Graph Embeddings

In order to extract useful structural information from graphs, one might want to try to embed its nodes in a geometric space.

Many important applications, including:
- node classification,
- node clustering and community detection,
- link prediction and missing links,
- visualization,
- anomaly detection.

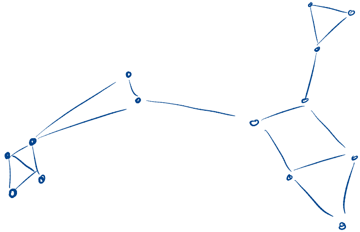
Graph Embedding

$G = (V, E)$ — weighted graph on the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$.

Graph Embedding

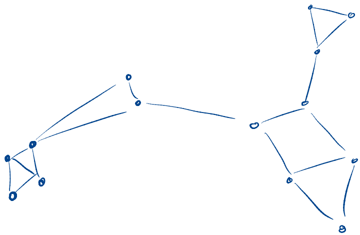
$G = (V, E)$ — weighted graph on the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$.

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



$G = (V, E)$ — weighted graph on the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$.

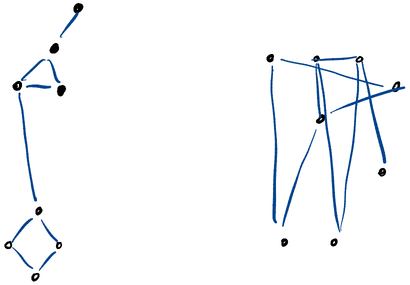
Embedding — function $\mathcal{E}\colon V \to \mathbb{R}^k$; k is much smaller than n.



$\mathcal{E}$ decreases the dimension but at the same time it tries to preserve pairwise proximity between nodes as best as possible.

Graph Embedding — Main Goal

There are many embedding algorithms (techniques from linear algebra, random walks, or deep learning) and the list grows (100+ algorithms). Results vary a lot (randomized algorithms) and are affected by parameters.



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

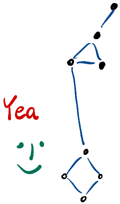
Graph Embedding — Big Picture

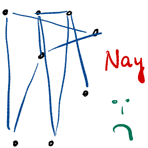
Input: Graph $G = (V, E)$ on n vertices with the degree distribution $\mathbf{w} = (w_1, \ldots, w_n)$.
Embedding of vertices of V, $\mathscr{E} : V \to \mathbb{R}^k$ (typically many).

Graph Embedding — Big Picture

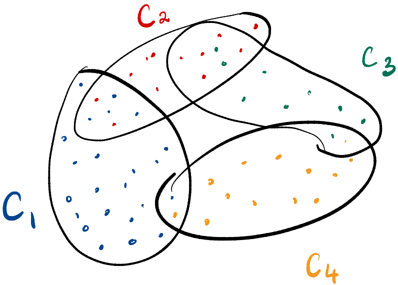
Input: Graph $G = (V, E)$ on n vertices with the degree distribution $\mathbf{w} = (w_1, \ldots, w_n)$.
Embedding of vertices of V, $\mathcal{E} : V \to \mathbb{R}^k$ (typically many).

Output: “divergence score” assigned to $\mathcal{E}$ (smaller is better).



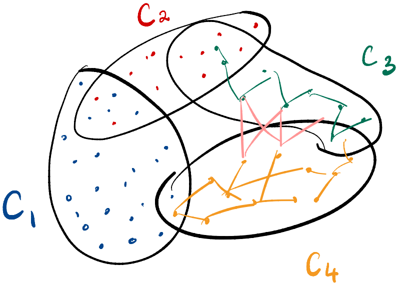


Graph Embedding — Big Picture



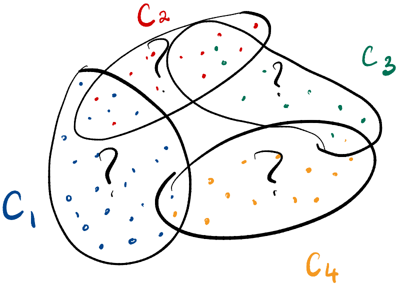
Step 1: Run some graph clustering algorithm to obtain a partition of V into ℓ communities $C_1, \ldots, C_\ell$.

Graph Embedding — Big Picture



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 $\mathscr{E}$.

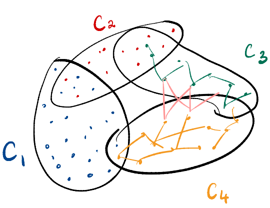
Graph Embedding — Big Picture



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 $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$.

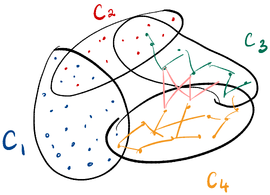
Note that we do not use G, only $\mathbf{w}$ (on top of $\mathcal{E}$).

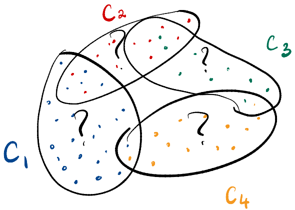
Graph Embedding — Big Picture



Step 4: Compute Δ_α: the Jensen-Shannon divergence between the two vectors.

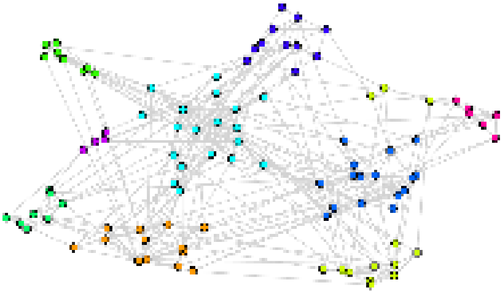
Graph Embedding — Big Picture

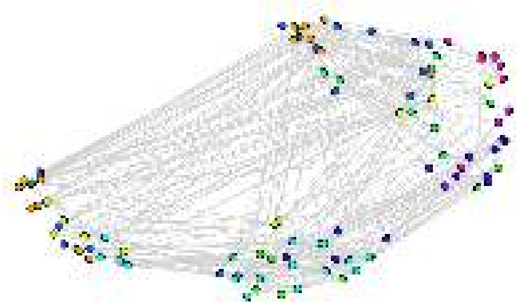




Step 5: Take the minimum Δ_α over various choices for α that measures "aversion" for long links.

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





One concrete application

Canadian Department of National Defence (Fall 2021 - Fall 2022)

Goal: detection bots (internet robots), cyborgs (hybrid accounts jointly managed by humans and bots), and hostile actors (foreign states, criminals, terrorists or activist groups trying to sway elections, introduce ideologies, undermine public confidence, and so forth).

One concrete application

Canadian Department of National Defence (Fall 2021 - Fall 2022)

Goal: detection bots (internet robots), cyborgs (hybrid accounts jointly managed by humans and bots), and hostile actors (foreign states, criminals, terrorists or activist groups trying to sway elections, introduce ideologies, undermine public confidence, and so forth).

Solution: user content (text, metadata) + network structure (graph, embeddings).

THE
END