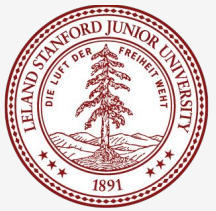
DEBS - June 28-29 2021

Tutorial on graph stream analytics

András Benczúr, Ferenc Béres,
Domokos Kelen, Róbert Pálovics

SZTAKI



Stanford
University

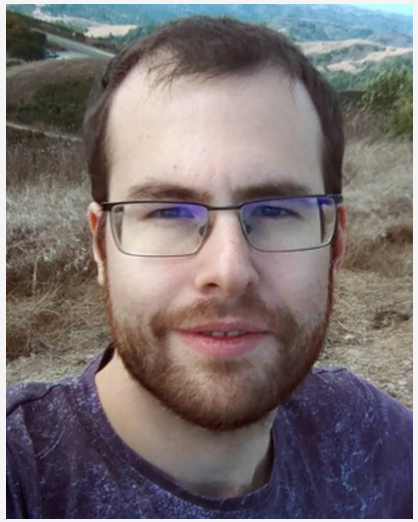
About the presenters

Andras Benczur benczur@sztaki.hu
Ph.D. MIT 1997

Leads a Data Science research lab at Institute for Computer Science and Control, Hungary

Scientific director of Artificial Intelligence National Laboratory, a consortium of 10 institutions - Single point of contact for AI partners in Hungary, milab@sztaki.hu

Ferenc Béres



Domokos Kelen

Róbert Pálovics
Stanford University

AB's current and past Doctoral students

Our related works

- Hands-on Tutorial on Open Source Online Learning Recommenders, RecSys 2017.
- Online machine learning in big data streams, chapters of the Encyclopedia of Big Data Technologies

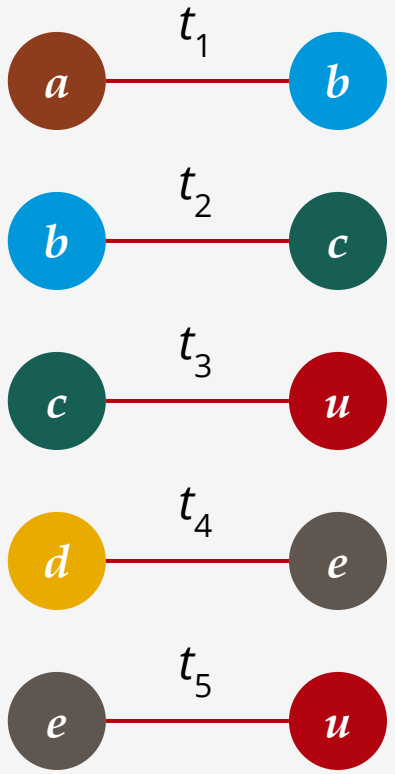
Main Resource

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

DEBS-graph-stream-tutorial

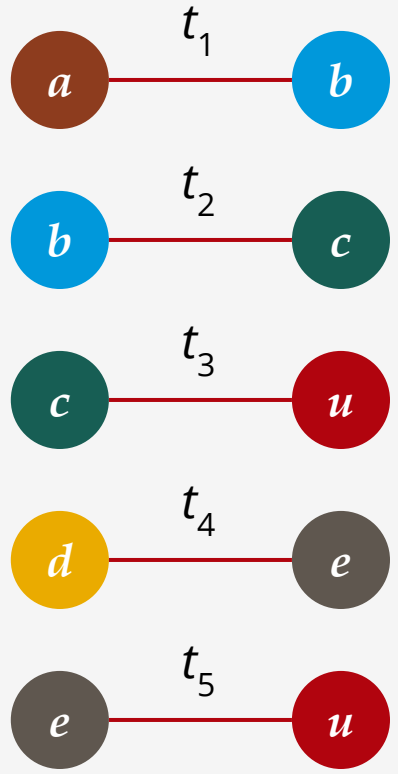
Temporal networks

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



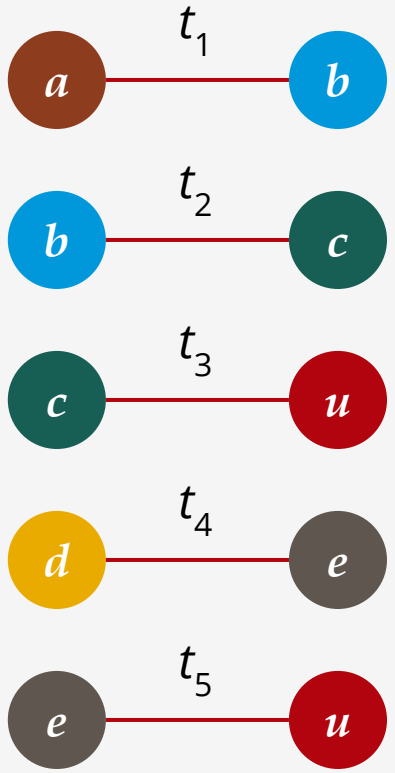
Temporal networks

- 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 [our works]
  - Assessing Ethereum mixer anonymity
  - Tracking Bitcoin transactions in the blockchain
  - Lightning Network (Bitcoin overlay) analysis

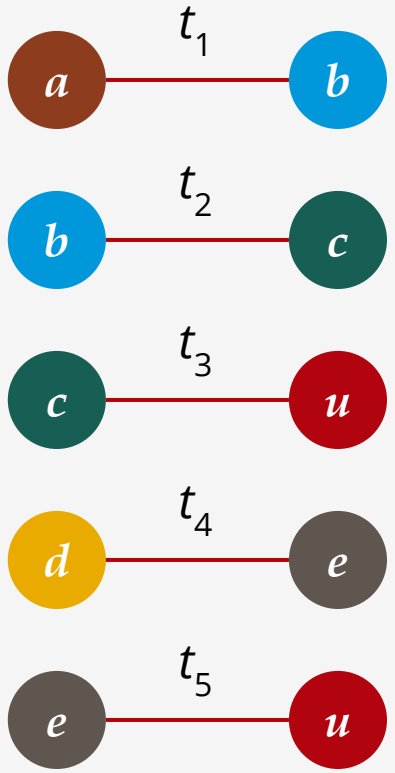


Temporal graph queries in databases

Goal is to find and online update frequent graph patterns

Most related methods build indexes

- Tree and graph search queries in streaming data [***Shasha, Wang, Giugno. PODS 2002. Algorithmics and applications of tree and graph searching***]
  - Uses direct indexing of the most frequent subgraphs
- the streaming version of retrieving graphs quickly from a large database via graph-based indices [***Yan, Yu, Han. SIGMOD 2004. Graph indexing: A frequent structure-based approach***]
  - Uses path indexing



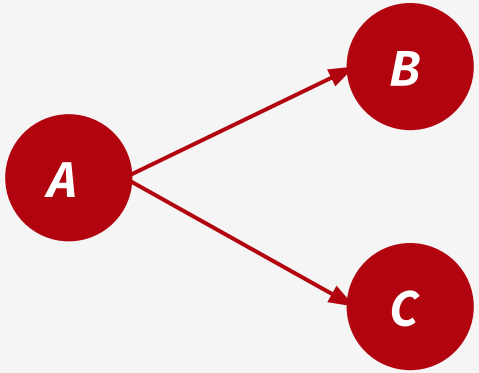
Structure of this Tutorial 60 mins Part I and 45 mins Part II

07:00	Showcase of a Twitter graph stream generator, F.B.
12:30	Basic graph algorithms, triangle counting, sampling, A.B.
28:00	Link prediction by online ML - the Alpenglow toolkit, R.P.
45:30	The notion of temporal walks, A.B.
	15 min Q&A, break
00:00	Online graph embedding algorithms and an experiment F.B.
26:00	Online network centrality algorithms, R.P.
34:30	Outlook, Conclusions, Resource list, Closing A.B.

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

```
1  from twittercrawler.crawlers import TwythonStreamCrawler
2  from twittercrawler.data_io import SocketWriter
3  from twittercrawler.search import get_time_termination
4  import datetime, sys
5
6  if len(sys.argv) != 3:
7      print("Usage: <twitter_api_key> <port>")
8  else:
9      api_fp = sys.argv[1]
10     port = int(sys.argv[2])
11
12     # prepare writers
13     sw = SocketWriter(port=port, export_filter="mention")
14
15     # initialize crawler
16     stream = TwythonStreamCrawler([sw], api_fp)
17
18     # query
19     search_params = {
20         "q":'#BREAKING OR #BREAKINGNEWS OR #breakingnews',
21         "lang":"en",
22     }
23     stream.set_search_arguments(search_args=search_params)
24
25     # run stream search (pause for 0.5 second after each tweet)
26     try:
27         # YOU MUST TERMINATE THIS SEARCH MANUALLY!
28         stream.search()
29     except:
30         raise
31     finally:
32         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.
4. Scalability, distributed processing (only as outlook).

Appeared in 2019
All in one ArXiv: 1802.05872

Issue 1: Algorithmic limitations

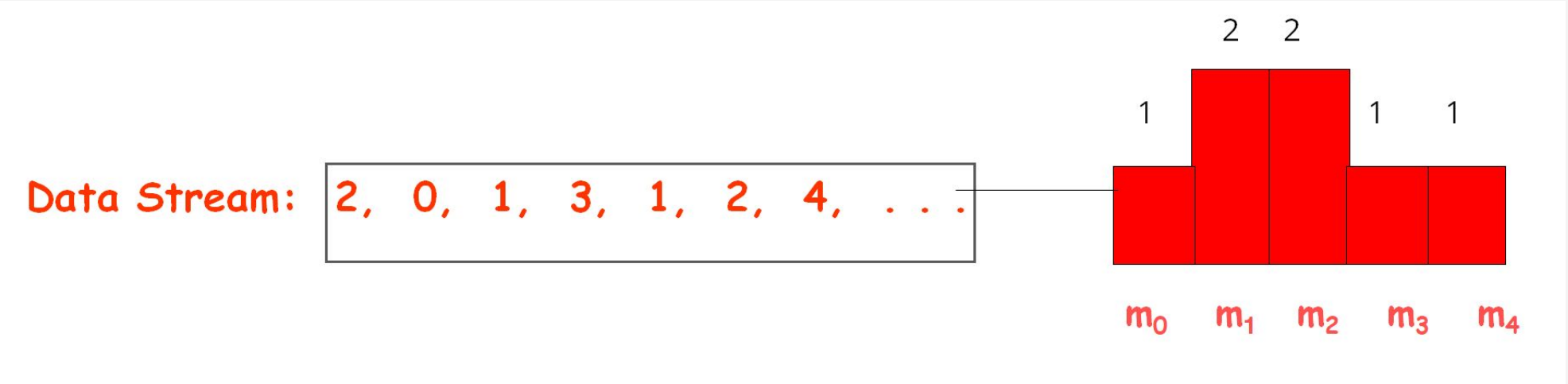
Example: Machine Learning in the data streaming computational model

- No access to (majority of) past data
  - No stochastic gradient descent: we cannot iterate over the data
    - Online gradient descent is possible []
  - 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]

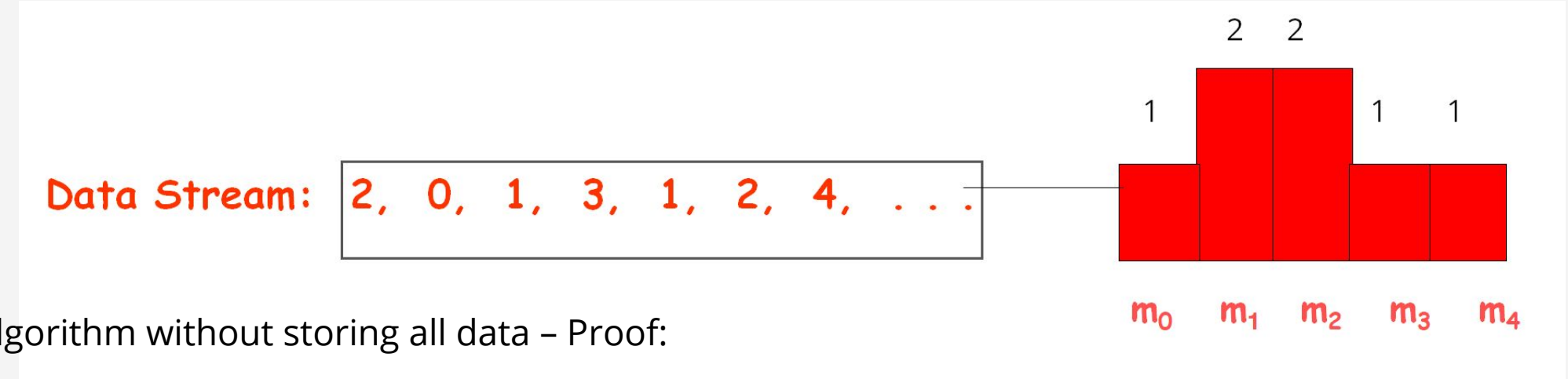
    [Vasiloudis, Beligianni, Morales. BoostVHT, CIKM 2017.]

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)

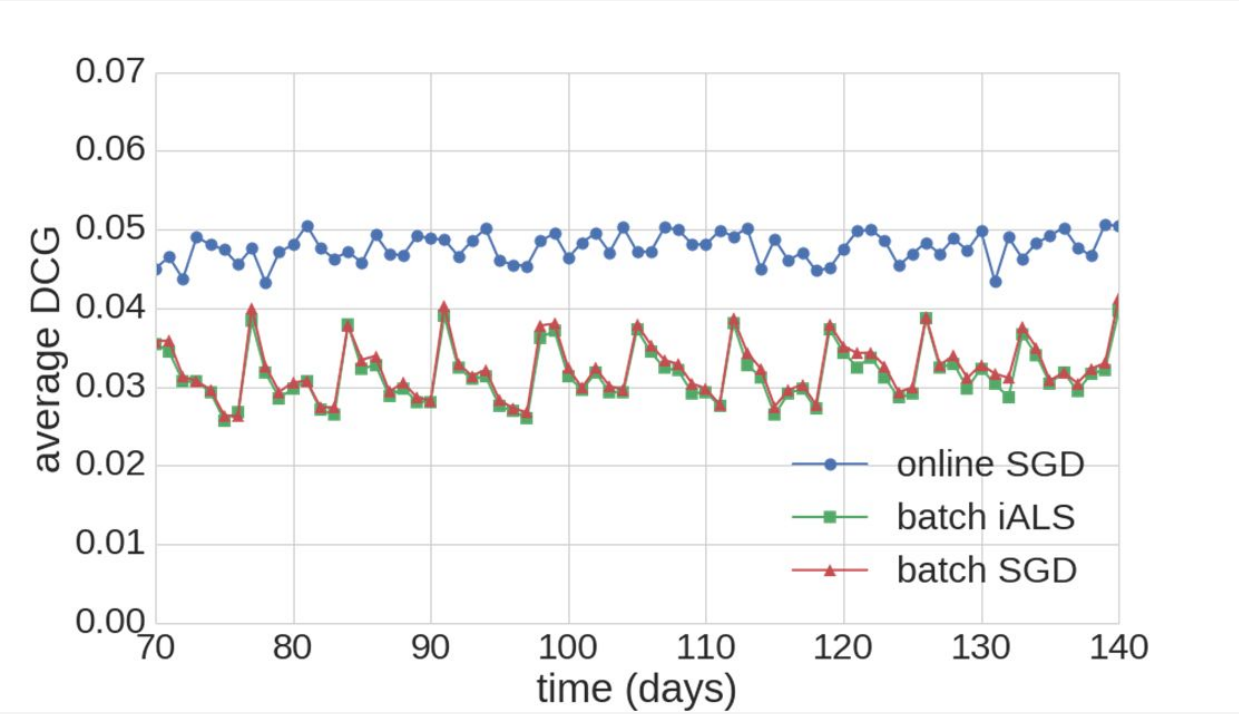
Issue 2: Difficulty in ML evaluation

- 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.P.: Present position and potential developments: Some personal views: Statistical theory:The prequential approach. Journal of the Royal Statistical Society. Series A (General) pp. 278–292 (**1984**)

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

Basic Graph Algorithms

Graph statistics in edge streams

- Simplest example: degree, temporal degree
- Graph statistics [Kim, Anderson. Temporal node centrality in complex networks. Phys. Rev. 2012]: only snapshot based
  - closeness, betweenness
- Graph streaming algorithms in database queries [Henzinger, Raghavan, Rajagopalan. 1999. Computing on data streams.]:
  - node with maximum degree
  - largest connected component
  - node pair connected by the largest number of paths

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]

Closeness, Betweenness centrality

- Closeness centrality: sum of inverse temporal shortest path distances to all other nodes
- Betweenness centrality: proportion of shortest paths passing through the given node
- Version in [Kim, Anderson 2012] considers time intervals
  - Very high computational cost for both measures
  - Algorithms for snapshots, no easy immediate update

Short detour to triangle counting

- The most actively researched algorithm, the “Hello World!” task of graph streaming
- Application: clustering coefficient
- Definition: Number of adjacent edge pairs divided by number of triangles (fraction of closed triangles

- [Buriol et al. PODS 2006. Counting triangles in data streams]
- [De Stefani et al., Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. TKDD 2017]

Example: Idea of the TRIEST triangle counting stream algorithm

- At time t, add the current edge to a sample of size M with probability M/t
- Update the neighborhood counters of the new sample edge
- If the sample is full, discard an edge after updating its counters
- Construct an unbiased estimate of the triangle count
- Standard stream sampling trick, key is the construction and analysis of the estimator (mean and variance)

Graph Sampling - a sample of results

- First results are semi-streaming [Feigenbaum, Kannan, McGregor et al., 2005]
- Algorithms and criteria to maintain properties both in static and in temporal graphs [Leskovec, Faloutsos KDD 2006]
- Graph Sketching for query estimation [Zhao et al., 2011]
- Graph Priority Sampling (GPS), an order–based reservoir sampling [Ahmed et al., 2017]

Example: link prediction

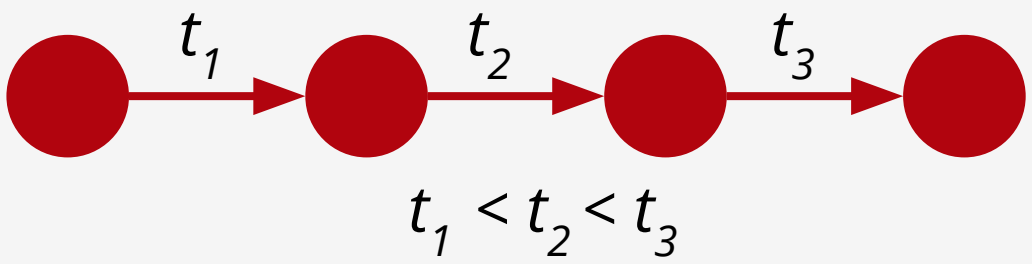
Predict the next element of the stream
Or, rather, the next element adjacent to a node in question

Temporal Walks

Predict the next element of the stream
Or, rather, the next element adjacent to a node in question

Time respective paths

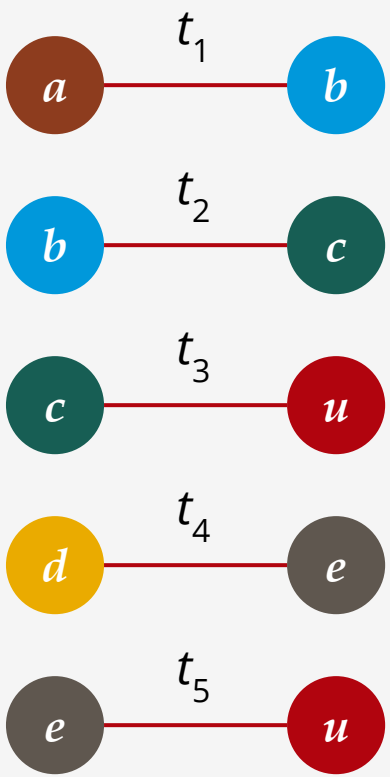
- Adjacent edges that are ordered in time
- Models a flow, e.g.
  - information flow in social networks
  - flow of funds or goods in the economy
- Concept
  - delay t_2-t_1 is small, then flow is more likely

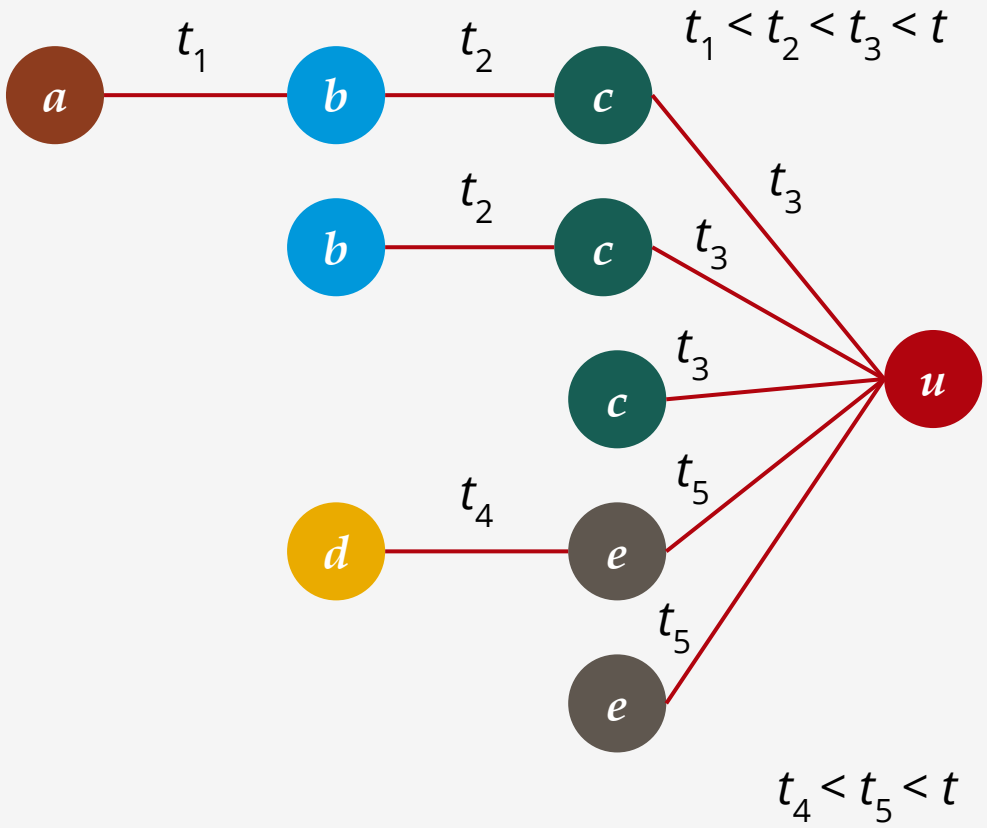


Network centrality metrics

Temporal Katz Centrality

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





Temporal Katz Centrality

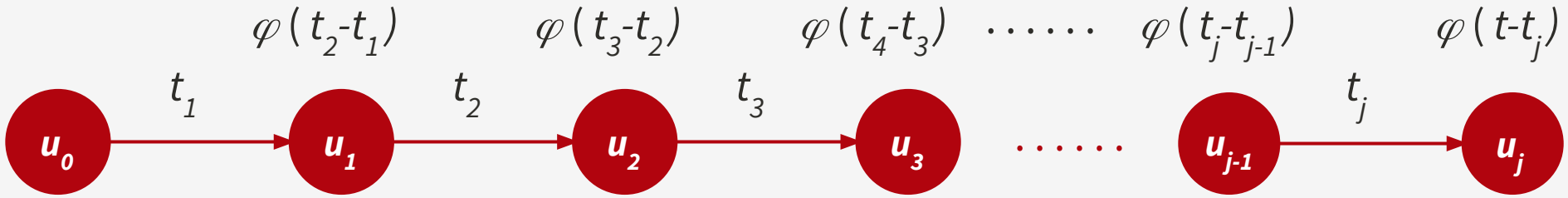
- Centrality for node *u* at time *t*

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

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

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

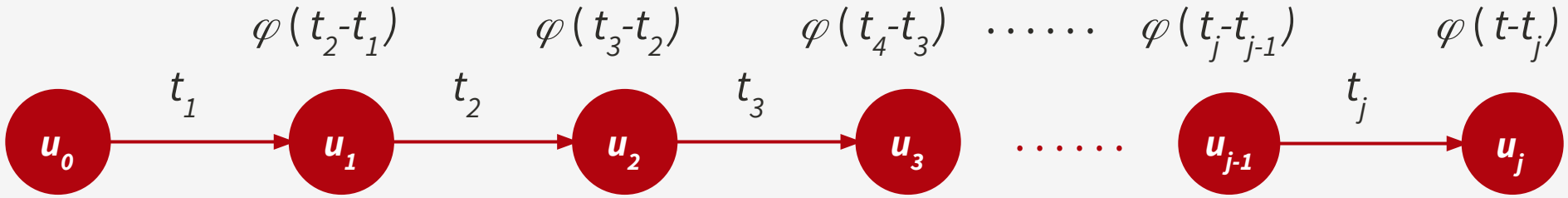
- where edges appeared at (t_1 , t_2 , . . . , t_j) for walk *z*



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

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

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

- Given an underlying graph with edge set of size *E*
- We sample uniform random *T* edges

Goal: calculate the expected value of temporal Katz centrality

Expected value - φ: = β

- Given an underlying graph with edge set of size *E*
- We sample uniform random *T* edges
- 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

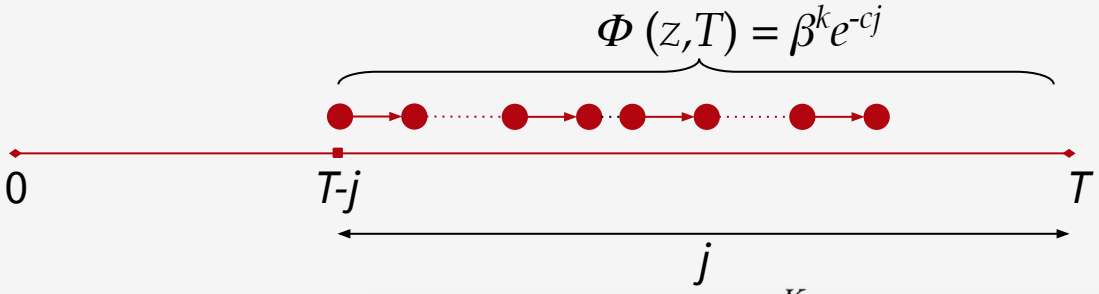
$$\vec{\text{TemporalKatz}} = \mathbf{1} \cdot \sum_{k=0}^{K} \beta^k A^k \binom{T}{k} \cdot E^{-k} \simeq \mathbf{1} \cdot \sum_{k=0}^{K} \beta^k A^k (T/E)^k / k!$$

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 $\exp\{c\} = \exp\{c'/E\} \approx 1 + c'/E$

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

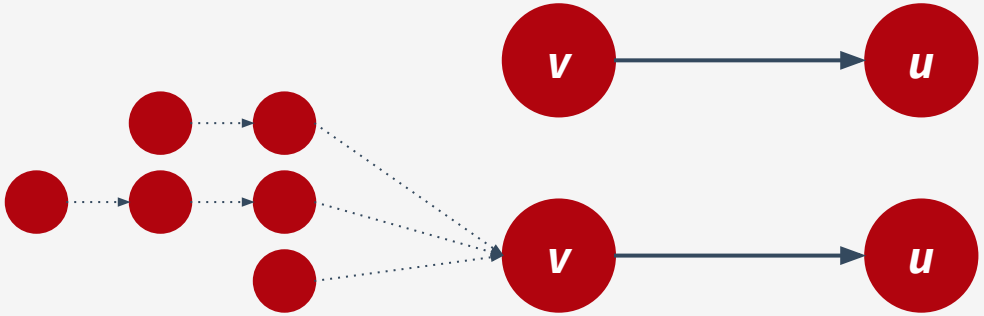
Temporal Katz converges to static Katz on uniformly sampled edge streams

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 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!



Temporal Katz Centrality - computation

- For each node u we initialize $r(u) := 0$
- We maintain $r_u(t)$, w_{vu} and t_{vu}
- When edge uv appears
  - we calculate the current value of r_v

$$r_v := \sum_{zv \in E(t)} w_{zv} \cdot \varphi(t - t_{zv})$$

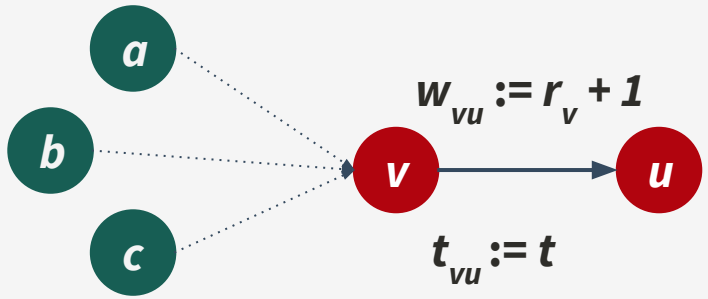
  - $w_{vu} := r_v + 1$
  - $t_{vu} := t$
- since $\varphi(a) \cdot \varphi(b) = \varphi(a + b)$

$$r_v := r_v \cdot \varphi(t - t_v);$$ update r_v

$$r_u := r_u \cdot \varphi(t - t_u) + (r_v + 1) \cdot \beta;$$ add new walks ending in u

$$t_u := t, \quad t_v := t,$$ update old walks ending in u

- no need to store w_{vu}

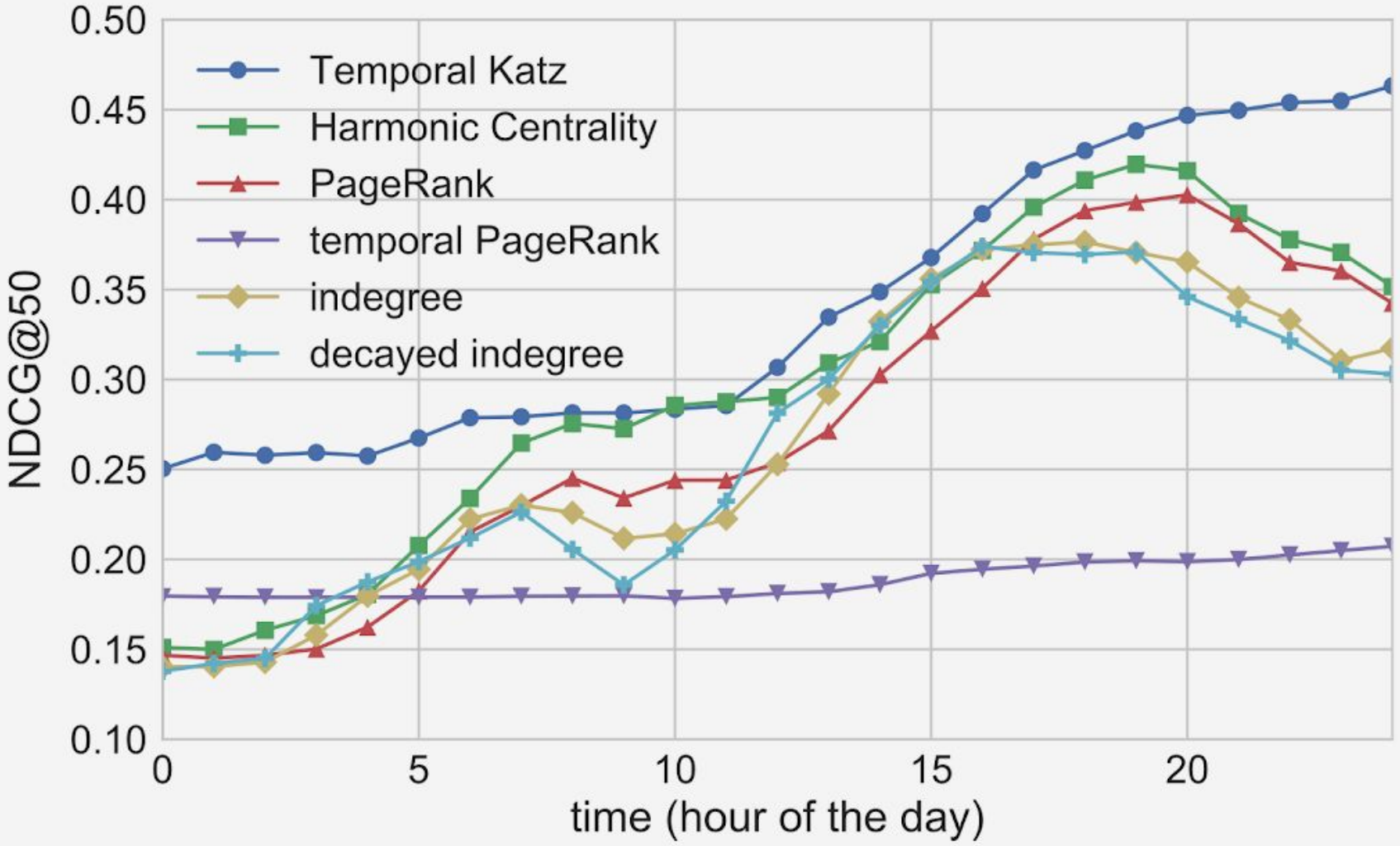


Experiments on Twitter data

- 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

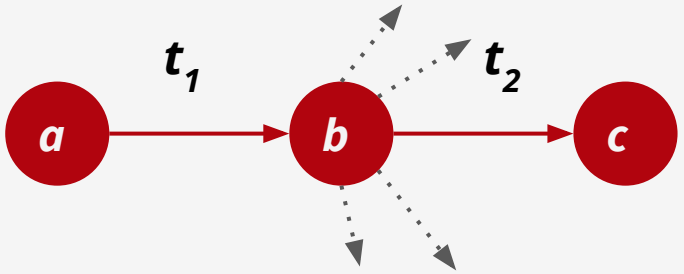
1	Roland-Garros	@rolandgarros	0
2	Stanislas Wawrinka	@stanwawrinka	1
3	Andy Murray	@andy_murray	1
4	Simona Halep	@Simona_Halep	0
5	Rafa Nadal	@RafaelNadal	1
6	Dominic Thiem	@ThiemDomi	1
7	Timea Bacsinszky	@TimeaOfficial	0
8	Rohan Bopanna	@rohanbopanna	0
9	Ana Ivanovic	@AnaIvanovic	0
10	WTA	@WTA	0
11	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



Temporal PageRank

- Polina Rozenstein & Aris Gionis
- Different φ () weighting function
  - for adjacent edges (a, b, t_1) and (b, c, t_2)
  - L := number of edges (b, x, t) where $t_1 < t < t_2$
  - $\varphi \sim |\alpha|^L$, $\alpha < 1$



Network node embeddings

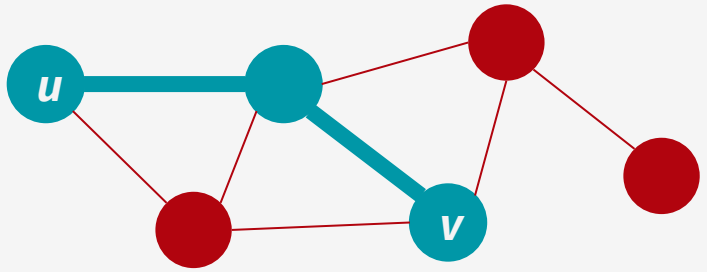
Network node embeddings

- 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

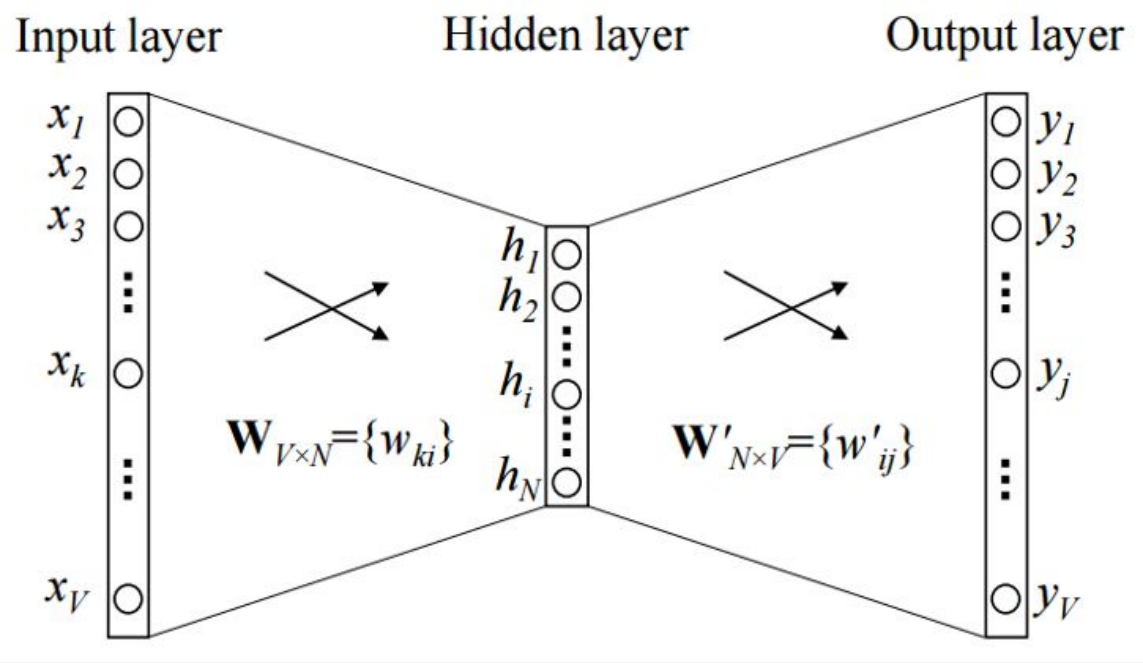


- Random walk based methods

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

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

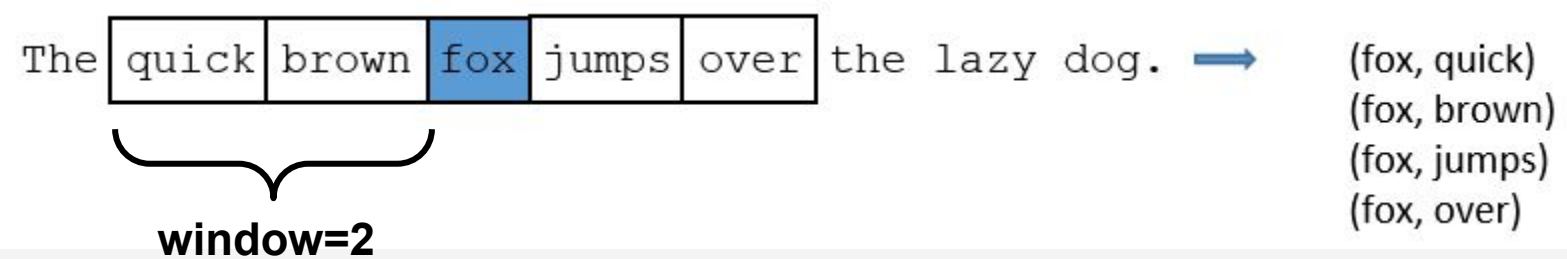
Word2Vec



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

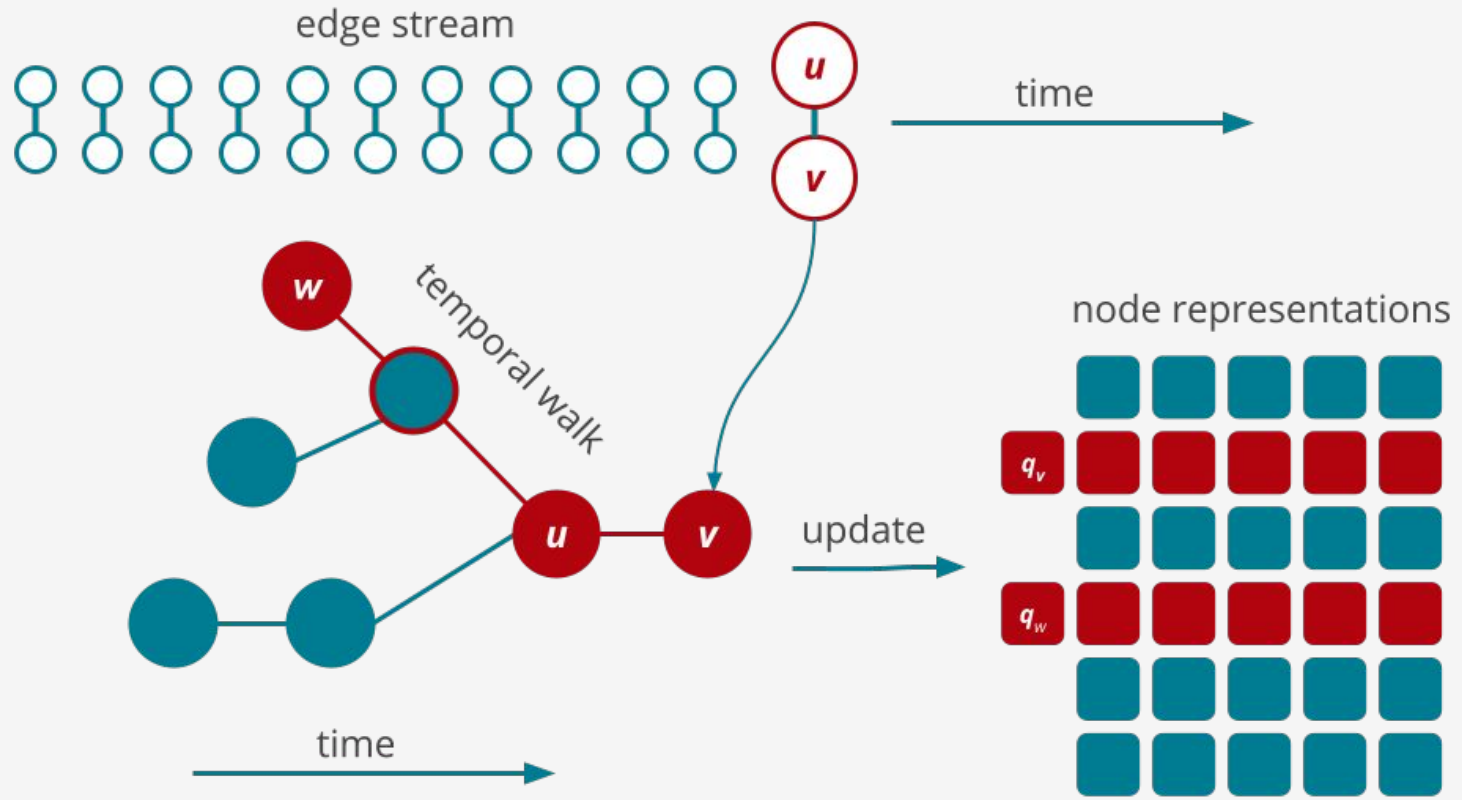


Static node embedding methods

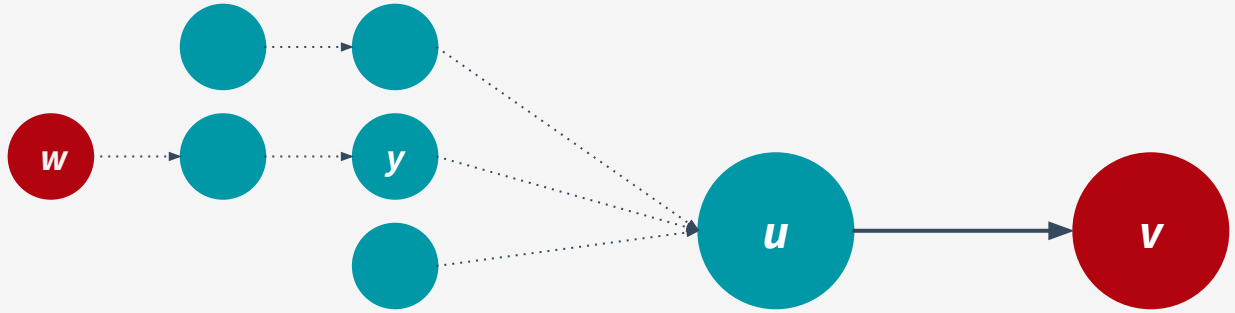
- DeepWalk: [Perozzi, Al-Rfou, Skiena. KDD 2014 Deepwalk: Online learning of social representations]
  - Corpus: nodes of the graph
  - Sentences: random walks
  - no control over the explored neighborhood
- Node2Vec: [Grover, Leskovec. KDD 2016: Scalable feature learning for networks]
  - Return parameter: 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

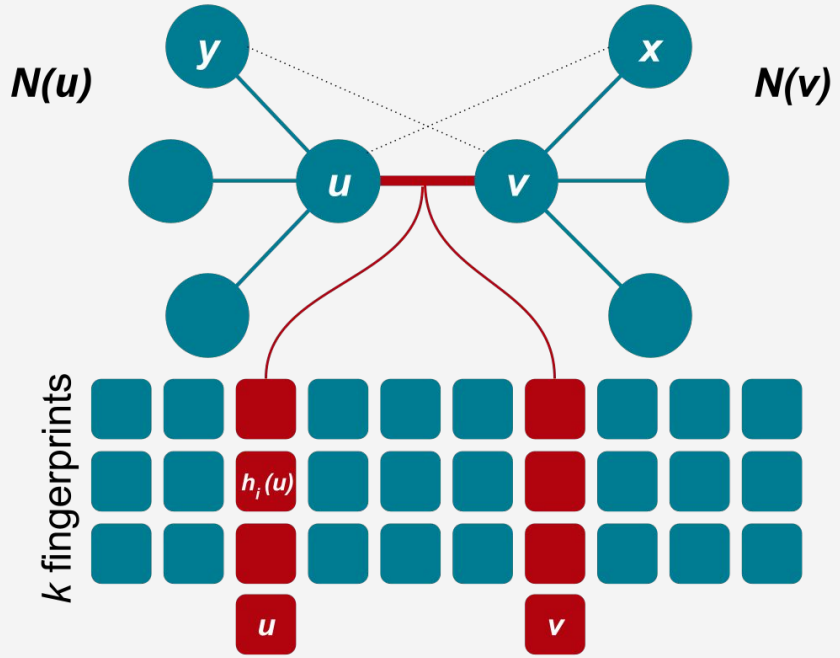


- Process edges in temporal order
- For each edge (u,v)
  - learn that p_u and p_v are similar
  - start random time-respective walks $(w,\ldots,u,v)$ to the past ending in (u,v)
  - learn for (w,v) that p_w and p_v are similar

- 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

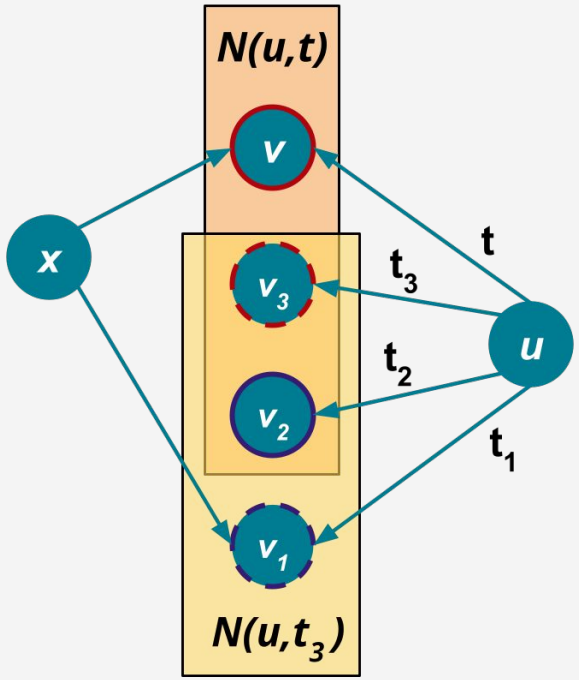
- Optimizes the embedding to match time-aware Jaccard similarity of the neighborhood of *u* and *x*
- Temporal neighborhood:
  - multi-set N(*u*, t)
  - w(y) = exp(−c(t − t(*uy*))) for edge *uy*
  - Emphasize the importance of new edges:
    discard $y \in N(u, t)$ with probability 1 − w(*y*)
    when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



Summary

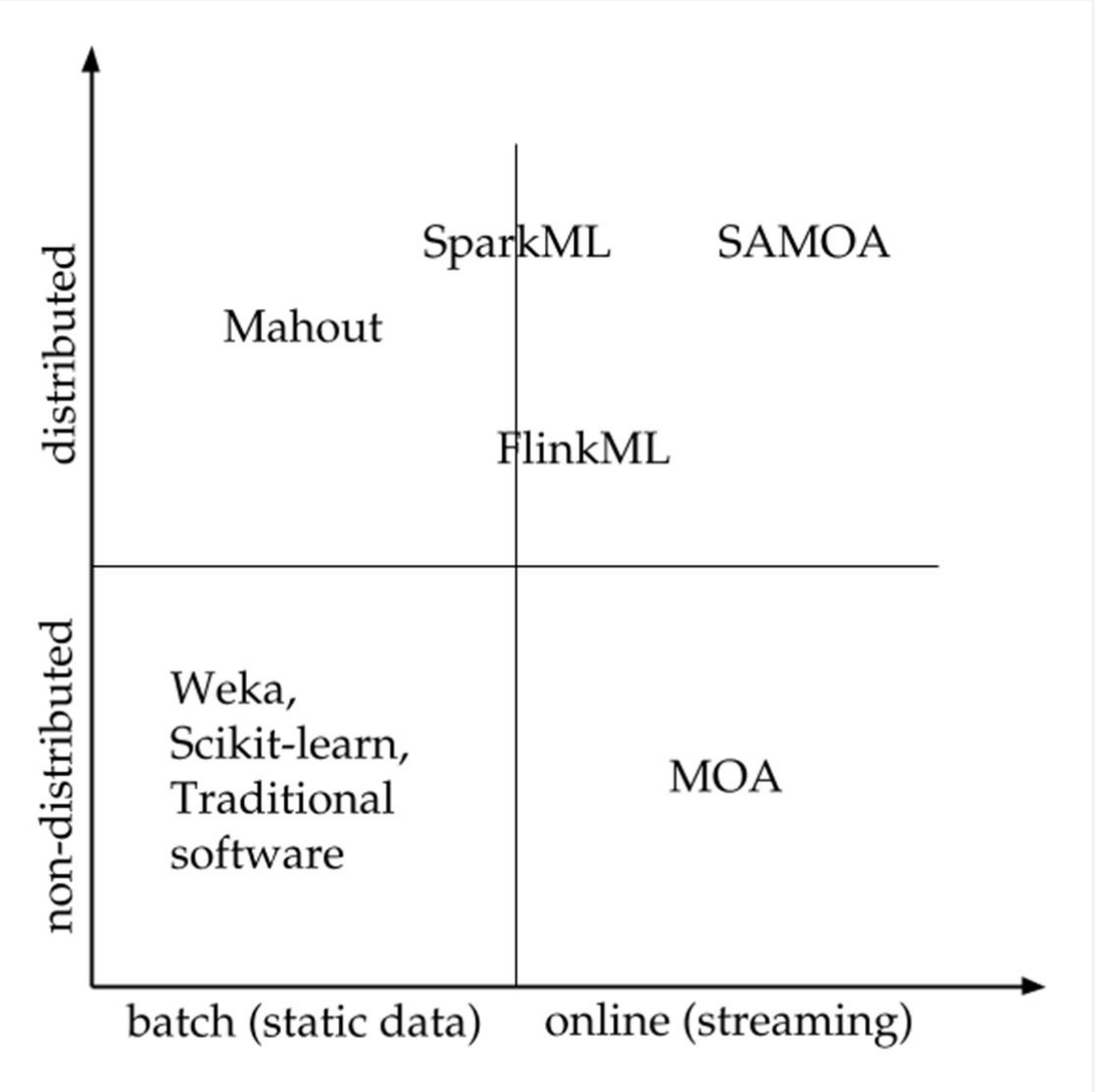
- We surveyed results to analyze and model edge streams:
  - Interactions in a social network service
  - Queries in graph databases with real-time stream updates
- Ingredients:
  - Data streaming computational model
  - Temporal networks
  - Prequential evaluation
- Algorithms that update the results while the edge stream is processed:
  - Traditional graph properties (sampling, subgraph counting, graph query evaluation)
  - Low-rank approximation, link prediction
  - Network embedding, similarity search
  - Network centrality algorithms

Outlook

Data stream processing systems
Concept drift detection

Distributed stream processing architectures

- Easiest and typical way to train models is a single-server (maybe multicore or GPU but still in-memory) using static data
- Distributed is much less convenient
  - Install, optimize performance
  - Training labels are rarely available in huge quantities
- Online learning seems like a small niche application now
- Distributed and streaming
  - SAMOA: standalone library with connectors to many stream processing engines
  - SparkML: although Spark can process streams in mini-batches, SparkML methods are batch
  - FlinkML under development with low community support
- Model Serving is an important need



Concept drift

- Recall the success of online methods against static counterparts
  - Recommender systems: users can have different moods in different sessions, need to quickly re-learn
  - Link prediction, similarity search and centrality in social networks can be affected by physical events (Tournament schedule)
- Traditional learners assume finite, static, identically distributed data sets

Concept drift detection

- A very early result that introduces concept-adapting decision tree learner [Hulten, Spencer, Domingos. "Mining time-changing data streams." *KDD* 2001]
  - Evaluation on Web page request data
- Survey and performance comparison, mostly considering tree learning [Gama, Sebastiao, Rodrigues. On evaluating stream learning algorithms, 2013]
  - Prequential error estimates to detect drift
- Six challenges [Zliobaite et al. "Next challenges for adaptive learning systems." KDD 2012]:
  - Making adaptive learning systems scalable
  - Evaluation with real data
  - Usability and trust
  - Integrating expert knowledge
  - Application needs
  - From adaptive algorithms to adaptive tools

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

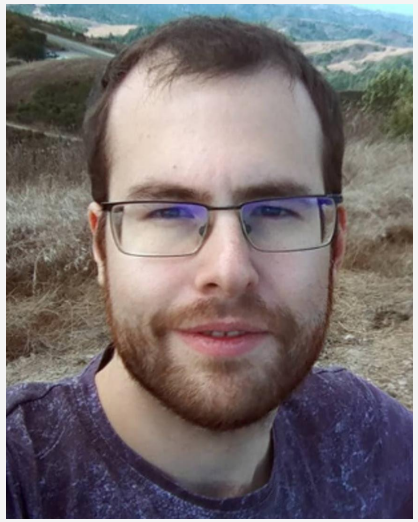
Conclusions

- Graph streaming has several applications in monitoring online events
- Streaming algorithms react faster than static counterparts that process graph snapshots
- Computational challenge to update models, representations, statistics immediately after a new edge arrives
- Several algorithms in the data stream computational model since ~2000
- Most of our methods are semi-streaming and store relative large representations in memory
- Fast update is however crucial

The presenters

Andras Benczur
benczur@sztaki.hu

Ferenc Béres
beres@sztaki.hu



Domokos Kelen
kdomokos@ilab.sztaki.hu

Róbert Pálovics
palovics@stanford.edu

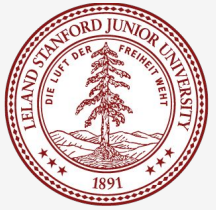
Thank You!

DEBS - June 28-29 2021

Tutorial on graph stream analytics

SZTAKI

András Benczúr, Ferenc Béres,
Domokos Kelen, Róbert Pálovics



Stanford University

Thank You!