

# Mining Networks

## Chapter 6 — Graph Embeddings

---

Bogumił Kamiński, Paweł Prałat, and François Théberge

Updated: 2026/01/22

*Department of Mathematics, Toronto Metropolitan University*  
File: DS8014-Chapter06

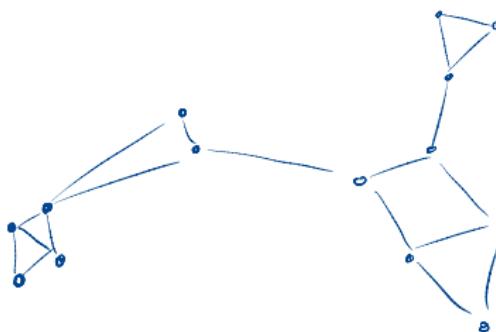


# Overview

1. Problem Formalization
2. Techniques
3. Unsupervised Benchmark
4. A Few Applications
5. Experiments

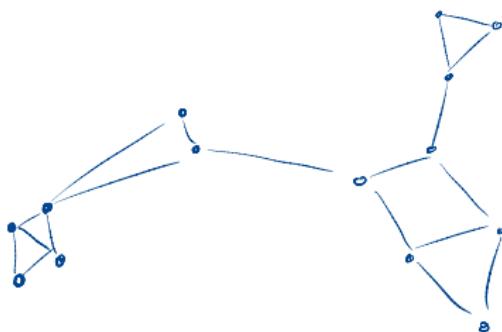
# Introduction

To **extract** useful **structural information** from graphs, it is convenient to **embed** it in a **geometric space** by assigning **coordinates** to each **node**.



# Introduction

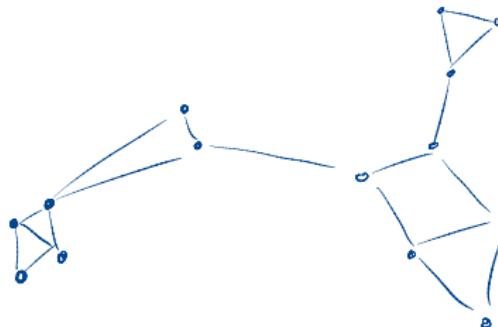
Classical embeddings: nearby nodes are more likely to share an edge than those far from each other, or are similar in some sense.



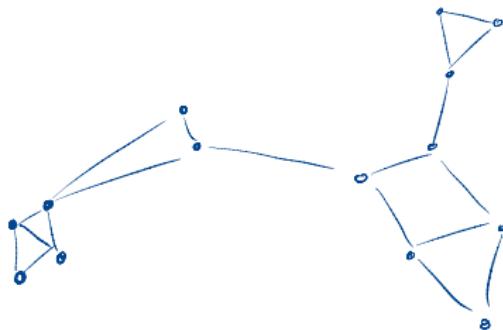
# Introduction

**Classical** embeddings: **nearby nodes** are more likely to **share an edge** than those far from each other, or **are similar** in some sense.

**Structural** embeddings: nodes with similar **structure** of their local **ego-nets** should be embedded close to each other.



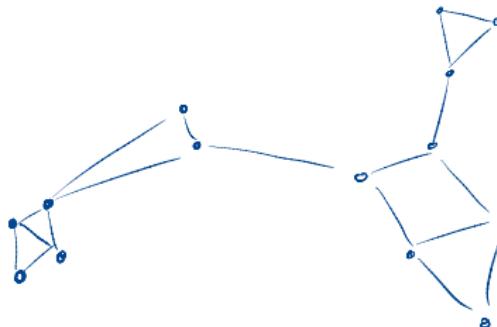
# Introduction



There are many **applications**, including:

- Node Classification (say, bot detection): **structural**,
- Node Clustering and Community Detection: **classical**,
- Link Prediction and Missing Links: **classical**,
- Visualization.

# Introduction



There are many **applications**, including:

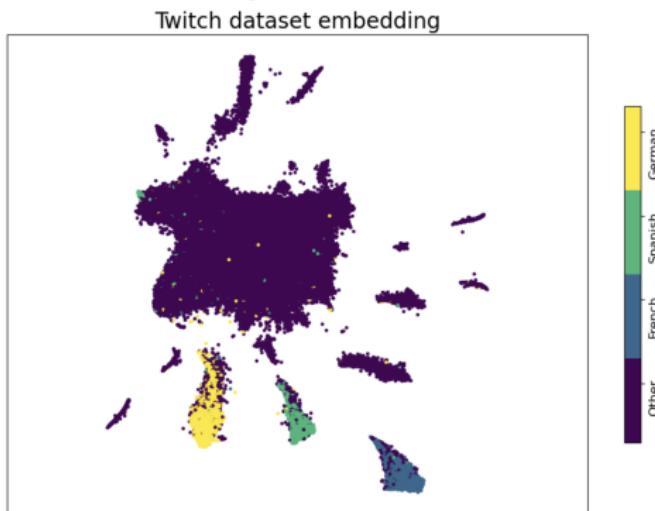
- Node Classification (say, bot detection): **structural**,
- Node Clustering and Community Detection: **classical**,
- Link Prediction and Missing Links: **classical**,
- Visualization.

**Drawback:** graph embedding often requires domain experts.

# Illustration

Twitch gamers social network:

- 168,114 nodes representing users
- 6,797,557 (undirected) edges — mutual followers



Node2Vec + UMAP

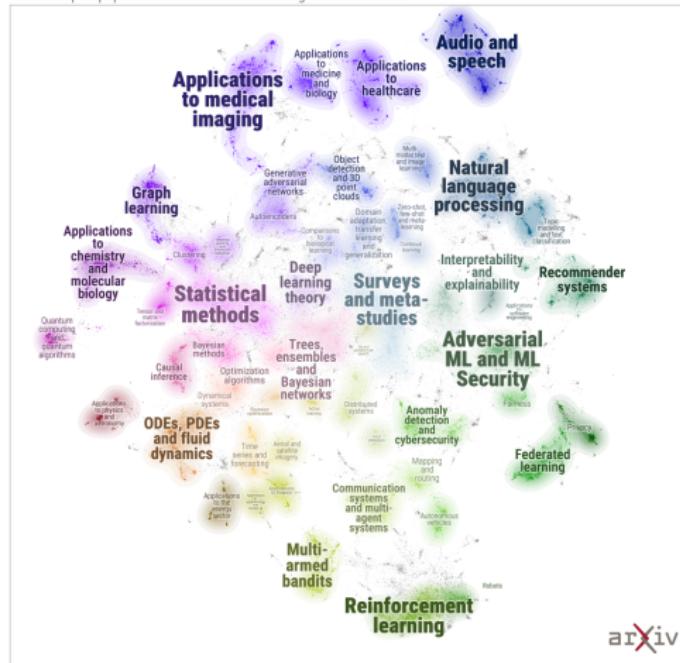
# Illustration

## ArXiv ML Landscape

(Example of DataMapPlot in action)

### ArXiv ML Landscape

A data map of papers from the Machine Learning section of ArXiv



arXiv

## Problem Formalization

# Embedding

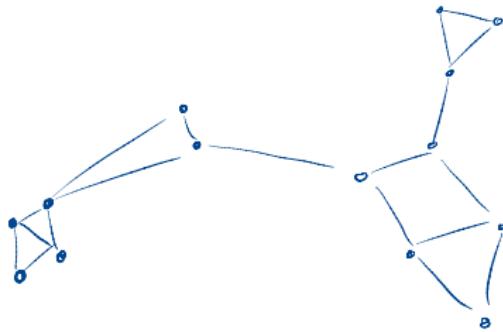
$G = (V, E)$  — weighted **undirected** graph on the set of nodes  
 $V = \{v_1, v_2, \dots, v_n\}$ . (We will discuss **directed** graphs later on.)

# Embedding

$G = (V, E)$  — weighted **undirected** graph on the set of nodes

$V = \{v_1, v_2, \dots, v_n\}$ . (We will discuss **directed** graphs later on.)

**Embedding** — function  $\mathcal{E}: V \rightarrow \mathbb{R}^k$ ;  $k$  is **much smaller than  $n$** .



# Algorithms Based on Random Walk

These methods are inspired by the **Word2Vec** algorithm for word embedding in **Natural Language Processing (NLP)**.

**Main idea:** words are known by the company they keep.

# Algorithms Based on Random Walk

These methods are inspired by the **Word2Vec** algorithm for word embedding in **Natural Language Processing (NLP)**.

**Main idea:** words are known by the company they keep.

Uses the model known as **SkipGram**. Consider a context window of size 5 (that is, we take  $\ell = 2$ ) and the sentence:

*“Graph embedding maps **nodes** to vector space”.*

# Algorithms Based on Random Walk

These methods are inspired by the **Word2Vec** algorithm for word embedding in **Natural Language Processing (NLP)**.

**Main idea:** words are known by the company they keep.

Uses the model known as **SkipGram**. Consider a context window of size 5 (that is, we take  $\ell = 2$ ) and the sentence:

“Graph *embedding* maps **nodes** to vector space”.

The model is **trained** to **predict** the words in *italics* given the word “**nodes**” as input.

# Algorithms Based on Random Walk

Similarly to HOPE, SkipGram associates two embeddings with each word:  $E_s$  ("source") and  $E_t$  ("target").

They try to capture relationships between the input words and the words that they aim to predict and vice versa.

# Algorithms Based on Random Walk

Similarly to HOPE, SkipGram associates two embeddings with each word:  $\mathbf{E}_s$  (“source”) and  $\mathbf{E}_t$  (“target”).

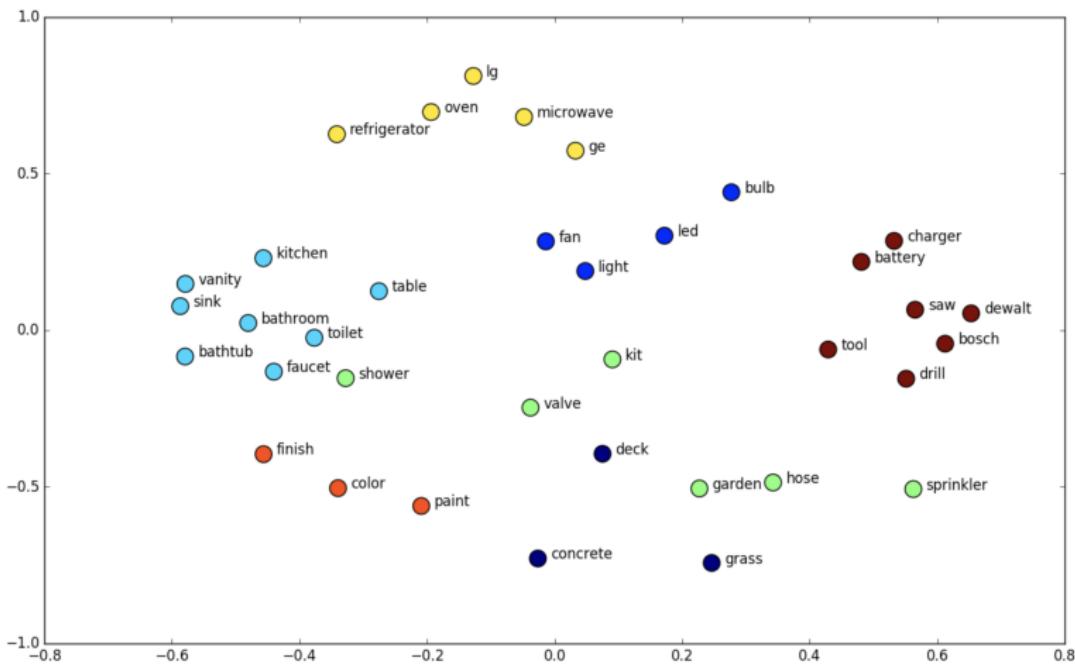
They try to capture relationships between the input words and the words that they aim to predict and vice versa.

Given a word  $i$ , the probability  $p_{i,j}$  that we see word  $j$  in its neighbourhood is approximated by the softmax function:

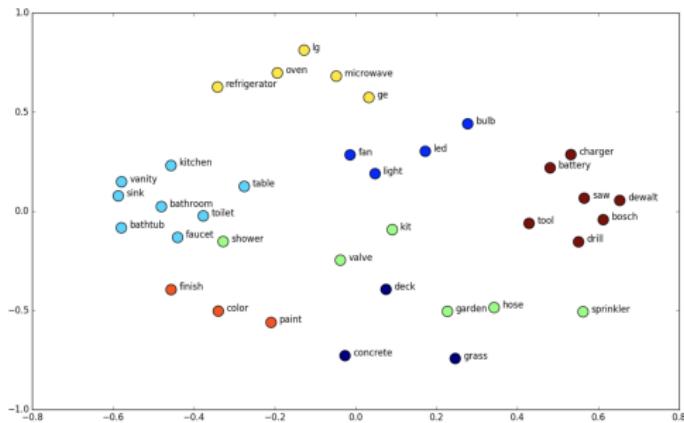
$$q_{i,j} = \frac{\exp(\mathbf{e}_{t,j}^T \mathbf{e}_{s,i})}{\sum_{\ell=1}^{|W|} \exp(\mathbf{e}_{t,\ell}^T \mathbf{e}_{s,i})}.$$

The model is trained using maximum likelihood estimation.

# Algorithms Based on Random Walk



# Algorithms Based on Random Walk



Unsupervised learning (no syntactic or semantic relationships among words are provided).

Yet, word embeddings seem to capture these relationships!

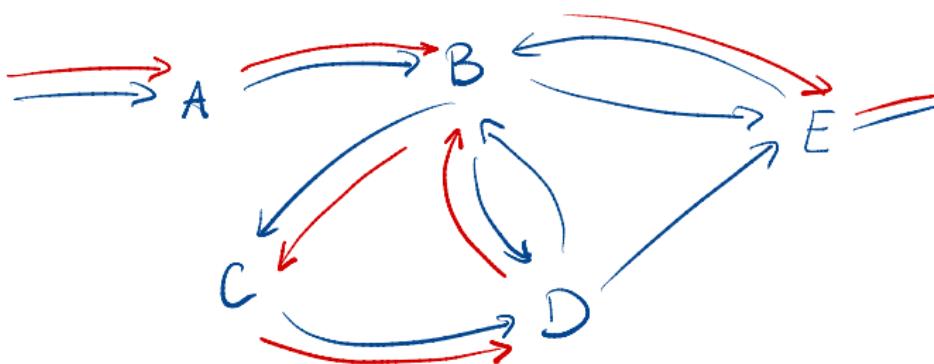
king - man + woman  $\approx$  queen

Paris - France + Germany  $\approx$  Berlin

# Algorithms Based on Random Walk

Generalization to graphs.

- the words are simply the nodes of a graph,
- sentences (sequences of nodes) are generated via random walks on a graph (exact procedure depends on the algorithm).



Walk : ... ABCD BE ...

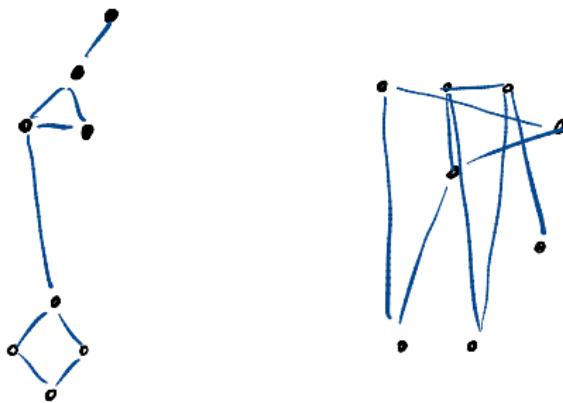
## Unsupervised Benchmark

---

# Main Goal

There are **many** embedding algorithms (techniques from linear algebra, random walks, or deep learning) and the list **grows**.

Results **vary** a lot.



**How can we evaluate these embeddings?** Which one is the best and should be used? Important question: **GIGO**.

# Big Picture

**Input:** Graph  $G = (V, E)$  on  $n$  nodes with the **degree distribution**  $\mathbf{w} = (w_1, \dots, w_n)$ .

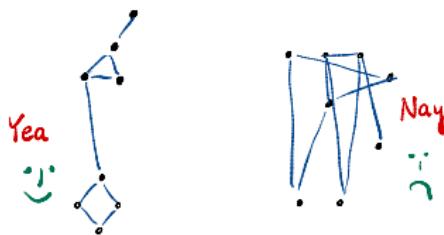
**Embedding** of nodes of  $V$ ,  $\mathcal{E} : V \rightarrow \mathbb{R}^k$  (typically many).

# Big Picture

**Input:** Graph  $G = (V, E)$  on  $n$  nodes with the **degree distribution**  $\mathbf{w} = (w_1, \dots, w_n)$ .

**Embedding** of nodes of  $V$ ,  $\mathcal{E} : V \rightarrow \mathbb{R}^k$  (typically many).

**Output:** two “divergence scores” (**global** and **local**) assigned to  $\mathcal{E}$  (smaller is better).



# Global Score

## Global score:

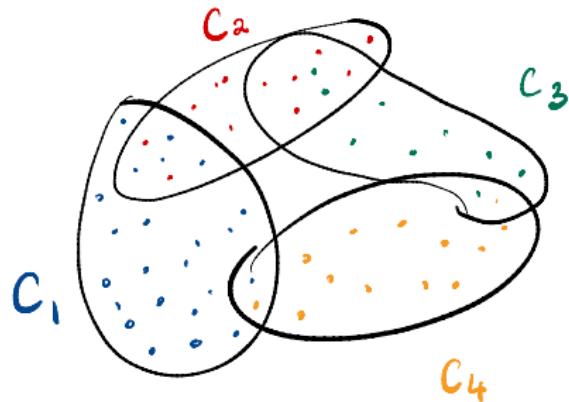
- looks at the network and the associated embeddings “from the distance”;
- evaluates the embeddings based on their ability to capture global properties of the network, namely, edge densities;

# Global Score

**Global** score:

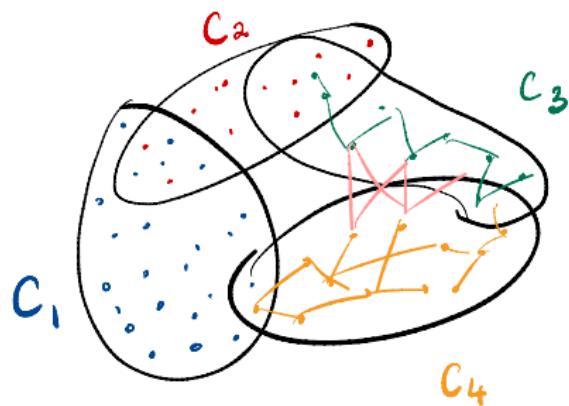
- looks at the network and the associated embeddings “from the distance”;
- evaluates the embeddings based on their ability to capture global properties of the network, namely, edge densities;
- **goodness-of-fit** test for the provided embedding, rather than simply checking its predictive power—its aim is similar to the well-known **Hosmer–Lemeshow** test for logistic regression (used frequently in **risk prediction** models).

# Global Score



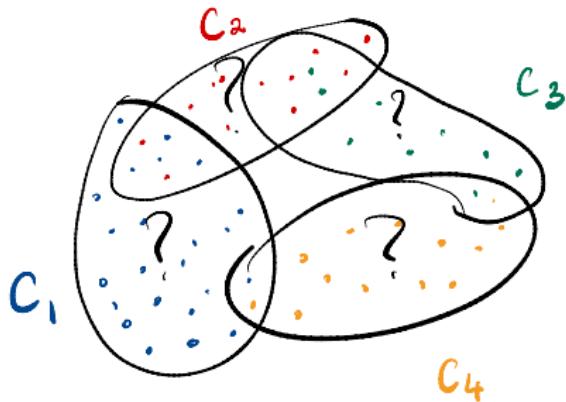
Step 1: Run some graph **clustering** algorithm to obtain a **partition** of  $V$  into  $\ell$  communities  $C_1, \dots, C_\ell$ .

# Global Score



Step 2: Compute  $c_{i,j}$  (including  $j = i$ ): proportion of edges with one endpoint in  $C_i$  and the other one in  $C_j$ .  
Note that we do **not** use  $\mathcal{E}$ .

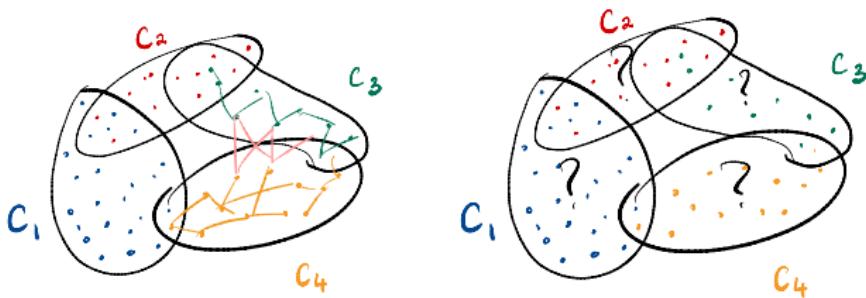
# Global Score



Step 3: Compute  $b_{i,j}$  (including  $j = i$ ): the **expected proportion of edges** with one endpoint in  $C_i$  and the other one in  $C_j$ , in the **geometric Chung-Lu model**  $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$ .

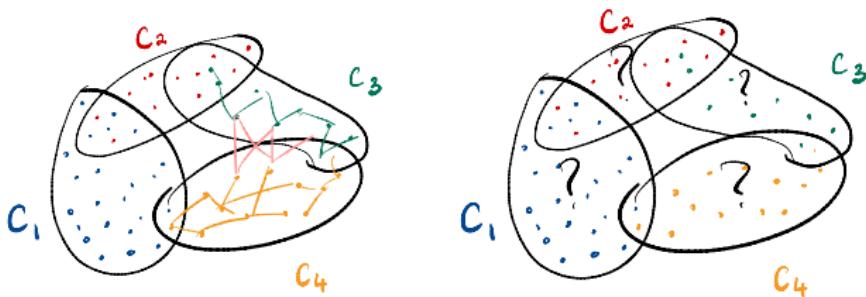
Note that we do **not** use  $G$ , only  $\mathbf{w}$  (on top of  $\mathcal{E}$  and parameter  $\alpha$ , the “**aversion**” for **long** links).

# Global Score



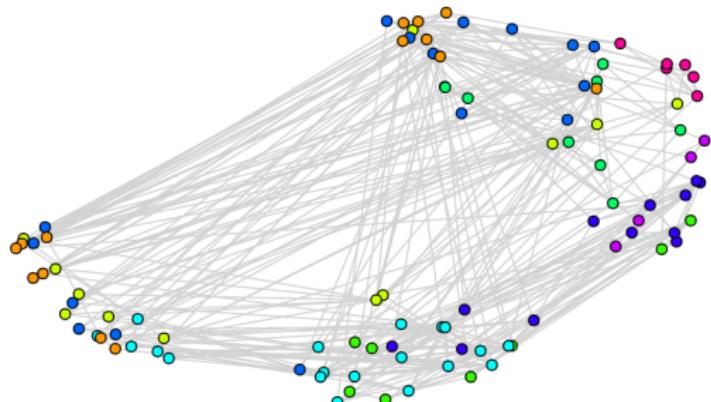
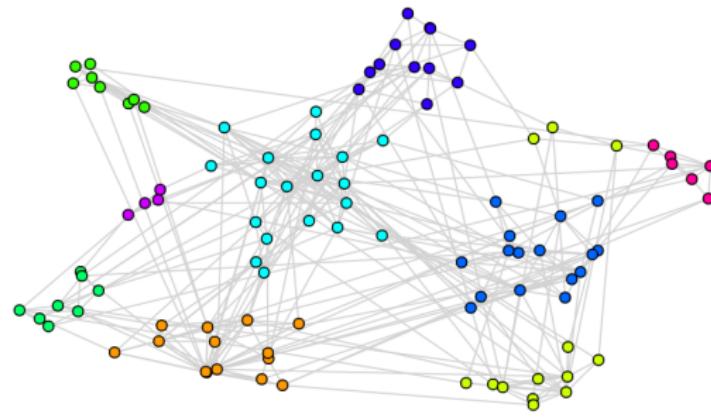
Step 4: Compute  $\Delta_\alpha$ : the **Jensen-Shannon divergence** between the two vectors (smoothed version of the **Kullback-Leibler** divergence we mentioned earlier).

# Global Score

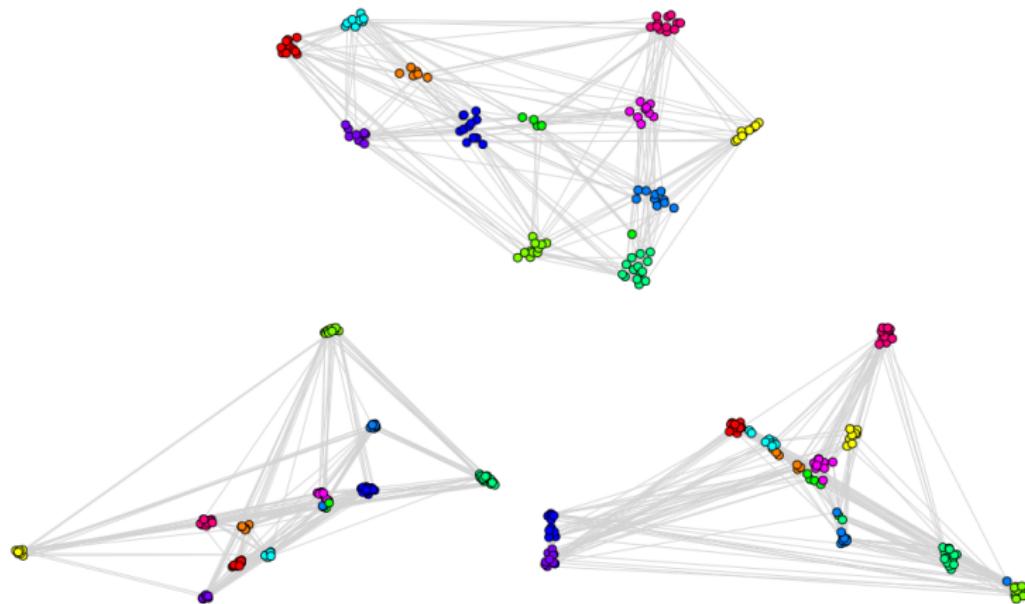


Step 5: Take the minimum  $\Delta_\alpha$  over various choices for parameter  $\alpha$  that measures the “aversion” for **long** links.  
It defines the **global** score.

# The worst and the best LFR graph ( $\mu = 0.35$ )



# The worst and the best College Football graph



# Local Score

**Local** score:

- provides a complementary test,
- looks at the network and embeddings “under the microscope”, trying to see if a local structure of a graph is well reflected by the associated embedding.

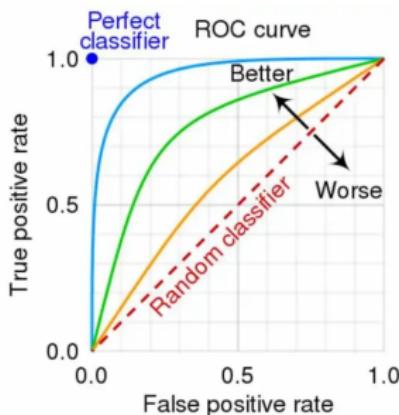
# Local Score

**Local** score:

- provides a complementary test,
- looks at the network and embeddings “under the microscope”, trying to see if a local structure of a graph is well reflected by the associated embedding.
- $S^+$  and  $S^-$ : edges and non-edges,
- $p(u, v)$ : the probability of an edge  $uv$  to be present in the **geometric Chung-Lu model**  $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$ ,
- build a binary **classifier** to see if  $p(u, v)$  can be used to distinguish edges from non-edges.

## Local Score

- The **ROC** (Receiver Operating Characteristic) is a curve showing the performance of a classification model at all classification thresholds.
  - y-axis:** **True Positive Rate (Sensitivity)** — the proportion of actual positive instances the model correctly identifies
  - x-axis:** **False Positive Rate (1-Specificity)** — the proportion of actual negative instances incorrectly classified as positive



## Local Score

- The **AUC** (area under the ROC curve) provides an aggregate measure of performance across all possible classification thresholds which can be interpreted as the probability that a randomly chosen positive sample is ranked higher than a negative sample:

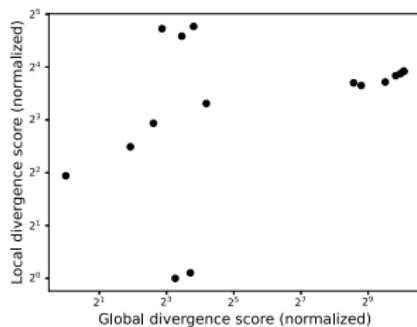
## Local Score

- The **AUC** (area under the ROC curve) provides an aggregate measure of performance across all possible classification thresholds which can be interpreted as the probability that a randomly chosen positive sample is ranked higher than a negative sample:

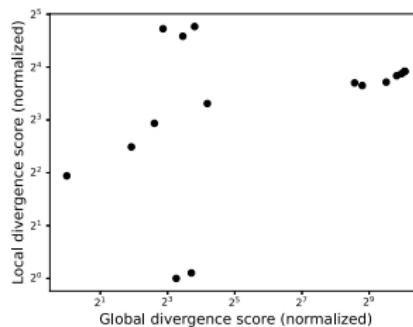
$$\frac{\sum_{st \in S^+} \sum_{uv \in S^-} \mathbf{1}\{p(s, t) > p(u, v)\}}{|S^+| \cdot |S^-|}.$$

(Can be easily approximated by **sampling**.)

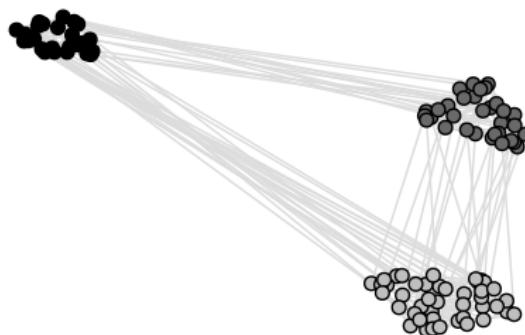
# Example: ABCD (3 communities)



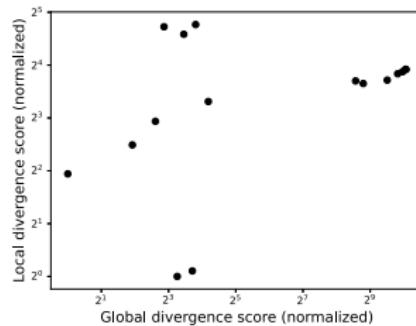
# Example: ABCD (3 communities)



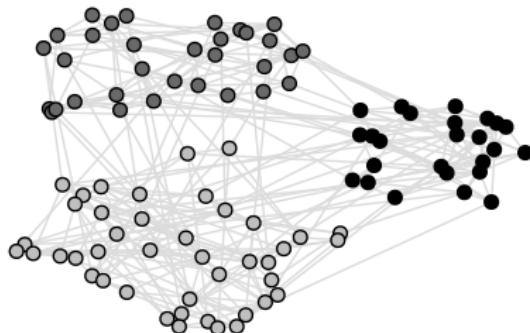
Low global divergence



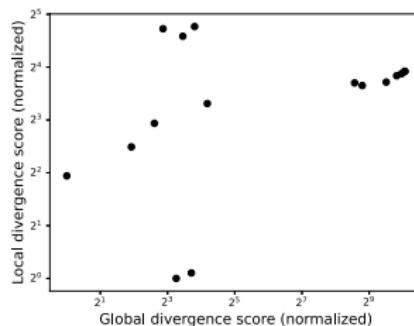
# Example: ABCD (3 communities)



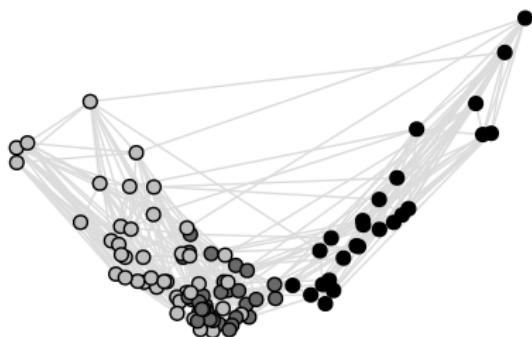
Low local divergence



## Example: ABCD (3 communities)



Both divergences high



# Geometric Chung-Lu model

We **generalized** the **Chung-Lu** model:

$$p_{i,j} = x_i x_j g(d_{i,j}),$$

where  $d_{i,j} = \text{dist}(\mathcal{E}(v_i), \mathcal{E}(v_j))$  and  $g(x) = g_\alpha(x)$  is some decreasing (similarity) function such as

$$g(d) = \left(1 - \frac{d - d_{\min}}{d_{\max} - d_{\min}}\right)^\alpha,$$

where  $d_{\min}$  and  $d_{\max}$  are the **minimum**, and respectively the **maximum**, distance between nodes in embedding  $\mathcal{E}$ .

## Geometric Chung-Lu model

$x_i$ 's have to be **tuned** so that  $\mathbb{E}[\deg(v_i)] = w_i$ .

This part requires  $\Theta(n^2)$  steps and so it is not feasible for large graphs.

## Geometric Chung-Lu model

$x_i$ 's have to be **tuned** so that  $\mathbb{E}[\deg(v_i)] = w_i$ .

This part requires  $\Theta(n^2)$  steps and so it is not feasible for large graphs.

We propose **scalable** (approximation) algorithm with running time  $O(n \ln n)$ .

## Experiments

---

# Experiments

Jupyter notebooks and datasets that accompany the textbook/slides are available on GitHub:

<https://github.com/ftheberge/GraphMiningNotebooks>

[Python\\_Notebooks\\_Second\\_Edition](#)

[Julia\\_Notebooks\\_2nd](#)



Tutorials for Python notebooks can be found on YouTube:

<https://www.youtube.com/@MiningComplexNetworks>

THE  
END