

Comp790-166: Computational Biology

Lecture 23

April 5, 2022

Today

- Graph alignment [Regal]
- Refining graph alignments [REFINA]
- Homework will be available today or tomorrow. Watch out for an email.

Announcements

- Project Presentations April 25 and April 27. Please visit the signup sheet. https://docs.google.com/spreadsheets/d/1_z1NBffJF8do8JrasTQ1-8pS-ATR2ScI7SutRDPIf80/edit?usp=sharing
- Final Project LaTeX template https://github.com/natalies-teaching/Comp790-166-CompBio-Spring2022/tree/main/Project_Final_Writeup

Problem

Here is a formal definition of the graph matching problem.

PROBLEM 1. *Given two graphs G_1 and G_2 with node-sets \mathcal{V}_1 and \mathcal{V}_2 and possibly node attributes \mathcal{A}_1 and \mathcal{A}_2 resp., devise an efficient **network alignment method** that aligns nodes by learning **directly comparable** node representations \mathbf{Y}_1 and \mathbf{Y}_2 , from which a node mapping $\phi : \mathcal{V}_1 \rightarrow \mathcal{V}_2$ between the networks can be inferred.*

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

Node Attributes

- Information about nodes un-related from connectivity
- Such as protein modification status of a protein, despite other interacting proteins

Overview of Regal Method

Regal \rightarrow Representation Based Graph Alignment.

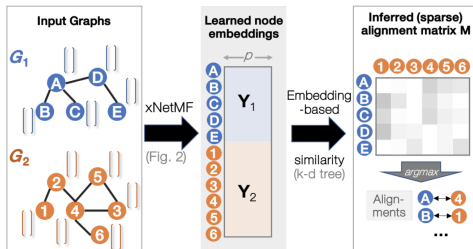


Figure: from Heimann *et al.* CIKM 2018. Similar to Node2vec, the authors want to find a representation for each node

What does this mean for us? Aligning nodes across multiple biological networks, for example.

Typical Network Alignment Problem Formulation

Given two graphs, \mathbf{A}_1 and \mathbf{A}_2 , find the permutation matrix, \mathbf{P} that minimizes the following.

$$\|\mathbf{P}\mathbf{A}_1\mathbf{P}^T - \mathbf{A}_2\|_F^2$$

Many of the proposed solutions might start with a 'seed' alignment to make the problem easier. We will see that the REGAL solution does not require any seeds.

Defining Node Identity

- Previously (e.g. node2vec), we saw node representations were defined in terms of their neighbors through random walks.
- In this setting, of quantifying relationships between nodes between graphs, we cannot walk because we have two different graphs.
- The solution is to instead focus on nodes with similar structural roles (e.g. degree, degree of neighbors, etc).

Calculating Node Similarity Within and Between Graphs

Define distance between nodes u and v in terms of structure (\mathbf{d}), or attributes (\mathbf{f}) as,

$$\text{sim}(u, v) = \exp \left[-\gamma_s \cdot \|\mathbf{d}_u - \mathbf{d}_v\|_2^2 - \gamma_a \cdot \text{dist}(\mathbf{f}_u, \mathbf{f}_v) \right]$$

Each \mathbf{d}_u is defined as,

$$\mathbf{d}_u = \sum_{k=1}^K \delta^{k-1} \mathbf{d}_u^k$$

Here, δ is a discount factor for greater hop distances.

Expensive Formulation

Given a factorization approach, write the similarity matrix, \mathbf{S} as,

$$\mathbf{S} \approx \mathbf{Y}\mathbf{Z}^T$$

Here, \mathbf{Y} gives the node-to-embedding matrix.

Intuitively, we want the following to be as close as possible to 0,

$$\|\mathbf{S} - \mathbf{Y}\mathbf{Z}^T\|$$

An Approximation with Landmarks

- The punchline is that \mathbf{S} will be approximated with a low-rank matrix, $\tilde{\mathbf{S}}$, which is never explicitly computed!
- The solution is to choose $p \ll n$ 'landmark' nodes, chosen across both graphs (G_1, G_2) .
- Compute similarity between each node and each landmark. This produces an $n \times p$ matrix, \mathbf{C} .
- Further, the $p \times p$ landmark-to-landmark submatrix can also be extracted (\mathbf{W}).

The Low-Rank Matrix, $\tilde{\mathbf{S}}$

Finally, the low-rank matrix $\tilde{\mathbf{S}}$ is given as,

$$\tilde{\mathbf{S}} = \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^T$$

Here \mathbf{W}^{-1} is computed as the pseudoinverse of \mathbf{W}

- The problem is that because we need to consider similarity between all pairs of nodes, this is still an n^2 computation.

Approximation for the Embedding Matrix, \mathbf{Y}

THEOREM 3.1. *Given graphs $G_1(\mathcal{V}_1, \mathcal{E}_1)$ and $G_2(\mathcal{V}_2, \mathcal{E}_2)$ with $n \times n$ joint combined structural and attribute-based similarity matrix $\mathbf{S} \approx \mathbf{Y}\mathbf{Z}^T$, its node embedding matrix \mathbf{Y} can be approximated as*

$$\tilde{\mathbf{Y}} = \mathbf{C}\mathbf{U}\Sigma^{1/2},$$

where \mathbf{C} is the $n \times p$ matrix of similarities between the n nodes and p randomly chosen landmark nodes, and $\mathbf{W}^\dagger = \mathbf{U}\Sigma\mathbf{V}^\top$ is the full rank singular value decomposition of the pseudoinverse of the small $p \times p$ landmark-to-landmark similarity matrix \mathbf{W} .

So to Summarize the Entire xNetMF

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

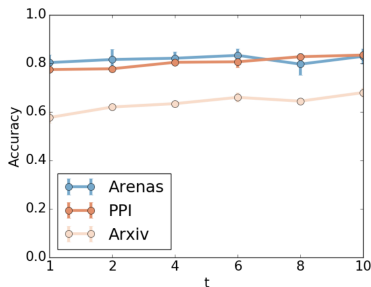
```
1: ===== STEP 1. Node Identity Extraction =====
2: for node  $u$  in  $\mathcal{V}_1 \cup \mathcal{V}_2$  do
3:   for hop  $k$  up to  $K$  do            $\triangleright$  counts of node degrees of  $k$ -hop neighbors of  $u$ 
4:      $d_u^k = \text{CountDegreeDistributions}(\mathcal{R}_u^k)$             $\triangleright 1 \leq K \leq \text{graph diameter}$ 
5:   end for
6:    $d_u = \sum_{k=1}^K \delta^{k-1} d_u^k$             $\triangleright$  discount factor  $\delta \in (0, 1]$ 
7: end for

8: ===== STEP 2. Efficient Similarity-based Representation =====
9: ===== STEP 2a. Reduced  $n \times p$  Similarity Computation =====
10:  $\mathcal{L} = \text{ChooseLandmarks}(G_1, G_2, p)$             $\triangleright$  choose  $p$  nodes from  $G_1, G_2$ 
11: for node  $u$  in  $\mathcal{V}$  do
12:   for node  $v$  in  $\mathcal{L}$  do
13:      $c_{uv} = e^{-\gamma_s \cdot \|d_u - d_v\|_2^2 - \gamma_a \cdot \text{dist}(f_u, f_v)}$ 
14:   end for
15: end for            $\triangleright$  Used in low-rank approx. of similarity graph (not constructed)

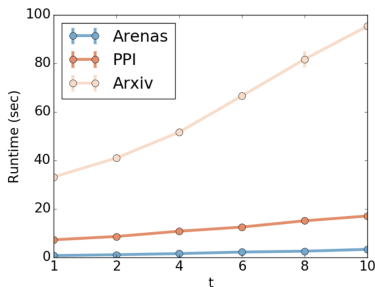
16: ===== STEP 2b. From Similarity to Representation =====
17:  $W = C[\mathcal{L}, \mathcal{L}]$             $\triangleright$  Rows of  $C$  corresponding to landmark nodes
18:  $[U, \Sigma, V] = \text{SVD}(W^\dagger)$ 
19:  $\tilde{Y} = CU\Sigma^{-\frac{1}{2}}$             $\triangleright$  Embedding: implicit factorization of similarity graph
20:  $\tilde{Y} = \text{Normalize}(\tilde{Y})$             $\triangleright$  Postprocessing: make embeddings have magnitude 1
21:  $\tilde{Y}_1, \tilde{Y}_2 = \text{Split}(\tilde{Y})$             $\triangleright$  Separate representations for nodes in  $G_1, G_2$ 
22: return  $\tilde{Y}_1, \tilde{Y}_2$ 
```

We Still Have Another Step (matching nodes between graphs)

But a question of interest is, how many landmarks do we need? These landmarks will be our effective embedding dimension.



(a) Accuracy w.r.t. # of landmarks



(b) Runtime w.r.t. # of landmarks

Figure: Here $p = t \log_2 n$

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

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

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

- Match nodes according to these similarity scores.
- **Soft Scoring Approach:** From kd-tree, find $\alpha \ll N$ 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

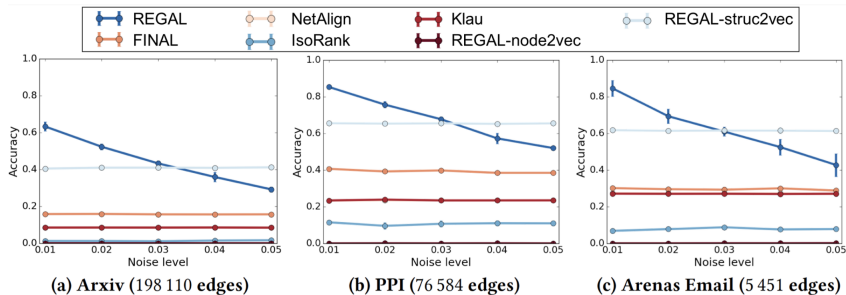


Figure: from Fig. 4, Replacing the embedding step with struct2vec produces better results when there is more noise.

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

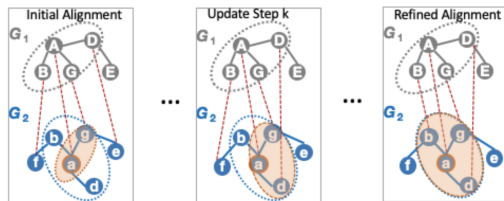


Figure: from Heimann *et al.* 2021. The alignment is iteratively connected so nodes within a neighborhood of one graph are mapped to a similar neighborhood in the other graph.

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.
 - $\tilde{\mathcal{N}}_{G_2}^\pi(i) = \{j \in \mathcal{V}_2 : \exists k \in \mathcal{N}_{G_1}(i) \text{ s.t. } \pi(k) = j\}$

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

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

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 be rewritten in matrix form, such that $\text{MNC}(i,j) = \mathbf{S}_{ij}^{\text{MNC}}$ as,

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

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,

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

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

```
1: Input: adjacency matrices  $\mathbf{A}_1, \mathbf{A}_2$ , initial align-
   alignment matrix  $\mathbf{M}_0$ , number of iterations  $K$ , token
   match score  $\epsilon$ 
2: for  $k = 1 \rightarrow K$  do                                 $\triangleright$  Refinement iterations
3:    $\mathbf{M}_k = \mathbf{M}_{k-1} \circ \mathbf{A}_1 \mathbf{M}_{k-1} \mathbf{A}_2$        $\triangleright$  MNC update
4:    $\mathbf{M}_k = \mathbf{M}_k + \epsilon$                                  $\triangleright$  Add token match scores
5:    $\mathbf{M}_k = \text{Normalize}(\mathbf{M}_k)$        $\triangleright$  By row then column
6: end for
7: 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 iterations.

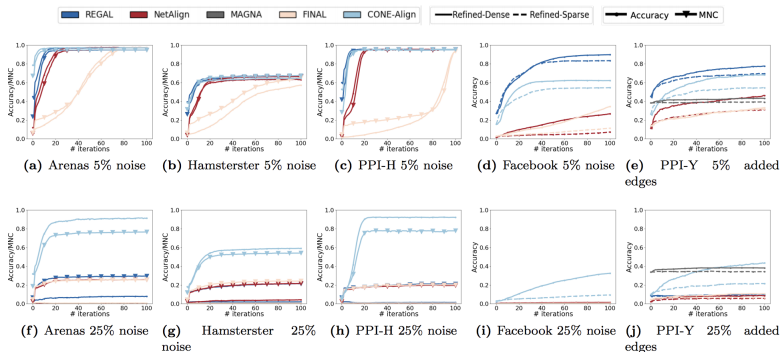


Figure: from Fig. 7

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} , $\tilde{\mathbf{A}} = \mathbf{P}\mathbf{A}\mathbf{A}^T$. Then remove edges with some probability, p_s .

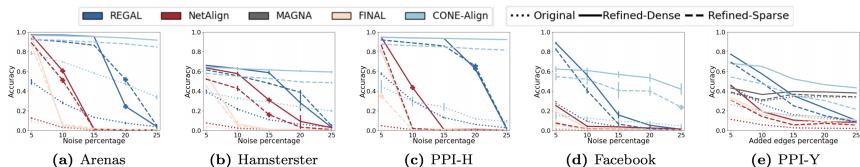


Figure: from Fig. 2. Refinement with the sparse or dense formulation helps networks with different structures.

Run-time

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

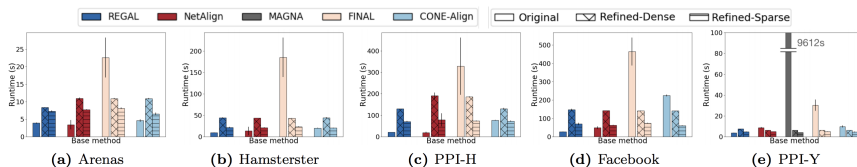


Figure: from Fig. 4.

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.

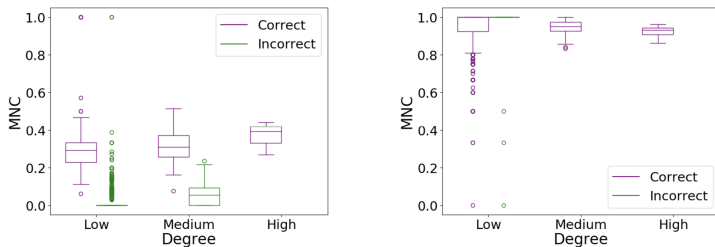


Figure: from Fig. 5

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 **M**