Top-k Correlative Graph Mining

Yiping Ke* James Cheng† Jeffrey Xu Yu‡

Abstract

Correlation mining has been widely studied due to its ability for discovering the underlying occurrence dependency between objects. However, correlation mining in graph databases is expensive due to the complexity of graph data. In this paper, we study the problem of mining top-k correlative subgraphs in the database, which share similar occurrence distributions with a given query graph. The search space of the problem is prohibitively large since every subgraph in the database is a candidate. We propose an efficient algorithm, TopCor, which mines the top-k correlative graphs by exploring only the candidate graphs in the projected database of a query graph. We develop three key techniques for TopCor: an effective correlation checking mechanism, a powerful pruning criteria, and a set of useful rules for candidate exploration. The three key techniques are very effective in directing the search to those highly correlative candidate graphs. We justify by experiments the effectiveness of the three key techniques and show that TopCor is more than an order of magnitude faster than CGSearch, the state-of-the-art threshold-based correlative graph mining algorithm.

1 Introduction

Graph is a general tool for modeling structural relationships between data objects. It has been prevalently used in a wide range of application domains, such as protein interaction graphs in biology [3], chemical compound structures in chemistry [1], food webs in ecology [17], social networks in social science [2], as well as Web graphs [20] and XML documents. With the increasing popularity of graph databases in various applications, discovering useful knowledge from graph databases emerges as one of the most important mining problems.

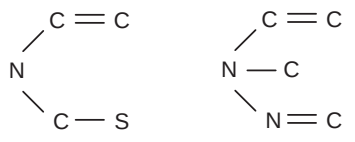
In literature, there have been a number of studies on mining interesting patterns from graph databases. Most of them focus on finding *frequent subgraphs (FGs)* [10, 13, 28, 8] and its compact representations, such as *closed frequent subgraphs (CFGs)* [29] and *maximal frequent subgraphs (MFGs)* [9, 24]. However, little attention has been paid to mining *correlative subgraphs* from graph databases, despite that correlation has been recognized as an interesting and useful type of patterns due to its ability to reveal the underlying occurrence dependency between data objects. Two correlative subgraphs have mutual implications on their occurrences or absences in the database. Therefore, correlative subgraphs indicate hidden properties of the graphs in the database, which are very useful in many applications. For example, a set of user traversal graphs can be extracted from the Web log of a Web site. Correlative subgraphs mined from these graphs have similar occurrence distributions and thus represent the navigation patterns of a group of users who share common interests. By analyzing these correlative graphs, the Web site owner is able to understand user behaviors better in order to improve the Web site structure and detect some abnormal navigation patterns, which is especially important and useful in E-commerce.

In our earlier work [11, 12], we study the problem of finding correlative graphs whose correlation with a query graph is at least a given minimum correlation threshold θ ($0 < \theta \leq 1$). However, since different graph databases can have very different characteristics, the number of correlative graphs obtained for the same value of θ can vary significantly for different queries and databases. We illustrate this problem using the following example.

Example 1.1. Figure 1 shows two query graphs for a real chemical compound structure database. When θ is set to be 0.5, there are 66 correlative graphs for query A, while there are as many as $5,660$ correlative graphs for query B. Given such a great difference in the number of answers, it is hard for a user to specify a suitable value of θ for each specific query graph. As a result, a user may need to try many times in order to obtain a set of patterns that is truly useful to him, while trying each value of θ itself is an expensive process especially when θ is small. □

To address this problem, we propose to discover the top-k correlative subgraphs in graph databases, which is formalized as follows. *Given a graph database $\mathcal{D}$ that contains N graphs, a query graph q and an integer k, find the top-k subgraphs in $\mathcal{D}$ that have the*

*The Chinese University of Hong Kong, ypke@se.cuhk.edu.hk
†Nanyang Technological University, j.cheng@acm.org
‡The Chinese University of Hong Kong, yu@se.cuhk.edu.hk



(a) Query A (b) Query B

Figure 1: Two Example Query Graphs

highest correlation with q. The usage of k has several advantages. It circumvents the need for a *user-specified* minimum correlation constraint θ and allows a user to directly control the number of patterns discovered. Therefore, a user is able to specify how to identify interesting patterns by choosing a suitable k.

Mining top-k correlative subgraphs poses significant challenges. First, the search space of the problem is prohibitively large. Each subgraph of a graph in $\mathcal{D}$ can be a candidate top-k answer graph while there are exponentially many subgraphs in $\mathcal{D}$. Second, correlation measures do not have the anti-monotone property. As a result, the apriori-like pruning strategy cannot be applied to reduce the search space. Finally, graph operations such as subgraph isomorphism testing are expensive. In order to compute the correlation between a candidate graph g and the query graph q, a number of subgraph isomorphism testings need to be performed to obtain their occurrence probabilities (also called *support*). It is obviously intractable to compute the correlation for every subgraph in $\mathcal{D}$.

A straightforward solution for the top-k correlation mining problem is by applying the θ-threshold-based correlation mining algorithm, CGSearch [11], as follows. We first set a high initial value of θ and gradually lower θ until k answer graphs are obtained. However, this approach has several drawbacks. First, there is no obvious correspondence between the values of θ and k. Therefore, given a specific k, there is no clue what value of θ gives the k highest correlative answer graphs. Therefore, different queries may require very different values of θ in order to find the top-k answers using CGSearch, for the same k. Second, in the process of trying different values of θ, many correlative subgraphs are re-computed and thus the computation time is wasted. As a result, this approach is not efficient and not scalable for the top-k mining task.

Example 1.2. Using the two queries in Figure 1, if we want to find their top-20 correlative graphs using CGSearch, the answers are obtained at $\theta = 0.5$ for query A but at $\theta = 0.9$ for query B. Therefore, we cannot set a universal θ for different queries. If we set $\theta = 0.9$, we miss some answers for query A. If we set $\theta = 0.5$ instead, we obtain $5,660$ answers for query B, the majority of which are not useful to the user. More importantly, setting a low θ for query B results in 40 times longer processing time (0.45 vs. 17.91 seconds), which is not necessary at all.

Instead of choosing a universal θ, a better way of using CGSearch for top-k mining is to lower θ gradually until k answer graphs are obtained. For example, if we start at $\theta = 0.9$, we need only 0.45 seconds for query B. However, we need to perform several iterations of lowering θ to process other queries such as query A, which requires 1.9 seconds. By lowering θ to find top-k answers is still not efficient enough since we need only 0.75 seconds for processing query A if we can directly set θ at the correct value of 0.5. Unfortunately, the correct θ for processing a query is not known.

Therefore, an algorithm for finding top-k correlative graphs directly is highly demanded. The algorithm we propose in this paper requires only 0.26 and 0.11 seconds for processing queries A and B, which are even faster than using CGSearch with a known correct θ. □

In this paper, we propose an efficient algorithm, *TopCor*, for computing the top-k correlative graphs for a given query. We adopt Pearson's correlation coefficient ϕ [30] as the correlation measure; but we remark that other measures [23] can also be applied in a similar way. Instead of directly searching the whole database $\mathcal{D}$, TopCor limits the search space to the *projected database* $\mathcal{D}_q$, of q and explores only candidate graphs from $\mathcal{D}_q$. Here, the projected database of q refers to the subset of graphs in $\mathcal{D}$ that are supergraphs of q. Since $\mathcal{D}_q$ is usually much smaller than $\mathcal{D}$, mining from $\mathcal{D}_q$ is much more efficient than from $\mathcal{D}$.

Transforming the search space to the projected database alone is still inadequate. In order to more effectively direct the search to those highly correlative subgraphs, we develop three key techniques: *an effective correlation checking mechanism*, *a powerful pruning criteria*, and *a set of useful rules for candidate exploration*.

First, we investigate the property of the correlation function ϕ and derive a maximum value $\phi_{max}(g)$ that a candidate graph g can achieve. By comparing $\phi_{max}(g)$ with the smallest correlation value in the current top-k list (denoted as ϕ_{min}), we can check whether g is a potential answer without computing its exact ϕ value. This ϕ_{max}-based *checking mechanism* effectively prunes unqualified candidate graphs and saves a lot of correlation computations. Second, we further study the property of $\phi_{max}(g)$ and derive a minimum support threshold as a function of ϕ_{min}, denoted as $\varsigma(\phi_{min})$, which is used as a *pruning criteria* for mining the correlative subgraphs from $\mathcal{D}_q$. The distinguished property of $\varsigma(\phi_{min})$ is that $\varsigma(\phi_{min})$ progressively becomes tighter

during the mining process and thereby significantly increases the pruning power. Finally, we develop *a set of five rules* to further save the correlation computations and prune false-positive candidate graphs. Based on these rules, we design a bi-directional search strategy to avoid the expensive correlation computations on both the subgraphs and supergraphs of a candidate graph.

We integrate the above-mentioned three key techniques into our mining algorithm TopCor. Our comprehensive experiments show that each one of the key techniques is a crucial contributor to the efficiency of TopCor. Compared with the CGSearch algorithm, TopCor is over an order of magnitude faster, which, together with the flexibility on the usage of k, demonstrates the benefit of designing a new algorithm specifically for the top-k mining task. Experimental results also show that TopCor achieves very impressive and stable performance when varying the query support, database size, as well as the density of the graphs in the database.

We summarize the contributions of this paper as follows.

- We propose the problem of discovering top-k correlations from graph databases, which provides the user with the flexibility to control the number of correlative subgraphs to be obtained.
- We develop three key techniques to facilitate the mining of top-k correlations, all of which play an important role in reducing the large search space to a smaller and more promising one.
- We propose an efficient algorithm, TopCor, to solve the problem of top-k correlation mining in graph databases.
- We conduct extensive experiments that verify the efficiency of our algorithm, as well as the effectiveness of the key techniques.

The rest of the paper is organized as follows. We introduce some background knowledge on graph databases and frequent subgraph mining in Section 2. We define the top-k correlation mining problem in Section 3. We present our solution, TopCor, in Section 4. Then, we evaluate the performance of TopCor in Section 5. Finally, we review related work in Section 6 and conclude the paper in Section 7.

2 Background

In this paper, we focus on *undirected labelled connected graphs*, while our approach can be straightforwardly extended to handle directed and unlabelled graphs.

Let $g = (V, E, l)$ denote a graph g, where V is the set of vertices, E is the set of edges and l is a labelling function that assigns a label to each vertex and edge. A *graph database* is a collection of N graphs, denoted as $\mathcal{D} = \{g_1, g_2, \ldots, g_N\}$. Given two graphs, $g = (V, E, l)$ and $g' = (V', E', l')$, we call that g is a *subgraph* of g' (or g' is a *supergraph* of g), denoted as $g \subseteq g'$ (or $g' \supseteq g$), if there exists an injective function $f\colon V \to V'$, such that for every edge $(u, v) \in E$, we have $(f(u), f(v)) \in E'$, $l(u) = l'(f(u))$, $l(v) = l'(f(v))$, and $l(u, v) = l'(f(u), f(v))$. The injective function f is called a *subgraph isomorphism* from g to g'. It have been proved that testing subgraph isomorphism is an *NP*-complete problem [7].

Given a graph database $\mathcal{D}$ and a graph g, we denote the set of all graphs in $\mathcal{D}$ that are supergraphs of g as $\mathcal{D}_g = \{g' : g' \in \mathcal{D},\ g' \supseteq g\}$. We call $\mathcal{D}_g$ the *projected database* of $\mathcal{D}$ on g. The *frequency* of g in $\mathcal{D}$, denoted as $freq(g; \mathcal{D})$, is defined as the number of graphs in $\mathcal{D}$ that are supergraphs of g, i.e., $freq(g; \mathcal{D}) = |\mathcal{D}_g|$. The *support* of g in $\mathcal{D}$, denoted as $supp(g; \mathcal{D})$, is defined as the percentage of graphs in $\mathcal{D}$ that are supergraphs of g, i.e., $supp(g; \mathcal{D}) = \frac{freq(g;\mathcal{D})}{|\mathcal{D}|} = \frac{|\mathcal{D}_g|}{|\mathcal{D}|}$. For simplicity, we use $freq(g)$ and $supp(g)$ to represent $freq(g; \mathcal{D})$ and $supp(g; \mathcal{D})$, respectively, when $\mathcal{D}$ is clear in the context. The support measure has the *anti-monotone* property, i.e., if $g_1 \subseteq g_2$, then $supp(g_1) \geq supp(g_2)$.

Given two graphs g_1 and g_2, we define the *joint frequency* of g_1 and g_2 in $\mathcal{D}$, denoted as $freq(g_1, g_2)$, as the number of graphs in $\mathcal{D}$ that are supergraphs of both g_1 and g_2, i.e., $freq(g_1, g_2) = |\mathcal{D}_{g_1} \cap \mathcal{D}_{g_2}|$. Similarly, we define the *joint support* of g_1 and g_2 as $supp(g_1, g_2) = \frac{freq(g_1, g_2)}{|\mathcal{D}|}$. Joint support has the following properties: $supp(g_1, g_2) \leq supp(g_1)$ and $supp(g_1, g_2) \leq supp(g_2)$.

A graph g is called a *Frequent subGraph* (*FG*) [10] in $\mathcal{D}$ if $supp(g) \geq \sigma$, where σ $(0 \leq \sigma \leq 1)$ is a user-specified *minimum support threshold*. For an FG g, if there exists no supergraph of g with the same support value, g is called a *Closed Frequent subGraph* (*CFG*).

Example 2.1. Figure 2 shows an example graph database, $\mathcal{D}$, that contains ten graphs, $g_1, \ldots, g_{10}$. For clarity of presentation, all the nodes are of the same label, which is omitted in the figure; while the edges are labelled with a, b and c.

The graph g_8 is a subgraph of g_2. The projected database of g_8, i.e., $\mathcal{D}_{g_8}$, is $\{g_2, g_3, g_6, g_7, g_8\}$, where the corresponding subgraph isomorphism is indicated by the colored nodes. The frequency of g_8 in $\mathcal{D}$ is computed as $freq(g_8) = |\mathcal{D}_{g_8}| = 5$. The support of g_8 in $\mathcal{D}$ is $supp(g_8) = \frac{freq(g_8)}{|\mathcal{D}|} = \frac{5}{10} = 0.5$. Similarly for g_9, we have $\mathcal{D}_{g_9} = \{g_6, g_7, g_9\}$, $freq(g_9) = 3$ and $supp(g_9) = 0.3$. The joint frequency of g_8 and g_9 is computed as $freq(g_8, g_9) = |\mathcal{D}_{g_8} \cap \mathcal{D}_{g_9}| = |\{g_6, g_7\}| = 2$. Therefore, the joint support of g_8 and g_9 is computed as

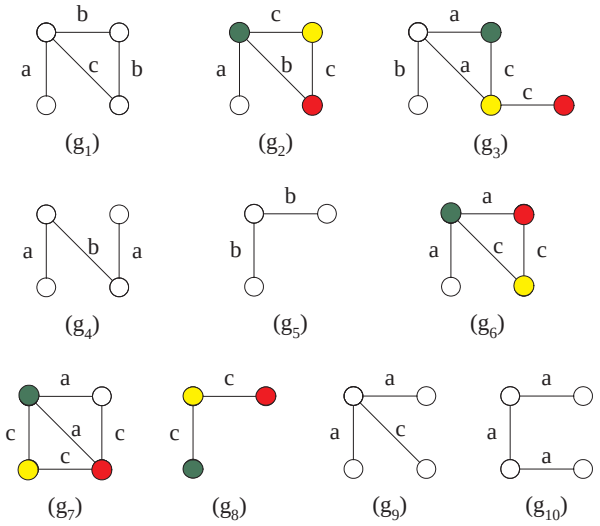


Figure 2: An Example Graph Database, $\mathcal{D}$

$supp(g_8, g_9) = \frac{freq(g_8,g_9)}{|\mathcal{D}|} = 0.2$. If we set the minimum support threshold $\sigma = 0.4$, then g_8 is an FG while g_9 is not. Graph g_8 is also a CFG since all of its supergraphs have less support values. □

3 Problem Definition

The top-k correlative graph mining problem we study in this paper can be formalized as follows. *Given a graph database $\mathcal{D} = \{g_1, g_2, \ldots, g_N\}$, a query graph q, and a threshold k, find the top-k subgraphs in $\mathcal{D}$ with the highest correlation with respect to q.*

The correlation between two graphs is defined on their occurrence distributions in the graph database $\mathcal{D}$. We use the following example to illustrate the concept of correlation between two graphs.

EXAMPLE 3.1. Consider the graph database $\mathcal{D}$ in Figure 2. Given a graph that contains two connected edges labelled with a (denoted as graph aa), its occurrence distribution in $\mathcal{D}$ is shown by the solid line in Figure 3. The occurrence of aa is set to be 1 for each graph in $\mathcal{D}$ that is a supergraph of aa, and 0 otherwise. For example, g_6 is a supergraph of aa; therefore, the value of the solid line at g_6 is 1. Similarly, we can plot the occurrence distribution of a graph ac as shown by the dotted line in Figure 3.

The correlation between two graphs measures the similarity of their occurrence distributions. As shown in Figure 3, the correlation between aa and ac is high since their occurrence distributions take the same value for most graphs in $\mathcal{D}$. □

Any existing correlation measure [23] can be used to characterize correlation between graphs. In this paper,

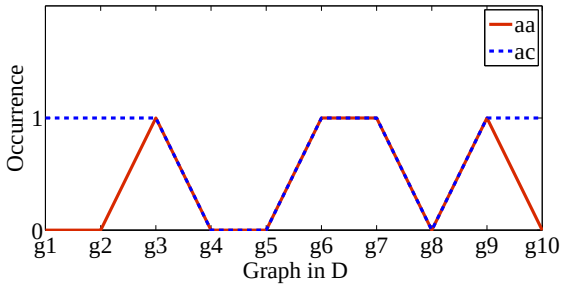


Figure 3: Occurrence Distributions of Graphs aa and ac in $\mathcal{D}$

we focus on *Pearson's correlation coefficient* [21], while other correlation measures can also be adopted in a similar way [12].

DEFINITION 3.1. (Pearson's Correlation Coefficient) *Given two graphs g_1 and g_2, the* Pearson's correlation coefficient *of g_1 and g_2, denoted as $\phi(g_1, g_2)$, is defined as*

(3.1)
$$\phi(g_1, g_2) = \frac{supp(g_1,g_2) - supp(g_1)supp(g_2)}{\sqrt{supp(g_1)supp(g_2)(1-supp(g_1))(1-supp(g_2))}}.$$

When $supp(g_1)$ or $supp(g_2)$ is equal to 0 or 1, $\phi(g_1, g_2)$ is defined to be 0.

The value of $\phi(g_1, g_2)$ falls within the range of $[-1, 1]$. The value 0 indicates that the occurrences of g_1 and g_2 in the database are independent; positive value indicates that the occurrences of g_1 and g_2 are positively correlative, while negative value means negative correlation. In this paper, we focus on finding top-k positive correlations.

We now present a property of Pearson's correlation coefficient, which will be used to develop effective pruning techniques in the following section. We omit the proof due to limited space.

PROPERTY 3.1. *If both $supp(g_1)$ and $supp(g_1, g_2)$ are fixed, then $\phi(g_1, g_2)$ is monotonically decreasing with $supp(g_2)$.*

4 Our Solution

To solve the top-k correlative graph mining problem, a naive approach is to mine all subgraphs in $\mathcal{D}$, compute their correlation with the query graph, sort them and extract the top-k correlative subgraphs. This approach is apparently intractable, especially when the database $\mathcal{D}$ is large or the graphs in $\mathcal{D}$ are diverse (i.e., the number of distinct subgraphs in $\mathcal{D}$ is large).

In our earlier work [11], we proposed an efficient algorithm, CGSearch, to find all subgraphs in $\mathcal{D}$ whose correlation with q is higher than a given threshold θ. For

top-k correlation mining, one possible way is to utilize the CGSearch algorithm by initially setting a high θ and progressively lowering it until the size of the answer set is larger than k. Finally, the top-k answers are obtained by sorting the results returned by CGSearch. However, this approach is not efficient enough since it is hard to specify a value of θ that is corresponding to a given k. The lower the value of θ, the higher the complexity of the mining problem. As a result, a lot of computation effort is wasted in trying different θ values.

We propose an efficient solution that is able to direct the searching to those highly correlative subgraphs. Similar to the work in [11], we explore the search space by mining FGs from the projected database of the query graph, $\mathcal{D}_q$. This is natural since correlation measures the co-occurrences of a subgraph and the query graph q. Furthermore, mining from $\mathcal{D}_q$ is much more efficient than mining from the entire database $\mathcal{D}$, since $\mathcal{D}_q$ is much smaller than $\mathcal{D}$.

After we transform the search space from $\mathcal{D}$ to $\mathcal{D}_q$, however, we still have three major challenges remaining for the top-k correlation mining task. First, it is inefficient or even infeasible to mine every possible subgraph from the projected database. Thus, we need a powerful *pruning criteria* to prune unpromising candidates in the mining process as many as possible, while at the same time guaranteeing the completeness and correctness of the top-k answer graphs.

Second, for each candidate graph g, we need to obtain its support $supp(g)$ and joint support $supp(g, q)$ in order to compute its correlation $\phi(g, q)$. The joint support $supp(g, q)$ can be obtained at a relatively low cost from the smaller projected database; however, computing $supp(g)$ is in general expensive (even though we may obtain $supp(g)$ through a querying operation using a graph index such as [6]). Therefore, we need an effective *correlation checking mechanism* to determine whether a candidate graph is a top-k answer without obtaining its support.

Finally, since the search space of the problem is prohibitively large, we need to develop useful *rules for candidate exploration* so that the search can be directed more quickly to those potentially high-correlative graphs.

4.1 Key Techniques When mining FGs from the projected database, we maintain a queue of current top-k answer graphs discovered so far, which is denoted as Q_{cur}. Each graph $g \in Q_{cur}$ also has a corresponding correlation value $\phi(g, q)$. The graphs in Q_{cur} are sorted in the descending order of their ϕ values. The ϕ value of the k-th graph in Q_{cur} is also denoted as ϕ_{min}. For each newly discovered candidate graph g, its correlation value $\phi(g, q)$ should be at least ϕ_{min}; otherwise, g is not a top-k answer graph and can thus be pruned.

During the mining process in the projected database $\mathcal{D}_q$, the support of a candidate graph g in the projected database, i.e., $supp(g; \mathcal{D}_q)$, can be obtained. Therefore, we can also obtain the value of the joint support of g and q wrt the whole database $\mathcal{D}$ by the following equation:

$$supp(g, q) = supp(g; \mathcal{D}_q) \cdot supp(q). \tag{4.2}$$

We now discuss a checking mechanism which can effectively determine whether a candidate graph is a potential top-k answer graph. More specifically, we derive an upper bound on the ϕ value that a candidate graph g can achieve given its joint support $supp(g, q)$.

LEMMA 4.1. *Given a candidate graph g and its joint support $supp(g, q)$, the correlation $\phi(g, q)$ achieves the following maximum value when $supp(g) = supp(g, q)$:*

$$\phi_{max}(g) = \sqrt{\frac{(1 - supp(q))supp(g, q)}{supp(q)(1 - supp(g, q))}}. \tag{4.3}$$

Proof. When computing $\phi(g, q)$ by Eq. (3.1), both the values of $supp(q)$ and $supp(g, q)$ are fixed. By Property 3.1, $\phi(g, q)$ is monotonically decreasing with $supp(g)$. Since $supp(g) \geq supp(g, q)$, it follows that $\phi(g, q)$ achieves its maximum value when $supp(g) = supp(g, q)$. The above $\phi_{max}(g)$ thus follows by replacing $supp(g)$ with $supp(g, q)$ in Eq. (3.1). □

According to Lemma 4.1, we can compute the maximum correlation $\phi_{max}(g)$ for each candidate graph g once it is mined from the projected database. Based on Lemma 4.1, we derive the following theorem, which can be utilized as an effective checking mechanism to quickly prune the unqualified candidate graphs without knowing their support values in $\mathcal{D}$.

THEOREM 4.1. *A candidate graph g can be safely pruned if $\phi_{max}(g) < \phi_{min}$.*

We further investigate the property of $\phi_{max}(g)$ in order to achieve more pruning as well as to obtain a pruning criteria for the mining process.

LEMMA 4.2. *Given two candidate graphs g_1 and g_2, if $supp(g_1, q) \geq supp(g_2, q)$, then $\phi_{max}(g_1) \geq \phi_{max}(g_2)$.*

Proof. The lemma follows since $\phi_{max}(g)$ is monotonically increasing with $supp(q, g)$ by Eq. (4.3). □

By Lemma 4.2, we can determine whether it is necessary to explore the branch rooted at g in the search tree. Once we prune a graph g by Theorem 4.1, i.e.,

$\phi_{max}(g) < \phi_{min}$, we can safely prune all the supergraphs of g. This is simply because the supergraphs of g can only have less ϕ_{max} than g by Lemma 4.2 and thus can be pruned since their ϕ_{max} values are less than ϕ_{min} for sure. Based on Lemma 4.2, we further develop a pruning criteria for the mining process as stated in the following theorem.

THEOREM 4.2. *When mining the projected database $\mathcal{D}_q$, the following function of ϕ_{min} can be used as a minimum support threshold:*

$$\varsigma(\phi_{min}) = \frac{\phi_{min}^2}{\phi_{min}^2 \cdot supp(q) + 1 - supp(q)}. \tag{4.4}$$

Proof. We only need to prove that given any candidate graph g, if $supp(g; \mathcal{D}_q) < \varsigma(\phi_{min})$, then $\phi(g, q) < \phi_{min}$. By Eq. (4.2), we have $supp(g, q) < (\varsigma(\phi_{min}) \cdot supp(q))$. By treating $(\varsigma(\phi_{min}) \cdot supp(q))$ as a joint support of a graph g' and plugging it into Eq. (4.3), the corresponding $\phi_{max}(g')$ exactly equals to ϕ_{min}. According to Lemma 4.2, since $supp(g, q) < supp(g', q)$, we have $\phi_{max}(g) < \phi_{max}(g') = \phi_{min}$. Therefore, we have proved that $\phi(g, q) \leq \phi_{max}(g) < \phi_{min}$ and thus the theorem follows. □

Theorem 4.2 states a minimum support threshold that can be used in the process of mining the projected database without missing any top-k answer graph. The minimum support threshold is a function of ϕ_{min} and the value of ϕ_{min} increases in the mining process since we keep on updating the queue Q_{cur} with new current top-k answers. Therefore, the minimum support threshold is also increasing and the pruning on the search space becomes more and more effective.

In the following, we further develop a number of useful rules that can be applied in the mining process to facilitate the correlation computation and to achieve greater pruning.

RULE 4.1. *Given a candidate graph g, if $g \supseteq q$, then $\phi(g, q) = \phi_{max}(g)$.*

Proof. Since $g \supseteq q$, we have $supp(g) = supp(g, q)$. By Lemma 4.1, $\phi(g, q) = \phi_{max}(g)$. □

According to Rule 4.1, for each supergraph g of q mined from the projected database, in order to compute $\phi(g, q)$, we do not need to perform an expensive querying operation to obtain its $supp(g)$ since $\phi(g, q)$ exactly equals to $\phi_{max}(g)$. This saves the correlation computation time for the supergraphs of q.

RULE 4.2. *Given two candidate graphs g_1 and g_2, if $g_1 \supset g_2$ and $supp(g_1, q) = supp(g_2, q)$, then $\phi(g_1, q) \geq \phi(g_2, q)$.*

Proof. Since $g_1 \supset g_2$, we have $supp(g_1) \leq supp(g_2)$. The result $\phi(g_1, q) \geq \phi(g_2, q)$ follows since $\phi(g, q)$ is monotonically decreasing with $supp(g)$ when $supp(g_1, q) = supp(g_2, q)$ by Property 3.1. □

According to Rule 4.2, we know that a CFG in the projected database always has a higher correlation than its graphs with the same support values (we call the set of these subgraphs the *closure* of a CFG). Therefore, we can design the following strategy when computing correlation values: we query the $supp(g)$ for a CFG first and then determine whether it is necessary to query the subgraphs in its closure. For a CFG, if it is not added to Q_{cur} after checking its correlation, we can safely prune all the subgraphs in its closure by Rule 4.2 and save the querying time to obtain their support values.

RULE 4.3. *Given two candidate graphs g_1 and g_2, if $g_1 \supset g_2$ and $supp(g_1)$ is known, the following two statements are true:*

(a) If $supp(g_2, q) < supp(g_1)$, then $\phi_{max}(g_2)$ can be updated as

$$\phi_{max}(g_2) = \frac{supp(g_2, q) - supp(q)supp(g_1)}{\sqrt{supp(q)(1-supp(q))supp(g_1)(1-supp(g_1))}}.$$

(b) If $supp(g_2, q) < f(supp(g_1), \phi_{min})$, then $\phi(g_2, q) < \phi_{min}$, where

$$\begin{aligned} & f(supp(g), \phi_{min}) \\ = \; & \phi_{min}\sqrt{supp(q)(1 - supp(q))supp(g)(1 - supp(g))} \\ & + supp(q)supp(g). \end{aligned}$$

Proof. We first prove Part (a). Since $g_1 \supset g_2$ and $supp(g_1) > supp(g_2, q)$, we have $supp(g_2) \geq supp(g_1) > supp(g_2, q)$. In Lemma 4.1, $\phi_{max}(g_2)$ is obtained by using $supp(g_2, q)$ as the lower bound of $supp(g_2)$. According to Property 3.1, $\phi(g_2, q)$ is monotonically decreasing with $supp(g_2)$. Since we now have a lower bound of $supp(g_2)$, i.e., $supp(g_1)$, which is higher than $supp(g_2, q)$, a tighter $\phi_{max}(g_2)$ can be obtained by replacing $supp(g_2)$ with $supp(g_1)$ in Eq. (3.1).

We now prove Part (b). It is easy to prove that the function f is monotonically increasing with $supp(g)$ when ϕ_{min} is fixed (we omit the detailed proof due to limited space). Since $g_1 \supset g_2$, we have $supp(g_1) \leq supp(g_2)$. We then further have $supp(g_2, q) < f(supp(g_1), \phi_{min}) \leq f(supp(g_2), \phi_{min})$. The result $\phi(g_2, q) < \phi_{min}$ thus follows by replacing $supp(g_2, q)$ with $f(supp(g_2), \phi_{min})$ in Eq. (3.1). □

By Rule 4.3(a), after checking the correlation of a graph g (i.e., $supp(g)$ is known), we can update the ϕ_{max}

of its subgraph whose joint support is less than $supp(g)$ to achieve better pruning. On the other hand, in Rule 4.3(b), the condition $supp(g_2, q) < f(supp(g_1), \phi_{min})$ also implies that $\phi(g_1, q) < \phi_{min}$. Therefore, if we prune a candidate graph g_1, we can also prune all its subgraphs whose joint support values are less than $f(supp(g_1), \phi_{min})$.

Rules 4.2 and 4.3 can be applied to achieve better pruning for the subgraphs of a given candidate graph. We now develop the following two rules that work for supergraphs of a candidate graph.

RULE 4.4. *Given two candidate graphs g_1 and g_2, if $g_1 \subset g_2$ and $supp(g_1) = supp(g_1, q)$, then $\phi(g_2, q) = \phi_{max}(g_2)$.*

Proof. Since $g_1 \subset g_2$ and $supp(g_1) = supp(g_1, q)$, we have $supp(g_2) = supp(g_2, q)$. Therefore, it follows that $\phi(g_2, q) = \phi_{max}(g_2)$ by Lemma 4.1. □

According to Rule 4.4, if we find a candidate graph whose support equals to its joint support, we can avoid querying all its supergraphs since their exact ϕ values are essentially their ϕ_{max} values.

RULE 4.5. *Given two candidate graphs g_1 and g_2, where $g_1 \subset g_2$ and $supp(g_1)$ is known, if $supp(g_2, q) \geq f(supp(g_1), \phi_{min})$, then $\phi(g_2, q) \geq \phi_{min}$.*

Proof. Since $g_1 \subset g_2$, we have $supp(g_1) \geq supp(g_2)$. As stated in the proof of Rule 4.3, the function f is monotonically increasing with $supp(g)$ when ϕ_{min} remains the same. Therefore, we have $supp(g_2, q) \geq f(supp(g_1), \phi_{min}) \geq f(supp(g_2), \phi_{min})$. By replacing $supp(g_2, q)$ with $f(supp(g_2), \phi_{min})$ in Eq. (3.1), we have $\phi(g_2, q) \geq \phi_{min}$. □

Notice that in Rule 4.5, the condition $supp(g_2, q) \geq f(supp(g_1), \phi_{min})$ also implies that $\phi(g_1, q) \geq \phi_{min}$. Therefore, Rule 4.5 can be applied as follows: if we add a candidate graph g_1 to Q_{cur}, we can also add its supergraphs whose joint support values are at least $f(supp(g_1), \phi_{min})$ to Q_{cur} without performing any querying operation to obtain their support values.

The following example demonstrates how these key techniques can be applied for search exploration.

EXAMPLE 4.1. Figure 4 shows an example search tree (a prefix tree) when mining the projected database $\mathcal{D}_q$. Each node in the search tree represents a subgraph in $\mathcal{D}_q$ and a child node is a supergraph of its parent node. When a candidate graph g is mined from $\mathcal{D}_q$, its support in $\mathcal{D}_q$, denoted as $supp(g; \mathcal{D}_q)$, is also obtained. Suppose that the query graph q corresponds to graph g_6. By Rule 4.1, we do not need to compute the exact ϕ values for

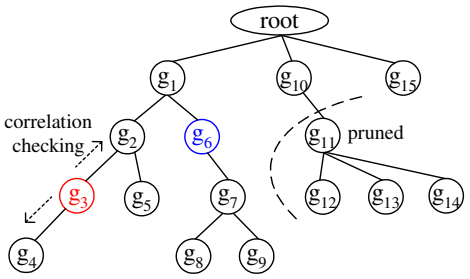


Figure 4: An Example Search Tree

g_7, g_8 and g_9 which are supergraphs of q, since their ϕ values equal to their ϕ_{max} values. Moreover, ϕ_{max} can also be used for early correlation checking. For example, if we find that $\phi_{max}(g_5)$ is less than ϕ_{min} at the time when it is mined, we can directly prune g_5 by Theorem 4.1. Given ϕ_{min}, we can compute a minimum support threshold $\varsigma(\phi_{min})$ for the mining process by Theorem 4.2. For example, if $supp(g_{11}; \mathcal{D}_q)$ is less than $\varsigma(\phi_{min})$, we can immediately prune the branch rooted at g_{11} as indicated in the figure. Suppose that g_3 is a CFG and g_1 and g_2 are its subgraphs with the same support in $\mathcal{D}_q$ (i.e., g_1 and g_2 are in the closure of g_3). By Rule 4.2, $\phi(g_3, q)$ is guaranteed to be higher than $\phi(g_1, q)$ and $\phi(g_2, q)$. Therefore, we can skip the correlation checking of g_1 and g_2 until finding its CFG g_3. We then obtain $supp(g_3)$ by performing a querying operation and compute $\phi(g_3, q)$. After that, we can explore both directions of g_3 by applying Rule 4.3 for its subgraphs and Rules 4.4 and 4.5 for its supergraphs, which is shown in Figure 4. □

4.2 Our Algorithm: TopCor We apply the three key techniques to devise an efficient algorithm, **TopCor**, as shown in Algorithm 1. We use a queue, Q_{cur}, to keep the current top-k correlative subgraphs with respect to a query q. TopCor mines the top-k answers from the projected database $\mathcal{D}_q$ of q, which can be obtained using a graph index [6]. Then, the mining process starts from a graph with a single edge e and invokes **MineTopCor** to grow the graph in a depth-first manner by adding one edge at each step. Intuitively, the mining process conceptually constructs a prefix tree and thus a *valid* graph refers to a graph that is represented as a node in the prefix tree (similar to [29]).

Before we invoke MineTopCor, we first compute $supp(g; \mathcal{D}_q)$ in $\mathcal{D}_q$, which can be used to obtain $supp(g, q)$ by Eq. (4.2). We apply Theorems 4.1 and 4.2 to determine whether we need to invoke MineTopCor or MineTopCor-1 for a graph g (Line 7 of Algorithm 1, Line 12 of Procedure 2, and Line 1 of Procedure 3). This step is the key to the mining efficiency since we can prune all supergraphs that are to be grown from a

Algorithm 1 TopCor

Input: Graph database $\mathcal{D}$, a query q, and an integer k.
Output: The top-k correlative graphs with respect to q.

1. Initialize an empty queue, Q_{cur}, of size k;
2. Obtain $\mathcal{D}_q$;
3. Find the set of distinct edges, E, in $\mathcal{D}_q$;
4. Set ϕ_{min} and $\varsigma(\phi_{min})$ initially as 0;
5. **for each** $e \in E$ **do**
6. Obtain $supp(e; \mathcal{D}_q)$;
7. **if**($\phi_{max}(e) \geq \phi_{min}$ and $supp(e; \mathcal{D}_q) \geq \varsigma(\phi_{min})$)
8. Invoke **MineTopCor**(e);
9. Output the graphs in Q_{cur};

Procedure 2 MineTopCor(g)

1. **if**($g \supseteq q$)
2. **MineTopCor-1**(g); /* By **Rule 4.1** */
3. **else**
4. Let G be the set of *valid* graphs that are grown from g by adding one edge to g;
5. Compute $supp(g'; \mathcal{D}_q)$ for each $g' \in G$;
6. **if**(g is a CFG)
7. Compute $supp(g)$;
8. **if**($\phi(g, q) \geq \phi_{min}$)
9. Push g into Q_{cur} and refine $\varsigma(\phi_{min})$ by **Theorem 4.2**;
10. **for each** $g' \in G$ **do** /* Explore supergraph g' of g */
11. Apply **Rule 4.5** to determine whether to push g' into Q_{cur};
12. **if**($\phi_{max}(g') \geq \phi_{min}$ and $supp(g'; \mathcal{D}_q) \geq \varsigma(\phi_{min})$)
13. **if**($supp(g)$ is computed and $supp(g) = supp(g, q)$))
14. **MineTopCor-1**(g'); /* By **Rule 4.4** */
15. **else**
16. **MineTopCor**(g');
17. **if**($supp(g)$ is not computed)
18. Apply **Rules 4.2** and **4.3** to determine whether $supp(g)$ needs to be computed;
19. **if**($supp(g)$ is not computed and $\phi_{max}(g) \geq \phi_{min}$)
20. Compute $supp(g)$;
21. **if**($\phi(g, q) \geq \phi_{min}$)
22. Push g into Q_{cur} and refine $\varsigma(\phi_{min})$ by **Theorem 4.2**;

Procedure 3 MineTopCor-1(g)

1. **if**($\phi_{max}(g) \geq \phi_{min}$) /* By **Rules 4.1** and **4.4** */
2. Push g into Q_{cur} and refine $\varsigma(\phi_{min})$ by **Theorem 4.2**;
3. Grow g by one edge;
4. **for each** *valid* graph g' grown from g **do**
5. **MineTopCor-1**(g');

graph g if $\phi_{max}(g) < \phi_{min}$ or $supp(g; \mathcal{D}_q) < \varsigma(\phi_{min})$. This pruning is very effective since the pruning power increases when ϕ_{min} increases (Lines 9 and 22 of Procedure 2 and Line 2 of Procedure 3).

To compute $\phi(g, q)$, we need both $supp(g, q)$ and $supp(g)$. However, computing $supp(g)$ is costly since $\mathcal{D}$ is large, while computing $supp(g, q)$ is more efficient since $\mathcal{D}_q$ is in general much smaller. Therefore, we attempt to avoid computing $supp(g)$ as much as possible.

First, Lines 1-2 and Lines 13-14 of Procedure 2 invoke **MineTopCor-1** to process the supergraphs of a graph g. Since $\phi(g, q) = \phi_{max}(g)$ by Rules 4.1 and 4.4, we can avoid computing $supp(g)$ from $\mathcal{D}$. Line 1 of Procedure 2 invokes a subgraph isomorphism testing; however, this is avoided in our implementation since we first start growing the graphs from q in order to fill Q_{cur} with a set of potentially higher correlative subgraphs. In this way, we also obtain a higher ϕ_{min} and $\varsigma(\phi_{min})$ in an early stage to achieve greater pruning.

Second, we do not compute $supp(g)$ for every graph g; rather, we only compute $supp(g)$ if g is a CFG. When $supp(g)$ is computed, we can apply Rule 4.5 to determine whether a supergraph grown from g can be counted as the current top-k answers (Line 11 of Procedure 2). If g is not a CFG, we determine whether it is necessary to compute $supp(g)$ when MineTopCor returns from processing the supergraph, g', of g (Lines 17-18 of Procedure 2). First, if $supp(g', q) = supp(g, q)$ and g' is not in Q_{cur}, then we can safely prune g by Rule 4.2. Second, if $supp(g, q) < f(supp(g'), \phi_{min})$, then we can safely prune g by Rule 4.3(b). Third, if $supp(g') > supp(g, q)$, then we can refine $\phi_{max}(g)$ by Rule 4.3(a). Thus, we compute $supp(g)$ only if g is not pruned and $\phi_{max}(g) \geq \phi_{min}$ (Lines 19-20 of Procedure 2).

Finally, the top-k answers are obtained as the graphs in Q_{cur} when the depth-first mining process terminates.

5 Performance Evaluation

In this section, we evaluate the performance of our algorithm. We compare our algorithm, TopCor, with CGSearch [11], which is the only existing work that finds the set of correlative graphs for a given query, with respect to a *user-specified* correlation threshold θ. For both TopCor and CGSearch, we use FG-index [6] as the graph index to obtain the projected database $\mathcal{D}_q$ and the support $supp(g)$ of a candidate graph g. The parameters in FG-index are set as the default values suggested in [6]. We run the experiments on a linux machine with an AMD Opteron 248 CPU and 4GB RAM.

Datasets: We evaluate the performance of our algorithm on both real and synthetic datasets. We use real datasets from the NCI Open Database and we name them as *NCI datasets*. The NCI datasets contain the compound structures of cancer and AIDS data (detailed characteristics of the data can be found from their web-

site[1]), and we extract six datasets that consist of 10K, 20K, 40K, 60K, 80K, and 100K graphs, respectively. We also generate four synthetic datasets[2] of 100K graphs that have average densities[3] of 0.05, 0.1, 0.15 and 0.2, respectively, because the average density of the graphs in the NCI datasets is only 0.1.

Query-sets: For each dataset, we randomly generate four sets of queries, F_1, F_2, F_3 and F_4, each of which contains 100 query graphs with support values ranging in $[0.001, 0.005]$, $(0.005, 0.01]$, $(0.01, 0.03]$ and $(0.03, 1)$. Assume that commonness is related to the support value, these query-sets correspond to *very rare*, *rare*, *rare-to-common*, *common* queries. But for clarity of presentation (there are 8 lines in a figure otherwise), we combine the four query-sets to represent a more general set of queries for the experiments in Sections 5.3-5.5.

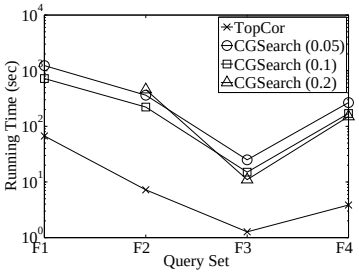
5.1 Effectiveness of Three Key Techniques We first assess the effectiveness of the three key techniques in TopCor by disabling each of them in TopCor. We compute the top-10 answers on the NCI dataset with 100K graphs. Table 1 reports the average query response time and peak memory consumption for discovering the top-10 answers, for the query-set F_3 (only the complete version of TopCor is able to obtain the results for F_1 and F_2). The result shows that without applying either Theorem 4.1 or 4.2, the algorithm immediately becomes infeasible. This justifies that both the correlation checking mechanism by ϕ_{max} and the pruning criteria for the mining process by $\varsigma(\phi_{min})$ are essential to the efficient discovery of top-k answers. In addition, applying Rules 4.1-4.5 also tremendously speeds up the mining process by over an order of magnitude. Thus, the result clearly demonstrates the significance of the three key techniques in TopCor.

Table 1: Effectiveness of Pruning in TopCor

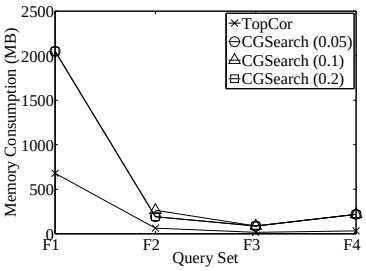
	Response Time	Peak Memory
TopCor	7.14 sec	65 MB
Without applying Theorem 4.1	$>$ 3 hr	$>$ 1 GB
Without applying Theorem 4.2	$>$ 3 hr	$>$ 1 GB
Without applying Rules 4.1-4.5	230 sec	70 MB

5.2 Performance on Different Query-Sets We now test the performance of TopCor for processing queries with different support ranges, corresponding to *very rare*, *rare*, *rare-to-common*, *common* queries. We use the NCI dataset with 10K graphs and set $k = 50$.

[1]http://cactus.nci.nih.gov/ncidb2/download.html.
[2]We use GraphGen: http://www.cse.ust.hk/graphgen.
[3]Density of a graph = $\frac{Size\ of\ the\ graph}{Size\ of\ its\ complete\ graph}$.



(a) Average Response Time



(b) Peak Memory Consumption

Figure 5: Performance on Different Query-Sets

CGSearch for top-k: To use CGSearch for computing the top-k answers, we need to specify the value of θ corresponding to the top-k answers. However, it is difficult to guess the correct θ for different queries. Thus, in our experiments, we set $\theta = (1 - s * i)$, where $i = 1, 2, 3, \ldots$ and s is the amount that θ is decreased for each step i, until the top-k answers are found by CGSearch. We also remark that it may seem natural to apply the binary search strategy to find the best value of θ. However, it does not really work because the set of answer graphs at a low value of θ covers that at a high θ. Once a low value of θ is tested, there is no need to test higher θ values. Moreover, a low θ can easily result in long processing time for CGSearch, which is definitely undesirable.

We report the results for $s = 0.05$, 0.1, and 0.2, which give the best performance of CGSearch among other values of s. We denote them as CGSearch(s) in the figures. Figure 5 presents the average query response time and peak memory consumption of TopCor and CGSearch(s). The results show that TopCor is over an order of magnitude faster than CGSearch(s) for processing all query-sets, and also consumes significantly less memory, especially for queries of low support.

The results also show that TopCor uses more time and memory for processing F_1 queries than others. This is because a query $q \in F_1$ has a small $supp(q)$, which means a small $\varsigma(\phi_{min})$ according to Eq. (4.4). Therefore, F_1 queries have a much larger number of candidates due to the small value of the pruning threshold $\varsigma(\phi_{min})$, which results in longer processing time. How-

ever, the same trend is also observed for CGSearch(s).

We also observe an unusual trend on the performance of processing F_4 queries using both TopCor and CGSearch(s). Our examination finds that many F_4 queries have very high support and hence a much larger projected database (some are even close to the entire database), which results in a higher cost of computing the support of a graph in the projected database.

Among the three values of s for CGSearch(s), we find that $s = 0.1$ gives the best average performance. This is because $s = 0.05$ results in too many iterations in running CGSearch to obtain the top-k answers, while $s = 0.2$ lowers the value of θ too rapidly and returns much more than k answers. We find that CGSearch(0.05) is only more efficient than CGSearch(0.1) for a small subset of the queries and therefore, on average, CGSearch(0.05) is always worse than CGSearch(0.1). CGSearch(0.2) is sometimes more efficient but can also be much worse than CGSearch(0.1); in particular, CGSearch(0.2) for processing F_1 is not reported in Figure 5 because the result is exponentially worse (even drawn in the logarithmic scale). Thus, for the subsequent experiments, we only report the results for CGSearch(0.1).

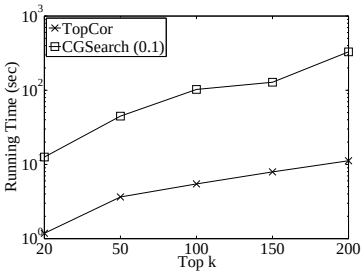
The results of this experiment clearly demonstrate the advantages of an algorithm that finds the top-k correlative graphs automatically and directly, without the need of any user-specified thresholds that can be hardly determined for different queries and datasets.

5.3 Performance on Varying k We then assess the performance of varying the value of k from 20 to 200. We use the NCI dataset with 10K graphs and combine the four query-sets, F_1 to F_4, to represent a more general set of queries.

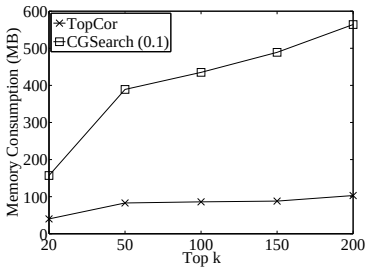
We present the average query response time and peak memory consumption of both TopCor and CGSearch(0.1) in Figure 6. The results show that TopCor is over an order of magnitude faster than CGSearch(0.1) and also consumes significantly less memory for values of k. TopCor is also more stable than CGSearch(0.1) since the difference in performance becomes more obvious when k becomes bigger. This result is because CGSearch needs to find the right θ step-wise, while TopCor finds the top-k answers directly and can more effectively prune the unpromising search space.

5.4 Performance on Varying Dataset Sizes We now evaluate TopCor by varying the size of the NCI dataset from 20K to 100K graphs. We set $k = 50$ and also combine the four query-sets, F_1 to F_4.

Figure 7 reports the average query response time and peak memory consumption of both TopCor and

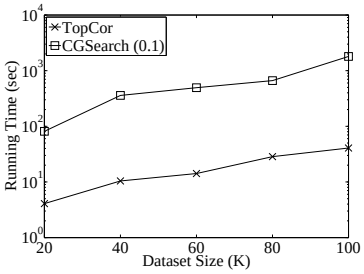


(a) Average Response Time

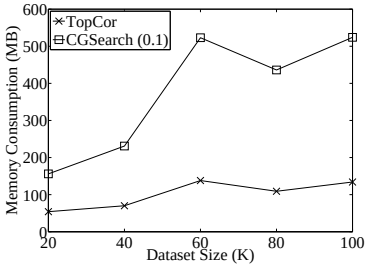


(b) Peak Memory Consumption

Figure 6: Performance on Different k



(a) Average Response Time

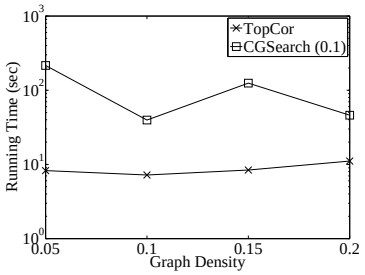


(b) Peak Memory Consumption

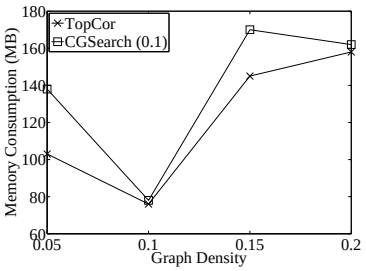
Figure 7: Performance on Different Dataset Sizes

CGSearch(0.1) for different dataset sizes. The results are consistent with those of the previous experiments that TopCor is over an order of magnitude faster than CGSearch(0.1) and also consumes significantly less memory, for all dataset sizes.

However, we notice that there is an upheaval in the memory consumption at the dataset size of 60K graphs. We examine the queries and find that for one particular



(a) Average Response Time



(b) Peak Memory Consumption

Figure 8: Performance on Different Graph Densities

query, the top-50 answers span over a wide range of ϕ values and hence both TopCor and CGSearch(0.1) generate a much larger number of candidates than usual, which results in much higher memory consumption. Since we record the peak memory consumption rather than the average, it creates the upheaval in the curve for memory consumption in Figure 7(b). This abnormal result is not revealed in Figure 7(a), for both TopCor and CGSearch(0.1), because the query response time is averaged over all the queries.

5.5 Performance on Varying Graph Densities

We now assess the effect of graph density on the performance of TopCor using the synthetic datasets. We set $k = 50$ and use the combined query set.

We report the performance for both TopCor and CGSearch(0.1) in Figure 8. The results show that TopCor is very stable in response time, while that of CGSearch(0.1) varies greatly. CGSearch(0.1) is not stable because it cannot effectively prune the unpromising search space for finding the top-k answers as does TopCor. As a result, TopCor outperforms CGSearch(0.1) by about an order of magnitude on average.

The memory consumption of TopCor and CGSearch(0.1) follows a similar trend. Moreover, the difference in the memory consumption of TopCor and CGSearch(0.1) for processing the synthetic datasets is not as big as for processing the real datasets. One reason is that there are less abnormal queries for the synthetic datasets as we observe for the real datasets.

In general, the algorithms consume more memory when the graph density increases. The exception that the memory consumption is not the lowest when the density is equal to 0.05 is because at the density of 0.05, the top-50 correlative graphs have much lower ϕ values than those of the other densities. Since a lower ϕ implies a lower pruning threshold $\varsigma(\phi_{min})$ (as indicated in Eq. (4.4)) and hence a larger number of candidates, and the memory consumption increases as a result.

6 Related Work

Correlation mining has been widely studied in various contexts. In market-basket data, many studies [4, 15, 14, 16, 27, 26, 25] aimed to discover correlative patterns that are formed by basket items. A number of correlation measures were proposed or adopted to define different correlative patterns, including the χ^2 test [4, 15], the interest measure [4], the m-pattern measure [14], any-confidence [16], all-confidence (or equivalently h-confidence) [16, 27], bond [16], Pearson's correlation coefficient [26, 25], etc. All the above work set a threshold θ for the correlation measure, except [25], which studied the top-k mining. Our work is incomparable to [25] due to different nature of market-basket data and graph data. In stream data [22, 19], lagged correlation based on Pearson's correlation coefficient was proposed to study the relationship between time series. In multimedia data [18], correlation between multimedia objects was studied for the task of automatic captioning of media data. In the context of graph databases, there are only two existing studies on correlation mining, the stepwise correlated pattern mining in [5] and CGSearch in [11] with its extended version in [12]. Although the work of [5] also studied top-k correlated graph mining, their algorithm is specifically designed for graph classification by investigating the correlation of a subgraph with a class attribute. Our work, on the other hand, studies the correlation between a subgraph with any given query graph, which is more general. As for CGSearch, we have shown in the experiments that our TopCor algorithm significantly outperforms it.

Our work is also related to graph pattern mining, which mainly focuses on mining FGs [10, 13, 28, 8] and its compact representations, CFGs [29] and MFGs [9, 24]. There are two typical search strategies of the mining algorithms: breadth-first search (also called Apriori-like levelwise search) [10, 13] and depth-first search [28, 8, 29, 9, 24]. These studies discovered the whole set of frequent subgraphs from the database without investigating their inter-relationship. In this paper, we study the correlation between subgraphs, which reveals their occurrence dependencies. Frequent subgraph mining is used as a module in our algorithm to explore candidate graphs.

7 Conclusions

In this paper, we study the problem of mining top-k correlative graph patterns. We develop three key techniques to address the problem. First, we devise a correlation checking mechanism, by which we determine whether a candidate is a potential top-k answer or is an unpromising result that can be pruned, and more effectively, pruned together with its supergraphs. Second, we derive a pruning criteria for the mining process which is dynamically refined with the current top-k answers, thereby allowing us to quickly prune a large portion of the search space. Third, we develop a set of useful rules for efficient candidate exploration, by which we minimize the execution of the expensive operations during the mining process. We integrate the three key techniques and propose an efficient algorithm, TopCor, for mining the top-k correlative graphs. Our extensive experiments show that all these three key techniques are essential to the efficiency of computing the top-k answers. We also show that TopCor is significantly more efficient than the threshold-based correlative graph mining algorithm, CGSearch, for a wide range of query sets, database sizes, values of k, and graph densities.

Acknowledgement. The project was supported by the grant of RGC Hong Kong SAR, China (No. CUHK419008).

References

[1] National library of medicine. *http://chem.sis.nlm.nih.gov/chemidplus*.

[2] The International Network for Social Network Analysis. *http://www.insna.org/*.

[3] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[4] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. In *SIGMOD*, pages 265–276, 1997.

[5] B. Bringmann, A. Zimmermann, L. D. Raedt, and S. Nijssen. Don't be afraid of simpler patterns. In *PKDD*, pages 55–66, 2006.

[6] J. Cheng, Y. Ke, and W. Ng. FG-Index: Towards verification-free query processing on graph databses. *Proc. of SIGMOD*, 2007.

[7] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. of STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[8] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proc. of ICDM*, page 549, 2003.

[9] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *Proc. of KDD*, pages 581–586, 2004.

[10] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. of PKDD*, pages 13–23, 2000.

[11] Y. Ke, J. Cheng, and W. Ng. Correlation search in graph databases. In *Proc. of KDD*, pages 390–399, New York, NY, USA, 2007. ACM.

[12] Y. Ke, J. Cheng, and W. Ng. Efficient correlation search from graph databases. *To appear in IEEE TKDE*, 2008.

[13] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. of ICDM*, pages 313–320, 2001.

[14] S. Ma and J. L. Hellerstein. Mining mutually dependent patterns. In *ICDM*, pages 409–416, 2001.

[15] S. Morishita and J. Sese. Traversing itemset lattice with statistical metric pruning. In *Proc. of PODS*, pages 226–236, 2000.

[16] E. R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE TKDE*, 15(1):57–69, 2003.

[17] R. T. Paine. Food webs: Linkage, interaction strength and community infrastructure. *The Journal of Animal Ecology*, 49(3):667–685, 1980.

[18] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *Proc. of KDD*, pages 653–658, 2004.

[19] S. Papadimitriou, J. Sun, and P. S. Yu. Local correlation tracking in time series. In *ICDM*, pages 456–465, 2006.

[20] S. Raghavan and H. Garcia-Molina. Representing Web graphs. In *Proc. of ICDE*, pages 405–416, 2003.

[21] H. Reynolds. *The analysis of cross-classifications*. The Free Press, New York, 1977.

[22] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD Conference*, pages 599–610, 2005.

[23] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proc. of KDD*, pages 32–41, 2002.

[24] L. T. Thomas, S. R. Valluri, and K. Karlapalem. Margin: Maximal frequent subgraph mining. In *ICDM*, pages 1097–1101, 2006.

[25] H. Xiong, M. Brodie, and S. Ma. Top-cop: Mining top-k strongly correlated pairs in large databases. In *ICDM*, pages 1162–1166, 2006.

[26] H. Xiong, S. Shekhar, P.-N. Tan, and V. Kumar. TAPER: A two-step approach for all-strong-pairs correlation query in large databases. *IEEE TKDE*, 18(4):493–508, 2006.

[27] H. Xiong, P.-N. Tan, and V. Kumar. Hyperclique pattern discovery. *DMKD*, 13(2):219–242, 2006.

[28] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. of ICDM*, page 721, 2002.

[29] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *Proc. of KDD*, pages 286–295, 2003.

[30] G. U. Yule. On the methods of measuring association between two attributes. *Journal of the Royal Statistical Society*, 75(6):579–652, 1912.