2013 IEEE 13th International Conference on Data Mining

BIG-ALIGN: Fast Bipartite Graph Alignment

Danai Koutra*
Carnegie Mellon University
danai@cs.cmu.edu

Hanghang Tong
City College of New York
tong@cs.ccny.cuny.edu

David Lubensky
IBM T.J. Watson
davidlu@us.ibm.com

***Abstract*—How can we find the virtual twin (i.e., the same or similar user) on LinkedIn for a user on Facebook? How can we effectively link an information network with a social network to support cross-network search? Graph alignment – the task of finding the node correspondences between two given graphs – is a fundamental building block in numerous application domains, such as social networks analysis, bioinformatics, chemistry, pattern recognition.**

In this work, we focus on aligning *bipartite* graphs, a problem which has been largely ignored by the extensive existing work on graph matching, despite the ubiquity of those graphs (e.g., users-groups network). We introduce a new optimization formulation and propose an effective and fast algorithm to solve it. We also propose a fast generalization of our approach to align *unipartite* graphs. The extensive experimental evaluations show that our method outperforms the state-of-art graph matching algorithms in both alignment accuracy and running time, being up to $10\times$ more accurate or $174\times$ faster on real graphs.

I. INTRODUCTION

Can we spot the *same* people in two different social networks, say LinkedIn and Facebook? An equally interesting question is how to find *similar* people across different graphs. In both settings, a key step is to align[1] the two graphs in order to reveal similarities between the nodes of the two networks.

Informally, the problem is defined as follows: given two graphs, $G_A(\mathcal{N}_A, \mathcal{E}_A)$ and $G_B(\mathcal{N}_B, \mathcal{E}_B)$ where $\mathcal{N}$ and $\mathcal{E}$ are the node and edge sets respectively, how can we permute their nodes, so that the graphs have as similar structure as possible? This is a core building block in many desciplines as it essentially enables us to link the different networks so that we can search and/or transfer valuable knowledge across different networks. The notions of graph similarity and alignment appear in many disciplines such as protein-protein alignment [3] [7], chemical compound comparison [23], information extraction for finding synonyms in a single language or translation between different languages [3], answering similarity queries in databases [17], and pattern recognition [9] [29].

Among others, bipartite graphs stand for an important class of real graphs and appear in many different settings, such as author-conference publishing graphs, user-group membership graphs, user-movie rating graphs. Despite their ubiquity, most, if not all, of the existing work on graph alignment is tailored for unipartite graphs and, thus, might be sub-optimal for bipartite graphs.

In this paper, we mainly focus on the alignment of such bipartite graphs. Our main contributions are:

* Work done during an internship at IBM T.J. Watson.
[1]Throughout this work we use the words "align(ment)" and "match(ing)" interchangeably.

1) *Formulations.* We introduce a powerful primitive with new constraints for the graph matching problem.
2) *Algorithms.* We propose an effective and fast procedure, BIG-ALIGN, to solve the constrained optimization problem with careful handling of many subtleties. Then, we further generalize it for matching unipartite graphs (UNI-ALIGN).
3) *Evaluations.* We conduct extensive experiments, which demonstrate that our algorithms, BIG-ALIGN and UNI-ALIGN, are superior to existing graph matching methods in terms of both accuracy and and efficiency.

The rest of the paper is organized as follows: Section II presents the formal definition of the graph matching problem we are addressing; Section III our proposed method; and Section V the experimental results. Finally, we give the related work, discussion, and conclusions in Sections VI, VII and VIII respectively.

II. PROPOSED PROBLEM FORMULATION

In the past three decades, numerous communities studied the problem of graph alignment, as it arises in many settings. However, most of the research work has focused on *unipartite* graphs, i.e. graphs that consist of only one type of nodes. Formally, the problem addressed in the past is the following: Given two *unipartite* graphs, G_A and G_B, with adjacency matrices $\mathbf{A}$ and $\mathbf{B}$, find the permutation matrix $\mathbf{P}$ that minimizes the cost function f_{uni}:

$$\min_{\mathbf{P}} f_{uni}(\mathbf{P}) = \min_{\mathbf{P}} ||\mathbf{P}\mathbf{A}\mathbf{P}^T - \mathbf{B}||_F^2,$$

where $|| \bullet ||_F$ is the Frobenius norm of the corresponding matrix. We list the frequently used symbols in Table I. The permutation matrix $\mathbf{P}$ is a square binary matrix with exactly one entry 1 in each row and column, and 0s elsewhere. Effectively, it reorders the rows of the adjacency matrix $\mathbf{A}$, while its transpose reorders the columns of the matrix, so that the resulting reordered matrix is "close" to $\mathbf{B}$.

In this work, we introduce the problem of aligning *bipartite* graphs, i.e., graphs whose edges connect two disjoint sets of vertices (there are no edges within the two node sets). One example of such graphs is the user-group graph; the first set of nodes consists of users, the second set of groups, and the edges represent user memberships. Throughout the paper we will consider the alignment of the "user-group" LinkedIn graph ($\mathbf{A}$) with the "user-group" Facebook graph ($\mathbf{B}$). In a more general setting, the reader may think of the first set consisting of nodes and the second set of communities.

First, we extend the traditional *unipartite* graph alignment problem definition to *bipartite* graphs:

TABLE I: Description of major symbols.

Notation	Description
$\mathbf{A}, \mathbf{B}$	adjacency matrix of bipartite graph G_A, G_B
$\mathbf{A}^T, \mathbf{B}^T$	transpose of matrix $\mathbf{A}$, $\mathbf{B}$
$\mathcal{N}_A$, $\mathcal{N}_B$	set of nodes of $\mathbf{A}$, $\mathbf{B}$
$\mathcal{E}_A$, $\mathcal{E}_B$	set of edges of $\mathbf{A}$, $\mathbf{B}$
n_{A1}, n_{A2}	number of nodes of graph $\mathbf{A}$ in set 1 and 2 resp.
n_{B1}, n_{B2}	number of nodes of graph $\mathbf{B}$ in set 1 and 2 resp.
$\mathbf{P}$	user-level (node-level) correspondence matrix
$\mathbf{Q}$	group-level (community-level) correspondence matrix
$\mathbf{P}^{(v)}$	row or column vector of matrix P
$\mathbf{1}$	vector of 1s
$\|\|A\|\|_F$	$= \sqrt{Tr(A^T A)}$, Frobenius norm of $\mathbf{A}$
λ, μ	sparsity penalty parameters for $\mathbf{P}$, $\mathbf{Q}$ resp. (equiv. to lasso)
η_P, η_Q	step of gradient descent for $\mathbf{P}$, $\mathbf{Q}$
ϵ	small constant (> 0) for the convergence of grad. descent

Problem 1 (Adaptation of Traditional Definition): Given two **bipartite** graphs, G_A and G_B, with adjacency matrices $\mathbf{A}$ and $\mathbf{B}$, find the **permutation** matrices $\mathbf{P}$ and $\mathbf{Q}$ that minimize the cost function f_0:

$$\min_{\mathbf{P},\mathbf{Q}} f_0(\mathbf{P},\mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}} ||\mathbf{PAQ} - \mathbf{B}||_F^2,$$

where $|| \bullet ||_F$ is the Frobenius norm of the matrix.

We note that in this case there are two different permutation matrices that reorder the rows and columns of $\mathbf{A}$ “independently”. However, this formulation has two main shortcomings:
[S1] It is hard to solve, due to its *combinatorial* nature.
[S2] The permutation matrices imply that we are in search for *hard assignments* between the nodes of the input graphs. However, finding hard assignments might not be possible nor realistic. For instance, in the case of input graphs with perfect ‘star’ structure, aligning their spokes (peripheral nodes) is impossible, as they are identical from the structural viewpoint. In other words, any way of aligning the spokes is *equiprobable*. In such and more complicated and realistic cases, soft assignment may be more valuable than hard assignment.

To deal with these issues, we relax Problem 1 that is directly adapted from the well-studied case of unipartite graphs, and state it in a more realistic way:

Problem 2 (Soft, Sparse Bipartite Graph Alignment): Given two **bipartite** graphs, G_A and G_B, with adjacency matrices $\mathbf{A}$ and $\mathbf{B}$, find the **correspondence** matrices $\mathbf{P}$, $\mathbf{Q}$ that minimize the cost function f:

$$\min_{\mathbf{P},\mathbf{Q}} f(\mathbf{P},\mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}} ||\mathbf{PAQ} - \mathbf{B}||_F^2$$

under the following constraints:

(1) [Probabilistic] each matrix element is a probability, i.e. $0 \le P_{ij} \le 1$ and $0 \le Q_{ij} \le 1$, and
(2) [Sparsity] the matrices are sparse, i.e. $||\mathbf{P}^{(v)}||_0 \le t$ and $||\mathbf{Q}^{(v)}||_0 \le t$ for some small, positive constant t. The $||\bullet||_0$ denotes the l_0-norm of the enclosed vector, i.e., the number of its non-zero elements.

The *first constraint*, the requirement of non-integer entries for the matrices, has two advantages:
[A1] It solves both shortcomings of the traditional-based problem. The optimization problem is easier to solve, and has a realistic, probabilistic interpretation; it does not provide only the 1-to-1 correspondences, but also reveals similarities between nodes across networks. The entries of the correspondence matrix $\mathbf{P}$ (or $\mathbf{Q}$) describe the probability that a LinkedIn user (or group) corresponds to a Facebook user (or group). We note that these properties are not guaranteed when the correspondence matrix is required to be permutation or even doubly stochastic (square matrix with non-negative real entries, where each row and column sums to 1), which is common practice in the literature.
[A2] The matrices $\mathbf{P}$ and $\mathbf{Q}$ do *n*ot have to be square, which means that the matrices $\mathbf{A}$ and $\mathbf{B}$ can be of different size. This is yet another realistic requirement, as very rarely do two networks have the same number of nodes. Therefore, our formulation addresses not only graph alignment, but also *subgraph* alignment.

The *second constraint* follows naturally from the first one, as well as the large size of the social, and other networks. We want the correspondence matrices to be as sparse as possible, so that they encode few potential correspondences per node. Allowing every user/group of LinkedIn to be matched to every user/group of Facebook is not realistic and, actually, it is problematic for large graphs, as it has quadratic space cost w.r.t. the size of the input graphs.

To sum up, the existing approaches do not distinguish the nodes by types (e.g. users and groups), treat the graphs as unipartite, and, thus, aim at finding a permutation matrix $\mathbf{P}$, which gives a hard assignment between the nodes of the input graphs. In contrast, our formulation separates the nodes in categories, and can find correspondences at different granularities at once (e.g., individual and group-level correspondence in the case of the “user-group” graph).

III. BIG-ALIGN FOR BIPARTITE GRAPHS

Now that we have formulated the problem, we move on to the description of a technique to solve it. The design objective is two-fold. In terms of effectiveness, given the non-convexity of Problem 2, our goal is to find a ‘good’ local minimum. In terms of efficiency, we focus on carefully designing the search procedure.

Our method, BIG-ALIGN, comprises two main ideas:
[Idea 1] an alternating, projected gradient descent approach to find the local minima of the newly-defined optimization problem (Problem 2), and
[Idea 2] a series of optimizations: (a) a network-inspired initialization (NET-INIT) of the correspondence matrices to find a good starting point, (b) automatic choice of the steps for the gradient descent, and (c) handling the node-multiplicity problem, i.e. the “problem” of having nodes with exactly the same structure (e.g. peripheral nodes of a star) to improve both effectiveness and efficiency.

Next, we start by building the core of our method, continue with the description of the three optimizations, and conclude with the pseudocode of the overall algorithm.

A. Alternating Projected Gradient Descent (APGD): Mathematical formulation

Following the standard approach in the literature, in order to solve the optimization problem (Problem 2), we first relax the sparsity constraint, which is mathematically represented by the l_0-norm of the matrices' columns, and replace it with the l_1-norm, $\sum_i |\mathbf{P}_i^{(v)}| = \sum_i \mathbf{P}_i^{(v)}$, where we also use the probabilistic constraint. Therefore, the sparsity constraint now takes the form: $\sum_{i,j} P_{ij} \leq t$ and $\sum_{i,j} Q_{ij} \leq t$. By using this relaxation and applying Linear Algebra operations, the bipartite graph alignment problem takes the following form.

Theorem 1: **[Augmented Cost Function]** The optimization problem for the alignment of the bipartite graphs G_A and G_B, with adjacency matrices $\mathbf{A}$ and $\mathbf{B}$, under the probabilistic and sparsity constraints (Problem 2), is equivalent to:

$$\min_{\mathbf{P},\mathbf{Q}} f_{aug}(\mathbf{P},\mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}}\{||\mathbf{PAQ}-\mathbf{B}||_F^2 + \lambda\sum_{i,j} P_{ij} + \mu\sum_{i,j} Q_{ij}\}$$
$$= \min_{\mathbf{P},\mathbf{Q}}\{Tr(\mathbf{PAQ}(\mathbf{PAQ})^T - 2\mathbf{PAQB}^T) + \lambda\mathbf{1}^T\mathbf{P}\mathbf{1} + \mu\mathbf{1}^T\mathbf{Q}\mathbf{1}\}, \quad (1)$$

where $||\bullet||_F$ is the Frobenius norm of the enclosed matrix, $\mathbf{P}$ and $\mathbf{Q}$ are the user- and group-level correspondence matrices, and λ and μ are the sparsity penalties of $\mathbf{P}$ and $\mathbf{Q}$ respectively.

Proof: See Lemma 1 in Appendix A. ■

In summary, we solve the minimization problem by using a variant of the gradient descent algorithm. Given that the cost function in eq. (1) is bivariate, we use an alternating procedure to minimize it. We fix $\mathbf{Q}$ and minimize f_{aug} w.r.t. $\mathbf{P}$, and vice versa. If during the two alternating minimization steps, the entries of the correspondence matrices become invalid temporarily, we use a projection technique to guarantee the probabilistic constraint: If $P_{ij} < 0$ or $Q_{ij} < 0$, we *project* the entry to 0. If $P_{ij} > 1$ or $Q_{ij} > 1$, we *project* it to 1. The update steps of the alternating, projected gradient descent approach (APGD) are given by the following theorem.

Theorem 2: **[Update Step]** The update steps for the user- ($\mathbf{P}$) and group-level ($\mathbf{Q}$) correspondence matrices of APGD are given by:

$$\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_P \cdot \left(2(\mathbf{P}^{(k)}\mathbf{AQ}^{(k)} - \mathbf{B})\mathbf{Q}^{T^{(k)}}\mathbf{A}^T + \lambda\mathbf{1}\mathbf{1}^T\right)$$
$$\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_Q \cdot \left(2\mathbf{A}^T\mathbf{P}^{T^{(k+1)}}(\mathbf{P}^{(k+1)}\mathbf{AQ}^{(k)} - \mathbf{B}) + \mu\mathbf{1}\mathbf{1}^T\right),$$

where $\mathbf{P}^{(k)}$, $\mathbf{Q}^{(k)}$ are the correspondence matrices at iteration k, η_P and η_Q are the steps of the two phases of the APGD and $\mathbf{1}$ is the all-1 column-vector.

Proof: See Lemmas 2-3, and Obs. 3 in Appendix A. ■

We note that the assumption in the above formulas is that $\mathbf{A}$ and $\mathbf{B}$ are rectangular, adjacency matrices of bipartite graphs. It turns out that this formulation has a nice connection to the standard formulation for unipartite graph matching if we treat the input bipartite graphs as unipartite (i.e., symmetric, square, adjacency matrix). We summarize this equivalence in the following proposition.

Proposition 1: **[Equivalence to Unipartite Graph Alignment]** If the rectangular adjacency matrices of the bipartite graphs are converted to square matrices, then the minimization is done w.r.t. the coupled matrix $\mathbf{P}^*$:

$$\mathbf{P}^* = \begin{pmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} \end{pmatrix}.$$

That is, Problem 2 becomes $min_{\mathbf{P}*}||\mathbf{P}^*\mathbf{A}\mathbf{P}^{*T} - \mathbf{B}||_F^2$, which is equivalent to the unipartite graph problem introduced at the beginning of Section II.

B. Optimizations

Up to this point, we have the mathematical foundation at our disposal to build our algorithm, BIG-ALIGN. But first we have to make three design decisions:
(D1) How to initialize the correspondence matrices?
(D2) How to choose the steps for the APGD?
(D3) How to handle the structurally equivalent nodes?

The baseline approach, which we will refer to as BIG-ALIGN-BASIC, consists of the simplest answers to these questions: (D1) uniform initialization of the correspondence matrices, (D2) "small", constant step for the gradient descent, (D3) no specific manipulation of the structurally equivalent nodes. Next, we elaborate on sophisticated choices for the initialization and optimization step that render our algorithm more efficient. We also introduce the "node-multiplicity" problem, i.e., the problem of structurally equivalent nodes, and propose a way to deal with it.

(D1) How to initialize the correspondence matrices?

The optimization problem is non-convex (not even bi-convex), and the gradient descent gets stuck in local minima, depending heavily on the initialization. There are several different ways of initializing the correspondence matrices $\mathbf{P}$ and $\mathbf{Q}$, such as random, degree-based, eigenvalue-based [24] [10]. While each of these initializations has its own rationality, they are designed for unipartite graphs and hence ignore the skewness of the real, large-scale bipartite graphs. To address this issue, we propose a network-inspired approach (NET-INIT), which is based on the following observation about large-scale, real biparite graphs:

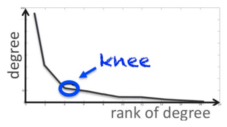
Observation 1: Large, real networks have skewed or power-law-like degree distribution ([1], [8], [11]). Specifically in bipartite graphs, usually one of the node sets is significantly smaller than the other, and has skewed degree distribution.

The implicit assumption[2] of NET-INIT is that a person is almost equally popular in different social networks, or, more generally, an entity has similar "behavior" across the input graphs. In our work, we found that such behavior can be well captured by the node degree. However, the technique we describe below can be naturally applied to other features (e.g., weight, ranking, clustering coefficient) that may capture better the node behavior.

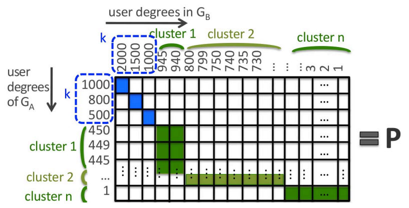
Our initialization approach consists of four steps. For the description of the approach, we refer to the example of LinkedIn and Facebook bipartite graphs, where the first set consists of users, and the second set of groups. Assume that the set of groups is significantly smaller than the set of users. The steps, which are pictorially shown in Fig. 1(b), are:

Step 1. Match 1-by-1 the top-k high-degree groups in the LinkedIn and Facebook graphs. To find k, we borrow the idea of scree plot, which is used in Principal Component Analysis (PCA): we sort the unique degrees of each graph in descending order, and create the plot of unique degree vs. rank of node (Fig. 1(a)). In this plot, we detect the "knee" and up to the

[2]If the assumption does not hold, no method is guaranteed to find the alignment based purely on the structure of the graphs, but they can still reveal similarities between nodes.



(a) "Scree-like" plot for NET-INIT.



(b) Pictorial initialization of $\mathbf{P}$.

Fig. 1: (a) Choise of k in Step 1 of NET-INIT. (b) Initialization of the node/user-level correspondence matrix by NET-INIT.

corresponding degree we "safely" match the groups of the two graphs one-by-one, i.e. the most popular group of LinkedIn is aligned initially with the most popular group of Facebook etc. For the automatic detection of the knee, we consider the plot piecewise, and assume that the knee occurs when the slope of a line segment is less than 5% of the slope of the previous segment.

Step 2. For each of the matched groups, **align their neighbors** based on their Relative Degree Difference (RDD):

*Definition 1 (*RDD*):* The Relative Degree Distance function that aligns node i of graph $\mathbf{A}$ to node j of $\mathbf{B}$ is:

$$\mathrm{rdd}(i,j) = \left(1 + \frac{|\deg(i) - \deg(j)|}{(\deg(i) + \deg(j))/2}\right)^{-1} \tag{2}$$

where $\deg(\bullet)$ is the degree of the corresponding node.

The idea behind this approach is that a node in one graph corresponds most probably to a node with similar degree in another graph, than to a node with very different degree. The above function assigns higher probabilities to alignments of similar nodes, and lower probabilities to alignments of very dissimilar nodes w.r.t. their degrees.

We note that the RDD function, $\mathrm{rdd}(i,j)$, corresponds to the degree-based similarity between node i and node j. However, it can be generalized to other properties (than the degree) that the nodes are expected to share across different graphs. Equation (2) captures one additional desired property: it penalizes the alignments based on the relative difference of the degrees. For example, two nodes of degrees 1 and 20 respectively are less similar than two nodes with degrees 1001 and 1020.

Step 3. Create c_g **clusters of the remaining groups** in both networks, based on their degrees. **Align the clusters 1-by-1** according to the degrees (e.g., "high", "low"), and initialize the correspondences *within* the matched clusters using the RDD.

Step 4. Create c_u **clusters of the remaining users** in both networks, based on their degrees. Align the users using the RDD approach within the corresponding user clusters.

(D2) How to choose the steps for the APGD?

One of the most important parameters that come up in the APGD method is η (the step of approaching the minimum point), which determines its convergence rate. In an attempt to automatically determine the step, we use the *line search* approach [6], which is described in Algorithm 2. Line search is a strategy that finds the local optimum for the step. Specifically, in the first phase of APGD, line search determines η_P by treating the objective function, f_{aug}, as a function of η_P (instead of a function of $\mathbf{P}$ or $\mathbf{Q}$) and loosely minimizing it. In the second phase of APGD, η_Q is determined similarly. Next we introduce 3 variants of our method that differ in the way the steps are computed.

Variant 1: BIG-ALIGN-Points. Our first approach consists of approximately minimizing the augmented cost function: we randomly pick some values for η_P within some "reasonable" range, and compute the value of the cost function. We choose the step η_P that corresponds to the minimum value of the cost function. Similarly we define η_Q. This approach is computationally expensive, as we shall see in Sec. V.

Variant 2: BIG-ALIGN-Exact. By carefully handling the objective function of our optimization problem, we can find closed (exact) forms for η_P and η_Q, which are given in the next theorem.

Theorem 3: **[Optimal Step Size for P]** In the first phase of APGD, the value of the step η_P that exactly minimizes the augmented function, $f_{aug}(\eta_P)$, is given by:

$$\eta_P = \frac{2\,\mathrm{Tr}\,\{(\mathbf{P}^{(k)}\mathbf{AQ})(\Delta_{\mathbf{P}}\mathbf{AQ})^T - (\Delta_{\mathbf{P}}\mathbf{AQ})B^T\} + \lambda \sum_{i,j} \Delta_{Pij}}{2||\Delta_P \mathbf{AQ}||_F^2}, \tag{3}$$

where $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_P \Delta_{\mathbf{P}}$, $\Delta_{\mathbf{P}} = \nabla_{\mathbf{P}} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}}$ and $\mathbf{Q} = \mathbf{Q}^{(k)}$.

Proof: See Appendix B. ■

Similarly, we find the appropriate value for the step η_Q of the second phase of APGD.

Theorem 4: **[Optimal Step Size for Q]** In the second phase of APGD, the value of the step η_Q that exactly minimizes the augmented function, $f_{aug}(\eta_Q)$, is given by:

$$\eta_Q = \frac{2\,\mathrm{Tr}\,\{(\mathbf{PAQ}^{(k)})(\mathbf{PA}\Delta_Q)^T - (\mathbf{PA}\Delta_{\mathbf{Q}})B^T\} + \mu \sum_{i,j} \Delta_{Qij}}{2||\mathbf{PA}\Delta_Q||_F^2}, \tag{4}$$

where $\Delta_{\mathbf{Q}} = \nabla_{\mathbf{Q}} f_{aug}|_{\mathbf{Q}=\mathbf{Q}^{(k)}}$, $\mathbf{P} = \mathbf{P}^{(k)}$, and $\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_Q \Delta_{\mathbf{Q}}$.

Proof: Omitted for brevity. ■

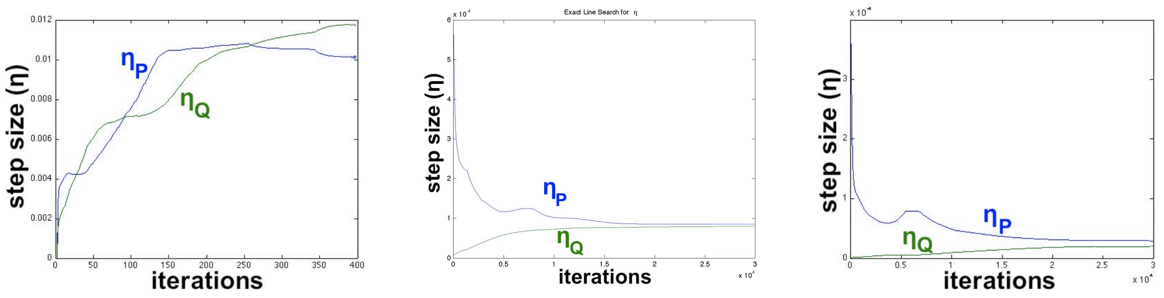
BIG-ALIGN-Exact is significantly faster than BIG-ALIGN-Points. It turns out that we can increase the efficiency even more, as experimentation with real data revealed that the values of the gradient descent steps that minimize the objective function do not change drastically in every iteration (Fig. 2). This led to the third variation of our algorithm:

Variant 3: BIG-ALIGN-Skip. This variation does exact line search for the first few (e.g., 100) iterations, and then updates the values of the steps every few (e.g., 500) iterations. This significantly reduces the computations for determining the optimal step sizes.

(D3) How to handle the structurally equivalent nodes?

One last observation that renders BIG-ALIGN more efficient is the following:

Observation 2: In the majority of graphs, there is a significant number of nodes that cannot be distinguished, because they have exactly the same structural features.



(a) Bipartite graph with 50 nodes. (b) Bipartite graph with 300 nodes. (c) Bipartite graph with 900 nodes.

Fig. 2: (Hint for speedup.) Size of optimal step for $\mathbf{P}$ (blue) and $\mathbf{Q}$ (green) vs. the number of iterations. We observe that the optimal step sizes do not change dramatically in consecutive iterations, and, thus, skipping some computations almost does not affect the accuracy at all.

For instance, in many real-world networks, a commonplace structure is stars [14], but it is impossible to tell the peripheral nodes apart. Other examples of non-distinguishable nodes include the members of cliques, full bipartite cores etc.

To address this problem, we introduce a pre-processing phase at which we eliminate nodes with identical structures by aggregating them in super-nodes. For example, a star with 100 peripheral nodes which are connected to the center by edges of weight 1, will be replaced by a super-node connected to the central node of the star by an edge of weight 100. This subtle step not only leads to a better optimization solution, but also improves the efficiency by reducing the scale of graphs that are actually fed into our BIG-ALIGN.

C. BIG-ALIGN: *Putting everything together*

The previous subsections shape up the proposed algorithm, BIG-ALIGN, the pseudocode of which is given in Algorithms 1 and 2.

In our implementation, the only parameter that the user is required to input is the sparsity penalty, λ. The bigger this parameter is, the more entries of the matrices are forced to be 0. We set the other sparsity penalty $\mu = \frac{\lambda * (\text{elements in } \mathbf{Q})}{\text{elements in } \mathbf{P}}$, so that the penalty per non-zero element of $\mathbf{P}$ and $\mathbf{Q}$ is the same.

It is worth mentioning that, in contrast to the approaches found in the literature, our method does not use the classic Hungarian algorithm to find the hard correspondences between the nodes of the bipartite graphs. Instead, we rely on a fast approximation: we align each row i (node/user) of $\mathbf{P}^T$ with the column j (node/user) that has the maximum probability. It is clear that this assignment is very fast, and even parallelizable, as each node alignment can be processed independently. Moreover, it allows aligning multiple nodes of one graph with the same node of the other graph, a property that is desirable especially in the case of structurally equivalent nodes.

Figure 3 depicts how the cost and accuracy of the alignment change with respect to the number of iterations of the gradient descent algorithm.

Algorithm 1 BIG-ALIGN-Exact: Bipartite Graph Alignment

```
INPUT: $\mathbf{A}$, $\mathbf{B}$, $\lambda$, MAXITER
$\epsilon = 10^{-6}$; $cost(0) = 0$; $k = 1$;
/* STEP 1: pre-processing for node-multiplicity */
aggregating identical nodes
/* STEP 2: initialization */
[$\mathbf{P}0$, $\mathbf{Q}0$] = NET-INIT-ialization
$cost(1) = f_{aug}(\mathbf{P}0, \mathbf{Q}0)$
/* STEP 3: alternating projected gradient descent (APGD) */
while $|cost(k-1) - cost(k)|/cost(k-1) > \epsilon$ AND $k <$ MAXITER do
    $k++$
    /* PHASE 1: fixed $\mathbf{Q}$, minimization w.r.t. $\mathbf{P}$ */
    $\eta_{Pk} = $ LINESEARCH_P$(\mathbf{P}^{(k)}, \mathbf{Q}^{(k)}, \nabla_{\mathbf{P}} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}})$
    $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_{Pk} \nabla_{\mathbf{P}} f_{aug}(\mathbf{P}^{(k)}, \mathbf{Q}^{(k)})$
    VALIDPROJECTION$(\mathbf{P}^{(k+1)})$
    /* PHASE 2: fixed $\mathbf{P}$, minimization w.r.t. $\mathbf{Q}$ */
    $\eta_{Qk} = $ LINESEARCH_Q$(\mathbf{P}^{(k+1)}, \mathbf{Q}^{(k)}, \nabla_{\mathbf{Q}} f_{aug}|_{\mathbf{Q}=\mathbf{Q}^{(k)}})$
    $\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_{Qk} \nabla_{\mathbf{Q}} f_{aug}(\mathbf{P}^{(k+1)}, \mathbf{Q}^{(k)})$
    VALIDPROJECTION$(\mathbf{Q}^{(k+1)})$
    $cost(k) = f_{aug}(\mathbf{P}, \mathbf{Q})$
end while
return $\mathbf{P}^{(k+1)}$, $\mathbf{Q}^{(k+1)}$

/* PROJECTION STEP */
function VALIDPROJECTION$(\mathbf{P})$
  for all $i, j$
    if $\mathbf{P}_{ij} < 0$ then $\mathbf{P}_{ij} = 0$
    else if $\mathbf{P}_{ij} > 1$ then $\mathbf{P}_{ij} = 1$
end function
```

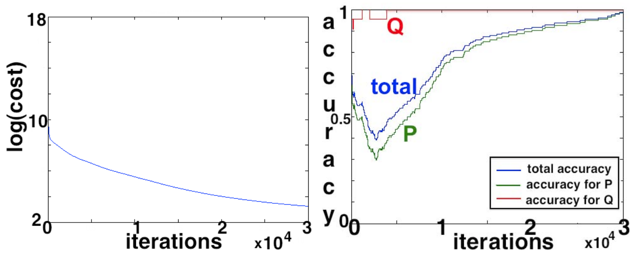
IV. UNI-ALIGN: EXTENSION TO UNIPARTITE GRAPHS

Although our primary target for BIG-ALIGN is bipartite graphs – which by themselves already stand for a significant portion of real graphs –, as a side-product, BIG-ALIGN also offers an alternative, fast solution to the alignment problem of unipartite graphs. Our approach consists of two steps:

Step 1: Uni- to Bi-partite Graph Conversion. The first step involves converting the $n \times n$ unipartite graphs to bipartite graphs. Specifically, we can first extract d node features, such as degree, edges in a node's egonet (= induced subgraph of the node and its neighbors), and clustering coefficient. Then, we can form the $n \times d$ bipartite graph node-to-feature, where $n \gg d$. The runtime of this step depends on the time complexity of extracting the selected features.

Algorithm 2 Line Search for η_P and η_Q

```
function LINESEARCH_P($\mathbf{P}, \mathbf{Q}, \Delta_\mathbf{P}$)
    return
    $\eta_P = \frac{2\,\mathrm{Tr}\,\{(\mathbf{P}^{(k)}\mathbf{AQ})(\Delta_\mathbf{P}\mathbf{AQ})^T - (\Delta_\mathbf{P}\mathbf{AQ})B^T\} + \lambda\sum_{i,j}\Delta_{Pij}}{2||\Delta_P\mathbf{AQ}||_F^2}$
end function
function LINESEARCH_Q($\mathbf{P}, \mathbf{Q}, \Delta_\mathbf{Q}$)
    return
    $\eta_Q = \frac{2\,\mathrm{Tr}\,\{(\mathbf{PAQ}^{(k)})(\mathbf{PA}\Delta_Q)^T - (\mathbf{PA}\Delta_\mathbf{Q})B^T\} + \mu\sum_{i,j}\Delta_{Qij}}{2||\mathbf{PA}\Delta_Q||_F^2}$
end function
```



(a) Cost function. (b) Accuracy.

Fig. 3: BIG-ALIGN (900 nodes, $\lambda = 0.1$): As desired, the cost of the objective function drops with the number of iterations, and at the same time the accuracy both on node- (green) and community-level (red) increases. The blue line corresponds to the total accuracy, i.e., the accuracy of all the alignments independently of the node type (user or group). The exact definition of accuracy is given in Sec. V-B.

Step 2: Finding P. We note that in this case, the alignment of the second sets of the bipartite graphs is known, i.e., $\mathbf{Q}$ is an identity matrix, since we extract the same type of features from the graphs. Thus, we only need to align the nodes that belong to the first sets of the graphs, i.e., compute $\mathbf{P}$. We revisit Eq. (1) of our initial minimization problem, and now we want to minimize it only w.r.t. $\mathbf{P}$. By setting the derivative of f_{aug} w.r.t. $\mathbf{P}$ equal to 0, we have:

$$\mathbf{P}\cdot(\mathbf{AA^T}) = \mathbf{BA^T} - \lambda/2\cdot\mathbf{11^T},$$

where $\mathbf{A}$ is a $n \times d$ matrix. If we do SVD (Singular Value Decomposition) on this matrix, i.e., $\mathbf{A} = \mathbf{USV}$, the Moore-Penrose pseudo-inverse of $\mathbf{AA^T}$ is $(\mathbf{AA^T})^\dagger = \mathbf{US^{-2}U^T}$. Therefore, we have

$$\begin{aligned}\mathbf{P} &= (\mathbf{BA^T} - \lambda/2\mathbf{11^T})(\mathbf{AA^T})^\dagger\\ &= (\mathbf{BA^T} - \lambda/2\mathbf{11^T})(\mathbf{US^{-2}U^T})\\ &= \mathbf{B}\cdot(\mathbf{A^TUS^{-2}U^T}) - \mathbf{1}\cdot(\lambda/2\cdot\mathbf{1^TUS^{-2}U^T})\\ &= \mathbf{B}\cdot\mathbf{X} - \mathbf{1}\cdot\mathbf{Y} \qquad (5)\end{aligned}$$

where $\mathbf{X} = \mathbf{A^TUS^{-2}U^T}$ and $\mathbf{Y} = \lambda/2\cdot\mathbf{1^TUS^{-2}U^T}$.

Hence, we can exactly (*non*-iteratively) find $\mathbf{P}$ from Eq. (5). It can be shown that the time complexity for finding $\mathbf{P}$ is $\mathbf{O(nd^2)}$ (after omitting the simpler terms), which is linear on the number of nodes of the input graphs.

What is more, we can see from Eq. (5) that $\mathbf{P}$ itself has the low-rank structure. In other words, we do not need to store $\mathbf{P}$ in the form of $n \times n$. Instead, we can represent (compress) $\mathbf{P}$ as the multiplication of two low-rank matrices $\mathbf{X}$ and $\mathbf{Y}$, whose additional space cost is just $O(nd + n) = O(nd)$.

TABLE II: Graph Alignment Algorithms: name conventions, short description, type of graphs for which they were designed ('uni-' for unipartite, 'bi-' for bipartite graphs), and reference.

Name	Description	Graph	Source
`Umeyama`	eigenvalue-based	uni-	[24]
`NMF-based`	NMF-based	uni-	[10]
`NetAlign-full`	BP-based with uniform init.	uni-	Modified
`NetAlign-deg`	BP-based with same-degree init.	uni-	from [3]
BIG-ALIGN-Basic	APGD (no optimizations)	**bi-**	current
BIG-ALIGN-Points	APGD + approx. Line Search	**bi-**	current
BIG-ALIGN-Exact	APGD + exact Line Search	**bi-**	current
BIG-ALIGN-Skip	APGD + skip some Line Search	**bi-**	current
UNI-ALIGN	BIG-ALIGN-inspired (SVD)	uni-	current

V. EXPERIMENTAL EVALUATION

In this section, we evaluate the proposed algorithms, BIG-ALIGN and UNI-ALIGN, w.r.t. alignment accuracy and runtime, and also compare them to the state-of-the-art methods. The code for all the methods is written in Matlab and the experiments were run on Intel(R) Xeon(R) CPU 5160 @ 3.00GHz, with 16GB RAM memory.

A. Baseline Methods

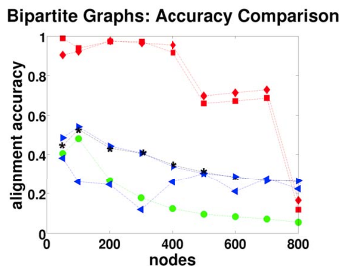
To the best of our knowledge, no graph matching algorithm has been designed for bipartite graphs. Throughout this section, we compare our algorithms to 3 state-of-the-art approaches, which are succinctly described in Table II: (i) `Umeyama`, the influential eigenvalue decomposition-based approach proposed by Umeyama [24]; (ii) `NMF-based`, a recent approach based on Non-negative Matrix Factorization [10]; and (iii) `NetAlign-full` and `NetAlign-deg`, two variations of a fast, and scalable Belief Propagation-based (BP) approach [3]. Some details about these approaches are provided in the Related Work (Section VI).

In order to use the state-of-the-art approaches for aligning bipartite graphs, we convert the latter to unipartite by using Proposition 1. In addition to that, the BP-based approach requires not only the two input graphs, but also a bipartite graph that encodes the possible matchings per node. To render the method applicable in our setting, we use two heuristics for forming the required bipartite 'matching' graph: (a) `full` bipartite graph, which essentially conveys that we have no domain information about the possible alignments, and each node of the first graph can be aligned with *any* node of the second graph (`NetAlign-full`); and (b) `degree`-based bipartite graph, where only nodes with the same degree in both graphs are considered possible matchings (`NetAlign-deg`).

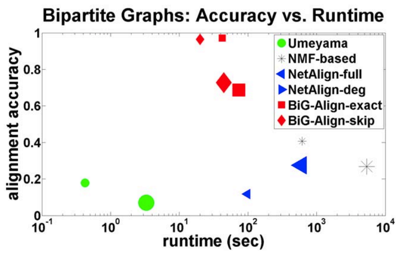
B. Evaluation of BIG-ALIGN

For the experiments on bipartite graphs, we use the movie-genre graph of the MovieLens network[3]. Each of the 1,027 movies is linked to at least one of the 23 genres (e.g., comedy, romance, drama). Specifically, from this network, we extract

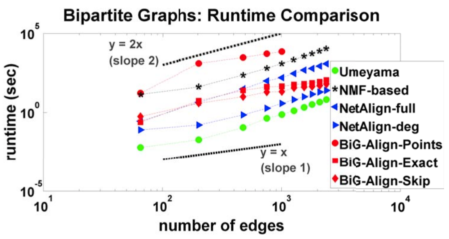
[3] `http://www.movielens.org`



(a) (Higher is better.) Accuracy of alignment vs. number of nodes.

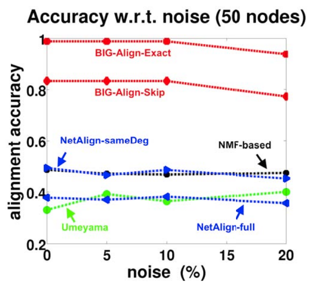


(b) (Higher and left is better.) Accuracy of alignment vs. runtime (in seconds / logscale) for graphs with 300 nodes (small markers), and 700 nodes (big markers).

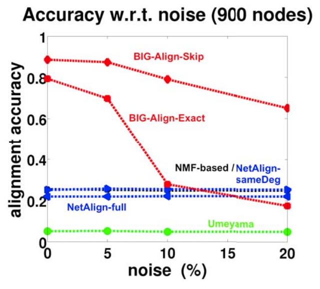


(c) (Lower is better.) Runtime in seconds vs. the number of edges in the graphs in log-log scale.

Fig. 5: Accuracy and runtime of alignment of bipartite graphs. (a) BIG-ALIGN-Exact and BIG-ALIGN-Skip (red lines) significantly outperform, in terms of accuracy, all the alignment methods for almost all the graph sizes; (b) BIG-ALIGN-Exact and BIG-ALIGN-Skip (red squares/ovals) are more accurate and, at the same time, faster than the baselines for both graph sizes. (c) The BIG-ALIGN variants are faster than all the baseline approaches, except for `Umeyama`'s algorithm.



(a) Graphs of 50 nodes.



(b) Graphs of 900 nodes.

Fig. 4: (Higher is better.) Accuracy of bipartite graph alignment vs. level of noise (0-20%). BIG-ALIGN-Exact (red line with square marker), almost always, outperforms the baseline methods.

subgraphs of different sizes. Then, following the tradition in the literature [10], for each of the subgraphs we generate permutations, $\mathbf{B}$, with $noise$ from 0% to 20% using the formula $\mathbf{B}_{ij} = (\mathbf{PAQ})_{ij} \cdot (1 + noise * r_{ij})$, where r_{ij} is a random number in $[0, 1]$. For each noise level and graph size, we generate 10 distinct permutations of the initial subnetwork. We run the alignment algorithms on all the pairs of the original and permuted subgraphs, and report the mean accuracy and runtime. For all the variants of BIG-ALIGN, we set the sparsity penalty $\lambda = 0.1$.

How do we compute the accuracy of the methods? For the state-of-the-art methods, which find "hard" alignments between the nodes, the accuracy is computed as usual: only if the true correspondence is found, the corresponding matching is deemed correct. In other words, we use the state-of-the-art algorithms off-the-shelf. For our method, BIG-ALIGN, which has the advantage of finding "soft", probabilistic alignments, we consider two cases for evaluating its accuracy: (i) *Correct Alignment.* If the true correspondence coincides with the most probable matching, we count the node alignment as correct; (ii) *Partially Correct Alignment.* If the true correspondence is *among* the most probable matchings (tie), the alignment thereof is deemed partially correct and weighted by (# of nodes in tie)/ (total # of nodes).

Accuracy. Figures 4 (a) and (b) present the accuracy of the methods for two different graph sizes and *varying level of noise* in the permutations. We observe that BIG-ALIGN outperforms all the other methods in most cases with a large margin. In Fig. 4(b), the only exception is the case of 20% of noise in the 900-nodes graphs where `NetAlign-deg` and `NetAlign-full` perform slightly better than our algorithm, BIG-ALIGN-Exact. The results for other graph sizes are along the same lines, and therefore are omitted for space.

Figure 5(a) depicts the accuracy of the alignment methods *for varying graph size*. For graphs with different sizes, the variants of our method achieve significantly higher accuracy (70%-98%) than the baselines (10%-58%). Moreover, surprisingly, BIG-ALIGN-Skip performs slightly better than BIG-ALIGN-Exact, although the former skips several updates of the gradient descent steps. The only exception is for the smallest graph size, where the consecutive optimal steps change significantly (Fig. 2(a)), and, thus, skipping computations affects the performance. `NetAlign-full` and `Umeyama`'s algorithm are the least accurate methods, while `NMF-based` and `NetAlign-deg` achieve medium accuracy. Finally, the accuracy vs. runtime plot in Fig. 5(b) shows that our algorithms have two desired properties: they achieve *better* performance, *faster* than the baseline approaches.

Runtime. Figure 5(c) presents the runtime as a function of the number of edges in the graphs. `Umeyama`'s algorithm and `NetAlign-deg` are the fastest methods, but at the cost of accuracy; BIG-ALIGN is upto 10× more accurate in the cases that it performs slower. The third best method is BIG-ALIGN-Skip, closely followed by BIG-ALIGN-Exact. BIG-ALIGN-Skip is upto 174× faster than the `NMF-based` approach, and upto 19× faster than `NetAlign-full`. However, our simplest algorithm that uses line search, BIG-ALIGN-Points, is the slowest approach that takes considerable amount of time for graphs with more than 1.5K edges (and, thus, we omit several data points in the plot).

It is worth mentioning that currently BIG-ALIGN is a single machine implementation, but it has the potential for further speed-up. For example, it could be parallelized by splitting the optimization problem to smaller subproblems (by decomposing the matrices, and doing simple column-

TABLE III: Runtime (top) and accuracy (bottom) comparison of the BIG-ALIGN variants: BIG-ALIGN-Basic, BIG-ALIGN-Points, BIG-ALIGN-Exact, and BIG-ALIGN-Skip. BIG-ALIGN-Skip is not only faster, but also comparably or more accurate than BIG-ALIGN-Exact.

	BIG-ALIGN-Basic		BIG-ALIGN-Points		BIG-ALIGN-Exact		BIG-ALIGN-Skip	
Nodes	mean	std	mean	std	mean	std	mean	std
RUNTIME (SEC)								
50	0.07	0.00	17.3	0.05	0.24	0.08	0.56	0.01
100	0.023	0.00	1245.7	394.55	5.6	2.93	3.9	0.05
200	31.01	16.58	2982.1	224.81	25.5	0.39	10.1	0.10
300	0.032	0.00	5240.9	30.89	42.1	1.61	20.1	1.62
400	0.027	0.01	7034.5	167.08	45.8	2.058	21.3	0.83
500	0.023	0.01	-	-	57.2	2.22	36.6	0.60
600	0.028	0.01	-	-	64.5	2.67	40.8	1.26
700	0.029	0.01	-	-	73.6	2.78	44.6	1.23
800	166.7	1.94	-	-	86.9	3.63	49.9	1.06
900	211.9	5.30	-	-	111.9	2.96	61.8	1.28
ACCURACY								
50	0.071	0.00	0.982	0.02	0.988	0	0.904	0.03
100	0.034	0.00	0.922	0.07	0.939	0.06	0.922	0.07
200	0.722	0.37	0.794	0.01	0.973	0.01	0.975	0.00
300	0.014	0.00	0.839	0.02	0.972	0.01	0.964	0.01
400	0.011	0.00	0.662	0.02	0.916	0.03	0.954	0.01
500	0.011	0.00	-	-	0.66	0.20	0.697	0.24
600	0.005	0.00	-	-	0.67	0.20	0.713	0.23
700	0.004	0.00	-	-	0.69	0.20	0.728	0.19
800	0.013	0.00	-	-	0.12	0.02	0.165	0.03
900	0.015	0.00	-	-	0.17	0.20	0.195	0.22

row multiplications). Moreover, instead of the basic gradient descent algorithm, we can use a variant method, the stochastic gradient descent, which is based on sampling.

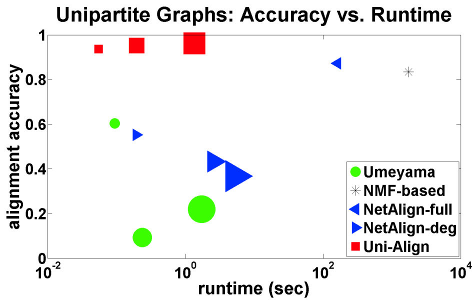
Variants of BIG-ALIGN. Before we continue with the evaluation of UNI-ALIGN, we present in Table III the runtime and accuracy of all the variants of BIG-ALIGN for aligning movie-genre graphs with varying sizes and permutations with noise level 10%. The parameters used in this experiment are $\epsilon = 10^{-5}$, and $\lambda = 0.1$. For BIG-ALIGN-Basic, η is constant and equal to 10^{-4}, while the correspondence matrices are initialized uniformly. This is not the best setting for all the pairs of graphs that we are aligning, and it results in very low accuracy. On the other hand, BIG-ALIGN-Skip is not only $\sim 350\times$ faster than BIG-ALIGN-Points, but also more accurate. Moreover, it is $\sim 2\times$ faster than BIG-ALIGN-Exact with higher or equal accuracy. The speedup can be further increased by skipping more updates of the gradient descent steps. Overall, the results show that a naive solution of the optimization problem, such as BIG-ALIGN-Basic, is not sufficient, and the optimizations we propose in Section III are crucial and render our algorithm efficient.

C. *Evaluation of* UNI-ALIGN

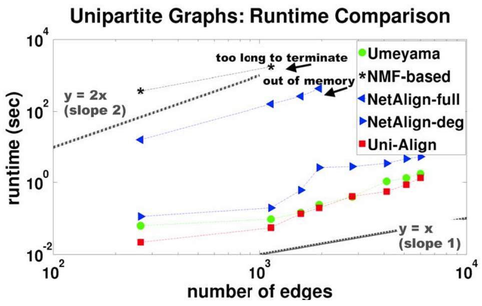
To evaluate our proposed method, UNI-ALIGN, for aligning unipartite graphs, we use the Facebook who-links-to-whom graph [25], which consists of approximately $64K$ nodes. In this case, the baseline approaches are readily employed, while our method requires the conversion of the given unipartite graph to bipartite. We do so by extracting some unweighted egonet[4] features for each node (degree of node, degree of egonet[5], edges of egonet, mean degree of the node's neighbors). As before, from the initial graph we extract subgraphs of size 100-800 nodes (or equivalently, 264-6K edges), and create 10 noisy permutations (per noise level) as before.

[4]As a reminder, egonet of a node is the induced subgraph of its neighbors.

[5]The degree of an egonet is defined as the number of incoming and outgoing edges of the subgraph, when viewed as a super-node.



(a) (Higher and left is better.) Accuracy of alignment vs. runtime (in seconds / logscale) for facebook frienship subgraphs of size 200 (small markers), 400 (medium markers), and 800 (big markers).



(b) (Lower is better.) Runtime (in seconds) vs. number of edges in log-log scale.

Fig. 6: Accuracy and runtime of alignment of unipartite graphs. (a) UNI-ALIGN (red points) is more accurate and faster than all the baselines for all graph sizes. (c) UNI-ALIGN (red squares) is faster than all the baseline approaches, followed closely by `Umeyama`'s approach (green circles).

Accuracy. The accuracy vs. runtime plot in Fig. 6(a) shows that UNI-ALIGN outperforms all the other methods in terms of accuracy and runtime for all the graph sizes depicted. Although NMF achieves a reasonably good accuracy for the graph of 200 nodes, it takes too long to terminate; we stopped the runs for graphs of bigger sizes as the execution was taking too long. The remaining approaches are fast enough, but yield poor accuracy.

Runtime. Figure 6(b) compares the graph alignment algorithms w.r.t. their running time (in logscale). UNI-ALIGN is the fastest approach, closely followed by Umeyama's algorithm. `NetAlign-deg` is some orders of magnitude slower than the previously mentioned methods. However, `NetAlign-full` ran out of memory for graphs with more than 2.8K edges; we stopped the runs of the `NMF-based` approach, as it was taking too long to terminate even for small graphs with 300 nodes and 1.5K edges. The results are similar for other graph sizes that, for simplicity, are not shown in the figure. For graphs with 200 nodes and $\sim 1.1K$ edges (which is the biggest graph for which all the methods were able to terminate), UNI-ALIGN is $1.75\times$ faster than `Umeyama`'s approach; $2\times$ faster than `NetAlign-deg`; $2,927\times$ faster than `NetAlign-full`; and $31,709\times$ faster than the `NMF-based` approach.

VI. Related Work

The graph alignment problem is of such great interest that there are more than 150 publications proposing different solutions for it, and spanning numerous research fields: from data mining to security and re-identification [13] [18], bioinformatics [4] [15] [22] , databases [17], chemistry [23], vision, and pattern recognition [9]. Among the suggested approaches are genetic, spectral, clustering algorithms [20], decision trees, expecation-maximization [16], graph edit distance [21], simplex [2], non-linear optimization [12], iterative HITS-inspired [5][28]. Notice that all these works are designed for unipartite graphs, while we focus on bipartite graphs.

One of the well-known approaches is Umeyama's near-optimum solution for nearly-isomorphic graphs [24]. The method solves the optimization problem $min_{\mathbf{P}}||\mathbf{PAP}^T - \mathbf{B}||$ (where $\mathbf{P}$ is permutation matrix) based on the eigendecomposition of the matrices, and operates on unipartite, weighted graphs with the *same* number of nodes. The Hungarian algorithm [19] is employed at the end to find the node correspondences. The constraint that $\mathbf{P}$ is doubly stochastic matrix is imposed in [26] and [29], where the proposed formulation, PATH, is based on convex and concave relaxations. Ding et al [10] recently proposed a Non-Negative Matrix Factorization (NMF) approach, which starts from Umeyama's solution, and then applies an iterative algorithm to find the orthogonal matrix $\mathbf{P}$ with the node correspondences.

Bradde et al. [7] propose distributed, heuristic, message-passing algorithms, based on Belief Propagation [27], for protein alignment and prediction of interacting proteins. Independently, Bayati et al [3] formulate graph matching as an integer quadratic problem, and also propose message passing algorithms for aligning sparse networks. A sparse and weighted bipartite graph, whose edges represent the possible node matchings between the two graphs is required by these algorihms. The use of the full bipartite graph was proposed earlier by Singh et al. [22].

In all these works, the graphs that are studied are unipartite, while we focus on bipartite graphs, and also propose an extension of our method to handle unipartite graphs.

VII. Discussion

The experiments show that BiG-Align solves efficiently a problem that has been neglected in the literature: the alignment of bipartite graphs.

Given that all the efforts have been targeted at aligning unipartite graphs, why matching bipartite graphs deserves being studied separately? Firstly, bipartite networks are omnipresent: people like webpages, belong to online communities, access shared files in companies, post in blogs, co-author papers, attend conferences etc. All these settings can be modeled as bipartite graphs. Secondly, although it is possible to turn them to unipartite and apply an off-the-shelf algorithm, as shown in the experiments, knowledge of the specific structural characteristics can prove useful in achieving better quality alignments. Lastly, this problem enables emerging applications. For instance, one may be able to link the clustering results from different networks by applying soft clustering on the input graphs, and subsequently our method on the obtained node-cluster membership graphs.

Although the main focus of our paper is bipartite graph alignment, the latter inspires an alternative way of matching unipartite graphs, by turning them to bipartite. Therefore, we show how our framework can handle any type of input graphs, without any restrictions on its structure.

Finally, is our approach simply gradient descent? The answer is negative; gradient descend is the *core* of our algorithm, *but* the projection technique, appropriate initialization and choice of the gradient step, as well as careful handling of known graph properties are the critical design choices that make our algorithm successful (as shown in Sec. V, where we compare our method to simple gradient descend, BiG-Align-Basic).

VIII. Conclusion

In this paper, we study the problem of graph matching for an important class of real graphs, bipartite graphs. Our contributions can be summarized as follows:

1) *Formulations.* We introduce a powerful primitive with new constraints for the graph matching problem.
2) *Algorithms.* We propose an effective and efficient algorithm, BiG-Align, based on gradient descent (APGD) to solve our constrained optimization problem with careful handling of many subtleties. We also give a generalization of our approach to align unipartite graphs (Uni-Align).
3) *Evaluations.* Our extensive experiments show that BiG-Align and Uni-Align are superior to state-of-the-art graph matching algorithms in terms of both accuracy and efficiency, for bipartite as well as unipartite graphs.

Future work includes extending our problem formulation to subgraph matching by revisiting the initialization of the correspondence matrices.

Acknowledgements

This material is partially supported by the National Science Foundation under Grant No. IIS1017415 and IIS-1217559, the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053, and Defense Advanced Research Projects Agency (DARPA) under Contract Number W911NF-11-C-0200 and W911NF-12-C-0028. The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *CoRR*, cond-mat/0106096, 2001.

[2] H. A. Almohamad and S. O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE TPAMI*, 15(5):522–525, 1993.

[3] M. Bayati, M. Gerritsen, D. Gleich, A. Saberi, and Y. Wang. Algorithms for large, sparse network alignment problems. In *ICDM09*, pages 705–710, 2009.

[4] J. Berg and M. Lässig. Local graph alignment and motif search in biological networks. *PNAS*, 101(41):14689–14694, Oct. 2004.

[5] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. V. Dooren. A measure of similarity between graph vertices. *CoRR*, 2004.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[7] S. Bradde, A. Braunstein, H. Mahmoudi, F. Tria, M. Weigt, and R. Zecchina. Aligning graphs and finding substructures by a cavity approach. *Europhysics Letters*, 89, 2010.

[8] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Comput. Netw.*, 33(1-6):309–320, June 2000.

[9] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.

[10] C. H. Q. Ding, T. Li, and M. I. Jordan. Nonnegative matrix factorization for combinatorial optimization: Spectral clustering, graph matching, and clique finding. In *ICDM*, pages 183–192, 2008.

[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262, Aug. 1999.

[12] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE TPAMI*, 18(4):377–388, 1996.

[13] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos. It's who you know: graph mining using recursive structural features. In *KDD*, KDD, pages 663–671, 2011.

[14] U. Kang and C. Faloutsos. Beyond 'caveman communities': Hubs and spokes for graph compression and mining. In *ICDM*, pages 300–309, 2011.

[15] G. W. Klau. A new graph-based method for pairwise global network alignment. *BMC*, 10(S-1), 2009.

[16] B. Luo and E. R. Hancock. Iterative procrustes alignment with the em algorithm. *Image Vision Comput.*, 20(5-6):377–396, 2002.

[17] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, 2002.

[18] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *SSP*, pages 173 –187, may 2009.

[19] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[20] H. Qiu and E. R. Hancock. Graph matching and clustering using spectral partitions. *IEEE TPAMI*, 39(1):22–34, 2006.

[21] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950 – 959, 2009.

[22] R. Singh, J. Xu, and B. Berger. Pairwise global alignment of protein interaction networks by matching neighborhood topology. In *RECOMB07*, pages 16–31, 2007.

[23] A. Smalter, J. Huan, and G. Lushington. GPM: A Graph Pattern Matching Kernel with Diffusion for Chemical Compound Classification. In *ICBBE*, 2008.

[24] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE TPAMI*, 10(5):695–703, 1988.

[25] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *WOSN09*, August 2009.

[26] J. T. Vogelstein, J. M. Conroy, L. J. Podrazik, S. G. Kratzer, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe. Fast inexact graph matching with applications in statistical connectomics. *CoRR*, abs/1112.5507, 2011.

[27] J. S. Yedidia, W. T. Freeman, and Y. Weiss. *Understanding belief propagation and its generalizations*, pages 239–269. Morgan Kaufmann Publishers Inc., 2003.

[28] L. Zager and G. Verghese. Graph similarity scoring and matching. *Applied Mathematics Letters*, 21(1):86–94, 2008.

[29] M. Zaslavskiy, F. Bach, and J.-P. Vert. A path following algorithm for the graph matching problem. *IEEE TPAMI*, 31(12):2227–2242, Dec. 2009.

Appendix A: Derivation of the APGD Equations

Here we give the lemmas and proofs that are used to derive the updating steps of the APGD method.

Lemma 1: The minimization of f in Problem 2 can be reduced to the problem: $\min_{\mathbf{P},\mathbf{Q}} \{||\mathbf{PAQ}||_F^2 - 2\,\mathrm{Tr}\,\mathbf{PAQB}^T\}$.

Proof: Starting from the definition of the Frobenius norm of $\mathbf{PAQ} - \mathbf{B}$, we obtain:

$$\begin{aligned} ||\mathbf{PAQ} - \mathbf{B}||_F^2 &= \mathrm{Tr}\,(\mathbf{PAQ} - \mathbf{B})(\mathbf{PAQ} - \mathbf{B})^T \\ &= ||\mathbf{PAQ}||_F^2 - 2\,\mathrm{Tr}\,(\mathbf{PAQB}^T) + \mathrm{Tr}\,(\mathbf{BB}^T), \end{aligned}$$

where we used the fact that $\mathrm{Tr}\,(\mathbf{PAQB}^T) = \mathrm{Tr}\,(\mathbf{PAQB}^T)^T$. Notice that the last term, $\mathrm{Tr}\,(\mathbf{BB}^T)$, does not depend on $\mathbf{P}$ or $\mathbf{Q}$, and does not affect the minimization. ■

Lemma 2: The derivative of the objective function, $f(\bullet)$, w.r.t. $\mathbf{P}$ is given by: $\frac{\partial f(\mathbf{P},\mathbf{Q})}{\partial \mathbf{P}} = 2(\mathbf{PAQ} - \mathbf{B})\mathbf{Q}^T\mathbf{A}^T$.

Proof: By using properties of matrix derivatives, we obtain:

$$\begin{aligned} &\frac{\partial(||\mathbf{PAQ}||_F^2 - 2\,\mathrm{Tr}\,(\mathbf{PAQB}^T))}{\partial \mathbf{P}} = \\ &= \frac{\partial \mathrm{Tr}\,(\mathbf{PAQQ}^T\mathbf{A}^T\mathbf{P}^T)}{\partial \mathbf{P}} - 2\frac{\partial \mathrm{Tr}\,(\mathbf{PAQB}^T)}{\partial \mathbf{P}} \\ &= 2(\mathbf{PAQ} - \mathbf{B})\mathbf{Q}^T\mathbf{A}^T \end{aligned}$$

■

Lemma 3: The derivative of the cost function, $f(\bullet)$, w.r.t. $\mathbf{Q}$ is given by:

$$\frac{\partial f(\mathbf{P},\mathbf{Q})}{\partial \mathbf{Q}} = 2\mathbf{A}^T\mathbf{P}^T(\mathbf{PAQ} - \mathbf{B})$$

Proof: By using properties of matrix derivatives, and the invariant property of the trace under cyclic permutations $\mathrm{Tr}\,(\mathbf{PAQQ}^T\mathbf{A}^T\mathbf{P}) = \mathrm{Tr}\,(\mathbf{A}^T\mathbf{P}^T\mathbf{PAQQ}^T)$, we obtain:

$$\begin{aligned} &\frac{\partial(||\mathbf{PAQ}||_F^2 - 2\,\mathrm{Tr}\,\mathbf{PAQB}^T)}{\partial \mathbf{Q}} = \\ &= \frac{\partial Tr(\mathbf{A}^T\mathbf{P}^T\mathbf{PAQQ}^T)}{\partial \mathbf{Q}} - 2\frac{\partial \mathrm{Tr}\,(\mathbf{PAQB}^T))}{\partial \mathbf{Q}} = \\ &= 2\mathbf{A}^T\mathbf{P}^T(\mathbf{PAQ} - \mathbf{B}) \end{aligned}$$

■

Observation 3: The partial derivative w.r.t. $\mathbf{P}$ of the sparsity penalty term of the cost function, f_{aug}, is $\frac{\partial(\mathbf{1}^T\mathbf{P}\mathbf{1})}{\partial \mathbf{P}} = \mathbf{1}\mathbf{1}^T$.

Appendix B: Step Choice

To find the step η_P that minimizes $f_{aug}(\eta_P)$, we take its derivative and set it to 0:

$$\frac{df_{aug}}{d\eta_P} = \frac{d(Tr\{\mathbf{P}^{(k+1)}\mathbf{AQ}(\mathbf{P}^{(k+1)}\mathbf{AQ})^T - 2\mathbf{P}^{(k+1)}\mathbf{AQB}^T\} + \lambda\sum_{i,j} P_{ij}^{(k+1)})}{d\eta_P} = 0, \tag{6}$$

where $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_P\Delta_P$, where $\Delta_P = \nabla_{\mathbf{P}} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}}$. It also holds that

$$\begin{aligned} &\mathrm{Tr}\,(\mathbf{P}^{(k+1)}\mathbf{AQ}(\mathbf{P}^{(k+1)}\mathbf{AQ})^T) - 2\mathbf{P}^{(k+1)}\mathbf{AQB}^T) = \\ &||\mathbf{P}^{(k)}\mathbf{AQ}||_F^2 - 2\,\mathrm{Tr}\,\mathbf{P}^{(k)}\mathbf{AQB}^T + \eta_P^2||\Delta_P\mathbf{AQ}||_F^2 + \\ &+2\eta_P\,\mathrm{Tr}\,(\Delta_P\mathbf{AQB}^T) - 2\eta_P\,\mathrm{Tr}\,(\mathbf{P}^{(k)}\mathbf{AQ})(\Delta_P\mathbf{AQ}) \end{aligned} \tag{7}$$

Substituting Eq. (7) in (6), and solving for η_P yields the 'best value' in the line search point of view. The computations are symmetric for η_Q, and, thus, omitted.