

Multimetric Link Prediction on GPUs

CS 7387: Research in Computer Science

Andres Gonzalez*
Texas State University
andres.gonzalez@txstate.edu

May 2026

Abstract

Accurately predicting new links in graphs is an important operation, but distinguishing between likely and unlikely links is both difficult and slow. Prior work has overcome these challenges by using CPU parallelization to speed up the computation and by using topological characteristics to improve the solution quality. Many useful metrics have been proposed to estimate the likelihood of a new link, but the leading approaches only compute and consider a single metric at a time. In this study, we present ECL-LP, a parallel GPU implementation that simultaneously evaluates 10 popular similarity-based metrics used throughout the literature. It outperforms the state of the art on uniform-degree networks by an order of magnitude with respect to runtime performance and is competitive on power-law graphs. Preliminary results regarding solution quality by using a combination of metrics are promising, achieving better predictions than when considering a single metric at a time, but determining optimal weights for combining these metrics is ongoing work.

1 Introduction

The Link Prediction problem has been a topic of study for more than 20 years, but as of the early 2020s, has recently gained much traction [1, 2, 3, 4, 5]. Given a graph or a network where nodes have corresponding representations, we can represent connections, relationships, or links via edges. That is, a graph G is defined by its vertices V and edges E , such that $G = (V, E)$. Over time, social networks—that is, graphs where nodes represent individuals, and edges represent a connection between those individuals—have been extensively studied.

Link Prediction has played a central role in applications such as product and movie recommender systems. Additionally, Link Prediction has been used in graph analyses involving terrorist networks, online market transactions, and project management [5, 6]. For example, a graph can represent a collection of online shoppers and the products that have been individually-purchased by each of those customers. Using Link Prediction, a shopper can be suggested a product to buy based on the purchases of other customers via the underlying graphs of those customers. As another example, individuals in a criminal organization may be connected even though there has been no concrete evidence of their communication, and Link Prediction can be used to predict which individuals in those organizations are likely communicating. In a company or business organization, individuals in previous projects might be selected for a new collaborative project based on Link Prediction. For a scientific community, researchers in particular subject domains may eventually collaborate on a study or paper, and Link Prediction can be used to reveal those collaborations.

Many real-world networks are heterogeneous in structure and topology [7]. In any particular graph that is representative of real-world data, there tends to be a graph structure where few nodes have many connections and many nodes have few connections—this sparse form of graph is called “power-law” in nature. An example of this graph could be a social network where few individuals are followed by the masses of others, and likewise, there may be many individuals that only have a connection with a few others. Conversely, there are also many real-world graph representations that are “uniform degree”—that is, most of the nodes in the graph tend to have a near-equal degree. Road networks and protein-structure graphs usually have this topology, and correspondingly, Link Prediction can be used to predict (e.g.) new molecular or protein interactions. A graph could, therefore, be anywhere in the spectrum of power-law to uniform-degree.

Link Prediction studies have used a variety of metrics for predicting connections where there are currently none, of which the “Common Neighbor” metric has been the most widely studied [1]. The concept of the “Common Neighbor” is easily graspable when used in the context of social networks—the more common connections

*Advisor: Martin Burtcher; Texas State University; burtcher@txstate.edu

Table 1: Local Node-Scoring Metrics

Abbr.	Metric
CN	Common Neighbor
JC	Jaccard Coefficient
SOI	Sorensen Index
SC	Salton Cosine
HP	Hub Promoted
HD	Hub Depressed
LHN	Leicht-Holme-Newman
AA	Adamic-Adar
RA	Resource Allocation
PA	Preferential Attachment

any one particular individual may have with another may, the more likely it is that those two individuals (that are not currently connected) are actually connected in some way (e.g., perhaps they just have not met via their social circles of interactions with other individuals).

A recent survey of Link Prediction methods found more than 400 articles published between 2021 and 2023 [3, 5, 7, 8, 9]. A handful of general approaches have been the focus of graph analyses:

- Similarity Metric-based approaches (e.g., “common neighbor”)
- Matrix-based approaches (e.g., eigenvector and dimensionality-reduction approaches)
- Machine Learning-based approaches (e.g., GNNs with node and edge feature analyses)

Matrix-based factorization methods that aim to reduce dimensionality by embeddings is intractable for large graphs, and machine learning approaches suffer from lack of quality datasets [3]. Popular datasets for the topic of Link Prediction have normally focused on networks with node counts in the thousands [8]. However, many network types (including and especially social networks) contain nodes in the millions with edge-counts exceeding the billions. Much of the struggle with regard to large and complex graphs merely stems from the simple fact that matrix-based approaches are intractable for these large graphs [5]. For example, a graph with 1,000,000 nodes results in an n^2 matrix with 1,000,000,000,000 entries. If each entry is represented with a 4-byte integer, this makes the application of matrix-based codes difficult or impossible for a typical computer, where the hardware configuration may perhaps only support up to 256 GB of DRAM.

Additionally, many real-world-representative networks tend to be sparse (as opposed to dense), where there are a low-percentage of entries relative to the entire possible connectivity of the graph. For example, the “eu-rope.osm” graph contains 50,912,018 nodes, but only contains 108,109,320 non-zeros, giving a density measure of 0.000004%—for reference, the highest density of the graphs in this study is 0.0451%. This roughly corresponds to an edge-prediction pool of $(50,000,000 * 50,000,000) - 100,000,000 = 2,499,999,900,000,000$, a seemingly impossibly-large candidate list of edges to correctly-predict from. Surprisingly, the likelihood for correctly-predicting an edge significantly increases via the use of the aforementioned similarity metric-based approaches. In this work, we focus on the Similarity Metric based approach using GPUs.

2 Similarity Metrics

Listed below are each of the metrics used to score edges in this study, as they are frequently used in the literature [6, 8, 10]. The corresponding abbreviations for each metric is listed in Table 1.

In the following notations, “gamma” with any subscript (e.g., the symbol “ Γ_x ”) denotes “the neighbors of vertex x ”. For example, in looking at the graph in Figure 1, the notation “ Γ_0 ” would refer to the list of nodes 1, 3, 4, and 5. Furthermore, the vertical bars around the neighbor-notation (e.g., “ $|\Gamma_x|$ ”) denote “the magnitude of”, and to continue on the earlier notation for the example graph, the notation “ $|\Gamma_0|$ ” would be equivalent to the value 4. The notation is generalized to any vertex by means of lowercase letters as in “ Γ_a ” or “ $|\Gamma_a|$ ” to denote “the neighbors of a ” or “the number of neighbors of a ” (i.e., the degree), respectively.

Additionally, the $\cap$ symbol denotes “the set intersection between the listed vertices’ neighbor lists”. For example, the equation “ $\Gamma_0 \cap \Gamma_8$ ” denotes the set intersection of the neighbor lists of both vertices 0 and 8. If a graph had vertices 0 and 8, where $\Gamma_0 = \{1, 2, 3, 4, 5\}$ and $\Gamma_8 = \{3, 4, 5, 6, 7\}$, then $\Gamma_0 \cap \Gamma_8 = \{3, 4, 5\}$ and $|\Gamma_0 \cap \Gamma_8| = 3$.

Conversely, the $\cup$ symbol denotes the set union. For example, if $\Gamma_0 = \{1, 2, 3, 4, 5\}$ and $\Gamma_8 = \{3, 4, 5, 6, 7\}$, then $\Gamma_0 \cup \Gamma_8 = \{1, 2, 3, 4, 5, 6, 7\}$ and $|\Gamma_0 \cap \Gamma_8| = 7$.

The “Common Neighbor” metric is the simplest, taking a tally of how many neighbors a pair of vertices have in common [11].

$$\text{CN}(a, c) = |\Gamma_a \cap \Gamma_c| \quad (1)$$

The “Jaccard Coefficient” metric extends the “Common Neighbor” metric by giving priority to pairs of vertices that have a higher ratio of common neighbors [12].

$$\text{JC}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a \cup \Gamma_c|} \quad (2)$$

The “Sorensen Index” metric, unlike the Jaccard Coefficient, only regards lower number of neighbors as more important, giving a higher priority to edges whose vertices contain low degrees [13].

$$\text{SI}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a| + |\Gamma_c|} \quad (3)$$

The “Salton Cosine” metric takes a measure of the angle between the two vectors that describe the neighbor lists of the given vertices [14].

$$\text{SC}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\sqrt{|\Gamma_a| \cdot |\Gamma_c|}} \quad (4)$$

Similar to graph node centrality measures such as PageRank that incorporate choosing neighbors by random chance, the “Leicht-Holme-Newman” metric gives priority to pairs of vertices whose number of common neighbors are higher than would be expected by random chance [15].

$$\text{LHN}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{|\Gamma_a| \cdot |\Gamma_c|} \quad (5)$$

The “Hub Promoted” metric takes the stance of thinking that potential for new links increases when either vertex in the vertex-pair of an edge have fewer connections (regardless of the number of edges already established by the other vertex) [16].

$$\text{HP}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\min(|\Gamma_a|, |\Gamma_c|)} \quad (6)$$

The “Hub Depressed” metric takes the opposite stance of that of the “Hub Promoted” metric wherein any edge between a pair of vertices is less likely to be a candidate edge if either of its vertices already has a high degree [16].

$$\text{HD}(a, c) = \frac{|\Gamma_a \cap \Gamma_c|}{\max(|\Gamma_a|, |\Gamma_c|)} \quad (7)$$

Unlike the similarity measures that incorporate common neighbors, the “Resource Allocation”, “Adamic-Adar”, and “Preferential Attachment” metrics incorporate a measure of the neighbor list of the wedge-central node. The *AA* metric lessens the effect of central nodes whose degrees are high, whereas *RA* metric maintains a proportional relationship to the number of neighbors in that central node, and the *PA* metric plays a role in the growth of a network, making nodes that establish new connections more favorable for establishing even more connections [17, 18, 19]. In the notation below, k denotes “the degree of”, as in “the degree of node a ” for k_a .

$$\text{AA}(a, c) = \sum_{b \in \Gamma_a \cup \Gamma_c} \frac{1}{\log |\Gamma_b|} \quad (8)$$

$$\text{RA}(a, c) = \sum_{b \in \Gamma_a \cup \Gamma_c} \frac{1}{|\Gamma_b|} \quad (9)$$

$$\text{PA}_{ac} = |\Gamma_a| \cdot |\Gamma_c| = k_a \cdot k_c \quad (10)$$

3 Related Work

Previous research has been carried out on the CPU with parallelization using OpenMP by Sahu and Kothapalli using the same collection of similarity-metrics presented here with the exception of the “Preferential Attachment” metric [10]. Their implementation aims to predict links based off of a single metric (rather than the 10 metrics simultaneously as has been done in this study), and their codes can be found on github¹. Their approach utilizes a priority-queue-per-thread implementation. The traversal of the edges-to-be-computed is carried out in parallel, where each thread keeps a priority-queue of the top-scoring edges along the traversal of the thread’s chunk of edges. The authors acknowledge that due to the parallel nature of the implementation, and for the potential of edges to score the same given their single-metric evaluation, their resulting implementation is nondeterministic. Additionally, their implementation only aims to predict True Positives. In other words, their implementation does not entirely disqualify edges from being predicted as True Positives; they do not aim to predict True Negatives. In this study, we evaluate the graphs analyzed by Sahu and Kothapalli in addition to a collection of others that together represent a diverse set of graph characteristics (see Table 3).

Yin et al. introduce Super Resolution Graph “SRG”, a deep generative model for predicting the evolution of social-network temporal graphs—the points at which new links are established and the points at which links are broken [9]. The analogy for preserving link-formation or link-destroying across a model is inspired by the idea of taking a low resolution image and using a model to generate a higher resolution image. In terms of an evolving social network, a collection of (“low resolution”) graphs at different timestamps in the social network are used as the analogous “low resolution images” in order to construct a “higher resolution graph” that more effectively represents the evolving nature of the network, using a conditional Normalizing Flow to capture the graph’s dynamic transmission pattern over time. However, it is important to note that this approach is applied to denser graphs of under 1,000 vertices. Their code is available on github².

Belth et al. introduce *LinkWaldo*, a model for predicting links between node pairs given proximity and Stochastic Block resemblance models [3]. Naturally, the search space for finding new links is quadratic with node count in graphs. Similarity-based approaches are limited to a proximity of 2 nodes and normally result in too small a dataset of True Positives in order to effectively carry out machine learning approaches—*LinkWaldo* aims to resolve this proximity and ground truth imbalance problem and is able to further predict edges from nodes that are farther away with a focus on high recall by leveraging node-pair link resemblance. Their dual model based approach is evaluated on graphs with node counts below 100k (except for their “Digg” benchmark of 300k nodes). Their code is available on github³.

Wang et al. use a machine learning approach to avoid tying the learned edge-types from heterogeneous networks against the prediction of edges in the general case, a problem that the authors describe as “edge disturbance” [7]. Their disturbance-resilient prediction method “DRPM” constructs individual edge-type experts that then aggregate their data for a general predictor via 3 stages: (i) a structural characterizer that embeds the heterogeneous network into a low-dimensional representation, (ii) a type differentiator where the edge-type experts maintain their learned features, and (iii) a resilient-predictor which aggregates the model’s previous knowledge to formulate a prediction. Like previously-mentioned machine learning methods, the authors’ experiments evaluate their model in graphs of less than 40k nodes, but their model performs better than a handful of other state-of-the-art machine learning approaches. Wang et al. study a similar approach to edge-type discrimination [4].

Saifi et al. use a similarity-based approach much like this study, however unlike this study’s implementation, they also incorporate similarity-metrics based on paths [5]. Additionally, due to their usage of path metrics, their evaluations are limited to graphs’ individual connected-components. It is important to note that their evaluations use datasets that have less than 10k nodes except for two instances, where the nodes for those graphs are less than 20k. Through parallelization and various algorithmic optimizations, they show that a graph decomposition, where every connected component can be evaluated by a separate processor, can speed up computations.

Hu et al. focus on Link Prediction via Cross-Domain Collaboration Prediction “CDCP” [6]. Through meta-path sampling, multi-task learning, and a semantic encoding method for representing researchers’ motivations for collaborations, the authors show that their model can surpass predictive capabilities of other approaches based on mechanisms such as (e.g.) random walks and graph neural networks. They also analyze variations on the inclusion of their semantic encoding and metapath sampling methods on heterogeneous networks of less than 30k nodes.

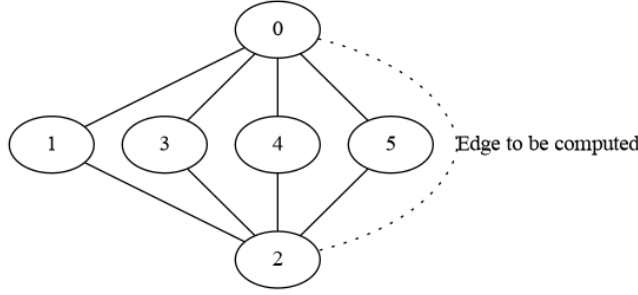
Unlike the majority of other works that have focused on machine-learning and matrix-based methods, this

¹<https://github.com/puzzlef/neighborhood-link-prediction-openmp>

²<https://github.com/yinyanting123/SRG>

³<https://github.com/GemsLab/LinkWaldo>

Figure 1: Example Graph 1



study focuses on the similarity-metric edge-scoring approach on GPUs using their massive parallelism capabilities. Similar to Sahu and Kothapalli, every edge that needs to be computed (i.e., node pairs that are a 2-hop distance that do not currently have an edge) has a collection of metrics computed simultaneously, and the resulting score is used to rank the top- k edges. This set of predicted “most-likely” candidates then forms the resulting list of edges that would correspond to True Positives (should they exist in the set of ground truth edges). Furthermore, the implementation in this study aims to associate each metric with a given weight relative to the other metrics, effectively focusing attention on characteristics that are favorable for different types of inputs. In other words, roadmap networks may tend towards a specific metric that may be different from the metrics favored by social networks. Although finding optimal weights is ongoing research, preliminary results have shown that, indeed, there is benefit to taking into account different metrics at once. That is, solutions of higher-quality can be found when using more than a single-metric (see Section 9). Further still, it is also important to note that the implementation presented is analyzed with graphs whose node counts are in the hundreds of millions and whose edge counts surpass the billions (see Table 3). This is a stark contrast to previous studies that focus mainly on graphs whose node counts are fewer than numbers ranging in the tens of thousands.

4 Approach

4.1 Exhaustive Brute-Force

ECL-LP-CPU-BF The naïve Brute-Force CPU implementation introduces a baseline. However, it is immediately clear that it has extreme limitations. The idea of the exhaustive brute force is straightforward: for every edge that does not exist between two nodes, compute the metric scores and keep the top-scoring edges. Being that this approach is $O(N^2)$, this solution is especially intractable for large graphs, with a graph of few nodes taking an unacceptable amount of time to compute. Hence, the optimizations listed below are necessarily incorporated into the algorithm used in this study.

4.2 Optimizations

4.2.1 2-Hop DFS

The literature incorporates a simple approach for limiting the number of edges being computed for any input, where the computation of edge scores is limited to nodes whose distance are length of 2 [10]. Since the majority of the metrics computed incorporate a measure of “common neighbors”—that is, the set intersection of the neighbor lists of the vertices in question—there would not be a need (at least in an initial iteration) to compute edges of nodes whose distance is, e.g., 5, because these two nodes naturally will not have any common neighbors. This optimization alone rids the algorithm of computing edges whose scores would cause their rejection from the list of top-scoring edges.

4.2.2 Eliminating Duplicate Computations

An insightful optimization with regard to filtering unnecessary edge score computation is the notion that a currently-nonexisting edge could be attribute to several wedge-central vertices. For example, in a simple graph with 6 nodes, a 2-hop DFS from node 0 to 2 could perhaps occur through the wedges formed via the nodes 1, 3, 4, and 5, and this in turn means that the edge between 0 and 2 would be computed 4 times (see Figure 1). The insight here is that in order to compute this edge from 0 to 2 only once, we should take into account only the vertex that is the *first* common neighbor between both nodes (in the case of Figure 1, this would be vertex

ID 1). At runtime, this means that the edge from vertex 0 to vertex 2 induced between node triples $(0 - 3 - 2)$, $(0 - 4 - 2)$, and $(0 - 5 - 2)$ would all be discarded, leaving only the one computation needed for scoring the edge.

4.2.3 Chunked Edge Processing with Minimum Edge Score

Due to the size of some of the larger graphs, whose node counts are well into the millions with a large average degree, being able to compute all edges requires only computing a certain number of edges at a time. In addition, this means that an ongoing list of top-scoring edges must be maintained. Furthermore, a minimum edge score can also be maintained along with the list of top-scoring edges, such that any edges that does not breach the minimum score as part of its computation can be safely discarded as there are more edges whose scores are higher and fulfill the resulting 10% edge list.

4.2.4 Simply Ignoring Vertices

A strategy also incorporated in the literature is that of ignoring vertices whose degree is above a certain threshold [10]. The rationale behind ignoring stems back to an idea from social networks—a user that has many followers (otherwise known as a “hub”) is far less likely to establish meaningful relationships with other users than a user with only a few connections. Therefore, it is more likely that we can predict an edge between the user with fewer links and another such user. This strategy also aids in removing edges from the combinatorial explosion that comes from the upper-triangular matrix of neighbor lists that result from each node. Disregarding these large “hubs” results in faster runtimes at little expense to the solution qualities obtained. Furthermore, this strategy can be extended to ignore the top-percentile of vertices with respect to degree—that is, we can safely ignore the vertices with the most connections with little danger of losing significant solution quality.

5 The Algorithm

In short, the resulting algorithm can be outlined as follows

1. Keep a running list of “top-k” edges that would ultimately replace the 10% that were initially removed from the static graph.
2. For all edges that need to be computed:
 - 2.1 Process a chunk of edges, keeping only edges that score above the top-k’s current minimum score
 - 2.2 Sort that chunk’s list of edges by each edge’s total score from each metric
 - 2.3 Merge that sorted list with the ongoing list of top-k edges
 - 2.4 Copy back “k” edges to the running list
 - 2.5 Update the minimum score
3. Compare the resulting top-k list of edges with the original graph and compute F1 score

6 Computing Better Weights

Due to our algorithm computing several metrics simultaneously, the algorithm has a parameterized approach to optionally give each metric a weight so as to vary which metrics dominated inputs for any given run. The literature describes some metrics as performing better than others for graphs of specific types (e.g., road versus social networks), but our investigation pose the additional question of “Is there benefit to considering multiple metrics simultaneously”? In this regard, we can combine metrics in any particular combination by attributing each metric with a weight. This also poses the question of “What is a good combination of weights?” for any given input. We describe two methods for attempting to find good weights below.

6.1 Top-K * Factor (“TopKF”)

In this approach, we take the weight-neutral computation for each edge in terms of every metric, and we compute a factor of the 10% of edges that were removed (in the case of results presented here, the factor was 5). This gives a certain number of edges for which we can then determine as True Positives or False Positives and feed the metrics-and-classifications into a Logistic Regression model, the hope being that True Positives will steer the weights towards favorable configurations for the given input.

6.2 Compute Removed Edges (“CRE”)

In this alternative approach, we take the weight-neutral computation for each removed-edge in terms of every metric (giving the original “True Positive dataset” of the edges we are attempting to predict), and this is combined with an equal number of randomly-generated False Positives for the same input—the metrics for these False Positives are also computed. This approach attempts to balance the ratio of True-Positives-to-False-Positives for the Logistic Regression’s dataset for finding optimal weights. As described above, the “Top-K * Factor” approach suffers from a large imbalance of few True Positives and many False Positives, and this approach aims to remedy that imbalance.

6.3 Grid Search

As an additional approach of finding optimal weights, the fast runtimes of some inputs with regard to the algorithm presented in this study allowed for a grid search of weight-attribution and, thusly, best-F1 score. Each weight was given a possible value of 0.0, 0.5, or 1.0, and the corresponding inputs were combinatorially run with all possible weight-combinations of these values. The larger inputs do not run in a sufficient amount of time to allow for this grid search.

7 Experimental Methodology

Austin Our test system “Austin” consisted of dual Intel Xeon Gold 6226R processors with 16 cores operating at 2.9 GHz, using gcc 12.3.1 and running the Fedora Linux 37 operating system. The node was also equipped with an NVIDIA RTX 3090 with 24 GB memory, an NVIDIA A100 with 40 GB memory, and used nvcc 12.0. Other GPUs also used in this study are listed in Table 2.

Table 2: GPU Specifications

	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

8 Benchmarks

The benchmarks used in this study are listed in Table 3 in three groups. The first group denotes a standard set of benchmarks used by ECL for characterization of algorithms with respect to varying graph structure. The second group is the list of graphs from Sahu and Kothapalli, and the third is from the same Sahu work’s currently-listed inputs from their github repository⁴ [10].

8.1 Generating Inputs

When considering graphs that represent social networks, newly-formed links are the result of established relationships over time. One could take a snapshot of a social network at a particular timestamp, and some time later, take another timestamped snapshot, and perform link prediction on the earlier graph to subsequently compare against the latter. When it comes to static graphs, a number of edges are removed randomly, such that the process of Link Prediction involves predicting those removed links. A good measure of removing links has been established as 10% of the static graph’s existing edges, which allows the network to maintain structure without degrading context [3, 20].

9 Results

It is important to remember that the implementation reflects two metrics in terms of performance for which we aim to improve: (1) runtime—that is, how quickly we can arrive at a result—and (2) F1 Score—that is, the result’s solution quality with regard to how many edges that were predicted were actual True Positives. Results

⁴<https://github.com/puzzlef/neighborhood-link-prediction-openmp>

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
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
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

Table 3: Benchmarks

Figure 2: Runtime Comparison of kmer_A2a to Sahu

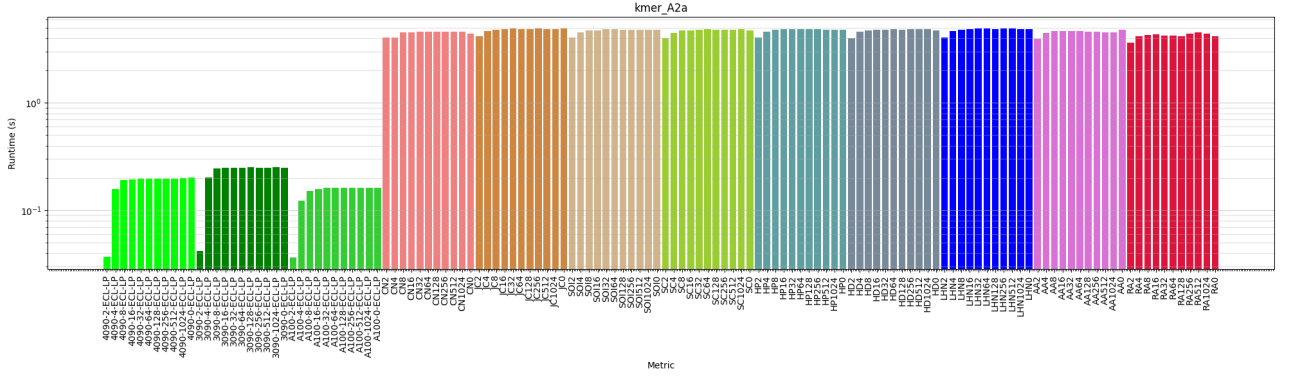
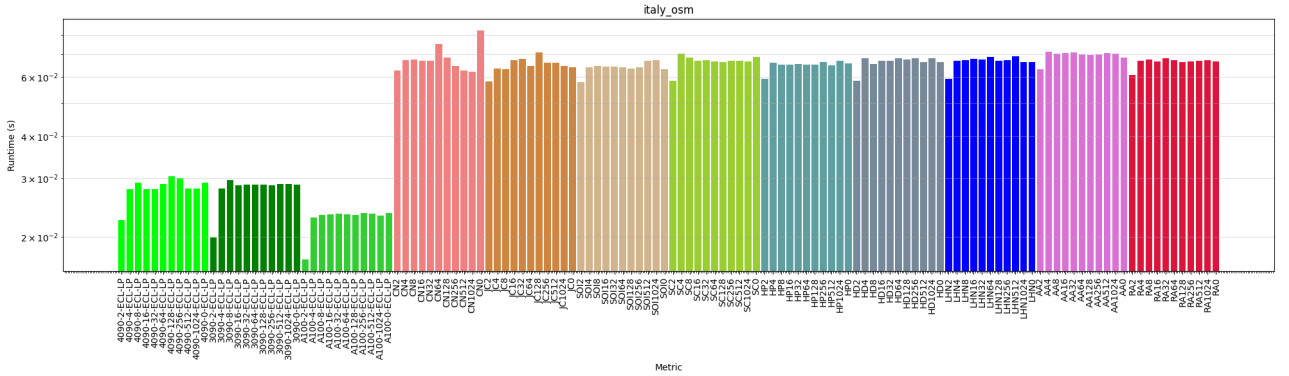


Figure 3: Runtime Comparison of italy_osm to Sahu



for runtime performance are presented below. The Appendix in section 12 contains additional figures for inputs concerning ECL-LP’s runtime behavior against third-party code.

9.1 Runtime Performance

On *uniform degree* inputs, ECL-LP consistently outperforms the leading CPU OpenMP implementation in terms of runtime [10]. The “kmer_A2a” and “italy_osm” inputs are shown in Figures 2 and 3, respectively. Each of these figures depicts GPU runtimes alongside the time achieved for the OpenMP CPU implementation, which only computes the single metric per run. The 4090, 3090, and A100 all achieved faster runtimes than that of the CPU (denoted by the far left columns of each figure). Each GPU is grouped in powers of 2 by the lowest vertex-degree cutoff on the left, beginning with 2^1 and ending with 2^{10} , with one additional column depicting the runtime without a vertex degree cutoff (e.g., “4090-0”). The chart then shows the same depictions of cutoffs for each single metric as computed by the cutoff, where each abbreviation stands for each metric as depicted in Table 1. As shown in each figure, the GPU implementation for each respective cutoff is over an order of magnitude faster than the respective 3rd party code. The plateau of the no-increase-in-runtime-performance is expected due to the structure of the input, where the open street map nodes or protein structure nodes typically have an average degree of slightly above 2. In other words, there is no difference between considering a vertex-degree cutoff of 4 against a vertex-degree cutoff of 8 due to the majority of nodes having a maximum of only 3 neighbors.

As graph structure begins to move toward the *power-law* end of the vertex-degree spectrum, where vertex average-degrees begin to rise, runtime performance begins to vary. As cutoffs get lower, runtime performance naturally gets faster, and this “downward curve” can be seen in each of “web-Google” and “coPapersCiteseer”’s runs (see Figures 4 and 5, respectively). The “web-Google” graph experiences a larger jump from no-cutoff to a cutoff of 1024 due to the max degree of the graph (6332) when compared to the max degree of “coPapersCiteseer” (1188). However, their runtime-trends follow a similar behavior. Graph “coPapersCiteseer” has an average degree of under 74, which is where the plateau begins for no-further-improvement, and graph “web-Google”’s average degree is under 10, which is why no further improvement in runtime performance can be seen for cutoffs of 8 and below. Interestingly, “web-Google”’s runtime performance is consistently faster than the CPU code,

Figure 4: Runtime Comparison of web-Google to Sahu

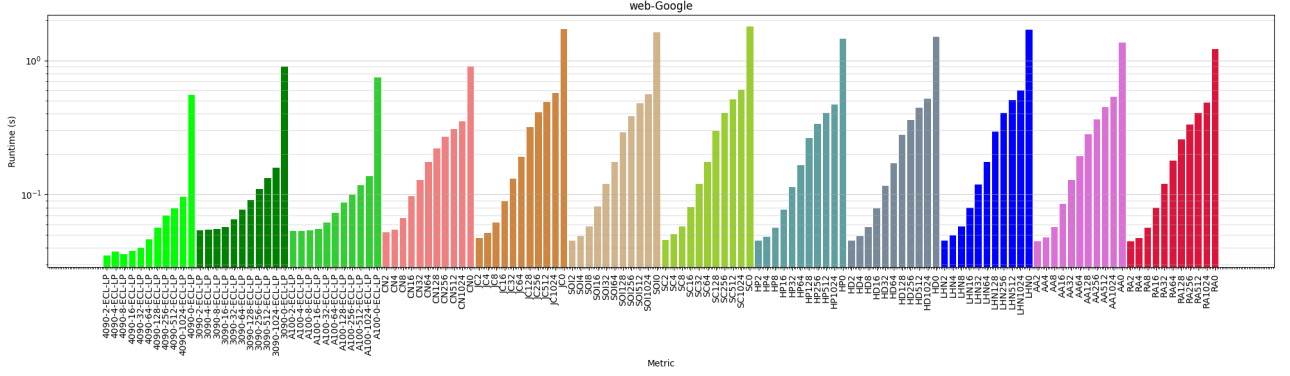
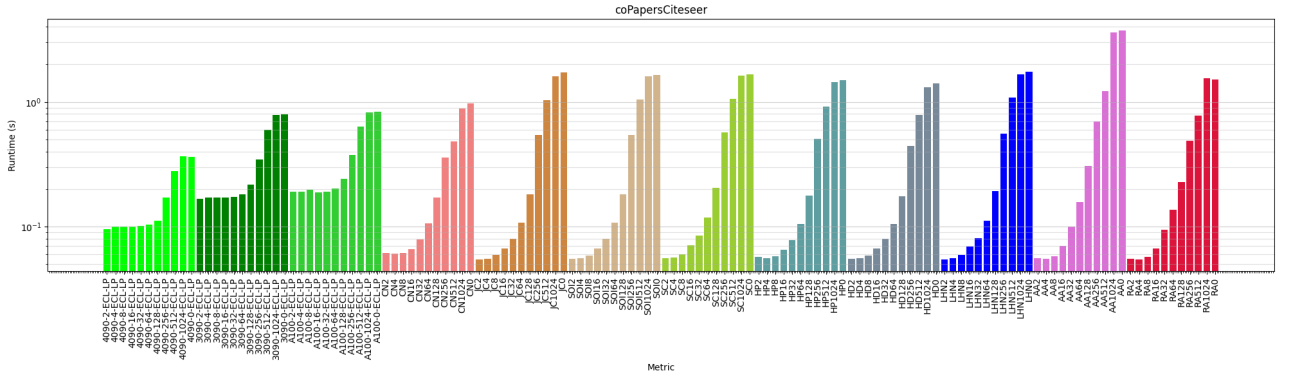


Figure 5: Runtime Comparison of coPapersCiteseer to Sahu



but this is not the case for “coPapersCiteseer”, where the GPU code is faster without a cutoff but is slower with the cutoffs.

For large power-law graphs, the GPU implementation does not perform as well as the CPU implementation (see Figures 6 and 7), and we believe this is due to the structure of the algorithm. In order to determine whether an edge needs to be computed, the edge’s endpoint vertices first have to be derived from the neighbor lists of vertices with the currently-existing edges, and this is only known at runtime due to the large size of the inputs. Every 2-hop DFS is generated by each vertex’s neighbor list, giving $((|N_a|)(|N_a| - 1))/2$ combinations per vertex. Once the non-existing edges’ endpoints are obtained via these combinations, we then have to determine whether the edge should be filtered according to the neighbor-list intersection “first common neighbor” approach and according to whether there already is an edge between the vertices in question. This computation causes overhead for which eventually the work is discarded due to the filtering, and this trend in “amount of valid work” can be observed in Figure 8. The larger, power-law graphs tend towards the left, where many edges are filtered and, thusly, a low percentage of “valid” work is computed. Naturally, the uniform-degree inputs tend towards the right due to the graph not having many overlaps in edges to consider.

An interesting visualization of the runtime benefits are the speed up curves generated by each cutoff from 2^1 to 2^{10} , as seen in Figure 9. The topology of each graph becomes even more apparent as the curves differ in when they benefit from specific thresholds for degree cutoffs, especially with regard to the average degree of each graph. Towards the lower right of the visualization are the curves that are more uniform degree, with some inputs such as “USA-road-d.NY” and “europe.osm” not experiencing speedups until the degree cutoff is 2. Others, such as “soc-LiveJournal1” and “rmat22.sym” immediately stand to benefit from ignoring vertices above degree 1024. Graphs that sit in between the extremities of uniform-degree and power-law exhibit curves in the middle such as “amazon0601”.

An unexpected result was exhibited by a small set of the larger inputs, where the amount of memory allocated for the chunk size was varied between 5% and 100%. This is shown in Figure 10. Given the longest runtime with only 5% of memory allocated for edge processing in chunks, allocating 100% of memory for chunked edge-processing only exhibited a 5% speedup, indicating that the optimized approach is compute-bound.

Figure 6: Runtime Comparison of web-Stanford to Sahu

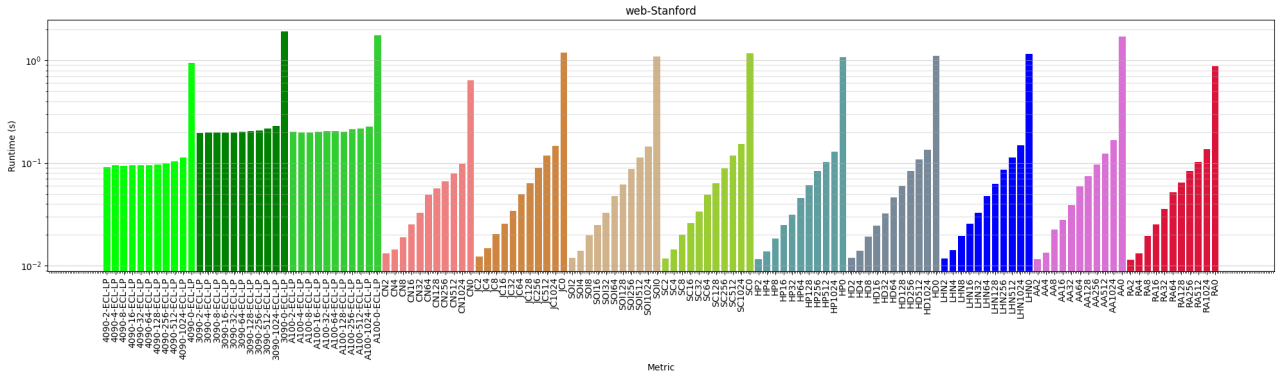


Figure 7: Runtime Comparison of arabic-2005 to Sahu

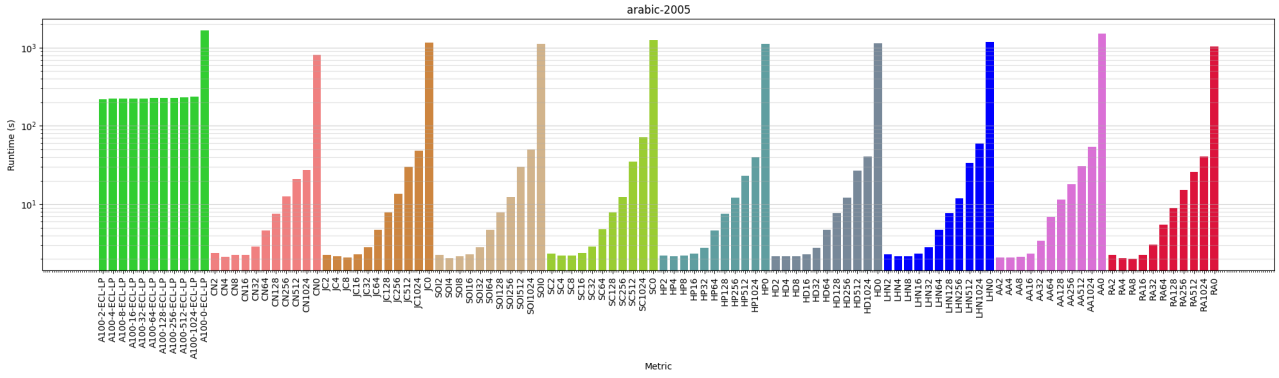


Figure 8: Valid vs. Invalid Computed Edges

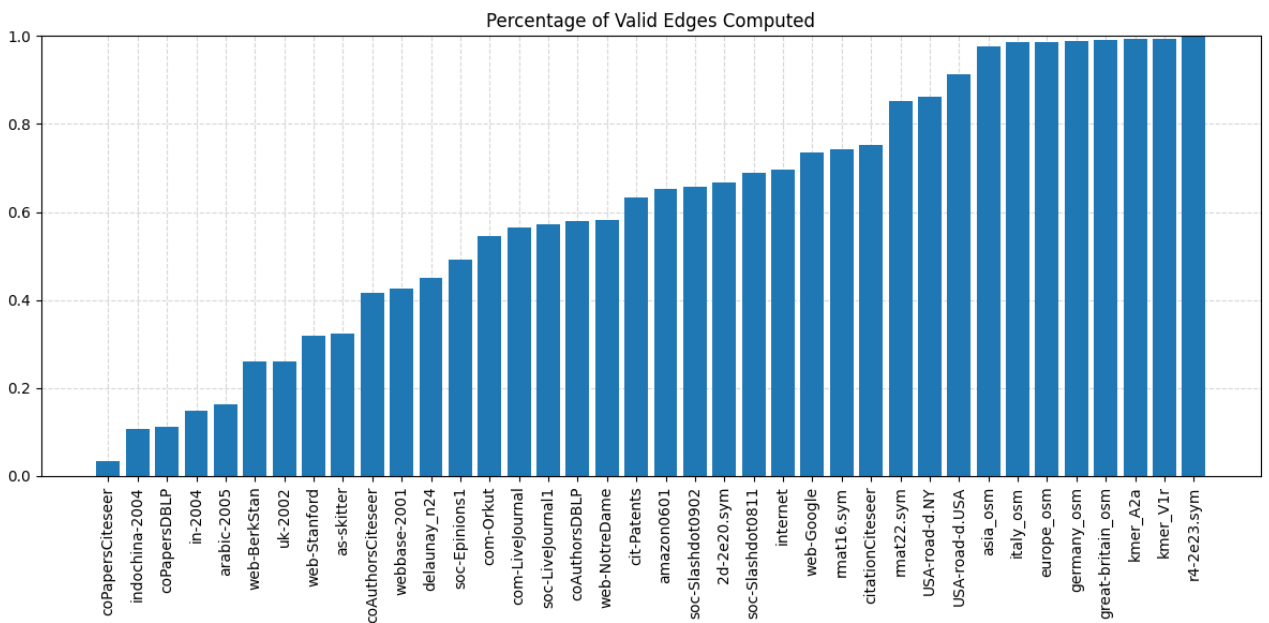


Figure 9: Speed Up Curves vs. Degree Cutoff

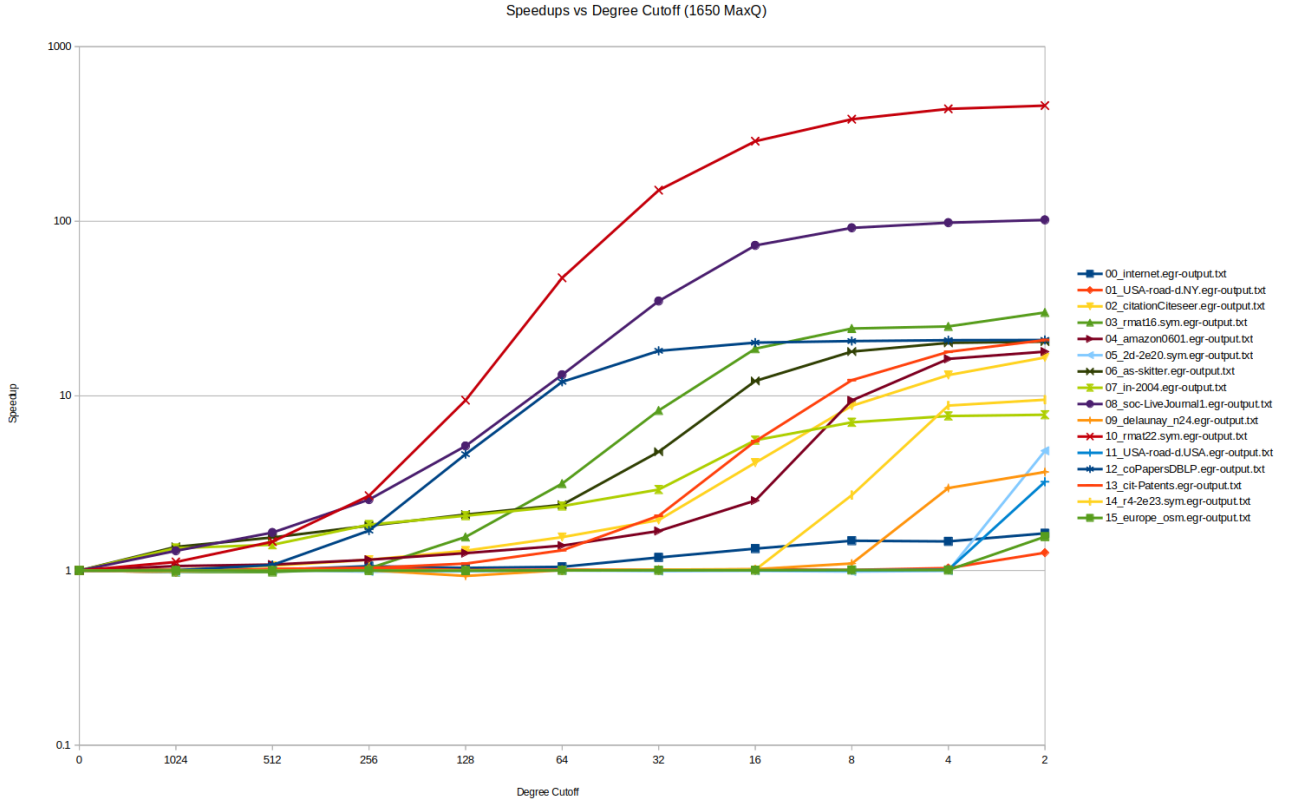


Figure 10: Runtime vs. Amount of Memory for Edge Processing

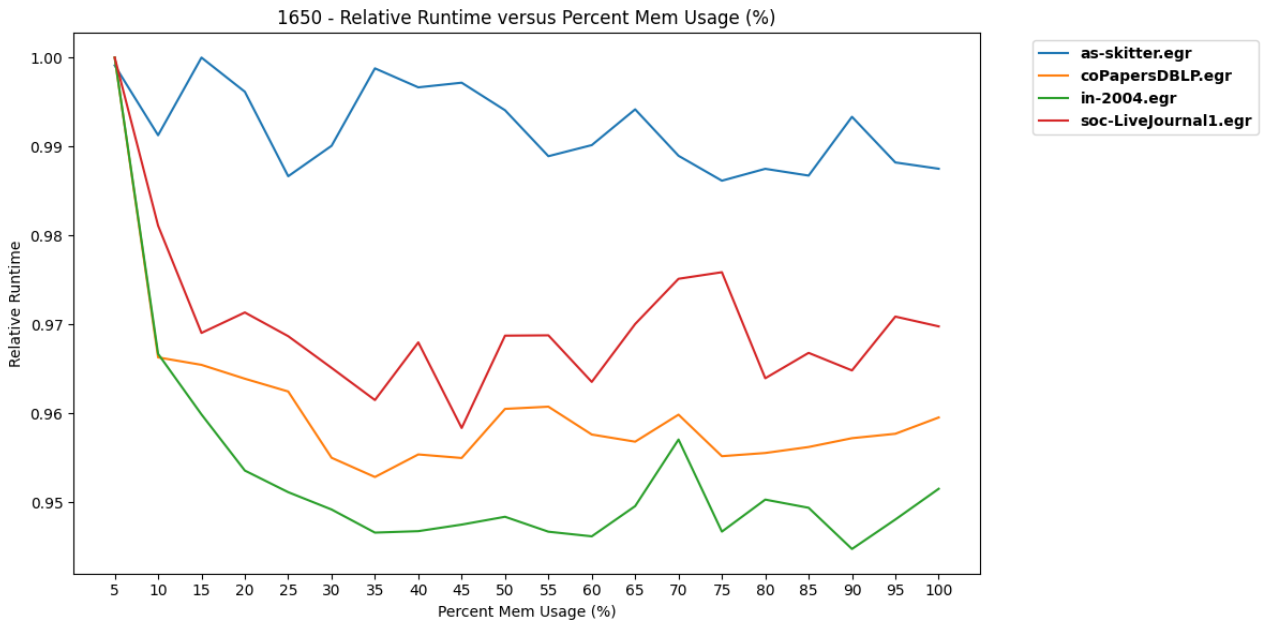
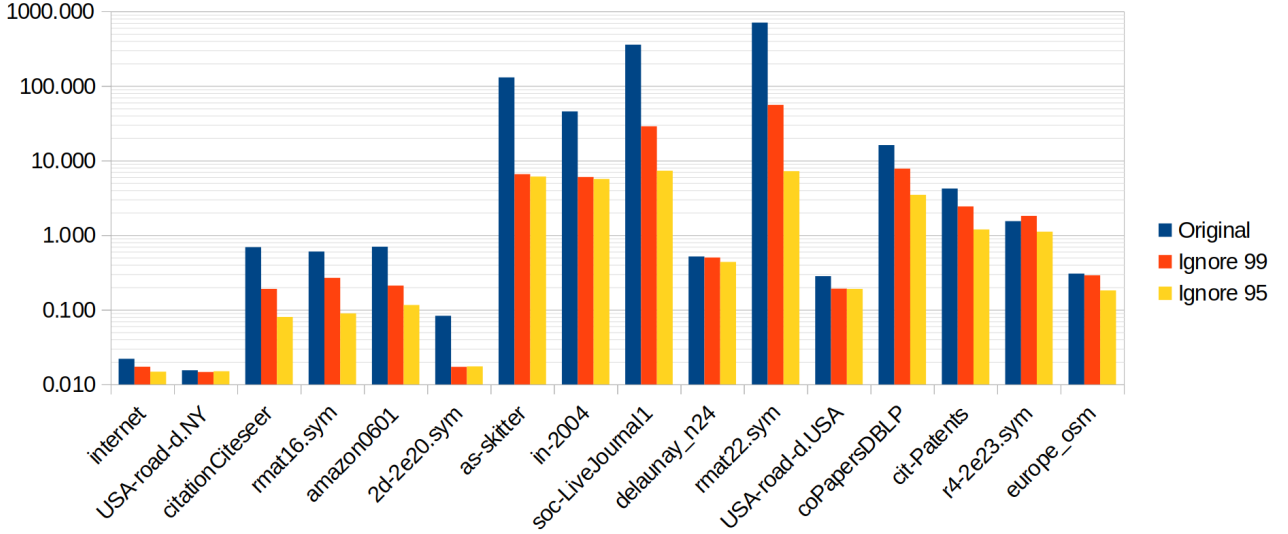


Figure 11: Ignoring Percentiles - Runtimes



9.2 Solution quality - F1 Score

9.2.1 Ignoring Vertices and F1 Score

We next discuss observed results of F1 Score. Figures 11 and 12 depict runtimes and F1 scores for a few inputs with regard to the strategy of ignoring the top percentile of vertices whose degree exceed a threshold. As can be expected, the more vertices are ignored, the faster the runtime proceeds, and this is appropriately reflected in Figure 11. When it comes to discussion of F1 Score, however, it can be noted in Figure 12 that there tends to be no apparent correlation between including or disregarding the larger-degree vertices. However, it is a fair conclusion that F1 scores remain relatively the same or near each other when large-degree vertices are ignored—that is, there are no large jumps in solution quality. As a benefit of ignoring large vertices, we can then conclude that we can speed up the computation for little tradeoff in solution qualities. For example, inputs “as-skitter” and “in-2004” experience a decrease in overall runtime by an order of magnitude when ignoring beyond the 99th and 95th percentiles, though their F1 scores remain mostly unchanged. In fact, the F1 score for “as-skitter” decreases slightly when ignoring beyond the 99th percentile, but then becomes better than not ignoring vertices at all when ignoring beyond the 95th percentile. The same can be stated for the “internet” and “soc-LiveJournal1” inputs.

9.2.2 Computing Weights and F1 Score

In an attempt to find optimal weights per input, we compared the F1 score results of using two different datasets as described above in section 6. Figures 13 and 14 depict the resulting weight distribution with regard to the 10 metrics used in this study as listed in Table 1.

In comparing figures 13 and 14, it can be observed that the Logistic Regression from both datasets tends to provide somewhat equal priority to the *CN* metric. However, there is a stark difference between the “Salton Cosine” *sai* metric and beyond. For the “Top-K” model, the Logistic Regression gives the *sai* metric a priority of less than 0.50, but the “Compute Removed Edges” model gives the same full priority. Additionally, the TopKF model indicates that the “Jaccard Coefficient” (or “Jaccard Index”) is inversely correlated with True Positives, but the CRE model disregards the same metric entirely. There is also disagreement between the *hpi* metric, but the remainder of the metrics have somewhat correlated priorities—that is, the *hdi*, *lnni*, *pa*, *aa*, and *ra* metrics are regarded in the same direction but with differing strengths.

The rest of the inputs have a similar depiction of differences in weights. Figure 15 shows a collection of other resulting weights for other inputs, and it can be easily seen that inputs vary with priority to specific metrics.

The results for comparing the F1 scores for each of the approaches used are depicted in Figure 16. Each group-of-3 columns in this figure depicts, in order, the F1 score that results from using “original” weights (which were arbitrarily-chosen), the F1 score that results from using the weights derived from the TopKF approach for finding optimal weights, and the F1 score that results from using the weights derived from the CRE approach for finding optimal weights, and each group-of-3 is associated with the input of the specified label. A quick observation is that there is no apparent correlation between weights and solution qualities. Sometimes,

Figure 12: Ignoring Percentiles - F1 Scores
Ignore top 5 percentile deg vs. Ignore top 1 percentile deg vs. No Ignore - F1 Scores

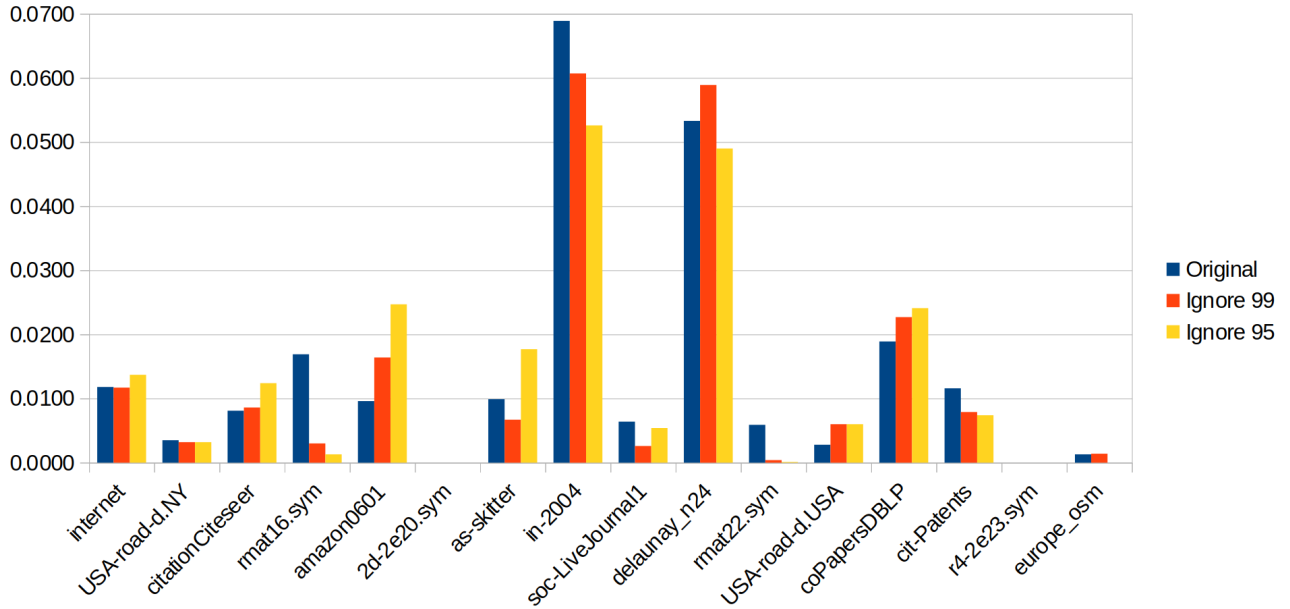


Figure 13: “Top-K * Factor” Weights for Input “Amazon0601”

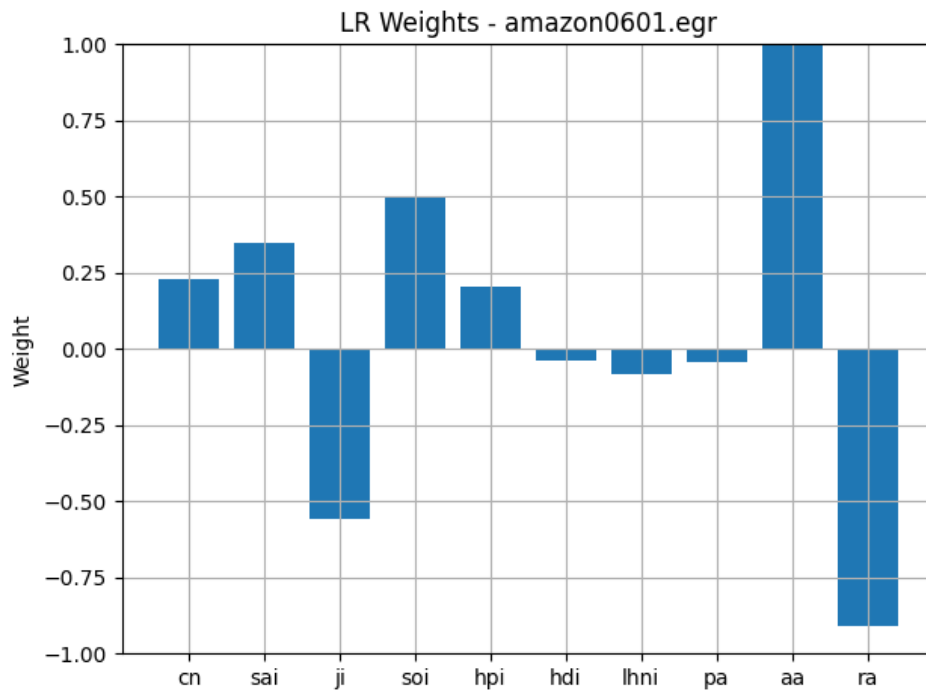


Figure 14: CRE Weights for Amazon

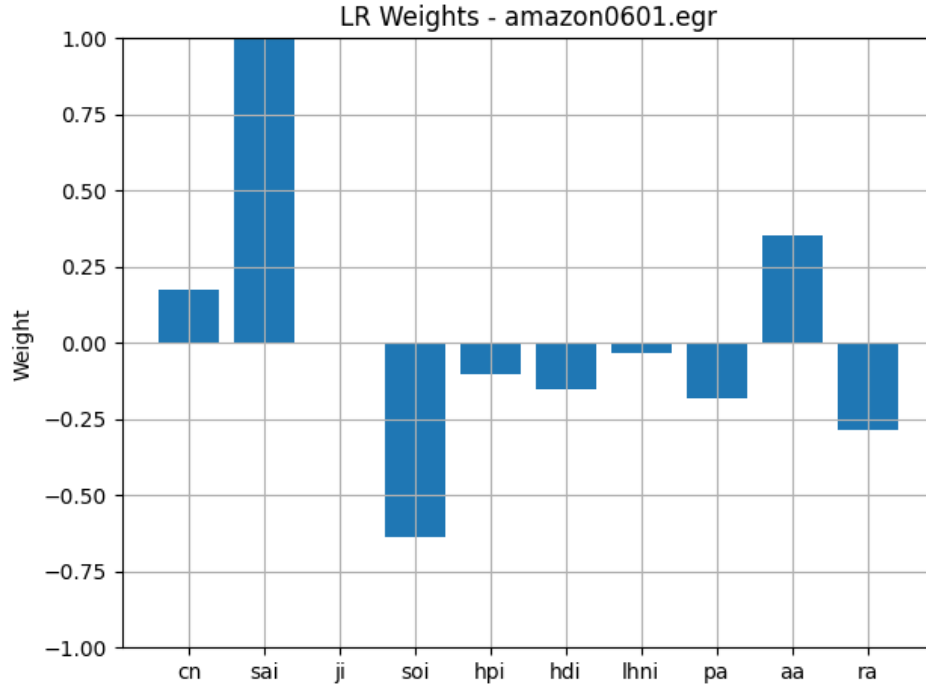


Figure 15: Grouped Weights for All Inputs



Figure 16: F1 Scores Comparing Weight Methods

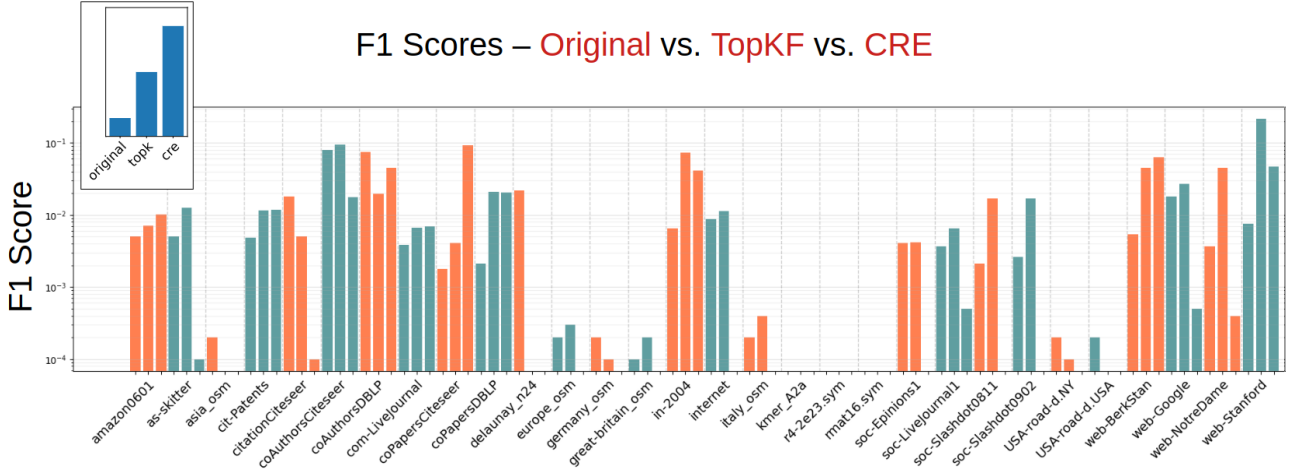
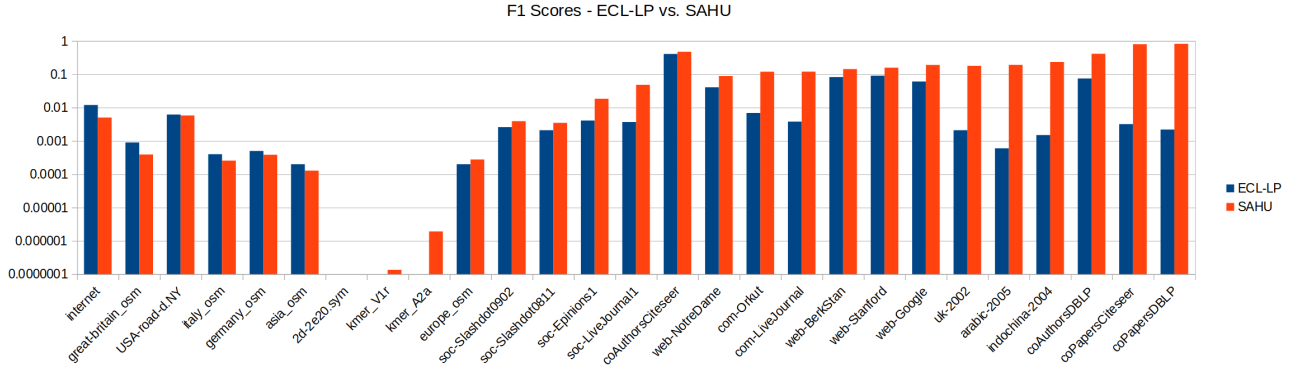


Figure 17: Final F1 Scores with Weights - ECL-LP vs. SAHU



the Logistic Regressions (LR) improve the F1 score, as is the case with inputs “amazon0601”, “cit-Patents”, “com-LiveJournal”, “coPapersCiteSeer”, “coPapersDBLP”, and “web-Berkstan”. In other cases, such as inputs “asia_osm”, “delauney_n24”, “europe_osm”, “germany_osm”, “USA-road-d.NY”, and “USA-road-d.USA”, the F1 score degrades or becomes completely eliminated (meaning there were extremely few or no True Positives in the resulting list of predicted edges). In yet other cases, there is no correlation, as the LR scores were lower than the original arbitrary-weight score as in input “coAuthorsDBLP”, the TopKF LR score was better than the original arbitrary-weight score, but the CRE LR score was lower than either as in input “web-NotreDame”, or the TopKF score was better than the original arbitrary-weight score, but the CRE score was eliminated as in the europe and great-britain road networks.

Nonetheless, the usage of “optimal weights” (the term “optimal” being used somewhat loosely) has shown to be beneficial. Figure 17 shows inputs’ F1 scores for current-best achieved F1 score by ECL-LP compared to Sahu[10]. The inputs are sorted by difference in F1 score, where ECL-LP leads by the most on the left-most bars and Sahu leads on the right-most bars. It is clear that when considering multiple metrics simultaneously, there are better predictions to be had when comparing against implementations that only use a single metric at a time. For example, on input “internet”, we can achieve a better set of predicted edges with multiple metrics than any use of a single-metric. The “current best” result of ECL-LP in this Figure is depicted due to the left-most inputs arriving at better F1 scores due to the metric-weight grid search as described in section 6.3. Other inputs’ “current best” is the result of the usage of original weights, the TopKF weights, or the CRE weights.

10 Discussion and Future Work

Figure 17 reveals that considering more than a single metric at a time when analyzing any graph has the potential to reveal more intrinsic information than running the same analyses with any single-metric alone. The grid-search method however, which allowed for finding weights that led to better F1 scores, is discrete in nature,

and does not consider the continuous influence that varying weights would otherwise have on the resulting score. In other words, discrete approaches to finding optimal weights only offer a certain amount of resolution between each metric, and grid searching more combinations is not a favorable approach for finding optimal weights due to the combinatorial explosion of possible weight combinations.

10.1 Limitations of Logistic Regression

The Logistic Regression approach used in this study may stand to improve upon philosophy or direction but is ultimately limited by the quality of the input dataset used to train the model. The TopKF input dataset suffered from too great an imbalance of True Positives against False Positives—that is, there are certainly not enough True Positives to steer the model towards higher precision and recall when it comes to model evaluation, and arguably, the model performs really well at predicting True Negatives. The intention of balancing the input dataset made an appearance with the “Compute Removed Edges” LR model, where an equal number of True Positives and True Negatives were represented in the input dataset. However, this also lacks some information. Consider the scenario where two pairs of vertices whose neighborhoods are identical have an edge (and thusly a True Positive) in one pair of vertices, but the other pair does not. The evaluation of the collective metrics would ultimately arrive at the same scores, but one pair of vertices actually has the True Positive and the other does not. This information is not captured as part of the LR model and graph evaluation, which may call for alternative or additional methods of predicting an edge or finding optimal weights in conjunction with the approach used here.

10.2 The Average Case

The static-graphs analysis approach has the limitation of evaluation through randomness. In other words, randomly-removing edges per run per input can result in vastly non-correlated data and, thus, metric weights. It should be noted that the implementation aims to be fast and good in solution quality for the average case, such that many runs over time would eventually lead to better results, discarding the nature of the randomness associated with the graph evaluation. This is furthered by the idea that graphs span many domains, and naturally, the best average-case would be different per domain. Indeed, this effort could be furthered by the necessity to find non-discrete approaches for finding optimal-weights.

11 Summary and Conclusions

In this study, we present ECL-LP, a parallel GPU implementation that is competitive with parallel CPU codes in terms of runtime performance. Utilizing a handful of optimizations allows for the implementation to compute multiple edge-scoring metrics that can potentially reveal more intrinsic information from inputs than when considering any one metric alone, resulting in better edge-prediction capabilities per input. Fast runtimes allowed grid search for finding weights that can be attributed to each metric to result in better F1 scores than the leading CPU implementation. We aim to find better methods of computing optimal metric-weights to consistently improve F1 scores across graphs that are representative of varying domains. This can ultimately help recommendation systems improve upon their suggestions or better establish connections or links in social networks.

References

- [1] Wei Cui et al. “Bounded link prediction in very large networks”. en. In: *Physica A: Statistical Mechanics and its Applications* 457 (Sept. 2016), pp. 202–214. ISSN: 03784371. DOI: 10.1016/j.physa.2016.03.041. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0378437116300395> (visited on 03/30/2026).
- [2] David Liben-Nowell and Jon Kleinberg. “The Link Prediction Problem for Social Networks”. en. In: (2003).
- [3] Caleb Belth, Alican Buyukcakil, and Danai Koutra. “A Hidden Challenge of Link Prediction: Which Pairs to Check?” en. In: *2020 IEEE International Conference on Data Mining (ICDM)*. Sorrento, Italy: IEEE, Nov. 2020, pp. 831–840. ISBN: 978-1-7281-8316-9. DOI: 10.1109/ICDM50108.2020.00092. URL: <https://ieeexplore.ieee.org/document/9338267/> (visited on 03/30/2026).

- [4] Huan Wang, Guoquan Liu, and Po Hu. “TDAN: Transferable Domain Adversarial Network for Link Prediction in Heterogeneous Social Networks”. en. In: *ACM Transactions on Knowledge Discovery from Data* 18.1 (Jan. 2024), pp. 1–22. ISSN: 1556-4681, 1556-472X. DOI: 10.1145/3610229. URL: <https://dl.acm.org/doi/10.1145/3610229> (visited on 03/30/2026).
- [5] Abdelhamid Saifi, Farid Nouioua, and Samir Akhrouf. “Fast approach for link prediction in complex networks based on graph decomposition”. en. In: *Evolving Systems* 15.2 (Apr. 2024), pp. 303–320. ISSN: 1868-6478, 1868-6486. DOI: 10.1007/s12530-023-09492-2. URL: <https://link.springer.com/10.1007/s12530-023-09492-2> (visited on 03/30/2026).
- [6] Zhenyu Hu et al. “Predicting cross-domain collaboration using multi-task learning”. en. In: *Expert Systems with Applications* 255 (Dec. 2024), p. 124570. ISSN: 09574174. DOI: 10.1016/j.eswa.2024.124570. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0957417424014374> (visited on 03/30/2026).
- [7] Huan Wang et al. “Resisting the Edge-Type Disturbance for Link Prediction in Heterogeneous Networks”. en. In: *ACM Transactions on Knowledge Discovery from Data* 18.2 (Feb. 2024), pp. 1–24. ISSN: 1556-4681, 1556-472X. DOI: 10.1145/3614099. URL: <https://dl.acm.org/doi/10.1145/3614099> (visited on 03/30/2026).
- [8] Djihad Arrar, Nadjat Kamel, and Abdelaziz Lakhffif. “A comprehensive survey of link prediction methods”. English. In: *The Journal of Supercomputing: An International Journal of High-Performance Computer Design, Analysis, and Use* (Sept. 2023), pp. 1–41. ISSN: 09208542. DOI: 10.1007/s11227-023-05591-8. URL: <https://research.ebsco.com/linkprocessor/plink?id=0fc01c3b-562b-32ec-9692-ec7b1944a437> (visited on 03/30/2026).
- [9] Yanting Yin et al. “Super Resolution Graph With Conditional Normalizing Flows for Temporal Link Prediction”. en. In: *IEEE Transactions on Knowledge and Data Engineering* 36.3 (Mar. 2024), pp. 1311–1327. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2023.3295367. URL: <https://ieeexplore.ieee.org/document/10183879/> (visited on 03/30/2026).
- [10] Subhajit Sahu and Kishore Kothapalli. “A Fast Parallel Approach for Neighborhood-Based Link Prediction by Disregarding Large Hubs”. en. In: *Concurrency and Computation: Practice and Experience* 37.2 (Jan. 2025), e8331. ISSN: 1532-0626, 1532-0634. DOI: 10.1002/cpe.8331. URL: <https://onlinelibrary.wiley.com/doi/10.1002/cpe.8331> (visited on 03/30/2026).
- [11] M. E. J. Newman. “Clustering and preferential attachment in growing networks”. en. In: *Physical Review E* 64.2 (July 2001). arXiv:cond-mat/0104209, p. 025102. ISSN: 1063-651X, 1095-3787. DOI: 10.1103/PhysRevE.64.025102. URL: <http://arxiv.org/abs/cond-mat/0104209> (visited on 03/30/2026).
- [12] Paul Jaccard. “The Distribution of the Flora in the Alpine Zone”. In: *The New Phytologist* 11.2 (1912), pp. 37–50. ISSN: 0028-646X. URL: <https://www.jstor.org/stable/2427226> (visited on 05/09/2026).
- [13] Th Sorenson. “A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons”. In: *Biol Skar* 5 (1948), p. 1.
- [14] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.
- [15] E. A. Leicht, Petter Holme, and M. E. J. Newman. “Vertex similarity in networks”. en. In: *Physical Review E* 73.2 (Feb. 2006). arXiv:physics/0510143, p. 026120. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.73.026120. URL: <http://arxiv.org/abs/physics/0510143> (visited on 03/30/2026).
- [16] E. Ravasz et al. “Hierarchical Organization of Modularity in Metabolic Networks”. In: *Science* 297.5586 (2002), pp. 1551–1555.
- [17] Albert-László Barabási and Réka Albert. “Emergence of Scaling in Random Networks”. In: *Science* 286.5439 (1999), pp. 509–512.
- [18] Lada A Adamic and Eytan Adar. “Friends and neighbors on the Web”. en. In: *Social Networks* 25.3 (July 2003), pp. 211–230. ISSN: 03788733. DOI: 10.1016/S0378-8733(03)00009-1. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0378873303000091> (visited on 03/30/2026).
- [19] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. “Predicting missing links via local information”. en. In: *The European Physical Journal B* 71.4 (Oct. 2009), pp. 623–630. ISSN: 1434-6028, 1434-6036. DOI: 10.1140/epjb/e2009-00335-8. URL: <http://link.springer.com/10.1140/epjb/e2009-00335-8> (visited on 03/30/2026).

Figure 18: Runtime Comparison of kmer_A2a to Sahu

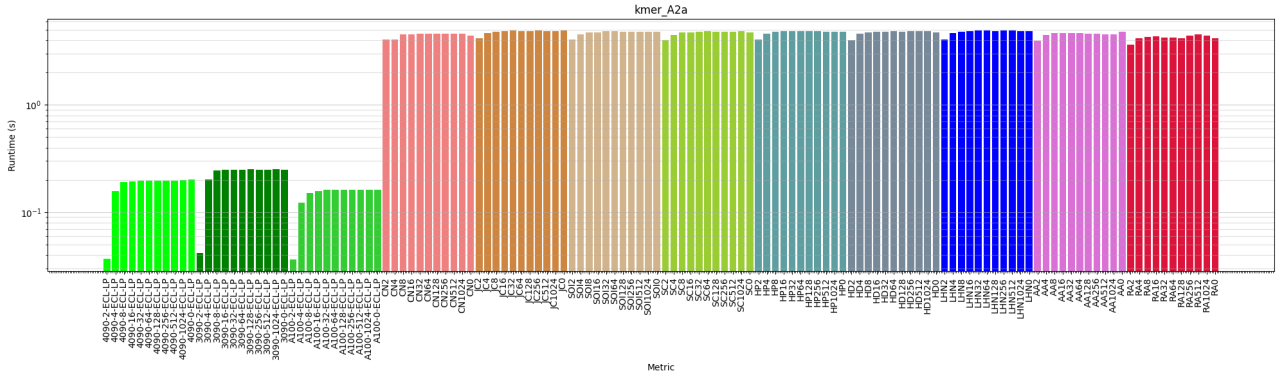


Figure 21: Runtime Comparison of asia_osm to Sahu

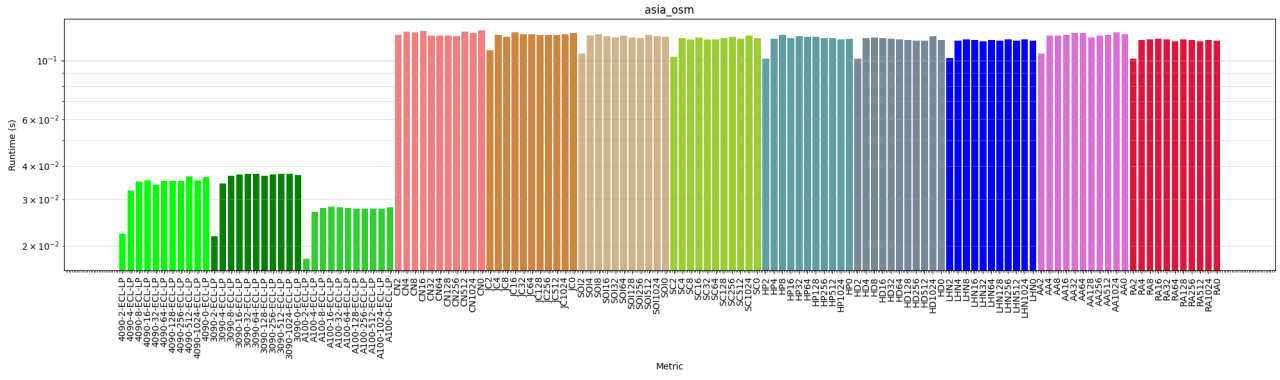


Figure 22: Runtime Comparison of germany_osm to Sahu

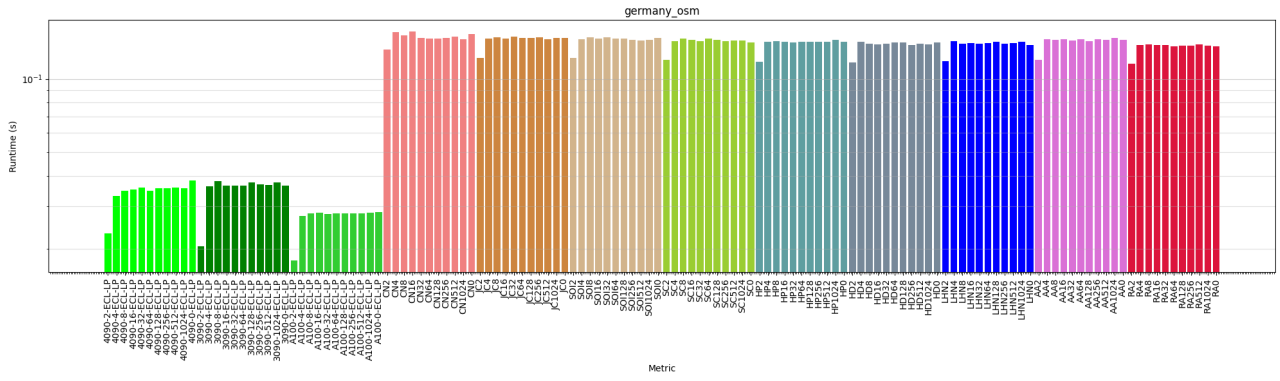


Figure 23: Runtime Comparison of europe_osm to Sahu

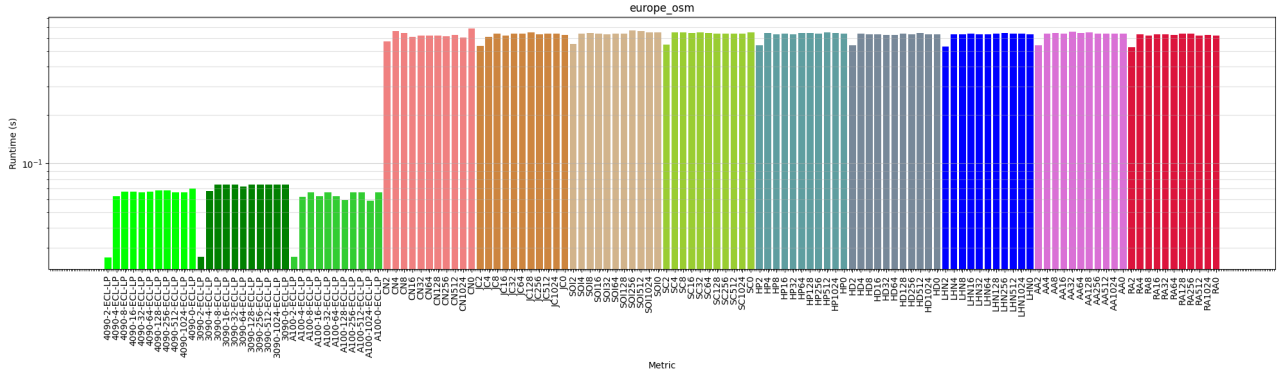


Figure 24: Runtime Comparison of great-britain_osm to Sahu

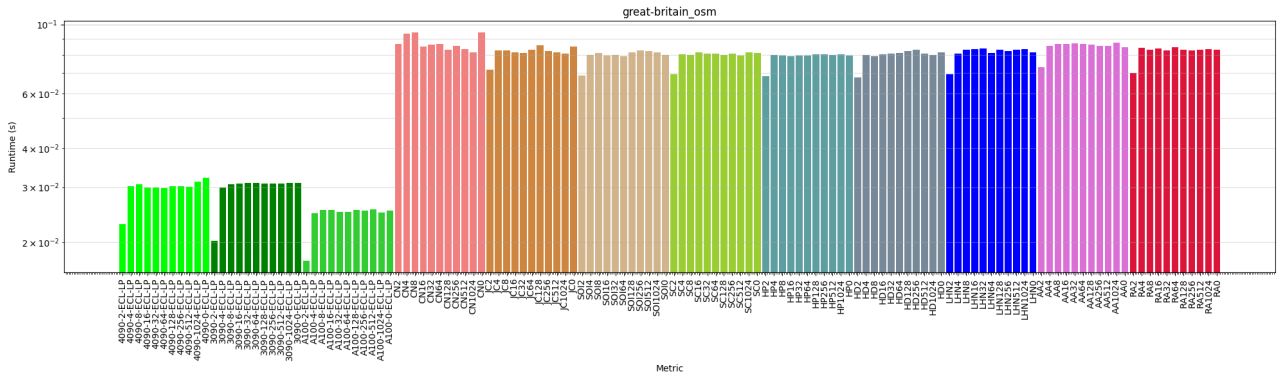


Figure 25: Runtime Comparison of web-Google to Sahu

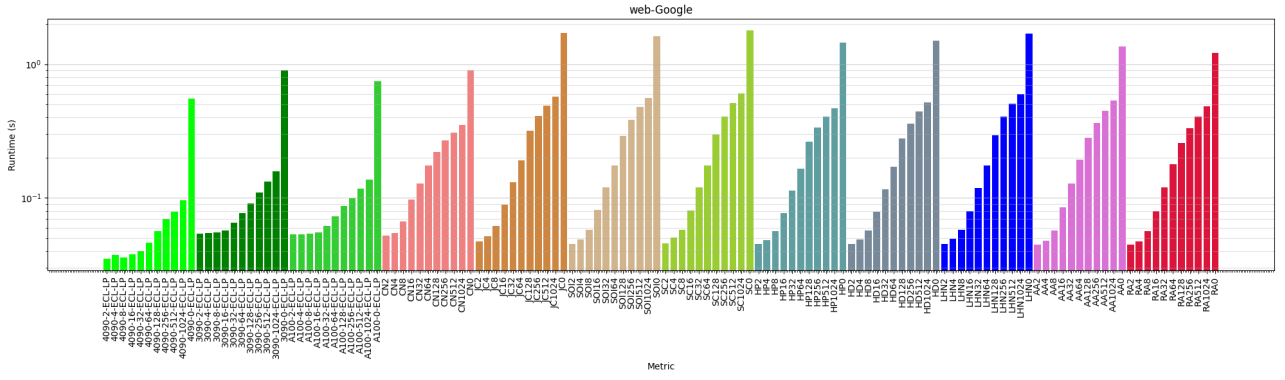


Figure 26: Runtime Comparison of soc-Slashdot0811 to Sahu

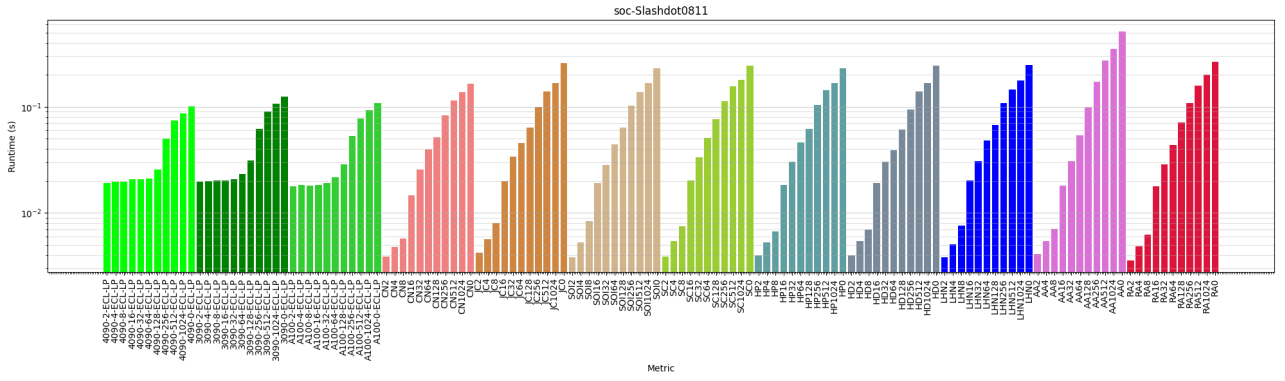


Figure 27: Runtime Comparison of soc-Slashdot0902 to Sahu

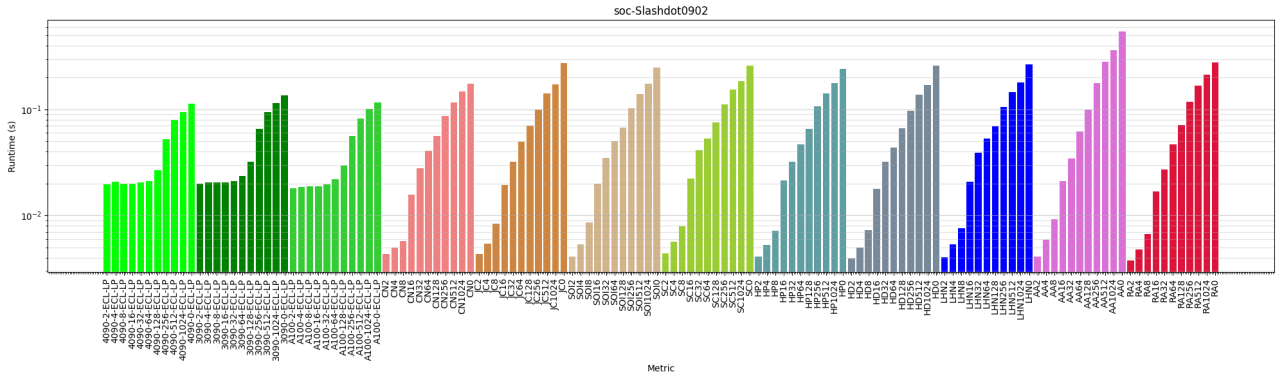


Figure 28: Runtime Comparison of soc-Epinions1 to Sahu

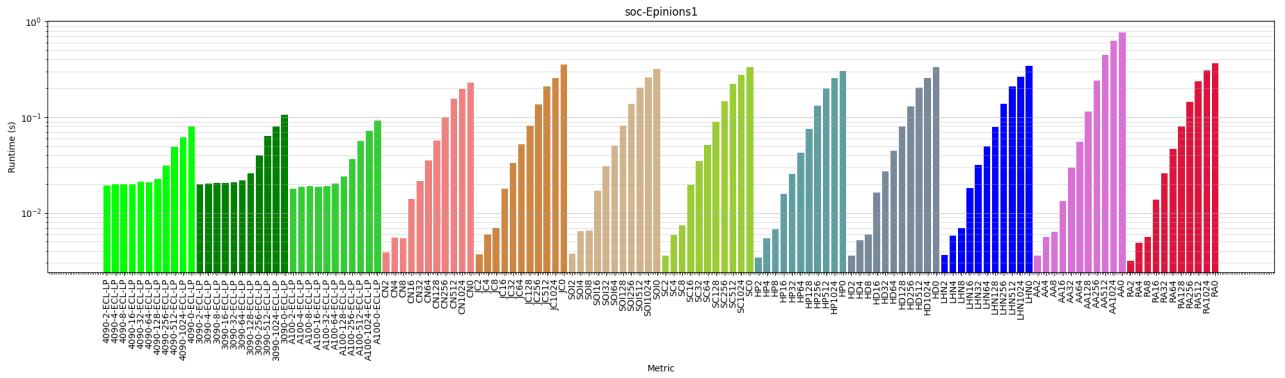


Figure 29: Runtime Comparison of com-LiveJournal to Sahu

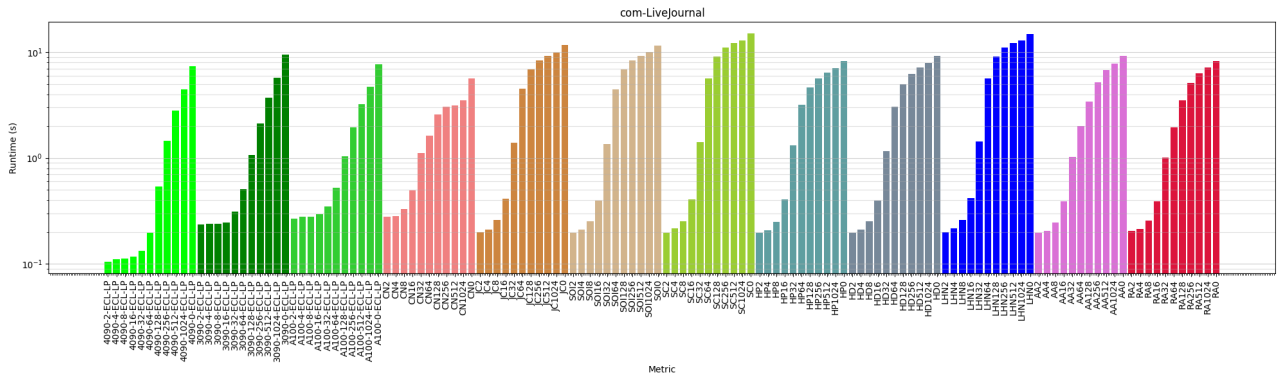


Figure 30: Runtime Comparison of soc-LiveJournal1 to Sahu

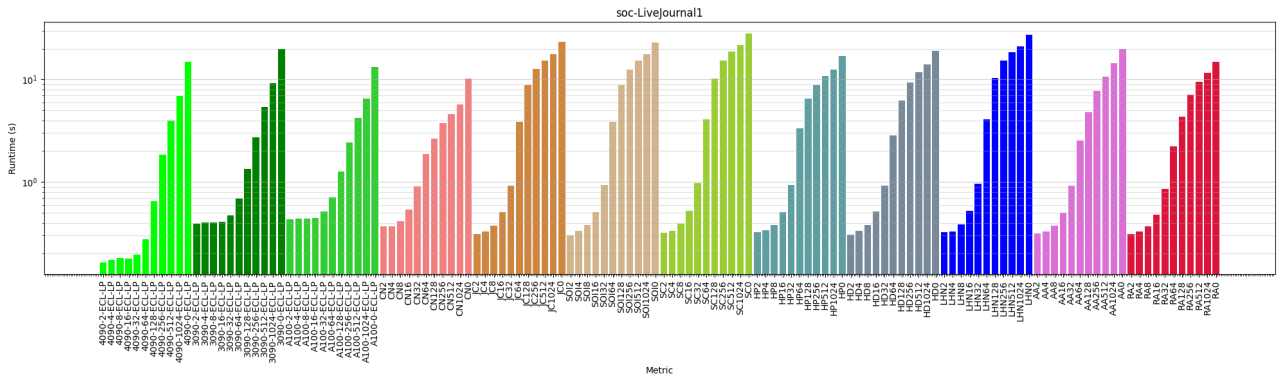


Figure 31: Runtime Comparison of coAuthorsDBLP to Sahu

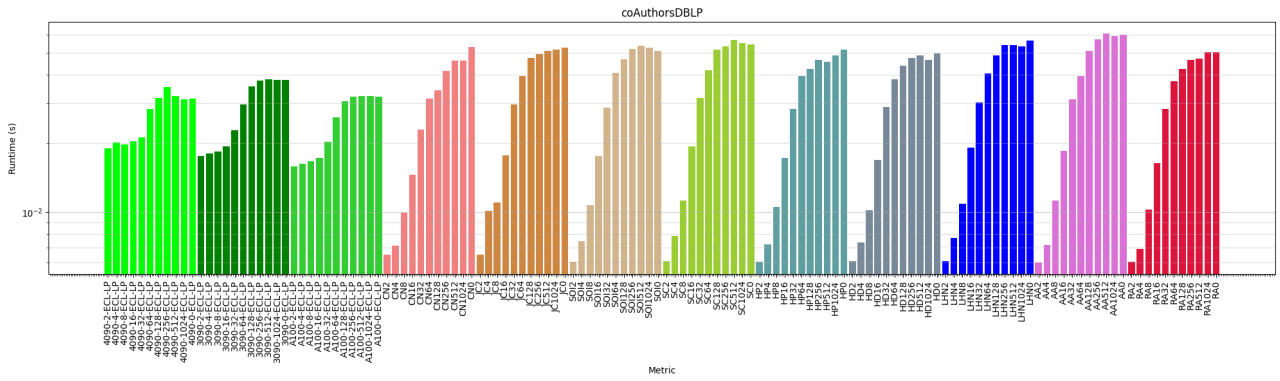


Figure 32: Runtime Comparison of coPapersCiteseer to Sahu

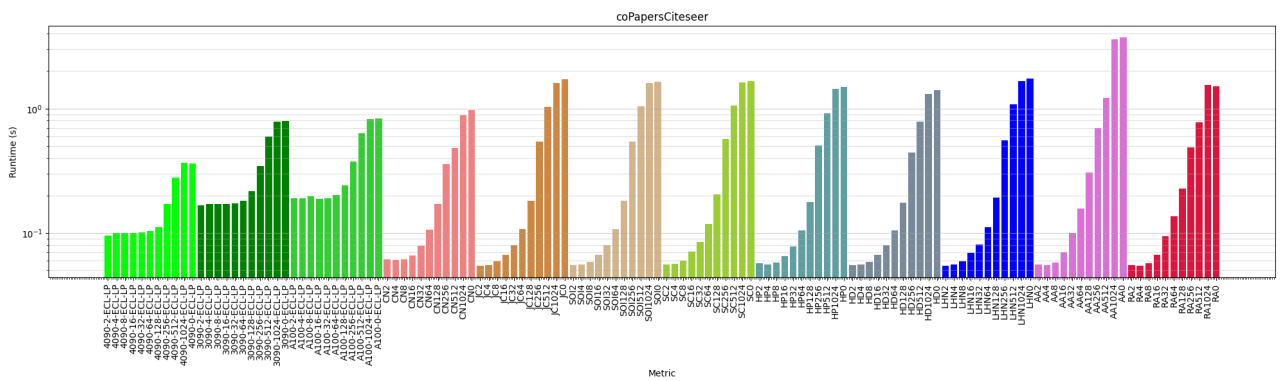


Figure 33: Runtime Comparison of com-Orkut to Sahu

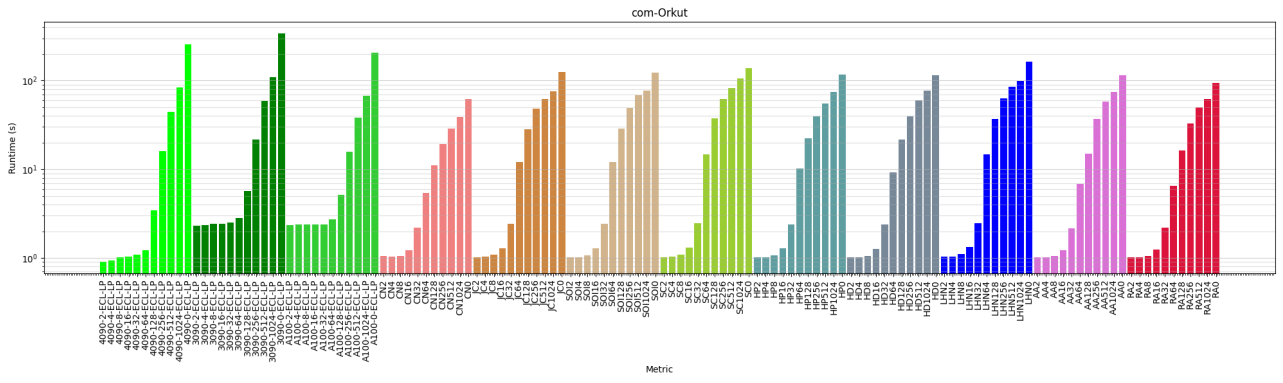


Figure 34: Runtime Comparison of coPapersDBLP to Sahu

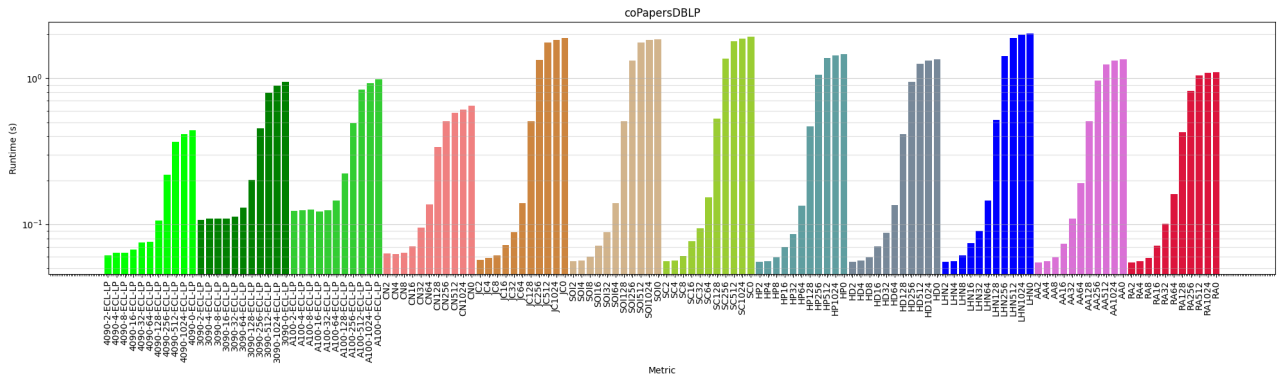


Figure 35: Runtime Comparison of web-Stanford to Sahu

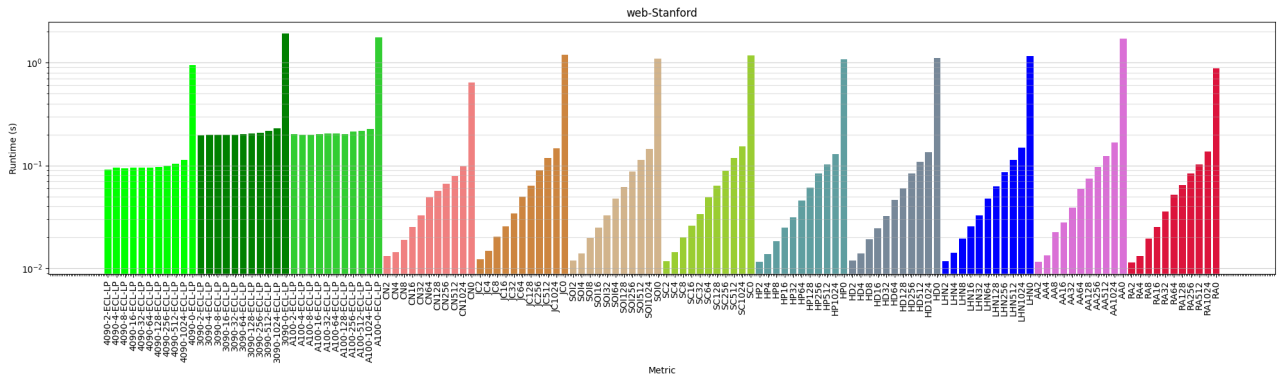


Figure 36: Runtime Comparison of web-NotreDame to Sahu

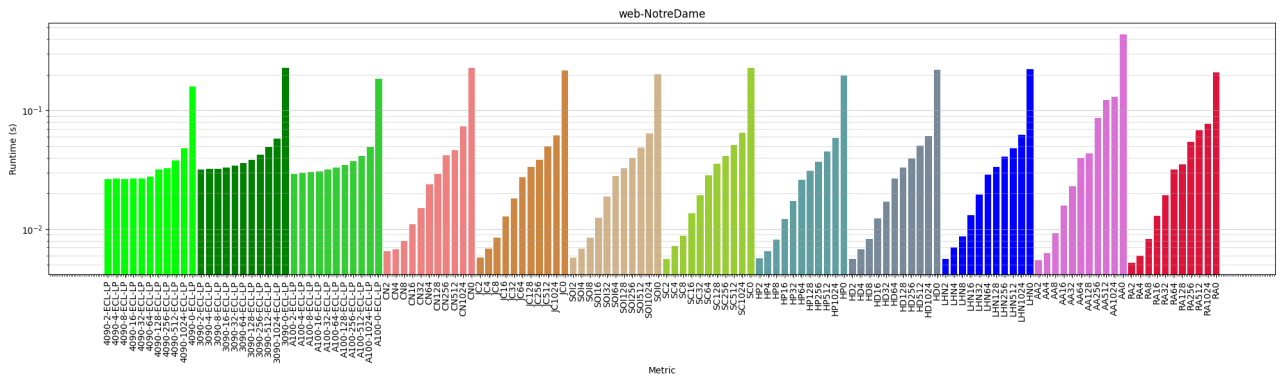


Figure 37: Runtime Comparison of web-BerkStan to Sahu

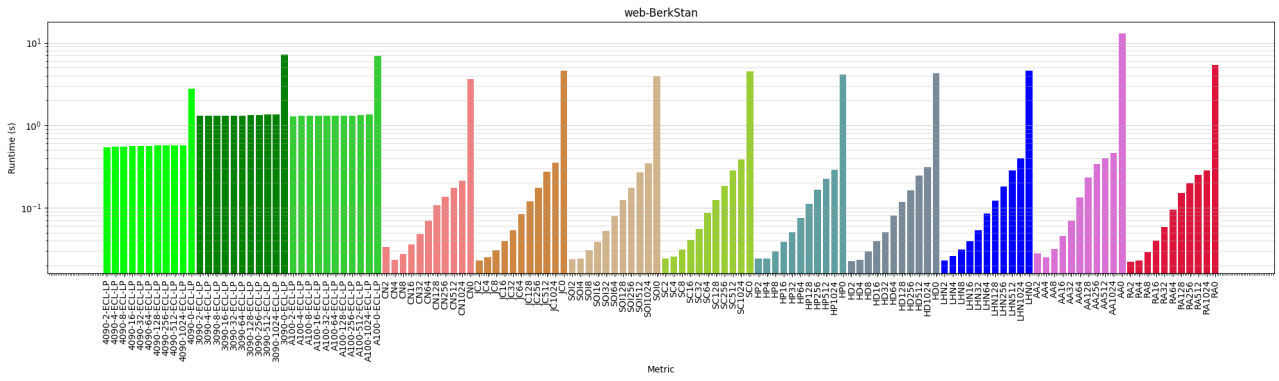


Figure 38: Runtime Comparison of uk-2002 to Sahu

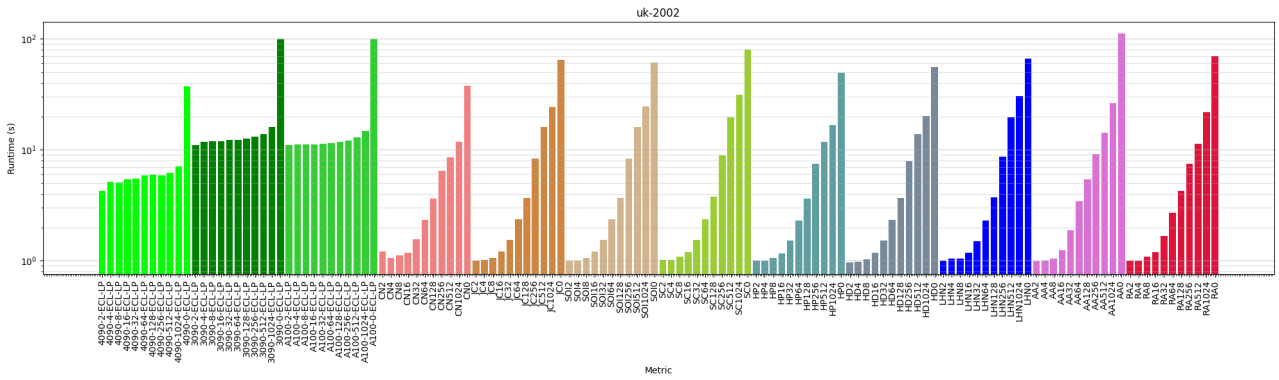


Figure 39: Runtime Comparison of coAuthorsCiteseer to Sahu

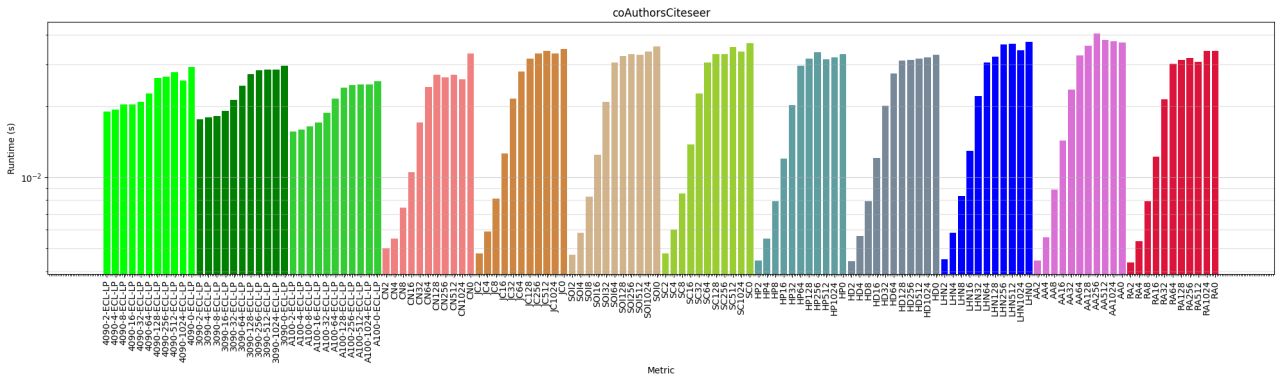


Figure 40: Runtime Comparison of indochina-2004 to Sahu

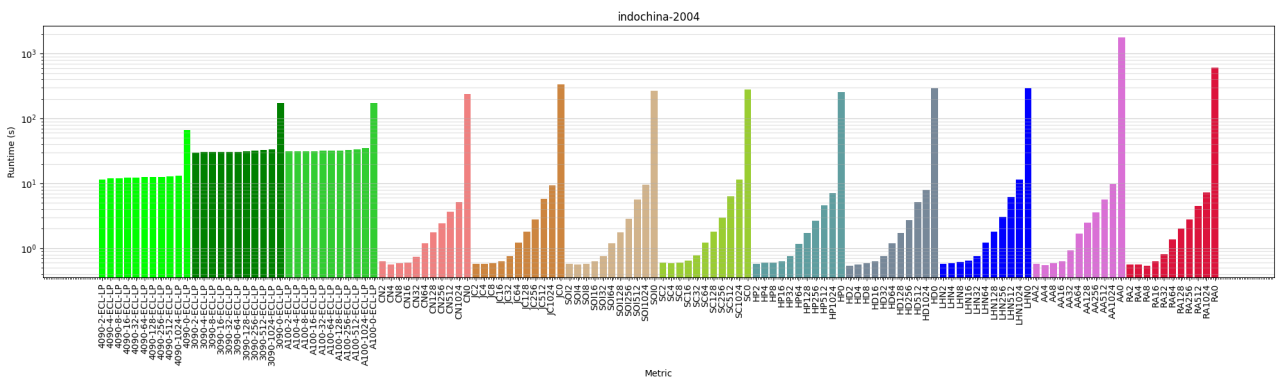


Figure 41: Runtime Comparison of arabic-2005 to Sahu

