Comp790-166: Computational Biology

Lecture 24

April 11, 2022

Good Morning Question

- Can someone summarize REGAL and its main steps?
- How are landmark nodes used and how may someone choose landmarks?

Today

- Refining graph alignments [REFINA]
- Graph Summarization

Announcements

- Homework is online.
- Project Presentations April 25 and April 27. Please visit the signup sheet. https://docs.google.com/spreadsheets/d/1_z1NBffJF8do8JrasTQl-8pS-ATR2ScI7SutRDPIf80/edit?usp=sharing
- Final Project LaTeX template https://github.com/natalies-teaching/Comp790-166-CompBio-Spring2022/tree/main/Project_Final_Writeup
- I encourage in person attendance for the project presentation days.

Overview of Regal Method

Regal → Representation Based Graph Alignment.

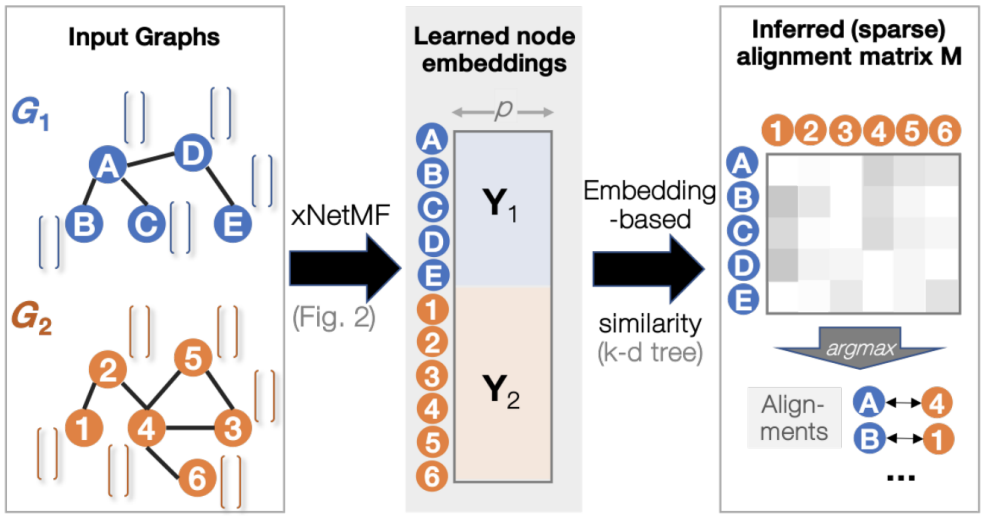


Figure: from from Heimann *et al.* CIKM 2018. Similar to Node2vec, the authors want to find a representation for each node

Graph Alignment is a Hard Problem

- With REGAL, there was some fancy linear algebra required
- **Another reasonable assumption:** Nodes within the a common neighborhood in one graph should be mapped to nodes that are close (e.g. within or in a close neighborhood) in the other graph.
- This feels a bit like Leiden- where the partition will be ‘corrected‘ to make sure that communities contain graphs with a connected path between most pairs within the community.

Illustration of the Idea

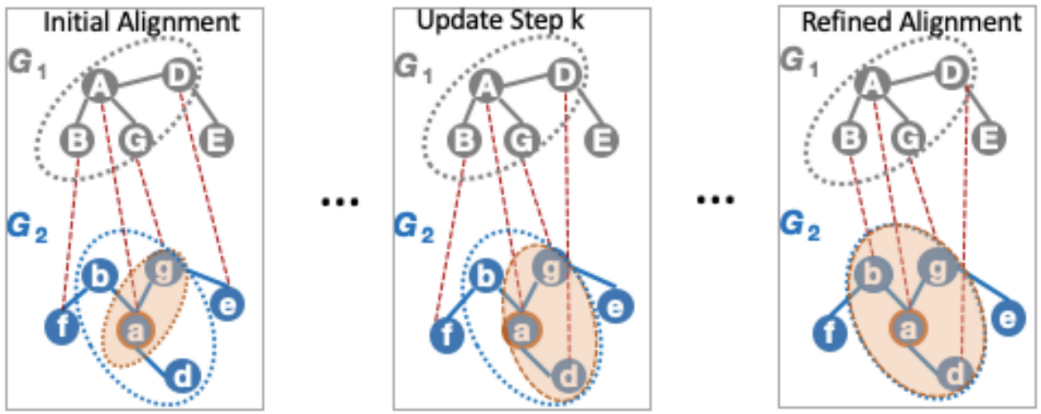


Figure: from Heimann *et al.* 2021. The alignment is iteratively connected so nodes within a neighborhood of one graph are mapped to a similar neighborhood in the other graph.

A Similar Formulation of the Graph Alignment Problem from REGAL

- A node alignment is a function, $\pi : \mathcal{V}_1 \rightarrow \mathcal{V}_2$ that maps the nodes of G_1 to the nodes of G_2.
- This can also be presented by a $|\mathcal{V}_1| \times |\mathcal{V}_2|$ matrix, $\mathbf{M}$, where M_{ij} is the similarity between node i in G_1 and node j in G_2.

Then similar to what we saw the REGAL, specify the greedy alignment as,

$$\pi(i) = \arg\max_j \mathbf{M}_{ij}$$

Matched Neighborhood Consistency (MNC)

- **Neighbors of node i in G_1** $\rightarrow \mathcal{N}_{G_1}(i) = \{j \in \mathcal{V}_1 : (i,j) \in \mathcal{E}_1\}$
- **Mapped Neighborhood:** The set of nodes onto which π maps i's neighbors.
  - $\tilde{\mathcal{N}}^{\pi}_{G_2}(i) = \{j \in \mathcal{V}_2 : \exists k \in \mathcal{N}_{G_1}(i) \text{ s.t. } \pi(k) = j\}$

Then, given a node i in G_1 and a node j in G_2, define the neighborhood consistency as,

$$\text{MNC}(i,j) = \frac{\left|\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cap \mathcal{N}_{G_2}(j)\right|}{\left|\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cup \mathcal{N}_{G_2}(j)\right|}$$

Welcome RefiNA

- Let $\mathbf{M}_0$ be the initial alignment returned by any graph alignment method.
- The goal is to refine the initial solution, $\mathbf{M}_0$ into a more refined solution, $\mathbf{M}$ that better preserves this neighborhood overlap.
- This is in contrast to other approaches that ‘seed‘ the alignments and only uses the structure of the graph to try to make a better alignment.

Rewriting the MNC

Recall the MNC (matched neighborhood consistency) between the graphs. This can we rewritten in matrix form, such that $\text{MNC}(i,j) = \mathbf{S}_{ij}^{\text{MNC}}$ as,

$$\mathbf{S}^{MNC} = \mathbf{A}_1\mathbf{M}\mathbf{A}_2 \oslash (\mathbf{A}_1\mathbf{M}\mathbf{1}^{n_2} \otimes \mathbf{1}^{n_2} + \mathbf{1}^{n_1} \otimes \mathbf{A}_2\mathbf{1}^{n_2} - \mathbf{A}_1\mathbf{M}\mathbf{A}_2)$$

Here,

- $\mathbf{M}$ is a binary alignment matrix
- $\oslash$ is element-wise division
- $\otimes$ is outerproduct

* aka writing Jaccard similarity in matrix format

Computing a Refined Alignment

Compute refined alignments, $\mathbf{M}'$ by multiplicative updating each node's alignment score (in $\mathbf{M}$) with its matched neighborhood consistency as,

$$\mathbf{M}' = \mathbf{M} \circ \mathbf{S}^{\mathrm{MNC}}$$

Here, $\circ$ is Hadamard product.

- The proposed refinement score should iteratively increase alignment scores for nodes that have high MNC.

Modifying Updates with Some Important Intuition

- **Higher degree nodes are easier to align**. The part of MNC we care about is counting nodes' matched neighbors. This simplifies the update rule to,

$$\mathbf{M}' = \mathbf{M} \circ \mathbf{A}_1\mathbf{M}\mathbf{A}_2$$

Now the MNC update is simply,

$$\mathbf{A}_1\mathbf{M}\mathbf{A}_2$$

.

Intuition, Continued

- **Do Not Rely Too Much on Initial $\mathbf{M}_0$.** Add a small ϵ at each iteration to correct for false negatives.
- **Normalization:** To encourage performance and to keep **M** from exploding, row normalize **M**, followed by column normalization at every iteration.

Summary

Algorithm 1 RefiNA $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{M}_0, K, \epsilon)$

Input: adjacency matrices $\mathbf{A}_1, \mathbf{A}_2$, initial alignment matrix $\mathbf{M}_0$, number of iterations K, token match score ϵ
for $k = 1 \rightarrow K$ **do** ▷ Refinement iterations
    $\mathbf{M}_k = \mathbf{M}_{k-1} \circ \mathbf{A}_1\mathbf{M}_{k-1}\mathbf{A}_2$ ▷ MNC update
    $\mathbf{M}_k = \mathbf{M}_k + \epsilon$ ▷ Add token match scores
    $\mathbf{M}_k = \text{Normalize}(\mathbf{M}_k)$ ▷ By row then column
end for
return $\mathbf{M}_K$

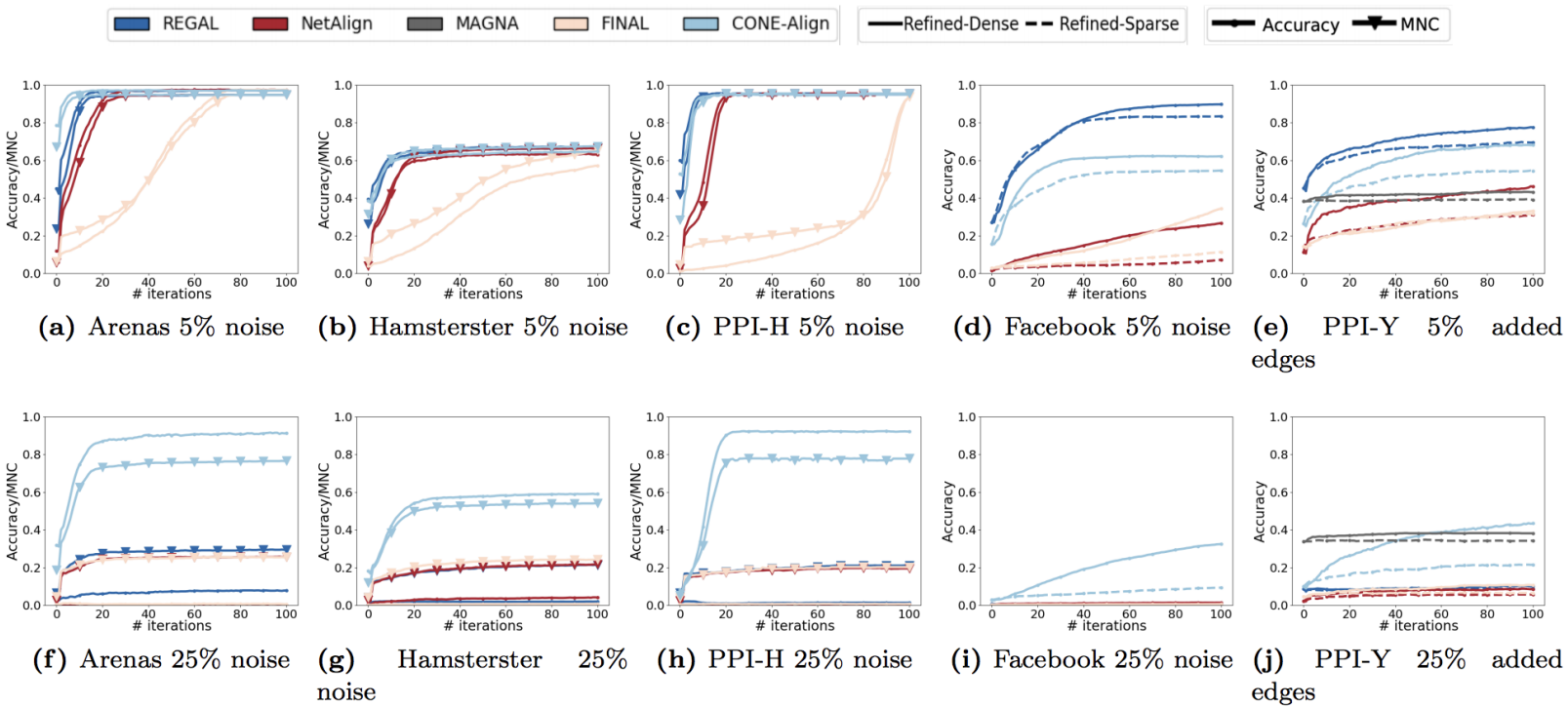
Figure: from Algorithm 1

Question

Which order neighborhood should be considered? It seems that the authors only consider immediate neighbors. Do you think there is benefit to considering higher order neighborhoods?

Convergence

Here's what happens with MNC and accuracy with increasing multiplicative interations.



(a) Arenas 5% noise **(b)** Hamsterster 5% noise **(c)** PPI-H 5% noise **(d)** Facebook 5% noise **(e)** PPI-Y 5% added edges

(f) Arenas 25% noise **(g)** Hamsterster 25% noise **(h)** PPI-H 25% noise **(i)** Facebook 25% noise **(j)** PPI-Y 25% added edges

Figure: from Fig. 7

Making RefiNA Sparse

- To scale RefiNA to large graphs, the authors focus on updating a small number of alignment scores for each node.
- Intuitively, forgo updating scores of likely unaligned node pairs.
- The solution is to only update some entries of $\mathbf{M}$.

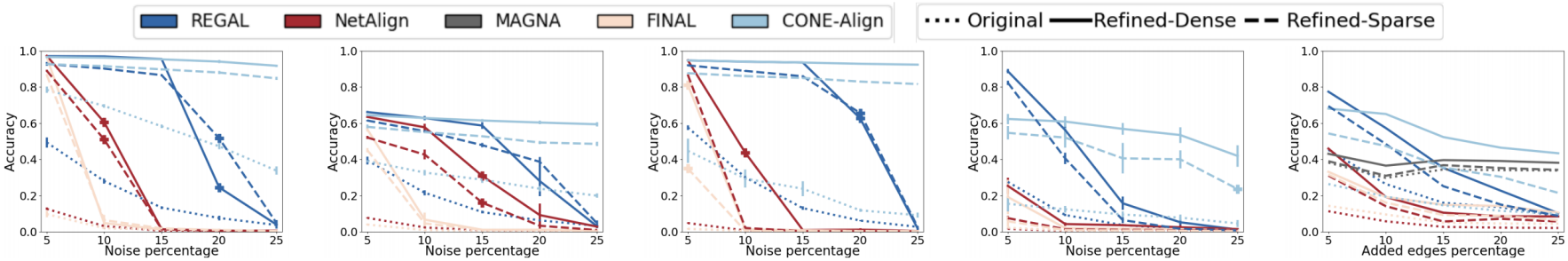
Specifically the updates to $\mathbf{M}$ are modified as,

$$\mathbf{M}_{k|\mathbf{U}_k} = \mathbf{M}_{k-1|\mathbf{U}_k} \circ \mathbf{U}_k$$

- $\mathbf{U}_k = \text{top} -\alpha\left(\mathbf{A}_1\mathbf{M}\mathbf{A}_2\right)$ is only the top α entries per row.
- $\mathbf{M}_{k|\mathbf{U}_k}$ is only the non-zero elements in $\mathbf{U}_k$

Noise Experiments

Create a permuted copy of $\mathbf{A}$, $\tilde{\mathbf{A}} = \mathbf{PAA}^T$. Then remove edges with some probability, p_s.



(a) Arenas (b) Hamsterster (c) PPI-H (d) Facebook (e) PPI-Y

Figure: from Fig. 2. Refinement with the sparse or dense formulation helps networks with different structures.

Run-time

Notice REGAL is faring pretty well (even with the dense formulation...)

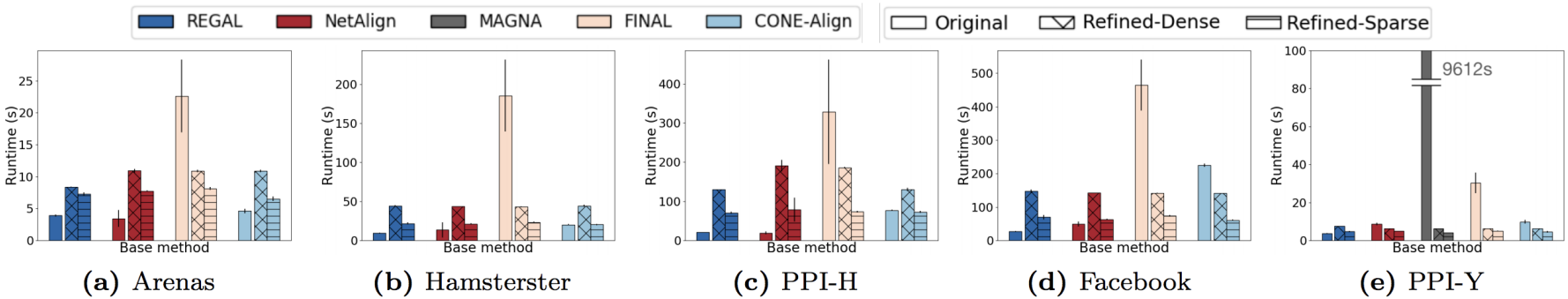


Figure: from Fig. 4.

Performance Based on a Binned Degree Distribution

Nodes were binned by degree into $\{low, medium, high\}$. The MNC of these nodes were further visualized based on accuracy.

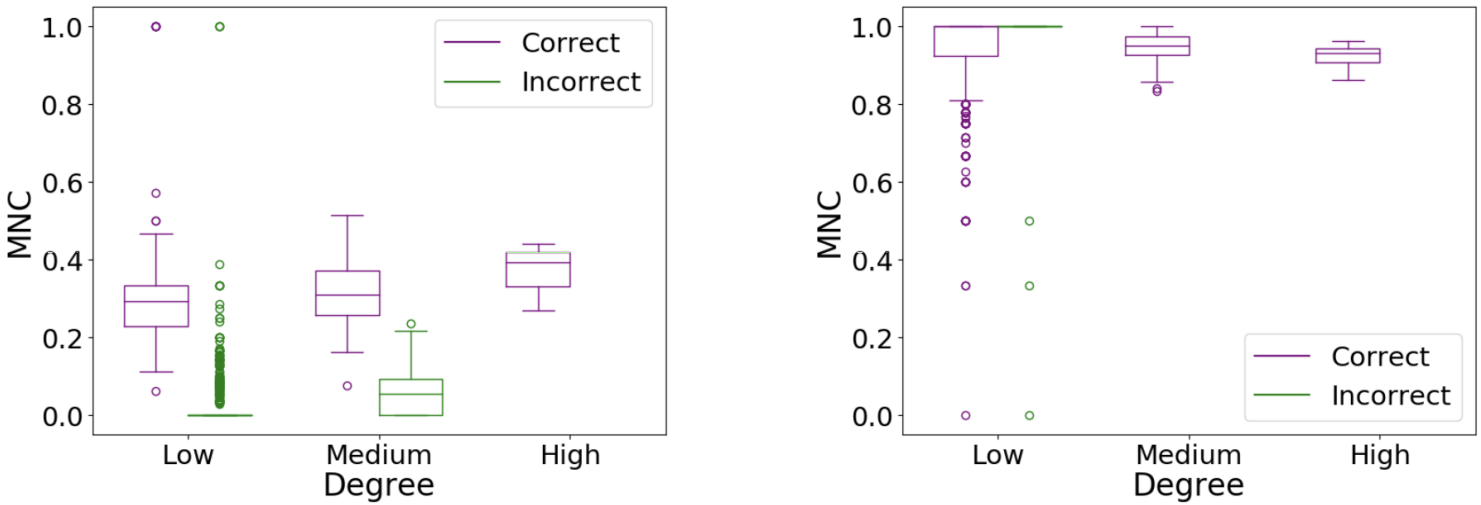


Figure: from Fig. 5

RefiNA Conclusion

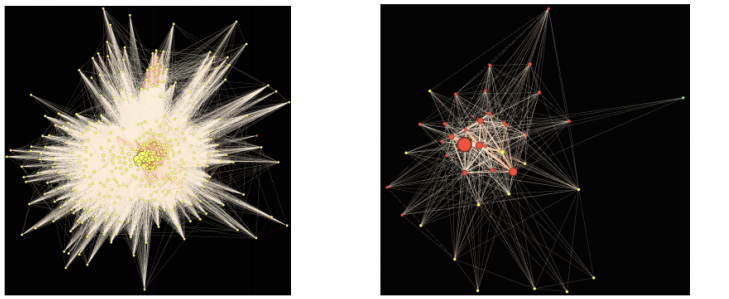
- RefiNA is a posthoc method to clean up a network alignment.
- You can start with an alignment generated with any algorithm (though REGAL looks good!)
- You have the ability to make the updates more sparse by zeroing out entries in $\mathbf{M}$

Graph Summarization

- The goal is to find a representation for the graph that is independent of the number of nodes
- Very simple things (yet uninformative) : number of nodes, number of edges, degree distribution
- More complex and interesting
  - Counts of particular subgraph types (stars, triangles, k-cliques, etc)

Example - Condense

Look for predefined structural patterns and reduce those patterns to a single node.



(a) Prior work [29] (b) Our method: CONDENSE-STEP

Figure: from `https://gemslab.github.io/papers/liu-2018-reducing.pdf`. 2018

One Way to Get Structures Quickly, k-cores

- A k-core is a subgraph in which all nodes have degree k or more.

Algorithm 2 KCBC: k-core-based Graph Clustering

Input: graph G

While the graph is nonempty

Step 1: Compute core numbers (max k for which the node is present in the decomposition) for all nodes in the graph.

Step 2: Choose the maximum k (k_{max}) for which the decomposition is non-empty, and identify nodes which are present in the decomposition as the "decomposition set." Terminate when $k_{max} = 1$.

Step 3: For the induced subgraph from the decomposition set, identify each connected component as a structure.

Step 4: Remove all edges in the graph between nodes in the decomposition set—they have been identified as structures already.

return set of all identified structures

Figure: from `https://arxiv.org/pdf/1511.06820.pdf`

One Application: Network Compression

The graph represents clusters of orthologous genes from multiple contexts, obtained from the String Database. The algorithm finds certain cliques and summarizes with a sparser representation (like a star).

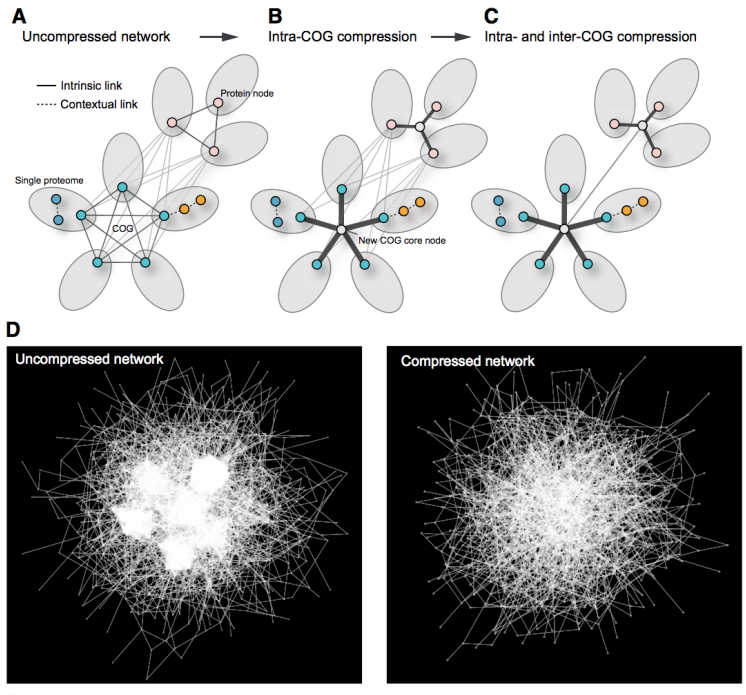


Figure: from Lisewski *et al.* Cell. 2014