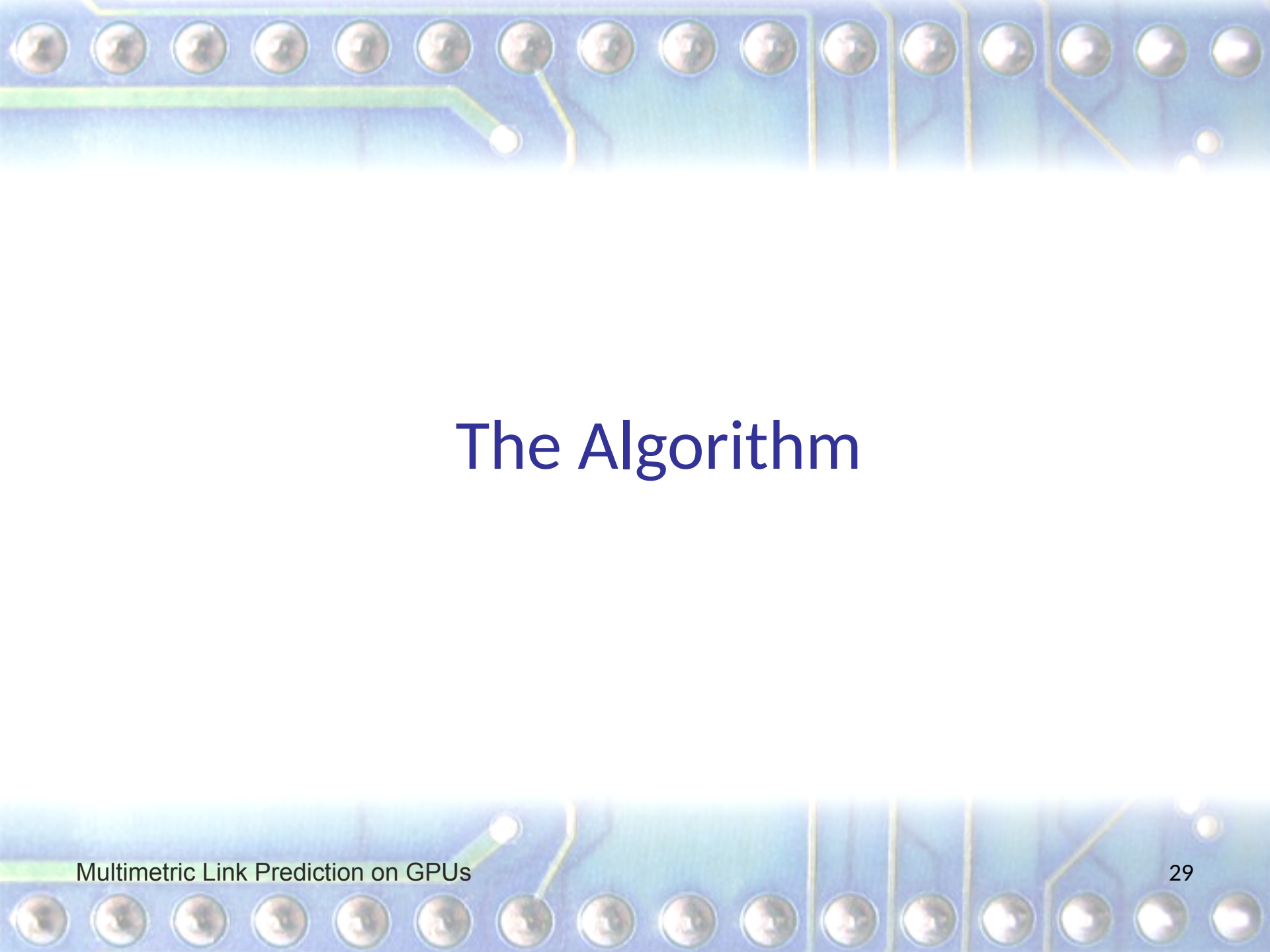
- **Ignore vertices**

- Ignore vertices of degree within top percentile
- Degree above threshold

- **Example**

- Vertex 0 **already has many links**
- Other vertices more likely to make new links





The Algorithm

Algorithm Overview

- Keep **running list** of **top-k edges** to replace the 10% of edges that were removed
- For all edges that need to be computed
 - **Process chunk** of edges
 - **Sort** chunk's list of edges by metric scores' total
 - **Merge** with ongoing list of top-k
 - **Copy back** "k" edges to running list
 - **Update minimum** score

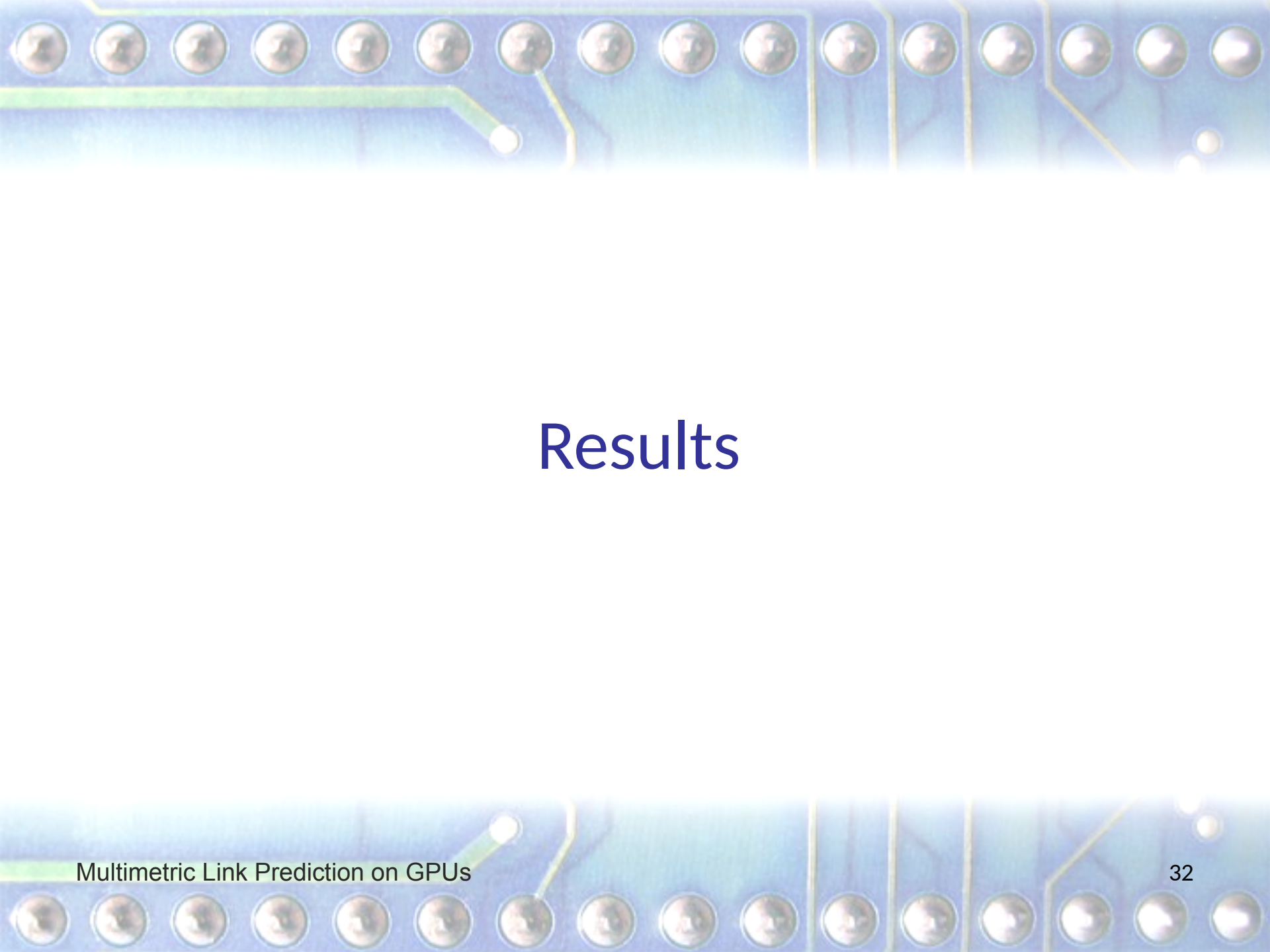
Algorithm Overview (cont'd)

- Compare resulting top-k edges to original graph
 - True Positives – top-k edges that **are in** the original
 - False Positives – top-k edges that **are not in** the original
- Compute F1 Score

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1 score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$



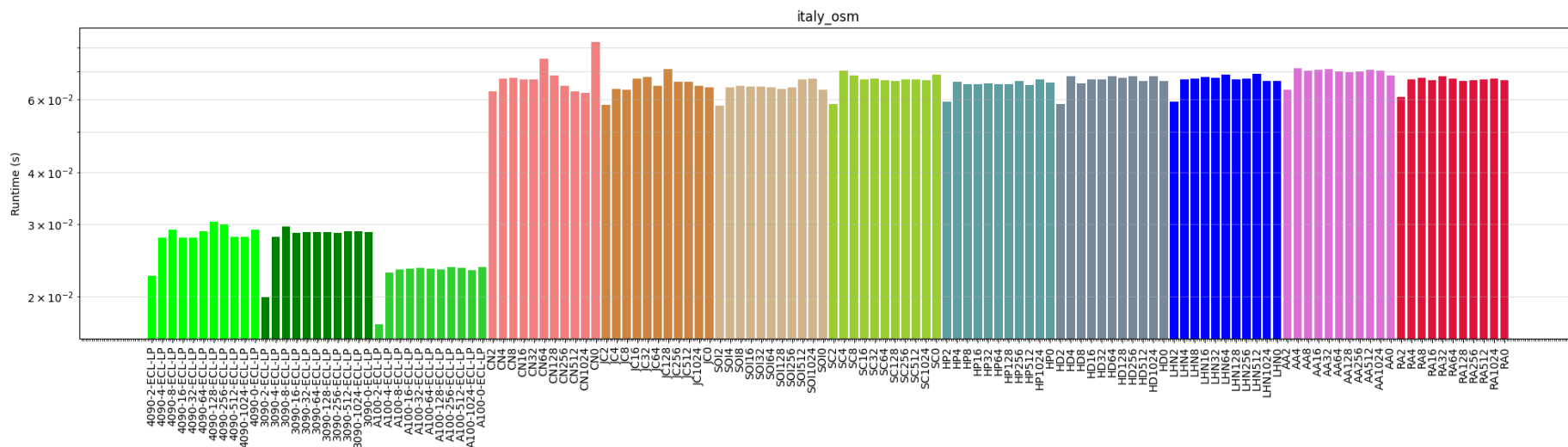
Results



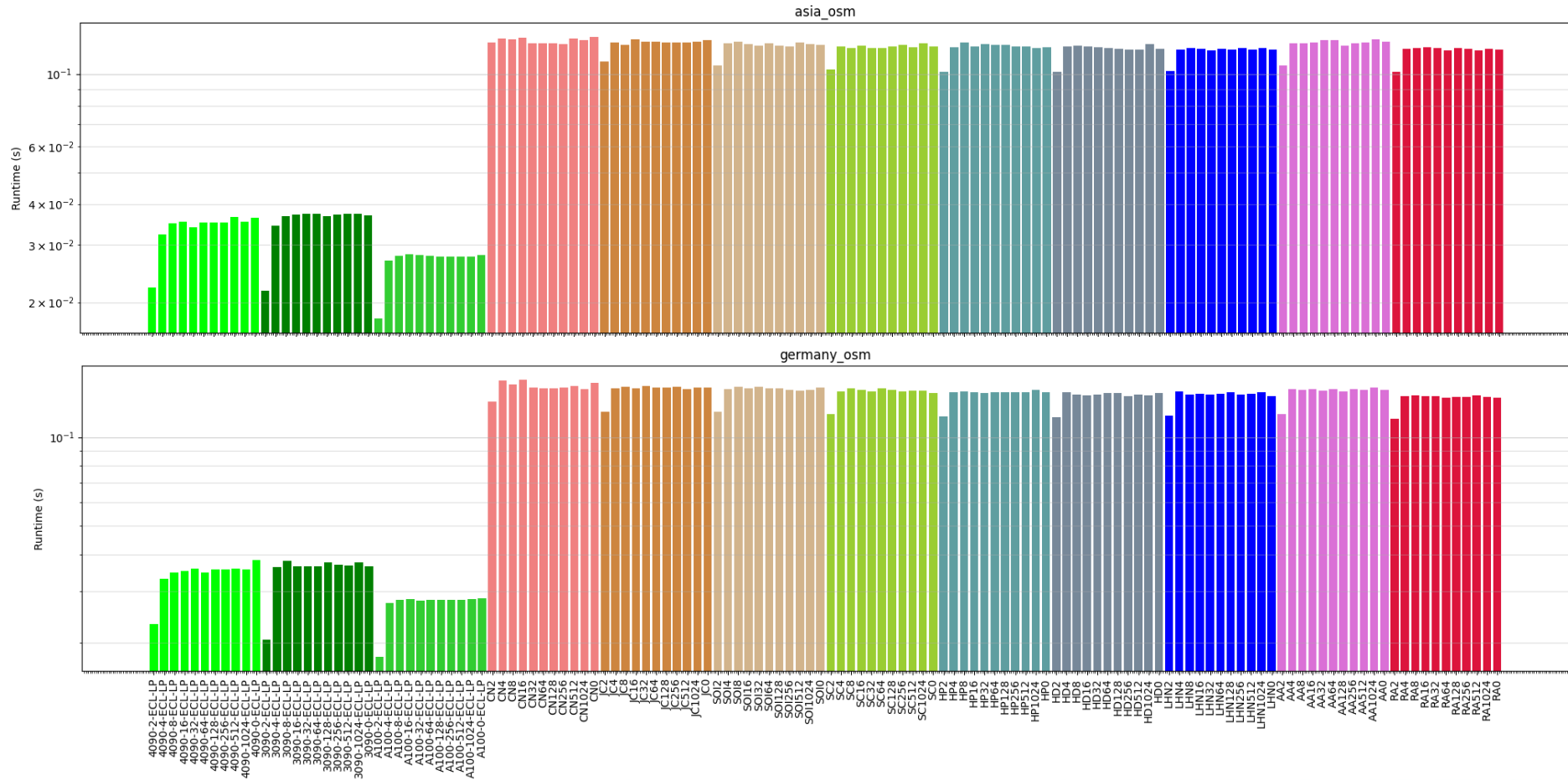
Runtime

Runtime - Compare to 3rd Party Code

- 3rd Party code **SAHU** uses **CPU OpenMP**
 - GPU code outperforms parallel CPU for **uniform degree**
 - GPU code computes **all 10 metrics simultaneously!**

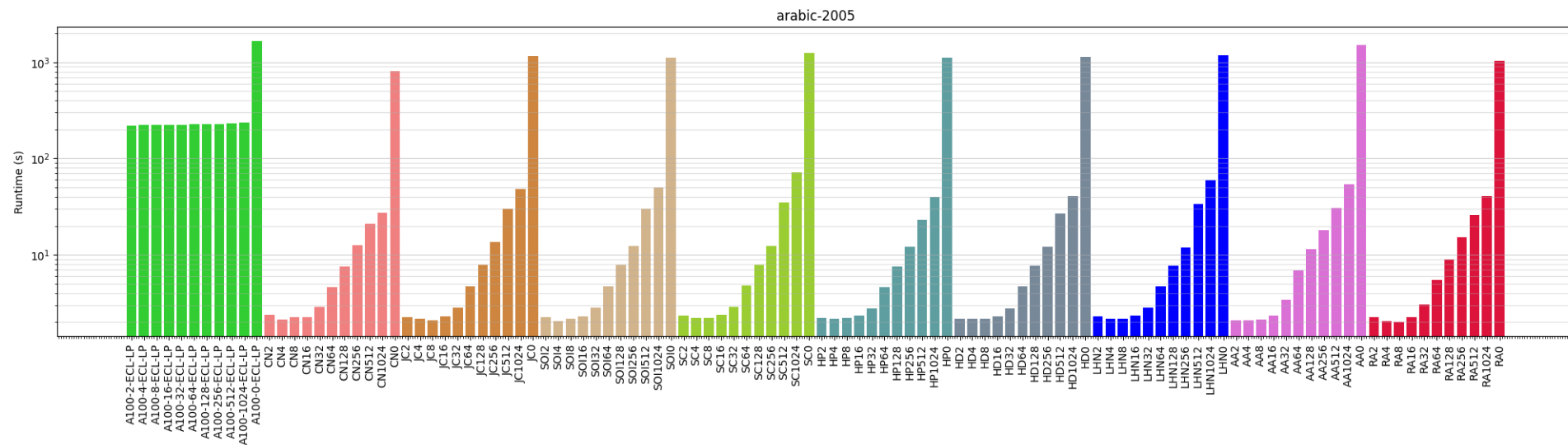


Runtime – Compare to 3rd Party Code

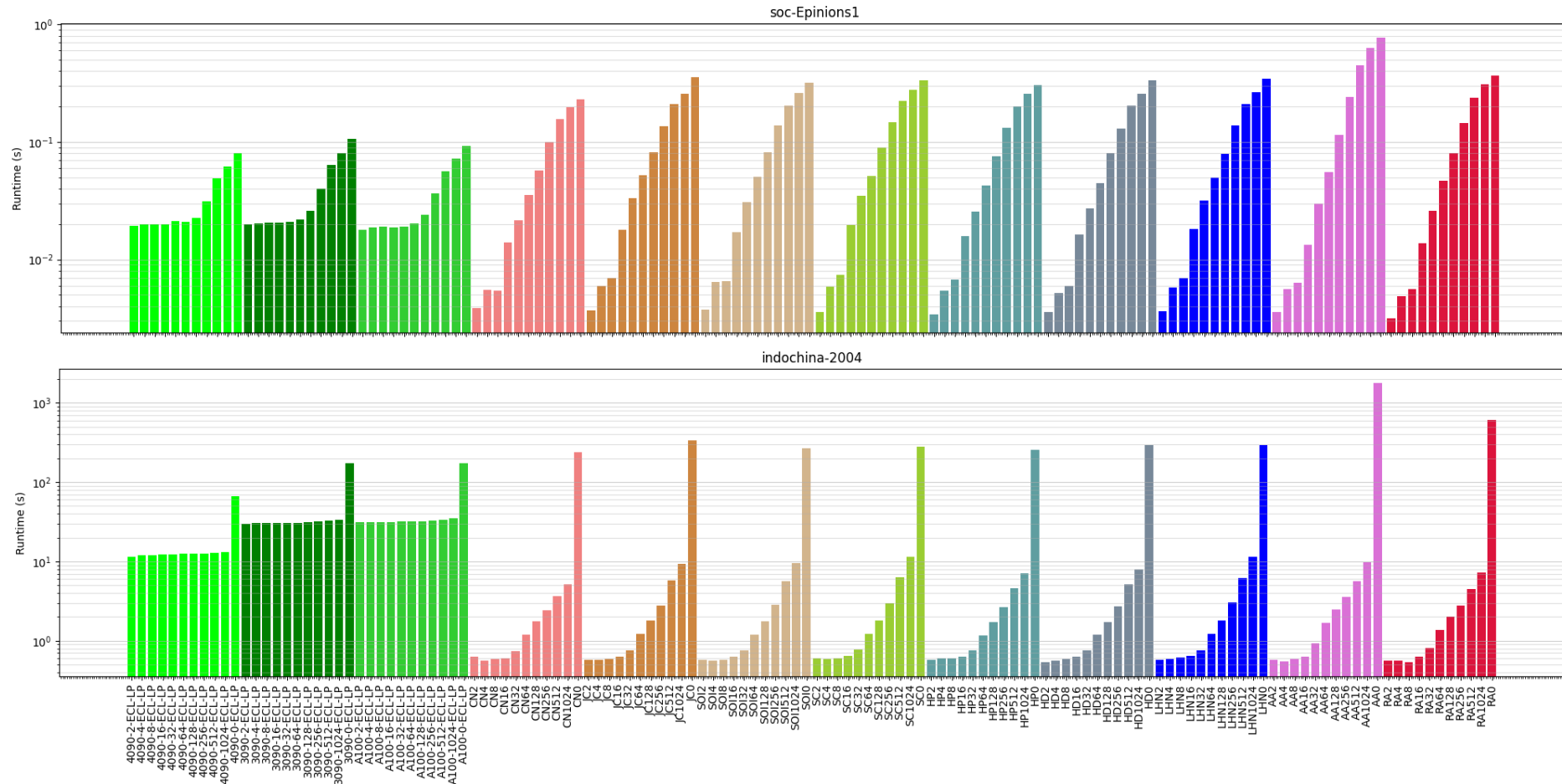


Runtime – Compare to 3rd Party Code

- Need to find solution for **Power Law** graphs
 - GPU sometimes faster / sometimes slower

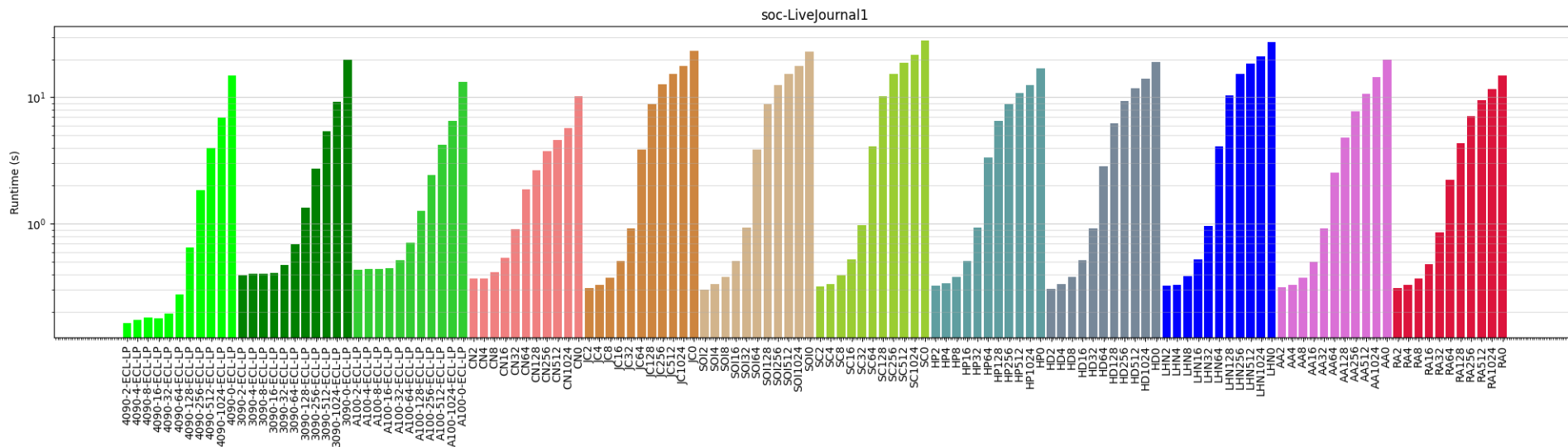


Runtime – Compare to 3rd Party Code

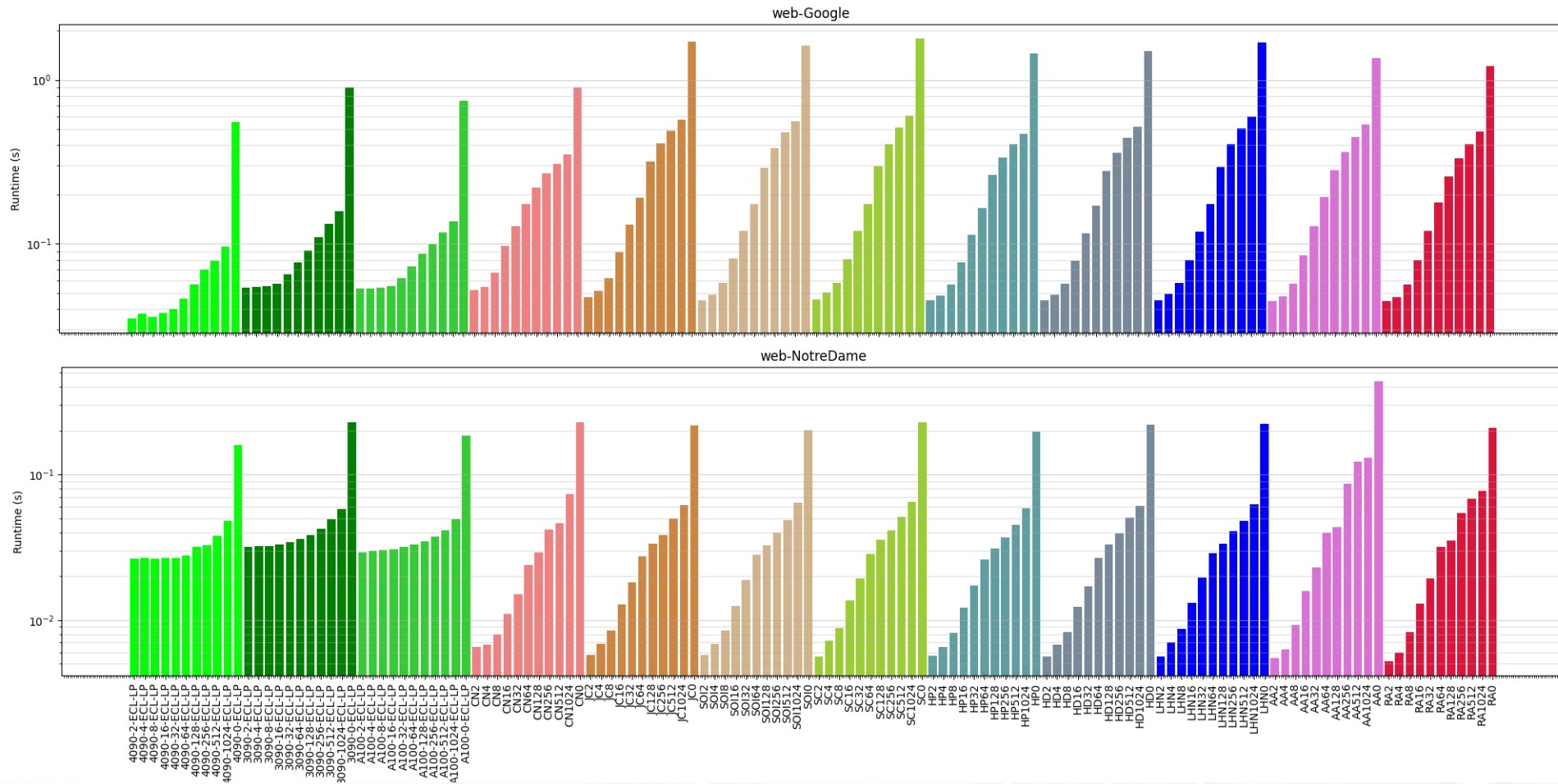


Runtime – Compare to 3rd Party Code

- Other power-law graphs perform on-par or slightly better
 - Good! Remember that GPU computes **all** metrics!

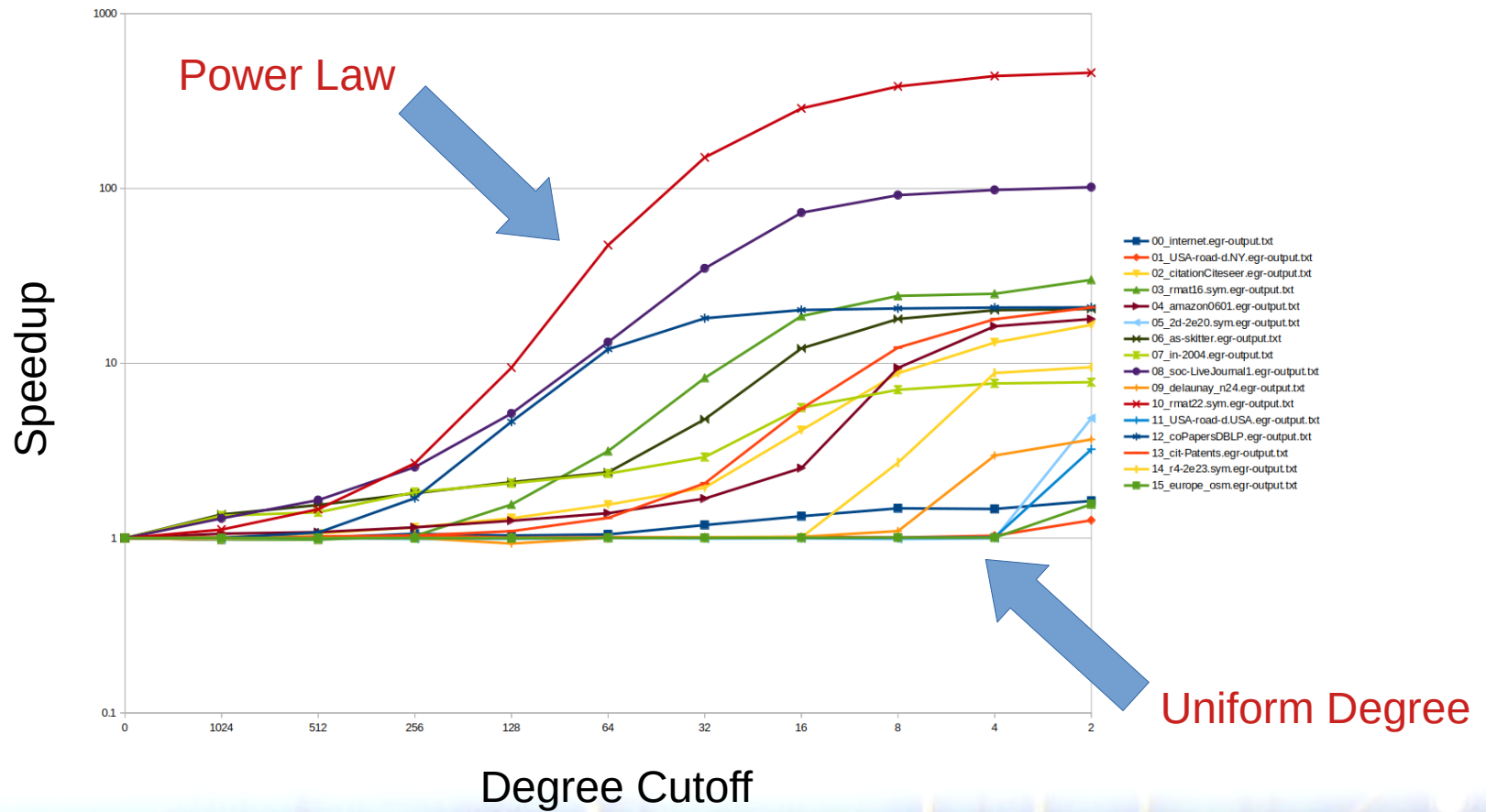


Runtime - Compare to 3rd Party Code



Ignoring High-Degree Vertices

Speedup vs. Degree Cutoff (1650)





F1 Score

Scoring Metrics

$$\text{CN}(a, c) = |\Gamma_a \cap \Gamma_c|$$

$$\text{JC}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a \cup \Gamma_c|}$$

$$\text{SI}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a| + |\Gamma_c|}$$

$$\text{SC}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\sqrt{|\Gamma_a| \cdot |\Gamma_c|}}$$

$$\text{LHN}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a| \cdot |\Gamma_c|}$$

$$\text{HP}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\min(|\Gamma_a|, |\Gamma_c|)}$$

$$\text{HD}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\max(|\Gamma_a|, |\Gamma_c|)}$$

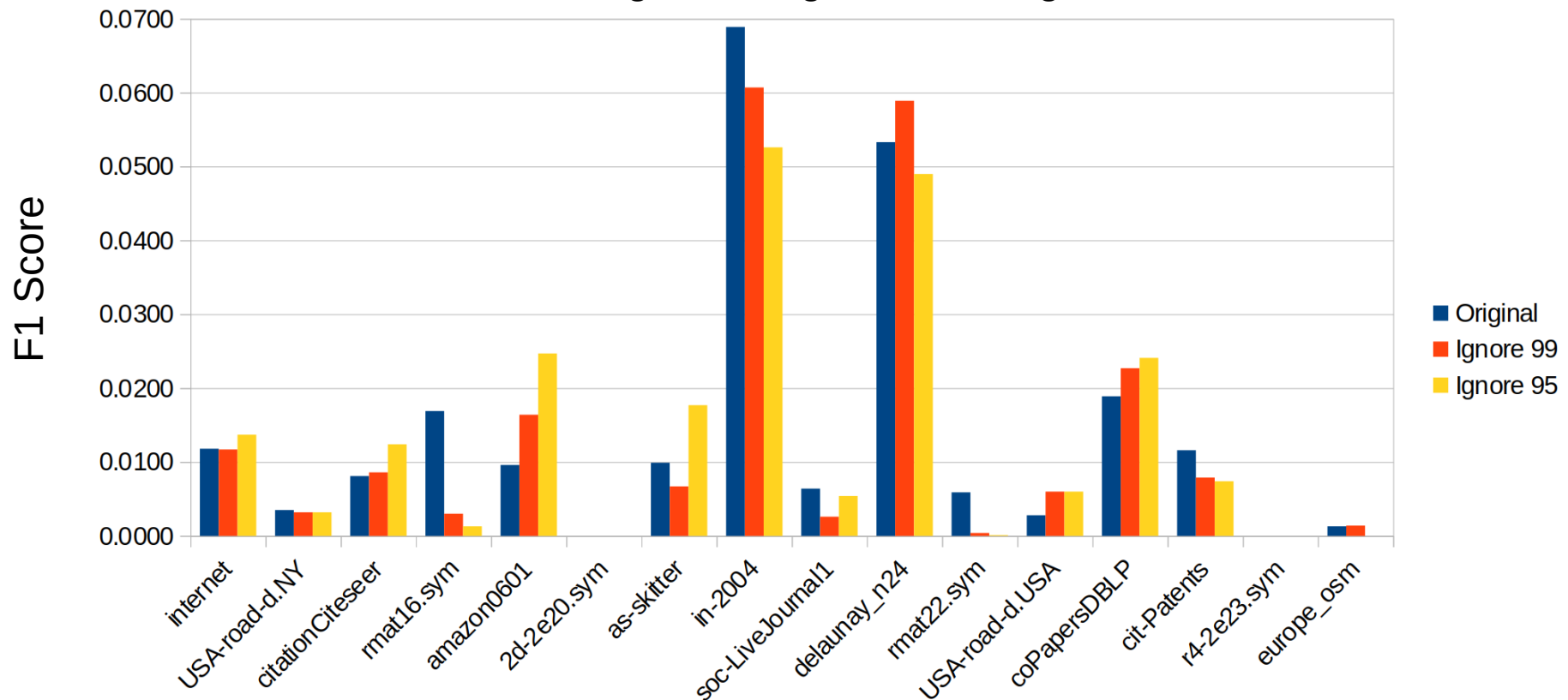
$$\text{AA}(a, c) = \sum_{b \in \Gamma_a \cup \Gamma_c} \frac{1}{\log |\Gamma_b|}$$

$$\text{RA}(a, c) = \sum_{b \in \Gamma_a \cup \Gamma_c} \frac{1}{|\Gamma_b|}$$

$$\text{PA}_{xy} = k_a * k_c$$

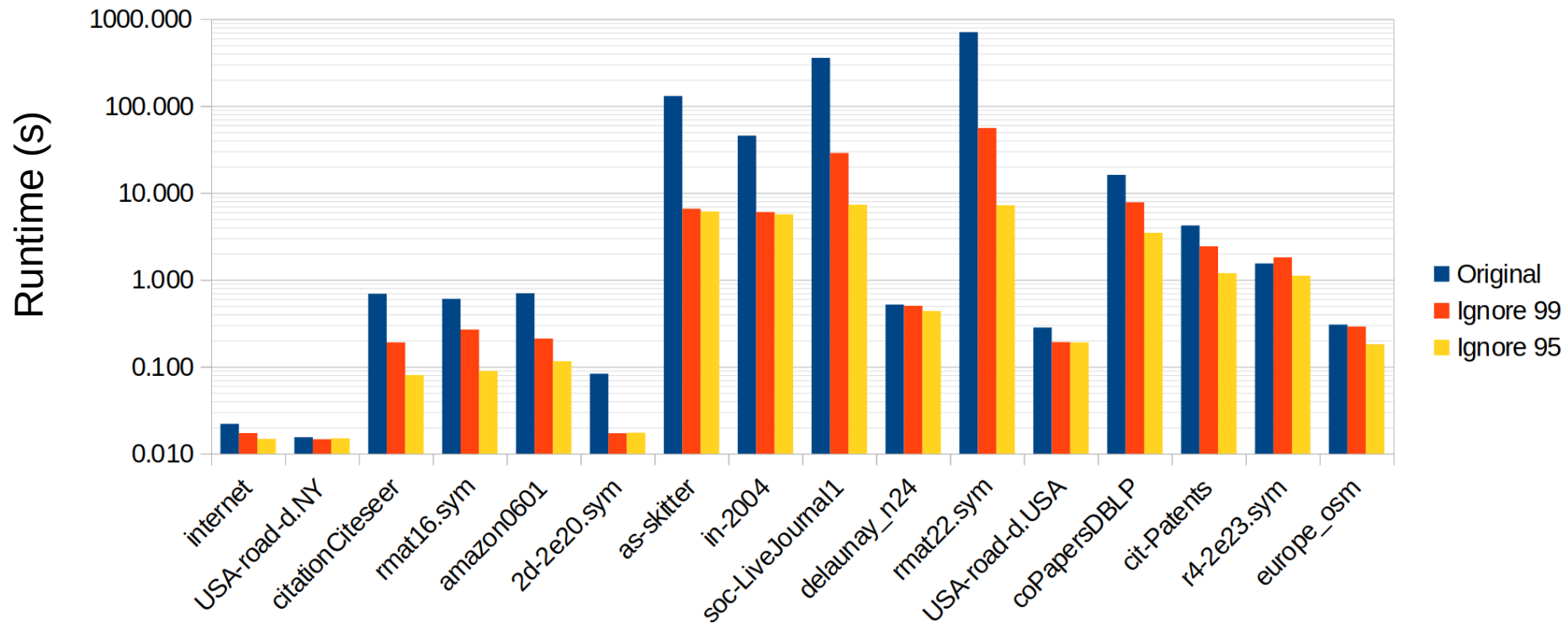
Ignoring Top Percentiles – F1 Scores

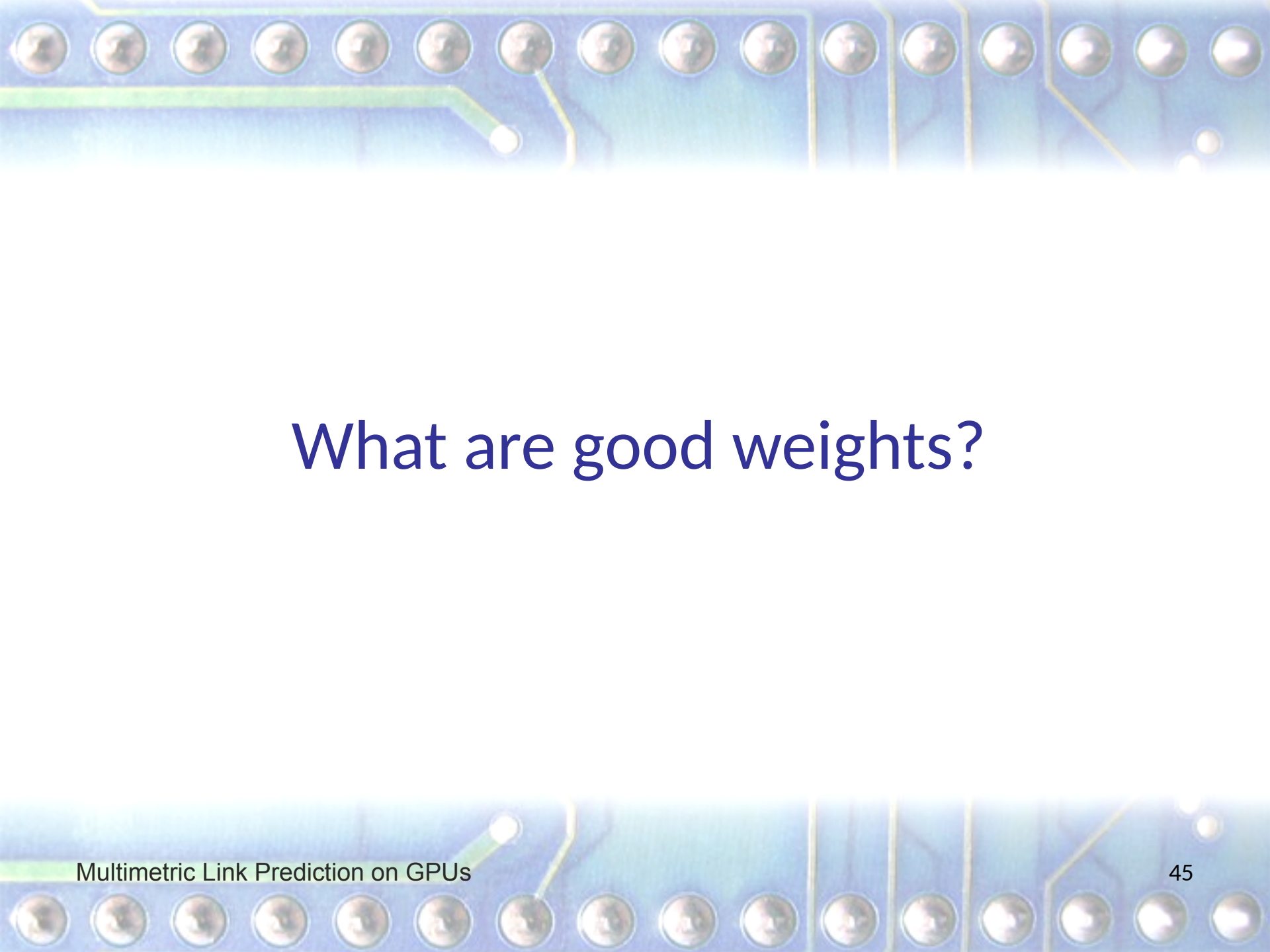
F1 Scores - Original vs. Ignore99 vs. Ignore95



Ignoring Top Percentiles – Runtime

Runtimes - Original vs. Ignore99 vs. Ignore95





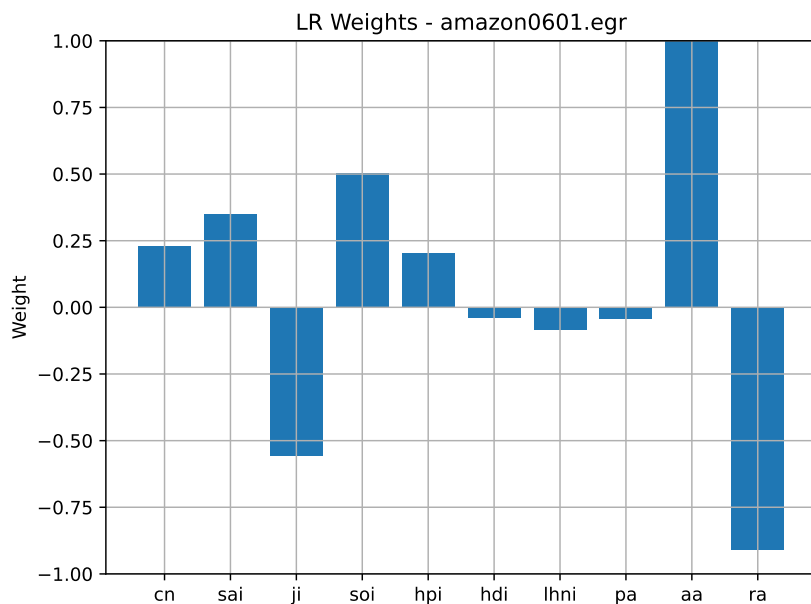
What are good weights?

Finding Best Weights – Logistic Reg.

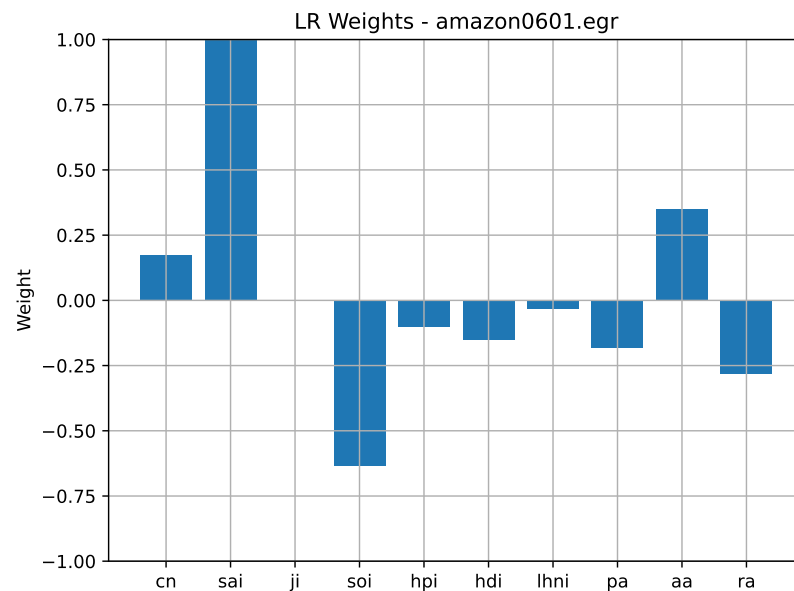
- “TopK * Factor” (**TopKF**)
 - Expand running list of edges to size $10\% * 5$
 - Run **Logistic Regression** on resulting edge list
 - Too few True Positives
- “Compute Removed Edges” (**CRE**)
 - Use **removed edges** (the **original Tp**) and randomly-generated non-Tp edges for the logistic regression
 - Balances Logistic Regression data

Finding Best Weights (cont'd)

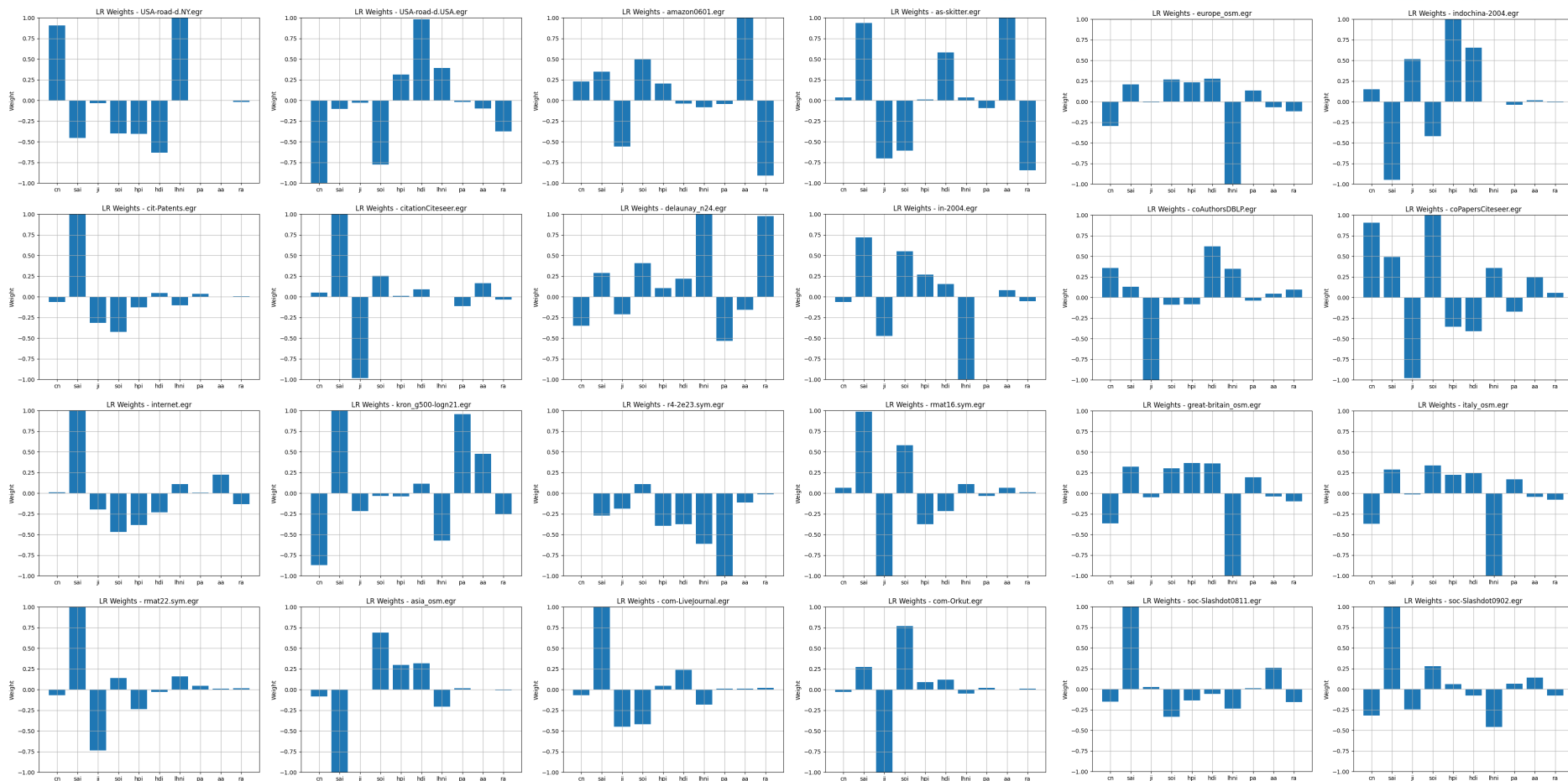
TopK * Factor



Compute Removed Edges

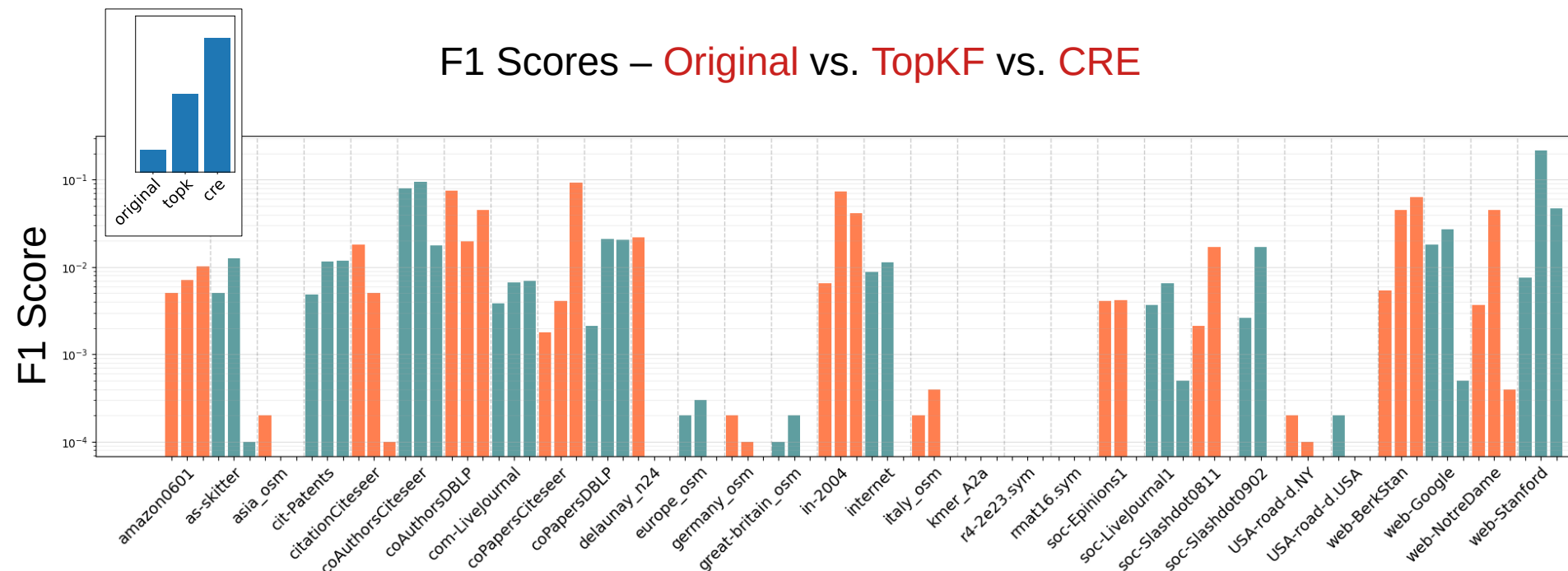


Finding Best Weights (cont'd)



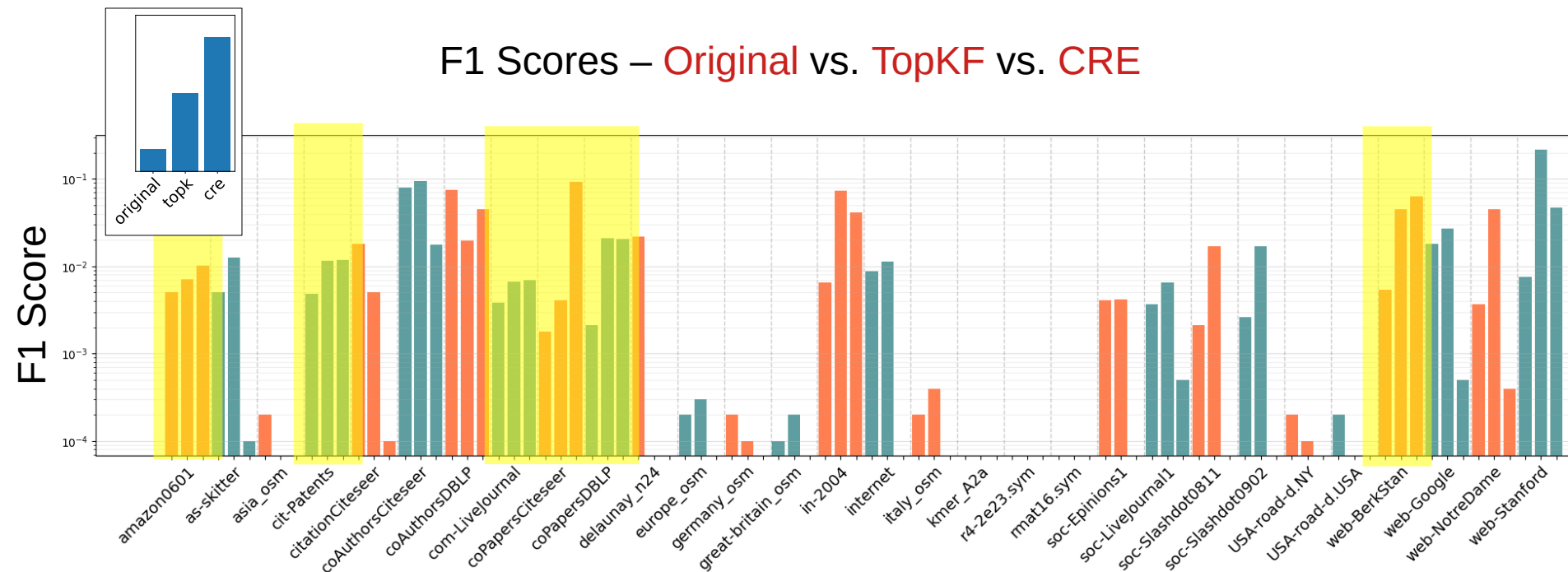
Finding Best Weights

- No apparent correlation between weights and solution quality



Finding Best Weights

- No apparent correlation between weights and solution quality

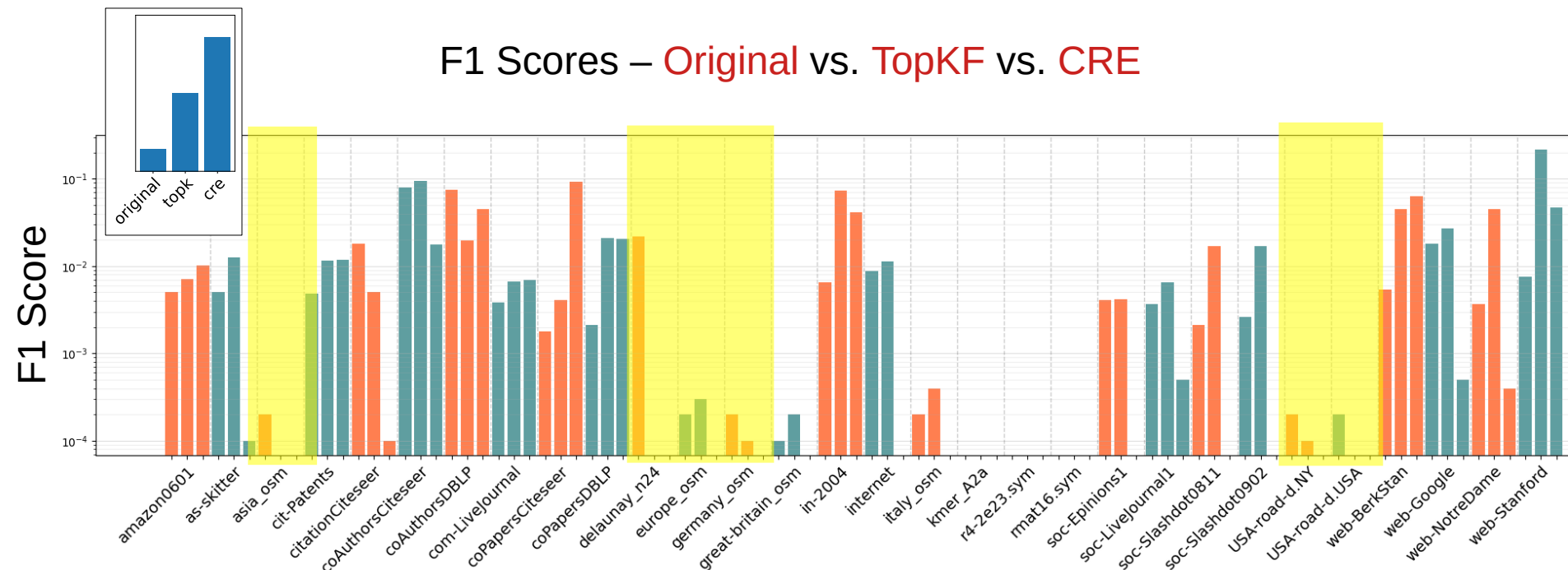


Sometimes, Logistic Regressions improve F1 Score!

Finding Best Weights

- No apparent correlation between weights and solution quality

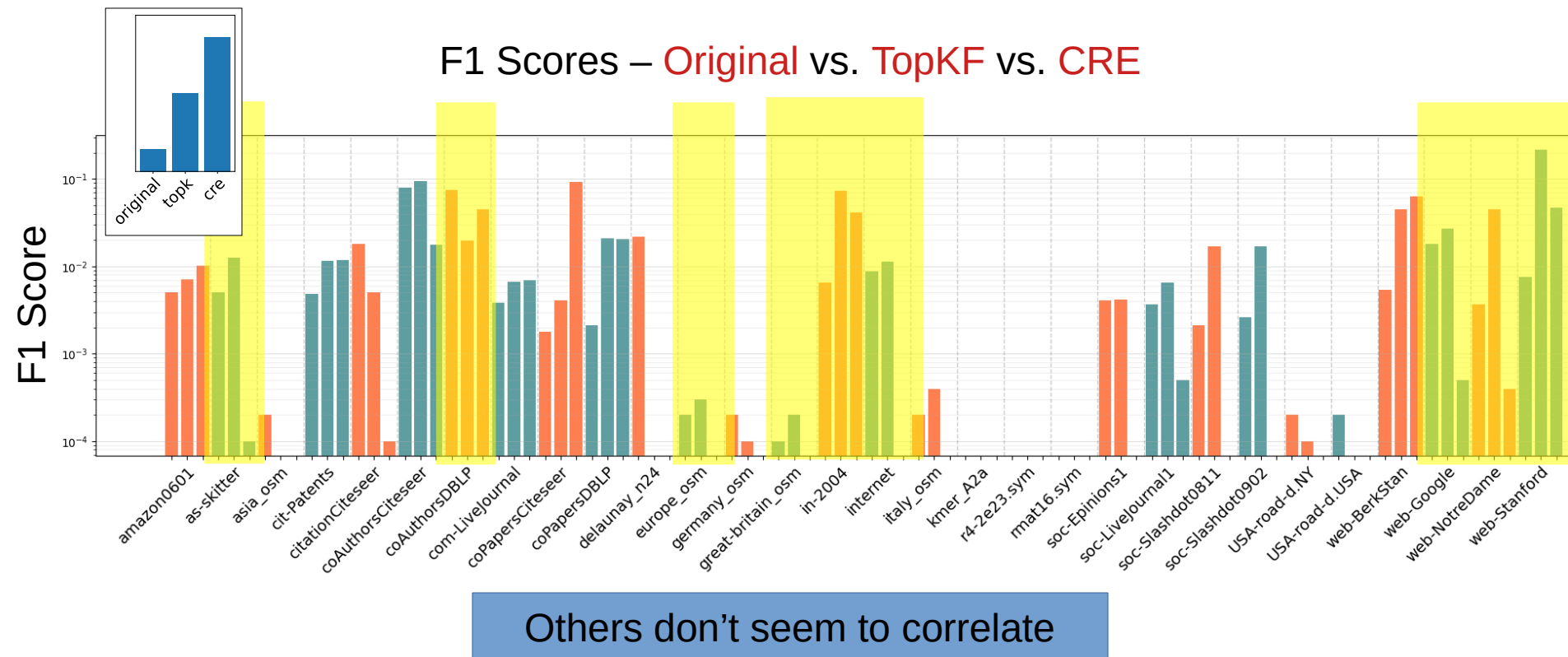
F1 Scores – Original vs. TopKF vs. CRE



Sometimes, Logistic Regressions degrade F1 Score!

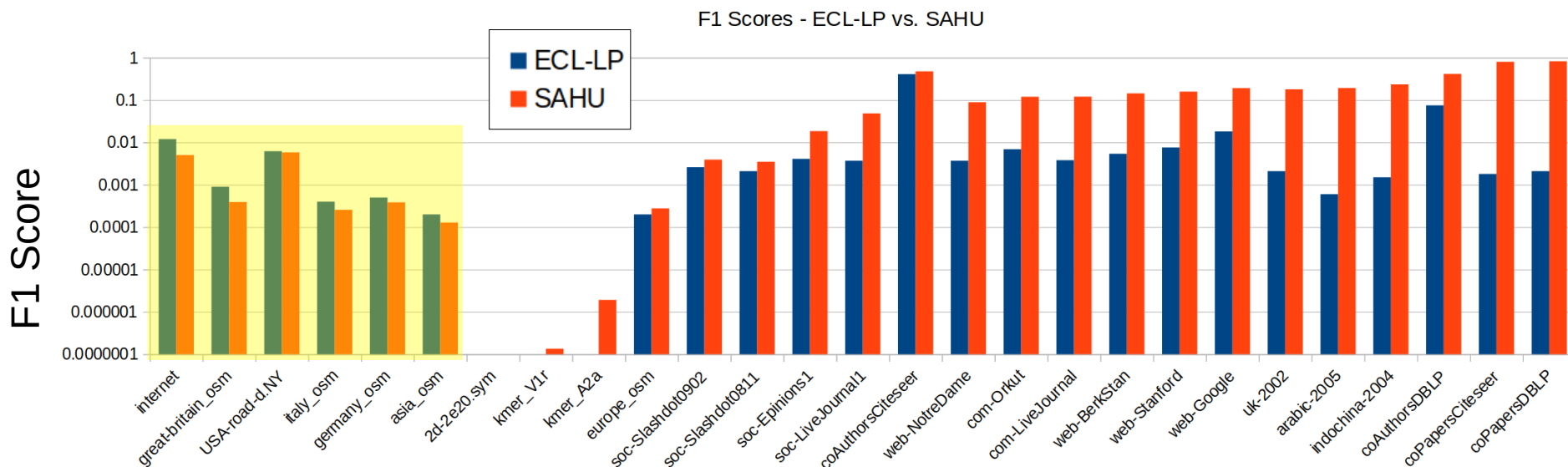
Finding Best Weights

- No apparent correlation between weights and solution quality



Comparison To 3rd Party Code - SAHU

- **Grid search** reveals there is more information from observing **all metrics simultaneously**!



Certain weight combinations score higher than single metric alone!

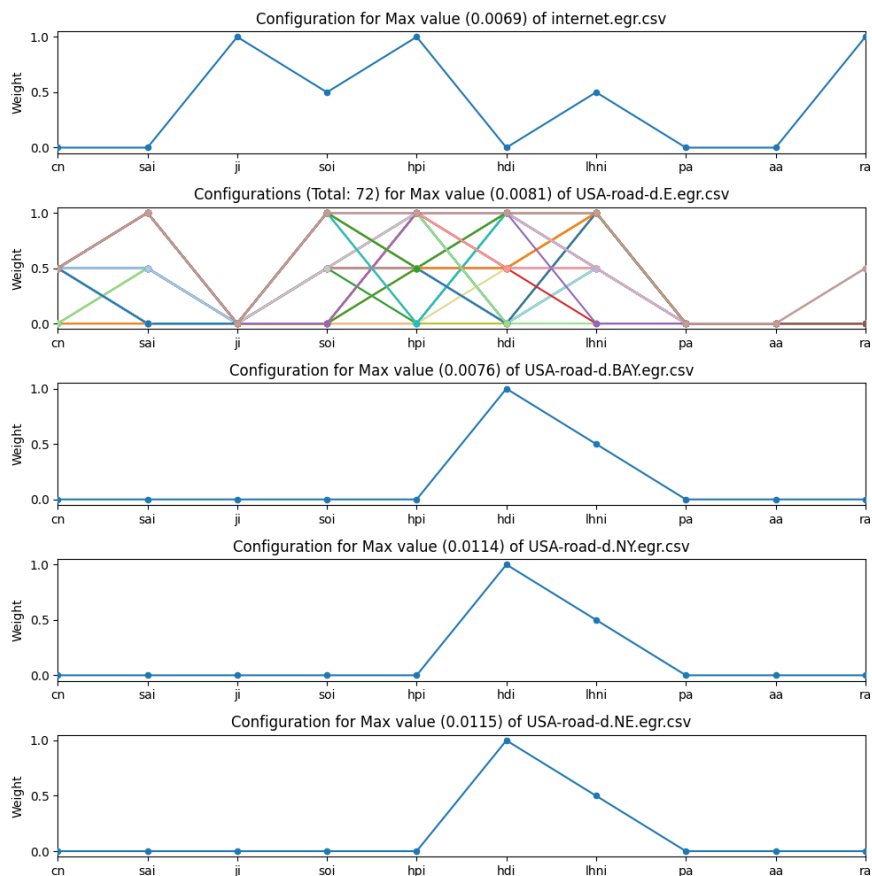
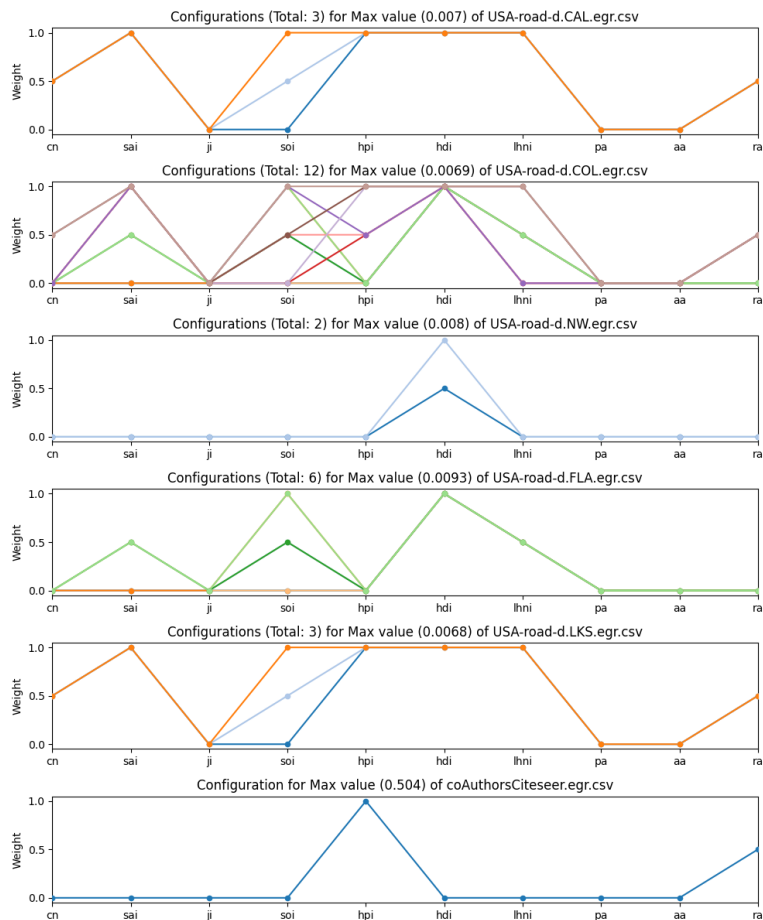
Summary

- I present **ECL-LP**, a fast parallel GPU implementation that is competitive with parallel CPU codes in terms of runtime performance
- I compute **all metrics at once**, whereas 3rd party code only computes a single metric at a time
- Looking to **find good weights!**

Grid Search for Best Weights

- Which combination of **weights-per-metric** gets the highest scoring edges?
- Combinations of **0.1, 0.5, 1.0** per metric
- Different inputs correspond to different favorable metrics

Grid Search for Best Weights (cont'd)



Valid Edges Computed

