Béres *et al. Applied Network Science* (2019) 4:64
https://doi.org/10.1007/s41109-019-0169-5

Applied Network Science

RESEARCH Open Access

Node embeddings in dynamic graphs

Ferenc Béres[1,2*], Domokos M. Kelen[1,2,3], Róbert Pálovics[4] and András A. Benczúr[1,3]

*Correspondence: beres@sztaki.hu
[1]Institute for Computer Science and Control, Hungarian Academy of Sciences, (MTA SZTAKI) Kende Street 13-17, H1111 Budapest, Hungary
[2]Eötvös University Budapest Pázmány s. 1, H-1117 Budapest, Hungary
Full list of author information is available at the end of the article

Abstract

In this paper, we present algorithms that learn and update temporal node embeddings on the fly for tracking and measuring node similarity over time in graph streams. Recently, several representation learning methods have been proposed that are capable of embedding nodes in a vector space in a way that captures the network structure. Most of the known techniques extract embeddings from static graph snapshots. By contrast, modeling the dynamics of the nodes in temporal networks requires evolving node representations. In order to update node representations that reflect the temporal changes in the local graph structure, we rely on ideas for data stream algorithms. For example, we assess neighborhood overlap by a MinHash fingerprint-based algorithm.

To evaluate our methods, in addition to the standard link prediction task, we provide dynamic ground truth data for the quantitative evaluation of similarity search by using online updated node embeddings. In our experiments, we constructed tennis tournament Twitter mention graphs as edge streams and compiled dynamic ground truth by using tournament schedule as external source. Our new algorithms outperformed snapshot-based batch methods for both link prediction and similarity search.

Introduction

The need for machine learning over data streams is motivated by a rapidly growing number of industrial applications of graph algorithms (Wang et al. 2017; Nie et al. 2017; Zhou et al. 2017; Wei et al. 2017) and online machine learning (Bifet et al. 2010; De Francisci Morales et al. 2016; Zhu and Shasha 2002; Žliobaite et al. 2012). In graph streams, the combination of the two areas, edges arrive continuously over time from a large network and have no duration (McGregor 2014). We intend to apply online machine learning (Bifet et al. 2010) for link prediction and similarity search by learning and updating node feature representations on the fly from graph streams.

The principal task of online machine learning is to learn a concept incrementally by processing data immediately after creation (Widmer and Kubat 1996), for example, after each mention in a Twitter mention graph. Traditional, batch learners build static models from finite, static data sets, which do not change over time. By contrast, stream learners build models that evolve over time. For example, in graphs, more recent edges can form a more relevant picture of the current network structure than older ones. The final model will strongly depend on the order of examples generated from a continuous, non-stationary flow of data. Modeling is therefore affected by potential concept drifts or changes in distribution (Gama et al. 2013). Online learning seems more restricted than batch learning, which can iterate over the data set several times, and thus one could expect inferior

SpringerOpen

results from online methods. By contrast, in some cases (Frigó et al. 2017), online methods perform surprisingly strongly.

To track node properties in a graph stream, we adapt the highly successful technique of node representations. Representation learning methods on graphs encode the nodes of the network to points in a low-dimensional vector space. In general, representations in the embedded space should reflect the structure of the original graph. The research area of node embeddings has been recently catalyzed by the Word2Vec algorithm (Mikolov et al. 2013), developed for natural language processing. Several node embedding methods have been proposed recently (Perozzi et al. 2014; Tang et al. 2015; Grover and Leskovec 2016; Qiu et al. 2018) and applied successfully for multi-label classification and link prediction in a variety of real-world networks from diverse domains.

In order to generate node embeddings, we have to solve the challenge of maintaining node embeddings for tracking and measuring node properties and similarities as the edges arrive. Most graph algorithms are difficult to update online. For example, to compute random walk-based embeddings (Grover and Leskovec 2016), we have to be able to maintain not just the embedding but also the set of walks whenever a new edge appears in the stream.

Time-aware relevance evaluation also becomes troublesome in the presence of fast changes in network structure. For link prediction (Liben-Nowell and Kleinberg 2007), in an edge stream we can update our model immediately after the new edge arrives, and predict a completely new list in the next step. For this so-called predictive sequential (abbreviated as prequential) evaluation (Dawid 1984), we have to define new evaluation metrics. The same difficulties for evaluating streaming recommenders was first observed in (Lathia et al. 2009).

To fully utilize the power of graph embedding, our main focus for evaluation is similarity search, in which we assess the information encoded in the embedding about node pairs rather than just a global property required for predicting links. For similarity search, the prime source of difficulty lies in ground truth compilation. Static relevance measures such as precision, recall, or NDCG already require ground truth labeling, which itself often requires tedious human effort such as the effort that has been made for TREC topics (Clarke et al. 2004). In a dynamic graph stream, depending on time granularity, the same human data curation may be required in each time step.

Algorithms for temporal graphs have already started to emerge in publications; however, very few graph learning algorithms are capable of immediately updating their models from edge streams. Similarly, in the literature we rarely find real graph streaming methods where node labels are highly dynamic: even link prediction tasks are evaluated in batches for sets of edges that appear over a longer period in time. Any embedding method can be applied in dynamic graphs by considering graph snapshots in time. However, such solutions do not only react slowly, but also build new representations for every snapshot, hence they require an entire model retraining for downstream machine learning tasks (Hamilton et al. 2017a). The more natural part of the task is updating the embedding: gradient descent is a commonly used optimization procedure, which naturally lends itself to online learning algorithms (Juang and Lin 1998) as well. However, walk-based embedding methods published so far could only efficiently rebuild the walks for snapshots or larger batches of insertions and deletions, and were not able to update the set of walks for every single new edge in the stream.

Present work. StreamWalk, our first algorithm updates the node embedding online to track and measure node properties and similarity from a graph stream (Rozenshtein and Gionis 2016). StreamWalk is based on the recent concept of temporal walks containing edges ordered in time. As illustrated in Fig. 1, the StreamWalk algorithm picks node samples for a single node from its temporal neighborhood, using temporal walks ending in the node. Given the sample, we optimize to make the embedding of the sample similar to the source node. Our algorithm performs online machine learning (Bifet et al. 2010) by continuously updating a model as we read the graph stream. Its key ingredients are online gradient descent optimization (Juang and Lin 1998) and time respecting temporal walks (Rozenshtein and Gionis 2016).

StreamWalk improves over static node representations as applied to graph streams in three key ways:

- It accounts for the ordering of edges by sampling only from time respecting random walks, capturing richer information about graph structure.
- It includes an efficient data structure to sample random walks online without storing the entire edge set.
- The node representations evolve over time to reflect changes in network structure.

Our second algorithm directly learns the neighborhood similarity of node pairs in the graph stream, which we call second order similarity. By MinHash fingerprinting (Fogaras and Rácz 2005), we efficiently approximate the neighborhood Jaccard similarity of any two nodes at a given time. Then we optimize the embedding to make pairs similar, proportional to the overlap of their neighborhood.

Our main results are twofold:

- We design two algorithms that can update their node representations quickly in large graph streams, and outperform the baselines among others in link prediction tasks.
- We design a quantitative experiment for accessing the quality of temporal node embeddings based on the Twitter tennis tournament mention graphs of (Béres et al. 2018), which include temporally changing node labels. In a supervised experiment,

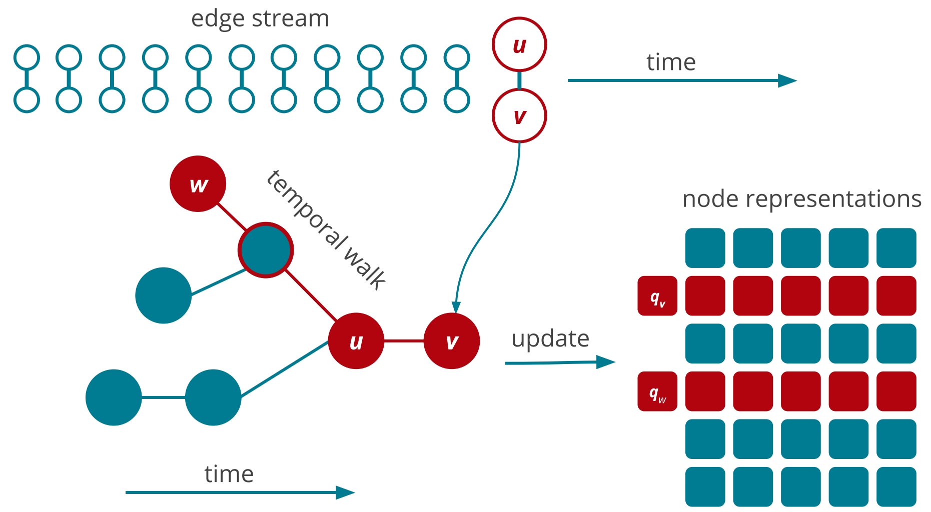


Fig. 1 Concept of StreamWalk. When *uv* edge arrives in the stream, vertices from the temporal neighborhood of *u* are sampled via temporal random walks. The method optimizes for the similarity of *v* and the sampled node *w*

we show that online updateable embeddings capture node similarities better than static embeddings.

The rest of this paper is organized as follows. First we summarize the related works in "Related works" section. In "Dynamic vector space embedding methods in edge streams" section we introduce StreamWalk and our method for learning neighborhood similarities directly from graph streams. In "Similarity search experiments" section we first examine the quality of dynamic node embeddings on the RG17 and UO17 Twitter data sets. Finally, in "Online link prediction" section we consider the online link prediction problem as another evaluation of our methods.

Related works

Temporal networks. A large variety of temporal network algorithms have appeared for connectivity, spanning trees, matchings, and many more, which are surveyed, for example, in (Holme and Saramäki 2012; Aggarwal and Subbian 2014). The usual approach for analyzing temporal graphs is to use timestamps to create a series of static graph snapshots (Kumar et al. 2010). High temporal granularity networks are considered in the edge or graph stream model (McGregor 2014) where edges must be processed once they arrive in the stream; for example, a random walk algorithm is described in (Sarma et al. 2011).

The concept of time respecting paths, in which adjacent edges must be ordered in time, is key in our results and directly used in one of our embedding models. The concept was perhaps introduced in (Moody 2002) for analyzing diffusion in networks. In another terminology, temporal walks were used to construct time-aware centrality metrics in (Rozenshtein and Gionis 2016; Béres et al. 2018).

Online machine learning. The area of online machine learning covers algorithms that work from data streams with only a limited possibility to store past data (Bifet et al. 2010). We define our models over graph streams where the data stream consists of the edges of the graph. Our models are online updateable, hence they are capable of adapting to concept drift.

Link prediction from graph streams is one of our tasks where the goal is to predict the next edge appearing in the edge stream. The problem is closely related to recommender systems where the strength of online machine learning has been observed recently (Ling et al. 2012; Frigó et al. 2017). Note that for link prediction we use the prequential evaluation, which was published more than thirty years ago (Dawid 1984) but has only recently come into widespread use (Gama et al. 2013) for streaming algorithms.

Representation learning on graphs. Embedding methods on graphs encode the nodes of the network to vectors in a low-dimensional vector space. In general, representations in the embedded space should reflect the structure of the original graph. Perhaps the most well-known method is Laplacian eigenmaps (Belkin and Niyogi 2002). Another class of models is based on the adjacency matrix of the graph; one popular example is graph factorization (Ahmed et al. 2013). Recently, random walk-based approaches have been proposed, like Node2Vec (Grover and Leskovec 2016), LINE (Tang et al. 2015), and DeepWalk (Perozzi et al. 2014). These methods sample node pairs that co-occur in random walks, and then optimize for their similarity in the embedded space. Walk sampling is motivated by the skip-gram model from natural language processing (Mikolov et al. 2013). Furthermore, the aforementioned techniques can be unified under a matrix factorization framework (Qiu et al. 2018).

We briefly review the methodology of the above approaches by following (Hamilton et al. 2017b). Static embedding methods learn an embedding vector q_u for each node u in the graph. Usually the objective is to learn vectors that are similar for neighboring nodes. Let $s(u)$ denote the neighborhood of u; then our goal is to satisfy $q_v \approx q_u$ for $v \in s(u)$. Shallow embedding approaches for static graphs differ in the objective function they use to ensure the similarity of the embeddings, and in the definition of the network neighborhood $s(u)$.

Graph factorization (Ahmed et al. 2013), GraRep (Cao et al. 2015), and HOPE (Ou et al. 2016) optimize for the squared error (SE) over node pairs in the neighborhood:

$$\sum_{u} \sum_{v \in s(u)} \left[q_u q_v - \mathrm{sim}(u, v)\right]^2; \tag{1}$$

where $\mathrm{sim}(u, v)$ is the similarity of two nodes measured from the graph structure. The definition of the neighborhood is based on the adjacency matrix. Graph factorization calculates with adjacent neighbors, while GraRep uses higher powers of the adjacency matrix, for example, two-hop neighbors.

As a different method, random walk-based approaches (Grover and Leskovec 2016; Perozzi et al. 2014) sample vertices from the neighborhood of a node. Sampling is done by initiating random walks from node u. Instead of SE, these approaches optimize for cross-entropy loss:

$$\sum_{u} \sum_{v \in s^*(u)} = -\log\left[\frac{\exp(q_u q_v)}{\sum_w \exp(q_u q_w)}\right]; \tag{2}$$

where $s^*(u)$ is a random sample from the neighborhood of u.

Many of the above mentioned algorithms use the Word2Vec model as an underlying abstraction by training the model, using sampled walks analogously to sentences, and using the learned embeddings as node embeddings. We follow this approach, and also investigate the use of either the input ($W1$) or the output embedding ($W2$) of the model (Press and Wolf 2016) as the vector space representation of the graph.

The above models learn static embeddings on graph snapshots; however, they mention extensions towards online learning from graph streams. In DeepWalk (Perozzi et al. 2014), the possibility of an online incremental update is proposed but not analyzed. An incremental update for LINE with a batch of edge insertions and deletions is described in (Yu et al. 2018), but no attempt is made to analyze the online, single edge insertion behavior. Closest to our work is the continuous-time dynamic network embedding result (Nguyen et al. 2018), which does not learn online but computes an embedding for a single point in time. Similarly, the HTNE algorithm (Zuo et al. 2018) produces temporal node embeddings, but training is done in batch instead of executing online updates. A promising direction for computing the embedding dynamically involves recurrent neural networks, for example, Long Short-Term Memory networks (Wang et al. 2015); however, the applicability for graphs is not yet explored.

Dynamic vector space embedding methods in edge streams

We describe two node embedding approaches that are applicable in edge streams. The input of both algorithms consists of an edge stream (u, v, t) ordered by time t in which each edge can occur multiple times. As required by the data stream algorithmic model, we process the edges in the order of arrival without storing the entire input.

Our goal is to dynamically learn node representations by reflecting the current node similarity structure of the evolving graph as we dynamically change the location of the nodes in the vector space. To this end, we give two embedding methods in the next two subsections. Two nodes are required to be mapped close in the vector space whenever they lie on short paths formed by recent edges in the first model, and whenever the set of their recent neighbors is similar in the second model.

Similarity based on reachability through short temporal walks

In our first algorithm, our goal is to enforce that the embedding of node v be similar to the embedding of nodes with the ability to reach v across edges that appeared recently, as shown in Fig. 1. In other words, the embedding of a node should be similar to the embedding of nodes in its temporal neighborhood. We define time respecting *temporal walks* (Rozenshtein and Gionis 2016) in order to sample for each node u at any time t nodes from its temporal neighborhood. As seen in Fig. 2, a temporal walk consists of adjacent edges ordered in time:

$$z = (u_0, u_1, t_1), (u_1, u_2, t_2), \cdots, (u_{j-1}, u_j, t_j); t_{i-1} \leq t_i. \tag{3}$$

For example, there are three temporal walks leading to node v in Fig. 4: e_1, e_3, and e_2, e_3. Since edges can appear multiple times, we consider the edge set as a multiset and distinguish between the walk (e_2, e_3), which is a temporal walk, from (e_2, e_1), which is not, since e_1 comes earlier than e_2.

To define the similarity, we want to give more weight to shorter walks and more weight to fresh edges. Towards this end, for a temporal walk where edges appeared at $(t_1, t_2, \ldots, t_j)$, we define the probability of the walk at time t as

$$p(z, t) := \beta^{|z|} \cdot \prod_{i=1}^{j} \gamma(t_{i+1} - t_i); \tag{4}$$

where $\beta \leq 1$ is an exponential decay on the length of the walk, $t = t_{j+1}$, and $\gamma(\tau)$ is a time-aware weighting function that is based on the delay τ between adjacent edges. The concept of (4) is that a walk is more likely if edges along the walk appeared close to each other in time. We use exponential time weight $\gamma(\tau) := \exp(-c\tau)$. Since $\gamma(a) \cdot \gamma(b) = \gamma(a + b)$, the probability of the path in (4) becomes

$$p(z, t) = \beta^{|z|} \exp(-c(t - t_j)) \cdot \ldots \cdot \exp(-c(t_2 - t_1)) = \beta^{|z|} \exp(-c(t - t_1)). \tag{5}$$

The notation is summarized in Table 1.

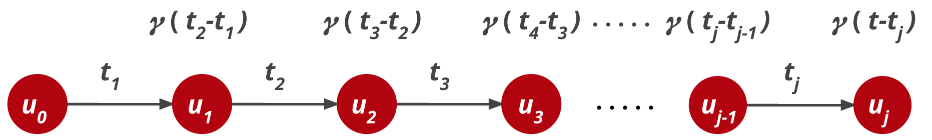


Fig. 2 Temporal walk. Adjacent edges in the graph are in time respecting order. The probability of the walk is based on the delays between the apperances of the adjacent edges

Table 1 Notations used in the StreamWalk algorithm

z	a temporal walk
uv	the new directed edge that is being processed in the edge stream
$t(xy)$	time of arrival of a multi-edge instance
t_u	last time an edge arrived leading to u
β	decay exponent for the length of the walk
γ	time-aware edge weighting function
$p(z,t)$	weight of walk z at time t
$p(w,t)$	sum of weight of walks ending in node w at time t
$p(xy)$	sum of weight of walks ending with edge xy at time $t(xy)$

Temporal walk sampling from edge stream

Given a node v, a naive idea would be to compute the walk weight $\sum\{p(z,t)\ :\ z$ is a temporal walk from w to $v\}$ for all other nodes w and set the embedding of w close to that of v proportional to the walk weight. The problem with this approach is that it requires a time consuming walk enumeration procedure at each time instance, and has no ability to update the similarity measure by focusing only on the new edges as they arrive.

Given the new edge uv that arrives at time t, we would like to only consider walks from any w that reach v by the new edge uv. Towards this end, we propose a sampling update procedure for temporal walks as follows. We select a start node w of a random temporal walk z ending in u with probability proportional to $p(z,t)$ in (5); see Fig. 3. We generate the walks by taking steps backwards from u. To make sure the walks are temporal, we always use edges that appeared before the previous one. Among the possible edges entering the current node, we select proportional to the time-aware weighting function γ. For example, in Fig. 3, we select t_5 backwards from u, and then t_3 backwards from the next node. Finally, we also compute a stopping probability corresponding to the length decay β so that we select no new edge from w in the example; the actual formula (10) is explained later.

The actual implementation is somewhat tricky in that we have to handle multi-sets of edges. A way to illustrate the implementation is to consider an edge uv that appears before another wu and then reappears, see Fig. 4. The second instance can form a temporal walk w, u, v, while the same walk is not temporal with the first instance of uv. However, the

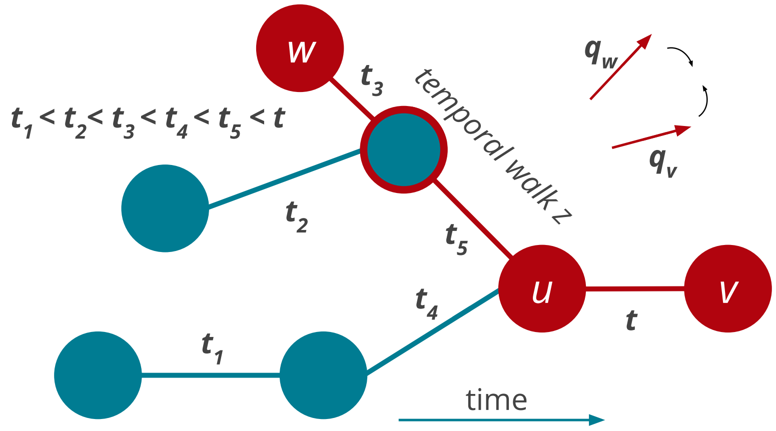


Fig. 3 StreamWalk: learning embeddings based on random walks to the past in temporal networks

second instance of uv has a higher edge weight γ, hence we have to store the weight of the first instance as well to be able to correctly compute the weight of all temporal walks that reach node v.

The implementation of the StreamWalk algorithm

In Algorithm 1, we describe StreamWalk, our implementation of temporal walk sampling. Recall that the notation is summarized in Table 1. For every edge uv in the multi-set of edges arriving in the stream, we maintain the total weight of all walks ending at v at time $t(uv)$:

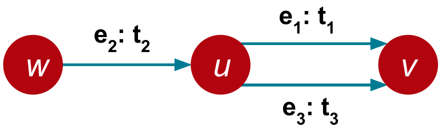
$$p(v, t(uv)) = \sum_z p(z, t(uv)); \tag{6}$$

where we sum over all temporal walks z ending in v using edges arriving no later than $t(uv)$. The actual computation in procedure UPDATEWALKS accumulates the weight of the walks seen in Fig. 5. There is a new single edge temporal walk uv with weight β. Furthermore, we can continue each temporal walk z that ended in u before $t(uv)$ with uv. The total weight of these walks is $p(u, t_u) \cdot \beta \cdot \exp(-c(t(uv) - t_u))$ where t_u is the most recent timestamp for which $p(u, t_u)$ is known. In other words, t_u denotes the last time an edge entering u arrived in the edge stream. The exponential term accounts for the time decay of temporal walk weights since the arrival of this last edge entering u. Finally, we add all the walks that terminated at v before, with exponential time decay. The final formula becomes

$$p(v, t(uv)) \leftarrow \beta \cdot (1 + p(u, t_u) \cdot \exp(-c(t(uv) - t_u))) + p(v, t_v) \cdot \exp(-c(t(uv) - t_v)); \tag{7}$$

where t_v is the most recent timestamp for which $p(v, t_v)$ is known. The update rule is illustrated in the last step in Figs. 4 and 5.

For each edge uv in the stream, we finally update the embedding of v by sampling a fixed number of temporal walks ending in u; we do this by calling procedure SAMPLEWALKS k times as described at the end of this section. Given the start node w of a walk in the sample, we optimize for the similarity of the embedding pair (q_v, q_w) with stochastic gradient descent. For loss function, we either set MSE or cross-entropy as in Eqs. (1) and (2). In the case of MSE, for each w we apply online negative sampling (Pálovics et al. 2014) by selecting pairs vw' proportional to the popularity of w' in the edge stream up to the current timestamp. We refer to (Kaji and Kobayashi 2017) for online incremental updates for cross-entropy based loss.



time	$p(e_1, t)$	$p(e_2, t)$	$p(e_3, t)$	$p((e_2, e_3), t)$	$p(v, t)$
$t = t_1$	β	n/a	n/a	n/a	$p(e_1, t_1) = \beta$
$t = t_2$	$\beta\gamma(t_2 - t_1)$	β	n/a	n/a	$p(e_1, t_2) = \beta\gamma(t_2 - t_1)$
$t = t_3$	$\beta\gamma(t_3 - t_1)$	$\beta\gamma(t_3 - t_2)$	β	$\beta^2\gamma(t_3 - t_2)$	$p(e_3, t_3) + p((e_2, e_3), t_3) + p(e_1, t_3) =$ $\beta(1 + p(e_2, t_3)) + \gamma(t_3 - t_2)p(v, t_2)$

Fig. 4 Computation of $p(v, t)$ for the arrival times $t_1 < t_2 < t_3$ of the three edges e_1, e_2, and e_3. The bottom right cell illustrates the update formula (7)

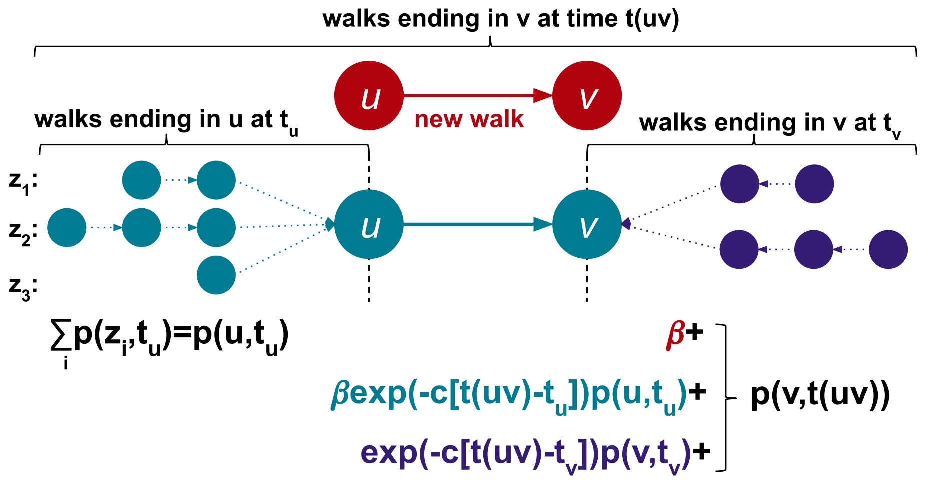


Fig. 5 Whenever a new edge uv appears, a new walk starts from u (red), and each temporal walk (z_1, z_2, z_3) that ended in u up to time t_u continues via uv (blue). We get $p(v, t(uv))$ by summing up the contribution of the previous two type of walks (red and blue) with the decayed weight of walks that have already reached node v (purple) before time $t(uv)$

Since we train by sampling k walks per edge, time complexity is affected by the cost of sampling temporal walks. To reduce storage, we can work over a sliding window of the stream and periodically remove the oldest edges; these edges will already have a very small γ value.

Finally, we describe the algorithm to sample temporal walks as implemented in Procedure SAMPLEWALKS of Algorithm 1. Our goal is to sample proportional to $p(y, \tau)$ at a given time τ. We define a random walk backwards from y. We select a backward edge with probability proportional to the weight of walks ending with that edge, which we define as

$$p(xy) = \sum_z p(z, t(xy)); \tag{8}$$

where z are temporal walks ending with the given instance of the edge xy that appeared at time $t(xy)$. Recall that the edges are taken from a multi-set. The value of $p(xy)$ can be calculated as follows. From the total temporal walk weight ending in y at time $t(xy)$, we have to subtract the total weight of all walks ending in y before $t(xy)$; the difference contains the weight of only those paths that use the edge instance xy of timestamp $t(xy)$:

$$p(xy) = p(y, t(xy)) - p(y, \bar{t}) \cdot \exp(-c(t(xy) - \bar{t})); \tag{9}$$

where $\bar{t} < t(xy)$ is the timestamp of the last edge in the stream entering y before $t(xy)$. The exponential term corresponds to the time decay of the walk weight since time $\bar{t}$.

We also define the termination for the walk, which is based on the contribution of the single node y as a zero-edge walk relative to all other walks that end at y. At any time of observation τ, the weight of the zero-edge walk is 1, and the total weight of the remaining walks is $p(y, t)$ for the last recorded time $t \leq \tau$, decayed proportional to the elapsed time, $\tau - t$. Hence with the probability below, we take no further steps but stop the walk:

$$1/(1 + p(y, t) \cdot \exp(-c(\tau - t))). \tag{10}$$

The steps of Procedure SAMPLEWALKS are summarized as follows.

1 We start the random walk from $y \leftarrow u$ and set $\tau =$ now.

Algorithm 1 *StreamWalk.*

```
procedure UpdateWalks($u, v$)
                                        ▷ Update the weight for all walks ending at $v$
    $t_u, t_v \leftarrow$ last timestamp such that $p(u, t_u)$ and $p(v, t_v)$ are known, respectively
    $p(v, \text{now}) \leftarrow \beta \cdot (1 + p(u, t_u) \cdot \exp(-c(\text{now} - t_u))) + p(v, t_v) \cdot \exp(-c(\text{now} - t_v))$
    $t(uv) \leftarrow \text{now}$
end procedure

procedure SampleWalks($y, \tau$)        ▷ Recursively sample a temporal walk ending at $y$
    $t \leftarrow$ most recent timestamp with $t \leq \tau$ such that $p(y, t)$ is known
    $p(y, \tau) \leftarrow p(y, t) \cdot \exp(-c(\tau - t))$
    With probability $1/(1 + p(y, \tau))$ do
        return $y$
    else
        for all $xy$ multi-edges with $t(xy) < \tau$ do
            Select $x$ with probability $p(xy) \cdot \exp(-c(\tau - t(xy)))/p(y, \tau)$
        end for
        return SampleWalks($x, t(xy)$)
end procedure


procedure StreamWalk($u, v$)
                                        ▷ Update embedding for $v$
    call *UpdateWalks*($u, v$)
    repeat $k$ times
        $w \leftarrow$ SampleWalks($u$, *now*)
        Optimize the representations $q_w$ and $q_v$ by Eqs. (1) or (2)
end procedure
```

2 With probability such as in Eq. 10, we stop the walk and return the current node y.
3 Optionally, we can also terminate the walk if its length reaches a predefined limit.
4 Else, we select an edge xy with $t(xy) < \tau$ with probability proportional to the time-decayed total weight of walks ending with xy, which is $p(xy) \cdot \exp(-c(\tau - t(xy)))$ by definition.
5 We repeat from step 2 by setting $y \leftarrow x$ and $\tau \leftarrow t(xy)$.

As the final implementation details, we can sample by selecting a random value between zero and $p(y, \tau)$ and binary search in the multi-set of xy edges ordered by $t(xy)$. For a given edge xy, we compute $p(xy)$ by Eq. 9 and continue the binary search based on the time-decayed value $p(xy) \cdot \exp(-c(\tau - t(xy)))$. Lastly, it can happen that sampling intends to select a very old edge that was already deleted from the sliding window. This happens when binary search does not terminate at the oldest t still kept in the records. In this case, we can repeat the sampling with a new random value.

Online learning of second order node similarity

Our next online algorithm optimizes the embedding to match the neighborhood similarity of the nodes, which we call second order proximity by following (Tang et al. 2015). Our goal is to optimize for (1) online, by considering $\mathrm{sim}(u,x)$ as a time-aware Jaccard similarity of the neighborhood of u and x, as illustrated in Fig. 6. We consider the neighbors y of u as a multi-set $N(u,t)$ in which we use the decayed weight of edge uy as the weight of y:

$$w(y) = \exp(-c(t - t(uy))); \tag{11}$$

where $t(uy)$ is the time the corresponding instance of edge uy appeared in the stream. Whenever we add a new edge to u, we discard elements $y \in N(u,t)$ with probability $1 - w(y)$. This way we emphasize the importance of new edges and also limit the size of $N(u,t)$ by discarding old edges with low weight that have little effect on similarity values.

In order to design a streaming algorithm to compute second order similarity, we face the same problems as in the StreamWalk algorithm: we want to focus on the increase of similarity when we add a new edge uv, and we want to avoid the costly full computation of similarities of u with all neighbors x of v. Note that the similarity of x and u depend on their neighborhood, which means that all nodes of distance two from v should be enumerated for the full computation. In the next subsection, we describe a randomized approximation method for neighborhood similarities based on (Fogaras and Rácz 2005), which will be used in our final algorithm.

Approximation by fingerprinting

Our algorithm relies on MinHash fingerprinting (Broder et al. 2000) to approximate the Jaccard similarity. The notations are summarized in Table 2. Let there be k independent random permutations over the nodes π_i for $i = 1 \ldots k$. We define the k fingerprints of A as

$$h_i(A) := \mathrm{argmin}\{\pi_i(a) : a \in A\}; \quad h_i(\emptyset) = \mathrm{NaN}; \quad i = 1 \ldots k; \tag{12}$$

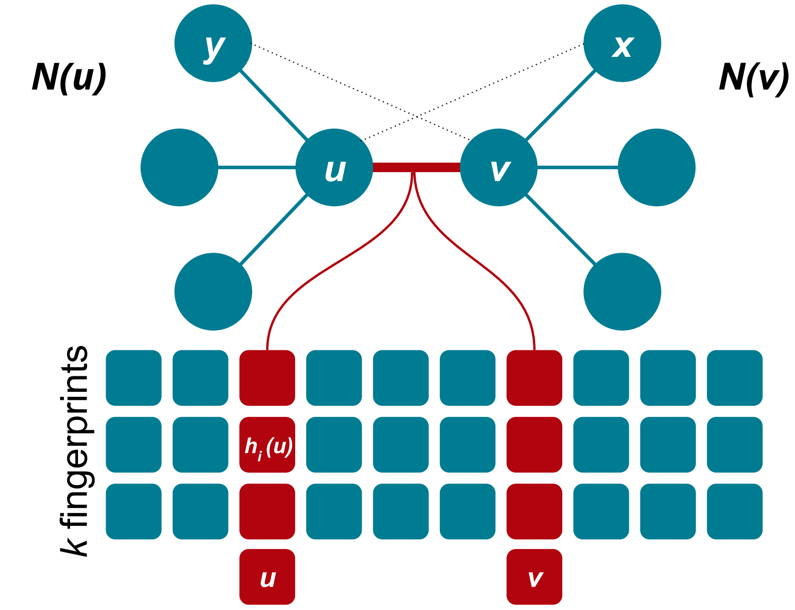


Fig. 6 Online learning of second order node similarity

Table 2 Notations used in the second oder similarity algorithm

uv	the new directed edge that is being processed in the edge stream
$t(xy)$	time of arrival of a multi-edge instance
π_i	a permutation of the entire vertex set, fixed in time
$N(u,t)$	the pruned neighbors of u at time t
$h_i(u) = h_i(N(U,t))$	the i-th fingerprint of u at time t

For short,

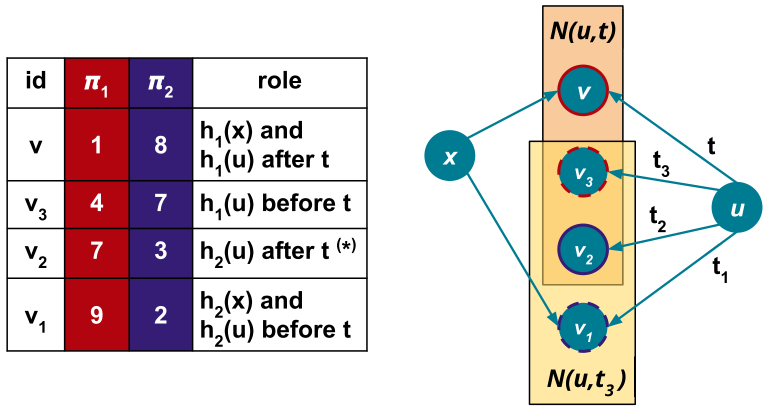
$$h_i(u) := h_i(N(u,t)). \tag{13}$$

We maintain k fingerprints defined in (12) for the neighborhood of each node where the weights of the elements are defined by (11). We approximate the time-aware Jaccard similarity of any node pair with the fraction of common fingerprint values:

$$\mathrm{sim}(u,v,t) \approx \sum_{i=1}^{k} I[\,h_i(u) = h_i(v)]\,/k. \tag{14}$$

We illustrate the fingerprinting idea in Fig. 7 for $k = 2$. The two fingerprints of u, $h_1(u)$ and $h_2(u)$, are defined based on two permutations π_1 and π_2 of the entire vertex set. The permutations are fixed, but the fingerprints change in time as new edges arrive and past edges become too old and get removed from $N(u,t)$.

Next, we show how the similarity of u and a neighbor x of v can be approximated in the example of Fig. 7. Assume that $h_1(x) = v$ and $h_2(x) = v_1$. By using formula (14), before edge uv arrives, the similarity approximation is $\mathrm{sim}(u,v,t_3) \approx (0+1)/2$ as $h_2(x) = h_2(u) = v_1$ at time t_3. When edge uv arrives, the similarity will on one hand increase, since $h_1(u)$ gets assigned with v. On the other hand, the similarity can decrease as edges become too old. For example, if we drop edge uv_1, equation $h_2(x) = h_2(u) = v_1$ will no longer hold. However, since we want to avoid the cost of updating $h_i(x)$ for all i and all neighbors x of v, we heuristically only consider the increase of similarity, which can be caused by adding v as new fingerprint of u.



id	π_1	π_2	role
v	1	8	h_1(x) and h_1(u) after t
v_3	4	7	h_1(u) before t
v_2	7	3	h_2(u) after t (*)
v_1	9	2	h_2(x) and h_2(u) before t

Fig. 7 Illustration of how the fingerprints of node u change when adding the new edge uv. Neighbors of u are ordered in time as $t_1 < t_2 < t_3 < t$. Two fixed random permutations π_1 and π_2 define the fingerprints $h_1(u)$ and $h_2(u)$. In π_1 (red), v has minimum value, hence the previous $h_1(u)$ will be reassigned to v. In π_2 (purple), the minimum is the oldest node v_1, which becomes too old and gets removed from $N(u,t)$. The correct value for $h_2(u)$ would be v_2 ($\star$). Instead we heuristically set $h_2(u) = v$ after the removal of v_1

Algorithm 2 Online learning second order similarity

```
procedure UpdateFingerprints($u, v$)
    for all $i$ in $1 \dots k$ do
        if $h_i(u)$ is too old or $\pi_i(v) < \pi_i(h_i(u))$ then
            $h_i(u) \leftarrow v$
        end if
    end for
end procedure

procedure GetSimilarityDelta($u, v, x$)
    $\ell \leftarrow 0$
    for all $i$ in $1 \dots k$ do
        if $h_i(u) = h_i(x) = v$ then
            $\ell \leftarrow \ell + 1$
        end if
    end for
    return $\ell$
end procedure

procedure OnlineSecondOrderSim($u, v$)
    UpdateFingerprints($u, v$)
    for all in-neighbors $x$ of $v$ that are not too old do
        $\ell \leftarrow$ GetSimilarityDelta($u, v, x$)
        Optimize the representations $q_u$ and $q_x$ repeated $\ell$ times, by using Eqs. (1) or (2)
    end for
    Repeat with $v$ and $u$ swapped and edge directions reversed
end procedure
```

As a final heuristic, in our implementation we always replace fingerprints corresponding to pruned neighbors by v, since obtaining the π_i values of the entire neighborhood is computationally costly. In the example of Fig. 7, we drop edge uv_1 as $t_1 \ll t$. The correct new value of $h_2(u)$ would be the next oldest vertex v_2, however this can only be calculated by enumerating all neighbors of u. Instead, in our implementation we heuristically assign $h_2(u) \leftarrow v$.

Algorithm for online learning second order similarity

Our method is described in Algorithm 2 by using the notations in Table 2. Our goal is to approximate the change of similarity between u and the in-neighbors x of v, and modify the embedding vectors whenever certain x gets more similar to u after adding the new edge uv. Note that x becomes more similar if the edge xv also appeared recently; in terms of fingerprints, this means that for some fingerprint index i, both x and u have v as fingerprint node. We perform the steps below to update the fingerprints of u and check for v as fingerprint in the in-neighbors x of v:

1 For node u, we maintain the present neighborhood $N(u)$ by removing very old edges, and recompute the k fingerprints $h_i(u)$ for $i = 1 \dots k$ by calling

Procedure UPDATEFINGERPRINTS. Fingerprint $h_i(u)$ can take the new value v for the new edge uv if it is too old or if $\pi_i(v)$ becomes the new MinHash value for permutation i. In the former case, we can either heuristically replace $h_i(u)$ with the new neighbor v or compute the true MinHash value $\text{argmin}\{\pi_i(y) : y \in N(u)\}$.

2 Finally, for each in-neighbor $x \in N(v)$, we compute the number of fingerprints ℓ that match those of u and have value v in Procedure GETSIMILARITYDELTA, and ℓ times optimize the representations q_u and q_x by using Eqs. (1) or (2).

Symmetrically, we also check for the similarity increase of v with the out-neighbors of u by performing the same steps, replacing u and v on the reverse direction graph.

Similarity search experiments

In this section, we describe our main evaluation, in which we assess how well the closeness of two nodes in the embedding reflect their similarity against an external ground truth. Towards this end, we first describe a network enriched with a time dependent external similairty ground truth information. Then, at a time instance, we compute the list of nodes closest to selected ones in the embedding, and compare these lists against the similarity ground truth.

We analyze node embedding methods for similarity search over the Twitter tennis tournament collections of (Béres et al. 2018). For the quantitative analysis, we use the annotation of the nodes for the accounts of the tennis players that participate in a game on a given day. In this sense, we expect that the players of the same day are more similar than other players and non-player accounts, as we will describe in "Evaluation metrics" section. We compare the performance of StreamWalk and online second order similarity with online and static baseline methods, which we will describe in "Baseline models" section.

Tennis tournament twitter collection data

In (Béres et al. 2018), we compiled two separate tweet collections: RG17 for Roland-Garros, the French Open Tennis Tournament, and UO17 for US Open, the United States Open Tennis Championships, which we use in our first experiment. We use the mention graphs extracted from the last 15 and 14 days of RG17 and UO17, respectively. Based on the approximate time of the games, we consider a Twitter account n **active** on the given day, if it belongs to a tennis player who participated in a completed, canceled, or resumed game.

Evaluation metrics

We evaluate similarity search by a supervised experiment in which for each active account we consider the other similar active accounts on the given day. For each embedding algorithm, we generate 128-dimensional node representations every six hours (6:00, 12:00, 18:00, 24:00). For online methods, we perform continuous updates over the edge stream. For the static methods, we build the corresponding graph snapshots.

We use NDCG (Al-Maskari et al. 2007) to evaluate how other active accounts are similar to a selected one. NDCG is a measure for ranked lists that assigns higher score if active accounts appear with higher rank in the similarity list. In our experiments, we compute the average of the NDCG@100 for the active accounts as query nodes to measure the performance of a single model in any given snapshot.

Baseline models

We compare StreamWalk and online second order similarity to *online* (or time-aware) and *static* (or batch) embedding methods. Online models are updated after the arrival of each edge. By contrast, static representations are only updated once every six hours when the graph snapshot ends. At hour t a static model is computed on the graph constructed from edges arriving in time window $[t - T, t]$ from the edge stream. For each batch baseline, we experimentally select the best value of T.

We consider four static centrality measures as baseline:

- DeepWalk (Perozzi et al. 2014)
- Node2Vec (Grover and Leskovec 2016)
- LINE (Tang et al. 2015): the first and the second order versions of LINE, as well as the combination of the two versions
- Static indegree, calculated in time window $[t - T, t]$ by counting each edge with multiplicity

Furthermore, we compare our proposed algorithms with a simple online baseline:

- Decayed indegree, defined for node u at time t as

$$\sum_{zu \in E(t)} \exp(-c(t - t_{zu})); \tag{15}$$

  where $E(t)$ is the multi-set of edges that occurred up to time t with edge activation time t_{zu}.

We use the 128-dimensional representations of StreamWalk, second order similarity, DeepWalk, Node2Vec, and LINE to measure node similarity over time. For the two degree methods, we rank by degree without reference to the query node in the NDCG@100 formula.

Results

In our experiments, we measure how the similarity of node representations evolves over time by a supervised evaluation in which the active nodes should be similar to each other. We show two different ways to describe the performance of a single model:

1 For each day, we present the mean NDCG@100 of the snapshots evaluated at 6:00, 12:00, 18:00, and 24:00.
2 As a single global value (NDCG@100), we take the average of NDCG@100(u) for each daily player u in every snapshot.

For a given parametrization of every embedding-based method, we always show the average performance of ten independent instances.

During our experiments, we found that the following parameters had a great impact on the quality of online node embeddings, see Table 3:

W1 and W2: In Word2Vec, we have the option to optimize node representations for the input ($W1$) or the output ($W2$) matrices (Press and Wolf 2016). It is application dependent whether $W1$ or $W2$ yields the better representation. For SW, we achieved the best results by $W2$.

Initialization: We experimented with Xavier (Glorot and Bengio 2010) and uniform random initialization of $W1$ and $W2$.

Table 3 Mean global performance (NDCG@100) of StreamWalk with different settings on the first three day of RG17 and UO17 respectively

matrix	init type	mirror	decayed	RG17	UO17
W1	Xavier	yes	no	0.1865	0.1695
W1	uniform	yes	no	0.2001	0.1818
W2	uniform	yes	no	0.2898	0.2386
W2	uniform	no	no	0.3272	0.2898
W2	uniform	no	yes	0.3341	0.2999

Mirror: In our algorithms, the input to Word2Vec consists of node pairs. Given a training instance (x, y), we *mirror* if we feed both (x, y) and (y, x), not just (x, y).

Decay: We heuristically map the representations of nodes with no recent activity to the null vector.

Negative sampling rate and past positive samples used: Key parameters of Word2Vec analyzed separately in Figs. 8–9. Past positive samples are edges that appeared longer time ago; using such edges for negative training helps forgetting the past.

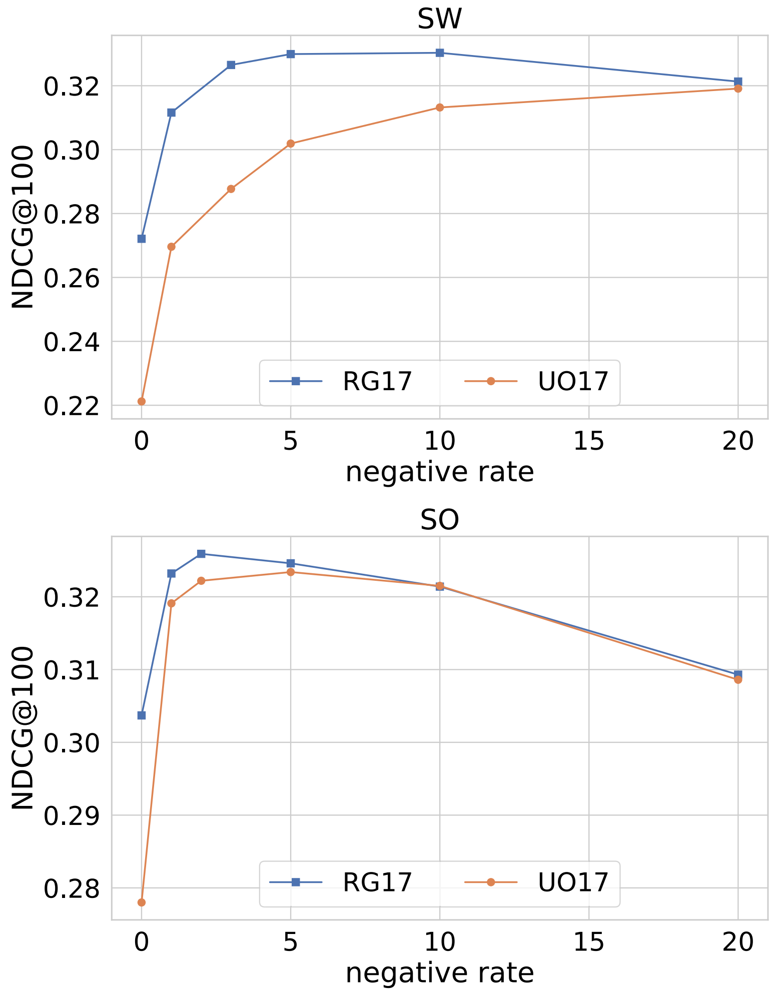


Fig. 8 The effect of negative sample rate on the global mean performance (NDCG@100) of StreamWalk (**top**) and online second order similarity (**bottom**)

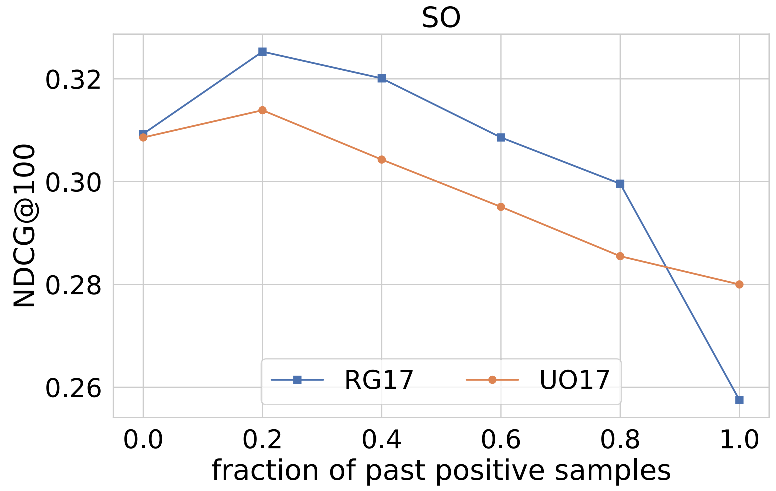


Fig. 9 The effect of past positive samples used as negative samples for SO when the latest edge arrives

Next, we examine the quality of node representations with respect to node pair *sampling-related parameters*:

Time decay and half-life: By transforming the time decay parameter c, we show our results as the function of half-life $h = \ln(2)/c$ in Fig. 10.

Number of walks sampled: The number of new training instances for SW and SO for a new edge arrival is a parameter analyzed in Fig. 11.

We also combine the output of StreamWalk and second order similarity by using the weighted average of the corresponding inner products as similarity. This method denoted as SW+SO outperforms SW and SO, as seen in Fig. 12. The optimal weight of SO in the combination is 0.3 for both RG17 and UO17.

In Table 4 we present the best global mean performance for each model. Fig. 13 shows the daily mean performance of the best models.

For illustration, in Table 5 we present the 20 accounts most similar to that of Rafael Nadal for different node embeddings on 2017-May-31 18:00. Since Rafael Nadal played on this day, the active accounts (yellow) belong to tennis players who participated in a game on this day. The combined model SW+SO has the highest number of active player accounts. Furthermore, accounts present in both SW and SO columns (e.g. *BMATTEK*, *DjokerNole*, *GrigorDimitrov*, etc.) typically achieve higher position by SW+SO than SW. It is interesting to see that SW and SO find different active accounts, which explains why the combination SW+SO achieves superior performance. While static LINE and Node2Vec have less relevant hits than our online methods, most of the irrelevant accounts still belong to tennis players (e.g. *andy_murray*, *stanwawrinka*, etc.). The main difference is that the daily active players are better found by the online than the static methods.

Online link prediction

Next, we address the online, time-aware variant of the link prediction problem, in which we give a prediction for the next edge as we process the stream edge by edge. Our goal

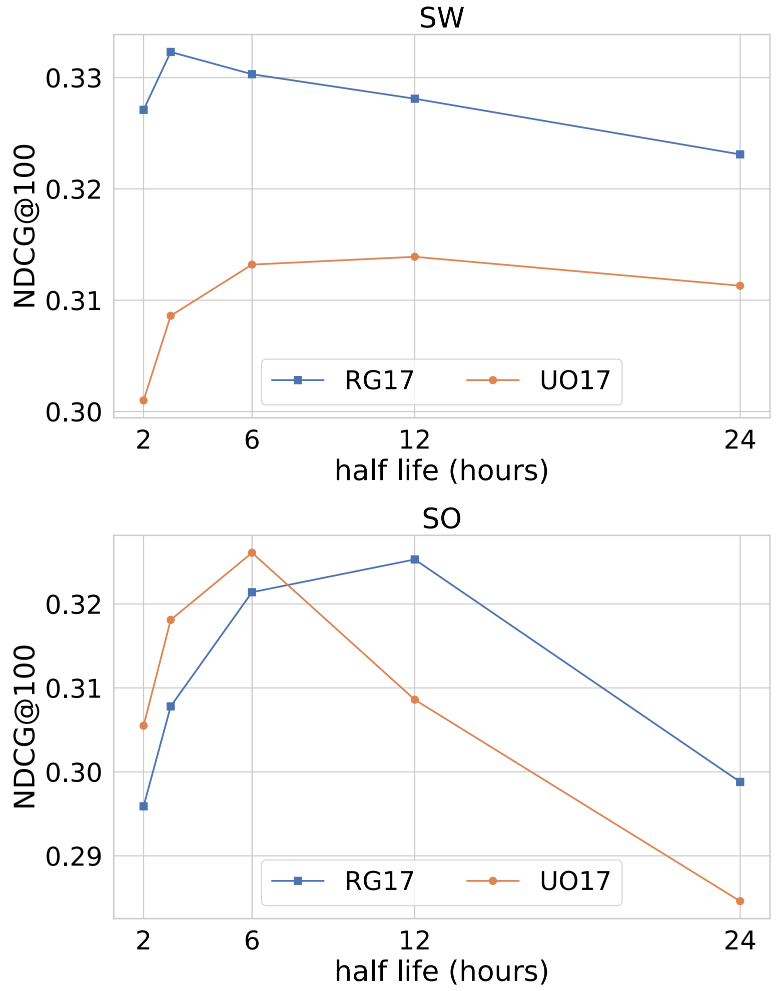


Fig. 10 The effect of half-life on the global mean performance (NDCG@100) of StreamWalk (**top**) and online second order similarity (**bottom**)

is to predict a new link at a given time based on all events that appeared before, including the arrival of the most recent edges. Compared to the predictions given at time t, at time $t+1$, we can potentially reconfigure our model based on the edges appeared at time t and give a very different prediction for the next set of links. If we compare with a traditional model based on graph snapshots, the traditional model will output the exact same ranked list of links between two snapshots. While our modeling technique provides much stronger time awareness, it poses a challenge for evaluation, since we cannot compare just a single prediction against a larger set of edges, but a large set of potentially very different predictions ordered in time.

To evaluate online link prediction methods, we use the prequential evaluation framework (Dawid 1984). As explained in Fig. 14, before a new edge (u, v, t) arrives in the graph stream, we first give an attempt to predict this edge, then reveal the edge and update the model using the new edge. In this way, we can incorporate information on the most recent edges in our model and evaluate potentially completely different predictions, coming from modified models, at every new time tick.

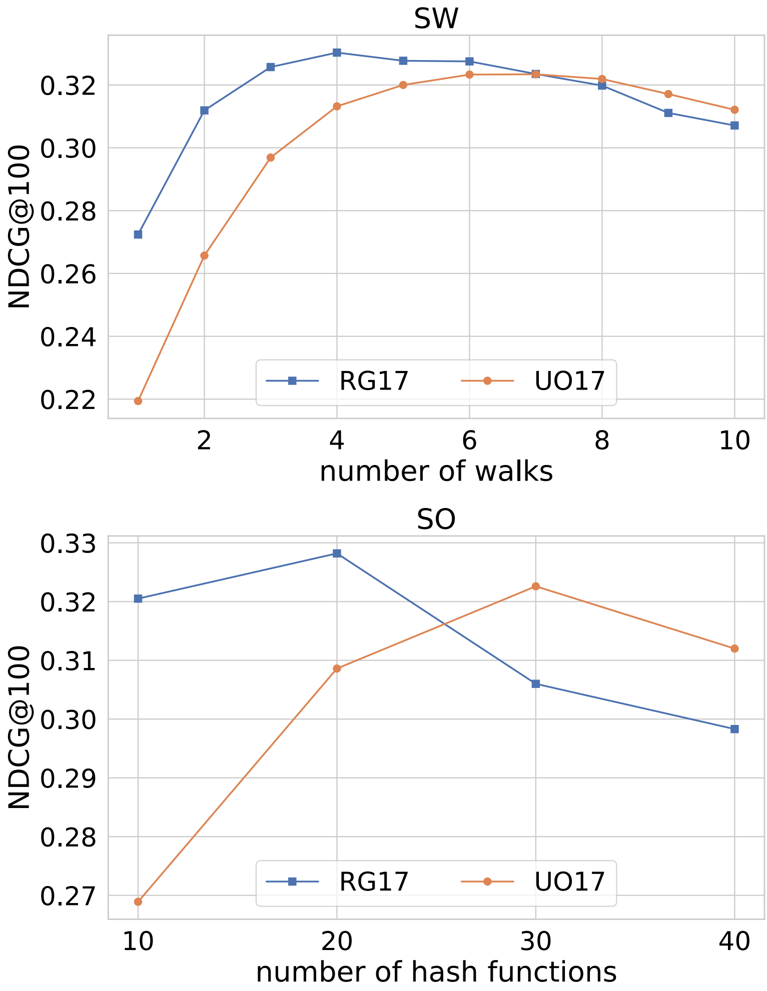


Fig. 11 The effect of sampled walks (**top**) and hash functions (**bottom**) on the global mean performance (NDCG@100) for SW and SO respectively. These parameters control the number of sampled node pairs we feed to online Word2Vec at every edge arrival

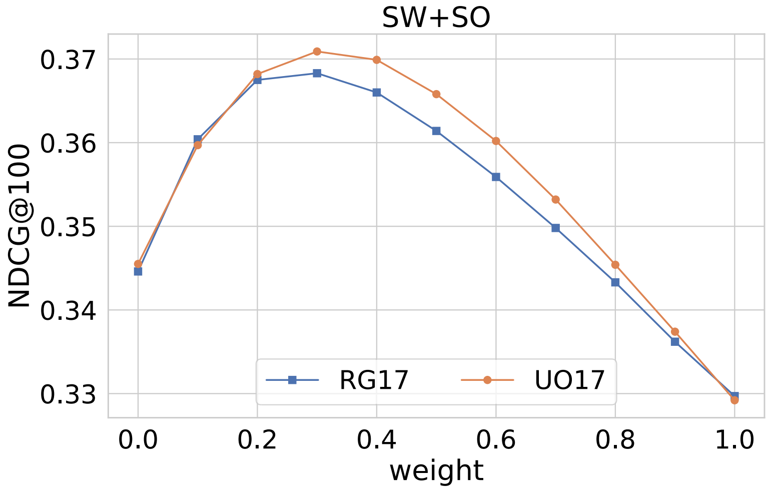


Fig. 12 Global mean performance of the combination of SW and SO with respect to the weight of SO. The combined model (SW+SO) has superior performance with the optimal weight 0.3

Table 4 Best mean global performance (NDCG@100) of each model for the Twitter tennis data sets

model	RG17	UO17
SW+SO	**0.3683**	**0.3709**
SW	0.3446	0.3455
SO	0.3283	0.3261
LINE	0.2946	0.2936
Node2Vec	0.2617	0.2562
decayed indegree	0.2973	0.2548
static indegree	0.2288	0.2073

Best performing methods are marked boldface

For evaluation, we use a single-point variant of NDCG (Pálovics et al. 2014) in which there is always exactly one relevant item, the actual new edge, and the higher the rank of the relevant edge, the higher the score. The overall evaluation of the model is the average of the single-point DCG@20 values over all events in the graph stream. We can also assess performance trends by computing daily or weekly averages of the DCG. We note that we ignore reappearing edges and only evaluate the prediction for those edges that appear the first time in the stream.

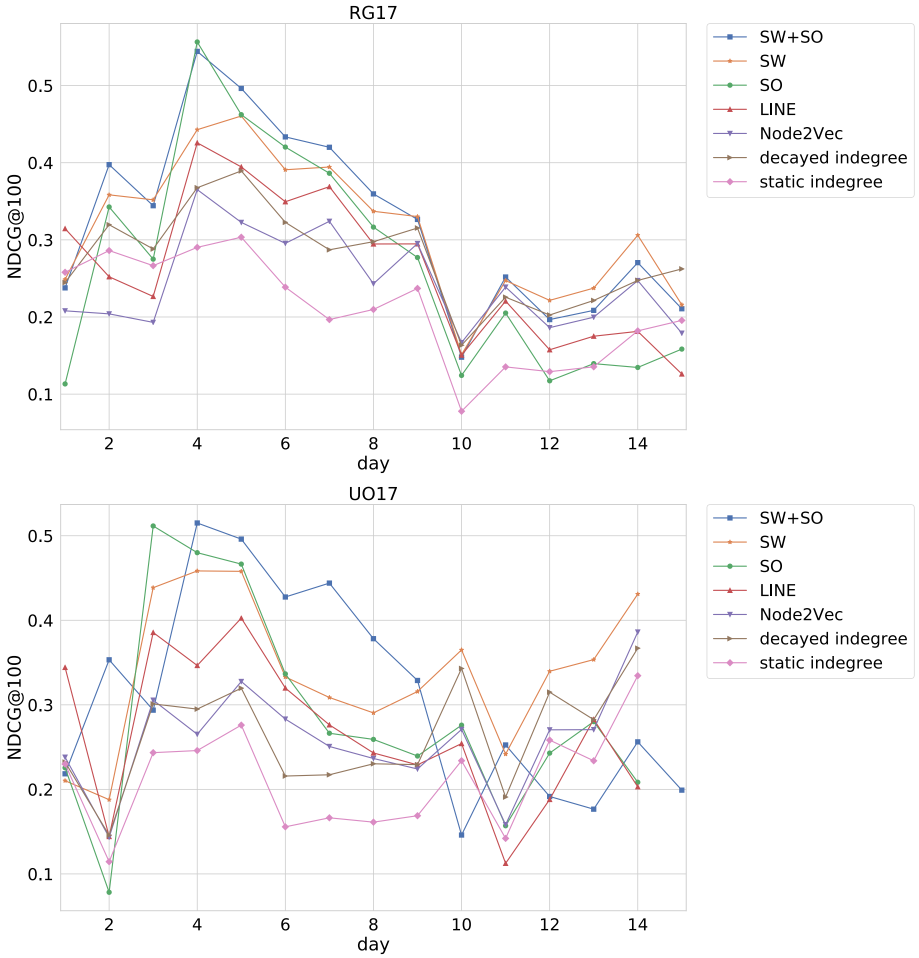


Fig. 13 Best model performance over time for RG17 (**top**) and UO17 (**bottom**) tennis data sets. For each day the average NDCG@100 over the four daily snapshots is presented

Table 5 Similarity list of Rafael Nadal for models based on embeddings generated at 18:00 on May 31 (fourth day of RG17)

	SW+SO	SW	SO	LINE	Node2Vec
1	robin_haase	robin_haase	NikaBasil	DjokerNole	renzolii
2	KikiMladenovic	KikiMladenovic	Amandine_Hesse	benoitpaire	Gael_Monfils
3	BMATTEK	SJohnson_89	GarbiMuguruza	andy_murray	KikiMladenovic
4	SJohnson_89	BMATTEK	GrigorDimitrov	FerVerdasco	DjokerNole
5	borna_coric	borna_coric	DjokerNole	GarbiMuguruza	dieschwartzman
6	rolandgarros	rolandgarros	milosraonic	tsonga7	tsonga7
7	DjokerNole	CaroWozniacki	p2hugz	robin_haase	andy_murray
8	CaroWozniacki	DjokerNole	AljazBedene	delpotrojuan	AndreAgassi
9	cicibellis99	cicibellis99	Petra_Kvitova	Gael_Monfils	keinishikori
10	SaraErrani	joaosousa30	rolandgarros	CaroWozniacki	HoracioZeballos
11	GarbiMuguruza	SaraErrani	WTA	stanwawrinka	GarbiMuguruza
12	joaosousa30	GarbiMuguruza	ppauline86	Venuseswilliams	LuksiMladenovic
13	GrigorDimitrov	Venuseswilliams	ThiemDomi	KikiMladenovic	geniebouchard
14	Venuseswilliams	GrigorDimitrov	KikiMladenovic	ThiemDomi	NickKyrgios
15	milosraonic	renzolii	Cibulkova	M_Granollers	TKokkinakis
16	tsonga7	serenawilliams	alizecornet	NickKyrgios	delpotrojuan
17	jimchardy	tsonga7	David__Goffin	Petra_Kvitova	babolat
18	AnettKontaveit	AnettKontaveit	Gael_Monfils	CaroGarcia	Venuseswilliams
19	ThiemDomi	kyle8edmund	BMATTEK	JoKonta91	FerVerdasco
20	David__Goffin	BolelliSimone	TimeaOfficial	richardgasquet1	DavidFerrer87

Active player accounts are highlighted in yellow. With the exception of a few renamed accounts (since 2017) most of the presented accounts can be still searched for on Twitter

Data sets

We experiment on three standard network data sets from KONECT (Kunegis 2013). We selected networks from the collection with timestamped edges evenly distributed in time. For example, we discarded networks that were crawled for several weeks, but a significant part of their edges appeared within one day due to anomalies in the crawl. We discarded self-loops and similar edges with the same timestamps from each data set and processed the links in temporal order.

Enron: The Enron email network consists of emails sent between employees of Enron. Nodes in the network are individual employees and edges are individual emails. The data has 308,708 edge events in 365 days between 27,972 nodes with 90,177 unique edges.

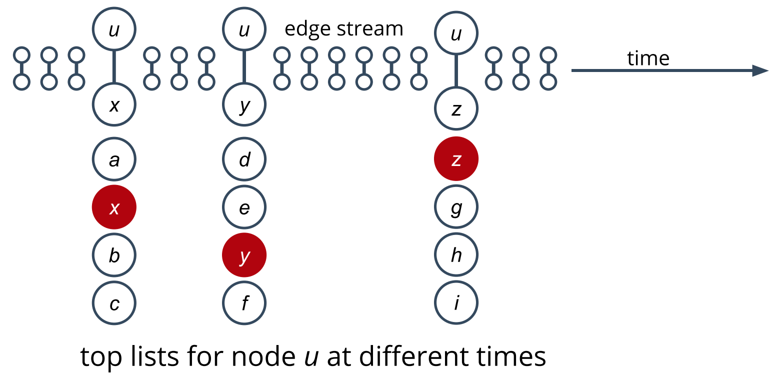


Fig. 14 The online link prediction problem. For each edge in the stream, first we query the model to predict the next interaction. Then we train the model on the observed edge. Whenever node *u* interacts with another node, the model may generate different, updated top-*k* predictions

Linux kernel: The communication network of the Linux kernel mailing list. The nodes are people, and each directed edge represents a reply from a user to another. The data has 487,355 edge events in 1380 days between 16,449 nodes with 88,855 unique edges.

Facebook: The nodes of this network are Facebook users, and each edge represents one post, linking the user writing a post to the user whose wall the post is written on. The data has 16,868 edge events in 658 days between 16,868 nodes with 61,582 unique edges.

Baseline methods

We compare our methods to three batch baselines: Node2Vec, DeepWalk (DW), and Graph Factorization (GF). For these three models, we retrained the model over all past data periodically. We set the periodicity for one day on the Enron data set. We trained models with weekly batch updates on the Linux and Facebook data sets. Furthermore, we give two online baselines. Besides simply updating the degree of each node and using it as a predictor, we experimented with the online version of Graph Factorization. This corresponds to the version of StreamWalk where we do not take any samples but optimize only for the similarity of the node pairs along the edges in the stream.

Results

We analyze the results for online link prediction for the best parameter setting of each algorithm. The results are summarized in Table 6 and in Fig. 15. Our key observation is that the online learning methods, online graph factorization (GF), StreamWalk (SW) and SecondOrder (SO) show very strong performance compared to the batch methods. Note that batch methods such as GF read the list of edges several times to perform stochastic gradient descent, while online methods must process the edges in the order of arrival, without the possibility to access past edges again.

The surprisingly good performance of the online methods is due to the fact that they put emphasis on the more recent edges. For gradient descent with negative sampling, the embedding is always optimized towards the freshly arrived edges, while negative sampling has the effect of forgetting the past. The advantage of model freshness is strikingly strong for the link prediction experiments where the low time granularity prequential evaluation method is used. Online methods also outperform batch embeddings for the Twitter tennis data sets. Note that ground truth data is only available daily in our collections. Higher performance difference can be obtained with labels of higher time granularity, as seen for the link prediction.

When comparing the methods that update the embeddings from an edge stream, online graph factorization (GF), StreamWalk (SW), and SecondOrder (SO), we observe that they

Table 6 Link prediction results with best parameter settings for each algorithm: *batch:* Graph Factorization (GF), Node2Vec, DeepWalk (DW), *online:* degree, online Graph Factorization (GF), and the best of StreamWalk (SW) and SecondOrder (SO)

data	degree	GF	DW	Node2Vec	online GF	SW/SO
enron	0.0212	0.0381	0.0259	0.0279	0.0943	**0.0955**
linux	0.0731	0.0469	0.0482	0.0482	**0.1163**	0.0997
facebook	0.0092	0.0094	0.0201	0.0201	0.0109	**0.0298**

Best performing methods are marked boldface

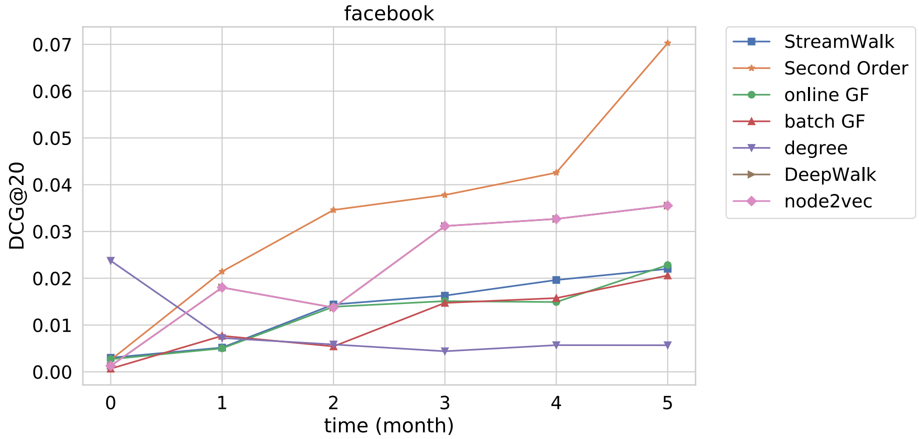


Fig. 15 Comparison of the batch and online methods for the link prediction problem on the Facebook data set

perform very similarly and their relative order depends on the data sets. Also note that GF is a special case of SW with walks of a length of one.

Conclusion

We introduced two online machine learning algorithms to extract temporal node representations from graph streams. The StreamWalk algorithm optimizes for the similarity of node pairs extracted along temporal walks from the data stream, whereas online second order similarity efficiently learns neighborhood similarity over graph streams by MinHash fingerprinting.

We measured the quality of these models in two tasks. In the RG17 and UO17 Twitter collections, we analyzed the similarity of node representations over time for both online and static node embedding algorithms. Our methods SW and SO significantly outperformed static Node2Vec, LINE, and simple degree related baselines. The combination of SW and SO achieved superior performance in the supervised evaluation task that we implemented using daily changing node relevance labels.

In a second experiment, we addressed the temporal link prediction task in three commonly used network data sets. We observed that online learning methods are superior to the snapshot-based batch algorithms. For some graphs, our walk-based embedding methods performed better than online matrix factorization.

Abbreviations

DCG: Discounted cumulative gain; DW: DeepWalk (Perozzi et al. 2014); GF: Graph factorization (Ahmed et al. 2013); NDCG: Normalized discounted cumulative gain; RG17: Our twitter data set about Roland-Garros 2017, the French Open Tennis Tournament; SO: online second order similarity, our model based on temporal node neighborhoods; SW: StreamWalk, our proposed model based on temporal walks; SW+SO: The combination of StreamWalk and online second order similarity; TREC: Text retrieval conference; UO17: Our twitter data set about US Open 2017, the United States Open Tennis Tournament

Acknowledgements

See Fundings section.

Authors' contributions

FB implemented online second order similarity and measured all algorithms for the similarity search experiments. DMK implemented online Word2Vec and contributed to link prediction measurements. RP implemented StreamWalk and contributed to algorithm descriptions. AB contributed with ideas for the algorithms and provided funding. All authors read and approved the final manuscript.

Funding

Support from Project 2018-1.2.1-NKP-00008: Exploring the Mathematical Foundations of Artificial Intelligence and the "Big Data—Momentum" grant of the Hungarian Academy of Sciences.

Availability of data and materials

Our data sets and software code are publicly available in the GitHub repository https://github.com/ferencberes/online-node2vec.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Institute for Computer Science and Control, Hungarian Academy of Sciences, (MTA SZTAKI) Kende Street 13-17, H1111 Budapest, Hungary. [2]Eötvös University Budapest Pázmány s. 1, H-1117 Budapest, Hungary. [3]Széchenyi University, Gyõr Egyetem tér 1, H-9026 Gyõr, Hungary. [4]Department of Computer Science, Stanford University, 353 Serra Mall Stanford CA 94305 USA.

Received: 17 April 2019 Accepted: 8 July 2019
Published online: 23 August 2019

References

Aggarwal C, Subbian K (2014) Evolutionary network analysis: A survey. ACM Comput Surv (CSUR) 47(1):10

Ahmed A, Shervashidze N, Narayanamurthy S, Josifovski V, Smola AJ (2013) Distributed large-scale natural graph factorization. In: Proceedings of the 22nd International Conference on World Wide Web. ACM, Geneva. pp 37–48

Al-Maskari A, Sanderson M, Clough P (2007) The relationship between IR effectiveness measures and user satisfaction. In: Proc. SIGIR. ACM, New York. pp 773–774

Belkin M, Niyogi P (2002) Laplacian eigenmaps and spectral techniques for embedding and clustering. In: NIPS. Neural Information Processing Systems, San Diego. pp 585–591

Béres F, Pálovics R, Oláh A, Benczúr AA (2018) Temporal walk based centrality metric for graph streams. Appl Netw Sci 3(1):32

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) Moa: Massive online analysis. J Mach Learn Res 11(May):1601–1604

Broder AZ, Charikar M, Frieze AM, Mitzenmacher M (2000) Min-wise independent permutations. J Comput Syst Sci 60(3):630–659

Cao S, Lu W, Xu Q (2015) Grarep: Learning graph representations with global structural information. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM, New York. pp 891–900

Clarke CL, Craswell N, Soboroff I (2004) Overview of the trec 2004 terabyte track. In: TREC. NIST 100 Bureau Drive, Gaithersburg Vol. 4. p 74

Dawid AP (1984) Present position and potential developments: Some personal views: Statistical theory: The prequential approach. J R Stat Soc Ser A Gen 147(2):278–292

De Francisci Morales G, Bifet A, Khan L, Gama J, Fan W (2016) Iot big data stream mining. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York. pp 2119–2120

Fogaras D, Rácz B (2005) Scaling link-based similarity search. In: Proceedings of the 14th World Wide Web Conference. International World Wide Web Conferences Steering Committee Republic and Canton of Geneva, Switzerland. pp 641–650

Frigó E, Pálovics R, Kelen D, Benczúr AA, Kocsis L (2017) Online ranking prediction in non-stationary environments. In: Proceedings of the 1st Workshop on Temporal Reasoning in Recommender Systems, Co-located with 11th International Conference on Recommender Systems. CEUR Workshop Proceedings Vol-1922

Gama J, Sebastião R, Rodrigues PP (2013) On evaluating stream learning algorithms. Mach Learn 90(3):317–346

Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of M achine Learning Research Vol. 9. pp 249-256

Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York. pp 855–864

Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: NIPS. pp 1024–1034

Hamilton WL, Ying R, Leskovec J (2017) Representation learning on graphs: Methods and applications. NIPS Neural Information Processing Systems, San Diego

Holme P, Saramäki J (2012) Temporal networks. Phys Rep 519(3):97–125

Juang C.-F., Lin C.-T. (1998) An online self-constructing neural fuzzy inference network and its applications. IEEE Trans Fuzzy Syst 6(1):12–32

Kaji N, Kobayashi H (2017) Incremental skip-gram model with negative sampling. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Copenhagen. pp 363–371

Kumar R, Novak J, Tomkins A (2010) Structure and evolution of online social networks. In: Link Mining: Models, Algorithms, and Applications. Springer, New York. pp 337–357

Kunegis J (2013) Konect: the koblenz network collection. In: Proceedings of the 22nd International Conference on World Wide Web. ACM, Geneva. pp 1343–1350

Lathia N, Hailes S, Capra L (2009) Temporal collaborative filtering with adaptive neighbourhoods. In: Proceedings of the 32nd International ACM SIGIR Conference. ACM, New York. pp 796–797

Liben-Nowell D, Kleinberg J (2007) The link-prediction problem for social networks. J Am Soc Inf Sci Technol 58(7):1019–1031

Ling G, Yang H, King I, Lyu MR (2012) Online learning for collaborative filtering. In: The 2012 International Joint Conference on Neural Networks (IJCNN). IEEE. pp 1–8
McGregor A (2014) Graph stream algorithms: a survey. ACM SIGMOD Rec 43(1):9–20
Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv preprint. arXiv:1301.3781
Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: NIPS. Neural Information Processing Systems, San Diego. pp 3111–3119
Moody J (2002) The importance of relationship timing for diffusion. Soc Forces 81(1):25–56
Nguyen GH, Lee JB, Rossi RA, Ahmed NK, Koh E, Kim S (2018) Continuous-time dynamic network embeddings. In: Companion Proceedings of the The Web Conference 2018. International World Wide Web Conferences Steering Committee, Geneva. pp 969-976
Nie F, Zhu W, Li X (2017) Unsupervised large graph embedding. In: Thirty-first AAAI Conference on Artificial Intelligence. AAAI Press, Palo Alto
Ou M, Cui P, Pei J, Zhang Z, Zhu W (2016) Asymmetric transitivity preserving graph embedding. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York. pp 1105–1114
Pálovics R, Benczúr AA, Kocsis L, Kiss T, Frigó E (2014) Exploiting temporal influence in online recommendation. In: Proceedings of the 8th ACM Conference on Recommender Systems. ACM, New York. pp 273–280
Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York. pp 701–710
Press O, Wolf L (2016) Using the output embedding to improve language models. CoRR abs/1608.05859:157–163
Qiu J, Dong Y, Ma H, Li J, Wang K, Tang J (2018) Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. ACM, New York. pp 459–467
Rozenshtein P, Gionis A (2016) Temporal pagerank. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer International Publishing Switzerland, Cham. pp 674–689
Sarma AD, Gollapudi S, Panigrahy R (2011) Estimating pagerank on graph streams. J ACM JACM 58(3):13
Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: Large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, Geneva. pp 1067–1077
Wang P, Qian Y, Soong FK, He L, Zhao H (2015) A unified tagging solution: Bidirectional lstm recurrent neural network with word embedding. arXiv preprint. arXiv:1511.00215
Wang X, Cui P, Wang J, Pei J, Zhu W, Yang S (2017) Community preserving network embedding. In: Thirty-First AAAI Conference on Artificial Intelligence. AAAI Press, Palo Alto
Wei X, Xu L, Cao B, Yu PS (2017) Cross view link prediction by learning noise-resilient representation consensus. In: Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, Geneva. pp 1611–1619
Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23(1):69–101
Yu Y, Yao H, Wang H, Tang X, Li Z (2018) Representation learning for large-scale dynamic networks. In: International Conference on Database Systems for Advanced Applications. Springer, Cham. pp 526–541
Zhou C, Liu Y, Liu X, Liu Z, Gao J (2017) Scalable graph embedding for asymmetric proximity. In: Thirty-First AAAI Conference on Artificial Intelligence. AAAI Press, Palo Alto
Zhu Y, Shasha D (2002) Statstream: Statistical monitoring of thousands of data streams in real time. In: Proceedings of the 28th International Conference on Very Large Data Bases. VLDB Endowment, Franklin. pp 358–369
Žliobaite I, Bifet A, Gaber M, Gabrys B, Gama J, Minku L, Musial K (2012) Next challenges for adaptive learning systems. ACM SIGKDD Explor Newsl 14(1):48–55
Zuo Y, Liu G, Lin H, Guo J, Hu X, Wu J (2018) Embedding temporal network via neighborhood formation. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, New York. pp 2857–2866

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.