MINING QUASI-FREQUENT SUBNETWORKS IN GRAPH NETWORKS USING EDGE-EDGE SUMMARY GRAPH

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Priyanka Dawar

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

April 2017

Fargo, North Dakota

NORTH DAKOTA STATE UNIVERSITY

Graduate School

Title

MINING QUASI-FREQUENT SUBNETWORKS IN GRAPH NETWORKS USING EDGE-EDGE SUMMARY GRAPH

By

Priyanka Dawar

The supervisory committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Saeed Salem

Chair

Dr. Simone A. Ludwig

Dr. Lauren Hanna

Approved:

April 12, 2017

Date

Dr. Brian Slator

Department Chair

ABSTRACT

In today's computing world, graphs have become increasingly important in modeling sophisticated structures, entities and their interactions, with broad applications including Bioinformatics, Computer Vision, Web analysis etc. For an example, multiple gene expressions samples over the same set of genes are recorded to strengthen the evidence of co-expression patterns. These can be modeled by forming a set of graphs for these samples. The problem is how to dig into such multiple sources of information to make better inferences.

In this paper, I have presented an efficient method to find useful subnetworks from graph networks. The idea is to create a summary graph from these networks and then find subnetworks using this graph. I have given a detailed comparison between an already existing approach called vertex-vertex summary graph approach and the approach discussed in this paper. The results I have found are more promising than for the existing approach.

ACKNOWLEDGMENTS

I would first like to express my deepest gratitude and regards to my advisor, Dr. Saeed Salem. His guidance was invaluable in helping me finish this paper and he consistently steered me in the right the direction whenever he thought I needed it. His profound knowledge of the discipline of computer science and enthusiasm has always been a source of inspiration for me.

I would like to express my gratitude to the committee members, Dr. Simone Ludwig and Dr. Lauren Hanna, for taking their precious time to help and guide me throughout.

I also want to thank North Dakota State University and the Department of Computer Science for offering me the opportunity to study at such a great school, full of smart and diverse people who contributed to a great environment for research and studies.

Finally, I must express my very profound gratitude to my parents and to my brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this paper. This accomplishment would not have been possible without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

G	A sample graph having vertices and edges
V	The total set of vertices for a graph
E	The total set of edges for a graph
G_c	The given input graph collection
T_{vv}	A Vertex-Vertex template graph
T_{ee}	An Edge-Edge template graph
E_T	Set of edges in template graph
V_T	Set of vertices in template graph
min_d	Threshold density for DME Algorithm
$\mathcal{F}$	The set of cluster files generated from DME
θ	Quasi appearance of a subgraph
γ	Minimum Frequency Threshold
min_s	Minimum number of nodes in a cluster
M_{res}	The result matrix

CHAPTER 1. INTRODUCTION AND RELATED WORK

In this Era, data is continuing to grow at unprecedented rates. Every single transaction made online gets stored in the organizational data. For example, if you book a flight online or scan an item at Walmart or make a withdrawal or deposit in your bank account, everything gets stored in the organization's database as a part of organizational data. This data is multiplying in companies and organizations everywhere, but its growth doesn't stop there. Millions of people around the world transfer data onto the Internet in bulk every second of each day. So, the question arises: How much data is out there exactly?

Google receives over 3.5 million queries every second from the internet population across the globe today. This is very high in comparison to 10,000 queries per day in 1998 . Every minute, Facebook users share nearly 2.5 million pieces of content in form of posts, photos, videos, messages etc. Twitter users tweet nearly 400,000 times. Instagram users post nearly 220,000 new photos. YouTube users upload 72 hours of new video content. Apple users download nearly 50,000 apps. Email users send over 200 million messages. Amazon generates over $80,000 in online sales [1].

So, there is a lot of data that is generated every second but there was no means to harness this data in the past to discover useful patterns that further help in decision making and hence helps the business to grow. This data seemed to be useless a few decades ago but today because of technologies and efficient algorithms, businesses are able to convert this data into useful information or patterns that help the senior executives in making critical business decisions. For example: Which cable TV customers are going to drop their service plan before their contract expires or which customers are highly likely to renew their contract? Which credit card transactions are fraudulent? Which businessmen are likely to take a loan within the next eight months?

Finding this useful information from the plethora of data is like finding a white cat in a snowstorm. But thanks to the field of data mining which is the science of digging into these big data sets to look for relevant or useful information and answers to the above questions. The idea is that businesses these days collect enormous amount of data that may be homogeneous or heterogeneous and could be manually entered or automatically collected. Decision-makers need access to smaller, more specific pieces of data from these large sets. So, they use data mining to uncover these pieces of information that would help plot the course of their business. The data of interest may take various forms like vectors, tables, texts, images, and so on.

Generally, data mining (sometimes called knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into useful information. There are many data mining methods including:

1. Clustering and Classification
2. Pattern Mining

Classification consists of predicting a certain outcome based on a given input. There are two kind of datasets used: Training and Testing(or prediction set). For predicting the outcome, a classification algorithm processes a training set containing a set of attributes and the outcome, usually called goal or prediction attribute. The technique tries to discover relationships between the attributes that would make it possible to predict the outcome. Next the classification algorithm is given a data set not seen before, called prediction set, which contains the same set of attributes, except for the prediction attribute whose value needs to be found out. The algorithm analyses the input and produces a prediction or a classification label. The measure of prediction accuracy defines how "good" the algorithm is in predicting the outcome.

Clustering is the grouping of a set of objects based on their characteristics and aggregating them according to their similarities based on these characteristics. In regards to data mining, this technique basically partitions the data into groups called clusters using a specific algorithm, most suitable for the desired information analysis.

The other approach referred to as soft partitioning states that in a calculated degree, every object belongs to a cluster. More definitive divisions are possible for creating like objects belonging to more than one cluster which may force an object to engage in a single cluster or construct hierarchical relationships.

Pattern Mining involves construction of data mining algorithms capable of discovering useful and interesting patterns in data. Pattern mining algorithms may be applicable to various types of data which include graphs, transactional data, strings etc. Interesting patterns are the patterns of interest by a researcher. For example an interesting pattern could be defined as a pattern that appears frequently in a database. Other researchers want to discover rare patterns, patterns with a high confidence, the top patterns, etc. Finding interesting patterns in databases supports decision support, selective marketing, financial forecast, medical diagnosis and many other applications. One such type of Pattern mining is called frequent itemset mining which is discussed below.

Frequent Itemset Mining: Frequent itemsets are groups of items shared by no less than a given number of minimum transactions in the input database. Frequent Itemset Mining is fundamentally a subset of data mining dedicated to mining itemsets, patterns, and objects occurring frequently in a dataset. These mined itemsets may be further employed for discovering association rules beneficial for classification. A frequent pattern (a set of items, subsequences, substructures, etc.) is a pattern that occurs frequently in a given data set. The need for finding frequent patterns in datasets is to make various business decisions like :What products were often purchased together? What are the subsequent purchases after buying a PC? One of the crucial achievement of data mining research is the discovery of efficient algorithms for finding frequent itemsets in very large transactional databasesa[2, 3, 4, 5]. Mining the frequent sets allow us to construe association rules among them. This helps us to find the likelihood of co-occurrence of these sets.

The utility of mining frequent patterns shows up in a number of other domains too and market basket analysis (MBA) is one of them. Market basket analysis mines the sets of items that are frequently bought together at a supermarket by analyzing the items that a customer usually buys together. Mining the frequent sets allow us to construe association rules among them. This helps us to find the likelihood of co-occurrence of these sets. For example, in the case of market baskets, we can find rules like, "Customers who buy Milk and Cereal also tend to buy Bananas", which may prompt a grocery store to co-locate bananas in the cereal aisle. Another example is the weblog scenario. Frequent sets allow us to extract rules like, "Users who visit the sets of pages *main, laptops* and *rebates* also visit the pages *shopping-cart* and *checkout*, indicating, perhaps, that the special rebate offer is resulting in more laptop sales.

For any data mining application, two datasets need to be defined: set of items and set of transactions denoted by (I) and (T) respectively to mine frequent itemsets in a given database. The set I=x1, x2, ..., xm is the set of items in questions and set(T)= t1, t2, ... , tn represents the unique events in which different sets of items bought at a supermarket. Examples of sets (I) and (T) are the set of all possible items presents in a grocery store and transactions done by different customers to buy items from set(I) respectively. Let X and t denote the subsets of I and T respectively (X I and t T) such that t(X) are the transactions in which items in X appear. The subset t is also called tidset. The Eclat algorithm is used to find the itemsets that are frequently bought together by representing the given database vertically.

The basic idea behind the algorithm is that the support (variable that represents how frequently an itemset appears in the dataset) of a candidate itemset can be computed by intersecting the tidsets of suitably chosen subsets. In general, given t(X) and t(Y) for any two frequent itemsets X and Y, we have $t(XY) = t(X) \cap t(Y)$ and the support for XY is the cardinality of t(XY)), i.e., $sup(XY) = |t(XY)|$.

There is a lot of existing work on mining flat transactional data by devising data mining algorithm that mine patterns from flat data i.e., sets of items and the datasets with hierarchy, structures, etc. work well with flat representations. But it becomes challenging to represent the data collected from social networks, Chemical compounds (Cheminformatics), Protein structures, biological pathways/networks (Bioinformactics) as flat data. This real world data is usually structured and semi-structured and can be easily modeled using graphs, thus generally suited to graph representations.

Graph Mining is essentially the problem of mining useful patterns (subgraphs) occurring in the input graphs. To give one example, if we consider protein-protein interaction networks (a common application area for graph mining) these can be represented in a graph format such that the vertexes indicate genes, and the directed or undirected edges indicate physical interactions or functional associations.

Because of the ease with which structured and semi-structured data can be represented in graph formats, there has been much interest in the mining of graph data. 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.

One of the growing areas is mining of dense subgraphs in networks and many problems in social, biological and financial networks require finding cliques, or quasi-cliques (densely connected subgraphs).

Graph, in general is defined to be a set of vertexes (nodes) which are interconnected by a set of edges. As a general data structure, graphs have become increasingly important in modeling sophisticated structures and their interactions in these fields. Entities and the relationships among them are represented by directed or undirected graphs. In many real world applications, these entities are associated with different types of relationships in same or different contexts.

For example, the same set of people can form a network on Facebook and Twitter. Another example would be multiple gene expressions samples over the same set of genes are recorded to strengthen the evidence of coexpression patterns. These can be modeled by forming a set of graphs over the same set of vertices but different set of edges in the graphs.

In graph mining, finding dense components or subgraphs is very important. A dense subgraph is a subgraph that satisfies a user defined constraint [6]. These constraints can include vertices with a minimum number of edges to other vertices in the same subgraph, the vertices with minimum number of outgoing edges or other similar constraints. Finding dense subgraphs in graphs can represent important information. In a Protein- Protein Interaction network, proteins which are members of the same dense subgraph can represent protein complexes [7] which is another name for dense protein networks. Although there are ways of detecting protein complexes experimentally, there are weaknesses to each of those ways [8].

Thus it is important to use data currently available and extrapolate other proteins that might belong to these complexes and find new complexes. This can help identify disease causing genes to identify people at risk and start preventative

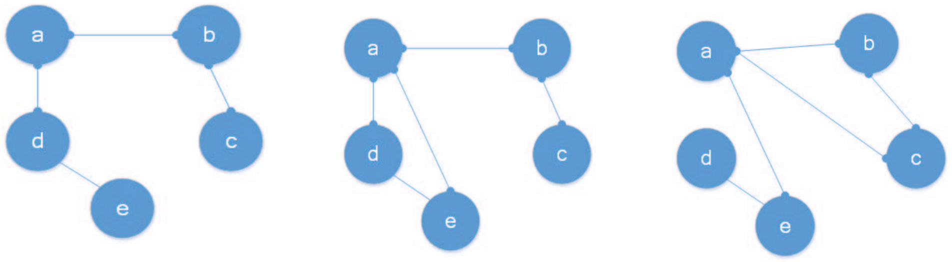


Fig. 1. A collection of graphs

treatment of possible future conditions. Dense subgraphs in a Facebook network may represent groups of interrelated people. This information can be utilized to find out people who are likely to be friends but do not have a connection between them yet. Mining dense subgraphs will help recommend they befriend each other. Another use would be to find commercial items which a majority of the people in the group own and sell them to the other people in the group.

The above approach discussed for mining Frequent Itemset can be applied to finding dense subgraphs in a graph. The nodes and their neighbors in a graph can be related to items and their common transactions. The concept of minimum support in frequent itemset mining can be related to the concept of minimum density for a subgraph to be dense in graph mining.

Testing every combination of vertices within a graph is one way to enumerate all the possible dense subgraphs within the graph. This collection of all possible combinations of items of a set S is called the power set, or P(S) of S. A set enumeration tree is an efficient way to enumerate the power set of a set. Creating a set enumeration tree is efficient and makes sure no combination is enumerated twice [12]. An example of a set enumeration tree for the set {A,B,C,D} is shown in Figure 2.

A set enumeration tree can be created by sorting the set and setting the root as null. Then the first level nodes in the tree are created. Each node in the first level contains a unique item of the set(called member set) in a sorted order.

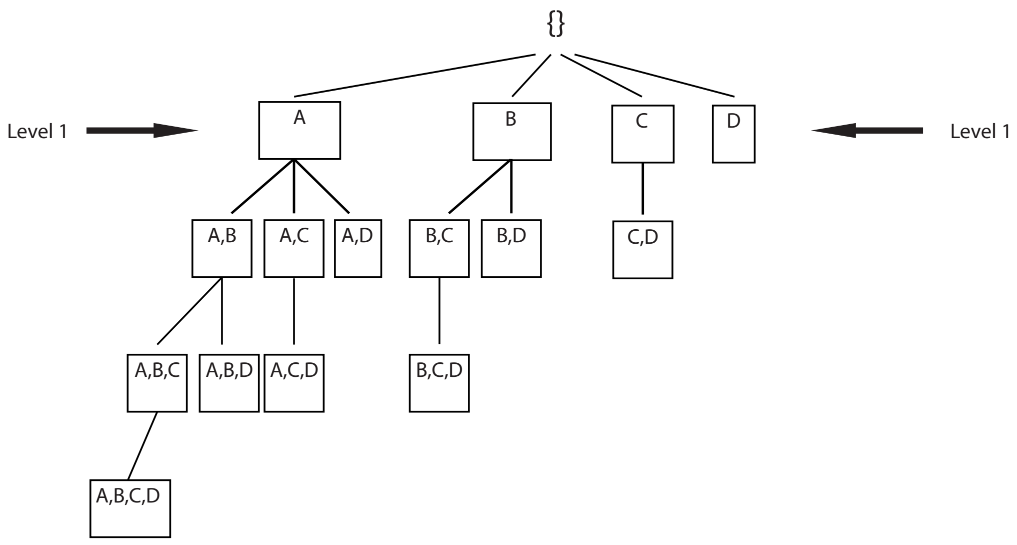
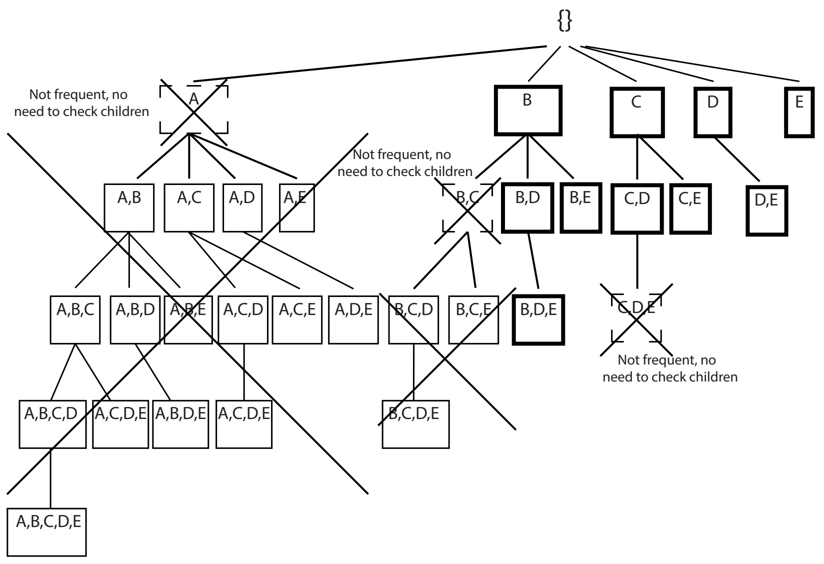


Fig. 2. Set enumeration tree for {A,B,C,D}

In Figure 2, the depth-first traversal for the first level node with A as the member set would be {Af}, {A,B}, {A,B,C}, {A,B,C,D}, {A,B,D}, {A,C}, {A,C,D}, {A,D}. Then the depth-first traversal is done for node {B} on the first level and so on.

This set enumeration tree can also be used for finding frequent itemsets in association rule mining. An example of this is shown in Figure 3. A number of transactions containing items are mined with minimum support (minsup)=2. An itemset is frequent if the number of transactions in which the itemset appears is greater than or equal to the minimum support. Using the set enumeration tree, the member set of each node is tested for frequency. In frequent itemset mining if an itemset is not frequent then no superset of that itemset will be frequent(anti-monotone property). In the itemset enumeration tree, this allows for pruning of all children of a node if the parent node is not frequent. In Figure 3, the search branches rooted at {A}, {B},{C}, and {C,D,E} are pruned this way. The frequent itemsets are shown as bolded boxes in Figure 3.

Finding frequent subgraphs across a set of graphs is another area of graph



t1	B,C,D,E
t2	A,B,E
t3	B,D,E
t4	C,D
t5	D,E

Fig. 3. Frequent itemset enumeration tree with minimum support of 2

mining. By considering the labels on the network nodes, relevant algorithms discover frequent subnetworks in networks.

The power of using graphs to model complex datasets has been recognized by various researchers in chemical domain [9, 10], computer vision [11], image and object retrieval [12, 13], and machine learning [14, 15]. Dehaspe et al. [9] applied Inductive Logic Programming (ILP) to obtain frequent patterns in the toxicology evaluation problem. Inductive Logic Programming has been actively used for predicting carcinogenesis, which is able to find all frequent patterns that satisfy a given criteria. Another approach that has been developed is using a greedy scheme [26, 16] to find some of the most prevalent subgraphs. These methods are not complete, as they may not obtain all frequent subgraphs, although they are faster than the ILP-based methods.

Inokuchi et al. [16] presented a computationally efficient algorithm called AGM, that can be used to find all frequent induced subgraphs in a graph database that satisfy a certain minimum support constraint. AGM finds all frequent induced subgraphs using an approach similar to that used by Apriori [2], which extends

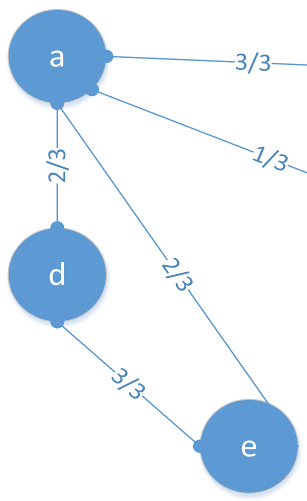


Fig. 4. A sample template graph

subgraphs by adding one vertex at each step. Experiments reported in [16] show that AGM achieves good performance for 2 synthetic dense datasets, and it required 40 minutes to 8 days to find all frequent induced subgraphs in a dataset containing 300 chemical compounds, as the minimum support threshold varied from 20% to 10%.

Lee et al. [6] proposed an approach that builds an unweighted summary graph that only has edges which occur in at least a minimum number of graphs. Clustering the aggregate graph results in false positive modules since the links between the edges in a given module can be scattered across the graphs but appear together in the aggregate graph. To overcome these false positive modules, Hu et al. [4] proposed the CODENSE algorithm, a two-step approach for mining coherent dense subgraphs. In the first phase, dense modules are extracted from the aggregate graph. The second phase uses the edge occurrence similarity and partition these dense modules into smaller modules whose edges show high edge occurrence similarity.

As discussed above, one of the main subject areas of graph based data mining is finding frequently occurring subgraph patterns from graphs, there has been a significant work done in recent past to mine graph data efficiently. Frequent Subgraph Mining (FSM) is the essence of graph mining. The objective of FSM is to extract all the frequent subgraphs in a given set of graphs, whose occurrence counts are above a specified threshold. Joint mining of multiple data networks can often discover interesting, novel, and reliable patterns which cannot be obtained solely from any single network. For instance in bioinformatics, gene clusters that show coherent expression patterns can be found, and protein interaction sets can be produced by joint mining of protein interaction data and gene expression. Such clusters may be potential pathways in finding related diseases.

The rest of the paper is structured as follows. Section 2 gives the problem definition and algorithm. The results of the algorithm on real world data is shown in Section 3. Lastly, Section 4 presents the conclusion and how this work might be extended in the future.

CHAPTER 2. PROBLEM DESCRIPTION

In this section, before describing the problem, I define some terms that are used throughout the paper and are essential for understanding the problem statement. The graphs considered for this paper are *simple* graphs. A simple graph is a graph which only has undirected edges. In addition, a simple graph has no self-directed edges or multi-edges.

2.1. Graph Terminology

Definition 1 (Graph). *A graph $G = (V, E)$, contains a set of vertices $V = \{v_1, v_2, \cdots, v_m\}$, and a set of edges $E = \{e_1, e_2, \cdots, e_p\}$ connecting the vertices where each edge itself is a pair of vertices. Throughout this paper,we assume that the graph is undirected and labeled i.e each vertex has a label associated with it.*

Figure 1 shows a collection of three graphs. The first graph in the collection has vertices $V = \{a, b, c, d, e\}$, and a set of edges $E = \{ab, ac, ad, ae, bc, de\}$. All the graphs in this set are undirected and unweighted.

Definition 2 (Subgraph). *A graph $G_s = (V_s, E_s)$ is called a subgraph of a graph $G = (V, E)$ if the set of vertices V_s in G_s is a subset of V and the set of edges E_s is a subset of of E i.e. $V_s \subseteq V$ and $E_s \subseteq E$.*

Figure 5 shows an example of a subgraph of the last graph(say G) in the graph collection shown in Figure 1. The vertices and edges that comprise this subgraph are the subsets of vertices and edges in G.

Definition 3 (Frequent Subgraphs). *A frequent subgraph is a graph G_f that appears frequently across a collection of graphs. A graph G is frequent across a collection G_c if the number of graphs in which it appears is greater than or equal to minimum frequency of appearance (γ).*

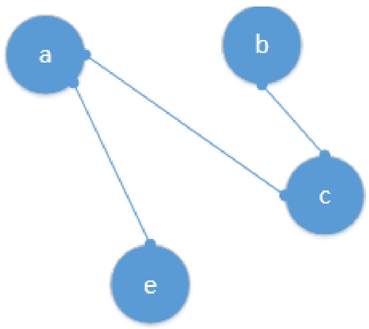


Fig. 5. An example of a subgraph

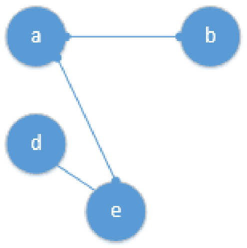


Fig. 6. An example of frequent subgraph

Consider Figure 1 as the input graph collection and γ=2 (i.e. for a subgraph to be frequent it should appear atleast in two graphs). Figure 6 shows an example of a such a subgraph.

Definition 4 (Quasi Appearance)**.** *If a sub-network does not appear fully in a graph, it is said to have quasi appearance in the graph. For example if 90% of a graph Gáppears in a bigger graph G, then the quasi appearance of Gín G is 90%.*

Figure 7 is an example of quasi appearance of a sub graph in a graph. The graph G_q with vertices $V = \{a, b, c, d\}$ and $E = \{ae, eb, bc\}$ has quasi appearance in the main graph since not all the edges in this subgraph appear in the graph. For example edge *be* doesn't appear in the parent graph. The percentage of quasi appearance is 66.7%.

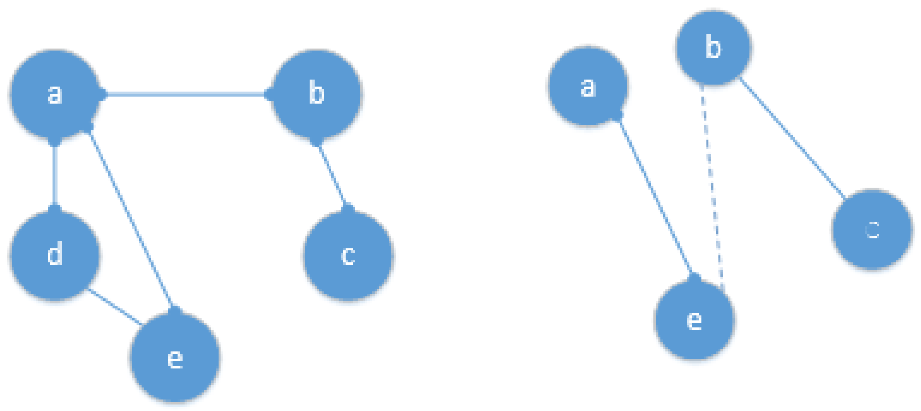


Fig. 7. An example of Quasi appearance

Definition 5 (Template Graph)**.** *A Template Graph is a summary network created from a collection of graphs and has weighted edges. The weight on each edge is the number of graphs in which the edge appears divided by the total number of graphs in the collection. We will be demonstrating two kinds of template graphs which basically differ in what actually represents the nodes and the edges between them.*

Figure 4 shows an example of a template graph created from graph collection in Figure 1. Each graph in the collection has vertices $V = \{a, b, c, d, e\}$. So the resulting template graph has the same vertices and edges $E_t = \{ab, ac, ad, ae, bc, de\}$, which is the union of all the edges in the collection and the weights $W_t = \{3/3, 1/3, 2/3, 2/3, 3/3, 1/3\}$. The weight for the edge ab in the template graph is 2/3 or 0.66 because it appears in two of the three graphs in the collection.

Definition 6 (Vertex-Vertex Template graph)**.** *A template graph in which a node represents an actual vertex in the collection of graphs and an edge between two nodes represents the edge between the vertices representing these nodes in the database of graphs is called a Vertex-Vertex Template graph. So, a node in this network (template graph) is an actual vertex that appears in the collection of graphs.*

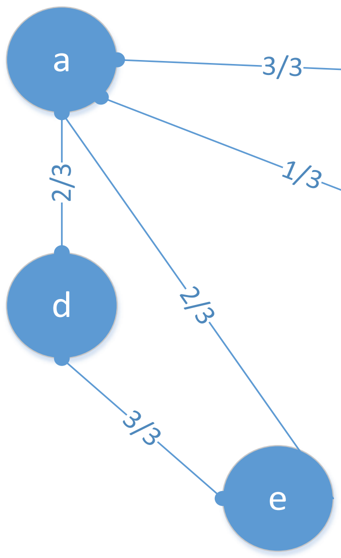


Fig. 8. A vertex-vertex template graph

Figure 8 shows an example of a vertex-vertex template graph constructed from the graph collection in Figure 1. The nodes in this template graph are the vertices in the input graph and edges are weighted.

Definition 7 (Edge-Edge Template graph)**.** *A template graph that is constructed from the input graphs in such a way that nodes in the graph are basically edges that appears in the input graph collection is called a Edge-Edge Template graph. The edges are constructed in this graph based on the common vertex that appears between two edges in any given graph in the collection.*

Figure 9 is an example of an edge-edge template graph constructed from Figure 1. Each node in this graph, $V_t = \{ab, bc, ac, ad, ae, de\}$ is an edge in the input graph set. The edges $E_t = \{\{ab, bc\}, \{bc, ac\}, \{ab, ac\}, \{ab, ad\}, \{ab, ae\}, \{ac, ae\}, \{ad, ae\}, \{ad, de\}, \{de, ae\}\}$ are added to the graphs based on the appearance of a common vertex in two nodes. For example an edge appears between the nodes ab and bc because these two nodes which are edges in the input graphs have a common vertex b.

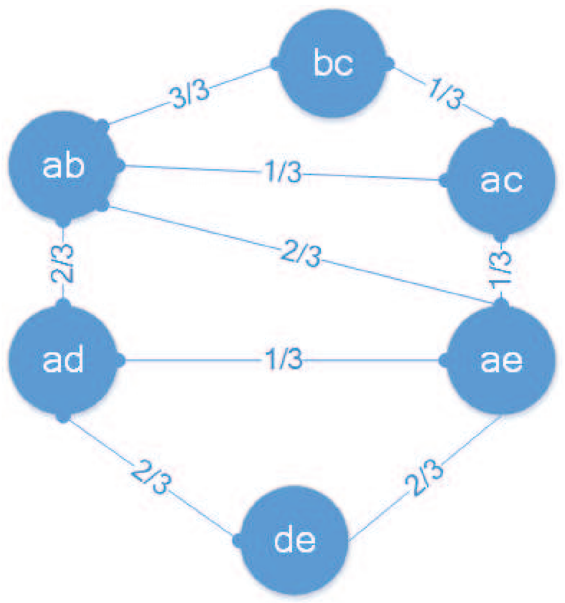


Fig. 9. An edge-edge template graph

2.2. Problem Definition

Consider there are m different network topologies on the same set of vertices V. For example we are given m different biological networks with the same set of molecules. Let us denote the networks in this collection G by graphs $G_i = (V, E_i)$, for each value of i = 1, 2, ..., m. The given problem is to find frequent dense sub-networks over the entire set of network topologies.

One of the approaches is creating a summary graph from the graphs in the collection and then exploiting this graph to find the modules that appear frequently in the graphs. This summary graph consists of the nodes as the total possible vertices that appear in the given collection of graphs with the respective edges. The weight of an edge between two vertices is the ratio of the number of graphs in which that edge appears to the number of graphs in the network. This summary graph is also called the template network. Since the nodes in this summary graph represent the vertices in the network, we call this graph as Vertex-Vertex template graph.

So, a Vertex-Vertex Template graph is a graph in which a node represents an actual vertex in the collection of graphs and an edge between two nodes represents the edge between two vertices representing these nodes in the collection of graphs. So, a node in this network (template graph) is an actual vertex that appears in the collection of graphs.

The approach presented in this paper is called the Edge-Edge Template graph approach which is about creating an Edge-Edge summary graph and then mining quasi frequent subgraphs from this graph. This means finding out the quasi appearance of these subgraphs in the collection. A vertex in this summary graph represents an edge in any of the graphs in the collection. So, an Edge-Edge Template graph is a graph in which a node represents an actual edge in the collection of graphs and an edge between two nodes represents the connection between two edges present in graph collection.

2.3. Algorithms

In this project, I developed algorithms for discovering quasi frequent subnetworks in a given collection of graph. This is done by creating an edge-edge template summary graph from these graphs and mining this graph to find dense subgraphs.

The first algorithm builds a vertex-vertex template graph with a set of graph networks and minimum support as the inputs.The set of graphs that have the same set of vertices. Each graph in this set is represented as an adjacency list of the edges with equal weight on each edge.

The second algorithm builds an edge-edge template graph with a set of graph networks, minimum support as the inputs. Each graph in the collection has the same set of vertices, though the connection amongst these vertices may vary across the graphs. Each graph in this set is represented as an adjacency list of the edges with equal weight on each edge. The nodes in the resulting graph are the edges from the individual graphs.

The third algorithm discovers the quasi frequent subgraphs obtained from the vertex-vertex template graph obtained from Algorithm 1. The algorithm used to find the dense modules or clusters from the template graphs is DME algorithm which takes the template graph and minimum density as inputs and outputs the dense subgraphs based on this minimum density. The details of how this algorithm works are given later in the paper. These dense clusters are then analyzed to see if they have frequent quasi appearance across the graphs in the collection using this algorithm.

The last algorithm discovers the quasi frequent subgraphs obtained from the edge-edge template graph obtained from Algorithm 2. These dense clusters are then analyzed to see if they have frequent quasi appearance across the graphs in the collection using this algorithm. We then compare the two approaches and see which is more promising in terms of mining denser subgraphs.

2.4. Pseudo-code

Algorithm 1 shows the pseudo-code for generating a vertex-vertex template graph. The algorithm takes as input a collection of graphs $\mathcal{G}$, and outputs the template graph T_{vv} with $\mathcal{E}_{\mathcal{T}}$ and $\mathcal{V}_{\mathcal{T}}$ as edge set and vertex set respectively. The template graph is finally written to a matrix M_{vv} that can be further used to mine subnetworks in the graph collection. In the algorithm, the template graph is first declared as an empty graph and then built as a set of edges with each edge represented by a pair of vertices. For a given graph in the collection, each and every edge is scanned to see if it is present in the template graph. If it is present, its weight is increased by 1. If the edge is not present in the graph, it is added to the graph and the weight of the edge in the template graph is set to 1. This is repeated for the all the graphs in the collection. In the end, the template graph with edges having calculated weights

Algorithm 1: Creating a Vertex-Vertex Template Graph

```
Input:
  $G_c$: The given set of graphs
Output:
  $\mathcal{T}_{\sqsubseteq\sqsubseteq}$:Vertex-Vertex Template Graph
  $\mathcal{E}_{\mathcal{T}}$: Set of Edges in Vertex-Vertex Template Graph
  $\mathcal{V}_{\mathcal{T}}$: Set of Vertices in Vertex-Vertex Template Graph
  $M_{vv}$: Vertex-Vertex Template Matrix
1.   $\mathcal{E}_{\mathcal{T}} = \emptyset$ ▷Set of edges in the vertex-vertex template graph
2.   ▷Find the set of edges and their weights in the Template graph
3.   for each $g_i \in G_c$:
4.   $E$=$getEdges(g_i)$ ▷Set of Vertices in $\mathcal{G}$
5.   for each $e_j \in E$:
6.     if $e_j \in E_T$ Then
7.       $W(e_j) = W(e_j) + 1$
8.     else
9.      $E_T = E_T \cup e_j$
10.      $W(e_j)$ =1
11.    end if
12.  end for
```

is returned.

Algorithm 2 shows the pseudo-code for generating an edge-edge template graph. The algorithm takes as input the same collection of graphs $\mathcal{G}$, and outputs the template graph T_{ee}, with $\mathcal{E}_{\mathcal{T}}$ and $\mathcal{V}_{\mathcal{T}}$ as edge set and vertex set respectively. As explained, each node in the edge-edge template graph represents an edge in the collection. So each edge in the template graph is formed by using a pair of edges from the graph collection. The edge-edge template graph is finally written in the form of a matrix M_{ee} that can be further used to mining subnetworks in the graph collection.

In the algorithm, each graph is scanned in the collection for each pair of edges $e_ij = e_{ik} \cup e_{kj}$. The edges e_{ik} and e_{kj} in the given input graph represent the nodes in the template graphs and the resulting edge is e_{ij} which is represented as an edge in the template graph. If this edge is found in the template graph, its weight is increased by 1 in the graph. Otherwise this edge is added to the graph and given an initial

Algorithm 2: Creating an Edge-Edge Template Graph

```
Input:
  $G_c$: The given set of graphs
Output:
  $T_{ee}$: Edge-Edge Template Graph
  $\mathcal{E}_{\mathcal{T}}$: Set of Edges in Edge-Edge Template Graph
  $\mathcal{V}_{\mathcal{T}}$: Set of Vertices in Edge-Edge Template Graph
  $M_{ee}$: Edge-Edge Template Matrix
1.    $\mathcal{E}_{\mathcal{T}} = \emptyset$ ▷Set of edges in the vertex-vertex template graph
2.    ▷ Find the set of edges and their weights in the Template graph
3.    for each $g_i \in G_c$:
4.    $E$=$getEdges(g_i)$ ▷Set of Vertices in $\mathcal{G}$
5.    for each pair $(e_ik, e_kj) \in E$ where i<j:
6.      $e_ij = e_{ik} \cup e_{kj} \in E_T Then$
7.      if $e_{ij} \in E_T$ Then
8.        $W(e_{ij}) = W(e_{ij}) + 1$
9.      else
10.       $E_T = E_T \cup e_{ij}$
11.       $W(e_{ij})$ =1
12.     end if
13.   end for
```

weight 1. This is repeated for all the graphs in the input graph collection.

Algorithm 3 shows the pseudo-code for mining quasi frequent subnetworks using the vertex-vertex template graph approach. The algorithm takes as input a collection of graphs $\mathcal{G}$, the set of input cluster files $\mathcal{F}$ obtained from DME algorithm using template graph, T_{vv} and the array of DME densities as inputs, the vertex-vertex template graph obtained from Algorithm 1, the array of DME densities used to generate the DME clusters, minimum frequency of appearance γ of a cluster in the collection of graphs to to declare a subgraph as frequent, the array containing quasi appearance (θ) values. The DME algorithm creates one file per density value.

The algorithm outputs the result matrix that has the percentage of subnetworks that is frequent across the graph collection for a given Quasi appearance (θ) and a Minimum frequency of appearance (γ). The results are found out for quasi-appearances (θ)= (50, 60, 70, 80, 90, 100) and density threshold for DME min_d= (50, 55, 60, 65, 70, 75, 80, 85). The algorithm is run for γ (frequency of appearance)= 8 and 9.

In the algorithm, each cluster file is scanned to loop through the clusters in the file. For each cluster in the file, the intersection graphs are found by intersecting the current cluster being scanned with each of the input graphs, $g_int = g_i \cup c_g$. For each intersection graph, it is checked if the length of the intersection graph is greater than or equal to current θ (quasi appearance)value. If this quasi clearance condition is met, then the minimum frequency of appearance condition is checked. If the appearance of this intersection is in graphs equal to greater than γ value, the DME cluster to which this intersection graph belongs is considered as a quasi-frequent subgraph.

This is done for all the θ values used in the experiment for the current cluster. The same conditions are checked for all other clusters in the current input cluster file containing clusters obtained from DME algorithm for a given DME density.

Algorithm 4 shows the pseudo-code for mining quasi frequent subnetworks using the edge-edge template graph approach. The algorithm takes as input the collection of graphs $\mathcal{G}$, the set of input cluster files $\mathcal{F}$ obtained from DME algorithm, the edge-edge template graph T_{ee} obtained from Algorithm 2, the array of DME densities used to generate the DME clusters, minimum frequency of appearance γ of a cluster in the collection of graphs to to declare a subgraph as frequent, quasi appearance (θ)values.

The algorithm outputs the result matrix M_{res} that has the percentage of subnetworks that is frequent across the graph collection for a given Quasi appearance (θ) and a Minimum frequency of appearance(γ). The results are found out for the same set of quasi-appearances(θ) and density threshold for DME as in edge-edge clustering. The algorithms are run for $\gamma = 8$ and 9.

Algorithms 3 and 4 are similar except in case of Algorithm 3 you have to basically create the subgraph for each DME cluster using the template graph (Steps 11-13). In case of edge-edge template graph approach, each node in a DME cluster is an edge since the DME clusters are found from edge-edge template graph. So the nodes in the cluster represent the actual subgraph.

Algorithm 3: Mining Vertex-Vertex Template Graphs

```
Input:
  $G_c$: The given collection of input graphs
  $\mathcal{F}$: The set of input cluster files generated from DME algorithm
  $T_{vv}$: Vertex- Vertex template graph
  $\mathcal{E}_{\mathcal{T}}$: Set of Edges in Vertex- Vertex Template Graph
  $min_d$: Density Threshold for DME
  $\gamma$: Minimum frequency of appearance of cluster in Input Graphs
  $\theta$: Quasi appearance of a cluster in an Input Graph
  $min_s$: Minimum number of nodes in a cluster
Output:
  $M_{res}$: The result matrix with for each DME density and each quasi appearance
1.    for each file $f_i \in F$:
2.      $C$=$getClusters(f_i)$ ▷Set of Vertices in $\mathcal{G}$
3.      $\theta$=(50,60,70,80,90)
4.      for each cluster $c_i \in C$:
5.        if($length(c_i) < min_s$ then
6.          go to step 4
7.        end if
8.        $C_{total} = C_{total} + 1$
9.        $counts = (0, 0, 0, 0, 0, 0)$
10.       for each node $j \in c_i$:
11.           $x = getIndexes(T_{vv}[, 1], j)$
12.           $y = getIndexes(T_{vv}[, 2], j)$
13.           $c_g = x \cap y$ ▷ subgraph for the current cluster
14.       end for
15.       ▷ find if the cluster appears in the input graphs
16.       for each graph $g_i \in G_c$
17.           $g_int = g_i \cup c_g$
18.           $q_app = length(g_int)/length(c_i)$
19.           for each threshold $\theta_i \in \theta$:
20.             if($q_app >= \theta_i$) then
21.               for file $c_i$ $count(\theta_i) = count(\theta_i) + 1$
22.               end for
23.             end if
24.           end for
25.       end for
26.       ▷find if the current cluster is frequent across the input graphs or not
27.       for each count in counts
28.           if($count >= \gamma$) then
29.             $C_{freq}[\theta_i] = C_{freq}[\theta_i] + 1$
30.           end if
31.       end for
32.     end for
33.       ▷calculate the number of frequent clusters contained this file for each $\theta$
34.       for each index x in $C_{freq}$
35.         $M_{res}[i, x] = C_{freq}/C_{total}$ ▷ i is the file number being processed
36.     end for
37.   end for
```

Algorithm 4: Mining Edge-Edge Template Graphs

Input:

G_c: The given set of input graphs

T_{ee}: Edge-Edge template graph

$\mathcal{E}_{\mathcal{T}}$: Set of Edges in Edge-Edge Template Graph

F_{lkp}: The lookup file having edge to vertex mapping

$\mathcal{F}$: The set of input cluster files generated from DME algorithm

min_d: Density Threshold for DME

γ: Minimum frequency of appearance of cluster in Input Graphs

θ: Quasi appearance of a cluster in an Input Graph

min_s: Minimum number of nodes in a cluster

Output:

M_{res}: The result matrix with for each DME density and each quasi appearance

```
1.  for each file $f_i \in F$:
2.   $C$=$getClusters(f_i)$
3.   $\theta$=(50,60,70,80,90)
4.   for each cluster $c_i \in C$:
5.    if ($length(c_i) < min_s$) then
6.     go to step 4
7.    end if
8.    $C_{total} = C_{total} + 1$
9.    $counts = (0,0,0,0,0,0)$
10.     ▷ find if the cluster appears in the input graphs
11.     for each graph $g_j \in G_c$:
12.       $g_{int} = c_i \cup g_j$
13.       $q_{app} = length(g_{int})/length(c_i)$
15.      for each threshold $\theta_i \in \theta$:
16.        if($q_app >= \theta_i$) then
17.         $counts(\theta_i) = counts(\theta_i + 1)$
18.        end if
19.      end for
20.     end for
21.     ▷find if the current cluster is frequent across the input graphs or not
22.     for each count in counts
23.      if ($count >= \gamma$) then
24.        $C_{freq}[\theta_i] = C_{freq}[\theta_i] + 1$
25.      end if
26.     end for
27.    end for
28.  ▷calculate the number of frequent clusters contained this file for each $\theta$
29.     for each index x in $C_{freq}$
30.      $M_{res}[i,x] = C_{freq}/C_{total}$ ▷ i is the file number being processed
31.    end for
32.   end for
```

CHAPTER 3. EXPERIMENTS

To discover the effectiveness of Edge-Edge Template graph approach in finding dense clusters, I tested its effectiveness on a real-world dataset. The dataset I used is constructed from Genetic Network Analysis Tool (GNAT) and consists of genetic data for 35 human tissues with one graph for each human tissue. The network consists for 10,000 genes and 5 million links among those genes. So, the collection consists of 35 graphs with 10,000 vertices and 5 million edges in total.

The first step is to create the vertex-vertex and edge-edge template graphs from this collection of graphs. I first cleansed the 35 graphs based on the minimum threshold of appearance of an edge across the graphs in the network. The idea behind this is to remove the edges which appear in significantly low number of graphs. The minimum threshold for my experiments is 10 for both the approaches.

Figure 10 shows the template graph built using the vertex-vertex template graph approach constructed using Algorithm 1. The graph consists of 9886 vertices and 55458 edges. Figure 11 shows the template graph built using the edge-edge template graph approach constructed using Algorithm 2. The graph consists of 55443 vertices and 689286 edges. The minimum initial frequency used for the construction of these graphs is 10.

After constructing the template graphs for both approaches, the next step is to find out the clusters in these graphs. DME (Dense Module Enumeration) algorithm has been used here for discovering dense modules in both edge-edge template graph and vertex-vertex template graph. Given an undirected weighted graph G with node set V, let W represent the set of weights between all pairs of nodes and let U be a subset of nodes or a subgraph.

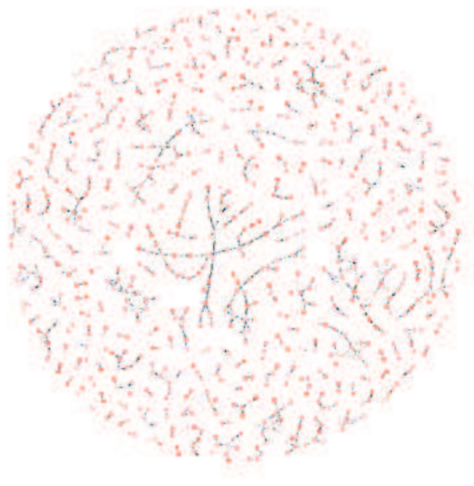


Fig. 10. A Vertex-Vertex Template Graph on GNAT data

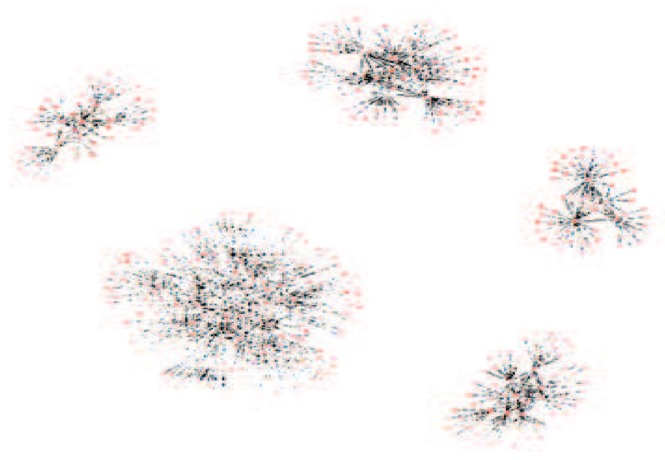


Fig. 11. An Edge-Edge Template Graph on GNAT data

Then the density of this set of nodes U with respect to W is defined as the average pairwise interaction weight between all the nodes that are members of U. Dense Module Enumeration detects all node subsets that satisfy a user-defined minimum density threshold.

In our case, the undirected weighted subgraphs are T_{vv} and T_{ee} and sets of edges and nodes associated with each of these graphs are represented by E_t and V_t respectively. Dense Module Enumeration algorithm works on both weighted and non-weighted undirected graphs. The graph collection used for the experiments in this paper consists of unweighted graphs. For unweighted graphs, it calculates the weights for its edges on its own. Dense Module Enumeration algorithm works on a set of input parameters. Following is basic enumeration method along with the explanation of the input parameters:

./dme m ⟨input_file⟩ ⟨output_file⟩ ⟨density_threshold⟩ [options]

Input file: This is the file consists of the edges along with the weight on each edge in the graph. There are three columns in total with first column and second column consisting of the node number and the third column consists for the corresponding interaction weight between the nodes. The input files for our experiment are the template_vv and template_ee files consisting of vertex-vertex template graph and edge-edge template graph respectively.

Output File: The output file consists of the cluster modules obtained by running DME algorithm on the input files. There are 4 columns: ranking score, density in the module (between 0 to 1), module size and module nodes. We will be working on the 4th column of the output file which consists of the nodes constituting the clusters.

Density Threshold: This input parameter specifies the minimum density threshold for the modules. This value should be between 0 and 1. The density thresholds used in the experiments are 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85. A threshold density of 0.50 means that a node in the set U must interact with 50% of other nodes in U. The density threshold is represented by min_d in this paper.

Options: The only option I am using as the input parameter is the min_{size} option which specifies what should be the minimum size of a module in the output file. Those with size less than minimal size are not included in the output.

Because I have run the DME algorithm for 8 different threshold densities(min_d), there are 8 output files after running the algorithm on each of the template graphs. The next step is to find out if each of the cluster modules in each of the files is frequent in the input graphs in the network or not. The two constraints used to analyze this are:

Minimum Frequency of Appearance (γ): The number of graphs the given cluster module should appear in. I have included the results for γ=8 and γ=9.

Quasi Appearance (θ) : The cluster is not always required to be present fully in a graph. So we are finding out the quasi appearance of the clusters in the collection of graphs. This constraint specifies the percentage or the part of the cluster that should be present in a graph. For example, if θ=50%, then at least a half of the cluster should appear in a graph to be considered as a frequent candidate.

Following tables show the results of my experiments run on the DME cluster files for both vertex-vertex template graph and edge-edge template graph.

Table 1. Result Matrix for Vertex-Vertex Template Graph approach

min_d	$Clusters$	$AvgSize$	$\theta = 100$	$\theta = 90$	$\theta = 80$	$\theta = 70$	$\theta = 60$	$\theta = 50$
50	2936	4.7	24%	24%	25%	26%	28%	30%
55	970	4.5	26%	26%	27%	28%	30%	32%
60	373	4.4	23%	23%	24%	26%	28%	31%
65	152	4.2	28%	28%	28%	31%	32%	36%
70	41	4.3	39%	39%	39%	44%	44%	44%
75	14	4	36%	36%	36%	36%	36%	36%
80	4	4	25%	25%	25%	25%	25%	25%
85	0	0	0%	0%	0%	0%	0%	0%

Table 1 shows the percentage of dense subnetworks (clusters) discovered from Vertex- Vertex template graph that actually appear in the initial set of graphs. The DME densities are 50, 55, 60, 65, 70, 75, 80, 85 (in percentages). For each density value, the table shows the total number of clusters discovered using the DME approach. For density 0.5 (50%) the number of clusters is maximum and for density 0.85 (85%) it's minimum which is because there would be a fewer number of subgraphs that are denser as compared to the ones that are less dense. The average cluster size for less dense subgraphs will be more as compared to the ones with a higher density value. The reason again is because it's more likely that a fewer nodes will be involved in stronger interaction. Next given are the percentages of subgraphs discovered from the vertex-vertex template graph that are quasi frequent in the collection of graphs for a different densities of clustering.

For example if the value of quasi appearance (θ) is 100% which means a subgraph should appear fully in the set of networks and density= 0.5, the percentage of subgraphs discovered from vertex- vertex template graph that are frequent across the given set of graphs is 24%. Another example is, for quasi appearance (θ)= 50% and DME density (min_d)= 50% the percentage for subgraphs (clusters) that are frequent across the collection of graphs is 30%. It can be inferred from the table that as the value of quasi appearance increases, the percentage of frequent subgraphs decreases.

Table 2. Result Matrix for Edge-Edge Template Graph approach

min_d	$Clusters$	$AvgSize$	$\theta = 100$	$\theta = 90$	$\theta = 80$	$\theta = 70$	$\theta = 60$	$\theta = 50$
50	5310	5.8	39%	39%	78%	95%	99%	100%
55	1615	5.3	49%	55%	84%	96%	99%	100%
60	571	4.8	63%	67%	85%	97%	99%	100%
65	210	4.5	72%	73%	85%	99%	99%	100%
70	73	4.5	75%	75%	88%	100%	100%	100%
75	33	4.4	67%	67%	79%	100%	100%	100%
80	15	4.1	87%	87%	93%	100%	100%	100%
85	3	4.1	87%	67%	100%	100%	100%	100%

Table 2 shows the percentage of dense subnetworks discovered from Edge- Edge template graph that actually appear in the initial set of graphs. The DME densities again are 50, 55, 60, 65, 70, 75, 80, 85. For each density value, the table shows the total number of clusters discovered using the DME approach.

The observations regarding the number of clusters, average cluster size are the same as in vertex-vertex template graph. The percentage of subnetworks discovered from the edge-edge template graph that actually appears in the set of networks for a given density of clustering are listed in the table. For example, consider the quasi appearance is 100% which means a subnetwork appears 100% in the set of networks. For density= 0.5, the percentage of subgraphs discovered from vertex-vertex template graph that actually exist in the give set of graphs is 39%.

Tables 1 and 2 show that Clustering on Edge-Edge template graph gives more promising figures than clustering on Vertex-Vertex template graph. Following are the observations:

1. The number of clusters discovered using edge-edge template graph approach is almost double than that by vertex-vertex template graph approach. For example, for DME Density = 0.5 the cluster size of edge-edge ad template-template are 5310 and 2936 respectively.

2. The average size of clusters for a given density is greater in case of edge-edge template graph approach. For example, considering the DME Density = 0.5 and quasi appearance is 50% then the average cluster size for edge-edge template graph and vertex-vertex template graphs are 4.7 and 5.8 respectively.

3. The percentage of dense subgraphs appearing in given set of networks is way higher in case of edge-edge template graph approach. For example, according to the tables for DME density = 0.5 the quasi appearance 100% which means the

whole subgraph appears in the given set of graphs, 39% of the total clusters are frequent if edge-edge template graph approach is used which is higher than the percentage in case of vertex-vertex template graph approach. Another example is if DME density (min_d)=0.5 and the quasi appearance $\theta = 50\%$ then 99.7% of the total clusters found using edge-edge template graph approach are frequent, which is a lot higher than the percentage of clusters that are frequent using vertex-vertex template graph (which is 30.42%) for the same set of constraints.

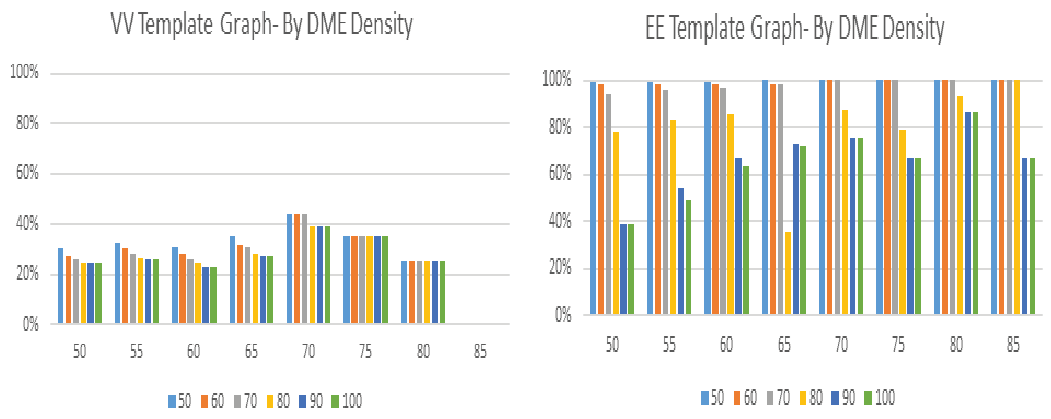


Fig. 12. Percentage of Quasi Frequent Subgraphs shown by Density

Figure 12 shows the result plots for the percentage of quasi frequent subgraphs for each of DME densities $min_d = \{50, 55, 60, 65, 70, 75, 80, 85\}$ using both vertex-vertex and edge-edge template graph approaches. For each of the min_d values, the results are further shown for θ=$\{50, 60, 70, 80, 90, 100\}$.

For example in case of vertex-vertex template graph approach, for min_d= 50 the percentage of subgraphs that are quasi frequent in the input graph collection for θ ={ 50, 60, 70, 80, 90, 100 } are {30, 28, 26, 25, 24, 24} respectively. And in case of edge-edge template graph approach, for the same value of min_d, the values corresponding to θ ={50, 60, 70, 80, 90, 100} are {100, 99, 95, 78, 39, 39} respectively (in percentage).

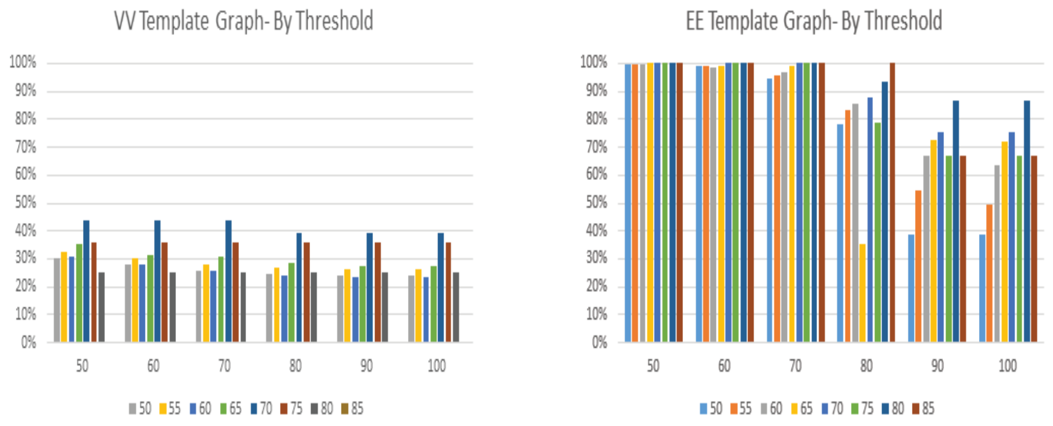


Fig. 13. Percentage of Quasi Frequent Subgraphs shown by Threshold

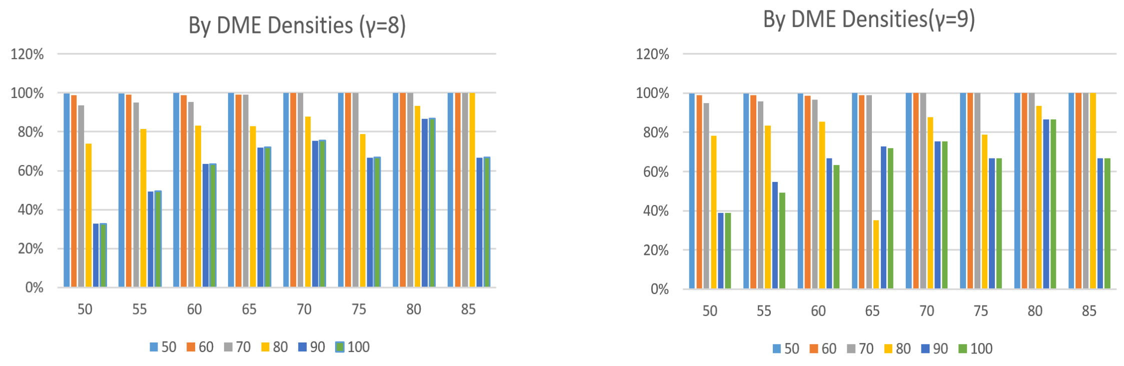
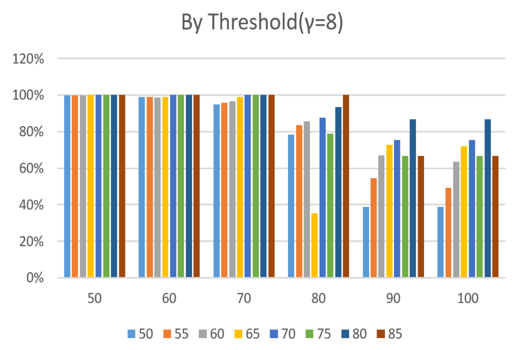


Fig. 14. Results for $\gamma = 8$ and 9 by min_d for Edge-Edge Template Graph

Figure 13 shows the result plots for the percentage of quasi frequent subgraphs for each of the values in Quasi Frequent set $\theta = \{50, 60, 70, 80, 90, 100\}$ using both vertex-vertex and edge-edge template graph approaches. For each of the θ values, the results are further shown for $min_d = \{50, 55, 60, 65, 70, 75, 80, 85\}$.



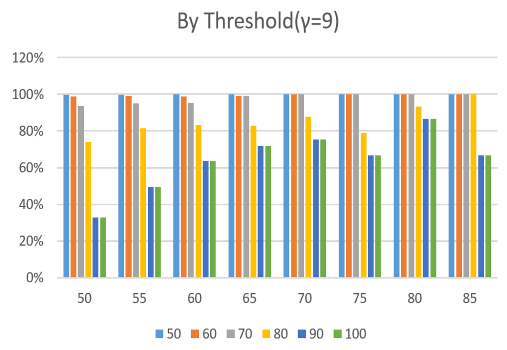


Fig. 15. Results for $\gamma = 8$ and 9 by θ for Edge-Edge Template Graph

For example in case of edge-edge template graph approach, for $\theta = 70$ the percentage of subgraphs that are quasi frequent in the input graph collection for $min_d = \{50, 55, 60, 65, 70, 75, 80, 85\}$ are $\{26,\ 28,\ 26,\ 31,\ 44,\ 36,\ 25,\ 0\}$ respectively. And in case of edge-edge template graph approach, for the same value of θ, the values corresponding to $min_d = \{50, 55, 60, 65, 70, 75, 80, 85\}$ are $\{95,\ 96,\ 97,\ 99,\ 100,\ 100,\ 100,\ 100\}$ respectively (in percentage).

Figure 14 compares the results for of quasi frequent subgraphs for $\gamma = 8$ and 9 in case of edge-edge template graph approach using bar plots by DME densities . The variable γ is the minimum frequency of appearance of a subparagraph in the given graph collection. The bar plots show that there is not much difference in percentages when γ is changed from 8 to 9. We see similar kind of observation in Figure 15 where the plots are by minimum threshold of appearance (θ).

CHAPTER 4. CONCLUSION AND FUTURE WORK

In this paper I have presented a new method to find quasi frequent subgraphs or frequent subgraphs in large network sets that can be used to discover recurrent patterns in scientific, spatial, and relational datasets. Such subgraphs can play an important role for understanding the nature of these datasets and can be used as input to other data-mining tasks. Using a summary graph to find communities in networks helps find interesting interactions and there are many established algorithms to do it.

The edge-edge template graph approach presented in this paper looks very promising in terms of finding larger number of denser clusters in the set of networks. My experimental evaluation shows that this approach can scale reasonably well to very large graph databases if the value if the value of support (γ) is not low. One of the reasons is that the connections that are not frequent are eliminated at the initial stage. So, the template graph is built on the edges that appear frequently across the set of networks. The results clearly show that edge-edge template graph approach is much more efficient in finding the dense interactions across networks. The algorithm is also capable of finding the quasi appearance of a subgraph across the networks. In this paper, the value of quasi appearance are 50%, 60%, 70%, 80%, 90%, 100%.

There are a few limitations that can be addressed as a part of future work. Firstly, we assumed that the vertices across all the graphs are the same in the collection. It sometimes doesn't hold true in real world networks, For example two different social networks might not have the same people connected in them. There could be more people involved in a network or two different sets of people in two different networks. So, the algorithm can be improved to work for different sets of vertices across the graphs in a graph collection. That way this algorithm can be used in the area of mining social networks to find useful connections.

Another functionality that can be added as a part of future work is further enhancing this algorithm to find quasi frequent subnetworks when the edges in the graphs have attributes attached to them. For example, the edges (connections) in a network can have different attributes associated with them that define the nature of these connections. For example, in a Facebook network the attributes associated with two people connected in the network are their common friends (or connections), common page likes, common TV show likes etc. The algorithm can be modified to find frequent subgraphs based upon the values of these attributes assigned to different connections or edges in the graphs.

The elimination of edges that are not frequent across the graphs during the initial stage is important since it makes the algorithm efficient in terms of running time and space required. But in many cases the edges that sparsely appear across the collection are significant for some analyses and should not be eliminated. Using the algorithm in that case may take larger amount of time and space. So the algorithm needs to be optimized to run for huge datasets if initial elimination step has to be skipped.

REFERENCES

[1] Susan Gunelius. The data explosion in 2014 minute by minute- infographic. 2014. http://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic/.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, 1994.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering*, ICDE '95, pages 3–14, 1995.

[4] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 USA, 2001.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. School of Computing Science, Simon Fraser University, Canada, 2000.

[6] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. A survey of algorithms for dense subgraph discovery. In Haixun Wang Charu C. Aggarwal, editor, *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 303–336. Springer US, 2010.

[7] Roded Sharan, Trey Ideker, Brian P. Kelley, Ron Shamir, and Richard M. Karp. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. In *Proceedings of the eighth annual international conference on Resaerch in computational molecular biology*, RECOMB '04, pages 282–289, 2004.

[8] Xiaoli Li, Min Wu, Chee-Keong Kwoh, and See-Kiong Ng. Computational approaches for detecting protein complexes from protein interaction networks: a survey. *BMC Genomics*, 11(Suppl 1):S3, 2010.

[9] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 30–36, 1998.

[10] R. N. Chittimoori, L. B. Holder, and D. J. Cook. Applying the subdue substructure discovery system to the chemical toxicity domain. In *Proceedings of the 12th International Florida AI Research Society Conference*, pages 90–94, 1999.

[11] H. Kalviainen and E. Oja. Comparisons of attributed graph matching algorithms for computer vision. In *Proceedings of STEP-90, Finnish Artificial Intelligence Symposium*, pages 354–368, Oulu, Finland, June 1990.

[12] V. A. Cicirello. Intelligent retrieval of solid models. Master's thesis, Geometric and Intelligent Computing Laboratory, Department of Mathematics and Computer Science Philadelphia, Drexel University, Pa. 19104, 1999.

[13] D. Dupplaw and P. H. Lewis. Content-based image retrieval with scale-spaced object trees. In *Proceedings of SPIE: Storage and Retrieval for Media Databases*, volume 3972, pages 253–261, Oulu, Finland, 2000.

[14] C.-W. K. Chen and D. Y. Y. Yun. Unifying graph-matching problem with a practical solution. In *International Conference on Systems, Signals, Control, Computers*, 1998.

[15] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In *Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.

[16] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD'00, pages 13—23, Lyon, France, 2000.