清華大學 Tsinghua University

media and network lab

IEEE ICDM 2018

Billion-scale Network Embedding with Iterative Random Projection

Ziwei Zhang
Tsinghua U

Peng Cui
Tsinghua U

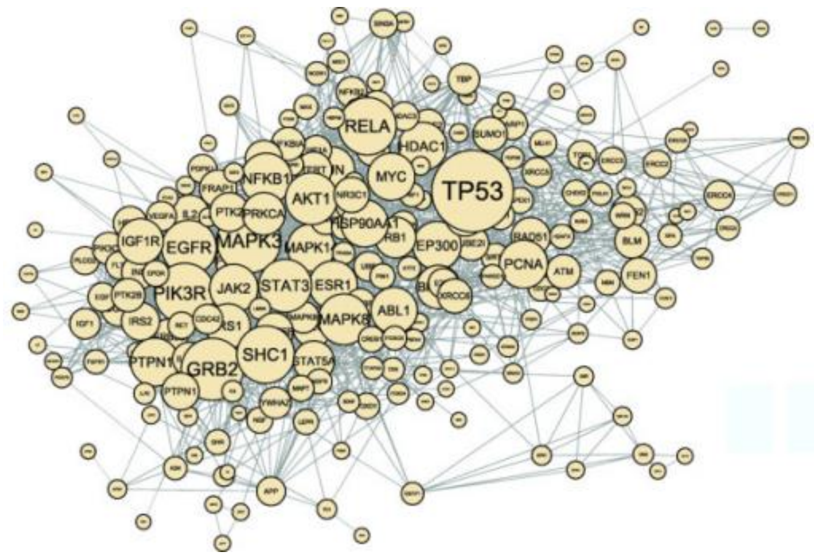
Haoyang Li
Tsinghua U

Xiao Wang
Tsinghua U

Wenwu Zhu
Tsinghua U

Network Data is Ubiquitous

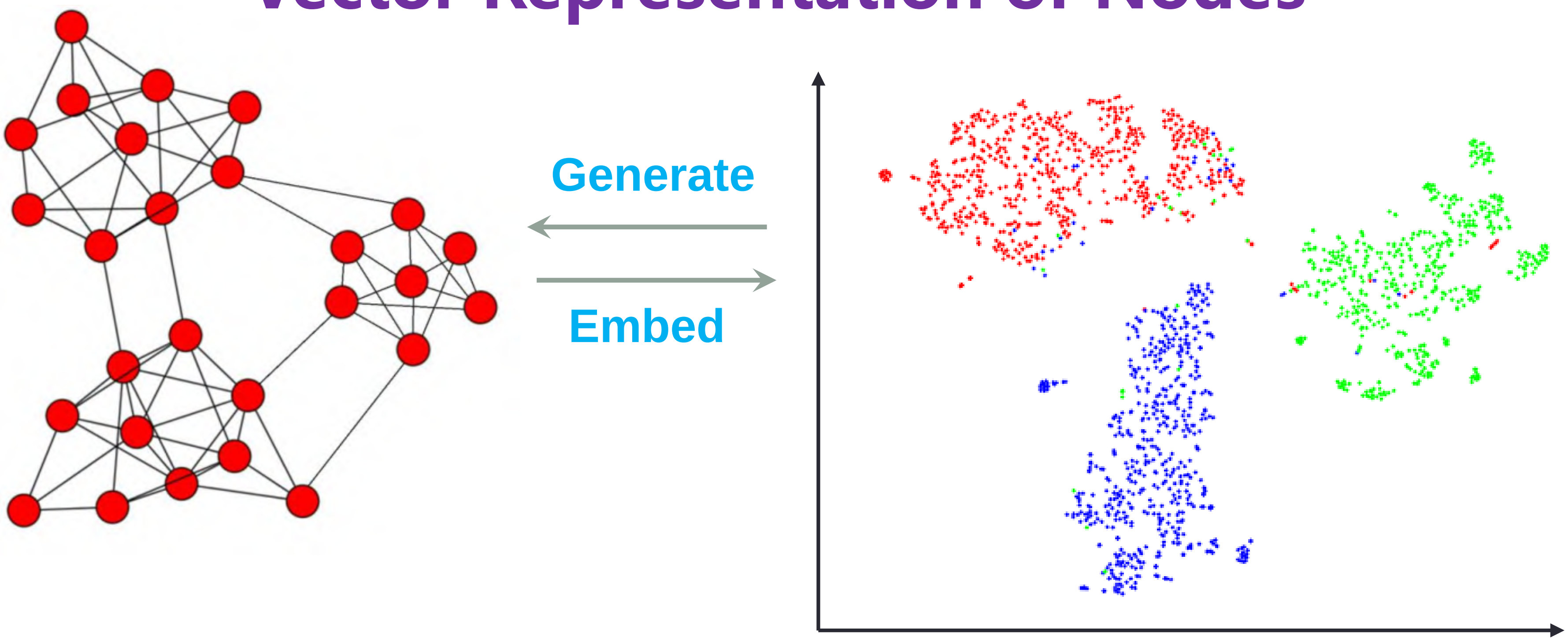
Social Network



Biology Network

Traffic Network

Network Embedding: Vector Representation of Nodes



- Apply feature-based machine learning algorithms
- Fast computing of nodes similarity
- Support parallel computing

- Applications: link prediction, node classification, community detection, centrality measure, anomaly detection ...

Challenge: Billion-scale Network Data

Social Networks

- WeChat: 1 billion monthly active users (March, 2018)
- Facebook: 2 billion active users (2017)

E-commerce Networks

- Amazon: 353 million products, 310 million users, 5 billion orders (2017)

Citation Networks

- 130 million authors, 233 million publications, 754 million citations (Aminer, 2018)

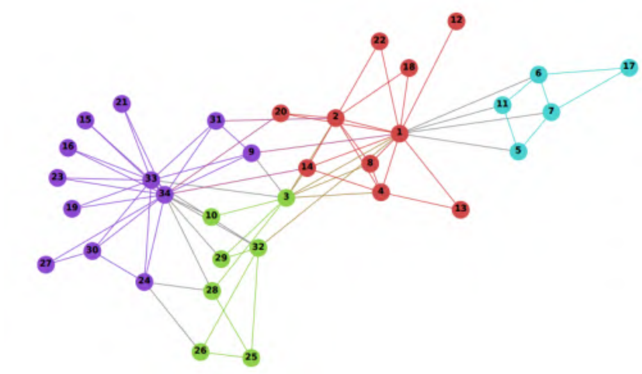
How to conduct network embedding for such large-scale network data?

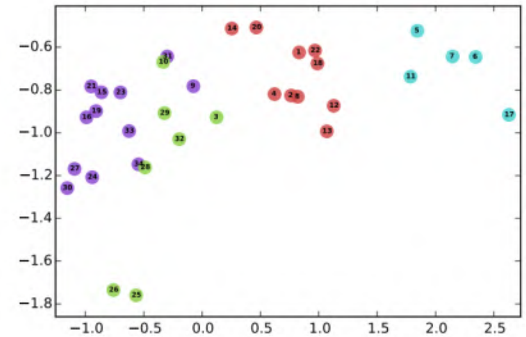
Bottleneck of Existing Methods

- Methods based on random-walks
  - DeepWalk, B. Perozzi, et al. *KDD 2014.*
  - LINE, J. Tang, et al. *WWW 2015.*
  - Node2vec, A. Grover, et al. *KDD 2016.*
- Methods based on matrix factorization
  - M-NMF, X. Wang, et al. *AAAI 2017.*
  - AROPE, Z. Zhang, et al. *KDD 2018.*
- Methods based on deep learning
  - SDNE, D. Wang, et al. *KDD 2016.*
  - DVNE, D. Zhu, et al. *KDD 2018.*
- Common bottleneck: based on sophisticated optimization
  - Computationally expansive
  - Hard to resort to distributed computing scheme
    - Optimization is entangled and needs global information
      → Communication cost is high
- Only handle thousands or millions of nodes and edges

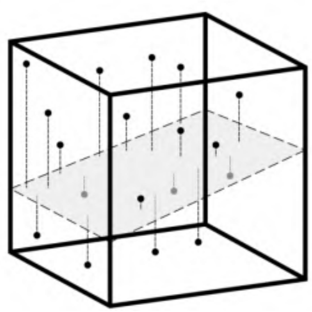
Random Projection

- Network embedding: essentially a dimension reduction problem



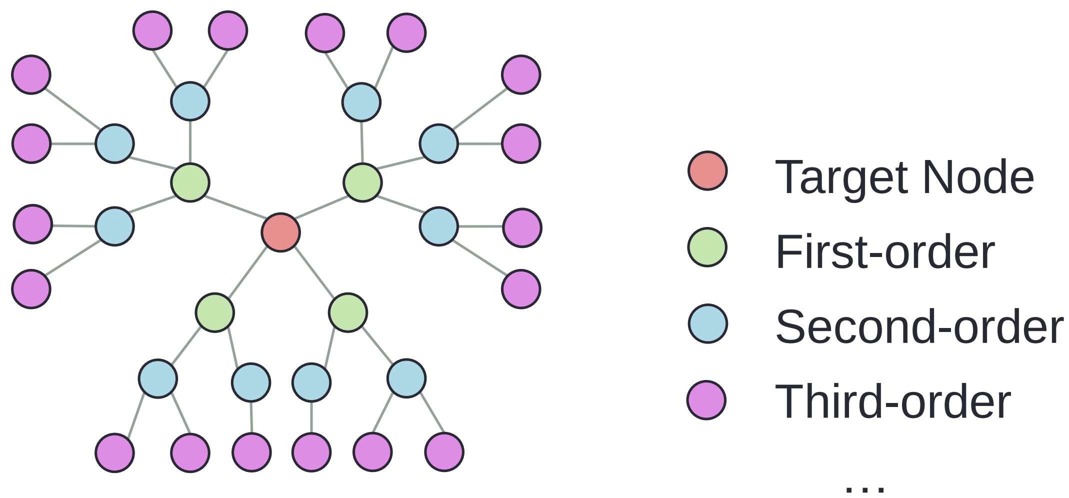


- Random projection: optimization-free for dimension reduction
  - Basic idea: randomly project data into a low-dimensional subspace
  - Extremely efficient and friendly to distributed computing



High-Order Proximity

- Key network property: high-order proximity



  - Can solve the network sparsity problem
  - Measure indirect relationship between nodes

→ How to design a high-order proximity preserved random projection?

Problem Formulation

- Objective function: matrix factorization of preserving high-order proximity

$$\min_{U,V} \|S - UV^T\|_p^2$$

$$S = f(A) = \alpha_1 A^1 + \alpha_2 A^2 + \cdots + \alpha_q A^q$$

- Slight modification: assuming positive semi-definite and using 2 norm

$$\min_{U} \|SS^T - UU^T\|_2$$

$$S = f(A) = \alpha_1 A^1 + \alpha_2 A^2 + \cdots + \alpha_q A^q$$

- Random projection:
  - Denote $R \in \mathbb{R}^{N \times d}$ as a Gaussian random matrix

$$R_{ij} \sim \mathcal{N}\left(0, \frac{1}{d}\right)$$

  - Surprisingly simple result:

$$U = SR$$

Theoretical Guarantee

- Theoretical guarantee

Theorem 1. *For any similarity matrix* $\mathbf{S}$*, denote its rank as* $r_{\mathbf{S}}$*. Then, for any* $\epsilon \in \left(0, \frac{1}{2}\right)$*, the following equation holds:*

$$P\left[\left\|\mathbf{S}\cdot\mathbf{S}^T - \mathbf{U}\cdot\mathbf{U}^T\right\|_2 > \epsilon\left\|\mathbf{S}^T\cdot\mathbf{S}\right\|_2\right] \leq 2r_{\mathbf{S}}e^{-\frac{\left(\epsilon^2-\epsilon^3\right)d}{4}},$$

where $\mathbf{U} = \mathbf{S}\cdot\mathbf{R}$ *and* $\mathbf{R}$ *is a Gaussian random matrix.*

- Basically, random projection can effectively minimize the objective function
- However, calculating S is still very time consuming

Iterative Projection

- Iterative projection:

$$U = SR = \left(\alpha_1 A^1 + \alpha_2 A^2 + \cdots + \alpha_q A^q\right)R$$
$$= \alpha_1 \boxed{A^1 R} + \alpha_2 \boxed{A^2 R} + \cdots + \alpha_q \boxed{A^q R}$$

$\times$ A $\quad$ $\times$ A $\quad$ $\times$ A

  - Can be calculated iteratively
- Why efficient?
  - A: $N \times N$ sparse adjacency matrix
  - R: $N \times d$ low-dimensional matrix
  - Associative law of matrix multiplication

Sparse

$$AA \ldots AA\boxed{AR}$$ Low-dimensional

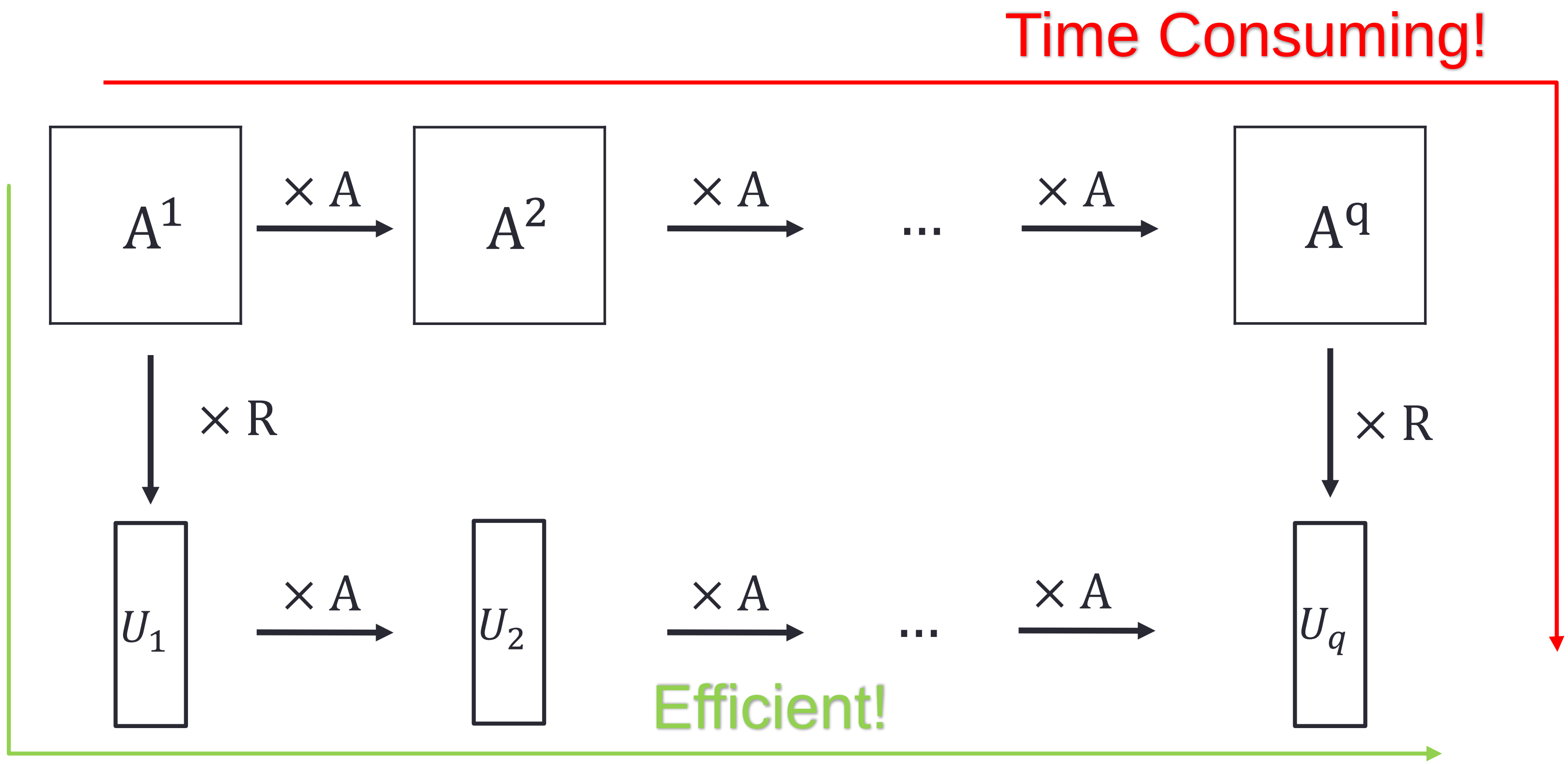
Sparse

Sparse matrix multiplication! $$AA \ldots A\boxed{A(AR)}$$ Low-dimensional

Sparse

$$AA \ldots \boxed{A(AAR)}$$ Low-dimensional

Iterative Projection



RandNE: Iterative Random projection Network Embedding

Algorithm 1 RandNE: Iterative Random Projection Network Embedding

Require: Adjacency Matrix $\mathbf{A}$, Dimensionality d, Order q, Weights $\alpha_0, \alpha_1, ..., \alpha_q$

Ensure: Embedding Results $\mathbf{U}$

1: Generate $\mathbf{R} \in \mathbb{R}^{N \times d} \sim \mathcal{N}(0, \frac{1}{d})$

2: Perform a Gram Schmidt process on $\mathbf{R}$ to obtain the orthogonal projection matrix $\mathbf{U}_0$

3: **for** i in 1:q **do**

4:     Calculate $\mathbf{U}_i = \mathbf{A} \cdot \mathbf{U}_{i-1}$

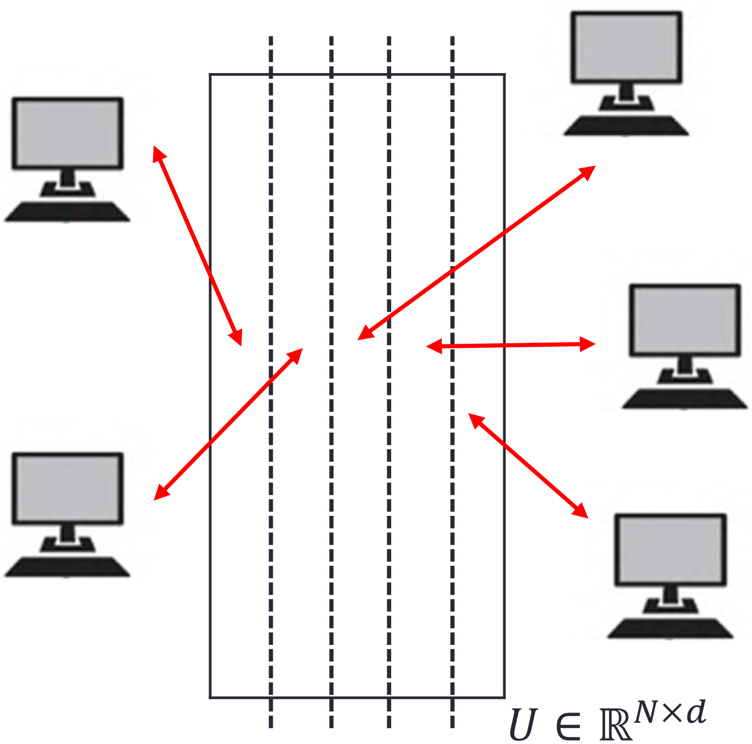
5: **end for**

6: Calculate $\mathbf{U} = \alpha_0 \mathbf{U}_0 + \alpha_1 \mathbf{U}_1 + ... + \alpha_q \mathbf{U}_q$

- Time Complexity: $O(qMd + Nd^2)$
  - N/M: number of nodes/edges; d: dimension; q: order
  - Linear w.r.t. network size
  - Only need to calculate q sparse matrix products
    - Orders of magnitude faster than existing methods!
- Advantages:
  - Distributed Calculation
  - Dynamic Updating

Distributed Calculation

- Iterative random projection only involves matrix product $U_i = AU_{i-1}$
  - Each dimension can be calculated separately
    - Property of sparse matrix product
  - No communication is needed during calculation!



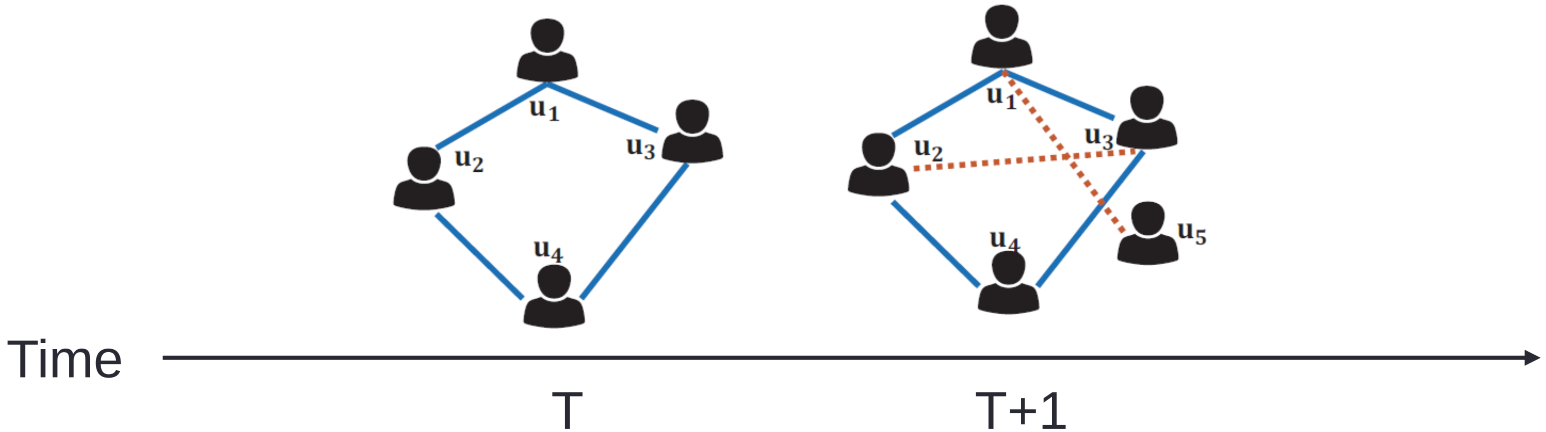
Algorithm 2 Distributed Calculation of RandNE

Require: Adjacency matrix $\mathbf{A}$, Initial Projection $\mathbf{U}_0$, Parameters of RandNE, K Distributed Servers
Ensure: Embedding Results $\mathbf{U}$

Broadcast $\mathbf{A}$, $\mathbf{U}_0$ and parameters into K servers
Set i = 1
repeat
    if There is an idle server k **then**
        Calculate $\mathbf{U}(i,:)$ in server k
        i = i + 1
        Gather $\mathbf{U}(i,:)$ from server k after calculation
    end if
until $i > d$
Return $\mathbf{U}$

Dynamic Updating

- Networks are dynamic in nature
  - E.g., in social networks, users add/delete friends, new users join, old users leave



- Changes of edges → Calculate incremental parts!

$$U_i + \Delta U_i = (A + \Delta A) \cdot (U_{i-1} + \Delta U_{i-1})$$

$$\rightarrow \quad \Delta U_i = A \cdot \Delta U_{i-1} + \Delta A \cdot U_{i-1} + \Delta A \cdot \Delta U_{i-1}$$

- Changes of nodes → adjust the dimensionality

Dynamic Updating

Algorithm 3 Dynamic Updating of RandNE

Require: Adjacency Matrix $\mathbf{A}$, Dynamic Changes $\Delta\mathbf{A}$, Previous Projection Results $\mathbf{U}_0, \mathbf{U}_1, ..., \mathbf{U}_q$

Ensure: Updated Projection Results $\mathbf{U}'_0, \mathbf{U}'_1, ..., \mathbf{U}'_q$

```
if ΔA includes N' new nodes then
    Generate an orthogonal projection Û_0 ∈ R^{N'×d}
    Concatenate Û_0 with U_0 to obtain U'_0
    Add N' all-zero rows in U_1...U_q
end if
Set ΔU_0 = 0
for i in 1:q do
    Calculate ΔU_i using Eq. (7)
    Calculate U'_i = U_i + ΔU_i
end for
```

- Linear scalability w.r.t. number of changed nodes/edges

  Theorem 3. *The time complexity of dynamic updating is linear with the number of changed nodes and number of changed edges respectively.*

- No error accumulation
  - Identical results as re-running the algorithm

Experimental Setting: Moderate-scale Networks

- Datasets: BlogCatalog, Flickr, YouTube

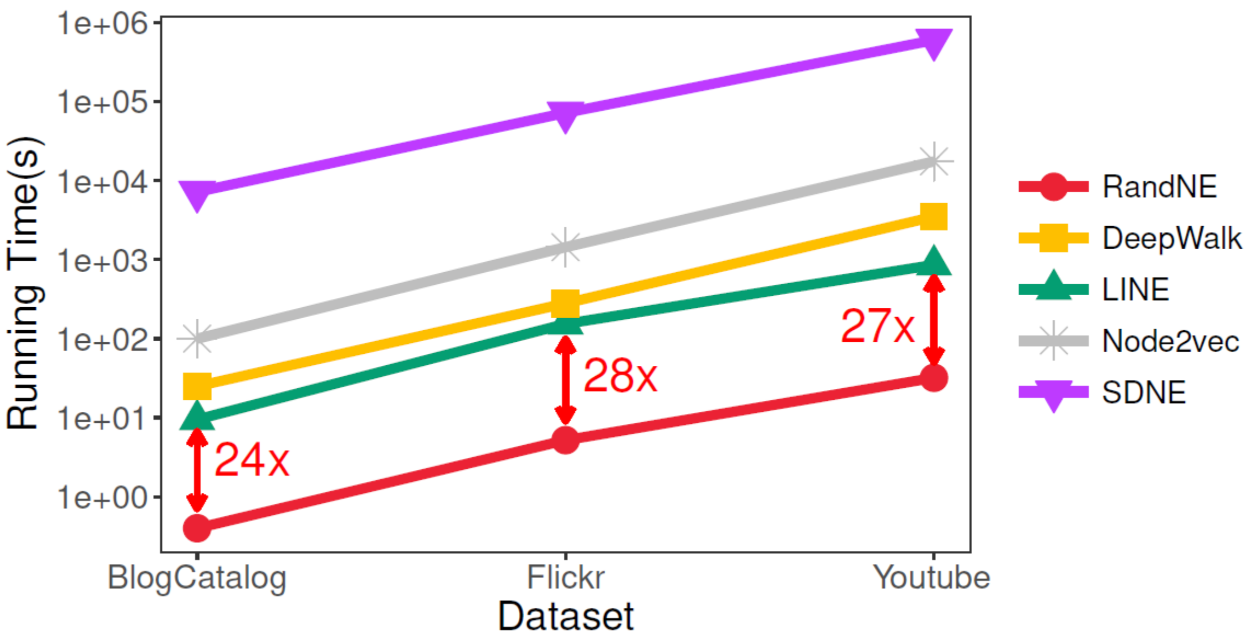
TABLE I
THE STATISTICS OF DATASETS

Dataset	# Nodes	# Edges	# Labels
BlogCatalog	10,312	667,966	39
Flickr	80,513	11,799,764	47
Youtube	1,138,499	5,980,886	195

- Baselines:
  - DeepWalk (KDD 2014): DFS random walk + skip-gram
  - LINE (WWW 2015): BFS random walk + skip-gram
  - Node2vec (KDD 2016): biased random walk + skip-gram
  - SDNE (KDD 2016): deep auto-encoder

Experimental Results

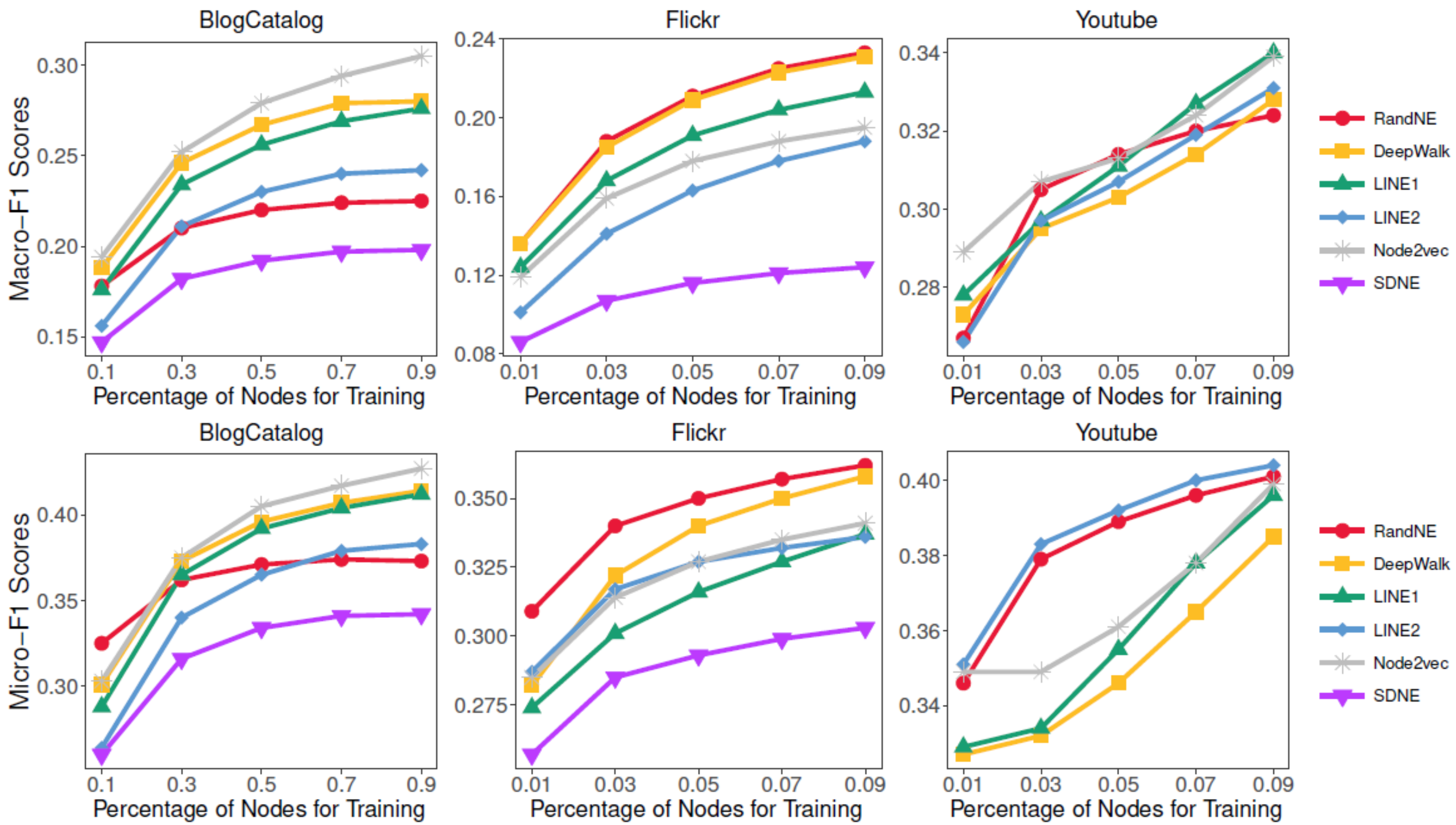
- Running time



At least dozens of times faster

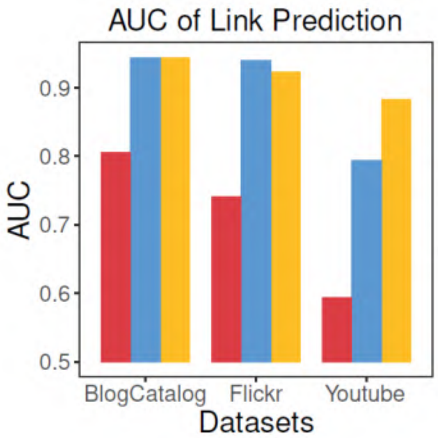
Experimental Results

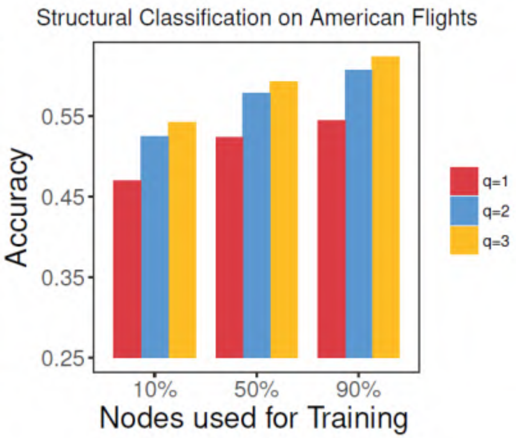
- Node Classification



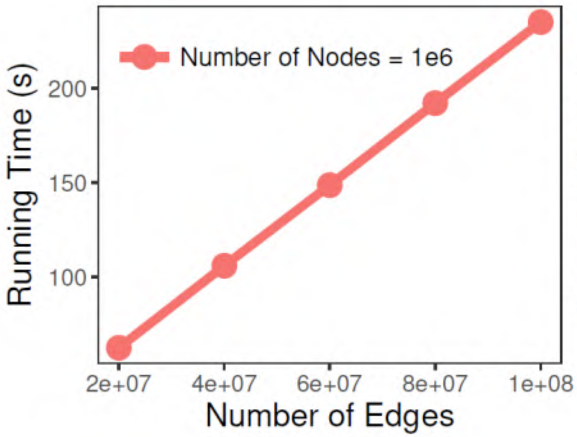
Experimental Results

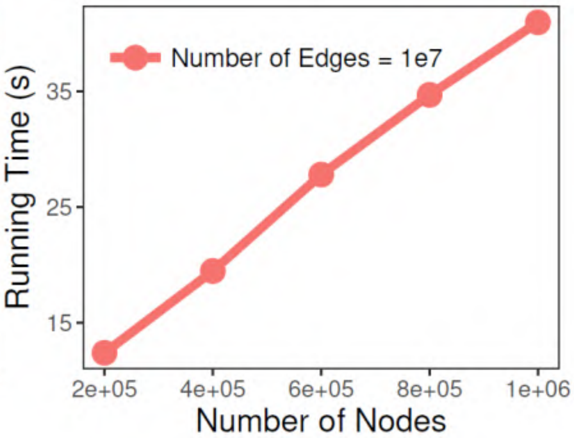
- Parameter analysis:
  - Effectiveness of preserving high-order proximity





  - Scalability





<4 minutes for network with 1 million nodes, 100 million edges with one PC

Experiments on a Billion-scale Network

- Experimental results on WeChat
  - 250 millions nodes, 4.8 billion edges
  - Network Reconstruction

Method	AUC
RandNE	**0.989**
Common Neighbors	0.783
Adamic Adar	0.783
Random	0.500

  - Dynamic link prediction

Table 3: AUC scores of dynamic link prediction on WeChat.

Observed Edges	30%	40%	50%	60%	70%
RandNE-D	**0.646**	**0.689**	**0.726**	**0.756**	**0.780**
RandNE-R	**0.646**	**0.689**	**0.726**	**0.756**	**0.780**
Common Neighbors	0.575	0.611	0.647	0.681	0.712
Adamic Adar	0.575	0.611	0.647	0.681	0.712
Random	0.500	0.500	0.500	0.500	0.500

Better results and no error accumulation!

Experimental Results

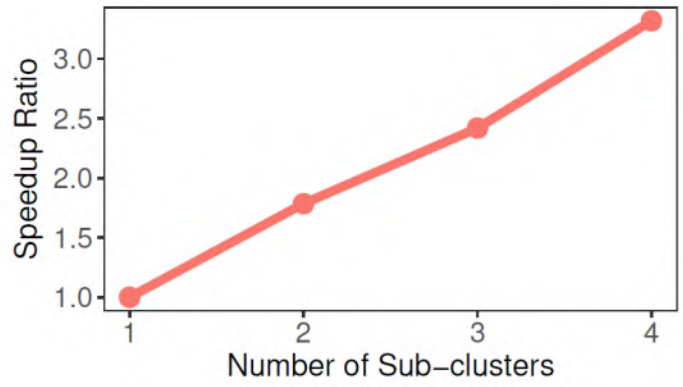
- Running time and acceleration ratio

Table 4: The running time of our method via distributed computing.

Number of Computing Nodes	4	8	12	16
Running Time(s)	82157	46029	33965	24757

<7 hours!

- Practical running time for real billion-scale networks



Support distributed computing

Conclusion

- RandNE: a billion-scale network embedding method
  - Based on iterative random projection to preserve high-order proximities
  - Much more computationally efficient
  - Distributed algorithm
  - Handle dynamic networks
- Experimental results on moderate-scale networks
  - At least one order of magnitude faster
  - Better or comparable performance
  - Linear scalability
- Experiments on WeChat, a real billion-scale network
  - Better results in network reconstruction and link prediction
  - No error accumulation
  - Linear acceleration ratio

Thanks!

Ziwei Zhang, Tsinghua University

zw-zhang16@mails.tsinghua.edu.cn

http://zw-zhang.github.io/

http://nrl.thumedialab.com/