Leveraging CUDA for Local Similarity-Based Link Prediction in Large-Scale Graphs

Ahmad Al Musawi
Department of Computer Science
Virginia Commonwealth University
Richmond, VA, USA
almusawiaf@vcu.edu

***Abstract*—Graph link prediction, a key component in network analysis, plays a pivotal role in deciphering the underlying dynamics of complex systems. This paper explores the potential of leveraging CUDA (Compute Unified Device Architecture) for accelerating local similarity-based link prediction algorithms in large-scale graphs. Traditional computational methods often struggle to handle the expansive data sets typical in domains such as social networking and biological systems, where the volume of connections and the rate of data generation can be substantial. To address these challenges, we implement and evaluate various similarity measures, primarily focusing on the common neighbors approach, within a CUDA-enabled environment. Our experiments demonstrate significant speedup, especially evident in larger graphs, underscoring the efficiency of parallel processing in graph analytics. The outcomes of this study not only validate the effectiveness of CUDA in handling large-scale network data but also open avenues for integrating advanced parallel computing techniques with existing analytical tools and frameworks.**

***Index Terms*—Link prediction, local similarity, High-performance computing, CUDA**

I. Introduction

Link prediction, a fundamental task in graph mining and network analysis, aims to predict the likelihood of the existence of a link between two nodes in a network based on observed data. This prediction is crucial in understanding and extrapolating the behavior and evolution of complex networks. Its applications span various fields, such as social networks, biological systems, and medical knowledge bases.

In **social networks**, link prediction plays a pivotal role in enhancing user experience and facilitating network growth. It helps in:

- *Friendship and Interaction Prediction*: By analyzing user interactions and connections, social platforms can predict potential friendships or connections, enhancing social integration and user engagement.
- *Recommender Systems*: Link prediction aids in developing sophisticated recommender systems that suggest content, groups, or connections, thus improving content relevance and user retention.
- *Community Detection*: Predicting links contributes to identifying emerging communities within the network, which can be crucial for targeted marketing, information dissemination, and understanding social dynamics.
- *Information Spread Modeling*: Understanding potential future links helps in modeling how information, like news or public health advisories, might spread across the network.

In **biological networks**, link prediction is vital for unraveling complex biological processes and interactions:

- *Protein-Protein Interaction (PPI) Networks*: Predicting interactions between proteins can lead to a deeper understanding of cellular processes and the development of new drugs.
- *Gene Regulatory Networks*: By forecasting links between genes and regulatory elements, scientists can better understand genetic controls and implications in diseases.
- *Disease-Gene Association*: Link prediction can help identify unknown associations between genes and diseases, paving the way for novel therapeutic approaches and personalized medicine.

In the realm of **medical knowledge**, link prediction has significant implications:

- Drug Repurposing: Predicting interactions between drugs and target molecules can aid in identifying new uses for existing drugs, accelerating drug development.
- Patient Symptom Analysis: Link prediction can analyze patient data to forecast potential health risks or disease progression, contributing to preventive medicine and early intervention strategies.
- Medical Research Collaboration: By predicting potential collaborations between researchers or institutions based on previous research outputs, link prediction can foster partnerships that accelerate medical breakthroughs.

A. Utilizing CUDA for Accelerated Link Prediction Processing

The increasing complexity and scale of networks, particularly in the realms of social media, biology, and medical informatics, necessitate more efficient computational approaches for link prediction. This is where CUDA (Compute Unified Device Architecture) comes into play as a parallel computing platform and application programming interface model created by NVIDIA. CUDA enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

- Handling Large-scale Data: Networks, especially social and biological ones, can encompass millions of nodes and billions of links. CUDA facilitates the handling of such large-scale data by parallelizing the computation, significantly speeding up the processing time compared to traditional CPU-based methods.
- Complex Algorithm Implementation: Link prediction often involves complex algorithms, including machine learning models, which can be computationally intensive. CUDA allows for these algorithms to be executed more efficiently by distributing the workload across multiple GPU cores.
- Real-time Analysis: In scenarios like social networks where real-time data analysis is crucial for features like recommendation systems or trend analysis, CUDA's parallel processing capabilities enable faster computations, allowing for near real-time link prediction.
- Energy Efficiency: GPUs, particularly those optimized for CUDA, are not only faster but also more energy-efficient for large-scale computations compared to CPUs. This aspect is increasingly important in large data centers and for environmentally conscious computing.

By employing CUDA, we can overcome the computational barriers associated with link prediction in complex networks, leading to more timely insights and actions in various fields. This approach not only speeds up the processing but also opens new horizons for employing more sophisticated, data-intensive algorithms that were previously impractical due to computational limitations.

B. Formulation of the Problem

- Each dataset is split into two separate, non-overlapping graphs: the training graph G^t and the probe/test graph G^p. The training graph G^t is formed by randomly sampling from the original graph G, while G^p comprises the edges not included in G^t. The edge sets of G^t and G^p are denoted as E^t and E^p respectively, where $E = E^t + E^p$ and E^t and E^p are mutually exclusive.
- We allocate 80% of the edges to E^t and the remaining 20% to E^p, see Fig. 1.
- The link prediction algorithm takes G^t as input and predicts future links, resulting in a new graph $G^{'}$. The algorithm's performance is measured by the number of true positive (TP) and false positive (FP) edges. A TP edge exists in both $G^{'}$ and G^p, whereas an FP edge is present in $G^{'}$ but not in the original graph G.
- To ensure reliability, we repeat the experiments several times with randomly generated G^t and G^p each time, and report average results over these runs.

II. Approaches Based on Similarity Metrics

This section discusses a fundamental and prevalent strategy for predicting potential links in a network: analyzing the similarity in attributes between non-linked node pairs.

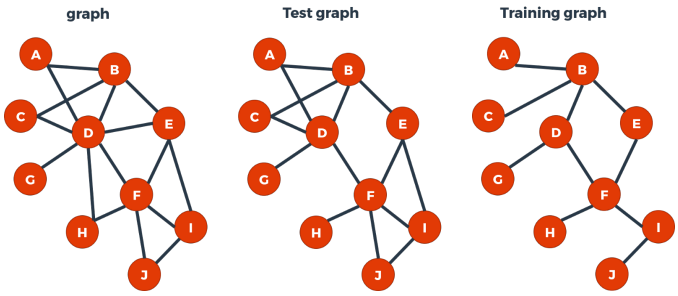

Fig. 1. Link prediction representation using a synthetic graph. The aim is to predict all non-existed edges in the train graph and validate them with the edges of the test graph.

A. Local Similarity Methods

Local similarity methods focus on the immediate neighbors of a node u, defined as the set Γ_u which contains nodes directly connected to u in the network. The underlying principle here is that two nodes are more likely to establish a link if they have common neighbors. One such method is the Common Neighbor (CN) approach. It evaluates the likelihood of a link formation between nodes u, v based on the number of mutual neighbors they have, expressed as $S^{CN}_{u,v} = |\Gamma_u \cap \Gamma_v|$ [1].

B. Additional Similarity Measurement Techniques

Numerous techniques for assessing node similarity in graphs have been proposed and developed, as documented in [2]. The effectiveness of our current implementation significantly influences these computations, enabling the scaling of processing to accommodate large-scale networks.

C. Algorithms

Local similarity-based approaches use node neighborhoods to measure the similarity of each node with other nodes in the network. Local approaches are faster than non-local approaches and it is highly parallelizable and efficient for dynamic networks. However, all of the following algorithms have a computation complexity of $O(vk^3)$ except for the preferential attachment which has a computation complexity of $O(vk^2)$; v refers to the number of vertices (nodes) and k refers to the degree of the node. Most of these methods are well explained in V. Martinez et. al [3].

1) *Common Neighbors [1]* (CN) is the simplest and fundamental local technique. It measures the number of shared neighbors between two nodes x, y. A confirmed hypothesis [4] shows that for two distinct nodes, there is a correlation between the number of shared neighbors and the probability of being connected. The formula for $S^{CN}_{x,y}$ is as follows:

$$S^{CN}_{x,y} = |\Gamma_x \cap \Gamma_y| \tag{1}$$

2) *Adamic-Adar Index [5]* (AA) is another variation of common neighbors which measures the similarity be-

tween x, y by logarithmically penalizing the shared neighbors.

$$S_{x,y}^{AA} = \sum_{z \in \Gamma_x \cap \Gamma_y} \frac{1}{\log |\Gamma_z|} \tag{2}$$

3) *Resource Allocation Index[6]* (RA) is another variation of both common neighbors and the Adamic-Adar index which models the unit of resources between two unconnected nodes through neighborhood nodes. The number of resource units transmitted from node x using x's neighbors and received by node y reflects the degree of similarity between x, y.

$$S_{x,y}^{RA} = \sum_{z \in \Gamma_x \cap \Gamma_y} \frac{1}{|\Gamma_z|} \tag{3}$$

4) *Preferential attachment[7]* (PA) is based on a premise that in a large set of real networks, node degrees tend to follow a power law distribution resulting in scale-free networks. The probability of having an edge between two nodes increases as their degrees increase.

$$S_{x,y}^{PA} = |\Gamma_x||\Gamma_y| \tag{4}$$

5) *The Jaccard Index[8]* (JA) is a widely used similarity measurement that measures the ratio of shared neighbors in the complete set of neighbors for two nodes.

$$S_{x,y}^{JA} = \frac{|\Gamma_x \cap \Gamma_y|}{|\Gamma_x \cup \Gamma_y|} \tag{5}$$

6) *Sorensen Index[9]* (SI) is a very similar method to the Jaccard index, used to compare the similarity between different ecological community data samples.

$$S_{x,y}^{SI} = \frac{|\Gamma_x \cap \Gamma_y|}{|\Gamma_x| + |\Gamma_y|} \tag{6}$$

7) *Hub Promoted Index [10]* (HPI) measures the similarity between x, y by comparing the ratio of common neighbors of nodes x, y to the minimum degree of either node.

$$S_{x,y}^{HPI} = \frac{|\Gamma_x \cap \Gamma_y|}{min(|\Gamma_x|, |\Gamma_y|)} \tag{7}$$

8) *Hub Depressed Index [10]* (HDI) measures the similarity between x, y by comparing the ratio of common neighbors of nodes x, y to the maximum degree of either node.

$$S_{x,y}^{HDI} = \frac{|\Gamma_x \cap \Gamma_y|}{max(|\Gamma_x|, |\Gamma_y|)} \tag{8}$$

9) *Local Leicht-Homle-Newman Index [11]* (LLHN) is a model where the similarity between x, y nodes is measured as the ratio of common neighbors of the x, y nodes to the multiplication of neighbors of the x, y nodes.

$$S_{x,y}^{LLHN} = \frac{|\Gamma_x \cap \Gamma_y|}{|\Gamma_x||\Gamma_y|} \tag{9}$$

III. Parallelization Approach

A list of steps were implemented toward parallelization of the link prediction models.

A. Graph Preparation

Graphs can be read, saved, and processed in several ways. However, we did the following to read and prepare the graph:

1) *Graph File*: A graph can be encapsulated using a variety of data structures. In the current implementation, the graph is ingested from files, a process that involves parsing each line of the file. These files can come in numerous formats, including but not limited to plain text (.txt), edge lists (.edges), or graph modeling language files (.gml). In this experiment, we used (.edges) file format. Each line within these files denotes a pair of interconnected nodes, effectively defining an edge. Consequently, the total number of lines within the file provides a direct count of the edges contained in the graph. This approach enables the flexible handling of graph data and facilitates subsequent computational tasks, especially shuffling the edges (to be used in the splitting of the edges into train and test).
2) Nodes are represented as string variables, therefore no confusion can be made later in the implementation.
3) Each line is read, split, and the nodes are saved if they have not already been stored. This operation is computationally intensive due to the requirement of a node search for each single edge, consuming $O(vk)$.
4) Upon reading an edge, it is saved in its dedicated location. This process results in the creation of two structures: an *adjacency matrix* and an *adjacency list*. The adjacency matrix, of Boolean type, indicates whether there is an edge between any given pair of nodes. Additionally, for efficient neighbor search, another structure is created to hold the $< Node : Neighbors >$ mapping. These structures are critical for the kernel, as they are heavily relied upon to generate similarity scores.

B. The Kernels

Having read the graph and prepared the necessary data structures, we now compute the similarity score for all missing edges in the training graph. We have developed a distinct kernel for each type of similarity score. The inputs to these kernels are as follows:

1) **Adjacency Matrix, A**: This is a Boolean matrix where $A_{uv} = true$ if an edge exists between nodes u and v, and $false$ otherwise.
2) **Adjacency List, Neighbors** and its **Offset**: This list is an array of nodes, the length of which is equal to the number of edges. In this array, we concatenate the neighbors of each node sequentially. The offset is another array utilized to store the necessary index offset to locate the address of the first neighbor for a given node. The size of the offset array is equal to the number of nodes.
3) **Results Adjacency Matrix**: We use an additional adjacency matrix (results) of size $|V| * |V|$, with a data type of float, to store the calculated similarity scores.

All the adjacency matrix, adjacency list, offset, and results matrices are first initialized on the host and then transferred

to the device using the *cudaMemcpy* function with the *HostToDevice* parameter.

In the kernel, each thread is responsible for calculating the local-based similarity measure for its assigned (*row, col*) position. If the adjacency matrix indicates *false* at the current position (signifying a missing edge between the node at the *row*'th position and the node at the *col*'th position), the corresponding measurement is executed and the resulting score is stored at the same position. Subsequently, we retrieve the results by copying them back from the device to the host using the *cudaMemcpy* function with the *DeviceToHost* parameter. Algorithm 1 outlines the pseudocode for the common neighbor score kernel.

Algorithm 1 Kernel for Calculating Common Neighbors

procedure KERNEL_CN(A, $neighbors$, $offsets$, $results$, N)
  $col \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$
  $row \leftarrow blockIdx.y \times blockDim.y + threadIdx.y$
  $start1 \leftarrow offsets[col]$
  $end1 \leftarrow offsets[col + 1]$
  $start2 \leftarrow offsets[row]$
  $end2 \leftarrow offsets[row + 1]$
  $inter \leftarrow 0$
  if $row < N$ **and** $col < N$ **then**
    if not $A[row \times N + col]$ **then**
      for $i \leftarrow start1$ **to** $end1 - 1$ **do**
        for $j \leftarrow start2$ **to** $end2 - 1$ **do**
          if $neighbors[i] = neighbors[j]$ **then**
            $inter \leftarrow inter + 1$
            break
          end if
        end for
      end for
      $results[row \times N + col] \leftarrow inter$
    end if
  end if
end procedure

IV. Performance Evaluation

In the realm of link prediction algorithms, each absent link is assigned a score $S_{u,v}$, representing the likelihood of its future existence. A link is predicted to form in the subsequent temporal unit if $S_{u,v}$ reaches or surpasses a predefined threshold. The performance of these algorithms on selected datasets is quantified using the Area Under the receiver operating characteristic Curve (AUC). The AUC metric gauges the probability that a randomly selected existent link is assigned a higher score $S_{u,v}$ compared to a randomly chosen non-existent link. For calculating AUC, the scores of existent (True Positive, TP) and non-existent (False Positive, FP) links are contrasted across n independent comparisons. If n_1 comparisons yield a higher score for existent links and n_2 comparisons result in equal scores, the AUC is computed as follows:

$$AUC = \frac{n_1 + 0.5n_2}{n} \tag{10}$$

An effective link prediction algorithm is characterized by an AUC value approaching 1.

On the other side, to evaluate the performance of the CUDA implementation, we compare the results collected from a Python-based module and a C/C++ CUDA module using two factors, speedup and P-estimated.

$$Speedup = \frac{T_s}{T_p} \tag{11}$$

$$P - estimated = \frac{(1/Speedup) - 1}{(1/N) - 1} \tag{12}$$

such that T_s, T_p are time consumed in running serial and parallel programs respectively, N is the number of threads per block used in CUDA's kernel.

V. Results

A. Dataset

In this experiment, two different social network datasets (Facebook) have been used. The first set of datasets [12] represents a collection of anonymized user networks from Facebook, where nodes represent individuals and edges denote friendships. Each node comes with obscured attribute vectors to maintain user privacy. The structure enables the analysis of social patterns without revealing personal details. The second set of datasets [13] represents blue verified Facebook page networks of different categories. Nodes represent the pages and edges are mutual likes among them. The nodes been reindexed to achieve a certain level of anonimity. The csv files contain the edges – nodes are indexed from 0, including 8 different distinct types of pages. These are listed below. For each dataset, we listed the number of nodes and edges, see Table. I.

B. Discussion

Table. I list all results for implementing Python-based and CUDA-based C++ link prediction implementation for the common neighbor metric. In the code, we also programmed the rest listed metrics in section II-C. Two different sizes of networks were incorporated in the analysis. Apparent progression was achieved on all networks.

Considering the performance in Fig. 2 reveals interesting patterns in the relationship between network characteristics (such as the number of nodes and edges) and the computational speedup achieved with different thread counts in parallel processing. For instance, networks like 'new_sites_edges' and 'company_edges', which have a relatively larger number of nodes and edges, show a significant increase in speedup, particularly with higher thread counts (256 and 1024 threads). This suggests a strong scalability in larger networks when utilizing more threads. Fig. 3 presents a heatmap illustrating the correlation between the features of the network and the achieved speedup.

TABLE I

PERFORMANCE COMPARISON BETWEEN SERIAL EXECUTION (IMPLEMENTED IN PYTHON) AND CUDA (IMPLEMENTED IN C++) FOR COMMON NEIGHBOR METRIC. *HTTPS://SNAP.STANFORD.EDU/DATA/GEMSEC-FACEBOOK.HTML, **HTTPS://SNAP.STANFORD.EDU/DATA/EGO-FACEBOOK.HTML

Networks	Nodes	Edges	Serial	64 Threads			256 Threads			1024 Threads		
				Time (sec)	Speedup	P-est	Time (sec)	Speedup	P-est	Time (sec)	Speedup	P-est
new_sites_edges, *	27917	206259	2574.572	4.73	543.80	1.014	2.76	933.62	1.003	4.32	595.52	0.999
company_edges, *	14113	52310	583.284	0.61	953.33	1.015	0.37	1579.25	1.003	0.56	1047.38	1.000
athletes_edges, *	13866	86858	593.140	1.04	571.82	1.014	0.64	930.44	1.003	0.95	623.48	0.999
public_figure_edges, *	11565	67114	412.548	0.56	733.09	1.014	0.34	1211.20	1.003	0.51	811.54	1.000
government_edges, *	7057	89455	143.786	0.55	261.24	1.012	0.34	420.07	1.002	0.64	224.16	0.997
politician_edges, *	5908	41729	104.992	0.20	532.56	1.014	0.16	666.27	1.002	0.38	278.53	0.997
tvshow_edges, *	3894	17262	46.663	0.05	952.06	1.015	0.03	1502.83	1.003	0.05	893.22	1.000
facebook_107, **	1034	53498	3.110	0.03	95.38	1.005	0.02	128.83	0.996	0.04	82.60	0.989
facebook_1684, **	786	14024	1.688	0.01	164.32	1.010	0.01	224.82	0.999	0.01	161.84	0.995
facebook_1912, **	747	60050	1.765	0.03	67.94	1.001	0.02	73.38	0.990	0.03	57.04	0.983
facebook_3437, **	534	4813	0.725	0.003	248.02	1.012	0.003	253.43	1.000	0.003	245.24	0.997
facebook_0, **	333	2519	0.261	0.001	196.94	1.011	0.001	238.58	1.000	0.002	118.90	0.993
facebook_348, **	224	3192	0.119	0.001	81.27	1.003	0.001	90.83	0.993	0.003	47.10	0.980
facebook_686, **	168	1656	0.065	0.001	65.69	1	0.002	38.26	0.978	0.001	63.26	0.985
facebook_414, **	150	1693	0.052	0.001	66.76	1.001	0.001	37.88	0.977	0.001	67.24	0.986
facebook_698, **	61	270	0.008	0.002	4.80	0.804	0.001	6.53	0.850	0.0004	20.52	0.952
facebook_3980, **	52	146	0.006	0.0004	15.69	0.951	0.001	5.16	0.809	0.001	5.35	0.814

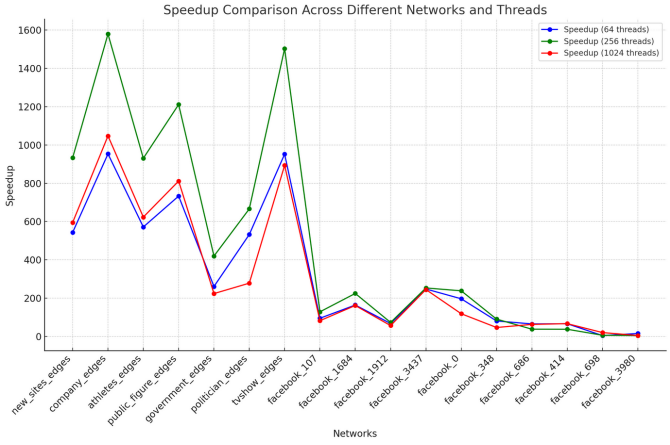

Fig. 2. Performance Trends in CUDA Parallelization: speedup achievements across diverse networks with 64, 256, and 1024 threads per block. The graph distinctly illustrates the varying scalability of different network data sets, highlighting the significant gains in speedup with higher thread counts in certain networks, while also revealing the diminishing returns or lesser impact in others.

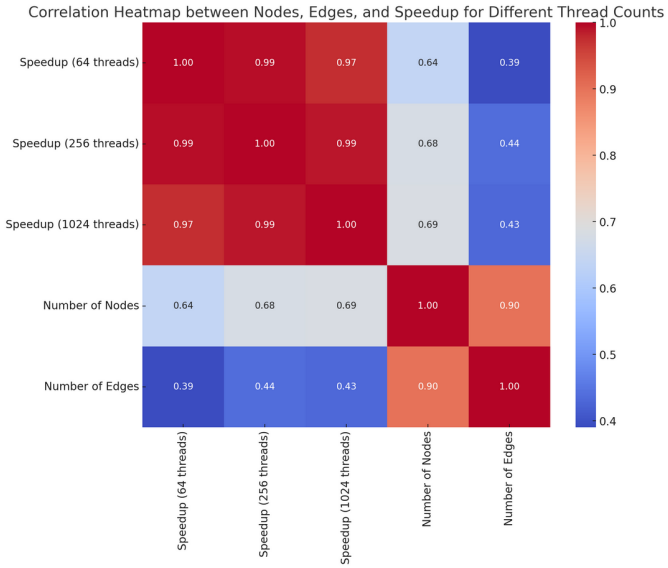

Fig. 3. Heatmap of Correlations: understanding the relationship between network size (Nodes, Edges) and speedup across 64, 256, and 1024 block threads. This figure highlights the degree of association between network characteristics and CUDA parallelization performance.

In contrast, smaller networks, such as those in the 'facebook_*' series, exhibit a varied response to increased thread counts. Some show moderate improvements or even diminishing returns at higher thread counts, indicating that the efficiency of parallel processing may depend on the specific structure and complexity of the network, beyond just its size.

Furthermore, the variation in speedup across different networks at the same thread count implies that factors like the distribution of edges and the nature of connections within the network also play a crucial role in determining the effectiveness of parallelization strategies. This data could be crucial in optimizing parallel computing approaches for different types of network structures and sizes.

VI. CONCLUSION

This paper has highlighted the considerable impact of applying parallel computing techniques to graph link prediction. Our experiments focused on evaluating node similarity through common neighbors, revealing notably enhanced speedup, particularly in large-scale graphs. A clear correlation was observed between speedup and both the number of nodes and the volume of missing edges. Notably, kernels configured with 256 threads per block outperformed others in terms of speedup. These findings pave the way for further research into broader applications and potential integration with existing software platforms, such as Gephi. The resources for this work, including code, are available at https://github.com/almusawiaf/LinkPredictionInCUDA.

REFERENCES

[1] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[2] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[3] V. Martínez, F. Berzal, and J.-C. Cubero, "A survey of link prediction in complex networks," *ACM computing surveys (CSUR)*, vol. 49, no. 4, pp. 1–33, 2016.

[4] M. E. Newman, "Clustering and preferential attachment in growing networks," *Physical review E*, vol. 64, no. 2, p. 025102, 2001.

[5] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social networks*, vol. 25, no. 3, pp. 211–230, 2003.

[6] T. Zhou, L. Lü, and Y.-C. Zhang, "Predicting missing links via local information," *The European Physical Journal B*, vol. 71, no. 4, pp. 623–630, 2009.

[7] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[8] P. Jaccard, "Etude de la distribution florale dans une portion des alpes et du jura," *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, no. 142, pp. 547–579, January 1901.

[9] T. Sorensen, "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," 1948.

[10] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási, "Hierarchical organization of modularity in metabolic networks," *science*, vol. 297, no. 5586, pp. 1551–1555, 2002.

[11] E. A. Leicht, P. Holme, and M. E. Newman, "Vertex similarity in networks," *Physical Review E*, vol. 73, no. 2, p. 026120, 2006.

[12] J. Leskovec and J. Mcauley, "Learning to discover social circles in ego networks," *Advances in neural information processing systems*, vol. 25, 2012.

[13] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton, "Gemsec: Graph embedding with self clustering," 2018.