# Scaling Attributed Network Embedding to Massive Graphs

Renchi Yang, Jieming Shi, Xiaokui Xiao,
Yin Yang, Juncheng Liu, Sourav S. Bhowmick

# Outline

- Problem Definition & Applications
- Existing Work & Limitations
- Basic Idea & Challenges
- Random Walks & Affinity Measure
- Objective Function & Solution
- Parallel Implementations
- Experiments

# Attributed Network

- **Input**: a graph $G$, where each node $u$ has some attributes from a set $R$, e.g., a social network user has an attribute “gender”


Mark Zuckerberg

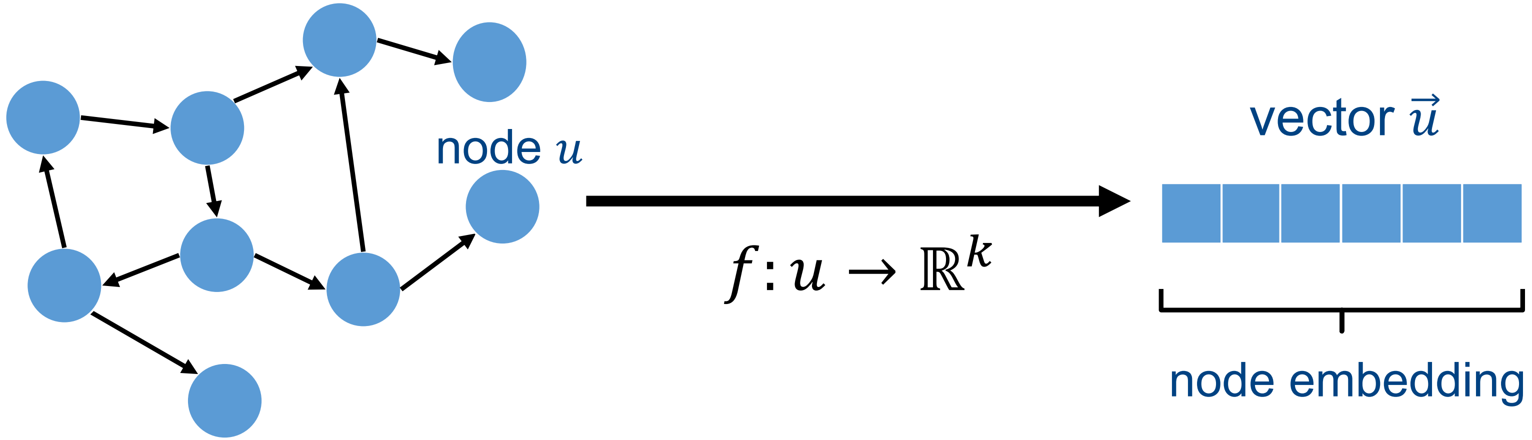
| |
|---|
| Gender: M |
| Country: USA |
| Age: 37 |
| #fans: 116M |

| |
|---|
| M: 1 |
| F: 0 |
| Other: 0 |

- Each node-attribute pair $(u, r)$ has a weight $w(u, r)$ that indicates the strength of association

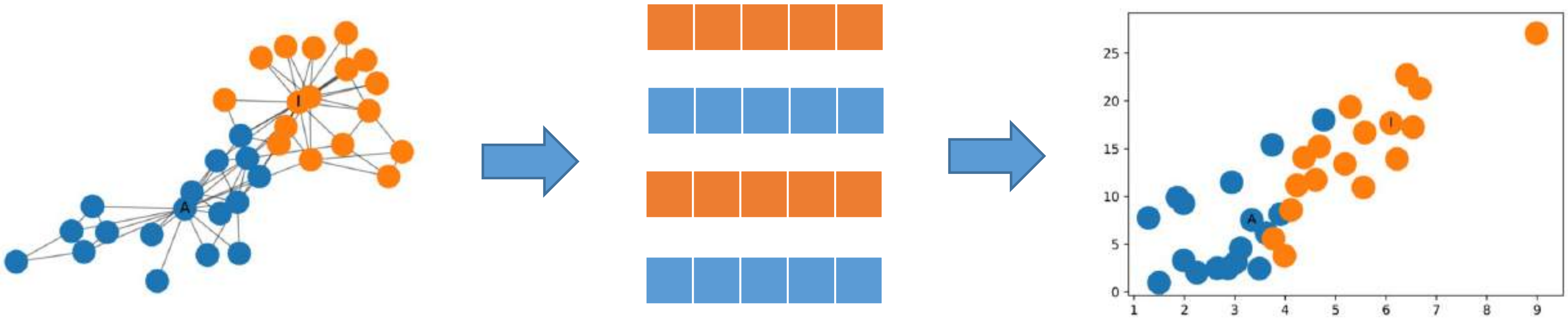
# Attributed Network Embedding (ANE)

- **Objective**: Map each node to an embedding vector, which can then be used as input to downstream machine learning tasks



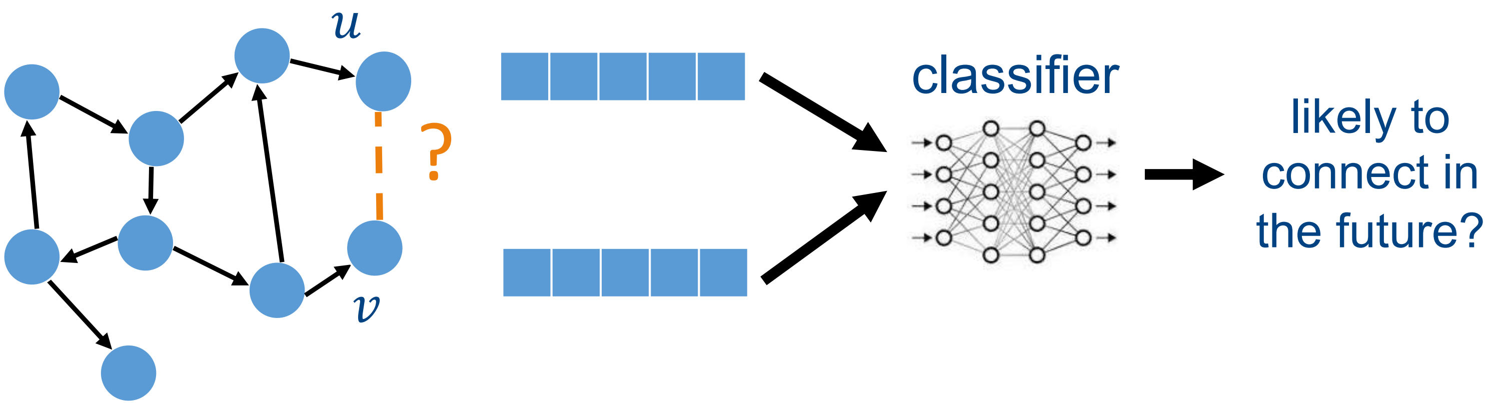
# Applications

- Node classification
  - user tagging in social networks
  - fraud detection in financial networks
  - cancer biomarkers identification in biological networks



# Applications

- Link Prediction
  - image/video recommendation in Pinterest [Ying KDD'18]
  - product recommendation in Alibaba [Cen KDD'19]



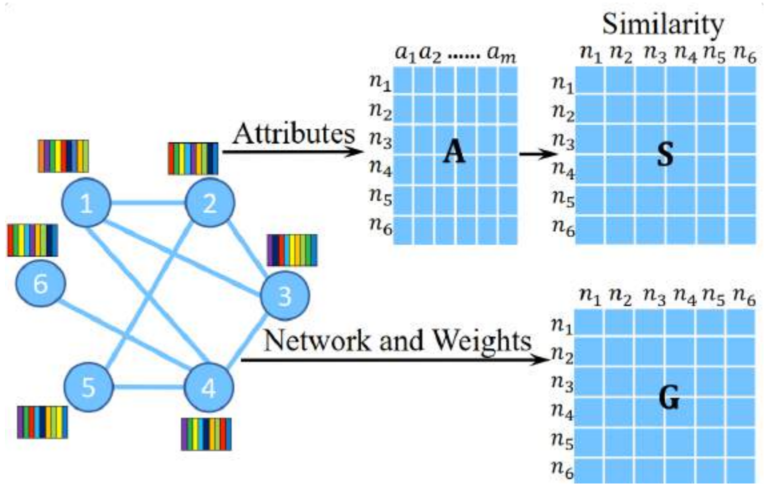


Ying et al. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. KDD'18.

Cen et al. Representation Learning for Attributed Multiplex Heterogeneous Network. KDD'19.

# Existing work

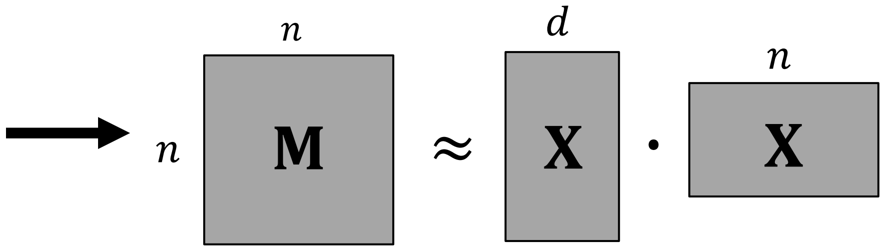
- Matrix Factorization-based methods




attributed graph $G$

Affinity

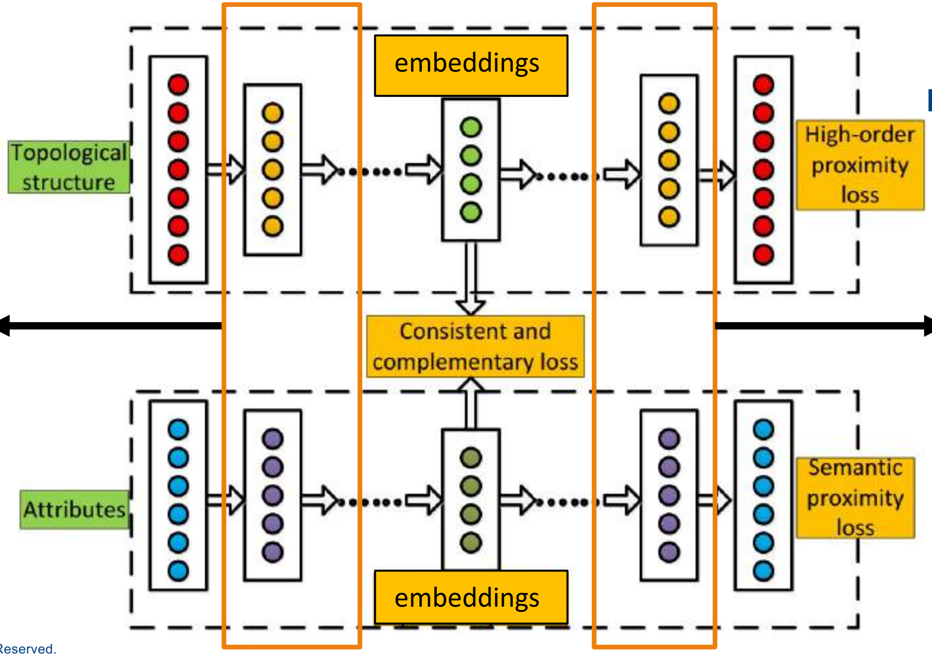
Factorize the combined matrix of $\mathbf{S}$ and $\mathbf{G}$



# Existing work

- Neural networks-based methods

**Encoders**: neural networks that compress data into embeddings

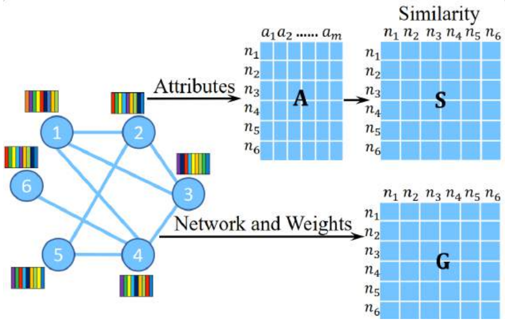


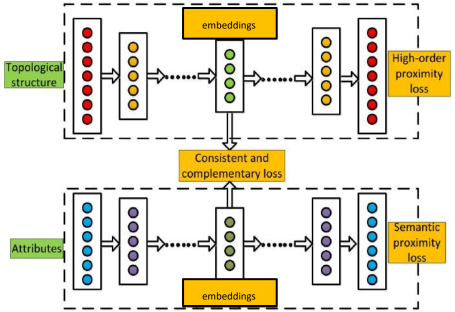

**Minimize reconstruction loss**

**Decoders**: neural networks that reconstruct input data from the embeddings

# Limitations

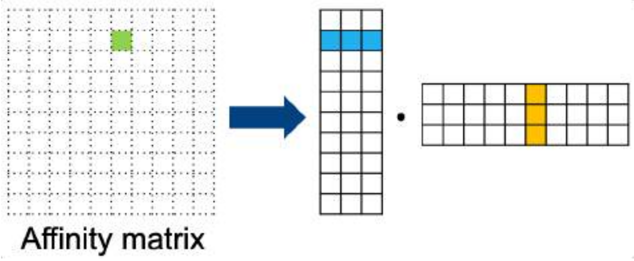
- Existing ANE solutions
  - generate effective embeddings based on expensive learning techniques (e.g., auto-encoders). This makes them difficult to handle large graphs.
  - are more efficient but produce low-quality embeddings.



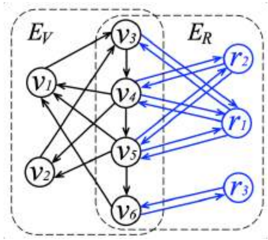
# Basic idea & challenges

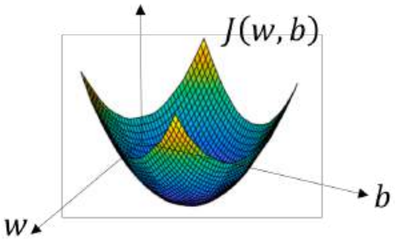
- Objective: Construct high-quality embeddings without significant computation cost
- Basic idea: formulate ANE as a new matrix factorization problem
- Challenges
  - Two types of affinity to capture: node-node & node-attribute
    - How to model these affinities?
    - How to compute & store these affinities? ($O(n^2)$ space, $n$=#nodes)
  - Two affinity matrices
    - Which one are we factorizing?
    - How to design the objective function?

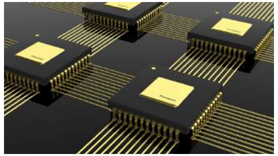


# Solution Overview: PANE

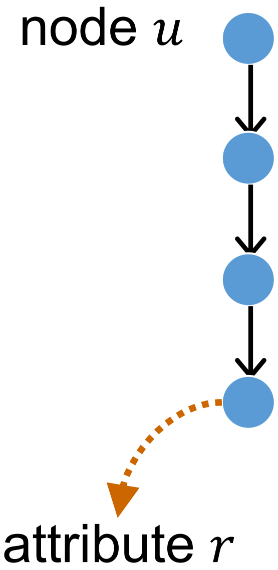
- Construct node-attribute affinity matrix $\mathbf{F}$ & attribute-node affinity matrix $\mathbf{B}$
  - forward & backward random walk models
  - indirectly model node-node affinity via $\mathbf{F}$ & $\mathbf{B}$
- Joint factorization of affinity matrices $\mathbf{F}$ & $\mathbf{B}$
  - initialize embeddings via singular value decomposition
- Parallelize PANE for higher efficiency
  - parallel singular value decomposition

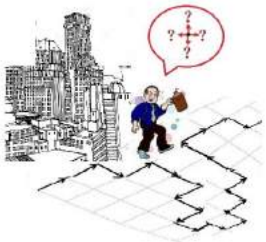


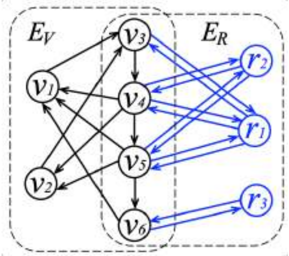



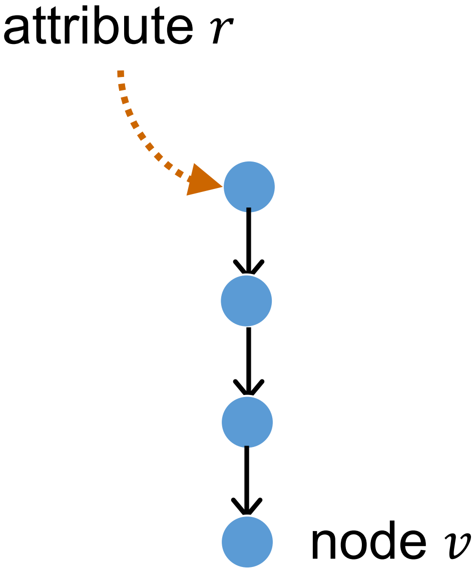



# Two Types of Random Walks





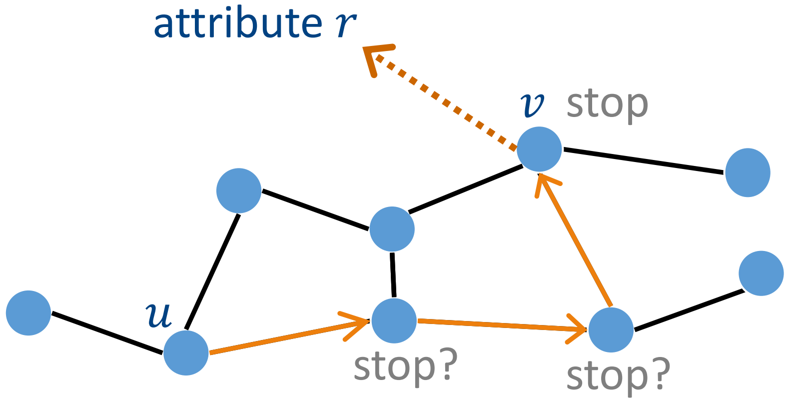




Forward: node-to-attribute

Backward: attribute-to-node

# Forward Random Walks

- Forward random walk from node $u$:
  - Start from $u$
  - At each step, stop with $\alpha$ probability
  - After stopping at a node $v$, pick an attribute $r$ with probability $\propto \mathrm{w}(v, r)$
- Intuition: it samples an attribute $r$ from the vicinity of $u$

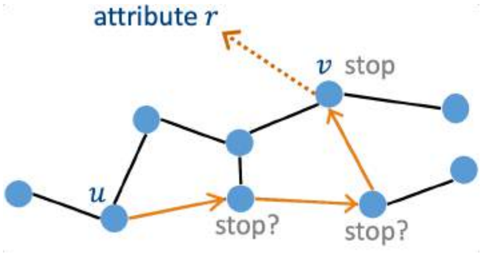



- $p_f(u, r)$ is the probability that a forward random walk from $u$ samples $r$ in the end

# Node-Attribute Affinity

- Our node-attribute affinity measure:

$$\mathbf{F}[u,r] = \log\left(\frac{n \cdot p_f(u,r)}{\sum_{v \in V} p_f(v,r)} + 1\right)$$



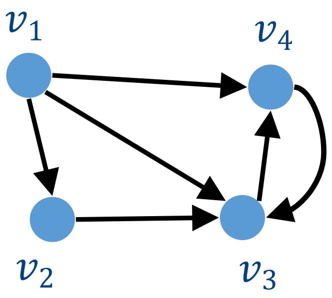

- Understand $\mathbf{F}[u,r]$ using PMI (pointwise mutual information) in information theory:
  - PMI $\log(\frac{\Pr(r|v)}{\Pr(r)})$ measures the co-occurrence of elements and in a collection of elements $S$
  - all random walks as $S$, $\sum_{v \in V} p_f(v,r)$ as $\Pr(r)$ and $p_f(u,r)$ as $\Pr(r|v)$
  - $\mathbf{F}[u,r]$ measures how frequently $u, r$ co-occur on all random walks to $r$
  - Word2vec is implicitly factorizing a PMI matrix to obtain word embeddings [Levy NeurIPS'14]



Levy et al. Neural word embedding as implicit matrix factorization. NeurIPS 2014.

# Computing node-attribute affinity

- Node-attribute affinity
  - $\mathbf{F}[u,r] = \log\left(\frac{n \cdot p_f(u,r)}{\sum_{v \in V} p_f(v,r)} + 1\right)$
  - $p_f(u,r) = \alpha \sum_{i=0}^{t} (1-\alpha)^i \mathbf{P}^i \cdot \mathbf{R}_\mathrm{r}[u,r]$
  - $O(mdt)$ time using power method

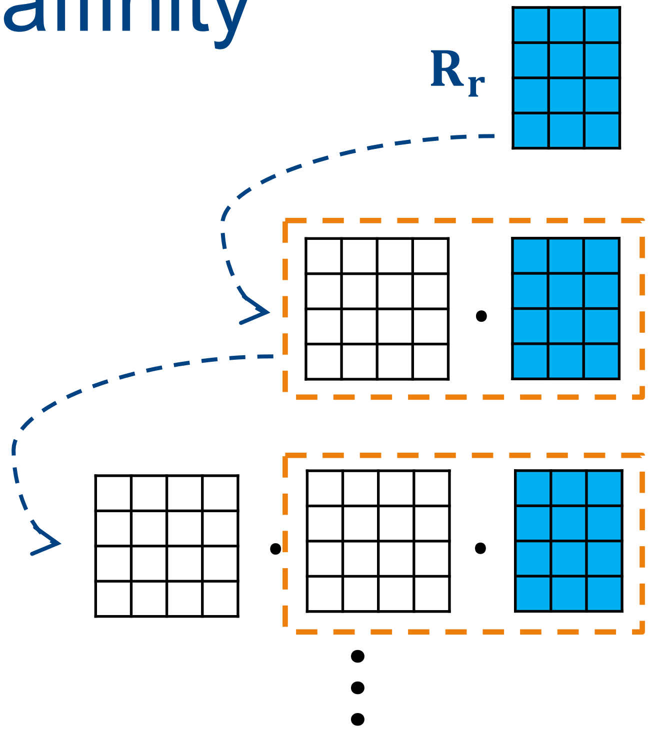



$$\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ v_2 & 0 & 0 & 1 & 0 \\ v_3 & 0 & 0 & 0 & 1 \\ v_4 & 0 & 0 & 1 & 0 \end{array}$$

**Transition matrix $\mathbf{P}$**

$$\begin{array}{c|ccc} & r_1 & r_2 & r_3 \\ \hline v_1 & 1/2 & 1/2 & 0 \\ v_2 & 1/3 & 0 & 2/3 \\ v_3 & 1/3 & 1/3 & 1/3 \\ v_4 & 1 & 0 & 0 \end{array}$$

**Row-normalized attribute matrix $\mathbf{R}_r$**



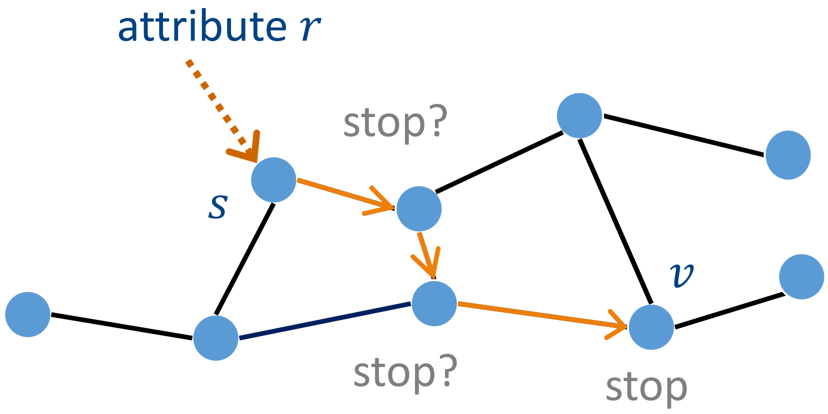

Weighted sum

# Backward RW & Attribute-Node Affinity

- Backward random walk from attribute $r$
  - Randomly pick a node $s$ with probability $\propto \mathrm{w}(s, r)$
  - Start a random walk from $s$
  - At each step, stop with $\alpha$ probability
  - Let $v$ be the stopping point of the walk
- Our attribute-to-node affinity measure:

$$\mathbf{B}[r, v] = \log\left(\frac{d \cdot p_b(r, v)}{\sum_{r' \in R} p_b(r', v)} + 1\right)$$

  - where $R$ is the set of all attributes,
  - $p_b(r, v)$ is the probability that a backward random walk from $r$ samples $v$ in the end



# Why node-attribute & attribute-node affinities

- We can indirectly model node-node affinities with much less space using node-attribute + attribute-node affinities


- Intuition:
  - Consider a forward random walk from $u$ to $r$ & a backward random walk from $r$ to $v$
  - They form an extended walk from $u$ to $v$, in which we "teleport" through a virtual connection by $r$
  - Such random walks could be combined to model node-node affinity

# Capturing node-node affinity

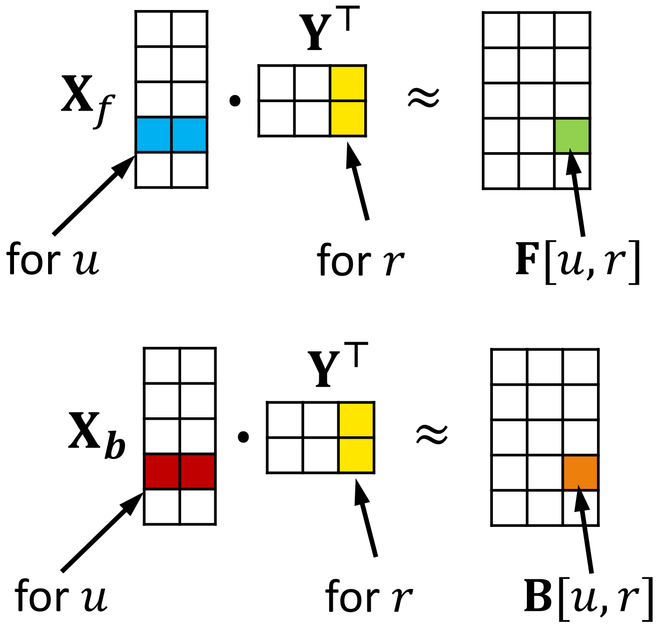
- Our node-node affinity can be indirectly constructed via:

$$p(u,v) = \sum_{r \in R} \mathbf{F}[u,r] \cdot \mathbf{B}[r,v]$$


- Thus, we do not need an $n \times n$ node-node affinity matrix explicitly
  - space overhead: $O(n^2) \rightarrow O(nd), d \ll n$
  - $d$ = #attributes, $n$ = #nodes

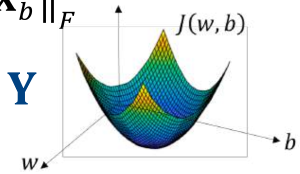
# Objective function

- We construct
  - two embedding matrices $\mathbf{X}_f, \mathbf{X}_b \in \mathbb{R}^{n \times k}$ for the nodes, and
  - one embedding matrix $\mathbf{Y} \in \mathbb{R}^{d \times k}$ for attributes
- Optimization objective:
  - $\min_{\mathbf{X}_f, \mathbf{Y}, \mathbf{X}_b} \left\| \mathbf{F} - \mathbf{X}_f \mathbf{Y}^{\top} \right\|_F^2 + \left\| \mathbf{B} - \mathbf{X}_b \mathbf{Y}^{\top} \right\|_F^2$
  - $\mathbf{X}_f \cdot \mathbf{Y}^{\top} \approx \mathbf{F}$, to capture node-attribute affinity
  - $\mathbf{X}_b \mathbf{Y}^{\top} \approx \mathbf{B}$, to capture attribute-node affinity

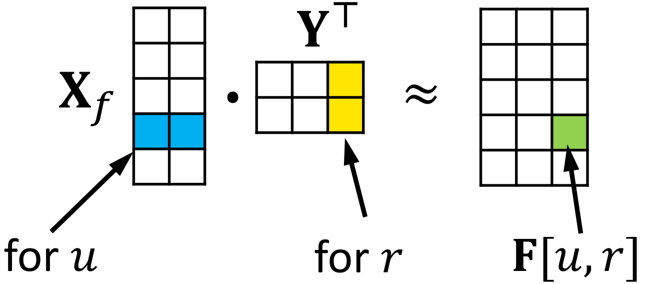


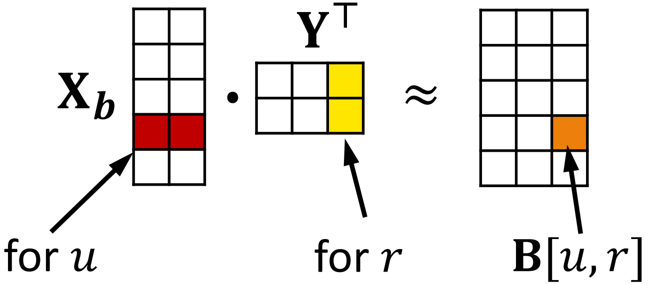
# Solving the optimization objective

- Optimization objective:
  - $\min_{\mathbf{X}_f,\mathbf{Y},\mathbf{X}_b} \left\|\mathbf{F} - \mathbf{X}_f\mathbf{Y}^{\mathsf{T}}\right\|_F^2 + \left\|\mathbf{B} - \mathbf{Y}\cdot\mathbf{X}_b^{\mathsf{T}}\right\|_F^2$




- We can obtain $\mathbf{X}_f$, $\mathbf{X}_b$, and $\mathbf{Y}$ using gradient descent
  - a large number of iterations are required till convergence
  - jointly updating $\mathbf{X}_f$, $\mathbf{X}_b$ and $\mathbf{Y}$ involves intensive computation
- Our idea for efficiency:
  - find a good initialization for $\mathbf{X}_f$, $\mathbf{X}_b$ and $\mathbf{Y}$ based on singular value decomposition (SVD)



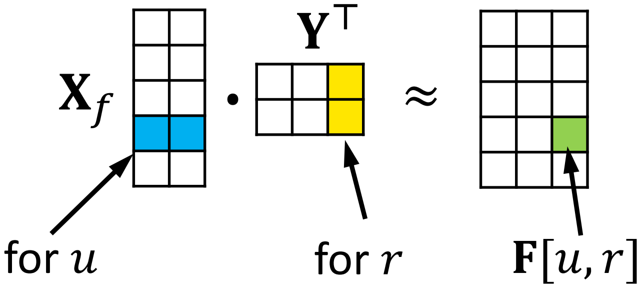



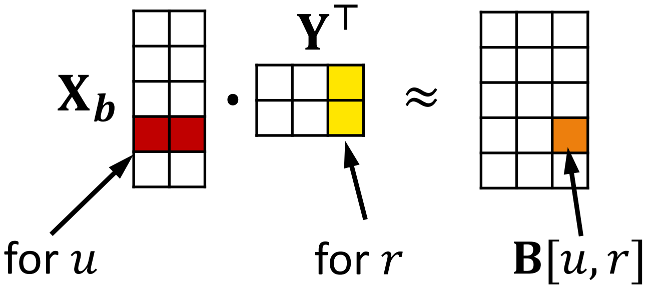
# Initialize embeddings via SVD

- Initial solution for the first part via randomized SVD

$$\mathbf{F} \approx \underbrace{\boxed{\mathbf{U} \cdot \boldsymbol{\Sigma}}}_{\mathbf{X}_f} \cdot \underbrace{\boxed{\mathbf{V}}}_{\mathbf{Y}}{}^{\mathrm{T}}$$

- Initial solution for the second part
  - approximate singular vectors $\mathbf{Y}$=$\mathbf{V}$ is semi-unitary, i.e., $\mathbf{Y}^{\mathrm{T}}\mathbf{Y} = \mathbf{I}$
  - Intuitively, if we want $\mathbf{X}_b\mathbf{Y}^{\mathrm{T}} = \mathbf{B}$, $\mathbf{X}_b = \boxed{\mathbf{X}_b\mathbf{Y}^{\mathrm{T}}}\mathbf{Y} = \boxed{\mathbf{B}} \cdot \mathbf{Y}$

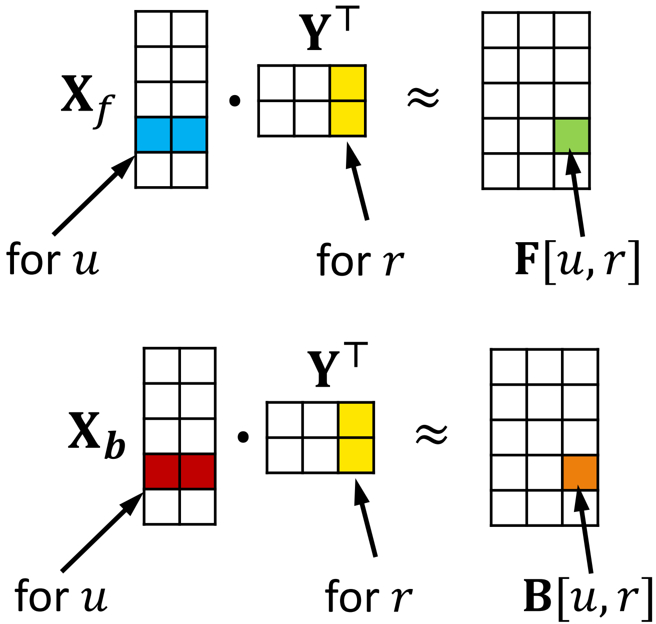


# Space & time complexities

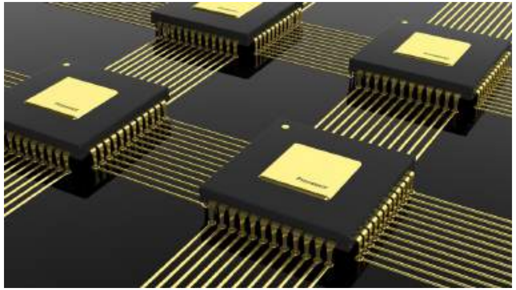
- Time complexity:
  - $O(mdt + ndkt)$
  - $t$ = #iterations of gradient descent ($t = 5$ in our experiments)
  - $k$ = the embedding size
  - $m$ = #edges
  - $n$ = #nodes
  - $d$ = #attributes

- Space complexity
  - $O(m + nd + nk)$



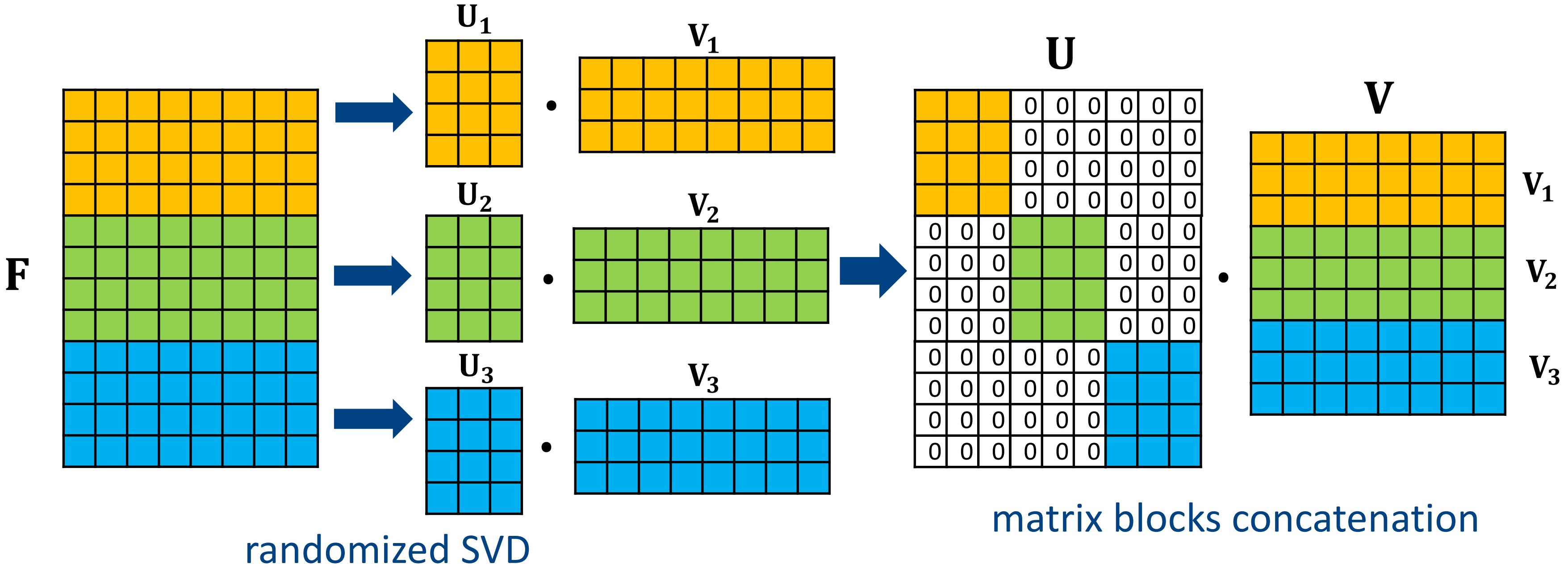
# Parallel implementation of PANE

- Parallel computation of $\mathbf{F}$ and $\mathbf{B}$
  - Matrix multiplications can be easily parallelized
- Parallel gradient descent. In each iteration,
  - $\mathbf{X}_f[u]\ \forall u \in V$ or $\mathbf{X}_b[v]\ \forall v \in V$ can be updated independently
  - $\mathbf{Y}[r]$ for each attribute $r$ can be updated independently
- Parallel SVD over $\mathbf{F}$
  - The SVD over $\mathbf{F} \neq$ direct SVD over each matrix blocks of $\mathbf{F}$
  - How to perform SVD over $\mathbf{F}$ in parallel?



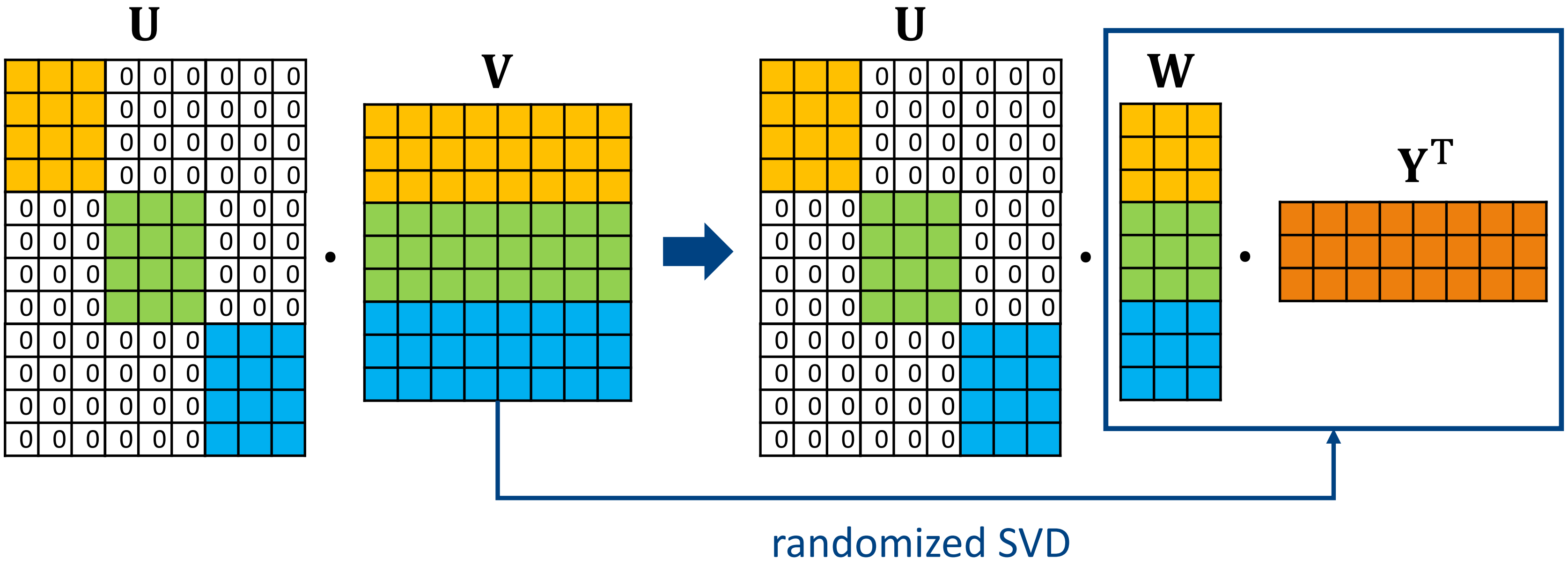
# Parallel SVD

- Given $\mathbf{F} \in \mathbb{R}^{12\times 8}$, we want $\mathbf{X}_f \in \mathbb{R}^{12\times 3}$and $\mathbf{Y} \in \mathbb{R}^{8\times 3}$s.t. $\mathbf{X}_f \cdot \mathbf{Y}^\top \approx \mathbf{F}$



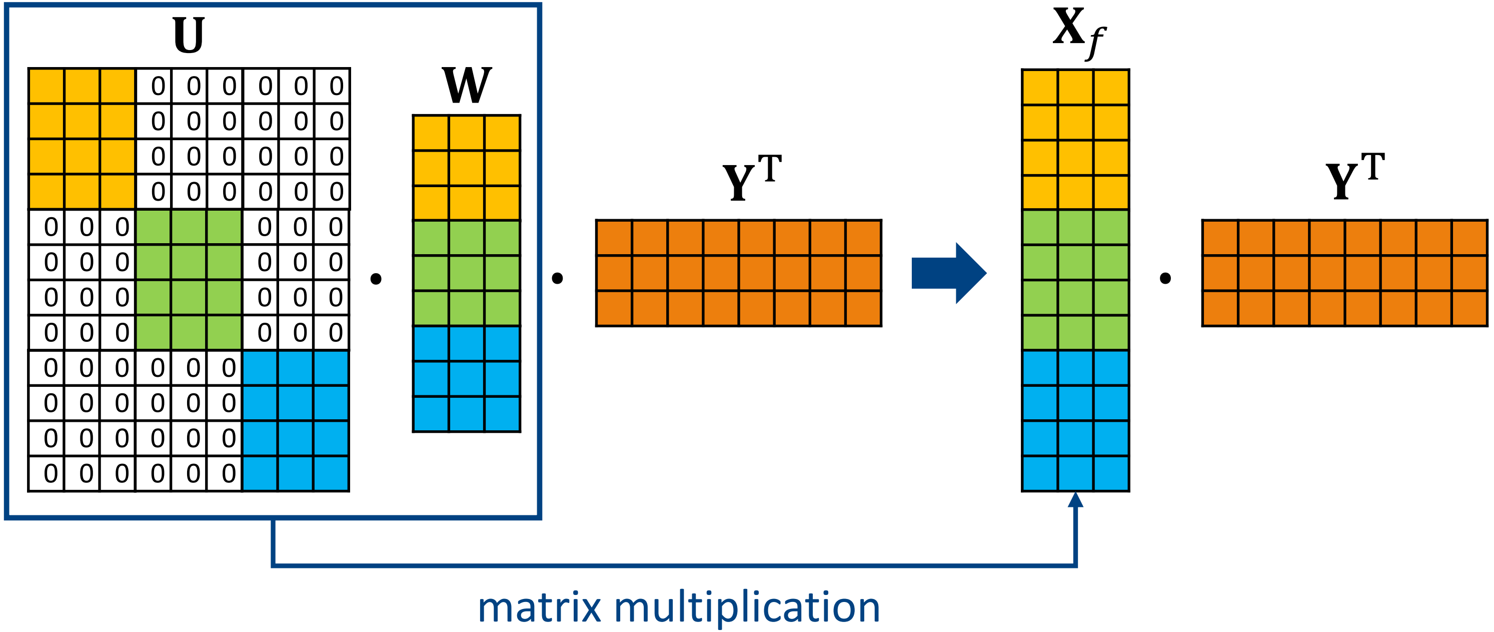
# Parallel SVD

- Given $\mathbf{F} \in \mathbb{R}^{12\times 8}$, we want $\mathbf{X}_f \in \mathbb{R}^{12\times 3}$ and $\mathbf{Y} \in \mathbb{R}^{8\times 3}$ s.t. $\mathbf{X}_f \cdot \mathbf{Y}^\top \approx \mathbf{F}$



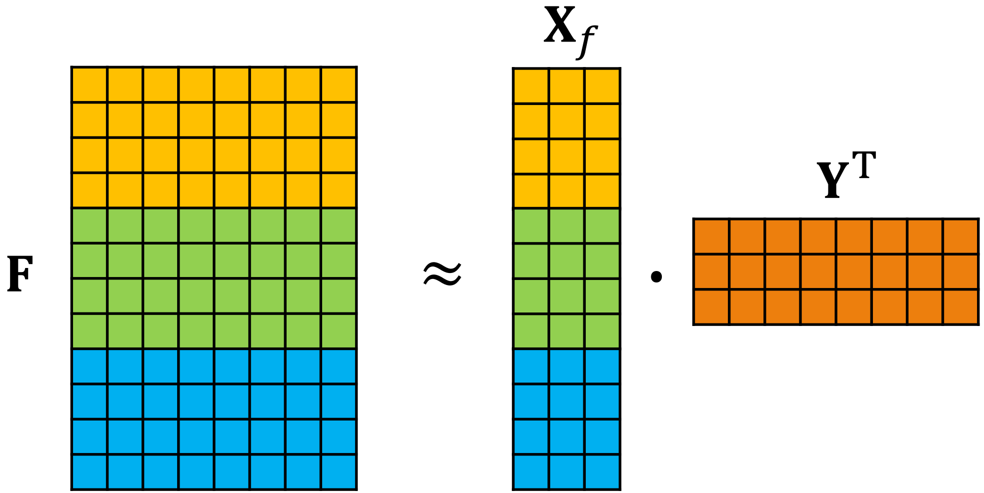
# Parallel SVD

- Given $\mathbf{F} \in \mathbb{R}^{12\times8}$, we want $\mathbf{X}_f \in \mathbb{R}^{12\times3}$and $\mathbf{Y} \in \mathbb{R}^{8\times3}$s.t. $\mathbf{X}_f \cdot \mathbf{Y}^\top \approx \mathbf{F}$



# Parallel SVD

- Given $\mathbf{F} \in \mathbb{R}^{12\times 8}$, we want $\mathbf{X}_f \in \mathbb{R}^{12\times 3}$and $\mathbf{Y} \in \mathbb{R}^{8\times 3}$s.t. $\mathbf{X}_f \cdot \mathbf{Y}^\top \approx \mathbf{F}$



# Experiments: Datasets

| Name | # of nodes | # of edges | # of distinct attributes | # of attributes per node | # of distinct labels |
|---|---|---|---|---|---|
| Cora | 2.7k | 5.4k | 1.4k | 18.2 | 7 |
| Citeseer | 3.3k | 4.7k | 3.7k | 31.9 | 6 |
| Facebook | 4k | 88.2k | 1.3k | 8.3 | 193 |
| Pubmed | 19.7k | 44.3k | 0.5k | 50.2 | 3 |
| Flickr | 7.6k | 479.5k | 12.1k | 24.0 | 9 |
| Google+ | 107.6k | 13.7M | 15.9k | 2793.7 | 468 |
| TWeibo | 2.3M | 50.7M | 1.7k | 7.3 | 8 |
| MAG | 59.3M | 978.2M | 2.0k | 7.3 | 100 |

# Experiments: Competitors

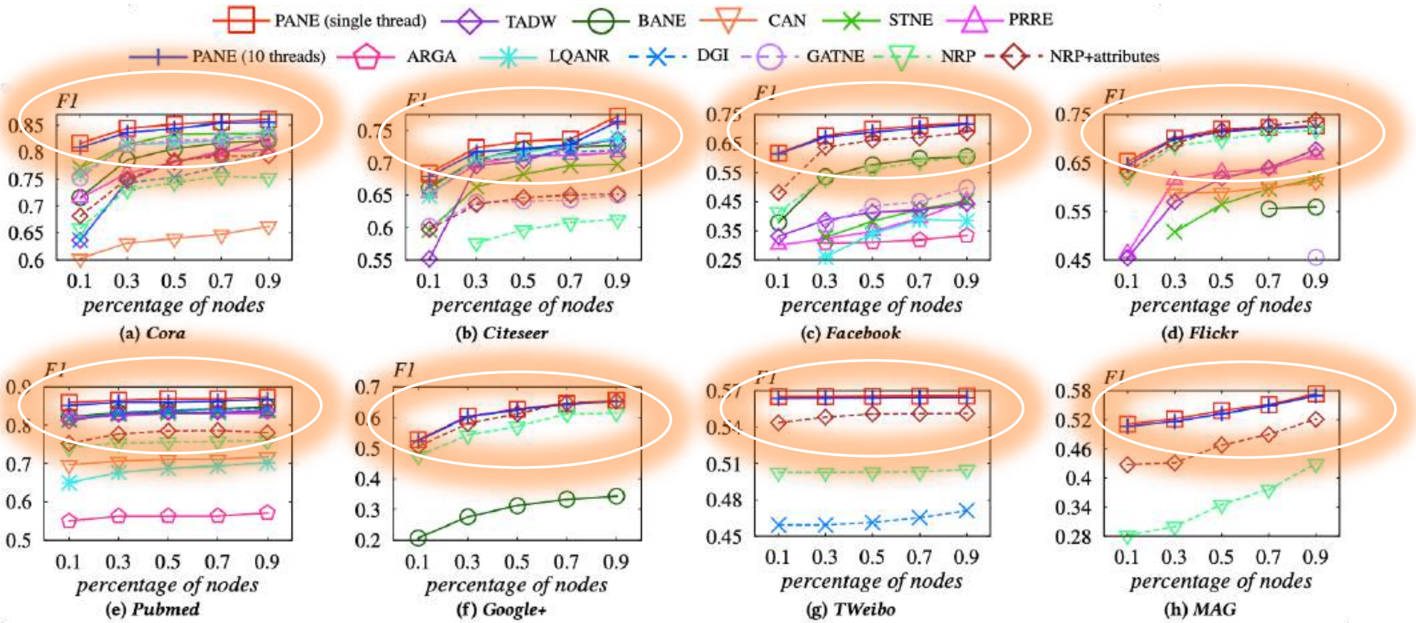
- PANE: Our solution for attributed graphs
- Default embedding dimensionality: $k = 128$
- CPU: Intel Xeon 2.2GHz

- 6 neural-based methods
  - STNE [KDD 2018]
  - ARGA [IJCAI 2018]
  - LQANR [IJCAI 2019]
  - CAN [WSDM 2019]
  - DGI [ICLR 2019]
  - GATNE [KDD 2019]

- 3 factorization-based methods
  - TADW [IJCAI 2015]
  - BANE [ICDM 2018]
  - NRP [VLDB 2020]

- 1 other method
  - PRRE [CIKM 2018]

# Results: Node Classification

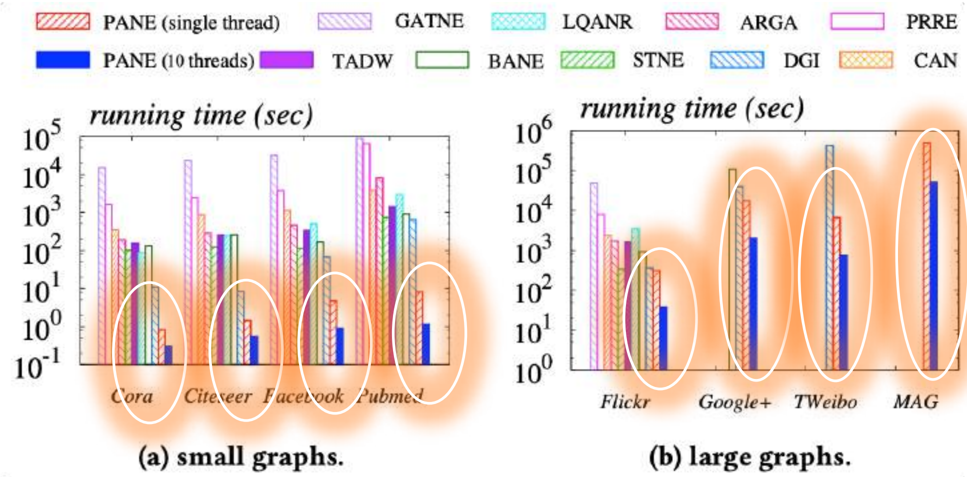



- Percentage of nodes used for training: 10% ~ 90%
- PANE vs. SOTA: improvements of 3.4%-17.2% in terms of F1 measure

# Results: Link Prediction

| Method | Area Under Curve (AUC) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | *Cora* | *Citeseer* | *Pubmed* | *Facebook* | *Flickr* | *Google+* | *TWeibo* | *MAG* |
| NRP | 0.796 | 0.86 | 0.87 | 0.969 | 0.909 | 0.989 | 0.967 | 0.915 |
| GATNE | 0.791 | 0.687 | 0.745 | 0.961 | 0.805 | - | - | - |
| TADW | 0.829 | 0.895 | 0.904 | 0.752 | 0.573 | - | - | - |
| ARGA | 0.64 | 0.637 | 0.623 | 0.71 | 0.676 | - | - | - |
| BANE | 0.875 | 0.899 | 0.919 | 0.796 | 0.64 | 0.56 | - | - |
| PRRE | 0.879 | 0.895 | 0.887 | 0.899 | 0.789 | - | - | - |
| STNE | 0.808 | 0.71 | 0.789 | 0.962 | 0.638 | - | - | - |
| CAN | 0.663 | 0.734 | 0.734 | 0.714 | 0.5 | - | - | - |
| DGI | 0.51 | 0.5 | 0.73 | 0.711 | 0.769 | 0.792 | 0.721 | - |
| LQANR | 0.886 | 0.916 | 0.904 | 0.951 | 0.824 | - | - | - |
| PANE (single thread) | 0.933 | 0.932 | 0.985 | 0.982 | 0.929 | 0.987 | 0.976 | 0.96 |
| PANE (10 threads) | 0.929 | 0.929 | 0.985 | 0.98 | 0.927 | 0.984 | 0.975 | 0.958 |

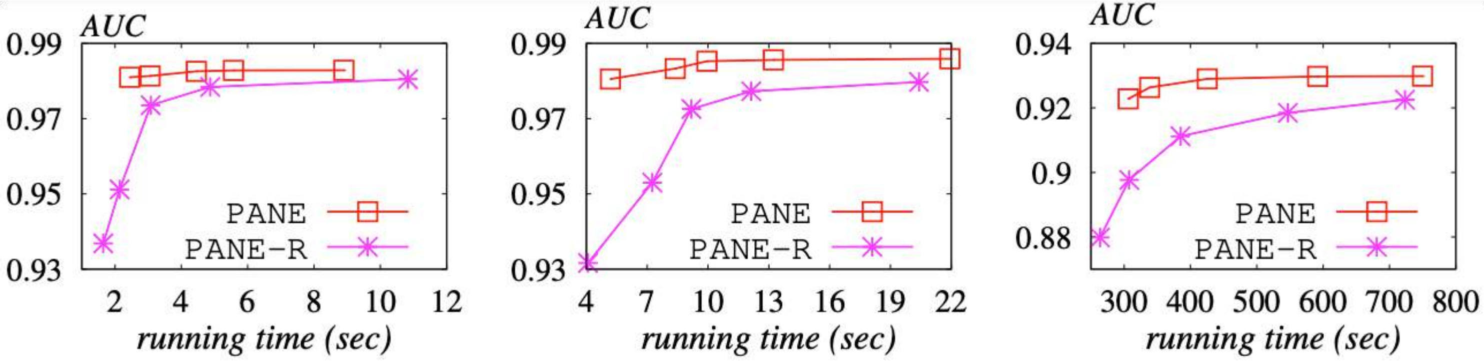
# Results: Efficiency




- Compared to the state of the art, PANE is orders of magnitude faster
- On the MAG dataset with 0.98 billion edges, PANE can terminate within 12 hours using 10 CPU cores (Intel Xeon 2.2GHz)

# Effectiveness of SVD-based initializations

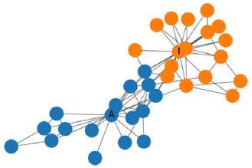
**PANE-R**: the algorithm that uses random initializations
**PANE**: the algorithm that uses SVD-based initializations

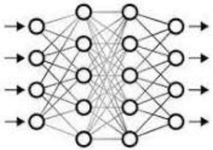


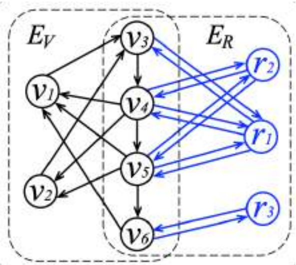

(a) *Facebook.* (b) *Pubmed.* (c) *Flickr.*

Link prediction performance vs. running time
when varying #iteration for the gradient descent from 1 to 20

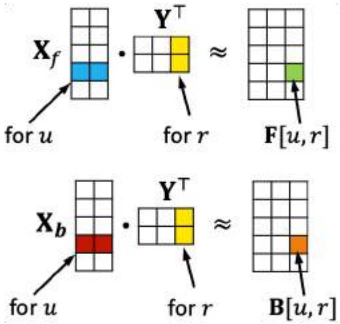
# THANK YOU

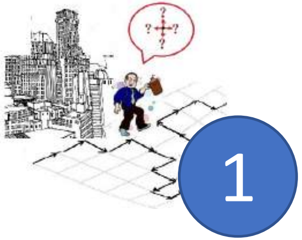





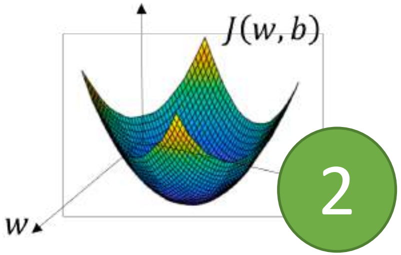




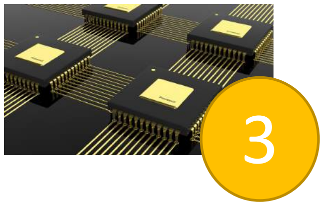



Random walks




Joint matrix factorization




Parallelization