BIG-ALIGN: Fast Bipartite Graph Alignment

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

Hanghang Tong
City College, CUNY
tong@cs.ccny.cuny.edu

David Lubensky
IBM TJ Watson Researchl
davidlu@us.ibm.com

ABSTRACT

How can we find the virtual twin (i.e., the same or similar user) on Twitter 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, etc.

In this work, we focus on the alignment of bi-partite graphs, which despite their ubiquity, has been largely ignored by the extensive existing work on graph matching. We introduce a new optimization formulation for aligning bipartite graphs (e.g., users-groups graph); and propose an effective and fast algorithm to solve it. The extensive experimental evaluations show that our method outperforms the state-of-art graph matching algorithms in both matching accuracy and running time.

1. INTRODUCTION

Can we spot the *same* people in two different social networks, say Twitter 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 so that we can find similarities between the people 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 nodes and edges sets respectively -, how can we permute their nodes, so that the graphs have as much similar structure as possible? This is a core building block in many desciplines as it essentially enables us to link the different networks together so that we can search and/or transfer valuable knowledge across different networks. To name a few, the notion of graph similarity and alignment appears in protein-protein alignment [5, 2], chemical compound comparison [18], information extraction for finding synonyms in a single language or translation between different languages [2], answering similarity queries in databases [12], pattern recognition [6, 24] and many more.

[1]Throughout this work we use the words 'align(ment)' and 'match(ing)' interchangeably.

KDD '13 Chicago, IL, USA

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, etc. Despite their ubiquity, most - if not all - of the existing work on graph alignment/matching are 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:

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 our constrained optimization problem with careful handling of many subtleties. 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, for both bipartite graphs and unipartite graphs.

The rest of the paper is organized as follows: Section 2 presents the formal definition of the graph matching problem; Section 3 our proposed method; and Section 5the experimental results. Finally, we give the related works and conclusions in Section 6 and 7.

2. PROPOSED PROBLEM FORMULATION

Table 1: 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.
$\mathbf{P}$	node (user)-level correspondence matrix
$\mathbf{Q}$	community (group)-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_1, η_2	step of gradient descent for $\mathbf{P}$, $\mathbf{Q}$
ϵ	small constant (> 0) for the convergence of grad. descent

The alignment of graphs is a problem that has been studied in numerous communities in the past about three decades due to its occurrence in many applications. However, most of the research has been focused on *unipartite* graphs, i.e. graphs that consist of only one type of nodes. In this work, we introduce the problem

of aligning bipartite graphs (i.e., graphs that consist of edges only between two disjoint sets of vertices, e.g. user-group graph, where the edges represent that a user belongs to a specific group). First, we give the definition of *bipartite* graph alignment by extending the traditional *unipartite* graph alignment problem definition, and then we introduce a new formulation that also accommodates the requirements of the current applications in data mining. We list the frequently used symbols in Table 1.

PROBLEM 1 (ADAPTATION OF TRADITIONAL DEFINITION). *Given two **bipartite** graphs, G_A and G_B, with adjacency matrices $\mathbf{A}$ and $\mathbf{B}$, we want to 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 corresponding matrix.

The permutation matrix (i.e., a square binary matrix with exactly one entry 1 in each row and column, and 0s elsewhere) $\mathbf{P}$, which reorders the rows of the adjacency matrix $\mathbf{A}$, encodes the 1-to-1 correspondences between the nodes of the first set of the input bipartite graphs. Similarly, $\mathbf{Q}$, which reorders the columns of $\mathbf{A}$, is related to the second set of the input bipartite graphs.

The above-mentioned problem is not only hard to solve due to its combinatorial nature, but also the permutation matrices imply that we are in search for hard assignments between the nodes of the input bipartite graphs. However, finding hard assignments might not be possible nor realistic. For instance, suppose that the input graphs have perfect star structure: aligning the spokes of the two stars is impossible, since they have exactly the same structural "footprint"; any way of aligning the spokes is *equiprobable*, and soft assignment may be more valuable than hard assignment.

That said, we relax problem 1 that is directly adapted from the well-studied case of unipartite graphs, and formulate 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 $\mathbf{B}$, we want to 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. $0 \leq P_{ij} \leq 1$ and $0 \leq Q_{ij} \leq 1$, and
(2) [Sparsity] the matrices are sparse, i.e. $||\mathbf{P}^{(v)}||_0 \leq t$ and $||\mathbf{Q}^{(v)}||_0 \leq t$ 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.

Throughout the paper we will refer to the following example in order to simplify our description: Let $\mathbf{A}$ be the "user-group" Twitter graph, and $\mathbf{B}$ the corresponding Facebook graph. The optimization problem given in Problem 2 involves finding how we should permute the users of Twitter ($\mathbf{P}$), as well as its groups or communities ($\mathbf{Q}$), so that it resembles structurally the Facebook network as much as possible.

The first constraint of Problem 2 lends a probabilistic interpretation to the node matchings: the entries of the correspondence matrix $\mathbf{P}$ (or $\mathbf{Q}$) describe the probability that a person (or community) of Twitter corresponds to a user (or group) of Facebook. The requirement of non-integer entries for the matrices (a) renders the optimization problem easier to solve, and (b) has a nice, realistic interpretation for the large networks that are of interest nowadays; it does not provide only 1-to-1 correspondence, but also reveals similarities between people/communities across networks. 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, each of whose rows and columns sum to 1), which is common practice in the literature. Another important property of our formulation is that 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). Therefore, our formulation includes 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 person/group of Twitter to be matched to every person/group of Facebook is not realistic and, actually, it is problematic -if not impossible- for large graphs, as it would have quadratic space cost w.r.t. the size of the input graphs.

Note that 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 community-level correspondence in the case of the "user-group" graph.)

3. BIG-ALIGN FOR BIPARTITE GRAPHS

Here, we present our algorithm, BIG-ALIGN. 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. We also carefully design the search procedure to improve the efficiency of BIG-ALIGN. To this end, BIG-ALIGN comprises several important ideas: (i) a projected, alternating gradient descent (PAGRAD) approach to find the local minima of the newly-defined optimization problem 2, (ii) a net-inspired initialization (NET-INIT) of the correspondence matrices to find a good starting point, (iii) automatic choice of the step(s) of the gradient descent, and (iv) handling the node-multiplicity problem, i.e. the "problem" of having nodes with exactly the same structural "footprint" (e.g. spokes of a star) to improve both effectiveness and efficiency. Next, we describe these individual components before we present the overall algorithm, BIG-ALIGN.

3.1 PAGRAD: Mathematical formulation

In order to solve the optimization 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 / non-negativity constraint. Therefore, the sparsity constraint now takes the form: $\sum_{i,j} P_{ij} \leq t$ and $\sum_{i,j} Q_{ij} \leq t$. Based on this approach, our bipartite graph alignment problem is equivalent to the problem described in the following theorem.

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), can be equivalently described as:

$$\begin{aligned}\min_{\mathbf{P},\mathbf{Q}} f_{aug}(\mathbf{P},\mathbf{Q}) &= \\ &= \min_{\mathbf{P},\mathbf{Q}}\{||\mathbf{PAQ}-\mathbf{B}||_F^2 + \lambda\sum_{i,j}P_{ij} + \mu\sum_{i,j}Q_{ij}\} \\ &= \min_{\mathbf{P},\mathbf{Q}}\{Tr(\mathbf{PAQ}(\mathbf{PAQ})^T - 2\mathbf{PAQB}^T) + \\ &\qquad + \lambda\mathbf{1}^T\mathbf{P1} + \mu\mathbf{1}^T\mathbf{Q1}\},\end{aligned} \tag{1}$$

where $||\bullet||_F$ is the Frobenius norm of the enclosed matrix, $\mathbf{P}$ and $\mathbf{Q}$ are the node- and community-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. □

Note that the probabilistic constraint is not explicitly accommodated by the augmented cost function (f_{aug}); instead we apply the *projection* technique to the solution matrices: 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.

We solve the minimization problem by using a variant of the gradient descent algorithm. First, notice that the cost function (1) is bivariate, as it encompasses the minimization of both $\mathbf{P}$ and $\mathbf{Q}$. Therefore, 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 matrices become invalid temporarily, we use the projection method described above (probabilistic constraint guaranteed). The update steps of our projected, alternating gradient descent approach (PAGRAD) are given by the following theorem.

THEOREM 2. **[Update Step]** *The update steps for the node ($\mathbf{P}$) and community-level ($\mathbf{Q}$) correspondence matrices of* PAGRAD *are given by:*

$$\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_1 \cdot \left(2(\mathbf{P}^{(k)}\mathbf{AQ}^{(k)} - \mathbf{B})\mathbf{Q}^{T^{(k)}}\mathbf{A}^T + \lambda\mathbf{1}\mathbf{1}^T\right)$$

$$\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_2 \cdot \left(2\mathbf{A}^T\mathbf{P}^{T^{(k+1)}}(\mathbf{P}^{(k+1)}\mathbf{AQ}^{(k)} - \mathbf{B}) + \mu\mathbf{1}\mathbf{1}^T\right),$$

where $\mathbf{P}^{(k)}$, $\mathbf{Q}^{(k)}$ are the correspondence matrices at iteration k, η_1 and η_2 are the steps of the alternating gradient descent, and $\mathbf{1}$ is the all-1 column-vector.

PROOF. See Lemmas 2, 3, and Observation 3 in Appendix A. □

Note that in the above formulas, we assume that $\mathbf{A}$ and $\mathbf{B}$ are the rectangular, adjacency matrices of the 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:

$$min_{\mathbf{P}*}||\mathbf{P}^*\mathbf{AP}^{*T} - \mathbf{B}||_F^2.$$

3.2 NET-INIT: Initialization of Alignment

Up to this point, we have the whole arsenal of the mathematical foundation at our disposal to build our algorithm, BIG-ALIGN. There is only one basic component missing: the initialization of the correspondence matrices. Our optimization problem is non-convex (not even bi-convex), and the gradient descent is known for getting 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 as in [19] and [7]. 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). Our initialization apograph is based on the following observations of large-scale, real biparite graphs:

OBSERVATION 1. *Large, real networks have skewed or power-law-like degree distribution. Specifically in bipartite graphs, usually one of the node sets is significantly smaller than the other, and has skewed degree distribution.*

The implicit assumption of NET-INIT is that a person is almost equally popular in different social networks, or more generally, the same entity has almost similar 'behavior' in 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 metrics/features (e.g., weight, ranking, etc) that may capture better the node behavior.

Our initialization approach consists of 4 steps. We refer to the example of Twitter and Facebook that we mentioned above; the first set of the bipartite graphs consists of users, and the second set of groups. Assume that the set of groups is significantly smaller than the set of users. In a nutshell, the steps, which are pictorially shown in Fig. 1(b), are:

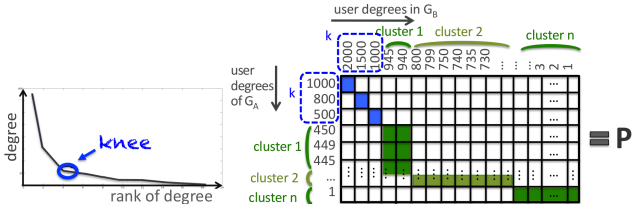
1. **Match 1-by-1 the top-k** high-degree groups in the Twitter and Facebook graphs.
2. For each of the matched groups, **align their neighbors** based on their relative degree difference (RDD), which we explain next.
3. Create c_g **clusters of the remaining groups** in both networks, based on their 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 approach.
4. Create c_u **clusters of the remaining users** in both networks, based on the degrees. Align the users using the RDD approach withing the corresponding user clusters.

Finding the top-k high-degree nodes of Step 1.

To find k, we borrow the idea of scree plot, which is used in the 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 corresponding degree we "safely" match the users of the two graphs one-by-one, i.e. the most popular user of Twitter is aligned initially with the most popular user of Facebook etc. For the automatic detection of the knee, we use the following heuristic: we assume that we have detected the knee if the slope of a piecewise line in the plot is less than 5% of the slope of the previous line.

*Relative Degree Distance (*RDD*).*

As mentioned above, we match the nodes in corresponding clusters using the RDD method. 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. Therefore, we are in search of a function that assigns higher prob-



(a) "Scree-like" plot for NET-INIT. (b) Pictorial initialization of $\mathbf{P}$.

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

abilities to matchings of similar nodes, and lower probabilities to matchings of very dissimilar nodes w.r.t. their degrees.

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

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

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

Notice that $\text{rdd}(i,j)$ corresponds to the similarity between the nodes i and j. Equation (2) captures one additional desired property: it penalizes the alignments based on the relative difference of the degrees, e.g., two nodes of degrees 1 and 20 respectively are less similar than two nodes with degrees 1001 and 1020 respectively.

3.3 Step choice for PAGRAD

One of the most important parameters that come up in the projected, alternating gradient descent 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 "line search" (Algorithm 2).

Here, we explain how line search works only for the first phase of PAGRAD, since it's symmetric for the second phase. The step η_1 is used in the the first phase, where we are minimizing the objective function w.r.t. $\mathbf{P}$, and the correspondence matrix $\mathbf{Q}$ is considered fixed. Line search consists of viewing the augmented cost function, f_{aug}, as a function of η_1 only (not as a function of $\mathbf{P}$ and $\mathbf{Q}$), and the goal is to find the value of η_1 that loosely minimizes it.

The baseline approach (BIG-ALIGN-Points) consists of approximately minimizing the augmented cost function: we randomly pick some values for η_1 within some reasonable range, and compute the value of the cost function. For the current gradient descent step, we choose the value of η_1 that corresponds to the minimum cost function value. This approach is computationally expensive, as we shall see in Section 5.

By carefully handling the objective function of our optimization problem, we can find closed forms for η_1 and η_2. We call the version of our algorithm that uses exact line search for choosing the gradient descent steps BIG-ALIGN-Exact.

THEOREM 3. **[Optimal Step Size for P]** *In the first phase of* PAGRAD*, the value of the step η_1 that exactly minimizes the augmented function, $f_{aug}(\eta_1)$, is given by:*

$$\eta_1 = \frac{2\,\text{Tr}\,\{(\mathbf{P}^{(k)}\mathbf{A}\mathbf{Q})(\Delta_\mathbf{P}\mathbf{A}\mathbf{Q})^T - (\Delta_\mathbf{P}\mathbf{A}\mathbf{Q})B^T\} + \lambda\sum_{i,j}\Delta_{Pij}}{2||\Delta_P\mathbf{A}\mathbf{Q}||_F^2}, \quad (3)$$

where $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_1\Delta_\mathbf{P}$, $\Delta_\mathbf{P} = \nabla_\mathbf{P} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}}$ and $\mathbf{Q} = \mathbf{Q}^{(k)}$.

PROOF. See Appendix B. □

Similarly, we find the appropriate value for the step η_2 of the second phase of PAGRAD.

THEOREM 4. **[Optimal Step Size for Q]** *In the second phase of* PAGRAD*, the value of the step η_2 that exactly minimizes the augmented function, $f_{aug}(\eta_2)$, is given by:*

$$\eta_2 = \frac{2\,\text{Tr}\,\{(\mathbf{P}\mathbf{A}\mathbf{Q}^{(k)})(\mathbf{P}\mathbf{A}\Delta_Q)^T - (\mathbf{P}\mathbf{A}\Delta_\mathbf{Q})B^T\} + \mu\sum_{i,j}\Delta_{Qij}}{2||\mathbf{P}\mathbf{A}\Delta_Q||_F^2}, \quad (4)$$

where $\Delta_\mathbf{Q} = \nabla_\mathbf{Q} f_{aug}|_{\mathbf{Q}=\mathbf{Q}^{(k)}}$, $\mathbf{P} = \mathbf{P}^{(k)}$, and $\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_2\Delta_\mathbf{Q}$.

PROOF. Omitted for brevity. □

Compared with BIG-ALIGN-Points, BIG-ALIGN-Exact is significantly faster. It turns out that we can do even better based on the following observation. 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, BIG-ALIGN-Skip, which does line search for the first few (say, 100) iterations, and then updates the values of the steps every few (say, 500) iterations, thus leading to significantly fewer computations to search for optimal step sizes.

3.4 Handling the node-multiplicity problem

Before presenting our algorithm, BIG-ALIGN, we mention one more observation that is important when trying to solve the alignment problem for real bipartite graphs.

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.*

For instance, in many real-world networks, a commonplace structure is stars, but it is impossible to tell the "spokes" apart. Other examples of non-distinguishable nodes include the members of a cliques, etc.

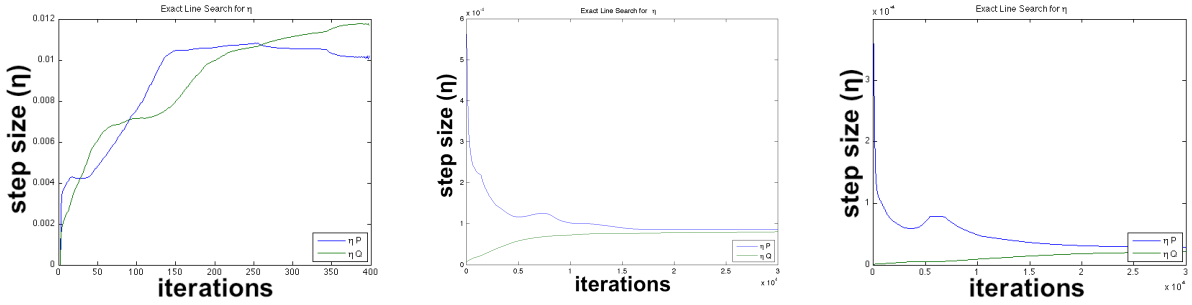
To address this problem, we introduce a pre-processing phase at which we eliminate nodes with identical "structural footprints" by aggregating them in super-nodes. For example, a star with 100 spokes which are connected to the center by edges of weight 1, will be replaced by a super-node connected to 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.

3.5 BIG-ALIGN: Putting everything together

The previous subsections shape up our proposed algorithm, BIG-ALIGN, whose pseudocode 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. Although the optimization problem contains one more sparsity penalty, μ, we set $\mu = \frac{\lambda * (\text{elements in } \mathbf{Q})}{\text{elements in } \mathbf{P}}$ so that we put same amount of penalty for each non-zero element of $\mathbf{P}$ and $\mathbf{Q}$.

It is worth mentioning that 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, P_{ij}. It is clear that this assignment is very fast, and even parallelizable; the assignment



(a) Graph with 50 nodes. (b) Graph with 300 nodes. (c) Graph with 900 nodes.

Figure 2: (Hint for speedup.) Size of optimal step for $\mathbf{P}$ (blue) and $\mathbf{Q}$ (green) vs. the number of iterations. Notice 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.

per row is independent of all other row assignments. What is more, this strategy brings another desirable property - in the case of duplicate nodes (which is often the case in real bipartite graphs), it is desirable to align multiple nodes of graph G_A (all the duplicate nodes) to the same node of graph G_B.

Algorithm 1 BIG-ALIGN-Exact: Bipartite Graph Alignment

INPUT: $\mathbf{A}$, $\mathbf{B}$, λ, MAXITER
$\epsilon = 10^{-6}$; $cost(0) = 0$; $k = 1$;
/* STEP 1: pre-processing for node-multiplicity */
aggregating identical nodes
/* STEP 2: initialization */
[P0, Q0] = NET-INIT-ialization
$cost(1) = f_{aug}(\mathbf{P}0, \mathbf{Q}0)$
/* STEP 3: alternating gradient descent with projection */
while $|cost(k) - cost(k+1)| > \epsilon$ & $k <$ MAXITER **do**
  $k++$
  /* PHASE 1: fixed $\mathbf{Q}$, minimization w.r.t. $\mathbf{P}$ */
  $\eta_{1k} = \text{LINESEARCH-P}(\mathbf{P}^{(k)}, \mathbf{Q}^{(k)}, \nabla_{\mathbf{P}} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}})$
  $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_{1k} \nabla_{\mathbf{P}} f_{aug}(\mathbf{P}^{(k)}, \mathbf{Q}^{(k)})$
  VALIDPROJECTION($\mathbf{P}^{(k+1)}$)
  /* PHASE 2: fixed $\mathbf{P}$, minimization w.r.t. $\mathbf{Q}$ */
  $\eta_{2k} = \text{LINESEARCH-Q}(\mathbf{P}^{(k+1)}, \mathbf{Q}^{(k)}, \nabla_{\mathbf{Q}} f_{aug}|_{\mathbf{Q}=\mathbf{Q}^{(k)}})$
  $\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_{2k} \nabla_{\mathbf{Q}} f_{aug}(\mathbf{P}^{(k+1)}, \mathbf{Q}^{(k)})$
  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

Algorithm 2 Line Search for η_1 and η_2

function LINESEARCH-P($\mathbf{P}, \mathbf{Q}, \Delta_{\mathbf{P}}$)
 return

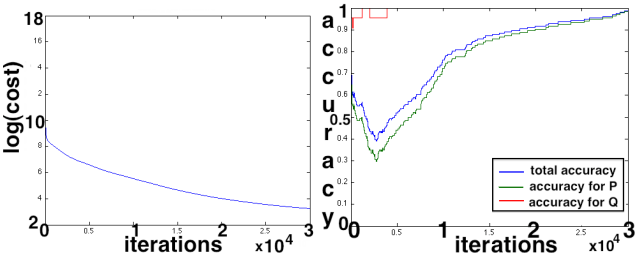
$$\eta_1 = \frac{2\,\mathrm{Tr}\{(\mathbf{P}^{(k)}\mathbf{A}\mathbf{Q})(\Delta_{\mathbf{P}}\mathbf{A}\mathbf{Q})^T - (\Delta_{\mathbf{P}}\mathbf{A}\mathbf{Q})B^T\} + \lambda \sum_{i,j} \Delta_{Pij}}{2||\Delta_P \mathbf{A}\mathbf{Q}||_F^2}$$

end function

function LINESEARCH-Q($\mathbf{P}, \mathbf{Q}, \Delta_{\mathbf{Q}}$)
 return

$$\eta_2 = \frac{2\,\mathrm{Tr}\{(\mathbf{P}\mathbf{A}\mathbf{Q}^{(k)})(\mathbf{P}\mathbf{A}\Delta_Q)^T - (\mathbf{P}\mathbf{A}\Delta_{\mathbf{Q}})B^T\} + \mu \sum_{i,j} \Delta_{Qij}}{2||\mathbf{P}\mathbf{A}\Delta_Q||_F^2}$$

end function



(a) Cost function. (b) Accuracy.

Figure 3: BIG-ALIGN: 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- (red) level 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).

4. 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 extract d node features (invariants), and form the $n \times d$ bipartite graph node-to-feature, where $n \gg d$.

Step 2: Finding P. 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:

$$\mathbf{P} \cdot (\mathbf{A}\mathbf{A}^{\mathbf{T}}) = \mathbf{B}\mathbf{A}^{\mathbf{T}} - \lambda/2 \cdot \mathbf{1}\mathbf{1}^{\mathbf{T}}$$

Note that $\mathbf{A}$ is $n \times d$. If we do SVD 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

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

where $\mathbf{X} = \mathbf{A^TUS^{-2}U^T}$ and $\mathbf{Y} = \lambda/2\cdot\mathbf{1^TUS^{-2}U^T}$. In other words, we can exactly (and *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 Equation (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 $O(nd + n) = O(nd)$.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate our proposed algorithms, BIG-ALIGN and UNI-ALIGN, w.r.t. alignment accuracy and runtime, and also compare them to the state-of-art methods.

5.1 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 2: (i) the influential eigenvalue decompositio based approach proposed by Umeyama [19], (ii) a recent NMF-based approach (Non-negative Matrix Factorization) [7], and (iii) a fast, and scalable Belief Propagation-based (BP) approach [2].

It should be pointed out that these algorithms are designed for unipartite graphs, so before applying them to the bipartite graphs that we study, we convert them to unipartite graphs as in Proposition 1. Moreover, the BP-based approach is not readily applicable in our setting; the input of the algorithm is not only the two graphs that we want to align, but also a bipartite graph that encodes the potential alignments for each node of the input graphs. Given that this information is not available in our setting, we use the following heuristics: (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; and (b) `deg`ree-based bipartite graph, where only nodes with the same degree in both graphs are considered possible matchings.

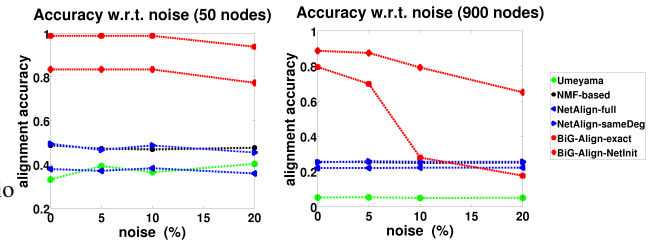
Table 2: Graph Alignment Algorithms: name conventions, short description, type of graphs on which they were designed for ('uni-' for unipartite, 'bi-' for bipartite graphs), and reference.

Name	Description	Graph	Source
`Umeyama`	eigenvalue-based	uni-	[19]
`NMF-based`	NMF-based	uni-	[7]
`NetAlign-full`	BP-based with uniform init.	uni-	Modified
`NetAlign-deg`	BP-based with same-degree init.	uni-	from [2]
BIG-ALIGN-Points	PAGRAD + approx. Line Search	**bi-**	current
BIG-ALIGN-Exact	PAGRAD + exact Line Search	**bi-**	current
BIG-ALIGN-Skip	PAGRAD + skip some Line Search	**bi-**	current
UNI-ALIGN	BIG-ALIGN-inspired (SVD)	uni-	current

5.2 BIG-ALIGN

Setup. For the experiments on bipartite graphs, we use the movie-genre graph of the MovieLens network [2]. Each of the 1,027 movies is linked to at least one of the 23 genres (e.g., comedy, romance, drama). To evaluate the accuracy and runtime of our method, we extract from the MovieLens network subgraphs with different sizes. For each of them, following the tradition in the literature, we generate permutations, $\mathbf{B}$, with noise level ($noise$) 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 of the pairs of subnetworks, and report the mean accuracy and runtime.

Accuracy. First, we compare the alignment algorithms with respect to the accuracy of the matchings. Figures 4 (a) and (b) present the accuracy of the methods for varying levels of noise in the permutations, $\mathbf{B}$, of initial graphs, $\mathbf{A}$, of two different sizes. We observe that BIG-ALIGN outperforms all the competitor methods in most cases with a large margin. The only exception is the case of 20% noise in the graphs with 900 nodes 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.



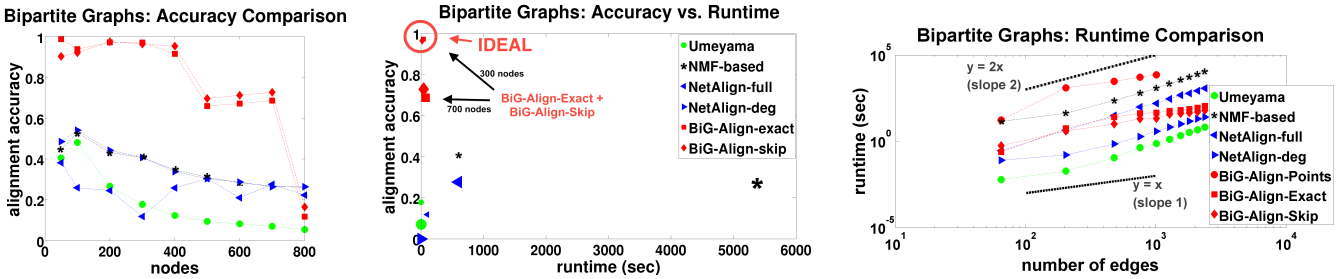
(a) Graphs of 50 nodes. (b) Graphs of 900 nodes.

Figure 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.

Figure 5(a) depicts the accuracy of the alignment approaches 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 graphs of size 50, where the consecutive optimal step sizes 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. As shown in Fig. 5(c) with runtime vs. number of edges in the graphs, `Umeyama`'s algorithm and `NetAlign-deg` are the fastest methods (but at the cost of accuracy). 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 non-optimized algorithm, 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).

[2] `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) 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.

Figure 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 points) 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.

It is worth mentioning that currently BIG-ALIGN is a single machine implementation, but it has the potential for further speedup. For example, it could be parallelized by splitting the optimization problem to smaller subproblems (by decomposing the matrices, and doing simple column-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. Table 3 presents the runtime and accuracy of BIG-ALIGN-Points, BIG-ALIGN-Exact, and BIG-ALIGN-Skip, for graphs with different sizes. Note that BIG-ALIGN-Skip is not only $\sim 350\times$ faster than the non-optimized variant, BIG-ALIGN-Points, but also more accurate. In addition, it is $\sim 2\times$ faster than BIG-ALIGN-Exact with higher or equal accuracy. This speedup can be further increased by skipping more updates of the gradient descent steps.

Table 3: Runtime (top) and accuracy (bottom) comparison of the BIG-ALIGN variants: 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-Points		BIG-ALIGN-Exact		BIG-ALIGN-Skip	
Nodes	**mean**	**std**	**mean**	**std**	**mean**	**std**
RUNTIME (SEC)						
50	17.3	0.05	0.24	0.08	0.56	0.01
100	1245.7	394.55	5.6	2.93	3.9	0.05
200	2982.1	224.81	25.5	0.39	10.1	0.10
300	5240.9	30.89	42.1	1.61	20.1	1.62
400	7034.5	167.08	45.8	2.058	21.3	0.83
500	-	-	57.2	2.22	36.6	0.60
600	-	-	64.5	2.67	40.8	1.26
700	-	-	73.6	2.78	44.6	1.23
800	-	-	86.9	3.63	49.9	1.06
900	-	-	111.9	2.96	61.8	1.28
ACCURACY						
50	0.982	0.02	0.988	0	0.904	0.03
100	0.922	0.07	0.939	0.06	0.922	0.07
200	0.794	0.01	0.973	0.01	0.975	0.00
300	0.839	0.02	0.972	0.01	0.964	0.01
400	0.662	0.02	0.916	0.03	0.954	0.01
500	-	-	0.66	0.20	0.697	0.24
600	-	-	0.67	0.20	0.713	0.23
700	-	-	0.69	0.20	0.728	0.19
800	-	-	0.12	0.02	0.165	0.03
900	-	-	0.17	0.20	0.195	0.22

5.3 UNI-ALIGN

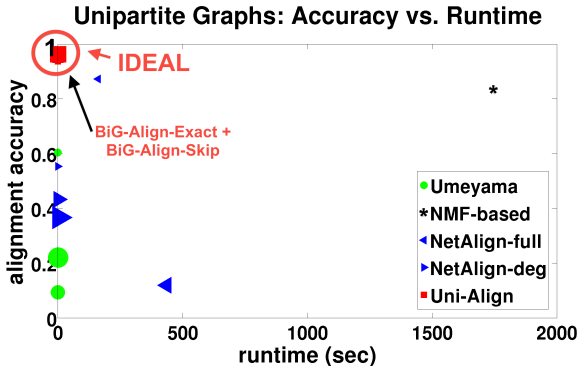
Setup. To evaluate our proposed method, UNI-ALIGN, for aligning unipartite graphs, we use the 63731×63731 Facebook who-links-to-whom graph [20]. In this case, the baseline approaches are readily employed, while our method requires the conversion of the given unipartite graph to bipartite. We do so by extracting some unweighted egonet features for each node (degree of node, degree of egonet, 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.

Accuracy. The accuracy vs. runtime plot in Fig. 6(a) shows that UNI-ALIGN outperforms all 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 killed the runs for graphs of bigger sizes as the execution was too long. The rest approaches are fast enough, but yield poor accuracy.

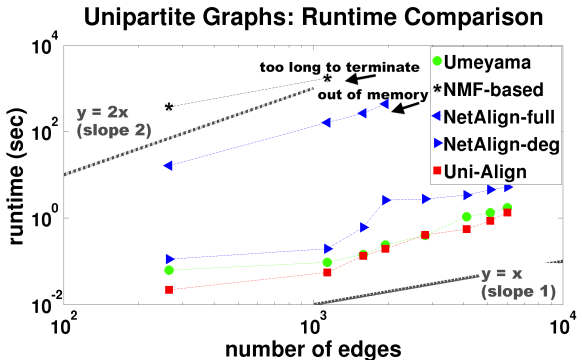
Runtime. Figure 6(b) compares the graph alignment algorithms w.r.t. their running time. UNI-ALIGN is the fastest approach, closely followed by Umeyama's algorithm. `NetAlign-deg` is some orders of magnitude slower than the previously mentioned methods. However, `NetAlign-full` ran out of memory for graphs with more than 2.8K edges; we killed `NMF-based` 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.

6. RELATED WORK

The graph alignment problem is of such great interest that the number of publications exceeds 150 and spans numerous research fields: from data mining to security and re-identification [13, 9], bioinformatics [17, 10, 3], databases [12], chemistry [18], vision, and pattern recognition [6]. Among the suggested approaches are genetic, spectral, clustering algorithms [15], decision trees, expecation-maximization [11], graph edit distance [16], simplex [1], non-linear optimization [8], iterative HITS-inspired [4, 23]. Notice that all



(a) (Higher and left is better.) Accuracy of alignment vs. runtime (in seconds) 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.

Figure 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).

these works are designed for unipartite, while we focus on bipartite graphs.

One of the well-known approaches is Umeyama's near-optimum solution for nearly-isomorphic graphs [19]. 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 [14] is employed at the end to find the node correspondences. The constraint that $\mathbf{P}$ is doubly stochastic matrix is imposed in [21], and [24], where the proposed formulation, PATH, is based on convex and concave relaxations. Ding et al [7] 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. [5] propose distributed, heuristic, message-passing algorithms - based on Belief Propagation [22] - for protein alignment and prediction of interacting proteins. Independently, Bayati et al [2] 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. [17].

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

7. 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 (PAGRAD) 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 and efficiency, for both bipartite graphs and unipartite graphs.

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

8. REFERENCES

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

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

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

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

[5] 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.

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

[7] 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.

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

[9] 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.

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

[11] B. Luo and E. R. Hancock. Iterative procrustes alignment with the em algorithm. *Image Vision Comput.*,

20(5-6):377–396, 2002.

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

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

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

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

[16] 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.

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

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

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

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

[21] 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.

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

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

[24] 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 PAGRAD Equations

Here we give the lemmas and proofs that are used to derive the updating steps of the PAGRAD 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 η_1 that minimizes $f_{aug}(\eta_1)$, we take its derivative and set it to 0:

$$\frac{df_{aug}}{d\eta_1} = \frac{d(Tr\{\mathbf{P}^{(k+1)}\mathbf{AQ}(\mathbf{P}^{(k+1)}\mathbf{AQ})^T - 2\mathbf{P}^{(k+1)}\mathbf{AQB}^T\} + \lambda\sum_{i,j}P_{ij}^{(k+1)})}{d\eta_1} = 0, \quad (6)$$

where $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_1\Delta_P$, where $\Delta_P = \nabla_{\mathbf{P}} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}}$. It also holds that

$$\begin{aligned}&\mathrm{Tr}\,(\mathbf{P}^{(k+1)}\mathbf{AQ}(\mathbf{P}^{(k+1)}\mathbf{AQ})^T) - 2\mathbf{P}^{(k+1)}\mathbf{AQB}^T) = \\ &||\mathbf{P}^{(k)}\mathbf{AQ}||_F^2 - 2\,\mathrm{Tr}\,\mathbf{P}^{(k)}\mathbf{AQB}^T + \eta_1^2||\Delta_P\mathbf{AQ}||_F^2 + \\ &+2\eta_1\,\mathrm{Tr}\,(\Delta_P\mathbf{AQB}^T) - 2\eta_1\,\mathrm{Tr}\,(\mathbf{P}^{(k)}\mathbf{AQ})(\Delta_P\mathbf{AQ}) \quad (7)\end{aligned}$$

Substituting Eq. (7) in (6), and solving for η_1 yields the 'best value' in the line search point of view. The computations are symmetric for η_2, and, thus, omitted.