Comp790-166: Computational Biology

Lecture 17

April 6, 2021

Announcements

- Homework 2 to be assigned on Thursday and due April 22.
- Comments on projects? How are they going?

Today

- A few remaining remarks on REGAL
- Refining Graph Alignments
- Graph Summarization

REGAL Reminder

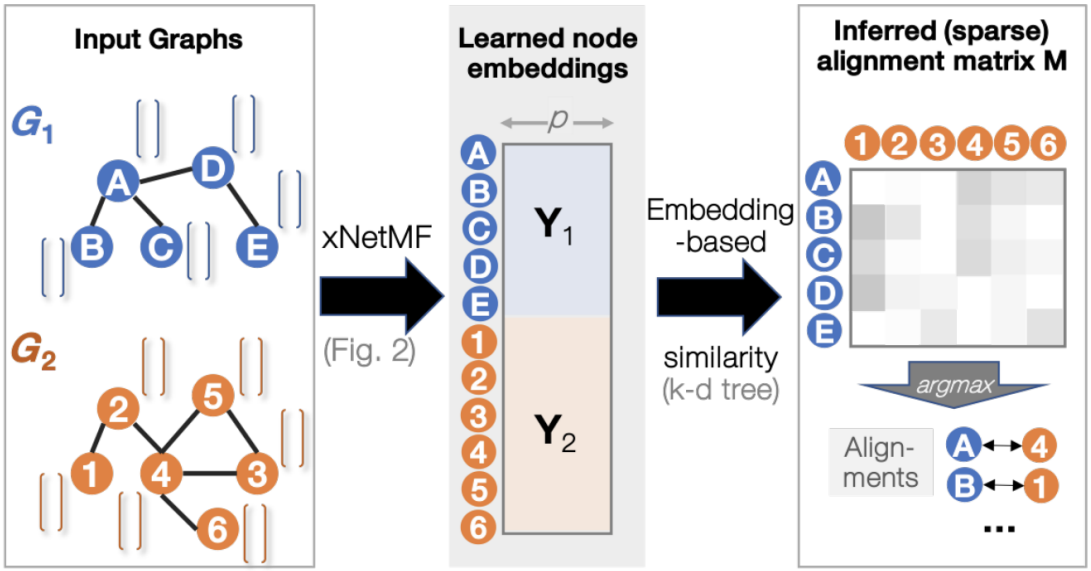


Figure: from Heimann *et al.* CIKM. 2018

The Overall Approach for the Embedding Algorithm

- There are two phases $\rightarrow$ embedding and matching nodes between graphs based on the learned embeddings.

Algorithm 2 xNetMF $(G_1, G_2, p, K, \gamma_s, \gamma_a)$

========== **STEP 1. Node Identity Extraction** ==========
for node u in $\mathcal{V}_1 \cup \mathcal{V}_2$ **do**
  for hop k up to K **do** ▷ counts of node degrees of k-hop neighbors of u
    $\mathbf{d}_u^k$ = CountDegreeDistributions($\mathcal{R}_u^k$) ▷ $1 \leq K \leq$ graph diameter
  end for
  $\mathbf{d}_u = \sum_{k=1}^{K} \delta^{k-1} \mathbf{d}_u^k$ ▷ discount factor $\delta \in (0, 1]$
end for
========= **STEP 2. Efficient Similarity-based Representation** =========
========== **STEP 2a. Reduced $n \times p$ Similarity Computation** ==========
$\mathcal{L}$ = ChooseLandmarks(G_1, G_2,p) ▷ choose p nodes from G_1, G_2
for node u in $\mathcal{V}$ **do**
  for node v in $\mathcal{L}$ **do**
    $c_{uv} = e^{-\gamma_s \cdot ||\mathbf{d}_u - \mathbf{d}_v||_2^2 - \gamma_a \cdot \text{dist}(\mathbf{f}_u, \mathbf{f}_v)}$
  end for
end for ▷ Used in low-rank approx. of similarity graph (not constructed)
========== **STEP 2b. From Similarity to Representation** ==========
$\mathbf{W} = \mathbf{C}[\mathcal{L}, \mathcal{L}]$ ▷ Rows of C corresponding to landmark nodes
$[\mathbf{U}, \Sigma, \mathbf{V}] = \text{SVD}(\mathbf{W}^{\dagger})$
$\tilde{\mathbf{Y}} = \mathbf{C}\mathbf{U}\Sigma^{-\frac{1}{2}}$ ▷ **Embedding**: *implicit* factorization of similarity graph
$\tilde{\mathbf{Y}} = Normalize(\tilde{\mathbf{Y}})$ ▷ **Postprocessing**: make embeddings have magnitude 1
$\tilde{\mathbf{Y}}_1, \tilde{\mathbf{Y}}_2 = \text{Split}(\tilde{\mathbf{Y}})$ ▷ **Separate** representations for nodes in G_1, G_2
return $\tilde{\mathbf{Y}}_1, \tilde{\mathbf{Y}}_2$

The Last Step: Distances Between Nodes in Different Graphs via Embeddings

Letting $\tilde{Y}_1$ and $\tilde{Y}_2$ be the embeddings for graphs 1 and 2 respectively, then pairwise node similarities between the graphs can be computed as,

$$\text{sim}_{emb}\left(\tilde{\mathbf{Y}}_1[u], \tilde{\mathbf{Y}}_2[v]\right) = e^{-\left\|\tilde{\mathbf{Y}}_1[u] - \tilde{\mathbf{Y}}_2[v]\right\|_2^2}$$

- Match nodes according to these similarity scores.
- **Soft Scoring Approach:** From kd-tree, find $\alpha << N$ top matches of nodes from the other graph.

Experimental Evaluation

Given an adjacency matrix, $\mathbf{A}$, generate a random permutation matrix, $\mathbf{P}$ and generate a new network as,

$$\mathbf{A}' = \mathbf{PAP}^T$$

Further, remove edges from $\mathbf{A}'$ with probability p_s, and permute attributes with probability, p_a.

Results from Adding Noise

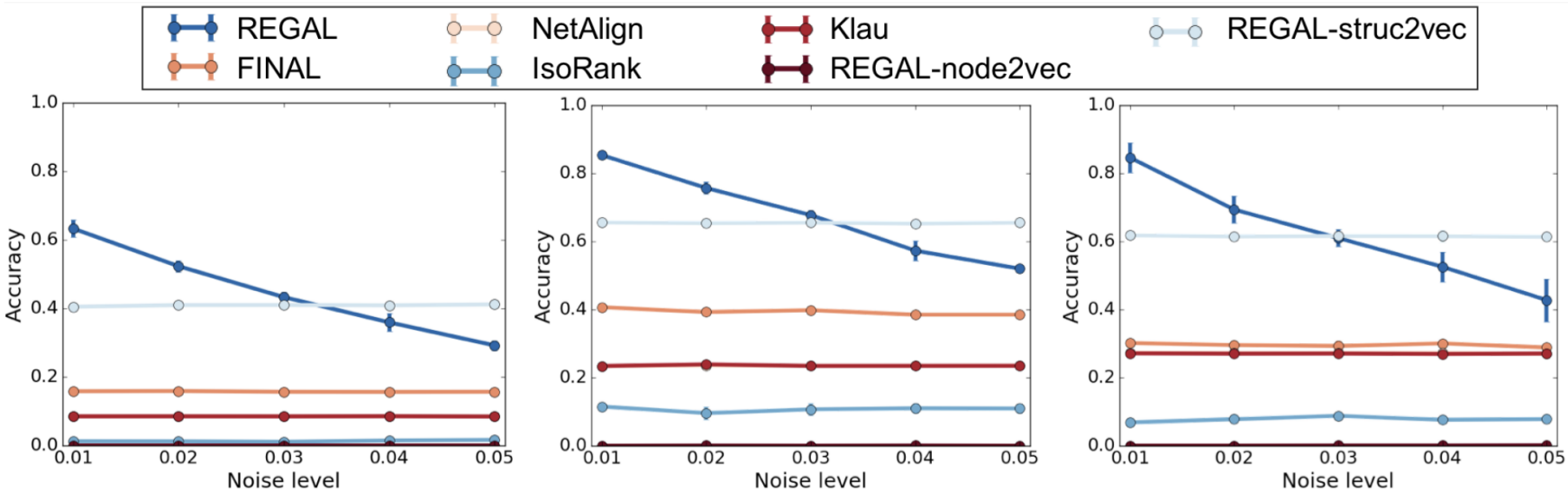


Figure: from Fig. 4, Replacing the embedding step with struct2vec produces better results when there is more noise.

REGAL Is More Ideal wrt Run-time

Dataset	Arxiv	PPI	Arenas
FINAL	4182 (180)	62.88 (32.20)	3.82 (1.41)
NetAlign	149.62 (282.03)	22.44 (0.61)	**1.89 (0.07)**
IsoRank	**17.04 (6.22)**	**6.14 (1.33)**	**0.73 (0.05)**
Klau	1291.00 (373)	476.54 (8.98)	43.04 (0.80)
REGAL-node2vec	709.04 (20.98)	139.56 (1.54)	15.05 (0.23)
REGAL-struc2vec	1975.37 (223.22)	441.35 (13.21)	74.07 (0.95)
REGAL	**86.80 (11.23)**	**18.27 (2.12)**	2.32 (0.31)

Figure: from Table 4. The methods that perform better in ‘noise‘ experiments also have higher run-time.

Conclusion

- REGAL computes embeddings via landmark points.
- The magic is in approximating the pairwise node similarity matrix through landmark points
- Performance is generally better than baselines for most ‘noise‘ levels.

Questions for You

- How do you think landmarks should be chosen? At random, or something more principled?
- What about extending this to more than two graphs?

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

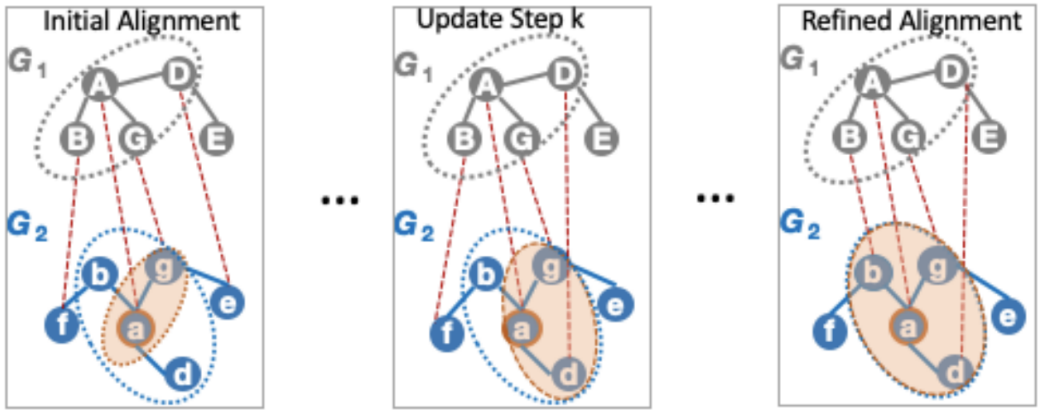


Figure: from Heimann *et al.* 2021. The alignment is iteratively connected so nodes within a neighborhood of one graph are mapped to a similar neighborhood in the other graph.

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 $\mathbf{M}$ from exploding, row normalize $\mathbf{M}$, 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 $\mathbf{M}_K$

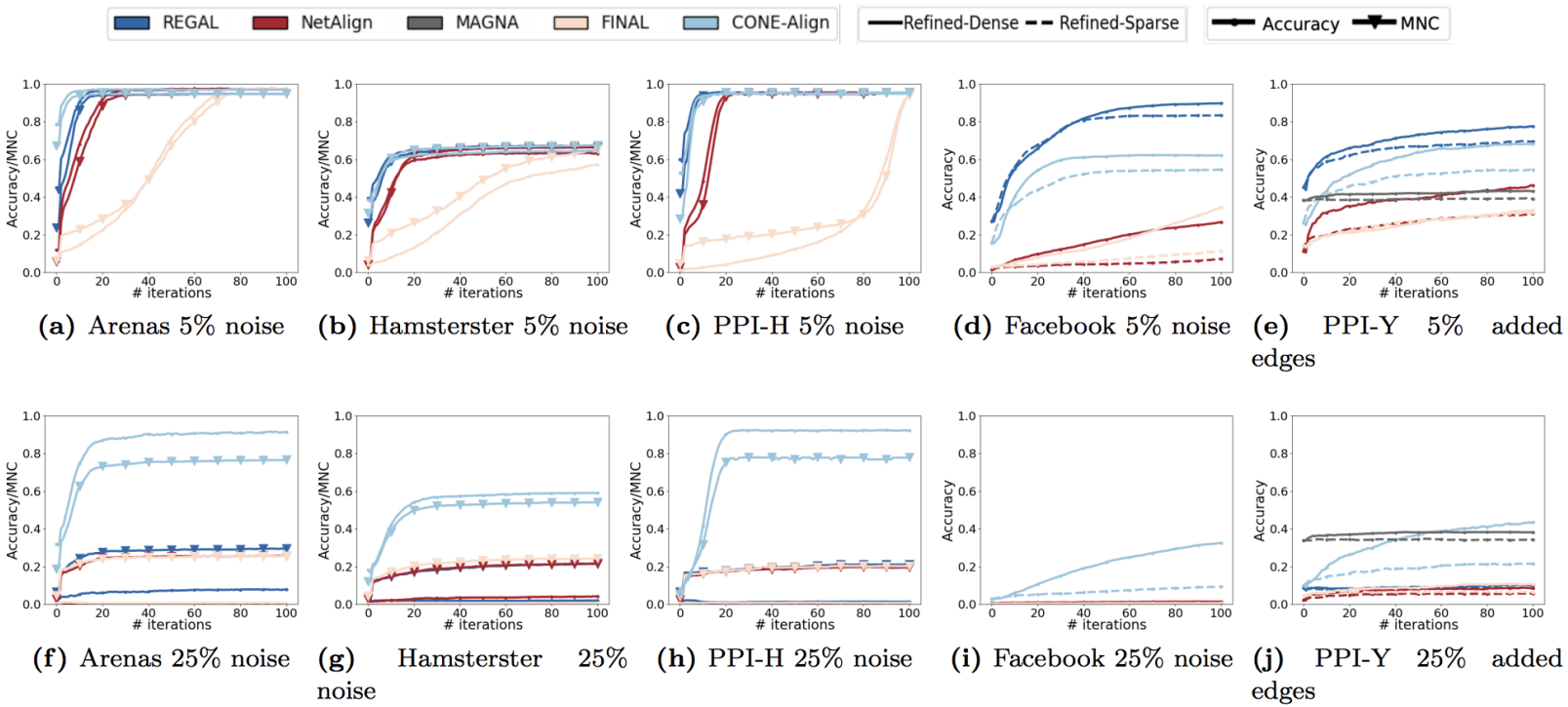
Figure: from Algorithm 1

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 $\mathbf{M}$.

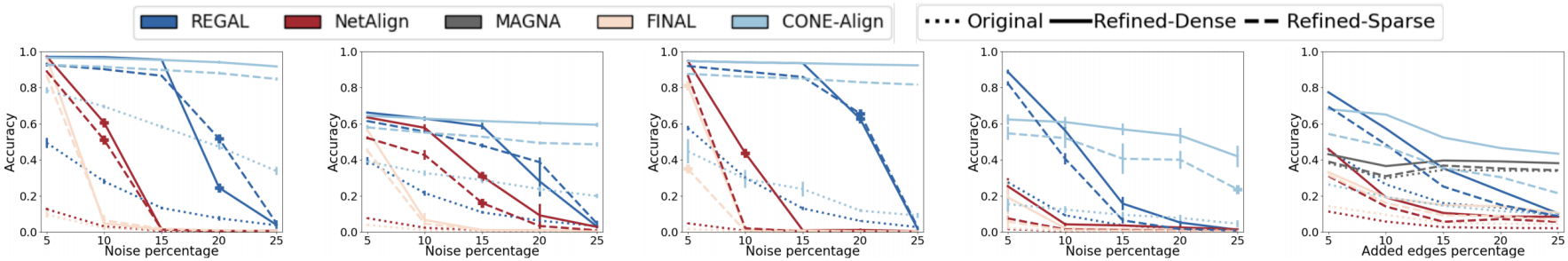
Specifically the updates to $\mathbf{M}$ are modified as,

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

- $\mathbf{U}_k = \text{top} -\alpha\,(\mathbf{A}_1\mathbf{M}\mathbf{A}_2)$ 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...)

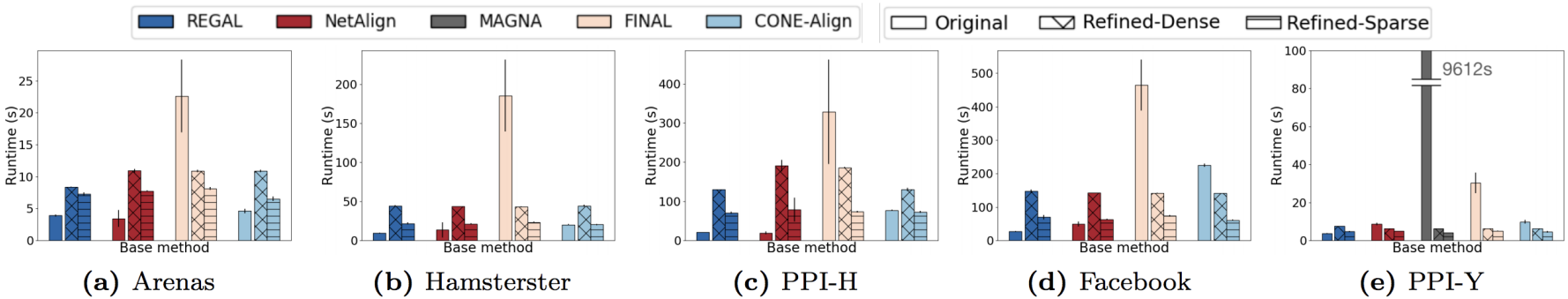


Figure: from Fig. 4.

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.

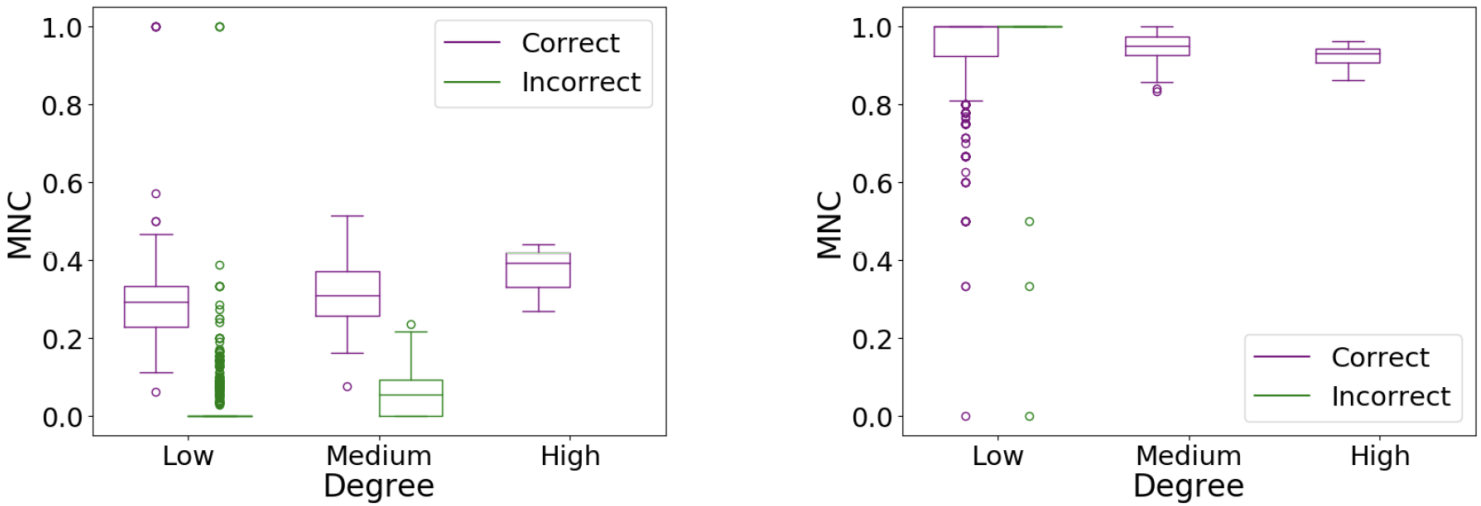


Figure: from Fig. 5

RefiNA Conclusion

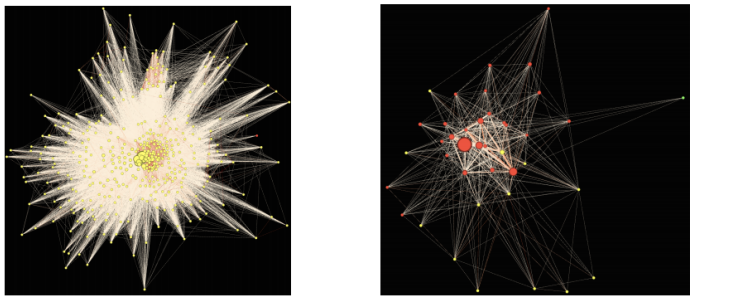
- RefiNA is a posthoc method to clean up a network alignment.
- You can start with an alignment generated with any algorithm (though REGAL looks good!)
- You have the ability to make the updates more sparse by zeroing out entries in $\mathbf{M}$

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).

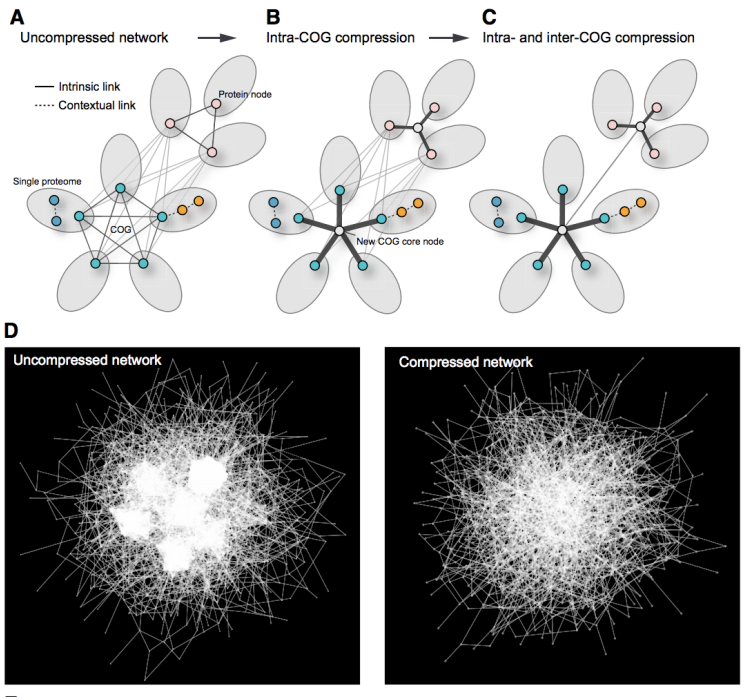


Figure: from Lisewski *et al.* Cell. 2014

The Interplay Between Graph Embedding and Summarization

- Imagine your graph has 1 billion nodes
- Node embedding is effectively summarization, yet depends on the size of the graph.
- Storing 1 billion $\times$ 128 is still a lot.
- **Latent Network Summarization:** Find a low-dimensional representation that is independent of the size of the graph.

Formal Statement

Notice that we should be able to use the summary for downstream tasks!

> DEFINITION 1 (LATENT NETWORK SUMMARIZATION). *Given an arbitrary graph $G = (V, E)$ with $|V| = N$ nodes and $|E| = M$ edges, the goal of latent network summarization is to map the graph G to a low-dimensional $K \times C$ representation $\mathcal{J}$ that summarizes the structure of G, where $K, C \ll N, M$ are independent of the graph size. The output latent representations should be usable in data mining tasks, and sufficient to derive all or a subset of node embeddings on the fly for learning tasks (e.g., link prediction, classification).*

Figure: from Jin *et al.* KDD. 2019

Notation/Definition (Relational Operator)

DEFINITION 4 (RELATIONAL OPERATOR). *A relational operator* $\phi(\mathbf{x}, \mathcal{R}) \in \Phi$ *is defined as a basic function* (e.g., *sum*) *that operates on a feature vector* $\mathbf{x}$ *associated with a set of related elements* $\mathcal{R}$ *and returns a single value.*

For example, let $\mathbf{x}$ be an $N \times 1$ vector of the adjacency matrix and $\mathcal{R}$ be the neighborhood, $\mathcal{N}_i$ of node i. Let ϕ denote sum. Then $\phi(\mathbf{x}, \mathcal{R})$ would return the number of nodes reachable from node i.

Base Features, $\mathbf{F}_0$

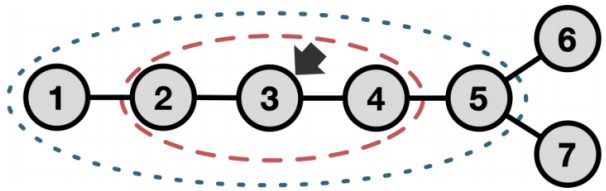
As a simple case, let's assume to compute our base features, we will simply count the degree for each node.

$$f_b\langle \mathbf{b}, \mathcal{N} \rangle = \left[f_b\left(\mathbf{b}, \mathcal{N}_1\right), f_b\left(\mathbf{b}, \mathcal{N}_2\right), \ldots, f_b\left(\mathbf{b}, \mathcal{N}_N\right) \right]^T, \mathbf{b} \in \mathcal{B}$$

The key we will use is that we want to study this for multiple l-order neighborhoods. Having a base level understanding of the statistic for each node is not enough.

Example of All This Business

Here, let **x** represent node degrees.



(a) 1- and 2-hop neighborhood of node 3

Node id: ① ② ③ ④ ⑤ ⑥ ⑦

$$\mathbf{x}^T = [1 \;\; 2 \;\; 2 \;\; 2 \;\; 3 \;\; 1 \;\; 1]$$

$$\max\langle \mathbf{x}, \mathcal{N} \rangle^T = [2 \;\; 2 \;\; 2 \;\; 3 \;\; 3 \;\; 3 \;\; 3]$$

Figure: from Fig. 3

Higher-Order Features Through Histograms

- An example, a histogram of degree distributions of k-hop neighborhoods.
- For each node, make a histogram for all of the degrees within l hops. [REGAL]
- You can make these statistics arbitrarily complex. The important thing is that you will do it for multiple levels of l.

k-order graph, illustrated

In this example, consider the subgraph induced by all nodes within k hops or less. For our purpose, we would just record those node indices and their degrees to put in a histogram.

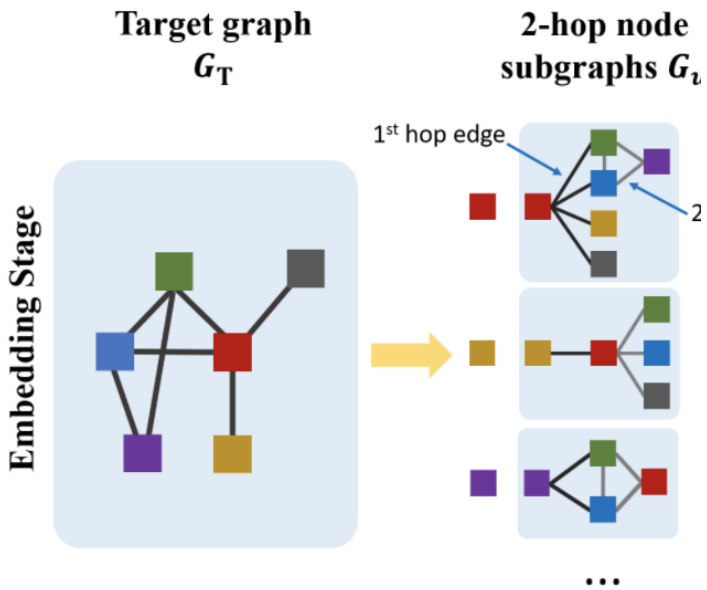


Figure: from `http://snap.stanford.edu/subgraph-matching/`

Multilevel Summarization

- Consider the notation where $\mathbf{H}^\ell$ gives a histogram for the l-th order ego network.
- $\mathbf{H}^\ell = \mathbf{Y}^\ell \mathbf{S}^\ell$ $(N \times c \rightarrow (N \times k) \cdot (k \times c))$
- Here, $\mathbf{Y}^\ell$ is the dense embedding matrix that we don't actually want.

Here, the solution is to do an SVD on $\mathbf{H}^\ell$, with,

- $\mathbf{Y}^\ell = \mathbf{U}^\ell \sqrt{\Sigma^\ell}$
- $\mathbf{S}^\ell = \sqrt{\Sigma^\ell} \mathbf{V}^{\ell T}$

* Note same story as always with Σ storing singular values on the diagonal.

Recap, $\mathbf{S}^\ell$

- $\mathbf{S}^\ell$ contains the best k-dimensional subspace for a given context, H^ℓ.
- You can compute the $\mathbf{Y}^\ell$ on the fly if you for some reason need an embedding.