BME Matematikai Modellalkotás Szeminárium

Embedding temporal networks into vector spaces

András Benczúr, Ferenc Béres,
Domokos Kelen, Róbert Pálovics

SZTAKI

Stanford University

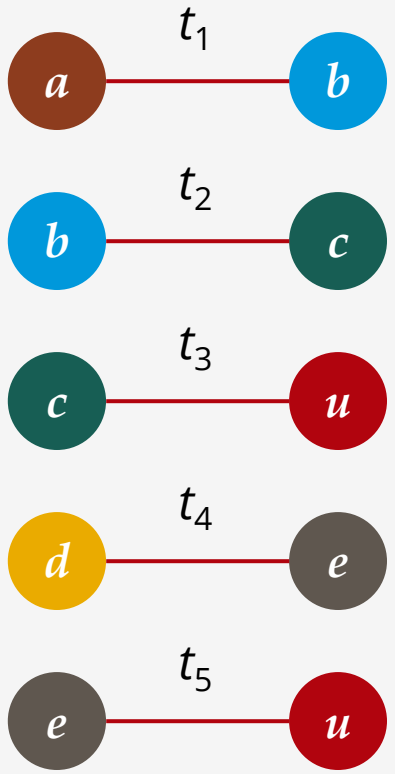
Main Resource

https://github.com/ferencberes/DEBS-graph-stream-tutorial

DEBS-graph-stream-tutorial

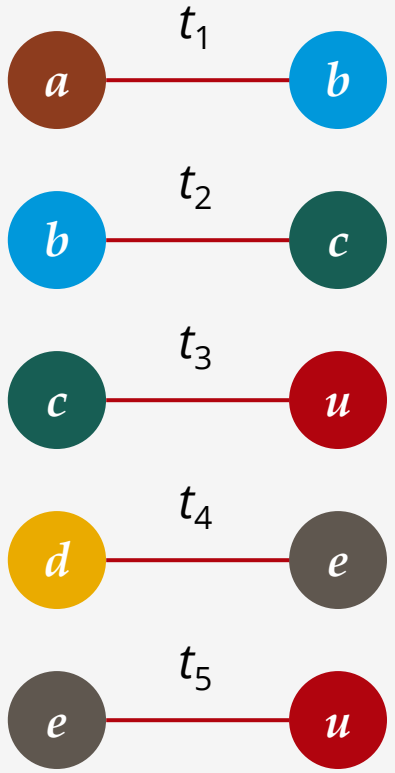
Temporal networks

- Network changes over time
- Edge stream
  - time series of edges: each link has a timestamp
  - edges may occur several times
  - example: Twitter mention network



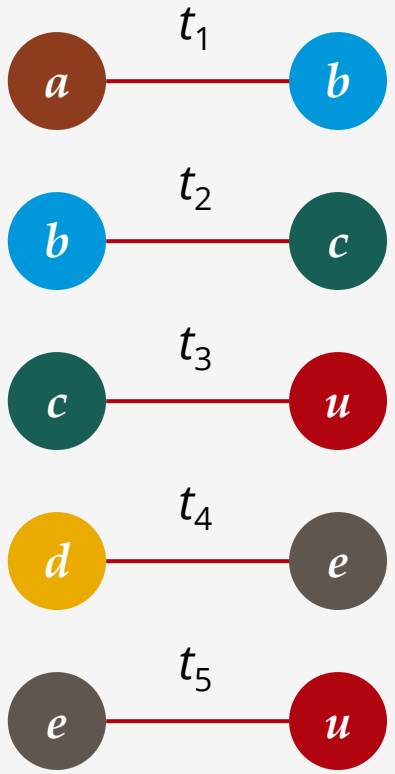
Temporal networks

- First results consider network snapshots, long term changes:
  - Kumar et al. 2010, Structure and evolution of online social networks: Flickr, Yahoo! community changes in time
  - Rosvall, Bergstrom. 2010. Mapping change in large networks: scientific citation structure change in a decade
  - Sun, Faloutsos, Papadimitriou, Yu. GraphScope. KDD07: graph streams are mentioned but consider 50-1000 incremental snapshots in different graphs
- The real temporal model:
  - McGregor. Graph stream algorithms: a survey. SIGMOD 2014
  - Streaming and semi-streaming algorithms for connectivity, sparsification, spanners, trees, matchings



Temporal network application results

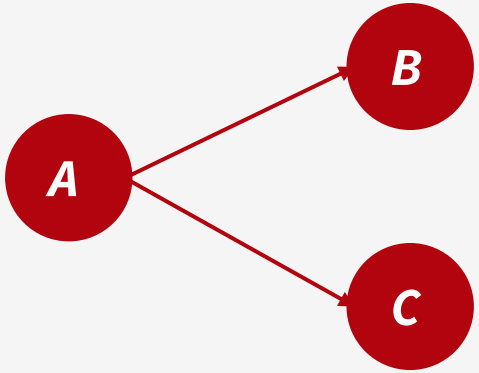
- Social interactions as temporal networks
  - “retweets” on Twitter with corresponding hashtags
  - track popularity of a political party during the election period to estimate the popularity [Aragón et al. Communication dynamics in twitter during political campaigns: The case of the 2011 Spanish national election. Policy & Internet 2013]
  - mobile communication [Aledavood et al., 2015]
- Network of cryptocurrency transactions can be analyzed for assessing anonymity [Béres, Seres, B 2020, 2021]
  - Assessing Ethereum mixer anonymity
  - Tracking Bitcoin transactions in the blockchain
  - Lightning Network (Bitcoin overlay) analysis



Example: Twitter mention stream

- Edge stream: Twitter @-mentions

  A: “This is a joint work with @B and @C”



- Streaming framework: producer -> socket -> consumer

DEBS-graph-stream-tutorial/graph_stream

Twitter mention stream - producer

DEBS-graph-stream-tutorial/graph_stream

Define output channel: socket

Connect to the Twitter Streaming API +
Search query setup: English tweets with:
#BREAKING, #BREAKINGNEWS, #breakingnews

Run tweet search in real-time

```python
from twittercrawler.crawlers import TwythonStreamCrawler
from twittercrawler.data_io import SocketWriter
from twittercrawler.search import get_time_termination
import datetime, sys

if len(sys.argv) != 3:
    print("Usage: <twitter_api_key> <port>")
else:
    api_fp = sys.argv[1]
    port = int(sys.argv[2])

    # prepare writers
    sw = SocketWriter(port=port, export_filter="mention")

    # initialize crawler
    stream = TwythonStreamCrawler([sw], api_fp)

    # query
    search_params = {
        "q":'#BREAKING OR #BREAKINGNEWS OR #breakingnews',
        "lang":"en",
    }
    stream.set_search_arguments(search_args=search_params)

    # run stream search (pause for 0.5 second after each tweet)
    try:
        # YOU MUST TERMINATE THIS SEARCH MANUALLY!
        stream.search()
    except:
        raise
    finally:
        stream.close()
```

Twitter mention stream - consumer

Read from the socket

Extract mentions

Print edge stream

```python
from twittercrawler.data_io import SocketReader
import sys

def listen_for_news(port, hide_accounts=True):
    reader = SocketReader(port=port)
    try:
        for tweet in reader.read():
            ts = tweet["created_at"]
            tweet_id = tweet["id_str"]
            source = tweet["user"]["id_str"] if hide_accounts else tweet["user"]["screen_name"]
            if "entities" in tweet and "user_mentions" in tweet["entities"]:
                for mention_obj in tweet["entities"]["user_mentions"]:
                    target = mention_obj["id_str"] if hide_accounts else mention_obj["screen_name"]
                    record = {
                        "time":ts,
                        "tweet_id":tweet_id,
                        "source":source,
                        "target":target
                    }
                    print(record)
    except:
        raise
    finally:
        reader.close()
```

DEBS-graph-stream-tutorial/graph_stream

Algorithms on Data Streams

1. Limitations of the data steam model. Limited memory.
2. Certain evaluation methods fail since predictive models can change during evaluation.
3. Concept Drift can occur (not in this talk).
4. Scalability, distributed processing (not in this talk).

Appeared in 2019
All in one ArXiv: 1802.05872

Issue 1: Algorithmic limitations

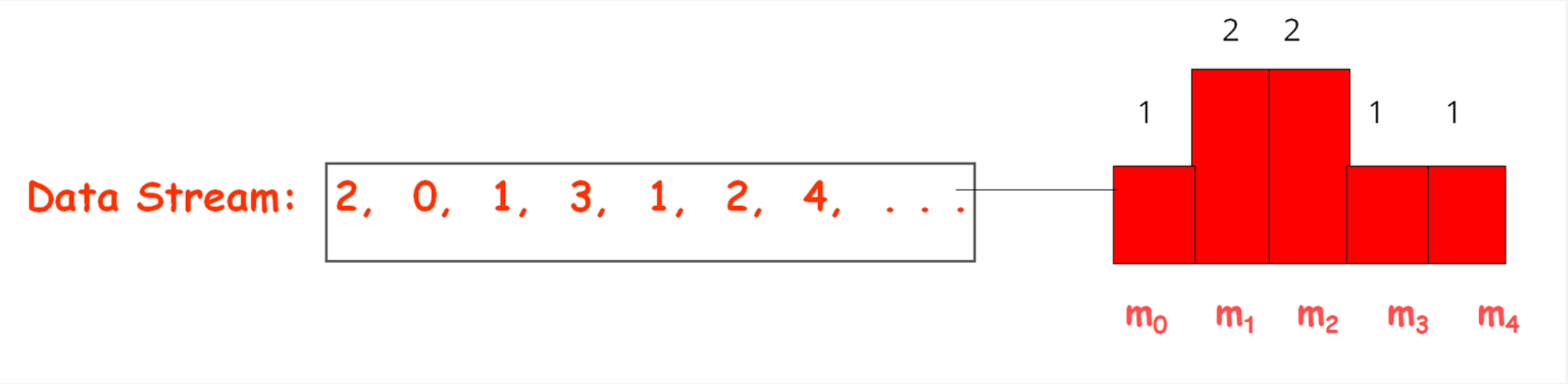
Example: Machine Learning in the data streaming computational model

- No access to (majority of) past data
  - No stochastic gradient descent: we cannot iterate over the data
    - Online gradient descent is possible [Palovics, B 2014]
  - No decision trees: after deciding about a split, we cannot use data inside the new nodes for a next split
    - If confident about a split seeing some data, build further splits by the new data
    - Use concept drift statistics to rebuild certain branches

      [Kourtellis, Morales, Bifet, Murdopo, Vertical Hoeffding tree. Big Data 2016]

      [Vasiloudis, Beligianni, Morales. BoostVHT, CIKM 2017.]

Illustration of the computational models: Histogram



In-Memory

- Hash tables

On disk

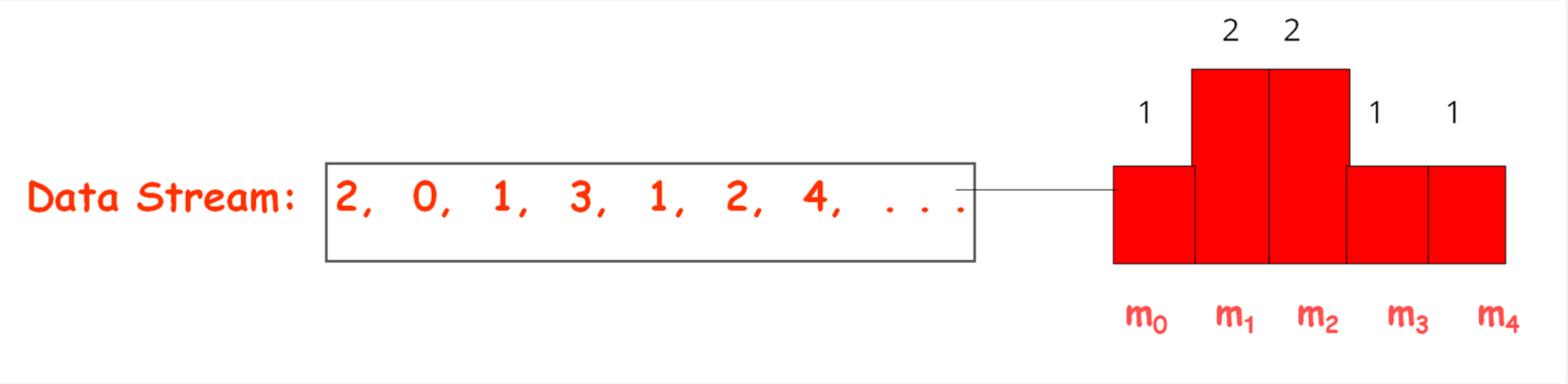
- Sort (mergesort)

Distributed

- Map-reduce – basically sorting again

Stream?

Number of different elements in data stream



- No exact algorithm without storing all data – Proof:
  - Trivial information bound to decide if the next element is new or already present earlier in the stream
  - We may reconstruct the entire past stream as a set by probing with next elements
- Random sampling fails very bad on rare elements
  - Assume sample consist of identical elements only
  - A likely answer is that there are no other elements
  - But we may have, with large probability, a large number of other elements that appear only once
  - Numerical example: 20% random sample, we can construct data stream where relative error on count > 20%
- Distinct sampling: read all data, make adaptive decisions

Muthukrishnan, S., et al.: Data streams: Algorithms and applications. Foundations and Trends in Theoretical Computer Science1(2), 117–236 (2005)

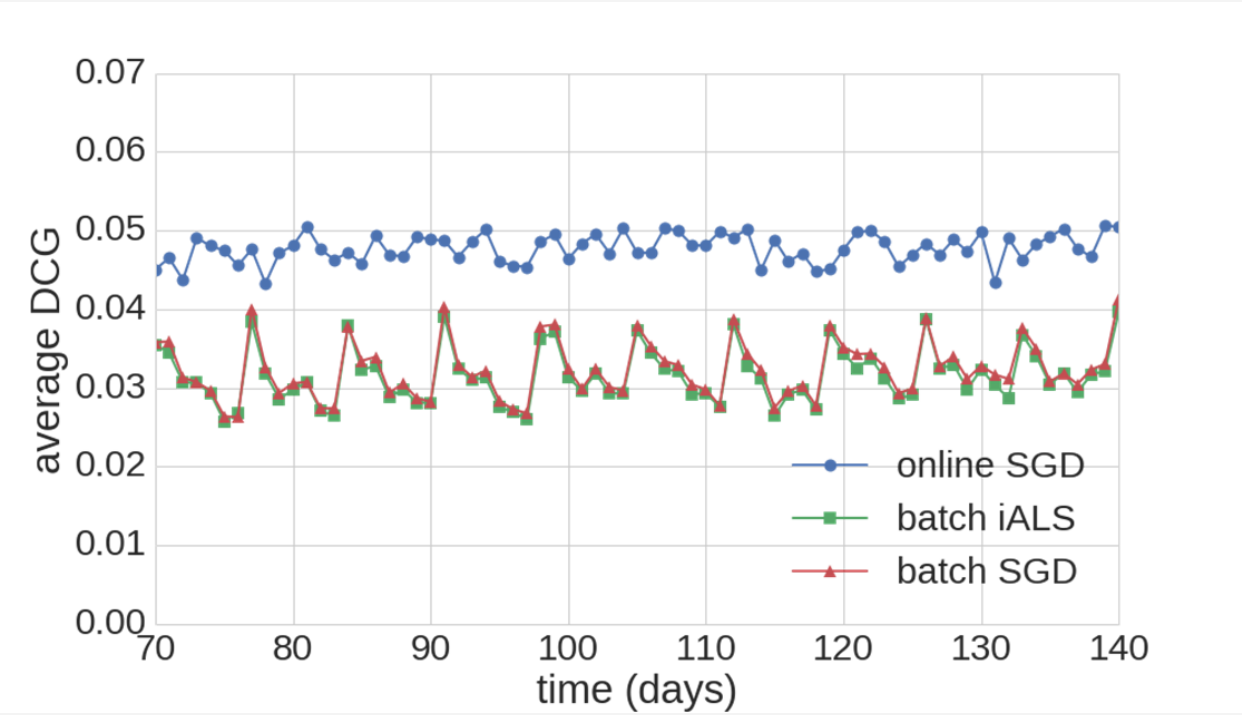
Issue 2: Difficulty in ML evaluation

- Model changes right after prediction is made
  - Precision, Recall and many other metrics compare a SET of predictions against the ground truth
  - But for the next prediction, a new model may potentially give completely new results
- A natural streaming evaluation metric is clickthrough rate
  - Equivalent of the “Precision” of a single item
- AUC for classification is also a problem
- Prequential (predictive sequential) evaluation
  1. Give a prediction for the next data point
  2. Read its label, compare to the prediction and update quality metrics
  3. Update the model to be used for the next data point
- Slightly modified metrics are needed

Dawid, A. Philip. "Present position and potential developments: Some personal views - statistical theory - the prequential approach." *Journal of the Royal Statistical Society: Series A (General)* 147.2 (**1984**): 278-290.

Issue 3: Concept drift - when online algorithms can be better

- Model performance very often deteriorates in time
- Observe the weekly retraining periods in performance on the right
- Concept drift detection either
  - Detects sharp changes in distribution, e.g. failures, or
  - Measures deterioration to schedule retraining
- Auto-forgetting old events can be an option
  - Gradient descent with negative sample generation



Last.fm “30M” Music listening dataset crawled by the CrowdRec team

Temporal Walks

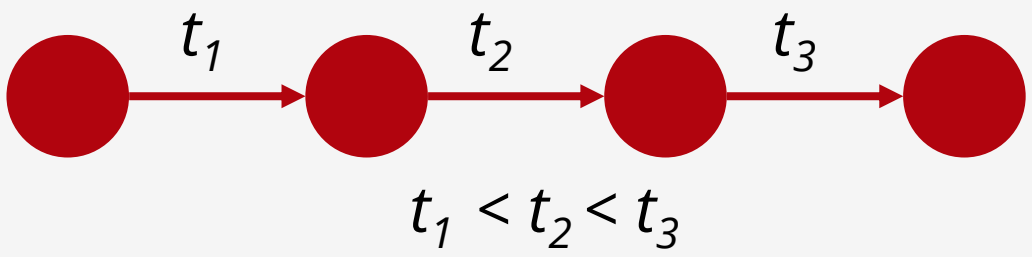
Model the information flow in the network
Use for embedding in vector space and defining centrality

Example: temporal degree

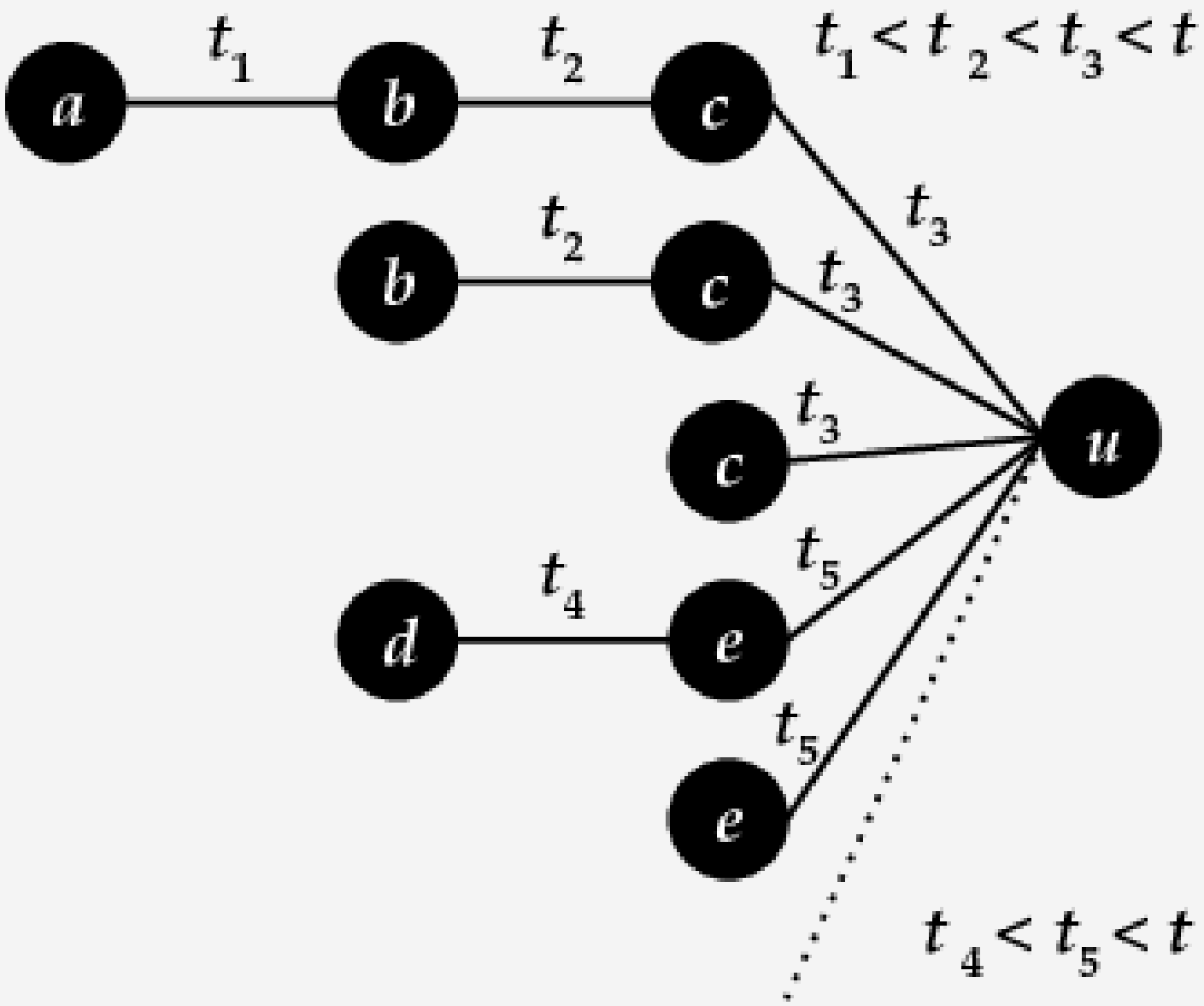
- Add time decay to the degree
- Exponential decay
  - $\varphi(\tau) = \beta \exp\{-c\,\tau\}$ as the function of the age τ
    - Diminishes fast, we can keep only recent edges
  - Since $\varphi(a) \cdot \varphi(b) = \varphi(a + b)$:
    - Easy to update as time elapses
    - Can even give weight to lists of edges (walks)
- Performs well for several tasks, explicitly mentioned in [Kim, Anderson 2012]

Time respective paths

- Adjacent edges that are ordered in time
- Models a flow, e.g.
  - information flow in social networks
  - flow of funds or goods in the economy
- Concept
  - delay t_2-t_1 is small, then flow is more likely
  - No flow in the opposite direction, from a future to a past edge



Example: temporal walks ending at node u before time t



Give weight to walks: recall the temporal degree

- For a temporal walk with edges at times (t1,t2,...,tk) and decay φ, weight is

$$\Phi(z,t) := \prod_{i=1}^{j} \varphi(t_{i+1} - t_i)$$

- Exponential decay
  - $\varphi(\tau) = \beta \exp\{-c\,\tau\}$ as the function of the age τ
    - Diminishes fast, we can keep only recent edges
  - Since $\varphi(a) \cdot \varphi(b) = \varphi(a + b)$:
    - Easy to update as time elapses
    - Easy to compute weight to lists of edges (walks)
      - $\varphi(tj - t1)$ in the example

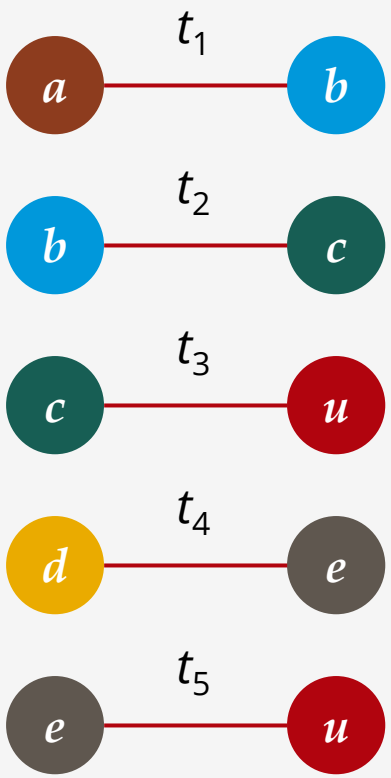
Use of time respecting walks

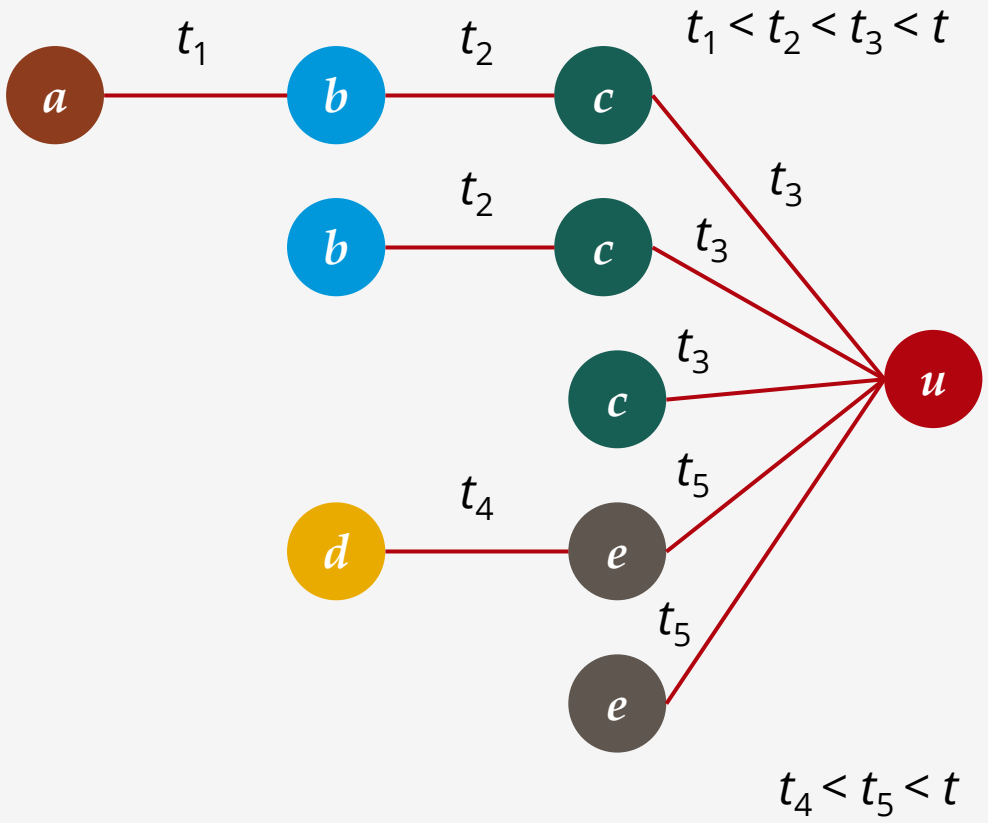
- Walk based network embedding with temporal walks, e.g. [Grover, Leskovec 2016, Node2Vec]
- Walk-based centrality
  - First result [Rozenshtein, Gionis. "Temporal pagerank." ECML 2016.]
  - Our result based on Katz centrality [Katz (1953) A new status index derived from sociometric analysis]

Network centrality metrics

Temporal Katz Centrality

Definition: weighted sum of all time respecting walks that end in node *u*





Temporal Katz Centrality

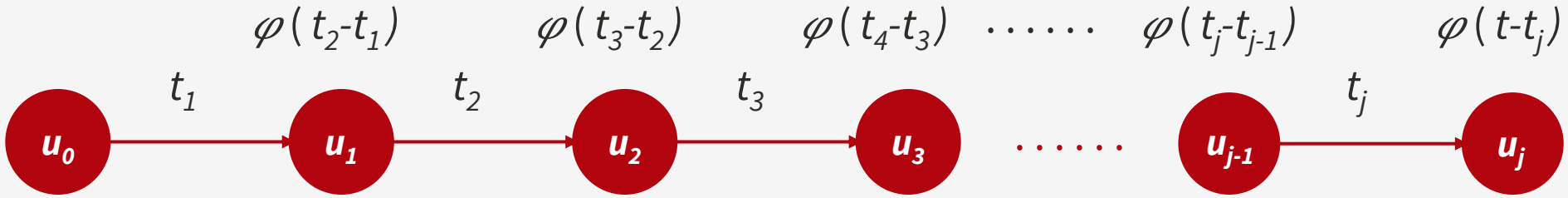
- Centrality for node *u* at time *t*

$$r_u(t) := \sum_{v} \sum_{\substack{\text{temporal paths } z \\ \text{from } v \text{ to } u}} \Phi(z,t)$$

- where $\Phi(z,t)$ is the weight of a single path

$$\Phi(z,t) := \prod_{i=1}^{j} \varphi(t_{i+1} - t_i)$$

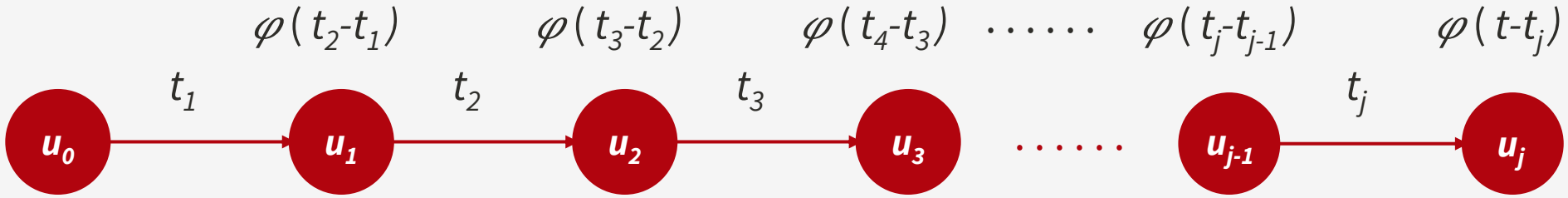
- where edges appeared at (t_1 , t_2 , . . . , t_j) for walk *z*



Weighting functions

- Constant $\beta < 1$
  - $\varphi(\tau) = \beta$
  - walk length penalized with β
  - $\Phi(z,t) = \beta^{|z|}$
- Exponential decay
  - $\varphi(\tau) = \beta \exp\{-c\,\tau\}$
  - as $\varphi(a) \cdot \varphi(b) = \varphi(a + b)$, for an arbitrary path

$$\Phi(z,t) = \beta \exp\left(-c\left[t - t_j\right]\right) \ldots \beta \exp\left(-c\left[t_2 - t_1\right]\right) = \beta^{|z|} \exp\left(-c\left[t - t_1\right]\right)$$



Relation to Katz Centrality

- Katz centrality

$$\vec{\text{Katz}} = \mathbf{1} \cdot \sum_{k=0}^{\infty} \beta^k A^k,$$

$$\vec{\text{Katz}}(u) := \sum_{v} \sum_{k=0}^{\infty} \beta^k |\{\text{paths of length } k \text{ from } v \text{ to } u\}|$$

- Theorem. Given an underlying graph with edge set of size *E*, if we sample *T* edges uniform at random, the expected value of temporal Katz centrality is the Katz centrality

Expected value - $\varphi := \beta$

- Not true if $\varphi := \beta$

The expected number of times the edges of a *given* path of length *k* appear in a given order:

$$s_{T,k} = \binom{T}{k} \cdot E^{-k}$$

since a given edge has a probability of 1/E to appear at a given position

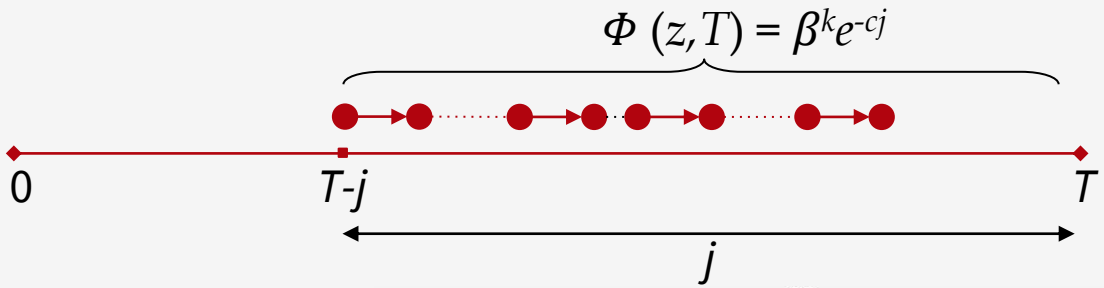
$$\vec{\text{TemporalKatz}} = \mathbf{1} \cdot \sum_{k=0}^{K} \beta^k A^k \binom{T}{k} \cdot E^{-k} \simeq \mathbf{1} \cdot \sum_{k=0}^{K} \beta^k A^k (T/E)^k / k!$$

Expected value - exponential decay

$$\vec{\text{TemporalKatz}} = \lim_{T\to\infty} \mathbf{1}\cdot\sum_{k=0}^{K} A^k s_{T,k} = \mathbf{1}\cdot\sum_{k=0}^{K} A^k \lim_{T\to\infty} s_{T,k}$$

- $s_{T,k}$: the expected total weight of a *given* path of length k
- Each occurrence of a path of length k starting at time $(T-j)$ has the weight $\beta^k \exp(-cj)$

$$s_{T,k} = \beta^k \frac{1}{E^k} \sum_{j=k}^{T} \binom{j-1}{k-1} e^{-cj}$$



Since $\sum_{n=m}^{\infty} \binom{n}{m} x^n = x^m/(1-x)^{m+1}$,

$$\lim_{T\to\infty} s_{T,k} = \lim_{T\to\infty} \left(\frac{\beta}{E}\right)^k \sum_{j=k}^{T} \binom{j-1}{k-1} e^{-cj}$$

$$= \left(\frac{\beta}{E}\right)^k e^{-c} \sum_{j=k}^{\infty} \binom{j-1}{k-1} e^{-c(j-1)}$$

$$= \left(\frac{\beta}{E}\right)^k \frac{e^{-ck}}{(1-e^{-c})^k} = \left(\frac{\beta}{E}\right)^k \frac{1}{(e^c-1)^k}.$$

$$\vec{\text{TemporalKatz}} = \mathbf{1}\cdot\sum_{k=0}^{K} A^k \lim_{T\to\infty} s_{T,k} = \mathbf{1}\cdot\sum_{k=0}^{K} A^k \left(\frac{\beta}{E}\right)^k \left(\frac{1}{e^c-1}\right)^k$$

- let $c := c'/E$ with $c' \ll E$
- hence $c/E \ll 1$ and $\exp\{c\} = \exp\{c'/E\} \approx 1 + c'/E$

$$\vec{\text{TemporalKatz}} = \mathbf{1}\cdot\sum_{k=0}^{K} A^k \left(\frac{\beta}{E}\right)^k \left(\frac{1}{1+c'/E-1}\right)^k = \mathbf{1}\cdot\sum_{k=0}^{K} A^k \left(\frac{\beta}{c'}\right)^k$$

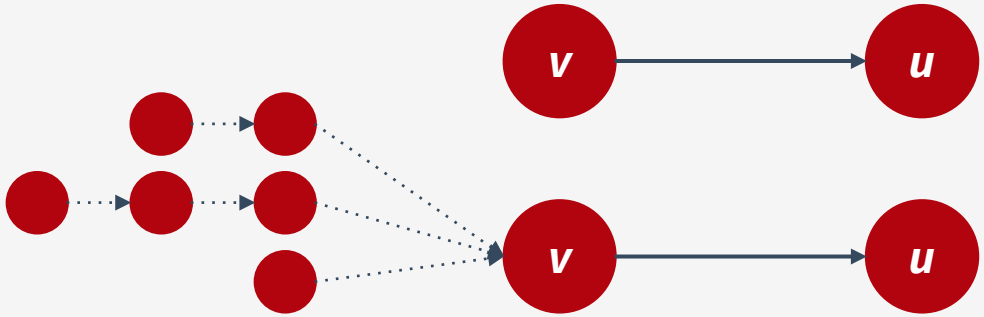
Temporal Katz converges to static Katz on uniformly sampled edge streams

Temporal Katz Centrality - computation

- When edge *vu* appears at time t_{vu}
- The centrality of node *u* at time *t* increases as
  - a new time respecting walk appears
  - all walks that ended in *v* continue via edge *vu* to *u*
- Hence the total increase is
  $(1 + r_v(t_{vu})) \cdot \varphi(t - t_{vu})$
- Recursive definition

$$r_u(t) = \sum_{vu \in E(t)} (1 + r_v(t_{vu}))\, \varphi(t - t_{vu})$$

- Note that $w_{vu} := r_v(t_{vu})$ does not depend on time!

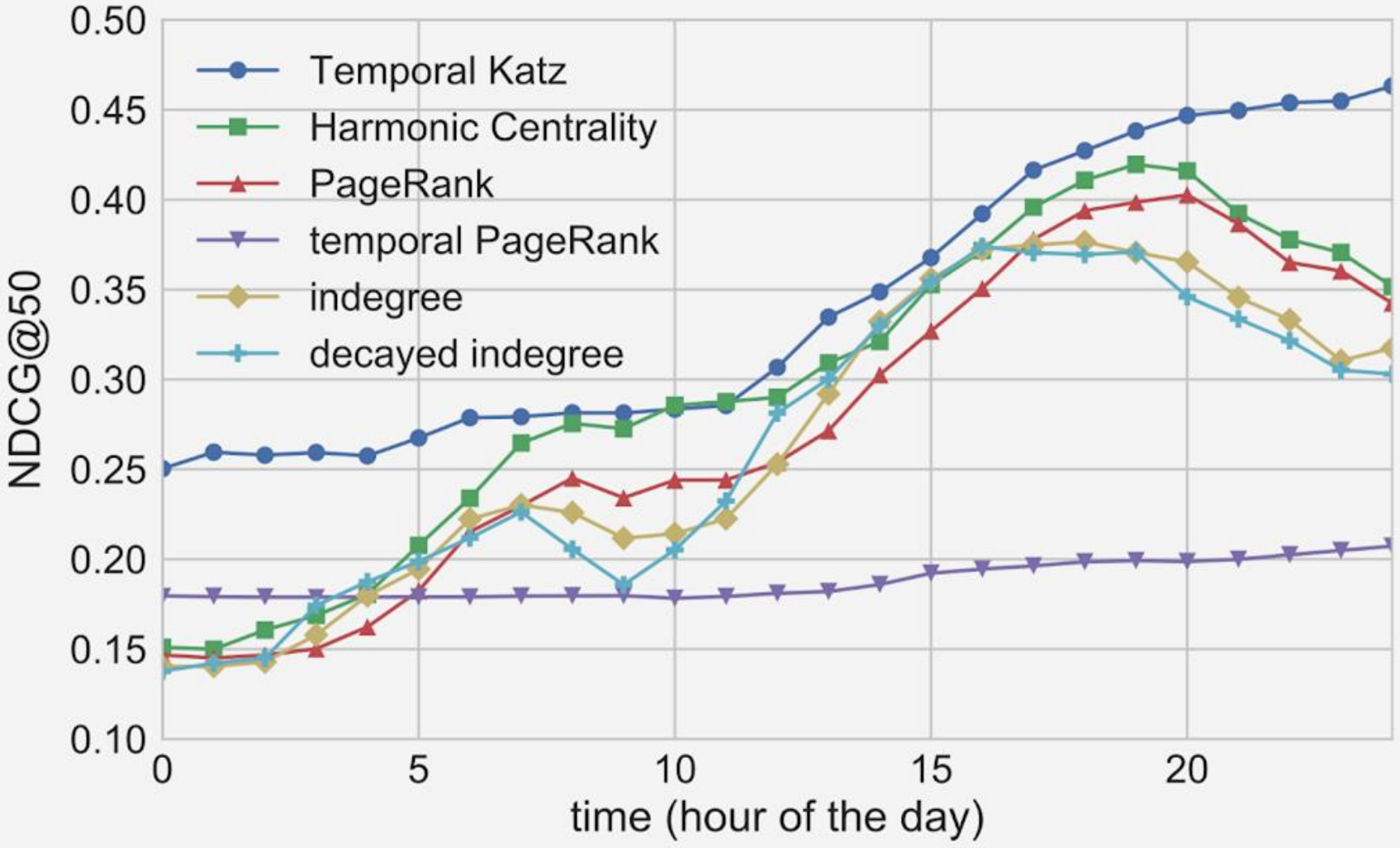


Experiments on Twitter data

- Centrality metrics are difficult to evaluate overall
- Current need: temporal network with temporal labels
- Data: Twitter mentions during a tennis tournament
  - nodes: Twitter accounts
  - edges: user mentions
  - labels:
    - players participating on a given day,
    - other tennis players

1	Roland-Garros	@rolandgarros	0
2	Stanislas Wawrinka	@stanwawrinka	1
3	Andy Murray	@andy_murray	1
4	Simona Halep	@Simona_Halep	0
5	Rafa Nadal	@RafaelNadal	1
6	Dominic Thiem	@ThiemDomi	1
7	Timea Bacsinszky	@TimeaOfficial	0
8	Rohan Bopanna	@rohanbopanna	0
9	Ana Ivanovic	@Analvanovic	0
10	WTA	@WTA	0
11	Gaby Dabrowski	@GabyDabrowski	0
12	Tennis Channel	@TennisChannel	0
13	Rafa Nadal Academy	@rnadalacademy	0
14	Karolina Pliskova	@KaPliskova	0
15	yonex.com	@yonex_com	0
16	Gusti Fernandez	@gustifernandez4	0
17	rolandgarrosFR	@rolandgarros_FR	0
18	Eurosport.es	@Eurosport_ES	0
19	ATP World Tour	@ATPWorldTour	0
20	Caroline Garcia	@CaroGarcia	0

Experiments on Twitter data



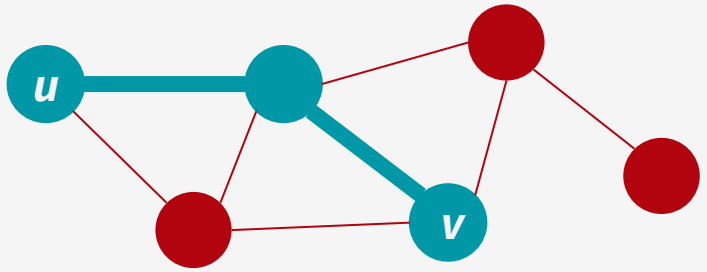
Network node embeddings

Network node embeddings

- Find embedding of nodes to *d* dimensions
- Similar nodes in the graph have embeddings that are close together
- Already existed, e.g. recSys MF, graph factorization
- Revisited: random walk based methods,
  e.g. DeepWalk, LINE, node2vec
- Simple graph factorization loss function

$$L = \Sigma_{uv}[\, p_u p_v - A_{uv}]^2$$

  - model parameters: p_u
  - optimization via SGD

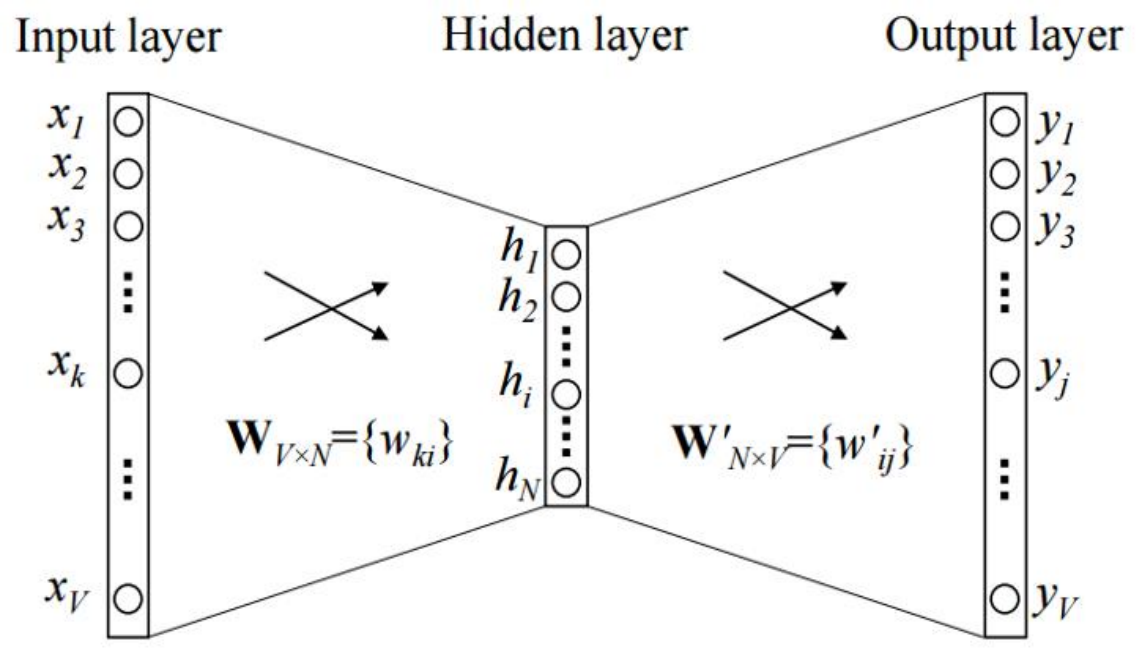


- Random walk based methods

$$L = \Sigma_{D\,(u,v)} - \log(\, \text{softmax}(p_u p_v\,)\,)$$

  - *D* is a set of generated random walks, e.g. *k* from each node

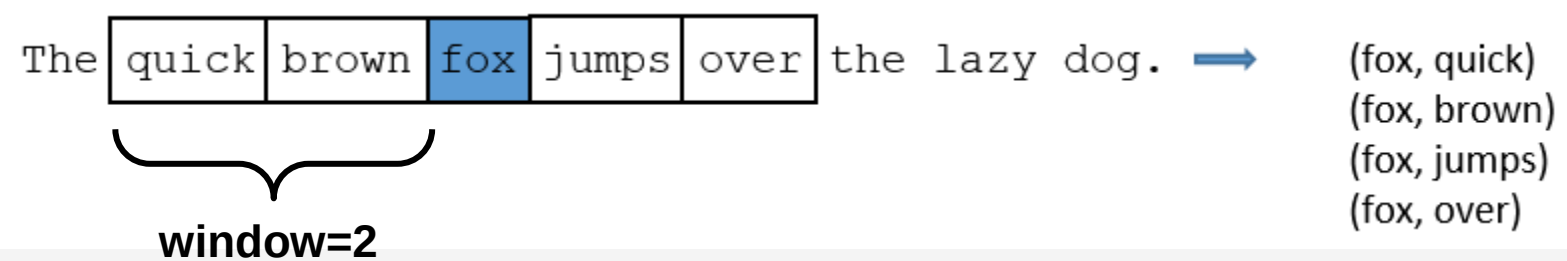
Word2Vec



Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems 2013.

V: Number of words in the corpus

N: Hidden layer neurons **==** Dimension of the embedding space

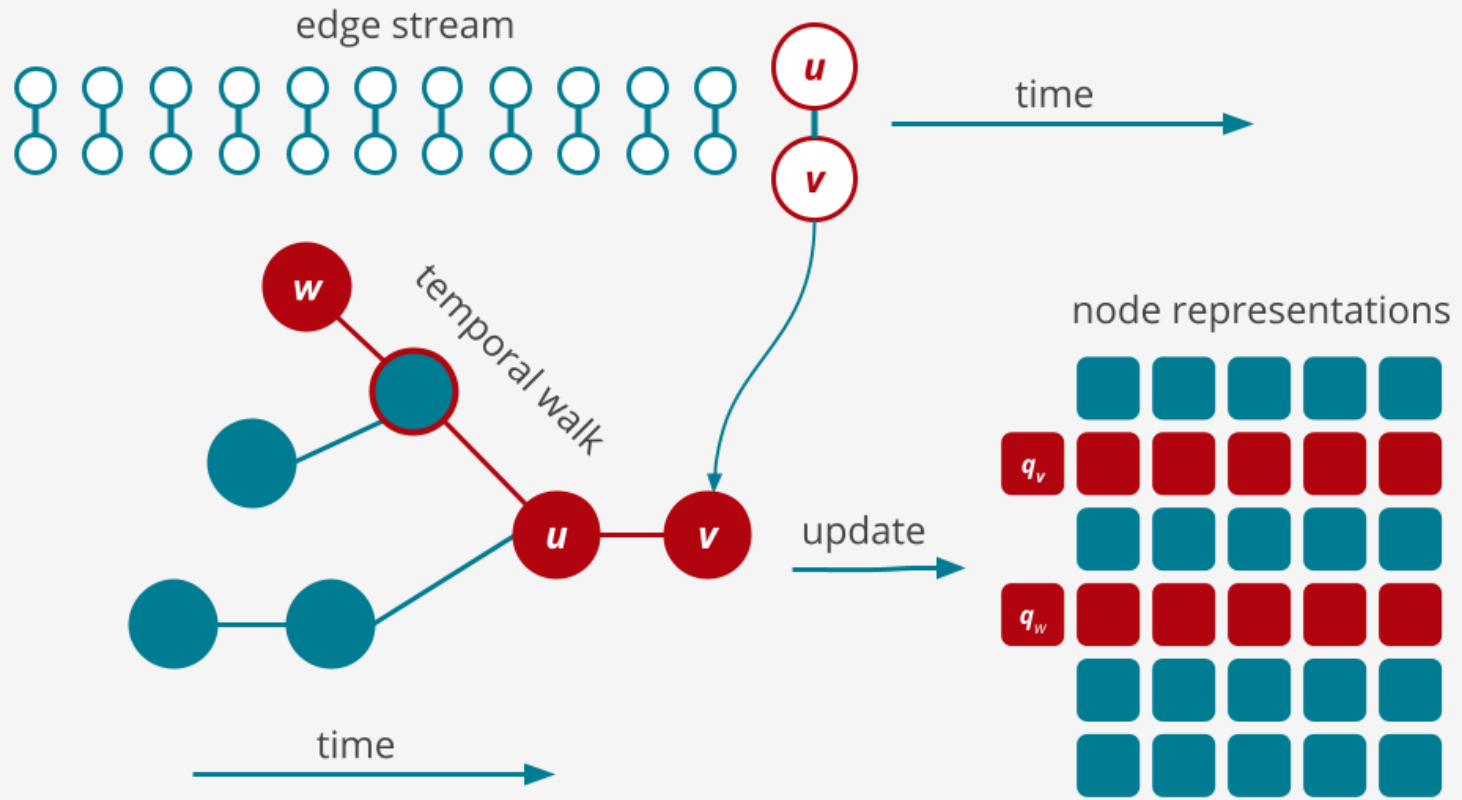


Static node embedding methods

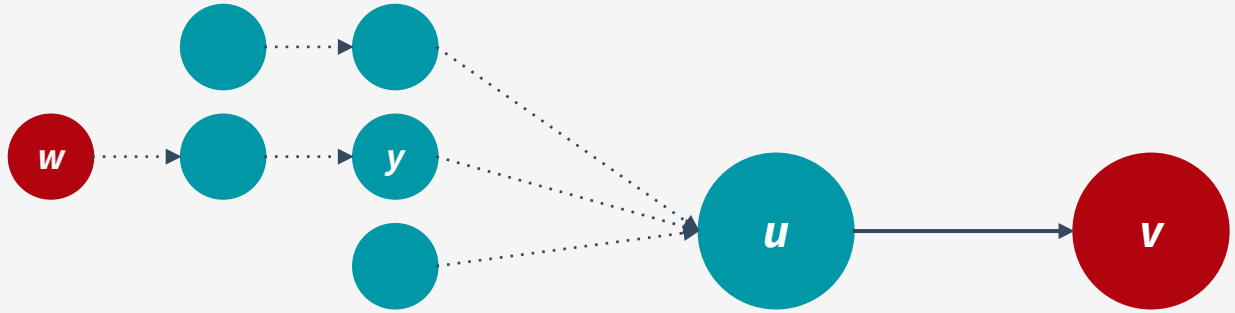
- DeepWalk: [Perozzi, Al-Rfou, Skiena. KDD 2014 Deepwalk: Online learning of social representations]
  - Corpus: nodes of the graph
  - Sentences: random walks
  - no control over the explored neighborhood
- Node2Vec: [Grover, Leskovec. KDD 2016: Scalable feature learning for networks]
  - Return parameter: p
  - In-out parameter: q
- LINE: [Tang, Qu, Wang, Zhang, Yan, Mei. WWW 2015 Line: Large-scale information network embedding]
  - first-order proximity: edge weights
  - second-order proximity: similarity of neighborhoods

Online node embeddings - objective

- Define online node embedding model which is
  - edge stream based
  - online updateable
  - generates temporal embeddings
  - adapts to concept drift



StreamWalk

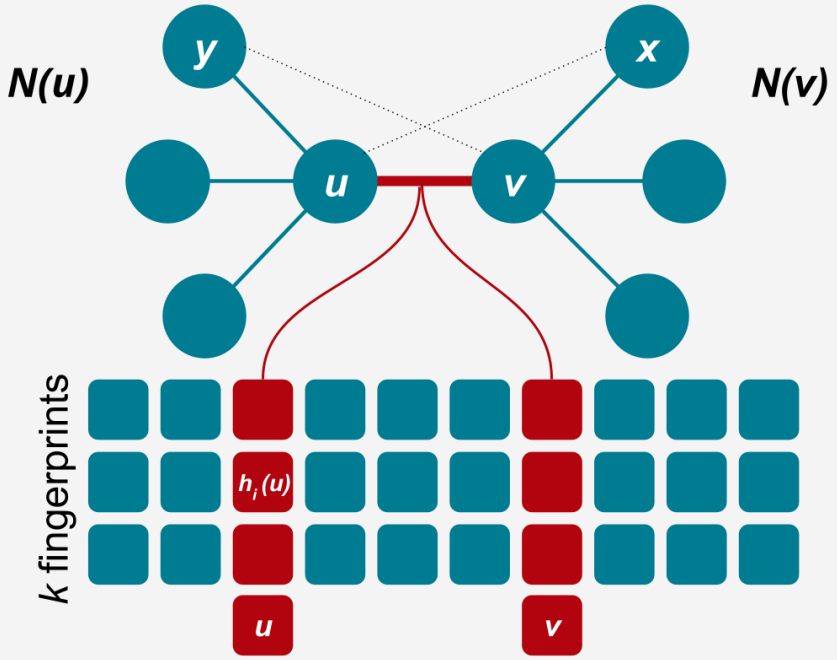


- Process edges in temporal order
- For each edge (u,v)
  - learn that p_u and p_v are similar
  - start random time-respective walks $(w,\ldots,u,v)$ to the past ending in (u,v)
  - learn for (w,v) that p_w and p_v are similar

- Since we can not store every temporal walk leading up to u, we define a recursive temporal walk sampling procedure:
  - update $p(v,t)$: weight of all t. walks ending at v at time t
  - sample a temporal walk ending at u (wrt. multi-edges) then continue recursively with y

Online second order similarity

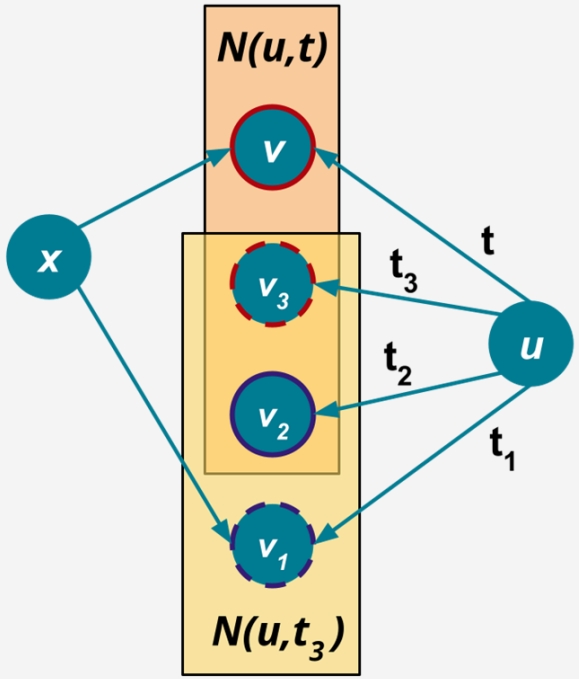
- Optimizes the embedding to match time-aware Jaccard similarity of the neighborhood of *u* and *x*
- Temporal neighborhood:
  - multi-set $N(u, t)$
  - $w(y) = \exp(-c(t - t(uy)))$ for edge *uy*
  - Emphasize the importance of new edges:
    discard $y \in N(u, t)$ with probability $1 - w(y)$
    when a new edge is added to *u*
- Time-aware similarity approximation:
  - maintain k MinHash fingerprints



Online second order similarity - fingerprints

- new edge *uv* arrives at time t
- Neighbors of *u* are ordered $t_1 < t_2 < t_3 < t$
- random permutations π_1 and π_2 define the fingerprints $h_1(u)$ and $h_2(u)$
- Dashed circles indicate MinHash fingerprints before the arrival of *uv* while the full colored circles are the new MinHash fingerprints.

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



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Blockchain is Watching You

Profiling and Deanonymizing Ethereum Users

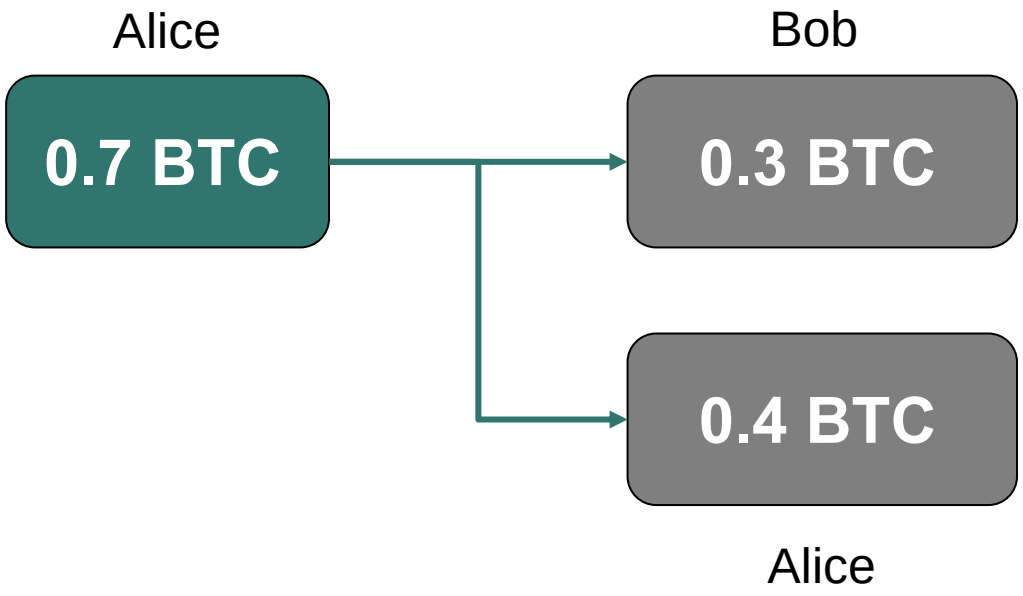
Ferenc Béres, István A. Seres, András A. Benczúr, Mikerah Quintyne-Collins

Our contributions

- In the cryptocurrency domain: first to quantitatively assess the performance of node embeddings
- We establish several heuristics to decrease the privacy guarantees of non-custodial mixers on Ethereum
- We collect and analyze a wide source of Ethereum related data

ferencberes/ethereum-twitter

UTXO vs Account model



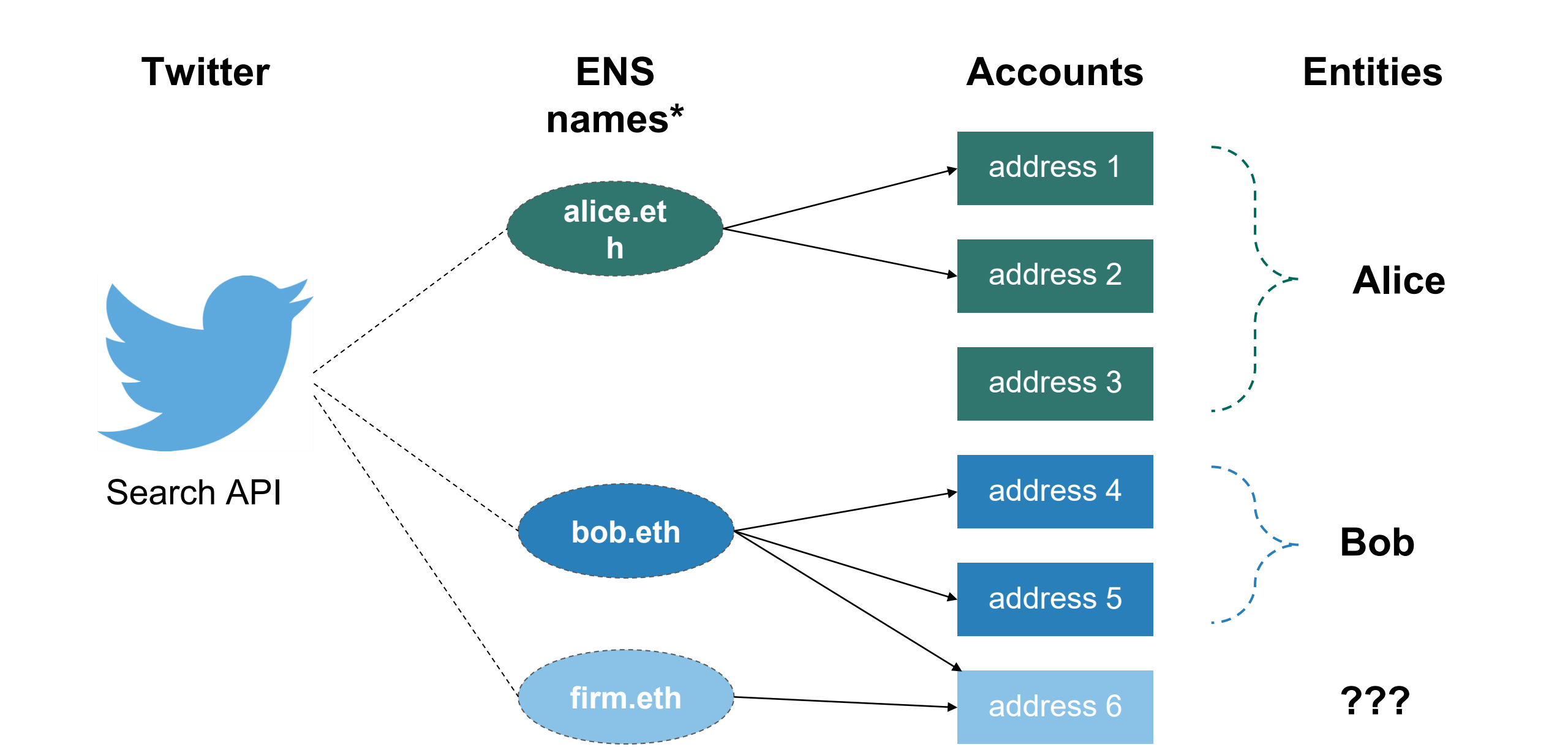
Timeline

Prominent Ethereum community members started to show their ENS names on Twitter

November 2019 — April 2020 — June 2020

Data collection from multiple sources (Twitter, Etherscan, Web3.py etc.)

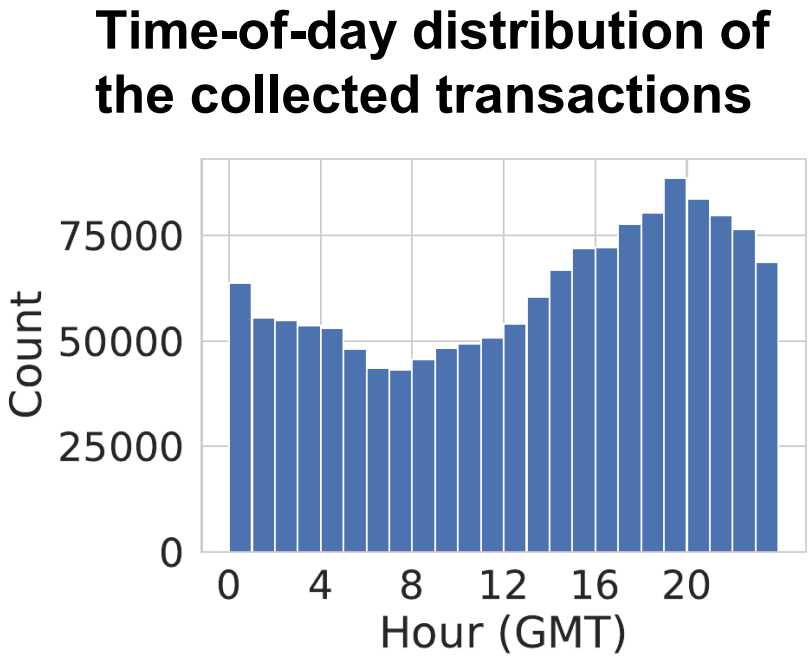
*ENS (Ethereum Name Service): mapping string (e.g. alice.eth) to long address hashes



*ENS (Ethereum Name Service): mapping string to long address hashes

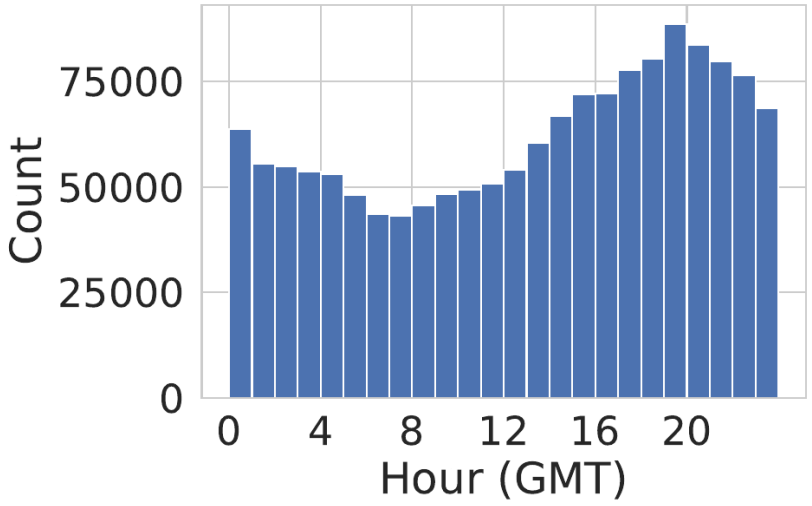
Quasi-identifiers of Ethereum accounts

Time of day activity



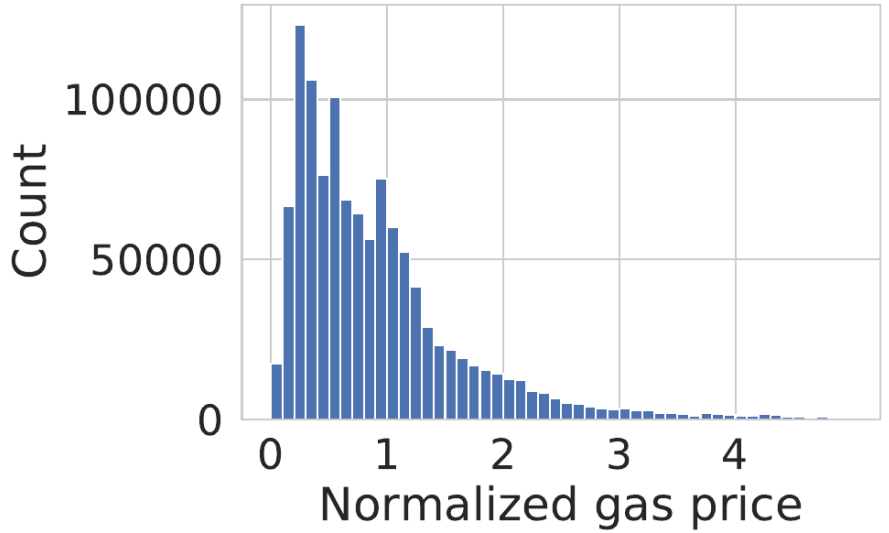
Time of day activity

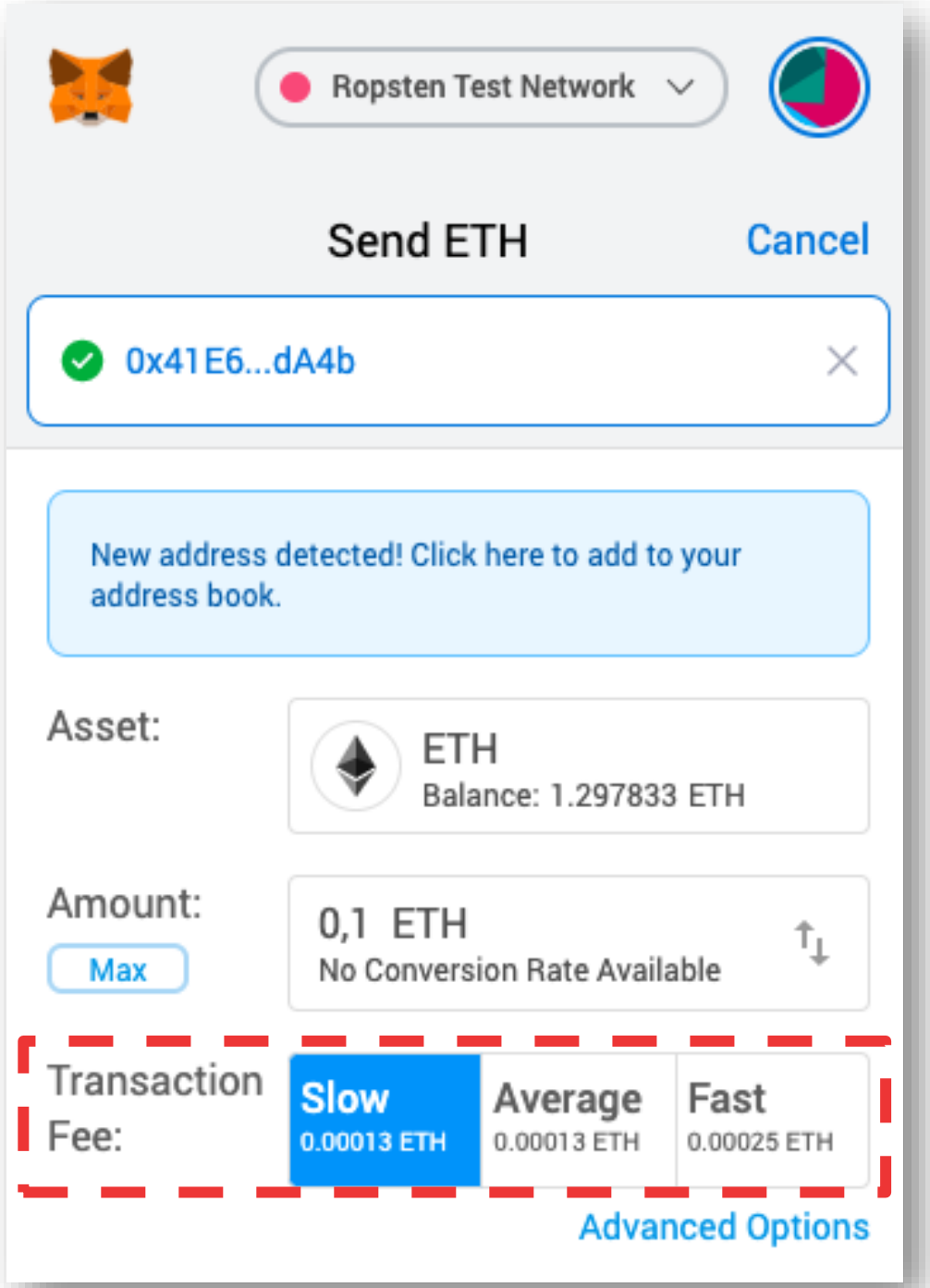
Time-of-day distribution of the collected transactions



Gas price selection

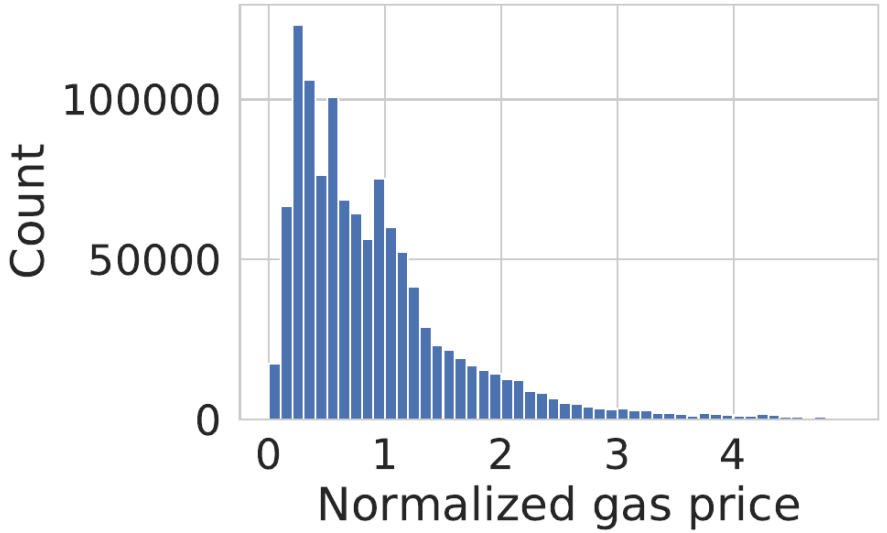
Normalized gas price distribution of the collected transactions





Gas price selection

Normalized gas price distribution of the collected transactions



Linking Ethereum accounts of the same entity

Overview

Address A and B belong to the same ENS name.

Task: For A try to find B (and also A for B) out of the total 3321 addresses

Representations: Map each address to a vector. Predict addresses based on distance:

- A -> [a_1, a_2, a_3, …, B, … a_N], (N=3320)

rank=?

Evaluation: is based on multiple A, B pairs

- 129 ENS names with exactly two addresses **(ground truth)**

Metrics:

- average rank
- entropy gain

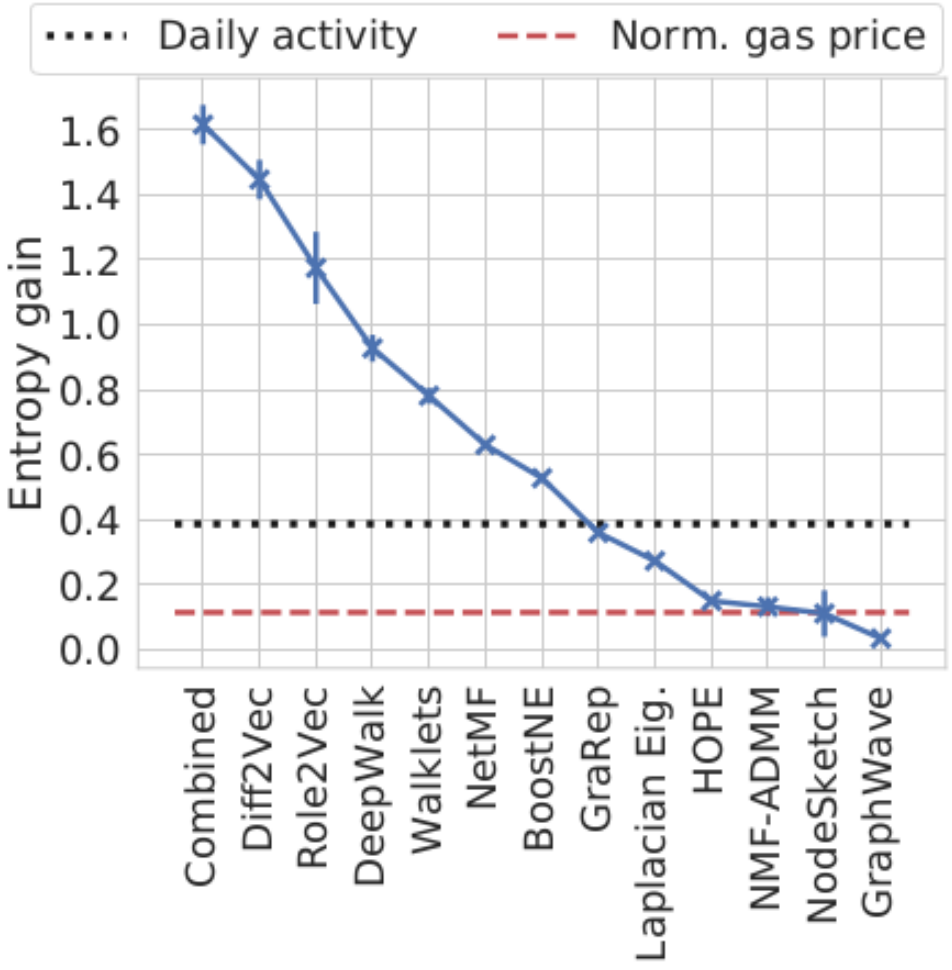
Linking performance analysis

Baselines: Daily activity and normalized gas price

Best model: combination of Diff2Vec and Role2Vec

Anonymity set size reduction: $2^{1,6} \approx 3.0314$

Interpretation: addresses of the same entity are usually in the same cluster (Diff2Vec) performing similar roles (Role2Vec)



Summary

- We surveyed results to analyze and model edge streams:
  - Interactions in a social network service
  - Graph algorithms with real-time stream updates
- Ingredients:
  - Data streaming computational model
  - Temporal networks
  - Prequential evaluation
- Algorithms that update the results while the edge stream is processed:
  - Link prediction
  - Network embedding, similarity search
  - Network centrality algorithms

Resources

Resource list: software

https://github.com/ferencberes/DEBS-graph-stream-tutorial

1. Twitter graph stream generator https://github.com/ferencberes/DEBS-graph-stream-tutorial/blob/main/graph_stream
2. Online graph embedding algorithms
   http://info.ilab.sztaki.hu/~fberes/debs_tutorial/OnlineNodeEmbeddings.slides.html https://github.com/ferencberes/DEBS-graph-stream-tutorial/blob/main/node_embedding/OnlineNodeEmbeddings.ipynb
3. Link prediction by online ML - the Alpenglow toolkit
   https://github.com/rpalovics/Alpenglow

DEBS-graph-stream-tutorial

Resource list: Our papers

1. Pálovics, Benczúr. "Temporal influence over the Last. fm social network." (2015)
2. Frigó, E., Pálovics, R., Kelen, D., Kocsis, L., & Benczúr, A. (2017). Online ranking prediction in non-stationary environments.
3. Frigó, Pálovics, Kelen, Kocsis, Benczúr. (2017). Alpenglow: Open source recommender framework with time-aware learning and evaluation.
4. Béres, Pálovics, Oláh, Benczúr, (2018). Temporal walk based centrality metric for graph streams. Applied network science
5. Béres, Kelen, Pálovics, Benczúr, (2019). Node embeddings in dynamic graphs. Applied Network Science
6. Béres, Seres, Benczúr, Quintyne-Collins (2021). Blockchain is Watching You: Profiling and Deanonymizing Ethereum Users

BME Matematikai Modellalkotás Szeminárium

Embedding networks into vector spaces

András Benczúr, Ferenc Béres,
Domokos Kelen, Róbert Pálovics

Stanford University

Thank You!

The research was supported by the Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program and by Project 2018-1.2.1-NKP-00008: Exploring the Mathematical Foundations of Artificial Intelligence.