Addressing Computational Bottlenecks in Higher-Order Graph Matching with Tensor Kronecker Product Structure

Charles Colley, Huda Nassar, and David F. Gleich

Abstract—Graph matching, also known as network alignment, is the problem of finding a correspondence between the vertices of two separate graphs with strong applications in image correspondence and functional inference in protein networks. One class of successful techniques is based on tensor Kronecker products and tensor eigenvectors. A challenge with these techniques is memory and computational demands that are quadratic (or worse) in terms of problem size. In this manuscript we present and apply a theory of tensor Kronecker products to tensor based graph alignment algorithms to reduce their runtime complexity from quadratic to linear with no appreciable loss of quality. In terms of theory, we show that many matrix Kronecker product identities generalize to straightforward tensor counterparts, which is rare in tensor literature. Improved computation codes for two existing algorithms that utilize this new theory achieve a minimum 10 fold runtime improvement.

Index Terms—Graph Matching, Network Alignment, Image Correspondence, Tensor Eigenvectors, Tensor Kronecker Product

1 INTRODUCTION

GRAPH matching is an old problem that continues to find new applications [1], [2]. Standard graph matching techniques involve finding a one-to-one correspondence between the vertices of the two graphs designed to align the edges of the graphs. In computer vision the primary applications are based on vertex correspondence problems such as automatic panoramic stitching [3], [4], shape and local geometry matching [5], [6], [7], [8], n-dimensional symmetry analysis [9], and learning textures [10]. In bioinformatics, network alignment is used to generate orthologous protein pairs in different species [11], [12], [13], [14], [15], [16] and in genetic pathway analysis [17] (see two other surveys, such as [18], [19]). In ontology analysis, database schema matching, and translation, matching algorithms are common [20], [21], [22].

Graph matching problems, posed as optimization problems, can be combined with any number of customizations including adding weights to guide the alignments and restrictions to prohibit known erroneous matches. Algorithms for the problem date to the 1950’s due to the relationship with the quadratic assignment problem and facility location [24], [25], [26]. Methods to address the problem range from relax and round on integer problems [25], [27], to Lagrangian relaxation techniques [28], to well-motivated heuristics [11], [29], and many other types of techniques [13], [14]. Recent eigenvector-inspired spectral approaches [11], [14], [30], [31] have been among the most scalable and versatile.

Given that standard graph matching is one of a number of possible solutions for the demands of various applications, we note that there are many potential improvements to the frameworks and heuristics. For instance, an emerging practice is to align graphs by seeking matches among the higher order structures, or motifs, present in the graph – such as triangles. These methods and frameworks then seek to find a vertex correspondence that aligns many of these higher-order structures. The authors of [8], [12], [32] all derive similar methods that involve using tensor eigenvectors as guidance for an alignment. Earlier proposals for similar ideas include [20]. All of these methods fit into the growing literature on how higher-order thinking helps graph algorithms [33], [34], [35], [36], [37].

The primary difficulty of these methods is the combinatorial explosion of their complexity. In an ideal scenario, their runtime is proportional to the number of motifs present in both graphs. This can easily be super-linear in the size of the input graph as most real-world networks have many more triangles than edges. In this research we’re specifically interested in two higher order algorithms: TAME [12] and HOFSAM [38], both of which show promising results for higher-order alignment but have steep computing requirements. For instance, in TAME, the memory required for aligning two graphs with \( n \) vertices and \( m \) motifs is quadratic in the number of vertices, i.e. \( n^2 \). This is large, but feasible for graphs up to 50k vertices. (Increasing memory capacities allow even larger problems.) Each iteration of the algorithm in TAME also requires quadratic computation in terms of the number of motifs, i.e. \( m^2 \) computation, where \( m \) can easily be one million even when \( n \) is around 10,000. Put another way, a graph can easily have one million triangles even when it has 10k vertices, and thus, runtimes for such higher-order alignment problems are long. This remains true even using the optimized sophisticated implicit algorithms as is done in [12], [32], [38].

The long runtimes for both TAME [12] and HOFSAM [38] arise because both use Kronecker products between tensors in the main iteration. Tensor Kronecker products are the straightforward generalizations of the matrix Kronecker product and were originally introduced in [39]. Subsequently,
they have been used in a variety of scenarios [12], [38], [40], [41], [42]. In the matrix case, the Kronecker product has widely known identities that enable various efficient operations. For instance, the eigenvectors of a Kronecker product between two matrices are Kronecker products between pairs of eigenvectors from each matrix. These results have been deployed for edge-based network alignment to greatly accelerate the algorithms [30], [31]. The theory we develop in this paper shows that the natural tensor networks from the Biogrid repository [44] collected (§4). We evaluate the new algorithms and implementations of higher-order alignment algorithms including TAME and HOFASM. These new algorithms show at least a factor 10 runtime improvement with no loss in performance (§4).

The remainder of our paper formally establishes these results. It begins with a brief overview of our preliminaries for hypergraphs and tensors along with definitions of our specific graph alignment problems (§2) and how Kronecker structures arise in two specific instances. From there we will move on to our three primary contributions, (1) a generalization of some core identities for the matrix Kronecker product (§3), (2) a novel extremal Z-eigenbound for tensor Kronecker products (§3.3), and (3) the application of this theory to two graph matching applications TAME and HOFASM (§4). We evaluate the new algorithms and implementations by recreating the original experiments and supplementing with additional tests as needed, in §5. For HOFASM we use the synthetic experiment framework of [43]. For our TAME methods we test aligning protein-protein interaction (PPI) networks from the Biogrid repository [44] collected by the Local vs. Global Network Alignment (LVGNA) [16] and TAME [12] projects, with additional random geometric and Erdős-Rényi graphs. Our code is available (see §5) and we have attempted to make our results as reproducible as possible by including the experiment driver codes as well.

2 Definitions & Preliminaries

2.1 Notation

Much of the notation in our paper in standard and we review it for clarity. Graphs consist of vertex sets $V$ and edge sets $E$. They can be weighted with a positive weight for each edge and an implicit zero weight for each non-edge, or they can be unweighted in which case edges have an implicit weight of 1 and non-edges have weight 0. All of the graphs we consider are undirected. The adjacency matrix then corresponds with a symmetric matrix of edge weights for a fixed order of the vertices. We bold matrices and vectors as in $A$ and $x$. The vector of all ones of length $n$ is $\mathbb{1}_n$.

2.2 Tensor notation and tensor eigenvectors

Tensors are bold and underlined, as in $A$. We use a bold tuple of indices $i = (i_1, \ldots, i_k)$ to denote each element of the tensor $A(i) = A(i_1, \ldots, i_k)$ [15] $\S12.4.2$. The dimensions of a tensor are called modes, and so a 2-mode tensor is a matrix. We call a tensor symmetric if the entries are the same in all permutations of the indices. When the dimension for each mode of a tensor is the same, we call this tensor cubical. Let $[n]^k$ be the set of all multi-indices of length $k$ for $1 \ldots n$ in each dimension.

Tensor eigenvectors involve two operations. Let $A$ be a $k$-mode, cubical tensor of dimension $n$. First, the polynomial

$$\sum_i A(i)x(i_1)x(i_2) \cdots x(i_k)$$

is written as $A^k x$, and generalizes the quadratic $x^TAx = \sum_{ij} A_{ij}x_ix_j$ which we could write as $A^2 x$. The second is the tensor-vector product, written

$$Ax^{k-1}$$

and defined as $\sum_i A(i)x(i_2) \cdots x(i_k) I(:, i_1)$, where $I(:, i_1)$ is the $i_1$th column of the identity matrix. The result is a vector, and note that $x^TAx^{k-1} = A^k x$.

There are a variety of notions of tensor eigenvectors [46]. We use the Z-eigenvector of Qi [47] or the $i_2$ eigenvectors of Lim [48]. A Z-eigenpair of a tensor $A$, is a pair $(\lambda, x)$ with $\lambda$ scalar and $x$ an $n$-vector, where

$$Ax^{k-1} = \lambda x \quad \|x\|_2 = 1.$$

Equivalently, a tensor Z-eigenpair is a KKT point of the optimization problem

$$\maximize \ Ax^k \quad \text{subject to} \quad \|x\|_2 = 1.$$

There is an exponential number of tensor eigenvectors [49] as the number of modes $k$ grows. For symmetric tensors, the eigenvectors can be computed via the Laserre hierarchy and convex programming [50], although these techniques do not scale to large tensors. The higher-order power method (HOPM) [51], symmetric shifted higher-order power method (SSHOPM) [52], its generalizations [53], and dynamical systems [54] are among the scalable ways to compute tensor eigenvectors. Although these scalable methods may not have the most satisfactory theoretical guarantees, they are practical and useful.

2.3 Hypergraphs from graphs via motifs

A motif is simply a graph – usually small, like a triangle – and an instance of a motif in a graph is simply an instance of an isomorphic induced subgraph [55]. From any graph we can induce a $k$-regular hypergraph $H$ by identifying hyperedges with the presence of motifs with $k$ vertices [12], [56], [57], [58]. For instance, suppose the motif is a triangle. Each instance of the motif – i.e. each triangle – gives us one hyperedge. Then the full edge set of the hypergraph involves enumerating all the instances of the motif $M$. This can be computationally demanding for complicated motifs, but is fast for simple motifs like triangles and small cliques [59]. Analogously to the adjacency matrix, we use an adjacency tensor $A$ to denote the presence of these motifs, or equivalently, hyperedges. Formally,

$$A(i_1, \ldots, i_k) = \begin{cases} 1 & \text{if nodes } i_1, \ldots, i_k \text{ form motif } M, \\ 0 & \text{else.} \end{cases}$$

Each permutation of the indices corresponds to a different orientation of the motif $M$. Consequently, the adjacency tensor of a hypergraph is a symmetric, cubical tensor for the motifs we consider.
2.4 Kronecker products of tensors and vectorization

The Kronecker product $\otimes$ between matrices arises from treating the pair of matrices $B$ and $A$ in $Y = AXB^T$ as a linear operator from $X$ to $Y$. (The order arises from how the matrix $X$ is linearized, see more below.) The way we present the definition of $B \otimes A$ for matrices involves a more complicated seeming interleaving of indices from $A$ and $B$, but this will enable a seamless generalization to tensors. Let $i,i',j,j'$ represent a linearization, or vectorization, of the pair $i,i'$ to a single index. For instance, if $i$ ranges from 1 to $m$, then $i,i'$ represents the linearized index $m(i'-1)$ where we have vactorized by the first index. This joint index notation is exactly the vectorization:

$$\text{vec}(X)[i,i'] = X(i,i'),$$

where we use the “matrix-to-vector” operator vec, which converts matrix-data into a vector by columns. We extend this definition to an interleaving of pairs of indices as in $(i,j)(i',j') \rightarrow (i,i',j,j')$. This gives us the matrix Kronecker product

$$(B \otimes A)[i,i',j,j'] = A(i,j)B(i',j').$$

Using this notation we have for $Y = AXB^T$, let $y = \text{vec}(Y)$, $x = \text{vec}(X)$ and

$$y[i,i'] = \sum_{j,j'} (B \otimes A)[i,i',j,j']x[j,j'] = \sum_{j,j'} A(i,j)B(i',j')x[j,j'].$$

The nice thing about this notation is it gives us a seamless way to generalize to tensors. Given a $k$-tuple of indices $i$ and $i'$, we have $i,i' = (i_{1},i_{1}',i_{2},i_{2}',\ldots,i_{k},i_{k}')$. Then if $A$ and $B$ are two $k$-mode cubical tensors, we have the element-wise definition

$$(B \otimes A)[i,i',j,j'] = A(i)B(i').$$

Equivalently, we can define this in terms of single element tensors. Let $E_{i}$ be a tensor with a 1 in the $i$ entry and zero elsewhere. Then $A = \sum_{i} A(i)E_{i}$ and $B = \sum_{i'} B(i')E_{i'}$, and

$$B \otimes A = (\sum_{i} B(i')E_{i'}) \otimes (\sum_{i} A(i)E_{i}) = \sum_{i,i'} A(i)B(i')E_{i'} \otimes E_{i}.$$

Using the element-wise definition above $E_{i'} \otimes E_{i}$ only has a single non-zero in the $i,i'$ entry.

2.5 Higher-order graph alignment

The higher-order graph alignment problem we study is defined in terms of a matching between the vertices of two graphs $[12,32,38]$. For a pair of graphs $A$ and $B$ we characterize a matching between their vertex sets as a matrix.

**DEFINITION 1 (Matching Matrix)** Let $A$ and $B$ be two graphs of size $m$ and $n$ respectively, then we define the matching matrix $X \in \{0,1\}^{m \times n}$ such that

$$X(i,i') = \begin{cases} 1 & \text{if } i \in V_{A} \text{ is matched to } i' \in V_{B} \\ 0 & \text{else} \end{cases}$$

and $X1_{m} \leq 1_{m}$ and $X^{T}1_{m} \leq 1_{n}$.

We find it easiest to understand higher-order graph alignment after introducing pairwise, edge-based graph alignment $[1,11,60]$. Consider a pair $(i,j)$ of vertices in graph $A$ and a pair $(i',j')$ of vertices in graph $B$. Then we assign a similarity score $S(i,j,i',j')$ to this set of four indices. In the simplest case, we can set $S(i,j,i',j') = 1$ when $(i,j)$ corresponds to an edge in $A$ and $(i',j')$ corresponds to an edge in $B$. In the framework of Feizi et al. [29], they assign a similarity $S(i,j,i',j')$ that depends on the three cases: both edges exist, only one edge exists, and both edges do not exist. More complex scenarios are also possible [9]. The goal of the alignment is then to find a matching matrix that optimizes

$$\text{maximize } \sum_{i,j} \sum_{i',j'} S(i,j,i',j')X(i,i')X(j,j') \text{ subject to } X \text{ is a matching}.$$

This objective rewards choices where vertex $i$ matches to $i'$ and $j$ matches to $j'$ and the similarity score $S(i,j,i',j')$ is high. The framework can be further customized with various constraints on the matching such that some entries are never matched [60] or such that there is a prior score $W(i,i')$ on how likely vertex $i$ and $i'$ can be matched. We often find it convenient to write this objective as

$$\text{maximize } \text{vec}(X)^{T}V \text{ vec}(X) = x^{T}Vx \text{ subject to } X \text{ is a matching}$$

where we convert the similarity tensor $S$ into a matrix $V$ indexed conformally with the vec operator. This tensor to “operator for vec($X$)” transformation is something we repeatedly use and write it as

$$S \leftrightarrow V \text{ means } \sum_{i,j} \sum_{i',j'} S(i,j,i',j') = \sum_{i,i'} V[i,i'] \otimes V[i,i'].$$

There are useful connections to the Kronecker product discussed in §2.6 where many choices for $S$ give rise to tensors $V$ with Kronecker structure. Another utility of this vec-form is that it makes the eigenvector-heuristic inspiration clear because eigenvectors optimize the Rayleigh quotient $x^{T}Ax$.

The idea in higher-order graph alignment is that we move from edges to motifs. For higher-order graph alignment, we suppose we are given a similarity tensor $S$ where entries can be indexed using a pair of tuples $i$ for a motif in graph $A$ and $i'$ for a motif from graph $B$. The value $S(i,i')$ indicates the similarity of the motif at indices $i$ to the motif at indices $i'$. This subsumes the edge-based alignment framework because i could have just been the pair $(i,j)$. Again, there are various choices for entries $S(i,i')$. In TAME [12], this similarity is 1 if there is a triangle at both $i$ in $A$ and $i'$ in $B$. In HOFA$\text{S}[38]$, the entry depends on the local features of an image. A simple form of higher-order graph alignment problem is then to optimize

$$\text{maximize } \sum_{i,j} [S(i,i')X(i,j)]X(i,j)X(i,j)X(i,j) \text{ subject to } X \text{ is a matching}.$$
Again, this framework is highly flexible. For simplicity, we define the following objective function that will guide our subsequent research.

**DEFINITION 2 (Global Graph Alignment)** Fix graphs $A$ and $B$ of sizes $m$ and $n$ respectively, an $m \times n$ prior weight matrix $W$, and a motif $M$ with $k$ vertices. Let $S$ be a $2k$-mode similarity tensor where the $S(i, i')$ entry denotes the similarity between the motifs induced by the vertices $i$ in graph $A$ and $i'$ in graph $B$. Then we wish to find a matching $X$ between the vertices in $A$ to the vertices in $B$ that optimizes

$$
\max_{X} \sum_{i,i'} [S(i, i')X(i_1, i'_1)X(i_2, i'_2)\cdots X(i_k, i'_k)] + \sum_{i,i'} W(i, i')X(i, i')
$$

subject to $X$ is a matching.

Equivalently, we let $V$ be the $k$-mode “vec-operator” form of $S$, i.e. $V = \sum_{i,i'} S(i,i') \, vec(X_{i,i'})$. Then the problem is

$$
\max_{V} V \cdot vec(X)^{k} + trace(W^T \cdot X)
$$

subject to $X$ is a matching

which makes the tensor-eigenvector inspiration clear (see Sec. 2.2).

The tensor $S$ will change depending on what structure we will consider for the higher order matching problem and we may adjust the weightings between the prior matrix and the affinity tensors. It need not be symmetric in permutations of the first $k$ entries, which are permutations of $i$, but in the problems we consider in this paper, it will be. Likewise for permutations of the last $k$ entries for $i'$. Note that this means that $V$ is a symmetric tensor. Although $S$ itself is not even cubical. As might be surmised from the various definitions of $S$ above, there is extensive structure in this tensor and our goal is to use that structure to make the computations faster. Note that our goal is to use structure far beyond simple symmetry—which could only give a factor six speedup for a third order tensor.

### 2.6 Kronecker structure in higher-order alignment

There are a number of problems and methods that fit the structure of the higher-order graph alignment problem (Definition 2). We discuss two in depth: HOFASM and TAME.

The higher-order method Higher Order Fast Spectral graph Matching (HOFSM) [38] is an improvement to Higher-Order Matching (HOM) method due to Cherktok and Keller [12]. This method is designed to solve the image correspondence problems over images of different scales using a power method on affinity tensors capturing angle similarity in triangles of two images. The authors demonstrated the effectiveness of marginalization technique and sparsification and provided theoretical justifications using the probabilistic hypergraph matching framework of Zass and Shashua [43].

Given an image, we identify keypoints using any suitable procedure, such as SIFT features [61]. These keypoints are the vertices. When we are trying to identify correspondences between images, we seek matching triplets of keypoints. The original HOM method uses a similarity tensor based on triplets of points in images where similarities are computed using differences of triangle interior angles to capture scale invariance. Other image features, such as line segments lengths, can be used for other goals. Let $\theta_1^{i1}, \theta_2^{i1}, \theta_3^{i1}$ be the angles among a triplet of keypoints $i = (i, j, k)$ in one image. Let $\theta_1^{i'}, \theta_2^{i'}, \theta_3^{i'}$ be the angles among $i', j', k'$ in the other image. Then, in HOM, we use the global alignment framework with

$$
S(i, j, k, i', j', k') = f(\theta_1^{i1}, \theta_2^{i1}, \theta_3^{i1}, \theta_1^{i'}, \theta_2^{i'}, \theta_3^{i'})
$$

for some similarity function $f$. Given two images with $n$ keypoints, this tensor is dense, with $O(n^6)$ entries and little additional structure, which makes using it for alignment challenging. In HOFASM, Park et al. [38] adopt an approximation of this tensor where

$$
S(i, j, k, i', j', k') = \sum_{r,s} H_{r,s}(i, j, k) B_{r,s}(i', j', k'),
$$

where $H_{r,s}$ and $B_{r,s}$ are a constant-sized set of image dependent tensors and there are about 1000 tensors where each pair is indexed by these pairs $(r, s)$. This gives

$$
V = \sum_{r,s} B_{r,s} \otimes H_{r,s}.
$$

(Recall that we reverse the order of the matrices in the Kronecker product.) Namely, HOFASM approximates the HOM tensor with a sum of Kronecker products of tensors and we detail the rationale behind this approximation in §4.1.

While HOFASM arose from the image correspondence literature, the TAME method arose from the network alignment literature in bioinformatics. It is a spectral method that uses a tensor-eigenvector heuristic to guide the alignment. At the core, the TAME method is a simple instance of the higher-order graph alignment framework (Definition 2). Given two graphs $A$ and $B$, first enumerate triangles in each to build triangle adjacency tensors $T_A$ and $T_B$. Then set $S(i, i') = T_A(i) \cdot T_B(i')$. That is, the similarity of two triangles is 1 if both are there or 0 otherwise. For this choice, we have

$$
S = vec(V) = vec(T_B) \otimes vec(T_A).
$$

(6)

The methods and algorithms also work with arbitrary motifs, although we only investigate triangles.

### 3 Multi-linear Kronecker Product

Here, we establish a number of useful theorems to enable faster computation of expressions such as

$$
(B \otimes A) \cdot vec(X)^{k-1}
$$

(7)

when $X$ has additional structure – in particular, low-rank structure. This tensor-vector product shows up as a key-subroutine in both the HOFASM and TAME methods. We state the results in more generality than we need for the applications with three mode tensors as the proofs simplify with the generality (and those results may be independently useful in future research). This involves introducing new notation as well to make our results look like their matrix analogues.
3.1 A simplified result

We begin with a simplified result that exemplifies a useful special case. This theorem regards a tensor generalization of the matrix-vector identity for Kronecker products: \((B \otimes A)(y \otimes x) = (By) \otimes (Ax)\). For the sake of notation in this section, let \(T\) be a general k-mode, symmetric tensor. The notation \(Tz^p\) for integer \(0 \leq p \leq k\) represents the \(k-p\) mode tensor

\[
(Tz^p) = \sum_i T(i)_1 z(i_2) \cdots z(i_p) E_{i_{p+1},i_{p+2},\ldots,i_k}
\]

where \(E_{i_{p+1},i_{p+2},\ldots,i_k}\) is a tensor with a 1 in the \(i_{p+1}, \ldots, i_k\) entry.

**Theorem 3** Given two k-mode, cubical tensors \(A\) and \(B\) of dimension \(m\) and \(n\), respectively, and the \(m \times n\) rank 1 matrix \(X = uv^T\), then

\[
(B \otimes A) vec(X)^p = (B \otimes A)(v \otimes u)^p = Bv^p \otimes Au^p,
\]

for \(1 \leq p \leq k\).

**Proof** First note that for any symmetric tensor \(T\), we have in our notation \(Tz^p = (Tz^{p-1})z\) for any \(2 \leq p \leq k\).

Our proof simply uses induction. The base case is \(p = 1\). Here, we split the multi-index \(i\) (for \(A\)) into its first component and tail \(i_1, j\), and do the same for the \(j' = (i_1', j')\) for \(B\). Then

\[
(Bu \otimes Av)(i,j)[i',j'] = \sum_{i_1} A(i_1, j)u(i_1)(\sum_{i_1'} B(i_1', j')v(i_1'))
\]

\[= \sum_{i_1} A(i_1, j)B(i_1', j')u(i_1)v(i_1')
\]

\[= \sum_{i_1} (B \otimes A)[i_1i_1', j][v \otimes u][i_1i_1']
\]

\[= (B \otimes A)(u \otimes v)[i,j'][i',j'].
\]

The remainder of the argument is as follows. Assume the result holds for up to some value of \(p\), then we can show it holds for \(p + 1\) via

\[
Bv^{p+1} \otimes Au^{p+1} = (Bv^p \otimes Au^p)v = (D \otimes C)(v \otimes u).
\]

Note that \(C\) and \(D\) are both symmetric, so we can apply the previous result (or the inductive hypothesis). Inductively, now, we have \(D \otimes C = (B \otimes A)(v \otimes u)^p\) and thus we are done using our initial note.

The more general case of this theorem with a rank \(r\) matrix requires just a bit more notation.

3.2 Modal products and a more general result

Recall that the Kronecker product \(B \otimes A\) for matrices arises by treating \(AXB^T\) as a linear operator for an \(m \times n\) matrix \(X\). For symmetric \(A\) and \(B\) the tensor case treats

\[
\sum A(i_1, i_2, i_3)X(i_1, i_1')X(i_2, i_2')B(i_1, i_2, i_3)
\]

as a multi-linear operator (in this third order example). Consequently, the more general case will require a bit more notation. The term \(\sum_{i_1} A(i_1, i_2, i_3)X(i_1, i_1)\) is commonly referred to as a modal product of the first mode, and written as \(A \times_1 X\). For just this 3rd order expression, this portion of the expression would be \(A \times_1 X \times_2 X\). In general, this will involve

\[
A \times_1 X \times_2 \cdots \times_p X.
\]

Moreover, we also often wish to use column or rank-wise expansions. Suppose \(X\) is \(m\) by \(r\) and \(A\) is \(k \geq p\) modes, cubical with dimension \(m\). Then

\[
[A \times_1 X \times_2 \cdots \times_p X](i, j) \in [r]^p, j \in [m]^{k-p}
\]

\[
= \sum_{\ell \in [m]^p} A(\ell, j)X(\ell_1, i_1) \cdots X(\ell_p, i_p)
\]

We introduce the following functional notation to simplify

\[
A(x_1, \ldots, x_p) = \sum_{\ell \in [m]^p} A(\ell, \ldots, x_1(\ell_1) \cdots x_p(\ell_p))
\]

where dimensions and modes of the result are the trailing \(k-p\) dimensions of \(B\). Note that \(A(x_1, \ldots, p) \times_i \ldots \times_x X = AX^p\). This notation allows us to write a general modal product as

\[
[A \times_1 X \times_2 \cdots \times_p X](i, j) = [A(X_1(:, i_1), X_2(:, i_2), \ldots, X_p(:, i_p))](j).
\]

The utility of this notation is that it gives us a generalization of Theorem 3 to a general matrix \(X = YZ^T\) where \(Y\) and \(Z\) have \(r\) columns. For the classic matrix case, we have vec\((AXZ^T) = vec(AYZ^T) A vec(I)\), where \(I\) is \(r \times r\).

**Lemma 4** Given two k-mode, cubical tensors \(A\) and \(B\) of dimension \(m\) and \(n\), respectively, and the matrix \(X \in \mathbb{R}^{m \times n}\) with the rank \(r\) decomposition \(YZ^T\), then

\[
(B \otimes A) vec(X)^p
\]

\[
= \sum_{i \in [r]^p} B(Z(:, i_1), \ldots, Z(:, i_p)) \otimes A(Y(:, i_1), \ldots, Y(:, i_p))
\]

\[
= ([B \otimes A](Y(:, i_1)) \cdots X(X(:, i_1)) \cdots X(X(:, i_p))) vec(I)^p,
\]

where \(I\) is the \(r \times r\) identity matrix.

**Proof** We show this directly on each element, starting with the first line. Let \(j \in [m]^{k-p}\) and \(j' \in [n]^{k-p}\). Then

\[
((B \otimes A) vec(X)^p)[i,j'] = \sum_{\ell \in [m]^p} B(\ell, j')X(\ell_1, i_1) \cdots X(\ell_p, i_p)
\]

\[
= \sum_{\ell \in [m]^p} (B(\ell, j') \otimes A(X(:, i_1)) \cdots X(X(:, i_p)) vec(I)^p)
\]

This gets us the second line after we convert to the Kronecker product. To get the last line, we can convert the summation over \(i\) into a double summation multiplied by an indicator. Essentially, we use \(\sum_{i} a_i b_i = \sum_{i,j} a_i b_j I_{ij}\). Note that because
A useful property of Kronecker products is that the eigenvalues and eigenvectors of $B \otimes A$ decouple into Kronecker products of the eigenvalues of $A$ and $B$, individually. This makes spectral analysis of matrix Kronecker products efficient. We call an eigenpair dominant if it is the global maximum of $|Ax|^k$ where $|x| = 1$. Here, we show that this decoupling property remains true for the dominant tensor eigenvalue of a Kronecker product of tensors.

**THEOREM 5** Let $A$ be a symmetric, $k$-mode, $m$-dimensional tensor and $B$ be a symmetric, $n$-dimensional tensor. Suppose that $(\lambda_A^*, u^*)$ and $(\lambda_B^*, v^*)$ are any dominant tensor $Z$-eigenvalues and vectors of $A$ and $B$, respectively. Then $(\lambda_A^* \lambda_B^*, v^* \otimes u^*)$ is a dominant eigenpair of $B \otimes A$. Moreover, any Kronecker product of $Z$-eigenvectors of $A$ and $B$ is a $Z$-eigenvector of $B \otimes A$.

**Proof** Let $x = \text{vec}(X)$ be any vector with $\|x\|_2 = \|X\|_F = 1$ where $X$ is an $m \times n$ matrix. Let $z(i) = \|X(:,i)\|_2$. We have $\|z\|_2 = 1$ as well. Then we create

$$Z = \text{diag}(z(1), \ldots, z(n))$$

and $Y$ so that $X = YZ$.

The $ith$ column of $Y$ is either normalized or entirely 0 (if $z(i) = 0$). Recall that the dominant eigenpair maximizes $|(B \otimes A)x|^k = \|(B \otimes A)\text{vec}(X)^k\|$. From Lemma 4 we have

$$\|(B \otimes A)\text{vec}(X)^k\| = \|A(Y(:,i_1), \ldots, Y(:,i_k))B(Z(:,i_1), \ldots, Z(:,i_k))\|$$

$$= \|B_1(A(Y(:,i_1), \ldots, Y(:,i_k))B(Z(:,i_1), \ldots, Z(:,i_k))k\|_1 z(i_j)B(I(:,i_1), \ldots, I(:,i_k))\|,$$

where $I(:,j)$ is the $j$th column of the identity matrix. Now, because $A$ is symmetric, we have that

$$|\lambda_A^*| = \text{maximize } |A(u_1, \ldots, u_k)| \text{ subject to } \|u_k\| = 0, 1.\$$

This follows from a relatively recent result on the best rank-1 approximation of a symmetric tensor [62, Theorem 2.1], where the result is with $\|u_k\| = 1$. We can handle cases with $\|u_k\| = 0$ (which we could have for zero columns of $X$) by simply noting that any such column gives a zero overall value, so the maximum will never occur for those. Thus, this gives us an upper-bound on $|A(Y(:,i_1), \ldots, Y(:,i_k))|$

$$\|(B \otimes A)\text{vec}(X)^k\| \leq |\lambda_A^*| \sum_{i=1}^k z(i_j)B(I(:,i_1), \ldots, I(:,i_k))\|.$$
We now explain how this gives rise to the sum of Kronecker
have
In $A$ there are 8 keypoints selected and in $B$ there are 7 keypoints.
We show two of the many triangles. The angles for each triangle are
centered at the respective vertex in the ordering. We give a few examples
for clarity. For triangle $i = (1, 4, 6)$ in $A$, we have $\theta_{1}^{[i]} = \theta_{a}, \theta_{2}^{[i]} = \theta_{b}, \theta_{3}^{[i]} = \theta_{c}$. If instead in $A$ we had $i = (4, 1, 6)$ then we have $\theta_{1}^{[i]} = \theta_{a}, \theta_{2}^{[i]} = \theta_{c}, \theta_{3}^{[i]} = \theta_{b}$. As another example, if in $B$ $i = (5, 2, 3)$ then we have $\theta_{1}^{[i]} = \theta_{b}, \theta_{2}^{[i]} = \theta_{c}, \theta_{3}^{[i]} = \theta_{a}$. Also note that $S = S(1, 4, 6, 3, 5, 2) = 
\exp(-\frac{(\theta_{1}^{[i]} - \theta_{2}^{[i]})^{2} + (\theta_{1}^{[i]} - \theta_{3}^{[i]})^{2} + (\theta_{3}^{[i]} - \theta_{2}^{[i]})^{2}}{c^{2}}}$.

framework (Definition 2) for image correspondence between
triples of image keybords with

$S(i, j, k, \theta_{1}^{[i]}, \theta_{2}^{[i]}, \theta_{3}^{[i]}, \theta_{2}^{[j]}, \theta_{3}^{[j]}, \theta_{3}^{[k]}) = \exp(-\frac{(\theta_{1}^{[i]} - \theta_{2}^{[j]})^{2} + (\theta_{1}^{[i]} - \theta_{3}^{[k]})^{2} + (\theta_{3}^{[k]} - \theta_{2}^{[j]})^{2}}{c^{2}})}.$

where $c$ is a RBF parameter that determines the similarity
scale. The way we map between triangles and angles is to
use the index order in $i$ (or $i'$) to determine angle order based
on the associated center vertex. This is illustrated in Figure 1 To avoid the $O(n^6)$ runtime implied by the HOM method,
HOFASM adopts an approximation of this tensor. The
authors noticed that triangles with similar angles lead to
similar entries in their similarity tensor $S$. This suggests
the following simplification: quantize the set of angles in
one image. The idea is general enough to be able to quantize
angles in both images, but the HOFASM authors only used
one. This reduces the set of distinct entries in the tensor $S$. We now explain how this gives rise to the sum of Kronecker
product structure, which reduces the stored information to
be proportional to $O(n^3)$ where the constant depends on
the quantization.

Let $f(\theta_{1}, \theta_{2}, \theta_{3}, \theta_{1}', \theta_{2}', \theta_{3}')$ represent the similarity function
between two triangles of keypoints. The HOFASM
authors actually use the alternative

$f(\theta_{1}, \theta_{2}, \theta_{3}, \theta_{1}', \theta_{2}', \theta_{3}') = 4.5 - \frac{(\theta_{1} - \theta_{1}')^{2} + (\theta_{2} - \theta_{2}')^{2} + (\theta_{3} - \theta_{3}')^{2}}{6\epsilon^{2}}$ if $|\theta_{1} - \theta_{1}'| \leq 3\epsilon$ and $|\theta_{2} - \theta_{2}'| \leq 3\epsilon$ and $|\theta_{3} - \theta_{3}'| \leq 3\epsilon$

or 0 otherwise

to promote non-negativity and sparsity, but either choice
works for our exposition. The idea in HOFASM is that we
take the tensor $S$ and look at it as a sum of elements

$S = \sum_{i} \sum_{i'} f(\theta_{1}^{[i]}, \theta_{2}^{[i]}, \theta_{3}^{[i]}), \theta_{1}^{[i']}, \theta_{2}^{[i']}, \theta_{3}^{[i']}) E_{1...i',...}$

Here $E_{1...i',...} = E_{i,j,k,l,k',l'}$ is just the tensor with a one in
$i, j, k, l, k', l'$ element and zero elsewhere. So this expression
is simply writing out the tensor as a linear combination of
single-element tensors. (These single element tensors are
sometimes written with a Kronecker delta.) Note also that $f$
can be reduced to depend on 4 angles because we have
convex triangles where $\theta_{1}^{[i]} = 180 - \theta_{1}^{[i]} - \theta_{2}^{[i]}$.

$S = \sum_{i} \sum_{i'} f(\theta_{1}^{[i]}, \theta_{2}^{[i]}, \theta_{1}^{[i']}, \theta_{2}^{[i']}) E_{1...i',...}$

If we quantize the angles $\theta_{1}^{[i]}$ and $\theta_{2}^{[i]}$ for all $i = (i, j, k)$ in
the first image into a small set of at most $D$ values in each
angle, then we have

$S = \sum_{i} \sum_{i'} f(\theta_{1}^{[i]}, \theta_{2}^{[i]}, \theta_{1}^{[i']}, \theta_{2}^{[i']}) E_{1...i',...}$

where $\theta_{1}^{[i]}$ and $\theta_{2}^{[i]}$ are the quantized values. Let $\theta_{r}$ and
$\theta_{s}$ be the $r$th and $s$th quantized values for $\theta_{1}^{[i]}$ and $\theta_{2}^{[i]}$
respectively. We use angles binned in 5 degree increments as
in the HOFASM paper -- although any quantization strategy
works. Then we can separate the function by summing over all
choices for the quantization multiplied by an indicator function.
Formally, we intentionally sum over all possible
discretized angles

$S = \sum_{r,s} \sum_{i} \sum_{i'} (f(\theta_{r}, \theta_{s}, \theta_{1}^{[i]}, \theta_{2}^{[i]}) \text{Ind}[\theta_{1}^{[i]} = \theta_{r}] \text{Ind}[\theta_{2}^{[i]} = \theta_{s}]) E_{1...i',...}$

This is progress as the dependence in indices is now broken:

$f(\theta_{r}, \theta_{s}, \theta_{1}^{[i]}, \theta_{2}^{[i]}) \text{Ind}[\theta_{1}^{[i]} = \theta_{r}] \text{Ind}[\theta_{2}^{[i]} = \theta_{s}]$

depends only $i' = (i', j', k')$

which is a tell-tale sign of Kronecker structure. We now show
how to precompute tables to make this information easy to
work with. Let $H_{r,s}$ be a $3$-mode tensor corresponding to
the triangles that use discretized angles $\theta_{r}, \theta_{s}$. Formally,

$H_{r,s}(i) = \begin{cases} 1 & \text{Ind}[\theta_{1}^{[i]} = \theta_{r}] \text{Ind}[\theta_{2}^{[i]} = \theta_{s}] \\ 0 & \text{otherwise} \end{cases}$

Likewise, let $B_{r,s}$ be a $3$-mode tensor corresponding to the function values involved in aligning triangles with quantized
angles $\theta_{r}, \theta_{s}$ in image 1 with triangles in image 2, so we get

$B_{r,s}(i') = f(\theta_{r}, \theta_{s}, \theta_{1}^{[i']}, \theta_{2}^{[i']})$

This is, potentially, a dense tensor if all triangles are included.
In HOFASM, it is sparse because the function $f$ is zero
when $|\theta_{r} - \theta_{1}^{[i']}|$ and $|\theta_{s} - \theta_{2}^{[i']}|$ are large (bigger than $3\epsilon$).
Consequently, the information in $S$ can be expressed as the following
product

$S = \sum_{r,s} \sum_{i} H_{r,s}(i) B_{r,s}(i') E_{1...i',...}$

In this form, we can efficiently perform various contractions with $S$. In particular, if we re-organize $S$ into a
3-mode vec-operator tensor $V$. Then, we have $E_{1...i',...} \Rightarrow \text{vec}$
Assignment to find the correspondences between the images process operation. Procedurally, this computation is equivalent with computing a matrix $Z = B_{r,s} \otimes H_{r,s} \times_1 \text{vec}(I_m I_n^T) \times_2 \text{vec}(X_{\ell})$ which is the result of marginalizing $B_{r,s} \otimes H_{r,s}$ over one of the dimensions. (Due to symmetry, it does not matter which one, so we pick the first.) Then computing $Z \text{vec}(X_{\ell})$ to get a vectorized form of $X_{\ell+1}$. Marginalization can be decomposed into multiplication by a rank-1 matrix, so we can use Theorem \[35\] to get

$$Z = C_B B_{r,s} \otimes H_{r,s}(I_m) \otimes D_H H_{r,s}(I_n).$$

Note that $Z$ is actually a Kronecker product between matrices (at this point). So we can we can then compute the next iterate through the matrix-product

$$X_{\ell+1} = DX_{\ell}C^T.$$ 

Commonly $H_{r,s}$ will be quite sparse, as the sum of all the non-zeros of each $H_{r,s}$ is equal to the number of triangles in the quantized image. Consequently, we found the most efficient way to compute $DX_{\ell}$ term first, and then make use of the sparsity structure of $H_{r,s}$, which is manifest in $(DX_{\ell})C^T$, for additional efficiency.

We will show in Section 4.4 that this leads to a factor of 10 improvement over the fastest implementation of the original algorithm as described by the authors, while still leaving open the option of other speed-ups through the form of parallel implementations (like the authors originally suggested). In practice, we compute the marginalizations once as they do not change from iteration-to-iteration and simply store the results as matrices. There are a few other implementation details we discuss in Section 4.5 for instance, some simplifications due to permutations.

### 4.2 TAME and Exact LowRankTAME

While HOFASM arose from image correspondence, the TAME method arose from network alignment. It is a spectral method that uses a tensor-eigenvector heuristic to guide the alignment. Recall that the TAME method is a simple instance of the higher-order graph alignment framework (Definition \[2\]), where, given two graphs $A$ and $B$, we first enumerate triangles in each, in order to build triangle adjacency tensors $T_A$ and $T_B$. Then we set $S(i,i') = T_A(i)T_B(i')$. For this choice, we have

$$S \leftarrow V = T_B \otimes T_A$$

as can be verified, for instance, through a derivation with $E_{k,r,s}$ akin to the one for HOFASM. This results in the following idealized optimization problem for TAME

$$\text{maximize } (1 - \alpha)\text{trace}(W^T X) + \frac{\alpha}{6}(T_B \otimes T_A) \text{vec}(X)^3$$

subject to $X$ is a matching.

Here the value $\alpha/6$ arises because each triangle alignment gives 6 entries in $T_B \otimes T_A$ due to symmetry. The weight matrix $W$ gives flexibility to bias the alignment towards certain nodes. When no prior matrix is available, we make use of a rank 1 matrix $W = \frac{1}{m} I_m I_n$ which gives a uniform bias everywhere.

The heuristic procedure used in TAME is to deploy the SS-HOPM algorithm \[52\] to seek a tensor eigenvector, or near tensor eigenvector, of $V = T_B \otimes T_A$. At each iteration, we explicitly compute a matching and return the best solution with the highest downstream objective (triangle alignment, mixture, or some other combination). We show the procedure in Algorithm \[2\]. The authors also make use of a b-matching post processing algorithm, which can be run to further refine the alignments, but we won’t focus on that portion of the algorithm because the runtime of the TAME algorithm is dominated by the search of the tensor power sequence and the computing of the matchings. We also present an affine-shift variant of the TAME method that includes the
Algorithm 2 TAME \textsuperscript{[12]} with affine shift

\textbf{Require:} $k$-mode motif tensors $T_A, T_B$ for graphs $A$ and $B$, mixing parameter $\alpha$, shift $\beta$, tolerance $\varepsilon$, weights $W$

\textbf{Ensure:} Alignment matrix $X_{\text{best}}$

1: $X_0 = W/\|W\|_F$ \Comment*[r]{Normalize first iterate}
2: \textbf{for} $\ell = 0, 1, \ldots$ until $|\lambda_{\ell+1} - \lambda_\ell| < \varepsilon$ \textbf{do}
3: \Comment*[r]{SS-HOPM iteration}
4: $X_{\ell+1} = \text{unvec}(T_B \otimes T_A) \text{vec}(X_\ell)^{k-1}$ \Comment*[r]{Implicitly}
5: $\lambda_{\ell+1} = \text{trace}(X_\ell^T X_{\ell+1})$ \Comment*[r]{Estimate tensor-eval}
6: $X_{\ell+1} \leftarrow \alpha X_{\ell+1} + \beta X_\ell + (1 - \alpha)X_0$
7: $X_{\ell+1} \leftarrow X_{\ell+1}/\|X_{\ell+1}\|_F$
8: \textbf{end for}
9: \textbf{return} the matching of $X_{\ell+1}$ with the highest $t_{\ell+1}$

mixing parameter $\alpha$ to re-mix in the original iterate whereas TAME \textsuperscript{[12]} fixed $\alpha = 1$. This choice sometimes helps boost performance a little bit.

In TAME the authors make use of an implicit operation to compute the iterates of the tensor powers

$$T_B \otimes T_A \text{vec}(X)^{k-1} \quad (16)$$

without forming $T_B \otimes T_A$. This computation still takes $O(\text{nnz}(T_B) + \text{nnz}(T_A))$ work, where nnz is the number of non-zeros in the sparse tensor.

In the case of the uniform bias prior ($W = \frac{1}{mn} 1_m 1_n^T$), the first iterate is rank-1, so we could apply Theorem 3 to decouple the operation. Because of the shift $\beta$, however, subsequent iterations will not remain rank 1 as the following observation clarifies.

\textbf{Observation.} Suppose that $W$ is rank 1 and we are dealing with a $k$-mode tensor. Then the theory of our multilinear Kronecker products (Lemma 4, applied to the TAME iteration, states that if $X$ is rank $r$ then the next iterate has rank at most $rk^{-1} + r + 1$. This follows from the number of combinations of vectors in the Lemma combined with the addition of the $r$ rank factors for the previous iterate in the shift. Also $rk^{-1}$ can be reduced to $\left(\frac{r}{k-1}\right)$ for the symmetric case, but for simplicity we use the upper-bound $r^{k-1}$.

While this observation explains a simplistic analysis for the worst case scenario for the rank grow of the iterates, in practice we find it extremely conservative. (See evidence in §5.3) This means that there is still a useful low-rank strategy to employ with our theory. Namely, use Lemma 4 to compute the components of the next iterate and then compute an exact low-rank factorization. As long as the rank does not get too big, this will be faster.

\textbf{An exact low-rank TAME iteration.} Let $W = FG^T$ be the low-rank factors of the weight matrix $W$ and let $t$ be the rank of the matrix.

1) Given a rank $r$ factorization of iterate $X_t = Y_t Z_t^T$
2) Compute all $r^{k-1}$ terms in the summation expansion from Lemma 4 (This is $r^2$ for triangle tensors.) This results in $X_{t+1} = Y_{t+1} \tilde{Z}_{t+1}^T$ where we know the rank is less than $r^{k-1}$.
3) Append the previous low-rank factorizations to incorporate the shift by the previous iterate and the addition of the initial weight matrix; set

$$\hat{Y}_{t+1} \leftarrow \sqrt{\alpha} \hat{Y}_{t+1} \quad \sqrt{\alpha \beta} Y_t \quad \sqrt{1 - \alpha F}$$

$$\hat{Z}_{t+1} \leftarrow \sqrt{\alpha} \hat{Z}_{t+1} \quad \sqrt{\alpha \beta} Z_t \quad \sqrt{1 - \alpha F}$$

4) Use a rank-revealing factorization (such as the SVD or rank-revealing QR) on $\hat{Y}_{t+1}$ to get $\hat{Y}_{t+1} = UV^T$ as a rank $s$ factorization; also try the same with $\hat{Z}_{t+1}$ to get the minimal rank. Then we use

$$X_{t+1} = U \left(\hat{Z}_{t+1} V^T\right)$$

as the factorization. (Or the analogous term with the factors of $\hat{Z}_{t+1}$ instead for a smaller rank.)

The dominant terms in the overall runtime of this approach for $k$-node motifs is $O(\text{nnz}(T_A) + \text{nnz}(T_B))k^{k-1} + \text{RFF}(m, (r^{k-1} + r + t)) + \text{RFF}(n, (r^{k-1} + r + t))$ where RFF is the cost of the rank-revealing factorization. There are many options here including randomized and tall-and-skinny approaches. In our codes we use an SVD for simplicity. Representative values of the ranks $r$ are typically 100 and are much smaller than $n$ or $m$; see more discussion in §5.3.

This low-rank scenario offers a few benefits even beyond the reduced runtime. First, we are able to explicitly store a large number of TAME iterates as low-rank factorizations. This enables us to easily parallelize the step of estimating matchings at each iteration as we can simply assign matching problems to processors. (Parallelizing within each matching problem is challenging \textsuperscript{[65, 66]}; see the discussion in \textsuperscript{[67]}) Furthermore, this enables us to use fast low-rank matching heuristics from \textsuperscript{[51]} as well.

4.3 \textbf{A-TAME}

The inspiration for using SS-HOPM in TAME is that the spectral relaxation for TAME’s objective function (15) leads us to the dominant eigenvector problem for $T_B \otimes T_A$. With Theorem 5 we’ve shown that the dominant eigenvector is built from the dominant eigenvectors of $T_B$ and $T_A$. This suggests a new heuristic which can be run using only the tensor powers sequences of $T_B$ and $T_A$ independently, rather than combining them as is done in TAME. We then store each of the iterates into a pair of matrices $U$ and $V$, and we search over all pairs of vectors (one from $T_B$ and one from $T_A$). In fact, we can use any iterative algorithm including recent work on dynamical systems \textsuperscript{[54]}. Since each approximate eigenvector is rank 1, we can efficiently find a matching based on the rearrangement inequality—as was done in our past research for low-rank alignment techniques \textsuperscript{[31]} and the NSD method that inspired this approach \textsuperscript{[30]}. The resulting algorithm is given in Algorithm 3. We call this method A-TAME because it is inspired by our dominant Z-eigenvalue theorem.

In the algorithm, both $U$ and $V$ can be computed in time proportional to the number of non-zeros of their tensors times the total number of iterations. A matching between any pair of vectors in $U$ and $V$ can then be computed in $O(m \log(m))$ time. (Here, without losing any generality, $m \ge n$ is the larger size of the problem.) We typically only run the algorithms for each tensor for a small number of iterations—around 10-20, which means the total number of pairs of vectors is manageable. Again, we adopted an affine-shift variant of the TAME method that includes an $\alpha$ factor to reinroduce the original vector into the solution. This can
Algorithm 3 Λ-TAME

Require: \( k \)-mode motif tensors \( T_A, T_B \) for graph \( A \) and \( B \); mixing parameter \( \alpha \), shift \( \beta \), max iterations \( L \)

Ensure: Alignment matrix \( X \)

1: Initialize first columns: \( U(:,1) = \frac{1}{\sqrt{n}} \); \( V(:,1) = \frac{1}{\sqrt{n}} \)
2: for \( \ell = 1, \ldots, L \) do
3: \( U(:,\ell+1) = T_A U(:,\ell)^{k-1}; V(:,\ell+1) = T_B V(:,\ell)^{k-1} \)
4: \( U(:,\ell+1) \leftarrow \alpha U(:,\ell+1) + \alpha \beta U(:,\ell) + (1-\alpha) U(:,1) \)
5: \( V(:,\ell+1) \leftarrow \alpha V(:,\ell+1) + \alpha \beta V(:,\ell) + (1-\alpha) V(:,1) \)
6: \( U(:,\ell+1) = \frac{U(:,\ell+1)}{\| U(:,\ell+1) \|}; V(:,\ell+1) = \frac{V(:,\ell+1)}{\| V(:,\ell+1) \|} \)
7: for each pair \( i, j \) of vectors in \( U \) and \( V \) do
8: Compute a matching \( X \) based on \( U(:,i) \) and \( V(:,j) \)
9: Score the matching based on triangles, weights, etc.
10: Return the matching \( X \) with the highest score.

be set to 0 so that the iterates are exactly those from the SS-HOFM method, but there are cases where \( \alpha \neq 0 \) helps.

5 Empirical Comparisons

We have implemented both straightforward versions of the HOFASM and TAME algorithms with the original implicit tensor Kronecker products as well as new versions using our tensor Kronecker theory. We further implemented our new Λ-TAME method. This software are written in Julia and are available from [https://www.cs.purdue.edu/homes/colley/project_pages/TensorKroneckerProducts.html](https://www.cs.purdue.edu/homes/colley/project_pages/TensorKroneckerProducts.html). We will use this section to validate the algorithms and show how we can achieve similar results with greatly improved runtimes. Some highlights of our results:

1. HOFASM using the tensor Kronecker product theory can be computed very quickly – about 1 order of magnitude faster than the implicit tensor Kronecker product formulation – in experiments from [38] (see §5.1).
2. Iterates of TAME are low rank on real data and synthetic data and the exact LowRankTAME method computed them an order of magnitude faster (§5.3.5.4).
3. Λ-TAME is an effective heuristic, running 3 orders of magnitude faster than the C++ TAME implementation, with only a small reduction in the number of triangles matched (§5.4).

We use all the same parameters as the original research where they were accessible and will discuss our reasoning for our choices for unlistered parameters. Our experiment environment uses Intel Xeon Platinum 8168 CPUs (2.70GHz) with 24 cores. Our largest experiment makes use of approximately 10.4Gb of RAM.

5.1 HOFASM

Our theory suggests an implementation of HOFASM that will improve the runtime, so we use a subset of the author’s original experimental framework. This framework was originally used in [38]. It uses synthetic keypoint sets for simplicity. These are created by first generating a set \( S \) of \( n \), 2-dimensional points \( s \), where each coordinate is drawn from a standard normal distribution (i.e. \( S = \text{randn}(n, 2) \)) in Matlab or Julia. We then synthetically perturb these by adding random normal noise to each component with standard deviation \( \sigma \), so we get \( s_i' = s_i + \sigma \text{randn}(2) \). We then compute all \( \binom{n}{3} \) triangles among the points in \( S \) and likewise in \( S' \) to create the triangles and form the tensors as the HOFASM approximation, using angles discretized into 5 degree increments. (Note that a straightforward implementation of this idea as currently presented would need all \( n^3 \) triangles, in all permutations, but we explain below in the extra details paragraph how this can be avoided as is done in our code.)

We compare three different implementations of HOFASM for runtime in Figure 26 called “fully implicit”, “partially implicit”, and “our work.” Recall that our Kronecker theory shows that the tensors \( B_T \) and \( H_{rs} \) can be marginalized independently. This enables us to use a one-time marginalization as in Algorithm 4 which reflects the time of “our work.” In comparison, “partially implicit” is essentially what the HOFASM authors [38] describe. This corresponds to computing step 5 of Algorithm 1 using the steps: (i) explicitly compute the matrix \( Z = (B_T \otimes H_{rs}) \times 1 \text{vec}(I_r, I_s) \) but without building the Kronecker product, and then (ii) compute the matrix-vector product \( Z \text{vec}(X_i) \). Note that step (i) is constant in each iteration and although the HOFASM authors describe re-computing this at each iteration, we simply compute and store it once. The “fully implicit” algorithm involves computing step 5 with a fully implicit algorithm without any intermediaries or storage. Note that these algorithms all compute exactly the same quantities. The results in the figure show that our Kronecker theory results in at least a factor 10 reduction in runtime for the largest problems on 100 keypoints.

Finally, to validate the results of our HOFASM implementation and subsequent improvement, we compare the results of our Julia implementation of HOFASM with a similar reimplementation of Chertok and Keller’s HOM matrix marginalized method [32] (which inspired HOFASM). As in the HOFASM paper, we also find that this results improved results over HOM as illustrated in Figure 26.

Extra details about permutations and sparsity. There are a few subtleties in the HOFASM implementation due to reducing the data storage using permutations. When we enumerate triangles in keypoints, we only store each unique triangle once (instead of all 6 permutations). This involves picking a single binning of two triangle angles (instead of binning angles for all triangles separately, which may give slightly different bins) and then applying the permutations to the indices in the routines. This reduces the number of marginalized tensors by a factor of 6 (or 3 if another type of symmetry was used).
5.2 Data for network alignment experiments

There are two types of data that we use in evaluating the new network alignment algorithms. The first is a subset of the LVGNA [15] protein-protein interaction (PPI) graph collection along with the largest problem from Mohammadi et al. [12] Table 2. Each pair of networks in this collection gives an alignment problem. Network statistics are in the supplemental materials (Table 1). Each vertex represents a protein and the edges represent interactions. The networks range in size from 2871 to 16060 vertices and all but the largest networks have fewer triangles than edges.

The second type of data are synthetic Erdős-Rényi (ER) graphs and random geometric (RG) graphs. To generate the RG graphs, we randomly sample $n$ points in the unit square. Then each point adds undirected edges to the $k$ nearest neighbors, where $k$ is drawn from a log-normal distribution centered at $\log 5$ with $\sigma = 1$. We then get a problem by perturbing these graphs following [31] which involves randomly deleting edges with probability $q$ and randomly adding edges with probability $p = \frac{qn}{1 - q}$, where $\rho$ is the density of the graph. Consequently, the perturbation are a separate process from edge formation, which makes the network alignment framework appropriate. We further randomize the permutation of the second network to avoid any influences due to node order.

5.3 Low-rank structure in TAME

For our first experiment, we want to show that the iterations from TAME (Algorithm 2) remain low rank when we start with a uniform, or unbiased iterate as the weight matrix: $X_0 = \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T$, which is rank 1. To do this, we use the exact LowRankTAME algorithm instead of the raw TAME algorithm, which are identical in exact arithmetic. Given either a pair of networks from LVGNA, the largest alignment problem from Mohammadi et al. [12], or a synthetic RG, we plot the maximum rank from any iteration on any trial ($\alpha \in \{0.5, 1.0\}$ running 15 iterations) as determined by the rank function in Julia, as we vary the $\beta$ parameter in the alignment problem. (See Figure 3) These results, along with trendlines for the maximum rank over multiple repetitions of the synthetic experiments, show that the rank is often below 200 even though many networks have thousands of vertices.

These experiments show the synthetic tests have higher ranks than the LVGNA collection. Whereas we do see the rank grow mildly for the synthetic problems—up to a limit—for the LVGNA collection, large problems tend to have smaller rank. We also see that increasing $\beta$ produces higher ranks because these problems incorporate more of the previous iterate via an affine shift. We further verified that the max-rank of the $\beta = 10$ case on the synthetic problems does decrease as the problems get even bigger. All of these experiments support using the exact LowRankTAME algorithm compared with the original TAME algorithm.

5.4 Comparing network alignment algorithms

Now we will show how all our methods compare against one another as well as to L-GRAAL [14] and LowRankEigenAlign [31]. L-GRAAL computes graphlet degrees to guide alignments (similar to other GRAAL methods) and uses Lagrangian relaxations to seed and extend its matchings. L-GRAAL was one of the higher quality, but longer running algorithms in the original TAME paper [12] Fig. 1,2. LowRankEigenAlign utilizes low rank structure discovered in the EigenAlign [29] algorithm and improves its scalability at no cost to accuracy. LowRankEigenAlign has been tested

1. This choice of exact LowRankTAME vs. TAME is because it is faster to compute but also because preliminary experiments showed that TAME caused the finite precision rank to grow even when the result is mathematically rank 1 ($\alpha = 1.0$, $\beta = 0.0$ by Theorem 1). This is well-known to happen to finite precision computations, for instance in the power method. Details of this test are in the Supp. A.1.3.
on similar real world and synthetic alignment problems, and its low rank structure makes it a comparable method in terms of memory to $\Lambda$-TAME. Neither L-GRAAL nor LowRankEigenAlign needs any seed information for the alignment. We also run LowRankTAME using the low rank matching procedure from [31], denoted as LowRankTAME-(lrm) or LR-TAME-lrm. Our experiments show that not only do $\Lambda$-TAME and LowRankTAME offer high quality matchings, but they compute them much faster than the original TAME. We demonstrate this on both random graph experiments—so that we can evaluate ground truth performance (see Figure 4a)—and also the LVGNA collection of PPI networks (see Figure 4b).

Random Graph Accuracies. We start with comparing the alignments of ER and RG random graph models in terms of their triangle alignment score (how many triangles they aligned compared with the maximum possible) and also their ground truth accuracy (formally, the precision of the alignment). We do this as we vary the edge removal probability in the perturbation process and also as the networks grow for RG models. We include additional baselines: degree and random matching. Degree matchings result from aligning the lists of vertices sorted by their degree in each graph and random matchings enumerate a random permutation of the nodes in the second network. The results are shown in Figure 4a. These results show that LowRankTAME offers the best overall performance and also that triangle alignment scores are an accurate proxy for ground truth alignments even as the values get small.

LVGNA Runtimes and Performance. Next, we evaluate the runtime over pairs of alignment problems from LVGNA. We also show a performance profile plot over the triangle accuracies to understand the relative performance of the algorithms. These results show that LowRankTAME is roughly an order of magnitude faster than the implementation of TAME in C++ from [12]; in comparison, $\Lambda$-TAME is about three orders of magnitude faster. Note the difference in runtime between LowRankTAME and LowRankTAME-(lrm); the only difference between these is an exact maximum matching algorithm compared with an approximate low-rank matching, thus we see that as the problem sizes grow, the growth in runtime can be attributed to the growing cost of solving for the maximum weighted matchings instead of the tensor operations. We furthermore see from the performance profile plot LowRankTAME and TAME have the strongest overall performance in terms of triangle alignment. The differences in triangles matched between LowRankTAME and TAME amounts to differences in how floating point values round at intermediate steps and how ties in the matchings break differently as edges in the matchings are traversed using row major formatting in the C++ code, whereas Julia uses a column major formatting. Similar to our random graph experiments, we see that L-GRAAL and LowRankEigenAlign don’t fare particularly well, however this is to be expected as these are methods focus on optimizing the number of edges matched, rather than triangles. Prior experiments with TAME showed that aligning triangles combined with a few careful post-processing steps result in best-in-class alignment solutions on these types of problems [12]. Our new techniques simply provide a framework to make these computations faster.

6 Related research and discussion

The major focus of our paper is on demonstrating how the novel theory on tensor Kronecker products [43] enables us to accelerate two higher-order graph matching algorithms [44] – both by making the same algorithm faster (HOFSAM and LowRankTAME) and by giving a new, faster algorithm ($\Lambda$-TAME). This involves

1) a novel extremal eigenbound for $Z$-eigenvalues of tensor Kronecker products,
2) generalization of the mixed product property to tensor Kronecker products, and
3) experimental evidence of how this theory offers faster runtimes for graph matching and network alignment.

Related work. In terms of related work, we wish to mention how our low-rank methodologies tie into a variety of recent work on network embedding methods. Network embedding, inspired by deepwalk [68] and node2vec [69], in the context of network alignment shares a close relationship with our low-rank methods. In both the LowRankTAME and $\Lambda$-TAME methods, we maintain a set of vectors associated with each graph. These are $Y$ and $Z$ in LowRankTAME, which represent a low-rank representation of the current iterate $X = YZ^T$, and $U$, $V$ for $\Lambda$-TAME, which represent a basis for a space of outer-product of columns that we use for alignment. Consequently, both $Y$ and $U$ can be considered embeddings of the vertices of the graph $A$, or more precisely for our methods, an embedding of the motif-hypergraph of $A$. These bases are permutation invariant, which avoids the requirement of many embedding procedures that require a set of seed nodes across vertices.

The fashion in which we construct these embeddings is also closely related to various graph kernels [70, 71] including the random walk kernel on a direct product graph. Graph kernels have long been used to align small chemigraphs (graphs that represent small chemical molecules). In this case, we are able to generate a direct factorization of a graph kernel between vertices of two graphs into a product of features on each graph. This is a common paradigm [70] involving matrix Kronecker products—although we are unaware of any research on this for higher order analogues of the graph kernels involving tensor Kronecker products that would be needed for our perspective. When viewed in this light, our research has the potential to open new directions in this space in terms of efficient graph kernels on hypergraphs.

Discussion. In summary, our theory and experiments show how the computational demands of methods with tensor Kronecker products may be reduced by orders of magnitude with no change in quality, or accelerated even further with useful approximate results. We are excited about the future opportunities with tensor Kronecker products due to the widespread use of matrix Kronecker products, and suspect that these theorems, or alternative generalizations that use specific structure in novel problems, will be a key element in this future research.

References

[1] D. Conte, P. P. Foggia, C. Sansone, and M. Vento, “Thirty years of graph matching in pattern recognition,” Intern. J. Pattern. Recognit. Artif. Intell., vol. 18, no. 3, pp. 265–298, May 2004.
Fig. 4. We compare the results of network alignment methods on synthetic (a) and real world data (b). (a) We see that the fraction of aligned triangles is a good proxy for accuracy and the TAME method is more robust to our perturbation experiments, especially for random geometric graphs. We plot the true accuracy (bottom) and the ratio of total matched triangles to maximum matchable triangles (top) for the methods LowRankTAME (LR-TAME) and LowRankTAME with low-rank matching (LR-TAME-lrm). -TAME, LowRankEigenAlign (LR-EigenAlign), degree matching and random matching. We report the results for two different accuracy perturbation experiments, the noise levels q and the sizes n of the graphs and plot the median with the 20th-80th percentiles of 50 trials as ribbons. In the noise experiments the q = .01 gridline of the RG is marked to denote the x = 0 intercept of the size experiments because these are the same experiment. (b) −TAME is the fastest tested method with about 3 orders of magnitude improvement from TAME, although the triangle performance is slightly reduced. The left plot is a simple runtime plot comparing the speed of our -TAME, LowRankTAME both with maximum matching and a low rank matching (lrm) approximation, the original TAME (C++), and L-GRAAL. The rightmost plot contains the relative performance of each algorithm in terms of triangle matched ratios. For each alignment problem we compute the ratio of triangles matched by an algorithm, divided by the most triangles matched by any algorithm run and plotted each algorithm’s sorted list of ratios. We see that the three TAME methods have the best triangle matched ratios of all methods tested over all pairwise alignments in the LVGNA dataset. Methods closest to 1 had best performances, and methods which attain a value of 0 had experiments which failed to align any triangles. Experiment details for (a) and (b) in Supp. [A.1.2] [A.1.5] respectively.

[2] F. Emmert-Streib, M. Dehmer, and Y. Shi, “Fifty years of graph matching, network alignment and network comparison,” Information Sciences, vol. 346, pp. 180–197, 2016.

[3] M. Brown, D. G. Lowe et al., “Recognising panoramas.” in ICCV, vol. 3, 2003, p. 1218.

[4] L. Juan and G. Oubong, “Surf applied in panorama image stitching,” in 2nd international conference on image processing theory, tools and applications. IEEE, 2010, pp. 495-498.

[5] A. C. Berg, T. L. Berg, and J. Malik, “Shape matching and object recognition using low distortion correspondences,” in Proceedings of CVPR, 2005, pp. 26–33.

[6] T. Cour, P. Srinivasan, and J. Shi, “Balanced graph matching,” in Advances in Neural Information Processing Systems 19, B. Scholkopf, C. Platt, and T. Hofmann, Eds. MIT Press, 2007, pp. 313–320.

[7] F. Bernard, J. Thunberg, P. Szwoboda, and C. Theobalt, “Hippi: Higher-order projected power iterations for scalable multi-matching,” in Proceedings of ICCV, 2019.

[8] O. Duchenne, F. Bach, I.-S. Kweon, and J. Ponce, “A tensor-based algorithm for high-order graph matching,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 12, pp. 2383–2395, 2011.

[9] M. Chertok and Y. Keller, “Spectral symmetry analysis,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 7, pp. 1227–1238, 2010.

[10] J. Hays, M. Leordeanu, A. A. Efros, and Y. Liu, “Discovering texture regularity as a higher-order correspondence problem,” in European Conference on Computer Vision. Springer, 2006, pp. 522–535.

[11] R. Singh, J. Xu, and B. Berger, “Global alignment of multiple protein interaction networks with application to functional orthology detection,” Proc. Natl. Acad. Sci. U.S.A., vol. 105, no. 35, pp. 12763–12768, 2008.

[12] S. Mohammadi, D. F. Gleich, T. G. Kolda, and A. Grama, “Triangular alignment TAME: A tensor-based approach for higher-order network alignment,” IEEE/ACM Trans. Comput. Biol. Bioinform., vol. 14, no. 6, pp. 1446–1458, 2017.

[13] R. Patro and C. Kingsford, “Global network alignment using multiscale spectral signatures,” Bioinformatics, vol. 28, no. 23, pp. 3105–3114, 2012.

[14] N. Malod-Dognin and N. Pržulj, “L-graal: Lagrangian graphlet-based network aligner,” Bioinformatics, vol. 31, no. 13, pp. 2182–2189, 2015.

[15] V. Vijayan and T. Milenković, “Multiple network alignment via multimagna++,” IEEE/ACM Trans. Comput. Biol. Bioinform., vol. 15, no. 5, pp. 1669–1682, 2017.

[16] L. Meng, A. Striegel, and T. Milenković, “Local versus global biological network alignment,” Bioinformatics, vol. 32, no. 20, pp. 3155–3164, 2016.

[17] F. Emmert-Streib, “The chronic fatigue syndrome: a comparative pathway analysis,” J. Comput. Biol., vol. 14, no. 7, pp. 961–972, 2007.

[18] N. Atias and R. Sharan, “Comparative analysis of protein networks: hard problems, practical solutions,” Commun. ACM, vol. 55, no. 5, pp. 88–97, May 2012.

[19] A. Elmsallati, C. Clark, and J. Kalita, “Global alignment of protein-protein interaction networks: A survey,” IEEE/ACM Trans. Comput. Biol. Bioinform., vol. 10, no. 1, p. S59, 2009.

[20] G. Świą, “Exploiting patterns in ontology mapping,” in Proceedings of ISWC/ASWC, ser. LNCS, vol. 4825, Berlin, Heidelberg, November 2007, pp. 950–954.

[21] S. Melnik, H. Garcia-Molina, and E. Rahm, “A versatile graph matching algorithm and its application to schema matching,” in Proc. 18th Intl. Conf. Data Engineering (ICDE), 2002, pp. 117–128.

[22] S. Lacoste-Julien, B. Taskar, D. Klein, and M. I. Jordan, “Word alignment via quadratic assignment,” in Proceedings of NAACL, Association for Computational Linguistics, 2006, pp. 112–119.

[23] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren, “A measure of similarity between graph vertices: Applications to synonymy extraction and web searching,” SIAM review, vol. 46, no. 4, pp. 647–666, 2004.

[24] M. Bazzara and O. Kirc, “A branch-and-bound-based heuristic for solving the quadratic assignment problem,” Naval research logistics quarterly, vol. 30, no. 2, pp. 287–304, 1983.

[25] T. C. Koopmans and M. Beckmann, “Assignment problems and the theory of production,” Econometrica: journal of the Econometric Society, pp. 53–76, 1957.

[26] E. L. Lawler, “The quadratic assignment problem,” Management science, vol. 9, no. 4, pp. 586–599, 1963.

[27] R. Burkard, M. Dell’Amico, and S. Martello, Assignment Problems, revised ed. SIAM, 2012.

[28] G. W. Klau, “A new graph-based method for pairwise global network alignment,” BMC Bioinform., vol. 10, no. 1, p. S99, 2009.
**APPENDIX A**

**SUPPLEMENTAL INFORMATION**

**A.1 Experiment Parameters**

To ensure the replicability of our results, we include the experiment parameters used for our experiments.

**A.1.1 HOFASM Experiment Parameters**

For the experiments in Figure 2 we use $\varepsilon = 10^{-4}$, $\beta = 30$ (lines 2 and 7) for the graduated assignment parameters of both HOFASM and HOM and set a tolerance of $10^{-4}$ for the Bistochastic Normalization loop (starting at line 8) which runs for a maximum of 12 iterations. For our runtime experiments we fix $\sigma = 1$ and vary $n = \{10, 20, 30, 40, 60, 80, 100\}$ for the runtime experiments. For the accuracy experiments, we fix $n = 30$ and vary $\sigma \in \{0.25, 0.05, 0.0725, \ldots, 0.125, 0.15, 0.175, 0.2\}$. In accuracy ratio experiments we remove one HOM experiment ($\sigma = .15$) as its accuracy is 0.

**A.1.2 Random Graph TAME Accuracies**

In Figure 4a we use two random graph models, Erdős-Rényi (ER) graphs with $p_{\text{edge}} = 2 \frac{\ln n}{n}$ and random geometric (RG) graphs as described in the main paper. For the noise variational experiments we plot the results of 50 trials for $q \in \{0.0, 0.02, 0.04, 0.06, 0.08, 0.1, 0.15, 0.2, 0.04, 0.6, 0.1, 2\}$ for a pair of graphs with $n = 100$ for ER and RG models. For the variable size experiments we plot the results of 20 trials with $q = .01$ and $n \in \{100, 150, 200, 250, 300, 400, 500, 600, 700, 850, 1000, 1150, 1300, 1500, 1750, 2000\}$ using just the RG models.

**A.1.3 TAME rank-1 Singular Value Experiments**

![Graph](image)

Fig. 5. LowRankTAME accurately captures the rank-structure when it is provably rank 1. TAME frequently produces iterates $X_t$, with non-dominant singular values large enough to be non-zero for matrices which are provably rank 1 ($\alpha = 1.0$, $\beta = 0.0$, by theorem 3).

These experiments explain why we use the exact LowRankTAME iteration instead of the original TAME iteration to study rank. They show TAME produces iterates that would be detected as rank-2 even when the answer is theoretically rank-1, whereas LowRankTAME does not. The figures plot the maximum second largest normalized singular values of $X_t$, $\sigma_2$, of all 15 iterations for TAME and LowRankTAME of the rank 1 iteration case. Hollow points are values small enough to be considered zero (and hence, would be rank-1), and filled points are large enough to be measured as non-zero (and hence, would be rank-2). The LVGNA experiments align all pairs of distinct networks. The synthetic experiments are measured over 10 trials using random geometric graphs, over both $q \in \{.01, .05\}$ for $n \in \{100, 500, 1K, 2K, 5K, 10K, 20K\}$.

**A.1.4 TAME max rank experiments**

In Figure 3, the top figure corresponds experiments conducted on random geometric graphs for $n = \{100, 500, 1K, 2K, 5K, 10K, 20K\}$. We report maximum rank over 15 iterations, $\alpha \in \{.5, 1.0\}$, and for random graphs, we additionally maximize over $q \in .01, .05$ and 50 trials.

**A.1.5 PPI Runtime Experiments**

In Figure 4b all TAME implementations are reported for total runtimes of $\alpha \in \{.5, 1.0\}$ and $\beta \in \{0, 1, 10, 100\}$ using 15 iterations, LGRAAL used $\alpha \in \{.15, .5, .85\}$ but otherwise default parameters for the package, and we use 10 iterations for LowRankEigenAlign. The plotted lines are loess smoothing using 30% of approximate neighbors for fitting.

**A.2 PPI Graph Statistics**

We use networks from the LVGNA project and 2 from the TAME project [12]. Statistics on these are in Table 1 where the TAME networks are in blue. These networks have been aligned with a variety of contemporary methods in [15], [16], [51], to make our results comparable with prior research. We remove any directional edges from the network before the enumerating triangles. As our methods are focused on triangle motifs, we only use networks with more than 150 triangles.

**TABLE 1**

| Graph Name | Vertices | Edges | Triangles |
|------------|----------|-------|-----------|
| worm_Y2H1  | 2871     | 10388 | 536       |
| worm_PHY1  | 3003     | 11002 | 692       |
| fly_Y2H1   | 7094     | 46712 | 2501      |
| yeast_Y2H1 | 3427     | 22696 | 9503      |
| human_PHY2 | 9996     | 79968 | 15177     |
| human_PHY1 | 8283     | 39394 | 19190     |
| yeast_PHY2 | 3768     | 27308 | 26925     |
| fly_PHY1   | 7885     | 72542 | 58216     |
| yeast_PHY1 | 6168     | 164736| 381812    |
| human_PHY1 | 16060    | 315298| 525238    |
| BioGRID_full_yeast_net | 5850 | 79458 | 347079 |
| BioGRID_full_human_net  | 14867 | 126593| 407650   |

**A.3 LowRankTAME Algorithm**

The following algorithm outlines how our LowRankTAME method (Alg 5) is implemented for our experiments. The method follows our explanation in §5.3 closely. Here we use a rank-revealing method inspired by the R-SVD (which does a QR factorization before an SVD to reduce the work in the SVD) and the structure of our problem. More on the asymptotic runtime of the R-SVD vs SVD can be found in [45, Figure 8.6.1].
Algorithm 5 Exact LowRankTAME with affine shift

Require: $k$-mode motif tensors $\bar{T}_A, \bar{T}_B$ for graph $A$ and $B$, mixing parameter $\alpha$, shift $\beta$, tolerance $\varepsilon$, weights $W = UV^T$

Ensure: Alignment matrix $X_{\text{best}}$

1: $C = \text{trace}((V^T V)(U^T U))$
2: $U_0 = U/\sqrt{C}; V = V_0/\sqrt{C}$ \Comment{Normalize first iterate}
3: for $\ell = 0, 1, \ldots$ until $|\lambda_{\ell+1} - \lambda_{\ell}| < \varepsilon$
4: \Comment{Exact LowRank SS-HOPM iteration}
5: \Comment{Interleave refers to rank}
6: $U_{\ell+1}(; i, j) = T_A(U_\ell(; i), U_\ell(; j))$
7: $V_{\ell+1}(; i, j) = T_B(V_\ell(; i), V_\ell(; j))$
8: \Comment{Estimate tensor-eval}
9: $\lambda_{\ell+1} = \text{trace}((V_{\ell+1}^T V_{\ell+1})(U_{\ell+1}^T U_{\ell+1})))$
10: \Comment{Apply affine shift in low-rank factors}
11: $U_{\ell+1} \leftarrow [\sqrt{\alpha} U_{\ell+1} \sqrt{\alpha \beta} U_{\ell+1} \sqrt{1 - \alpha} U_0]$
12: $V_{\ell+1} \leftarrow [\sqrt{\alpha} V_{\ell+1} \sqrt{\alpha \beta} V_{\ell+1} \sqrt{1 - \alpha} V_0]$
13: \Comment{Reduce to lowest rank terms}
14: $Q_U, R_U = QR(U_{\ell+1}); Q_V, R_V = QR(V_{\ell+1});$
15: $\hat{U}, \hat{V}, \hat{\Sigma}^T = \text{svd}(R_U R_U^T)$
16: $U_{\ell+1} \leftarrow Q_U \hat{U}; V_{\ell+1} \leftarrow Q_V (\hat{V} \hat{\Sigma})$
17: \Comment{Normalize}
18: $C = \text{trace}((V_{\ell+1}^T V_{\ell+1})(U_{\ell+1}^T U_{\ell+1}))$
19: $U_{\ell+1} \leftarrow U_{\ell+1}/\sqrt{C}; V_{\ell+1} \leftarrow V_{\ell+1}/\sqrt{C};$
20: $X_{\ell+1} = U_{\ell+1} V_{\ell+1}^T$
21: Set $t_{\ell+1}$ to be the score of a matching from $X_{\ell+1}$
22: return the matching of $X_{\ell+1}$ with the highest $t_{\ell+1}$