Comp790-166: Computational Biology

Lecture 19

April 2, 2023

Today

- Finishing Up Graph Alignment
  - REGAL graph alignment.
  - Refining graph alignments with Refina

Quick Announcements

- Homework 2 will be assigned sometime this week.
- Final project presentations will be on the last two days of class (April 24 and April 26). Note that you can just give us an update. The writeup will be due on our Final Exam Day (May 1).

Review Questions

1. What was the meaning of the **w** and **x** inferred through the Mashup appproach?
2. How were the **x** and **w** used to estimate the random walk with restart probabilities?

Review of Motivation of Graph Alignment Problem

Graph Alignment: What happens now if we have multiple graphs, but we don't know how to register the nodes, in the sense of which nodes align based both on connectivity patterns and node-level features?

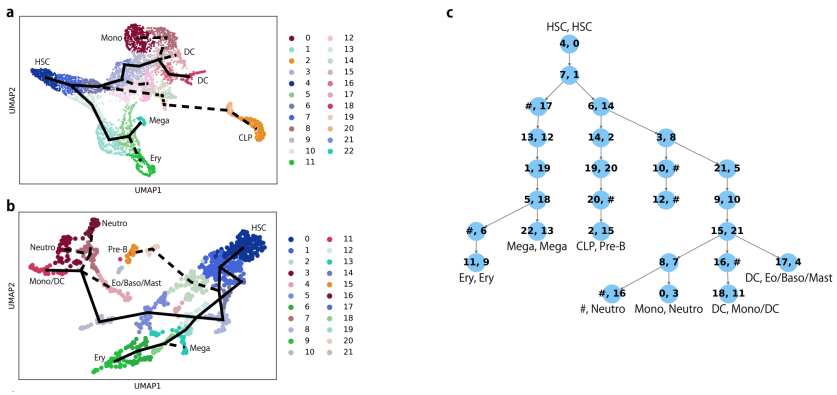


Figure: from Sugihara *et al.* Nature Communications, 2022

Overview of Regal Method

Regal → Representation Based Graph Alignment.

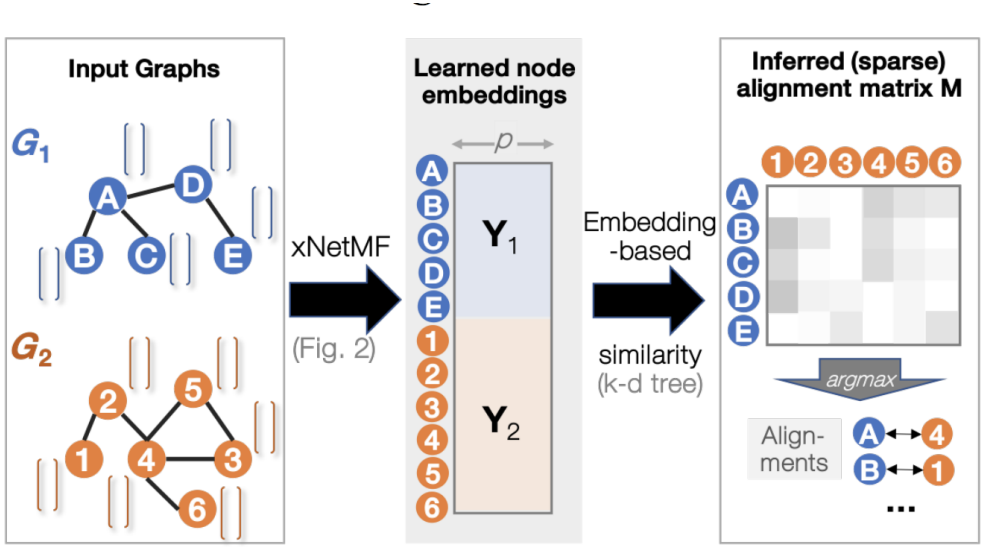


Figure: from from Heimann *et al.* CIKM 2018. Similar to Node2vec, the authors want to find a representation for each node

From our earlier example, we would get a mapping of cell-populations from one dataset to the other.

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,

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

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 $n \times n$ similarity matrix, $\mathbf{S}$ as,

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

Here, $\mathbf{Y}$ gives the node-to-embedding matrix (which is what we want).
Intuitively, we want the following to be as close as possible to 0,

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

($\mathbf{Z}$ and $\mathbf{Y}$ are both $n \times p$ matrices)

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 $p << n$ '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,

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

Here $\mathbf{W}^{\dagger}$ is computed as the pseudoinverse of $\mathbf{W}$

- The problem is that $\tilde{\mathbf{S}}$ is still an $n \times n$ matrix, and we still need to find the embedding matrix, $\mathbf{Y}$

Approximation for the Embedding Matrix, $\mathbf{Y}$

Theorem: Given graphs, $\mathcal{G}_1(\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2(\mathcal{V}_2, \mathcal{E}_2)$ with an $n \times n$ 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,

$$\tilde{Y} = CU\Sigma^{-1/2}. \tag{1}$$

Here, $\mathbf{C}$ is the $n \times p$ matrix of similarities between the n nodes and the p chosen landmarks and $\mathbf{W}^\dagger = \mathbf{U}\Sigma\mathbf{V}^T$ is the full rank SVD of the small $p \times p$ landmark similarity matrix matrix, $\mathbf{W}$

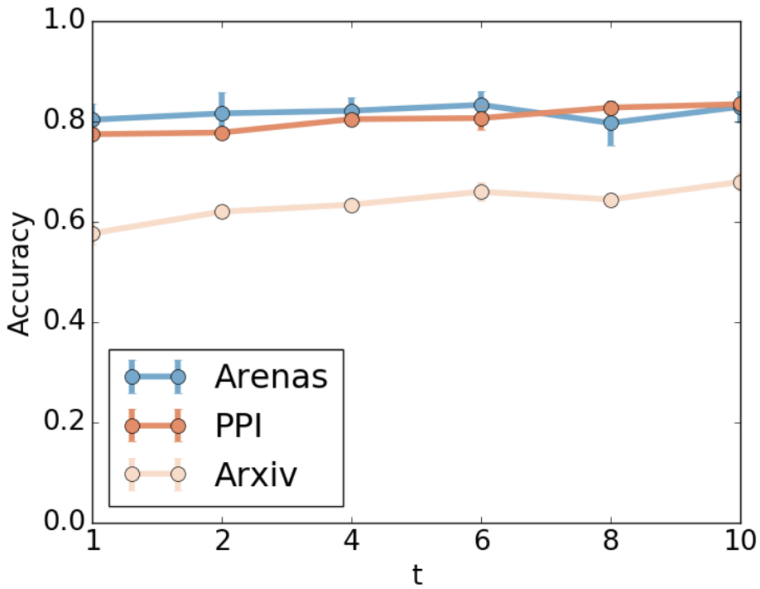
So to Summarize the Entire xNetMF

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

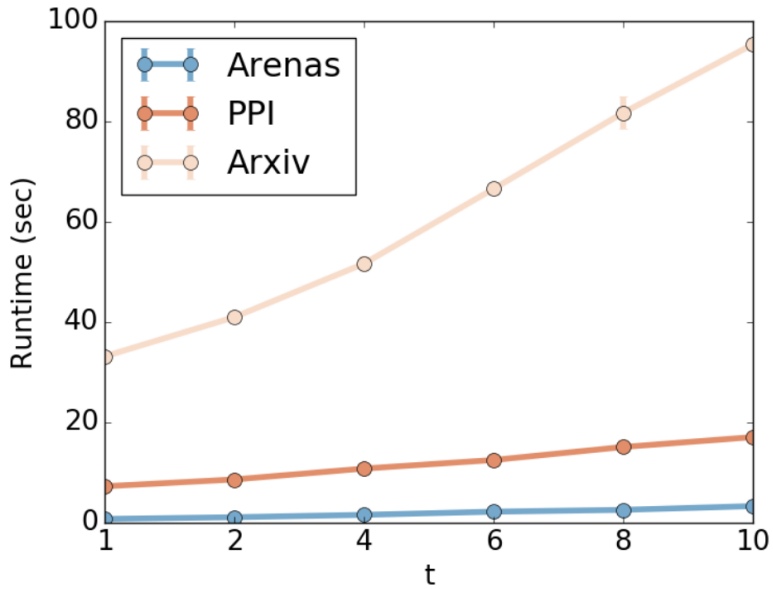
========== STEP 1. Node Identity Extraction ==========
for node u in $\mathcal{V}_1 \cup \mathcal{V}_2$ **do**
  for hop k up to K **do** ▷ counts of node degrees of k-hop neighbors of u
    $\mathbf{d}_u^k$ = CountDegreeDistributions($\mathcal{R}_u^k$) ▷ $1 \leq K \leq$ graph diameter
  end for
  $\mathbf{d}_u = \sum_{k=1}^{K} \delta^{k-1} \mathbf{d}_u^k$ ▷ 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 $\mathbf{C}$ corresponding to landmark nodes
$[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] = \text{SVD}(\mathbf{W}^\dagger)$
$\tilde{\mathbf{Y}} = \mathbf{C}\mathbf{U}\mathbf{\Sigma}^{-\frac{1}{2}}$ ▷ **Embedding**: *implicit* factorization of similarity graph
$\tilde{\mathbf{Y}} = Normalize(\tilde{\mathbf{Y}})$ ▷ **Postprocessing**: make embeddings have magnitude 1
$\tilde{\mathbf{Y}}_1, \tilde{\mathbf{Y}}_2$ = Split($\tilde{\mathbf{Y}}$) ▷ **Separate** representations for nodes in G_1, G_2
return $\tilde{\mathbf{Y}}_1, \tilde{\mathbf{Y}}_2$

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 $\tilde{Y}_1$ 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.

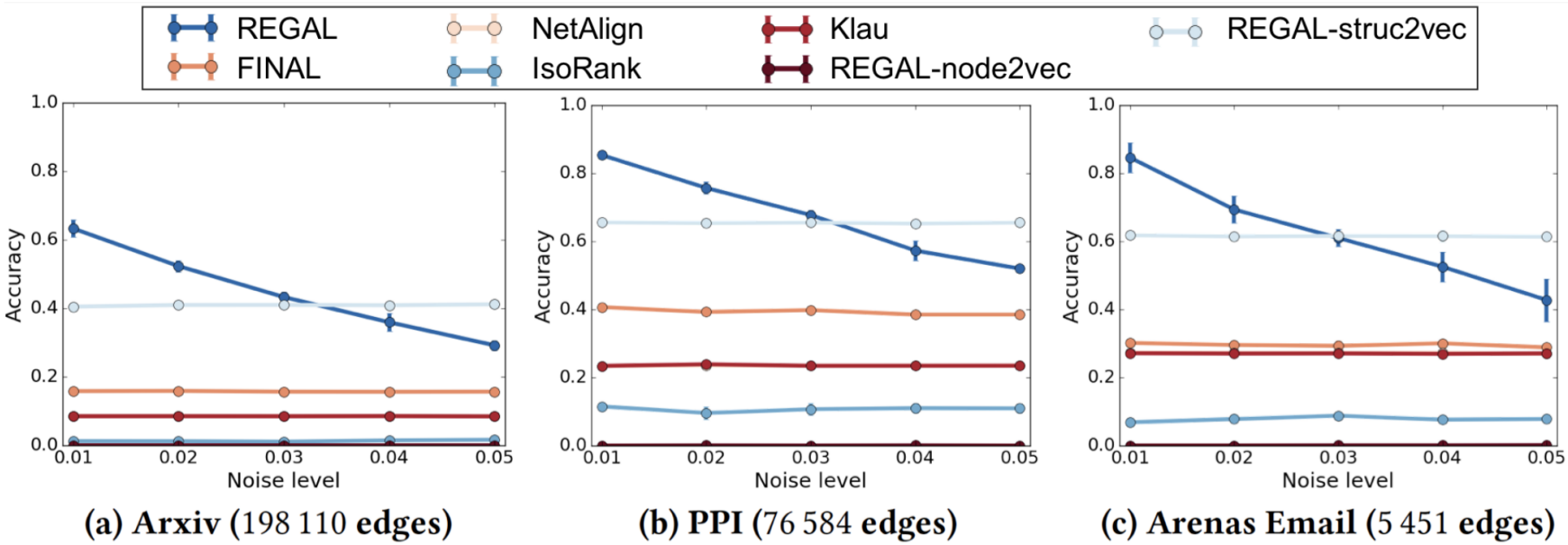
Experimental Evaluation

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



(a) **Arxiv** (198 110 **edges**) (b) **PPI** (76 584 **edges**) (c) **Arenas Email** (5 451 **edges**)

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

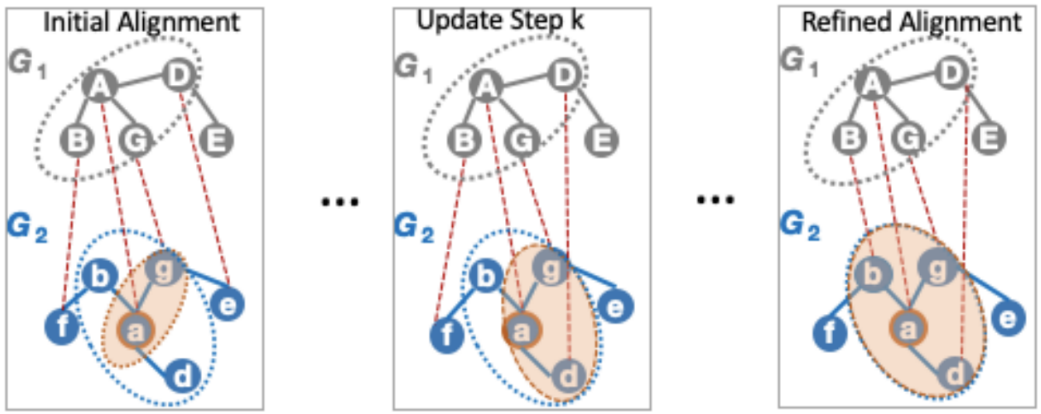


Figure: from Heimann *et al.* 2021. The alignment is iteratively connected so nodes within a neighborhood of one graph are mapped to a similar neighborhood in the other graph.

A Similar Formulation of the Graph Alignment Problem from REGAL

- A node alignment is a function, $\pi : \mathcal{V}_1 \to \mathcal{V}_2$ 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 $\mathbf{M}_K$

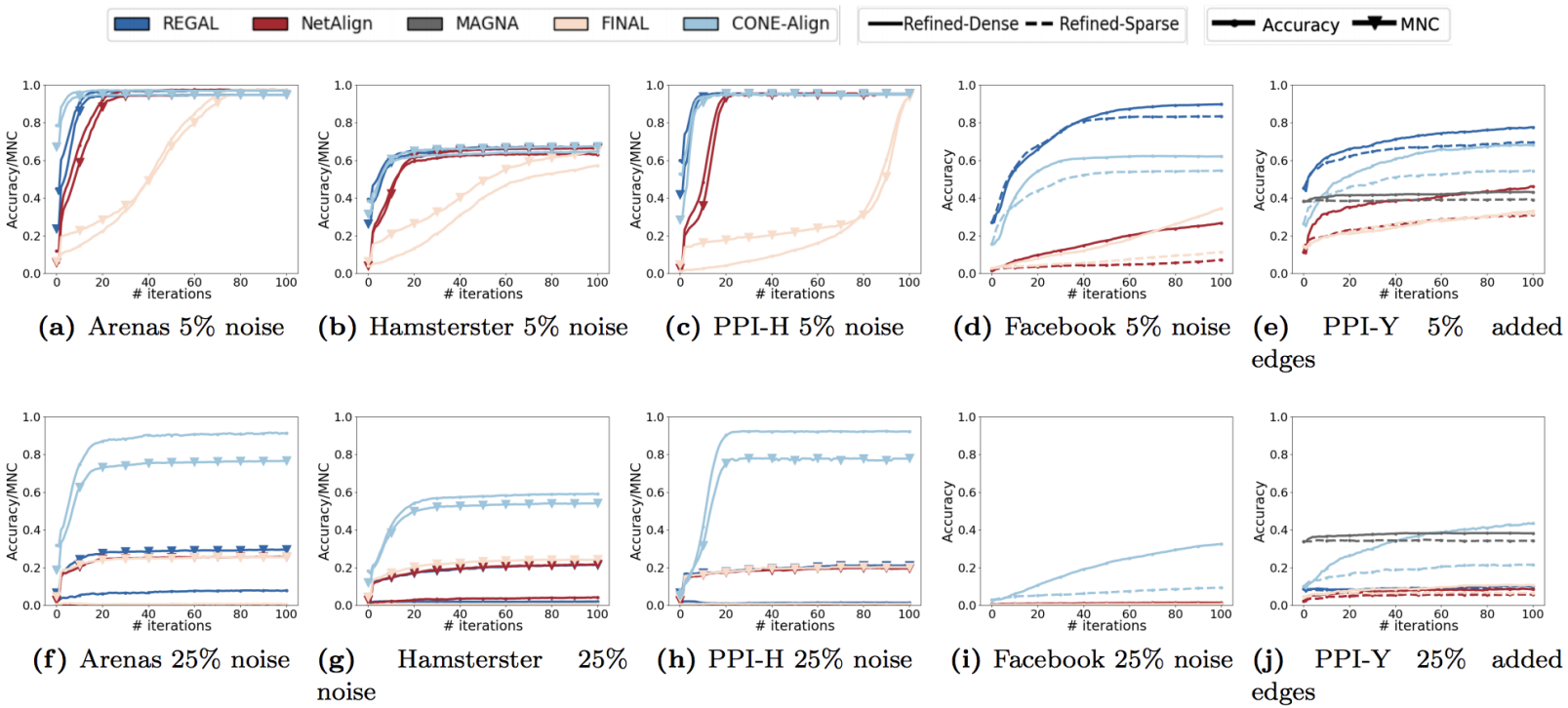
Figure: from Algorithm 1

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 $\mathbf{M}$.

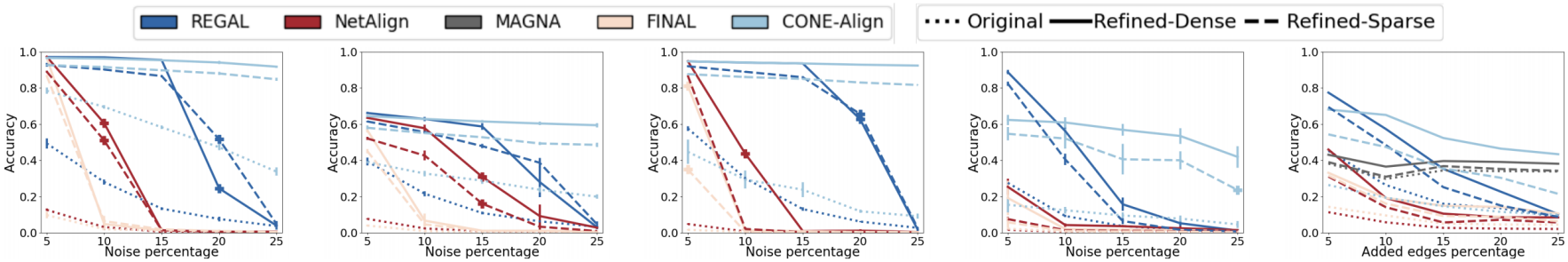
Specifically the updates to $\mathbf{M}$ are modified as,

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

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

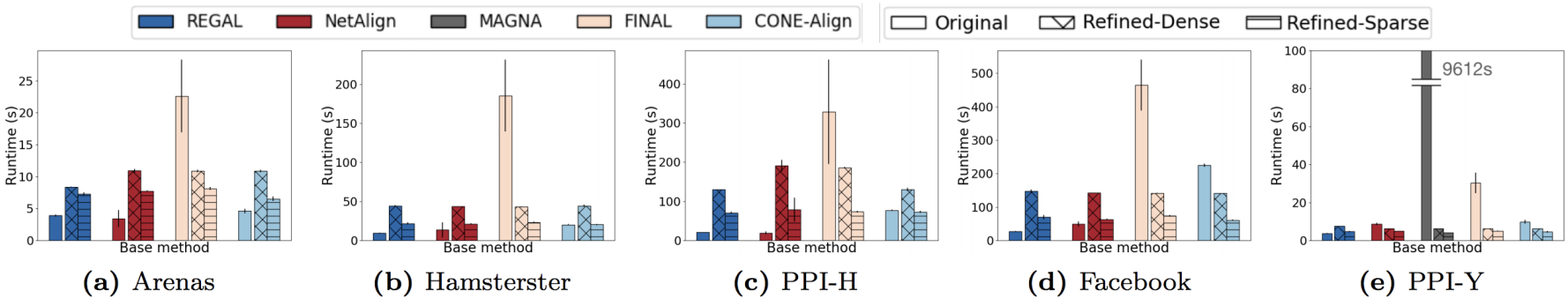


Figure: from Fig. 4.

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.

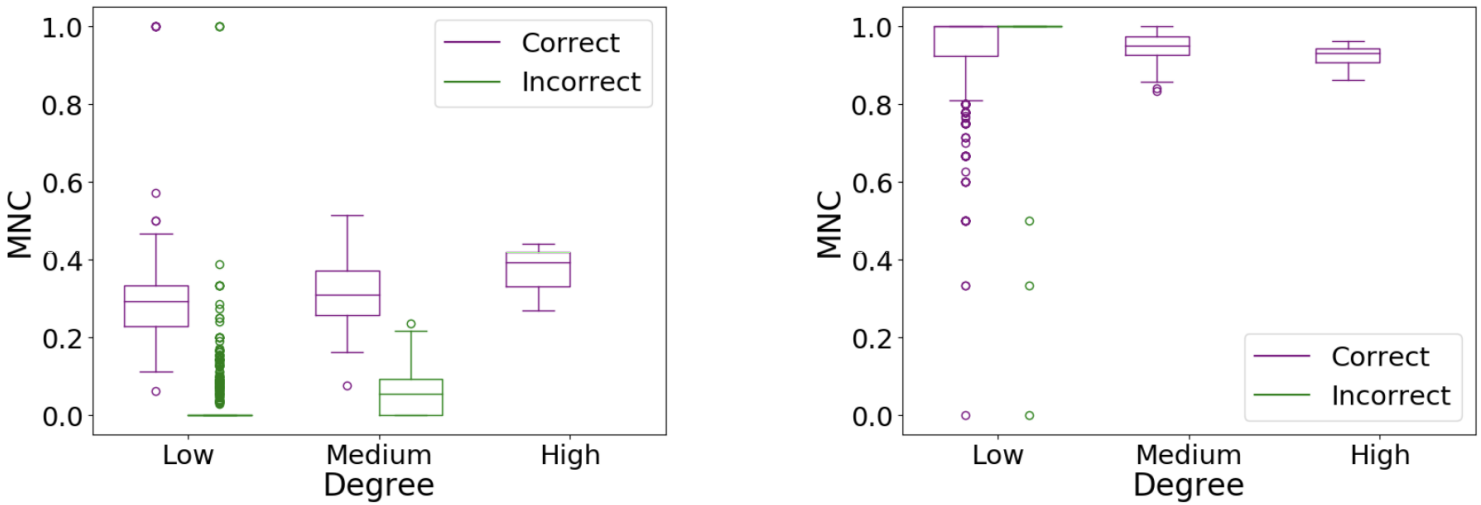


Figure: from Fig. 5

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 $\mathbf{M}$