Comp790-166: Computational Biology

Lecture 3

January 19, 2022

Announcements

- slack channel $\rightarrow$ check your email
- Reminder that our course page for all notes and suggested reading is `https://github.com/natalies-teaching/Comp790-166-CompBio-Spring2022`
- We will transition from relevant background to more research based paper soon, so keep your reading summaries in mind!

Today

- Finish our diffusion and Laplacian bridge from last time

Today

- Finish our diffusion and Laplacian bridge from last time
- k nearest neighbor graphs as a very simple graph that can be built from data

Today

- Finish our diffusion and Laplacian bridge from last time
- k nearest neighbor graphs as a very simple graph that can be built from data
- Visualizing Graphs
  - How can we use the graph structure to infer the ideal 2d coordinates for nodes?

Today

- Finish our diffusion and Laplacian bridge from last time
- k nearest neighbor graphs as a very simple graph that can be built from data
- Visualizing Graphs
  - How can we use the graph structure to infer the ideal 2d coordinates for nodes?
- Graph partitioning (over today and Monday)
  - Modularity-based
  - probabilistic via stochastic block model
  - probabilistic via affiliation model
  - Mining biological networks

Setup for Graph Diffusion

Diffusion offers the chance to study how some quantity (e.g. signal) relates to connectivity in the graph. Here we assume that some quantity can only flow to adjacent nodes.

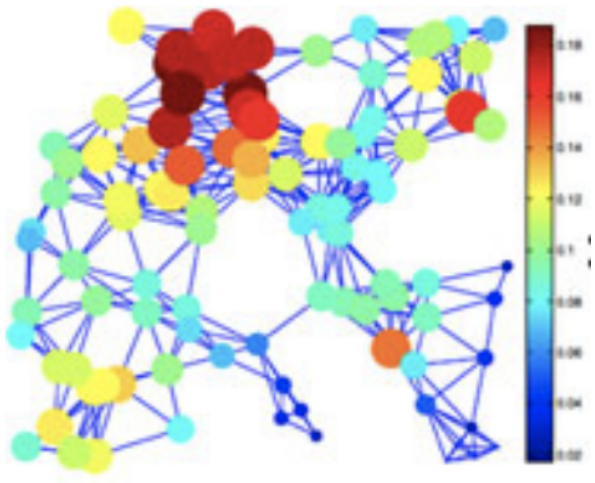


Figure: from `https://web.media.mit.edu/~xdong/paper/tsipn17.pdf`

Coming Attraction

Why should we care about this? Single-cell people should care! :D

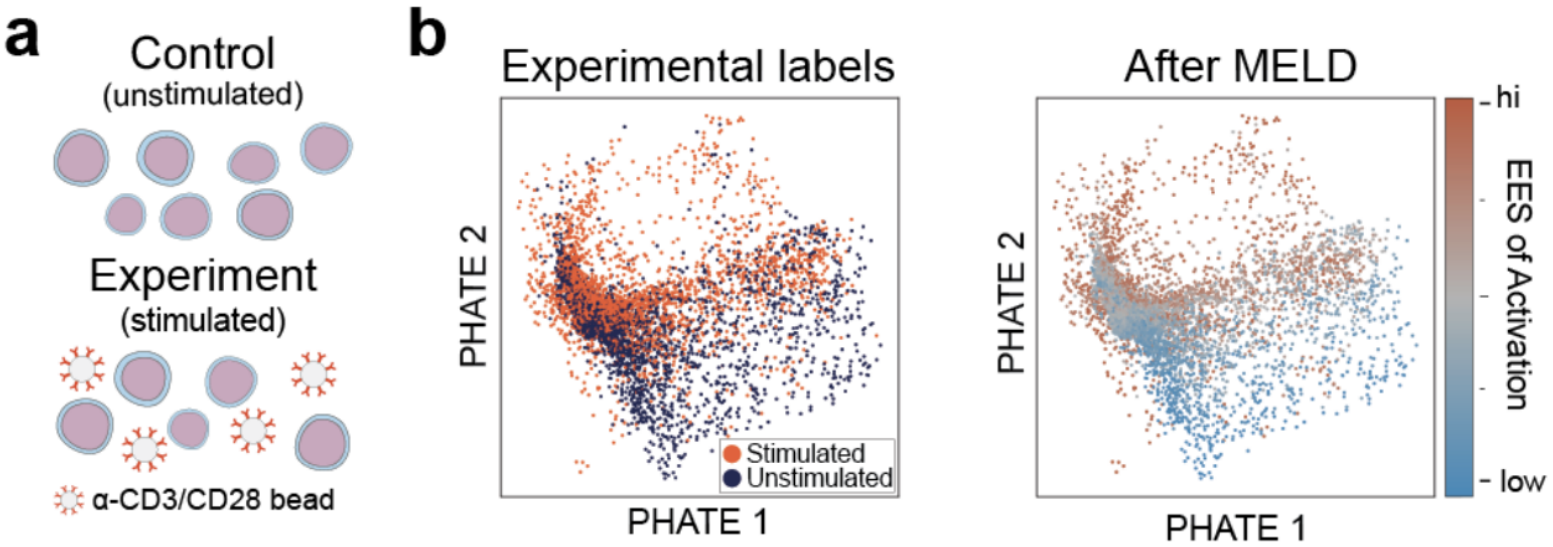


Figure: from `https://www.biorxiv.org/content/10.1101/532846v2`

Diffusion on Graphs

- Suppose we have some quantity that will spread or *diffuse* across a graph, given the graph's connectivity. Suppose there is a quantity of ψ_i at node i.

Diffusion on Graphs

- Suppose we have some quantity that will spread or *diffuse* across a graph, given the graph's connectivity. Suppose there is a quantity of ψ_i at node i.
- Suppose that this quantity flows along the graph from node j to an adjacent node i at a rate of $C(\psi_j - \psi_i)$. The rate at which ψ_i is changing is given as,

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C \sum_j A_{ij}(\psi_j - \psi_i) \tag{1}$$

Here, A_{ij} enforces the idea that the only terms in a sum are pairs of nodes connected by an edge.

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C \sum_j A_{ij}(\psi_j - \psi_i) \tag{2}$$

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j A_{ij}(\psi_j - \psi_i) \tag{2}$$

Splitting up terms, we can rewrite this as,

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j A_{ij}\psi_j - C\psi_i\sum_j A_{ij} = C\sum_j A_{ij}\psi_j - C\psi_i k_i \tag{3}$$

- Remember, $k_i = \sum_j A_{ij}$ is simply the degree of node i

Diffusion continued

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j A_{ij}\psi_j - C\psi_i k_i \tag{4}$$

further simplifies to,

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j (A_{ij} - \delta_{ij}k_j)\psi_j. \tag{5}$$

Here, k_i is the degree of node i and $\delta_{ij} = 1$ if $i = j$ and is 0 otherwise.

- That was just some massaging so that everything can be written as a $\sum_j$ to make our next step easier.

Massaging Explained

In case you were wondering how we reworked $C\psi_i k_i$ to be written in terms of $\sum_j$ as, $C \sum_j \delta_{ij} \psi_j k_j$

- We only need to count it for node i, so δ_{ij} is our indicator for indexing over j that ensures that we only add a non-zero value when $i = j$

Diffusion and Connection with the Graph Laplacian

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j (A_{ij} - \delta_{ij}k_j)\psi_j. \tag{6}$$

.

- This simplifies because $C\psi_i k_i = C\sum_j \delta_{ij}\psi_j k_j$

Diffusion and Connection with the Graph Laplacian

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j (A_{ij} - \delta_{ij}k_j)\psi_j. \tag{6}$$

.

- This simplifies because $C\psi_i k_i = C\sum_j \delta_{ij}\psi_j k_j$

This can further can be expressed in matrix form as,

$$\frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}t} = C(\mathbf{A} - \mathbf{D})\boldsymbol{\psi}. \tag{7}$$

Here, $\mathbf{D}$ is an $N \times N$ diagonal matrix where k_i is the ith diagonal element and represents the degree of node i.

Diffusion and Connection with the Graph Laplacian

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j (A_{ij} - \delta_{ij}k_j)\psi_j. \tag{6}$$

.

- This simplifies because $C\psi_i k_i = C\sum_j \delta_{ij}\psi_j k_j$

This can further can be expressed in matrix form as,

$$\frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}t} = C(\mathbf{A} - \mathbf{D})\boldsymbol{\psi}. \tag{7}$$

Here, $\mathbf{D}$ is an $N \times N$ diagonal matrix where k_i is the ith diagonal element and represents the degree of node i.

- The **Graph Laplacian** is the matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$, such that, $\frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}t} + C\mathbf{L}\boldsymbol{\psi} = 0$ (aka the diffusion equation).

Diffusion on a Bunny Using its Graph Representation

A small amount of signal spreads locally according to specified edges and takes many time steps to spread across the bunny.

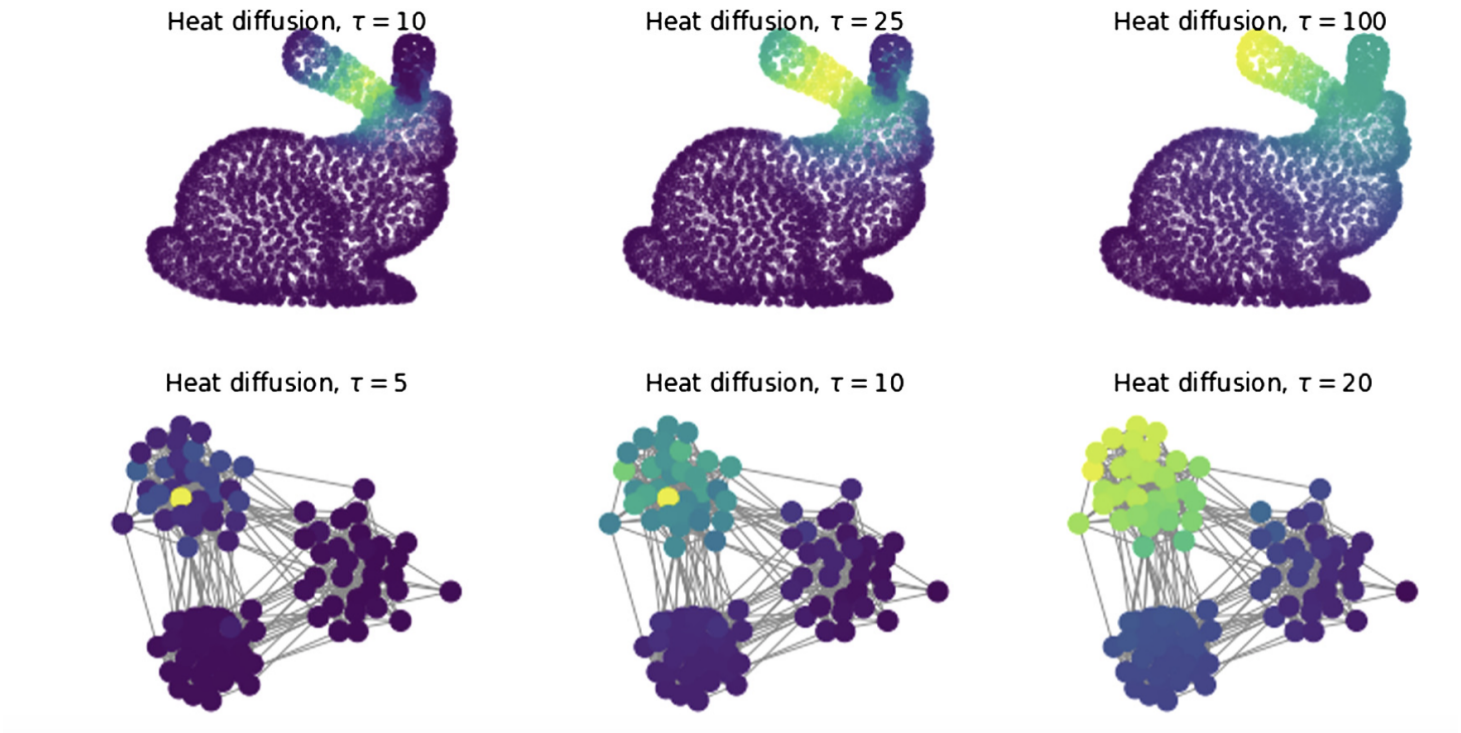


Figure: from `https://www.sciencedirect.com/science/article/pii/S1631070519301094`

Welcome Graph Laplacian

- Besides describing diffusion, the Graph Laplacian has many other nice properties.

Welcome Graph Laplacian

- Besides describing diffusion, the Graph Laplacian has many other nice properties.
- Notably, properties about **connectivity patterns** in the graph and can be useful for **graph partitioning**.

Welcome Graph Laplacian

- Besides describing diffusion, the Graph Laplacian has many other nice properties.
- Notably, properties about **connectivity patterns** in the graph and can be useful for **graph partitioning**.
- We are going to be using the graph Laplacian for many tasks!
  - For example, finding single-cell measurements that are prototypical of a condition of interest or denoising high-dimensional genomic data

Laplacian Matrix in More Detail

The elements of the Laplacian matrix are defined as follows for an undirected, unweighted matrix.

- Remember: $\mathbf{L} = \mathbf{D} - \mathbf{A}$

$$L_{ij} = \begin{cases} k_i & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and there is an edge } (i, j) \\ 0 & \text{otherwise} \end{cases}$$

There is also a weighted version of the Graph Laplacian

The normalized Graph Laplacian, L_N is defined as,

$$\mathbf{L}_n = \mathbf{D}^{-1/2}\mathbf{L}_u\mathbf{D}^{-1/2}$$

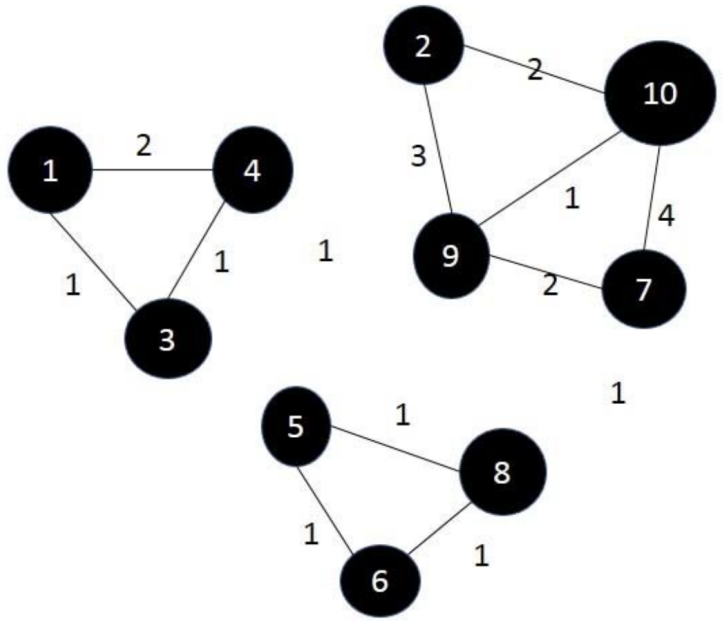
.

Here, L_u is the un-normalized Graph Laplacian.

The Graph Laplacian Has Many Nice Spectal Properties

One easy one...

- In a network with c components, the number of zero eigenvalues is equal to the number of components, (c). (Numerically, you may have more than that if there are very few connections between pairs of components and hence the eigenvalue is either very close to 0 or numerically 0.)



Building graphs from data

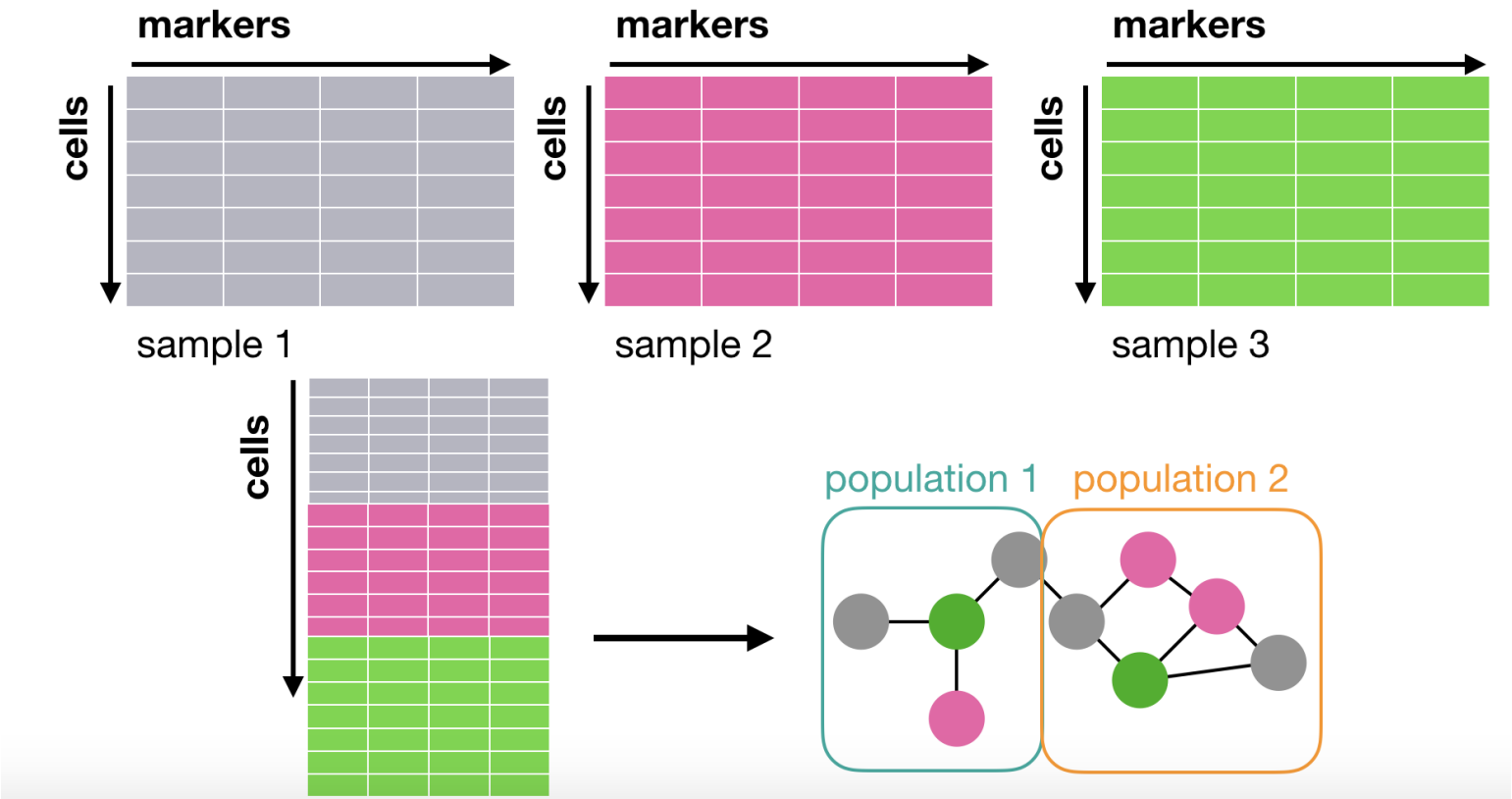


Figure: The task is to compute distances between all pairs of cells, and connect each cell with their nearest neighbors

k-Nearest Neighbor Graphs

- Suppose our dataset, $\mathcal{X}$ is defined as $\mathcal{X} = \{\mathbf{x}_i\}_{i=1\ldots N}, \mathbf{x}_i \in \mathbb{R}^d$.
- The general idea is to connect each node with its k nearest neighbors
  - Compute all pairwise similarities between all pairs of nodes
  - For a node, i, find the k nodes that are closest and draw edges.
- Computing all pairwise distances quickly becomes expensive ($O(N^2 d)$).

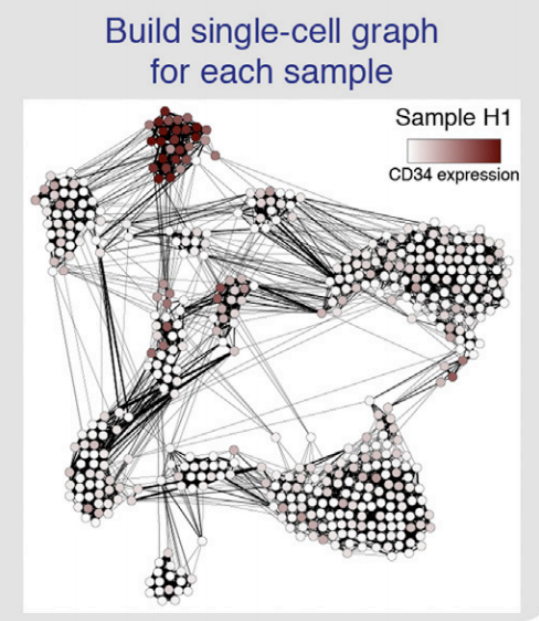


Figure: from Levine *et al.* Cell. 2015. Each cell is connected to its k-nearest neighbors.

Question for you: what do you think are the trade-offs for larger vs smaller k (for either application or computational reasons)?

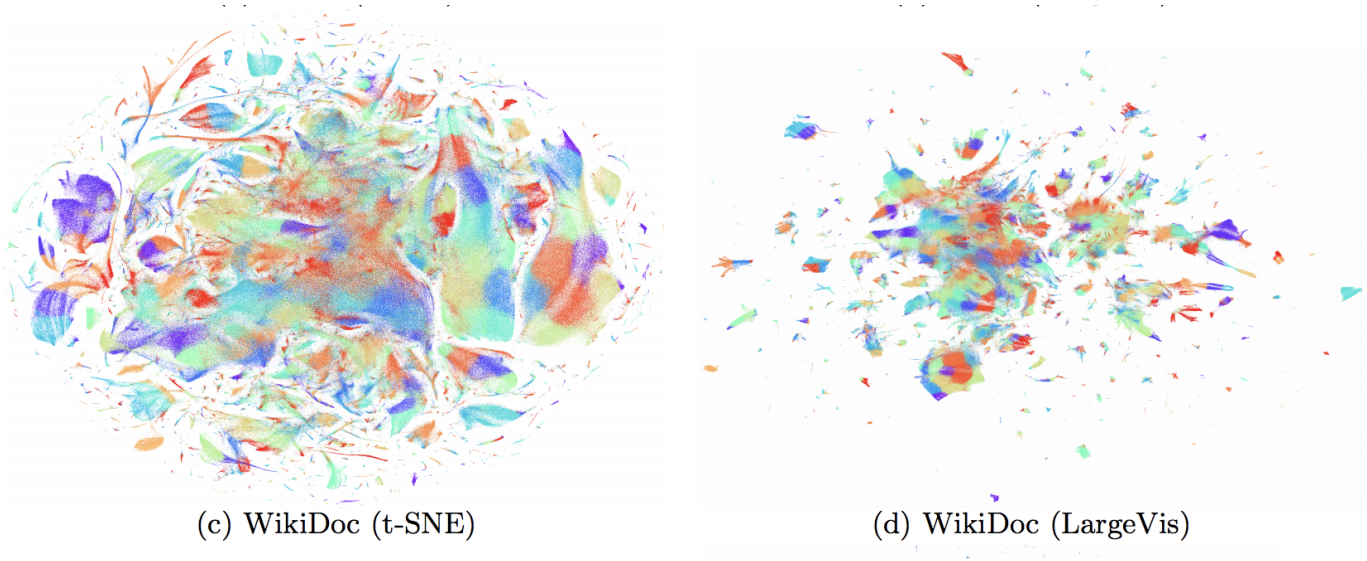
A Common way to Calculate Edge Weights in a Graph

For better or for worse, people commonly assume a Gaussian distribution on pairwise similarities between nodes. So, a weight between nodes i and j is often computed as,

$$W_{ij} = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \tag{8}$$

One Application of k-NN Graph is Visualization (LargeVis)

Using random projection trees, LagreVis creates an embedding for a set of points with a probabilistic model.



(c) WikiDoc (t-SNE)

(d) WikiDoc (LargeVis)

Figure: from Tang *et al.* The Web Conference (WWW). 2016. tSNE and 'string behavior' from being designed to preserve super local similarities.

A Probabilistic Model for Graph Embedding

For some nodes, i and j, let $\mathbf{y}_i$ and $\mathbf{y}_j$ be their embedding coordinates such that $\mathbf{y}_i, \mathbf{y}_j \in \mathbb{R}^d$ (for visualization purposes, $d = 2$). We can model the probability that an edge exists between nodes i and j as a function ($f(\cdot)$) of their learned embedding coordinates, or,

$$P(e_{ij} = 1) = f(||\mathbf{y}_i - \mathbf{y}_j||) \tag{9}$$

A Probabilistic Model for Graph Embedding

For some nodes, i and j, let $\mathbf{y}_i$ and $\mathbf{y}_j$ be their embedding coordinates such that $\mathbf{y}_i, \mathbf{y}_j \in \mathbb{R}^d$ (for visualization purposes, $d = 2$). We can model the probability that an edge exists between nodes i and j as a function ($f(\cdot)$) of their learned embedding coordinates, or,

$$P(e_{ij} = 1) = f(||\mathbf{y}_i - \mathbf{y}_j||) \tag{9}$$

- An important criterion to be satisfied is that $f(\cdot)$ must ensure a high probability when i and j are close, and low otherwise.
- For example: $f(x) = \frac{1}{1+ax^2}$

Writing Down the Likelihood of the Observed Graph

The authors define a similarity measure between pairs of nodes where there is an edge, based on the original data, $\mathbf{x}_i$ and $\mathbf{x}_j$. Here, γ being a constant fixed weight for negative edges.

Writing Down the Likelihood of the Observed Graph

The authors define a similarity measure between pairs of nodes where there is an edge, based on the original data, $\mathbf{x}_i$ and $\mathbf{x}_j$. Here, γ being a constant fixed weight for negative edges.
The likelihood of the graph can be written as,

$$O = \prod_{(i,j)\in E} P(e_{ij} = 1) \prod_{(i,j)\in \tilde{E}} (1 - P(e_{ij} = 1))^{\gamma}$$
$$\propto \sum_{(i,j)\in E} \log P(e_{ij} = 1) + \sum_{(i,j)\in \tilde{E}} \gamma \log(1 - P(e_{ij} = 1))$$

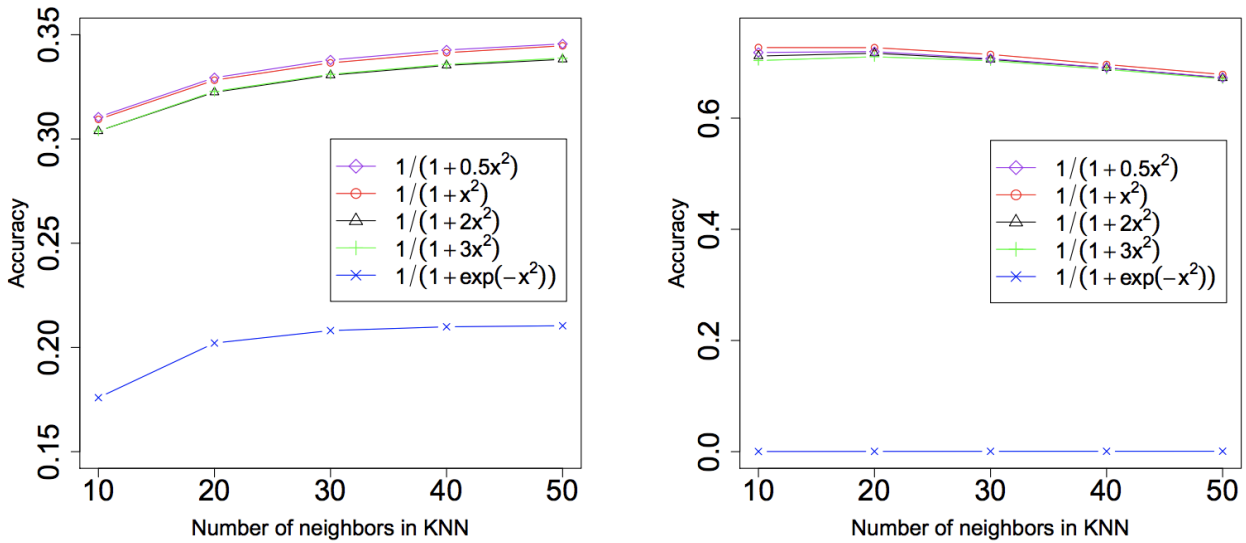
$\tilde{E}$ represents pairs of nodes with no edge.

Brief Overview of Optimization of Parameters

- Since these graphs are very sparse, there are a lot of 'negative edges' or edges $\in \tilde{E}$. So the authors used a smart negative sampling approach to only consider a subset of negative node pairs
  - For each node, i, consider sample some 'negative' pair nodes.
  - A node, j from the negative pair set to pair with i as a negative example is sampled with probability $\propto d_j^{0.75}$. Here d_j is the degree of node j.
  - Optimize parameters with asynchronous stochastic gradient descent (`https://papers.nips.cc/paper/2011/file/218a0aefd1d1a4be65601cc6ddc1520e-Paper.pdf`)

Choice of $f(\cdot)$

The authors tested how choice of $f(\cdot)$ affected the quality of their embedding. Looks like there is one bad choice, but overall, does not make a huge difference.



(a) WikiDoc (b) LiveJournal

Figure: from Tang *et al.* The Web Conference (WWW). 2016.

More Practical Reasons to Use LargeVis: Scales better than tSNE

Nothing would ever be as fast as PCA. But, if you need to understand more local types of similarities between data points, perhaps it's worth the wait....

Table 2: Comparison of running time (hours) in graph visualization between the t-SNE and LargeVis.

Algorithm	20NG	MNIST	WikiWord	WikiDoc	LiveJournal	CSAuthor	DBLPPaper
t-SNE	0.12	0.41	9.82	45.01	70.35	28.33	18.73
LargeVis	0.14	0.23	2.01	5.60	9.26	4.24	3.19
Speedup Rate	0	0.7	3.9	7	6.6	5.7	4.9

Figure: from Tang *et al.* The Web Conference (WWW). 2016. Comparison in run-time between tSNE and LargeVis.

Reminder of Steps in LargeVis

- Construct k-NN graph based on original data, $\mathbf{X}$

Reminder of Steps in LargeVis

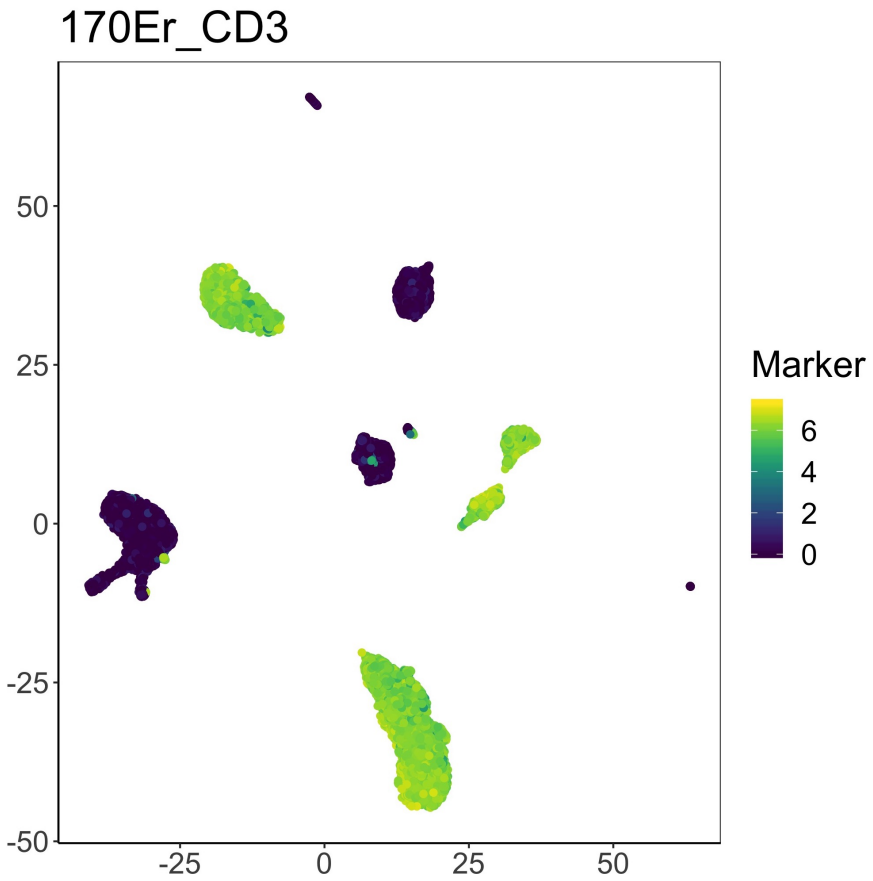
- Construct k-NN graph based on original data, $\mathbf{X}$
- After building the graph, project all nodes into a $\mathbb{R}^d$, with the learned coordinates of each node encoded in $\mathbf{Y}$.

Reminder of Steps in LargeVis

- Construct k-NN graph based on original data, $\mathbf{X}$
- After building the graph, project all nodes into a $\mathbb{R}^d$, with the learned coordinates of each node encoded in $\mathbf{Y}$.
- The probability of connection between nodes i and j was modeled as $f(||\mathbf{y}_i - \mathbf{y}_j||)$

Example Use Case- Single Cell Data!

Your input, $\mathbf{X}$ is the cells $\times$ marker matrix. Not only do you get the graph out, but we can easily find T-cells and different T-cell subsets!



Community Detection Illustrated

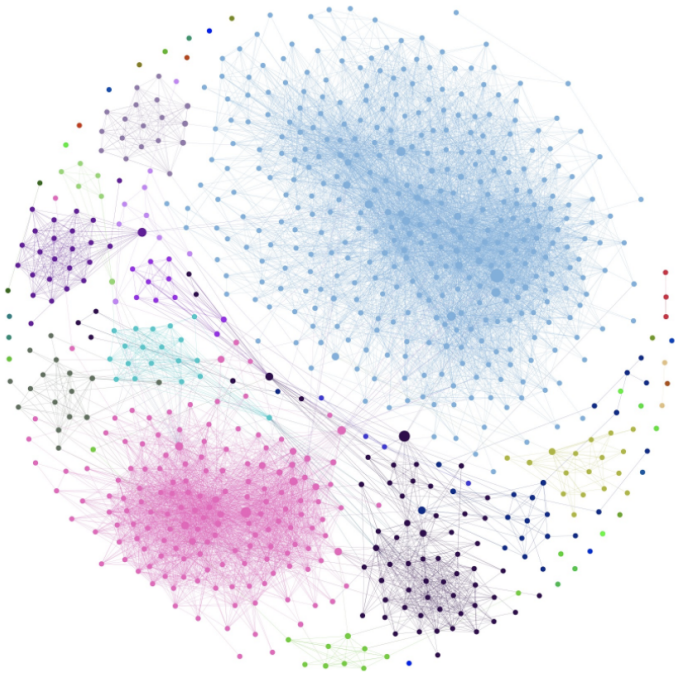


Figure: The high-level goal is to partition nodes into coherent node-subsets (communities), such that nodes within a community have on average more within group edges than between-group edges.

Community Detection (Graph Partitioning) is just Clustering

- Instead of your standard clustering problem in a matrix of N objects with P features, our input here is an adjacency matrix, $\mathbf{A}$, where we want to cluster nodes based on similarities in their connectivity patterns.

Community Detection (Graph Partitioning) is just Clustering

- Instead of your standard clustering problem in a matrix of N objects with P features, our input here is an adjacency matrix, $\mathbf{A}$, where we want to cluster nodes based on similarities in their connectivity patterns.
- Most community detection optimization problems seek a hard partition (each node assigned to a single community)

Community Detection (Graph Partitioning) is just Clustering

- Instead of your standard clustering problem in a matrix of N objects with P features, our input here is an adjacency matrix, $\mathbf{A}$, where we want to cluster nodes based on similarities in their connectivity patterns.
- Most community detection optimization problems seek a hard partition (each node assigned to a single community)
- There are some variants that learn a soft partition, or a propensity or probability that each node is assigned to each community.

Community Detection (Graph Partitioning) is just Clustering

- Instead of your standard clustering problem in a matrix of N objects with P features, our input here is an adjacency matrix, $\mathbf{A}$, where we want to cluster nodes based on similarities in their connectivity patterns.
- Most community detection optimization problems seek a hard partition (each node assigned to a single community)
- There are some variants that learn a soft partition, or a propensity or probability that each node is assigned to each community.
- Optimization for this problem can come in many flavors

Optimization Approaches

- Quality Function with a Null Model + Heuristic for Optimization $\leftarrow$
  - A null model describes a graph with no structure, for example, nodes connected randomly.

Optimization Approaches

- Quality Function with a Null Model + Heuristic for Optimization $\leftarrow$
  - A null model describes a graph with no structure, for example, nodes connected randomly.
- Probabilistic and Likelihood Optimization $\leftarrow$

Optimization Approaches

- Quality Function with a Null Model + Heuristic for Optimization ←
  - A null model describes a graph with no structure, for example, nodes connected randomly.
- Probabilistic and Likelihood Optimization ←
- Spectral Clustering Methods (Partition based on graph Lacplacian)

Optimization Approaches

- Quality Function with a Null Model + Heuristic for Optimization $\leftarrow$
  - A null model describes a graph with no structure, for example, nodes connected randomly.
- Probabilistic and Likelihood Optimization $\leftarrow$
- Spectral Clustering Methods (Partition based on graph Lacplacian)
- Most recently: graph embedding + clustering on embedding
  - We will touch on this briefly Thursday with node2vec

Why Might a Person Waste Time Partitioning a Graph?

Build a graph between cells based on marker expression and partition into cell-populations. Members of a cell-population should be phenotypically similar and therefore express the same markers.

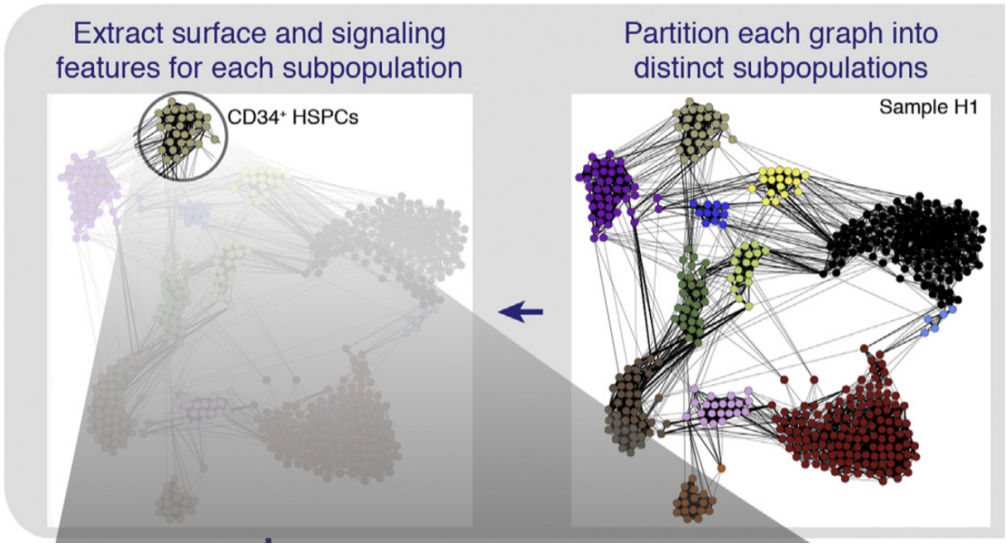


Figure: from Levine *et al.* Cell. 2015. This is the PhenoGraph algorithm.

PhenoGraph Uses Modularity Based Maximization

- **Intuition:** We first specify a null model of a network with no structure about the probability of two nodes being connected.Then we find the partition that is maximally different from this null model.

PhenoGraph Uses Modularity Based Maximization

- **Intuition:** We first specify a null model of a network with no structure about the probability of two nodes being connected.Then we find the partition that is maximally different from this null model.
- **A Simple Null Model:** An easy way to think about the probability of two edges being connected is based on some function of their degree.
  - Consider the null model, $\frac{k_i k_j}{2M}$
  - k_i and k_j give the degrees of nodes i and j
  - M (as always in our notation) is the total number of edges, or the sum of all edge weights.

PhenoGraph Uses Modularity Based Maximization

- **Intuition:** We first specify a null model of a network with no structure about the probability of two nodes being connected.Then we find the partition that is maximally different from this null model.
- **A Simple Null Model:** An easy way to think about the probability of two edges being connected is based on some function of their degree.
  - Consider the null model, $\frac{k_i k_j}{2M}$
  - k_i and k_j give the degrees of nodes i and j
  - M (as always in our notation) is the total number of edges, or the sum of all edge weights.
- **Configuration Model:** This is known as the configuration model, or fixing the degree sequence and connecting nodes at random.

Modularity Defined

$$Q = \frac{1}{2M} \sum_{i,j} [A_{ij} - \gamma \frac{k_i k_j}{2M}] \delta(c_i, c_j) \tag{10}$$

- A_{ij} is the adjacency matrix entry for node pair (i, j) (can be weighted and not just binary)
- $\delta(c_i, c_j)$ is an indicator function for whether or not nodes i and j were assigned to the same community.
- We need an algorithm to help us determine node-to-community assignments for all nodes i, such that Q is as large as possible.
- γ is a resolution parameter controlling the size of communities

Louvain: A Simple Algorithm that Makes a lot of Sense

Merge (if modularity increases), agglomerate, repeat until modularity doesn't increase anymore.

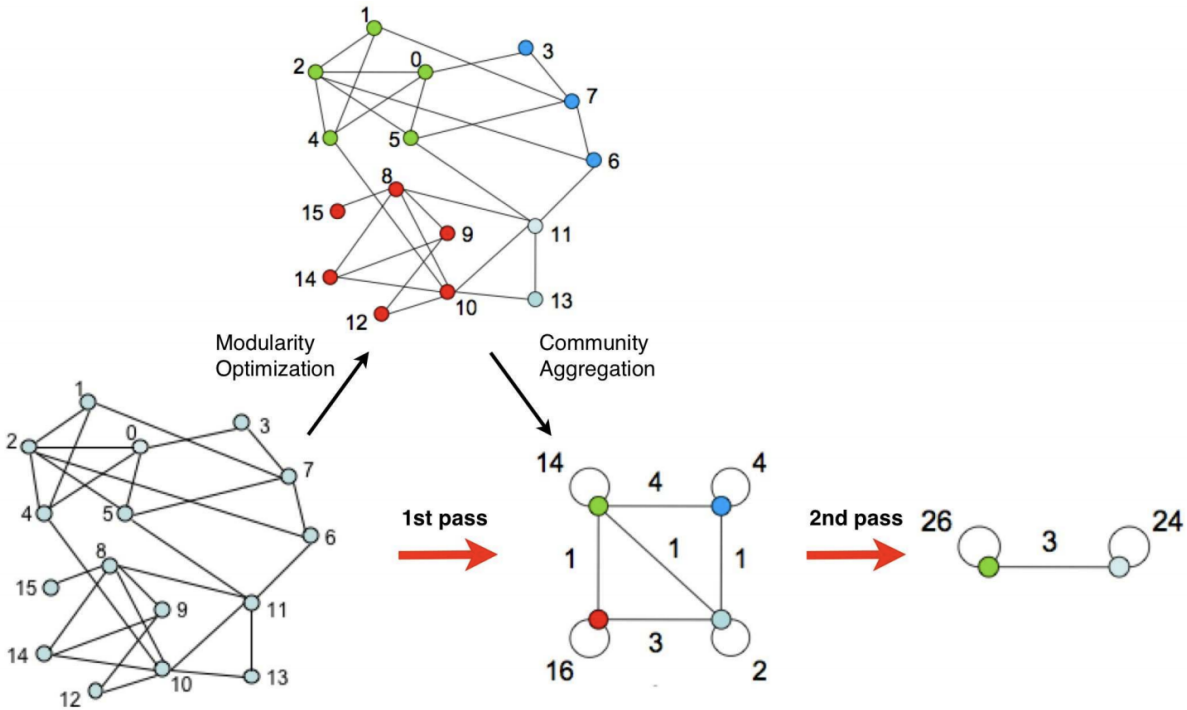


Figure: from Blondel *et al.* Journal of Statistical Mechanics. 2008.

Louvain is Fast Because Potential Merges are Easy to Compute

They show in Blondel *et al. 2008* that the change in modularity by moving a node into a community, C, can be computed in closed form as,

$$\Delta Q = [\frac{\sum_{in} + k_{i,in}}{2M} - (\frac{\sum_{tot} + k_i}{2M})^2] - [\frac{\sum_{in}}{2M} - (\frac{\sum_{tot}}{2M})^2 - (\frac{k_i}{2M})^2] \quad (11)$$

- $\sum_{in}$ the number of edges (or sum of the weights) of links inside of community C
- $\sum_{tot}$ is the number (or sum of the weights) of the edges connected to the nodes in C.
- k_i is the degree of node i
- $k_{i,in}$ is the sum of the edges (or edge weights) from nodes i to nodes in C

Practical Louvain Details

- Very fast, scalable, method. Works for most things if your graph is relatively sparse and or structured.

Practical Louvain Details

- Very fast, scalable, method. Works for most things if your graph is relatively sparse and or structured.
- Code: `https://pypi.org/project/louvain/`

Practical Louvain Details

- Very fast, scalable, method. Works for most things if your graph is relatively sparse and or structured.
- Code: `https://pypi.org/project/louvain/`
- A very good bet to get the job done quickly....

Practical Louvain Details

- Very fast, scalable, method. Works for most things if your graph is relatively sparse and or structured.
- Code: `https://pypi.org/project/louvain/`
- A very good bet to get the job done quickly....
- You do not need to specify the number of communities. The default resolution parameter is $\gamma = 1$.

Practical Louvain Details

- Very fast, scalable, method. Works for most things if your graph is relatively sparse and or structured.
- Code: `https://pypi.org/project/louvain/`
- A very good bet to get the job done quickly....
- You do not need to specify the number of communities. The default resolution parameter is $\gamma = 1$.
- A limitation is that is only allows for a hard partition of nodes.

All was Fine and Good Until they Realized a little Quirk

Louvain can produce communities that are internally disconnected. That is, the shortest path between a pair of nodes in the same community may require actually leaving the community.

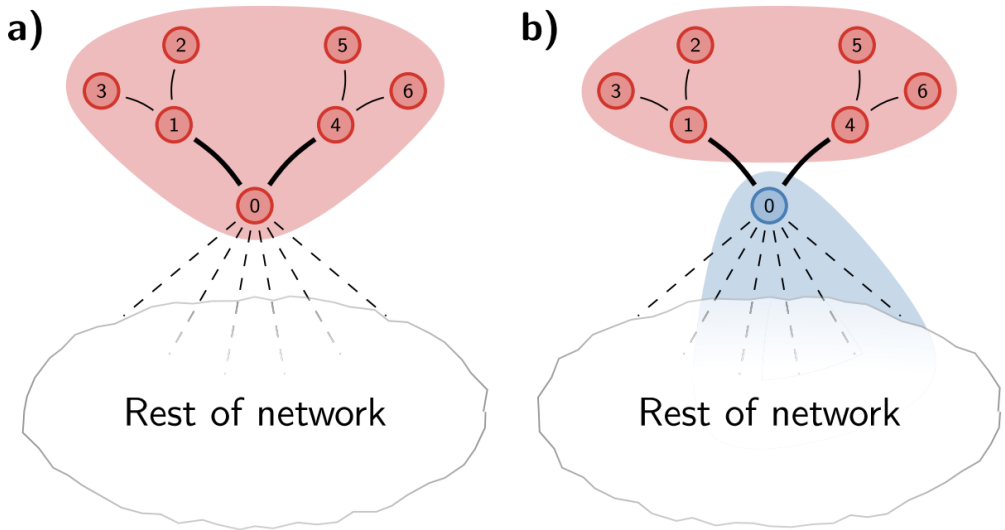


Figure: from Traag *et al.* Scientific Reports. 2018.

Overview of Leiden

Leiden makes a few modifications to guarantee well-connected communities.

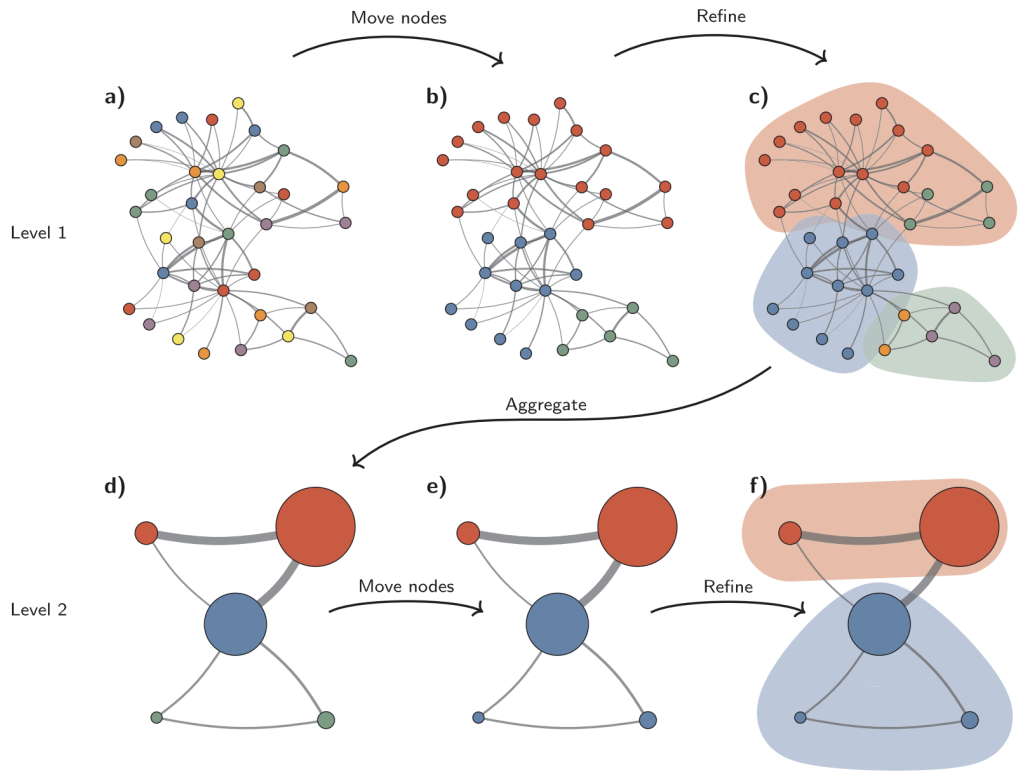


Figure: from Traag *et al.* Scientific Reports. 2018.

Very Similar to Louvain with a Small Extra Twist...

- In Louvain, a node (or group of nodes) might be merged into an existing group to produce some partition, $\mathcal{P}$.

Very Similar to Louvain with a Small Extra Twist...

- In Louvain, a node (or group of nodes) might be merged into an existing group to produce some partition, $\mathcal{P}$.
- Leiden takes $\mathcal{P}$ and creates $\mathcal{P}_{\text{refined}}$, which is sub-communities within each community of $\mathcal{P}$.

Very Similar to Louvain with a Small Extra Twist...

- In Louvain, a node (or group of nodes) might be merged into an existing group to produce some partition, $\mathcal{P}$.
- Leiden takes $\mathcal{P}$ and creates $\mathcal{P}_{\text{refined}}$, which is sub-communities within each community of $\mathcal{P}$.
- A singleton node within a community in $\mathcal{P}$ can only be merged into a sub-community of some community $c \in \mathcal{P}$ if it keeps c connected.

Punchline: Leiden will find higher quality communities (e.g. better connected) in less time than Louvain.
Code: `https://github.com/vtraag/leidenalg`

Indeed, there are less disconnected communities under leiden...

Depending on your application, this is important. At some point though if your graph gets large enough, what do we think?

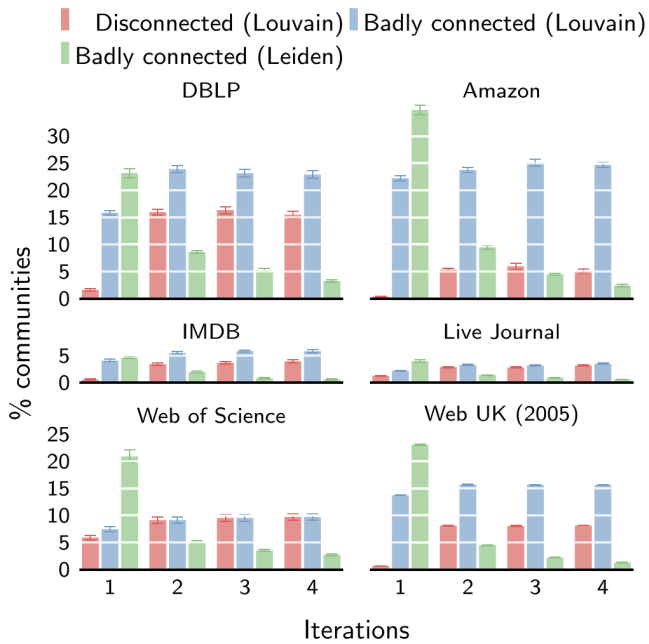


Figure: from Traag *et al.* Scientific Reports. 2018.

Badly connected $\rightarrow$ two nodes in the same community can be connected by walking outside of the community.

A Question

Can anyone think of a situation where disconnected communities might still be valid to consider?

Leiden Also Increases the Maximum Observed Modularity

	Nodes	Degree	Max. modularity	
			Louvain	Leiden
DBLP	317,080	6.6	0.8262	0.8387
Amazon	334,863	5.6	0.9301	0.9341
IMDB	374,511	80.2	0.7062	0.7069
Live Journal	3,997,962	17.4	0.7653	0.7739
Web of Science	9,811,130	21.2	0.7911	0.7951
Web UK	39,252,879	39.8	0.9796	0.9801

Figure: from Traag *et al.* Scientific Reports. 2018.

Modularity is more of a big picture score. I think whether you care about modularity or disconnectedness depends on your application.

Free Project Idea

A free idea for a course project: The authors define here a very particular notion of quality community. Are there other quality definitions. How does this observation change with different kinds of networks?

- You can find many of the standard networks people using for benchmarking and more on SNAP
  `https://snap.stanford.edu/data/`
- You can also use the biological networks in Choobdar *et al.*
  `https://www.nature.com/articles/s41592-019-0509-5`

Summary

- Diffusion and Graph Laplacian
- Building graphs from data and calculating a good 2d layout
- Start graph partitioning...