Comp683: Computational Biology

Lecture 4

January 23, 2024

Today

- Building and visualizing large-scale graphs
- Graph partitioning
  - Modularity-based approaches (Louvain, Leiden $\rightarrow$ PhenoGraph)
  - Application mining biological networks

Review from last time → 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 ($O(N^2 d)$).

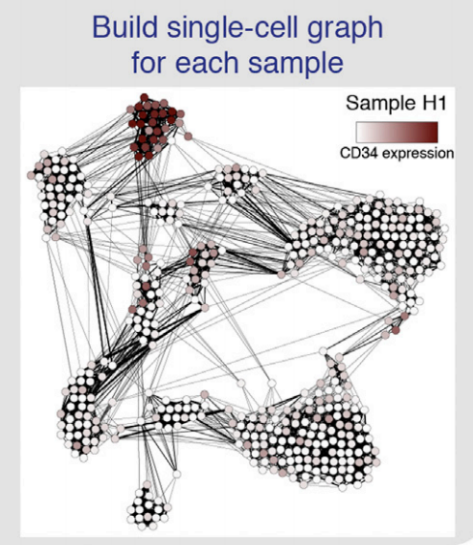


Figure: from Levine *et al.* Cell. 2015. Each cell is connected to its k-nearest neighbors

k-d tree

This is a partition of $\mathbb{R}^d$ hyper-rectangular cells based on the datapoints.

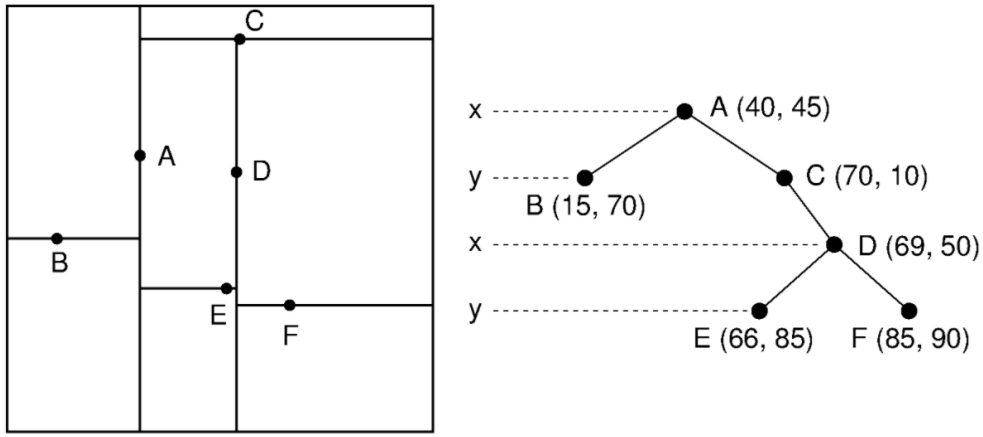


Figure: Alternate between splitting according to the ‘x‘ and ‘y‘ dimensions. First split according to ‘A‘. Based on partition by A, you can split horizontally according to C and B.

Random Projection Trees

Instead of doing axis parallel splits, split directions and split points are randomly selected.

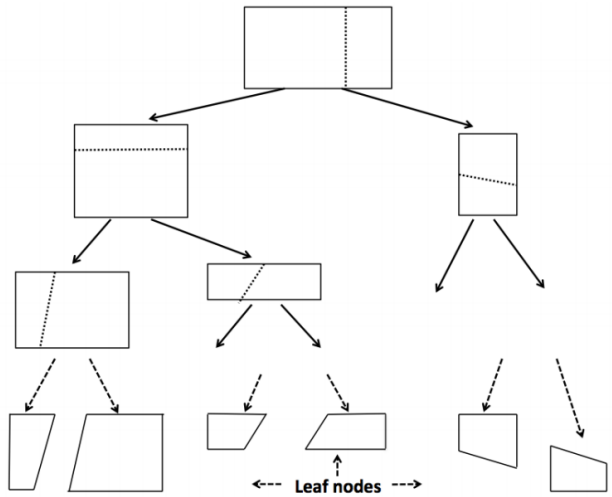
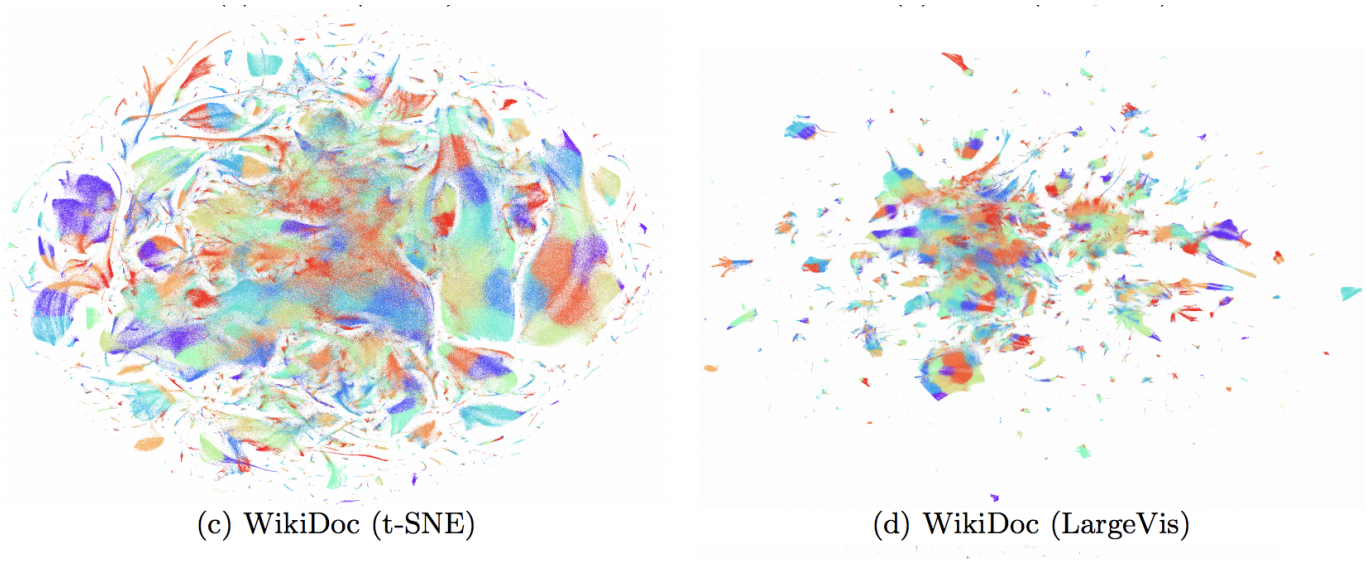


Figure: Random projection tree. (see `https://cseweb.ucsd.edu/~dasgupta/papers/exactnn-colt.pdf` for a discussion on randomized partition trees.)

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,

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

- 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.
The likelihood of the graph can be written as,

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

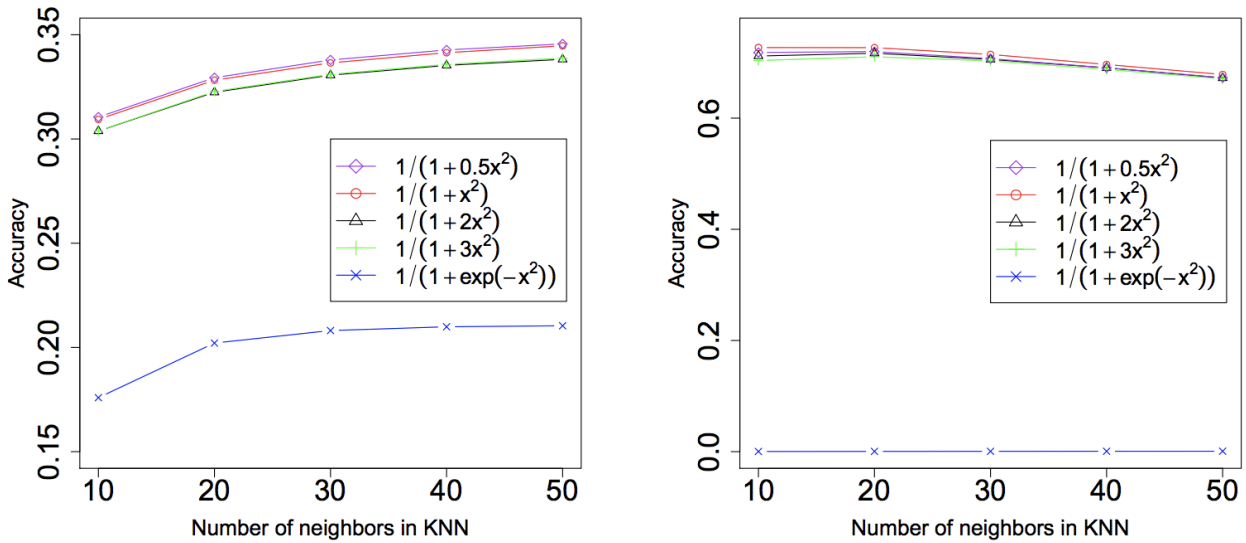
$\tilde{E}$ represents pairs of nodes with no edge.

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....

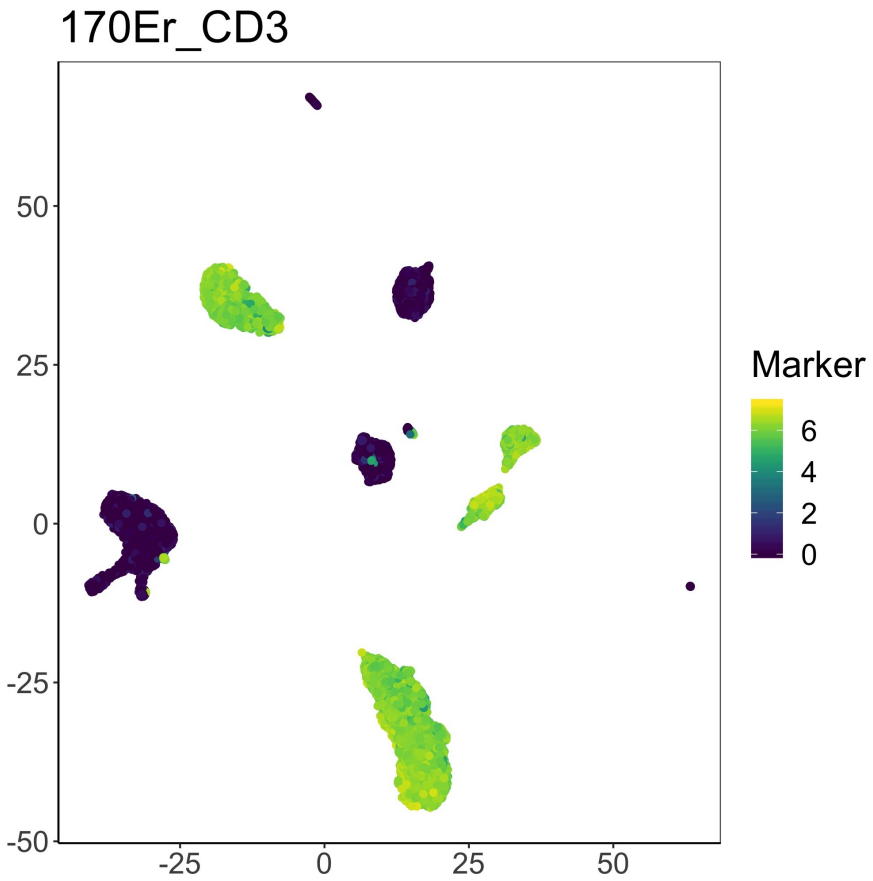
Table 2: Comparison of running time (hours) in graph visualization between the t-SNE and LargeVis.

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.

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!



Graph Partitioning Illustrated

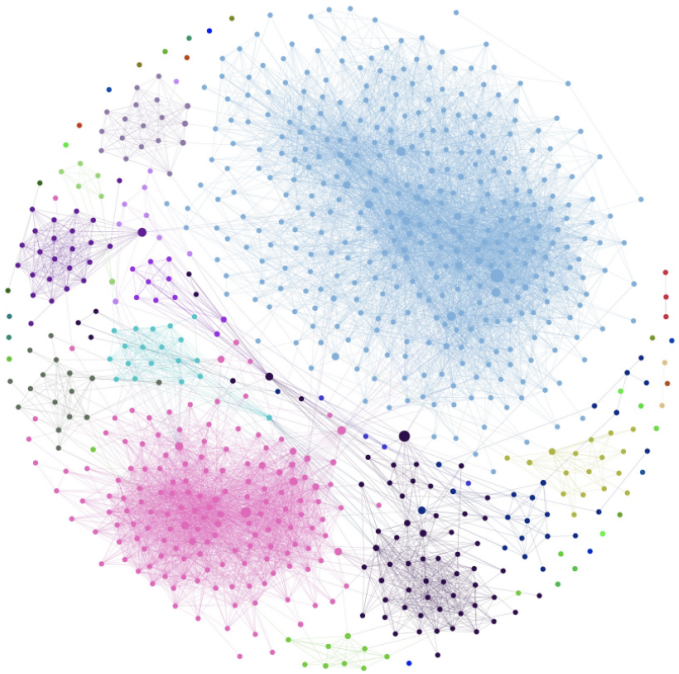


Figure: The high-level goal is to partition nodes into coherent node-subsets (communities), such that nodes within a group have on average more within group edges than between-group edges.

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 graph partitioning optimization problems seek a hard partition (each node assigned to a single cluster)
- 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 ←
  - 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 Laplacian)
- 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.

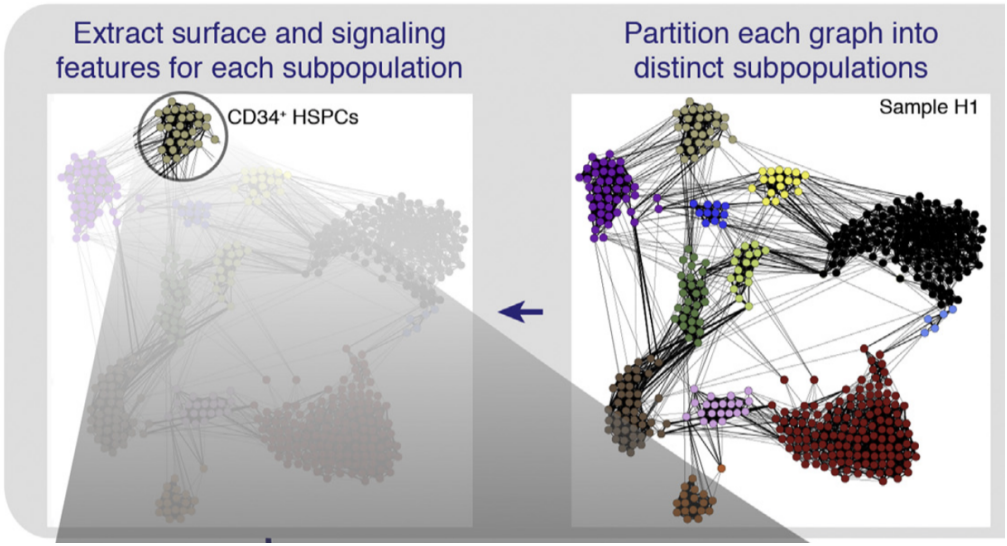


Figure: from Levine *et al.* Cell. 2015. This is the PhenoGraph algorithm.

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.
- **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.

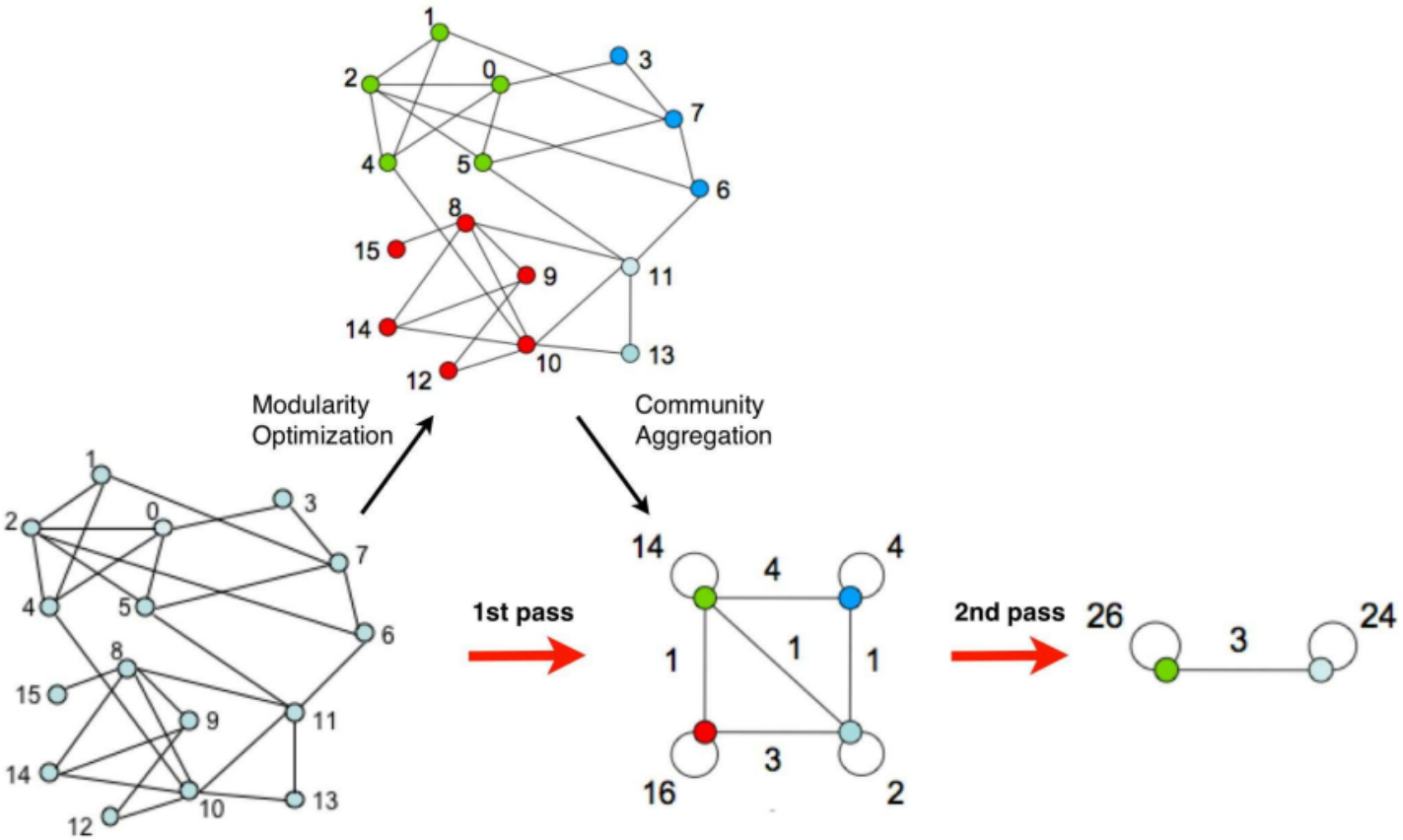
Modularity Defined

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

- 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.



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,

$$\Delta Q = [\frac{\sum_{in} + k_{i,in}}{2M} - (\frac{\sum_{tot} + k_i}{2M})^2] - [\frac{\sum_{in}}{2M} - (\frac{\sum_{tot}}{2M})^2 - (\frac{k_i}{2M})^2] \quad (3)$$

- $\sum_{in}$ 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.
- 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.

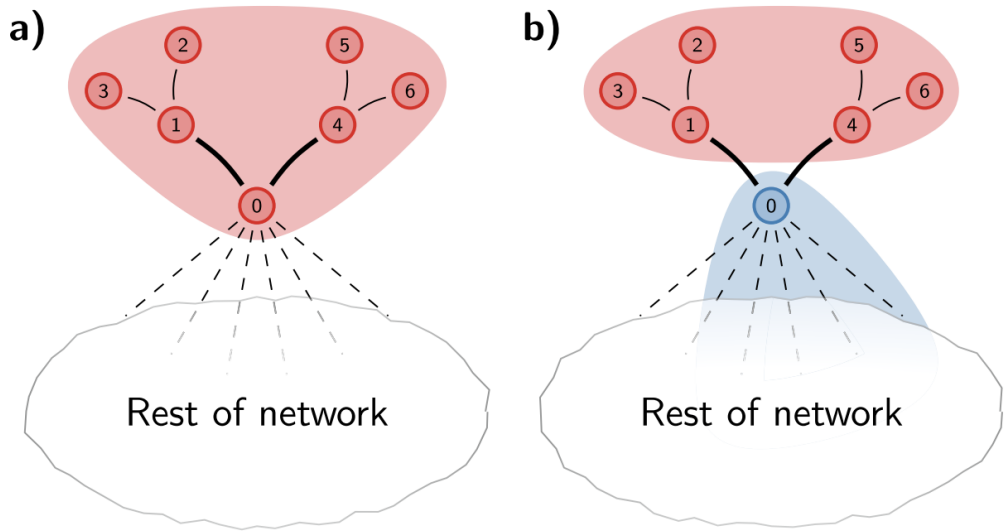


Figure: from Traag *et al.* Scientific Reports. 2018.

Overview of Leiden

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

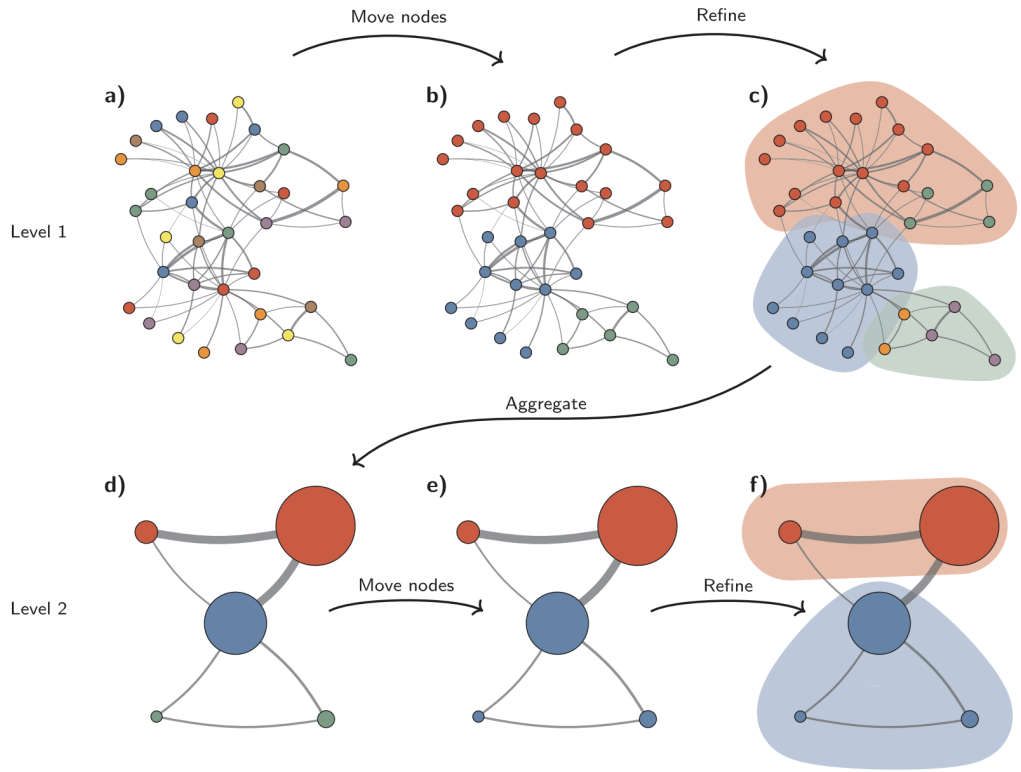


Figure: from Traag *et al.* Scientific Reports. 2018.

Methods of Clean-Up (1)

- **Smart Local Move:** Regular Louvain update of a merge, such that objective function cannot be increased
  - Before aggregating, consider the possibility of splitting each newly formed community
  - Gives the opportunity to split a subset of the newly formed community

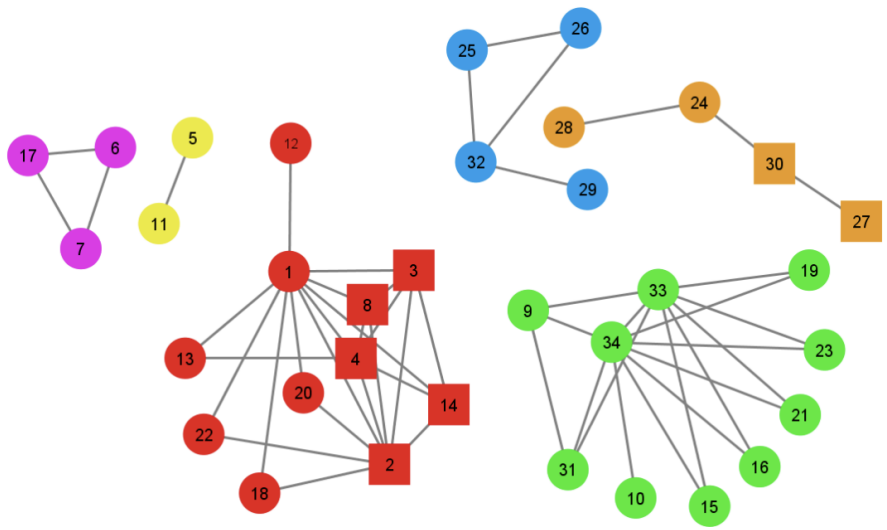


Figure: from `https://arxiv.org/pdf/1308.6604.pdf`

Methods of Clean-Up (2)

- **Random Moving:** For a particular node chosen at random, look at its neighbor in a different community and see if there is a gain in modularity if moving to that community
  - Intuition : The best community to join is the one where most of your neighbors are.

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

scanpy.tl.leiden

scanpy.tl.leiden(*adata, resolution=1, *, restrict_to=None, random_state=0, key_added='leiden', adjacency=None, directed=True, use_weights=True, n_iterations=-1, partition_type=None, neighbors_key=None, obsp=None, copy=False, **partition_kwargs*)

Cluster cells into subgroups [Traag18].

Cluster cells using the Leiden algorithm [Traag18], an improved version of the Louvain algorithm [Blondel08]. It has been proposed for single-cell analysis by [Levine15].

This requires having ran `neighbors()` or `bbknn()` first.

Example in practive $\rightarrow$ cell-types emerging

```
[31]: sc.tl.leiden(adata)

      running Leiden clustering
          finished: found 8 clusters and added
          'leiden', the cluster labels (adata.obs, categorical) (0:00:00)
```

Plot the clusters, which agree quite well with the result of Seurat.

```
[32]: sc.pl.umap(adata, color=['leiden', 'CST3', 'NKG7'])
```

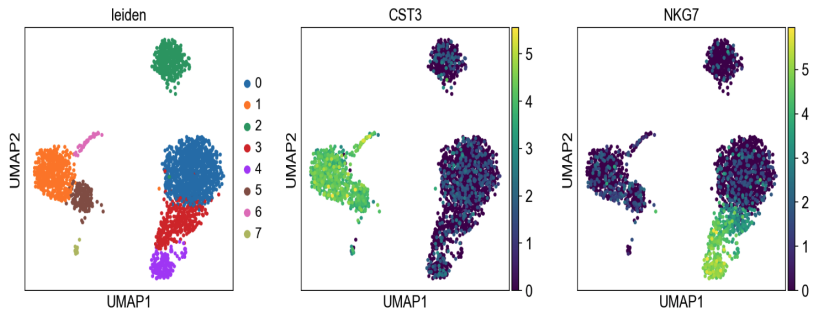


Figure: dataset of PBMCs from `https://scanpy-tutorials.readthedocs.io/en/latest/pbmc3k.html`

A Word of Caution for Single-Cell People

It is incorrect to say 'I clustered by single-cell data with scanpy' or to say 'I clustered by single-cell data with Seurat.' Look at which graph partitioning approach they were using!

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?

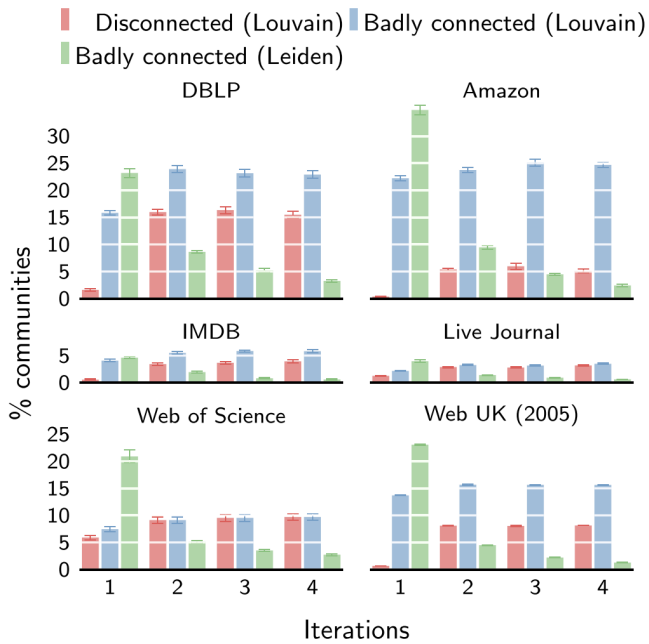


Figure: from Traag *et al.* Scientific Reports. 2018.

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`

Partitions of Biological Graphs

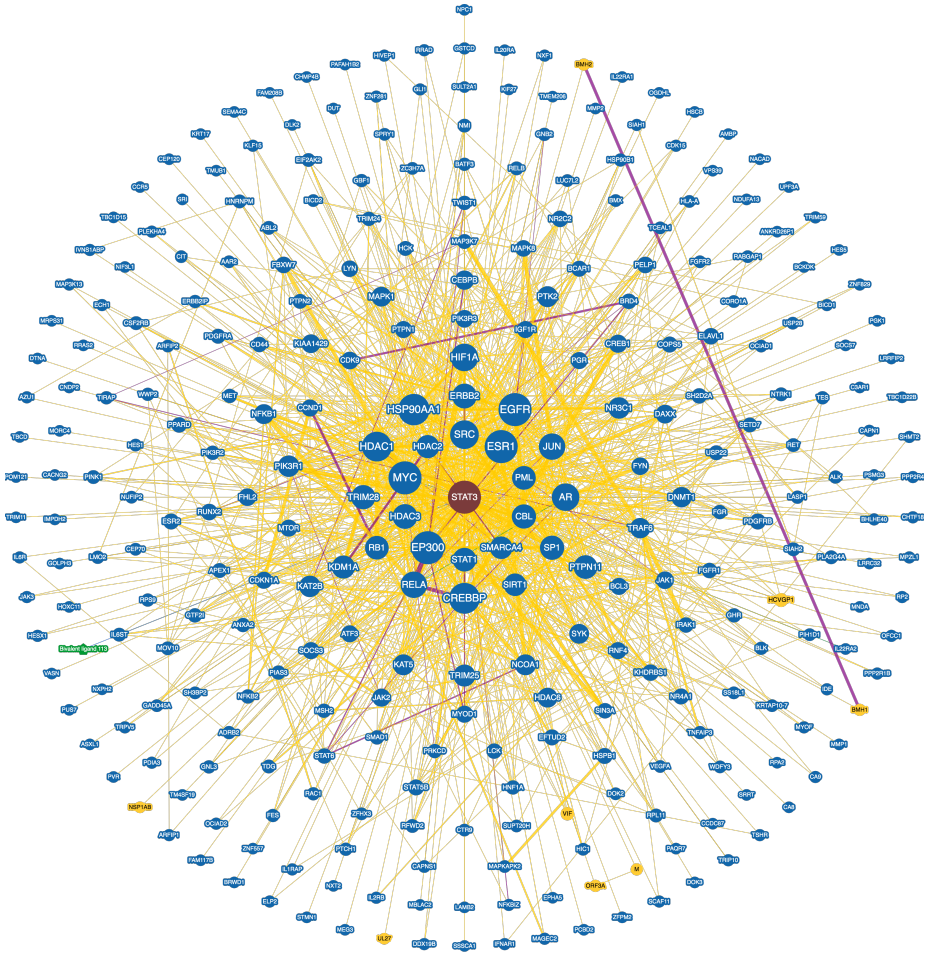


Figure: We are interested in the interactions with STAT3. These can be measured in different ways (gene expression, protein expression, physical interaction)

How Does Modularity-Based Optimization do in the Biological Network Benchmarking Challenge?

Here there is a really nice ‘ground truth‘ understanding, which is gene and protein interactions linked to particular diseases.

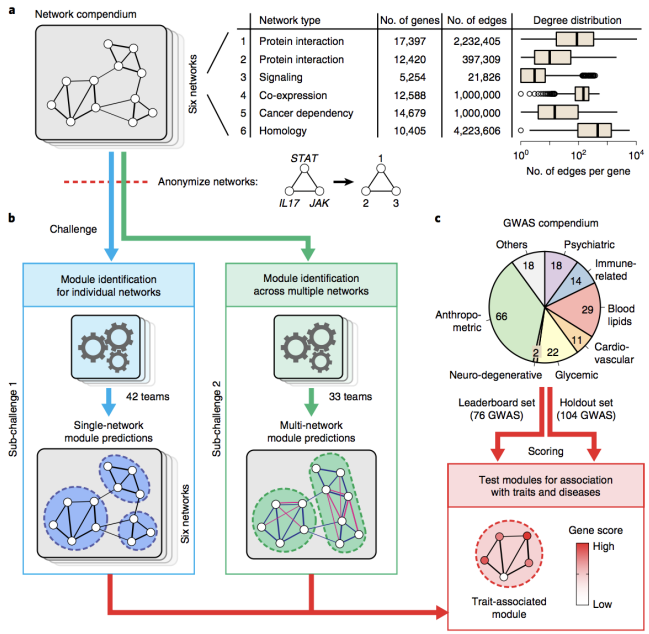
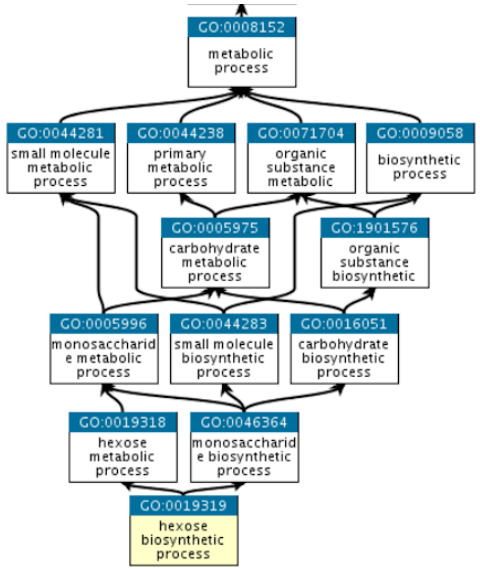


Figure: from Choobar *et al.* Nature Methods 2018.

Good Places to Look for Ground Truth to Validate Your Methods

- Gene Ontology → genes that are involved in the same pathway.
- MSigDB → genes related to certain categories (cancer, immune system, regulatory targets, etc.), text mined
- KEGG → metabolism
- Panther → genes and proteins



It seems that modularity is not only numerically useful

We use modularity to effectively score a candidate partition. This experiment shows that modularity is indeed correlated with the identification of biologically meaningful modules.

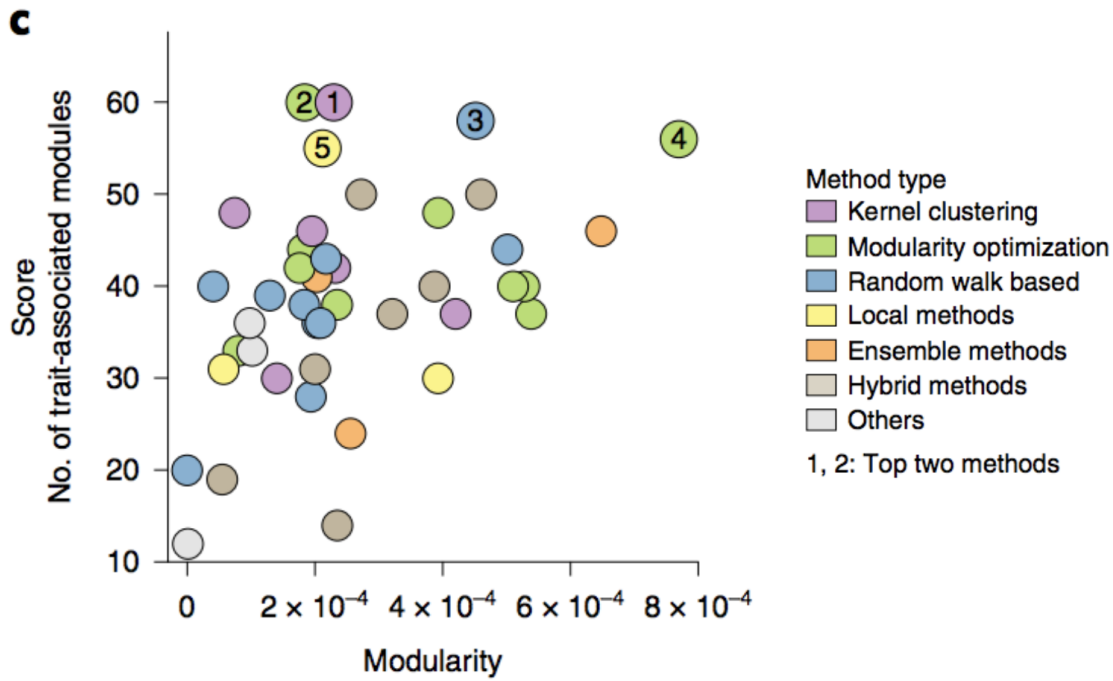


Figure: from Choobar *et al.* Nature Methods 2018.

Ranking Methods Based on Biological Relevance.

'M1' is a variant of what we have seen with Louvain. Interested readers can read more here, `https://arxiv.org/abs/physics/0703218`

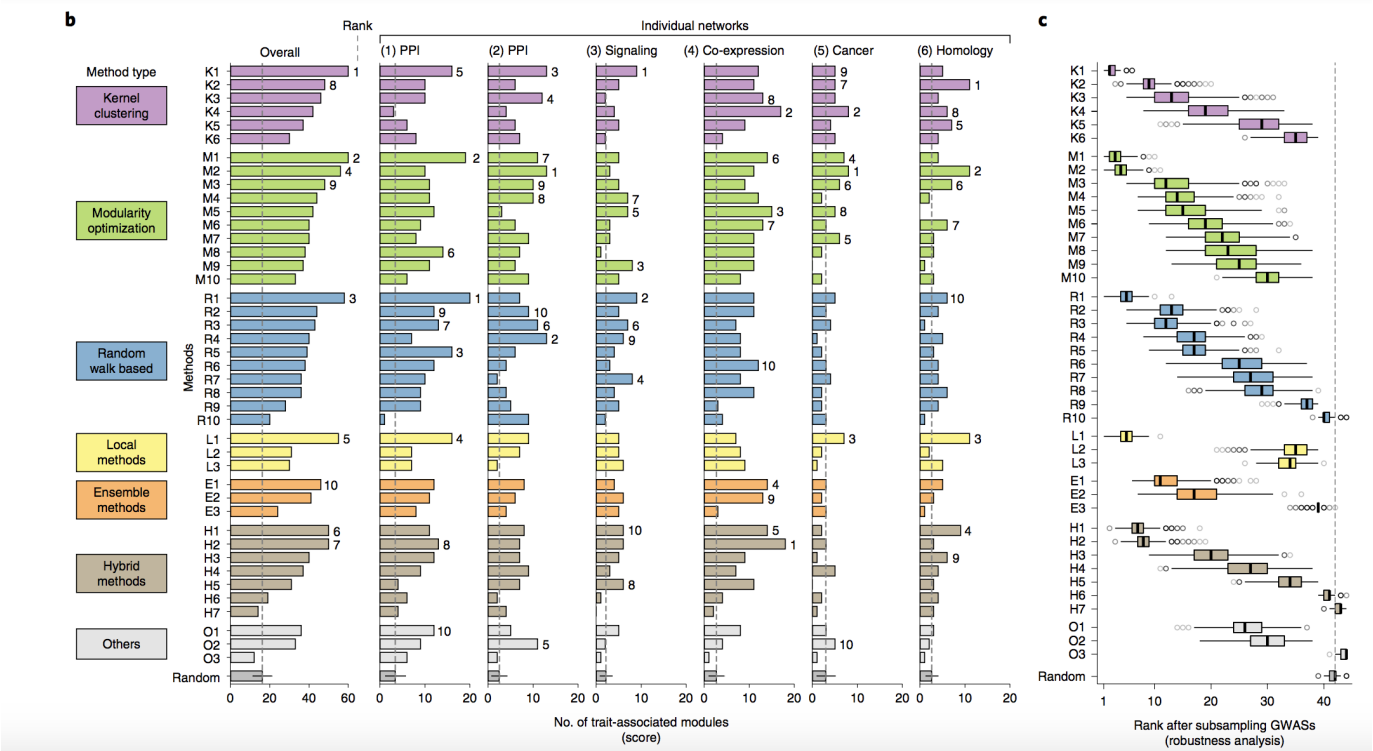


Figure: from Choobar *et al.* Nature Methods 2018.

Examples of Biologically-Relevant Modules

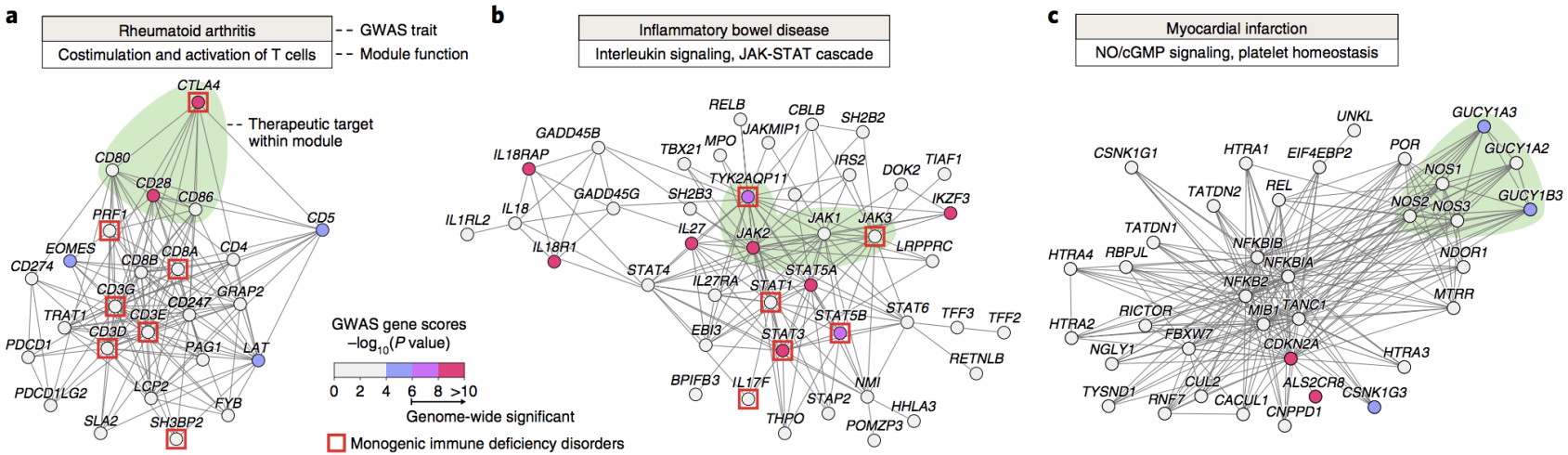


Figure: from Choobar *et al.* Nature Methods 2018.