Generating Evolving Property Graphs with Attribute-Aware Preferential Attachment

Amir Aghasadeghi
Drexel University
Philadelphia, PA
aa3657@drexel.edu

Julia Stoyanovich*
Drexel University
Philadelphia, PA
stoyanovich@drexel.edu

ABSTRACT

In recent years there has been significant interest in evolutionary analysis of large-scale networks. Researchers study network evolution rate and mechanisms, the impact of specific events on evolution, and spatial and spatio-temporal patterns. To support data scientists who are studying network evolution, there is a need to develop scalable and generalizable systems. Tangible systems progress in turn depends on the availability of standardized datasets on which performance can be tested.

In this work, we make progress towards a data generator for evolving property graphs, which represent evolution of graph topology, and of vertex and edge attributes. We propose an attribute-based model of preferential attachment, and instantiate this model on a co-authorship network derived from DBLP, with attributes representing publication venues of the authors. We show that this attribute-based model predicts which edges are created more accurately than a structure-only model. Finally, we demonstrate that synthetic graphs are indeed useful for evaluating performance of evolving graph query primitives.

ACM Reference Format:
Amir Aghasadeghi and Julia Stoyanovich. 2018. Generating Evolving Property Graphs with Attribute-Aware Preferential Attachment. In *DBTest'18: Workshop on Testing Database Systems , June 15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3209950.3209954

1 INTRODUCTION

Graphs are used to represent a plethora of phenomena, including the Web, social networks, biological pathways, transportation networks, and semantic knowledge bases. Considerable research and engineering effort is being devoted to developing effective and efficient graph representations and analytics. The phenomena that are represented by graphs can change over time. Consequently, many interesting questions about graphs, and about the networks they represent, are related to their evolution rather than to their static state. Researchers study graph evolution rate and mechanisms, (e.g., [1, 8]), the impact of specific events on evolution (e.g., [7]), and spatial and spatio-temporal patterns (e.g., [15]). Some areas where evolving graphs are being studied are social network analysis [12, 16, 17, 26], biological networks [4, 5, 27] and the Web [7, 23].

To support data scientists who are studying graph evolution, there is a need to develop scalable and generalizable systems. Tangible systems progress in turn depends on the availability of standardized datasets on which performance can be tested. In this work, we make progress towards a data generator for *evolving property graphs* [22]. Such graphs represent evolution of (1) graph topology and (2) of vertex and edge attributes. Our ultimate goal is to develop a flexible and principled graph generation method that can be used to simulate both dimensions of graph evolution. With this method in hand, we can generate synthetic graphs with different properties, such as size, density, clustering coefficient, or degree distribution, and judiciously evaluate performance of graph analysis primitives with respect to these properties. In this paper, we focus on an important aspect of this challenging task, and propose a *preferential attachment model* that uses both graph structure and information derived from vertex attributes.

Our model is based on the observation that the likelihood that a pair of vertices will form a connection may depend on the values of their attributes. For example, it is conceivable that two individuals are more likely to become friends if they speak the same language or live in close geographic proximity of each other. It is also conceivable that two individuals are less likely to become friends if they are members of opposing political parties. In general, information derived from attributes may increase, decrease, or have no impact on the likelihood that a pair of vertices will form a connection. The impact of attributes on this likelihood can be learned by observing past evolution, and incorporated into the model.

Related work. We leverage a rich body of work on generative models for graphs [1, 2, 11, 14, 17, 19, 28, 29]. Such models were used to simulate network evolution in different domains, including computer networks and the Web [11, 29], citation graphs [19], and social networks [17]. Notably, this work focuses solely on network structure, and did not account for vertex and edge attributes. As we will show in this paper, complementing structural information with information based on attribute evolution, helps reconstruct network structure more accurately, and is therefore potentially more useful for testing performance of query workloads over evolving graphs.

Another line of work focuses on generating property graphs, with attributes on nodes and edges [10, 25]. For example, the LDBC Social Network Benchmark (SNB) [10] is a realistic synthetic social network with multiple entity types such as person, post and comment. An interesting aspect of SNB is that attributes of the entities are used when creating relationship edges between them (e.g., it is more likely for people from the same school to become friends). SNB does not use a known generative model but rather

*This work was supported in part by NSF Grant No. 1750179.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DBTest'18, June 15, 2018, Houston, TX, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5826-2/18/06…$15.00
https://doi.org/10.1145/3209950.3209954

generates the power law distribution of vertex degrees directly. Further, although SNB entities have a creation time as one their attributes, these are generated uniformly at random.

This treatment of temporal information is problematic because, as it was shown in [17, 19, 29], vertex and edge creation times are not uniformly distributed in real evolving graphs. In our work, it is our goal to leverage a known and principled generative model for the generation of evolving property graphs such as the SNB. Specifically, we focus on preferential attachment models, and study and extend random walk [29], arguably the most fundamental model in this class, with attribute information.

We note that another family of generative models has been proposed recently, based on using a matrix operation, the Kronecker product, to generate graphs termed "Kronecker graphs" [18]. These graphs were used in scope of TrillionG, a scalable graph generator [24]. While the scalability of this model is appealing, it is not immediately clear how to extend Kronecker graphs to incorporate information about attributes, and we leave an investigation of this model to future work.

Contributions. We start by presenting preliminaries on evolving property graphs and on preferential attachment models (Section 2). We then present several important contributions towards a data generator for evolving property graphs. We propose an attribute-aware model of preferential attachment and develop a scalable distributed implementation of this model (Section 3). We instantiate this model on a co-authorship network derived from DBLP, with attributes representing publication venues of the authors (Section 4). We then demonstrate that synthetic graphs are indeed useful for evaluating performance of graph query primitives (Section 5). We conclude with a discussion of future work (Section 6).

2 PRELIMINARIES

2.1 Evolving Property Graphs

The evolving graphs we generate in this work adhere to the TGraph model, a logical model of an evolving graph proposed in [22]. A TGraph represents evolution of graph topology and of vertex and edge properties. TGraph extends the (non-temporal) property graph model [3], assigning periods of validity to graph nodes, edges and their properties.

We assume a linearly ordered discrete time domain Ω^T.

Definition 2.1. TGraph T is a 6-tuple $(V, E, P, \rho, \xi^T, \lambda^T)$, where: V is a finite set of *vertices*, E is a finite set of *edges*, and P is a set finite set of properties; $\rho : E \rightarrow (V \times V)$ is a total function that maps a directed edge to its source and destination vertex; $\xi^T : (V \cup E) \times \Omega^T \rightarrow B$ is a total function that maps a vertex or an edge and a time point to a Boolean, indicating existence of the vertex or edge during the time point; and $\lambda^T : (V \cup E) \times P \times \Omega^T \rightarrow Val$ is a partial function that maps a vertex or an edge, a property, and a time point to a value of the property at that particular time point.

Definition 2.1 uses a point-based representation of time to eliminate any ambiguity. For compactness, we use an *interval-based representation*: time intervals over $\Omega^T \times \Omega^T$ represent the constituent time instants. Interval-based representations do not add expressive power to the model [9]. Following the SQL:2011 standard, we use closed-open intervals.

TGraph is manipulated by the operators of temporal graph algebra (TGA), also described in [22], in accordance with *point semantics*. This semantics has two properties. The first, termed *snapshot reducibility*, requires that the result of applying a temporal operator to a database is equivalent to applying its non-temporal variant to each database state [6]. The second, termed *extended snapshot reducibility*, requires that timestamps are made available to operators by propagating time as data. This property allows users to reference time explicitly in predicates, but not manipulate time directly [6]. To support point semantics, TGraph must be *temporally coalesced*: value-equivalent temporally-adjacent facts must be represented by a single fact, with the corresponding period of validity. We will showcase the usefulness of our data generator on several operators of TGA in Section 5.

A *snapshot* of a TGraph T, denoted $\tau_p(\mathsf{T})$, is a conventional (non-temporal) property graph that represents the state of T at a specific time instant p. Here, τ is the time-slice operator that is commonly used in temporal languages.

Evolving graph model restrictions. The graph generator of Section 4 will produce valid TGraphs, but under the restriction that graph topology is *growth-only*. That is, while Definition 2.1 allows a vertex (resp. an edge) to exist during multiple non-overlapping time periods, in this work we will restrict our attention to graphs in which a vertex or an edge persists continuously form the time when it was created. Further, while TGraph is a multi-graph (its vertices and edges have identity, and multiple edges may connect a pair of vertices), in this work we focus on graphs in which at most one edge connects a given pair of vertices. We leave supporting deletion events, and handling multi-graphs, to future work.

2.2 Counting and Closing Triangles

In graph theory, a *triangle* is a triple of vertices (x, y, z) such that every pair of vertices in the triple is connected by an edge.

An important primitive in preferential attachment models is closing a triangle. A triple (x, y, z) is a *possible triangle* if there exists a path $x \rightarrow y \rightarrow z$ but there does not exist an edge (x, z). We refer to the pair (x, z) as a *candidate pair*. If an edge (x, z) is created, at some later time point, we will refer to this edge as a *triangle-closing edge* at that time point.

In accordance with Definition 2.1, we record graph evolution at discrete time points. When generating T, we count triangles and possible triangles, or add triangle-closing edges, in a given snapshot $\tau_p(\mathsf{T})$ at time p. We use $p-1$ and p to denote two consecutive time points during which some change occurred in T.

Let us denote by CP_{p-1} the set of candidate triangle-closing pairs at time $p-1$, and by CE_p the set of triangle-closing edges that were added at time p. We will denote by $P(\triangle_p)$ the probability that a possible triangle is closed at time p, and estimate it as:

$$\hat{P}(\triangle_p) = \frac{|CE_p|}{|CP_{p-1}|} \tag{1}$$

$\hat{P}(\triangle_p)$ is an estimate both because it is computed from observations, and because the number of possible (resp. actual) triangles and the number of candidate pairs (resp. closing edges) is not the

same. For example, we may have two paths of length two connecting a given pair of vertices ($x \rightarrow z \rightarrow y$ and $x \rightarrow w \rightarrow y$): two possible triangles and a single candidate pair (x, y) that closes both.

Equation 1 estimates the over-all probability of closing a triangle. We are also interested in estimating the probability of closing a triangle with edge (x, y) at time p, conditioned on the pair-wise similarity between vertices x and y at time $p-1$. This similarity is defined with respect to the attributes of x and y, and is denoted by the random variable S_{p-1}. The conditional triangle closing probability is estimated as:

$$\hat{P}(\triangle_p \mid S_{p-1}) = \frac{|\{(x,y) \in CE_p \wedge sim(x,y) = S_{p-1}\}|}{|\{(x,y) \in CP_{p-1} \wedge sim(x,y) = S_{p-1}\}|} \quad (2)$$

3 ATTRIBUTE-BASED PREFERENTIAL ATTACHMENT

We now describe our proposed attribute-based preferential attachment model (ABA). This model builds on random walk (RW) [29] — a simple model that underlies most other generative models, and on recursive search (RS) [29], which extends RW.

RW and RS are generative models that build a graph by adding vertices, in some pre-specified order, connecting the newly-added vertices to the graph, and closing triangles. Both RW and RS add and connect vertices one at a time. RW then performs a random walk on the graph: every time it visits a previously unvisited vertex, it creates a new edge with probability q, *closing a triangle*. Algorithm 1 gives the pseudocode of RW. RS is similar to RW, but it follows multiple paths through the graph simultaneously during the walking step, closing triangles with probability q.

Our algorithm, which we call attribute-based attachment (ABA), differs from RW and RS in that *a batch* of vertices is added to the graph at each iteration. Another assumption of the RW model that we relax is that each vertex will create all of its edges in one single iteration (immediately after it is added). This assumption holds true in some cases (e.g., in citation graphs), but not in many others, including collaboration graphs and social networks. In contrast to RW and RW, ABA will attempt to close all possible triangles in the graph at each iteration.

Another important component of ABA, which makes it attribute-based, is that edge-creation probability q will not be uniform for the entire graph. Rather, it will depend on the similarity between vertices x and y for a candidate edge (x, y).

Recall our discussion of triangles, possible triangles, and estimating a triangle closing probability from Section 2.2. Figure 2 presents these probability estimates derived from the co-authoriship graph of DBLP, for 2006-2016. In this graph, vertices represent authors, and undirected edges represent co-authorship. For each author, we compute a *venues* attribute — a set of venues in which the author published up to and including a given year. We compute pair-wise similarities based on the values of the venues attribute (details will be given in Section 4.1). Probability estimates in Figure 2 are computed using Equation 1 for the blue line and Equation 2 for the other lines. We observe that the probability of closing a triangle with edge (x, y) at time p does depend on $sim_{p-1}(x, y)$. When $sim_{p-1}(x, y)$ is below 0.25, $\hat{P}(\triangle_p \mid S_{p-1} \in [0.0.25])$ is consistently lower than the attribute-independent estimate $\hat{P}(\triangle_p)$. On the other hand, $sim(x, y) \geq 0.25$ at time $p-1$ leads to a higher probability of closing (x, y) at time p.

Based on this result, we incorporate pair-wise similarities into the edge selection process of ABA: instead of using $q = \hat{P}(\triangle_p)$ for all edges, we use different probabilities based on $sim(x, y)$.

Graph generator restrictions. We do not study the vertex addition mechanism in this paper, and focus on the edge creation probabilities during the walking step. Rather than connecting a new vertex to a random vertex in the existing graph, we take as input the set of new vertices *together with their edges* in the snapshot in which these edges first appeared, and then perform the walking step on the created graph. We also do not model attribute evolution in this work. As another input we take the value of the evolving attribute based on which we compute similarity. We start our generative models from the original graph snapshot in the year 2005 and from that we generate the rest of the snapshots.

Algorithm 1 Random Walk

Require: Set of vertices V', edge creation probability q.

```
initialize G(V = V'.getVertex(), E = ∅)
while x = V'.getVertex() do
    V ← x
    y = V.selectRandomElement()
    E ← (x, y)
    {Generate a random number between 0 and 1.}
    while (q ≤ Û(0, 1)) do
        z = G.selectRandomNeighbor(y)
        {A neighbor of y is reachable from y by an outgoing edge.
        Close the triangle (x,y,z).}
        E ← (x, z)
        y = z
    end while
end while
```

4 DATA GENERATION

Experimental Setup. For all experiments, we used a 16-slave in-house Open Stack cloud. Each of our nodes has four cores and 30 GB of RAM. GNU/Linux Ubuntu 14.04, Spark v2.01. stand alone cluster, and Hadoop2.6were used. We ran our experiments using Portal, a Spark-based system for querying evolving property graphs [21, 22] and GraphX [13]. Reported running times are averages of three runs. Because of Spark lazy evaluation, we use a materialize operation at the end of our timed experiments.

4.1 Data Gathering

For this research, we needed an evolving graph of a reasonable size (not too small) with specific characteristics: the graph must have attributes at least on the vertices, and it must have temporal information for vertices, edges and attributes. We generated such a dataset from DBLP [20]. Using the DBLP XML dump file we created a growth-only collaboration TGraph (per Section 2.1), with authors as vertices and with undirected co-authorship edges. Frequency of change in this graph is 1 year, and so periods of validity of vertices, edges and attribute values are at least one year. Note that 1-year temporal resolution is not an intrinsic restriction either of

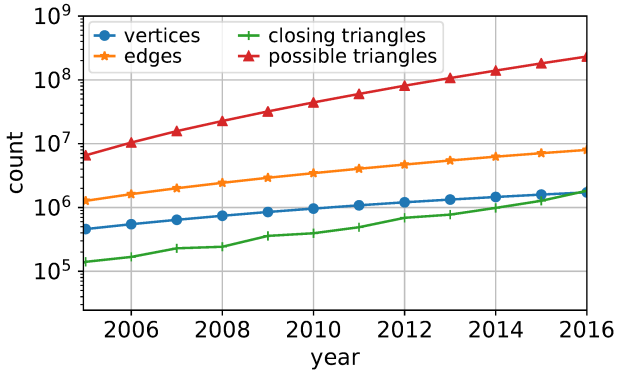

Figure 1: Number of vertices, edges, closing triangles, and possible closing triangles in DBLP, over time. Note logarithmic scale on the y-axis.

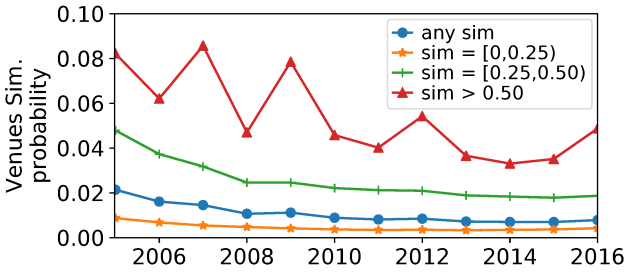

Figure 2: Triangle-closing probabilities in DBLP: *any sim* is per Equation 1, other series are per Equation 2.

the TGraph model or of the attribute-based attachment generative model, but rather is a property of the dataset with which we work.

An author vertex is valid from the year of the author's first publication and until 2016 (the end of the life-span of the TGraph). A collaboration edge between a pair of authors is valid from the year when the authors first published together and until 2016. Each author vertex has an evolving attribute called *venues* that represents the set of venues where the author has published papers, up to an including the given year.

Our dataset contains 12 yearly snapshots, from 2005 through 2016. We use the initial snapshot, 2005, for estimation only, and start predicting edges from 2006. Table 1 shows dataset statistics.

Table 1: DBLP Statistics

Time Span	2006- 2016
Number of Authors	1.3M
Number of Collaborations	10M
Number of Venues	1.6K
Number of Keywords	742K

As discussed in Section 3, our graph generation model and many other studied generation models (those based on preferential attachment) use triangle closing as their main method of edge creation. We introduced possible triangles and triangle-closing edges in Section 2.2. Figure 1 shows the number of vertices, number of edges, number of possible triangles and number of closing triangles for each snapshot of DBLP (note that the y-axis is in logarithmic scale). The number of possible triangles is an order of magnitude larger than the other statistics.

Closing triangles based on similarity. Our main assumption in this paper is that edge creation between more similar vertices is more probable. We studied this assumption by calculating Jaccard similarities on the *venues* attribute among the candidate pairs in each snapshot of the graph, and using these to estimate the conditional probability of closing a triangle. We grouped similarity values in three buckets $[0, 0.25)$, $[0.25, 0.5)$, $[0.5, 1]$. Figure 2 shows closing triangle probabilities for each of these buckets (per Equation 2). We also calculated unconditional triangle closing probability according to equation 1 ("any sim" in Figure 2). Based on this experiments, we find that higher similarity between a pair of vertices in a candidate pair yields a higher probability for a triangle to be closed.

4.2 Graph Generation

We now describe how we instantiate the graph generation process proposed in Section 3 on DBLP. In the generation process, we calculated average triangle closing probability for each similarity value range (bucket). Using those average probabilities, we perform the walking step of the Attribute-based Attachment (ABA) model. In addition to estimating probabilities from DBLP based on venues, we also computed reduced probabilities by dividing venues probabilities by four (we call this "Reduced Venues"), and studied the effect of this change on graph characteristics. We also generate a structure-only evolving graph (computing edge probabilities irrespective of vertex similarity), and use it in our comparison.

Figure 3 shows the number of edges for our different generated graphs. In this figure, "DBLP" shows the number of edges in the original dataset, and "Input" is the number of initial edges added to the graph when vertices are added. Since the number of vertices is the same in all graphs, we found the number of edges more informative than graph density. As expected, "Structure" and "Venues" models follow the same trend as the original DBLP, while "Reduced Venues" produces fewer edges. Note also that "Structure" approximates the total number of edges more closely than "Venues" until 2014. However, as we will show next, "Venues" is more accurate in deciding which edges to create.

Indeed, it turns out that the attribute-based model preserves the attribute similarity distribution better. To show this, we calculated the conditional triangle closing probability using venues similarity and structure similarity, and compare these to the real graph. Figure 4 shows the result — the mean square error (MSE) for those probabilities between the original graph and the generated model in each generated snapshot. We calculated MSE by comparing each similarity bucket probability of DBLP with the generated model, and averaging the error among all buckets for each snapshot. As can be seen, while the structure-only model works fairly well to reconstruct original probabilities, the attribute-based model preserves the original distribution better than the structure-only model — its error is consistently lower.

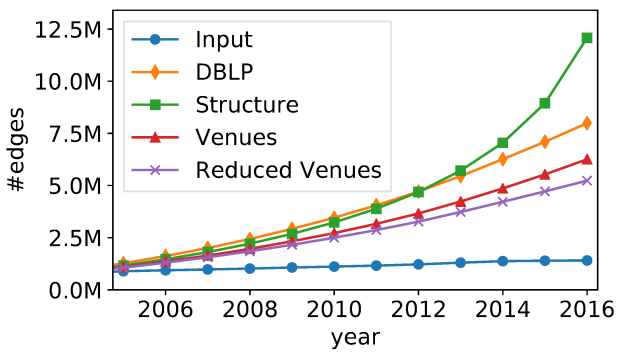

Figure 3: Number of edges for different generation models.

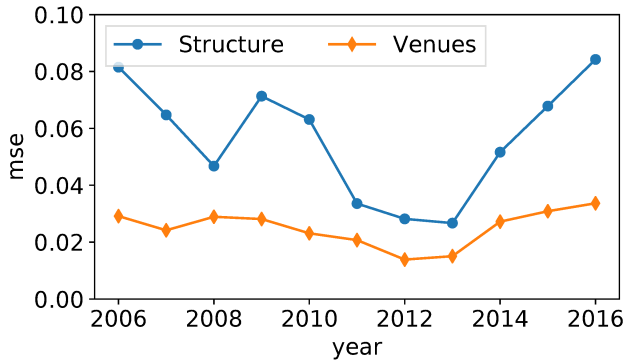

Figure 4: Mean Square Error (MSE) between probabilities of DBLP (ground truth), and graphs generated using structure only and venue-based similarity.

Finally, we find that the running time of our graph generator is reasonable: it takes 301 seconds in total to generate the graph using only the graph structure, 1028 seconds with venue-based similarity, and 928 seconds with reduced venue-based similarity. The bottleneck is computing pair-wise similarities between vertices in all candidate pairs, and so the structure-only variant is fastest because it does not perform this computation. As expected, venue-based similarity takes longer than its reduced variant, because fewer candidate pairs of vertices are considered.

5 EVALUATING QUERY PERFORMANCE

In this section we demonstrate that the graphs we generated in Section 4.2 are indeed useful for testing performance of evolving graph query primitives. The primitives we compute also allow us to investigate other interesting characteristics of the generated graph: average degree and clustering coefficient.

To compute average degree and clustering coefficient over time we used the Portal System [21]. We follow two goals in this experiment: first, to see how the generate models differ from the real graph, and second, to compare the execution time of each test on our different models.

Listing 1: Portal query for average vertex degree.

```
//load data
val g = GraphLoader.buildVE(data, -1, -1, range)
//compute degree per vertex
val degs = g.aggregateMessages[Int](sendMsg = (et =>
Iterator((et.dstId,1),(et.srcId,1))), (a,b) => a+b, 0,
    ↪  TripletFields.None)
//compute one vertex per rg with sum of degrees and
    ↪ count of vertices
val rgs = degs.vmap((vid, intv, attr) => (attr._2, 1),
    ↪  (0,0))
.createAttributeNodes( (a, b) => (a._1+b._1, a._2+b.
    ↪ _2))((vid,attr) => 1L)
val result = rgs.vmap((vid, intv, attr) => (attr._1 /
    ↪  attr._2.toDouble), 0.0)
result.vertices
```

Figure 5 shows the running time of the average degree query over time on each generated model, and the values of average degree over time for all models. As expected, the TGraph generated using venue similarity and the structure only TGraph yield result similar to DBLP, but the reduced probability TGraph has lower average degree. The execution times for different models are similar, which can be explained by the fact that all models use same set of vertices and have a relatively similar number of edges.

We also calculated the average clustering coefficient over time. Figure 6 shown the execution time and actual average values of the clustering coefficient over time for our graphs. As can be seen, the model that is less dense ("Reduced Venues") has a higher clustering coefficient. This is because graphs that are less dense have fewer candidate pairs (the denominators of Equations 1 and 2 are comparatively lower).

Listing 2: Portal query for clustering coefficient distribution.

```
//load data
val g = GraphLoader.buildHG(data, -1, -1, range)
//compute clustering coefficient per vertex
val coeff = g.clusteringCoefficient()
//compute one vertex per rg per clustering
    ↪ coefficient range
//i.e., 0-0.1, 0.1-0.2, etc.
val distro = coeff.vmap((vid, intv, attr) => (math.
    ↪ floor(attr._2*10)/10, 1),
(0.0,0)).createAttributeNodes( (a,b) => (a._1, a._2 +
    ↪  b._2))((vid,attr) => (attr._1*10).toLong)
distro.vertices
```

6 CONCLUSIONS AND FUTURE WORK

In this paper we presented a data generator for evolving property graphs, which represent evolution of graph topology, and of vertex and edge attributes. We proposed an attribute-based model of preferential attachment and instantiated this model on a co-authorship dataset derived from DBLP, with attributes representing publication venues. We demonstrated that synthetic graphs are indeed useful for evaluating performance of graph query primitives.

Much follow-up work remains. We will investigate automatic methods for attaching new nodes to the graph, and for modeling

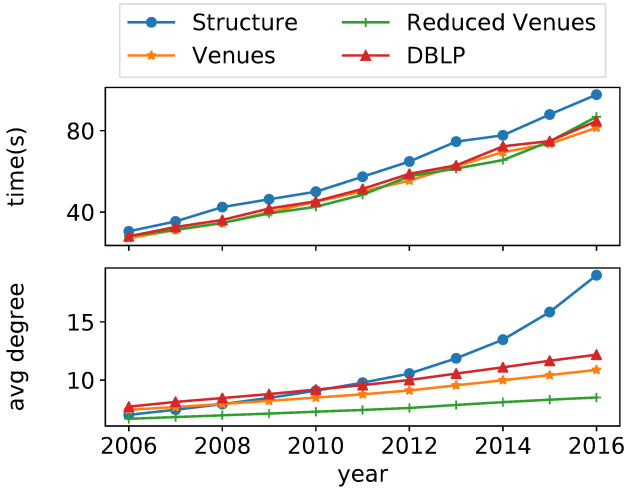

Figure 5: Execution time and average degree.

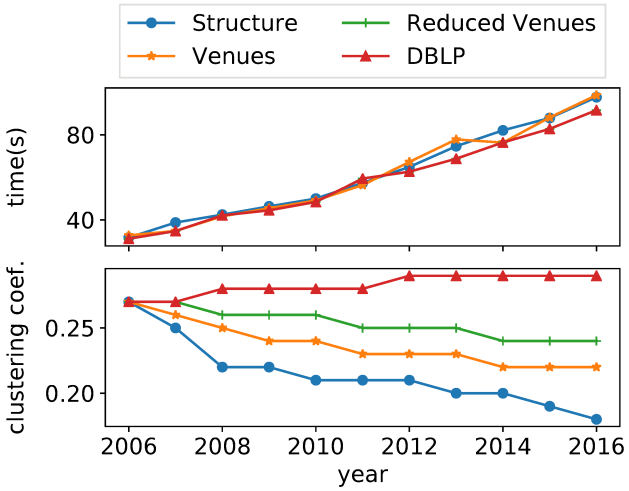

Figure 6: Execution time and clustering coefficient.

attribute value evolution. We will study how parameters of the generator can be adjusted in a principled way to influence characteristics of the resulting graphs, including size, degree distribution, density, and clustering coefficient. Finally, we will work on defining a workload of evolving graph query and analysis primitives.

REFERENCES

[1] Charu Aggarwal and Karthik Subbian. 2014. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 10.

[2] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of modern physics* 74, 1 (2002), 47.

[3] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017), 68:1–68:40. https://doi.org/10.1145/3104031

[4] Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. 2009. An event-based framework for characterizing the evolutionary behavior of interaction graphs. *TKDD* 3, 4 (2009). https://doi.org/10.1145/1631162.1631164

[5] Antje Beyer, Peter Thomason, Xinzhong Li, James Scott, and Jasmin Fisher. 2010. Mechanistic Insights into Metabolic Disturbance during Type-2 Diabetes and Obesity Using Qualitative Networks. *T. Comp. Sys. Biology* 12 (2010), 146–162. https://doi.org/10.1007/978-3-642-11712-1_4

[6] Michael H. Böhlen, Christian S. Jensen, and Richard T. Snodgrass. 2009. *Temporal Compatibility.* Springer US, Boston, MA, 2936–2939. https://doi.org/10.1007/978-0-387-39940-9_1059

[7] Jeffrey Chan, James Bailey, and Christopher Leckie. 2008. Discovering correlated spatio-temporal changes in evolving graphs. *Knowledge and Information Systems* 16, 1 (2008), 53–96. https://doi.org/10.1007/s10115-007-0117-z

[8] Junghoo Cho and H Garcia-Molina. 2000. The evolution of the web and implications for an incremental crawler. *VLDB '00 Proceedings of the 26th International Conference on Very Large Data Bases* (2000), 200–209. https://doi.org/10.1109/WI.2004.10097

[9] Jan Chomicki. 1994. Temporal Query Languages: A Survey. In *Temporal Logic, First International Conference, ICTL '94, Bonn, Germany, July 11-14, 1994, Proceedings.* 506–534. https://doi.org/10.1007/BFb0014006

[10] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. 2015. The LDBC social network benchmark: Interactive workload. In *ACM SIGMOD.* ACM, 619–630.

[11] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, Vol. 29. ACM, 251–262.

[12] Michaela Goetz, Jure Leskovec, Mary McGlohon, and Christos Faloutsos. 2009. Modeling Blog Dynamics. In *Proceedings of the Third International Conference on Weblogs and Social Media, ICWSM 2009, San Jose, California, USA, May 17-20, 2009.* http://aaai.org/ocs/index.php/ICWSM/09/paper/view/152

[13] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *OSDI.* 599–613. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez

[14] Pavel L Krapivsky and Sidney Redner. 2005. Network growth by copying. *Physical Review E* 71, 3 (2005), 036118.

[15] M. Lahiri and T.Y. Berger-Wolf. 2008. Mining Periodic Behavior in Dynamic Social Networks. In *2008 Eighth IEEE International Conference on Data Mining.* 373–382. https://doi.org/10.1109/ICDM.2008.104

[16] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. 2007. The dynamics of viral marketing. *TWEB* 1, 1 (2007). https://doi.org/10.1145/1232722.1232727

[17] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. 2008. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 462–470.

[18] Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker Graphs: An Approach to Modeling Networks. *Journal of Machine Learning Research* 11 (2010), 985–1042. https://doi.org/10.1145/1756006.1756039

[19] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph Evolution: Densification and Shrinking Diameters. *ACM Trans. Knowl. Discov. Data* 1, 1, Article 2 (March 2007). https://doi.org/10.1145/1217299.1217301

[20] Michael Ley. 2009. DBLP: some lessons learned. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1493–1500.

[21] Vera Zaychik Moffitt. 2017. *Framework for Querying and Analysis of Evolving Graphs.* Ph.D. Dissertation. Drexel University.

[22] Vera Zaychik Moffitt and Julia Stoyanovich. 2017. Temporal graph algebra. In *DBPL.* 10:1–10:12. https://doi.org/10.1145/3122831.3122838

[23] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. 2010. Web graph similarity for anomaly detection. *J. Internet Services and Applications* 1, 1 (2010), 19–30. https://doi.org/10.1007/s13174-010-0003-x

[24] Himchan Park and Min-Soo Kim. 2017. TrillionG: A trillion-scale synthetic graph generator using a recursive vector model. In *Proceedings of the 2017 ACM International Conference on Management of Data.* ACM, 913–928.

[25] Arnau Prat-Pérez, Joan Guisado-Gámez, Xavier Fernández Salas, Petr Koupy, Siegfried Depner, and Davide Basilio Bartolini. 2017. Towards a property graph generator for benchmarking. In *Proceedings of the Fifth International Workshop on Graph Data-management Experiences & Systems.* ACM, 6.

[26] Purnamrita Sarkar, Deepayan Chakrabarti, and Michael I. Jordan. 2012. Nonparametric Link Prediction in Dynamic Networks. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012.* http://icml.cc/discuss/2012/828.html

[27] Joshua M. Stuart, Eran Segal, Daphne Koller, and Stuart K. Kim. 2003. A gene-coexpression network for global discovery of conserved genetic modules. *Science* 5643, 302 (2003), 249−−255.

[28] Alexei Vázquez. 2001. Disordered networks generated by recursive searches. *EPL (Europhysics Letters)* 54, 4 (2001), 430.

[29] Alexei Vázquez. 2003. Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations. *Physical Review E* 67, 5 (2003), 056104.