

Algorithms for Graph Similarity and Subgraph Matching

Danai Koutra

Computer Science Department
Carnegie Mellon University
dkoutra@cs.cmu.edu

Ankur Parikh

Machine Learning Department
Carnegie Mellon University
apparikh@cs.cmu.edu

Aaditya Ramdas

Machine Learning Department
Carnegie Mellon University
aramdas@cs.cmu.edu

Jing Xiang

Machine Learning Department
Carnegie Mellon University
jingx@cs.cmu.edu

December 4, 2011

Abstract

We deal with two independent but related problems, those of graph similarity and subgraph matching, which are both important practical problems useful in several fields of science, engineering and data analysis. For the problem of graph similarity, we develop and test a new framework for solving the problem using belief propagation and related ideas. For the subgraph matching problem, we develop a new algorithm based on existing techniques in the bioinformatics and data mining literature, which uncover periodic or infrequent matchings. We make substantial progress compared to the existing methods for both problems.

1 Problem Definitions and Statement of Contributions

1.1 Graph Similarity

Problem 1

Given: two graphs $G_1(n_1, e_1)$ and $G_2(n_2, e_2)$, with possibly different number of nodes and edges, and the mapping between the graphs' nodes.

Find: (a) an algorithm to calculate the similarity of the two graphs, which returns (b) a measure of similarity (a real number between 0 and 1) that captures intuition well.

Innovations: a) We develop a method involving belief propagation, unseen in literature, to solve this problem b) The method (and its fast linearized approximate version) gives extremely agreeable results c) Except for scalability, we know of no shortcomings of this method.

1.2 Subgraph Matching

Problem 2

Given: a graph time series, where there are T number of graphs.

Find: (a) An algorithm to find approximate subgraphs that occur in a subset of the T graphs. (b) Where the approximate subgraphs may not occur in the majority of the time points, but in local sections of the time series

Innovations: a) We develop a principled approach to selecting the important time components from which subgraphs should be mined. Our method is also tailored for the problem of selecting subgraphs in biological networks. For this, we use sparse PCA which has not been for this application domain. b) Scalability: Our method is both fast and scalable to real biological data (1000s of nodes). However, it has not been demonstrated whether it can scale to extremely large networks of more than 10 000 nodes. c) The method gives results that are easy to interpret and biologically sensible.

Disclaimer of interests intersecting with course project

Aaditya may use the PhoneCall dataset for his DAP. Danai is interested in graph similarity and belief propagation for research. Ankur has used tensors for his research, but in a different context. Jing has used CODENSE before, and is interested in improving it for research purposes. None of the authors have other course projects this term.

The following are the papers read for this course (refer to the numbering in the references section): Jing [21], [28], [22], Ankur [20], [18], [9], Aaditya [5], [26], [27], Danai [10], [14], [15].

2 Introduction

Graphs arise very naturally in many situations - examples vary from the web graph of documents, to a social network graph of friends, to road-map graphs of cities. Over the last two decades, the field of graph mining has grown rapidly, not only because the number and the size of graphs has been growing exponentially (with billions of nodes and edges), but also because we want to extract much more complicated information from our graphs (not just evaluate static properties, but infer structure and make accurate predictions). This leads to challenges on several fronts - proposing meaningful metrics to capture different notions of structure, designing algorithms that can calculate these metrics, and finally finding approximations or heuristics to scale with graph size if the original algorithms are too slow. In this project, we tackle several of these aspects of two very interesting and important problems, graph similarity and subgraph mining, which we broadly introduce and motivate in the next few paragraphs. First, we briefly introduce our two sample datasets (graphs), which will recur throughout this report.

2.1 Datasets

We call the PhoneCall dataset *PC*. Imagine users (indexed by phone number) being the nodes, and there is an edge between two nodes if they spoke to each other, letting the total call duration be the weight (or its inverse) on that edge. Summing up durations over a week or month would give us several weighted graphs on the same set of nodes. The dataset consists of over 340,000 people in one city using one telephone service provider. It contains a list of all calls made from people in the network to others in the same network over 6 months. We also have a list of SMS's sent within the network (call it dataset *SMS*), which we may also use. Other properties (like the distribution of call durations, anomaly detection, reciprocity, etc) of this data have already been analyzed in [2, 1, 24].

We call the YeastCellCycle dataset *YCC*. In this setting, the genes are the nodes of each graph and there exists an edge between two nodes if two genes interact. *YCC* is a sequence of graphs, one for each of 24 time points. These graphs are generated by using Time-Varying dynamic Bayesian networks algorithm [12] on yeast cell cycle microarray data. Thus, the graphs vary over the different phases of the cell cycle, resulting in different patterns for each of the first growth (*G1*), synthesis (*S*), second growth and mitosis (*G2M*) phases. Similar to the yeast dataset, the Drosophila dataset (*DP*) is also a series of graphs that vary over time. Again, genes are the nodes of each graph and the edges represent interactions. The dataset consists of 1 graph per time point for a total of 66 time points. The graphs are generated using a kernel-reweighted logistic regression method [19]. Drosophila undergo several

stages of development which are the embryonic stage, larval stage, pupal stage, and adult stage. These changing stages of development result in variations between the graphs especially during the transition time points. The YCC and DP datasets will be referred to as GeneInteraction (GI) datasets.

List of Abbreviations	
PC	PhoneCall dataset
SMS	SMS dataset
YCC	Yeast Cell Cycle dataset
G1	Growth Phase of Cell Cycle
S	Synthesis Phase of Cell Cycle
G2M	Growth and Mitosis Phase of Cell Cycle
DP	Drosophila dataset
GI	Gene Interaction datasets
BP	Belief Propagation

2.2 Graph Similarity

Our setting for graph similarity is as follows. We have two graphs on the same set of N nodes, but with possibly different sets of edges (weighted or unweighted). We assume that we know the correspondence between the nodes of the two graphs (like the people in *PC* don't vary across graphs). Graph similarity involves determining the degree of similarity between these two graphs (a number between 0 and 1). Intuitively, since we know the node correspondences, the same node in both graphs would be similar if its neighbors are similar (and its connectivity, in terms of edge weights, to its neighbors). Again, its neighbors are similar if their neighborhoods are similar, and so on. This intuition guides the possibility of using *belief propagation (BP)* as a method for measuring graph similarity, precisely because of the nature of the algorithm and its dependence on neighborhood structure. We delve more into details of BP in a later section.

This can be a great tool for data exploration and analysis. For example, we might conjecture that the *PC* graphs are quite different during the day and night and also vary significantly from weekday to weekend (talk to colleagues more in the day/weekdays, and family or close friends at night/weekends). On the other hand, we may expect graphs of two consecutive months to be quite similar (family, close friends, colleagues don't change on such a short time scale).

2.3 Subgraph Matching

Our setting for subgraph matching is as follows. Consider a series of T graphs, each of them over the same set of N nodes, but with possibly different edges (weighted or unweighted). Assume that we know the correspondence between the nodes (the genes in GI don't change across time points). Subgraph matching involves identifying the coherent or well-connected subgraphs that appear in some or all of the T graphs. For example, the T time points may include several cell cycles, each involving a growth, synthesis and mitosis phase. Different sets of genes (subgraphs) may interact (appear to be strongly connected) in some phases and disappear during other phases. Of course, there may be some genes that interact across all phases as well. We would like to identify these subgraphs, even if they appear in a small number of time points (appear and disappear periodically).

When studying developmental processes in biology, it is important to identify the subsets of genes that interact across time. These can represent functional processes that are specific to certain stages and thus can help us elucidate the dynamic biological processes occurring. Such an automated way of picking out interacting subsets of genes (without manually examining large graphs over many time points) would permit faster analysis with more uniform objectivity. In addition, if the algorithms are scalable, they might be applicable to other domains. For example, it would be interesting to see if it is able to select subgraphs from the PC dataset that are periodic in nature (night/day or weekday/weekend).

3 Survey

3.1 Graph Similarity

Graph similarity has numerous applications in diverse fields (such as social networks, image processing, biological networks, chemical compounds, and computer vision), and therefore there have been suggested many algorithms and similarity measures. The proposed techniques can be classified into three main categories: edit distance/graph isomorphism, feature extraction, and iterative methods.

Edit distance/graph isomorphism One approach to evaluating graph similarity is graph isomorphism. Two graphs are similar if they are isomorphic [17], or one is isomorphic to a subgraph of the other, or they have isomorphic subgraphs. The drawback of graph isomorphism is that the exact versions of the algorithms are exponential and, thus, not applicable to the large graphs that are of interest today. The graph edit distance is a generalization of the graph isomorphism problem, where the target is to transform one graph to the other by doing a number of operations (additions, deletions, substitu-

tions of nodes or edges, and reversions of edges). This method associates each operation with a cost and it attempts to find the sequence of operations that minimizes the cost of matching the two graphs.

Feature extraction The key idea behind these methods is that similar graphs probably share certain properties, such as degree distribution, diameter, eigenvalues [25]. After extracting these features, a similarity measure [5] is applied in order to assess the similarity between the aggregated statistics and, equivalently, the similarity between the graphs. These methods are powerful and scale well, as they map the graphs to several statistics that are much smaller in size than the graphs. However, depending on the statistics that are chosen, it is possible to get results that are not intuitive. For instance, it is possible to get high similarity between two graphs that have very different node set size, which is not always desirable.

Iterative methods The philosophy behind the iterative methods is that “two nodes are similar if their neighborhoods are also similar”. In each iteration, the nodes exchange similarity scores and this process ends when convergence is achieved. Several successful algorithms belong to this category: the similarity flooding algorithm by Melnik et al. [14] applies in database schema matching; this algorithm solves the “matching” problem, that is, it attempts to find the correspondence between the nodes of two given graphs. What is interesting about the paper is the way the algorithm is evaluated: humans check whether the matchings are correct, and the accuracy of the algorithms is computed based on the number of adaptations that have to be done in the solutions in order to get the right ones. Although we are not solving the exact same problem (we are only interested in assessing the similarity of two given graphs with given correspondence), the ideas behind our approach are very similar to the ones presented in this paper. Another successful algorithm is SimRank [10], which measures the self-similarity of a graph, ie. it assesses the similarities between all pairs of nodes in one graph. Again this is a different problem from ours, but it is based on the notion that similar nodes have similar neighborhoods. The algorithm computes iteratively all pairs similarity scores, by propagating similarity scores in the A^2 matrix, where A is the adjacency matrix of the graph; the process ends when convergence is achieved. Furthermore, a recursive method related to graph similarity and matching is the algorithm proposed by Zager and Verghese [27]; this method introduces the idea of coupling the similarity scores of nodes and edges in order to compute the similarity between two graphs; the majority of the earlier proposed methods focuses on the nodes’ scores. In this work, the node correspondence is unknown and the proposed algorithm computes the similarity between all pairs of nodes, as well as all pairs of edges, in order to find the mapping between the nodes in the graph. Finally, Bayati et al. in [3] proposed two approximate sparse graph matching algorithms using message passing algorithms. Specifically, they formalized the

problem of finding the correspondence between the nodes of two given graphs as an integer quadratic problem and solved it using a Belief Propagation (BP) approach. Their problem formulation assumes that one somehow knows which are the possible correspondences between the nodes of the two graphs (ie. because of intuition someone expects the 1st node of graph 1 to correspond to one the nodes {1,5,1024,2048} of graph 2).

3.2 Subgraph matching

The subgraph matching problem occurs when you have a set of graphs and you're trying to extract a subset of nodes that are highly connected. Specifically, we are interested in approximate subgraph matching, where the connectivity within each subset of nodes is not exactly consistent between graphs. This problem comes up in several applications such as gene networks, social networks, and designing molecular structures. We describe some of the current approaches below.

Approximate Constrained Subgraph Matching One possible approach is to build a declarative framework for approximate graph matching where one can design various constraints on the matching [28]. For example, one group worked on a method where the potential approximation had to satisfy constraints such as mandatory and optional nodes and edges. In addition, there were also forbidden edges which were not to be included in the matching. While this leads to a more well-defined search, the user must have detailed information on the pattern he wants to match. The drawback of this method is that many times, we are searching for subgraphs without any prior knowledge of the pattern to be found. Thus, we may not have the prior knowledge necessary to effectively use such a method.

SAGA: Approximate Subgraph Matching using Indexing In response to existing graph matching methods being too restrictive, a tool called Substructure Index-based Approximate Graph Alignment (SAGA) [22] was developed. This technique allows for node gaps, node mismatches and graph structural differences and does not require any constraints to be designed in advance. To summarize, an index on small substructures of the graphs are stored in a database. The query graph is broken up into small fragments and then the database is probed using a matching algorithm to produce hits for substructures in the query. The disadvantages are that one has to maintain a database of small structures and that it is query based. In applications such as graph mining in biological networks, it's possible that we want to extract subgraphs without having identified queries.

Mining Coherent Dense Subgraphs The method called mining coherent dense subgraphs (CO-DENSE) [9] is probably the most suitable method for our application. It constructs a summary graph formed with edges that appear greater than k times in the set of graphs. It then mines subgraphs within the summary graph using an algorithm that recursively partitions the graph into subgraphs based on normalized cut. However, the limitations of this algorithm is that it finds subgraphs from a static graph constructed from all time points. Thus, it is unable to capture interactions that occur locally to a few time points.

Tensor Analysis Unlike the previous methods, Sun et al. [21] formulate the problem in the context of tensors. The first and second modes of the tensor correspond to the adjacency matrix of a graph, while the third mode corresponds to time. A tensor decomposition (i.e. a generalization of matrix PCA) is proposed which can find “clusters” in the tensor (i.e. correlated dimensions within the same mode and across different modes). The authors also present an incremental algorithm that can be applied in the online setting. This work seems related to our goal of finding recurring subgraphs. However, it is not entirely clear whether a “cluster” found across multiple time points by the tensor decomposition is equivalent to a recurring subgraph in practice. A set of genes may be highly connected (clustered together) in two time point t_1 as well as t_2 but the set of edges that connect them in t_1 may be completely different than those that connect them in t_2 . It also remains to be seen how the method performs for sparse networks that are common in biology.

Graph Scope GraphScope [20] is another method for finding coherent clusters in graphs over time. GraphScope assumes the sequence of graphs G_1, \dots, G_n are bipartite. It then partitions this sequence of graphs into segments using an information theoretic criterion and then finds clusters within each segment. This is an interesting approach but is limited by the fact that since it partitions the sequence of graphs into segments, it can only find clusters in neighboring time points. However, we seek to find recurring subgraphs that may not occur in adjacent or nearby time points.

Subgraph Matching via Convex Relaxation Schellewald et al. [18] propose a method for subgraph matching based on convex relaxation. In their formulation, there is a large graph G_L and a smaller graph G_K , and the goal is to find G_K in G_L (the correspondence of the vertices from G_K to G_L is not known). However, they also assume the existence of a distance function $d(i, j)$ where i is a node in G_K and j is a node in G_L . Using both the adjacency structure of G_K, G_L , and the distance function d they construct a quadratic integer program that is then relaxed to a convex program. This approach is mathematically principled, but has the drawback that it does not find subgraphs in G_K that match

those in G_L but rather seeks to match all of G_K in G_L . Thus it cannot extend to finding recurring subgraphs in a series of graphs (more than 2).

4 Proposed Method

4.1 Graph Similarity

As mentioned before, one of the key ideas in graph similarity is that “a node in one graph is similar to a node in another graph if their neighborhoods are similar”. The methods that are based on this notion are iterative and consist of “score-passing” between the connected nodes. The concept of “score-passing” seems very related to one successful guilt-by-association technique, loopy belief propagation (BP).

The methods in the literature that solve the graph similarity problem yield results that are not very intuitive. As we will see in the experiments section, our method manages to capture both the local and global topology of the graphs; therefore, it is able to spot small differences between the graphs and give results that agree with intuition. Also, our method is general and can be applied to both connected and disconnected graphs - note that the proposed spectral methods are not applicable on disconnected graphs.

4.1.1 Belief Propagation

Loopy belief propagation is an iterative message passing algorithm for performing approximate inference in graphical models, such as MRFs. It uses the propagation matrix and a prior state assignment for a few nodes and attempts to infer the maximum likelihood state probabilities of all the nodes in the Markov Random Field. Table 1 gives a list of symbols used.

Symbol	Definition
$m_{ij}(x_k)$	message from node i to node j
$\phi_i(x_k)$	prior belief of node i being in state x_k
$b_i(x_k)$	final belief of node i being in state x_k
$N(i)$	neighbors of node i
η	normalizing constant

Table 1: Symbols and Definitions for BP

In belief propagation, nodes pass messages to their neighbors iteratively until convergence or for a

maximum number of iterations that is specified by the user. The two main equations of BP are the update equation for messages and beliefs:

$$m_{ij}(x_l) = \sum_{x_k} \phi_i(x_k) \cdot \psi_{ij}(x_k, x_l) \cdot \prod_{n \in N(i) \setminus j} m_{ni}(x_k)$$

$$b_i(x_k) = \eta \cdot \phi_i(x_k) \cdot \prod_{j \in N(i)} m_{ij}(x_k)$$

We skip the details of the method because of lack of space, but all the definitions and explanations can be found in [26].

In graphs with loops, belief propagation might not converge and so can sometimes perform poorly. However, in numerous applications, the algorithm is effective and converges quickly to accurate solutions.

4.1.2 Belief Propagation for Graph Similarity

Original-BP graph similarity: The first BP-based algorithm we implemented for graph similarity uses the original BP algorithm as it is proposed by Yedidia in [26]. The algorithm is naive and runs in $O(n^2)$ time. We assume that the given graphs have the same number of nodes, n , if in the given edge files there are nodes that are missing, we assume that these nodes form single node connected components. The algorithm is the following:

```

for  $i = 1 \rightarrow n$  do
  initialize node's  $i$  prior belief to  $p$ 
  run BP for graphs 1 and 2 and
  get the  $b_{i1}$  and  $b_{i2}$  vectors of final beliefs
   $sim\_score_i = \text{sim-measure}\{b_{i1}, b_{i2}\}$ 
end for
similarity of graphs  $\leftarrow \text{avg}\{sim\_score\}$ 

```

In our experimental setup, we set the prior belief p of the initialized nodes, as well as the entries of the propagation matrix (which in our case can be summarized by only one number), to 0.9. Now, let's focus on the similarity measure that is mentioned in the above algorithm. We tried using various similarity measures; the cosine similarity measure, that we had mentioned in our proposal, is not suitable in our case, because the belief vectors that we are comparing do not have similar sizes and, so, measuring the angle between the vectors is not informative about their distance in the n -dimensional space. As a distance metric we used the euclidean distance (d), and we devised different ways of assessing the similarity (s) of the vectors given their euclidean distance; the ultimate goal was to get a number between 0 and 1, where 0 means completely dissimilar, while 1 means identical:

- $s = \frac{1}{1+d}$
- $s = 1 - \sqrt{d/\max\{d\}}$

We report some preliminary experiments on synthetic graphs in the experiments section. In our experiments we used the second similarity function, because it seems to have more discriminative power than the first one, and also agrees with our intuition.

As we mentioned in the previous report, our goal was to devise a more clever algorithm that does fewer runs of BP in order to assess the similarity of the given graphs. Towards this goal, we tried a variation of the naive algorithm which randomly picks a small number of nodes to be initialized, but it does not perform well.

In the following subsection, we describe FABP, a scalable and fast approximation of BP, in the graph similarity setting. This algorithm is preferable to BP, because we only need to do a matrix inversion in order to find the final beliefs of all nodes and all “one-node” initializations, and thus the method scales as well as the FABP method. Given that the FABP-based method is better than the original-BP based method, we did not run more experiments using the latter method, nor did we try to devise a way to achieve smaller computational complexity than the naive algorithm by carefully choosing the initial nodes for running the BP algorithm.

Linearized BP (FaBP) graph similarity: The second BP-based algorithm we tried uses the FABP algorithm proposed in [11]. In this paper, the original BP equations are approximated by the following linear system:

$$[\mathbf{I} + a\mathbf{D} - c'\mathbf{A}]\mathbf{b}_h = \vec{\phi}_h$$

where h_h is the “about-half” homophily factor, ϕ_h corresponds to the vector of the prior beliefs of the nodes, \mathbf{b}_h is the vector of the nodes’ final beliefs, and a, c' are the following constants $a = 4h_h^2/(1 - 4h_h^2)$, and $c' = 2h_h/(1 - 4h_h^2)$. Moreover, \mathbf{I} is the identity matrix, \mathbf{A} is the adjacency matrix of the graph and \mathbf{D} is the diagonal matrix of degrees.

As we briefly mentioned in the previous subsection, the advantage of this method is that we do not have to run it n times, where n is the number of nodes in the graphs; inverting the matrix $\mathbf{I} + a\mathbf{D} - c'\mathbf{A}$ for each graph and comparing the matrices column-wise by a way similar to the one described in the *original BP graph similarity* algorithm is enough, given that we initialize the same nodes in both graphs – the initialization information is encoded in the ϕ_h vector. Moreover, FABP can trivially take into account the importance of each edge, given that this information can be found in the adjacency matrix, \mathbf{A} , of a graph. In order to see how our graph similarity algorithm fares with weighted graphs, we report some results in graphs with weights in 5.1.

The experimental setup is the following: the prior belief of the initialized node is set to 0.51 (note that the “uninitialized” nodes have prior belief 0.5), and the homophily factor is computed using the L_2 -norm bound described in [11], so that the FABP method converges. As we mentioned in the previous report, this method yields small beliefs (in the order of $< 10^{-4}$) for the nodes, and so, we need a similarity measure that takes into account the variance of the beliefs; comparing the beliefs as absolute values results in high similarity between the vectors of beliefs, although the graphs we are comparing have significant differences.

Next we discuss the similarity measures that have been proposed and give the similarity measure that performs best in our application.

Similarity Measures	Types of vectors	Applications	Properties
Dot Product	binary vectors	text mining	# of matched query terms in the document unbounded
	weighted vectors		sum of products of weights of matched terms favors long docs with many unique terms measures matched terms BUT not unmatched terms
Cosine Similarity		text mining	normalized dot product does not depend on the length of the vectors [0,1]
Jaccard/ Intersection	binary vectors	sparse datasets	normalized inner product dice is highly related to jaccard [0,1]
Tanimoto	binary/non-binary		ignores the 0-0 matches extends Jaccard similarity [0,1]
Dice / Sorensen/ Czekannowski/ Hodgkin-Richards	binary vectors sets X and Y IR: string similarity	sparse datasets	gives less weight to outliers than euclidean distance a bit different from jaccard
Squared Chord		paleontological & pollen data analysis	[0,2]
Pearson'S coefficient			spots correlations in variables
Russell Rao	binary		used when there are big differences in magnitudes
Other: Harmonic mean, Motyka, Kulczynski Ruzicka, Kumar-Hasserbrook			could not find useful information about these measures

Table 2: Similarity measures proposed in the literature

4.1.3 Discussion on similarity measures

There have been proposed numerous similarities measures, each of which has its advantages and disadvantages. The similarity measures can be classified based on the kind of data they are applicable to: binary, continuous, categorical. In the case we are studying, we are interested in finding the similarity measures between vectors of real numbers, which also happen to be small (order of $< 10^{-4}$). In Table 2 we present some similarity measures and their properties and/or applications. All the similarity measures that are applied to binary vectors cannot be used in the graph similarity setting we are studying, since the vectors we are comparing have real numbers.

One of the most commonly used similarity measures is the cosine similarity and the dot product. The former measure takes values in $[-1, 1]$ and the latter is unbounded. As you may see in table 2, the cosine similarity is usually used in text mining for document comparison. When the vectors of the documents are binary, the dot product represents the number of words that appear in both documents. The cosine similarity computes the angle between the document vectors, without taking into account their lengths. This is a desirable property in text mining, since we are interested in finding similar documents no matter what their size is, and the cosine similarity assigns higher similarity to vectors that point roughly to the same direction.

However, it seems that in our setting, the cosine similarity is not as effective, because the “beliefs” of all the nodes are very small numbers (order of $< 10^{-4}$), and therefore the angle between the belief vectors of two different graphs is always very small. We will see in the experiments section that the cosine similarity cannot recognize the differences in the graphs that we are comparing.

In the literature there have been proposed many ways of converting a distance metric (d) to similarity measure (s). The most prevalent ways are the following:

- $s = 1 - d$
- $s = \frac{1}{d}$
- $s = \frac{1}{1+d}$ for unbounded d
- $s = e^{-d^2}$

In Section 5.1, we present the graph similarity results using 4 different measures: normalized-euclidean-based, cosine similarity, sample linear correlation and Spearman’s rank correlation (finds non-linear correlation between variables). The first measure is based on the euclidean distance (d) and we used the third formula to convert it to a similarity measure in $[0, 1]$. The key is that we are using the

following normalized version of the euclidean distance:

$$d = \sqrt{\sum_{i=1}^n \frac{(b_{1,i} - b_{2,i})^2}{(b_{clique,i} - \phi_{\mathbf{h}_i})^2}}$$

. The idea underlying the normalization of the euclidean distance is that the nodes would be assigned the maximum belief if all nodes were connected to each other forming a clique, and the minimum belief if there were no edges in the graph, which would be described by a final vector of beliefs equal to the prior beliefs. So, the denominator of our distance measure represents the maximum possible difference (in beliefs) for each node, and is of the same order as the numerator. The latter fact boosts the discriminative power of our measure by “solving” the issue of comparing very small numerical values, as we will see in Section 5.1.

4.1.4 Other Methods for Graph Similarity

“Eigenvalue method” As mentioned in the survey, one big category of graph similarity methods is based on feature extraction; admittedly, a feature that contains important information about a graph is the vector of its eigenvalues. Let A_1 and A_2 be the adjacency matrices of graphs G_1 and G_2 respectively. Let also $L_1 = D_1 - A_1$ and $L_2 = D_2 - A_2$ be the laplacians of the graphs, where D_1 and D_2 are the corresponding diagonal matrices of degrees. In this method, we find the eigenvalues of the laplacians and we define the similarity between the graphs as:

$$sim = \sum_{i=1}^k (\lambda_{1i} - \lambda_{2i})^2,$$

where k is chosen s.t.

$$\min_j \left\{ \frac{\sum_{i=1}^k \lambda_{ji}}{\sum_{i=1}^n \lambda_{ji}} > 0.9 \right\}$$

for $j = 1, 2$ (corresponding to the two graphs that we are comparing) ; that is, we keep the top k eigenvalues that contain 90% of the energy. Notice that the similarity this method gives is unbounded, $[0, \infty)$, and values close to 0 mean that the graphs are very similar, while high values show dissimilarity.

Algorithms in [15] We are comparing our method to some of the feature extraction methods that Papadimitriou et al. propose in [15] - vector similarity and signature similarity methods. The authors claim that both methods capture the structural changes that occur in web graphs.

Other Our graph similarity method is not readily comparable to Zager’s method in [27], which couples the scores of the nodes and the edges, because we are solving a different problem. The algorithm

described in this paper finds the similarity between all the pairs of nodes in the two graphs and outputs the best matching between the nodes (general graph matching problem). However, in our setting, we assume that the correspondence between the graphs' nodes is known and we only assess how similar are the two graphs.

4.2 Subgraph Matching

Recall that the problem of subgraph matching is how to find approximate subgraphs that recur over a set of graphs. This problem is challenging from many perspectives. First, we are interested in a variety of different patterns of recurring subgraphs in the graph data. An example of a simple pattern is a subgraph recurring locally over a set of consecutive time points. However, more complex patterns are possible such as in the case of periodic data, where subgraphs can occur in one period and then repeat in the next period. We seek to design a method that can robustly find recurring subgraphs that follow a variety of patterns.

Moreover, different subgraphs may share overlapping groups of nodes. For example, a subgraph that recurs over time points 1 through 5 may contain nodes a, b, c, d, e , and f . Another subgraph that recurs over time points 6 through 10 may contain nodes d, e, f, g , and h . Thus, d, e , and f are shared among the two different subgraphs but a, b, c are unique to subgraph 1 while g and h are unique to subgraph 2. An even more complicated scenario could arise when a node is part of multiple subgraphs in the same time point in addition to across different time points. This problem is nontrivial and as a result, standard techniques are not equipped to handle overlapping subgraphs.

4.2.1 Method 1: Subgraph Mining with Hierarchical Summary Graph Construction and Spectral Clustering

Constructing a Summary Graph with Local Information The existing method CODENSE [9] constructs a summary graph consisting of edges that appear in more than k of the graphs (where k is a threshold). However, this strongly favors subgraphs that appear in most of the time points and thus cannot extract subgraphs that only recur in a small, but significant subset of the time points. For example, the algorithm would not discover a subgraph if it appeared in 3 time points towards the beginning of a time series, and then 3 time points half way through because such edges would not be included in the summary graph if $k > 6$. However, lowering k would include many more false positives. In addition, it is susceptible to noise because a particular edge would be included in the summary graph if it appeared more than 6 times randomly throughout the series. Thus, it may be beneficial to try to incorporate local information before mining for graphs.

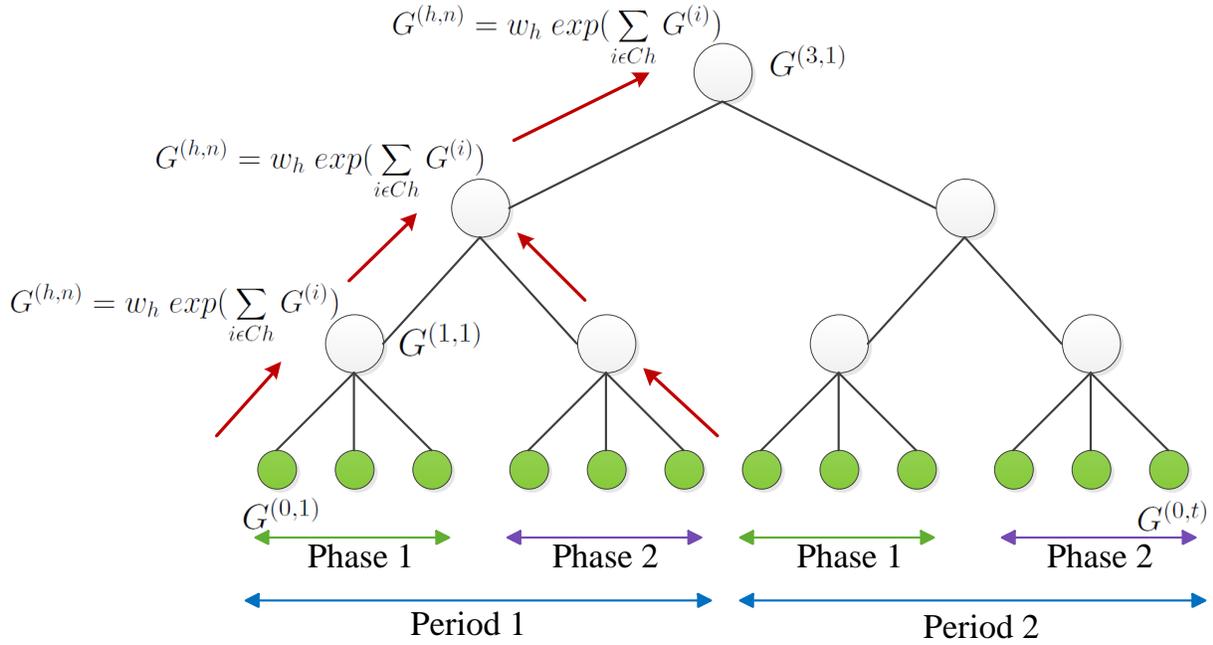


Figure 1: The hierarchical structure used to generate the summary graph. The leaf nodes represent the adjacency matrix of the graph at each time point for a total of t time points. Each node stores the weighted sum of the exponential of adjacency matrices stored at its children.

While creating multiple local summary graphs is a solution that was considered, it is not ideal because the amount of computation would increase with the number of summary graphs mined. Instead, we construct a single summary graph generated by a hierarchical structure over all graphs in the time series. The main objective is to include local information when selecting the edges in the summary graph. An example of how we compute the summary graph is shown in Figure 1. Each leaf node represents the original adjacency matrices of the graphs at each time point from $1..t$. We assume that the time series is divided into phases and that these divisions are prior knowledge. We show a simple example for illustration where the time series consists of 12 time points which are divided into 2 periods. In each period, phase 1 repeats and phase 2 repeats. This is important because we assume that certain subgraphs will be active in phase 1 while others will be active in phase 2. At each non-leaf node, we exponentiate the sum of all the child graphs of that node as written below:

$$G^{(h,n)} = w_h \exp\left(\sum_{i \in Ch} G^{(i)}\right); \quad (1)$$

where G is the adjacency matrix, h is the height of the node, and n is the index of the node at that

height. There is a different weight w for each height. We take the sum of all $G^{(i)}$, where the i^{th} graph is a child of the node n . We then exponentiate that value and weight it accordingly. We assign higher weights to the nodes of lower height to encourage the propagation of signal up to the root tree.

The resulting summary graph should then contain subgraphs that appear globally throughout the data and those that only occur in local consecutive segments. It will not contain noisy edges that appear frequently but randomly through the time series.

Mining Approximate Subgraphs Once the summary graph is obtained, there still remains the task of finding the subgraphs. For this we use spectral clustering [13]. There are many variants, but in general, spectral clustering works by computing the first K eigenvectors u_1, \dots, u_K , of some form of the graph Laplacian. These eigenvectors are then organized as columns of an N by K matrix U . Each node then "corresponds" to a row (1 by K vector) in U and k-means can be used to cluster the nodes in the new K dimensional space. If soft k-means is used, spectral clustering has the potential to find overlapping clusters which we explore in our synthetic experiments.

Existing Challenges There are several limitations of this approach that can arise. One problem involves mining the approximate subgraphs (clusters) in the summary graph. Since the edges in the summary graph may appear in different sets of time points it is possible that a cluster in the summary graph does not correspond to a cluster in any of the input graphs. Furthermore, if the hierarchy is asymmetric, i.e. different nodes at the same level have different numbers of children, then appropriate normalization needs to be done to ensure that the method is not biased towards the larger subtrees.

Effectively, using this method is not automatic and requires explicit domain knowledge. For instance, the user must specify the structure of the tree. The leaves must be grouped correctly according to the behavior of the time series. For example, for yeast cell cycle data, the user must know where in the time series a phase begins and ends. In addition, the user is also responsible for selecting the weights w_h , because it is not clear how to select them automatically. Because of these challenges, we chose to try other methods to do subgraph mining.

4.2.2 Method 2: Tensor Analysis for Mining Approximate Subgraphs

We investigate tensor analysis as an alternative method for mining subgraphs. A sequence of graphs of V vertices over T time steps can be represented using a $V \times V \times T$ tensor. The ability to represent relationships between genes and their relationship with time simultaneously is an advantage of using tensors. Tensor decompositions such as Tucker [23] and PARAFAC [8] can be used to characterize data

by providing the important components. The Tucker decomposition decomposes a tensor into a set of matrices and a core tensor however the results are difficult to interpret in the context of clusters. Thus, in this study we investigate PARAFAC, which decomposes a third order tensor \mathcal{A} as follows:

$$\mathcal{A} \approx \sum_{k=1}^K \lambda_k \mathbf{U}_{:,k}^{(1)} \circ \mathbf{U}_{:,k}^{(2)} \circ \mathbf{U}_{:,k}^{(3)} \quad (2)$$

where K is specified by the user and determines the quality of the approximation. $\mathbf{U}_{:,k}^{(i)}$ corresponds to the k^{th} column of matrix $\mathbf{U}^{(i)}$ and \circ corresponds to outer product. Thus PARAFAC is expressing \mathcal{A} as a sum of K rank one tensors.

One can then infer clusters as follows: let τ be a threshold and let $S_{\tau,k}^{(1)}$ be a set of indices such that $j \in S_{\tau,k}^{(1)}$ if and only if $\mathbf{U}_{j,k}^{(1)}$ is larger than τ (and similarly for $S_{\tau,k}^{(2)}$ and $S_{\tau,k}^{(3)}$). Then we can conjecture that the interactions of the vertices in $S_{\tau,k}^{(1)}$ and $S_{\tau,k}^{(2)}$ form an approximate subgraph for time points in $S_{\tau,k}^{(3)}$.

In the results section, we show that this approach works well on synthetic data to extract both non-overlapping as well as overlapping subgraphs. However, it is also challenging to apply this to real networks. One of the limitations of PARAFAC is that it is sensitive to the selection of the parameter k , which specifies the number of components that the decomposition should produce. PARAFAC differs from an SVD decomposition on 2D matrices in that it does not have the same properties. For SVD, the decomposition is equivalent to $\lambda_1 u_1 v_1^T + \lambda_2 u_2 v_2^T \dots + \lambda_n u_n v_n^T$ and regardless of how many components are requested, the first component returned will always be the same, as will the second, as will the m^{th} . However, the m^{th} component returned by PARAFAC differs depending on how many components, K , are selected. As we show in the succeeding section, if the selected K is correct, then PARAFAC will divide the data into the right clusters. However, if K is incorrect, the resulting decomposition is different and becomes difficult to interpret. In real settings, k is an unknown parameter which means that sensitivity to K is problematic. We hoped to amend this issue by using sparse PARAFAC [16] because imposing sparsity and non-negativity on the resulting components should provide more stable extraction of clusters. However, the optimization is non-convex and thus, the solution can change dramatically for small changes in the regularization parameters λ . In addition, it does not scale to the YCC dataset which is approximately $3000 \times 3000 \times 24$.

4.2.3 Method 3: Matricization and Sparse Principal Component Analysis

Because of the limitations of the previous methods, we changed the representation of the network and reformulated the problem into two major tasks. Instead of decomposing a 3D tensor, we matricize the

data so that it can be represented in a 2D framework. Each row in the matrix represents an edge and each column represents a time point. For the YCC dataset, because the network is sparse, we have a total of 66740 edges. Thus the data can be represented by a sparse matrix X of 66740×24 . The two major steps in this method are described below:

1. Find the Principal Components in Time In this step, we want to find components in time (vector of size $1 \times T$) which indicate where a significant fraction of edges occur. For instance, we might get the following shown in Figure 2 where the 1^{st} component shows that there is a concentration of edges in the early section of the time series, the 2^{nd} component shows activity towards the latter part of the time series etc. We can then extract the edges during these time intervals and mine them for subgraphs.

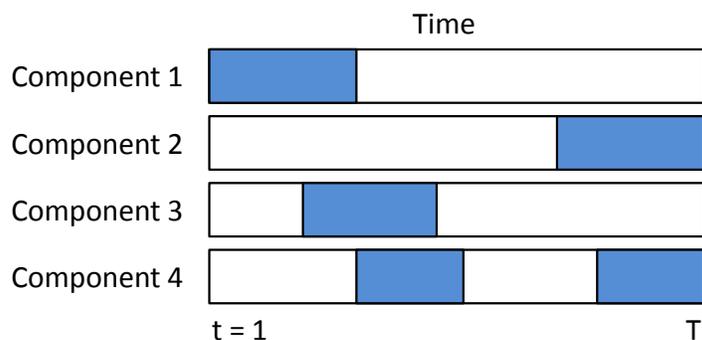


Figure 2: We use sparse PCA to find the principal components in time which shows where a significant fraction of edges occur. For instance, component 1 shows that many edges occur during an interval in the beginning of the time series, component 2 shows a significant number of edges occur in the latter section, component 3 shows edges in the middle of the time series, and component 4 shows that a significant amount of edges repeat in two intervals.

In general, we would use Principal Component Analysis (PCA) for this task since it finds linear combinations of the variables that correspond to directions of maximal variance. However, the principal components are usually linear combinations of all variables. Thus all weights in the linear combination (called loadings) are non-zero. However, in our application, the coordinates have a physical interaction, they represent edges connecting two vertices. Thus we need to be able to interpret the principal components and Sparse Principal Component Analysis (SPCA) allows us to do so because it involves very few non-zero coordinates. There are various different approaches available to perform SPCA including a method that solves a convex relaxation [6]. However, we use a greedy approach [6] because it provides the whole regularization path in $O(T^3)$ which allows us to choose the regularization parameter ρ easily.

The greedy approach works by starting with a set of cardinality one that contains the element of maximum variance. The algorithm then iteratively adds to this set by adding the variable with maximum variance contribution to generate the whole regularization path.

2. Subgraph Construction and Clustering Unlike PCA, SPCA only extracts one component at a time. After we find a time component, we extract all the edges that appear more than some threshold m percentage of times during that time component. The set of edges can be converted into an adjacency matrix that represents a (possibly not connected) edge set. We then delete these edges from the matrix and rerun SPCA to get a new sparse component. To make the results more interpretable, one can perform clustering on each edge set that corresponds to a particular time component. We used the Markov Cluster Algorithm (MCL) package for cluster extraction [7] because of its speed and popularity in computational biology applications. While there are advantages to simultaneously performing edge set extraction and clustering, we believe that treating each task separately leads to a faster and simpler algorithm.

The algorithm terminates when no edge appears more than m percentage of the times in the time component SPCA has most recently extracted. Thus K , the number of components that the user requests just serves as an upper bound on the number of components that will be extracted.

5 Results

5.1 Graph Similarity

In this section we try to answer the following questions:

1. how do our proposed algorithms compare to other methods reported in the literature?
2. which similarity measure gives the most intuitive results?
3. is the FABP-based graph similarity approach scalable?

5.1.1 Synthetic Tests

We note that the results mentioned in the following tables (3 and 6 - 8) are extremely intuitive, and here we provide short explanations why.

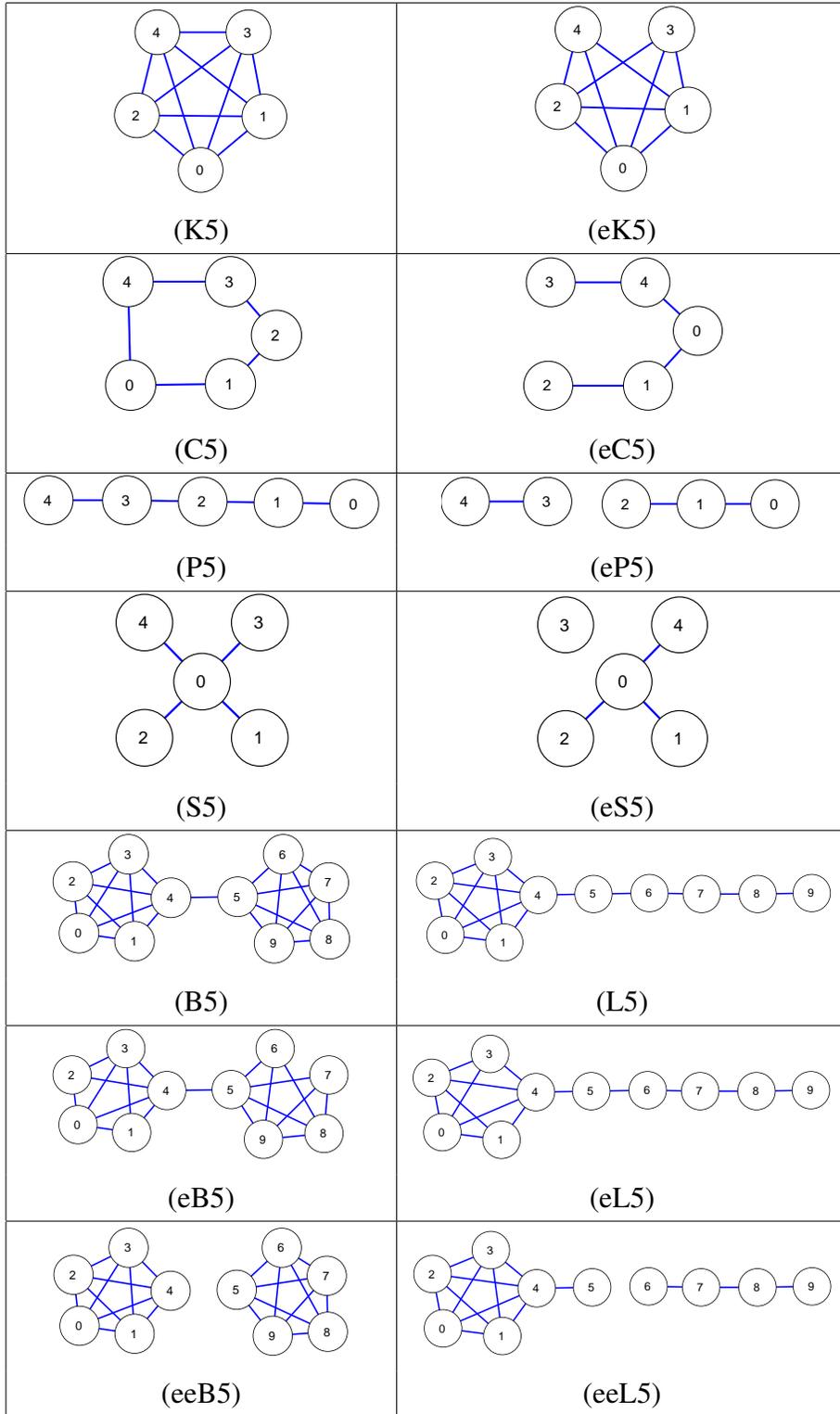


Figure 3: [BP TEST GRAPHS] K-complete, C-cycle, P-path, S-star, B-barbell, L-lollipop, R-random

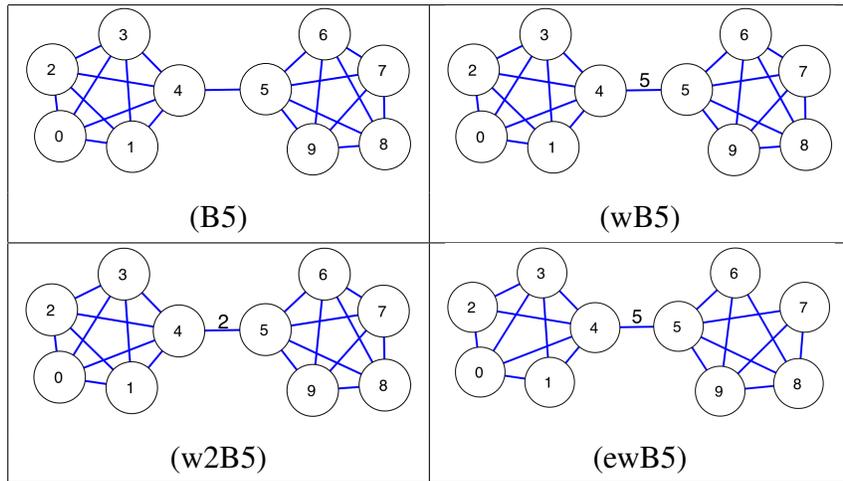


Figure 4: [WEIGHTED GRAPHS] The edges that are not annotated have unit weights.

Graph 1	Graph2	BP-Similarity
K5	eK5	0.97
K100	eK100	1
P5	eP5	0.6
P100	eP100	0.97
C5	eC5	0.62
C100	eC100	0.97
S5	eS5	0.62
S100	eS100	0.84
L5	eL5	0.97
L5	eeL5	0.51
B5	eB5	0.97
B5	eeB5	0.40
C5	S5	0.65
C100	S100	0.51

Table 3: Results of BP-similarity synthetic experiments

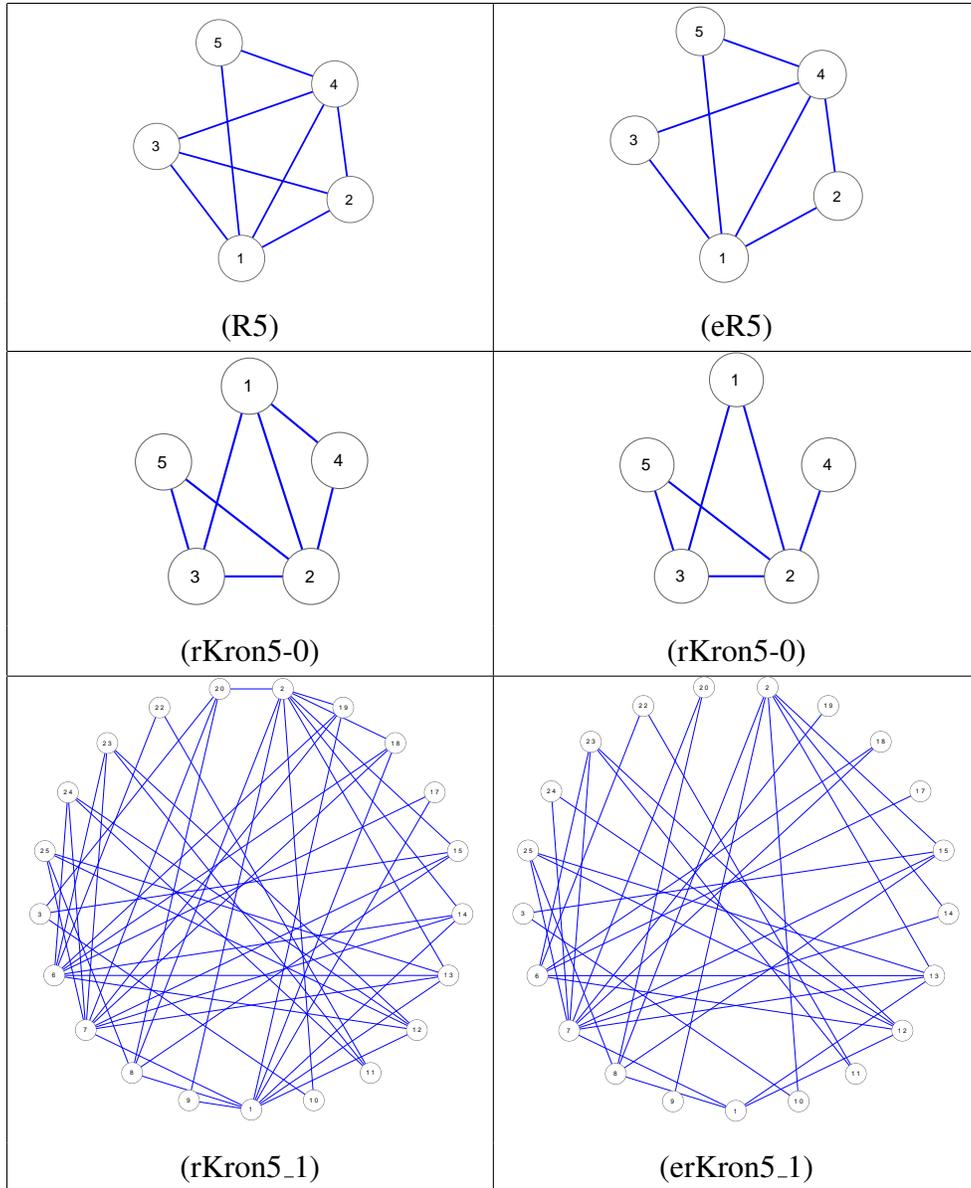


Figure 5: [Random Kronecker graphs] Random graphs, and their Kronecker graph of the first power. Here, instead of deleting an edge from the powered graph, we delete an edge from the base graph and take its power, so the two powered graphs differ in many edges but are broadly similar.

Graph	# Nodes	# Edges
R5	5	8
R25	25	238
R50	50	967
R75	75	2171
R100	100	3860
kron3_0	3	2
kron3_1	9	8
kron3_2	27	32
kron3_3	81	128
kron3_4	243	512
rKron5_0	5	7
rKron5_1	25	49
rKron5_2	125	343
rKron5_3	625	2401
rKron5_4	3125	16807
erKron5_0	5	6
erKron5_1	25	36
erKron5_2	125	216
erKron5_3	625	1296
erKron5_4	3125	7776

Table 4: Statistics for the random, kronecker, random-kronecker and edited-random-kronecker graphs.

Prefix	Short Definition	Description
e*	-1 edge	1 edge removed from the graph with the same name, but without prefix
ee*	- 2 edges	1 edge removed from the graph with the same name, but with the “e” prefix
e5*	-5 edges	5 edges removed from the graph with the same name, but with the “ee” prefix

Table 5: Definition of prefixes in the names of the random and kronecker graphs.

Graph 1	Graph 2	“Euclidean” based Sim.	Cosine Similarity	Linear Correlation	Spearman’s Correlation	Eigenvalue Method
K5	eK5	0.7663	0.9955	0.9899	0.7374	4.0000
P5	eP5	0.7338	0.9799	0.9754	0.7929	0.9098
C5	eC5	0.7689	0.9850	0.9812	0.7976	2.0000
S5	eS5	0.7257	0.9766	0.9593	0.8798	2.0000
L5	eL5	0.8872	0.9978	0.9971	0.9525	2.2422
L5	eeL5	0.8822	0.9970	0.9968	0.8656	0.6501
B5	eB5	0.8928	0.9981	0.9977	0.9756	4.0000
B5	eeB5	0.8558	0.9972	0.9970	0.9494	2.8953
wB5	w2B5	0.6448	0.9830	0.9729	0.9980	33.2884
wB5	eeB5	0.5645	0.9501	0.9302	0.8979	86.5037
wB5	B5	0.6029	0.9696	0.9540	0.9724	57.7474
w2B5	ewB5	0.6205	0.9796	0.9667	0.9705	37.2884
K100	eK100	0.9852	1.0000	1.0000	-0.1094	0.0000
P100	eP100	0.9837	1.0000	1.0000	0.9780	0.0459
C100	eC100	0.9838	1.0000	1.0000	0.9736	0.0676
S100	eS100	0.9486	0.9999	0.9999	0.8184	1.0000
C5	S5	0.5921	0.9550	0.9353	0.5677	9.0557
C100	S100	0.3984	0.9905	0.9929	0.1797	9489.3670

Table 6: Similarity scores for simple synthetic graphs: Columns 3-6 refer to the similarity scores obtained by applying the FABP method with different similarity measures. The last column presents the similarity score obtained by the “eigenvalue” method.

Graph 1	Graph 2	“Euclidean” based Sim.	Cosine Similarity	Linear Correlation	Spearman’s Correlation	Eigenvalue Method
R5	eR5	0.7690	0.9914	0.9866	0.9169	2.0000
R5	eeR5	0.6654	0.9843	0.9815	0.8374	4.3573
R5	e5R5	0.4848	0.9329	0.9541	0.9288	27.1716
eR5	e5R5	0.5295	0.9509	0.9555	0.8910	18.1716
R25	eR25	0.9520	0.9999	0.9999	0.9790	0.4659
R25	eeR25	0.9228	0.9998	0.9997	0.9641	1.1386
R25	e5R25	0.8652	0.9995	0.9994	0.9391	4.7201
eR25	e5R25	0.8846	0.9996	0.9995	0.9484	2.9115
R50	eR50	0.9766	1.0000	1.0000	0.9952	0.2426
R50	eeR50	0.9654	1.0000	1.0000	0.9933	0.6353
R50	e5R50	0.9296	0.9999	0.9999	0.9783	2.8650
eR50	e5R50	0.9424	0.9999	0.9999	0.9868	2.0348
R75	eR75	0.9848	1.0000	1.0000	0.9983	0.1899
R75	eeR75	0.9767	1.0000	1.0000	0.9973	0.4266
R75	e5R75	0.9538	1.0000	1.0000	0.9913	1.9375
eR75	e5R75	0.9592	1.0000	1.0000	0.9926	1.2943
R100	eR100	0.9886	1.0000	1.0000	0.9992	0.1146
R100	eeR100	0.9809	1.0000	1.0000	0.9982	0.3031
R100	e5R100	0.9659	1.0000	1.0000	0.9956	1.3292
eR100	e5R100	0.9726	1.0000	1.0000	0.9971	0.9454

Table 7: Similarity scores for random graphs: Columns 3-6 refer to the similarity scores obtained by applying the FABP method with different similarity measures. The last column presents the similarity score obtained by the “eigenvalue” method.

Graph 1	Graph 2	“Euclidean” based Sim.	Cosine Similarity	Linear Correlation	Spearman’s Correlation	Eigenvalue Method
kron3_2	ekron3_2	0.9560	0.9995	0.9994	0.9993	1.4193
kron3_2	eekron3_2	0.9192	0.9989	0.9989	0.9973	2.8385
kron3_2	e5kron3_2	0.8262	0.9972	0.9971	0.9547	7.3252
ekron3_2	e5kron3_2	0.8499	0.9977	0.9976	0.9554	5.0110
kron3_3	ekron3_3	0.9836	1.0000	1.0000	1.0000	2.8340
kron3_3	eekron3_3	0.9674	0.9999	0.9999	0.9999	3.3222
kron3_3	e5kron3_3	0.9216	0.9998	0.9998	0.9837	5.2987
ekron3_3	e5kron3_3	0.9338	0.9998	0.9998	0.9838	2.3285
kron3_4	ekron3_4	0.9934	1.0000	1.0000	1.0000	1.6498
kron3_4	eekron3_4	0.9847	1.0000	1.0000	0.9944	2.6498
kron3_4	e5kron3_4	0.9662	1.0000	1.0000	0.9944	6.4609
ekron3_4	e5kron3_4	0.9719	1.0000	1.0000	0.9944	4.8111
rKron5_0	erKron5_0	0.7878	0.9911	0.9861	0.8622	1.5147
rKron5_1	erKron5_1	0.7652	0.9940	0.9944	0.8935	45.394
rKron5_2	erKron5_2	0.6951	0.9981	0.9982	0.9515	1462.3
rKron5_3	erKron5_3	0.6510	0.9995	0.9995	0.9805	38258.8
rKron5_4	erKron5_4	0.6603	0.9998	0.9998	0.9919	925825

Table 8: Similarity scores for kronecker graphs: Columns 3-6 refer to the similarity scores obtained by applying the FABP method with different similarity measures. The last column presents the similarity score obtained by the “eigenvalue” method. The first column is our preferred score measure.

Synthetic Graphs Before we move on to the explanations, we briefly describe the synthetic graphs that we generated for our experiments. We generated several graphs with few nodes and specific “shapes” (Fig. 3 and 5), so that we can understand how each similarity algorithm behaves. Apart from those graphs, we also generated some random and kronecker graphs, more information for which is provided in Tables 4 and 5. The random graphs have each possible edge with probability 0.5. The initial matrix we used to generate the kronecker graphs is

$$K1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

and $kron3_2 = K_1 \otimes K_1$, $kron3_2 = K_1 \otimes K_1 \otimes K_1$, and $kron3_4 = K_1 \otimes K_1 \otimes K_1 \otimes K_1$. The meaning of the prefixes is explained in Table 5.

Finally, we also generated random kronecker graphs, $rKron5_i$, which start from a Erdos-Renyi random graph on 5 nodes and probability of edge p , and take the i -th kronecker power of those graphs. Here, $erKron5_i$ is generated by changing the probability of edge p of the base graph, and taking i -th kronecker power of this altered base graph. Hence, the larger powers may differ by many more edges due to propagate changes in the kronecker powering.

Explanations of results in Table 3 (original BP method) Removing one edge from $K5$ hardly changes anything, hence we expect a high similarity to $eK5$. Removing one edge from $P5$ disconnects the graph, and so we expect a high dissimilarity from $eP5$. Removing an edge from $S5$ isolates one node, so we expect low similarity. However, for all of these cases, moving to a graph on 100 nodes increases the similarity. One can intuitively understand this by seeing that a missing edge is far more noticeable when there are few edges and the graph is small, compared to when there are many edges and the graph is large. Also, as expected, there is a big difference between $S5$ and $C5$, and between $S100$ and $C100$.

Moving on to more complicated examples, we see how belief propagation captures the right intuition about which edges are more or less important than others. Looking at the lollipop graph $L5$ (a $K5$ connected to a $P5$), we see that deleting an edge from the $K5$ part made almost no difference, as noted with the high similarity to $eK5$, but deleting an important edge like one in the $P5$ part made a

big difference as noted by the small similarity to $eeK5$. Similarly, looking at the barbell graph (a $K5$ connected to a $K5$ by one edge), we see that deleting one of the $K5$ edges makes little difference (to $eB5$), but deleting the connecting edge between the $K5$ s causes a big decrease in similarity (to $eeB5$).

Explanations of results in Table 6 (FABP method) Let’s focus on the first column, which gives the similarity scores when we use our proposed normalized euclidean based similarity measure. In the first four rows of the table we observe that removing one edge from the clique, path, circle and star of 5 nodes yields medium similarity, and the similarity is lower when the removal of the edge creates disconnected components. The circle and the star consisting of 5 nodes have roughly the complementary edges and this is reflected by their similarity score (0.59). The corresponding similarity scores of graphs of 100 nodes are higher, which is intuitive as the “importance” of each edge in a big graph is smaller than its importance in a small graph with few edges.

In the case of the lollipop, although our method assigns smaller similarity to the pair $L5 - eeL5$ than to $L5 - eL5$, the scores are very close. The barbell graph with one missing edge in the clique ($eB5$) has higher similarity to $B5$, than the barbell graph with the missing bridge to $B5$, which is in accordance with our intuition. Moreover, the barbell graph with bridge of weight 5 is more similar to the barbell graph with weight 2, than to the graph without a bridge ($eeB5$). Note that the weight of the bridge represents its “importance”, and this is captured by our proposed graph similarity method; $B5 - eeB5$ have 0.86 similarity, but $wB5$ (bridge weight 5) is 0.56 similar to $eeB5$. Also, decrease in the weight of the barbell graph ($wB5$) by 4 units leads to 0.6 similarity. As expected, when we change both the weight of the barbell graph and remove an edge in the clique, we obtain smaller similarity than when we change only the weight (see pairs $wB5 - w2B5$ and $w2B5 - ewB5$).

Explanations of results in Table 6 (other similarity measures) The cosine similarity and the linear correlation capture the differences of the graphs we are comparing in the same way that the normalized-euclidean based similarity measure does; that is, the order of the pairs by similarity scores is the same when using these similarity measures. Notice that similarity 1.0000 is actually it is a bit smaller than 1 (say 0.999999..). What we observe in this table, is that the cosine similarity, does not have as big discriminative power, as our method. Also, linear correlation has better discriminative power than the cosine similarity, but still smaller than our method. Notice for example that these two methods do not “penalize” much the creation of disconnected components in a graph (see $B5-eeB5$), unless we explicitly state the “importance” of the bridge by assigning a non-unit weight (capture of

disconnected components is a desired property for a graph similarity algorithm).

Spearman’s correlation gives a slightly different order of pairs with respect to their similarity than the previous three methods, and its similarity scores are not always intuitive; for instance the pair $K100$ and $eK100$, which differ only in one edge have negative correlation/similarity! Finally, note that the eigenvalue method assigns small scores to similar graphs and high scores to dissimilar graphs. Although the results are intuitive when the graphs under comparison are connected, this method has 2 flaws: a) in case one of the graphs is disconnected, the results are not at all intuitive, since the eigenvalues correspond only to the biggest component (they are not global features any more). Thus, the graph similarity algorithm compares the graphs locally, which is not desirable. b) the similarity scores are unbounded, which is not desirable again, since a single similarity score cannot really inform us how much similar or dissimilar are the graphs (we only know that 0 means absolute similarity, but we don’t know how much the (dis)similarity of the graphs can become.

Explanations of results in Table 7 As mentioned above, the cosine similarity and the linear correlation have small discriminative power and fail to discern small differences in graphs with more than 25-50 nodes. Moreover, the Spearman’s correlation has better discriminative power than the previous two methods, but the results are not always intuitive; the pairs $R5 - eeR5$ and $R5 - e5R5$ get similarity scores 0.84 and 0.93 respectively, although in the first case they differ in two edges, and in the second case they differ in 3 more edges (the same as the first pair and 3 extra edges). Finally, the eigenvalue method is not satisfactory in the case of disconnected graphs as it only captures local structural differences.

On the other hand, we observe that our proposed method is able to spot small differences in graphs and its performance is not affected by disconnected components. As we expected, the similarity drops as we increase the number of missing edges, and the “importance” of each edge in a graph depends on its size (the larger the graph is, the smaller is the importance of each edge); we obtain higher similarity in large graphs that have the same number of missing edges as smaller graphs (see for example $R5 - eR5$ and $R100 - eR100$).

Explanations of results in Table 8 The observations are similar to the ones presented for the random graphs in Table 7. Let’s consider the graphs $R25$ with 238 edges and the corresponding graphs containing 1-5 fewer edges, as well as the kronecker graph $kron3_2$ which consists of 27 nodes, but 32 edges (see info in Table 4). Notice that the same difference in the number of edges for the random

and the kronecker graphs results in slightly higher similarity measures for the random graphs; this is normal, because the random graphs with *almost* the same number of nodes are denser, and therefore the contribution of a missing edge to the similarity score is smaller than the contribution of a missing edge in a sparse graph. Of course the similarity score depends on the “type” of edge that was removed in each case - if an edge removal leads to the creation of a new disconnected component in one graph, then our method “penalizes” this removal in a heavier way than it “penalizes” another edge.

5.1.2 Comparison to Papadimitriou-Molina

We obtained the code for two methods proposed in [15]. These were Perl scripts mainly under the names of compare-shingles and compare-edit-distance. With all due respect to the authors, the code obtained was of low standard. To be specific, none of the perl scripts worked without commenting an import statement. After that, the compare-edit-distance gave a compile error, and this was because `'!='` was used instead of `'ne'`. On correcting this, it was found that compare-edit-distance always calculated the edge intersection to be zero. This was because the edges from the first graph were stored as `'node1.-.node2'` but were searched for as `'node1.node2'` with a missing hyphen. On correcting this, it was found that it calculated the total number of nodes wrongly. The reason for this was also found, but not corrected, and it is not worth going into the details here.

Both the methods perform poorly with respect to our method, as seen in 9 (we omit the second one due to its mistakes). This is because both are broadly based on calculating node intersections and unions, and edge intersections and unions, without taking into account the local importance of different edges. For example, when tested on the Lollipop or Barbell graphs introduced earlier, they give the same similarity whether eB5 or eeB5 is used (similarly when eL5 or eeL5 is used), because the number of common edges and nodes is the same, even though a more important edge is missing from eeL5 and eeB5. We thus conclude that our method captures intuition better than simplistic formula-based methods.

5.1.3 Scalability of FaBP and Evaluation on Phone Call data set

We regret to find that the FaBP algorithm in its present state is not scalable. As we see in table 10 and figure 6, we find that the maximum size of the graph that it is able to handle is a few thousand nodes, with tens of thousands of edges. Hence, we were unable to test it on larger graphs like those generated in the phonecall dataset. For example, even 6 hours of PC dataset, on calls that last more than 120s,

Graph 1	Graph2	Shingles-Similarity
C5	eC5	0.8
S5	eS5	0.75
L5	eL5	0.93
L5	eeL5	0.93
B5	eB5	0.95
B5	eeB5	0.95

Table 9: Results of the Papadimitriou-Molina Shingles similarity measure

generated an extremely large sparse graph of over 100,000 nodes and edges. Reducing it to one hour still generated extremely sparse graphs over 20,000 nodes and edges, and we felt it did not make sense to shrink the size further. However, we extensively tested it on synthetic and random graphs, to show that it is, in principle, a very good method for testing graph similarity.

Nodes	Graph1 Edges	Graph2 Edges	Adjacency1 Inverse (s)	Adjacency2 Inverse (s)	K_n Inverse (s)	Similarity calculation (s)	Total Time (s)
100	15	15	0.0176	0.0007	0.0616	0.1880	0.27
500	226	218	0.0028	0.0020	0.0118	0.0716	0.088
1000	867	840	0.0074	0.0081	0.0764	0.1377	0.23
2000	3553	3437	0.2480	0.2382	0.3577	0.4818	1.33
3000	8160	7890	1.43	1.26	1.19	1.55	5.44
4000	14221	13751	3.17	3.024	2.27	3.77	12.23
5000	22341	21602	6.00	5.88	4.69	4.31232	20.88
6000	32383	31295	8.57	10.66	4.89	5.99	30.11
7000	44180	42715	15.01	24.69	34.436	9.48	83.61
8000	57736	54707	102.15	120.76	50.74	39.89	313.55
9000	73056	69192	257.30	201.11	123.99	89.12	671.52

Table 10: Scalability test on random graphs. All values represent time taken for that column's operation in seconds. Tested on machine with Intel Core I7, CPU 860 @ 2.8Ghz.

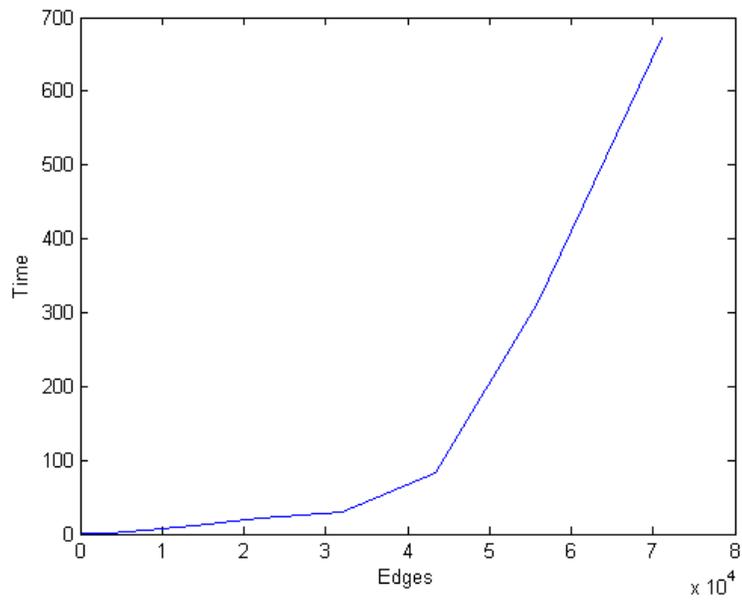
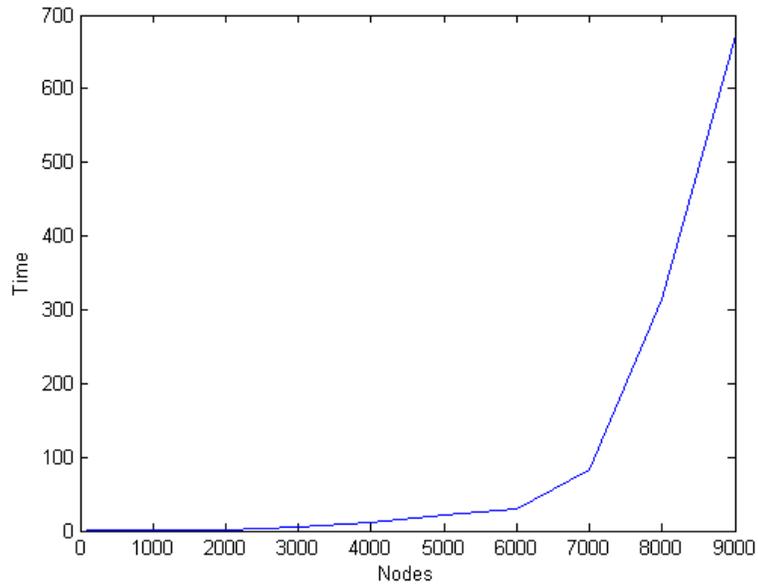


Figure 6: [Scalability of FaBP on Random Graphs] Time (sec) vs Edges/Nodes

5.2 Subgraph matching

5.3 Subgraph Matching Simulations with Spectral Clustering

In order to test the hierarchical summary graph approach, we simulated graphs in a time series with the phase and period format shown in Figure 1. For each time point, we generated the adjacency matrix of the graph and for each period, we selected 3 disjoint clusters of 100 nodes. If two nodes are in the same cluster, then we generate an edge with probability 0.80. Otherwise an edge is generated with probability 0.20. The resulting simulated graphs are shown in Figure 7.

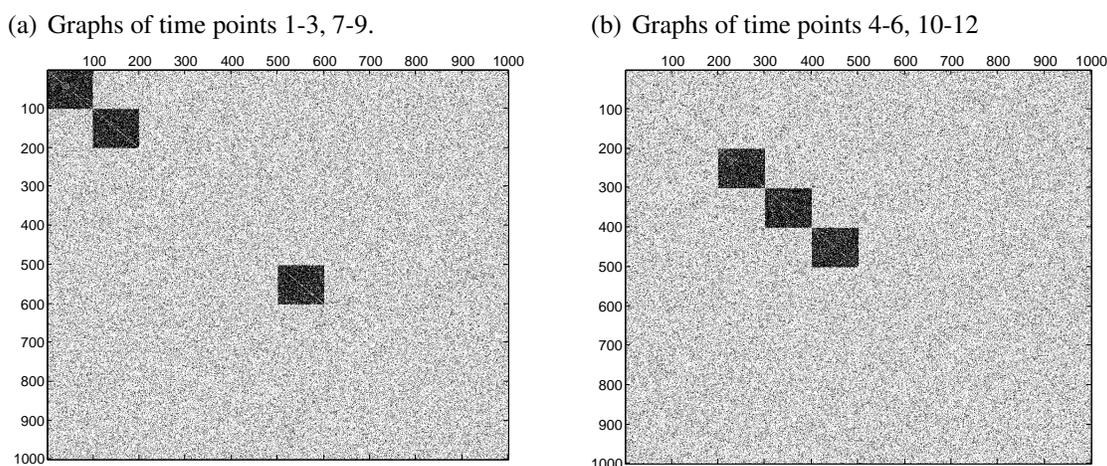


Figure 7: Shown above are the adjacency matrices of the graphs generated. Those time points that are part of phase 1 (see Figure 1).

The summary graph constructed by propagating the edge signals upwards throughout the tree is shown in Figure 8. The summary graph is able to incorporate both sets of clusters in each phase. If we then apply spectral clustering to the summary graph, it correctly puts clusters the 6 different groups of nodes and then places all other nodes into a separate clusters.

But often the data is more complex and nodes may be a member of multiple clusters at different time points. We simulate this scenario and perform soft spectral clustering. The time points listed in Figure 9a) exhibit a particular clustering pattern whereas, the time points that are part of phase 2 (Fig. 9b) are clustering differently. Soft spectral clustering does a decent job at finding the clusters as shown in Figure 10a). Each column represents a different cluster and the rows indicate membership of the nodes. The dark orange areas indicate a higher probability of membership whereas the blue areas indicate low probability of membership. This matches what we see in the summary graph (Fig 10b). For example, in column 2 of Figure 10a), the first 100 nodes have a high probability of membership

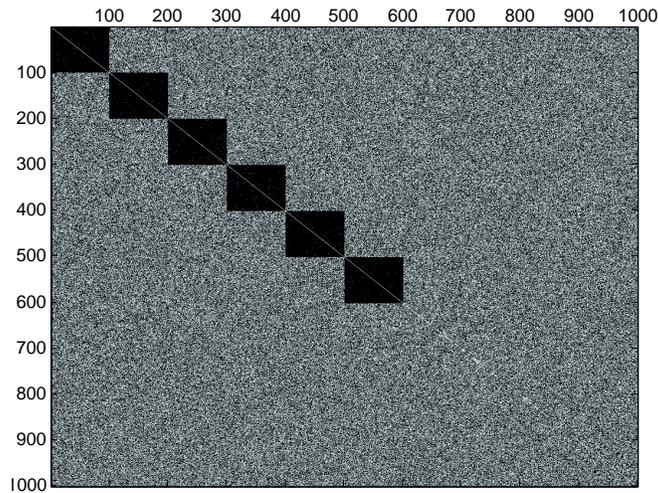
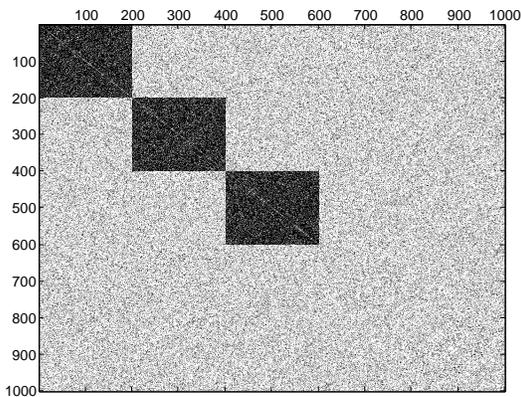


Figure 8: The summary graph generated from combining two periods of 2 phases of graphs (Fig. 7). From the figure, you can see that the summary graph incorporates both sets of clusters found in both phase 1 and phase 2.

whereas the yellow area represents a group nodes that have mixed membership. However, spectral clustering does not capture the cluster pattern perfectly because we are missing a cluster component where the nodes 201 - 300 have high probability of membership.

(a) Graphs of time points 1-3, 11-13, 21-23.



(b) Graphs of time points 4-7, 17-20.

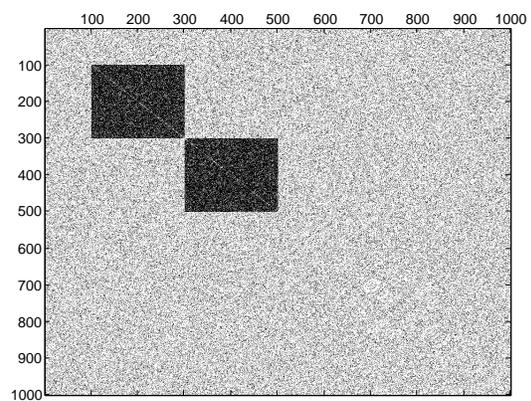


Figure 9: Shown above are the adjacency matrices of the graphs generated. Notice that nodes of particular cluster during one phase (a) and also members of other clusters in another phase (b).

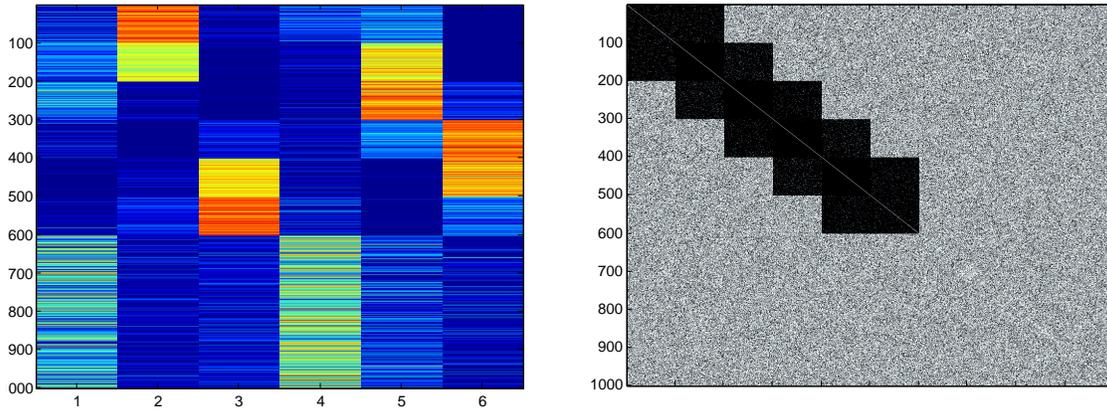
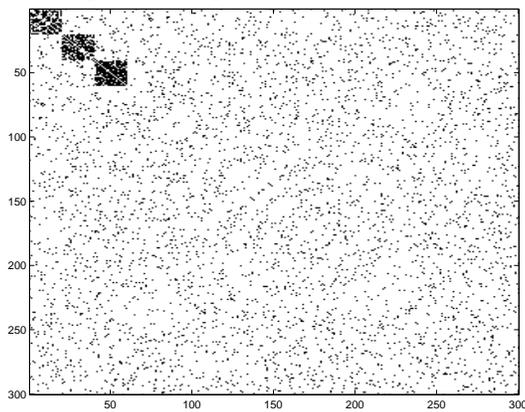


Figure 10: (a) Soft assignment of nodes to clusters generated from spectral clustering. (b) Summary graph of the time series with overlapping clusters.

5.4 Subgraph Matching Simulations with Matricized Sparse PCA method

For the matricized sparse PCA approach we have developed, we focus on our ability to discover the subsets of time during which groups of edges occur. We output the common (possibly disconnected) set of edges for these subsets. A clustering algorithm can then be used to cluster these edge sets to construct dense subgraphs. We show results on three synthetic graphs shown in Figures 11, 12 and 13. We set the number of components K equal to 10. However, the algorithm can terminate on its own before then, so K merely serves as an upper bound.

(a) Graphs of time points 1-3, 7-9.



(b) Graphs of time points 4-6, 10-12

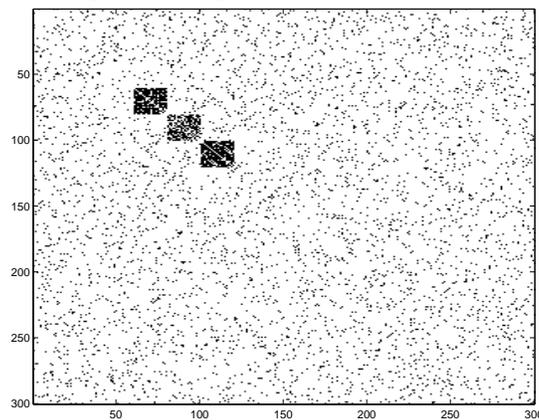
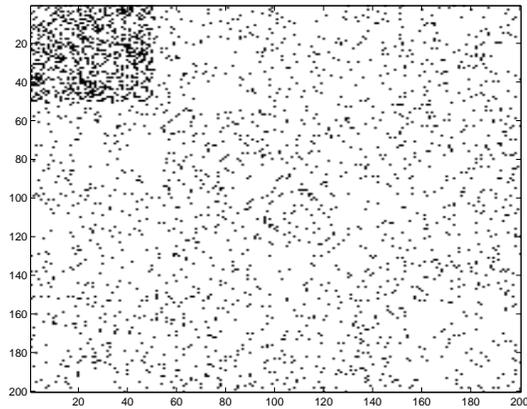


Figure 11: Synthetic Graph Series 1: Designed to test if algorithm can find small groups of highly connected vertices. There are 300 vertices and a total of 12 time points.

(a) Graphs of time points 4-6, 10-12



(b) Graphs of time points 4-6, 10-12

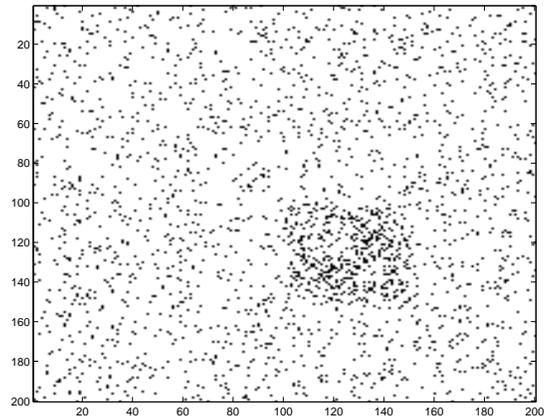
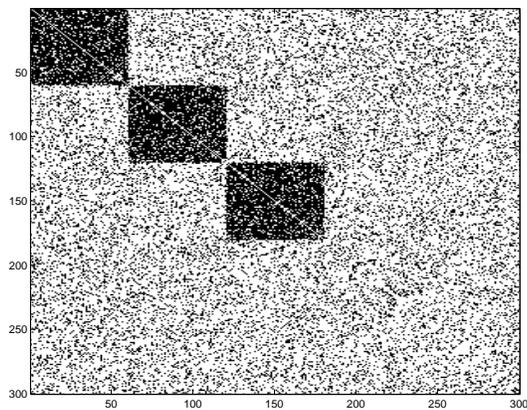


Figure 12: Synthetic Graph Series 2: Designed to test how algorithm performs in a lower signal to noise ratio scenarios. There are 200 vertices and a total of 12 time points.

(a) Graphs of time points 1-8



(b) Graphs of time points 9-16

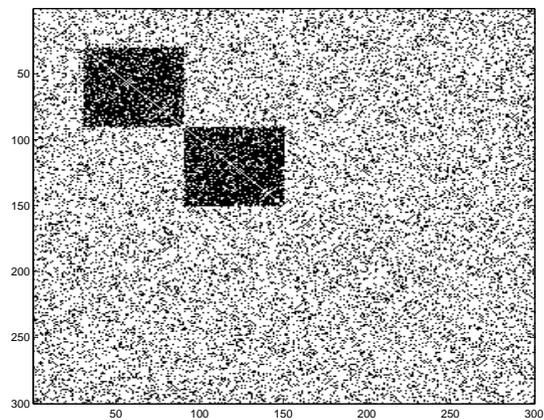


Figure 13: Synthetic Graph Series 3: Designed to test how algorithm performs in the case of overlapping clusters. There are 300 vertices and a total of 30 time points

As shown in Figure 5.4, our method obtains correct results for Synthetic Graph Series 1. The left most subfigure (a) shows the subset of time points that constitute each recovered component. Recall that these selected time points share a significant number of edges. Note that the method determines that there should only be two components and therefore components 3-10 are empty. Furthermore component 1 contains time points 1-3 and 4-6 while component 2 contains time points 4-6 and 10-12 which is correct. Subfigures (b) and (c) show the corresponding edge sets for each of these components. A clustering algorithm can then be used to cluster these edges if desired.

Similarly, even in the case of lower signal-to-noise ratio as in Synthetic Graph Series 2 (Figure 12), our method still finds the correct components as shown in Figure 15. However, some of the edge sets returned are rather sparse.

The graph series in Figure 16 provides a more challenging case. In reality the clusters are overlapping. However, our method will not be able to pick up the overlap explicitly because the data has been matricized. Instead the first component consists of the vertices that are part of clusters from times 1-16 while the second and third components focus on times 1-8 and 9-16 respectively.

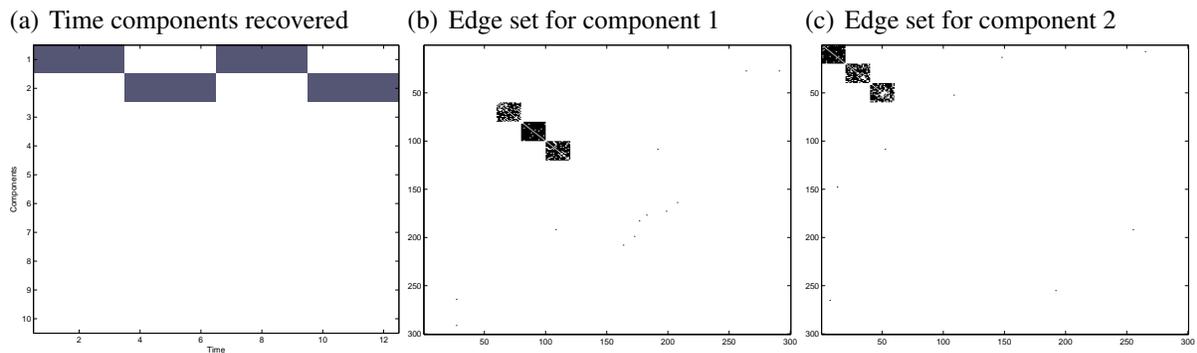


Figure 14: Performance of matricized sparse PCA method on Synthetic Graph Series 1

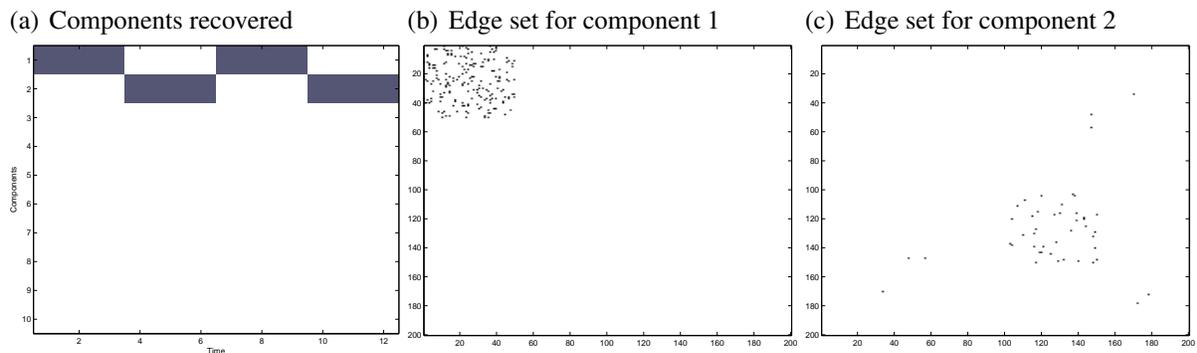


Figure 15: Performance of matricized sparse PCA method on Synthetic Graph Series 2

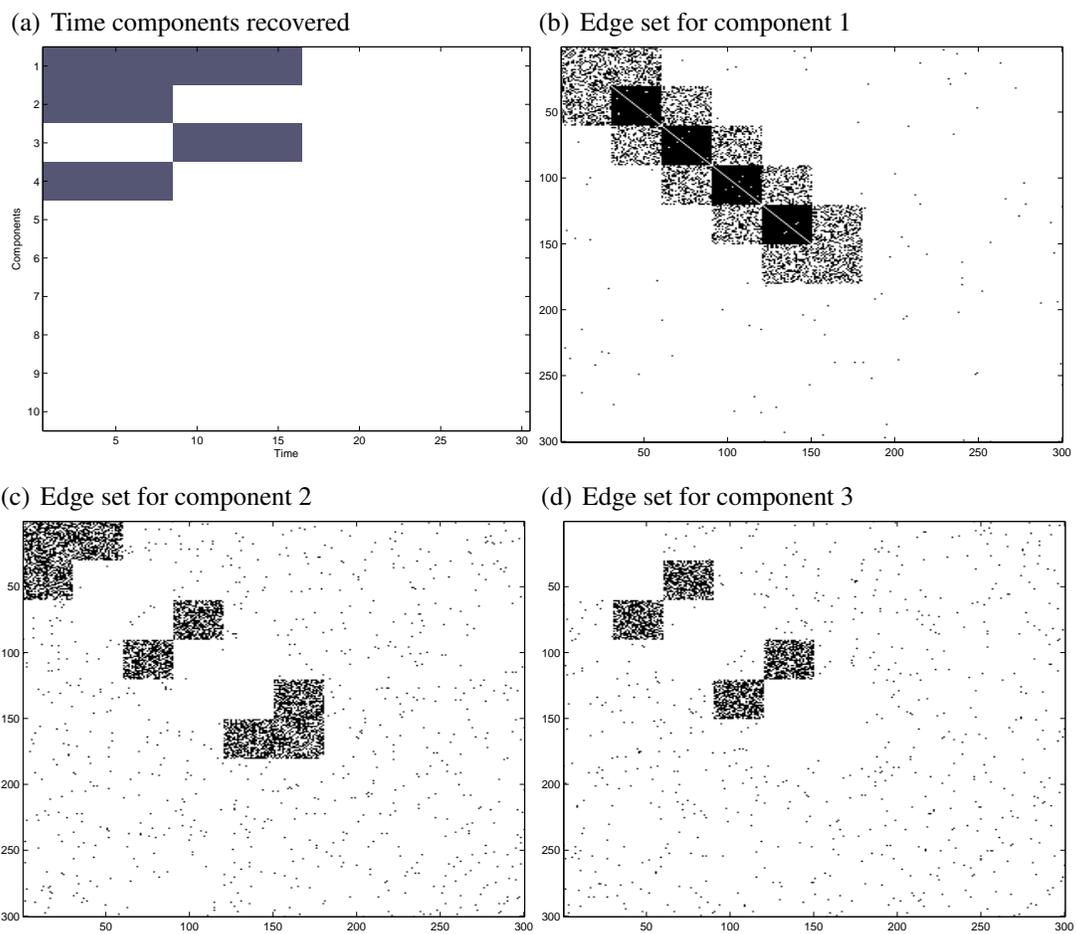


Figure 16: Performance of matricized sparse PCA method on Synthetic Graph Series 3

5.5 Subgraph Matching Simulations with PARAFAC Tensor Analysis

For completeness, we evaluated how nonnegative PARAFAC and sparse (nonnegative) PARAFAC (SPARAFAC) tensor decomposition perform on the above tasks. Both are significantly slower than the matricized sparse PCA approach. SPARAFAC in particular is not scalable. However, note that these methods do tensor decomposition which means that they simultaneously find time components and cluster vertices.

The results for (nonnegative) PARAFAC for all the datasets are shown in Figures 17, 18, 19. It can be seen that the first component returned is consistently not useful (i.e. it contains all the time points and all the vertices). However, the rest of the time components tend to be correct when PARAFAC is given either the correct number of components or less than the correct number. However, if it is given considerably more than the correct number, the resulting components are not correct. For example, in Figure 18 the algorithm performs well when 3 clusters are specified, but poorly if 7 are specified. One other problem is that all the coefficients are non-zero so there is no principled way to choose a threshold.

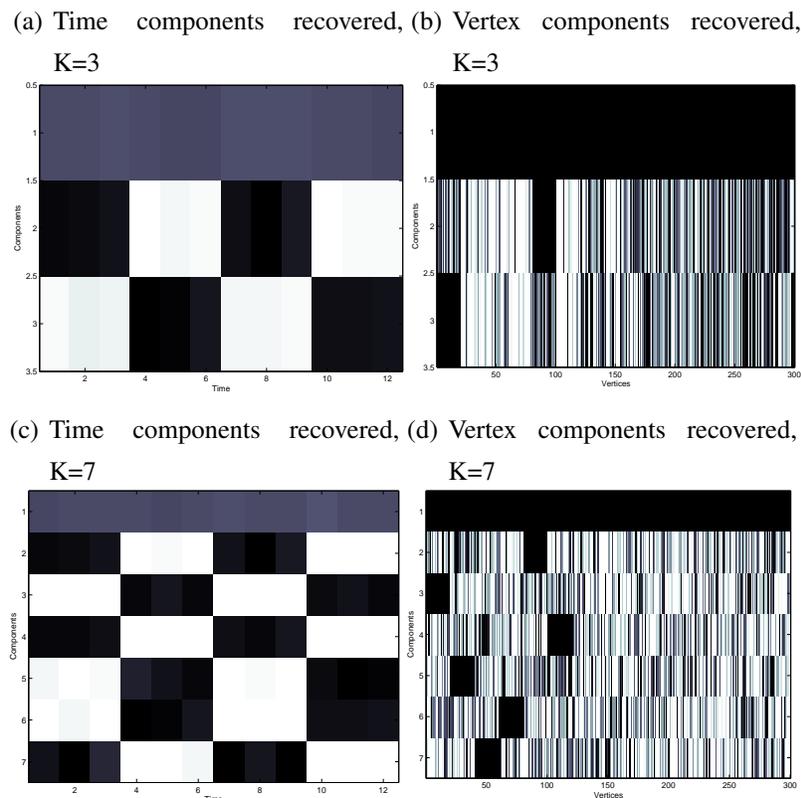
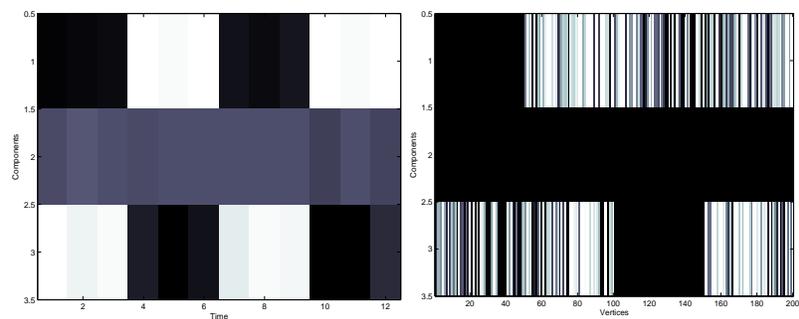


Figure 17: Performance of PARAFAC on Synthetic Graph Series 1

Because sparse PARAFAC does not scale well, we show the results of Sparse PARAFAC only on

(a) Time components recovered, (b) Vertex components recovered,
 $K=3$ $K=3$



(c) Time components recovered, (d) Vertex components recovered,
 $K=7$ $K=7$

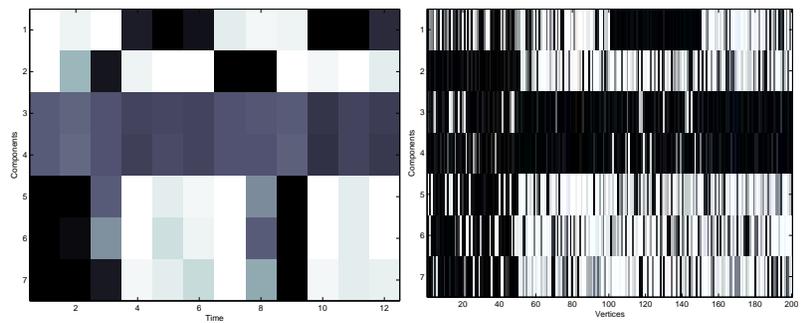
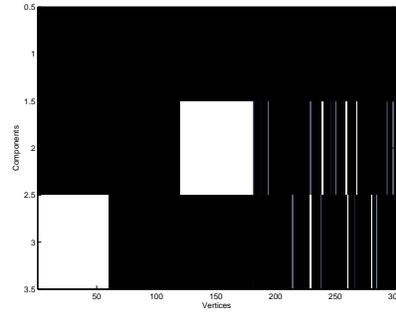
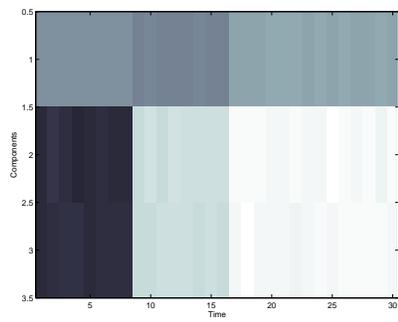


Figure 18: Performance of PARAFAC on Synthetic Graph Series 2

a) Time components recovered, $K=3$ b) Vertex components recovered, $K=3$



c) Time components recovered, $K=7$ d) Vertex components recovered, $K=7$

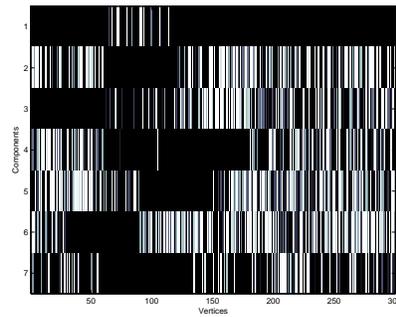
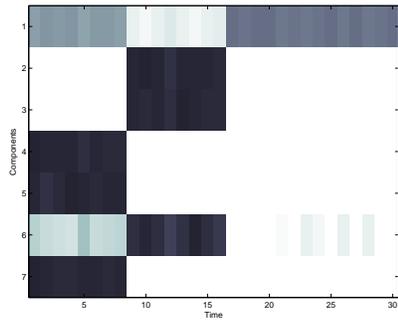


Figure 19: Performance of PARAFAC on Synthetic Graph Series 3

Synthetic Graph Series 2 in Figure 5.5, which is the smallest of the 3. In particular, the regularization parameter has to be tuned, so the program must be executed multiple times which is not desirable because each run is quite slow. However, it does not suffer from the same performance drawbacks that PARAFAC does. Specifically, as the number of components is increased, the regularization will set these redundant components to zero and still return the correct solution.

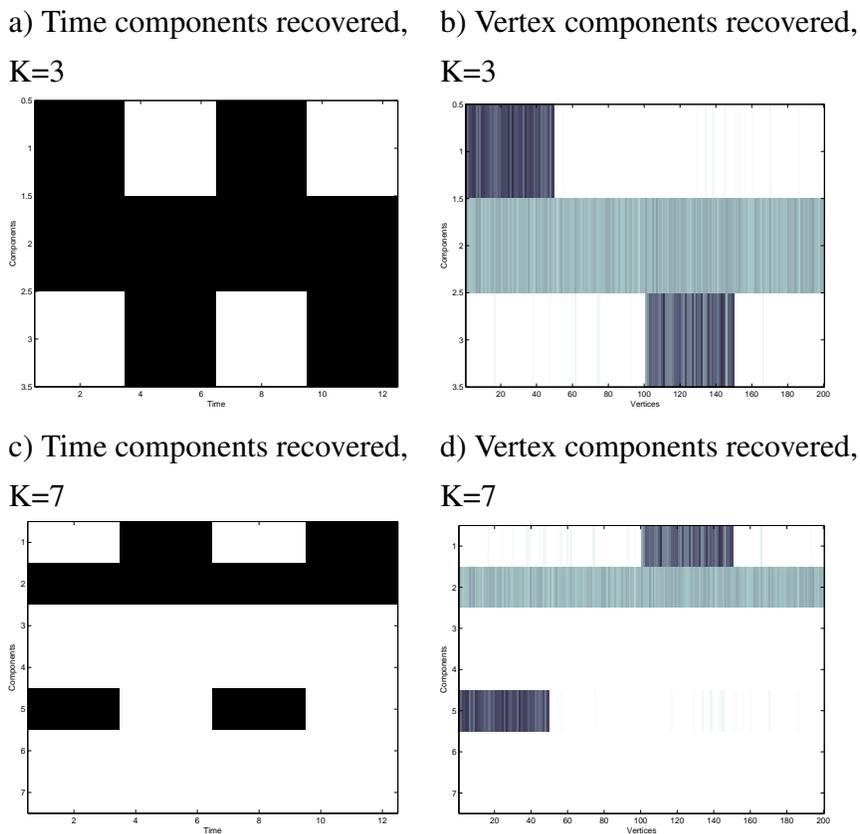


Figure 20: Performance of SPARAFAC on Synthetic Graph Series 2

5.6 Timing Comparisons

We compare the speed of PARAFAC and our matricized sparse PCA method (Sparse PARAFAC is considerably slower than PARAFAC and is omitted). Both are implemented in MATLAB. The number of time points is $T = 12$ and the number of vertices in each graph is varied from 300 to 2100. As shown in Figure 21 our matricized sparse PCA approach is orders of magnitude faster than PARAFAC since the sparse PCA greedy algorithm is much faster than the alternating minimization underlying PARAFAC. However, it should be noted that our method is only returning edge sets (and not clusters).

Therefore a clustering algorithm would have to be used as a post-processing step. PARAFAC however, simultaneously clusters the vertices and time points. Also, as expected, as the number of components in the PARAFAC decomposition increases, the computational costs increases as well.

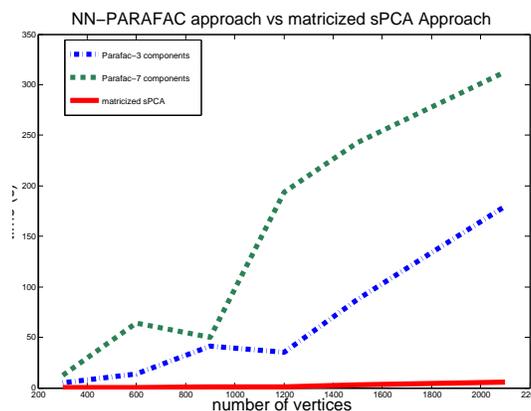


Figure 21: Comparison between our matricized SPCA approach and PARAFAC on synthetic data.

5.7 Evaluation on Yeast Cell Cycle Data

We apply the subgraph mining method to a yeast network learned by an existing method [12]. There are two cell cycles of data in this set. The table below shows the approximate intervals of the time series that correspond to the different cell cycle phases. Ideally, our matricized sparse PCA method should be able to capture these trends in the principal components that it finds.

Cell Cycle Phase	Timing in Cell Cycle 1	Timing in Cell Cycle 2
G1	Time points 1-6	Time points 13-18
S	Time points 5-10	Time points 17-21
G2M	Time points 10-14	Time points 22-24

Table 11: The estimated intervals of the time series where each cell cycle phase occurs.

The following are the top ten components that result from the decomposition of the covariance matrix $X^T X$ (Figure 22). The pink vertical line indicates where one cell cycle ends and another begins. It can be observed that the components we obtain are reasonable, as the first component represents that there is a significant amount of edge activity in the first cell cycle phase which corresponds to G1. Components 2, 6, and 8 are components that indicate activity in the middle of the cell cycle suggesting S phase interactions and components 7 and 10 occupy the end of the cell cycle (G2M phase).

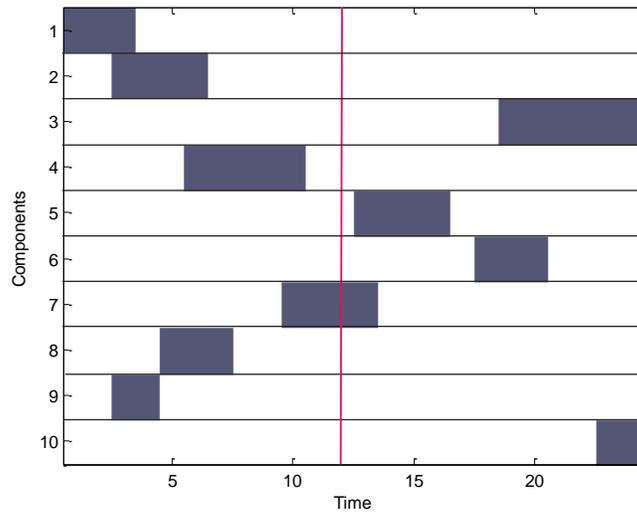


Figure 22: The components in time produced by sparse PCA. Each component (shown vertically) shows where the edges are active in time (shown horizontally).

We then made a summary graph of edges contained in each component and used the MCL package for cluster extraction [7]. In the tables below, we show examples of graphs obtained through this process and their biological relevance. Specifically, we use a GO (Gene Ontology) term mapper [4] to find common functional terms amongst the genes in our cluster. There are two examples listed below, one from G1 phase and one from S phase. The functions listed for Component 1 such as ribosome biogenesis indicate cell growth which is pertinent to G1, and the functions listed for Component 2 describe DNA synthesis which makes biological sense for S phase.

Component 1 (G1 Phase): GO Functional Annotation of Subgraphs	
Gene Ontology term	Cluster frequency
ribosome biogenesis	9 of 23 genes, 39.1%
ribonucleoprotein complex biogenesis	9 of 23 genes, 39.1%
ribosomal large subunit biogenesis	5 of 23 genes, 21.7%
cellular component biogenesis at cellular level	9 of 23 genes, 39.1%
Component 4 (S Phase): GO Functional Annotation of Subgraphs	
Gene Ontology term	Cluster frequency
lagging strand elongation	8 of 43 genes, 18.6%
DNA-dependent DNA replication	11 of 43 genes, 25.6%
DNA strand elongation involved in DNA replication	8 of 43 genes, 18.6%
DNA strand elongation	8 of 43 genes, 18.6%
DNA repair	13 of 43 genes, 30.2%
DNA metabolic process	17 of 43 genes, 39.5%
DNA replication	11 of 43 genes, 25.6%
response to DNA damage stimulus	13 of 43 genes, 30.2%
chromatin silencing at telomere	7 of 43 genes, 16.3%
nucleotide-excision repair	6 of 43 genes, 14.0%
base-excision repair	4 of 43 genes, 9.3%
cellular response to stress	14 of 43 genes, 32.6%
chromosome organization	12 of 43 genes, 27.9%

6 Conclusions

We tackle two related problems in data mining: graph similarity and subgraph matching. Both are motivated by similar applications and objectives to study and analyze graphs that occur naturally as biological networks, social networks, web graphs and many others. One of the challenges of determining the similarity between graphs is defining a measure for similarity. Here, we approach the problem with relevant ideas from belief propagation. We use the Linearized Belief Propagation (FaBP) algorithm with a similarity metric that is a normalized version of the euclidean distance. This produces extremely intuitive results and is effective in both the weighted and unweighted, connected and disconnected graph settings. One direction to consider for the future is to optimize the algorithm such that it is scalable to larger graphs of over 10 000 nodes. In addition to graph similarity, we investigate the related problem of subgraph matching; given a series of networks over time, the objective is to find subgraphs that approximately repeat in the time series in contiguous blocks. We develop a method that involves first extracting the important components in time, where edges are dominant. For this, we use sparse PCA which allows us to create a summary of edges specific to a particular subset of time points. After testing

several methods, we found that sparse PCA was desirable because it performs well and is fast. We then mine these local graphs for clusters using the Markov Clustering Algorithm. Our method produces biological relevant results and can easily handle thousands of nodes. However, more investigation is needed to determine whether it can scale is extremely large graphs, those containing tens of thousands of nodes. In addition, future work should also focus on providing a more principled and intuitive way of selecting the regularization and threshold parameters.

References

- [1] L. Akoglu and B. Dalvi. Structure, tie persistence and event detection in large phone and sms networks. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pages 10–17. ACM, 2010.
- [2] L. Akoglu and C. Faloutsos. Event detection in time series of mobile communication graphs. In *27th Army Science Conference*, volume 2, page 18, 2010.
- [3] Mohsen Bayati, David F. Gleich, Amin Saberi, and Ying Wang. Message passing algorithms for sparse network alignment. Submitted, 2011.
- [4] E.I. Boyle, S. Weng, J. Gollub, H. Jin, D. Botstein, J.M. Cherry, and G. Sherlock. Go:: Termfinderopen source software for accessing gene ontology information and finding significantly enriched gene ontology terms associated with a list of genes. *Bioinformatics*, 20(18):3710–3715, 2004.
- [5] Sung-Hyuk Cha. Comprehensive survey on distance / similarity measures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007.
- [6] A. d’Aspremont, L. El Ghaoui, M.I. Jordan, and G.R.G. Lanckriet. A direct formulation for sparse pca using semidefinite programming. *SIAM Review*, 49(3):434–448, 2007.
- [7] A.J. Enright, S. Van Dongen, and C.A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic acids research*, 30(7):1575, 2002.
- [8] R.A. Harshman. Foundations of the parafac procedure: Models and conditions for an” explanatory” multimodal factor analysis. *UCLA Working Papers in Phonetics*, 19, 1970.
- [9] H. Hu, X. Yan, Y. Huang, J. Han, and X.J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(suppl 1):212–239, 2005.
- [10] Glen Jeh and Jennifer Widom. SimRank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD ’02, pages 538–543, New York, NY, USA, 2002. ACM.
- [11] D. Koutra, T.Y. Ke, U. Kang, D. Chau, H.K. Pao, and C. Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. *Machine Learning and Knowledge Discovery in Databases*, pages 245–260, 2011.
- [12] M. Kolar L. Song and E.P. Xing. Time-varying dynamic bayesian networks. *Advances in Neural Information Processing Systems*, 22:1732–1740, 2009.
- [13] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.
- [14] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *18th International Conference on Data Engineering (ICDE 2002)*, 2002.
- [15] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):1167, 2008.
- [16] E.E. Papalexakis and N.D. Sidiropoulos. Co-clustering as multilinear decomposition with sparse latent factors. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 2064–2067, 2011.
- [17] M Pelillo. Replicator equations, maximal cliques, and graph isomorphism. *Neural Computation*, 11(8):1933–1955, 1999.
- [18] C. Schellewald and C. Schnörr. Probabilistic subgraph matching based on convex relaxation. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 171–186. Springer, 2005.
- [19] L. Song, M. Kolar, and E.P. Xing. Keller: estimating time-varying interactions between genes. *Bioinformatics*, 25(12):i128–i136, 2009.

- [20] J. Sun, C. Faloutsos, S. Papadimitriou, and P.S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 687–696. ACM, 2007.
- [21] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383. ACM, 2006.
- [22] Y. Tian, R.C. Mceachin, C. Santos, et al. Saga: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(2):232, 2007.
- [23] L.R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.
- [24] P. Vaz de Melo, L. Akoglu, C. Faloutsos, and A. Loureiro. Surprising patterns for the call duration distribution of mobile phone users. *Machine Learning and Knowledge Discovery in Databases*, pages 354–369, 2010.
- [25] Duncan J Watts. *Small Worlds*, volume 19. Princeton University Press, 1999.
- [26] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. *Understanding belief propagation and its generalizations*, pages 239–269. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
- [27] L Zager and G Verghese. Graph similarity scoring and matching. *Applied Mathematics Letters*, 21(1):86–94, 2008.
- [28] S. Zampelli, Y. Deville, and P. Dupont. Approximate constrained subgraph matching. *Principles and Practice of Constraint Programming-CP*, pages 832–836, 2005.