

Advanced Data Visualization

CS 6965

Fall 2019

Prof. Bei Wang Phillips

University of Utah



Lecture 23

Project 04

- Read 3 research papers provided the the instructor
- For each research paper, comes up with 4 different ideas of extending the proposed research
- How can we combine ideas from the 3 directions?

ML for graph vis
Graph-based ML
ML on graphs

NV

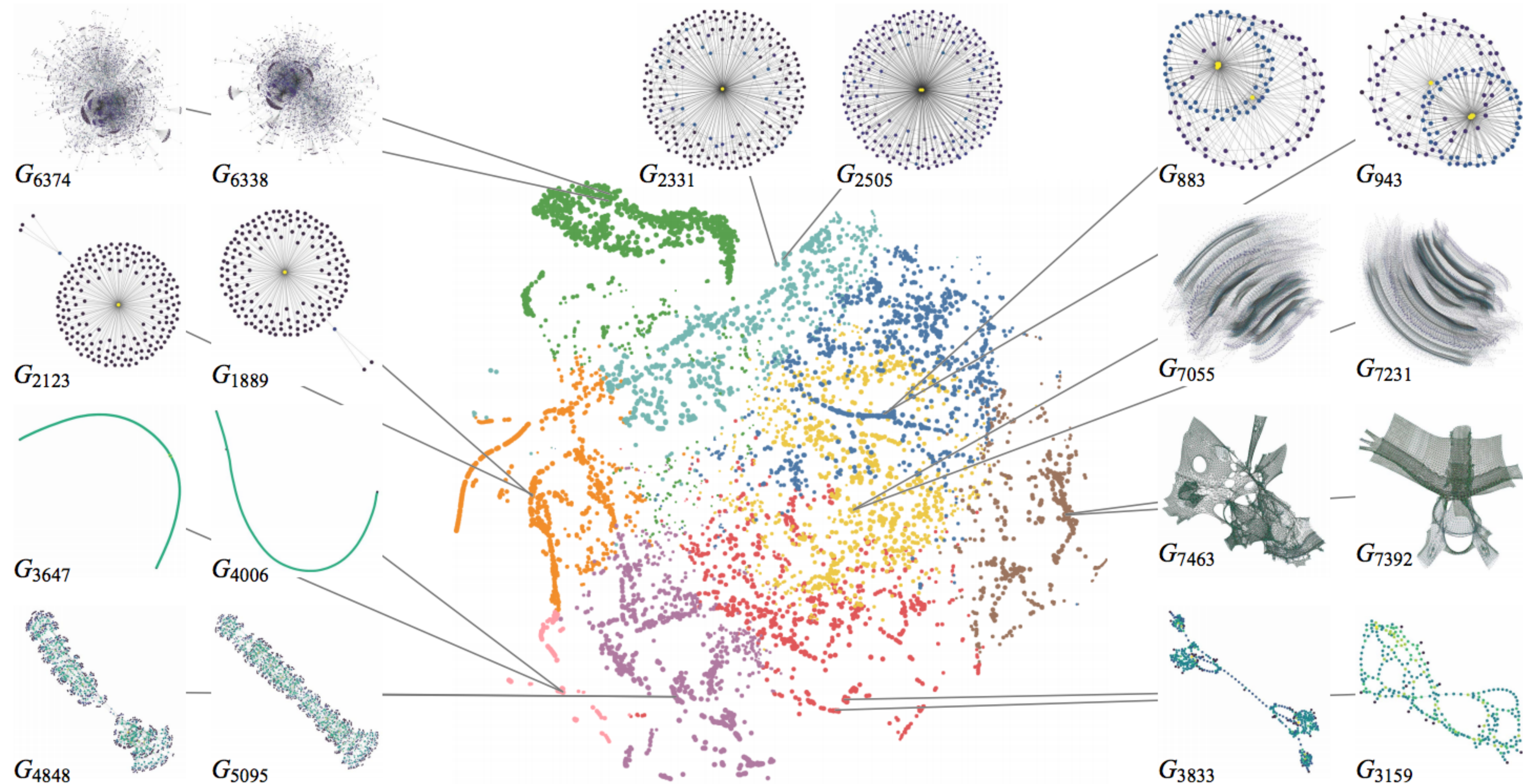
ML approach to large graph visualization

Main Challenges

- Training datasets
- Model Architecture
- Loss function design

Idea 1: when are two graphs look the same?

Machine learning approach to large graph visualization



Cluster graphs by their topological similarities.

Graphlets

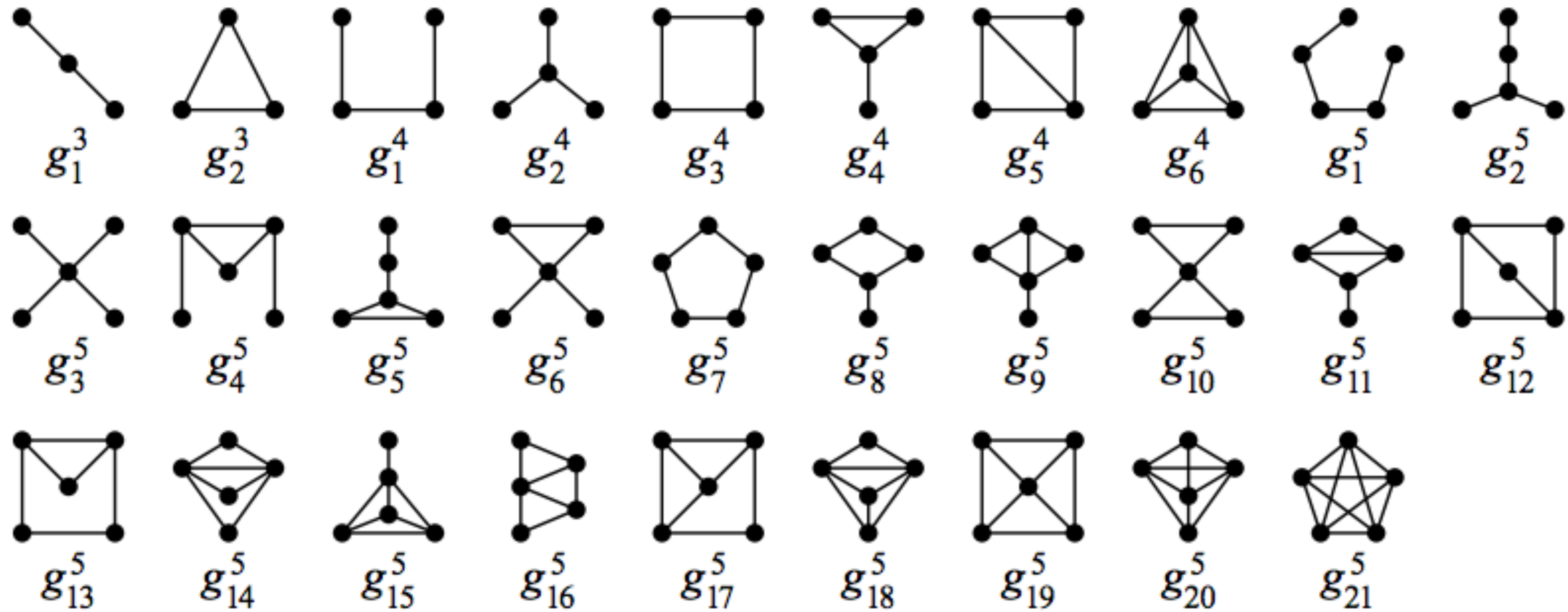


Fig. 2. All connected graphlets of 3, 4, or 5 vertices.

Graphlets Frequencies

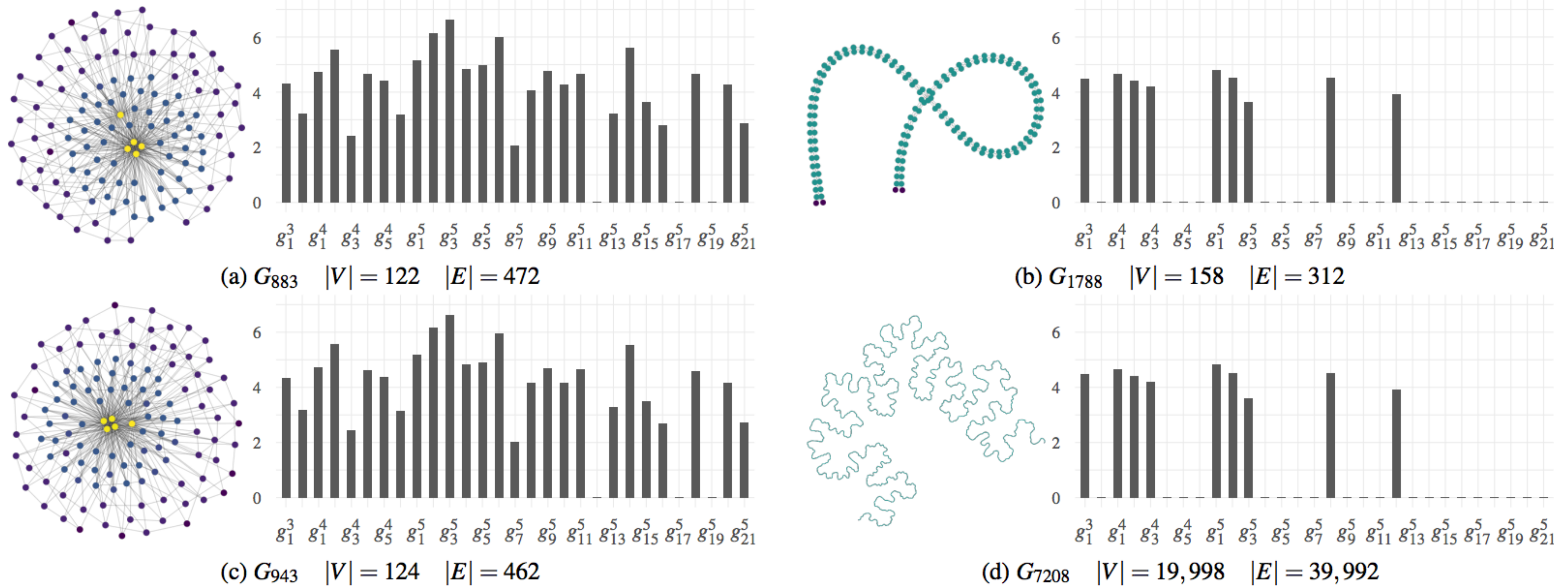


Fig. 3. Examples of graphlet frequencies. The x-axis represents connected graphlets of size $k \in \{3, 4, 5\}$ and the y-axis represents the weighted frequency of each graphlet. Four graphs are drawn with sfdp layouts [45]. If two graphs have similar graphlet frequencies, i.e., high topological similarity, they tend to have similar layout results (a and c). If not, the layout results look different (a and b). However, in rare instances, two graphs can have similar graphlet frequencies (b and d), but vary in graph size, which might lead to different looking layouts.

Topological similarities: kernels

Cosine similarity (COS): Most existing graphlet kernels use the dot product of two graphlet frequency vectors in Euclidean space, then normalize the kernel matrix. This is equivalent to the cosine similarity of two vectors, which is the L_2 -normalized dot product of two vectors:

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \frac{\mathbf{x} \cdot \mathbf{x}'^T}{\|\mathbf{x}\| \|\mathbf{x}'\|}$$

Gaussian radial basis function kernel (RBF): This kernel is popularly used in various kernelized machine learning techniques:

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

where σ is a free parameter.

Laplacian kernel (LAPLACIAN): Laplacian kernel is a variant of RBF kernel:

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_1}{\sigma}\right)$$

where $\|\mathbf{x} - \mathbf{x}'\|_1$ is the L_1 distance, or Manhattan distance, of the two vectors.

Treating graphlets frequencies as feature vectors. Define dot product of two graphlet frequency vectors.

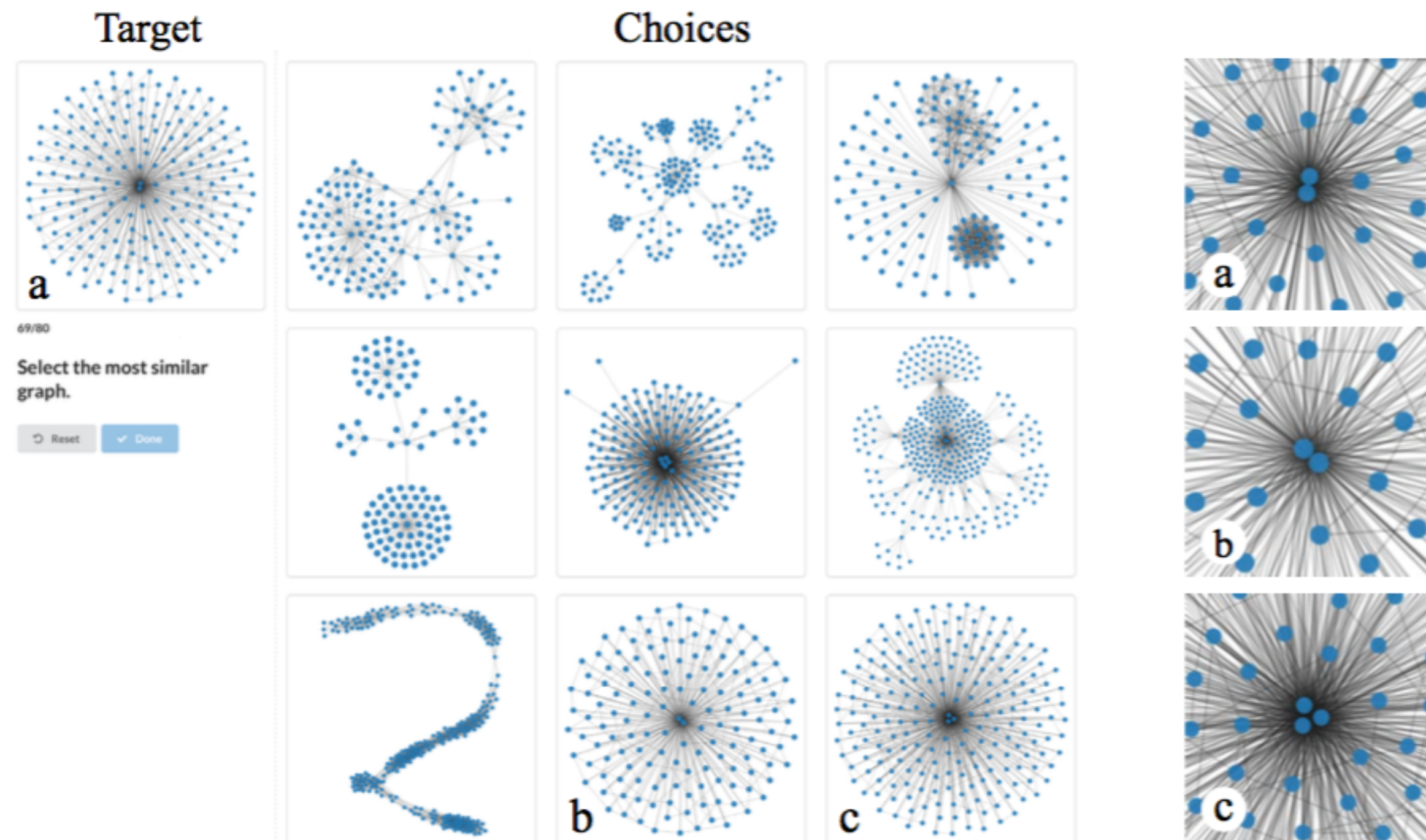
Aesthetic criteria for graph VIS

- Crosslessness: Minimizing the number of edge crossings
- Minimum angle metric: maximizing the minimum angle between incident edges on a vertex.
- Edge length variation: Uniform edge lengths.
- Shape-based metric: Mean Jaccard similarity between the input graph and the shape graph.

$$\text{MJS}(G_1, G_2) = \frac{1}{|V|} \sum_{v \in V} \frac{|N_1(v) \cap N_2(v)|}{|N_1(v) \cup N_2(v)|}$$

What would a graph look like in this layout?

- Compare accuracy of ML predicted similar graph layout vs human chosen layout



***Idea 2: how to find a
good layout?***

<http://kwonoh.net/dgl/>

Finding a Good Layout is Challenging

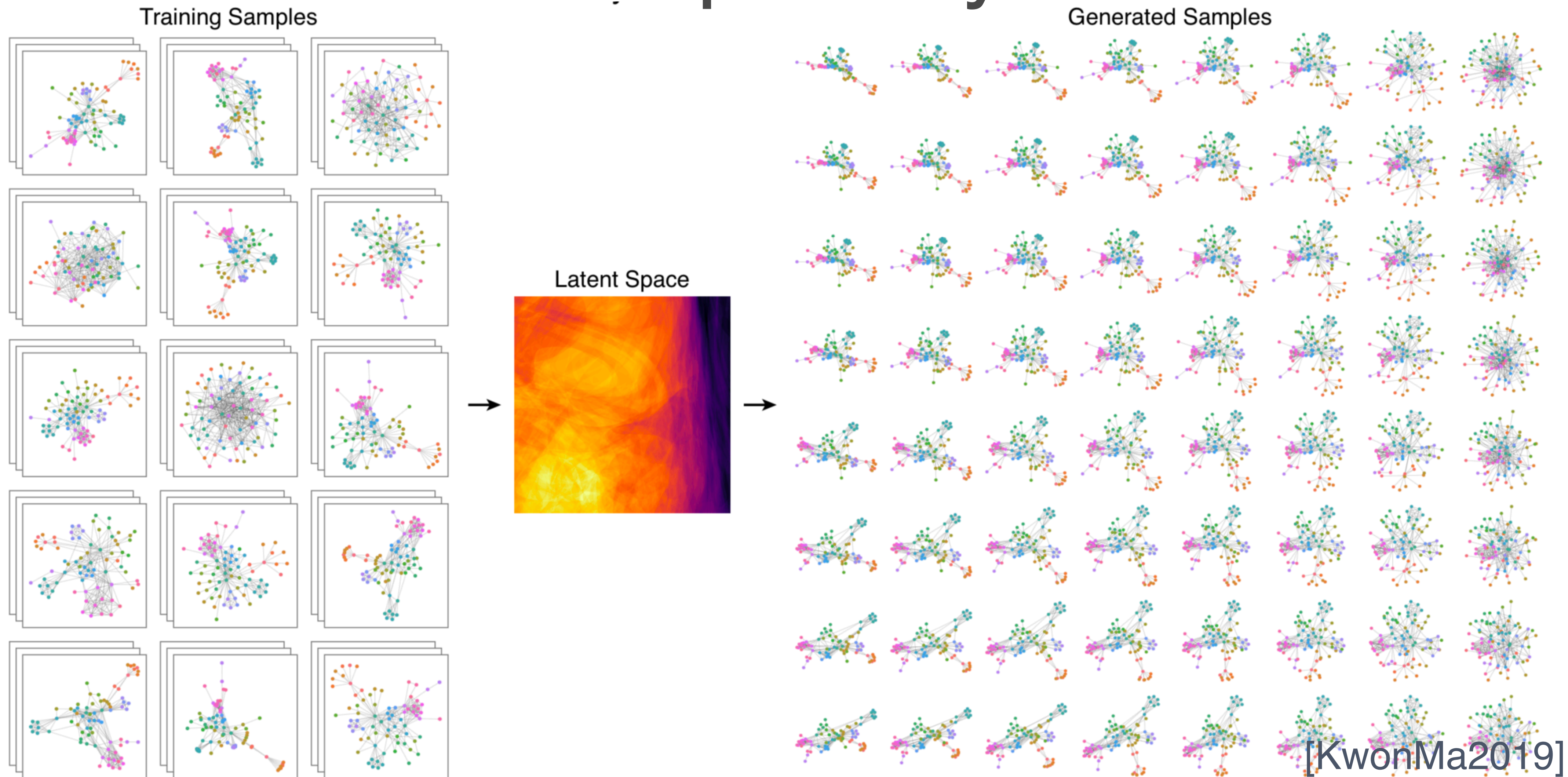
Things to consider:

- the structure of the given graph
- the purpose of the visualization (e.g., highlighting communities)
- subjective preferences

Existing heuristics (e.g., reducing edge crossings) are not enough

So far, there is no automatic method to find a good layout

A Deep Generative Model for Graph Layout



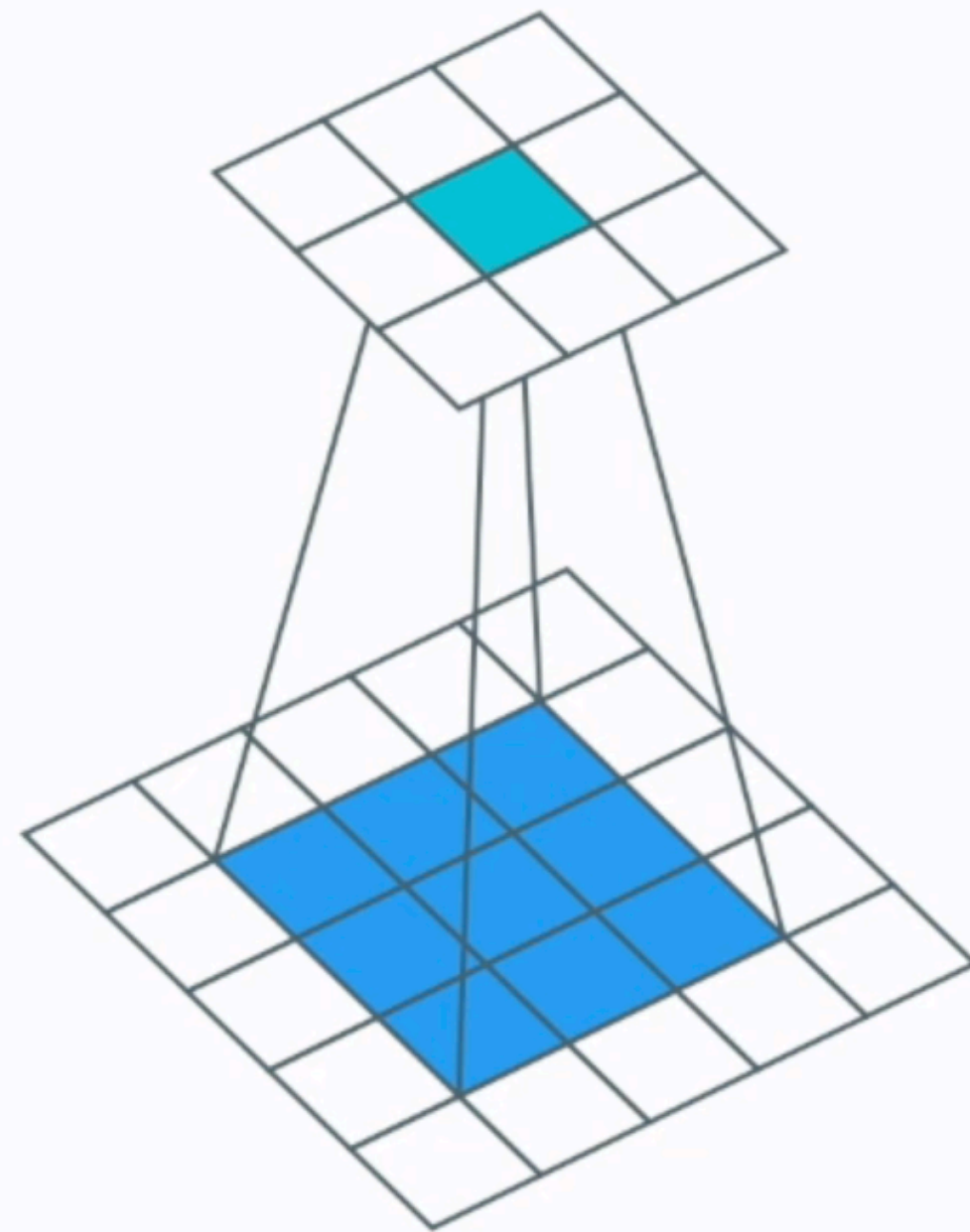
Detailed Approach

- **Training data collection:** using multiple layout methods using random search, where each layout is computed using randomly assigned parameter values of a given layout method
- **Layout features:** We use the pairwise Euclidean distance of nodes in a layout as the feature of the layout
- **Structural Equivalence:** compare two different layouts of the same set of Structurally equivalent nodes (SENs_ using the Gromov–Wasserstein (GW) distance; Two nodes of a graph are said to be structurally equivalent if they have the same set of neighbors.

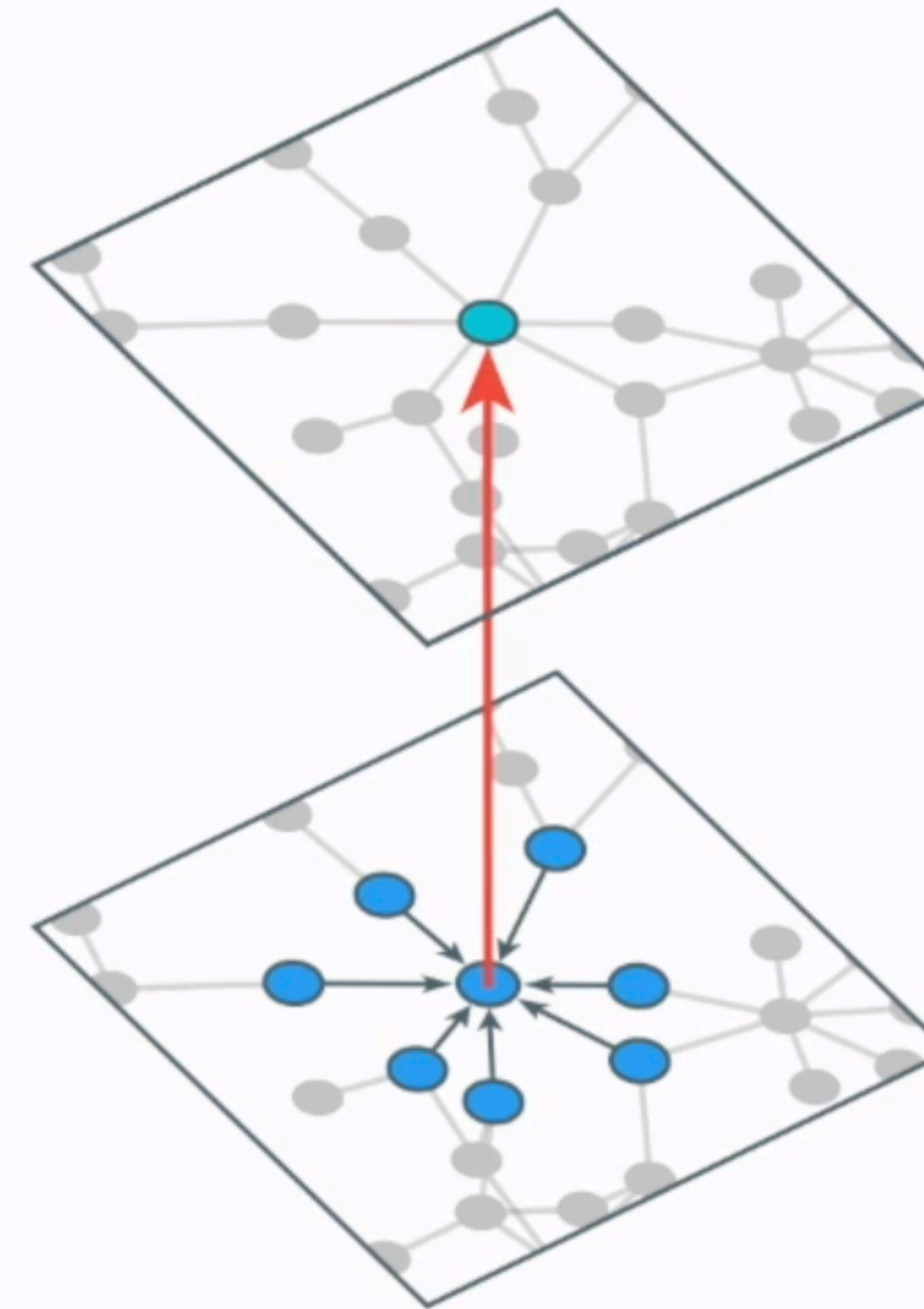
Detailed Approach

- **Architecture**
 - An autoencoder (AE) learns to encode a high-dimensional input object to a lower-dimensional latent space and then decodes the latent representation to reconstruct the input object
 - VAEs extend the classical AEs by minimizing variational loss which measures the difference of the distribution of the input objects' latent representations and a prior distribution.
- **Training:** learn the parameters of the neural network used in our model by minimizing the reconstruction loss and the variational loss

Graph Neural Networks

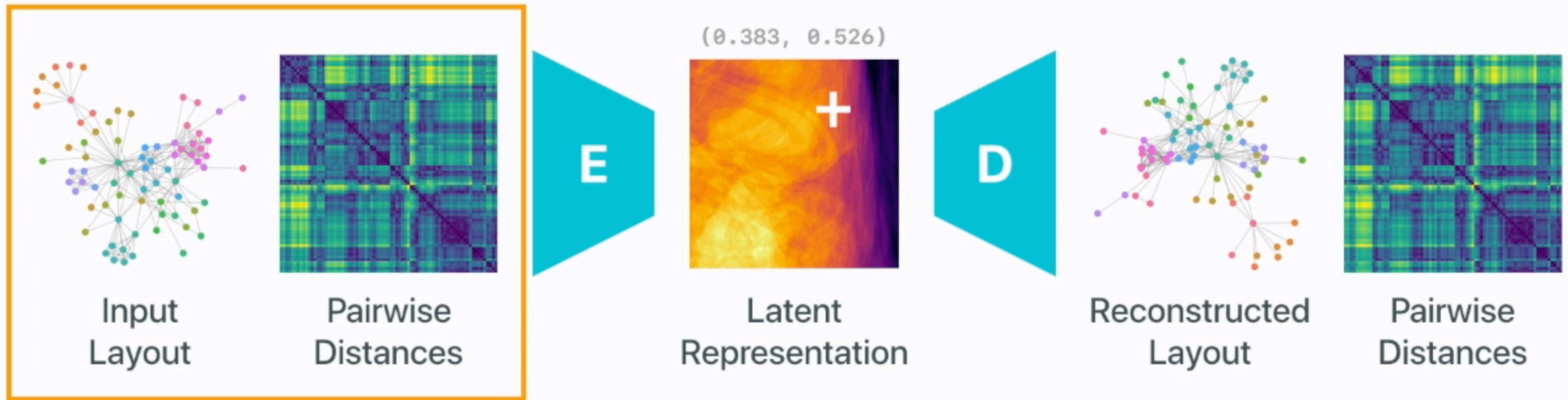


Convolution on Images

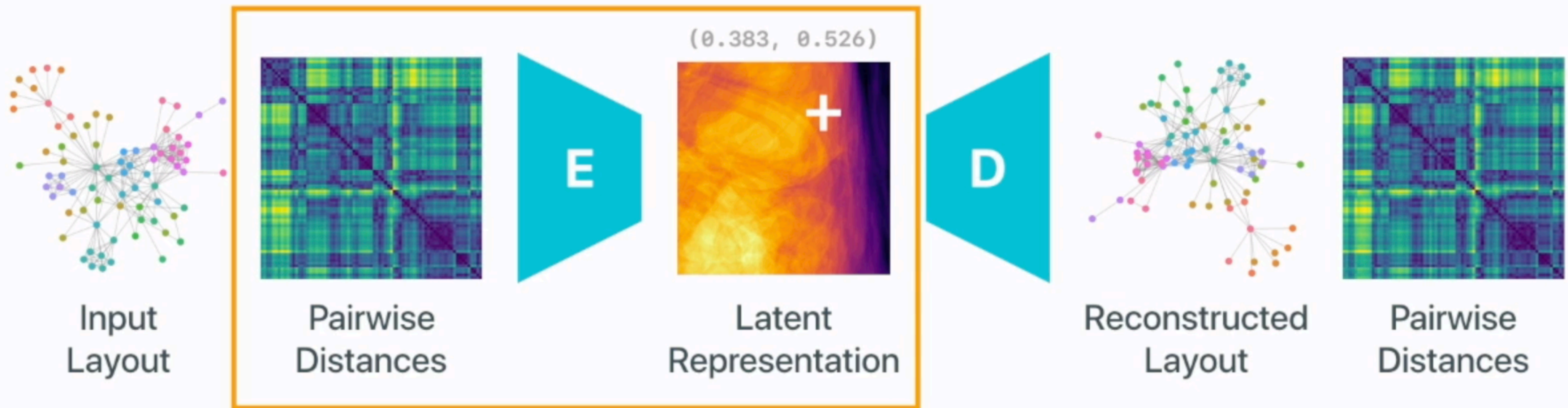


Convolution on Graphs

Positions → Pairwise Distance Matrix



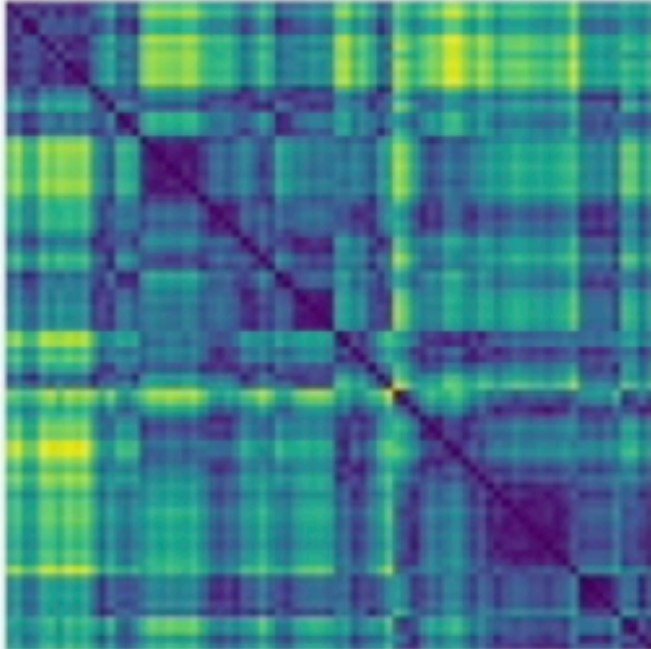
Pairwise Distance Matrix \rightarrow Latent Representation



Latent Representation → Positions (Reconstructed)



Input Layout



Pairwise Distances



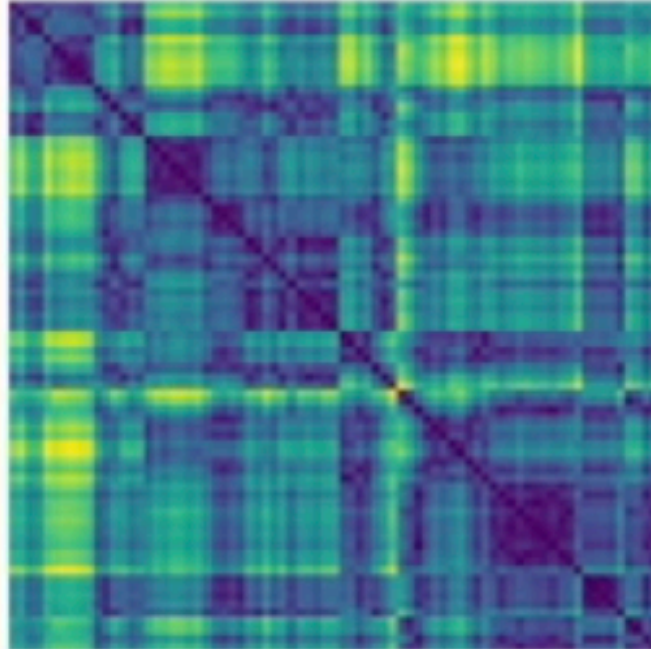
(0.383, 0.526)

A heatmap representing a latent representation, with a white '+' symbol in the upper right quadrant.

Latent Representation

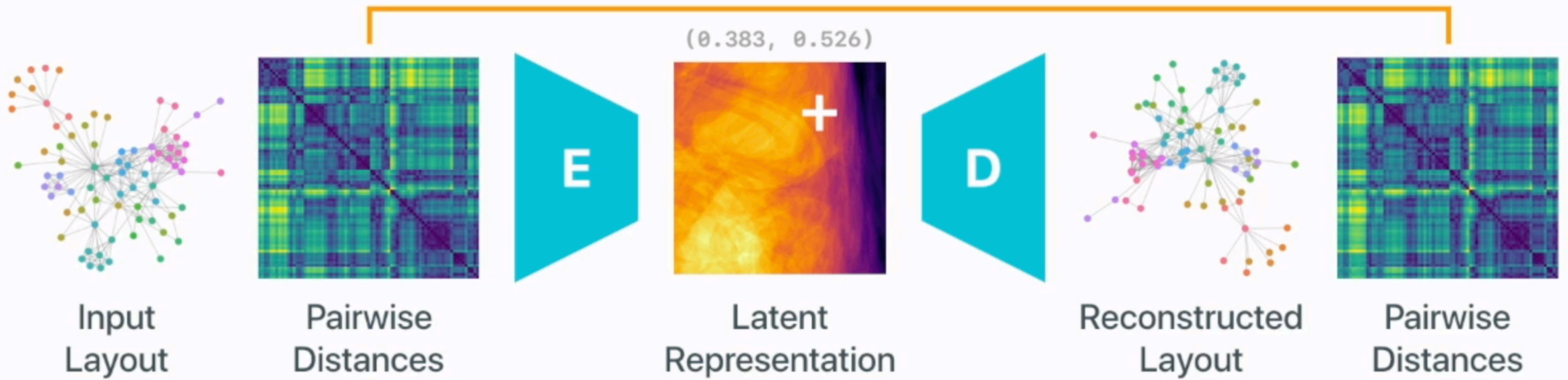


Reconstructed Layout

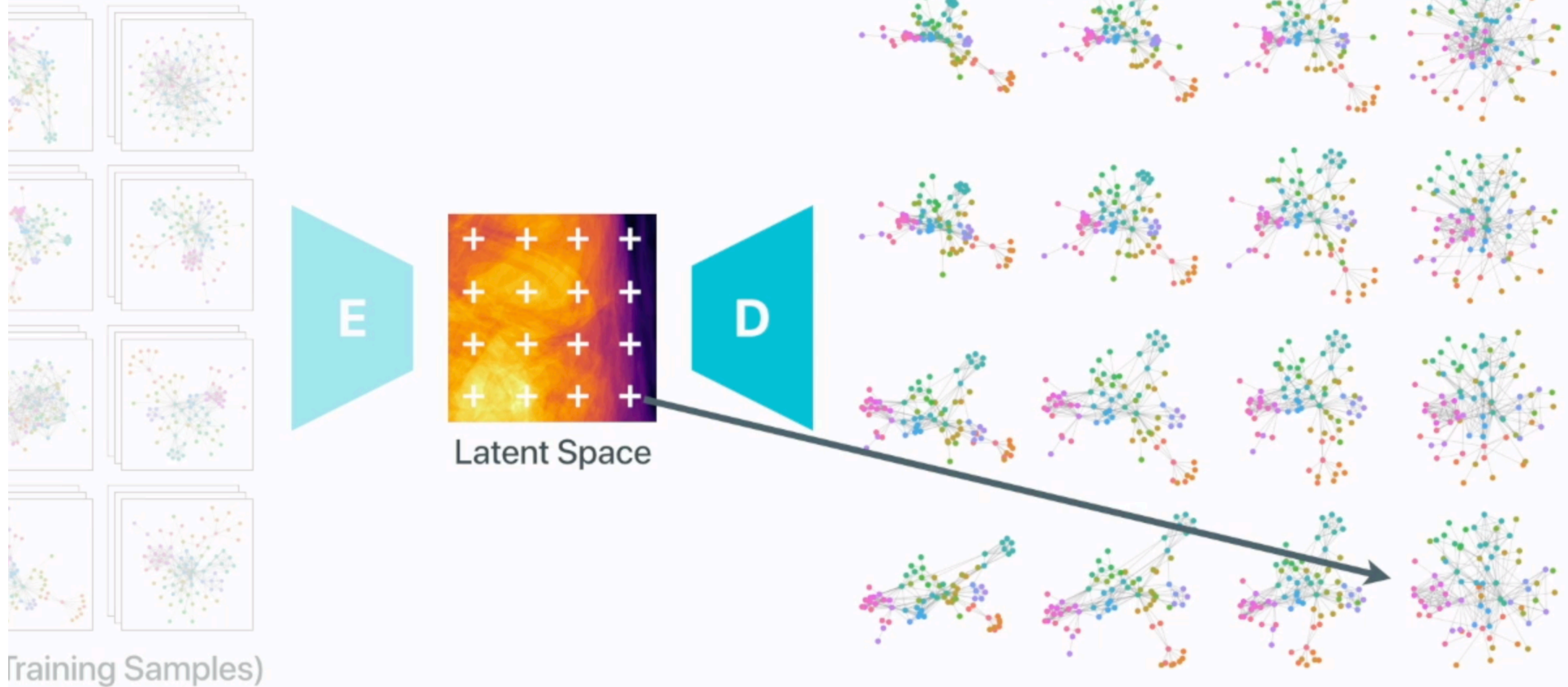


Pairwise Distances

Minimize Difference



Sample Grid



Limitations

Does not generalize for different graphs

→ Needs to train a separate model for each graph

→ We are working on it!

Training time (few minutes to few hours)

→ Incremental training, unsupervised learning

→ The user's time vs the computer's time

Graph size (~1K nodes, ~3K edges, and 40–100 layouts per batch)

→ Limited by the GPU memory

→ Sampling-based GNNs

Artificial Intelligence Augmentation

Using artificial intelligence to augment human intelligence

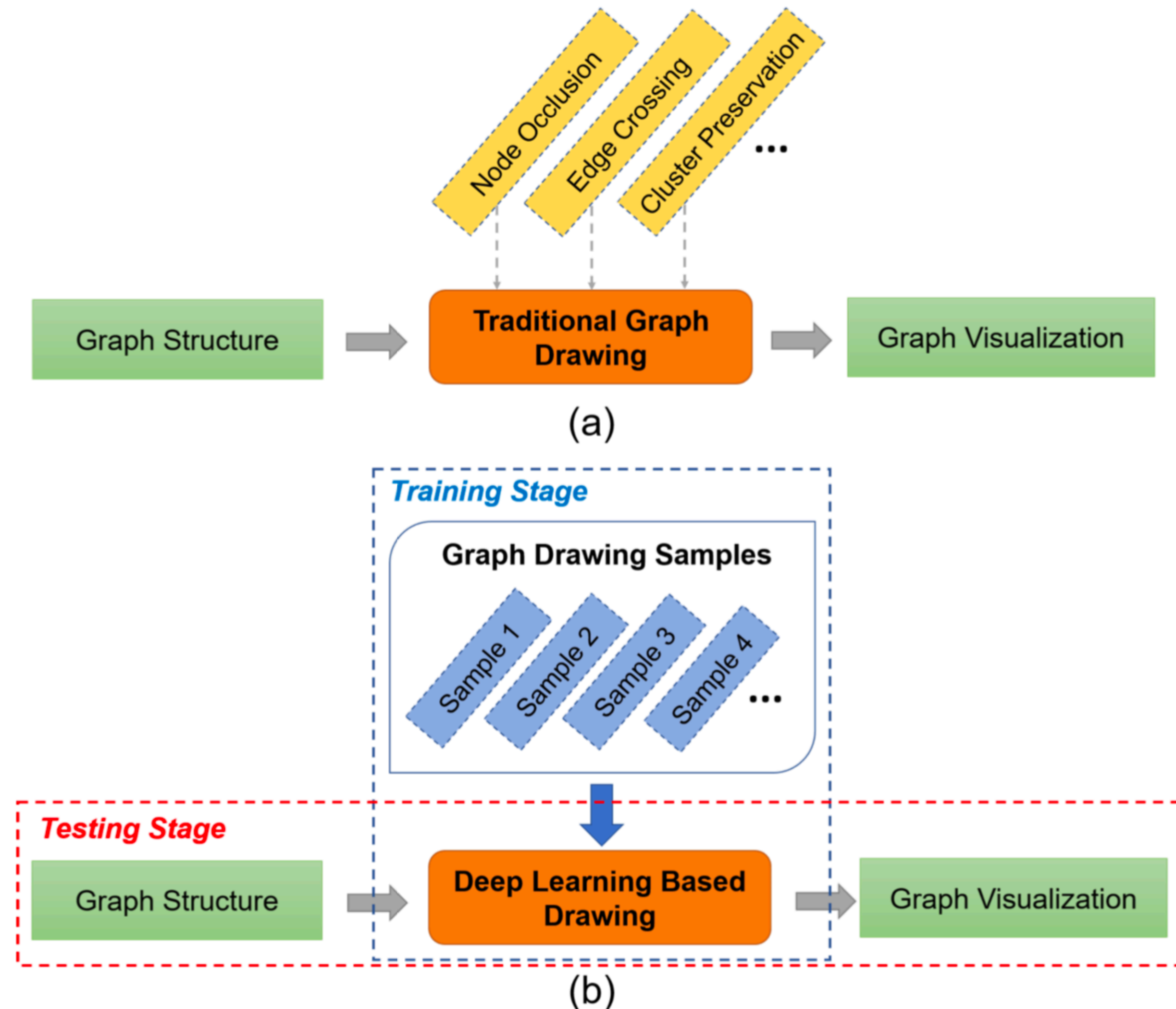
Shan Carter and Michael Nielsen, Distill, 2017

Our model builds a new type of user interface (i.e., layout latent space) to augment a human intelligence task (i.e., graph visualization design).

Machine learning can assist users in producing effective visualizations for data analysis and communication.

Idea 3: when is a prediction correct?

DeepDrawing: A Deep Learning Approach to Graph Drawing



Approach

- Training: Given a set of graph drawing examples with desirable aesthetic properties and their structures, the deep learning model is trained to learn the mapping and corresponding algorithm-specific parameters for determining the desirable graph drawings
- Testing: After training, when given a new graph, it can automatically analyze the graph structure and directly generate a layout that carries the common visual properties of the drawing examples (edge crossings, community preservation and node occlusion that are shared by the training graph drawings).
- The deep learning model learns one specific drawing style from a certain training dataset.

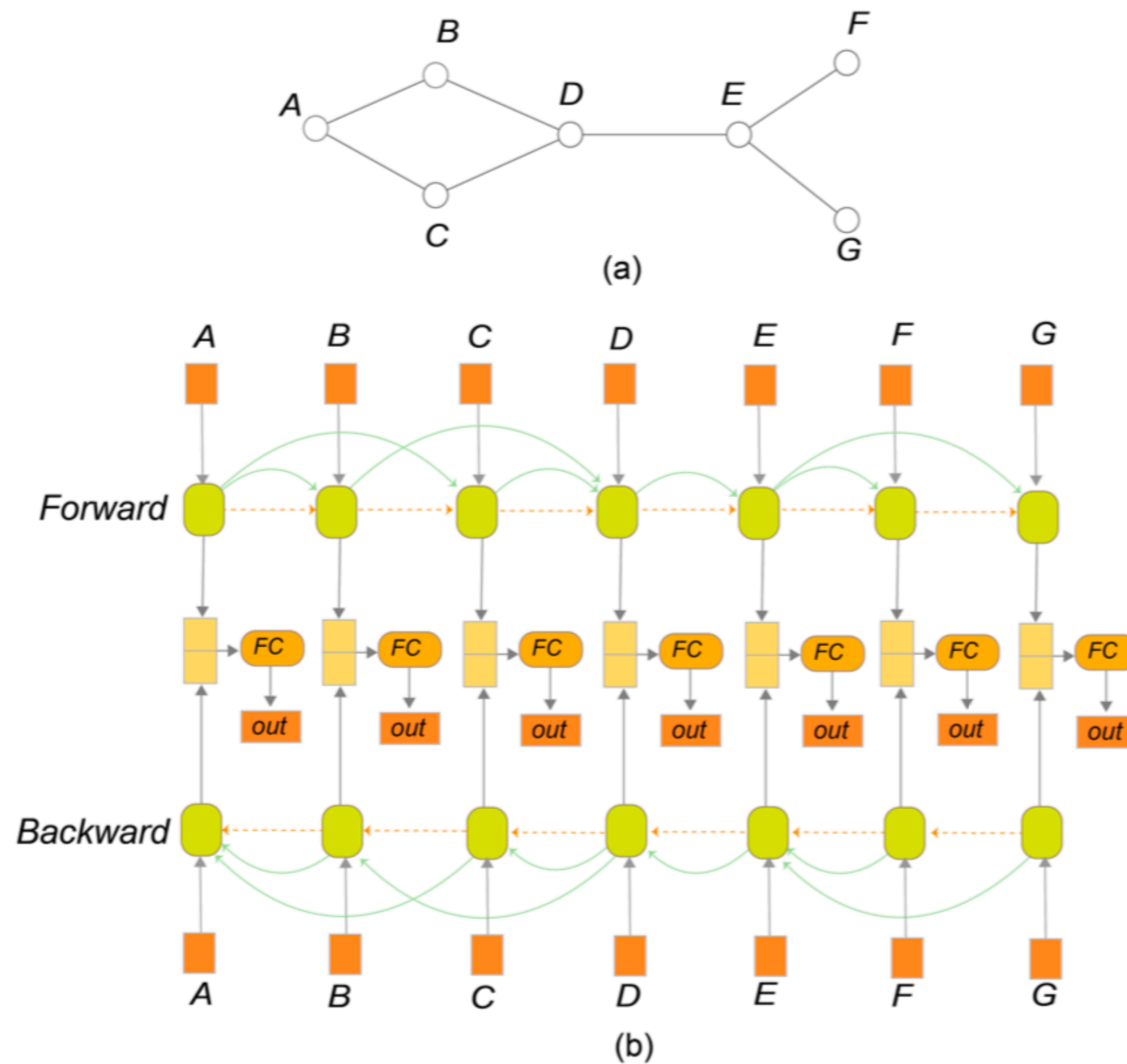


Fig. 2. An illustration of the proposed graph-LSTM-based model architecture: (a) an example graph input, (b) the proposed model to process the input graph. The graph nodes are sorted using BFS, with each node represented by an adjacency vector encoding its connections with predecessor nodes. The dotted yellow arrows (“fake” edges) propagate the prior nodes’ overall influence on the drawing of subsequent nodes, and the curved green arrows (real edges of graphs) explicitly reflect the actual graph structure, enhancing the graph drawing details. The information of both forward and backward rounds is considered for generating the final 2D node layouts.

Loss function

- The graph drawings in the training dataset are regarded as the ground-truth drawings, and the purpose of the loss function is to guide the proposed model to generate graph layouts as “similar” to the corresponding training data as possible.

between two drawings of the same graph. Procrustes Analysis is a widely-used technique in statistics and shape comparison [31] and it has also been used in graph drawing [45]. For two graph drawings of the same graph with n nodes, Procrustes Analysis will explicitly translate, scale and rotate the drawings to align them. Suppose the corresponding coordinates of all the n nodes in the two drawings are $C = [c_1, \dots, c_n]^T$ and $\bar{C} = [\bar{c}_1, \dots, \bar{c}_n]^T$, where $c_i = (x_i, y_i)$ and $\bar{c}_i = (\bar{x}_i, \bar{y}_i)$, the **Procrustes Statistic** will be calculated to indicate the shape difference between them as follows:

$$R^2 = 1 - \frac{(\text{tr}(C^T \bar{C} \bar{C}^T C)^{1/2})^2}{\text{tr}(C^T C) \text{tr}(\bar{C}^T \bar{C})} \quad (14)$$

where $0 \leq R^2 \leq 1$. It is essentially the squared sum of the distances between C and \bar{C} after a series of best possible transformations. $R^2 = 0$ denotes both graph drawings are exactly the same, while $R^2 = 1$ indicates that the two graph drawings are totally different and cannot be matched by any transformations.

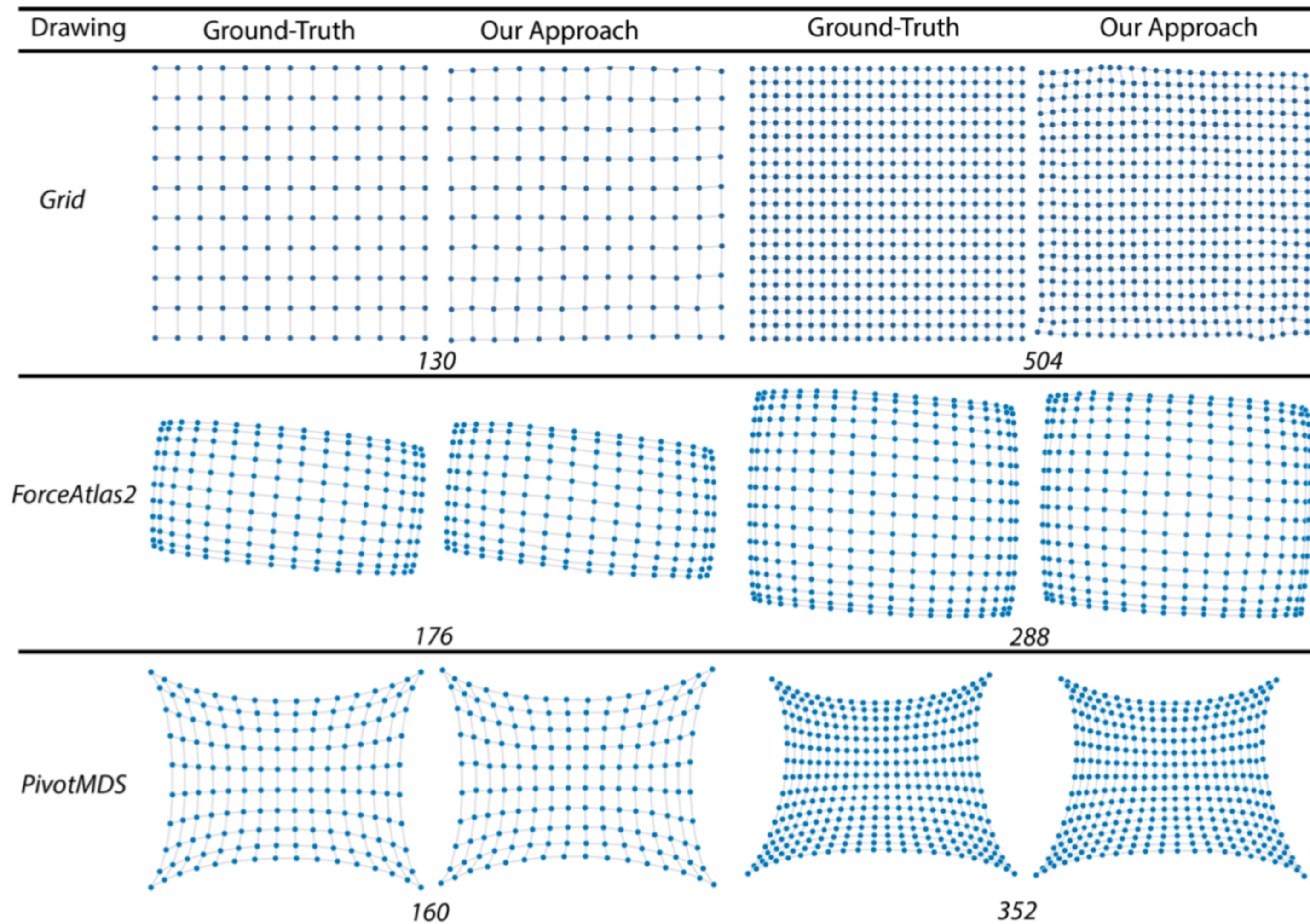
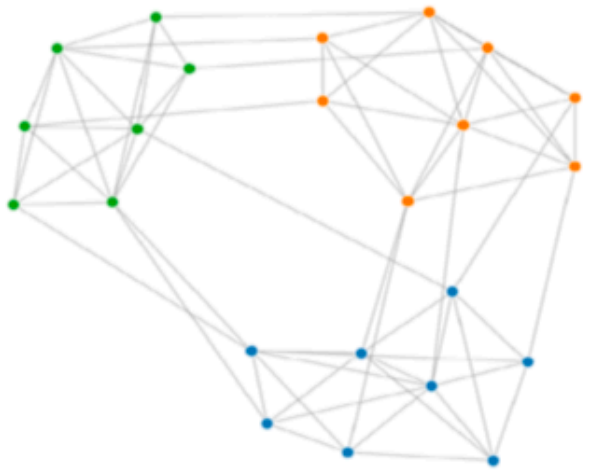
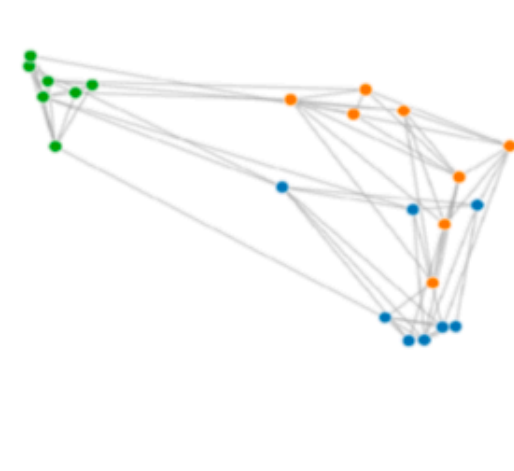
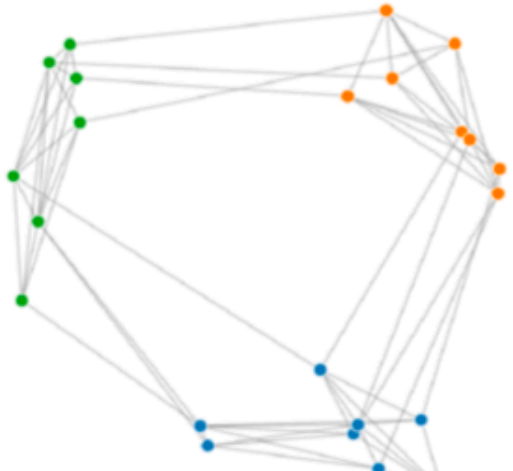

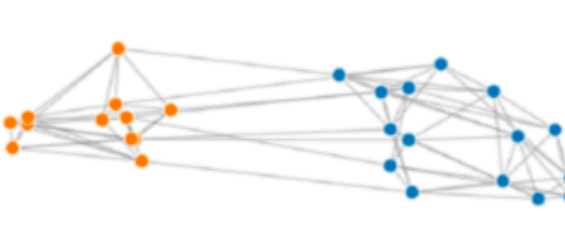


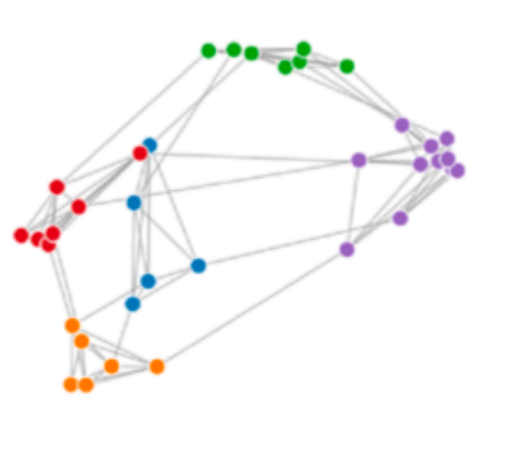



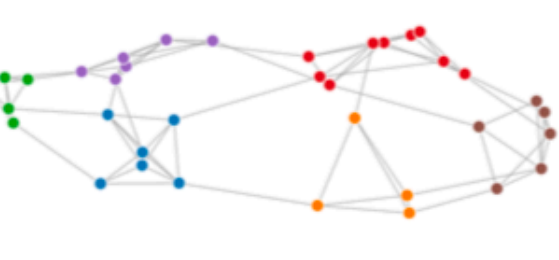
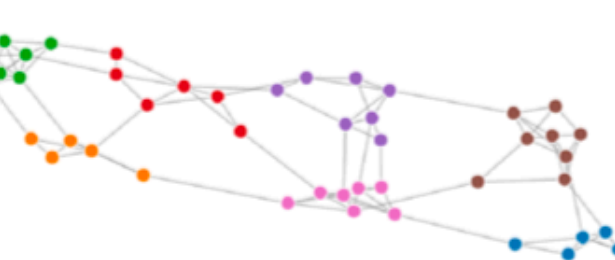
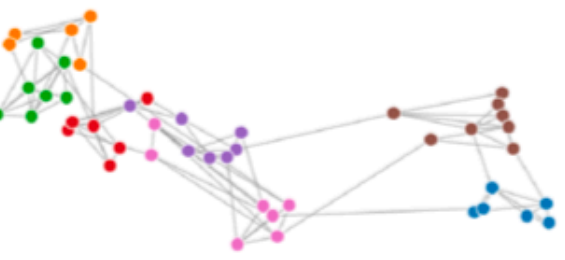
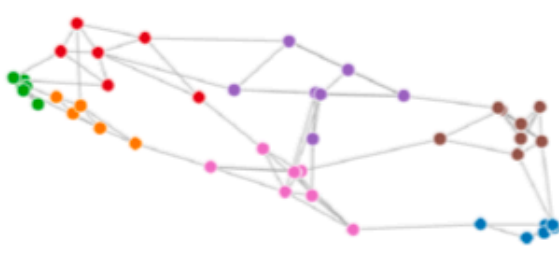
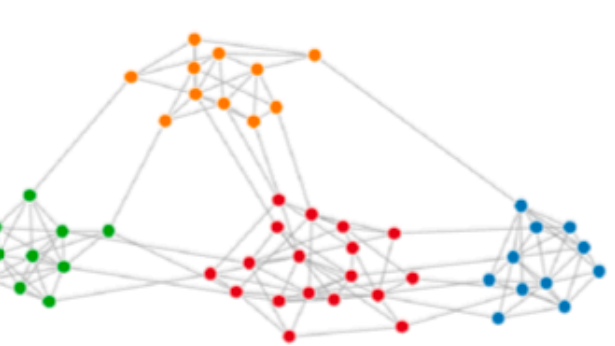
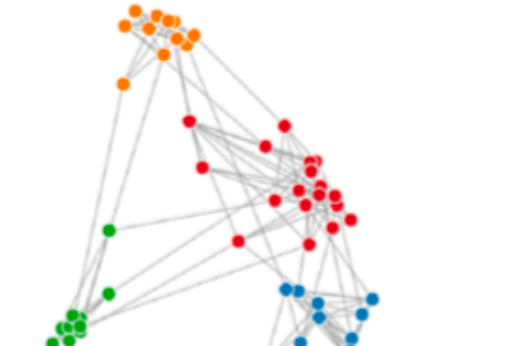
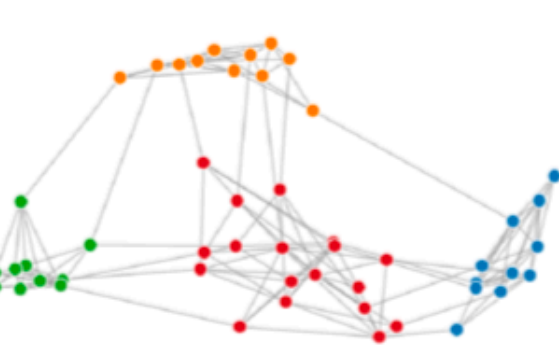


Fig. 4. Qualitative evaluation results on grid graphs: the ground truth and the graph drawing generated by our approach are compared side by side. Each row shows the results of a specific drawing style (i.e., perfect grid layout, ForceAtlas2 and PivotMDS) and the number of graph nodes is shown in the bottom.

Graph Size	Ground-Truth(ForceAtlas2)	Baseline Model	Our Approach
23 Nodes 3 Communities			
27 Nodes 2 Communities			
36 Nodes 5 Communities			
36 Nodes 6 Communities			
46 Nodes 7 Communities			
50 Nodes 4 Communities			

Baseline: other generative models

[WangJinWang2019]

Apply ML to Graph Drawing

Existing approaches

- Approaches that learn from human interaction: learn information from users based on their interactions to a graph drawing system.
- Approaches that are not based on human interaction: gather and evolve knowledge about how to draw a graph from the results of other automatic graph drawing algorithms or from the graph structure itself.

Rate by users

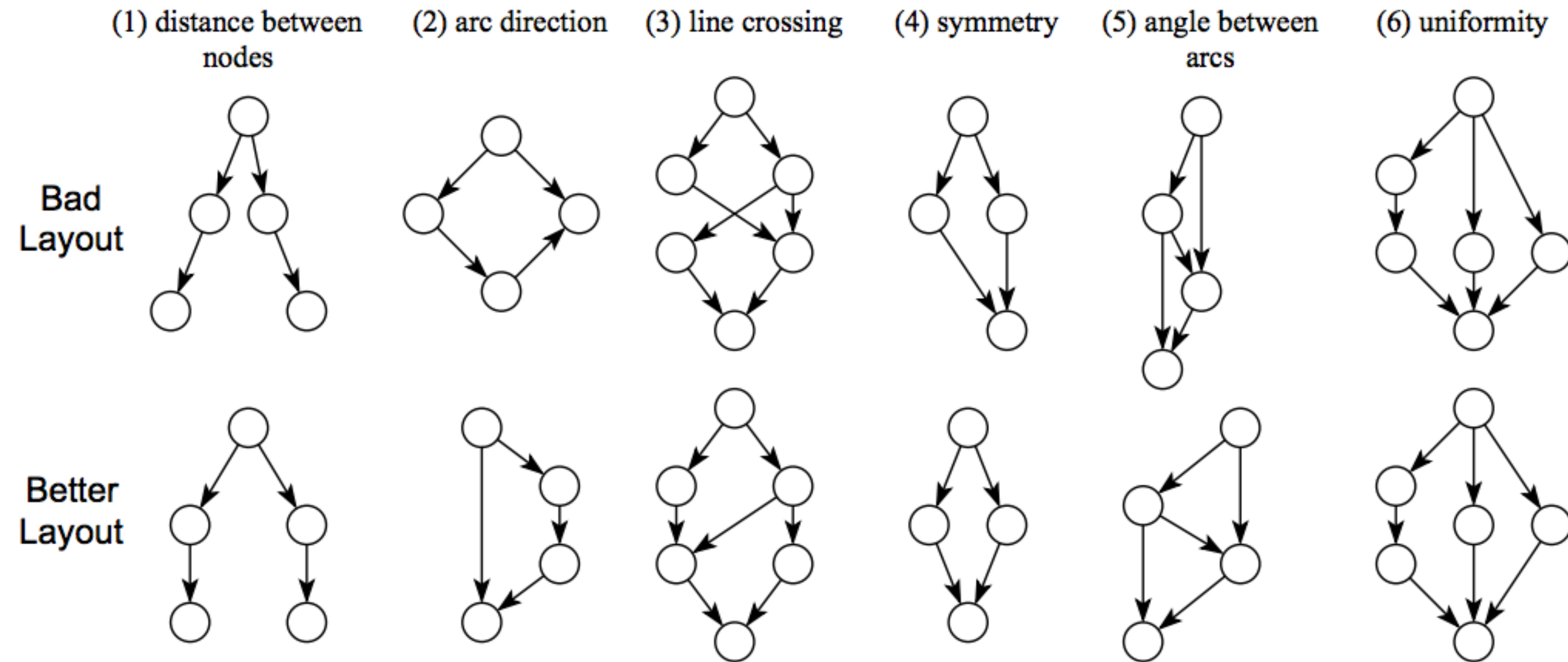


Figure 1: Constraints used in the layout of directed graphs.

Learning optimal graph drawing for clustered graph

- Construct a handcrafted feature vector of a cluster from a number of graph measures: number of vertices, diameter, and maximum vertex degree.
- Find an optimal layout for each cluster.

Using ML to improve layout quality

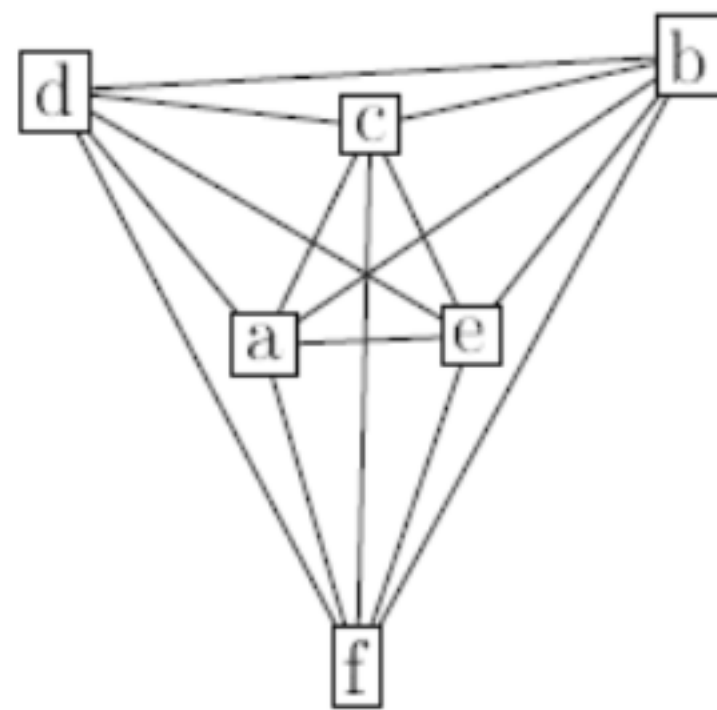
- Human in the loop evaluation.
- Using neural network algorithms to optimize a layout for certain aesthetic criteria.

•

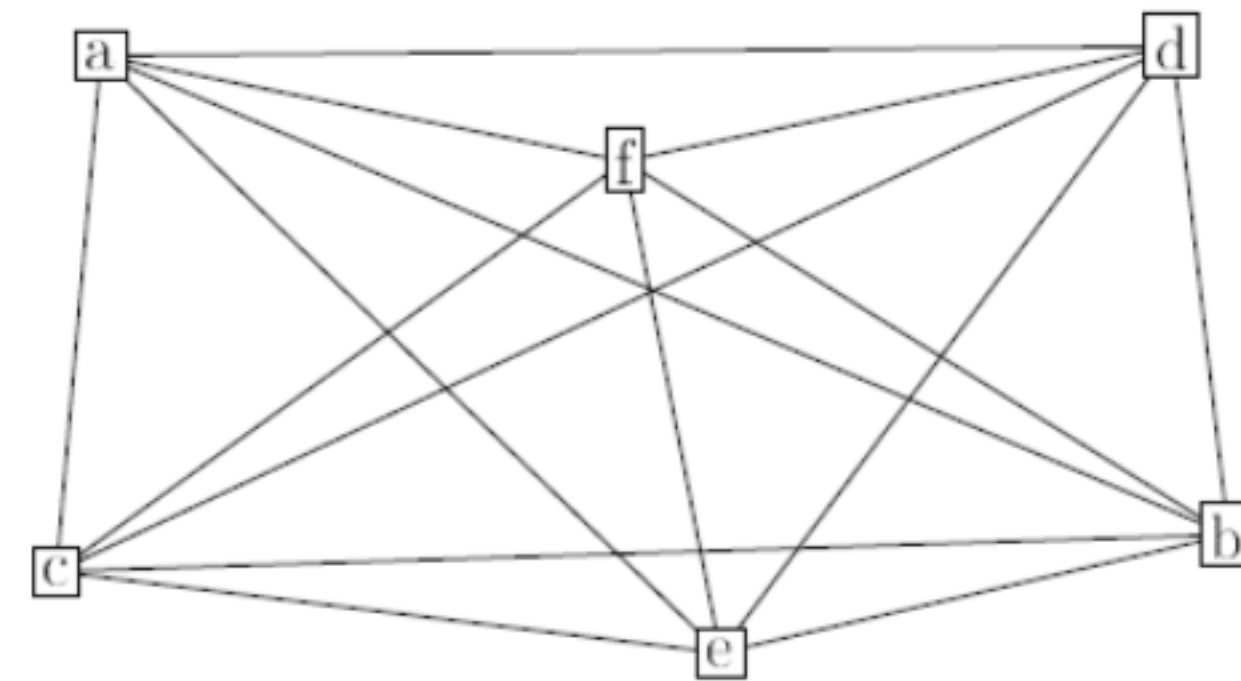
Self-organizing graphs

B. Meyer: Self-organizing graphs - a neural network perspective of graph layout

- Not building a general neural network that learns aesthetic criteria or hints about how to draw a graph. Instead, it models the graph structure and the drawing problem as a network coupled with an energy system.



(a)



(b)

Future directions?

- Using neural network algorithms to optimize a layout for certain criteria
- Really **learn** how to draw a good graph

Graph-Based ML

<https://research.googleblog.com/2016/10/graph-powered-machine-learning-at-google.html>

“We firmly believe is that it's at the intersection of machine learning and graph technology where the next evolution lies and where new disruptive companies are emerging,” says Ash Damle, Founder and CEO at [Lumiata](#) which helps healthcare organisations makes predictions.

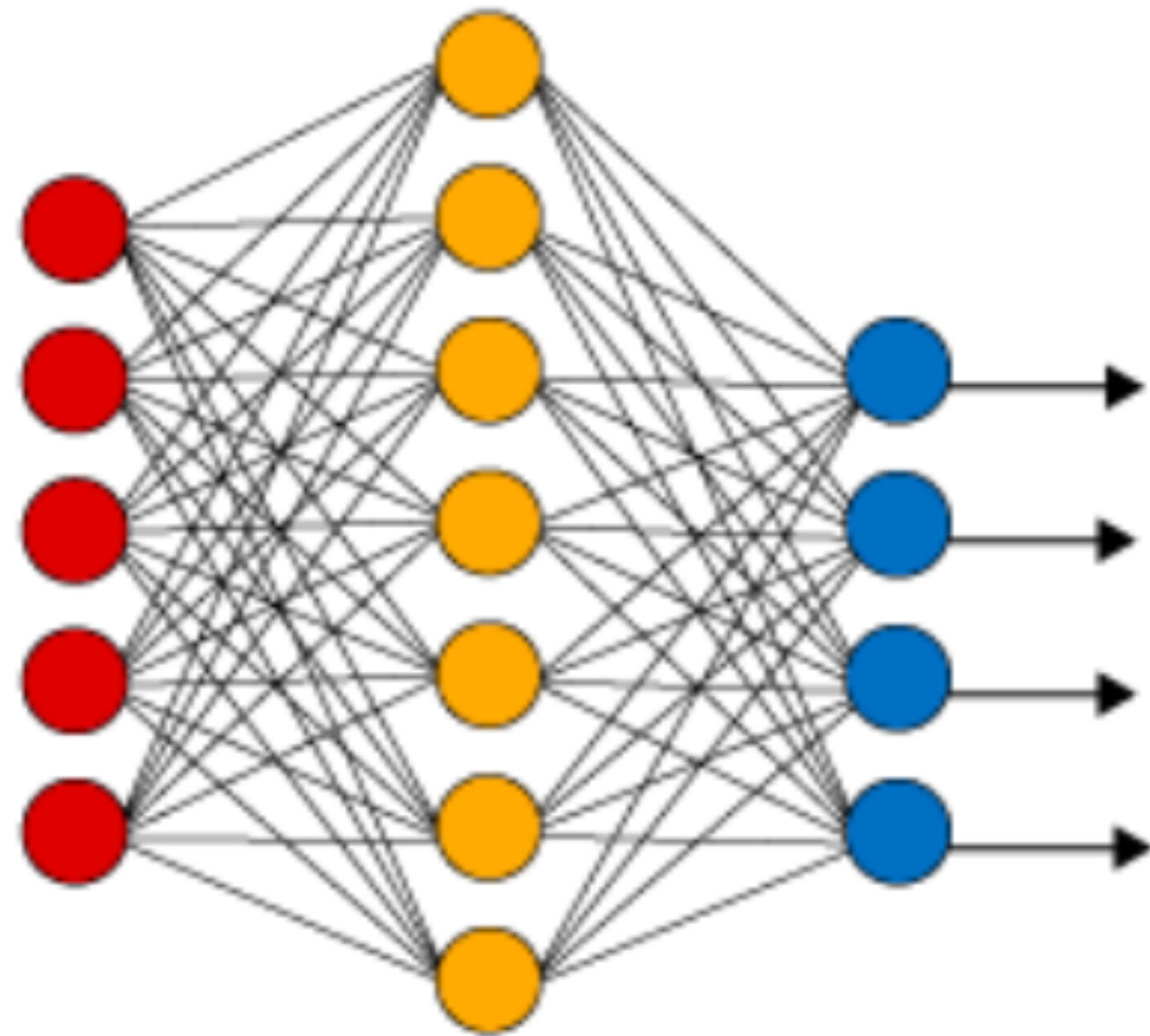
“It's only recently that companies can use graph at true scale and, now, by integrating with ML, we're moving much more into a core understanding of artificial intelligence, deep neural networks and image recognition.”

<http://www.idgconnect.com/abstract/18124/the-wave-disruption-graph-machine-learning>

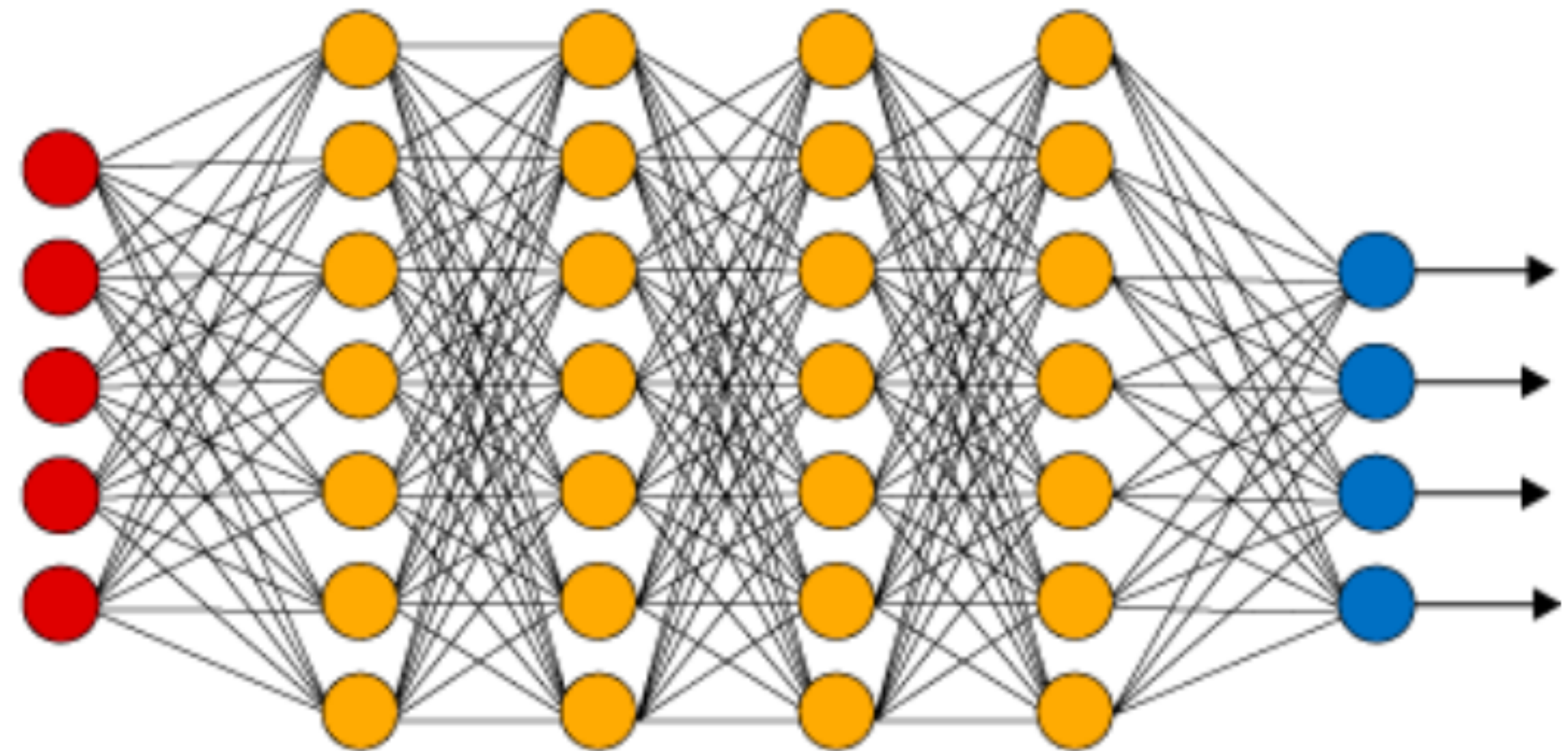
What ML algorithms are based on graphs?

Deep Learning!

Simple Neural Network



Deep Learning Neural Network



● Input Layer ● Hidden Layer ● Output Layer

First train a system using labeled data with features and then apply the trained system to unlabeled data

<http://www.global-engage.com/life-science/deep-learning-in-digital-pathology/>

Graph-powered Machine Learning at Google

Thursday, October 06, 2016

Posted by Sujith Ravi, Staff Research Scientist, Google Research

Recently, there have been significant advances in [Machine Learning](#) that enable computer systems to solve complex real-world problems. One of those advances is Google's large scale, [graph-based](#) machine learning platform, built by the Expander team in Google Research. A technology that is behind many of the Google products and features you may use everyday, graph-based machine learning is a powerful tool that can be used to power useful features such as [reminders in Inbox](#) and [smart messaging in Allo](#), or used in conjunction with deep neural networks to power the latest image recognition system in [Google Photos](#).



[https://
research.googleblog
.com/2016/10/
graph-powered-
machine-learning-
at-google.html](https://research.googleblog.com/2016/10/graph-powered-machine-learning-at-google.html)

Graph-based semi-supervised learning

- Sparse training data
- Model labeled and unlabeled data jointly during learning, leveraging the underlying structure in the data
- Easily combine multiple types of signals (for example, relational information from Knowledge Graph along with raw features) into a single graph representation and learn over them.

***Graph-Based
Semi-Supervised
Learning***

Semi-Supervised Learning Tutorial

Xiaojin Zhu

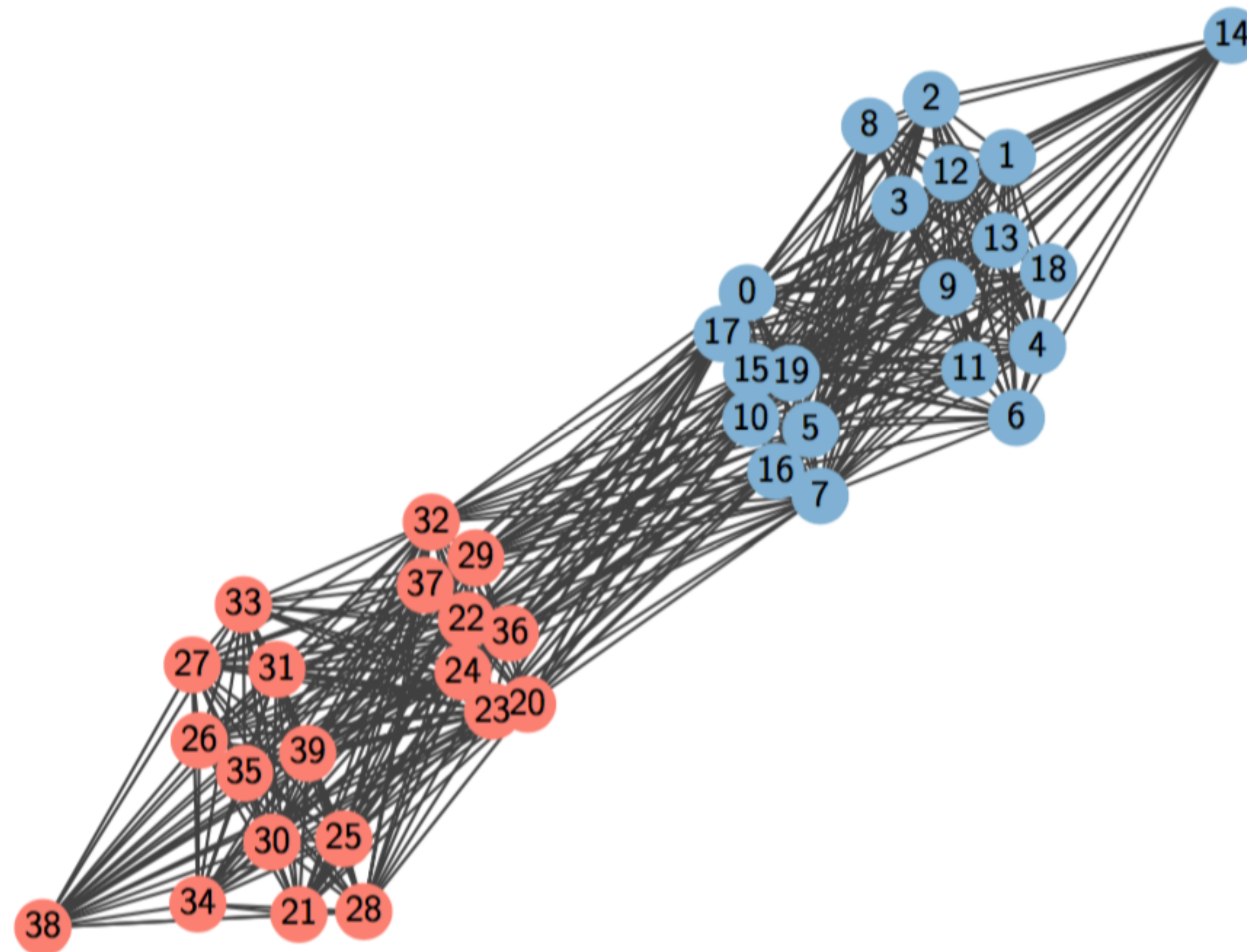
Department of Computer Sciences
University of Wisconsin, Madison, USA

ICML 2007

<http://pages.cs.wisc.edu/~jerryzhu/icml07tutorial.html>

Spectral Clustering

Spectral Clustering



<https://arxiv.org/abs/1708.08436>

Spectral clustering algorithm for graphs. We use the Ng-Jordan-Weiss algorithm [39] to perform spectral clustering of graphs. Let n_0 be the number of vertices in a graph. Recall the *affinity matrix* $A \in \mathbb{R}^{n_0 \times n_0}$ is a matrix where $A_{ij} (\geq 0)$ captures the affinity (i.e. measure of similarity) between vertex i and vertex j . In our setting, A_{ij} corresponds to the weight of edge e_{ij} in the diagonal edge weight matrix W_1 . The spectral clustering algorithm in [39] can be summarized as follows:

1. Compute the diagonal matrix $\Delta \in \mathbb{R}^{n_0 \times n_0}$ with diagonal elements Δ_{ii} being the sum of A 's i -th row, that is, $\Delta_{ii} = \sum_j A_{ij}$.
2. Construct the matrix $M = \Delta^{-1/2} A \Delta^{-1/2}$.
3. Find u_1, u_2, \dots, u_k , the eigenvectors of M corresponding to the k largest eigenvalues (chosen to be orthogonal to each other in the case of repeated eigenvalues), and form the matrix $X = [u_1 u_2 \dots u_k] \in \mathbb{R}^{n_0 \times k}$ by stacking the eigenvectors in columns.
4. Form the matrix Y from X by re-normalizing each of X 's rows to have unit length, that is, $Y_{ij} = X_{ij} / \left(\sum_j X_{ij}^2 \right)^{1/2}$.
5. Treating each row of Y as a point in \mathbb{R}^k , cluster them into k clusters via the k -means algorithm.
6. Finally, assign the original vertex v_i to cluster j if and only if row i of the matrix Y is assigned to cluster j .

Label Propagation

Label propagation on graphs. We implement a simple version of the iterative label propagation algorithm [60] based on the notion of stochastic matrix (i.e. random walk matrix) $P = \Delta^{-1}A$, where A is the affinity matrix and Δ is the diagonal matrix with diagonal elements $\Delta_{ii} = \sum_j a_{ij}$ (as defined in Section 5.1).

The matrix P represents the probability of label transition. Given P and an initial label vector \mathbf{y} , we iteratively multiply the label vector \mathbf{y} by P . If the graph is *label-connected* (i.e. we can always reach a labeled vertex from any unlabeled one), then P^t converges to a stationary distribution, that is, $P^t \mathbf{x} = \mathbf{x}$ for a large enough t .

Suppose there are two label classes $\{+1, -1\}$. Without loss of generality, assume that first l of the n vertices are assigned labels initially, represented as a length- l vector \mathbf{y}_l . Given a graph $G(V, E)$ and labels \mathbf{y}_l , the algorithm is given as:

1. Compute A , Δ , and $P = \Delta^{-1}A$.
2. Initialize $\mathbf{y}^{(0)} = (\mathbf{y}_l, \mathbf{0})$, $t = 0$.
3. Repeat until convergence:

$$\mathbf{y}^{(t+1)} = P\mathbf{y}^{(t)},$$

$$\mathbf{y}_l^{(t+1)} = \mathbf{y}_l^{(t)}.$$

4. Return $\text{sgn}(\mathbf{y}^{(t)})$.

Consider P to be divided into blocks as follows:

$$P = \begin{pmatrix} P_{ll} & P_{lu} \\ P_{ul} & P_{uu} \end{pmatrix}$$

where l and u index the labeled and unlabeled vertices with the number of vertices $n_0 = l + u$. Let $\mathbf{y} = (\mathbf{y}_l, \mathbf{y}_u)$ be the labels at convergence, then \mathbf{y}_u is given by :

$$\mathbf{y}_u = (I - P_{uu})^{-1} P_{ul} \mathbf{y}_l$$

As long as our graph is connected, it is also label-connected and $(I - P_{uu})$ is non-singular. So we can directly compute the labels at convergence without going through the iterative process described above.

Additional Reading

Graph-based Semi-supervised Learning

Zoubin Ghahramani

**Department of Engineering
University of Cambridge, UK**

`zoubin@eng.cam.ac.uk`

`http://learning.eng.cam.ac.uk/zoubin/`

**MLSS 2012
La Palma**

`https://www.youtube.com/watch?v=HZQOvm0fkLA`

`http://mlg.eng.cam.ac.uk/zoubin/talks/lect3ssl.pdf`

Graph-Based ML Plus Visualization

Visualizing Execution of Algorithm

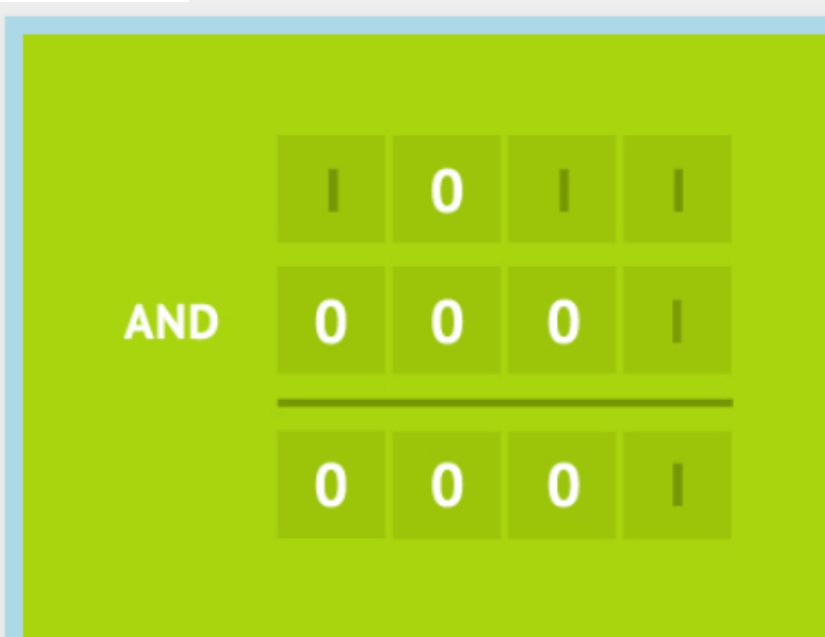
VISUALGO.NET/EN

<https://visualgo.net/en>

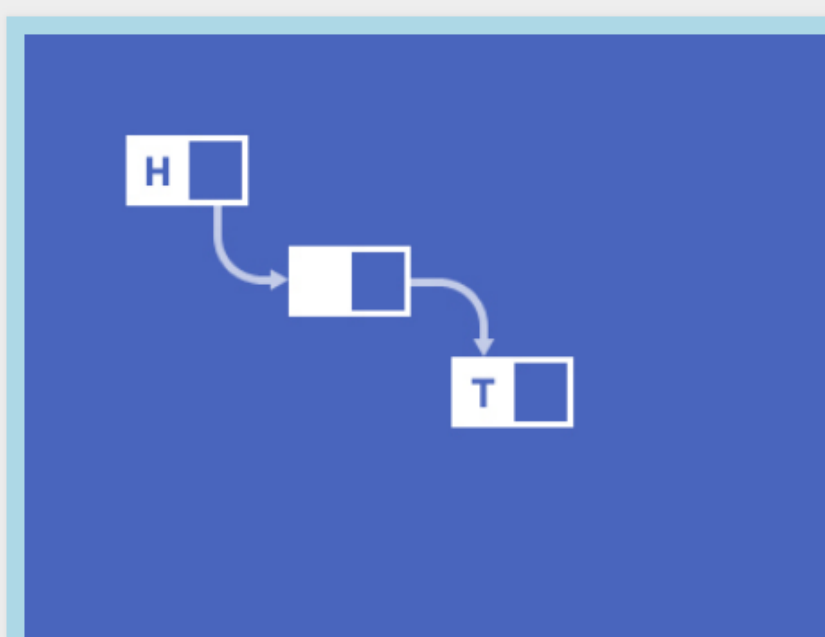
visualising data structures and algorithms through animation



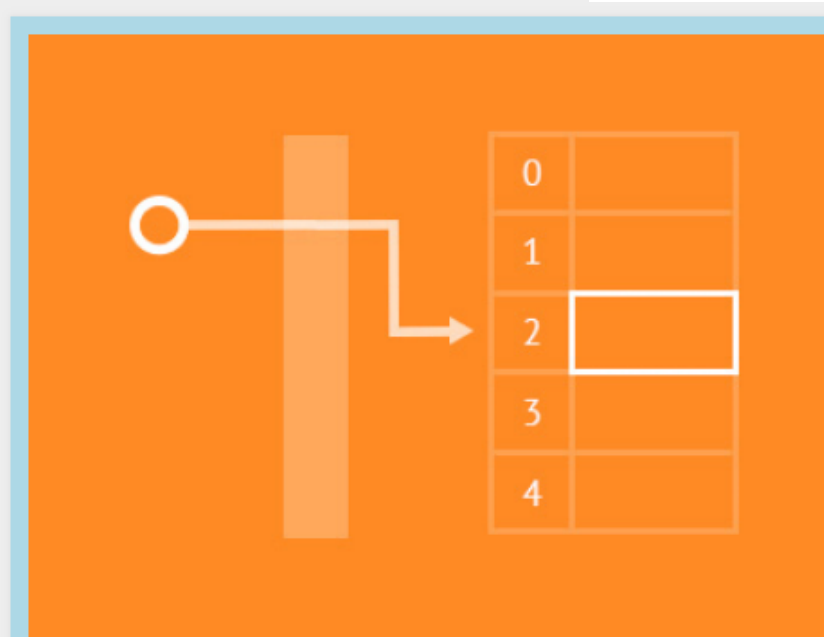
Sorting Training
array algorithm bubble select insert



Bitmask Training
bit manipulation set cs3233 array list ds



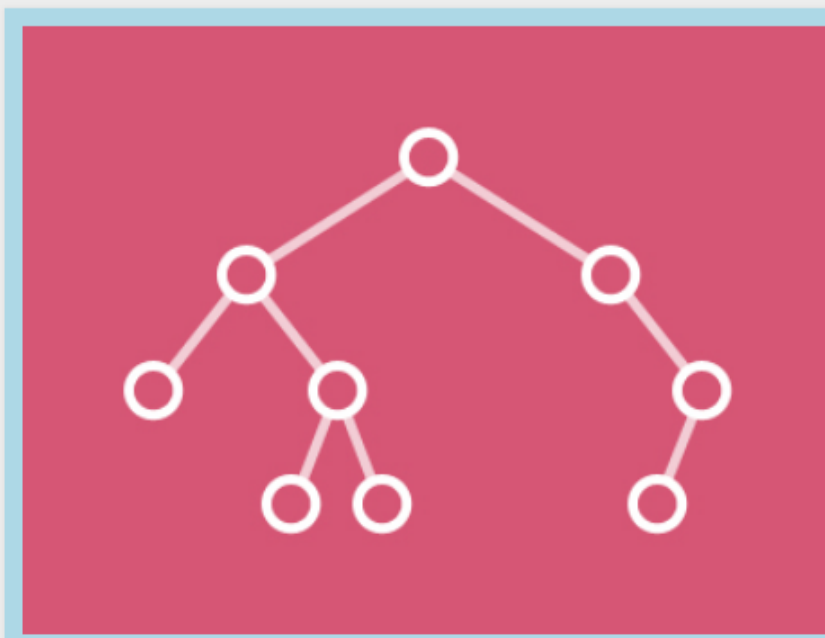
Linked List Training
stack queue doubly deque cs1020 cs2020



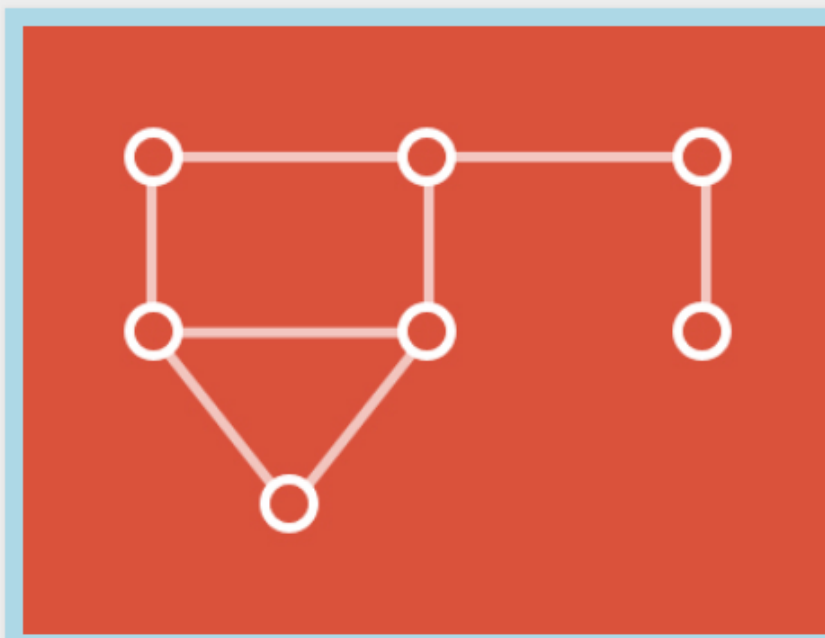
Hash Table Training
open addressing linear quadratic probing



Binary Heap Training
priority queue recursive cs2010 cs2020



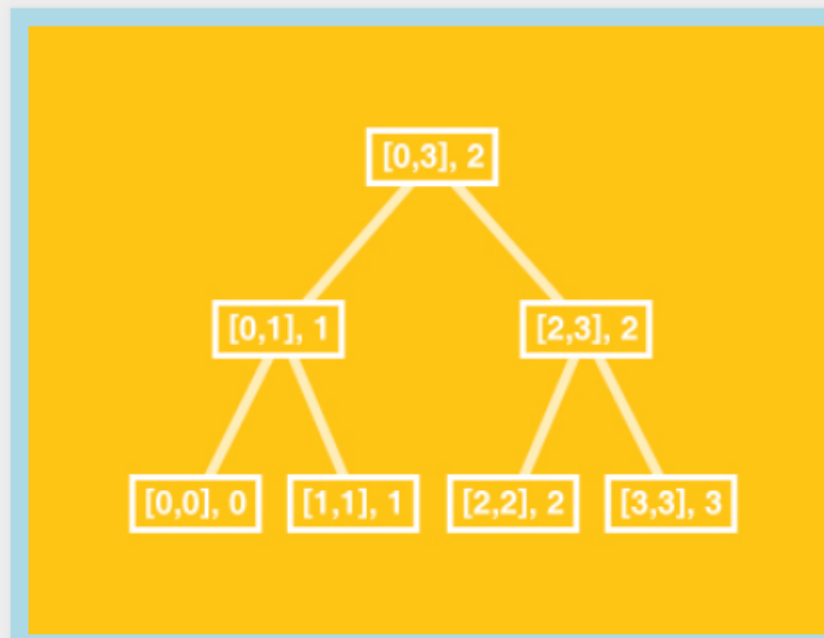
Binary Search Tree Training
adelson velskii landis set table avl cs2010



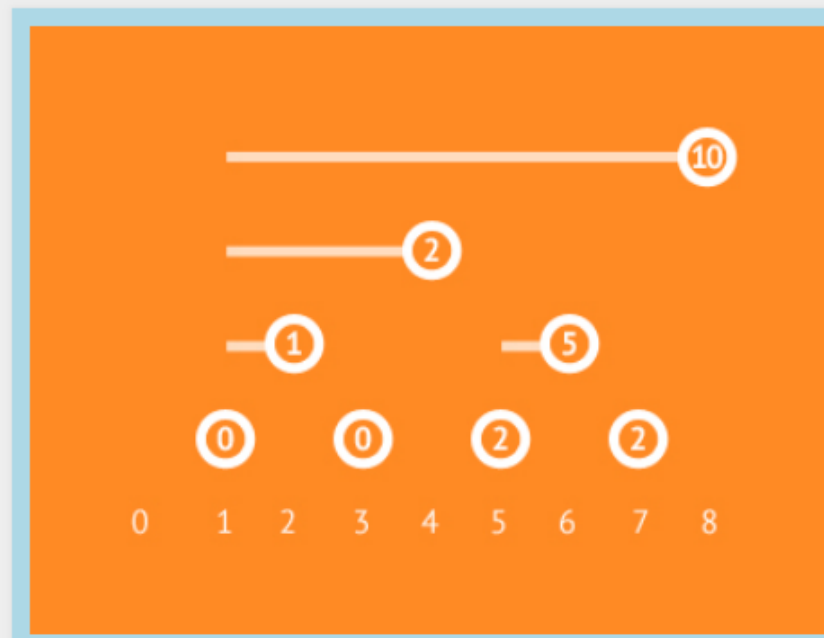
Graph Structures Training
tree complete bipartite dag cs2010 cs2020



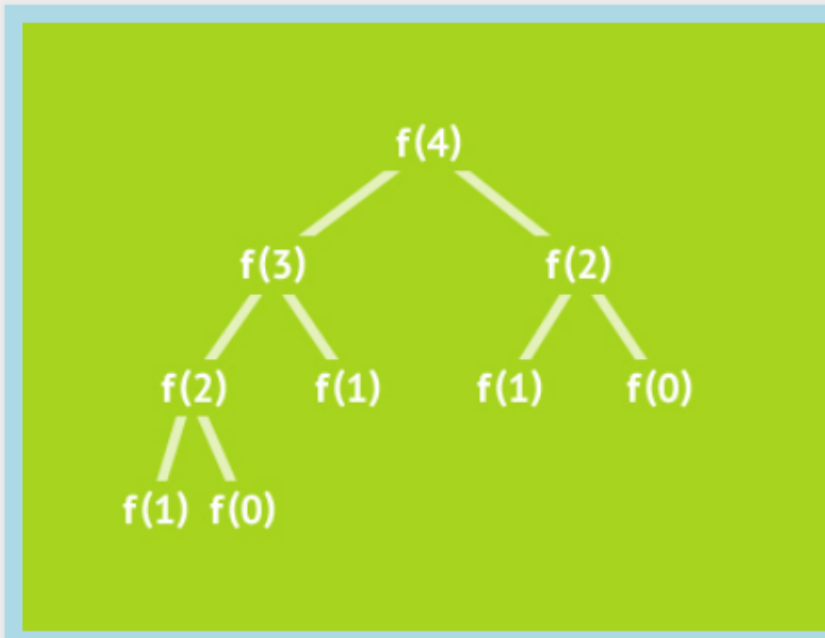
Union-Find DS Training
path compression disjoint set data structure



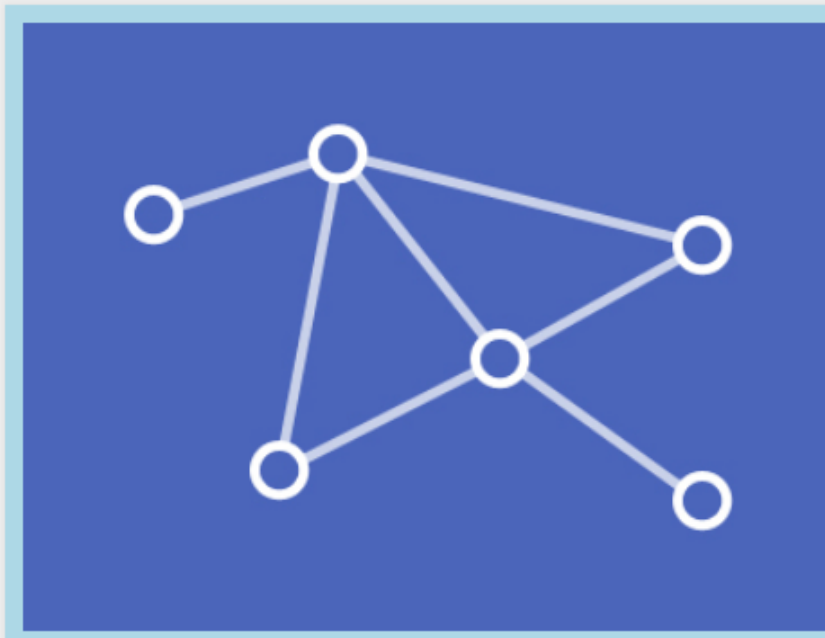
Segment Tree Training
dynamic range sum min max cs3233



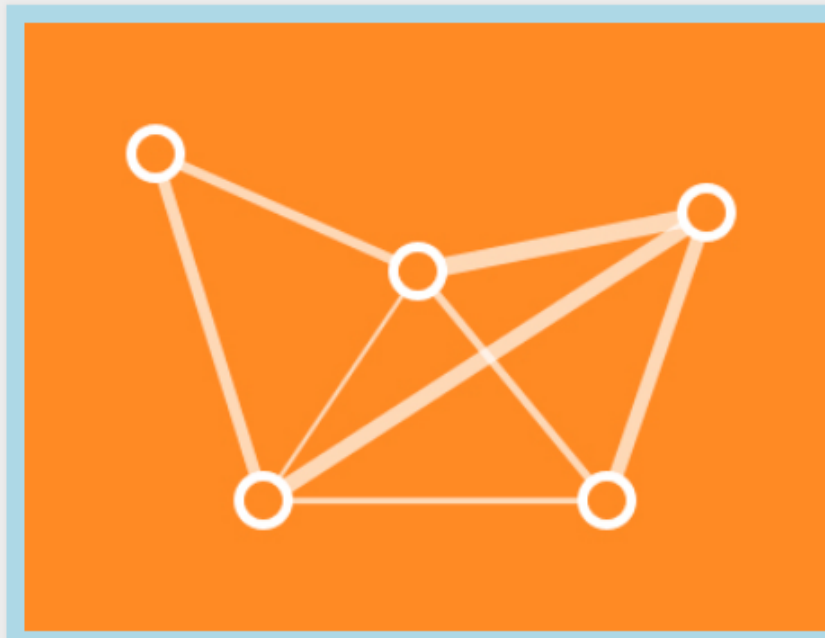
Fenwick Tree Training
binary indexed tree bit dynamic fenwick range



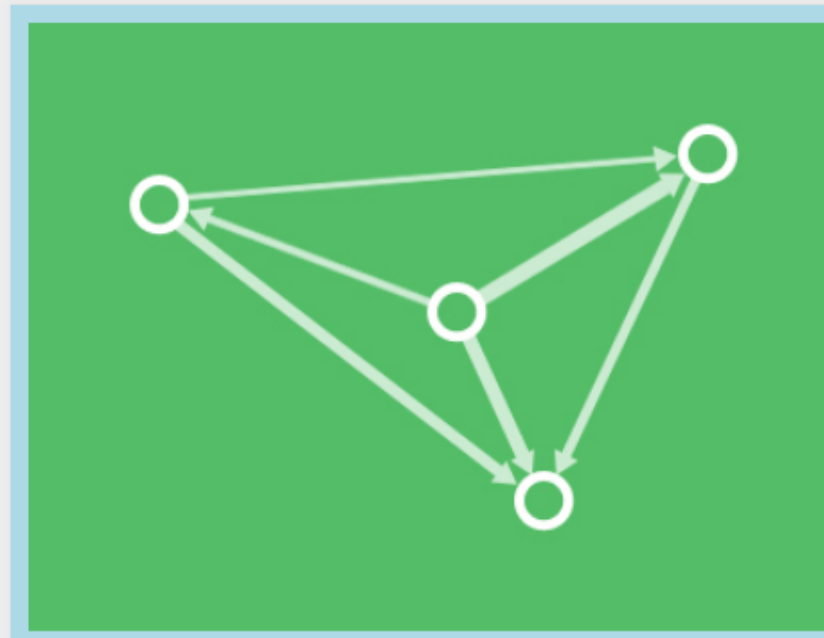
Recursion Tree/DAG Training
dynamic programming dp generic cs1010



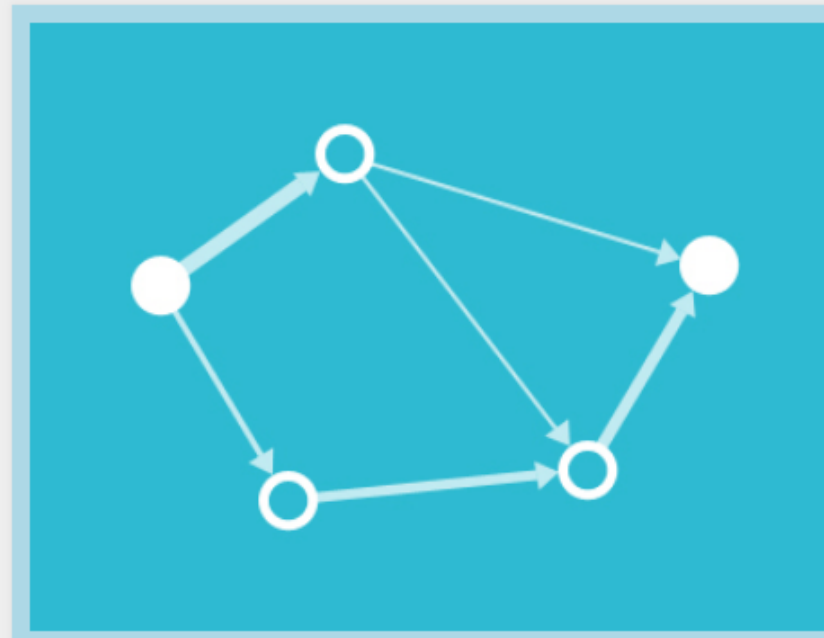
Graph Traversal Training
bfs dfs cs2010 cs2020 cs2040 bipartite



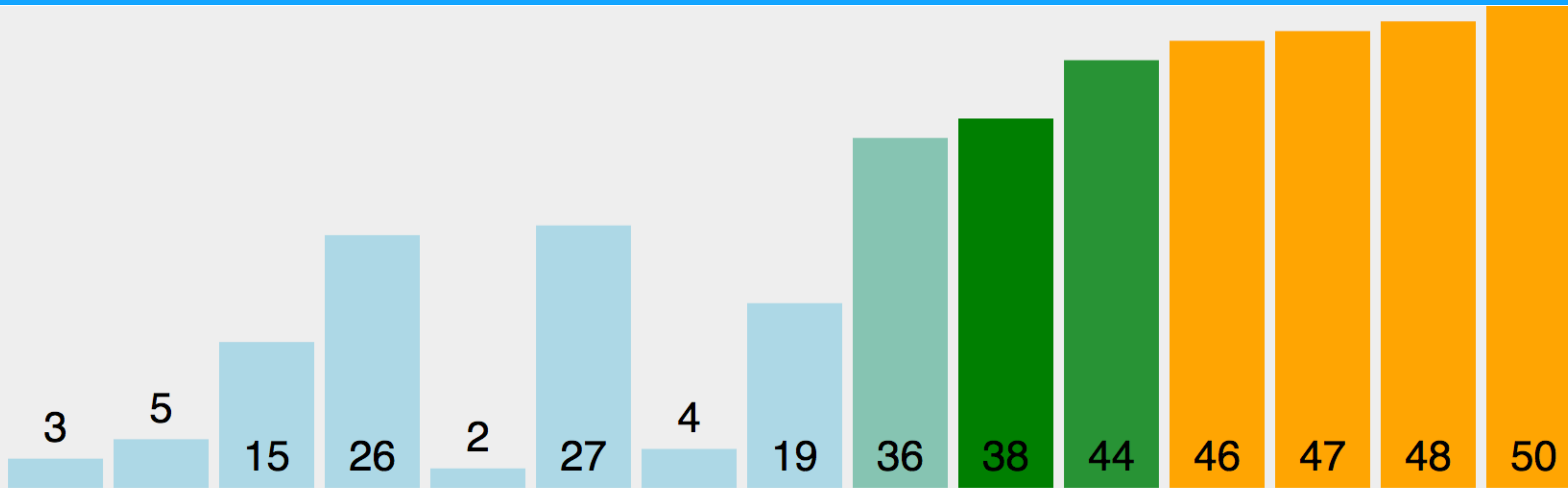
Min Spanning Tree Training
mst prim kruskal graph min spanning



SS Shortest Paths Training
sssp single-source bfs dijkstra bellman ford



Network Flow Training
max flow edmonds karp min cut dinic

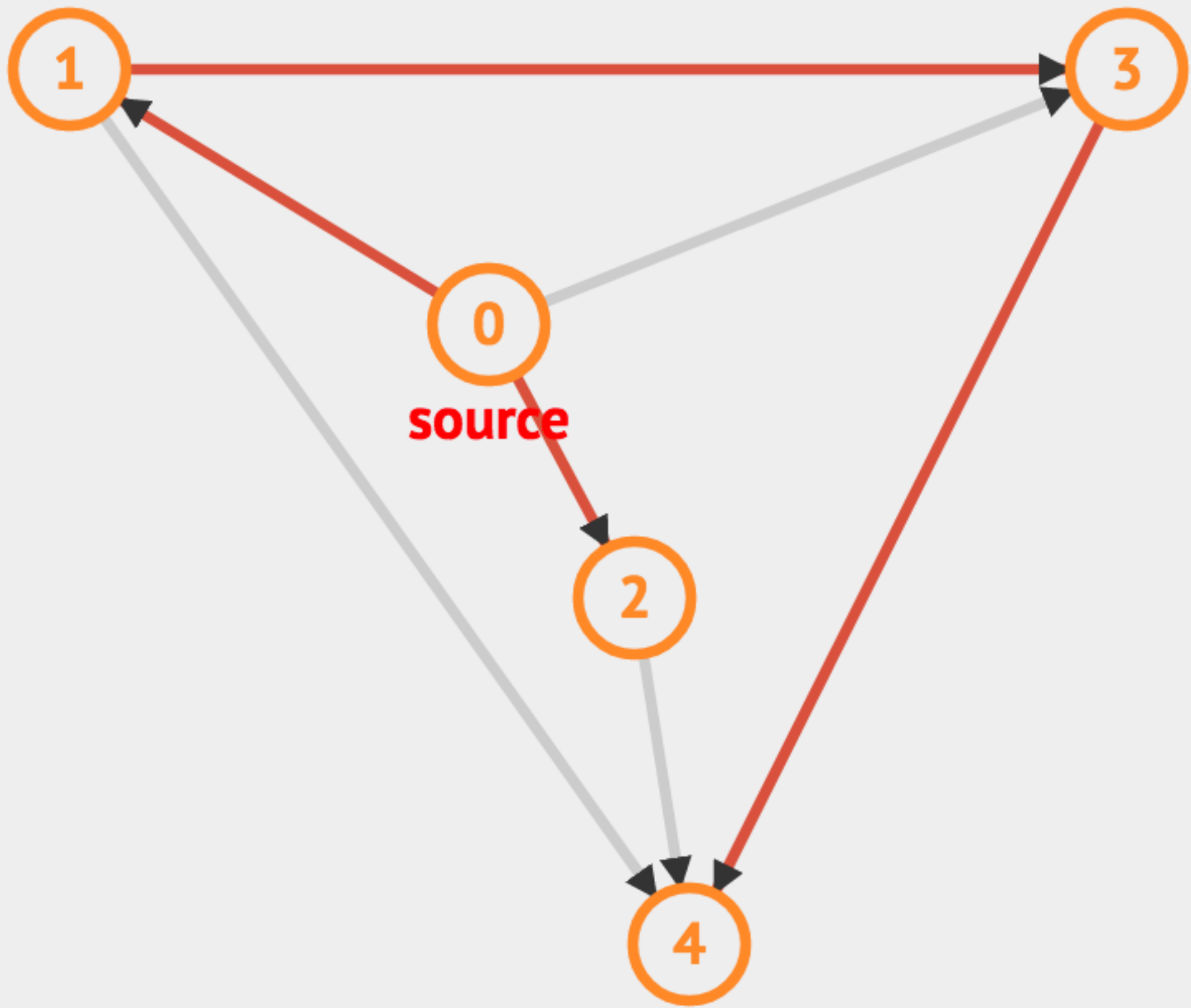


Bubble Sort

Checking if $38 > 44$ and swap them if that is true.
The current value of swapped = true.

```
do
  swapped = false
  for i = 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap(leftElement, rightElement)
      swapped = true
  while swapped
```

<https://visualgo.net/en/sorting>



DFS (0)

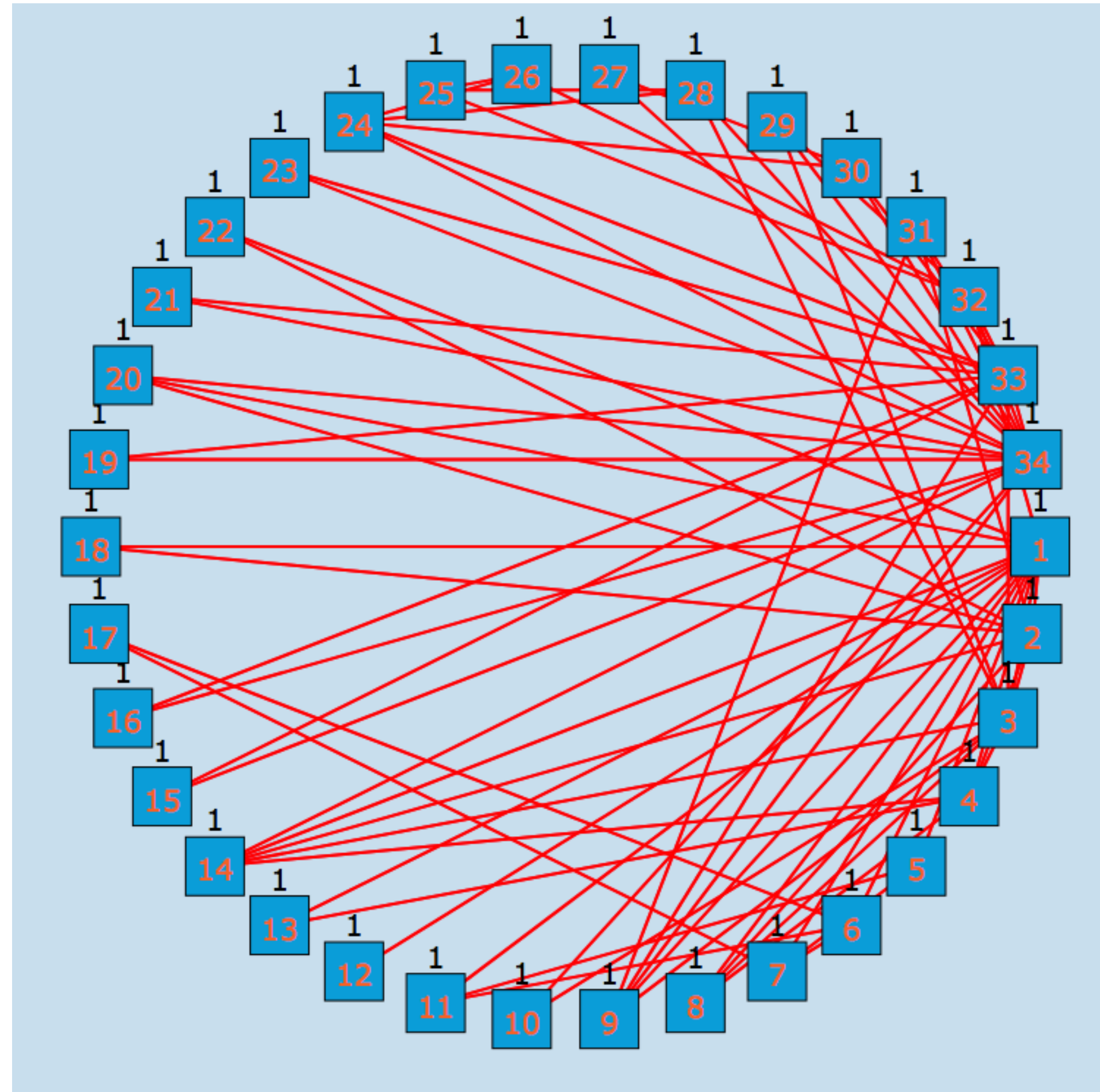
DFS(0) is completed. Red/grey/blue edge is tree/cross/forward/back edge of the DFS spanning tree, respectively.

```
DFS (u)
for each neighbor v of u
    if v is unvisited, tree edge, DFS (v)
    else if v is explored, bidirectional / back edge
    else if is visited, forward / cross edge
// ch4_01_dfs.cpp / java, ch4, CP3
```

<https://visualgo.net/de/dfsdfs>

Visualizing LP?

- Visualization of random walks on graphs?





Thanks!

Any questions?

You can find me at: beiwang@sci.utah.edu

CREDITS

Special thanks to all people who made and share these awesome resources for free:

- ☐ Presentation template designed by [Slidesmash](#)
- ☐ Photographs by [unsplash.com](#) and [pexels.com](#)
- ☐ Vector Icons by [Matthew Skiles](#)

Presentation Design

This presentation uses the following typographies and colors:

Free Fonts used:

<http://www.1001fonts.com/oswald-font.html>

<https://www.fontsquirrel.com/fonts/open-sans>

Colors used

