Network Similarity Prediction in Time-evolving Graphs: A Machine Learning Approach

Keyvan Sasani, Mohammad Hossein Namaki, Assefaw H. Gebremedhin
Washington State University
{ksasani, mnamaki, assefaw}@eecs.wsu.edu

***Abstract*—Temporal graphs—graphs that change with time—are becoming increasingly relevant and important, especially in areas such as web graphs of documents, online social networks, and transportation traffic networks. In such areas, the exponential growth of the sizes of the graphs, and the high-velocity streams of changes in the graphs have called for effective *incremental* algorithms—algorithms that use *only* the changes in input to provide an updated solution—for a range of network analysis operations. Measuring similarity between a pair of graphs is an example of such a network analysis operation.**

In this paper, we consider the problem of tracking graph similarities when the graphs are continuously changing over time. We aim to leverage the temporal information in the graphs to avoid recalculating the similarities from scratch. We propose a new approach, namely incremental approximation of similarity measures in time-evolving graphs where correspondence between the nodes of the two graphs is known. In particular, we use machine learning to build a model to predict the graph similarity by considering only the changes in the graphs. Our experimental results over both real-world and synthetic graphs show the effectiveness and agility of the proposed machine-learning approach.

I. Introduction

Temporal networks analysis is becoming increasingly important in emerging applications and is being fueled by the growing availability of temporal graph data in a variety of areas, including social networks, communication networks, and transportation traffic networks of cities [1], [2]. Comparing graphs or computing their similarities is an important problem with many applications ranging from biological networks to web searching [3], [4].

Network similarity is a useful analytical tool for numerous applications. It can be used in almost every situation where detecting whether the structure of a network has significantly changed is important [5]. In transfer learning applications, it is used to transfer knowledge from one trained network to a related network [6]. One can use network similarity to detect anomalies or intrusion by considering whether a graph has changed more than a given threshold. The same basic idea is also used to find temporal anomalies in urban environments [7]. For example, an occurrence of an accident in a specific place may increase the density of the edges in the corresponding part of the graph. In other applications, to understand behavioral patterns in social networks, graph similarities can be used to gauge the amount of similarity or difference in structures. In yet another application, graph similarity can be used to classify brain connectomes to understand the correlation between structures of connectomes and peoples' level of creativity [5].

Graph similarity computations can be broadly classified into two groups; (1) graph similarity with known node correspondence and (2) graph similarity with unknown node correspondence. In this paper, we focus on the first group. In our context, given two graphs G_1 and G_2, the outcome of the computation is a similarity score $sim(G_1, G_2)$, typically normalized in the range $[0, 1]$. This score indicates how similar the two graphs G_1 and G_2 are, where 1 means that the two are identical.

For analysis of temporal graphs, an incremental computation is likely to have significantly better performance than computation from scratch, especially when the magnitude of the changes is relatively small. It is well-known that while real-world graphs are constantly evolving, the changes are rather small. For example, only 5% to 10% of nodes are updated weekly in a Web graph [8]. Therefore, intuitively, it is much less costly to incrementally maintain similarity measures between two graphs than recomputing the similarity score over entire graphs after each batch of updates.

In this work, given a stream of incoming graphs, we consider the questions *"Can we compute the similarity between consecutive snapshots incrementally without recomputing from scratch?"* and *"Can we compute the similarity measures by looking only at the changes between the snapshots?"*. To address these questions, we developed a machine learning framework to predict the similarity of two graphs by only making use of computationally efficient change-centric features and statistics from affected nodes, without imposing any assumptions on graph data or change stream. We experiment with synthetic graphs as well as real-world drawn from various applications with different characteristics. Our experiments show that the

proposed machine-learning approach can predict the similarity of any two consecutive graphs with an accuracy as high as 99% and average of 86%.

The rest of the paper is organized as follows. We review related work in Section II. In Section III, we formally state the problem. In Section IV, we discuss the computation method we use to address the problem and in the Section V, we discuss the proposed framework in detail. We describe the datasets and our implementations in Section VI and show our experimental results in Section VII. In Section VIII, we conclude and outline directions for future work.

II. Related work

The problem of detecting anomalies by measuring the amount and the significance of changes in consecutive web crawl snapshots created by search engines was investigated in [4]. The authors introduced five graph similarity functions for directed, time-evolving web graphs: Vertex/edge overlap, Vertex ranking, Vertex/edge vector similarity, Sequence similarity, and Signature similarity. Among these metrics, the last one performs the best in terms of change detection in web graphs, which is computed as the Hamming distance between the fingerprints of two graphs.

To get such fingerprints, a similarity hash function is applied to the graphs being compared. In other words, the method translates each graph into a set of features that are randomly projected to lower-dimensional feature space (signatures) where features are the nodes and edges of the input graphs, weighted appropriately by their PageRanks. The resulting vectors are then compared. In addition, they stated three requirements for similarity measures in the context of anomaly detection over web graphs namely, scalability, sensitivity, and coverage.

DeltaCon [5] is another recent work dealing with the network similarity problem. It mainly addressed two questions. 1) *How to compare two networks efficiently?* and 2) *How to evaluate their similarity score?* The authors formalized the principles and properties of the desired similarity measure by stating three axioms (identity, symmetry, zero property) and four properties (edge importance, edge submodularity, and weight awareness, focus awareness). They proposed a method that compares pairwise node affinities of consecutive snapshots of the graph sequence and reports an anomaly if the changes are greater than a threshold. The node affinities are computed by a fast variant of Belief Propagation (FaBP) [9]. They showed that DeltaCon satisfies all the axioms and properties introduced in the paper, and gives similarity scores that agree with common sense and can be easily explained. In this work, we selected our similarity measure as DeltaCon since it supports the aforementioned axioms and is sensitive to small but important changes in the graph.

Sim-Watchdog is another related work that investigated the problem of anomaly detection in temporal graphs [10]. The authors proposed a framework that can be summarized as follows. First, for each candidate similarity measure, it trains an individual classifier from the previously labeled snapshots of the graph. The labels indicate if graph faces an abnormal behavior in that snapshot. Then, given the classification results of each individual classifier on the training data, it selects a set of similarity measures with high discriminative power and form an ensemble of classifiers. Finally, using this ensemble of classifiers, it performs anomaly detection on the newly observed sequence of graph snapshots, and reports whether each snapshot is anomalous or not.

Another related work on graph similarity calculation is called NetSimile [11]. The author proposed a framework that (1) extracts some local and neighborhood-based features for all nodes in the input graphs, then (2) aggregates those features by calculating median, mean, standard deviation, skewness, and kurtosis of all those features to generate a feature vector as a signature of each graphs. At last, (3) similarities between those graphs can be calculated by computing the distances between the graph signatures using any distance metric (NetSimile used Canberra Distance). The authors showed that similarity calculation using feature extraction can lead to fast, scalable, and yet promising framework.

While measuring the similarity between two graphs has received a significant amount of research attention, to the best of our knowledge, no existing work has devised an incremental approach such that the algorithm does not need to visit the entire graphs for computing their similarities. In this paper, we predict the similarity of two consecutive graph snapshots by only visiting a fraction of the graph that is affected due to the changes. The approach, as a result, would require less amount of computation.

III. Problem Statement

Graph Stream. In this paper, we consider a time-evolving graph $\mathcal{G}_\mathcal{T}$ as a sequence of snapshots $\{G_0, ..., G_T\}$ where snapshot G_t arrives at timestamp t ($t \in [0, T]$) [12], [13]. A snapshot graph $G_t = (V_t, E_t)$ is a graph where V_t is a set of nodes, and $E_t \subseteq V_t \times V_t$ is a set of directed temporal edges at timestamp t.

A graph stream $\mathcal{G}_\mathcal{T}$ can be viewed as transaction streams. In particular:

(1) An edge transaction is either an edge insertion (denoted as e^+) or deletion (denoted as e^-).

(2) $\mathcal{G}_\mathcal{T}$ contains (a) snapshot G_0, and (b) at each timestamp t ($t \in [0, T]$), a batch of edge

transactions ΔE_t (i.e., batch update) that modifies snapshot G_t to G_{t+1}.

We will interchangeably refer to $\mathcal{G}_\mathcal{T}$ as a stream of snapshots, or corresponding edge transactions [14]. Assuming that the graphs are collected over a stable set of entities (e.g., email addresses, IP addresses, etc.), we have that $V_t = V$ for any t [10].

Incremental similarity maintenance. Given a graph stream $\mathcal{G}_\mathcal{T}$, our goal is to incrementally predict the similarity between two consecutive graph snapshots G_t and G_{t+1} by only monitoring 1) edge transactions ΔE 2) the fraction of data that is affected by the changes, and 3) the previously computed similarities, independent from the size of the graph stream. The similarity score $sim(G_t, G_{t+1})$ is a value in the range $[0, 1]$, where 0 means totally different graphs, and 1 means identical graphs.

IV. Similarity Computation

A. Overview

Many algorithms and similarity measures have been suggested to address the graph similarity problem. We can categorize the current algorithms in three main categories: graph edit distance based methods, iterative methods, and feature extraction based methods. The key idea behind current feature extraction approaches is that similar graphs probably share certain properties, such as degree distribution, diameter, and eigenvalues [15]. After extracting these features, a similarity measure is applied in order to assess the similarity between the aggregated statistics and equivalently, the similarity between the graphs.

As we discussed earlier, to design a predictive framework for the temporal graph similarity problem, we focus on the state-of-the-art similarity measure DeltaCon [5]. We predict the DeltaCon similarity values and we do so by only looking at the changes in the two graphs. Hence, in subsection IV-B, we review the underlying method which uses fast belief propagation [9]. In section V, we propose an efficient machine-learning-based framework to predict DeltaCon value as the similarity between snapshots. The intuition is that if we can define a graph as a set of features to compare with other graphs in terms of their similarity, we might be able to also define the dissimilarity of two graphs as a set of features. Hence, our final approach would be extracting features from edge transactions ΔE and touched nodes (e.g. the nodes that affected directly by changes) to compute their influence on the changes of graph similarity. Then, we propose a model to predict the similarity changes of two consecutive graph snapshots.

Procedure $DeltaCon_0$
Input: adjacency matrices A_1, A_2,
$n = |V|$, $numOfIterations$
Output: similarity score $simScore \in$ (0,1]
1. $S_1 := ApproxInverse(A_1, n, numOfIterations)$
2. $S_2 := ApproxInverse(A_2, n, numOfIterations)$
/* Compute DeltaCon similarity score */
3. $simScore := \frac{1}{(1+\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(\sqrt{S_{1,ij}}-\sqrt{S_{2,ij}})^2})}$
4. **return** $simScore$

Figure 1. Procedure $DeltaCon_0$ [5]

Procedure $ApproxInverse$
Input: adjacency matrix A, $n = |V|$,$numOfIterations$
Output: the results of Equation 1
1. $D := diag(d_1, d_2, ..., d_n)$ /* Calculate degree matrix */
2. $c_1 := \sum_{i=0}^{n}(D_{ii}) + 2$ /* sum of all degrees */
3. $c_2 := \sum_{i=0}^{n}(D_{ii}^2) - 1$ /* sum of all squared degrees */
/* Compute constant h_h about-half homophily factor to guarantee convergence */
4. $h_h := \sqrt{\frac{-c_1+\sqrt{c_1^2+4*c_2}}{8*c_2}}$
/* Compute constant a_h and c_h (epsilons) */
5. $a_h := \frac{4*h_h^2}{(1-4*h_h^2)}$
6. $c_h := \frac{2*h_h}{(1-4*h_h^2)}$
/* Compute invert factor M*/
7. $M := c_h * A - a_h * D$
/* Calculate the inverse */
8. $Inv := I$
9. $Mat := M$
10. $pow := 1$
11. **while** (biggest element $\in Mat$) $> (10^{-09})$ and
12. $(pow < numOfIterations)$ **do**
13. $Inv := Inv + Mat$
14. $Mat := Mat * M$
15. $pow := pow + 1$
16. **return** Inv

Figure 2. Procedure $ApproxInverse$

B. Baseline algorithm

DeltaCon compares the pairwise node affinities of consecutive snapshots of the graph sequence and reports an anomaly if the changes are greater than a threshold. DeltaCon computes the similarity between two graphs in three steps:
Step-1: The node affinities are computed by a fast variant of Belief Propagation (FaBP) [9] as shown in the following equation.

$$S = [s_{ij}] = [I + \epsilon^2 * D - \epsilon * A]^{-1} \quad (1)$$

Here, S is the matrix of node affinities, $\epsilon \in (0, 1)$ is a decaying factor, and I, D, and A are identity, degree and adjacency matrices, respectively.

A matrix inversion approximation technique is employed to calculate an affinity matrix between all pairs of nodes for each graph. This affinity-measure approximately corresponds to the random-walk distance between all node pairs. However,

in practice, DeltaCon does not compute the exact inversion of the matrix, instead they approximate it. Since the details of the inverse procedure is important for understanding the features we have used, we provide a detailed outline of the inverse procedure in Fig. 2. First, the algorithm initializes ϵ as a_h and ϵ' as c_h (Lines 5-6), calculates a matrix M out of $\epsilon * A - \epsilon' * D$ (Line 7) and initializes Inv with the identity matrix (Line 8). Then, iteratively, it accumulates $Inv + M * M$ for a fixed number of iterations. In the original experiments in the DeltaCon paper, the number of iterations is set to 10, which means that affinities are computed at most based on 10 hops away from each node. (See [9] for details)

Step-2: The matrices of pairwise node similarities are then compared using the Matusita Distance (MD) (which is related to the Euclidean Distance).

Step-3: Since the calculated value MD is a distance and it is unbounded, the value is finally transformed to similarity, using $\frac{1}{MD+1}$ to be bounded in [0,1].

The entire DeltaCon algorithm is presented in Fig. 1.

V. Our framework

In this section, we present our network similarity prediction framework which uses DeltaCon as its core similarity measure.

A. Overview

The framework is illustrated in Fig. 3. Following statistical learning, the framework derives a prediction model based on training sets (i.e., offline learning), and then predicts similarity measures of two "unseen" consecutive snapshots based on the derived model (i.e., online prediction). Our goal is then to adapt the framework to discover the similarities effectively and fast by extracting meaningful features.

We approach the graph similarity problem as a regression problem. First, we introduce the metrics, then we formulate the problem.

B. Metrics

A metric in our context must measure how well the similarity of future snapshots are likely to be predicted. We seek to minimize absolute value of the error to be able to get closer to the value computed by $DeltaCon_0$. Therefore, we use mean absolute error (*MAE* or *MeanAE*), a widely used evaluation metric in regression problems [16], [17] to evaluate our model. Given n pair of consecutive snapshots, *MAE* is computed as follows:

$$\frac{1}{n}\sum_{i=1}^{n}|\hat{y} - y|$$

where y and $\hat{y}$ are the actual values and predicted values of graph similarities, respectively. The lower the *MAE* value is, the more accurate the model is.

To empirically verify the robustness of our model, we also consider four other statistical metrics besides *MAE*: *median absolute error* ($MedianAE$), *maximum absolute error* ($MaximumAE$), *mean square error* (MSE) and *R-Squared* (R^2). MSE computes the mean squared of the differences between predicted values and actual values. R^2 is computed as follows:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where y, $\hat{y}$, and $\bar{y}$ are the actual values, predicted values, and the mean of actual values, respectively. The larger the R^2 value is (up to 1), the more accurate the model is.

The problem is to learn, using regression, a function $f(x) = \hat{y}$ that maps a feature vector x (that encodes the features extracted from given two G_1 and $G_2 = G_1 + \Delta E$ graphs) to a continuous value $\hat{y}$ corresponding to the exact value (y) of similarities between two G_1 and G_2 graphs. The goal is to minimize $|\hat{y} - y|$. That is, the closer $\hat{y}$ is to y, the closer MAE is to 0.

C. Learning phase

The learning phase procedure consists of the following.

(1) The framework uses the set of previous snapshots DeltaCon similarities $[1...L]$ to train the model. To this end, it observes the previous snapshots and computes their DeltaCon similarity by invoking the original algorithm shown in Fig 1 and collects the similarity metrics and features for each pair of snapshots to construct the training instances.

(2) The predictive model is then derived by solving a regression problem formulated as follows.

- **Input**: A graph G_1, and the edge updates ΔE such that $G_1 + \Delta E = G_2$.
- **Output**: a prediction model $\mathcal{M}$ that predicts the graph similarity $Sim(G_1, G_2)$ by learning a model that minimizes MAE on training data samples.

D. Features

To derive the learning process, we extract features by considering the nature of the problem, previous practical learning on graph studies[18], and DeltaCon algorithm. The features, their description, and intuition are provided in Table I. Furthermore, we have reported the importance of each feature by the average percentage of their contribution in the prediction model for our experiments. The detailed

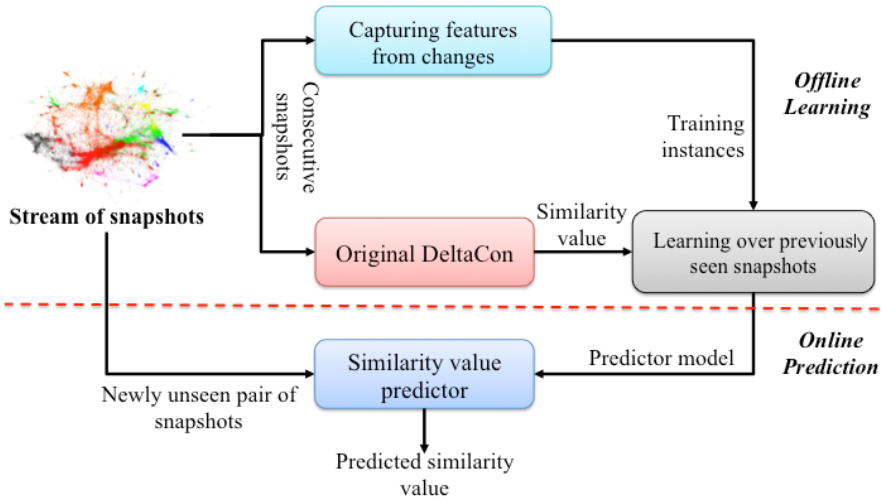


Figure 3. Learning framework for graph similarity prediction

analysis of feature importances can be found in section VII.

E. Online prediction

The prediction model is applied to predict the similarity of newly arrived snapshot with the current one. Upon receiving a new snapshot, the framework (1) computes the features based on the changes, and (2) predicts the similarity value between two graphs. The predicted results can then be readily applied for anomaly detection or other applications.

F. Predictive model

We have used random forest learning algorithm as the core of our predictive model since significant improvements in predictors' accuracy can be obtained by growing an ensemble of trees by asking them to vote for the predicted value [19]. Ensemble methods allow us to combine diverse weak regression trees by taking different samples of the original data set and then combining their outputs [20]. A random forest is a meta-estimator that benefits by combining some regression trees on various subsamples of the dataset and use averaging to improve the predictive accuracy and control overfitting [21]. We have empirically studied another competitive ensemble method called XGBoost [22], as our inner predictive model. XGBoost in contrast, trains each subsequent model using residuals of current prediction and true values. We found that random forest slightly outperforms XGBoost for our predictive task in most cases (see Figure 4 for details).

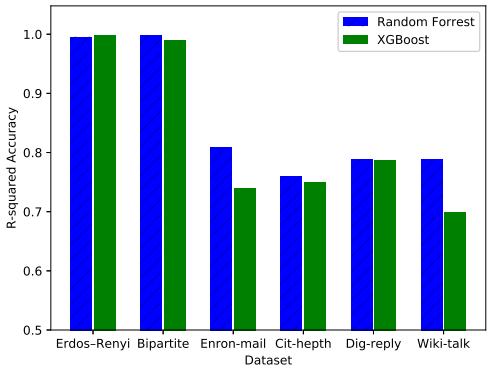


Figure 4. R-squared accuracy, RF and XGBoost Algorithms

VI. Implementation

In this section, we discuss the implemented algorithms and the datasets that we have used in our experiments.

A. Algorithms

We have implemented the following, all in Python.
(1) $DeltaCon_0$, the baseline algorithm that computes the similarity of two arbitrary graphs.
(2) Our machine-learning framework that, given two graphs, extracts the features from their changes and consults a predictive model for prediction. We have used random forest implementation of scikit-learn library [21].

B. Datasets

Enron-email. A large set of email messages, the Enron corpus, was made public during the legal investigation concerning the Enron corporation. The raw corpus is currently available on the web[1] [23].

[1] http://www-2.cs.cmu.edu/~enron/

Table I
FEATURES DESCRIPTION, INTUITION AND IMPORTANCE

Feature Name	Description/Intuition
# of touched nodes	Number of nodes that gained new edges or lost some. The more number of nodes has been changed, the more dissimilar two graphs are.
# of edge changes	The hamming distance of two graphs. The number of edges should be removed or added to reach from one graph to another. The more edit distance, the more dissimilar two graphs are.
# of added edges	How many more edges the second graph have in comparison with the first graph? The more number of edge additions, the less similar two graphs are.
# of removed edges	How many edges the second graph does not have in comparison with the first graph? The more number of edge removal, the less similar two graphs are.
Clus-coef before/after	The average clustering coefficients of touched nodes in the first/second graph. The lower the density of the graphs are, the higher similarity would be changed. Furthermore, the difference between clus-coef. before and after the change is an indicator of similarity.
Betweenness before/after	The average betweenness of touched nodes in the first/second graph (10 hops). If the edge is a bridge between two components, it has higher contribution in the similarity. Furthermore, the difference of betweenness in two graphs is an indicator of similarity.
Diff. avg. deg.	The difference of average degree of touched nodes between first and second graph. The more change in the degree of the nodes, the more dissimilar two graphs are.
ah parameter	one of the parameters of FaBP introduced in guilt-by-association.

Table II
DATASET STATISTICS

	|Nodes|	|Edges|	Avg Deg	Max Deg	|Triangles|	Avg Local Clus Coef
Enron-mail	87K	1M	26	39K	822M	0.18
Cit-hepth	23K	3M	233	12K	934M	0.81
Digg-reply	30k	86K	5	298	15K	0.01
Wiki-talk	1.1M	7.8M	16.7	488k	169K	0.20
Erdos–Renyi	500	83K	335.1	373	3.6M	0.55
Bipartite	500	37.9K	173.6	233	-	-

The Enron email network consists of $1,148,072$ emails sent among employees of Enron between 1999 and 2003. Nodes in the network are individual employees and edges are individual emails. It is possible to send an email to oneself, and thus this network contains loops. The edges are labeled by the timestamps of the emails.

Cit-hepTh. Arxiv HEP-TH (high energy physics theory) citation graph is from arXiv[2] library and covers all the citations. An edge from node u to node v indicates that paper u cited paper v and is labeled by the publication date of the paper u as its timestamp. If a paper cites or is cited by, a paper outside the dataset, the graph does not contain any information about this. The data is of the papers in the period from January 1993 to April 2003 [2].

Digg-reply. This is the reply network of the social news website Digg. Each node in the network is a user of the website, and each directed edge denotes that a user replied to another user [24]. The edges are labeled by the timestamps of replies.

Wiki-Talk. This dataset contains the user interaction networks of Wikipedia of 28 different languages. Nodes (orininal wikipedia user IDs) represent users of the Wikipedia, and an edge from user A to user B denotes that user A wrote a message on the talk page of user B at a certain timestamp [25], [26].

Synthetic graphs. In addition to these four real-world datasets, we experimented with synthetic graphs that we generated ourselves. For this purpose, we implemented a synthetic graph stream generator. The generator is controlled by several parameters including type of graph (e.g. bipartite, Erdos-Renyi), time period T, size of snapshots, size and type of batch updates. It simulates standard graph evolution model [27]. We used it to generate two 500 nodes, Erdos-Renyi and Bipartite graphs.

The statistics of the used datasets are provided in Table II.

[2]http://www.arxiv.org

VII. Experiments

In order to verify the effectiveness and speed of our proposed method empirically, we conducted a series of experiments over the real-world and synthetic datasets we described above. In this section, first, we discuss the performance of our method in predicting the similarity of two consecutive graphs. Second, we experimentally show that the learner model requires a small amount of training data to reach a desired accuracy. Third, we demonstrate the impact of changing the learner parameters on its performance. Finally, we analyze the importance of the features in the proposed predictive model.

Offline processing time. Similar to any other machine-learning approach, the prediction method has two steps. 1) offline training and 2) online prediction. To learn a prediction model, although we can learn from consecutive pair of snapshots to aggregate our training instances, we simply feed $\binom{n}{2}$ pairwise snapshots where n is the number of previous seen snapshots. Given 100 snapshots of Enron dataset, only 78 milliseconds are required to train the predictor. This does not include the feature extraction procedure. This procedure takes additional time which is related to amount of changes. However, it is less than a half second on average for each pair of snapshots.

Accuracy. Table III shows the effectiveness of the learned model on predicting the graph similarities using several metrics. One can verify that the accuracy of temporal synthetic graphs are higher than the real-world graphs due to the fact that synthetic graphs have more predictable behavior. In Bipartite and Erdos-Renyi graphs, the MAE is 0.2% in comparison with 4% in real-world datasets, which means we have 1% error on predicting the similarity score on average. In addition, Maximum Absolute Error is 11.6% for Erdos-Renyi, 4% for Bipartite and 9.8% on average for real-world datasets. It means that we can capture the difference of two consecutive graphs of real world networks with a boundary of less than nearly 15% percent.

Actual versus predicted similarity values are visualized in Fig. 5 to get a better insight on the framework's effectiveness. The blue line shows the actual similarity and the red line demonstrates the predicted values over nearly 100 pair of unseen snapshots in all the aforementioned graph streams. As a reader can see, the red line almost captures the vibrations of similarity score in a way that it is even hard to distinguish the differences.

We also test the variation of R^2 prediction accuracy based on the number of training instances and visualize it as a learning curve in Fig. 6. It is shown that the proposed method can reach an acceptable accuracy using 50 examples in synthetic and 60 samples in real-world graphs.

Tuning RF parameters. Random Forest model has various set of parameters to tune. These parameters affect both the performance and complexity of the model. Hence, it is important to investigate different settings to find the optimal one. Two most important parameters of the random forests are number of trees (estimators) and maximum depth of each tree. In fact, as the number of trees or the maximum depth of the trees increases, the complexity of the model grows since it tries to fit the training data better and also it may cause the overfitting problem. In favor of getting most out of our model, we have tried different settings for the model and figured out that 3 estimators with maximum tree depth of 3 are enough to have both simplicity and acceptable accuracy for the model. The details of our experiments are shown in Fig. 7.

Feature Analysis. In order to analyze the features we have used to build our predictor model in terms of their importance, we scored the contribution of each feature using this general rule: The relative rank (i.e. depth) of a feature used as a decision node in a tree can be used to assess the relative importance of that feature with respect to the predictability of the target variable. Features used at the top of the tree contribute to the final prediction decision of a larger fraction of the input samples. The expected fraction of the samples they contribute to can be used as an estimation of the relative importance of the features [21].

In Fig. 8, the top 10 features used by model are listed with their relative importance score in such a way that the summation of their score is equal to 1. As it is shown, number of touched nodes, and number of edges changes in each snapshot have a significant contribution to the similarity score which is easy to compute and consistent with the intuition. In Fig. 8, each bar shows the feature importance on average of different trees on the trained forests of the aforementioned datasets and black lines show their standard deviations range.

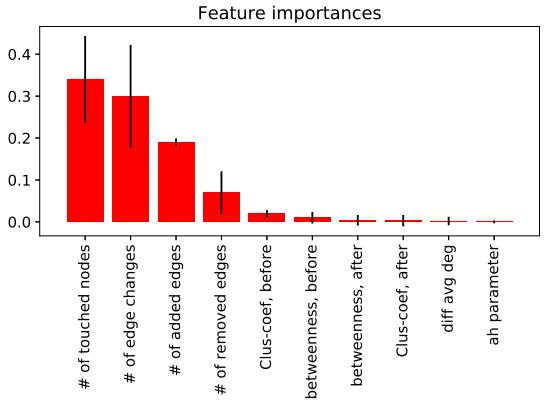
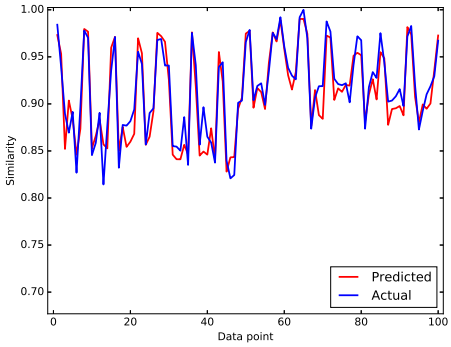


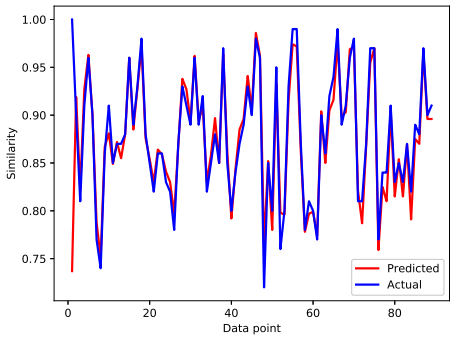
Figure 8. feature importance (%), vertical bars show their standard deviation among different trees

Table III
EFFECTIVENESS OF PREDICTING THE GRAPH SIMILARITIES OVER REAL-WORLD AND SYNTHETIC GRAPHS

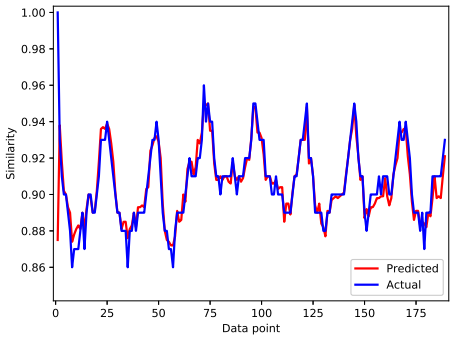
	R-Squared	MSE	MeanAE	MedianAE	MaximumAE
Enron-mail	0.809	3.67E-04	1.4E-03	1.1E-02	6.8E-02
Cit-hepth	0.760	1.01E-03	7.8E-03	7.8E-03	2.0E-02
Dig-reply	0.789	4.51E-03	4.0E-03	4.1E-03	1.3E-01
Wiki-talk	0.789	1.13E-02	1.0E-03	2.4E-03	1.8E-01
Erdos–Renyi	0.994	7.36E-05	2.1E-03	6.69E-04	1.1E-01
Bipartite	0.998	2.64E-05	2.0E-03	7.3E-04	4.1E-02



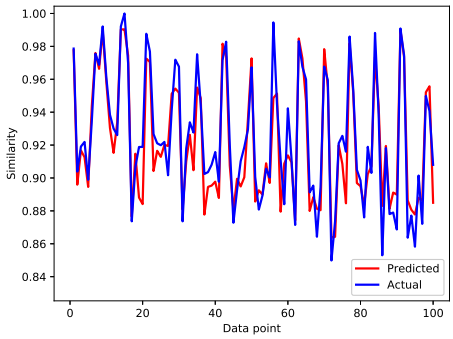
(a) Enron mail graph



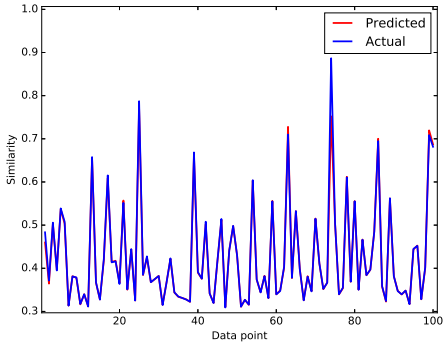
(b) Cit-hepth graph



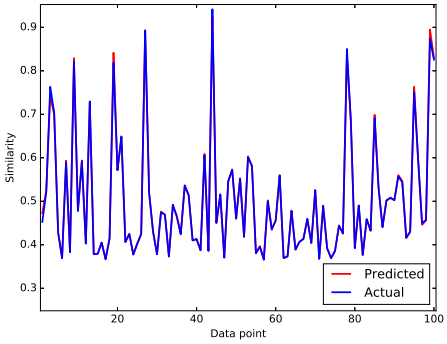
(c) digg-reply graph



(d) wiki-talk

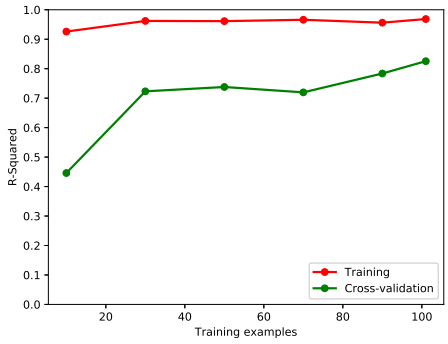


(e) Erdos–Renyi synthetic graph

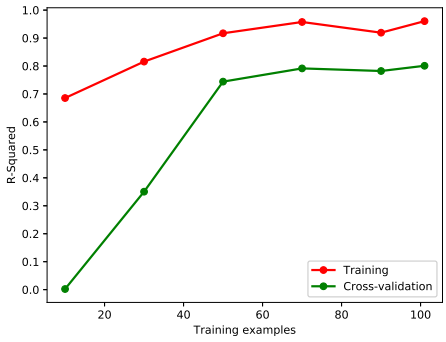


(f) Bipartite synthetic graph

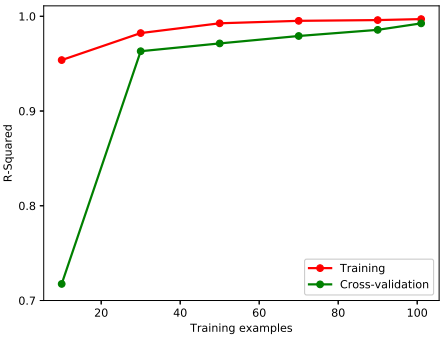
Figure 5. Similarity Scores, DeltaConactual values and proposed predicted values



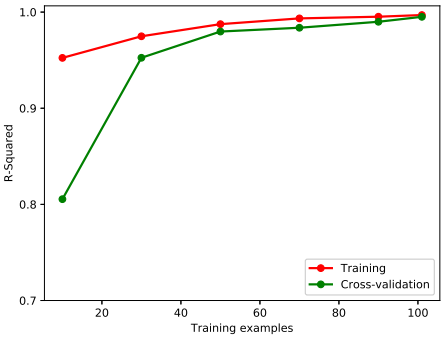
(a) Enron mail graph



(b) Cit-hepth graph

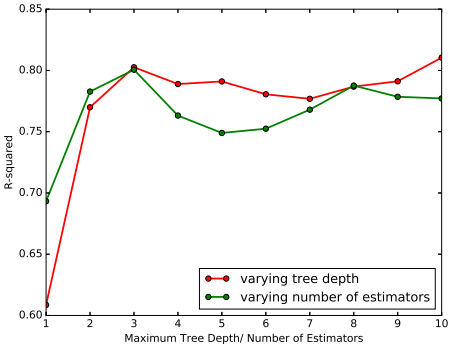


(c) Erdos–Renyi synthetic graph

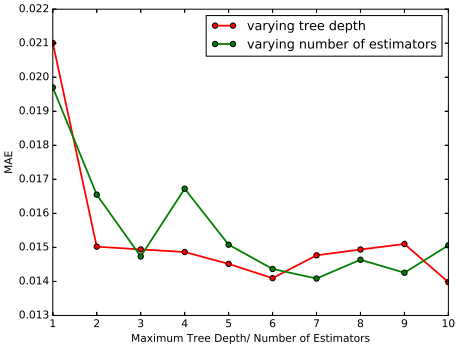


(d) Bipartite synthetic graph

Figure 6. Learning curve: the R^2 accuracy versus the number of training instances



(a) Erdos–Renyi synthetic graph



(b) Bipartite synthetic graph

Figure 7. Variation of R^2 accuracy by varying the number of estimators and maximum depth of the estimator

VIII. Conclusion and Future Work

We introduced the problem of similarity prediction for two consecutive snapshots in time-evolving graphs. We implemented a state-of-the-art algorithm known as DeltaCon and proposed an approach to solve the incremental similarity computation problem by approximating it. We presented a learning framework to predict similarity of two graphs by extracting features based on the difference between the graphs. We showed that by exploiting computationally efficient features from changes in the graphs, the similarity values can be accurately predicted using regression models. Our experimental study over multiple real-world and synthetic graphs verifies the effectiveness of the learned predictors.

This is an ongoing project and we are testing our learning framework over other real-world graph datasets. In addition, we aim to investigate transfer learning such that we learn similarity prediction in a graph and generalize it to other data graphs by addressing the question “can we learn over one or more graphs and use our prediction models for

other graphs?". Furthermore, since our solution is an approximate solution to predict the similarity, we aim to investigate if one can find a confidence measure to go back to the original DeltaCon algorithm when one is not confident about the prediction.

Furthermore, we will investigate if we can enrich our feature set by finding other computationally feasible new features to improve the accuracy of our prediction or even an unsupervised approach to come with representative features. An interesting direction to extend this work would be using representation learning algorithms[28], [29] to assign a low-dimensional feature vector to each node of the graph and the δE changes. In these approaches, the goal is to embed structural informations of the nodes in the graph to a low-dimensional space. By using this technique, the obtained feature vector can be used to maintain graph similarity over time with conventional vector comparison techniques. At last, since the proposed algorithm is a machine learning approach, the training set is playing a very important part. Another good further direction for this project would be how to find good training examples to capture all graph changes to get the most out of our dataset.

IX. Acknowledgments

This work is supported in part by NSF award IIS-1553528.

References

[1] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Computing Surveys (CSUR)*, 2014.

[2] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *KDD*, 2005, pp. 177–187.

[3] H. Bunke, P. J. Dickinson, M. Kraetzl, and W. D. Wallis, *A graph-theoretic approach to enterprise network dynamics*. Springer Science & Business Media, 2007, vol. 24.

[4] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *Journal of Internet Services and Applications*, pp. 19–30, 2010.

[5] D. Koutra, N. Shah, J. T. Vogelstein, B. Gallagher, and C. Faloutsos, "Deltacon: Principled massive-graph similarity function with attribution," *TKDD*, pp. 28:1–28:43, 2016.

[6] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li, "Rolx: structural role extraction & mining in large graphs," in *KDD*. ACM, 2012, pp. 1231–1239.

[7] Y. Zheng, H. Zhang, and Y. Yu, "Detecting collective anomalies from multiple spatio-temporal datasets across different domains," in *SIGSPATIAL*, 2015, p. 2.

[8] W. Fan, X. Wang, and Y. Wu, "Incremental graph pattern matching," *TODS*, p. 18, 2013.

[9] D. Koutra, T.-Y. Ke, U. Kang, D. H. P. Chau, H.-K. K. Pao, and C. Faloutsos, "Unifying guilt-by-association approaches: Theorems and fast algorithms," in *ECML-PKDD*, 2011, pp. 245–260.

[10] G. Yan and S. Eidenbenz, "Sim-watchdog: Leveraging temporal similarity for anomaly detection in dynamic graphs," in *ICDCS*. IEEE, 2014, pp. 154–165.

[11] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Netsimile: A scalable approach to size-independent network similarity," *arXiv preprint arXiv:1209.2684*, 2012.

[12] M. H. Namaki, K. Sasani, Y. Wu, and T. Ge, "Beams: bounded event detection in graph streams," in *ICDE*, 2017, pp. 1387–1388.

[13] M. H. Namaki, Y. Wu, Q. Song, P. Lin, and T. Ge, "Discovering graph temporal association rules," in *CIKM*, 2017.

[14] M. H. Namaki, P. Lin, and Y. Wu, "Event pattern discovery by keywords in graph streams," in *IEEE Big Data*, 2017.

[15] D. J. Watts, *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 1999.

[16] Q. Guo, R. W. White, S. T. Dumais, J. Wang, and B. Anderson, "Predicting query performance using query, result, and user interaction features," in *Adaptivity, Personalization and Fusion of Heterogeneous Information*, 2010.

[17] K. Sasani, M. H. Namaki, Y. Wu, and A. Gebremedhin, "Multi-metric graph query performance prediction," in *DASFAA*, 2018.

[18] M. H. Namaki, K. Sasani, Y. Wu, and A. H. Gebremedhin, "Performance prediction for graph queries," in *Proceedings of the 2nd International Workshop on Network Data Analytics*. ACM, 2017, p. 4.

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[20] G. Seni and J. F. Elder, "Ensemble methods in data mining: improving accuracy through combining predictions," *SLDMKD*, pp. 1–126, 2010.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *JMLR*, pp. 2825–2830, 2011.

[22] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*, 2016, pp. 785–794.

[23] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *European Conference on Machine Learning*. Springer, 2004, pp. 217–226.

[24] M. D. Choudhury, H. Sundaram, A. John, and D. D. Seligmann, "Social synchrony: Predicting mimicry of user actions in online social media," in *Proc. Int. Conf. on Computational Science and Engineering*, 2009, pp. 151–158.

[25] J. Sun, J. Kunegis, and S. Staab, "Predicting user roles in social networks using transfer learning with feature transformation," in *Proc. ICDM Workshop on Data Mining in Networks*, 2016.

[26] "Wikipedia talk, english network dataset – KONECT," Apr. 2017. [Online]. Available: http://konect.uni-koblenz.de/networks/wiki_talk_en

[27] B. Naveh *et al.*, "Jgrapht," *Internet: http://jgrapht. sourceforge. net*, 2008.

[28] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *arXiv preprint arXiv:1711.08752*, 2017.

[29] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.