MINING FREQUENT COHERENT PATTERNS FROM WEIGHTED GRAPHS

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

By

Syed Kutub Uddin Ahmed

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

June 2014

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Mining Frequent Coherent Patterns from Weighted Graphs

By

Syed Kutub Uddin Ahmed

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. Sameer Abufardeh

Dr. Shafiqur Rahman

Approved:

9-18-2014
Date

Dr. Kenneth Megel
Department Chair

ABSTRACT

Current research on network analysis; such as community detection, pattern mining and many other graph mining application mostly focus on large social or biological networks. Such experiments may find interesting patterns that helps us to understand the unknown relationship within the network. Sometimes the size of the input networks are so big that it needs an efficient algorithm to overcome the time and space complexities. In this paper we modify an existing algorithm that finds the maximal patterns from a set of input networks. Maximal patterns are those patterns that are not part of any frequent patterns. We introduce a new relational attributes to our algorithm from the input networks, we call them the edge attributes. We have tested our algorithm on a co-author relationship database; and after analyzing we have found some interesting characteristics of the input dataset.

ACKNOWLEDGMENTS

First of all, I would like to thank the Almighty to give me the courage, strength and knowledge to accomplish this study.

My sincere thanks to my thesis advisor Dr. Saeed Salem for his continuous guidance, patience and direction at every stage from the very beginning to the end of this research. Through his teaching and guidelines, he has greatly contributed to my understanding of scientific research on the field of Data Mining. His respected comments and discussions helped me to write this paper.

I would also love to express my gratitude to Dr Shafiqur Rahman and Dr Sameer Abufardeh for serving on my committee.

I am very grateful to Dr. Kendall Nygard. His valuable advise and support has helped me conquer the challenges throughout my graduate studies.

I also want to thank North Dakota State University and the Department of Computer Science for offering such an wonderful environment for research and study.

I would also like to offer my thanks to Tyler Brazier and Rami Alroobi for all the suggestions they gave to me during my research. Their valuable support was of a great source of inspiration for my completion of this degree.

DEDICATION

To My beloved Parents and Family, who have always inspired me with unconditional support, blessings and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\mathcal{I}$ ..... A set containing $\{x_1, x_2, \ldots, x_m\}$ items

$\mathcal{I}^k$ ..... Set of all possible subsets of $\mathcal{I}$ of size k

$\mathcal{D}$ ..... A database of n transactions

$\mathcal{F}$ ..... The set of all frequent patterns

σ^* ..... User defined support threshold

f_k ..... The frequent pattern found at level k

$\mathcal{M}$ ..... The set of all maximal patterns

$\mathcal{C}$ ..... The set of all closed patterns

G ..... A graph containing vertices and edges connecting the vertices

V ..... The total set of vertices for a graph

E ..... The total set of edges for a graph

$|V|$ ..... The number of vertices in a graph

$|E|$ ..... The number of edges in a graph

$|G|$ ..... The size of the graph, G

$\mathcal{G}$ ..... A relation graph database

g_k ..... The subgraph of size k

$N(e_i)$ ..... Neighboring edges of e_i

c_k ..... Candidate edgeset of subgraph g_k

d ..... The set of discovered edges

MFS ..... The set of maximal patterns

δ^* ..... User defined cohesive threshold

$w_i(e_j)$ ..... Weight of edge e_j in graph G_i

$\mathcal{P}_k$ ..... Pattern of size k

1. INTRODUCTION AND RELATED WORK

Data mining algorithms works fine with the traditional data and finds useful information from it. In fact, the research is increasing day by day to grasp complex scientific and commercial domains [4, 1, 3, 13, 33]. Both sequential and non-sequential approaches work fine to search for frequent patterns, but non-traditional data needs a more comprehensive way of presenting their complex structure. Graph structure has emerged as a popular data structure for storing the relationship in a complex system.

Many researchers have proposed efficient algorithms for graph mining [16, 31, 9, 20]. Since then, graphs have become very popular to model relationships among entities in various areas. It gives a clear and concise representation of complex data with nodes corresponding to entities and edges reflect relationships between the entities.

An example of a graph database is a protein-protein interaction (PPI) network, where a vertex represents a protein and physical interactions between various proteins are represented with edges connecting them. Analysis of PPI networks using graph-theory framework reveals important molecular interaction informations that helps understanding cellular organization, functional hierarchy and evolutionary conservation [17, 27, 22, 8, 15].

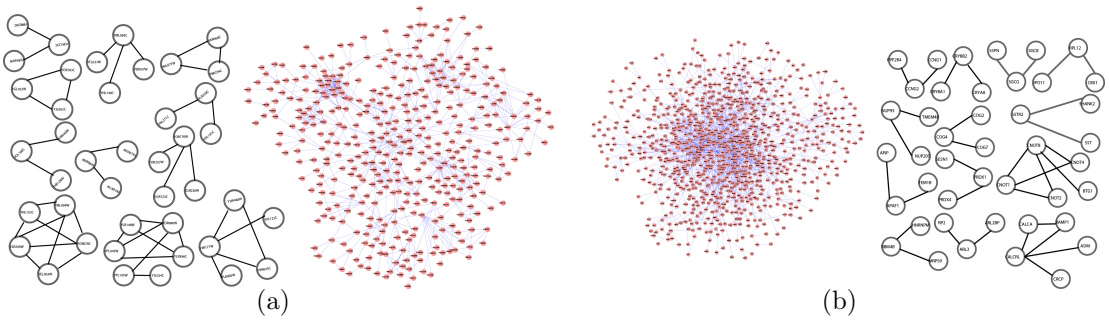


Figure 1. Yeast Gene (a) and Human Gene (b) Interaction network.

Another area where graphs are used to model interactions is the social networks. Social networks are widely used for communication and sharing contents between individuals. Here, a vertex represents a person and an edge connecting two vertices exhibits the friendship among them. It is a group of communities where all the people inside the community interact frequently with each other. It is likely that if a largest number of people have interest in one particular subject, then any personal choice will be interesting to others as well.

The problem is how to efficiently detect groups or communities from these type of giant networks since no explicit information is available [7]. Newman and Girvan [23] uses a *betweenness* property to find out the community structure. Ruan et al. [25] has shown that using contents and link information in graph structure has eliminated noise while discovering communities. Tang et al. [28] presented a joint optimization framework to detect communities by integrating multiple data sources. Qi et al. [24] proposed an algorithm to show that instead of node content, the edge content has the flexibility to detect community effectively.

Many data mining applications uses frequent itemset mining technique as their fundamental approach [12]. It is the first step for important discovery tasks like finding association rules, strong rules, correlations, sequential rules, episodes, multi-dimensional pattern [34]. Frequent itemset mining is described as; find frequent set of items from a given large database with item transactions and a user-defined threshold known as support. An itemset is frequent if it has occurred in at least number of graphs equals to support.

Frequent itemset mining is always costly especially when the size of the database is huge. Most of the frequent itemset mining approach are a variant of the *A-priori* algorithm [4], a bottom up approach that finds frequent items efficiently. *A-priori* uses a downward closure to prune the search space to find the frequent itemsets. Still

the size of the frequent itemsets is big since it has redundant data, as subsets of a larger frequent itemset also be present in the database. Another modified approach of mining frequent itemset is mining frequent maximal itemset, i.e. keeping only the larger patterns that will have the same properties but allows to compress the output data.

Some scientific applications uses multi-dimensional complex database and needs more comprehensive way to present. In these cases a graph structure is more easy to represent. Various objects from the database can be represented as node, interaction between the nodes can be represented as an edge. A mining approach on the graph database can find interesting behavior among the objects. Mining for frequent subgraphs is similar to itemset mining; find all the frequent subgraphs that has occurred frequently enough in the entire graph database.

Some of the approaches require that all the nodes in the graph has to be unique (MULE [18]), i.e. no two nodes in the graph can have the same label. Algorithms like FSG [20], gspan [31], Spin [16] does not require the labels to be unique. But these approaches has helped solving problems that was not possible to solve by basic itemsset approach like common motif discovery in DNA, finding recurrent substructure in chemical compounds and etc.

Using of *Graph theoretic formalism* has simplified the analysis of commercial, scientific and technological data. It describes efficient frameworks for clustering, shortest-path computation, graph matching, graph alignment, subgraph homeomorphism and graph mining, but still finding these informations from a graph is costly. Most of these data analysis includes frequent pattern mining from the graph database. It is similar to the frequent item set mining in data mining literature. Due to subgraph isomorphism, finding frequent patterns from a graph database NP-hard problem [20].

Interactions in networks can be weighted. Examples include; the number of posts in Facebook, the number of papers in a co-author relationship and the correlation of expression in gene co-expression networks. In this paper we develop a method to use a summary graph to mine maximal-cohesive patterns. This algorithm uses several powerful pruning techniques to prune the search space, thus improving the performance of the algorithm. The proposed algorithm is based on the MULE algorithm[18]. MULE does not take weights of a interaction in a networks, hence there is a need for a new algorithm.

The rest of the paper is structured as follows: Section 2 describes the basics of frequent pattern mining, Section 3 gives the problem definition and some preliminary definitions related to graph mining. Section 4 describes the basic MULE algorithm and our proposed methods. The results of the algorithm on real world data is shown in Section 5. Lastly, Section 6 presents the conclusion and how this work might be extended in the future.

2. MINING FREQUENT PATTERNS

Finding frequent patterns in a graph database follows a similar approach to that employed in frequent item set mining in the market-basket example [1]. In retail businesses, management analyzes the customer shopping carts to mine interesting patterns. The analysis might find some patterns about how likely are two sets of items to co-occur or to conditionally occur. For example rules like *"Customers who buy Bread and Milk also tend to buy Eggs"* can be deduced from the analysis, which may result in arranging bread, milk and egg in the same aisle in a grocery store. Discovering common subsequences and motifs in biomolecules is another example of frequent pattern mining [6]. Recent research on molecular biology has evolved to produce a new generation of bimolecular interaction data, analyzing these relationships and interactions convey functional, structural, and evolutionary information [14]. Here, in this section we introduce the itemset mining algorithm to familiarize the reader with the topic.

	A	B	C	D
1	0	1	1	1
2	1	1	0	0
3	0	1	0	1
4	0	0	1	1
5	0	0	0	1

(a)

	t(i)
1	BCD
2	AB
3	BD
4	CD
5	D

(b)

	A	B	C	D
$\mathbf{t}(X)$	2	1 2 3	1 4	1 3 4 5

(c)

Figure 2. Example of different database representations. (a) shows the Binary Database , (b) shows the Transaction Database and (c) shows the Vertical Database.

In this paragraph we present the database representation as described in [32]. Let $\mathcal{I} = \{x_1, x_2, \ldots, x_m\}$ be the set of items. A set X of size k is said to be an *itemset* only if $X \subseteq \mathcal{I}$, and is called a k-itemset. The set of all possible subsets of $\mathcal{I}$ of size k is denoted by $\mathcal{I}^k$. We define a database of n transactions as $\mathcal{D} =$

$\{(1, T_1), (2, T_2), \ldots, (n, T_n)\}$. For an itemset X, the set of all subset of X, i.e. the power set of X is denoted as 2^X. The set of items contained in transaction, tid t is denoted as $\mathbf{i}$(t). Figure 2(b) shows the corresponding transaction database, the binary database represented in 2(a); 2(c) shows the vertical representation of the binary database. So the first transaction in figure 2(b) is (1,{B,C,D}) means items B, C and D are present in transaction one. The vertical notation of item D would be (D,{1,3,4,5}) meaning that item D appears in transactions {1,3,4,5} (shown in 2(c)).

Support of an itemset can be defined in terms of cardinality of its corresponding tidset $\mathbf{t}(X)$. Here tidset(X) is the set of transactions that contain all the items in X. More formally:

$$t(X, \mathcal{D}) = \{(t_i, \mathbf{i}(t_i)) \in \mathcal{D} \text{ — } X \subseteq \mathbf{i}(t_i)\}$$

The support of an itemset X in database $\mathcal{D}$ is the number of transactions in $\mathcal{D}$ that contains X, denoted as $sup(X, \mathcal{D})$:

$$sup(X, \mathcal{D}) = |\mathbf{t}(X, \mathcal{D})|$$

Frequent Itemset: An itemset X is said to be frequent if $sup(X) \geq minsup$, where $minsup$ is a user defined threshold, the set of all frequent itemsets is denoted as $\mathcal{F}$.

Definition 1 (Frequent Itemset)**.** *If the frequency of an itemset X is greater than minsup then FREQ(X)=TRUE, i.e. $FREQ(X) = TRUE \iff sup(X) \geq \sigma^*$.*

Definition 2 (Frequent Itemset Mining Problem)**.** *Given a database $\mathcal{D}$ of n transactions and a minsup σ^*, then the problem is defined as to find $\mathcal{F}$, the set of frequent patterns.*

To find out the frequent item sets, a brute force approach is to run over all the items in power set of $\mathcal{I}; P(\mathcal{I})$ [29] and check their frequency. This will ensure no combination of items in set S is tried twice. Rymon [26] proposed similar way of searching through systemic set enumeration.

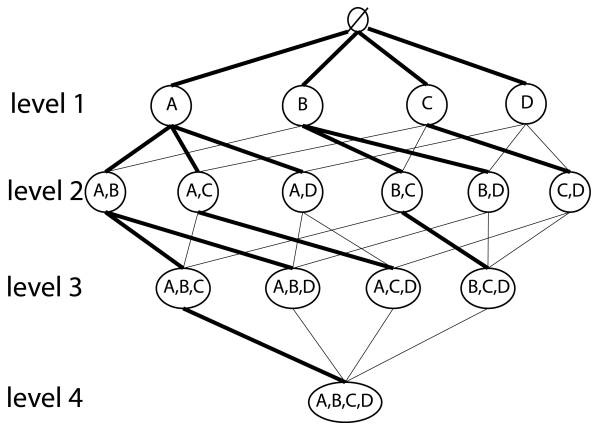


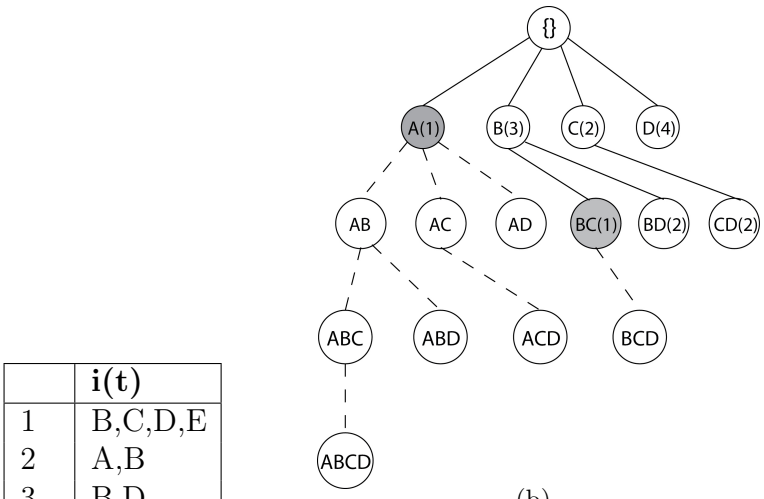
Figure 3. Set enumeration tree of items in $\mathcal{I}$. It shows a lattice structure for the database shown in figure 2(a)), here any two itemsets X and Y are linked iff $X \subseteq Y$ and $|X| = |Y| - 1$. The itemsets in the lattice can be enumerated using either breadth-first (BFS) or depth-first (DFS) in the prefix tree, drawn with bold line. Here in the prefix tree two items X and Y are connected by bold line iff X is a direct subset and prefix of Y.

An example of a set enumeration tree for the set $\mathcal{I} = \{A, B, C, D\}$ is shown in Figure 3 with bold line. The Set enumeration tree starts with an *empty set* or *null* to indicate that at level zero it has no member in its database. Then it discovers all its members one by one in sorted order and adds them in a breadth-first or depth-first approach. In a breadth-first approach, it will find all the single frequent items (ex: $\{A, B\}$) at level one, at level two it will have all the frequent paired items (ex: $\{AB, AC\}$), so at level n it will discover all the frequent item sets which has exactly n items in it. Obviously, it is easier to understand that for a large sized set $\mathcal{I}$, it may run out the system memory before it finds out all the frequent items. Depth-first approach works with an extra modification. It builds it database for single frequent items exclusively out side of the main process. This set of single valued frequent items is known as *member set*. Now within a recursive procedure it combines two frequent items to grow a newer pattern (so at level $k + 1$ it will create $f_{k+1} = f_k \cup f'_k$) and checks it frequency. In Figure 3, the depth-first traversal order starts with item $\{A\}$, as it traverse grows the patterns

$\{A, B\}, \{A, B, C\}, \{A, B, C, D\}, \{A, B, D\}, \{A, C\}, \{A, C, D\}, \{A, D\}$ chronologically all patterns with a prefix "A". Then it will go back to the item $\{B\}$ on the first level. Here we define an anti-monotone constraint:

Definition 3 (Anti-monotone constraint)**.** *A constraint P is anti-monotone for an itemset, X, if the following condition is satisfied:*

$$P(X) = TRUE \implies P(X') = TRUE \,, \forall\, X' \subseteq X$$



	i(t)
1	B,C,D,E
2	A,B
3	B,D
4	C,D
5	D

(a)

(b)

Figure 4. Effect of pruning on the prefix tree after applying *A-priori*. The dotted line in the figure shows the pruned branches. Solid line represents the itemsets that has been generated by *A-priori*. Itemsets with shaded oval ($\{A, BC\}$) are the infrequent patterns.

Let $X, Y \subseteq \mathcal{I}$ be any two itemsets. We can say that $sup(X) \geq sup(Y)$ when X is a subset of Y, i.e. $X \subseteq Y$. So from this relation we can say that if X is frequent then any $P \subseteq X$ is also frequent and if X is not frequent then any $Q \supset X$ is not frequent. Agrawal and Srikant [2] developed *A-priori* algorithm based on this observation using set enumeration tree. It stops generating candidate patterns when

it reaches an infrequent pattern as no superset can be frequent. In *A-priori* it avoids any candidate that has an infrequent subset. The rest of the *A-priori* algorithm is similar to the level wise breadth-first approach describe earlier in this chapter.

Figure 4 shows the effect of pruning with a support of 2 on the database shown in figure 2(b). Here, since support of A is one, i.e. $sup(A) = 1$, which is not frequent, so none of its supersets $\{AB, AC, AD, ABC, ABD, ACD, ABCD\}$ will be grown by *A-priori* algorithm. The reason is from the observation that, since the subset $\{A\}$ is not frequent none of its superset can be frequent. Same thing is true for the itemset $\{BC\}$, so its superset $\{BCD\}$ will not be generated by the algorithm. Comparing to to the set enumeration approach shown in figure 3, *A-priori* checks less number of items, reports the same frequent itemsets, and is much faster.

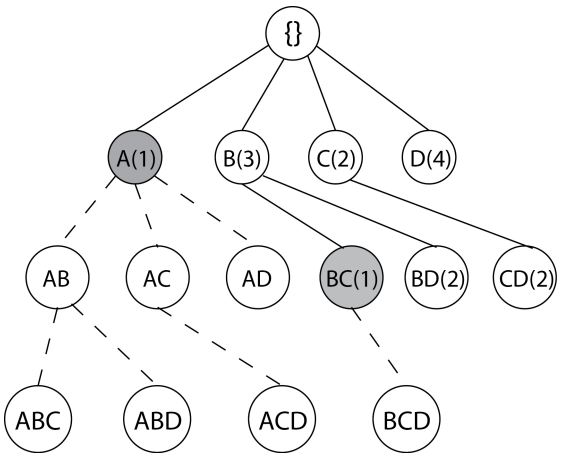


Figure 5. Tidset intersection approach on a database. The database shown in Figure 2.

Zaki and Gouda [33] proposed the Tidset Intersection algorithm for finding frequent itemsets from the database $\mathcal{D}$ shown in Figure 6. The approach is simple, instead of generating subsets of each transaction and count their support, the transaction id sets (tidsets) has been used directly in the algorithm. If two itemsets are frequent in the current iteration, then these two can be merged together to create a new itemset in the next iteration. So the occurring tidsets of the new pattern can

be computed simply by intersecting the tidsets of the candidate sets. For example in figure 2, tidsets of B i.e. $\mathbf{t}(B) = \{1, 2, 3\}$ and $\mathbf{t}(D) = \{1, 3, 4, 5\}$. Now the support of the itemset BD can be determined by $\mathbf{t}(BD) = \mathbf{t}(B) \cap \mathbf{t}(D) = \{1, 2, 3\} \cap \{1, 3, 4, 5\} = \{1, 3\}$, so the frequency of itemset BD is 2. The algorithm does not include the infrequent patterns, so pruning strategy is also employed by this is faster approach of support computation. The set enumeration tree for the example is shown in Figure 5.

Algorithm 1: Algorithm IntersectTidsets

```
// Initial Call : IntersectTidsets({⟨i, t(i)⟩ : i ∈ I},minsup)
IntersectTidsets(P, minsup):
1.    foreach ⟨X, t(X)⟩ ∈ P do
2.      P_X ← ∅
3.      foreach ⟨Y, t(Y)⟩ ∈ P with Y > X do
4.        N_XY = X ∪ Y
5.        t(N_XY) = t(X) ∩ t(Y)
6.        if sup(N_XY) ≥ minsup then
7.          P_X ← P_X ∪ {⟨N_XY, t(N_XY)⟩}
8.          print N_XY, sup(N_XY)
9.        endif
10.     endfor
11.     IntersectTidsets(P, minsup)
12.   endfor
```

Figure 6. Tidset Intersection Approach - The Eclat Algorithm.

Both the breadth-first and Depth-first approach work fine if the intention is finding the frequent patterns only. But numerous scientific and commercial application domains produces abundant transactional data, where the search space is enormous. A complete search for frequent patterns has to compute over an overwhelming size of data which is beyond the scope for analysis. Now most research focuses on how to reduce the frequent patterns to a smaller summary set that contains representative, non-redundant and discriminative patterns [5].

It is evident that frequency constraint is an anti-monotone which can be employed in frequent pattern set mining to prune the searching branches. But identifying huge number of similar frequent itemsets can be redundant. The results from a frequent itemset mining can be narrowed down. We can only keep the large frequent itemsets, as all the subset of a large frequent itemset will also be frequent. So instead of the largest set of groups, consider only those itemsets that still satisfies the *frequent* property will be more interesting. Gouda and Zaki [10] proposed a method to mine condensed representation of the frequent itemset known as *maximal frequent itemset*, representing partial frequent itemsets removing all redundant information that still holds the same characteristics of the original set enumeration tree. This has reduced the overhead to analyze the correlations found, at the same time reduces the cost of data storage and computation. In itemset mining concept, a frequent itemset $X \in \mathcal{F}$ is called maximal iff it has no frequent supersets.

Definition 4 (Maximal frequent)**.** *An itemset, X, is maximal if the following condition is satisfied:*

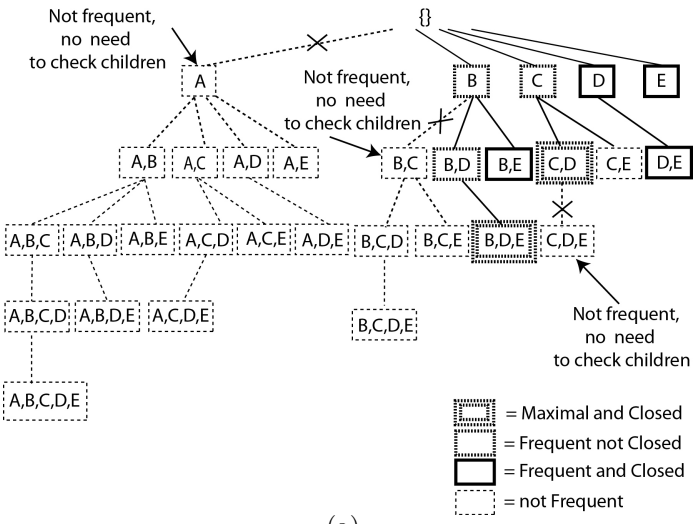
$$FREQ(X) = TRUE \, , \nexists \, X' \supseteq X \wedge FREQ(X') = TRUE$$

Enumerating only maximal itemsets offers more opportunities for pruning. Any node with a frequent child cannot be maximal. After pruning for frequency, only the leaf nodes are potential maximal frequent nodes. Let $\mathcal{M}$ be the set of all maximal frequent itemsets, then $\mathcal{M}$ is defined as :

$$\mathcal{M} = \{X | X \text{ is frequent and } \nexists Y \supset X, \text{ such that } Y \text{ is frequent}\}$$

Maximal patterns from a database gives us information about all the frequent items. If we find out all the subsets of maximal patterns then we will know all the

frequent patterns in the database. Figure 7 shows an example of set enumeration tree with support 2. Itemset $\{BDE\}$ is a maximal pattern since it has no frequents superset.



(a)

	i(t)
t_1	B,C,D,E
t_2	A,B,E
t_3	B,D,E
t_4	C,D
t_5	D,E

(b)

sup	**i(t)**
4	D,E
3	B,BE,DE
2	C,BD,CD,BDE

(c)

	itemset
$\mathcal{F}$	B,C,D,E,BD, BE,CD,DE,BDE
$\mathcal{C}$	D,E,BE, CD,DE,BDE
$\mathcal{M}$	CD,BDE

(d)

Figure 7. Set enumeration tree for $\mathcal{I}$= {A,B,C,D,E}. (a) shows the frequent itemset enumeration tree with minimum support of 2. The table in (b) shows the items in each transaction. Itemset that has frequency from 2-4 is listed in (c). Frequent($\mathcal{F}$), Closed($\mathcal{C}$) and Maximal($\mathcal{M}$) itemsets are listed in (d).

But maximal pattern is a lossy compression, since we can not generate all frequent patterns with their frequencies from the set of maximal frequent patterns. Therefore we need another summarization technique, [34] proposed an algorithm to mine all the frequent closed patterns from a database while given a user defined support. A close pattern doesn't have any superset with the same frequency. So subset of a close pattern tells us that they occurred at-least more than the pattern. Let $\mathcal{C}$ be the set of all closed frequent itemsets, which can be written as:

$$\mathcal{C} = \{X | X \text{ is frequent and } \nexists Y \supset X \text{with } sup(X) = sup(Y)\}$$

Itemset $\{BD\}$ in Figure 7, is not closed since its superset $\{BDE\}$ has same support, i.e. $sup(BD) = sup(BDE)$ and $BDE \supset BD$. Itemset $\{DE\}$ has no child, but still it is not a maximal pattern as $BDE \supset DE$. Itemset $\{B\}$ is frequent but not closed again for the same reason that one of its superset $\{BE\}$ has the same support. The frequent patterns $\mathcal{F}$, the maximal patterns $\mathcal{M}$ and the closed patterns $\mathcal{C}$ for $\mathcal{I} = \{A, B, C, D, E\}$ are highlighted in Figure 7(c).

3. PROBLEM DESCRIPTION

In this section, we define some terms related to graph mining that we will use throughout the chapter. The graphs considered here 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.

3.1. Preliminary Definitions

A graph $G = (V, E)$, consists of a set of vertices $V = \{v_1, v_2, \cdots, v_n\}$, and a set of edges $E = \{e_1, e_2, \cdots, e_m\}$ where $E \subseteq V \times V$ connecting the vertices. Two vertices u and v are *adjacent* if there is an edge connecting u and v. The degree of a vertex v is denoted by $deg(v)$ and is the number of edges connected to v. Nodes of a graph can have labels and edges can have weights. The weight on edge (u,v) $\in E$ is denoted as $w(u, v) \in \mathbb{R}$. The size of a graph G, denoted $|G|$, is the cardinality of the edge set (i.e., $|G| = |E|$). The vertex set and edge set of a graph G are denoted by $V(G)$, and $E(G)$, respectively.

A graph $G'(V', E')$ is a subgraph of G, denoted as $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. A subgraph G' of G is said to be induced if for $x, y \in V(G')$, there is an edge between x and y in G' if and only there is an edge between x, y in G, i.e. $(x, y) \in E(G)$. The subgraph G' is said to be induced from G by the vertex set $V(G')$ and is written as $G[V(G')]$.

Figure 8(a) is a uniquely labeled graph since none of its node has repeating labels, a graph is a *uniquelabeled* graph when none of its vertex labels are repeated, otherwise it is called as non-unique labeled graph. The subgraph $G' = \{V = (a, c, d, e, h), E = \{(a, c), (a, d), (e, d), (e, h)\}\}$ is not induced subgraph of G, since two edges $\{(e, c), (a, e)\}$ that are contained in G are not in G'. In 8(b) the subgraph in bold line is induced as, all the edges connecting node $\{a, b, c, d\}$ in G, are also in G'.

Definition 5 (Support)**.** *Given a relation graph dataset,* $\mathcal{G} = \{G_1, G_2, ..., G_n\}$*, where* $G_i = (V_i, E_i)$*, the support of a graph* G *is the number of graphs (in* $\mathcal{G}$*) where* G *is a subgraph, defined as:*

$$occurrences(G, \mathcal{G}) = \{G_i | G \subseteq G_i, G_i \in \mathcal{G}\}$$

$$sup(G, \mathcal{G}) = \mid occurrences(G) \mid$$

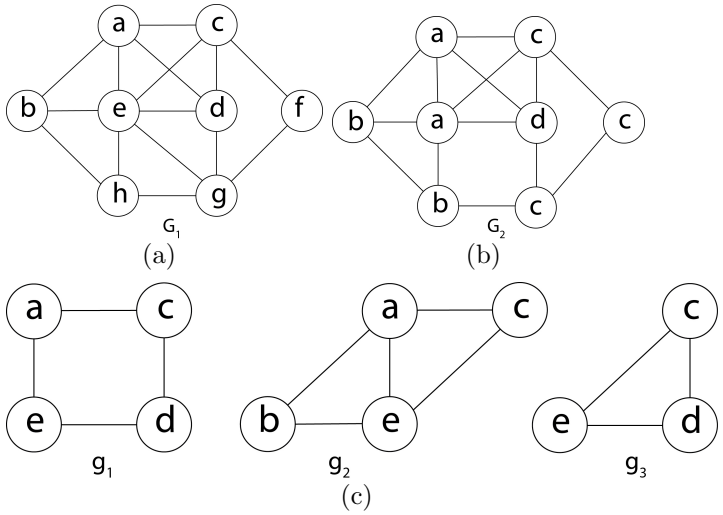


Figure 8. Unique and non-unique labeled graph. In the above figure, (a) shows a unique labeled graph (G_1), (b) shows a non-unique labeled graph (G_2). The subgraph in (c), $g_1 = \{V = (a, c, d, e), E = \{(a, c), (c, d), (a, e), (d, e)\}\}$ is not induced since (a, d) and (c, e) is not connected. $\{g_2, g_3\}$ are the induced subgraphs of G_1.

We can find support of a connected subgraph G' by simply iterating over each graph, $G_i \in \mathcal{G}$ and check whether $G' \subseteq G_i$; i.e., checking if the connected subgraph is a subgraph of G_i. If it is subsumed, then we simply increment the counter and move forward with the next graph. During frequency count, subgraph checking is sometimes very costly due to NP-hard subgraph isomorphism problem [30]. Kuramochi and Karypis [20] described a method, *Canonical Labeling* to overcome the duplicate consideration of subgraphs and hence avoids the redundancy during frequency count.

Definition 6 (Frequent Subgraph Mining)**.** *A subgraph G' is frequent in the graph database $\mathcal{G}$, if it is evident in minsup graphs. The goal of frequent graph-mining is to mine the set of frequent subgraphs $\mathcal{F} = \{G_1, G_2 \ldots, G_{|\mathcal{F}|}\}$ with $sup(G', \mathcal{G}) \geq \sigma^*$, where σ^* is a support threshold provided by the user.*

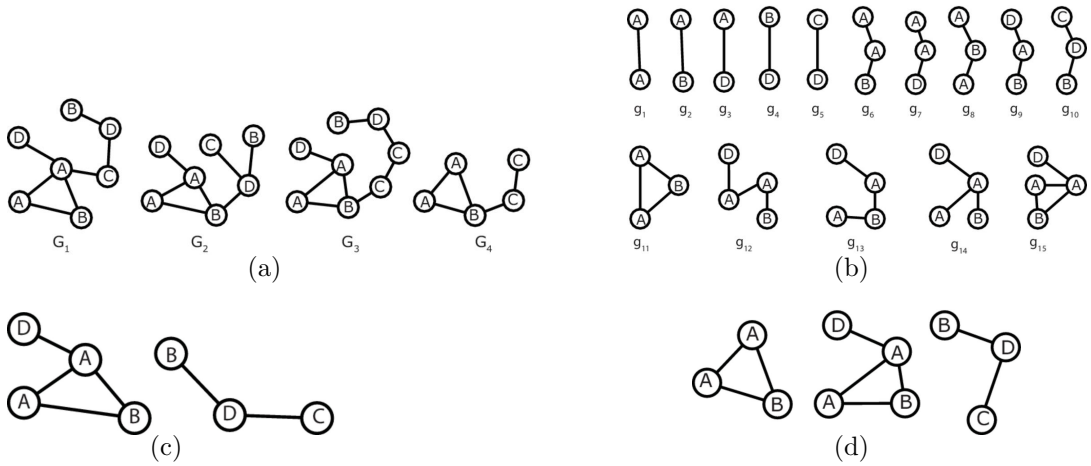


Figure 9. An example of non-unique labeled graph database. (a) shows the sample graph, (b) shows all the frequent patterns ($\mathcal{F}$) found in the database with $sup, \sigma^* = 3$. (c) shows the Maximal patterns ($\mathcal{M}$) and (d) shows the Closed patterns ($\mathcal{C}$).

Figure 9 shows an illustrating example of finding frequent subgraphs from the graph database shown in 9(a), which has 4 graphs in the database. The vertex set of the graph database $\mathcal{G}$ is $\{A, B, C, D\}$. Here the edges are not labeled.

Figure 9(b) shows all the frequent subgraphs found with support at least 3. For example the graph, g_8, $(V = \{a, a, b\}, E = \{(a, b), (a, b)\})$ is frequent since it is present in $\{G_1, G_2, G_3, G_4\}$.

In figure 9(b), it is noticeable that the frequent subgraphs found for the graph database in 9(a) are redundant. For example $\{g_1, g_2, g_6, g_8, \}$ in 9(b) are subset of g_{11}, if we look at the supersets(9(c)), we can see that subsets of these supersets are also frequent. So frequent subgraph mining needs a way of summarizing similar to maximal itemset mining approach described in chapter 2.

Definition 7 (Closed Frequent Subgraph)**.** *A frequent graph is closed if it has no frequent super-graph with the same support, i.e.* $\nexists G' : G \subseteq G'$ *and* $sup(G) = sup(G')$.

$$ClosedFrequentSubgraph, C = \{G' | G' is closed\}$$

The problem of mining frequent closed subgraph is, finding all the subgraph in the database that is closed, more formally the closed pattern set is described as:

$$\mathcal{C} = \{G' | G' \text{ is frequent and } \nexists G^* \text{ s.t. } G^* \supseteq G', \text{with } sup(G', \mathcal{G}) = sup(G^*, \mathcal{G})\}$$

Mining closed frequent subgraph from a graph database $\mathcal{G}$, is to find all the connected subgraphs G' such that $\sup(G', \mathcal{G}) \geq \sigma^*$, this means that G' is frequent subgraph in the graph database $\mathcal{G}$ and there is no frequent subgraph $G^* \in \mathcal{G}$ s.t. $G' \subseteq G^*$ and $\sup(G', \mathcal{G}) = \sup(G^*, \mathcal{G})$. Figure 9(d) shows the set of closed frequent subgraph patterns (total 3 closed frequent patterns found) with support = 3.

Mining for frequent subgraphs or closed frequent subgraphs does not help much, since it will produce huge amount of redundant data. From the figure in 9 we have seen 15 frequent subgraphs and 3 closed frequent subgraphs produced for simple graph database of size 4.

Again most of the frequent subgraphs in 9(b) are redundant. For example, the subgraph $V = \{a, b\}, E = \{(a, b)\}$ in g_2 appeared in $\{g_6, g_8, g_9, g_{11}, g_{12}, g_{13}, g_{14}, g_{15}\}$. So if we mine for a large graph database it will produce massive data which is another problem for data analysis and cause information overload. We want to shrink the resultant subgraphs that still holds the same characteristics with no loss of information. Huan et al. [16], Koyutürk et al. [18] and Hu et al. [15] proposed algorithms for mining maximal frequent subgraphs that has efficiently compressed the output of frequent subgraphs.

Definition 8 (Maximal frequent subgraph)**.** *A frequent graph is maximal if it has no frequent super-graph,i.e.* $\nexists G' : G \subseteq G'$ *and* G' *is frequent. Mining frequent*

subgraph is finding all the subgraph in the database that is frequent, denoted by $\mathcal{M}$ *and defined as :*

$$\mathcal{M} = \{\mathcal{G}' | \mathcal{G}' \text{ is frequent and } \nexists G^* \supset G', \text{ such that } G^* \in \mathcal{F}\}$$

Mining maximal frequent subgraphs refers to the problem of mining set of subgraphs $G' \in \mathcal{G}$ and $\text{supp}(G', \mathcal{G} \geq \sigma^*$, means they have to be frequent in graph database $\mathcal{G}$ and there is no such frequent subgraph G^* that subsumes G'. Fig 9(c) shows an example of maximal frequent pattern mining with support $\sigma^* = 3$. It shows that only 2 maximal frequent patterns found in the graph graph database shown in figure 9(a). By definition, the relationship among frequent, closed and maximal patterns can be stated as $\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{F}$. Looking at the patterns in frequent subgraphs [9(b)] and closed subgraphs [9(d)] it is clear that they are subsets of the maximal patterns in 9(c). So keeping only maximal patterns in the database will obviously occupy much less memory than keeping frequent or closed patterns and allow researchers to look at the summary set of frequent patterns.

We might want to mine interesting patterns. An interestingness constraint can be very general and involves many conditions. In this work, we only consider anti-monotone constraints so that the exponential search space of the problem can be reduced by pruning the search space.

Definition 9 (Anti-monotone constraint)**.** *A constraint* R *is anti-monotone if a graph* G *satisfies the constraint, implies that all its subgraphs also satisfy the same constraint, i.e.,* $R(G) = 1 \implies R(G') = 1$ *for all* $G' \subseteq G$*. Or inversely, if a graph does not satisfy the constraint, then none of its super-graphs do, i.e., if* $R(G) = 0 \implies R(G') = 0$ *for all* $G' \supseteq G$*.*

An example of an anti-monotone constraint is the frequency threshold, if it less than the user-defined threshold σ^* we can simply ignore that search space. If the

support of a subgraph G' is less than the user defined threshold then G' does not satisfy the constraint.

We can enforce more constraints that takes edge labels into consideration. Edges in the subgraph must have the same attribute values, or it can differ by at most a user-defined threshold.

4. MULE ALGORITHM AND OUR PROPOSED APPROACH

In this section we describe the original MULE algorithm[19] and how it works. MULE algorithm finds frequently occurring subgraph in metabolic pathways extracted from the *KEGG* database. Each enzyme is represented using an identical node, there exists a directed edge from one enzyme to another enzyme in the graph if and only if the second enzyme consumes a product of the first one [19]. So the graph model represents the metabolic pathways using simple directed graphs that efficiently captures these informations [19].

Mining all frequent patterns will result in redundant patterns since all subgraphs of a larger pattern are also frequent and will be in the database as well. Hence the MULE algorithm mines only maximally frequent patterns in the database to avoid redundancy. Moreover, as these are maximally occurring patterns it is guaranteed that there are no superset in the database that contains the same edgesets. To avoid considering the same edgeset twice, the algorithm enumerates frequent patterns by adding connected frequent candidate edges in a depth-first enumeration approach based on backtracking [11].

Koyutürk et al. [18] proposed this algorithm, the basic MULE for frequent subgraph mining shown in Figure 10. Initially the algorithm enumerates over the graph database, if an edge is present in the graph it is recorded. At the same time candidlate edges for each edges also updated. Finally when enumeration over the graph database is done, the edge list has the information about all edges that are present in the whole graph database. The total count of an edge, i.e., how many times an edge was seen during database read can be calculated from the edge list, if the count is greater than the user-defined support, then it is added to the frequent edge list. In line 2 (Figure 10) we iterate over all edges in c_k and we try to extend g_k

Algorithm 1: Basic MULE algorithm

procedure MinePathways(MFS, g_k, c_k, d)
▷MFS: Set of maximal frequent subgraphs
▷g_k: Frequent subgraph with k edges
▷c_k: Set of candidate edges
▷d: Set of already visited edges
1. $ismaximal \leftarrow$ **ture**
2. **for** all edges $e_i \in c_k$ **do**
3. $d \leftarrow d \cup \{e_i\}$
4. $g_{k+1} \leftarrow g_k \cup \{e_i\}$
5. **if** g_{k+1} is frequent **then**
6. $ismaximal \leftarrow$ **false**
7. $g_{k+1} \leftarrow (c_k \cup N(e_i)) \setminus D$
8. MinePathways(MFS, g_{k+1}, c_{k+1}, d)
9. **endif**
10. **endfor**
11. **if** $ismaximal$ **then**
12. **if** g_k has no superset in MFS **then**
13. $MFS \leftarrow MFS \cup g_k$
14. **endif**
15. **endif**

Figure 10. Depth-first enumeration algorithm for MULE.

with each edge. Here, g_k denotes the current pattern (of size k) which we are trying to extend with each edges e_j in c_k to make the new subgraph g_{k+1}. $N(e_i)$ in line 7, holds the neighboring edges of e_i. Candidate set, c_k, represents neighboring edges of subgraph g_k, meaning that edges in c_k also shares at least a common vertex with the edges in g_k. Since e_j is already in pattern g_{k+1}, there has to be a way of tracking already visited edges, so that e_j is not encountered again in later iteration.

The discovered set d keeps track of the edges that have already been visited in edgeset extension. Initially the maximal set MFS is empty. The algorithm keeps traversing the set enumeration tree, when it finds a maximal pattern at the end of the branch then it will be added to the maximal set, if it is not subsumed by another already found maximal pattern. It is a recursive approach, in each iteration it tries to

extend the subgraph by connecting an edge from the candidate edgeset. If the newly grown subgraph is frequent then the process continues, and the candidate list of the new pattern is updated. If it is not possible to extend the pattern then it checks if the pattern is subsumed by any maximal pattern previously mined. If not then the subgraph is included in the maximal set *MFS*.

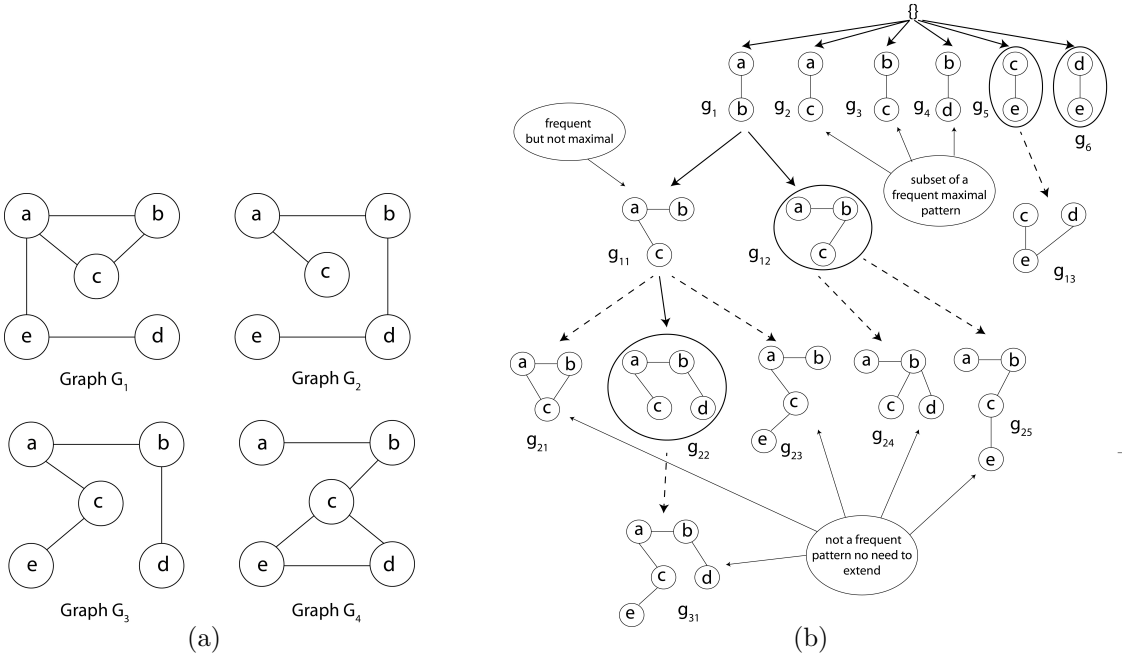


Figure 11. A sample execution of maximal frequent subgraph mining. (a) shows the input collections of unique labeled graphs, (b) shows the resulting enumeration tree with frequent subgraphs with support 2. The subgraphs in ovals are the maximal pattern found from the execution. Patterns that are not frequent are pointed by dotted lines.

Figure 11(a) shows an example database, Figure 11(b) show the pattern enumeration tree after running MULE. Here, initially the algorithm starts with an empty set, then it takes each frequent edges and tries to grow with its neighboring candidate edges. Here, in the example it starts with the edge (a, b). Its occurrence list is $\{1, 2, 3, 4\}$, we can see that it is present in all input graphs and so it is frequent. Now the frequent candidate edges of (a, b), are $\{(a, c), (b, c)\}$. In next step, (a, b) is ex-

tended with (a, c) to create a new subgraph $g_{11} = \{V = (a, b, c), E = \{(a, b), (a, c)\}\}$, candidate edges for the subgraph will be updated to $\{(b, c), (b, d), (c, e)\}$.

At this point, since $\{(a, b), (a, c)\}$, both the edges have been visited, they will be added to the discovered edgeset,d. Now it checks whether the new pattern g_{11} is frequent or not. Since it is frequent in $\{G_1, G_2, G_3\}$, then (a, b) is not maximal pattern. Next it extends g_{11} and keeps growing with its candidate edges $\{(b, c), (b, d), (c, e)\}$. This time it combines with (b, c) and creates the new pattern $g_{21} = \{V = (a, b, c), E = \{(a, b), (a, c), (b, c)\}\}$. The subgraph g_{21} can not be grown further since it is not frequent.

Now MULE backtracks to the previously grown pattern g_{11} and tries with its next candidate edge (b, d). The subgraph $g_{22} = \{V = (a, b, c, d), E = \{(a, b), (a, c), (b, d)\}\}$ is frequent so the discovered and candidate edges are updated. Now, it grows to form the pattern g_{31} [Figure 11(b)] and figures out that its is not frequent. So MULE gets back to the previous pattern g_{22} and marks it as the maximal pattern, since there are no frequent candidate edges to combine with as well as no frequent subgraphs already found that subsumes the subgraph g_{22}.

Again for the subgraph $g_{12} = \{V = (a, b, c), E = \{(a, b), (b, c)\}\}$ both of its extended subgraphs (g_{24}, g_{25}) are infrequent. So upon return, the subgraph is checked with previously found maximal pattern, as $g_{12} \not\subseteq g_{22}$, it is also marked as a maximal pattern. From the Figure 11(b), we can see that the frequent edges $\{(a, c), (b, c), (b, d)\}$ has not grown with its candidate edges as they are subset of the existing maximal subgraphs g_{12} and g_{22}. (c, e) grows with its only candidate edge (d, e), as the subgraph $g_{13} = \{V = (c, d, e), E = \{(c, e), (d, e)\}\}$ is not frequent it backtracks to (c, e) and marks as a maximal. (d, e) has no candidate edges to grow with, so it is marked as maximal patterns as well, as it is not found in other maximal patterns.

4.1. Mining Coherent Frequent Subgraphs

MULE does not consider edge weights. It works on binary database, an edge's presence is shown with a "1" and absence with a "0". For example, the database in Figure 12 has more information regrading the edge $(1, 2)$, its weight is 2 in graphs $\{G_0, G_2, G_4\}$, only 1 in $\{G_1\}$ and was not present in $\{G_3, G_5\}$. Let this example database represents authors and number of paper published in different conferences, then the edge represents that authors 1 and 2 published two papers in conference G_0. Similar to this sample database, there are more datasets that might carry valuable edge attributes, considering this data may find more interesting results.

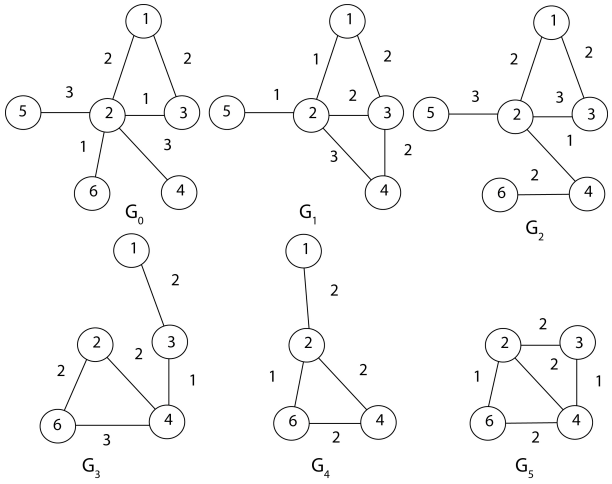


Figure 12. A sample graph database.

In this paper, we developed an algorithm for discovering maximal-cohesive frequent patterns. The modified MULE algorithm (Figure 13) can work on a graph database $\mathcal{G}$, similar to shown in Figure 12. Here, each graph G in the graph database has edges, e_i in the database with profile information $\{w_1(e_i), w_2(e_i), \ldots, w_n(e_i)\}$. Here, $w_1(e_i)$ represents weight of e_i in graph G_1. It takes two more parameters as input, frequency (σ^*) and cohesive (δ^*) threshold.

The first step of the algorithm is to iterate over each edge e_i and find out if it is frequent ($e_i.supp \geq \sigma^*$) and update max ($e_i.max$) and min ($e_i.min$) attributes. Initially for all frequent subgraphs with only one edge,

$$e_i.max = e_i.min = \{w_1(e_i), w_2(e_i), \ldots, w_n(e_i)\}.$$

At the end of the iteration one, it will produce a summery graph like in Figure 15(c). At this point, all the frequent edges have been discovered in the list of frequent edges $\mathcal{F}$. From the frequent edge list $\mathcal{F}$, the total occurrences of an edge can easily be found. Now, the algorithm iterates over each edges $e_i \in \mathcal{F}$, and updates its candidate edgeset $N(e_i)$. Each $e_j \in N(e_i)$ has to ensure that $e_j \cap e_i \neq \emptyset$, i.e., e_i and e_j shares one vertex within themselves. Once the neighboring information of the frequent edges has been updated, it starts extending each edge in $\mathcal{F}$ by calling the pattern extension algorithm *genMCFS* in Figures 13. The *genMCFS* algorithm takes two more sets for extending the pattern $\mathcal{P}_k$, the candidate set $\mathcal{C}_k$ and the set of already discovered edgeset $\mathcal{D}_k$.

The pattern extension algorithm ($genMCFS$) recursively calls itself if an existing pattern $\mathcal{P}_k$ can be extended with one of its candidate edge $c \in \mathcal{C}_k$ a new pattern $\mathcal{P}_{k+1}$ is generated by extending $\mathcal{P}_k$ with an edge. Max and Min attribute for the new pattern will be updated as,

$$\cup_{j=0}^{n}(\mathcal{P}_{k+1}[j].max) = max(\mathcal{P}_k[j].max, w_j(c)) \text{ and}$$
$$\cup_{j=0}^{n}(\mathcal{P}_{k+1}[j].min) = min(\mathcal{P}_k[j].min, w_j(c))$$

Now, here at any point $j = x$, when the new pattern has the difference between the max and the min attribute greater than the cohesive constraint, we set a flag and do not include it during frequency count. Frequency check is similar to the basic MULE algorithm, simply count the total occurrence, but in this case we ignore it if it is an ignore flag from frequency count. If the count is greater than the frequency threshold then it is frequent. If it is so then the parent edgeset $\mathcal{P}_k$ is set as not

maximal since $\mathcal{P}_k \subseteq \mathcal{P}_{k+1}$. This time the candidate edge c is added to the set of already visited edgeset $\mathcal{D}_k$, since it is already in the newly grown pattern $\mathcal{P}_{k+1}$. We update the new candidate edgeset $\mathcal{C}_{k+1}$ by removing c and merging it with $\mathcal{C}_k$ and neighboring edgeset of c, from *genMCFS* algorithm in Figure 13, line 8 shows how we update $\mathcal{C}_{k+1}$. The function again calls itself (line 9 from *genMCFS* in Figure 13) with the parameters $\mathcal{P}_{k+1}$, $\mathcal{C}_{k+1}$ and $\mathcal{D}_k$. It stops at a point when there is no more candidate in $\mathcal{C}_k$, means there is nothing more left for this current pattern to grow with and if this is frequent and cohesive it is added to the maximal-cohesive edgeset ($\mathcal{MCFS}$).

Figure 14 shows an illustrating example of our proposed method for finding cohesive patterns. Initially it finds the frequent edgesets $\{g_0, g_1, g_2, g_3\}$; and this time their attribute matrix is similar as in the input database. Now g_0 and g_1, i.e. edge $(2, 5)$ and $(2, 4)$ tries to extend since they have a common node "2". While it tries to extend, it updates the min and max attribute from their participating edges. Here, in the Figure 14, attribute M_{01} gets updated from the attribute set $\{M_0, M_1\}$. For example the pattern g_0 has $min = 1$ in $M_0[G_1, min]$; and g_1 has $min = 3$ in $M_1[G_1, min]$. So now the min attribute for the pattern g_{01} will be updated by the following formula;

$$M_{01}[G_x, min] = min(M_0[G_x, min], M_1[G_x, min])$$

Here, $min(M_0[G_1, min], M_1[G_1, min]) = min(1, 3) = 1$, so $M_01[G_1, min] = 1$. Similar way it updates the max values of M_{01}, by the following formula;

$$M_{01}[G_x, max] = min(M_0[G_x, max], M_1[G_x, max])$$

Once the attribute M_{01} is updated, it checks the cohesiveness, since the difference between $M_{01}[G_1, min]$ and $M_{01}[G_1, max]$ is greater than the cohesive threshold, it is replaced by the ignore condition(-1). Similar way $M_{01}[G_2, min]$ and $M_{01}[G_2, max]$ is replaced by the ignore condition. So now we calculate the support of the new

pattern from the updated matrix; the pattern has only true condition out of three. Hence the pattern g_{01} is frequent but not cohesive. So g_{01} is pruned, in the next iteration this pattern will not be considered for more extension.

Input:
▷ $\mathcal{G}$:unique labeled Graph database $\{G_1, G_2, \ldots, G_n\}$
▷ σ^*:frequency threshold
▷ δ^*:cohesive threshold
Output:
▷ $\mathcal{MCFS}$:Maximal Cohesive Frequent Subgraphs

$\varepsilon \leftarrow \{e = (u, v) : \exists G \in \mathcal{G}, (u, v) \in E(G)\}$ ▷Get all identical edge information
$MCFS = \emptyset$ ▷initialize Set of MCFS
foreach $e = (u, v) \in \varepsilon$ **do**
  $e.supp \leftarrow$ update frequency of $e \in \mathcal{G}$
  if $e.supp \geq \sigma^*$**then** ▷check if e is frequent
    $e.max \leftarrow \{w_1(e), w_2(e), \ldots, w_n(e)\}$
    $e.min \leftarrow \{w_1(e), w_2(e), \ldots, w_n(e)\}$
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{e\}$ ▷add into frequent edgeset
  endif
endfor
▷Depth-first traversal of the tree starting from each edge in level 1
foreach $e_i \in \mathcal{F}$ **do**
  $N(e_i) \leftarrow \{e_j \in \mathcal{F} : e_j \cap e_i \neq \emptyset\}$ ▷Neighboring edgeset of $\{e_i\}$
  genMCFS($\{e_i\}, N(e_i), \{e_1, e_2, \ldots, e_{i-1}\}$)
endfor
return $MCFS$

genMCFS($\mathcal{P}_k, \mathcal{C}_k, \mathcal{D}_k$)
▷**Input** $\mathcal{P}_k$: Frequent edgeset with k edges
▷**Input** $\mathcal{C}_k$: Set of candidate edges for edgeset extension
▷**Input** $\mathcal{D}_k$: Set of already visited edges

$isMaximal \leftarrow$ true
foreach $c \in \mathcal{C}_k$ do
  $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{c\}$ ▷Update the discover set
  $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k \cup \{c\}$ ▷New pattern in level $k + 1$
  update $\mathcal{P}_{k+1}.max$ and $\mathcal{P}_{k+1}.min$
  if $\mathcal{P}_{k+1}$ is frequent and cohesive **then**
    $ismaximal \leftarrow$ **false**
    $\mathcal{C}_{k+1} \leftarrow (\mathcal{C}_k \cup N(c)) \setminus \mathcal{D}_k$
    genMCFS($\mathcal{P}_{k+1}, \mathcal{C}_{k+1}, \mathcal{D}_k$)
  endif
endfor
if $isMaximal$ and $\mathcal{P}_k$ is cohesive **then**
  if $\nexists P' \in MCFS$ s.t $\mathcal{P}_k \subseteq P'$ **then**
    $MCFS \leftarrow MCFS \cup \mathcal{P}_k$
  endif
endif

Figure 13. Mining Maximal Cohesive Subgraphs.

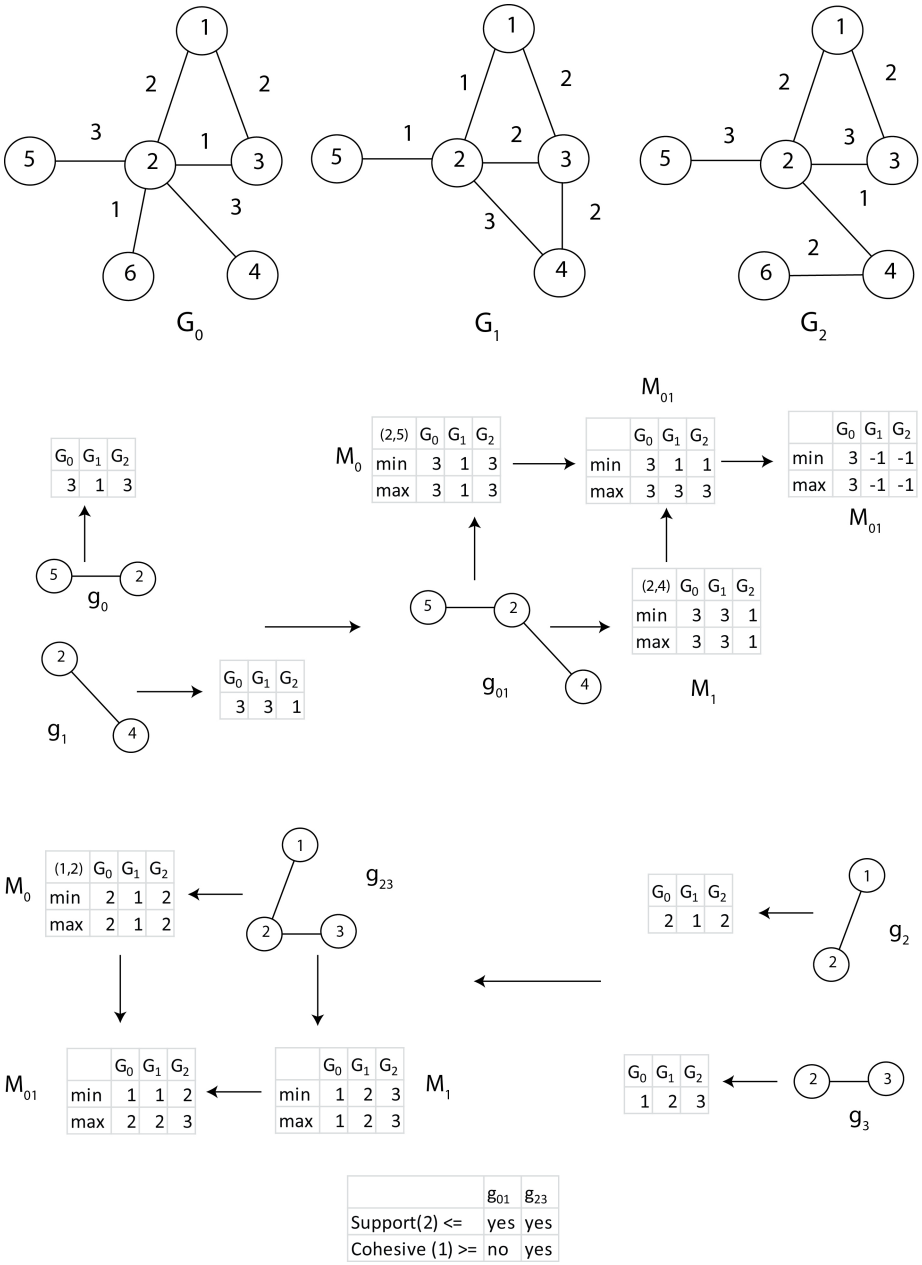
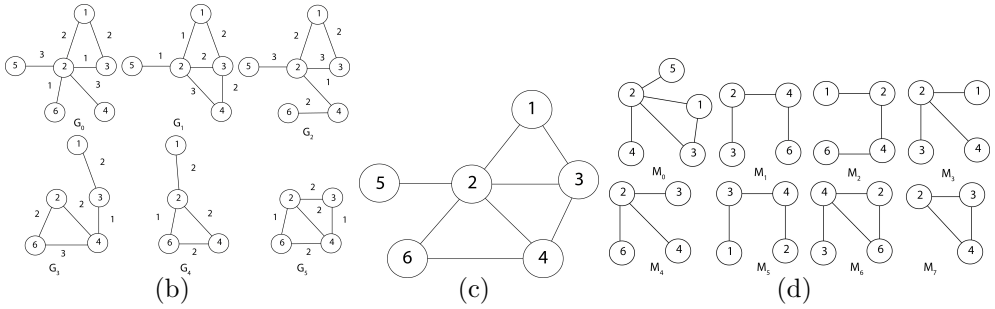


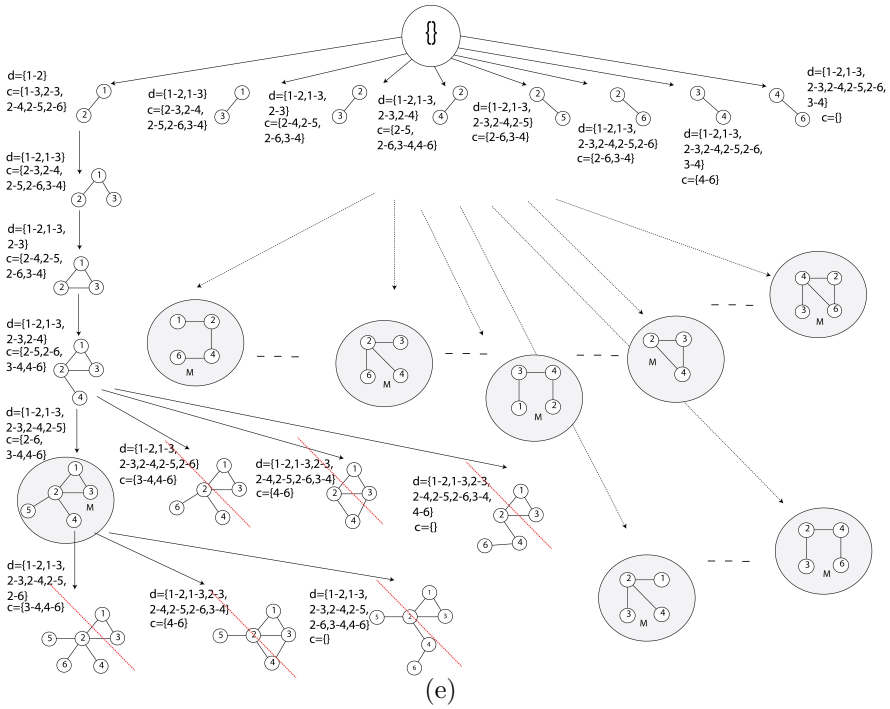
Figure 14. The graph database G_0, G_1, G_2. It is a part of the graph database shown in Figure 15(b). $\{g_0, g_1, g_2, g_3\}$ are the frequent edges for support threshold 2. For cohesive threshold 1, $\{g_{01}\}$ gets pruned while $\{g_{23}\}$ is frequent and cohesive.

Edge	Profile Info					
	G_0	G_1	G_2	G_3	G_4	G_5
1-2	2	1	2	0	2	0
1-3	2	2	2	2	0	0
2-3	1	2	3	0	0	2
2-4	3	3	1	2	2	2
2-5	3	1	3	0	0	0
2-6	1	0	0	2	1	1
3-4	0	2	0	1	0	1
4-6	0	0	2	3	2	2

(a)



(b) (c) (d)



(e)

Figure 15. The sample graph database. In (a) shows another sample graph database, (b) shows the illustrating network. (c) Resulting summary graph found at the end of level one. (d) Set of maximal frequent pattern, $\mathcal{F}$ found from the database in Figure (b) (min support, σ=3). (e) Resulting set enumeration tree of frequent edges shown in (c).

5. EXPERIMENTS

In this project modified MULE algorithm was implemented in python. We have used *Digital Bibliography & Library Project* (DBLP) for extensive testing used. The DBLP is a bibliography data in XML format which has all the computer science publication information since 1980s [21]. It was reported in 2013 that DBLP has more than 2.3 millions of articles.

5.1. Dataset

We designed and wrote a small script to extract the top 50 conference names from the DBLP dataset that has the largest number of articles. Some of the important conference names are KDD, ICDM, CIKM, SDM, SIGCOMM, SIGMETRICS, INFOCOM, MOBICOM and etc.

Figure 16 shows example of XML entry of the DBLP dataset. We can see the first journal name in "Journal" tag called "Further Normalization of the Data Base Relational Model" was published in 1971, the name of the author can be found in "author" tag; which is "E. F. Codd". There are more XML tags related to each entry, such as month, volume number, cdrom, ee, metadata for the article and etc.

From the DBLP dataset, if any paper was published in these conferences within years 2000 - 2013, then an edge is created between each pair of authors of the paper and added to the author mapping list. The conference attribute also records the published paper by updating the associated row (author mapping edge) and column (conference mapping graph).

The edge attribute X captures the occurrences of the co-author relationship in conference, so the number X_{ij} in conference graph, determines authors in the mapping edge (row i from author mapping list) has published X numbers of papers in the j^{th} conference. If X_{ij} is zero then there was no paper published in the conference by the authors. Year graph dataset was prepared in the same way, here in the dataset each

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>
<article mdate="2002-01-03" key="persons/CoddD74">
<author>E. F. Codd</author>
<author>C. J. Date</author>
<title>Interactive Support for Non-Programmers: The Relational and Network Approaches.</title>
<journal>IBM Research Report, San Jose, California</journal>
<volume>RJ1400</volume>
<month>June</month>
<year>1974</year>
</article>

<article mdate="2011-12-29" key="tr/trier/MI94-13" publtype="informal publication">
<author>Reiner Horst</author>
<author>Nguyen V. Thoai</author>
<title>An Integer Concave Minimization Approach for the Minimum Concave Cost Capacitated Flow Problem on Networks</title>
<journal>Universityaumlt Trier, Mathematik/Informatik, Forschungsbericht</journal>
<volume>94-13</volume>
<year>1994</year>
</article>

Figure 16. Example of XML entry of the DBLP data.

column represents a year. So if two authors have published a paper in year 2000 then there will a "1" in the column that represents year "2000".

Figure 17 shows the number of papers for the top 50 conferences. We found that a total of 192,650 papers were published in these conferences within the last fourteen years (2000-2013); and the conference called "IEEE GLOBECOM" has the highest number of publications (8228). Figure 18 shows a similar histogram related to the DBLP dataset. We represent the year information against the total number of papers published in these years. We can see that, gradually, the number of papers has increased from year 2000 to year 2013. We have not considered the current year since the DBLP dataset is incomplete for the current year.

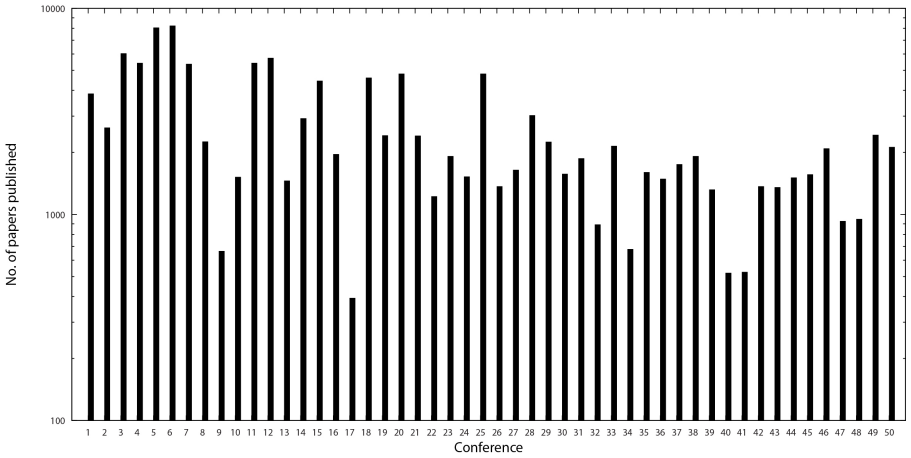


Figure 17. Paper published in each conference over the years from 2000 to 2013.

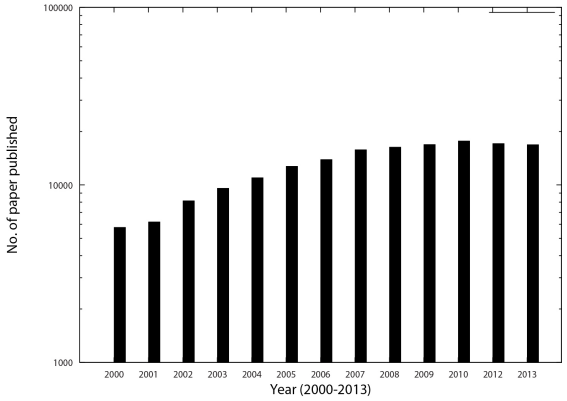


Figure 18. Paper published in conferences - each year from 2000 to 2013.

The histogram in Figure 19 shows another characteristics of the DBLP dataset. Here on the X-axis shows number of papers published and the Y-axis, shows how many authors published this many papers. An interesting observation is, there are few authors that have published many papers; i.e. as the number of papers published increases, the author count decreases.

5.2. Results

We ran the proposed algorithm for detecting maximal cohesive patterns with varying frequency constraint with constant cohesive threshold on two datasets. One with authorship network and conference attributes; and the other is authorship

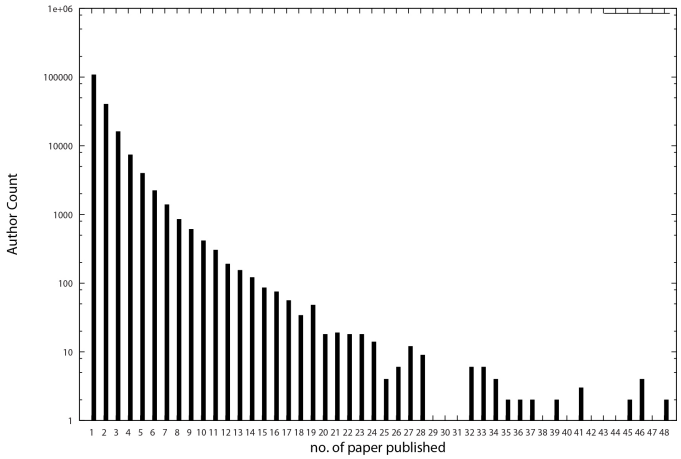


Figure 19. Number of papers distribution. Number of papers published and the number of authors contributed in paper publication.

network and year attributes. For author and conference graph, frequency threshold σ^* was varied from 4 to 13 in 1 increment with constant cohesiveness $\delta^* = 0$. Frequency threshold σ^*=4, means that we were looking for frequent subgraphs that is common at least in four conferences or years.

Table 1. Analysis of maximal patterns for varying support (i). To find the maximal cohesive patterns, the constant is the cohesiveness(δ^*=0). The author and conference graph database is used here for analysis which was found in the DBLP data.

Sup	$RunTime$	$\#FrequentEdge$	$\#Pattern$	$\#Pattern(\geq 2)$	$AVGSize(\geq 2)$	$Density$	$Sizeofthelargestpattern$
4	1881.85	4562	3982	432	2.45	0.61	15
5	11.94	1233	1136	169	2.64	0.60	10
6	2.05	305	290	10	2.70	0.58	6
7	1.73	103	100	2	2.50	0.58	3
8	1.68	34	32	2	2.00	0.67	2
9	1.61	13	13	0	0	1.00	1
10	1.68	3	3	0	0	1.00	1

Table 1 shows the experimental results on the DBLP data for author-conference graph database. For the 50 conference data, when the frequency threshold is 4 (i.e. frequent in 4 out of 50 conferences), the algorithm takes around 1881.85 sec to finish. The algorithm reports total of 4562 frequent edges at level one. Figure 20(a) shows the number of maximal cohesive patterns found for varying support. Here, we can see that, as we increase the support threshold, the number of maximal cohesive patterns

decreases. Also in Figure 20(b), we can see that as we increase the support threshold, the number of frequent edges found at level one also decreases. The number of maximal cohesive patterns, found with support threshold $\sigma^* = 4$ was 3982, and the largest pattern among these patterns has 15 edges, most of the patterns has 2 or 3 edges. The average size of the patterns is 2.45, which has overall density of 0.61. As the frequency threshold is increased 7, 8, 9 to 10 the execution time reduces to 1.73, 1.68, 1.61 and 1.68 seconds. Figure 20(c) shows how the change in support reduces the runtime. For support threshold 11, 12 and 13 the run times are almost same; mostly the time needed for reading level one.

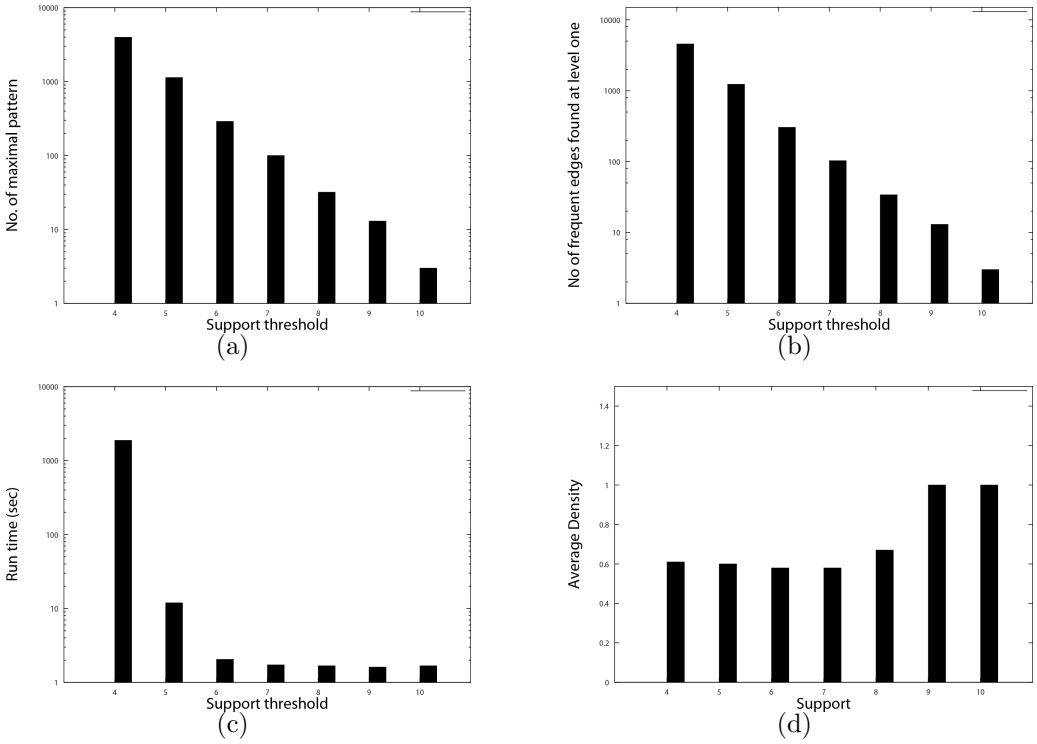
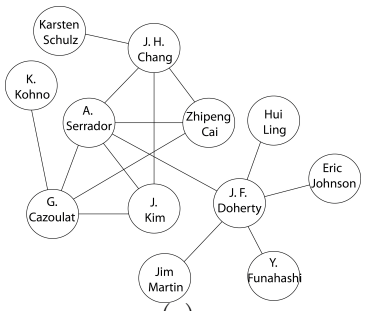


Figure 20. Author and Conference graph database analysis. The graph network was extracted from DBLP dataset, this time only constant parameter is $\delta^* = 0$. (a) Shows varying support vs number of maximal cohesive patterns found. (b) Varying support vs frequent edges found at level one. (c) Varying support vs runtime in seconds and (d) Shows varying support vs average density of the cohesive patterns.

Figure 20(d) shows the average density of the maximal patterns for varying support thresholds. Here, we can see that after support threshold is increased from 9 to 13 the density is "1" for all, this is because all the maximal cohesive pattern found this time have a single edge only.



(a)

Conference
icc
iros
vtc
pimcr

(b)

Figure 21. Largest pattern found in DBLP dataset. (a) shows the co-author relationship network, (b) shows the 4 conferences that all the edges from the pattern must have appeared.

Figure 21 shows an example of maximal cohesive pattern found in the DBLP dataset. We used conference graph with author information, this time the setting was support $\sigma^* = 4$ and cohesive $\delta^* = 0$. The pattern has 15 edges and 12 authors. We can see from the figure that there are nodes that are connected to more than one node. Since support threshold was four, we can say that this pattern was present at least in four conferences; i.e. these co-author relationship have appeared together in at least 4 conferences.

Table 2 shows experimental results for author-year graph, here we can see that for support threshold 4, the algorithm finishes in 55.31 sec. We found total 4153 frequent edges at level one, which is almost same as the number of frequent edges found at level one from the author-conference graph database (Table 1). Figure 22(a) shows the number of maximal cohesive patterns found in author-year graph database; and 22(b) shows the total frequent edges found at level one with varying

Table 2. Analysis of maximal patterns for varying support (ii). This time varying support σ^* is used with constant cohesiveness(δ^*=0) on DBLP dataset with author and year informations only.

Sup	*RunTime*	*#FrequentEdge*	*#Pattern*	*#Pattern*(≥ 2)	*AVGSize*(≥ 2)	*Density*	*Sizeofthelargestpattern*
4	55.31	4153	4136	21	2.00	0.67	2
5	11.25	1837	1835	2	2.00	0.67	2
6	3.03	887	887	0	0	1.00	1
7	1.41	460	460	0	0	1.00	1
8	0.85	248	248	0	0	1.00	1
9	0.70	123	123	0	0	1.00	1
10	0.66	69	69	0	0	1.00	1
11	0.65	30	30	0	0	1.00	1
12	1.29	13	13	0	0	1.00	1
13	0.67	7	7	0	0	1.00	1

support from author-year graph database. In both experiments we see that number of patterns discovered constantly decreases when we increase the support threshold.

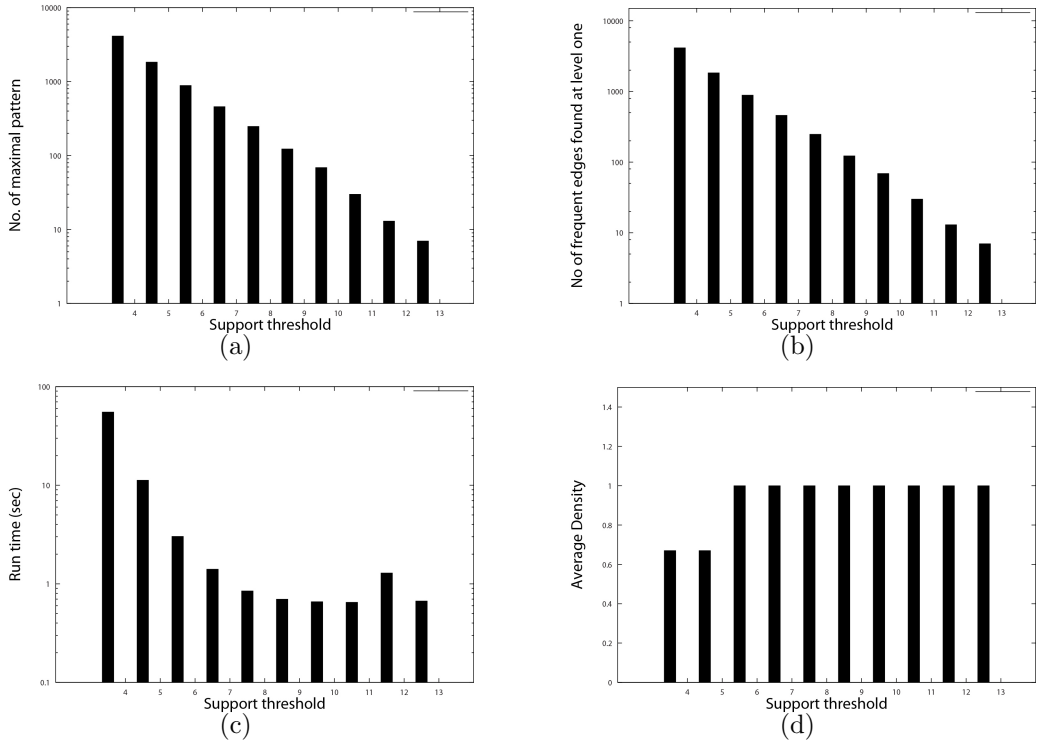


Figure 22. Author and year graph database analysis. (a) Varying support vs number of maximal patterns found from DBLP dataset. (b) Varying support vs frequent edges at level one. (c) Varying support vs runtime in seconds and (d) Varying support vs average density.

In Table 2, we can see that the number of maximal cohesive pattern is 4136 when support threshold is 4, and only 21 patterns have size of 2, i.e. these patterns have 2 edges as the size of the largest pattern reported was only 2-edges. Comparing to author-conference graph database, we found 432 maximal cohesive patterns that have at least 2 edges, more over the largest pattern has 15 edges 21. From the results we can see that, as we increase support threshold, the runtime also decreases; Figure 22(c) shows the runtime for different support thresholds used in the experiments.

We found some interesting measures from the experiments which is shown in Table 1 and Table 2. After support threshold 8, the maximal cohesive patterns found in author-conference graph database have single edge only; and for the author-year graph database the support threshold was 6 when we started seeing maximal cohesive patterns with single edges, some maximal patterns with single edge only found are "Fred S. Richardson and Yue-Jin Lv", "Michael G. Huchyj and David J. Dewit" and "Xianfeng Jiao and Sungchul Kim". In both experiments, when we started seeing single edged maximal cohesive patterns, their average size ($Averagesize = 1$) and density ($density = 1$) became constant. Figure 22(d) and 22(d) shows the changes in density when we increase the threshold for support.

Table 3. Analysis of maximal patterns for varying cohesive threshold. DBLP dataset with co-author relationship and conference attributes δ^* is used with constant support (σ^*=5).

Sup	$Cohesive$	$RunTime$	$\#FrequentEdge$	$\#Pattern$	$\#Pattern(\geq 2)$	$AVGSize(\geq 2)$	$Density$	$Sizeofthelargestpattern$
5	0	11.94	1233	1136	169	2.64	0.60	10
5	1	1284.70	1233	1041	162	2.60	0.59	10

Table 3 shows another analysis of DBLP data, this time we used varying cohesive threshold ("0" and "1") against constant support σ^*=5. We found that number of frequent edges at level one is exactly same in both cases. Number of maximal patterns reduced to 1041. But running time was around 30 minutes when we increased the cohesive threshold. So checking the cohesive constraint was expensive for increasing

cohesive threshold. Other measures such as average size, density, number of patterns of size greater than two and size of the largest pattern was almost same. We run our experiment with more increasing cohesive threshold but it didn't output any significant results as the average co-authorship count is around 1.5.

6. CONCLUSION

In this paper, we modified and extended an existing graph mining algorithm, called MULE. We introduce edge weights to the algorithm, our algorithm works on undirected graph networks with edge weights; what we call the edge attributes. We have seen that introducing edge attributes can find more interesting measures from a graph network which will be useful for graph pattern mining, pattern discovery and lot more graph mining applications. We have worked on DBLP dataset, we extracted best 50 conference names and co-author relationships for last 13 years(200-2013). We found some interesting characteristics of DBLP data after running our modified algorithm with varying support and cohesive threshold.

This paper only scopes to maximal cohesive patterns. Our algorithm can work fine to discover frequent closed patterns after slight modification to overcome lossy compression of graph network data. Our approach works fine up to certain support threshold. Here, we were constraint by the run time. We tried running our algorithm for low support, σ^*=3; and it was running for two consecutive days and did not finished. So, there is a need for efficient algorithms to mine frequent patterns. Parallel algorithms is one solution, specially multi-threaded algorithms. This is important since the modern machines have multiple core and we can run the parallel algorithm on personal computers.

7. BIBLIOGRAPHY

[1] Ramesh C Agarwal, Charu C Aggarwal, and VVV Prasad, *A tree projection algorithm for generation of frequent item sets*, Journal of parallel and Distributed Computing **61** (2001), no. 3, 350–371.

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

[3] Rakesh Agrawal and Ramakrishnan Srikant, *Mining sequential patterns*, Data Engineering, 1995. Proceedings of the Eleventh International Conference on, IEEE, 1995, pp. 3–14.

[4] Rakesh Agrawal, Ramakrishnan Srikant, et al., *Fast algorithms for mining association rules*, Proc. 20th Int. Conf. Very Large Data Bases, VLDB, vol. 1215, 1994, pp. 487–499.

[5] Mohammad Al Hasan and Mohammed Javeed Zaki, *Musk: Uniform sampling of k maximal patterns.*, SDM, 2009, pp. 650–661.

[6] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman, *Gapped blast and psi-blast: a new generation of protein database search programs*, Nucleic acids research **25** (1997), no. 17, 3389–3402.

[7] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan, *Group formation in large social networks: membership, growth, and evolution*, Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2006, pp. 44–54.

[8] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl, *Mining coherent subgraphs in multi-layer graphs with edge labels*, Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '12, 2012, pp. 1258–1266.

[9] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke, *Mafia: A maximal frequent itemset algorithm for transactional databases*, Data Engineering, 2001. Proceedings. 17th International Conference on, IEEE, 2001, pp. 443–452.

[10] Karam Gouda and Mohammed J. Zaki, *Genmax: An efficient algorithm for mining maximal frequent itemsets*, Data Mining and Knowledge Discovery: An International Journal **11** (2005), no. 3, 223–242.

[11] Karam Gouda and Mohammed Javeed Zaki, *Efficiently mining maximal frequent itemsets*, Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, IEEE, 2001, pp. 163–170.

[12] Jiawei Han, Micheline Kamber, and Jian Pei, *Data mining: concepts and techniques*, Morgan kaufmann, 2006.

[13] Jiawei Han, Jian Pei, and Yiwen Yin, *Mining frequent patterns without candidate generation*, ACM SIGMOD Record, vol. 29, ACM, 2000, pp. 1–12.

[14] Leland H Hartwell, John J Hopfield, Stanislas Leibler, and Andrew W Murray, *From molecular to modular cell biology*, Nature **402** (1999), C47–C52.

[15] Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou, *Mining coherent dense subgraphs across massive biological networks for functional discovery*, Bioinformatics **21** (2005), no. suppl 1, i213–i221.

[16] Jun Huan, Wei Wang, Jan Prins, and Jiong Yang, *Spin: mining maximal frequent subgraphs from graph databases*, Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2004, pp. 581–586.

[17] M. Koyutürk, Y. Kim, S. Subramaniam, W. Szpankowski, and A. Grama, *Detecting conserved interaction patterns in biological networks*, Journal of Computational Biology **13** (2006), no. 7, 1299–1322.

[18] Mehmet Koyutürk, Ananth Grama, and Wojciech Szpankowski, *An efficient algorithm for detecting frequent subgraphs in biological networks*, Bioinformatics **20** (2004), no. suppl 1, i200–i207.

[19] Mehmet Koyuturk, Yohan Kim, Shankar Subramaniam, Wojciech Szpankowski, and Ananth Grama, *Detecting conserved interaction patterns in biological networks*, Journal of Computational Biology **13** (2006), no. 7, 1299–1322.

[20] M. Kuramochi and G. Karypis, *Frequent subgraph discovery*, Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, IEEE, 2001, pp. 313–320.

[21] Michael Ley, Marc Herbstritt, Marcel R. Ackermann, Oliver Hoffmann, Michael Wagner, Stefanie von Keutz, and Katharina Hostert, *Dblp bibliography*, May 2013.

[22] 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** (2010), no. Suppl 1, S3.

[23] Mark EJ Newman and Michelle Girvan, *Finding and evaluating community structure in networks*, Physical review E **69** (2004), no. 2, 026113.

[24] Guo-Jun Qi, Charu C Aggarwal, and Thomas Huang, *Community detection with edge content in social media networks*, Data Engineering (ICDE), 2012 IEEE 28th International Conference on, IEEE, 2012, pp. 534–545.

[25] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy, *Efficient community detection in large networks using content and links*, Proceedings of the 22nd international conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2013, pp. 1089–1098.

[26] Ron Rymon, *Search through systematic set enumeration*, KR, 1992, pp. 539–550.

[27] Saeed Salem, Rami Alroobi, Syed Ahmed, and Mohammad Hossain, *Discovering maximal cohesive subgraphs and patterns from attributed biological networks*, Proceedings of the 5^{th} International Workshop on Bio-molecular Network Analysis (Philadelphia, PA, USA), IWBNA'12 (in conjunction with IEEE-BIBM '12), October 2012.

[28] Jiliang Tang, Xufei Wang, and Huan Liu, *Integrating social media data for community detection*, Modeling and Mining Ubiquitous Social Media, Springer, 2012, pp. 1–20.

[29] Andrzej Trybulec, *Enumerated sets*, Journal of Formalized Mathematics **1** (1989).

[30] Julian R Ullmann, *An algorithm for subgraph isomorphism*, Journal of the ACM (JACM) **23** (1976), no. 1, 31–42.

[31] Xifeng Yan and Jiawei Han, *gspan: Graph-based substructure pattern mining*, Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on, IEEE, 2002, pp. 721–724.

[32] M Zaki and Wagner Meira Jr, *Fundamentals of data mining algorithms*, 2012.

[33] Mohammed J Zaki and Karam Gouda, *Fast vertical mining using diffsets*, Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2003, pp. 326–335.

[34] Mohammed Javeed Zaki and Ching-Jiu Hsiao, *Charm: An efficient algorithm for closed itemset mining.*, SDM, vol. 2, 2002, pp. 457–473.