Expert Systems with Applications 41 (2014) 1847–1863

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Mining frequent correlated graphs with a new measure

Md. Samiullah [a], Chowdhury Farhan Ahmed [b,*], Anna Fariha [a], Md. Rafiqul Islam [c], Nicolas Lachiche [b]

[a] Department of Computer Science and Engineering, University of Dhaka, Bangladesh
[b] ICube Laboratory, University of Strasbourg, France
[c] School of Computing and Mathematics, Charles Sturt University, Australia

ARTICLE INFO

Keywords:
Data mining
Knowledge discovery
Correlated patterns
Graph mining

ABSTRACT

Correlation mining is recognized as one of the most important data mining tasks for its capability to identify underlying dependencies between objects. On the other hand, graph-based data mining techniques are increasingly applied to handle large datasets due to their capability of modeling various non-traditional domains representing real-life complex scenarios such as social/computer networks, map/spatial databases, chemical-informatics domain, bio-informatics, image processing and machine learning. To extract useful knowledge from large amount of spurious patterns, correlation measures are used. Nonetheless, existing graph based correlation mining approaches are unable to capture effective correlations in graph databases. Hence, we have concentrated on graph correlation mining and proposed a new graph correlation measure, *gConfidence*, to discover more useful graph patterns. Moreover, we have developed an efficient algorithm, *CGM* (*C*orrelated *G*raph *M*ining), to find the correlated graphs in graph databases. The performance of our scheme was extensively analyzed in several real-life and synthetic databases based on runtime and memory consumption, then compared with existing graph correlation mining algorithms, which proved that *CGM* is scalable with respect to required processing time and memory consumption and outperforms existing approaches by a factor of two in speed of mining correlations.

1. Introduction

Data mining extracts useful knowledge from databases. It also discovers patterns (Agrawal & Srikant, 1994; Ahmed, Tanbeer, Jeong, & Lee, 2009; Ahmed, Tanbeer, Jeong, & Choi, 2011; Ahmed, Tanbeer, Jeong, & Choi, 2012; Han, Pei, & Yin, 2000; Hu, Huang, & Kao, 2013; Inokuchi, Washio, & Motoda, 2000; Inokuchi, Washio, & Motoda, 2005; Kuramochi & Karypis, 2001; Nishi, Ahmed, Samiullah, & Jeong, 2013; Nori, Deypir, & Sadreddini, 2013; Tanbeer, Ahmed, Jeong, & Lee, 2009) hidden in data and useful correlations/affinities between the patterns. Correlation mining is a very interesting and important area of data mining which finds the underlying dependencies/affinities among the patterns/objects (Lee, Kim, Cai, & Han, 2003; Omiecinski, 2003; Tan, Kumar, & Srivastava, 2002; Xiong, Tan, & Kumar, 2003). The implicit information within databases, and mainly the interesting association relationships among sets of objects, those lead to association rules, may disclose useful patterns for decision support, financial forecast, marketing policies, even medical diagnosis and many other applications. Nowadays, as data mining techniques are being increasingly applied to non-traditional domains, existing approaches for finding frequent patterns cannot be used as they cannot model the requirements of these domains.

Graphs can be used as an alternate way of modeling the objects in datasets (Inokuchi et al., 2000; Kuramochi & Karypis, 2001; Lahiri & Berger-Wolf, 2008; Lahiri & Berger-Wolf, 2010; Yan & Han, 2003). Within that model, the problem of finding frequent patterns becomes that of discovering subgraphs that occur frequently over the entire set of graphs (Kuramochi & Karypis, 2001). In particular, each vertex of the graph will correspond to an entity and each edge will correspond to a relation between two entities. In this model, both vertices and edges may have labels associated with them which are not required to be unique. The graph structured data mining to derive frequent subgraphs from a graph dataset is difficult because the search for subgraphs is combinatorially explosive and includes subgraph isomorphism matching (Kramer, Pfahringer, & Helma, 1997) which is an NP-complete problem. Power of using graphs, to model complex datasets, has been recognized by many researchers in chemical informatics (Chittimoori, Holder, & Cook, 1999; Dehaspe, Toivonen, & King, 1998; Srinivasan, King, Muggleton, & Sternberg, 1997a; Srinivasan, King, Muggleton, & Sternberg, 1997b), computer vision (Klviinen & Oja, 1990; Piriyakumar, Murthy, & Levi, 1998), image and object retrieval (Cicirello, 1999; Dupplaw & Lewis, 2000), and machine learning (Chen & Yun, 2003; Holder, Cook, & Djoko, 1994; Yoshida & Motoda, 1995) domain.

* Corresponding author. Mob.: +33 629 271 568.
E-mail addresses: samiullah@cse.univdhaka.edu (Md. Samiullah), cfahmed@unistra.fr, farhan@cse.univdhaka.edu (C.F. Ahmed), purpleblueanna@gmail.com (A. Fariha), mislam@csu.edu.au (Md. Rafiqul Islam), nicolas.lachiche@unistra.fr (N. Lachiche).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.08.082

Correlation mining in graph databases is a very important graph mining task due to its wide range of application domains. Existing works (He & Singh, 2006; Holder et al., 1994; Raymond, Gardiner, & Willett, 2002; Williams, Huan, & Wang, 2007; Yan, Zhu, Yu, & Han, 2006) mainly focus on structural similarity search, which aim to find graphs those are similar in structure. However, in many applications, two structurally similar graphs do not imply that they are correlated or similar in characteristics. For example, in chemistry, isomers refer to molecules with the same chemical formula and similar structures. The chemical properties of isomers can be quite different due to different positions of atoms and functional groups. Consider the case that a chemist needs to find some molecules those share similar chemical properties with a given molecule. Structural similarity search is not relevant, since it mostly returns isomers of the given molecule that have similar structures but different chemical properties, which is undesirable.

In Ke, Cheng, and Ng (2008), the authors proposed an algorithm of mining graph correlation based on statistical similarity, that is, *CGS* (Correlated Graph Search) algorithm, which is able to obtain the molecules that share similar chemical properties but may or may not have similar structures to the given molecule. In particular, *CGS* is a searching algorithm, which works for searching correlation of a specific query graph with the database. Therefore, it has limitations in describing inherent correlation within graphs of graph databases and the domain knowledge is mandatory in using *CGS*, otherwise lots of queries would be meaningless.

Consider a scenario shown in Fig. 1, where two frequent graphs are found from a set of graphs representing a group of people in a social network. Each graph in the set represents friend circle of an individual where nodes represent individuals and edges represent interaction among individual pairs. The circles around graphs represent the interaction of a group of people all together (sub-group). Moreover, nodes in the frequent graphs represent the most interactive individuals and edges are their mutual interaction in various friend circle representing graphs. Integer values beside the edges and circles represent the frequencies of the edges and frequent graphs within the set of graphs respectively. The frequency of edges indicates the number of graphs where such interactions occur in the graph database (network) and the frequency of circles represents the number of graphs where such sub-grouping of individuals with their interactions occurred.

In order to determine the most correlated group between the two groups in Fig. 1, so that we can perform various operations on the most correlated group (as example, target group for task assignment, social/cyber crime investigation, common notification sending etc.), frequencies of the frequent graphs cannot provide any hint due to the tie in frequency of both graphs.

In this circumstance, our measure, which is proposed in Section 3.1, will suggest that people of G_2 are more correlated. Because, G_1's people interact together in 30 events and the maximum number of interactions between any pair in G_1 is 100. Therefore, according to our approach, $Correlation(G_1) = \frac{30}{100} = 0.3$. The second group's people interact together in 30 events and the maximum number of interactions between any pair in G_2 is 60, hence $Correlation(G_2) = \frac{30}{60} = 0.5$. Indeed, such correlation measure helps in mining more useful/meaningful graph patterns and knowledge, since it can discover inherent correlation among the elements of a graph. As a consequence, if the graph database, from which the two frequent subgraphs of Fig. 1 were extracted, can be used for mining correlated graphs with the correlation threshold value 0.4, then the first frequent subgraph will be pruned and the second one will be selected as a strong correlated subgraph among the graphs of the database.

These facts motivated us in developing such a new measure and to the best of our knowledge, such correlation mining in graph databases has not been proposed yet. The key contributions of this paper are as follows:

1. A new measure, *gConfidence*, is proposed to capture more interesting inherent correlation in graph databases.
2. Our proposed measure satisfies the downward closure property, consequently, allows to prune a large number of candidate patterns.
3. We have proposed an algorithm, *CGM* (Correlated Graph Mining), which uses the proposed measure and efficiently mines correlation by constructing a hierarchical reduced search space in large graph databases.
4. Elaborate descriptions with examples of real-life applications are given to explain the realistic usefulness of our approach. Advantages of *CGM* over existing graph correlation mining algorithms as well as the relationship with them are discussed and comprehensively analyzed.
5. An extensive performance study was conducted to show the efficiency, scalability, correctness and effectiveness of our approach. Real-life and complex-large synthetic graph datasets were used to compare our method with existing approaches with respect to runtime and memory consumption.

Our proposed algorithm can be applied in various real-life domains where data can be represented by graphs such as chemical informatics domain, gene sequence databases, bio-informatics, image processing, machine learning, neural networks and lot more.

Rest of the paper is organized as follows: Section 2 contains Related Works, our proposed scheme is presented in Section 3, where Section 4 focuses on the performance analysis of our proposed algorithm. We have discussed the applicability of our scheme in real-life scenarios in Section 5 and finally, we conclude our work in Section 6.

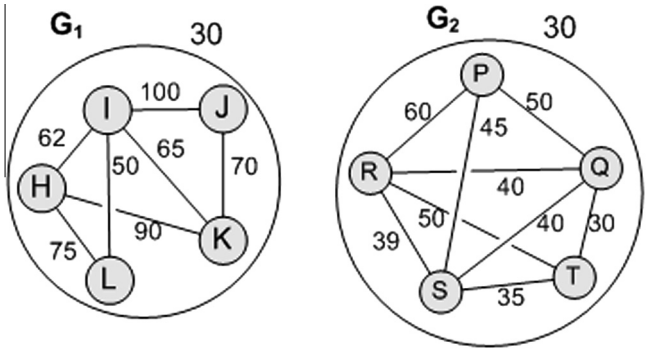


Fig. 1. An example scenario.

2. Related works

Data mining focuses on frequent data values in structured data, but in semi-structured and graph data, the emphasis is on frequent labels and common topologies. Difficulties arise in the discovery task from the complexity of some of the required sub-tasks, such as subgraph isomorphism. In any data mining algorithm which uses an Apriori-based approach, two issues arise: (1) the basic building block from which frequent patterns are composed; (2) making sure that at each step of the algorithm, all frequent patterns for that step are found (Inokuchi et al., 2000).

In graph mining domain, most of the graphs are considered labeled, that is, either the vertices or the edges or both contain a specific value. Transaction graphs can be represented by

$G = \{V(G), E(G), L(V(G)), L(E(G))\}$, that is, the set of vertices, the set of edges and set of labels for vertices and edges, respectively. Size of a graph can be important in several algorithms and varies in terms of the definition in many algorithms such as it can be number of nodes or edges or disjoint paths in a graph. For graph dataset, support and confidence can be stated as in Inokuchi et al. (2000), given a graph database, $\mathbb{GD}$, with G_s, G_b and G_h are subgraphs of any graph, $G \in \mathbb{GD}$, The metric support is:

$$Supp(G_s) = \frac{No.\ of\ graphs\ where\ G_s \subseteq G \in \mathbb{GD}}{Total\ No.\ of\ graph\ G \in \mathbb{GD}}$$

The metric confidence is:

$$Conf(G_b \Rightarrow G_h) = \frac{\{No.\ of\ graphs\ G | G_b \cup G_h \subseteq G \in \mathbb{GD}\}}{\{No.\ of\ graphs\ G_b \subseteq G \in \mathbb{GD}\}}$$

These imply that the support of G_s is the ratio between the frequency of G_s's super-graph and total number of transaction graphs. Moreover, confidence of G_b with respect to G_h is the ratio between the joint-occurrence frequency of G_b along with G_h and frequency of the super-graph for G_b.

To tackle graph isomorphism problem various labelings is introduced in many algorithms. In particular, the canonical labeling ensures the graph isomorphism solution. The canonical labeling for the normal form representation, X of a graph G can be defined as in Yan and Han (2002)

$$X_c = arg\ min_{X \in NF(G)}\ code(X)$$

That means, X_c is the minimum *DFS* (Depth First Search) *Code* among all possible *DFS Codes* of G. The *DFS Code* (Yan & Han, 2002) is a representation of any graph $G = (V,E)$, constructed from an edge sequence $e_i \in E(G)$ based on an ordering such that e_i comes before $e_{i+1} \in E(G)$ within the ordering, for $i = 0, \ldots, |E| - 1$. In such system, to define the ordering, an edge can be represented as a 5 *tuple*: (i, j, l_i, l_{ij}, l_j). To tackle the subgraph isomorphism problem while graph data mining, such representation and coding scheme are proved useful. The values of i and j correspond to the time stamp value of nodes in a graph while running a *DFS* traversal. Next three values represent the label of terminal nodes of an edge and the edge's label itself. This type of edge representation imposes an ordering on the edges for a graph. Moreover, the minimum among all possible edge orderings of the graph has a canonical aspect, that is, the canonical representation of such graph representation is unique and can solve the graph isomorphism problem in an effective way (Yan & Han, 2002).

The ordering among the edges can be defined as Yan and Han (2003), $e_1 < e_2$ **iff**

- $e_1, e_2 \in E_T^f$, and $j_1 < j_2$ or $i_1 > i_2 \wedge j_1 = j_2$
- $e_1, e_2 \in E_T^b$, and $i_1 < i_2$ or $i_1 = i_2 \wedge j_1 < j_2$
- $e_1 \in E_T^b, e_2 \in E_T^f$, and $i_1 < j_2$
- $e_1 \in E_T^b, e_2 \in E_T^f$, and $j_1 \leqslant i_2$

Where, $e_1 = (i_1, j_1, l_{i_1}, l_{i_1 j_1}, l_{j_1}) \in E(G)$ and $e_2 = (i_2, j_2, l_{i_2}, l_{i_2 j_2}, l_{j_2}) \in E(G)$ and E_T^f indicates the set of forward edges in the *DFS* tree of G. As well as E_T^b indicates the set of backward edges in the same tree of G. Moreover, the ordering among the terminal node labels and edge labels is straight forward and converges to lexicographic ordering. Hence, as an example, consider the graph in Fig. 2 with four vertices and four edges. The table residing in Fig. 2 contains the canonical representation, that is, minimum among all possible *DFS Codes* of the graph. Table 1 contains three potential *DFS Codes* among all possible potential *DFS Codes* of the graph in Fig. 2. Accordingly, the minimum code is the first one here, which is also considered as the canonical representation of the graph.

The frequent graph mining problem can be divided into:

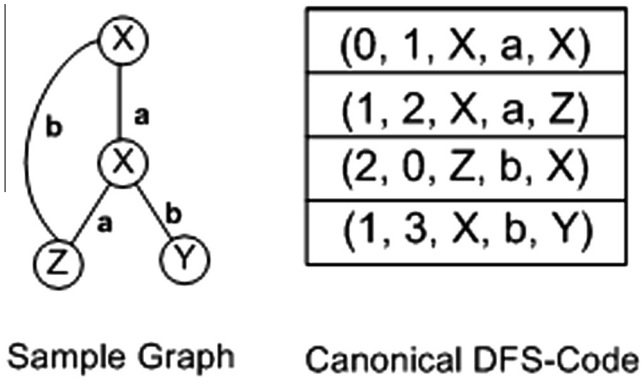


Fig. 2. *DFS Code* Representation of a Graph.

Table 1
Potential *DFS Codes* of graph in Fig. 2.

edge	DFS_Code_1	DFS_Code_2	DFS_Code_3
e_0	(0, 1, X, a, X)	(0, 1, X, a, X)	(0, 1, Y, b, X)
e_1	(1, 2, X, a, Z)	(1, 2, X, b, Y)	(1, 2, X, a, X)
e_2	(2, 0, Z, b, X)	(1, 3, X, a, Z)	(2, 3, X, b, Z)
e_3	(1, 3, X, b, Y)	(3, 0, Z, b, X)	(3, 1, Z, a, X)

- *Apriori-based approach*, which works in Apriori-based level by level wise *BFS* (Breadth First Search) manner. As for example *AGM* (Apriori-based Graph Mining) (Inokuchi et al., 2000), *FSG* (Frequent Subgraph Mining) (Kuramochi & Karypis, 2001) and Disjoint Path-based (Gudes, Shimony, & Vanetik, 2006) frequent graph pattern searching algorithms are apriori based algorithms.
- *Pattern growth-based approach*, which works in a pattern growth-based manner, that is, *DFS*-based approach of mining frequent graph patterns. For example, g*Span*(Yan & Han, 2002) and graph pattern (Li, Liu, & Gao, 2011) which works in pattern growth-based manner and performs much better than the former approach.

In Inokuchi et al. (2000), authors proposed a novel approach named *AGM*, which is a vertex-based algorithm, to efficiently mine association rules among the frequently appearing sub-structures in a given graph dataset. A transaction graph is represented by an adjacency matrix, and the frequent patterns appearing in the matrices are mined through the extended algorithm of the basket analysis. In Kuramochi and Karypis (2001), authors presented a new algorithm, named *FSG*, which considers edges as building block, for finding all connected subgraphs those appear frequently in a large graph database. This algorithm finds frequent subgraphs using the same level-by-level expansion adopted in Apriori (Agrawal & Srikant, 1994). It incorporates various optimizations for candidate generation and counting, which allows it to scale up to large graph databases.

The Apriori-like algorithms suffer two problems: (Yan & Han, 2002).

- *Costly subgraph isomorphism test*. Since subgraph isomorphism is an NP-complete problem, no polynomial time algorithm can solve it. Thus, testing of false candidates (false test or false search) degrades the performance a lot.
- *Costly candidate generation*. The generation of size $(k + 1)$ subgraph candidates, from size k frequent subgraphs, is more complicated and costly than that of itemsets, as observed in Kuramochi and Karypis (2001).

In Yan and Han (2002), authors developed *gSpan*, a new graph-based substructure mining algorithm, which requires no candidate generation and aims to avoid the two most significant costs mentioned above. The *gSpan* algorithm adopts *DFS* as opposed to *BFS* used inherently in Apriori-like algorithms. It provides a new canonical labeling system (*DFS* lexicographic order) to support *DFS*. Each graph is assigned a unique minimum *DFS Code*, based on which a hierarchical search tree is constructed. By pre-order traversal of the tree, *gSpan* discovers all frequent subgraphs with required support. Since the design combines the subgraph isomorphism test and frequent subgraph growth into one procedure, *gSpan* dramatically accelerates the mining process. The authors introduced methods for partitioning the frequent graphs according to the *DFS* lexicographic order and projection of graph datasets to fit the partitions. The partition and projection deliver good parallel and scale up properties.

Followed by frequent pattern mining, correlation mining is recognized as one of the most important data mining tasks for its capability to identify underlying dependencies between objects (Ke et al., 2008). For mining correlation there are several measures, most of which are used in mining correlation among itemsets in traditional domains (Tan et al., 2002). Authors of Ke et al. (2008) proposed a new problem of correlation mining from graph databases, called Correlated Graph Search (*CGS*). They adopted *Pearson's correlation coefficient* as the correlation measure to take into account the occurrence distributions of graphs. The *CGS* faces significant challenges, since every subgraph of a graph in the database is a candidate, but the number of subgraphs is exponential. Authors (Ke et al., 2008), devised an efficient algorithm that mines the candidate set from a much smaller projected database, and thus, *CGS* can obtain a significantly smaller set of candidates. Moreover, three heuristic rules were developed to refine the candidate set. They further improved their approach by introducing three additional heuristic rules and using some of the bounds defined in their paper, to directly answer high-support queries without mining the candidates.

In this paper, we have proposed a measure to effectively capture inherent correlations among graphs, as well as an efficient algorithm to mine the correlated graphs using the measure for addressing the limitations of existing approaches. However, this paper includes substantively new and different contributions beyond the preliminary conference version (Samiullah et al., 2013) including revised and optimized algorithm, enhanced motivation with real-life applications, rigorous analysis of the mining technique as well as the performance of the algorithm, elaborate explanation on applying the proposed algorithm in mining correlated graphs with our proposed measure, even in large size graph databases, with new figures and tables, a comprehensive presentation of the background, and detailed experimental results with discussions.

3. Proposed algorithm

Correlated graph mining is one of the most important graph mining tasks. In this paper, we have proposed a new measure, *gConfidence* and a new method, *CGM* (Correlated Graph Mining), to search correlation among graphs within graph databases. Since searching for frequent subgraphs is a prime need for correlation search, we have used a mechanism similar to *gSpan* (Yan & Han, 2002) to find frequent subgraphs. A hierarchical search space is constructed to facilitate correlated graph search efficiently. Moreover, correlation is searched based on user specified minimum correlation threshold, hence, the search space can be pruned based on two values, one for minimum support threshold and another one is minimum confidence threshold. Proposed *CGM* is an "edge"-based correlation mining algorithm, as well as the concept of "Projected Database" is used to reduce the costly searching operation for counting occurrences of any graph/subgraph.

3.1. Problem formulation

In this section, we have defined the proposed measure *gConfidence* and discussed some of its properties as well. Problem of graph correlation mining and *gConfidence Tree* construction along with their utilizations are described later.

In this paper, we have focused on undirected simple graph with vertices and edges having labels associated. However, our algorithm can fit in mining correlation to directed graph and unlabeled graph. With very few modifications, our algorithm can be extended to process non-simple graphs with self-loops and multiple edges.

Definition 1. (*gConfidence*). Given a graph database $\mathbb{GD} = \{G_1, G_2, \ldots, G_N\}$ of N graphs and any of its transaction graph $G = \{V,E\}$ where $V = \{v_0, v_1, \ldots, v_k\}$ and $E = \{e_0, e_1, \ldots, e_m\}$, then we have defined the *gConfidence* of $G_s \subseteq G$ as

$$gConfidence(G_s) = \frac{\{No.\ of\ graphs\ G | G_s \subseteq G \in \mathbb{GD}\}}{max(\{No.\ of\ graphs\ G_i \subseteq G \in \mathbb{GD} | \forall G_i \subseteq G_s\})} \quad (1)$$

Where "$max(\{No.\ of\ graphs\ G_i \subseteq G \in \mathbb{GD} | \forall G_i \subseteq G_s\})$" is the maximum support of any subgraph among possible subgraphs of G_s. This formal definition simply states that *gConfidence* is the smallest correlation of any graph G_s, that is, $gConfidence(G_s) = min\{correlation(G_s)\}$. The value of $gConfidence(G_s)$ falls within the range [0, 1], that is, $0 \leqslant gConfidence(G_s) \leqslant 1$.

The Eq. (1) is for computing correlation among the elements of a graph G_s where all possible subgraphs of a graph are considered to compute the correlation. However, the following Lemma 1 provides an optimistic solution to the exponential number of subgraph generation challenge of Eq. (1). The lemma is based on the Apriori-knowledge, that is, the frequency of any super-graph G_{sup} of a graph $G = (V,E)$ is smaller than or equal to the frequency of G. Hence, the smallest possible subgraph, G_{sub} of any graph $G = (V,E)$ is the one edge graph constructed by any edge $e_i, e_i \in E(G)$. Hence, during calculation of *gConfidence*, emphasis should be on the edges.

Lemma 1. *Given a graph database* $\mathbb{GD} = \{G_1, G_2, \ldots, G_N\}$ *of* N *graphs and any of its transaction graph* $G = \{V,E\}$, *where* $V = \{v_0, v_1, \ldots, v_k\}$ *with* $|V| \geqslant 2$ *and* $E = \{e_0, e_1, \ldots, e_m\}$ *with* $|E| \geqslant 1$, *then we can define the gConfidence of* $G_s \subseteq G$ *derived in* Eq. (1) *as*

$$gConfidence(G_s) = \frac{\{No.\ of\ Graphs\ G | G_s \subseteq G \in \mathbb{GD}\}}{max(\{\forall e_i \in E(G_s),\ No.\ of\ graphs\ G_j | e_i \in E(G_j),\ G_j \in \mathbb{GD}\})} \quad (2)$$

Proof. In Eq. (1), "$max(\{No.\ of\ graphs\ G_i \subseteq G \in \mathbb{GD} | \forall G_i \subseteq G_s\})$" represents the maximum value of the frequency of all possible subgraphs of G_s. Our approach of mining correlation is edge-based and we start from a single edge $\{e_0\}$ then we add 1 edge every time to get next subgraph pattern $\{e_0, e_1\}$. Now, Consider that $frequency(\{e_0\}) =$ "X" and $frequency(\{e_0, e_1\}) =$ "Y". Here, $frequency(\{e_0\}) \geqslant frequency(\{e_0, e_1\})$. In this manner, for larger patterns, the same scenario will be observed. Therefore, we can conclude that for maximum frequency of any subgraph of G_s, the individual edge frequency of G_s, that is, $frequency(e_i \in E(G_s))$ will dominate and we can replace "$max(\{No.\ of\ graphs\ G_i \subseteq G \in \mathbb{GD} | \forall G_i \subseteq G_s\})$" by "$max(\{\forall e_i \in E(G_s),\ No.\ of\ graphs\ G_j | e_i \in E(G_j),\ G_j \in \mathbb{GD}\})$" and calculate the metric *gConfidence*. Hence, Eq. (2) will provide same result as in Eq. (1). □

As an example, suppose we need to compute the inherent correlation of two groups of people shown in Fig. 1, as discussed in Section 1. We found that frequency of G_1 is 30, that is, G_1's people work together in 30 events and the maximum frequency of any edge of the graph representing G_1 is 100, i.e., highest number of interactions performed by any individual is 100. Therefore, *gConfidence*$(G_1) = 0.3$ and for the second group the frequency of G_2 is 30, that means, people of second group work in 30 events all together. As well as maximum frequency of any edge is 60, that is, highest interaction of any individual of G_2 is 60. Therefore, *gConfidence* $(G_2) = 0.5$ which implies that people of second group are more inherently correlated than people of first group.

Definition 2. (*Correlated Graph Mining*). Given a graph database $\mathbb{GD} = \{G_1, G_2, \ldots, G_N\}$, a user specified minimum support threshold, σ and a user specified minimum correlation threshold, θ, interesting graphs/subgraphs need to be searched, that is, the set of graphs $G_I = \{\forall G_i | supp(G_i) \geqslant \sigma;\ gConfidence(G_i) \geqslant \theta\}$ need to be mined.

It implies that the problem is to search for graphs having support count greater than or equal to σ and *gConfidence* value greater than or equal to θ.

For convenience, we assume $\mathbb{E}$ as the set of edges and $\mathbb{V}$ as the set of vertices, found in the graph database $\mathbb{GD}$ unless stated otherwise, that is, $\mathbb{GD} = \{\mathbb{V}, \mathbb{E}\}$.

Definition 3. (*Internal Occurrence*). For a graph database $\mathbb{GD} = \{G_1, G_2, \ldots, G_N\}$ and any frequent subgraph $G \subseteq G_i \in \mathbb{GD}$, where $G = \{V,E\}$ and $V = \{v_0, v_1, \ldots, v_n\}$, $E = \{e_0, e_1, \ldots, e_m\}$, any edge $e_j \in \mathbb{E}$, the internal occurrence of e_j *w.r.t.* G is $\left|OC_G \bigcap OC_{e_j}\right|$, where OC_G is the set of occurrence graphs of G and OC_{e_j} is the set of occurrence graphs for e_j.

Definition 4. (*External Occurrence*). For a graph database $\mathbb{GD} = \{G_1, G_2, \ldots, G_N\}$ and any frequent subgraph $G \subseteq G_i \in \mathbb{GD}$, where $G = \{V,E\}$ and $V = \{v_0, v_1, \ldots, v_n\}$, $E = \{e_0, e_1, \ldots, e_m\}$, any edge $e_j \in \mathbb{E}$, the external occurrence of e_j *w.r.t.* G is $\left|OC_{e_j} - \left\{OC_G \bigcap OC_{e_j}\right\}\right|$, where OC_G is the set of occurrence graphs of G and OC_{e_j} is the set of occurrence graphs for e_j.

Definition 5. (*Occurrence Ratio*). For a graph database $\mathbb{GD} = \{G_1, G_2, \ldots, G_N\}$ and any frequent subgraph $G \subseteq G_i \in \mathbb{GD}$, where $G = \{V,\ E\}$ and $V = \{v_0, v_1, \ldots, v_n\}$, $E = \{e_0, e_1, \ldots, e_m\}$, for any edge $e_j \in E \subseteq \mathbb{E}$ with the internal occurrence of e_j *w.r.t.* G being IOC_{e_j} and external occurrence of e_j *w.r.t.* G being EOC_{e_j}, then the Occurrence ratio of e_j *w.r.t.* G is the ratio among IOC_{e_j} and EOC_{e_j} in G.

In particular, the Occurrence ratio is an indication of correlation that takes into account the statistical distribution of $e_j \in \mathbb{E}$. It considers the contribution of that particular edge e_j in constructing a graph $\{G\}$ and the contribution of that same edge e_j in constructing other graphs $\{G' \in \mathbb{GD} - G\}$. Hence, the occurrence ratio is capturing internal correlation among the elements of a graph. As a consequence, in calculating *gConfidence*, the occurrence ratio of each edge and each subgraph is brought into consideration and internal correlation is captured.

As an example, say in any graph dataset $\mathbb{GD} = \{G_1, G_2, \ldots, G_k\}$, if any edge $e \in E(G_i \in \mathbb{GD})$ is found n times where the edge e co-occurs m times with a frequent subgraph $G \subseteq G_i \in \mathbb{GD}$, then the internal occurrence of e will be m with respect to G. Moreover, the external occurrence of e will be $n - m$ with respect to G. As a consequence, the *Occurrence Ratio* of e will be $\frac{m}{n-m}$ for the subgraph G.

For instance, consider the graph G_1 in Fig. 1, which consists of seven edges with frequencies in ascending order {50, 62, 65, 70, 75, 90, 100} in a particular graph database. However, the frequency of G_1 is 30 only in that graph database. Then the set of *Occurrence Ratio* for the edges will be $\{\frac{30}{20}, \frac{30}{32}, \frac{30}{35}, \frac{30}{40}, \frac{30}{45}, \frac{30}{60}, \frac{30}{70}\}$. As a consequence, the smallest value is $\frac{30}{70}$, which is the smallest occurrence distribution among any of the element of the graph G_1, indicating the smallest affection among the elements (vertices and edges) of the graph G_1 and considered as the analogous correlation value of *gConfidence*.

In actual sense, *gConfidence* is directly proportional to *Occurrence Ratio*. This can also be realized from the definition of *gConfidence*(Definition 1), that is, smallest *Occurrence Ratio* (Definition 5) of any subgraph indicates the correlation among the elements of any graph and conforms *gConfidence* for a particular graph. We should note that the maximum value will occur when the subgraph of G_s consists of a single edge. Adding additional edges neither increases the frequency of G_s nor the *gConfidence*, and, the *Occurrence Ratio* remains same or decreases as well.

We have listed some of the important properties of *gConfidence* measure below:

Property 1. *It has the downward closure property. If a subgraph/graph is passing a minimal gConfidence threshold, so is every one of its subgraphs.*

Proof. If a graph/subgraph $G_i = \{V_i, E_i\}$ does not support minimum confidence threshold, then further growing of such graph will not support the minimum confidence threshold, that is, if $support(G_i) < \sigma$ then $support(G_j) < \sigma$ where $G_i \subset G_j$ since $frquency(G_i) \geqslant frequency(G_j)$. In calculating *gConfidence* we have a denominator "$max(\{\forall e_i \in E(G_s),\ No.\ of\ graphs\ G_j | e_i \in E(G_j),\ G_j \in \mathbb{GD}\})$" in Eq. (2), where increasing of graph sequences will either increase the denominator and decrease the *gConfidence* value or the value will remain same. Hence, the extended sequence will not satisfy *gConfidence* threshold if original does not. Similarly, we can state that, if any graph pattern, "*GP*" satisfies the correlation threshold, that is, if $gConfidence(GP) \geqslant \theta$, then the denominator of Eq. (2) for its any subgraph pattern, "*SGP*", will be smaller or equal and $gConfidence(SGP) \geqslant \theta$, hence $gConfidence(SGP) \geqslant gConfidence(GP)$. Therefore, we can state that, *gConfidence* measure satisfies the downward closure property. □

For illustration, consider a graph that has correlation value as c and its subgraph has correlation value sc, then it must hold that $c \leqslant sc$. Hence, if $c \geqslant \theta$, then $sc \geqslant \theta$ must hold with θ being the correlation threshold. This corresponds to the Apriori property, that is, if a graph passes the correlation threshold barrier then all of its subgraph will surely pass the barrier.

The most important property of a metric in measuring correlation in a large database is Null-invariance. It's a property of some transactions known as null transactions. To be precise, a null transaction is referred to as a transaction where there are some items which are not examined in the measure. In a typical transaction database, a particular item appearing in very few transactions may have very small probability. Since most of the transactions do not contain such item, they are null transactions with respect to the item. If the correlation among a set of patterns being analyzed is affected by the transactions that contain none of them (i.e., null- transactions), such a measure is unlikely to be of high quality (Wu, Chen, & Han, 2007).

Property 2. *This measure is null-invariance. That is, it is not affected by null-transactions.*

Proof. In calculating *gConfidence*(G) for a graph $G = (V, E) \in \mathbb{GD}$, we consider only the edges $e \in E(G)$. We need not to consider those edges which don't belong to the set of edges of the graph i.e. $|\mathbb{E} - e|$. From Lemma 1, we found that *gConfidence*(G_s) for a graph G_s = (V_s,E_s) depends on two values, the *frequency* (G_s), i.e., we need to consider only the *frequency*($v \in V_s(G_s)$),*frequency*($e \in E_s(G_s)$) and "$max(\{\forall e_i \in E(G_s),\ No.\ of\ graphs\ G_j | e_i \in E(G_j),\ G_j \in \mathbb{GD}\})$", that is, only the edges $e_i \in E_s(G_s)$ are brought into consideration. Hence, we can state that the measure *gConfidence* is null-invariant. □

As an example, in calculating the correlation for the graphs in Fig. 1, we only considered the frequency of the elementary edges for a particular graph and never considered values which correspond to the frequency of elements other than elements of the graph itself.

Property 3. *gConfidence ensures a lower bound on confidence for any subgraph, which satisfies the minimum threshold.*

Proof. This particularly describes an implication of *gConfidence*, that is, the correlation calculated by this measure for a particular graph G_s = (V,E), indicates the lower bound of correlation for any subgraph of G_s, that is, *gConfidence*(G_s) = *Min*(*Correlation*(G_i)) where $\forall_i\ G_i \subseteq G_s$. Indeed, the Lemma 1 ensures that *gConfidence* is the ratio among *frequency*(G_s) and *Max*(*frequency*(e_i)) where $e_i \in E(G_s)$. Moreover, the denominator of Eq. (2) indicates that the edge-wise elementary maximum frequency contributes in *gConfidence* calculation. Hence, for that particular graph G_s, calculated correlation found by applying the Eq. (2), is smallest among all the correlation values. Now, the impact of numerator needs to be considered, which is constant for the same graph G_s. Hence, *gConfidence* safely holds this property. □

gConfidence provides information about implication that it is the minimal confidence among the set of subgraphs of G_s. As an example, consider the first frequent graph in Fig. 1, where correlation is calculated 0.3 which is the smallest value among the correlations calculated for all possible subgraphs of the graph. Hence, *gConfidence* can be regarded as the lower bound of the *confidence*/*correlation* of the graph.

Property 4. *gConfidence finds graphs/subgraphs those are inherently correlated and statistically co-occur.*

Proof. From Lemma 1, we can conclude that in order to calculate *gConfidence* of a graph G_s, we need to consider the ratio among *frequency*($G_s \in \mathbb{GD}$) and the *Max*(*freq*(e_i)) where $e_i \in E(G_s)$. Now, consider two graphs g_1 = (V_1,E_1) and g_2 = (V_2,E_2) with *frequency*(g_1) = f_1 and *frequency*(g_2) = f_2 respectively. Say, f_1 and f_2 are equal in this case. Also consider $max(g_1) = m_1$ and $max(g_2) = m_2$ where, $max(G)$ = *Max*(*frequency*(e_i)) where $e_i \in E(G)$ for a graph G = (V,E). Therefore, $gConfidence(g_1) = \frac{f_1}{m_1}$ and $gConfidence(g_2) = \frac{f_2}{m_2}$. Now, $gConfidence(g_1) \gg gConfidence(g_2)$ if $m_1 \ll m2$. This implies, g_2's elements are loosely connected within the graph g_2 with compared to g_1. Hence, we can claim that *gConfidence* finds graphs those are more inherently correlated. Furthermore, we have considered that f_1 and f_2 are same. Some of the edges/vertices of g_2 occur a large number of times with other graphs. Hence, the occurrence ratio of g_2, that is, the ratio among $IOc(g_2)$ and $EOc(g_2)$, is smaller than g_1's occurrence ratio. Hence, it can be stated that the measure *gConfidence* selects graphs those have more statistical co-occurrence ratio, that is, more statistically co-occurring graphs. □

This property describes that *gConfidence* is a measure where two most important characteristics, inherent correlation, i.e., correlation among various elements and statistical distribution of a graph, are considered. As an example, in the scenario, illustrated in Fig. 1, the second frequent graph is found more correlated than the first one. Because, the elements of second one are most statistically co-located and more inherently correlated.

3.1.1. gConfidence tree

In a *gConfidence Tree*, each node represents a graph or subgraph by storing corresponding *DFS Code* and also represents correlation by storing *gConfidence* value "*gC*", the relation between parent node and child node complies with the relation that a parent is one edge shorter in size than its child and a child is one edge larger than its parent and no child has "*gC*" greater than its parent. The relation between siblings is consistent with the *DFS* lexicographic order. That is, the pre-order search of *gConfidence Tree* follows the *DFS* lexicographic order.

If we are given a label set $L = \{l_0, l_1, \ldots, l_k\}$, a *gConfidence Tree* can be constructed by the metrics of Eq. (1) or Eq. (2), which could contain all possible graphs for this label set. In Fig. 3, we have shown a *gConfidence Tree* where the $(n+1)$-th level of the tree has nodes which contain *DFS Codes* of n-edge graphs and a value "*gC*", which represents the correlation of that particular subgraph with the other subgraphs in the tree.

The *DFS Code* and node in the *gConfidence Tree* are equivalent in the sense that one can be derived from the other. Any valid *DFS Code* has a unique corresponding node in the *gConfidence Tree* and any node in the *gConfidence Tree* contains a valid *DFS Code*. Certainly some of the nodes contain a minimum *DFS Code* while others do not. Nonetheless, there could be some nodes having "*gC*" values smaller than the minimum correlation threshold. This ensures second level of pruning of subgraph candidates in optimizing correlation calculation. The value for *gC* also maintains a parent and child relationship, that is, $gC(\alpha) \geqslant gC(\beta)$ where α = (a_0,a_1, …,a_m) and β = (a_0,a_1, …,a_m,b), that is, α is β's parent.

Eq. (1) can calculate *gConfidence* for a graph G_s = (V_s,E_s) by considering all possible subgraphs of G_s and Eq. (2) can optimize the calculation cost by only considering the edges $e \in E_s(G_s)$. However, we still need to compare the frequency of the edges $e \in E_s(G_s)$ to determine the most frequent edge. The following Lemma 2 performs another level of optimization by removing the necessity of computing maximum frequency among the frequency of the edges $e \in E_s(G_s)$.

Lemma 2. *For a graph database* $\mathbb{GD} = \{G_1, G_2, \ldots, G_N\}$ *of N number of graphs, gConfidence for a subgraph* G_s = (V_s,E_s) *of any transaction graph* G = (V,E) *where* $V = \{v_1, v_2, \ldots, v_m\}$ *with* $|V| \geqslant 2$ *and* $E = \{e_1, e_2, \ldots, e_m\}$ *with* $|E| \geqslant 1$, *i.e., any* $G_s \subseteq G$ *with* S = *Min(DFS_Code*(G_s)*)*

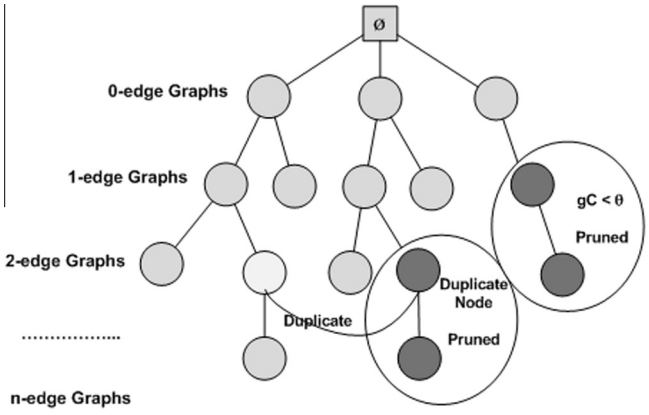


Fig. 3. *gConfidence Tree*: A search space.

and e_0 being the first edge in the minimum DFS Code representation of the subgraph G_s, then gConfidence(G_s), defined in Eq. (1) *can be defined as:*

$$gConfidence(G_s) = \frac{\{No.\ of\ Graphs\ G | G_s \subseteq G \in \mathbb{GD}\}}{\{No.\ of\ graphs\ G | e_0 \in S, e_0 \in E(G_s),\ G_s \subseteq G \in \mathbb{GD})\}} \quad (3)$$

Proof. From Lemma 1, we found that the denominator of Eq. (2) in calculating *gConfidence*(G_s) is the maximum value among the frequencies of the elementary edges of G_s. While constructing graphs as candidate by adding one edge at a time, the proposed algorithm represents the graphs by *DFS Code*, as proposed in *gSpan* (Yan & Han, 2002) with assurance that the edge will be inserted in the representation in descending order of their frequency. In this way, *frequency*(e_0) = *Max*(*frequency*($e \in E(G_s)$)), where e_0 is the first and most frequent edge in the minimum *DFS Code* representation of a graph $G_s = (V,E)$. Hence, *frequency*($e \in \{E(G_s) - e_0\}$) $\leqslant$ *frequency*(e_0). Therefore, it can be claimed that "*max*($\{\forall e_i \in E(G_s),\ No.\ of\ graphs\ G_j | e_i \in E(G_j),\ G_j \in \mathbb{GD}\}$)" and "$\{No.\ of\ graphs\ G | e_0 \in S, e_0 \in E(G_s),\ G_s \subseteq G \in \mathbb{GD})\}$" are equal and equivalent. □

As a consequence, we can calculate *gConfidence*(G_s) more efficiently and in a more optimistic way by using Eq. (3) instead of Eqs. (2) and (1).

3.1.2. The snapshot of the algorithm

Here we have stated the basic steps of *CGM*, required to calculate the graph correlation using our proposed measure *gConfidence*.

- Mine frequent vertices and edges along with their frequency.
- Keep the edges support count global.
- When any frequent subgraph is found, treat it as a candidate of being a node in *gConfidence Tree*.
- In each node of the tree, store *Max* and *gC* which represent maximum value of single item (edge) *frequency* and *gConfidence* respectively.
- To determine maximum elementary edge *frequency* of any graph *G*, keep the *frequency*(e_0)
  frequency()=function for counting frequency,
  e_0 = the first edge among the edges within *DFS Code* of graph *G*.
- Apply *gConfidence* measure to calculate correlation among graphs.
- At the termination we will have our frequent-interesting graph patterns.

For illustrating the working procedure of our *CGM* algorithm, we can consider the graph database for the chemical dataset shown in Fig. 4. We have assumed the support threshold, $\sigma = \frac{1}{3}$ and correlation threshold, $\theta = \frac{2}{3}$. For simplicity we have chosen such a graph database of Fig. 4, which contains no cyclic graph. Hence, in generating potential children we need not to consider the children found by adding a back edge or self loop.

According to our algorithm we have calculated *support* and *gConfidence* of each edge and vertex and then selected the frequent and correlated vertices and edges. Now, we have to construct single edge graph for each frequent-correlated edge and the edge set is also used to construct *GLOBAL EDGE MATRIX*. This matrix is sorted based on *support* count and *DFS Code* and can be used in a chronological order for constructing potential children (smallest *DFS Code* oriented child first) and also helps in counting maximum elementary edge support count of a graph. (See Fig. 5).

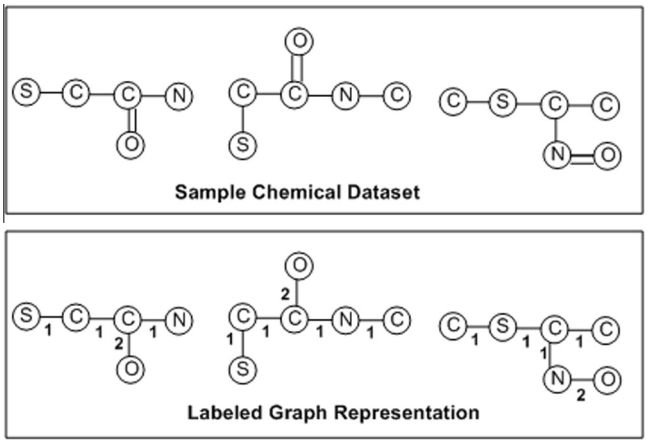
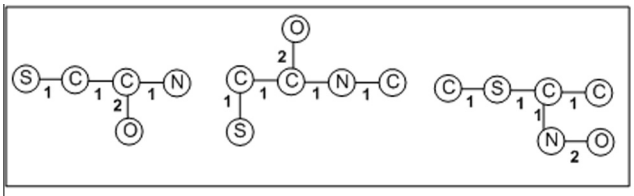


Fig. 4. Sample chemical and corresponding graph database.



Single Vertex Graph	Supp	gC	Status
C	3/3	3/3	✓
N	3/3	3/3	✓
O	3/3	3/3	✓
S	3/3	3/3	✓

Single Edge Graph	Supp	gC	Status
C–C	3/3	3/3	✓
C–N	3/3	3/3	✓
C–S	3/3	3/3	✓
C=O	2/3	2/2	✓
N=O	1/3	1/1	✓

Fig. 5. Selecting and sorting of vertices and edges.

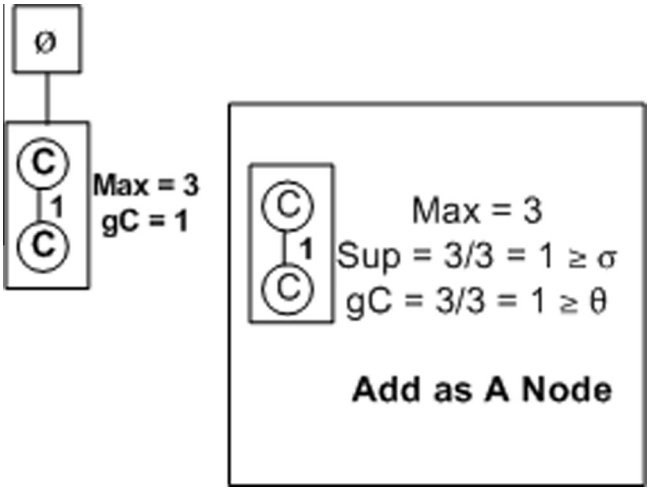


Fig. 6. Selecting Frequent-Correlated graph C_1.

We have created a Null-rooted *gConfidence Tree* and then started mining for the first 1-edge graph C_1. Since it satisfies both threshold values, we have added it in the search space which is

illustrated in Fig. 6 and started mining the correlation of its potential children recursively. Its first child $C_{1.1}$ is not frequent. Hence we have pruned it as shown in Fig. 7. The second child $C_{1.2}$ is frequent and correlated, hence added to the search space as shown in Fig. 8. However, recursive calculation of its children shows that $C_{1.2.1}$ and $C_{1.2.4}$ are frequent but not correlated and $C_{1.2.2}$ as well as $C_{1.2.3}$ are not frequent, hence we have to eliminate them as shown in Fig. 9. The fifth child of $C_{1.2}$, that is, $C_{1.2.5}$ is found correlated and added to *gConfidence Tree* and this is shown in Fig. 10. The eighth child of $C_{1.2.5}$ is found correlated and added in the search space as in Fig. 11 but first seven children of $C_{1.2.5}$ are not frequent hence we did not consider it for correlation searching using our proposed measure and pruned from the search space. Therefore, we have to search further for the correlation of all possible children of $C_{1.2.5.8}$.

Since no child of $C_{1.2.5.8}$ was found frequent-correlated, we can backtrack to $C_{1.2.5}$. However, already we have checked all possible children of $C_{1.2.5}$, hence we can trace back again to $C_{1.2}$ for checking correlation of its remaining children. We found sixth and seventh children of $C_{1.2}$ non-frequent and by skipping them we have calculated the correlation of eighth child of $C_{1.2}$, that is $C_{1.2.8}$, and added it in the *gConfidence Tree* because it is found frequent-correlated as shown in Fig. 12. Then we have to recursively mine the correlation of all potential children of $C_{1.2.8}$. However, all children of $C_{1.2.8}$ were found non-frequent and we have pruned them from the search space. Therefore, we backtrack to $C_{1.2}$ and no child of $C_{1.2}$ remains unchecked. Hence we go for checking remaining children of C_1 by tracing back one level up in the search space.

Then we get the next child of C_1, i.e., $C_{1.3}$ as frequent and correlated and added it to the search space as shown in Fig. 13. Now we have to check the correlation of all possible children of $C_{1.3}$. We found that first child of $C_{1.3}$, i.e., $C_{1.3.1}$ is frequent but not correlated and pruned it. Then the second to sixth children of $C_{1.3}$, that is, $C_{1.3.2}$ to $C_{1.3.6}$ are found non-frequent and pruned. Seventh and eighth children of $C_{1.3}$ are found duplicate by subgraph isomorphism test, that is, these are already mined. However, ninth child of $C_{1.3}$, i.e., $C_{1.3.9}$ was found frequent-correlated and has been added to *gConfidence Tree* as shown in Fig. 14.

Now we have to recursively mine correlation of all potential children of $C_{1.3.9}$. We found that $C_{1.3.9.2}$ is duplicate and $C_{1.3.9.1}$ and $C_{1.3.9.3}$ to $C_{1.3.9.5}$ are infrequent. Hence we trace back to $C_{1.3}$ and all of its children have been checked. Therefore, we have to backtrack to one level up in the *gConfidence Tree* and need to check for remaining children of $C_{1.3}$. All children of $C_{1.3}$ have been checked. So, we again trace back to one level up in the search space and tried for the next child of C_1, that is, $C_{1.4}$ and it is found frequent-correlated, hence, is added to the search space as shown in Fig. 15. Accordingly, we have added it in the search space.

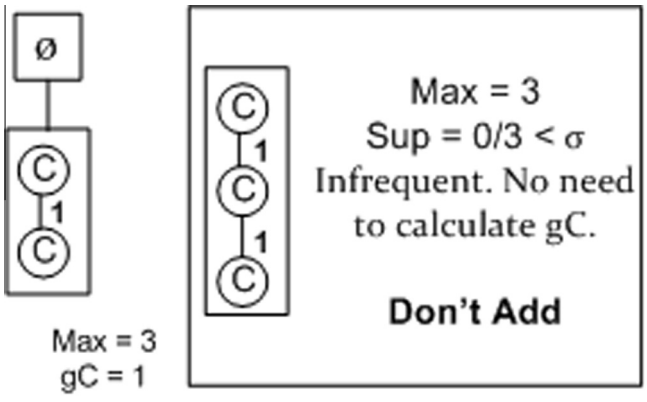


Fig. 7. Pruning infrequent graph $C_{1.1}$.

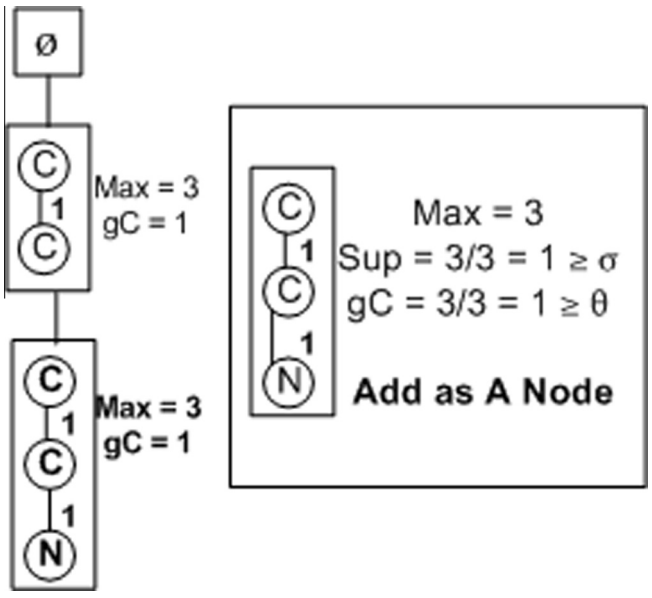


Fig. 8. Selecting $C_{1.2}$ for further extension.

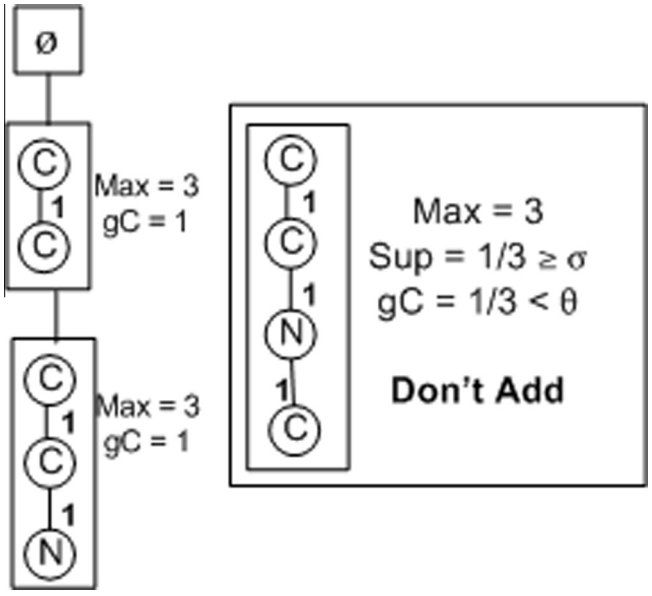


Fig. 9. Pruning of $C_{1.2.1}$ based on *gConfidence* metric.

However, no child of $C_{1.4}$ is found frequent-correlated and has been eliminated. Now we trace back to one level up in the tree and found no remaining child to be processed.

Now we start mining for second 1-edge graph C_2 and found it frequent-correlated and added in the search space as shown in Fig. 16. Then we have mined its all potential children for correlation searching and first child $C_{2.1}$ is frequent but not correlated, $C_{2.2}$ is duplicate and $C_{2.3}$, $C_{2.4}$ are found infrequent. However, $C_{2.5}$ is found frequent and correlated. Hence we added it to the search space as shown in Fig. 17. Now we find that its second child $C_{2.5.2}$ is duplicate and all other children are non-frequent and have been pruned. Then we have traced back to one level up in the search space and found that C_2 has no more unchecked child.

Then we started mining correlation of 3rd 1-edge graph, that is, C_3 and found it frequent and correlated. After adding it as shown in Fig. 18, we started mining correlation for its potential children. None of its children are found frequent except the first one, $C_{3.1}$, which is frequent but not correlated and the second child is

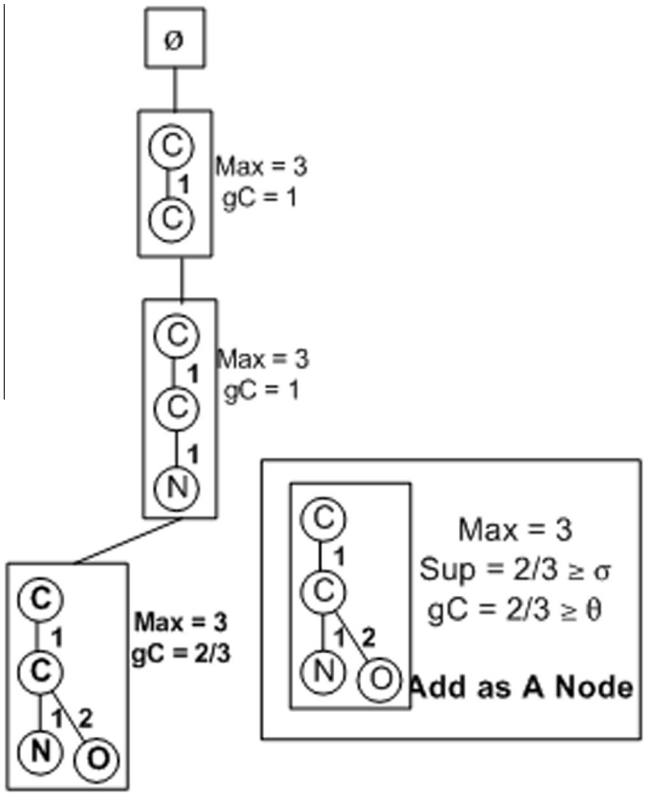


Fig. 10. *gConfidence* calculation and adding $C_{1.2.5}$ in *gConfidence Tree*.

duplicate. As a consequence, we have pruned all of C_3's children. Then we start mining correlation for fourth 1-edge graph C_4. C_4 is found frequent-correlated and added in the *gConfidence Tree* as shown in Fig. 19. We found none of C_4's children frequent-correlated and pruned all of them. Now, the last one edge graph C_5 should be considered in the hierarchy, which is found frequent and correlated based on the thresholds of support and correlation. Hence, C_5 can be safely added in the *gConfidence Tree* as a child node of the root, similarly as Fig. 20.

As a consequence, we have to recursively check for the potential children of C_5. However, the children of C_5 were found neither frequent nor correlated and were safely pruned from the search space. In this way, we have calculated the correlation of the given graph database and found a complete *gConfidence Tree* in Fig. 20. Nodes of the tree contain correlated graphs along with the amount of correlation and we have used the concept of using projected database to reduce the cost of searching databases. Since *gConfidence* satisfies the downward closure property, we have safely used the projected database to reduce the searching cost.

3.2. Algorithm description and analysis

In Section 3, we have presented our algorithm for mining graph correlation using our proposed measure *gConfidence*. Here we have presented the pseudo code of our algorithm, described the algorithm in a broader sense and then analyzed our algorithm to some extent.

3.2.1. Description of the proposed algorithm

Fig. 21 shows the pseudo code of the Correlated Graph Mining (*CGM*) algorithm, where from lines 2 to 13, sorting of vertices and edges and elimination of non frequent of them are done. Lines 14 and 15 relabel the vertices and edges in descending order of their frequency so that the nodes and edges having larger

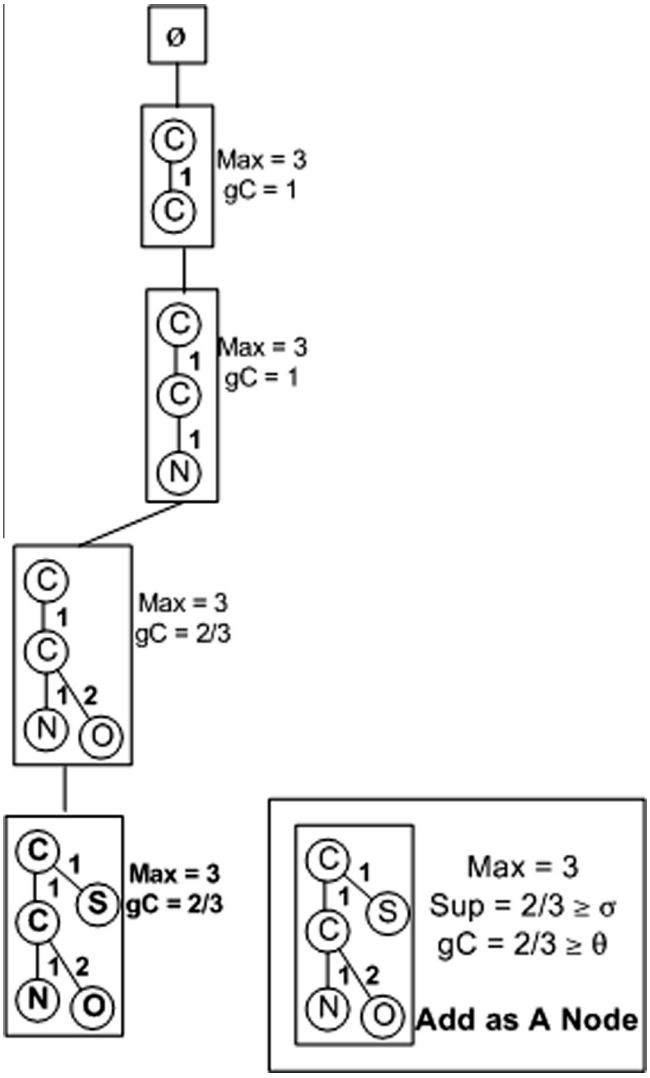


Fig. 11. *gConfidence* calculation and adding $C_{1.2.5.3}$ in *gConfidence Tree*.

frequency will be assigned minimum label to ensure minimum *DFS Code* assignment to frequently appearing elements. In line 16, we have constructed a global cost matrix for the easy retrieval of elementary frequency of each single element (edge), named *GLOBAL EDGE MATRIX*. Lines 17 to 19 constructed 1-edge frequent subgraphs and sorted them in the set, so that minimum *DFS Code* holder and maximum frequency oriented graphs can be processed first.

Now, the "*for loop*" in lines 20 to 26 processes each single edge graph by first initializing a single edge graph with their projected graph database. For efficient management of memory, instead of storing whole graph database, we have just stored projected database. However, for each pass of the "*for loop*" in lines 20 to 26, the *MAX* value that stores the maximum frequency among the all possible subgraphs of a graph, which is required for the calculation of *gConfidence* for that graph, is set to the frequency of a single edge graph which belongs to the *level* − 1 nodes in *gConfidence Tree*. According to Lemma 2, the graphs which can be formed by traversing any branch of the *gConfidence Tree*, will have frequency smaller than or equal to the frequency of the *level* − 1 graph of that particular branch.

Then we call for *gConfMining*() procedure of the algorithm to calculate *gConfidence* for that single edge graph. The procedure is recursive, which mines all the descendent of this graph along with

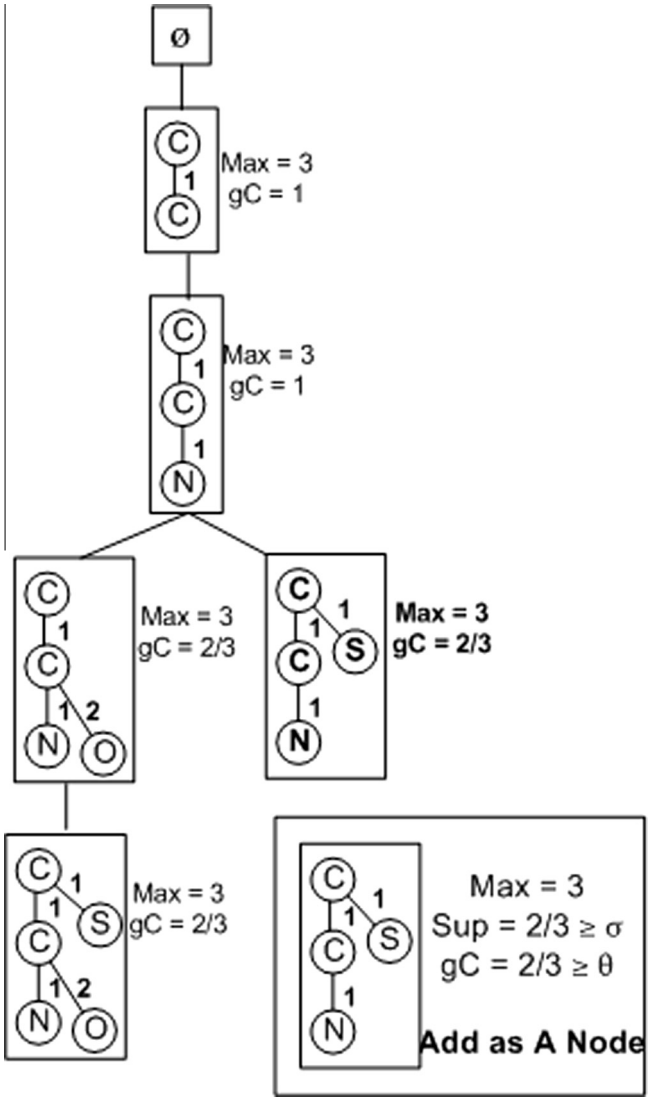


Fig. 12. *gConfidence* calculation and adding $C_{1.2.8}$ in *gConfidence Tree*.

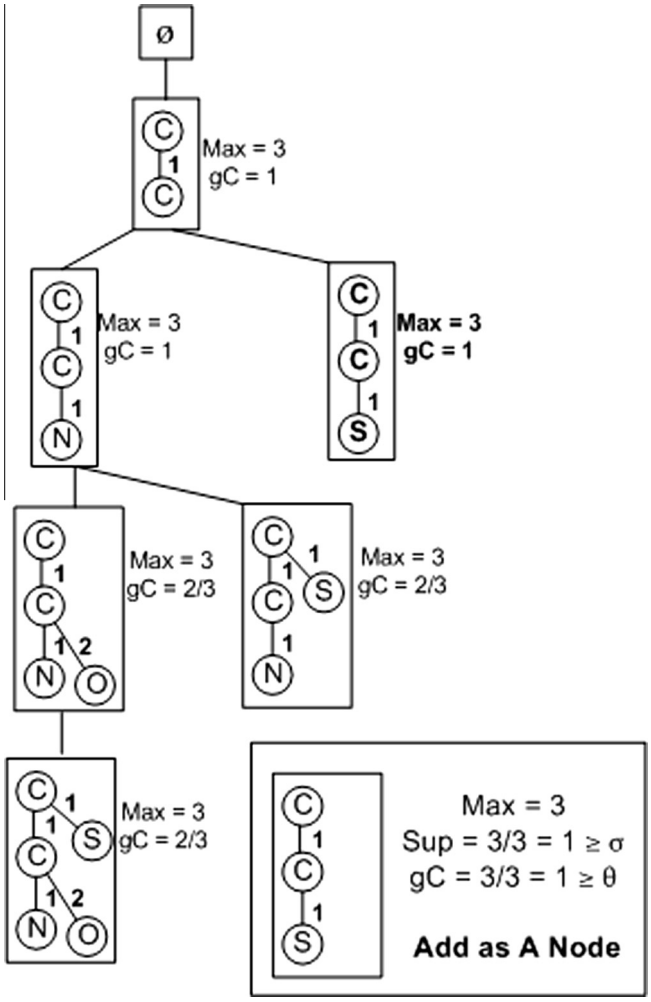


Fig. 13. *gConfidence* calculation and adding $C_{1.3}$ in *gConfidence* tree.

their *gConfidence* values. We have passed 5 parameters in the procedure *gConfMining*(), those are *GS*, *S*, *s*, *MAX* and S_1 representing *Projected* database of *s*, set of frequent-correlated graphs mined so far, the most recent subgraph which will be considered for expanding and set of frequent-correlated 1-edge graphs those can be used for children generation, respectively. After completing the *gConfMining*() procedure, we remove the edge from the edge set of the actual graph setting to ensure non-redundant children creation and processing.

In the procedure *gConfMining*() of Fig. 21, at first we check for whether the graph that needs to be processed has minimum *DFS Code* or not in lines 30 to 32. If the code is not minimum then it can be concluded that the graph has already been generated earlier. In lines 34 to 37, the *gConfidence* measure of minimum *DFS Code* oriented subgraph is calculated and then the value is compared against the threshold. If the graph support count does not satisfy support threshold and/or the *gConfidence* of the graph is less than the correlation threshold then the graph will be pruned and will not be included in the *gConfidence Tree*. If any graph satisfies all the three constraints imposed by *CGM* algorithm then it is added in the search space and its *gConfidence* value is added in the graph and recursively call the *gConfMining*() to mine its descendants in a *DFS* manner from lines 38 to 46.

3.2.2. Pruning by gConfidence

We can prune non-frequent nodes from extending within the *gConfidence Tree*, using a minimum support threshold, σ. As well as we have added an extra feature of pruning nodes from extending, using a minimum correlation threshold, θ. Because of the downward closure property of *gConfidence* measure, we can safely prune those graphs having *gConfidence* value smaller than θ along with all of the descendants of those graphs. This pruning is safe and advantageous. Because, it prunes a large number of graphs and subgraphs, those could be a candidate for being a node in *gConfidence Tree*.

3.2.3. Efficiency of selection by CGM over CGS

CGS algorithm extracts co-occurring graphs, those share similar properties in spite of having diverse structure, from a graph database, $\mathbb{GD} = \{G_1, G_2, \ldots, G_N\}$. Suppose, for the graph database, exposed in Fig. 22, the confidence threshold is $\theta = \frac{2}{3}$. Using *CGS* we can extract those graphs/subgraphs, $G = \{V', E'\}$ which can be found together in at least two graphs with a query graph, $q = \{v'', e''\}$. As instance, consider the query graphs in Fig. 22. In order to seek similarity between these two graphs, their co-occurrence can be calculated which indicates that these co-occur in all three graphs. Hence, according to the molecular characteristics, statistical co-occurrence, rather than structural similarity, can define uniqueness. Again, the *CGS* can look for correlation for any of the query

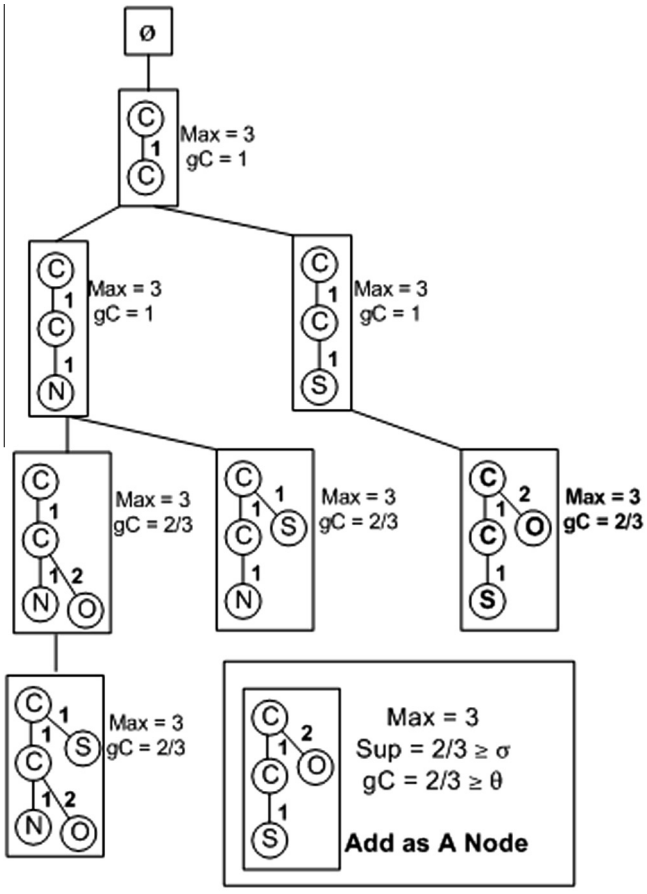


Fig. 14. *gConfidence* calculation and adding $C_{1.3.9}$ in *gConfidence Tree*.

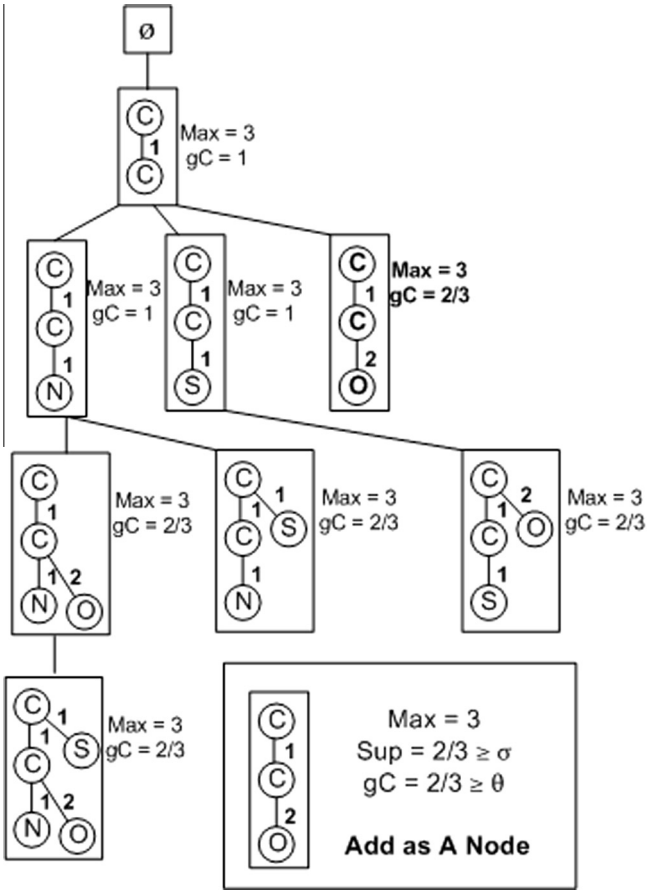


Fig. 15. *gConfidence* calculation and adding $C_{1.4}$ in *gConfidence Tree*.

graph with the graphs and subgraphs found in $\mathbb{GD}$ in almost similar approach.

Conversely, *CGM* mines the frequent-closed-correlated graph fragments $G'' = \{V'', E''\}$ from the graph database $\mathbb{GD}$ at the highest level of its hierarchy whilst processing the database. As an instance, for the same graph database of Fig. 22 with same correlation threshold, the closed-frequent-correlated database shown in the same figure will be generated. From Section 3.2.2, it is clear that *CGM* carries out another level of filtering using *gConfidence* as a correlation measure. Since the correlation threshold helps in selecting graphs/subgraphs with precise level of correlation affinities as well as both the measure of *CGM* and *CGS* emphasize on co-occurrence to mine correlation, the correlated closed graph mined by *CGM* contains the correlated graphs/subgraphs those co-occur, which is substantiated in the Property 4 of *gConfidence* measure. Moreover, Property 3 guaranteed that *gConfidence* of a graph is the minimum confidence for that particular graph. It can be noticed that both the query graphs are contained in the frequent-closed-correlated graph.

As a consequence, any graph query can be executed for finding correlation within a graph database by checking the query as a subgraph of the frequent-closed-correlated graph segment. This implies construct frequent-correlated-closed graphs from a database once and run query on these graph segments only, which are very smaller in number. That means, query optimization to an order of magnitude is done with an efficient filtering and selection of correlated graph segments using *CGM* algorithm and *gConfidence* measure. Hence, no domain knowledge is mandatory for the user and an overview can be found that whether and which graphs are in good affinities by computing all possible subgraphs of the closed-frequent-correlated graph segments.

4. Experimental results

We have performed a comprehensive performance study in our experiments on both synthetic and real-life datasets. We have used our own synthetic graph generator to construct synthetic dataset. The real-life datasets we have tested are cancer datasets provided in Pubchem (2011). In both type of data, our algorithm is proved to be sound and efficient as well as found scalable, faster and efficient enough that it can mine correlated graphs with any size and any level of complexity. The graph size can vary on number of vertices, number of edges, number of labels and density of the connectivity among nodes in a graph. Another form of variation in graph size or data size can be the number of graphs in a graph database. Moreover, we have compared the performance of our proposed *CGM* algorithm with *CGS* (Ke et al., 2008) and *gSpan* (Yan & Han, 2002) to ensure the efficiency of our proposed method and measure in mining correlated graphs.

All experiments of *CGM*, *CGS* and *gSpan* have been performed on a 2.1 GHz Intel (R) Core (TM) Duo PC with 1 GB RAM and Windows 7 operating system, using C/C++programming language. As real-life data we have used graph dataset provided in Yan (2011), which provides information on the biological activities of small molecules, containing the bioassay records for anti-cancer screen tests with different cancer cell lines found from Pubchem (2011), whereas Yan (2011) provides its equivalent graph formatted dataset. Each dataset belongs to a certain type of cancer screen with the outcome active or inactive. From these screen tests, we have collected two graph datasets with active labels among the eleven datasets with active and inactive labels. Two real-life datasets are MOLT-4 having 39765 number of graphs about Leukemia and NCI-H23 with 40353 number of graphs about Non-Small Cell Lung.

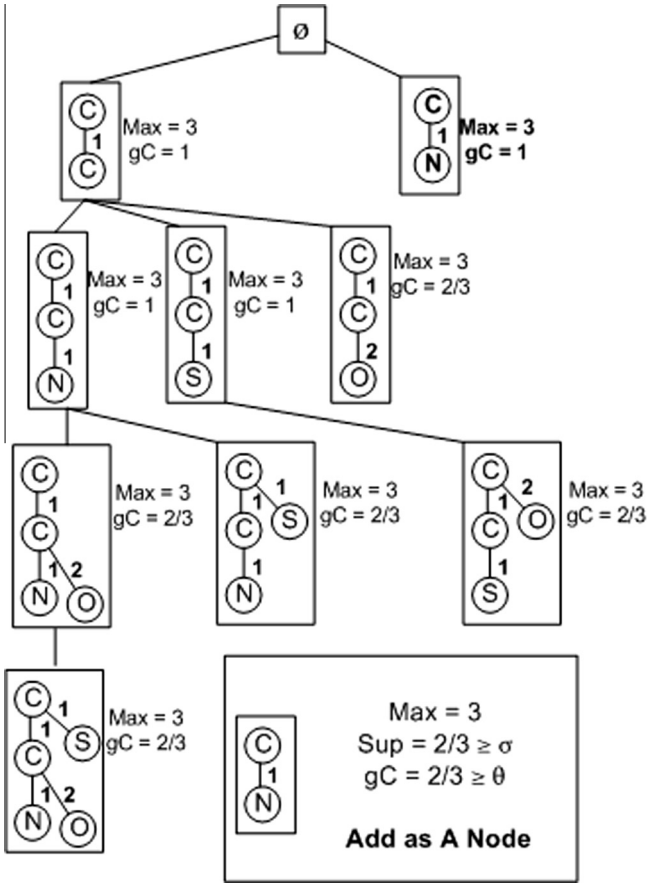


Fig. 16. *gConfidence* calculation and adding C_2 in *gConfidence Tree*.

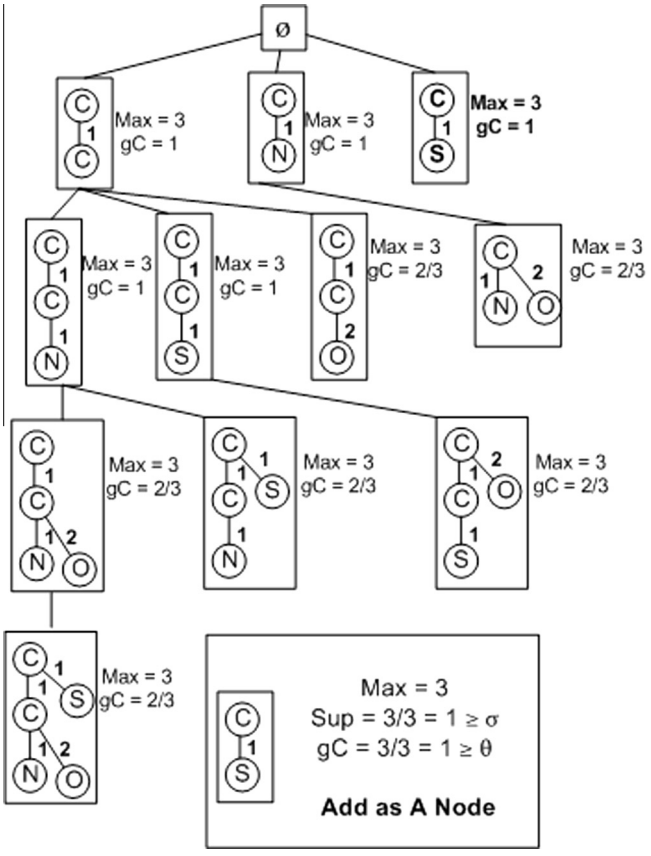


Fig. 18. *gConfidence* calculation and adding C_3 in *gConfidence Tree*.

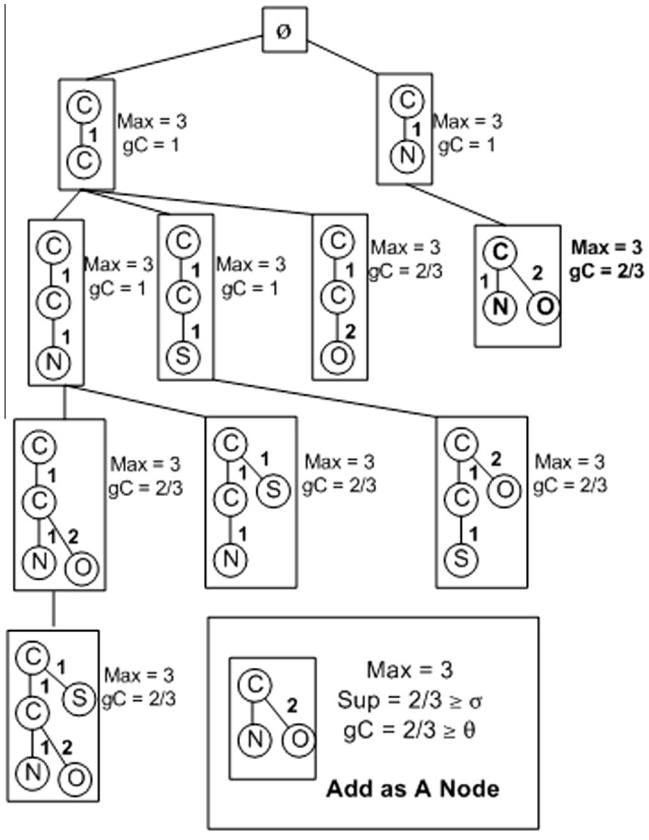


Fig. 17. *gConfidence* calculation and adding $C_{2.5}$ in *gConfidence Tree*.

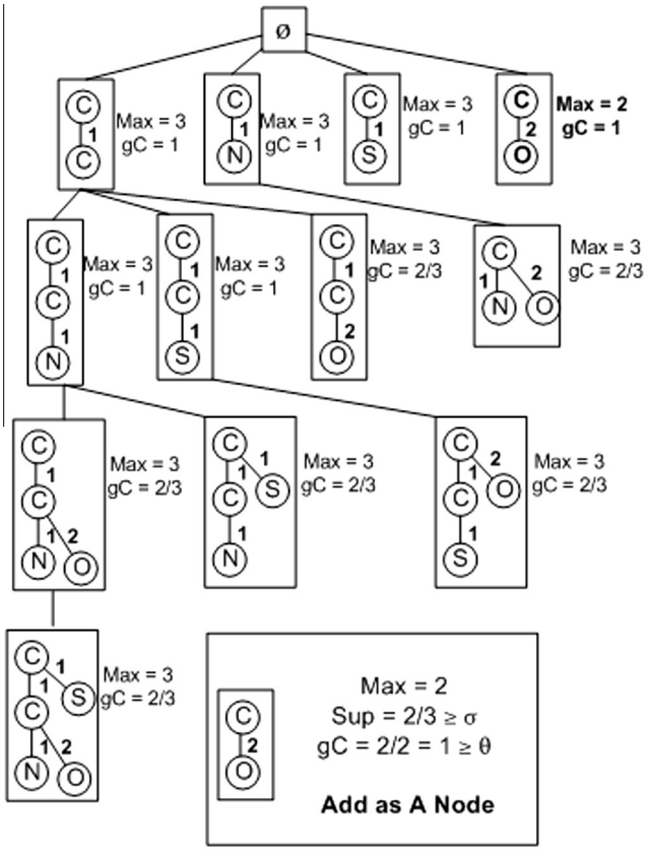


Fig. 19. *gConfidence* calculation and adding C_4 in *gConfidence Tree*.

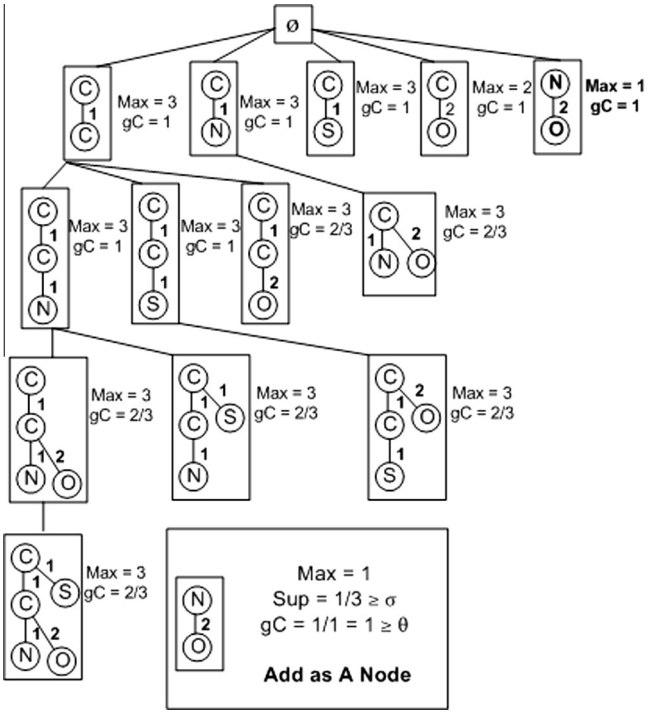


Fig. 20. *gConfidence* calculation and adding C_5 in *gConfidence Tree*.

On an average, the real datasets contain about 40 K graphs with average 25 nodes and 30 edges along with an average of 17 distinct labels for vertices and edges.

The synthetic graph generator can generate random graphs based on user specified number of vertices, edges and maximum number of supported labels. The synthetic dataset can be described by four parameters:

- $|D|$, the total number of graphs in the dataset.
- $|N|$, the number of distinct labels for vertices and edges in the dataset.
- $|T|$, the average size of the graphs in terms of edges.
- $|V|$, the average size of the graphs in terms of vertices.

Therefore, a dataset with 10 K graphs in total with 40 number of distinct labels along with 30 edges and 50 vertices can be described as *D*10*KN*40*T*30*V*50. For assessing the performance of our scheme with respect to various parameters, we have kept the support threshold, σ fixed at 5% and varied the correlation threshold, θ from 45% to 85% unless stated otherwise. In some cases, we have varied the size of database by adding or removing transaction graphs randomly from the actual database.

4.1. Performance analysis

In Fig. 23, we have shown the performance of our proposed algorithm with respect to *Processing Time vs. Data Density with varying Correlation Threshold*, when run on the MOLT-4 graph database. There were almost 40 K graphs initially and on an average 30 nodes with 21 labels and 35 number of edges. Since we are assessing the performance of our algorithm against graph density we have chosen such dense graph database. We have varied the size of the graph database by incrementally adding graphs with an initial choice of 15 K graphs. It can be noticed that maximum 150 s were needed and minimum required time was 25 s in mining such database with any confidence threshold within the specified range.

In Fig. 24, we have shown the performance of our proposed algorithm for *Scalability w.r.t. Time with varying Data Size*, when run on the NCI-H23 graph database. The database contains more than 40 K graphs initially and on an average 20 nodes with 14 labels and average 21 number of edges. Since we are assessing the scalability property, we have chosen such dense graph among the 11 given real-life database and from the real-life data we have initially taken first 20 K graphs and added every time 5 K graphs with the initial choice. We have found that 50 to 100 s are required in mining such database with any confidence threshold within the specified range.

For assessing our algorithm's *scalability w.r.t. memory consumption with varying Data Size*, in Fig. 25, we have used same graph setting as we have used earlier to verify scalability *w.r.t.* processing time. We found that our algorithm is scalable enough even in denser graph oriented database with the assurance that it never requires more than 40 MB of memory space.

Since the real graph datasets are smaller in size and have lower density, we have used our synthetic graph data generator to generate larger and denser graph dataset. The synthetic graph generator can generate random graphs based on user specified number of vertices, edges and maximum number of supported labels.

In Fig. 26, we have shown the performance of our proposed algorithm in terms of *Processing time vs. Data Density with varying Correlation Threshold*, when run on another synthetic graph database. There were initially 200 K graphs with 50 number of nodes in each graph, having 30 labels in average and number of edge was at most 80. Since we are assessing the performance of our algorithms against graph density, we have chosen such dense graph database. We have varied the size of the graph database in a similar way of MOLT-4. The remarkable fact was that minimum required time was 100 s and except some exceptional cases most of the time processing completes within 1000 s in mining such database with any confidence threshold within the specified range.

In Fig. 27, we have shown the performance of our proposed algorithm in terms of *Scalability w.r.t. Time with varying Data Size* when run on a synthetic graph database, where each graph contains average 40 number of edges, 30 number of nodes with 20 number of labels for nodes and edges, moreover, the graph database contains about 200 K graphs. For assessing scalability issue on synthetic data we have initially taken first 60 K graphs and added every time 30 K graphs with initial choice. The remarkable fact was that 300 to 1200 s are required in mining such database with any confidence threshold within range specified earlier.

To assess our algorithm's *Scalability w.r.t. Memory Consumption with Varying Data Size* for synthetic dataset, we have used same graph setting as used earlier in verifying scalability *w.r.t.* processing time. Our algorithm proved itself scalable in such a denser and larger graph database. Moreover, it can be noted that the algorithm never requires more than 170 MB. Fig. 28 shows the performance graphically.

4.2. Performance comparison with existing algorithms

Our proposed algorithm was found efficient enough in mining inherently correlated graphs and subgraphs within a graph database while comparing it with the existing correlation search algorithm, *CGS*. In fact, *CGS* is a correlation searching algorithm and *CGM* is a correlation mining algorithm as well as *CGM* performs correlation search for all possible subgraphs of a graph database. Hence, the outcome of *CGS* is inclusive within the outcome of *CGM* irrespective of the measure used in the approaches, which is described in Section 3.2.3. Comparison with *gSpan* is performed to illustrate the effectiveness of *CGM* in filtering out less interesting graph patterns.

Input: Set of Graphs: GS, Support threshold: σ, Correlation threshold: θ
Output: Set of frequent and correlated graphs: S

```
1  begin
2    Sort labels of the vertices, lb(v) where v ∈ V in GS, by the frequency.
3    Sort the label of edges, lb(e) where e ∈ E in GS, by the frequency.
4    foreach vertex v ∈ V do
5      if support(lb(v)) < σ then
6        remove v from V
7      end
8    end
9    foreach edge e ∈ E do
10     if support(lb(e)) < σ then
11       remove e from E
12     end
13   end
14   Relabel remaining vertices in descending order of frequency
15   Relabel remaining edges in descending order of frequency
16   Construct a GLOBAL EDGE MATRIX
17   Construct single edge graph set by frequent edges, S1
18   Sort S1 in Descending order of frequency and DFS lexicographic order
19   Put S1 in S as Frequent and Correlated subgraph set
20   foreach single edge graph g ∈ S1 do
21     Initialize s with e where e ∈ E(g)
22     Set s.GS = {g' | ∀g' ∈ GS, e ∈ E(g')}
23     Set MAX = frequency(g)
24     Call gConfMining(GS, S, s, MAX, S1)
25     Remove e from GS
26   end
27 end

28 Procedure gConfMining(GS, S, s, MAX, S1)
29 begin
30   if s, the DFS Code for a subgraph, is not minimum then
31     return
32   end
33   else
34     Set gC = supp(s)/MAX
35     if gC < θ or supp(s) < σ then
36       return
37     end
38     else
39       set s.gC = gC
40       Include s in S
41       Generate all potential children of s with one edge growth,
42       (u, v) ∈ E({g | g ∈ S1}) and u, v ∈ V(s(g)) or u ∈ V(s(g)), v ∉ V(s(g)), v ∈ V({g | g ∈ s.GS})
43       foreach c ∈ children(s) do
44         Call gConfMining(GS, S, c, MAX, S1)
45       end
46     end
47   end
48 end
```

Fig. 21. The Correlated Graph Mining (*CGM*) Algorithm.

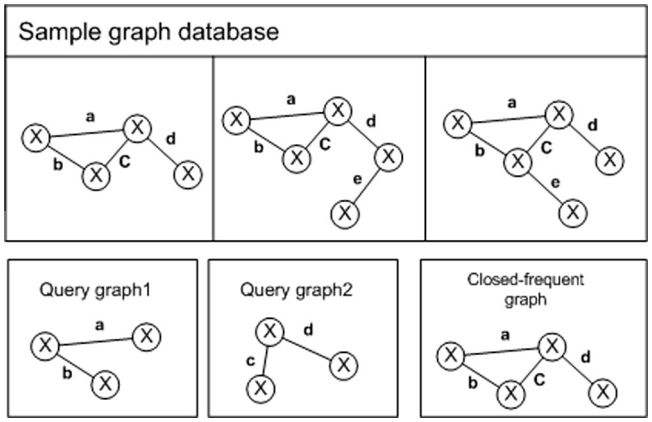


Fig. 22. Efficiency of *CGM* over *CGS*.

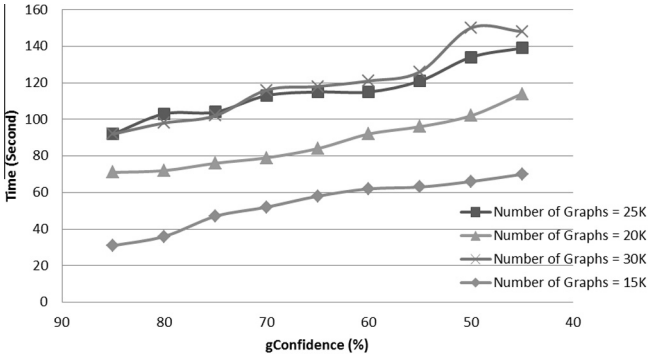


Fig. 23. Processing time *w.r.t.* Graph Density with various threshold for real-life data.

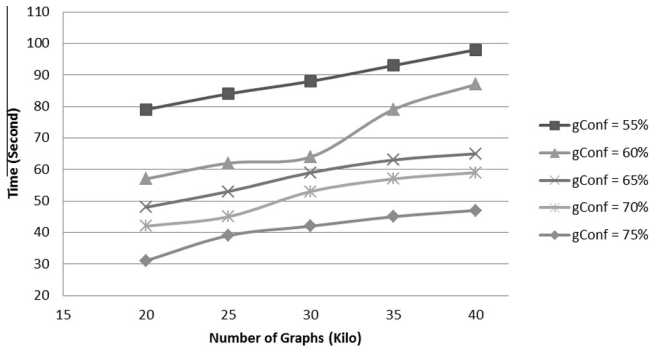


Fig. 24. Scalability of *CGM w.r.t.* processing time for real-life data.

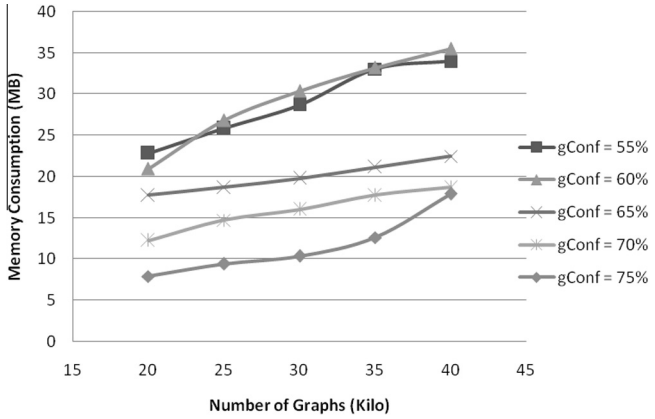


Fig. 25. Scalability of *CGM w.r.t.* memory consumption for real-life data.

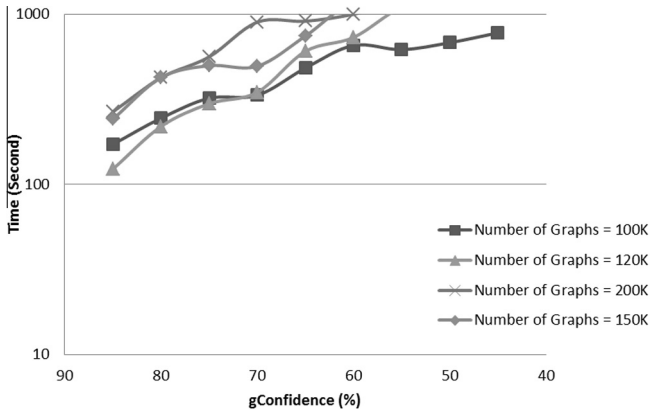


Fig. 26. Processing time *w.r.t.* Graph Density on synthetic data (D200KN30T80V50).

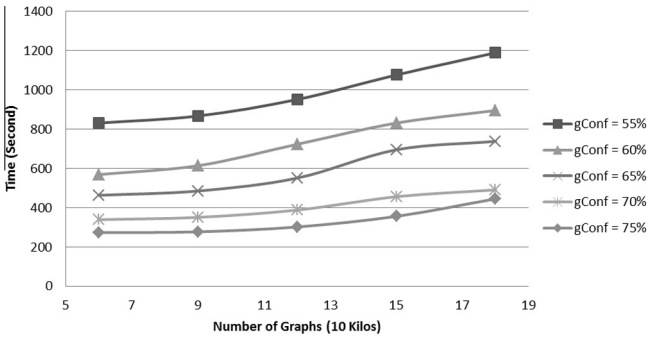


Fig. 27. Scalability of *CGM w.r.t.* processing time for synthetic data (D200KN20T40V30).

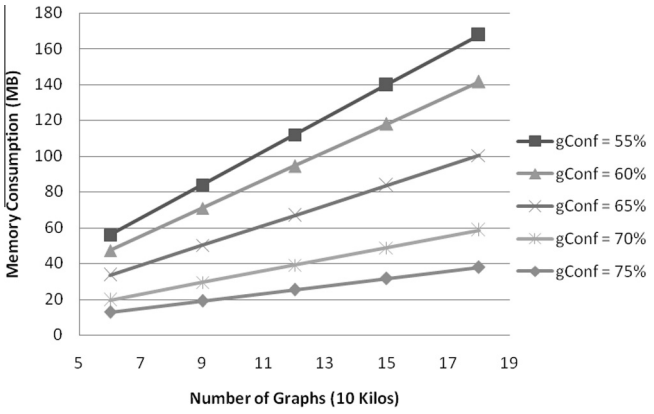


Fig. 28. Scalability of *CGM w.r.t.* memory consumption for synthetic data (D200KN20T40V30).

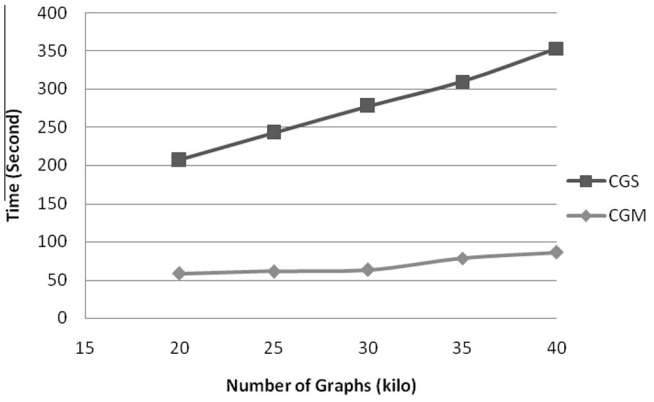


Fig. 29. Performance comparison with *CGS* on real-life dataset (NCI-H23).

For comparing the performance, once again we have used the denser real-life and synthetic dataset, which we have used in the scalability assessment earlier. The performance comparison with *CGS* algorithm can be found in Figs. 29 and 30 for real-life and synthetic dataset respectively, where a significant performance gain can be observed. It should be noted that in designing our version of *CGS* for mining correlation among all the graphs and subgraphs, we have first performed *gSpan* to search for frequent subgraphs within the database. Then we have calculated correlation for each subgraph with all other remaining subgraphs. However, while performing subgraph mining, we have done some optimizations in calculating *CGS*, such as, projected database calculation and subgraph list of a projected database is calculated by performing a set intersection operation which removes redundancy in subgraph mining for same subgraph/graph.

To prove the efficiency of our proposed algorithm we have compared it with the well known frequent subgraph mining algorithm, *gSpan*. As real datasets are too small in size and density, we have used the denser synthetic dataset used in the scalability assessment and the denser real-life dataset NCI-H23. We found our algorithm efficient enough in comparison with existing algorithms. The comparisons is shown in Figs. 31 and 32. Fig. 33 contains the percentage of graphs filtered by *CGM* with respect to the graphs selected by *gSpan*. As mentioned in the scenario of Section 1, *CGM* filters out un-correlated graphs that could be selected by *gSpan*.

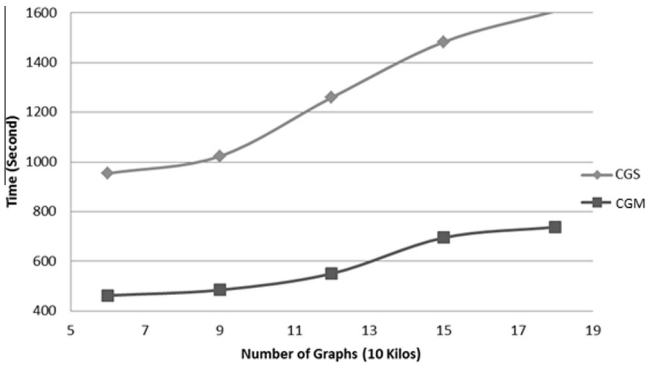


Fig. 30. Performance comparison with *CGS* on (D200KN20T40V30).

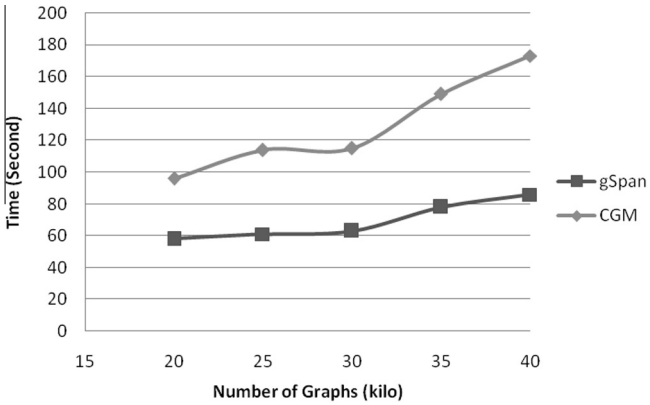


Fig. 31. Performance comparison with *gSpan* on real-life dataset (NCI-H23).

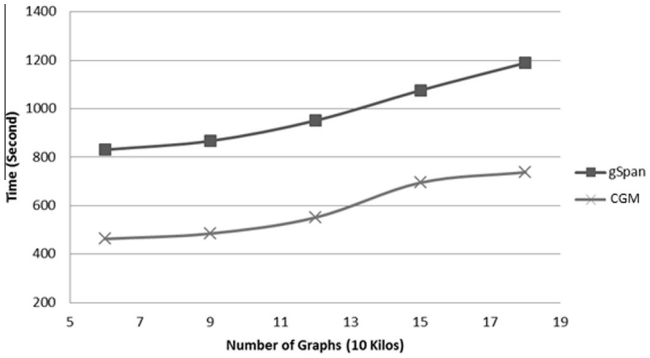


Fig. 32. Performance comparison with *gSpan* on (D200KN20T40V30).

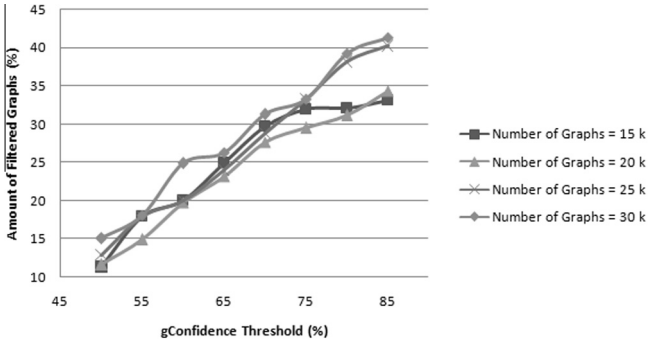


Fig. 33. Comparison based on amount of graphs filtered (%) on (MOLT-4).

Fig. 33 shows that the filtering percentage of *CGM* can be from 10% to 40% on a real-life dataset MOLT-4 for various *gConfidence* threshold.

5. Discussion

Our algorithm finds inherent correlation among the graphs in graph databases. As a result, we can apply this algorithm in those systems where correlation among various components or portions are essential. Some scenarios are following, where our algorithm can be applied and it can perform much better in mining correlated graphs than other correlated graph search algorithms.

The algorithm *CGM* and the measure *gConfidence* can be applied to solve the most discussed *Congestion Control* in computer and communication network. The communication network can be considered as a graph with the nodes representing active devices capable of sending/receiving data and edges representing the flow of data packet at any specific time. As a consequence, a set of graphs can be constructed by following the mechanism at specific time interval. Applying *CGM* and *gConfidence*, we can mine correlated graph segments among the set of graphs along with the amount of data load within the segment as amount of correlation value. Based on specific threshold of the correlation, the busiest road segments can be detected and dynamic route selection to mitigate the congestion can be performed.

Our algorithm *CGM* can be applied in any *Social Network*. As an example, suppose we need to compare the inherent correlation within a group of people, as shown in Fig. 1, where two different graphs represent two groups, with nodes of the graph representing individuals and edges represent interaction among individuals. If we apply the *gConfidence* measure on these graphs, then we find that second group of people is more inherently correlated than the first group of people. Now we can develop and offer some applications attractive for the group and also perform several analytical operations such as *cyber crime investigation* or *unfriend loosely connected friends* and/or *unnecessarily connected friends* to enhance *friend-ability* and well management of friendship as well as users can block unnecessary or harmful activities from a group of users.

We can apply our algorithm in *Automated Diagnosis System*. In such system, each graph corresponds to a disease history of a patient and labeled with a specific set of disease. Each node corresponds to a screening test or diagnosis and each edge represents internal connections between two disease based on the test. In such database, we can mine graph correlation by applying *CGM* algorithm utilizing *gConfidence* and get closely correlated graph segments. As a consequence, the newly coming suspected patients information graph can be tested for correlation. The correlation percentage will give an indication of the patient's current condition such as what can be the disease, how much the patient is attacked or what is the level of sickness or influence of disease as well as what can be the cure for the patient can be determined by the automated system.

Our proposed algorithm can be applied in *Automated Traffic and Congestion Control System*. Suppose the system is a collection of areas with roads, where graphs can be constructed from the available data, found in every specific time interval, where each node is a specific area and each edge represents road being used by a certain amount of vehicles. Correlation threshold specified by the system experts can be used to find the busiest road segment by utilizing our algorithm *CGM* (here correlation resembles the state of road segments being busy at a specific time period). Then based on the traffic load, the system will dynamically suggest for alternate ways for the vehicles to balance the load of traffic on the roads resulting in a lesser traffic jam.

Furthermore, the proposed algorithm can be fit in most evolving graph mining field, that is, graph clustering along with the machine learning problem in graph context, that is, graph classification can be performed most effectively by the proposed measure and the proposed algorithm. As an example, the various correlation threshold could be used for classification boundary and the set of graphs can be classified effectively by using the correlation measure. Accordingly, a set of graphs can be divided into various clusters by using the correlation value as a similarity measure.

6. Conclusions

Mining frequent patterns or sub-patterns with larger support threshold could miss some interesting patterns. At the same time, if the threshold considered is small enough to capture such rare but interesting items could generate lots of spurious patterns. Therefore, correlation analysis is used in association of frequent pattern mining for mining frequent-interesting patterns from a collection of patterns, but correlation searching is a challenging task. In graph databases, correlation searching is more challenging due to the fact that searching frequent subgraphs faces the graph isomorphism problem, which is NP-complete. Nonetheless, there are several correlation search algorithms with limited facilities and their specific limitations.

Therefore, we have proposed a new measure *gConfidence*, which can capture more interesting inherent correlation. The measure *gConfidence* has the downward closure property, which helps in pruning descendants of a non-correlated candidate and discover more meaningful sub-graphs. However, there seems to be no single correlation measure that works well for all cases. There are lots of interestingness measures in the literature of data and graph mining as discussed in Han and Kamber (2000) and Tan et al. (2002), unfortunately most of the measures do not have null-invariance property. Moreover, our proposed *CGM* algorithm works in a *DFS* manner and constructs a tree-like search space named *gConfidence Tree* and hierarchically mines the correlation, which can be applied in both traditional and non-traditional domains. We have conducted extensive performance analysis of *CGM* and found it scalable and fast enough which outperforms existing works in correlation search.

References

Agrawal, R., & Srikant, R., 1994. Fast algorithms for mining association rules in large databases. In *VLDB* (pp. 487–499).

Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., & Choi, H.-J. (2011). A framework for mining interesting high utility patterns with a strong frequency affinity. *Information Sciences, 181*(21), 4878–4894.

Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., & Choi, H.-J. (2012). Interactive mining of high utility patterns over data streams. *Expert Systems Applications, 39*(15), 11979–11991.

Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., & Lee, Y.-K. (2009). Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Transactions on Knowledge and Data Engineering, 21*(12), 1708–1721.

Chen, C. -W. K., & Yun, D. Y. Y., 2003. Discovering process models from execution history by graph matching. In *IDEAL* (pp. 887–892).

Chittimoori, R. N., Holder, L. B., & Cook, D. J., 1999. Applying the subdue substructure discovery system to the chemical toxicity domain. In *FLAIRS Conference* (pp. 90–94).

Cicirello, V. A., 1999. *Intelligent retrieval of solid models* (pp. 63–92). Drexel University.

Dehaspe, L., Toivonen, H., & King, R. D., 1998. Finding frequent substructures in chemical compounds. In *KDD* (pp. 30–36).

Dupplaw, D., & Lewis, P. H., 2000. Content-based image retrieval with scale-space object trees. In *Storage and retrieval for media databases* (pp. 253–261).

Gudes, E., Shimony, S. E., & Vanetik, N. (2006). Discovering frequent graph patterns using disjoint paths. *IEEE Transactions on Knowledge and Data Engineering, 18*(11), 1441–1456.

Han, J., Pei, J., & Yin, Y., 2000. Mining frequent patterns without candidate generation. In *SIGMOD Conference* (pp. 1–12).

Han, J., & Kamber, M. (2000). *Data mining: Concepts and techniques*. Morgan Kaufmann.

He, H., & Singh, A. K., 2006. Closure-tree: An index structure for graph queries. In *ICDE* (p. 38).

Holder, L. B., Cook, D. J., & Djoko, S., 1994. Substucture discovery in the subdue system. In *KDD workshop* (pp. 169–180).

Hu, Y.-H., Huang, T. C. K., & Kao, Y.-H. (2013). Knowledge discovery of weighted rfm sequential patterns from customer sequence databases. *Journal of Systems and Software, 86*(3), 779–788.

Inokuchi, A., Washio, T., & Motoda, H., 2000. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD* (pp. 13–23).

Inokuchi, A., Washio, T., & Motoda, H. (2005). A general framework for mining frequent subgraphs from labeled graphs. *Fundamental Information, 66*(1–2), 53–82.

Ke, Y., Cheng, J., & Ng, W. (2008). Efficient correlation search from graph databases. *IEEE Transactions on Knowledge and Data Engineering, 20*(12), 1601–1615.

Klviinen, H., & Oja, E., 1990. Comparisons of attributed graph matching algorithms for computer vision. In *Finnish Artificial Intelligence Symposium* (pp. 354–368).

Kramer, S., Pfahringer, B., & Helma, C., 1997. Mining for causes of cancer: Machine learning experiments at various levels of detail. In *KDD* (pp. 223–226).

Kuramochi, M., & Karypis, G., 2001. Frequent subgraph discovery. In *ICDM* (pp. 313–320).

Lahiri, M., & Berger-Wolf, T. Y., 2008. Mining periodic behavior in dynamic social networks. In *ICDM* (pp. 373–382).

Lahiri, M., & Berger-Wolf, T. Y. (2010). Periodic subgraph mining in dynamic networks. *Knowledge Information Systems, 24*(3), 467–497.

Lee, Y. -K., Kim, W. -Y., Cai, Y. D., & Han, J., 2003. Comine: Efficient mining of correlated patterns. In *ICDM* (pp. 581–584).

Li, J., Liu, Y., & Gao, H. (2011). Efficient algorithms for summarizing graph patterns. *IEEE Transactions on Knowledge and Data Engineering, 23*(9), 1388–1405.

Nishi, M. A., Ahmed, C. F., Samiullah, M., & Jeong, B.-S. (2013). Effective periodic pattern mining in time series databases. *Expert Systems Applications, 40*(8), 3015–3027.

Nori, F., Deypir, M., & Sadreddini, M. H. (2013). A sliding window based algorithm for frequent closed itemset mining over data streams. *Journal of Systems and Software, 86*(3), 615–623.

Omiecinski, E. (2003). Alternative interest measures for mining associations in databases. *IEEE Transactions on Knowledge and Data Engineering, 15*(1), 57–69.

Piriyakumar, D. A. L., Murthy, C. S. R., & Levi, P., 1998. A new a* based optimal task scheduling in heterogeneous multiprocessor systems applied to computer vision. In *HPCN*, Europe (pp. 315–323).

Pubchem web site for information on biological activities of small molecules, 2011. <http://pubchem.ncbi.nlm.nih.gov>.

Raymond, J. W., Gardiner, E. J., & Willett, P. (2002). Rascal: Calculation of graph similarity using maximum common edge subgraphs. *Comput. J., 45*(6), 631–644.

Samiullah, M., Ahmed, C. F., Nishi, M. A., Fariha, A., Abdullah, S. M., & Islam, M. R., 2013. Correlation mining in graph databases with a new measure. In *APWeb* (pp. 88–95).

Srinivasan, A., King, R. D., Muggleton, S., & Sternberg, M. J. E., 1997. Carcinogenesis predictions using ilp. In *ILP* (pp. 273–287).

Srinivasan, A., King, R. D., Muggleton, S., & Sternberg, M. J. E., 1997. The predictive toxicology evaluation challenge. In *IJCAI, Vol. 1* (pp. 4–9).

Tanbeer, S. K., Ahmed, C. F., Jeong, B.-S., & Lee, Y.-K. (2009). Efficient single-pass frequent pattern mining using a prefix-tree. *Information Sciences, 179*(5), 559–583.

Tan, P. -N., Kumar, V., & Srivastava, J., 2002. Selecting the right interestingness measure for association patterns. In *KDD* (pp. 32–41).

Williams, D. W., Huan, J., & Wang, W., 2007. Graph database indexing using structured graph decomposition. In *ICDE* (pp. 976–985).

Wu, T., Chen, Y., & Han, J., 2007. Association mining in large databases: A re-examination of its measures. In *PKDD* (pp. 621–628).

Xiong, H., Tan, P. -N., & Kumar, V., 2003. Mining strong affinity association patterns in data sets with skewed support distribution. In *ICDM* (pp. 387–394).

Yan, X., 2011. Graph dataset provided by xifeng yan. <http://www.cs.ucsb.edu/xyan/dataset.htm>.

Yan, X., & Han, J., 2002. gspan: Graph-based substructure pattern mining. In *ICDM* (pp. 721–724).

Yan, X., & Han, J., 2003. Closegraph: Mining closed frequent graph patterns. In *KDD* (pp. 286–295).

Yan, X., Zhu, F., Yu, P. S., & Han, J. (2006). Feature-based similarity search in graph structures. *ACM Transactions on Database Systems, 31*(4), 1418–1453.

Yoshida, K., & Motoda, H. (1995). Clip: Concept learning from inference patterns. *Artificial Intelligence, 75*(1), 63–92.