

Top-K Correlation Sub-graph Search in Graph Databases

Lei Zou¹, Lei Chen², and Yansheng Lu¹

¹ Huazhong University of Science and Technology, Wuhan, China,
{zoulei, lys}@mail.hust.edu.cn

² Hong Kong of Science and Technology, Hong Kong, China,
leichen@cse.ust.hk

Abstract. Recently, due to its wide applications, (similar) subgraph search has attracted a lot of attentions from database and data mining community, such as [13, 18, 19, 5]. In [8], Ke et al. first proposed correlation sub-graph search problem (CGSearch for short) to capture the underlying dependency between sub-graphs in a graph database, that is CGS algorithm. However, CGS algorithm requires the specification of a minimum correlation threshold θ to perform computation. In practice, it may not be trivial for users to provide an appropriate threshold θ , since different graph databases typically have different characteristics. Therefore, we propose an alternative mining task: *top-K correlation sub-graph search* (TOP-CGSearch for short). The new problem itself does not require setting a correlation threshold, which leads the previous proposed CGS algorithm inefficient if we apply it directly to TOP-CGSearch problem. To conduct TOP-CGSearch efficiently, we develop a *pattern-growth* algorithm (that is PG-search algorithm) and utilize graph indexing methods to speed up the mining task. Extensive experiment results evaluate the efficiency of our methods.

1 Introduction

As a popular data structure, graphs have been used to model many complex data objects and their relationships, such as, chemical compounds [14], entities in images [12] and social networks [3]. Recently, the problems related to graph database have attracted much attentions from database and data mining community, such as frequent sub-graph mining [7, 17], (similar) sub-graph search [13, 18, 19, 5]. On the other hand, correlation mining [15] is always used to discovery the underlying dependency between objects, such as market-basket databases [2, 10], multimedia databases [11] and so on. Ke et al. first propose correlation sub-graph search in graph databases [8]. Formally, correlation sub-graph search (CGSearch for short) is defined as follows [8]:

Given a query graph Q and a minimum correlation threshold θ , we need to report all sub-graphs S_i in graph database, where the Pearson’s correlation coefficient between Q and S_i (that is $\phi(Q, S_i)$, see Definition 4) is no less than θ .

Actually, the larger $\phi(Q, S_i)$ is, the more often sub-graphs Q and S_i appear together in the same data graphs. Take molecule databases for example. If two sub-structures S_1 and S_2 often appear together in the same molecules, S_1 and S_2 may share some similar chemical properties. In Fig. 1, we have 5 data graphs in database D and a query sub-graph Q and the threshold $\theta = 1.0$. In Fig. 1, for presentation convenience, we assume that all edges have the same label ‘x’. The number beside the vertex is vertex ID. The letter inside the vertex is vertex label. Since only $\phi(Q, S_2) = 1.0 \geq \theta$, we report S_2 in Fig. 1. $\phi(Q, S_i) = 1.0$ means S_i (or Q) always appears in the same data graph g , if Q

(or S_i) appears in data graph g . In CGSearch problem, each sub-graph S_i of data graphs in database D is a candidate, which is quite different from (similar) sub-graph search in previous work. In (similar) sub-graph search problem [13, 18], it reports all data graphs that (closely) contain query graph Q . Therefore, only data graphs are candidates. Thus, the search space of CGSearch problem is much larger than that of (similar) sub-graph search, which leads CGSearch to be a more challenging task.

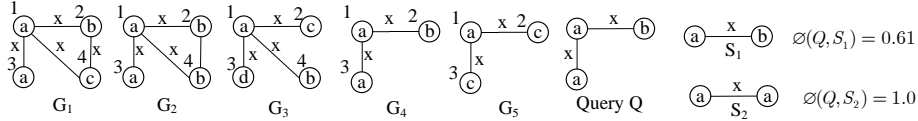


Fig. 1. A Graph Database

Furthermore, CGSearch is also different from frequent sub-graph mining. As we all know, all frequent sub-graph mining algorithms are based on “down-closure property”(that is Apriori property)[1] no matter they adopt bread-first search [9] or depth-first search [17]. However, Apriori property does not hold for $\phi(Q, S_i)$. It means that we cannot guarantee that $\phi(Q, S_i) \geq \phi(Q, S_j)$, if S_i is a sub-graph of S_j . Therefore, for CGSearch problem, we cannot employ the same pruning strategies used in frequent sub-graph mining algorithms.

In order to address CGSearch, Ke et al. propose CGS Algorithm [8], which adopts “filter-and-verification” framework to perform the search. First, according to a given minimum correlation coefficient threshold θ and a query graph Q , they derive the minimum support $lowerbound(sup(g))$. Any sub-graph g_i whose support is less than $lowerbound(sup(g))$ cannot be a correlation sub-graph w.r.t ³ query Q . Then, in the projected database (all data graphs containing query graph Q are collected to form the projected database, denoted as D_q), frequent sub-graph mining algorithms are used to find candidates g_i by setting minimum support $\frac{lowerbound(sup(g))}{sup(Q)}$ (that is *filter* process). After that, for each candidate g_i , the correlation coefficient is computed to find all true answers (that is *verification* process).

However, CGSearch in [8] requires users or applications to specify a minimum correlation threshold θ . In practice, it may not be trivial for users to provide an appropriate threshold θ , since different graph databases typically have different characteristics. Therefore, in this paper, we propose an alternative mining task: **top-K correlation sub-graph search**, *TOP-CGSearch*, which is defined:

Given a query graph Q , we report top-K sub-graphs S_i according to the Pearson’s correlation coefficient between Q and S_i (that is $\phi(Q, S_i)$).

Obviously, the above methods in CGSearch [8] cannot be directly applied to the TOP-CGSearch problem, since we do not have the threshold θ . Simply setting a large threshold and then decreasing it step by step based on the number of answers obtained from the previous step is out of practice, since we need to perform frequent sub-graph mining algorithms to obtain candidates in each iterative step. Furthermore, we also need to perform verification of CGSearch [8] in each iterative step.

³ with regard to

Can we derive top-K answers directly without performing frequent sub-graph mining algorithms for candidates and expensive verification several times? Our answer in this paper is YES. For TOP-CGSearch problem, we first derive an upper bound of $\phi(Q, S_i)$ (that is $upperbound(\phi(Q, S_i))$). Since $upperbound(\phi(Q, S_i))$ is a monotone increasing function w.r.t. $sup(S_i)$, we develop a novel **pattern-growth** algorithm, called *PG-search* algorithm to conduct effective filtering through the upper bound. During mining process, we always maintain β to be the K-largest $\phi(Q, S_i)$ by now. The algorithm reports top-K answers until all un-checked sub-graphs cannot be in top-K answers. Furthermore, in pattern-growth process, we only grow a pattern P if and only if we have checked all its parents P_i (see Lemma 2). Therefore, we significantly reduce the search space (the number of sub-graphs that need to be checked in *PG-search* algorithm). Finally, we utilize existing graph indexing techniques [13, 18] to speed up the mining task. In summary, we made the following contributions:

1. We propose a new mining task TOP-CGSearch problem. For TOP-CGSearch, we develop an efficient “pattern-growth” algorithm, called PG-search algorithm. In PG-search, we introduce two novel concepts, **Growth Element**(GE for short) and **Correct Growth** to determine the correct growth and avoid duplicate generation.
2. To speed up the mining task, we utilize graph indexing techniques to improve the performance in PG-search algorithm.
3. We conduct extensive experiments to verify the efficiency of the proposed methods. The rest of the paper is organized as follows: Section 2 discuss preliminary background. PG-search algorithm is proposed in Section 3. We evaluate our methods in experiments in Section 4. Section 5 discusses related work. Section 6 concludes this work.

2 Preliminary

2.1 Problem Definition

Definition 1. Graph Isomorphism. Assume that we have two graphs $G_1 < V_1, E_1, L_{1v}, L_{1e}, F_{1v}, F_{1e} >$ and $G_2 < V_2, E_2, L_{2v}, L_{2e}, F_{2v}, F_{2e} >$. G_1 is graph isomorphism to G_2 , if and only if there exists at least one bijective function $f : V_1 \rightarrow V_2$ such that: 1) for any edge $uv \in E_1$, there is an edge $f(u)f(v) \in E_2$; 2) $F_{1v}(u) = F_{2v}(f(u))$ and $F_{1v}(v) = F_{2v}(f(v))$; 3) $F_{1e}(uv) = F_{2e}(f(u)f(v))$.

Definition 2. Sub-graph Isomorphism. Assume that we have two graphs G' and G , if G' is graph isomorphism to at least one sub-graph of G under bijective function f , G' is sub-graph isomorphism to G under injective function f .

If a graph S is sub-graph isomorphism to another graph G , we always say that graph G contains S .

Definition 3. Graph Support. Given a graph database D with N data graphs, and a sub-graph S , there are M data graphs containing S . The support of S is denoted as $sup(S) = \frac{M}{N}$. Obviously, $0 \leq sup(S) \leq 1$.

Definition 4. Pearsons Correlation Coefficient.

Given two sub-graphs S_1 and S_2 , Pearson’s Correlation Coefficient of S_1 and S_2 , denoted as $\phi(S_1, S_2)$, is defined as:

$$\phi(S_1, S_2) = \frac{sup(S_1, S_2) - sup(S_1)sup(S_2)}{\sqrt{sup(S_1)sup(S_2)(1 - sup(S_1))(1 - sup(S_2))}} \quad (1)$$

Here, $-1 \leq \phi(S_1, S_2) \leq 1$

Given two graphs S_1 and S_2 , if $\phi(S_1, S_2) > 0$, then S_1 and S_2 are positively correlated; otherwise, they are negatively correlated. In this paper, we focus on positively correlated sub-graph search.

Definition 5. (Problem Definition) Top-K Correlation Sub-graph Search (TOP-CGSearch for short) Given a query graph Q and a graph database D , according to Definition 4, we need to find K sub-graphs S_i ($i=1\dots K$) in the graph database D , where $\phi(Q, S_i)$ are the first K largest.

Note that, in this work, we only focus on positively correlated sub-graphs, if there exist less than K positively correlated sub-graphs, we report all positively correlated sub-graphs. Now we demonstrate top- K correlation sub-graph search with a running example.

EXAMPLE 1(Running Example). Given the same graph database D with 5 data graphs and a query graph Q in Fig. 1, we want to report top-2 correlated sub-graphs in D according to Definition 5. In this example, the top-2 answers are S_1 and S_2 , where $\phi(Q, S_1) = 1.0$ and $\phi(Q, S_2) = 0.61$ respectively. For the other sub-graphs S_i , $\phi(Q, S_i) < 0.61$.

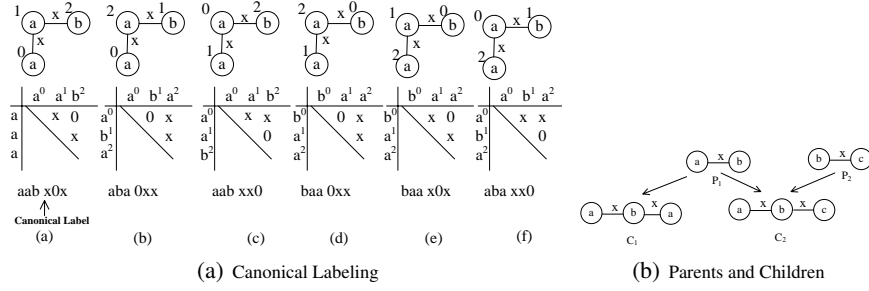
2.2 Canonical labeling

In order to determine whether two graphs are isomorphism to each other, we can assign each graph a unique code. Such a code is referred to as **canonical label** of a graph [4]. Furthermore, benefiting from canonical labels, we can define a total order for a set of graphs in a unique and deterministic way, regardless of the original vertex and edge ordering. The property will be used in Definition 8 and 12. It is known that the hardness of determining the canonical label of a graph is equivalent to determining isomorphism between graphs. So far, these two problems are not known to be either in P or in NP-complete [4]. There are many canonical labeling proposals in the literature [17]. In fact, any canonical labeling method can be used in our algorithm, which is orthogonal to our method. A simple way of defining the canonical label of a graph is to concatenate the upper-triangular entries of graph’s adjacency matrix when this matrix has been symmetrically permuted so that the obtained string is the lexicographically smallest over the strings that can be obtained from all such permutations.

Fig. 2(a) shows an example about obtaining the canonical label of a graph. In Fig. 2(a), we permute all vertex orderings. For each vertex ordering, we can obtain a code by concatenating the upper-triangular entries of the graphs adjacency matrix. Among all codes, the code in Fig. 2(a)a is the lexicographically smallest (‘0’ is smaller than all letters). Thus, in Fig. 2(a), “aab x0x” is the canonical code of the graph. Fig. 2(a)a shows the canonical form of the graph.

Definition 6. Canonical Vertex Ordering and ID. Given a graph S , some vertex ordering is called **Canonical Vertex Ordering**, if and only if this ordering leads to the canonical code of graph S . According to canonical vertex ordering, we define **Canonical Vertex ID** for each vertex in S .

The vertex ordering in Fig. 2(a)a leads to canonical code, thus, the vertex ordering in Fig. 2(a)a is called *canonical vertex ordering*. Vertex ID in Fig. 2(a)a is called *canonical vertex ID*.


Fig. 2. Canonical Labels and Pattern Growth

2.3 Pattern Growth

Definition 7. Parent and Child. Given two *connected* graphs P and C , P and C have n and $n + 1$ edges respectively. If P is a sub-graph of C , P is called a **parent** of C , and C is called a **child** of P .

For a graph, it can have more than one child or parent. For example, in Fig. 2(b), C_2 has two parents that are P_1 and P_2 . P_1 have two children that are C_1 and C_2 .

During mining process, we always “grow” a sub-graph from its parent, which is called **pattern growth**. During pattern growth, we introduce an extra edge into a parent to obtain a child. The introduced extra edge is called **Growth Element** (GE for short). The formal definition about GE is given in Section 3.1.

3 PG-search Algorithm

As mentioned in the Section 1, we can transfer a TOP-CGSearch problem into a threshold-based CGSearch. First, we set threshold θ to be a large value. According to CGS algorithm [8], if we can find the correct top-K correlation sub-graphs, then we report them and terminate the algorithm. Otherwise, we decrease θ and repeat the above process until we obtain the top-K answers. We refer this method as “Decreasing Threshold-Based CGS” in the rest of this paper. Obviously, when K is large, we have to set θ to be a small value to find the correct top-K answers, which is quite inefficient since we need to perform CGS algorithm [8] several times and there are more candidates needed to be investigated. In order to avoid this shortcoming, we propose a **pattern-growth search** algorithm (PG-search for short) in this section.

3.1 Top-K Correlation Sub-graph Query

The search space (the number of candidates) of TOP-CGSearch problem is large, since each sub-graph S_i in graph database is a candidate. Obviously, it is impossible to enumerate all sub-graphs in the database, since it will result in a combinational explosion problem. Therefore, an effective pruning strategy is needed to reduce the search space. In our algorithm, we conduct effective filtering through the upper bound. Specifically, if the largest upper bound for all un-checked sub-graphs cannot get in the top-K answers, the algorithm can stop. Generally speaking, our algorithms work as follows: We first generate all size-1 sub-graph (the size of a graph is defined in terms of number of edges, i.e. $|E|$) S_i , and insert them into heap H in non-increasing order of $upperbound(\phi(Q, S_i))$, which is the upper bound of $\phi(Q, S_i)$. The upper bound is a monotone increasing function w.r.t $sup(S_i)$. The head of the heap H is h . We set $\alpha = upperbound(\phi(Q, h))$. We compute $\phi(Q, h)$ and insert h into result set RS . β is

set to be the K th largest value in RS by now. Then, we delete the heap head h , and find all children C_i (see Definition 7) of h through pattern growth. We insert these C_i into heap H . For the heap head h , we set $\alpha = upperbound(\phi(Q, h))$. We also compute $\phi(Q, h)$, and then insert h into result set RS . The above processes are iterated until $\beta \geq \alpha$. At last, we report top- K answers in RS .

In order to implement the above algorithm, we have to solve the following technical challenges: 1) how to define $upperbound(\phi(Q, S_i))$; 2) how to find all h 's children; 3) Since a child may be generated from different parents (see Fig. 2(b)), how to avoid generating duplicate patterns; and 4) how to reduce the search space as much as possible.

Lemma 1. *Given a query graph Q and a sub-graph S_i in the graph database, the upper bound of $\phi(Q, S_i)$ is denoted as:*

$$\phi(Q, S_i) \leq upperbound(\phi(Q, S_i)) = \sqrt{\frac{1 - sup(Q)}{sup(Q)}} * \sqrt{\frac{sup(S_i)}{1 - sup(S_i)}} \quad (2)$$

Proof. Obviously, $sup(Q, S_i) \leq sup(S_i)$, thus:

$$\begin{aligned} \phi(Q, S_i) &= \frac{sup(Q, S_i) - sup(Q)sup(S_i)}{\sqrt{sup(Q)sup(S_i)(1 - sup(Q))(1 - sup(S_i))}} \\ &\leq \frac{sup(S_i) - sup(Q)sup(S_i)}{\sqrt{sup(Q)sup(S_i)(1 - sup(Q))(1 - sup(S_i))}} \\ &= \frac{sup(S_i)(1 - sup(Q))}{\sqrt{sup(Q)sup(S_i)(1 - sup(Q))(1 - sup(S_i))}} \\ &= \sqrt{\frac{1 - sup(Q)}{sup(Q)}} * \sqrt{\frac{sup(S_i)}{1 - sup(S_i)}} \end{aligned}$$

Theorem 1. *Given a query graph Q and a sub-graph S_i in a graph database D , the upper bound of $\phi(Q, S_i)$ (that is $upperbound(\phi(Q, S_i))$) is a monotone increasing function w.r.t $sup(S_i)$.*

Proof. Given two sub-graphs S_1 and S_2 , where $sup(S_1) > sup(S_2)$, we can know that the following formulation holds.

$$\frac{upperbound(\phi(Q, S_1))}{upperbound(\phi(Q, S_2))} = \sqrt{\frac{sup(S_1)}{sup(S_2)}} \sqrt{\frac{1 - sup(S_2)}{1 - sup(S_1)}} > 1$$

Therefore, Theorem 1 holds.

Property 1. (Apriori Property) Given two sub-graph S_1 and S_2 , if S_1 is a parent of S_2 (see Definition 7), then $Sup(S_2) < Sup(S_1)$.

Lemma 2. *Assume that β^* is the K -largest $\phi(Q, S_i)$ in the final results. Given a sub-graph S with $|S| > 1$, the necessary condition for $upperbound(\phi(Q, S)) > \beta^*$ is that, for all of its parents P_i , $upperbound(\phi(Q, P_i)) > \beta^*$.*

Proof. According to Property 1, we know that $Sup(S) \leq Sup(P_i)$. Since $upperbound(\phi(Q, S))$ is a monotone increasing function w.r.t $sup(S)$, it is clear that $upperbound(\phi(Q, P_i)) \geq upperbound(\phi(Q, S)) > \beta^*$. Therefore, Lemma 2 holds.

According to Lemma 2, we only need to check a sub-graph, if and only if we have checked all its parents P_i . It means that we can adopt ‘‘pattern-growth’’ framework in the mining process, that is to generate and check a size- $(n+1)$ sub-graph S from its parent P_i after we have verified all S 's parents. Furthermore, according to monotone increasing property of $upperbound(\phi(Q, S))$, we always ‘‘first’’ grow a sub-graph P with the highest support. We refer the method as ‘‘best-first’’. In order to combine ‘‘pattern-growth’’ and ‘‘best-first’’ in the mining process, we define Total Order in the following definition.

Definition 8. (Total Order) Given two sub-graphs S_1 and S_2 in graph database D , where S_1 is not isomorphism to S_2 , we can say that $S_1 < S_2$ if either of the following conditions holds:

- 1) $sup(S_1) > sup(S_2)$;
- 2) $sup(S_1) == sup(S_2)$ AND $size(S_1) < size(S_2)$, where $size(S_1)$ denotes the number of edges in S_1 .
- 3) $sup(S_1) == sup(S_2)$ AND $size(S_1) == size(S_2)$ AND $label(S_1) < label(S_2)$, where $label(S_1)$ and $label(S_2)$ are the canonical labeling of sub-graphs S_1 and S_2 respectively.

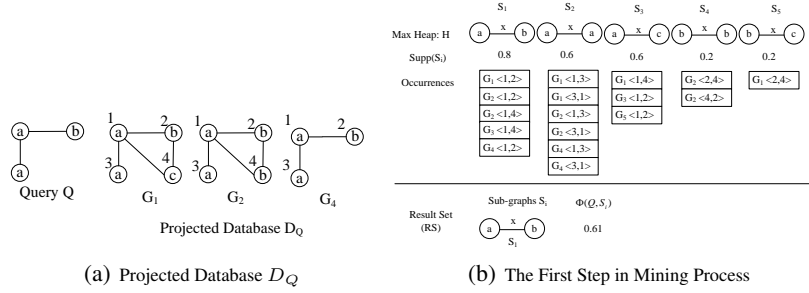


Fig. 3. Projected Database and The First Step

Furthermore, as mentioned in Section 2, we only focus on searching positively correlated graphs in this paper. Thus, according to Definition 4, if $\phi(Q, S_i) > 0$, then $sup(Q, S_i) > 0$, which indicates that Q and S_i must appear together in at least one graph in D . Therefore, we only need to conduct the correlation search over all data graphs that contain query graph Q , which is called *projected graph database* and denoted as D_Q . A projected database of the running example is given in Fig. 3(a). Similar with [8], we focus the mining task on the projected database D_Q . It means that we always generate candidate sub-graphs from the projected graph database through pattern growth. Since $|D_Q| < |D|$, it is efficient to perform mining task on D_Q .

At the beginning of PG-Search algorithm, we enumerate all size-1 sub-graph S_i in the projected graph database and insert them into the heap H in increasing order of *total order*. We also record their occurrences in each data graph. Fig. 3(b) shows all size-1 sub-graph S_i in the projected graph database in the running example. “ $G_1 < 1, 2 >$ ” in Fig. 3(b) shows that there is one occurrence of S_1 in data graph G_1 and the corresponding vertex IDs are 1 and 2 in graph G_1 (see Fig. 1). In Fig. 3(b), the head of the heap H is sub-graph S_1 . Therefore, we first grow S_1 , since it has the highest support. For each occurrence of S_1 in the projected database D_Q , we find all **Growth Elements** (GE for short) around the occurrence.

Definition 9. Growth Element. Given a connected graph C having n edges and another connected graph P having $n - 1$ edges, P is a sub-graph of C (namely, C is a child of P). For some occurrence O_i of P in C , we use $(C \setminus O_i)$ to represent the edge in C that is not in O_i . $(C \setminus O_i)$ is called **Growth Element** (GE for short) w.r.t O_i . Usually, GE is represented as a five-tuple $\langle (vid1, vlabel1), (elabel), (vid2, vlabel2) \rangle$, where $vid1$ and $vid2$ are **canonical vertex IDs** (defined in Definition 6) of P ($vid1 < vid2$). $vlabel1$ and $vlabel2$ are corresponding vertex labels, and $elabel$ is the edge label of GE. Notice that $vid2$ can also be “*”, and “*” means the additional vertex for P .

To facilitate understanding *GE*, we illustrate it with an example. Given a sub-graph S_1 in Fig. 4(a), there is one occurrence of S_1 in data graph G_1 , which is denoted as the shaded area in Fig. 4(a). For the occurrence, there are three GEs that are also shown in Fig. 4(a). Growing S_1 (parent) through each GE, we will obtain another size-2 sub-graph (child) in Fig. 4(b). The

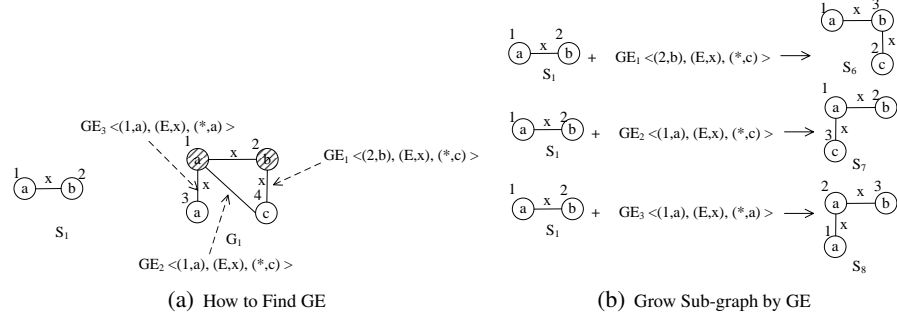


Fig. 4. Finding GE and Grow Sub-graph by GE

numbers beside vertexes in each sub-graph S_i are *canonical vertex IDs* in S_i . The numbers beside vertexes in data graph G are just vertex IDs, which are assigned arbitrarily. Notice that, canonical vertex IDs in a parent may not be preserved in its child. For example, in Fig. 4(b), the canonical vertex IDs in S_1 are different from that in S_6 . Given a size- $(n+1)$ sub-graph S_i (child), it can be obtained by growing from different size- n sub-graphs (parents). For example, in Fig. 5(a), we can obtain sub-graph S_6 by growing S_1 or S_5 . Obviously, duplicate patterns will decrease the performance of mining algorithm. Therefore, we propose the following definition about *correct growth* to avoid the possible duplicate generation.

Definition 10. Correct Growth. Given a size- $(n+1)$ sub-graph C (child), it can be obtained by growing some size- n sub-graphs P_i (parent), $i=1\dots m$. Among all growthes, the growth from P_j to C is called **correct growth**, where P_j is the largest one among all parents according to Total Order in Definition 8. Otherwise, the growth is incorrect one.

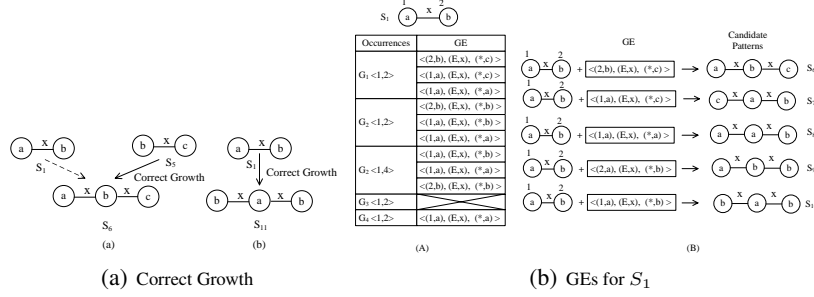
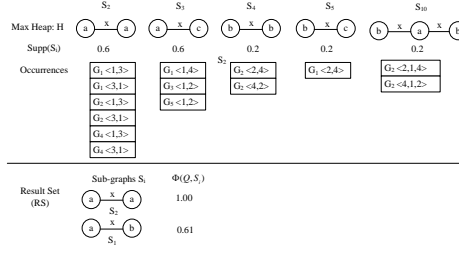
For example, in Fig. 5(a), S_9 has two parents, which are S_1 and S_5 . Since S_5 is larger than S_1 (see Definition 8), the growth from S_5 to S_6 is a correct growth, and the growth from S_1 to S_6 is an incorrect growth. Obviously, according to Definition 8, we can determine whether the growth from P to C is correct or incorrect. We will propose another efficient method to determine the correct growth in Section 3.2.

Now, we illustrate the mining process with the running example. In the running example, we always maintain β to be the K largest $\phi(Q, S_i)$ by now ($K = 2$ in the running example). Initially, we set $\beta = -\infty$. First, we conduct sub-graph query to obtain projected graph database D_Q , which is shown in Fig. 3(a). Then, we find all size-1 sub-graphs (having one edge) in the projected database D_Q . In Fig. 3(b), we insert them into heap H in increasing order according to Total Order in Definition 8. We also record all their occurrences in the graph database. Since S_1 is the head of heap H , we compute $\alpha = \text{upperbound}(\phi(Q, S_1)) = 1.63$ and $\phi(Q, S_1) = 0.61$. We inset S_1 into result set RS . Now, $\beta = -\infty < \alpha$, the algorithm continues.

We find all growth elements (GEs for short) around S_1 in D_Q , which are shown in Fig. 5(b). Since G_3 is not in the projected database, we do not consider the occurrences in G_3 when we find GEs. For each GE, we check whether it leads to a correct growth. In Fig. 5(b), only the last one is a correct growth. We insert the last one (that is S_{10}) into the max-heap H , as shown in Fig. 6. Now, the sub-graph S_2 is the heap head. We re-compute $\alpha = \text{upperbound}(\phi(Q, S_2)) = 1.0$, $\phi(Q, S_2) = 1.0$, and insert S_2 into RS . We update β to be Top-2 answer, that is $\beta = 0.61$. Since $\beta < \alpha$, the algorithm still continues. The above processes are iterated until $\beta \geq \alpha$. At last, we report top-2 answers in RS .

3.2 PG-search Algorithm

As discussed in Section 3.1, the heap H is ranked according to Total Order in Definition 8 (see Fig. 3(b) and 6). In Definition 8, we need $\text{sup}(S_i)$. Therefore, we maintain all occurrences of S_i


Fig. 5. Correct Growth and S_1 's Growth Elements

Fig. 6. The Second Step in Mining Process

in heap H . According to the occurrence list, it is easy to obtain $sup(S_i)$. However, it is expensive to maintain all occurrences in the whole database D , especially when $|D|$ is large. On the other hand, we need to find GE around each occurrence in the projected database D_Q (see Fig. 5(b)). Therefore, we need to maintain occurrences in the projected database. Furthermore, the projected database D_Q is always smaller than the whole database D . In order to improve the performance, in our PG-search algorithm, we only maintain the occurrence list in the projected database instead of the whole database.

However, Definition 8 needs $sup(S_i)$. Therefore, we propose to use $indexsup(S_i)$ to define Total Order instead of $sup(S_i)$, where $indexsup(S_i)$ is derived from sub-graph query technique.

Recently, a lot of graph indexing structures have been proposed in database literature [13, 18] for sub-graph query. Sub-graph query is defined as: *given a sub-graph Q , we report all data graphs containing Q as a sub-graph from the graph database*. Because sub-graph isomorphism is a NP-complete problem [4], we always employ a *filter-and-verification* framework to speed up the search process. In filtering process, we identify the candidate answers by graph indexing structures. Then, in verification process, we check each candidate by sub-graph isomorphism. Generally speaking, the filtering process is much faster than verification process.

Definition 11. Index Support. For a sub-graph S_i , index support is size of candidates in filtering process of sub-graph S_i query, which is denoted as $indexsup(S_i)$.

Actually, $sup(S_i)$ is the size of answers for sub-graph S_i query, which is obtained in verification process. According to graph indexing techniques [13, 18], we can identify the candidates without false negatives. Therefore, Lemma 3 holds.

Lemma 3. For any sub-graph S_i in the graph database, $sup(S_i) \leq indexsup(S_i)$, where $indexsup(S_i)$ is the size of candidates for sub-graph S_i query.

Furthermore, $indexsup(S_i)$ also satisfy Apriori Property. For example, in gIndex [18], if Q_1 is a sub-graph of Q_2 , the candidates for Q_2 query is a subset of candidates for Q_1 . It means the following property holds.

Property 2. (Apriori Property) Given two sub-graph S_1 and S_2 , if S_1 is a parent of S_2 (see Definition 7), then $indexsup(S_2) < indexsup(S_1)$.

Lemma 4. Given a query graph Q and a sub-graph S_i in the graph database, the upper bound of $\phi(Q, S_i)$ is denoted as:

$$\begin{aligned} \phi(Q, S_i) &\leq upperbound2(\phi(Q, S_i)) \\ &= \sqrt{\frac{1-sup(Q)}{sup(Q)}} * \sqrt{\frac{indexsup(S_i)}{1-indexsup(S_i)}} \end{aligned} \quad (3)$$

Proof. According to Lemma 1 and 3:

$$\begin{aligned} \phi(Q, S_i) &\leq \sqrt{\frac{1-sup(Q)}{sup(Q)}} * \sqrt{\frac{sup p(S_i)}{1-sup p(S_i)}} \\ &\leq \sqrt{\frac{1-sup p(Q)}{sup(Q)}} * \sqrt{\frac{index sup p(S_i)}{1-index sup p(S_i)}} \end{aligned}$$

Theorem 2. Given a query graph Q and a sub-graph S_i in a graph database D , the upper bound of $\phi(Q, S_i)$ (that is $upperbound2(\phi(Q, S_i))$) is a monotone increasing function w.r.t $indexsup(S_i)$.

Proof. It is omitted due to space limited.

In order to use graph index in the PG-Search algorithm, we need to re-define Total Order and Correct Growth according to *Index Support* as follows.

Definition 12. (Total Order) Given two sub-graphs S_1 and S_2 in graph database D , where S_1 is not isomorphism to S_2 , we can say that $S_1 < S_2$ if either of the following conditions holds:

- 1) $indexsup(S_1) > indexsup(S_2)$;
- 2) $indexsup(S_1) == indexsup(S_2)$ AND $size(S_1) < size(S_2)$, where $size(S_1)$ denotes the number of edges in S_1 .
- 3) $indexsup(S_1) == indexsup(S_2)$ AND $size(S_1) == size(S_2)$ AND $label(s_1) < label(s_2)$, where $label(S_1)$ and $label(S_2)$ are the canonical labeling of sub-graphs S_1 and S_2 respectively.

Lemma 5. According to Definition 12, if sub-graphs $S_1 < S_2$, then $upperbound2(\phi(Q, S_1)) \geq upperbound2(\phi(Q, S_2))$.

Definition 13. Correct Growth. Given a size-($n+1$) sub-graph C (child), it can be obtained by growing some size- n sub-graphs P_i (parent), $i=1..m$. Among all growthes, the growth from P_j to C is called the **correct growth**, where P_j is the largest parent among all parents according to Total Order in Definition 12. Otherwise, the growth is incorrect one.

We show the pseudo codes of PG-search algorithm in Algorithm 1. In PG-search algorithm, first, given a query Q , we obtain projected graph database D_Q by performing sub-graph Q query (Line 1). Then, we find all size-1 sub-graphs S_i (having one edge) in D_Q , and insert them into max-heap H in increasing order according to Definition 12. We also need to maintain the occurrences of sub-graphs S_i in the projected database D_Q (Line 2). The threshold β is recorded as the top-K $\phi(Q, S_i)$ in result set RS by now. Initially, we set $\beta = -\infty$ (Line 3). We set $\alpha = upperbound2(Q, h)$, where h is the heap head in H (Line 4). For heap head h in max-heap H , we perform sub-graph search to obtain $sup(h)$, and compute $\phi(h, Q)$. We also insert h into result set RS (Line 5). Then, we update β to be the Top-K answer in RS (Line 6). If $\beta \geq \alpha$, the algorithm reports top-K results (Line 7). Otherwise, we pop the heap head h (Line 8). We find all GEs with regard with h in the projected graph database D_Q (Line 9). For each GE g , if growing h through g is not a correct growth, it is ignored (Line 15). If the growth is correct, we obtain a new sub-graph C . We insert it into heap H according to Total Order in Definition 12 (Line 13). Then, we set $\alpha = upperbound2(\phi(Q, h))$, where h is the new head in H (Line 16). For the new heap head h , we perform sub-graph search to obtain $sup(h)$ and compute $\phi(h, Q)$. We insert h

Algorithm 1 PG-search algorithm

Require: **Input:** a graph database D and a query graph Q and K
Output: the top-K correlated sub-graph.

- 1: We conduct sub-graph Q search to obtain projected graph database D_Q .
- 2: Find all size-1 sub-graph S_i (having one edge), and insert them in max-heap H in order of Total Order in Definition 8. We also main their occurrences in projected graph database D_Q .
- 3: Set threshold $\beta = -\infty$.
- 4: For heap head h in max-heap H , we perform sub-graph query to obtain $sup(h)$. Then, we set $\alpha = upperbound2(\phi(h, Q))$ where $upperbound2(\phi(h, Q))$ is computed according to Equation 3.
- 5: For heap head h in max-heap H , we compute $\phi(h, Q)$ and insert it into answer set RS in non-increasing order of $\phi(h, Q)$.
- 6: Update β to be the Top-K answer in RS .
- 7: **while** ($\beta < \alpha$) **do**
- 8: Pop the current head h into the table T .
- 9: Find all GEs around the occurrences of h in the projected graph D_Q .
- 10: **for** each GE g **do**
- 11: Assume that we can get a sub-graph C through growing h by growth element g .
- 12: **if** The growth from h to C is correct growth **then**
- 13: Insert P into max-heap H according to Total Order in Definition 12. We also maintain the occurrence list for C in the projected graph database D_Q .
- 14: **else**
- 15: continue
- 16: Set $\alpha = upperbound2(\phi(Q, h))$, where h is the new head in H .
- 17: For the new heap head h in max-heap H , we perform sub-graph query to obtain $sup(h)$, and compute $\phi(h, Q)$. Then insert it into answer set RS in non-increasing order of $\phi(h, Q)$.
- 18: Update β to be the Top-K answer in RS .
- 19: Report top-K results in RS

into result set RS (Line 17). Update β to be the Top-K answer in RS (Line 18). We iterate Lines 7-21 until $\beta \geq \alpha$ (Line 7). As last, we report top-K answers (Line 19).

Notice that, in Line 4 and 16 of PG-search algorithm, we utilize graph indexing technique to support sub-graph query to obtain $sup(h)$. In Line 13, we also utilize indexes to obtain $indexsup(h)$, which is obtained in filtering process.

In order to determine the growth from h to C is correct or not (Line 12), we need to generate all parents P_i of h , and then perform sub-graph P_i query (only filtering process) to obtain $indexsup(P_i)$. At last, we determine whether h is the largest one among all P_i according to Definition 12. Obviously, the above method leads to numerous sub-graph query operations. We propose another efficient method for the determination as follows:

As we know, at each mining step, we always pop the heap head h_i from the heap H (Line 8). We can maintain all h_i into a table T , which is maintained in memory. For a growth from size- n sub-graph P to size- $(n+1)$ sub-graphs C , we generate all parents P_i of C . If all parents P_i except for P have existed in the table T , the parent P must be the largest one among all parents P_i (see Lemma 8 and proof). Therefore, the growth from P to C is correct one. Otherwise, the growth is incorrect. In the method, we only perform graph isomorphism test in table T . Furthermore, graph canonical labeling will facilitate the graph isomorphism. For example, we can record canonical

labels of all sub-graphs in table T . The hash table built on these canonical labels will speed up the determination.

We first discuss Lemma 6 and Lemma 7, which are used to prove the correctness of Lemma 8 and Theorem 3.

Lemma 6. *Assume that h_1 and h_2 are heads of heap H (Line 8) in PG-Search algorithm in i -th and $(i + 1)$ -th iteration step (Lines 8-18). According to Total Order in Definition 12, we know that $h_1 < h_2$.*

Proof. 1) If h_2 exists in heap H in i -th iteration step: Since the heap H is in increasing order according to Definition 12 and h_1 is the heap head in i -th iteration step, it is straightforward to know $h_1 < h_2$.

2) If h_2 does not exist in heap H in i -th iteration step: According to PG-search algorithm, h_2 must be generated by growing the heap head h_1 in i -step. It means that h_1 is a parent of h_2 . According to Property 2, $indexsup(h_1) \geq indexsup(h_2)$. Furthermore, $size(h_1) < size(h_2)$. Therefore, according to Definition 12, we know that $h_1 < h_2$.

Total Order in Definition 12 is “transitive”, thus, Lemma 7 holds.

Lemma 7. *Assume that h_1 and h_2 are heads of heap H in PG-Search Algorithm in i -th and j -th iteration step (Lines 8-18), where $i < j$. According to Definition 12, we know that $h_1 < h_2$.*

As discussed before, in PG-search algorithm, we maintain the heap head h of each step into the table T . The following lemma holds.

Lemma 8. *For a growth from size- n sub-graph P to size- $(n + 1)$ sub-graphs C , we generate all parents P_i of C . If all parents P_i except for P have existed in the table T , the parent P must be the largest one among all parents P_i .*

Proof. (Sketch) Since all parents P_i except for P have existed in the table T , it means all P_i exist before P in heap H . According to Lemma 7, we know $P_i < P$. Therefore, P is the largest one among all parents P_i .

Theorem 3. Algorithm Correctness. *PG-search algorithm can find the correct top- K correlated sub-graphs.*

Proof. (Sketch) Assume that $\beta \geq \alpha$ at n th iteration step of PG-search algorithm, where $\alpha = upperbound2(\phi(Q, h))$ and h is the heap head at n th step. According to Lemma 7, $h' < h$, where h' is the heap head at j th step, where $j > n$. According to Lemma 5, we know that $upperbound2(\phi(Q, h')) < upperbound2(\phi(Q, h)) = \alpha$. Since $\beta \geq \alpha$, $\alpha > upperbound2(\phi(Q, h'))$. It means that, in latter iteration steps, we cannot find top- K answers. Therefore, PG-search algorithm can find all correct top- K answers, if the algorithm terminates when $\beta \geq \alpha$ (Line 7). The correctness of PG-search algorithm is proved.

4 Experiments

In this section, we evaluate our methods in both real dataset and synthetic dataset. As far as we know, there is no existing work on top- K correlation sub-graph search problem. In [8], Ke et al. proposed threshold-based correlation sub-graph search. As discussed in Section 1, we can transfer top- K correlation sub-graph search into threshold-based one with decreasing threshold. In the following section, we refer the method as “Decreasing Threshold-Based CGS”(DT-CGS for short). We implement “DT-CGS” by ourselves and optimize it according to [8]. All experiments coded by ourself are implemented by standard C++ and conducted on a P4 1.7G machine of 1024M RAM running Windows XP.

1) Datasets. We use both real dataset (i.e. AIDS dataset) and synthetic dataset for performance evaluation.

AIDS Dataset. This dataset is available publicly on the website of the Developmental Therapeutics Program. We generate 10,000 connected and labeled graphs from the molecule structures and omit Hydrogen atoms. The graphs have an average number of 24.80 vertices and 26.80 edges,

and a maximum number of 214 vertices and 217 edges. A major portion of the vertices are C, O and N. The total number of distinct vertex labels is 62, and the total number of distinct edge labels is 3. We refer to this dataset as AIDS dataset. We randomly generate four query sets, F_1 , F_2 , F_3 and F_4 , each of which contains 100 queries. The support ranges for the queries in F_1 to F_4 are $[0.02, 0.05]$, $(0.05, 0.07]$, $(0.07, 0.1]$ and $(0.1, 1)$ respectively.

Synthetic Dataset. The synthetic dataset is generated by a synthetic graph generator provided by authors of [9]. The synthetic graph dataset is generated as follows: First, a set of S seed fragments are generated randomly, whose size is determined by a Poisson distribution with mean I . The size of each graph is a Poisson random variable with mean T . Seed fragments are then randomly selected and inserted into a graph one by one until the graph reaches its size. Parameter V and E denote the number of distinct vertex labels and edge labels respectively. The cardinality of the graph dataset is denoted by D . We generate the graph database using the following parameters with gIndex in [18]: $D=10,000$, $S=200$, $I=10$, $T=50$, $V=4$, $E=1$.

2) Experiment 1. Benefiting from graph indexing techniques, we can only maintain the occurrences in the projected database, as discussed in Section 3.2. In the experiment, we evaluate the indexing technique in PG-search. We use GCoding indexing technique [20] in the experiment. First, in Fig. 7(a) and 7(c), we fix the query sets F_1 and vary K from 10 to 100. Fig. 7(a) and Fig. 7(c) show the running time and memory consumption respectively. Second, in Fig. 7(b) and 7(d), we fix K to be 50 and use different query sets F_1 to F_4 . In “PG-search without Index”, we need to maintain the occurrence list in the whole database. Therefore, it needs more memory consumption, which is shown in Fig. 7(c) and 7(d). Observed from Fig. 7(a) and 7(b), it is clear that “PG-search” is faster than “PG-search without Index”.

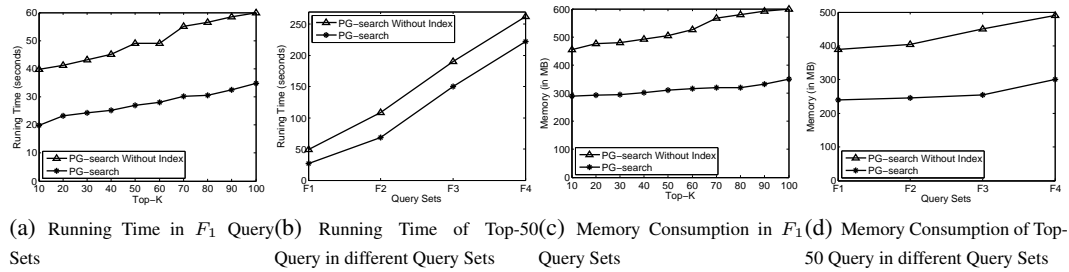


Fig. 7. Evaluating Indexing Technique in PG-search Algorithm in AIDS datasets

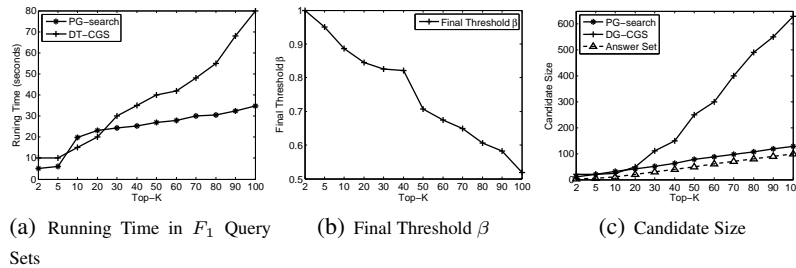


Fig. 8. PG-search VS. Decreasing Threshold-based CGS on AIDS Dataset

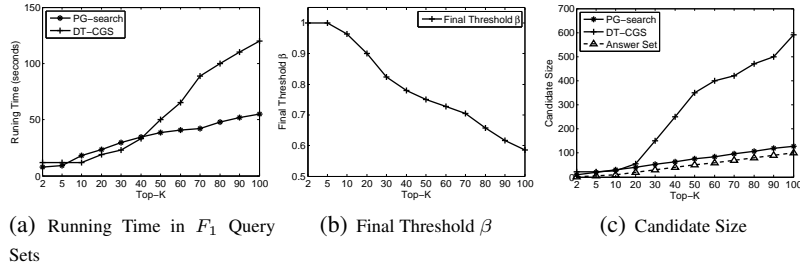


Fig. 9. PG-search VS. Decreasing Threshold-based CGS on Synthetic Dataset

3) Experiment 2. In the experiment, we compare PG-search with decreasing threshold-based CGS (DT-CGS for short). We fix query sets F_1 and vary K from 2 to 100. Fig. 9(a) reports running time. We also report the final threshold β for top- K query in Fig. 9(b). Observed from Fig. 9(a), PG-search is faster than DT-CGS in the most cases. In DT-CGS, it is difficult to set an appropriate threshold θ . Therefore, in the experiments, we always first set $\theta = 0.9$, and then decrease it by “0.02” in each following step based on the number of answers obtained from the previous step. For example, in Fig. 9(a), when $K = 2, 5$, the running time in DT-CGS keeps the same, because the final threshold are $\beta = 1.0$ and $\beta = 0.95$ respectively (see Fig. 9(b)). It means that we can find the top-2 and top-5 answers, if we set $\theta = 0.9$ in the first step in DT-CGS. As the increasing of K , the performance of DT-CGS decreases greatly. Fig. 9(b) explains the cause. When K is large, the final threshold β is small. Thus, we have to perform CGS algorithm [8] many times according to decreasing threshold θ . Thus, it leads to more candidates in DT-CGS.

Furthermore, we observed that the increasing trend of running time in PG-search is much slower than that in DT-CGS algorithm. In PG-search algorithm, we need to compute $\phi(Q, h)$ in each iteration step and h is the head of heap H . Therefore, we can regard h as a candidate answer. Fig. 9(c) reports candidate size in PG-search and DT-CGS algorithm. Observed from Fig. 9(c), we also find the increasing trend of candidate size in PG-Search is much slower than that in DT-CGS. We have the similar results on other query sets in the experiments. Due to space limit, we do not report them here. We also compare PG-search with DT-CGS on synthetic datasets in Fig. 9. The experiment results also confirm the superiority of our method.

5 Related Work

Given a query graph Q , retrieving related graphs from a large graph database is a key problem in many graph-based applications. Most existing work is about sub-graph search [13, 18]. Due to NP-complete hardness of sub-graph isomorphism, we have to employ *filtering-and-verification* framework. First, we use some pruning strategies to filter out false positives as many as possible, second, we perform sub-graph isomorphism for each candidate to fix the correct answers. Furthermore, similar sub-graph search is also important, which is defined as: *find all graphs that “closely contain” the query graph Q .*

However, the above similar subgraph search is structure-based. Ke et al. first propose correlation mining task in graph database, that is correlation graph search (CGSearch for short)[8]. CGSearch can be regarded as *statistical similarity* search in graph database, which is different from structural similarity search. Therefore, CGSearch provides an orthogonal search problem of structural similarity search in graph database. CGSearch in [8] requires the specification of a minimum correlation threshold θ to perform the computation. In practice, it may be not trivial for users to provide an appropriate threshold θ , since different graph databases typically have different characteristics. Therefore, we propose an alternative mining task: *top- K correlation sub-graph search, TOP-CGS.*

Correlation mining is always used to discover the underlying dependency between objects. The correlation mining task finds many applications, such as market-basket databases [2, 10], multimedia databases [11]. In [16], Xiong et al. propose an all-strong-pairs correlation query in a market-basket database. In [6], Ilyas et al. use correlation discovery to find soft functional dependencies between columns in relational database.

6 Conclusions

In this paper, we propose a novel graph mining task, called as TOP-CGS. To address the problem, we propose a pattern-growth algorithm, called PG-search algorithm. In PG-search, we first grow the sub-graph S_i with the highest support and maintain the threshold β to be the K-largest correlation value $\phi(Q, S_i)$ by now. We increase β in need until we find the correct top-K answers. Furthermore, in order to improve the performance, we utilize graph indexing technique to speed up the mining task. Extensive experiments on real and synthetic datasets confirm the efficiency of our methods.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.
2. S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD*, 1997.
3. D. Cai, Z. Shao, X. He, X. Yan, and J. Han. Community mining from multi-relational networks. In *PKDD*, 2005.
4. S. Fortin. The graph isomorphism problem. *Department of Computing Science, University of Alberta*, 1996.
5. H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, 2006.
6. I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, 2004.
7. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, 2000.
8. Y. Ke, J. Cheng, and W. Ng. Correlation search in graph databases. In *SIGKDD*, 2007.
9. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, 2001.
10. E. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE TKDE*, 15(1), 2003.
11. J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, 2004.
12. E. G. M. Petrakis and C. Faloutsos. Similarity searching in medical image databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(3), 1997.
13. D. Shasha, J. T.-L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS*, 2002.
14. P. Willett. Chemical similarity searching. *J. Chem. Inf. Comput. Sci.*, 38(6), 1998.
15. H. Xiong, M. Brodie, and S. Ma. Top-cop: Mining top-k strongly correlated pairs in large databases. In *ICDM*, 2006.
16. H. Xiong, S. Shekhar, P.-N. Tan, and V. Kumar. Exploiting a support-based upper bound of pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *KDD*, 2004.
17. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, 2002.
18. X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, 2004.
19. X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. pages 766–777, 2005.
20. L. Zou, L. Chen, J. X. Yu, and Y. Lu. A novel spectral coding in a large graph database. In *EDBT*, 2008.