Link Prediction in Graph Streams

ICDE Conference, 2016
Graph Streams

- Graph Streams arise in a wide variety of applications
  - Activity-centric social and information networks
  - Communication networks
  - Chat networks

- Graph streams are ubiquitous in settings in which activity is overlaid on network structure
Challenges

• Difficult to store the entire graph on disk because of high volume stream

• Graph applications such as link prediction require structural understanding of the graph

• Key is to design probabilistic summaries that can work in the link-prediction setting
The Link Prediction Problem

• Given a network $G = (N, A)$, determine the most likely pairs of nodes that receive a link between them.

• Typical classes of algorithms:
  – Neighborhood-based
  – Matrix factorization
  – Supervised
Contributions of this Work

- Link prediction algorithm for graph streams
- Design summarization schemes to use small-space data structures for prediction
- Is able to achieve results closer to their exact counterparts
- Design the real-time analogs to Jaccard, Adamic-Adar, and common neighbor.
Key Techniques Used

- We design MinHash based graph sketches to estimate Jaccard coefficient.

- Vertex-biased reservoir sampling based graph sketches to estimate common neighbor and Adamic-Adar.
  - Provide theoretical guarantees to exact estimation
  - Robust performance with respect to exact estimation
Streaming Setting

- We consider the streaming link prediction problem in a graph stream that receives a sequence \((e_0, e_1, \ldots, e_t, \ldots)\) of edges.
  - Each of which is in the form of \(e_i = (u, v)\) at the time point \(i\), where \(u, v \in V\) are incident vertices of the edge \(e_i\).

- We assume the underlying graph is \textit{undirected}, where the ordering of \(u\) and \(v\) of the edge is insignificant.

- The proposed graph sketches and corresponding estimation algorithms can be generalized to \textit{directed} graphs with minor revision.
Formalization

- At any given moment of time $t$, the edges seen thus far from the graph stream imply a conceptual graph $G(t) = (V(t), E(t))$, where $V(t)$ is the set of vertices, and $E(t)$ is the set of distinct edges up to time $t$.

- We use $\tau(u, t)$ to represent the set of adjacent vertices of the vertex $u$ in the graph $G(t)$.
  - In other words, $\tau(u, t)$ contains the distinct vertices adjacent to $u$ in the graph stream till $t$, and the degree of $u$ is denoted as $d(u, t) = |\tau(u, t)|$.  

- Given a graph stream $G(t)$ at time $t$, the streaming link prediction problem is to predict whether there is or will be an edge $e = (u, v)$ for any pair of vertices $u, v \in V(t)$ and $e \notin E(t)$. 
Target Measures

- In a graph stream $G(t)$ and for any $u, v \in V(t)$, the target measures for streaming link prediction are defined as follows,

1. **Preferential attachment**: $|\tau(u, t)| \times |\tau(v, t)|$;

2. **Common neighbor**: $|\tau(u, t) \cap \tau(v, t)|$;

3. **Jaccard coefficient**: $\frac{|\tau(u, t) \cap \tau(v, t)|}{|\tau(u, t) \cup \tau(v, t)|}$;

4. **Adamic-Adar**: $\sum_{w \in \tau(u, t) \cap \tau(v, t)} \frac{1}{\log(|\tau(w, t)|)}$
Observations

- These measures seem to be relatively trivial to compute in the static setting.

- However, in the streaming setting, we do not have a global view of the graph at any given time.

  - This makes simple computations surprisingly difficult.

- As an example, let us look at the simplest measure of preferential attachment.
Preferential Attachment

• Preferential attachment, it requires an accurate estimation of the number of distinct edges incident on each vertex, i.e., \(|\tau(u, t)|, u \in V(t)|.\)

  – This is not as easy as it sounds because the distinct edges cannot be explicitly maintained and updated for exact counting in a graph stream.

• Streaming methods for distinct element counting may be employed.

• Other neighborhood methods are much harder and will be the focus of the paper.
Jaccard Coefficient

- Natural approach is the min-hash index: has been used earlier in various applications for computation of Jaccard coefficient (Broder et al).

- Given a graph stream, we maintain a MinHash based graph sketch for each vertex $u$ with two key values: the minimum adjacent hash value, $H(u)$, and the minimum adjacent hash index, $I(u)$.

  \[
  H(u) = \min \mathcal{H}(v), (u, v) \in E(t) \tag{1}
  \]

  \[
  I(u) = \text{argmin} \mathcal{H}(v), (u, v) \in E(t) \tag{2}
  \]

- The Jaccard coefficient between a pair of nodes is equal to the probability that their min-hash indices are the same.
Basic Intuition

• Consider two columns in a binary matrix

• If you sort the rows randomly, what is the probability that both columns show values of 1, when at least one of them shows values of 1
  – Simulated by the Jaccard coefficient
  – Also captured by the min-hash index: Each hash function simulates a sort and multiple hash functions are used for robustness
  – Can use Chernoff bound to provide guarantee
Common Neighbor Estimation

- To estimate common neighbors of two vertices in a graph stream, a natural question arises: whether a sample of edges from the graph stream are good enough to estimate this target measure accurately?

  - Unfortunately, unbiased sampling provides a negative answer to this question mainly due to the power-law degree distribution of real-world graphs.
Example

- For example, consider a graph in which the top 1% of the high-degree vertices (degrees greater than 10) contain 99% of the edges.

- A down-sampling with a rate of 10% will capture the neighborhood information of such top 1% high-degree vertices robustly, but may not capture even a single edge for the remaining 99% low-degree vertices.

  - Result: poor predictions!
Solution: Vertex-biased Sampling

- To address this issue, we design the *vertex-biased* sampling based graph sketches, where for each vertex $u \in G(t)$, a *reservoir* $S(u)$ of budget $L$ is associated to dynamically sample $L$ incident edges of $u$.

- Note if an edge $(u,v)$ is sampled, the vertex $v$, not the edge itself, is maintained and updated in $u$’s reservoir, $S(u)$).

- This sampling approach is biased, because the number of sampled edges, $L$, is fixed for all vertices in the graph stream, and is therefore disproportionately higher for low-degree vertices than high-degree ones.
Issues

- Sampling $L$ incident edges of each vertex can be implemented by the traditional reservoir sampling method.

- Another problem arises for high-degree vertices instead: for two vertices with degrees larger than $L$, the number of common neighbors is hard to estimate accurately from their independent samples.

- For example, consider the case where the budget is $L = 10$, and the two vertices $u$ and $v$ have a degree 1,000 each with all such 1,000 incident vertices being common neighbors of $u$ and $v$.

- Then, if 10 adjacent neighbors of $u$ and $v$ are sampled independently, the expected number of common neighbors of
$u$ and $v$ is $10 \times 10/1000 = 0.1$, which definitely is not an accurate estimation.
Solutions

- To this end, we adopt this constant budget-based sampling approach with an important caveat: *random samples in different reservoirs of vertices are forced to be dependent on each other.*

- To model this dependency, we consider an implicit sorting order of vertices to impose a priority order on the vertex set of the graph stream.

- Such a priority order is enforced with the use of a hash function $G : u \in V \rightarrow (0, 1)$, where the argument $u$ is a vertex identifier, and the output is a real number in the range $(0, 1)$ with lower values indicating higher priority.
Specifically, vertices retained in the reservoir $S(u)$ are those having the highest $L$ priority orders among all incident neighbors of $u$. 
Sampling Rate

• In order to dynamically maintain and update $S(u)$ when the graph stream evolves, it is important to note that, if the degree of the vertex $u$ satisfies $d(u, t) > L$, only a fraction $\eta(u, t) = L/d(u, t)$ of all the incident vertices of $u$ can be retained in the sketch $S(u)$.

• To account for this, we define a threshold of the priority value $G(\cdot)$ for each incident neighbor of $u$ that can survive in the reservoir $S(u)$ as

$$\eta(u, t) = \min\{1, L/d(u, t)\} \quad (3)$$
Estimating number of common neighbors

• Let $\eta(u, t)$ and $\eta(v, t)$ be the fraction (threshold) of incident neighbors of $u$ and $v$, respectively, which are sampled. The number of common neighbors, $C_{uv}$, of $u$ and $v$ is

$$C_{uv} = \frac{|S(u) \cap S(v)|}{\max\{\eta(u, t), \eta(v, t)\}}$$  \hspace{1cm} (4)

• Paper discusses the trick to dynamically maintain $\eta(u, t)$ and $\eta(v, t)$. 
Adamic-Adar

- An important observation of Adamic-Adar is that the common neighbor $w$ of vertices $u$ and $v$ with a higher vertex-degree is weighted less, because of its proclivity to be a noisy vertex, as accounted for in the term $1/\log(|\tau(w,t)|)$.

- Such a weighting strategy makes it very challenging to estimate Adamic-Adar in graph streams.

- Very high-degree vertices contribute noice.
Truncated Adamic-Adar

- We consider an approximate variation of Adamic-Adar, which truncates the insignificant high-degree common neighbors of $u$ and $v$ for streaming link prediction, because they contribute little in the computation of Adamic-Adar.

- It turns out that such an approximation still preserves the accuracy of Adamic-Adar, but can be computed exactly based on the vertex-biased sampling based graph sketches.

- The truncated Adamic-Adar, denoted as $\overline{AA}(u, v)$, between two vertices $u$ and $v$, is defined in the same way as Adamic-Adar, except that the components contributed by common neighbors with degrees greater than $d_{max}$ are eliminated:

$$\overline{AA}(u, v) = \sum_{\{w \in \tau(u, t) \cap \tau(v, t) : |\tau(w, t)| \leq d_{max}\}} \frac{1}{\log(|\tau(w, t)|)}$$  (5)
Approach

• The core idea is that each of the components in the equation can be directly computed if both vertices \(u\) and \(v\) are present in the graph sketch \(S(w)\) of their common neighbor \(w\), whose degree is no larger than \(d_{max}\).

• At the beginning, we initialize the capacity of the reservoirs, \(L = d_{max}\), for all vertices.

• This is because vertices whose degrees are larger than \(d_{max}\) will not be considered in the computation of truncated Adamic-Adar
Truncated Adamic-Adar

• We then examine all the graph sketches one by one. For each vertex \( w \), both vertices \( u \) and \( v \) need be present in \( S(w) \) (Line 5).

• It is further checked whether \( w \) has a degree at most \( d_{max} \) by examining if the values of \( \eta(w) \) and \( \overline{\eta}(w) \) equal to 1, indicating that the graph sketch \( S(w) \) is not full yet.

• If so, \( (1/\log(|\tau(w,t)|)) \) is added to \( \overline{AA}(u,v) \).

• Note that the degree value, \( |\tau(w,t)| \), is equal to the current reservoir size, \( |S(w)| \).
Handling computational bottlenecks

- The main computational bottleneck is that the graph sketches of all vertices need to be scanned once for each incoming new edge \((u,v)\) in the graph stream.

- To alleviate this problem, we consider adopting inverted indexes to facilitate the computation of truncated Adamic-Adar.

- Specifically, for each vertex \(u\) in the graph stream, an auxiliary inverted index structure, \(L(u)\), is built to maintain vertex identifiers of \(v\) whose graph sketch \(S(v)\) contains \(u\), i.e., \(v \in L(u)\) if and only if \(u \in S(v)\).

- Inverted indices are dynamically maintained and they double the memory requirement.
Experimental Results

• We chose three real-world, publicly available graph datasets in our experimental studies.

• Note that their edges are attached with timestamps indicating when they are first created in the graphs, and thus can be ordered, formulated, and processed in a form of graph streams

  - DBLP, Amazon Product co-purchasing, Wikipedia citation network
Prediction Accuracy

• Compute accuracy with respect to a random predictor

• Ratio of the accuracy to that of a random predictor

• Values greater than 1 are good.
### DBLP Results

<table>
<thead>
<tr>
<th>Methods</th>
<th>Ext-Ja</th>
<th>App-Ja</th>
<th>Ext-CN</th>
<th>App-CN</th>
<th>Ext-Adar</th>
<th>App-Adar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>120.57</td>
<td>104.25</td>
<td>112.66</td>
<td>108.91</td>
<td>93.47</td>
<td>92.57</td>
</tr>
<tr>
<td>Methods</td>
<td>Katz</td>
<td>PrefAttach</td>
<td>PropFlow</td>
<td>R-PRank</td>
<td>Short-Path</td>
<td>SimRank</td>
</tr>
<tr>
<td>Accuracy</td>
<td>100.48</td>
<td>114.97</td>
<td>96.76</td>
<td>59.82</td>
<td>107.49</td>
<td>74.85</td>
</tr>
</tbody>
</table>
## Amazon Co-Purchasing Network

<table>
<thead>
<tr>
<th>Methods</th>
<th>Ext-Ja</th>
<th>App-Ja</th>
<th>Ext-CN</th>
<th>App-CN</th>
<th>Ext-Adar</th>
<th>App-Adar</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>116.15</td>
<td>106.74</td>
<td>121.07</td>
<td>109.70</td>
<td>116.57</td>
<td>116.15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Katz</th>
<th>PrefAttach</th>
<th>PropFlow</th>
<th>R-PRank</th>
<th>Short-Path</th>
<th>SimRank</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>88.38</td>
<td>116.23</td>
<td>98.75</td>
<td>71.48</td>
<td>147.70</td>
<td>110.95</td>
</tr>
</tbody>
</table>
Progression with Stream Size (DBLP)

(a) Jaccard

(b) Common neighbor

(c) Adamic-Adar
Performance with respect to algorithm parameters

(a) Jaccard
(b) Common Neighbor
(c) Adamic-Adar
Space with respect to algorithm parameters

(a) Jaccard

(b) Common Neighbor

(c) Adamic-Adar
## Runtime Cost vs Graph Stream Size

<table>
<thead>
<tr>
<th>Runtime Cost (in seconds)</th>
<th>Graph Stream Size $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.4M</td>
</tr>
<tr>
<td>App-Ja</td>
<td>0.004</td>
</tr>
<tr>
<td>Ext-Ja</td>
<td>21.68</td>
</tr>
<tr>
<td>App-CN</td>
<td>0.007</td>
</tr>
<tr>
<td>Ext-CN</td>
<td>18.0</td>
</tr>
<tr>
<td>App-Adar</td>
<td>0.29</td>
</tr>
<tr>
<td>Ext-Adar</td>
<td>16.44</td>
</tr>
</tbody>
</table>
Conclusions

• New method for link prediction in graph streams

• Generalizes common neighborhood methods for graph streams

• Future work will also generalize more advanced techniques for graph streams

• Design methods for incorporating content in link prediction