HashAlign: Hash-based Alignment of Multiple Graphs

Mark Heimann*, Wei Lee*, Shengjie Pan, Kuan-Yu Chen, and Danai Koutra

Computer Science and Engineering,
University of Michigan
{mheimann,weile,jessepan,kyuchen,dkoutra}@umich.edu

Abstract. Fusing or aligning two or more networks is a fundamental building block of many graph mining tasks (e.g., recommendation systems, link prediction, collective analysis of networks). Most past work has focused on formulating the *pairwise* graph alignment problem as an optimization problem with varying constraints and relaxations. In this paper, we study the problem of *multiple* graph alignment (i.e., the problem of collectively aligning multiple graphs at once) and propose HASHALIGN, an efficient and intuitive hash-based framework for network alignment that leverages structural properties and other node and edge attributes (if available) simultaneously. We introduce a new construction of LSH families, as well as robust node and graph features that are tailored for this task. Our method quickly finds the alignment between multiple graphs while avoiding the all-pairwise-comparison problem by expressing all alignments in terms of a chosen 'center' graph. Our extensive experiments on synthetic and real networks show that, on average, HASHALIGN is $2\times$ faster and 10 to 20% more accurate than the baselines in *pairwise* alignment, and $2\times$ faster while 50% more accurate in *multiple* graph alignment.

1 Introduction

Much of the data that is generated daily naturally form graphs, such as interactions between users in social media, communication via email or phone calls, question answering in forums, interactions between proteins, and more. Additionally, graphs may be inferred from non-network data [6,18]. For joint analysis, it is often desirable to fuse multiple graph data sources by finding the corresponding nodes across them. This task, known as graph alignment or matching, is the focus of our work. It is a core graph theoretical problem that has attracted significant interest, both in academia and industry, due to its numerous applications: identifying users in social networks [21], matching similar documents in lingual matching [4], brain graph alignment in neuroscience, protein-protein alignment [4,5], chemical compound comparison, and more.

In many applications, such as in aligning protein-protein interaction networks, brain graphs or social networks, the goal is to align *multiple* (more than two) networks at once. Most existing methods get as input two networks, so they handle multiple network alignment by expensively computing all pairwise alignments. In this paper, we seek to devise an efficient method that collectively aligns multiple networks and can readily adapt to the existence or not of other node/edge information in addition to the graph topology, without increasing its complexity.

* These authors contributed equally to this work.

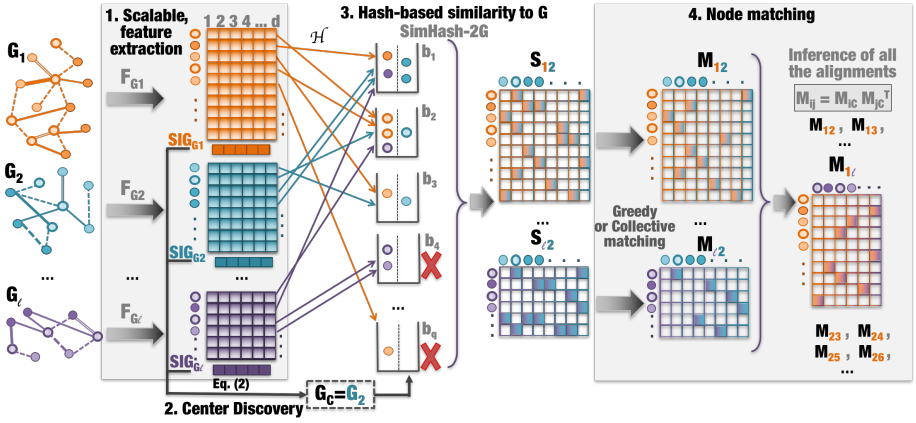


Fig. 1: Overview of proposed approach: HASHALIGN with input l undirected, weighted, attributed graphs (node/edge attributes are denoted with different shades/lines). The framework consists of four parts: (1) scalable, fast, robust, node-ID-invariant feature extraction per graph; (2) 'center' graph discovery, which is $G_C = G_2$ in this example; (3) efficient, hash-based similarity computation, $\mathbf{S}_{iC}$, between each graph G_i and G_C (buckets with red crosses do not contribute any pairwise similarity computations, and thus help with efficiency); and (4) node matching computation to find at most one matching per node in $\mathbf{M}_{ij}$.

Problem 1 (Multiple Graph Alignment with Side Information). **Given** l graphs, $G_1(\mathcal{V}_1, \mathcal{E}_1)$, $\ldots, G_l(\mathcal{V}_l, \mathcal{E}_l)$, where $\mathcal{V}_i$ and $\mathcal{E}_i$ are the node and edge sets of graph G_i, respectively, *with or without node/edge attributes*, we seek to **find** the correspondence between their nodes efficiently, so that the input graphs are as close to each other as possible.

To solve this problem, we propose HASHALIGN, an unsupervised method that is based on three key ideas: (i) inferring the similarity between nodes in different graphs based on structural properties and node/edge attributes; (ii) leveraging Locality Sensitive Hashing (LSH) [7] to minimize the number of pairwise node comparisons; (iii) choosing a 'center' graph out of l input graphs to which to align all the others, thereby avoiding solving $\binom{l}{2}$ pairwise graph alignment problems (instead solving $l-1$ alignments and quickly inferring the remaining node correspondences by applying simple transformations in the form of sparse matrix multiplications). A pictorial overview of our method is given in Fig. 1. Our main contributions are:

• **Flexible Framework.** We propose an efficient and accurate hashing-based family of algorithms, HASHALIGN, which solves the multiple network alignment problem. Our method is general and can readily incorporate any available node and edge attributes. HASHALIGN can be used as a standalone alignment method or provide its solution to initialize optimization problems for pairwise alignment (e.g., [4]).

• **Methods.** As part of our framework, we propose problem-specific choices of node and graph features, and introduce a new, robust construction of hash families.

• **Experiments.** We conduct extensive experiments on synthetic and real data, which show that HASHALIGN is 2-10× faster than the baselines that tackle either the multiple or pairwise alignment problem, while being equally or up to 50% more accurate.

For reproducibility, the code is available at `https://github.com/GemsLab/HashAlign.git`. Additional supplementary material is provided at `https://markheimann.github.io/papers/HashAlign-PAKDD18-full.pdf`.

2 Related Work

We review work that is relevant to our problem space and choices of techniques:

Graph Alignment. Scalable methods for pairwise graph alignment include a distributed, belief-propagation-based method for protein alignment [5], a message-passing algorithm for aligning sparse networks when some [4] or all [19] possible matchings are considered, alignment of bipartite networks [14,15], and attributed graph alignment [22]. *Multiple* network alignment, however, poses a further scalability challenge. For instance, the recent optimization-based formulation of [17] solves a bipartite matching problem in $O(n^3)$ time using the Hungarian algorithm and was only shown to scale to small networks. Zhang and Yu [21] introduce the notion of transitivity between graphs to align social networks more scalably, but also assume that some partial node matchings are provided (anchor links). Our method HASHALIGN preserves this notion of transitivity for any type of network, while seamlessly incorporating node and edge attribute information in intuitive, simple-to-implement, and highly scalable ways.

Locality-Sensitive Hashing. This technique for efficient similarity search has been used to accelerate the well-known k-nearest neighbor algorithm, often offering theoretical and practical improvements even over sophisticated data structures such as k-d trees [3]. It has also found use in matching problems in other domains, such as ontology matching in information retrieval [9]. In our proposed method, we leverage LSH to efficiently find nodes that are similar. For networks, [13] uses MinHash to find sets of similar nodes in a *single* attributed graph by relying on the adjacency matrix as features, but this is not applicable to the graph alignment setting. Thus, we introduce node-ID invariant representations and adapt LSH to find similarities *across* networks. Our contribution is orthogonal to prior works: a framework for network alignment, HASHALIGN, in which we propose design choices geared toward our specific domain.

3 Proposed Formulation: Two-Graph Alignment

In this section, we first introduce the alignment problem for two graphs. We then describe our proposed approach, and in the next section we extend it to the multiple graph alignment problem. Table 1 summarizes the main notations used in our analysis.

Table 1: Symbols and definitions. We use bold capital letters for matrices, bold lowercase letters for vectors and normal lowercase letters for scalars.

Symbols	Definitions
$G_p = (\mathcal{V}_p, \mathcal{E}_p)$	graph p with vertex set $\mathcal{V}_p$ and edge set $\mathcal{E}_p$
$\lvert\mathcal{V}_p\rvert = n_p, \lvert\mathcal{E}_p\rvert = m_p$	number of nodes and edges in graph G_p, resp.
$\boldsymbol{s}_{G_p}(v)$	$1 \times d_s$ vector of the structural invariants (e.g., PageRank) for node $v \in G_p$
$\boldsymbol{a_n}_{G_p}(v), \boldsymbol{a_e}_{G_p}(v)$	$1 \times d_{an}$ vector of node/edge attributes for node $v \in G_p$
$\mathbf{A}_{n_{G_p}}, \mathbf{A}_{e_{G_p}}$	the stacked node/edge attr. matrices of size $n_p \times d_{a_n}$ and $n_p \times n_p \times d_{a_e}$
$d = d_s + d_{an} + d_{ae}$	total number of (structural, node, and edge) features
$\mathbf{f}_{G_p}(v), \mathbf{F}_{G_p}$	$1 \times d$ all-feature vec. for node $v \in G_p$ and the resp. stacked $n_p \times d$ mat.
SIG_{G_p}	$1 \times 5d$ graph 'signature' vector representing graph G_p
$d(G_i, G_j)$	distance between graphs G_i and G_j
$\mathbf{S}_{ij}$	sparse $n_i \times n_j$ similarity matrix between graph G_i and G_j
$\mathbf{M}_{ij}$	$n_i \times n_j$ alignment between graph G_i and G_j
b_i	bucket i (hashing)
Z	number of bands (hashing)

3.1 Definition: Relaxed Two-Graph Alignment Problem

The typical graph alignment problem aims to find a one-to-one matching between the nodes of two input graphs. This problem is important, but in many applications it suffices to solve a relaxed version of it: finding *a small set of nodes* that are *likely* to correspond to a given node. For example, when aligning social networks to improve user recommendations, it is still useful to find the top-l most similar or likely-to-match individuals and incorporate this probabilistic information in the recommender system. Given this observation, we relax the original alignment problem as follows:

*Problem 2 (**Relaxed two-graph alignment**).* Given two graphs, $G_1(\mathcal{V}_1, \mathcal{E}_1)$ and $G_2(\mathcal{V}_2, \mathcal{E}_2)$, which may be (un)directed, (un)weighted and attributed / plain, **we seek to efficiently find** a *sparse*, weighted bipartite graph $G_S = (\mathcal{V}_1 \cup \mathcal{V}_2, \mathcal{E}_S)$ with edges representing potential matching pairs and being weighted by the likelihood of the match:

$$\forall \text{ potential matching pair } (u, v), u \in \mathcal{V}_1, v \in \mathcal{V}_2, \exists e \in \mathcal{E}_S : w_e = sim(u, v)$$
$$\text{and } |\mathcal{E}_S| < \alpha \cdot \max\{n_1, n_2\},$$

where $\alpha \in \mathbb{Z}$ $(\alpha > 1)$ is a small factor that controls the density of G_S.

To make sure that nodes are efficiently matched only to a few of their closest counterparts, the main requirement in Problem 2 is that the weighted bipartite graph G_S is sparse, i.e., $|\mathcal{E}_S| \ll n_1 \times n_2$. Most graph alignment methods find 1-1 matchings between the vertex sets [8,4], and a few approaches relax the requirements of the typical optimization problem to find probabilistic matchings [15], but each method targets a different type of graph (e.g., unipartite, undirected) and most of them rely only on the network structure. In this work we propose a different, similarity-based approach that encompasses all these settings. This approach is intuitive and leverages a suitably rich node representation to achieve superior accuracy.

A naive similarity-based method is to: (i) compute all the pairwise similarities between the nodes in G_1 and the nodes in G_2, and (ii) keep only the edges with similarities greater than a user-specified threshold. Although this approach results in a sparse graph G_S, it has several drawbacks. First, it is computationally expensive, since it computes *all* pairs of similarities, i.e., $n_1 \times n_2$ (which is big for large-scale networks), and later applies the threshold for edge filtering. Second, the threshold is arbitrary and affects the potential node matchings significantly. Third, among the possible options for representing the nodes, it is not clear how to choose the 'right' representation for similarity computations. Our proposed approach, which is shown pictorially in Figure 1, leverages hashing to overcome all these issues.

3.2 Node Representation: Handling Node and Edge Attributes

Our framework, HASHALIGN, requires a vector representation of each node. We want these to be comparable across graphs and also leverage node/edge attributes seamlessly.

We propose to represent each node u with a vector $\mathbf{f}(u)$ of structural features and node/edge attributes (if available). The benefit of this representation is that it can

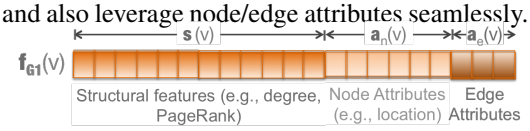


Fig. 2: Proposed feature-based, node-ID invariant representation of vertex v.

readily be adjusted to the type of the graphs and the richness of the available information *without any* changes in the problem formulation. Furthermore, it is *node-ID invariant* and can thus be meaningfully compared across graphs (where all nodes IDs are different: node $u \in \mathcal{V}_1$ may have neighbors $\{u_1, u_2, u_3\}$ in G_1, while a similar node $v \in \mathcal{V}_2$ may have neighbors $\{v_6, v_8, v_9\}$ in G_2). This is not true of representation learning methods such as DeepWalk and node2vec, which use random walks to sample context nodes by their IDs [10] and thus are not applicable to our multi-network problem setting [11]. Specifically, in Step 1 of our framework (Fig. 1), we concatenate d_s structural features, d_{an} node attributes, and d_{ae} edge attributes:

• **Structural features $\boldsymbol{s} \in \mathbb{R}^{1\times d_s}$.** Examples include the so-called local features (e.g., degree variants) and egonet features. The egonet of node u is defined as the induced subgraph of u and its neighbors, and structural features specific to the egonet include its number of edges, its degree, and more. In addition to these features, we also consider features that combine locality with globality, such as PageRank and various types of centrality. We choose specific structural features that are most robust to noise (Sec. 5.)

• **Node attributes $\boldsymbol{a_n} \in \mathbb{R}^{1\times d_{an}}$.** If a graph contains node attributes, the node feature vectors $\mathbf{f}$ can be extended to include those. Numerical features can be simply concatenated with the structural features, while categorical attributes can be incorporated by using 1-hot encoding and concatenated to the previously formed feature vector.

• **Edge attributes $\boldsymbol{a_e} \in \mathbb{R}^{1\times d_{ae}}$.** We propose converting numerical edge features to node attributes by applying an aggregate function $\xi : \mathcal{E}^{deg_u} \rightarrow \mathbb{R}$ (where the domain is the set of edges incident to $u \in \mathcal{V}$ and deg_u is its degree). Examples for $\xi()$ include sum, average, standard deviation, etc., which provide different edge-specific features for each node. For categorical features, we propose to encode the distribution of values per categorical feature. For example, if a feature has q possible values, then q entries with their frequencies will be concatenated with the previous features.

3.3 Proposed Hashing-based Computation of Potential Matchings

Now we have, for each node u in graph G_p, a real-valued vector $\mathbf{f}_{G_p}(u) \in \mathbb{R}^d$ constructed as described in Section 3.2. We propose to use hashing, and specifically Locality Sensitive Hashing (LSH) [7], in order to find a small number of potential matchings between nodes across graphs (i.e., nodes with high similarity) in a scalable way, without computing all pairs of $n_1 \times n_2$ similarities. In a nutshell, given a similarity function, LSH reduces the dimensionality of high-dimensional data while preserving their local similarities; that is, it efficiently maps similar data points (in our case, nodes) to the same buckets with high probability. Our proposed hashing approach for alignment takes as input the $\mathbf{F}_{G_1} \in \mathbb{R}^{n_1\times d}$ feature matrix (with row-wise node representations) for G_1 and $\mathbf{F}_{G_2} \in \mathbb{R}^{n_2\times d}$ for G_2, and hashes them row-wise using an LSH family $\mathcal{H}$.

Definition 1 (LSH-2G). *Given $\mathcal{V}_1$ and $\mathcal{V}_2$, the nodes in graph G_1 and G_2 respectively, along with a similarity function $\phi : \mathbb{R}^d \times \mathbb{R}^d \rightarrow [0, 1]$, $\mathcal{H}$ is an LSH-2G family of hash functions such that the probability of two nodes $u, v \in \mathcal{V}_1$ hashing to the same bucket is equal to their similarity, and additionally the probability of two nodes $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$ hashing to the same bucket is equal to their similarity: $Pr[h(u) = h(v)] = \phi(\mathbf{f}_{G_1}(u), \mathbf{f}_{G_2}(v))$.*

We propose an LSH-2G family based on the standard measure of cosine similarity, and describe another family based on Euclidean distance in the supplementary material. We introduce SimHash-2G, a modified version of SimHash [3] that is based on LSH-2G described above. SimHash-2G chooses K randomly generated column-vectors $\{\mathbf{r}_1, \ldots, \mathbf{r}_K\} \in \mathbb{R}^d$ that follow the standard Gaussian distribution (i.e., K random hyperplanes). The LSH-2G family consists of K hash functions: $h_k(u) = sign(\mathbf{f}_{G_p}(u) \cdot \mathbf{r}_k)$. Each of these projects node u on either side of the random hyperplane $\mathbf{r}_k$ (positive or negative sign). The low-dimensional representation of each node u is its K-bit vector $\mathbf{f}_{proj}(u) = [h_1(u), \ldots, h_K(u)]$. For random hyperplane k, the probability of two nodes $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$ being mapped to the same bucket is $Pr[h_k(u) = h_k(v)] = 1 - \frac{\theta_{uv}}{\pi}$, where $\theta_{uv} = cos^{-1} \frac{\mathbf{f}_{G_1}(u)\mathbf{f}_{G_2}(v)}{\|\mathbf{f}_{G_1}(u)\|_2 \|\mathbf{f}_{G_2}(v)\|_2}$. The angle $1 - \frac{\theta_{uv}}{\pi}$ captures the proximity of u and v. SimHash-2G computes only the similarity for pairs of nodes according to our proposed construction, SKD-construction (see below).

If a hash function $h_i \in \mathcal{H}$ maps two nodes to the same bucket, that indicates that they *could* be similar, but there is some probability of error. The technique of amplification creates a new LSH family $\mathcal{G}$ with hash function g defined over the functions in $\mathcal{H} = \{h_1, h_2, \ldots, h_K\}$, in order to reduce that probability of error. A standard technique is AND-construction where $g(u) = g(v) \implies \forall i \; h_i(u) = h_i(v)$.

However, the AND-construction is too strict and may lead to many false negatives when finding node matchings. To ameliorate that we use the banding technique: (i) we split each feature vector into Z equal bands, and (ii) per band z, we apply a corresponding LSH-2G family $\mathcal{H}_z$ using AND-construction. In each band, a node can fall into only one bucket, and thus collides with nodes in that same bucket (potential matchings). To handle the observed *skewed* distribution of nodes to buckets and guarantee that each node will have some potential matchings, we introduce the notion of 'importance' of a node collision within a band and propose the SKD-construction.

Definition 2 (Importance σ_{tot} of node collision). *Given two nodes $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$, and an LSH-2G family $\mathcal{H} = \{h_1, \ldots, h_K\}$ s.t. $\forall j \; h_j(u) = h_j(v)$ (i.e., both nodes are mapped to bucket $b_{\mathcal{H}}$), we define the* importance *of their collision based on $\mathcal{H}$ as the inverse of the size of the corresponding bucket: $\sigma_{\mathcal{H}}(u, v) = \frac{1}{|b_{\mathcal{H}}|}$. The total importance score of a node pair collision over all bands and their corresponding LSH families $\mathcal{H}' = \{\mathcal{H}_1, \ldots \mathcal{H}_Z\}$ is defined as: $\sigma_{tot}(u, v) = \sum_{\mathcal{H} \in \mathcal{H}'} \sigma_{\mathcal{H}}(u, v) \cdot \mathbb{1}_{h_j(u) = h_j(v), \forall h_j \in \mathcal{H}}$.*

Intuitively, the importance of a collision is higher if a few nodes are mapped to a bucket, as the bucket has higher discriminative power. The notion of importance tackles the skewness that we observe in the mapped nodes in the graph alignment setting. Based on this definition, we propose the SKD-construction (where SKD stands for SKeweD).

Definition 3 (SKD-construction). *Given $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$, and LSH-2G families $\mathcal{H}' = \{\mathcal{H}_1, \ldots, \mathcal{H}_Z\}$, a new family $\mathcal{G}$ with hash function g is based on SKD-construction:*

$$\text{if } g(u) = g(v) \implies \sigma_{tot}(u, v) \in \text{TOP}_\alpha(u),$$

where $\text{TOP}_\alpha(u)$ is the set of top-α total importance scores $\sigma_{tot}(u, v')$ for $v' \in \mathcal{V}_2$, and α is the small factor that controls the density of G_S in Problem 2.

Intuitively, SKD-construction computes the pairwise similarities of nodes that collide often (but not always, like AND-construction) and have important collisions that manage to distinguish the nodes (i.e., it penalizes functions that lead to skewed results).

3.4 From Similarities to Matchings

As shown in Step 3 of Fig. 1, the hashing approach that we introduced returns a small number of high similarities between the nodes of graphs G_1 and G_2, giving us an $n_1 \times n_2$ sparse matrix $\mathbf{S}$ with node similarities. Here we provide ways to use the similarity information in $\mathbf{S}$ to find the node matchings or correspondences $\mathbf{M} \in \mathbb{Z}^{n_1 \times n_2}$:

• **Greedy matching**: Assuming that the higher the similarity score, the more likely two nodes are to match [15,22], we can greedily make independent decisions for the best match of each node in G_1 through a function $\chi : \mathcal{V}_1 \rightarrow \mathcal{V}_2$ s.t. $\chi(u) = \text{argmax}_v\{\mathbf{S}_{uv}\}$. Since nodes are matched independently, this is very efficient and parallelizable, but may match more than one node in graph G_1 to the same node in G_2. It is a preferred method for very large networks or networks of different sizes, and also when multiple potential matchings are desired. In the latter case, it can be trivially extended by updating the function $\chi()$ to return more top potential matchings (instead of only the best one).

• **Collective matching**: An alternative is to leverage existing approaches that find 1-to-1 matchings collectively, given a similarity matrix $\mathbf{S}$. In Sec. 5 we consider scalable options for doing so and study their trade-offs.

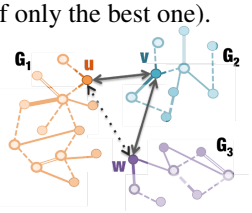


Fig. 3: Node matching consistency: If $u = v$ and $v = w$, then u should match to w (by transitivity).

4 HASHALIGN: Multiple Graph Alignment

In this section, we extend our HASHALIGN framework to multiple graph alignment and give a more formal definition which extends the relaxed 2-graph alignment problem.

*Problem 3 (**Relaxed multiple graph alignment**).* Given a set of graphs, $\mathcal{G} = \{G_1(\mathcal{V}_1, \mathcal{E}_1), \ldots, G_l(\mathcal{V}_l, \mathcal{E}_l)\}$, which may be (un)directed, (un)weighted and attributed / plain, we seek to efficiently **find** a sparse, weighted bipartite graph $G_{Sij} = (\mathcal{V}_i \cup \mathcal{V}_j, \mathcal{E}_{Sij})$ for each pair of graphs $<G_i, G_j>$, s.t. $\mathcal{E}_{Sij}$ has the potential matching pairs between their vertex sets and the weights describe how likely the nodes are to match.

Efficient Computation. The key insight to reduce computational cost is to use one of the l graphs as the 'baseline' graph G_C and align the remaining $l-1$ graphs with that, by applying the hash-based techniques of Section 3 to all the graphs in parallel. This approach avoids computing $O(l^2)$ pairwise graph alignments, instead leading to $l-1$ matching matrices $\mathbf{M}_{2C}, \ldots, \mathbf{M}_{lC}$ (w.l.o.g. we choose graph $G_C = G_1$ in our notation, but we will explain next the choice of G_C). Inspired by the idea of transitivity [21], which requires node matching consistency between pairs of graphs (Fig. 3), we efficiently infer all the remaining matching matrices $\mathbf{M}_{ij}$ (where $i, j \neq C$) via sparse matrix multiplications (Step 4 in Fig. 1): $\mathbf{M}_{ij} = \mathbf{M}_{iC} \cdot \mathbf{M}_{jC}^T$.

Choice of G_C. To reduce the induced alignment errors and their propagation to the inferred matchings, we propose the 'center' graph (i.e., the graph in $\mathcal{G}$ with the minimum total distance from the remaining graphs) as the baseline graph G_C (Step 2 in Fig.1):

$$\text{argmin}_C \sum_j d(G_C, G_j) = \sum_j ||\text{SIG}_{G_C} - \text{SIG}_{G_j}||_2,$$

where $d(G_C, G_j)$ is the distance between G_C and G_j, and SIG is a graph 'signature' (more details below). The intuition, which we have also found empirically, is that the center graph is as close as possible to all the other graphs, and thus the quality of the

Algorithm 1 HASHALIGN

Input: **(1)** $\mathcal{G}$=$\{G_1, G_2, \cdots, G_l\}$; **(2)** [OPT] Per graph i, node/edge attr. $\mathbf{A}_{n_{G_i}}^{n \times d_{a_n}}$ /$\mathbf{A}_{e_{G_i}}^{n \times n \times d_{a_e}}$
Output: A set of matching matrices $\{\mathbf{M}_{ij}\}$ for $i, j \in \{1, \ldots, l\}$

1: /* **STEPS 1&2**: NODE REPRESENTATION AND CENTER DISCOVERY */
2: **For** $G \in \mathcal{G}$ **do**
3: $\quad \mathbf{F}_G = \text{extractFeatures}(G, \mathbf{A}_G^{n \times d_{a_n}}, \mathbf{A}_G^{n \times n \times d_{a_e}})$ ▷ Sec. 3.2
4: $G_C = \text{findCenter}(\xi(\mathbf{F}_{G1}), \xi(\mathbf{F}_{G2}), \cdots, \xi(\mathbf{F}_{Gl}))$ ▷ Eq. (4) & aggregate function $\xi()$=SIG
5: /* **STEP 3**: HASH-BASED SIMILARITY (assuming q buckets in total) $\star$ /
6: $\{b_1, \ldots, b_q\}$ = SimHash-2G($\mathbf{F}_{G1}, \ldots, \mathbf{F}_{Gl}$) ▷ Sec. 3.3 (or EDHash-2G in Appendix B)
7: $\{\mathbf{S}_{1C}, \mathbf{S}_{2C}, \cdots, \mathbf{S}_{lC}\} = \text{computeSparseSimilarities}(b_1, \ldots, b_q)$ ▷ SKD-construction
8: /* **STEP 4**: NODE MATCHING */
9: $\{\mathbf{M}_{1C}, \mathbf{M}_{2C}, \cdots, \mathbf{M}_{lC}\} = \text{GREEDY or COLLECTIVE}(\mathbf{S}_{1C}, \mathbf{S}_{2C}, \cdots, \mathbf{S}_{lC})$ ▷ Sec. 3.4
10: **For** $i, j \in \{1, \ldots, l\}$ **do**
11: $\quad \mathbf{M}_{ij} = \mathbf{M}_{iC} \times \mathbf{M}_{jC}^T$ ▷ Sec. 4

center-based alignments are more precise (e.g., there exist more nodes that match, and the nodes are more similar) than the alignment based on any other graph in $\mathcal{G}$.

Since HASHALIGN is feature-based, we propose to leverage the same features in the graph signatures. A graph signature SIG_{G_j} can be created by applying an aggregate feature function $\xi()$ over its nodes. For example, a simple graph signature could contain average feature values: $\text{SIG}_{G_j} = \bar{\mathbf{f}}_{G_j} = \text{mean}\{\mathbf{f}_{G_j}(u_1), \mathbf{f}_{G_j}(u_2), \ldots, \mathbf{f}_{G_j}(u_{n_j})\}$. In our work, we collect the mean, median, standard deviation, skewness, and kurtosis of each of the d features to form the graph signature, giving us a $5d$-dimensional vector (shown in Step 1 of Fig. 1).

Hash-based similarity. After hashing all the feature-based node vectors of all the graphs in $\mathcal{G}$ as described in Sec. 3.3, we compute the similarity scores for possibly matching pairs of nodes according to the SKD-construction. We only compute the similarity between nodes in the center graph (right hand-side in the buckets in step 2 of Fig. 1) and nodes in the *peripheral*, or non-center, graphs (left hand-side).

Putting everything together. We propose HASHALIGN, a fast, hash-based, multiple graph alignment approach, which is described at a high level in Algorithm 1. The pictorial overview of HASHALIGN is given in Fig. 1, where Z =1 for simplicity. It consists of four main steps: (i) node representation, (ii) 'center' graph identification, (iii) hash-based similarity, and (iv) node matching. We note that in line 7 of Algorithm 1, SimHash-2G is applied to l graphs simultaneously, i.e., it hashes their nodes in parallel.

Computational Complexity of HASHALIGN. Our framework makes two main substitutions for computational savings. First, it replaces full pairwise similarity computations that are quadratic in the number of nodes with hashing in only $O(K \cdot n_p \cdot d)$ time for graph G_p with n_p nodes, if we use K hash functions on d-dimensional feature vectors. Second, it replaces all $\binom{l}{2}$ pairwise network alignments with only $l-1$ pairwise network alignments to a center graph (chosen in $O(l^2 \cdot d)$ time), inferring the remainder with sparse matrix multiplications. More details are given in the supplementary material.

5 Experimental Analysis

In this section, we seek to answer the following questions: **(1)** How robust is our framework compared to baselines for different levels of noise in the graphs (both in the structure and node/edge attributes)? **(2)** How could HASHALIGN help existing alignment

methods perform better and how could these help our method? **(3)** How do our methods scale when aligning multiple graphs collectively? We answer these questions on three datasets, described in Table 2. We also include additional experiments, such as a *sensitivity analysis* of HASHALIGN to different parameters, in the supplementary material.

Baselines. We consider 3 baseline methods commonly used in the literature: `NetAlign` [4], `Final` [22], and `IsoRank` [19]. We compare their performance against our method, HASHALIGN, where we infer alignments greedily from the hashing-based node similarities. The baselines accept a matrix $\mathbf{L}$ representing prior alignment information between the nodes of the original graphs. By default, we provide a thresholded similarity matrix based on the node attributes to assure good performance based on the attribute information, even for the baselines that are not formulated for it (`NetAlign` and `IsoRank`). We also consider two variants of HASHALIGN, namely HASHALIGN-NA and HASHALIGN-FN, which instead provide `NetAlign` and `Final` respectively with the similarity matrix $\mathbf{S}$ from HASHALIGN, so that these methods infer alignments from the node similarities as the final step of HASHALIGN (Sec. 3.4).

Data. We evaluate our proposed algorithms on three datasets along with the synthetic data that we generated from them (via permutations and added noise, as in [15,22]). Formally, given a graph G_1 with adjacency matrix $\mathbf{A}$, we create a noisy graph G_2 with matrix $\mathbf{B} = \mathbf{PAP}^\top$ (i.e., a permutation of itself), where $\mathbf{P}$ is a randomly generated permutation matrix (i.e., with one nonzero entry per row / column). Synthetic noise is applied to both graph structure and labels throughout our experiments to simulate real-world scenarios where the graphs are matchable but reasonably different. The noise level p indicates that with probability p, Gaussian noise with $std = 1$ is added to an edge weight; a binary edge label is flipped; or a categorical node / edge label is changed to another value.

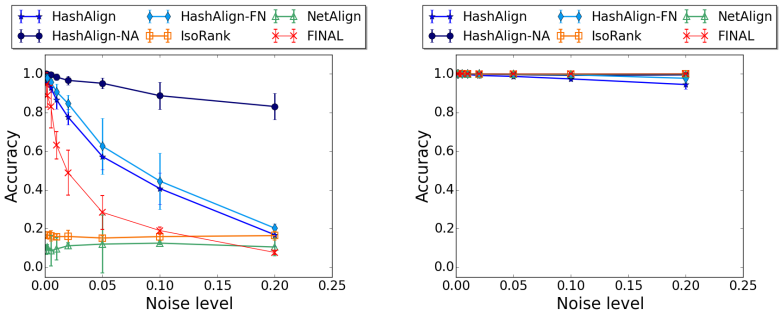
Table 2: Description of real datasets.

Datasets	# Nodes	# Edges	Graph Type	Labels	Description
Connectome [1]	941	9,622	Undirected	-	fMRI-inferred graphs
E-mail [2]	1,133	5,451	Undirected	5	email communications
DBLP [22]	42,252	210,320	Undirected	1	coauthorship network

Evaluation Metric. Following the literature, we compute the alignment accuracy as $\frac{\#\text{ correct matchings}}{\#\text{ total matchings}}$, where the total number of matchings between G_i and G_j is equal to the minimum number of nodes between the two graphs, $min\{n_i, n_j\}$ (which is the number of available ground truth matchings.)

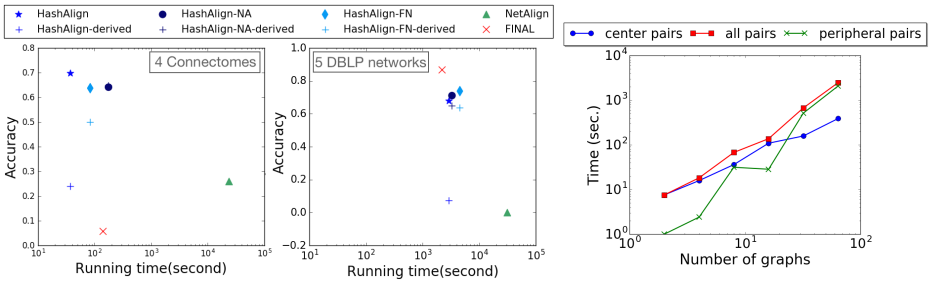
Experimental setup. In our experiments, we used the following structural attributes: degree, node betweenness centrality, PageRank, egonet degree, average neighbor degree, and egonet connectivity. We chose these attributes since they are robust to noise and thus, they are expected to help with aligning two graphs (which are often seen as noisy permutations of each other). More details are given in the supplementary material.

To test the ability of HASHALIGN to incorporate different kinds of features, we also generate synthetic node/edge attributes, if none are available in the real data. For each noise level p, we generate 3 pairs of graphs and report the average accuracy (along with a 95% confidence interval). HASHALIGN is implemented in Python2.7, and the structural feature extraction is based on SNAP [16]. We ran the experiments on Intel(R) Xeon(R) CPU E5@ 3.50GHz and 256G RAM.



(a) E-mail data with 1 node attribute **(b)** E-mail data with 5 node attributes

Fig. 4: E-mail dataset (experimental results on other datasets are similar): Effectiveness w.r.t noise on both attributes and the graph structure. Methods based on the HASHALIGN framework achieve highest accuracy, particularly with limited node attribute information.



(a) Runtime vs. accuracy for multiple graph alignment on four unattributed (on the left) and five attributed (on the right) graphs.

(b) Scalability of HASHALIGN.

Fig. 5: (a) HASHALIGN has stable efficiency across different kinds of networks. `Final` is fast, but its accuracy is subject to whether node/edge labels exist. (b) HASHALIGN scales linearly in terms of alignment with the center graph.

5.1 HASHALIGN: Accuracy and Runtime

Two-Graph Alignment. We conducted alignment of pairs of graphs on all the datasets (with G_1 the real graph and G_2 its noisy permutation at noise level p, generated as described above) and got consistent results. Only the result from the E-mail network is shown for brevity. With only 1 binary attribute (Fig. 4a), `NetAlign` and `IsoRank` perform poorly because the similarity matrix $\mathbf{L}$ built using just 1 attribute is not informative enough. However, these methods work significantly better as part of our framework, as indeed all HASHALIGN variants achieve superior results as noise levels increase. We note that in Fig. 4b, HASHALIGN-NA achieves perfect results in the presence of 5 node attributes. However, all methods perform essentially perfectly, since with abundant node attribute information, the alignment problem becomes much easier.

Multiple Graph Alignment. We evaluate HASHALIGN against other methods for multiple graph matching. The two datasets we experimented with include five connectome networks [1] without any labels, and four DBLP co-author networks extracted from the whole DBLP dataset following the settings in [22]. The DBLP coauthor networks have one categorical label, describing the most frequent conference that an author attends. Both experiments are conducted with $p = 2\%$ noise.

Figure 5a shows how different methods perform in terms of efficiency and accuracy. When there is no label information to help guide the alignment process (i.e.,

in the case of connectomes), HASHALIGN achieves best accuracy with short running time for peripheral-center graph pairs alignment, followed by HASHALIGN-FN and HASHALIGN-NA. As for the DBLP networks, since the label with 29 *distinct* values is very discriminative, `Final` can achieve very good efficiency, while HASHALIGN and its variants also have comparable performance. However without node labels, `Final` matches less than 10% of all node pairs, which can be boosted to over 60% if we feed it the hash-based similarity matrices of HASHALIGN. Pairwise graph alignment is the most computationally expensive for large numbers of graphs (see Fig. 5b), but for fewer graphs of the sizes in Fig. 5a, computing all pairwise alignments yields the highest accuracy, and is thus our recommendation if computational resources are not an issue. However, center graph alignment (the '*derived*' versions of HASHALIGN variants) often still outperforms the baselines, and in some cases matches the accuracy of the full pairwise comparisons (e.g., HASHALIGN-NA on the connectome data.)

These results clearly show that HASHALIGN leads to significant improvement over existing methods with regard to both accuracy and runtime. In summary, we see that on average, HASHALIGN (including its variants) are $2\times$ faster and 10 to 20% more accurate than the baselines in pairwise alignment, and $2\times$ faster while up to 50% more accurate in multiple graph alignment. However, these existing methods may have their place within our framework (see Step 4 of Fig. 1), where they may be used to accurately infer alignments from the hashing-based node similarities.

5.2 HASHALIGN: Scalability

A scalability test is conducted to verify that the proposed method scales as the number of graphs grows. For this purpose, we generated up to 64 synthetic graphs from the aforementioned connectome network with $p = 0.02$ noise, Z$= 2$ and $K = 40$. As shown in Fig. 5b, HASHALIGN's runtime scales linearly in terms of alignment with the center graph. The runtime for deriving the peripheral graph alignments (i.e., w/o the center graph) using sparse matrix multiplication scales subquadratically, as the slope indicates. We omitted the runtime for feature extraction as it is linear on the number of graphs, and does not contribute much to the runtime.

6 Conclusions

We study the problem of multiple graph alignment and propose HASHALIGN, an intuitive, fast and effective similarity-based approach that readily handles any type of input graph. Our method adapts LSH to graph alignment, with a new construction technique and an appropriate node-ID-invariant node representation for this task. Leveraging the rule of matching transitivity, it scales up to many graphs while avoiding solving the expensive alignment task for each pair of graphs separately. Our experiments on real data (incl. sensitivity analysis in the supplementary material) show that HASHALIGN can stand alone as a multi-network alignment tool or be combined with existing methods that require a small set of possible matchings as input. In most cases, it is more accurate, more robust to noise, and/or faster than the baselines. Our work suggests that hashing is a promising direction for scaling up network alignment. Future work could include extending HASHALIGN to use learned node representations specifically designed for multi-network problems, as in the very recent work of [12]. Here one challenge would be devising suitable graph signatures for efficient multiple graph alignment.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under Grant No. IIS 1743088, and the University of Michigan. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

1. COBRE. http://fcon_1000.projects.nitrc.org/indi/retro/cobre.html, 2012.
2. Konect: Koblenz network collection. http://konect.uni-koblenz.de/networks/, 2016.
3. A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*. IEEE, 2006.
4. M. Bayati, D. F. Gleich, A. Saberi, and Y. Wang. Message-Passing Algorithms for Sparse Network Alignment. *ACM TKDD*, 7(1):3:1–3:31, Helen Martin 2013.
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. I. Brugere, B. Gallagher, and T. Y. Berger-Wolf. Network structure inference, A survey: Motivations, methods, and applications. *CoRR*, abs/1610.00782, 2016.
7. M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pages 253–262. ACM, 2004.
8. 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*, 2008.
9. S. Duan, A. Fokoue, O. Hassanzadeh, A. Kementsietsidis, K. Srinivas, and M. J. Ward. Instance-based matching of large ontologies using locality-sensitive hashing. In *ISWC*. Springer, 2012.
10. P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *arXiv preprint arXiv:1705.02801*, 2017.
11. M. Heimann and D. Koutra. On generalizing neural node embedding methods to multi-network problems. In *KDD MLG Workshop*, 2017.
12. M. Heimann, H. Shen, and D. Koutra. Node representation learning for multiple networks: The case of graph alignment. *arXiv preprint arXiv:1802.06257*, 2018.
13. K. U. Khan, W. Nawaz, and Y. K. Lee. Set-based unified approach for attributed graph summarization. In *IEEE BDCC*, Dec 2014.
14. D. Koutra and C. Faloutsos. Individual and collective graph mining: Principles, algorithms, and applications. *Synthesis Lectures on Data Min. Knowl. Discov.*, 9(2):1–206, 2017.
15. D. Koutra, H. Tong, and D. Lubensky. Big-Align: Fast Bipartite Graph Alignment. In *ICDM*. IEEE, 2013.
16. J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM TIST*, 8(1):1, 2016.
17. E. Malmi, S. Chawla, and A. Gionis. Lagrangian relaxations for multiple network alignment. *Data Mining and Knowledge Discovery*, pages 1–28, 2017.
18. T. Safavi, C. Sripada, and D. Koutra. Scalable hashing-based network discovery. In *ICDM*. IEEE, 2017.
19. R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS*, 105 35:12763–8, 2008.
20. L. Zager and G. Verghese. Graph Similarity Scoring and Matching. *Applied Mathematics Letters*, 21(1):86–94, 2008.
21. J. Zhang and P. S. Yu. Multiple anonymized social networks alignment. In *ICDM*. IEEE, 2015.
22. S. Zhang and H. Tong. Final: Fast attributed network alignment. In *KDD*. ACM, 2016.

Supplementary Material

A Additional Related Work

In this Appendix, we provide an overview of some additional related works, which complement the discussion in Section 2. Specifically, we discuss some more existing formulations and methods for solving the problem of aligning pairs of graphs. We also describe some existing measures of node similarity in more detail, as this concept is relevant to our work.

Pairwise Graph Alignment. One of the most common formulations for aligning two graphs G_1 and G_2, with adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, is to find a matrix $\mathbf{P}$ that permutes the nodes in $\mathbf{A}_1$ so that it best matches $\mathbf{A}_2$: $\min_{\mathbf{P}} ||\mathbf{P}\mathbf{A}_1\mathbf{P}^T - \mathbf{A}_2||_F^2$, where $|| \bullet ||_F$ is the Frobenius norm of the corresponding matrix. The original requirement is that $\mathbf{P}$ is a permutation matrix (i.e., a square binary matrix with exactly one entry 1 per row and column, and 0s elsewhere), but several convex or concave relaxations to this problem have been proposed, such as $\mathbf{P}$ being a (doubly) stochastic matrix [29].

The Hungarian method [28] is a well-known spectral method for solving this formulation, but it requires graphs of the same size n and is computationally expensive, $O(n^3)$, and thus inapplicable to large-scale networks. Other suggested solutions vary: clustering algorithms [26], graph edit distance [27], iterative HITS-inspired [20], and more. Like the Hungarian method, however, most of these methods align pairs of graphs and do not necessarily scale well to large-scale graph alignment; in general, they are designed for more limited uses cases than HASHALIGN.

Node Similarity. There are many different ways of computing similarity scores between nodes, most of which operate on a single network (e.g., Personalized Random Walks [23], SimRank [24], and Belief Propagation [25]). Across two graphs, [15] and [4] use (weighted) degree-based node similarities to initialize the alignment matrix. Zager and Verghese [20] compute the similarity between two graphs via a recursive method that couples the similarity scores of nodes and edges. Unlike our work, the alignment in [20] is found *after* computing the similarity between *all* pairs of nodes, as well as *all* pairs of edges.

B Euclidean Distance as an LSH family for HASHALIGN

In our main method (Sec.3.3), we introduce an LSH family based on cosine similarity. Here we also introduce EDHash-2G, which is based on Euclidean distance (ED) as the measure of (dis)similarity and is preferred when the dimensionality of the node vectors d is quite high. However, in our experiments, we report results using SimHash-2G since it ties or outperforms EDHash-2G (Sec. E).

At a high level, the corresponding hash functions compute the projections of the input vectors on random lines. More formally, given a real number w, the k^{th} hash function is constructed so that it maps a real-valued vector in $\mathbb{R}^d$ into an integer (which corresponds to a 'segment' ID):

$$h_{\mathbf{r}_k, b_k}(u) = \lfloor \frac{\mathbf{f}_{G_p}(u) \cdot \mathbf{r}_k + \beta_k}{w} \rfloor,$$

where $\mathbf{r}_k$ is the corresponding column-vector in $\mathbb{R}^d$ with entries chosen independently from a Gaussian distribution (which is 2-stable [7]) and β_k is positive real number chosen uniformly from $[0, w]$. If the nodes $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$ map to the same w-long 'segment', they are considered similar, and their similarity is computed basd on the SKD-construction. The ED between two nodes is then proportional to the probability of them falling into the same 'segment'. Among the ways to convert a distance to similarity, we choose $sim = (1 + ED)^{-1}$, which gives bounded similarity scores in $[0, 1]$.

C Computational Complexity of HASHALIGN

In this section, we provide some more detail on the computational complexity of our proposed framework, HASHALIGN. The analysis is divided by step: **(1)** During the **node representation** phase, HASHALIGN extracts d-dimensional structural and node/edge attribute features in each of the l graphs. The complexity of this phase depends on the selected features (e.g., for G_p, the degree computation takes $O(m_p)$, while the betweenness centrality $O(n_p \cdot m_p)$). **(2)** The cost of finding the **center graph** from the l graphs to align is $O(l^2 \cdot d)$. The remaining steps can be parallelized per graph, so we report complexity analysis in terms of the size of the largest graph. **(3)** The hashing-based similarity computation consists of two parts: For K hashing functions and bucket size $|b_i|$, the complexity of **hashing** is $O(K \cdot \max_p n_p \cdot d)$ for graph G_p with n_p nodes. **(4)** For node **matching**, the similarity matrix must be constructed by computing the σ_{tot} scores over pairs of nodes in the same bucket in time $O(\max_i |b_i|^2)$. Then the greedy approach finds the maximum element for each of the n_p nodes in at most $O(n_p^2)$ time. In the case of multiple network alignment, we replace the expensive computation of pairwise similarity score and simply perform sparse matching matrix multiplications (with the center graph).

D Choice of Node Features for HASHALIGN

Since we believe that the inconsistency between graphs could be considered noise, the more stable the features are in the presence of noise, the better. Thus, we choose our node features after seeing that their distributions have little difference before and after adding noise to the graph. An example of a feature that is robust and one that is sensitive to noise is given in Fig. 6.

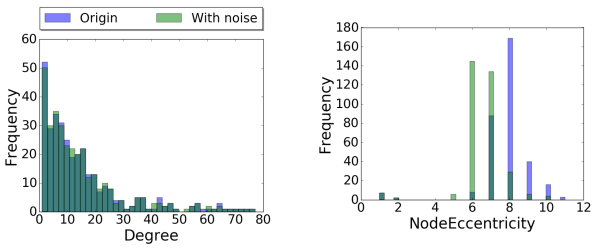
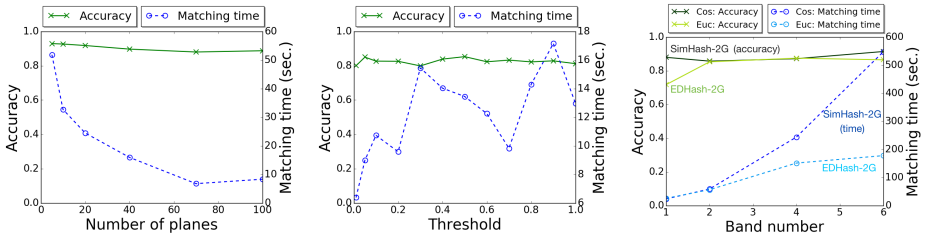


Fig. 6: Degree is robust, but eccentricity is sensitive to noise.

E HASHALIGN: Sensitivity Analysis

We provide additional analysis in this section to find out the patterns behind the various combinations of parameters in the synthetic graphs and LSH functions. The tuning



(a) Number of random hyperplanes K in SimHash-2G.

(b) Threshold α used in computing similar pairs.

(c) Efficiency with respect to the number of bands Z.

Fig. 7: Sensitivity analysis of HASHALIGN to parameters.

parameters for SimHash-2G / EDHash-2G are: the number of hash functions (or hyperplanes) K, the threshold α for computing the similarity of node pairs, and the number of bands Z. In Fig. 7, we present the results of a series of comprehensive experiments which vary these parameters. Based on these, we have the following observations:

- As shown in Fig. 7b, a higher threshold α leads to similar accuracy but potentially longer runtime. Computing pairwise similarity between 10 and 20% of the total number of nodes gives the best trade-off between runtime and accuracy.
- The number of node attributes or the length of feature vectors determines the performance of LSH. Given more features, LSH will generate more buckets, which decreases the runtime of filtering colliding nodes pairs and increases accuracy.
- In terms of choice of different types of LSHs, SimHash-2G (based on cosine similarity) tends to yield better randomness than EDHash-2G, while keeping runtime low. This is why we chose it for our other experiments. In Fig. 7c, we used segments of length $w = 4$ for EDHash-2G.
- The fewer bands, the better: more bands lead to roughly quadratic growth in runtime, since (1) there are more buckets from different bands to merge, and (2) there are fewer features in each band, making it easier for two nodes to collide (Fig. 7c).

References

23. T. H. Haveliwala. Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search. *TKDE*, 15(4):784–796, 2003.
24. G. Jeh and J. Widom. SimRank: A Measure of Structural-Context Similarity. In *KDD*, pages 538–543. ACM, 2002.
25. D. Koutra, T.-Y. Ke, U. Kang, D. H. Chau, H.-K. K. Pao, and C. Faloutsos. Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms. In *ECML PKDD*, pages 245–260, 2011.
26. H. Qiu and E. R. Hancock. Graph Matching and Clustering Using Spectral Partitions. *IEEE TPAMI*, 39(1):22–34, 2006.
27. 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.
28. S. Umeyama. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE TPAMI*, 10(5):695–703, 1988.
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.