The Link Regression Problem in Graph Streams

Bowen Dong
University of Illinois at Chicago
Chicago, IL, USA
bdong5@uic.edu

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown, NY, USA
charu@us.ibm.com

Philip S. Yu
University of Illinois at Chicago
Chicago, IL, USA
psyu@uic.edu

***Abstract*—We will study the problem of analyzing massive data streams in the context of the dynamic network-centered activities. Consider a network in which a time series activity data stream is associated with each edge in a massive network. Examples of such activities could include citation networks with continuously changing edge values, such as road network traffic, social network traffic, or email traffic. We introduce the problem of *link regression*, which models the prediction of the future stream *values* associated with a link from the history of all the stream values associated with the different links. This problem is a powerful generalization of both the large scale time series prediction problem and the link prediction problems in network analysis, each of which are known to be quite difficult in their own right. The generality of this framework provides it significant applicability for traffic analysis in road networks, congestion analysis in communication networks, interest trend analysis and bursty feature analysis in social networks. The problem is very challenging because of its massiveness both in network size and stream speed. We present an algorithm which uses streaming graph partitioning in conjunction with predictive regression. We present experimental results illustrating the effectiveness and efficiency of the approach.**

***Index Terms*—graph mining, data stream, regression**

I. Introduction

This paper introduces the *link regression problem*, which is defined as follows. Consider a network in which each link is associated with a continuous variable, which is updated in time. Thus, each link is associated with a time series data stream variable. We would like to predict the values on the links at some future period in time from the current history of time series stream values. Some examples of relevant applications are as follows:

- In a who-cites-whom citation network, the numbers on the edges represent the number of times that one author cites the other in a specified time period.
- In a co-authorship network, the numbers on the edges represent the number of times that one author collaborates with the other in the last time period.
- In road networks, the edges represent the road segments between the different junctions (nodes). The real values on the edges represent the traffic. Communication network traffic can be modeled in a similar way. It may be desirable to predict the traffic at a specific link (or congestion regions) in the future.
- In social networks, the edges represent the friendship links (or follower links), and the real values could represent the fraction of the messages (or tweets) which are related to a specific subject of interest. At any given point in the future, it may be desirable to predict the set of links which are most related to a specific subject of interest.

Our formulation for the link regression problem is fairly general, and two of its special cases are well known (and difficult) problems from the diverse domains of streaming and social network analysis respectively. Specifically, the two special cases are as follows:

- **Large Scale Multi-variate Time-Series Data Stream Prediction:** In this problem [18], [23], [29], [30], [33], we have a set of time series data streams, and it is desirable to predict one of more values of the stream, based on the past stream history. This problem can be modeled as a link-regression problem on a graph in which all edges are disconnected from one another.
- **Link Prediction Problem:** In this problem [3], [6], [9], [15], [16], [19], [25], [27], we we wish to predict future *formation* of links in the network. Thus, the approach is a binary decision problem. Furthermore, a number of temporal simplifications are typical. The prediction is typically not specific to a particular instant in time. Rather, the highest ranked node pairs are determined, between which links are likely to form *at any point in the future*. The model is typically not streaming either. Such convenient simplifications are not available in the case of the link regression problem.

Both the afore-mentioned problems are quite challenging in their own right. For example, the works in [18], [22], [29], [30], [33] deal with only hundreds or thousands of streams in the simplified (non-structural) scenario. On the other hand, realistic road networks, communication networks or social networks may have edges which could number in the millions or billions. Thus, the setting is more challenging not only from the perspective of the incorporation of structural information, but the typical scales of the problem in such domains are larger by *several orders of magnitude*. The structural information can however work both ways; while it increases the complexity of the problem, it also provides useful hints for the regression process. As is evident from problem of link prediction in social networks, structural locality plays an important role in prediction of edge values. In most reasonable applications of this problem that we can envision, structural proximity is also likely to play a key role. However, the correlations across different edges with typically vary with problem domain and data set; therefore, unlike the link prediction problem (in which specific structural measures are hard coded), the structural correlations between edges need to be *continuously learned* from the past history of the incoming stream. Furthermore, most applications of the link regression problem are quite different from link prediction, in that the predictions in this problem typically need to be made in real time in a fast evolving stream scenario. Thus, the scale of the problem adds to the complexity created by the use of structural information in the modeling process. In general, the streaming setting is known to be particularly difficult in the context of graphs [5].

This paper is organized as follows. The next section discusses a model for the link prediction problem. An algorithm for link prediction is proposed in section 3. The experimental results are presented in section 4. Section 5 contains the conclusions and summary.

A. Related Work

The problem of multi-variate time series regression has been studied widely in the stream mining literature [18], [29], [30], [33]. Most of these methods use scalable multi-variate regression analysis in order to predict the different stream values from one another, and they do not use structural locality for time series prediction. The problem of link prediction has also been studied extensively in the data mining and machine learning community [8]. Much of the relevant work on this problem is based on defining proximity-based measures on the nodes in the underlying network [2], [16], [17], [32]. The work in [16] studied the usefulness of different structural and topological features for link prediction. A second approach is to study the problem in the context of statistical relational models [1], [7], [10], [11], [13], [19], [24], [26], [31]. These methods use content information at nodes, and are therefore not relevant to our scenario.

The link prediction problem has also been studied more generally in the context of the classification problem [6], [15], since the link prediction problem can be considered a classification problem in which features and class labels (corresponding to existence or absence of links) can be associated with links to be predicted. In this paper, we will study the link regression problem, in which the goal is somewhat different, since it is desired to predict the quantitative values on the links.

II. Link Regression Model

The temporal network is denoted by $G(t) = (N(t), A(t), X(t))$ with node set $N(t)$ and edge set $A(t)$. The set $X(t)$ represents the streams associated with the different edges. Each edge $(i, j) \in A$ is associated a continuous series of real values $x_1^{ij} \ldots x_t^{ij} \ldots$, where the value x_t^{ij} is the value at the edge (i, j) at the tth discrete clock tick. The granularity of the clock ticks may depend on the application at hand, but it is generally assumed that it is fine-grained enough to accurately capture the trends in the underlying data streams.

One challenge with this problem is that the volume of the incoming data may be very large. For example, in a network containing a billion edges and a time-granularity of 1 minute, a single day of stream data requires more than 10^{12} values, which is clearly beyond the capabilities of most commodity hardware. In other words, the algorithms need to be *online*, and be able to work with modest space for predictive purposes. The link regression problem may be modeled as follows:

Definition 1 (Link Regression Problem): Given the temporal activity network $G(t_c) = (N(t_c), A(t_c), X(t_c))$ at current time t_c, predict the value x_t^{ij} on the edge (i, j) at future time t.

This problem could be formulated in multiple ways, depending upon the underlying application. For example, one could formulate the problem in terms of finding specific edges which satisfy certain threshold values. Some examples of query variations, which can be resolved using link regression are as follows:

- Determine all edges (i, j) for which the value of x_t^{ij} is above a given threshold α at future time t.
- Determine all connected subgraphs formed by the subset of edges (i, j) underlying edge values $x_t^{ij} \geq \alpha$ at future time t.

Clearly, these queries can be resolved relatively easily, as long as a method to solve the link regression problem is available. In the next section, we will describe the *NetRegress* approach, which performs link regression with the use of a combination of structural and traditional regression analysis in the streaming context.

III. The NetRegress Approach

One well known method for streaming regression is the SPIRIT approach [18]. This approach attempts to determine the hidden variables from the underlying data stream, and make predictions on this basis. However, the approach does not use the network structure, and in fact does not scale well with increasing number of variables. This is because the complexity of Principal Component Analysis (PCA) increases at least quadratically with increasing data dimensionality. It turns out that the network structure is useful information, which can be used in order to localize the regression process, and improve both the efficiency and the quality of the results.

Broadly speaking, the *NetRegress* approach uses a number of methods which are continuously performed over the data in order to perform the prediction.

- Edge clusters are continuously maintained over the data stream. These edges clusters are maintained on the basis of structural locality. The edge clustering is allowed to change periodically, as the clusters change over the data stream.
- For each edge cluster, we continuously perform streaming PCA. The idea is that edge clustering has already localized the variables which are most relevant to the prediction process. These are used in order to maintain the hidden variables in the data stream. Note that this structural localization decomposes the prediction process into smaller subproblems. These smaller subproblems can be more easily tackled since the size of the PCA matrix is much smaller for each subproblem. Furthermore, the relevant streams have been selected for each structural locality. This makes the overall analysis much sharper.
- Auto-regressive methods are used to perform prediction on the key hidden variables. These predicted hidden variables are then converted back to the original streaming values. The idea of performing prediction over the hidden variables corresponding to the largest principal components is that the noise effects are often removed, and they are much easier to predict than the individual values.

In this section, we will discuss each of these methods in detail. First, we will introduce some further notations and definitions.

A. Maintaining Edge Clusters

The edge clusters are maintained using an approach which is a modification of that discussed in [4]. The work in [4] uses a structural reservoir sampling approach in which the edge samples are used for the clustering process. One difference with the method in [4] is that the work is designed for the case of object streams, whereas we need to use it here for the case of continuously arriving edge streams. It is important to note that the clustering need not be perfect, as long as it approximately groups the edges in the different localities. The goal of the clustering is to group edges in different dense localities together, since the edge values in a particular structural locality are correlated with one another. In such cases, it suffices to define a clustering which is based on randomized sampling of the edges in the network. The clusters on the nodes are defined in terms of the connected components on the sampled edges. Then the node clusters are converted into edge clusters over *all* edges.

Since the connected components in edge samples are known to create weak clusters of densely connected nodes [14], such an approach is an excellent strategy for localizing the prediction process with the use of structural information. It should be pointed out that the work in [4] is designed for cases where the edges are unweighted, and may arrive at random points in the stream. In this case, the edges have weights denoted by x_t^{ij}. However, in this case, we are only interested in tracking structural locality on the basis of the behavior of edges which have non-zero weight. These can however be converted to the case, where $x_t^{ij} > 0$ corresponds to an edge weight of 1, whereas $x_t^{ij} = 0$ corresponds to an edge weight of 0. In some kinds of networks such as road networks, the edges stay constant over time, whereas in other kinds of networks such as citation networks, the edges may vary considerably with time. Nevertheless, since the values of x_t^{ij} are highly correlated with one another over consecutive time stamps, the overall structure of the network, which corresponds to edges with values of $x_t^{ij} > 0$ does not change significantly with time. In our case, for the purposes of structural clustering, we treat this as an edge stream in which an edge appears only for the first time, that it changes from $x_t^{ij} = 0$ to $x_t^{ij} > 0$. This is our re-constructed structural stream, which is useful for structural localization. Therefore, the volume of the edge stream is quite modest, since each edge changes exactly once from $x_t^{ij} = 0$ to $x_t^{ij} > 0$ *for the first time.* This means that each stream element is distinct. The total number of edges in this constructed stream is equal to the number of distinct edges in the network. Note that such a network stream would initially receive a large number of edge elements during the construction phase, but would gradually stabilize over the course of a larger time horizon. However, the level of stabilization may vary considerably depending upon the underlying application. For example, for a citation network, new edges will always be constantly formed over time at a steady-state rate. For a road-network on the other hand, the "formation" of edges slows down, as the structure of the mature network is learned with arrival of new edges.

A fast and online method is needed in order to create and maintain the dense partitions in this evolving network. It is well known that edge sampling [14] can be used to create dense partitions. For example, a sample of edges from a stream implicitly creates a set of clusters in terms of the connected components in this sample. Such connected components are much denser than randomly picked node sets in the graph, because of the inherent bias of edge sampling [14]. In this case, we also need to maintain certain structural properties, such as ensuring a minimum number of points in each cluster, or constraining the total number of clusters.

Furthermore, the approach needs to be adapted to the streaming scenario. While reservoir sampling [28] can dynamically maintain an unbiased sample from a stream of elements, it is not directly suited for structured graphs. In our case, it is desired to determine an unbiased sample of a structured graph, so that many natural and desirable structural properties of the sample are maintained. For example, in the case of our application, it is desirable for the clusters to be relatively balanced. This goal is achieved with the help of a monotonic set function of the underlying edges in the reservoir. A monotonically non-decreasing (non-increasing) set function is a function $f(\cdot)$ whose argument is a set, and value is a real number which always satisfies the following property:

- If S_1 is a superset (subset) of S_2, then $f(S_1) \geq f(S_2)$

The monotonic set function can be useful for regulating the structural characteristics of the graph over a given set of edges. In our specific case, we set the function value to the number of nodes in the the largest connected component in edge set S. This structural component is isolated for the analysis process.

Let $\mathcal{D}$ be a sorted set of edges. These edges can be added to S only in sort order priority; in other words, an edge cannot be included in a subset S, if all the elements which occur before it in the sort order are also included. Clearly, the number of such subsets of $\mathcal{D}$ is linear in the size of $\mathcal{D}$.

Definition 2 (Sort Sample with Stopping Criterion): Let $\mathcal{D}$ be a set of edges. Let $f(\cdot)$ be a monotonically non-decreasing (non-increasing) set function defined on the edges. A sort sample S from $\mathcal{D}$ with stopping threshold α is defined as follows:

- All edges in $\mathcal{D}$ are sorted in random order.
- The smallest subset S from $\mathcal{D}$ among all subsets which satisfy the sort-order priority is picked, such that $f(S)$ is at least (at most) α.

Such a set satisfies a minimality property. This means that if the last added element is removed, then that set (and all previous subsets) will not satisfy the stopping criterion. As a practical matter, the set which is obtained by removing the last added element is the most useful for processing purposes. For example, if $f(S)$ is the size of the largest connected component, the stopping criterion determines the smallest sample S, such that the size of the largest connected component is at least a user-defined threshold α. By dropping the last edge (v, w) which was added to S, it is guaranteed that the size of the largest connected component in $S - \{(v, w)\}$ is less than α. Correspondingly, a penultimate set for a sort sample is defined as follows.

Definition 3 (Penultimate Set): The penultimate set for a sort sample S is obtained by removing the last element in the sort order of sample S.

In the case of a static data set, this is achieved by sorting the edges in random order and adding them sequentially, until the stopping criterion can no longer be satisfied. However, in the case of a data stream, a random sample or reservoir needs to be maintained dynamically, which is much more of a challenge. Once edges have been dropped from the sample, their sort order cannot be compared to new incoming edges.

Therefore, a fixed random hash function is used to impose a sort order on the edges, and serve as a fixed landmark in order to meaningfully compare the sort order of dropped edges with respect to newly arriving edges. This hash function is computed as a function of the node labels on the edge, and does not change over the entire stream. The idea here is that the hash function fixes the sort order among the edges throughout the stream computation. For an edge (i, j) the hash function $h(i \oplus j)$ is computed, where $i \oplus j$ is the concatenation of the node labels i and j. The use of a sort on a random hash function value induces a random sort order on the stream elements. A stopping criterion on the sorted set of edges translates to a threshold on the hash function value. In other words, it is desirable to determine the smallest threshold q, such that the set S of edges which have hash function value at most q satisfy the condition that $f(S)$ is at least (at most) α. The key observation here is that the value of the threshold q varies over the life of the data set, as more edges arrive, and a smaller fraction of the edges in the stream need to be included in the reservoir in order to satisfy the structural constraint. The smaller fraction translates to lower hash thresholds. Therefore, the stopping hash threshold is monotonic over the life of the data stream. This implies that edges which are discarded will never be relevant over the life of the data stream. This is crucial to the correctness of the approach.

The maintenance algorithm is as follows. For each incoming edge, the hash function is applied, and it is added to the reservoir, if the hash value is less than the current threshold. The addition of an edge will never result in violation of the stopping criterion because of set function monotonicity. However, the set may no longer be the smallest sort-ordered set to do so. Therefore, edges may need to be removed to make it the smallest sort-ordered set to satisfy the stopping criterion. In order to achieve this goal, the edges in the reservoir are processed in decreasing order of the hash function value and removed, until the resulting reservoir is the smallest possible set which satisfies the stopping constraint. The corresponding hash threshold is then reduced to the largest hash function value of the remaining edges in the reservoir after removal. Therefore, the reservoir sample update process, at current hash threshold q, incoming edge (i, j) and hash function $g(\cdot)$ is as follows:

```
Compute hash value $g(i \oplus j)$;
**if** $g(i \oplus j) < q$ add $(i, j)$ to $S$;
Remove edges from $S$ with largest hash value
      until stopping criterion is met;
Reset threshold $q$ to largest hash-value in $S$;
```

The removal of edges may result in a reduction (or no change) of the hash threshold in each iteration. The penultimate set derived from the sort sample is always used for the purposes of maintaining the cluster partitions as the set of underlying connected components. This approach continuously creates the edge partitions in the network, which are indicative of structural locality. These node partitions can be used in order to create the local structurally clustered streams which are used for the link regression process.

```
**Algorithm** *MaintainStatistics*
**begin**
  **repeat**
    Receive next edge $(i, j)$;
    Update reservoir sample from next edge;
    Update node clusters from updates in
          reservoir sample;
    Update edge clusters from node clusters
          which have changed;
    Update covariance matrix statistics;
  **until**(end of stream);
**end**
```

Fig. 1. Overall Framework for Variable Maintenance

B. Hidden Variable Maintenance

In this process, we discuss the process of hidden variable maintenance in each structural locality of the edge stream. The hidden variable maintenance process uses PCA in order to infer the key concepts in the underlying data stream. We note that the reservoir samples induce a node partitioning on the network. Specifically, the connected components in the reservoir samples correspond to the node partitioned. Each node will always belong to exactly one partition. If a node is not present in the reservoir sample, it is assumed to be isolated. The first step is to convert the node partitioning into an edge partitioning. The first step is to find an appropriate assignment for edges whose end points lie across clusters. The edges in the data are of two types:

- Both the end points of an edge lie in the same cluster. In this case, the edge is assigned to the cluster in which the two end points lie.
- The end points of the edge lie in different clusters. In this case, the edge is assigned to the smaller of the two partitions. This is done in order to balance the edge clusters in the network as much as possible. Maintaining a balanced edge clustering is particularly useful for the problem of link regression analysis.

This creates a set of edge clusters from the data. Since the structural reservoir sampling approach maintains the node clusters in online and incremental fashion, the edge clusters are correspondingly updated, whenever the node clusters are updated. It is relatively easy to update the edge clusters, by updating the cluster membership only of edges, whose node membership has changed.

At each instant, we maintain the dominant hidden variables which characterize each edge cluster. Let F be the set of edges in a structural cluster containing d edges. For brevity, let us denote the series variables on the edges in this structural cluster by $\overline{y^1} \dots \overline{y^d}$. Note that each y_t^i corresponds to some edge value x_t^{kl}. The series variable $\overline{y^r}$ contains the time stamp values denoted by $y_1^r \dots y_t^r$. Then, a PCA-based method is used in order to maintain the hidden variables at a specific time stamp. The PCA method uses the covariance matrix of the data in order to determine the hidden variables, and maintain them in real time. Specifically, a $d \times d$ covariance matrix C is maintained, in which the covariance value C_{ij} is equal to the covariance between the streams $\overline{y^i}$ and $\overline{y^j}$. This results in a covariance matrix C.

The covariance matrix can easily be maintained incrementally for a data stream. Note that the covariance between the series $\overline{y^j}$ and $\overline{y^k}$ can be expressed as follows:

$$Cov(\overline{y^j}, \overline{y^k}) = \frac{\sum_{i=1}^{t} y_i^j \cdot y_i^k}{t} - \frac{\sum_{i=1}^{t} y_i^j}{t} \cdot \frac{\sum_{i=1}^{t} y_i^k}{t}$$

The computation of the above requires the maintenance of (i) the d additive sums of the values of each variable in the stream, (ii) the d^2 additive sums of the pairwise products of the time series values in the stream, and (iii) the number of clock ticks which have elapsed so far. These can easily be maintained in an incremental setting, by maintaining variables to which the different values can be added.

The covariance matrix would be too large to maintain in online fashion, if it were to be maintained for every pair of streams in the data. This is because such a covariance matrix would scale with the square of the *total* number of streams. However, when the edges have been partitioned into clusters, covariance matrices can be maintained for the different clusters in a much more scalable way. The major challenge here is that the edge clusters sometimes change during the course of the stream computation. In such cases, some of the entries of the covariance matrix are based on a relatively small number of observations, when some of the nodes which belonged to different clusters earlier now belong to the same cluster. This is however not necessarily a problem, since most of the covariance matrix can still be accurately estimated.

The matrix C is positive semi-definite, and its eigenvectors provide the key correlations among he different variables. Specifically, the eigenvectors corresponding to the largest eigenvalues are the most important ones for deciding the important directions of correlation. The covariance matrix can be diagonalized in order to determine the eigenvectors:

$$C = P \cdot D \cdot P^T \tag{1}$$

Here the columns of P correspond to the orthonormal eigenvectors. The matrix D is a diagonal matrix containing the eigenvalues. The eigenvectors corresponding to the largest eigenvalues correspond to the key hidden variables in the data. The eigenvectors need not be recomputed from scratch, every time the covariance matrix is updated. This is because the eigenvectors from covariance matrix in the previous time-stamp, already approximate the optimal solution very closely. Therefore, a few additional von Mises iterations [12] can approximate the eigenvectors of the updated matrix very effectively in most cases. This is true for the vast majority of the time-stamps, where the dimensionality of the covariance matrices do not change, and only the values in the covariance matrix are updated. In many cases, where the covariance matrix for a particular edge cluster does not change at all, the corresponding eigenvectors do not need to be recomputed at all.

Let $\overline{e^1} \dots \overline{e^k}$ be the k eigenvectors corresponding to the largest eigenvalues. Each eigenvector $\overline{e^r}$ has d components corresponding to $(e_1^r \dots e_d^r)$. Then, the set of time-series in F can be projected in terms of these k eigenvectors in order to summarize the large number of variables in F into a relatively smaller number of independent components. Then, if the new hidden variable series be denoted by $\overline{z^1} \dots \overline{z^k}$, with $\overline{z^r} = (z_1^r \dots z_t^r)$, we have:

$$z_t^r = \sum_{j=1}^{d} y_t^j \cdot e_j^r \tag{2}$$

These new hidden variables correspond to an "aggregated" representation of the principal components in the time series in which the correlations across the different series have been removed. The idea here is that since the correlations in the data are already accounted for, by using principal component analysis, this accounts for the structural relationships within a specific locality F. Subsequently, it is possible to use auto-regression on the variables in $\overline{z^1} \dots \overline{z^k}$ in order to predict the variables at the instant $(t+1)$ with the use of auto-regressive analysis. Since PCA has already accounted for correlations across series, the autoregressive analysis can account for correlations across time. Thus, the prediction implicitly uses the predictions across different series *as well as across* different time stamps.

Let $z_1^r \dots z_t^r \dots$ be the values in the time series corresponding to each of the hidden variables at times $1 \dots t$. In the auto-regressive model, the value of z_t^r is defined in terms of the auto-correlations in the values in the last window of length p. A linear function is used in order to perform the modeling:

$$z_{t+1}^r = \sum_{i=0}^{p-1} a_i \cdot z_{t-i}^r + c + \epsilon_{t+1}$$

A model which uses the last window of length p is referred to as an $AR(p)$ model. The values of the regression coefficients $a_1 \dots a_p, c$ need to be learned from the training data, which is the previous history of the time series. A set of linear equations between the coefficients can be created, by using each time stamp in the training data, along with its immediate history of length p. When the number of time-stamps available is much larger than p, this is an over-determined system of equations. The coefficients $a_1, \dots a_p, c$ can be approximated with *least-squares regression*, in order to minimize the square-error of the over-determined system. This is a standard regression modeling problem, which can be used in order to predict the value of z_{t+1}^r for each of the k different principal components. These methods can therefore be used in order to predict the hidden variables at time $t+1$ from the variables in the previous time periods. The overall process of maintaining the different data structures for maintaining edge clusters and the covariance matrices, which are required to maintain the hidden variables are illustrated in Figure 1.

C. Final Prediction of Regression Values

Note that the aforementioned approach can be used in order to predict the hidden variables at time $t+1$, which are denoted by $z_{t+1}^1 \dots z_{t+1}^k$. These hidden variables can then be converted back to the true values with the use of the k eigenvectors. This can be achieved by observing that the hidden variables intuitively represent weights along the different eigenvectors. Therefore, the vector $\overline{V(t+1)}$ of the d predicted values for the d streams in F is given by the following:

$$V(t+1) = \sum_{i=1}^{k} z_{t+1}^i \cdot \overline{e_i} \tag{3}$$

This provides the predicted values for the d streams in F. This process is performed over each edge cluster, in order to yield a comprehensive prediction over all streams.

IV. Experimental Results

We present experimental results which illustrated the accuracy and efficiency of the method. The goal is to show that our approach is significantly more effective, while retaining competitive efficiency in spite of a more complex processing mechanism.

A. Data Sets

We used real data sets drawn from *DBLP*, *IMDB* and *Cora* respectively. The pre-processing of the data sets is described below.

DBLP data set: The *DBLP* data set contained information about papers and authors in four areas from 2000 to 2010. There were a total of 179951 co-author edges. All papers were sorted using the published year, and each co-author edge was present in the data stream. The value on a co-author edge was the number of publications in that year. The task of this experiment is to read the publication information as a data stream and predict the values on each of the edges at any particular time in the future in real time.

IMDB data set: The problem formulation was similar to the *DBLP* data set, since a co-acting data set is very analogous to a co-author data set. The processing was a bit different in order to account for the large cast sizes associated with each movie. The *IMDB* dataset contained the information of movies and actors in four genres from 1996 to 2005. One problem with this data set was, that even a short film will have a long cast of actors, most of whom are not very relevant. This results in an extremely dense graph. For example, a single movie with a cast size of 200, will result in approximately 20000 edges. In order to reduce the density of the graph, we chose only the top 3 actors and top 3 actresses in the list. This is also a logical choice, since these actors are relatively better known than others and have a greater chance to co-act with other people. This results in 147562 edges.

Cora data set: The *Cora* data set is also a bibliographic data set like *DBLP*. As in the case of the *DBLP* data set, the papers were sorted in temporal order, and received in the form of a graph stream. There are a total of 24961 authors and 19396 papers in the *Cora* data set. The value on an edge was the number of publications in that year. The task on the data set was to predict the values on each of the edges at any particular time in future.

B. Performance Measures

We tested the approach for both effectiveness and efficiency. The effectiveness measured in terms of the sum of squared error of the prediction value. This is a natural measure to use for regression analysis problems. If e'_{ij} be the estimated value on an edge, and the true value is e_{ij}, then the *SSQ* error is defined as follows:

$$SSQ = \sum_{(i,j)} (e_{ij} - e'_{ij})^2 \qquad (4)$$

This is a standard measure which is used to compute the error in many regression analysis problems.

We also tested the efficiency of the approach, since the approach may sometimes be used in dynamic scenarios. For the case of efficiency, we tested the number of edges processed per second. In cases, where the processing rate was reported over the progression of the stream, the last reporting time window was used, whereas in cases where the processing rate was reported for a particular parameter value, it is was averaged over the life of the data stream.

C. Baselines

There are no known methods for performing the link regression in the context of a network. However, there are many methods which use regression analysis in order to make predictions. These methods will be used as baselines by our method. As baselines, we used a number of different methods, which use the correlations between the values of the different streams to make prediction:

1. We used a direct PCA-algorithm on the different edge streams in order to make prediction. Note that PCA is able to use the correlations in the values across the different edges in the stream.

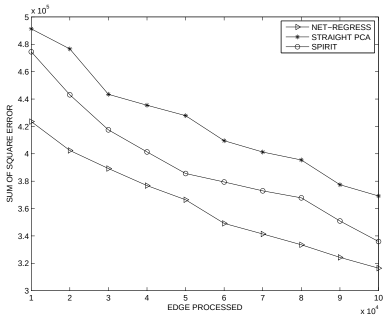
2. We used the SPIRIT [18] algorithm as the second baseline. While SPIRIT is based on the same principle, it has a number of features, which improve both its effectiveness and efficiency.

D. Effectiveness

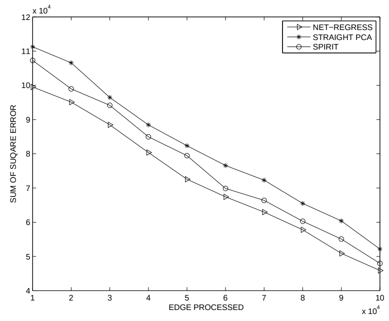
The main parameter used by the algorithm is the number of partitions k. Unless otherwise indicated, the default value of k was 12.

The effectiveness of the approach with stream progression for the *DBLP*, *IMDB* and *Cora* are illustrated in Table I (a) (b) (c) respectively. The stream progression in terms of the number of edges is illustrated on the X-axis, whereas the effectiveness is illustrated on the Y-axis in terms of the sum of squared error. It is evident that the prediction error of all three methods reduces with progression of the stream. This is because more training data is available as the stream progresses, and therefore the accuracy improves. In all cases, the *NetRegress* approach performs superior to the two baseline methods. Furthermore, the *SPIRIT* method performs superior to the PCA technique. This is because SPIRIT is better optimized than a simple application of PCA to temporal regression analysis.

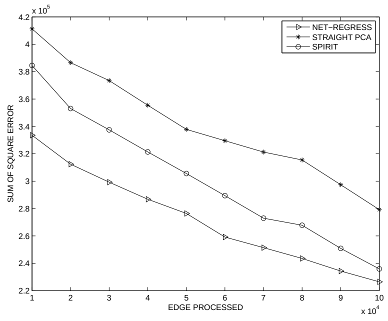
The effectiveness of the approach with increasing number of partitions for the *DBLP*, *IMDB* and *Cora* data streams are illustrated in Table I (d) (e) (f) respectively. Since the other schemes do not use the number of partitions as a parameter, the performance is shown as a horizontal line for those cases. For the case of the *NetRegress* approach, it can be seen that the error initially reduces with increasing number of partitions and then increases in both data sets. This is because when a small number of partitions is used, then the full advantage of structural locality is not leveraged. Each partition has too many nodes, and structural locality plays a much smaller role. On the other hand, if the number of partitions is too large, then there is are not enough number of nodes in each partitions in order to yield statistically robust results. Therefore, the best trade-off is usually achieved in the middle of the range. In both data sets, the *NetRegress* approach performs better than



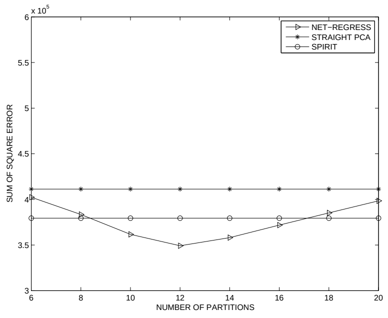
(a) Effectiveness with stream progression (DBLP)



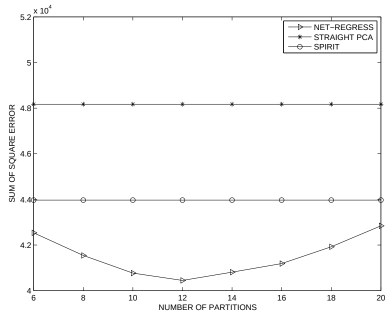
(b) Effectiveness with stream progression (IMDB)



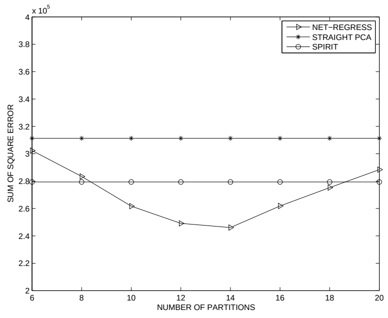
(c) Effectiveness with stream progression (Cora)



(d) Effectiveness with number of partitions (DBLP)



(e) Effectiveness with number of partitions (IMDB)



(f) Effectiveness with number of partitions (Cora)

TABLE I
EFFECTIVENESS RESULTS

the other baselines over most of the range, and in fact over the entire range in the case of the *IMDB* data set. The average performance of the two baselines are shown as horizontal lines, since they are not dependent on the number of partitions. The *SPIRIT* approach performed better than the *PCA* method, and this is consistent with our observations from the previous charts.

E. Efficiency Results

We also tested the efficiency of the different methods, over different choices of parameters. The efficiency with respect to stream progression for the *DBLP*, *IMDB* and *Cora* data sets are illustrated in Table II (a) (b) (c) respectively. The parameter k was set to its default value of 12. The stream progression is illustrated on the X-axis, and the average edges processed per second are illustrated on the Y-axis. The rate of processing of the *NetRegress* approach seemed to vary quite a bit over the progress of the stream. This was because of the drastic changes that occur in the partitioning over time, which results in varying number of nodes over the different partitions. Since the distribution of the nodes across the different partitions directly impacts the efficiency, this is reflected in the varying processing rate over time. Nevertheless, the *NetRegress* approach was at least competitive to both baseline algorithms, and is almost always more efficient than the PCA method. This is a very high level of efficiency, considering the fact that the superior effectiveness of the *NetRegress* approach is achieved by a more complex link-based mechanism in processing.

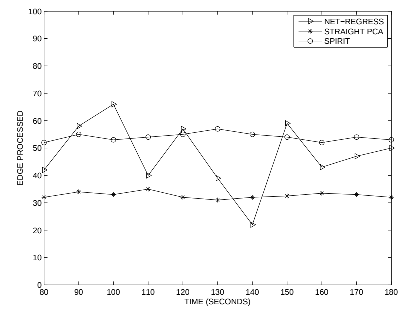
The efficiency of the approach with increasing number of partitions for the *DBLP*, *IMDB* and *Cora* data sets is illustrated in Table II (d) (e) (f) respectively. The other baselines are represented as horizontal lines in these figures, since they do not depend upon the number of partitions. It is evident that an increasing number of partitions slows down the *NetRegress* approach. This is because of the larger overhead and computations, when a larger number of partitions are used. In these cases, the algorithm seems to be linearly scalable with the number of partitions. When the *NetRegress* approach uses a smaller number of partitions, it is significantly more efficient than both baselines. It should be pointed out that the *NetRegress* approach can achieve greater effectiveness, when 5 or 6 partitions are used. Excellent efficiency is also achieved within this range. Therefore, the *NetRegress* approach is able to simultaneously achieve a high level of effectiveness and efficiency.

V. CONCLUSIONS

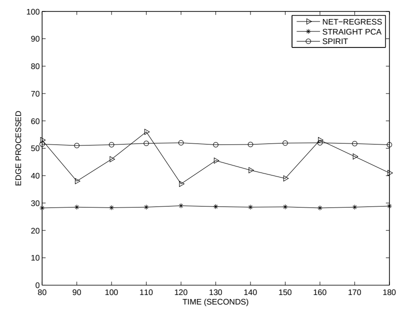
In this paper, we presented a new problem known as link regression, which can be considered a generalization of both the stream regression analysis and link prediction problem. This problem is very challenging because of its generality in the context of a networked streaming scenario. We presented a regression approach which was based on structural reservoir sampling, which implicitly sparsifies the network for further analysis. Our results show that our approach achieves a greater level of effectiveness than competitive baseline methods which can be adapted to this problem.

ACKNOWLEDGMENT

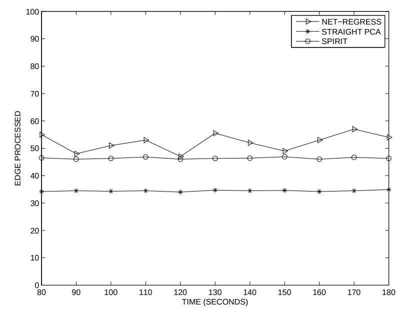
This work is supported in part by NSF under grants III-1526499, III-1763325, III-1909323, CNS-1930941, and CNS-1626432.



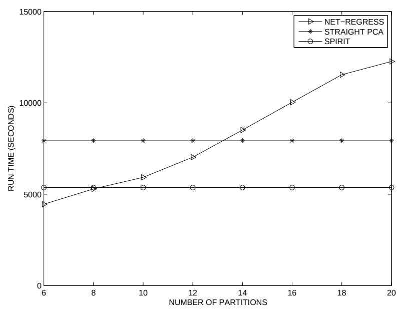
(a) Efficiency (Edges/sec.) with stream progression (DBLP)



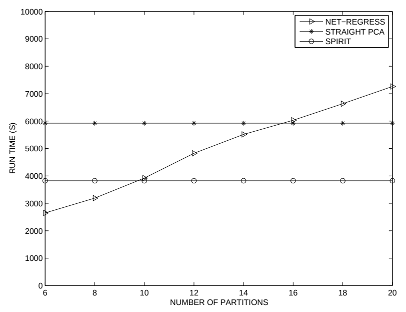
(b) Efficiency (Edges/sec.) with stream progression (IMDB)



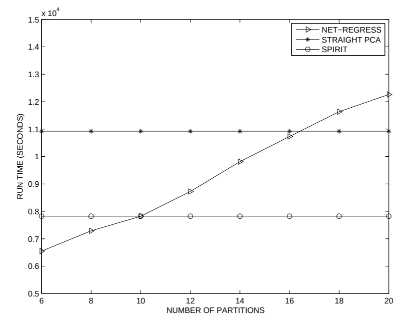
(c) Efficiency (Edges/sec.) with stream progression (Cora)



(d) Efficiency with number of partitions (DBLP)



(e) Efficiency with number of partitions (IMDB)



(f) Efficiency with number of partitions (Cora)

TABLE II
EFFICIENCY RESULTS

REFERENCES

[1] S. F. Adafre, M. Rijke, Discovering missing links in Wikipedia, *LinkKDD*, 2005.

[2] L. Adamic, E. Adar, Friends and neighbors on the web, *Social Networks*, 25, pp. 211–230, 2001.

[3] C. Aggarwal. *Social Network Data Analytics*, Springer, 2011.

[4] C. Aggarwal, Y. Zhao, and P. Yu. Outlier Detection in Graph Streams, *ICDE*, 2011.

[5] N. Ahmed, J. Neville, and R. Kompella. Network sampling: From static to streaming graphs. *ACM TKDD*, 8(2), 7, 2014.

[6] M. Al-Hassan, V. Chaoji, S. Salem, M. J. Zaki. Link prediction using supervised learning. *Workshop on Link Analysis, Counter-terrorism and Security (at SDM)*, 2005.

[7] M. Bilgic, G. Namata, L. Getoor. Combining collective classification and link prediction. *Workshop on Mining Graphs and Complex Structures (at ICDM)*, 2007.

[8] J. Doppa, J. Yu, P. Tadepalli, L. Getoor. Link mining: A survey. *SIGKDD Explorations*, 2005.

[9] J. R. Doppa, J. Yu, P. Tadepalli, L. Getoor. Chance constrained programs for link prediction. In *Analyzing Networks and Learning with Graphs*, 2009.

[10] L. Getoor, N. Friedman, D. Koller, B. Taskar. Learning probabilistic models of relational structure. In *ICML*, 2001.

[11] L. Getoor, N. Friedman, D. Koller, B. Taskar. Learning probabilistic models of link structure. *JMLR*, 3:679–707, 2002.

[12] G. Golub and C. van Van Loan. *Matrix Computations*, The Johns Hopkins University Press, 1996.

[13] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, M. Wang. A framework for semantic link discovery over relational data. In *CIKM*, 2009.

[14] D. R. Karger, Random sampling in cut, flow, and network design problems, In *STOC*, 1994.

[15] H. Kashima, N. Abe. A parameterized probabilistic model of network evolution for supervised link prediction. *ICDM*, 2006.

[16] D. Liben-Nowell, J. Kleinberg. The link prediction problem for social networks. In *LinkKDD*, 2004.

[17] M. E. J. Newman, Clustering and preferential attachment in growing networks, *Physical Review Letters*, 64, (2001).

[18] S. Papadimitriou, J. Sun, and C. Faloutsos. Dimensionality Reduction and Forecasting of Time-Series Data Streams, *Data Streams: Models and Algorithms, Springer*, 2007

[19] A. Popescul, L. Ungar, S. Lawrence, D. Pennock. Statistical relational learning for document mining. In *ICDM*, 2003.

[20] Y. Sun, R. Barber, Ma. Gupta, C. Aggarwal, J. Han. Co-author Relationship Prediction in Heterogeneous Bibliographic Networks. *ASONAM*, 2011.

[21] Y. Sun, J. Han, C. Aggarwal, N. Chawla. When will it happen – Relationship Prediction in Heterogeneous Information Networks, *WSDM*, 2012.

[22] B. Dong, S. Xie, J. Gao, W. Fan, P. Yu, OnlineCM: Real-time Consensus Classification with Missing Values. *SDM*, 2015.

[23] Z. Jiang and X. Chen and B. Dong and J. Zhang and J. Gong and H. Yan and Z. Zhang and J. Ma and P. S. Yu, Trajectory-based Community Detection (IEEE Transactions on Circuits and Systems II: Express Briefs) 2019

[24] B. Dong, C. Aggarwal, P. Yu, Transfer Learning for Network Classification. *IJCNN*, 2019.

[25] B. Dong, J. Zhang, C. Zhang, Y. Yang, P. Yu, Missing Entity Synergistic Completion across Multiple Isomeric Online Knowledge Libraries. *IJCNN*, 2019.

[26] B. Taskar, P. Abbeel, D. Koller, Discriminative probabilistic models for relational data, UAI, pp. 485–492, 2002.

[27] B. Taskar, M. F. Wong, P. Abbeel, D. Koller. Link prediction in relational data. In *NIPS*, 2003.

[28] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[29] Y. Sakurai, S. Papadimitriou, C. Faloutsos. BRAID: Stream Mining through Group Lag Correlations, *SIGMOD*, 2005.

[30] B.-K. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online Data Mining for Co-Evolving Time Sequences. *ICDE*, 2000.

[31] K. Yu, W. Chu, S. Yu, V. Tresp and Z. Xu, Stochastic relational models for discriminative link prediction, *NIPS*, 2006.

[32] P. Zhao, C. Aggarwal, and G. He. Link Prediction in Graph Streams, *ICDE Conference*, 2016.

[33] Y. Zhu, D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. *VLDB*, 2002.