Distributed Real Time Link Prediction on Graph Streams

Satya Katragadda*, Raju Gottumukkala¶, Murali Pusala†, Vijay Raghavan ‡, Jessica Wojtkiewicz*
*¶*Informatics Research Institute,* † *School of Computing,*‡*Center for Visual and Decision Informatics,* ‡¶*College of Engineering*
University of Louisiana at Lafayette
Lafayette, USA
*satya, ¶raju, †murali, ‡raghavan, §jessicaw1@louisiana.edu

***Abstract*—Link prediction refers to estimating the likelihood of a link appearing in the future based on the current status of a graph. Link prediction problem applications in various domains such as bioinformatics, social network analysis, cybersecurity and e-commerce. Some of these graphs are massive and are constantly evolving. Many applications require these graph streams to be processed them in real-time, to predict the link based on the most recent information as the graph features may change over time. Existing approaches to process large graphs for link prediction is non-trivial due to the following reasons: 1) Graphs required to predict the links are too large to be stored in a single RAM. Link prediction on these large graphs is expensive in terms of computation resources and time required to perform link prediction, 2) Sketch-based approaches are not suitable in applications where accuracy is critical (such as analyzing criminal social networks or supply chain networks) and 3) Sketch-based approaches also fail to handle dynamic graphs, where edges are not only added, but also removed. This results in changes to the graph topology, making the features previously computed to be obsolete. Distributed data stream frameworks such as Apache Flink could be potentially used for distributed graph processing. However, there are no techniques to handle link prediction on distributed graph streams. In this paper, we consider three fundamental, neighborhood-based link prediction measures, Jaccard coefficient, Preferential attachment, and common neighbors and enable an accurate measurement of them to address link prediction problem in dynamic graph streams. We propose a neighborhood-centric graph processing approach to handle graphs that exploits the locality, parallelism, and incremental computation of existing distributed frameworks to calculate these graph features with exact results. We perform experimental studies on various real-world graph streams. The results demonstrate that our graph measures are accurate and are more efficient than the existing vertex-centric approaches to graph processing.**

***Index Terms*—Dynamic graph streams, real-time link prediction, neighborhood-based processing, Apache Flink**

I. Introduction

Graphs are traditionally used to represent relationships between data in many data mining applications. The relationships include interactions or dependencies between various entities. One classic example of graph mining is the link prediction problem. Given a snapshot of a graph *G(t)* at the time t, link prediction problem aims to infer new edges that will appear in a graph snapshot $G(t_i)$ within the time interval $[t, ti]$, where $t < t_i$. Link prediction is an important problem in the data mining and the network science community. The link prediction problem has practical implications in the domains of social networks [1], bioinformatics [2], and the world wide web [3]. Existing link prediction algorithms use graph snapshots to design supervised or unsupervised models to predict new links. These snapshots are assumed to either fit in memory or disk and are readily available for graph computation in a standalone or a distributed environment [4].

Rapid growth in the availability of data has made the graphs too massive to be maintained in the main memory or the disks. For example, a social network graph with millions of users and billions of user-interactions, web history, and sharing history are too big to fit in the memory or disk of a single machine. In addition, these networks are often not static, instead evolve dynamically at an unprecedented rate. Majority of the applications where link prediction is used are time sensitive and it is beneficial to predict links in real-time [5].

Existing algorithms fail to address link prediction on graph streams due to three main reasons. First, link prediction algorithms rely on static graph snapshots. However, link prediction problem is a dynamic problem. The edges or link formed more recently have a higher impact on the way new links are formed compared to older relationships. Thus, using a static snapshot of data ignores the recency and seasonal patterns of data that can be captured in real-time. Second, a graph stream contains massive amounts of data that are too large to be materialized in memory of disks. Even in a distributed data processing environment, multi-pass traversals are required for any non-trivial graph computation. This presents major challenges in the areas of data partitioning, communication management, and computation parallelization. Third, majority of the graph streams are dynamic, i.e. these graph streams not only include insertion of edges, but also include deletion of edges and nodes. Thus, approaches like sketching on graphs streams that approximate graph features in real-time are not applicable on dynamic graph streams.

In order to develop a link prediction on dynamic real-time graph streams, the following challenges need to be addressed:

- Near Real-time Prediction: Dynamic graph streams are characterized by high velocity of streaming edges and nodes. The changes to the graph topology should be

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

TABLE I
ABSOLUTE MEAN SQUARED ERROR FOR SKETCHES IN TEMPORAL GRAPHS

Features	**Datasets**	
	Wikipedia	**DBLP**
Jaccard Coefficient	8.32%	7.74%
Common Neighbors	3.94%	6.49%
Preferential Attachment	5.18%	8.53%

TABLE II
LINK PREDICTION ACCURACY

2*Datasets	2*Sketch based	2*actual accuracy
Wikipedia	8.38	12.96
DBLP Dataset	80.91	98.24

incorporated into the graph as soon as the data is received to predict new links in the graph in near-real time.

- Single Pass of Data: Once the data is ingested into the graph, the graph measures extracted from the data should be computed in a single pass as multiple passes increases the complexity of computations. This increases the computational time of the algorithm.
- Recency of Prediction: As new information is included into the graph; the underlying graph topology also changes. This results in new patterns of the graph continuously evolving, while old patterns fade. The link prediction algorithm should be able to extract features from recent information while ignoring historic information.
- Exact Measures: Existing sketch-based feature extraction approaches to link prediction approximate features to decrease the computation time. While this is a valid trade off in some situations, this cannot be generalized to situations like counter terrorism or cybersecurity. Table I illustrates the percentage root mean squared error on various features with sketch-based approaches proposed in earlier works on Wikipedia and DBLP datasets. Table II shows the effect of these approximate measures on the accuracy of the link prediction approach. These benchmarks are computed based using the link prediction accuracy evaluation criteria in [1].
- Link Prediction on Dynamic Graphs: The link prediction approach should be able to handle both insertion and deletion of edges in graph streams.

In this paper, we consider three elementary, neighborhood-based link prediction measures, Jaccard coefficient, common neighbors, and preferential attachment. These measures are proven to be effective in link prediction in terms of link prediction accuracy. In order to handle large scale graph computations, we use a map-reduce based programming model to process streaming graph data in a distributed environment. We use Apache Flink's inherent support for streaming operations to design a neighborhood centric graph processing approach to efficiently compute various graph features in a single pass. This is accomplished by identifying parts of the graph that undergone recent change that led to change in the graph topology. The main contributions of this work are summarized as follows:

1) We design a real-time distributed feature extraction approach for dynamic graph streams to aid in link prediction. The features extracted include neighborhood based approaches include Jaccard coefficient, common neighbors, and preferential attachment. To the best of our knowledge, this is the first work addressing the distributed link prediction problem in dynamic graph streams.
2) We design a neighborhood centric graph processing approach that offers a lower level of abstraction than existing vertex centric approaches. This approach enables us to compute various measures for a vertex and its neighborhood without being restricted to its immediate neighbors. This approach reduces the communication costs among various nodes.
3) We carryout extensive experimental studies on two real-world graphs streams. Experimental results clearly demonstrate the effectiveness and efficiency of the proposed neighborhood centric graph processing approach.

II. RELATED WORK

In this work, we review existing work from three different perspectives: 1) Link prediction on networks, 2) graph streams, and 3) Distributed graph processing.

A. Link Prediction

Traditional link prediction algorithms can be categorized into two classes: unsupervised and supervised. The unsupervised link prediction algorithms are based on various graph proximity measures or various topology-based measures [6]. Simple neighborhood-based features like the number of common neighbors, katz centrality, Jaccard coefficient, preferential attachment, and adamic-adar are used to measure node proximity in unsupervised measures [7]. Link prediction problem can also be presented as a classification problem [8].

Majority of these link prediction algorithms are designed with the assumption that the graph snapshot is readily available in the memory or the disk. This assumption does not hold for real-world graphs as these graphs are too large to maintain in the memory of a single machine. Most of the measures used for link prediction are computationally expensive to extract in a distributed environment. Additionally, a dynamic graph has to extract the features from the graph before the next update to the graph changes the topology of the graph.

B. Graph Streams

Majority of the graph applications are not static graphs, instead streaming information modelled as entities and relationships between entities. Thus, most of these relationships can be processed as streaming graphs. These real-world streaming graphs are defined by their high velocity, which require the algorithms to handle these massive streaming graphs in real-time. There has been significant interest in processing graphs that are too large to store in the memory of a single machine or disk [9]. Most of the work on streaming

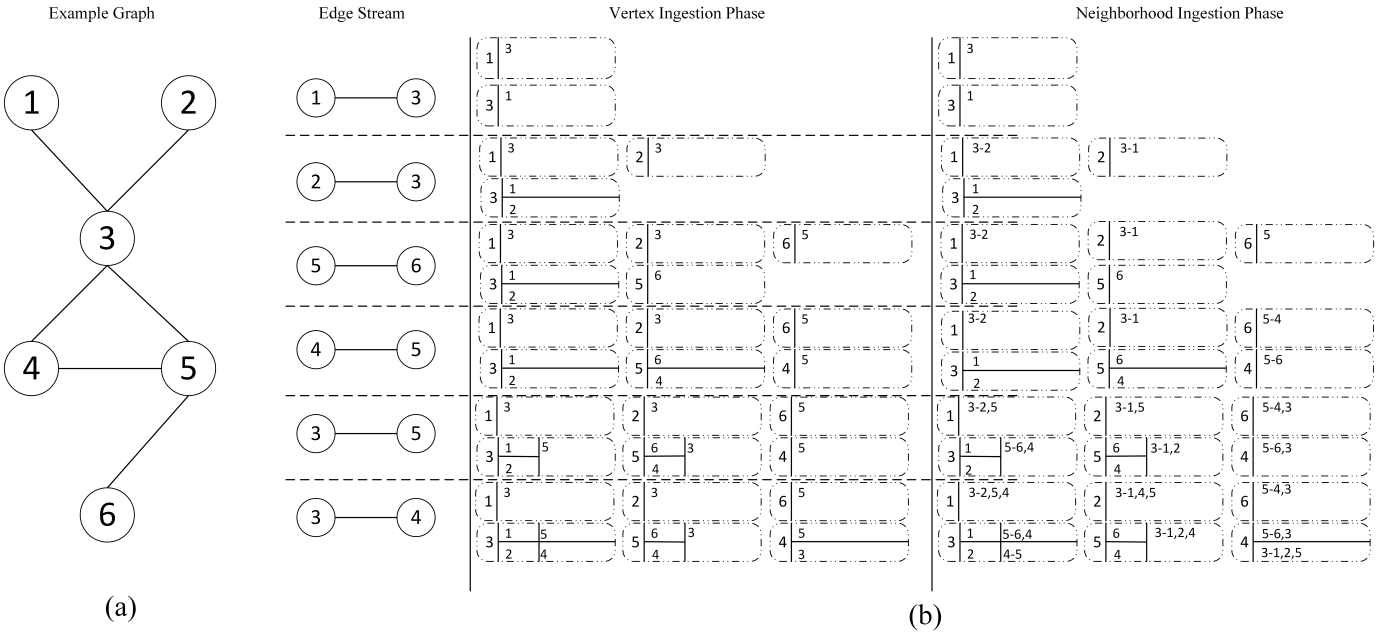

Fig. 1. Illustration of the Neighborhood Centric Approach (a) Example graph, (b) Illustration of the Neighborhood Centric approach

graphs is dedicated towards designing data structures that can summarize the observed network structure in real-time. Graph stream computations are performed on to solve the problems like graph connectivity, vertex cover, and page rank algorithms etc. [10]–[12].

Aggarwal et al. proposed one of the first approaches to cluster a graph in streaming fashion using hash-based streaming techniques [13]. A survey of streaming graph approaches is presented in [14].

The major challenge with processing graph streams is to balance the trade off between the cost of computation/storage of the graph and the accuracy of the online approximation of the sketches [4]. While majority of the proposed sketches process the graph streams in a real-time fashion, they sacrifice some of the accuracy of various graph measures. Majority of the sketches are designed for temporal graph streams and are not applicable for dynamic graph streams which also include deletion of edges.

C. Distributed Graph Processing

With the increase in the size of the graphs and availability of real-time graph streams, many distributed graph processing frameworks are proposed to efficiently process large scale graphs. In a distributed graph processing application, the graph is split into multiple partitions, where each partition is assigned to be processed on a single machine. Each partition interacts with other partitions by explicitly or implicitly passing messages about the status of the partition [15]. Various abstractions are designed to process graphs in a distributed environment. Pregel is a distributed vertex centric framework that maintains each vertex as a local state [16]. Vertices communicate with each other through messages and operations are executed synchronously at each on each vertex. A partition centric approach to distributed graph processing has been proposed to reduce the redundant communication and accelerate convergence of vertex centric programs [17].

Frameworks like GraphX and Gelly are proposed on existing distributed dataflow frameworks [18], [19]. These frameworks offer high-level APIs and libraries for graph processing on their dataflow engines. Kalavri et al. proposed a comprehensive analysis of various high-level programming abstractions for distributed graph processing [20]. The authors argue that vertex-centric model is well suited for iterative, value propagating algorithms. But, vertex centric models increase the amount of network traffic generated due to message passing. Neighborhood-centric model is designed to support operations on custom subgraphs [21]. We extend the neighborhood centric approach to handle dynamic streaming graphs. We also implement support for removal of edges and ability to perform various graph operations.

III. Methodology

We assume a standard graph stream as a sequence of edges $E = (e_0, e_1, e_2, \dots\ , e_t, \dots\ )$ of edges, where $e_i = (u, v, \tau)$, $i >= 0$, $u, v \in V$. An edge e represents an interaction between entities u and v of the underlying graph and τ represents the time when the edge is incident. The execution workflow and computer parallelization of the neighborhood-centric approach is shown in Fig. 1. Fig. 1(a) shows the current graph that we are trying to stream. The order of the edges is presented in the Fig. 1(b) along with the neighborhood graph generated. Each dashed box is a parallel executor whose key is the vertexId of the vertex in that executor. In our programming environment, we maintain two structures, vertex adjacency list and the

neighborhood graph. The vertex adjacency list contains the vertexIds and the neighborIds, i.e. vertexIds' of the neighbors. In the neighborhood graph, the 2-hop neighborhood of every vertex is stored. The vertex neighborhood is generated in the vertex ingestion phase and the neighborhood graph is Every new edge to the distributed graph processing system triggers one of the three cases:

1) No vertexId found for both of the vertices in the current set of neighborIds: This case is triggered when the edge is created between two vertices that are not either not present in the current graph or are not adjacent to any of the active vertices. Two new adjacency lists and neighborhood graphs are generated for both the vertex ids and the other vertex is added to the neighborhood of its adjacent vertex. Stages 1 and 3 in Fig. 1(b) are examples of this phenomena. The edges (1,3) and (5,6) are not attached to the existing graph structure when they are ingested into the distributed graph. The adjacency list is created during the vertex ingestion phase and the neighborhood graph is generated during the neighborhood creation phase. The neighborhood graph generated from these edges consists of a single edge that is ingested during the neighborhood graph creation process.
2) One of the vertexId is found in the neighborIds of existing vertex: This case is triggered when an ingested edge contains a previously known vertex and a new vertex. The new vertex is added to adjacency list of the existing vertex and a new adjacency list is created with for the new vertex during the vertex ingestion phase. During the neighborhood generation phase, the new vertex is added to the neighborhood graph of all vertices that are in the adjacency list of the existing vertex, and a new neighborhood graph is generated for the new vertex and all the neighbors of the adjacent vertex are added to the neighborhood of the new vertex. Stages 2 and 4 in Fig. 1(b) are an example of this stage. Edges (2,3) and (4,5) contained new vertices 2 and 4 respectively when they were ingested. These new vertices are added to the neighborhood of all the graphs that contained their neighbors 3 and 5. In addition to adding the neighbors, new neighborhood graphs for 2 and 4 are created and the current neighbors of 3 and 5 are added to the neighborhood.
3) Both the vertexIds are present in the neighborIds of existing neighborhood graphs: This case is triggered when both the vertices are connected to the existing graphs. New vertices are added to the adjacency graph of the other vertex in the edge. Then the newly added vertex is appended to the neighborhood graphs of all the other vertices in the adjacency graph. An example of this case can be found in stages 5 and 6 of Fig. 1(b).

Once a neighborhood graph is updated, the graph features for all the neighborIds in a neighborhood-based graph are computed for Jaccard Index, common neighbors and preferential attachment based on the following formula.

$$Jaccard\ index(a,b) = \frac{|\Gamma(a) \bigcap \Gamma(b)|}{|\Gamma(a) \bigcup \Gamma(b)|}$$

$$common\ neighbors(a,b) = |\Gamma(a) \bigcap \Gamma(b)|$$

$$preferential\ attachment(a,b) = |\Gamma(a)| \times |\Gamma(b)|$$

where $\Gamma(n)$ is the neighbors of node n. The features for node a and b are evaluated for the occurrence of a link in the future time period. Implementing this approach on existing graph libraries is difficult due to major issues with load balancing, resource management, and communication management. In order to solve these issues, we use a dataflow-based framework called Flink [19].

Apache Flink is a distributed general-purpose data processing platform that provides data distribution, communication, and fault tolerance for distributed computations over data streams. An application is represented by a Distributed Acyclic Graph of operations, where the data-parallel tasks are considered the vertices and the edges correspond to data flowing from one task to another. We build the proposed algorithm using Gelly, Flink's graph processing library.

IV. Results

A. Experimental Setup

We demonstrate the accuracy, efficiency and cost of the proposed approach of the proposed neighborhood centric approach in comparison with a graph sketch-based link prediction approaches from earlier works [4], [22]. The sketch-based implementations are rationally reconstructed in Java for standalone and distributed implementations based on the earlier work.

We reconstructed all our stand-alone experiments are carried out on a desktop machine with 2xIntel Xeon E5-2690 processor and 256 GB of memory running CentOS operating system. All the methods are implemented in Java. The distributed experiments are performed on 5 nodes each with 2xIntel Xeon E5-2690 processors, 256 GB of memory running CentOS operating system. The values of parameters K and L for number of minwise hash functions and reservoir budget in the graph sketch-based approaches are set to 100 for both the parameters based on earlier work in [4].

B. Datasets

We chose two real-world publicly available datasets. The edges in these datasets are tagged with temporal information on when an edge becomes active. These graphs can be processed as a stream of edges based on the timestamps provided.

1) DBLP: All the conference papers from 1956 - 2008 in the DBLP database are extracted. The database contains 590,314 authors and 597,113 papers in total. For each paper, authors are extracted, and unordered author pairs are generated. In total, 1,814,356 author pairs are generated from the co-authorship information. These author

TABLE III
RUNTIME COST (MS) TO COMPUTE GRAPH STRUCTURE IN DISTRIBUTED ENVIRONMENT ON DBLP

Approach	0.25	0.5	0.75	All
Vertex Centric Processing	103	139	162	198
Neighborhood Centric Processing	197	229	256	284

TABLE IV
RUNTIME COST (MS) TO COMPUTE GRAPH STRUCTURE IN DISTRIBUTED ENVIRONMENT ON WIKIPEDIA

Approach	0.25	0.5	0.75	All
Vertex Centric Processing	125	153	189	233
Neighborhood Centric Processing	224	251	306	342

pairs are treated as a graph stream to predict future authorship links.

2) Wikipedia: This dataset contains all articles extracted from a snapshot of Wikipedia [23]. The vertices are Wikipedia articles and there exists an edge if article i references article j. The time this link was created is added as the time-stamp to the link. The graph stream contains a stream of reference links between 1,832,117 vertices and 39,452,116 edges in total.

C. Evaluation Methods

The datasets are divided based on the criteria specified in [4]. We evaluate various algorithms based on three criteria:

1) Link prediction accuracy: The accuracy of the link prediction approaches when compared against a random classifier. The random classifier is built based on the criteria specified in [1].
2) Computation cost: We compare various approaches on the time taken to compute various measures for link prediction.
3) Space cost: The space consumed by various data structures used to compute the graph measures used in the link prediction algorithm.

V. RESULTS

A. Link Prediction

We evaluated the accuracy of link prediction for sketch based approaches and the distributed graph processing approaches that include vertex centric processing and neighborhood centric processing. The sketch-based approaches have a lower prediction accuracy compared to the distributed graph-based approaches in both DBLP and Wikipedia graph streams. The decrease in accuracy of sketch-based approaches is due to the approximations in sketch-based approaches that introduce some noise in the graph-based measures which is then translated to the link prediction accuracy. The percentage error introduced by the sketch-based methods and the effect on link prediction accuracy is presented in Tables I and II. The neighborhood centric approach computes the graph measures accurately.

TABLE V
RUNTIME COST (MS) TO COMPUTE GRAPH MEASURES ON DBLP

Approach	0.25	0.5	0.75	All
Sketch Based Processing	0.3	0.5	0.8	1.1
Distributed Sketch Based Processing	0.2	0.4	0.6	0.9
Vertex Centric Processing	73	106	142	197
Neighborhood Centric Processing	8	14	23	31

TABLE VI
RUNTIME COST (MS) TO COMPUTE GRAPH MEASURES ON WIKIPEDIA

Approach	0.25	0.5	0.75	All
Sketch Based Processing	0.4	0.7	0.8	1.1
Distributed Sketch Based Processing	0.3	0.6	0.7	0.9
Vertex Centric Processing	128	173	238	297
Neighborhood Centric Processing	9	16	27	37

B. Computation Cost

An important factor to evaluate link prediction in dynamic graph streams is the time required to extract features from the graphs. Link prediction algorithms should be able to compute graph measures in real-time. We compare the performance of the sketch-based link prediction on a desktop, the sketch-based implementation in a distributed environment, vertex centric processing and neighborhood centric processing. While the sketch-based link prediction is online in a desktop environment, the distributed variation of it requires that the vertex of a graph be materialized for the sketches to be computed. In order to enable a fair comparison, of the distributed link prediction models, we compare the cost involved to materialize the graph in a distributed environment. We then compare the time taken to compute all three measures to perform link prediction for a pair of vertices.

In Table III, we present the average run-time cost to ingest an edge into the graph structure in distributed processing for 25%, 50%, 75%, and total edges in the DBLP graph stream. The computation time includes the time taken to identify individual vertices and then add the current edge to existing vertices on vertex centric processing. The time for neighborhood centric processing includes the time taken to ingest an edge into all the neighborhood sub-graphs that the vertices of the edge appear. The time taken to build the vertex centric graph structure is lower compared to the time required to build a neighborhood-based structure for both the graph streams. This is due to the multiple map operations for a single edge in neighborhood centric processing compared to 2 map operations in vertex centric processing. The increased time to process Wikipedia graph datastream compared to DBLP graph datastream is due to high percentage of edges per vertex in Wikipedia data stream compared to DBLP graph stream. Table IV shows the average run-time cost to ingest the streams Wikipedia data streams.

In Table V and VI, we report the average time taken to compute Jaccard coefficient, preferential attachment, and number of common neighbors per vertex pair on DBLP and Wikipedia graph streams. The table shows the computation time for graph streams comprising of 25%, 50%, 75% and

all the edges of the total graph stream. Here, distributed sketch-based approach computes the features faster than sketch computation on a desktop.

The true cost of feature measurement should also include the time taken to compute the graph structure. If the time taken to compute the vertex structure is considered, the sketch-based approach is more efficient than the other approaches. As mentioned previously, the approximations of graph measures reduce the accuracy of link prediction. Neighborhood centric processing is more efficient than vertex centric processing even when considering the time taken to materialize the local graph structure.

C. Space Cost

The space complexity of neighborhood centric processing is $O(n + nd^2)$, where n is the number of vertices and d is the average degree of the current graph. The space complexity of the vertex centric processing is $O(n + nd)$. The space used to compute a 2-hop neighborhood for every vertex increases the space required by the graph exponentially. However, distributed computing machines are built to handle huge datasets. With the increased accuracy of link prediction, and minor increase in time taken to extract graph measures, we think extra space is a valid trade off while computing link prediction in a distributed environment.

VI. Conclusion and Future Work

In this paper, we studied the distributed link prediction problem in dynamic graph streams. Graph streams have a variety of applications in real-world in the domains of social sciences, recommender systems, bioinformatics, and security. In this paper, we demonstrated how basic link prediction measures like Jaccard coefficient, Preferential attachment, and common neighbors can be computed exactly and efficiently in a distributed environment. We designed a neighborhood centric graph processing approach based on partition centric approaches in distributed computing to materialize the 2-hop neighborhood of a graph in order to readily extract features from a graph. Our experimental studies demonstrated that the proposed approach enables the graph measures to be computed at that time period without significant loss in efficiency.

In the future work, we plan on extending the neighborhood centric approaches to include various path based features to improve the link prediction accuracy. Another direction of research is to reduce the space taken by the neighborhood subgraphs by bit mapping the neighborhood links without sacrificing the efficiency.

Acknowledgment

This work is supported by the National Science Foundation under grant numbers CNS-1429526 and CNS-1650551.

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] J. Menche, A. Sharma, M. Kitsak, S. D. Ghiassian, M. Vidal, J. Loscalzo, and A.-L. Barabási, "Uncovering disease-disease relationships through the incomplete interactome," *Science*, vol. 347, no. 6224, p. 1257601, 2015.

[3] Y. Dong, J. Zhang, J. Tang, N. V. Chawla, and B. Wang, "Coupledlp: Link prediction in coupled networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 199–208.

[4] P. Zhao, C. Aggarwal, and G. He, "Link prediction in graph streams," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 2016, pp. 553–564.

[5] W. Yu, C. C. Aggarwal, S. Ma, and H. Wang, "On anomalous hotspot discovery in graph streams," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 2013, pp. 1271–1276.

[6] C. C. Aggarwal, "On classification of graph streams," in *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 2011, pp. 652–663.

[7] P. Sarkar, D. Chakrabarti, and A. W. Moore, "Theoretical justification of popular link prediction heuristics." in *IJCAI proceedings-international joint conference on artificial intelligence*, vol. 22, no. 3, 2011, p. 2722.

[8] N. Z. Gong, A. Talwalkar, L. Mackey, L. Huang, E. C. R. Shin, E. Stefanov, E. R. Shi, and D. Song, "Joint link prediction and attribute inference using a social-attribute network," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 5, no. 2, p. 27, 2014.

[9] A. McGregor, "Graph stream algorithms: a survey," *ACM SIGMOD Record*, vol. 43, no. 1, pp. 9–20, 2014.

[10] S. Guha, A. McGregor, and D. Tench, "Vertex and hyperedge connectivity in dynamic graph streams," in *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. ACM, 2015, pp. 241–247.

[11] R. Chitnis, G. Cormode, M. T. Hajiaghayi, and M. Monemizadeh, "Parameterized streaming: Maximal matching and vertex cover," in *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2014, pp. 1234–1251.

[12] A. D. Sarma, S. Gollapudi, and R. Panigrahy, "Estimating pagerank on graph streams," *Journal of the ACM (JACM)*, vol. 58, no. 3, p. 13, 2011.

[13] C. C. Aggarwal, Y. Zhao, and P. S. Yu, "On clustering graph streams," in *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM, 2010, pp. 478–489.

[14] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, p. 10, 2014.

[15] P. Strandmark and F. Kahl, "Parallel and distributed graph cuts by dual decomposition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2085–2092.

[16] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[17] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From think like a vertex to think like a graph," *Proceedings of the VLDB Endowment*, vol. 7, no. 3, pp. 193–204, 2013.

[18] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013, p. 2.

[19] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[20] V. Kalavri, V. Vlassov, and S. Haridi, "High-level programming abstractions for distributed graph processing," *IEEE Transactions on Knowledge & Data Engineering*, no. 1, pp. 1–1, 2018.

[21] A. Quamar, A. Deshpande, and J. Lin, "Nscale: neighborhood-centric large-scale graph analytics in the cloud," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 25, no. 2, pp. 125–150, 2016.

[22] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[23] A. E. Mislove, "Online social networks: measurement, analysis, and applications to distributed information systems," Ph.D. dissertation, Rice University, 2009.