Correlation Search in Graph Databases

Yiping Ke
keyiping@cse.ust.hk

James Cheng
csjames@cse.ust.hk

Wilfred Ng
wilfred@cse.ust.hk

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

ABSTRACT

Correlation mining has gained great success in many application domains for its ability to capture the underlying dependency between objects. However, the research of correlation mining from graph databases is still lacking despite the fact that graph data, especially in various scientific domains, proliferate in recent years. In this paper, we propose a new problem of correlation mining from graph databases, called Correlated Graph Search (CGS). CGS adopts Pearson's correlation coefficient as a correlation measure to take into consideration the occurrence distributions of graphs. However, the problem poses significant challenges, since every subgraph of a graph in the database is a candidate but the number of subgraphs is exponential. We derive two necessary conditions which set bounds on the occurrence probability of a candidate in the database. With this result, we design an efficient algorithm that operates on a much smaller projected database and thus we are able to obtain a significantly smaller set of candidates. To further improve the efficiency, we develop three heuristic rules and apply them on the candidate set to further reduce the search space. Our extensive experiments demonstrate the effectiveness of our method on candidate reduction. The results also justify the efficiency of our algorithm in mining correlations from large real and synthetic datasets.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications - Data Mining

General Terms: Algorithms

Keywords: Correlation, Graph Databases, Pearson's Correlation Coefficient

1. INTRODUCTION

Correlation mining is recognized as one of the most important data mining tasks for its capability of identifying the underlying dependency between objects. It has a wide range of application domains and has been studied extensively on market-basket databases [5, 13, 15, 24, 23, 29], quantitative databases [11], multimedia databases [16], data streams [20], and many more. However, little attention has been paid to mining correlations from graph databases, in spite of the popularity of graph data model pertaining to various domains, such as biology [4, 10], chemistry [2], social science [3], the Web [17] and XML [1].

In this paper, we study a new problem of mining correlations from graph databases. We propose to use *Pearson's correlation coefficient* [21] to measure the correlation between a *query* graph and an *answer* graph. We formulate this mining problem, named *Correlated Graph Search* (*CGS*), as follows. Given a graph database $\mathcal{D}$ that consists of N graphs, a query graph q and a minimum correlation threshold θ, the problem of CGS is to *find all graphs whose Pearson's correlation coefficient wrt q is no less than θ*.

Pearson's correlation coefficient is shown to be one of the most desirable correlation measures in [21] for its ability to capture the departure of two variables from independence. It has been widely used to describe the strength of correlation among boolean variables in transaction databases [21, 23, 29]. This motivates us to apply the measure in the context of graph databases. However, graph mining is a much harder problem due to the high complexity of graph operations (e.g., *subgraph isomorphism testing* is NP-complete [7]). The difficulty of the problem is further compounded by the fact that the search space of CGS is often large, since a graph consists of exponentially many subgraphs and each subgraph of a graph in $\mathcal{D}$ can be a candidate graph. Thus, it poses great challenges to tackle the problem of CGS.

How can we reduce the large search space of CGS and avoid as many expensive graph operations as possible? We investigate the property of Pearson's correlation coefficient and derive two necessary conditions for the correlation condition to be satisfied. More specifically, we derive the lower bound and upper bound of the occurrence probability (also called *support*), $supp(g)$, of a candidate graph g. This effectively reduces the search space to be the set of *Frequent subGraphs* (*FGs*) [12] in $\mathcal{D}$ with the support values between the lower and upper bounds of $supp(g)$.

However, mining FGs from $\mathcal{D}$ is still expensive when the lower bound of $supp(g)$ is low or $\mathcal{D}$ is large. Moreover, we still have a large number of candidates and the solution is not scalable. Thus, we need to further reduce the number of candidates as well as address the scalability problem. Our solution to this problem is as follows.

Let $\mathcal{D}_q$ be the projected database of $\mathcal{D}$ on q, which is the set of all graphs in $\mathcal{D}$ that are supergraphs of q. We prove that, the set of FGs mined from $\mathcal{D}_q$ using $\frac{lowerbound(supp(g))}{supp(q)}$

KDD'07, August 12–15, 2007, San Jose, California, USA.

as the minimum support threshold is *complete* wrt the answer set. Since $\mathcal{D}_q$ is in general much smaller than $\mathcal{D}$ while $\frac{lowerbound(supp(g))}{supp(q)}$ is greater than $lowerbound(supp(g))$, our finding not only saves the computational cost for generating the candidate set, but also significantly reduces the number of candidates. Furthermore, we develop three heuristic rules to be applied on the candidate set to identify the graphs that are guaranteed to be in the answer set, as well as to prune the graphs that are guaranteed to be false positives.

In addition to the formulation of the new CGS problem and its efficient solution, the significance of our work also lies in its close connection to graph similarity search, which is an important research area of graph querying. There are two types of similarity: *structural similarity* (i.e., two graphs are similar in structure) and *statistical similarity* (i.e., the occurrence distributions of two graphs are similar).

Existing work [8, 18, 27, 22] mainly focuses on structural similarity search. However, in many applications, two graphs that are structurally dissimilar but always appear together in a graph in $\mathcal{D}$ may be more interesting. For example, in chemistry, *isomers* refer to molecules with the same chemical formula and similar structures. The chemical properties of isomers can be quite different due to different positions of atoms and functional groups. Consider the case that the chemist needs to find some molecule that shares similar chemical properties of a given molecule. Structural similarity search is not relevant, since it mostly returns isomers of the given molecule that have similar structures but different chemical properties, which is undesirable. On the contrary, CGS is able to obtain the molecules that share similar chemical properties but may or may not have similar structures to the given molecule. Therefore, our proposed CGS solves an orthogonal problem of structural similarity search.

Our extensive experiments on both real and synthetic datasets show that our algorithm, called *CGSearch*, achieves short response time for various queries with relatively small memory consumption. Compared with the approach whose candidate set is generated from the whole database with a support range, CGSearch is orders of magnitude faster and consumes up to 41 times less memory. The effectiveness of the candidate generation from the projected database and three heuristic rules is also demonstrated.

Contributions. The specific contributions of the paper are stated as follows.

- We formulate the new problem of correlation search in graph databases, which takes into account the occurrence distributions of graphs using Pearson's correlation coefficient.
- We derive theoretical bounds for the support of a candidate graph, which reduces the search space considerably.
- We propose to generate the candidate set by mining FGs from the projected database of the query graph. Three heuristic rules are developed to further reduce the size of the candidate set.
- We present an efficient algorithm to solve the problem of CGS. We also prove the soundness and completeness of the query results returned by the algorithm.
- A comprehensive set of experiments is conducted to verify the efficiency of the algorithm, and the effectiveness of the candidate generation and the heuristic rules.

Organization. We give preliminaries in Section 2. We define the CGS problem in Section 3. We propose the effective candidate generation from a projected database in Section 4. We present the algorithm, as well as the three heuristic rules, in Section 5. Then, we analyze the performance study in Section 6. Finally, we discuss related work in Section 7 and conclude our paper in Section 8.

2. PRELIMINARIES

In this paper, we restrict our discussion on *undirected, labelled connected graphs* (or simply *graphs* hereinafter), since most of the interesting graphs in practice are connected graphs; while our method can be easily extended to process directed and unlabelled graphs.

A graph g is defined as a 4-tuple (V, E, L, l), where V is the set of vertices, E is the set of edges, L is the set of labels and l is a labelling function that maps each vertex or edge to a label in L. We define the *size* of a graph g as $size(g) = |E(g)|$.

Given two graphs, $g = (V, E, L, l)$ and $g' = (V', E', L', l')$, a *subgraph isomorphism* from g to g' is an injective function f: $V \rightarrow V'$, such that $\forall (u, v) \in E$, $(f(u), f(v)) \in E'$, $l(u) = l'(f(u))$, $l(v) = l'(f(v))$, and $l(u, v) = l'(f(u), f(v))$. The subgraph isomorphism testing is known to be an *NP*-complete problem [7].

A graph g is called a *subgraph* of another graph g' (or g' is a *supergraph* of g), denoted as $g \subseteq g'$ (or $g' \supseteq g$), if there exists a subgraph isomorphism from g to g'.

Let $\mathcal{D} = \{g_1, g_2, \ldots, g_N\}$ be a *graph database* that consists of N graphs. Given $\mathcal{D}$ 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 $|\mathcal{D}_g|$. The *support* of g in $\mathcal{D}$, denoted as $supp(g; \mathcal{D})$, is defined as $\frac{freq(g;\mathcal{D})}{|\mathcal{D}|}$. A graph g is called a *Frequent subGraph* (FG) [9, 12, 25] in $\mathcal{D}$ if $supp(g; \mathcal{D}) \geq \sigma$, where σ ($0 \leq \sigma \leq 1$) is a user-specified *minimum support threshold*. For simplicity, we use $freq(g)$ and $supp(g)$ to denote the frequency and support of g in $\mathcal{D}$ when there is no confusion. Given two graphs, g_1 and g_2, we define the *joint frequency*, 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 $supp(g_1, g_2) = \frac{freq(g_1,g_2)}{|\mathcal{D}|}$.

The support measure is *anti-monotone*, i.e., if $g_1 \subseteq g_2$, then $supp(g_1) \geq supp(g_2)$. Moreover, by the definition of joint support, we have the following property: $supp(g_1, g_2) \leq supp(g_1)$ and $supp(g_1, g_2) \leq supp(g_2)$.

EXAMPLE *1.* Figure 1 shows a graph database, $\mathcal{D}$, that consists of 10 graphs, $g_1, \ldots, g_{10}$. For clarity of presentation, all the nodes are of the same label (not shown in the figure); while the characters a, b and c represent distinct edge labels.

The graph g_8 is a subgraph of g_2. The projected database of g_8, i.e., $\mathcal{D}_{g_8}$, is $\{g_2, g_3, g_6, g_7, g_8\}$. The frequency of g_8 is computed as $freq(g_8) = |\mathcal{D}_{g_8}| = 5$. The support of g_8 is $supp(g_8) = \frac{freq(g_8)}{|\mathcal{D}|} = 0.5$. As for g_9, we have $\mathcal{D}_{g_9} = \{g_6, g_7, g_9\}$. The joint frequency of g_8 and g_9 is computed as $freq(g_8, g_9) = |\mathcal{D}_{g_8} \cap \mathcal{D}_{g_9}| = |\{g_6, g_7\}| = 2$. The joint support of g_8 and g_9 is $supp(g_8, g_9) = \frac{freq(g_8,g_9)}{|\mathcal{D}|} = 0.2$. ■

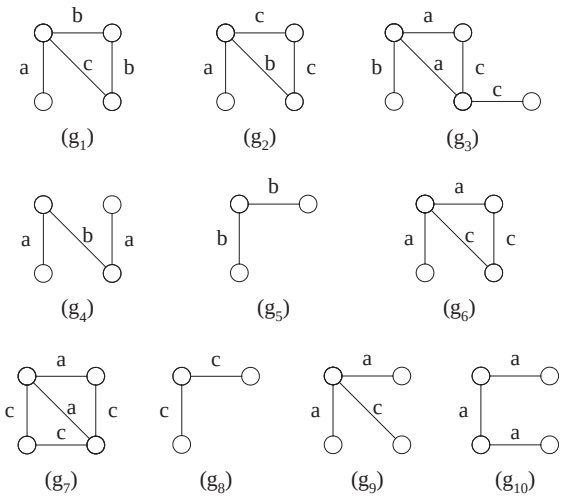


Figure 1: A Graph Database, $\mathcal{D}$

3. PROBLEM DEFINITION

We first define *Pearson's correlation coefficient* [19] for two given graphs. Pearson's correlation coefficient for boolean variables is also known as "*ϕ correlation coefficient*" [28].

DEFINITION 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 follows:*

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

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 range of $\phi(g_1, g_2)$ falls within $[-1, 1]$. If $\phi(g_1, g_2)$ is positive, then g_1 and g_2 are positively correlated; otherwise, g_1 and g_2 are negatively correlated. In this paper, we focus on positively correlated graphs defined as follows.

DEFINITION 2. (CORRELATED GRAPHS) *Two graphs g_1 and g_2 are* correlated *if and only if $\phi(g_1, g_2) \geq \theta$, where θ ($0 < \theta \leq 1$) is a user-specified* minimum correlation threshold.

We now define the correlation mining problem in graph databases as follows.

DEFINITION 3. (CORRELATED GRAPH SEARCH) *Given a graph database $\mathcal{D}$, a* correlation query graph *q and a minimum correlation threshold θ, the problem of* Correlated Graph Search (CGS) *is* to find the set of all graphs that are correlated with q. *The answer set of the CGS problem is defined as $\mathcal{A}_q = \{(g, \mathcal{D}_g) : \phi(q, g) \geq \theta\}$.*

For each correlated graph g of q, we include $\mathcal{D}_g$ in the answer set in order to indicate the distribution of g in $\mathcal{D}$. We also define the set of correlated graphs in the answer set as the *base* of the answer set, denoted as $base(\mathcal{A}_q) = \{g : (g, \mathcal{D}_g) \in \mathcal{A}_q\}$. In the subsequent discussions, a correlation query graph is simply called a *query*.

Table 1 gives the notations used throughout the paper.

Table 1: Notations Used Throughout

Notation	Description
$\mathcal{D}$	a graph database
q	a query graph
θ	a minimum correlation threshold, $0 < \theta \leq 1$
$\phi(q, g)$	Pearson's correlation coefficient of q and g
$\mathcal{A}_q$	the answer set of q
$base(\mathcal{A}_q)$	the base of the answer set
$\mathcal{D}_g$	the projected database of $\mathcal{D}$ on graph g
$freq(g)$, $supp(g)$	the frequency/support of g in $\mathcal{D}$
$freq(q, g)$, $supp(q, g)$	the joint frequency/support of q and g in $\mathcal{D}$
$freq(g; \mathcal{D}_q)$, $supp(g; \mathcal{D}_q)$	the frequency/support of g in $\mathcal{D}_q$
$freq(q, g; \mathcal{D}_q)$, $supp(q, g; \mathcal{D}_q)$	the joint frequency/support of q and g in $\mathcal{D}_q$
$lower(g)$, $upper(g)$	the lower/upper bound of $supp(g)$
$lower(q, g)$, $upper(q, g)$	the lower/upper bound of $supp(q, g)$

4. CANDIDATE GENERATION

A crucial step for solving the problem of CGS is to obtain the set of candidate graphs. Obviously, it is infeasible to test all subgraphs of the graphs in $\mathcal{D}$ because there are exponentially many subgraphs. In this section, we discuss how to effectively select a small set of candidates for a given query.

4.1 Support Bounds of Correlated Graphs

We begin by investigating the bounds on the support of a candidate graph, g, with respect to the support of a query q. We state and prove the bounds in Lemma 1.

LEMMA 1. *If q and g are correlated, then the following bounds of $supp(g)$ hold:*

$$\frac{supp(q)}{\theta^{-2}(1-supp(q))+supp(q)} \leq supp(g) \leq \frac{supp(q)}{\theta^2(1-supp(q))+supp(q)}.$$

PROOF. By the definition of the joint support, we have $supp(q, g) \leq supp(g)$ and $supp(q, g) \leq supp(q)$.

Since q and g are correlated, $\phi(q, g) \geq \theta$. By replacing $supp(q, g)$ with $supp(g)$ in $\phi(q, g)$, we have:

$$\frac{supp(g) - supp(q)supp(g)}{\sqrt{supp(q)supp(g)(1 - supp(q))(1 - supp(g))}} \geq \theta$$

$$\Rightarrow \quad supp(g) \geq \frac{supp(q)}{\theta^{-2}(1 - supp(q)) + supp(q)}.$$

Similarly, by replacing $supp(q, g)$ with $supp(q)$ in $\phi(q, g)$, we obtain the upper bound:

$$supp(g) \leq \frac{supp(q)}{\theta^2(1 - supp(q)) + supp(q)}.$$

□

For simplicity, we use $lower(g)$ and $upper(g)$ to denote the respective lower and upper bounds of $supp(g)$ with respect to q, as given in Lemma 1. The above lemma states a necessary condition for a correlated answer graph. Thus, a candidate graph should have support within the range of $[lower(g), upper(g)]$.

With the result of Lemma 1, we can obtain the candidate set by mining the set of FGs from $\mathcal{D}$ using $lower(g)$ as the minimum support threshold and $upper(g)$ as the maximum support threshold. However, according to the anti-monotone property of the support measure, the graphs with

higher support are always generated before those with lower support, no matter adopting a breadth-first or a depth-first strategy. As a result, the maximum threshold $upper(g)$ is not able to speed up the mining process. Therefore, generating the candidates by mining the FGs from $\mathcal{D}$ with a support range is still not efficient enough, especially when $lower(g)$ is small or $\mathcal{D}$ is large. This motivates us to devise a more efficient and effective approach to generate the candidates.

4.2 Candidate Generation From a Projected Database

From Definition 1, it follows that if $\phi > 0$, then $supp(q, g) > 0$. This means that q and g must appear together in at least one graph in $\mathcal{D}$. This also implies that $\forall g \in base(\mathcal{A}_q)$, g appears in at least one graph in the projected database of q, $\mathcal{D}_q$. Since $\mathcal{D}_q$ is in general much smaller than $\mathcal{D}$, this gives rise to the following natural question: can we mine the candidate set more efficiently from $\mathcal{D}_q$ instead of $\mathcal{D}$?

The challenge is that, however, we need to determine a minimum support threshold that can be used to mine the FGs from $\mathcal{D}_q$, so that no correlated answer graph is missed. Obviously, we cannot use a trivial threshold since it is too expensive. In this subsection, we derive a minimum support threshold which enables us to efficiently compute the candidates from $\mathcal{D}_q$. Our solution is inspired by the following important observation as stated in Lemma 2.

LEMMA 2. *Given a graph g, $supp(g; \mathcal{D}_q) = supp(q, g; \mathcal{D}_q) = \frac{supp(q,g)}{supp(q)}$.*

PROOF. By the definition of the projected database, every graph in $\mathcal{D}_q$ must contain q. Therefore, every graph in $\mathcal{D}_q$ that contains g must also contain q. Thus, $supp(g; \mathcal{D}_q) = supp(q, g; \mathcal{D}_q)$ holds. Since the number of graphs containing both q and g in $\mathcal{D}$ is the same as that in $\mathcal{D}_q$, that is, $freq(q, g) = freq(q, g; \mathcal{D}_q)$, we have $\frac{supp(q,g)}{supp(q)} = \frac{freq(q,g)/|\mathcal{D}|}{freq(q)/|\mathcal{D}|} = \frac{freq(q,g;\mathcal{D}_q)}{|\mathcal{D}_q|} = supp(q, g; \mathcal{D}_q)$. □

Lemma 2 states that the support of a graph g in $\mathcal{D}_q$ is the same as the joint support of q and g in $\mathcal{D}_q$. This prompts us to derive the lower bound and upper bound for $supp(q, g; \mathcal{D}_q)$, given that g is correlated with q. Then, we can use the bounds as the minimum and maximum support thresholds to compute the candidates from $\mathcal{D}_q$.

Since $supp(q, g; \mathcal{D}_q) = \frac{supp(q,g)}{supp(q)}$ by Lemma 2, we try to derive the bounds for $supp(q, g)$.

First, by the definition of the joint support, we obtain the upper bound of $supp(q, g)$ as follows:

$$supp(q, g) \leq supp(q). \tag{1}$$

Then, we construct a lower bound for $supp(q, g)$ from Definition 1. Given $\phi(q, g) \geq \theta$, we have the following inequality:

$$supp(q, g) \geq f(supp(g)), \tag{2}$$

where

$$f(supp(g)) = \theta\sqrt{supp(q)supp(g)(1 - supp(q))(1 - supp(g))} + supp(q)supp(g).$$

The lower bound of $supp(q, g)$ stated in Inequality (2) cannot be directly used, since it is a function of $supp(g)$, where g is exactly what we try to get using $supp(q, g)$. However, since we have obtained the range of $supp(g)$, i.e., $[lower(g), upper(g)]$ as stated in Lemma 1, we now show that this range can be used in Inequality (2) to obtain the lower bound of $supp(q, g)$.

By investigating the property of the function f, we find that, f is monotonically increasing with $supp(g)$ in the range of $[lower(g), upper(g)]$. Therefore, by substituting $supp(g)$ with $lower(g)$ in Inequality (2), we obtain the lower bound of $supp(q, g)$. We state and prove the bounds of $supp(q, g)$ in the following lemma.

LEMMA 3. *If q and g are correlated, then the following bounds of $supp(q, g)$ hold:*

$$\frac{supp(q)}{\theta^{-2}(1 - supp(q)) + supp(q)} \leq supp(q, g) \leq supp(q).$$

PROOF. The upper bound follows by the definition of the joint support.

To show that the lower bound holds, we need to prove that the function f is monotonically increasing within the bounds of $supp(g)$ given in Lemma 1. This can be done by applying differentiation to f with respect to $supp(g)$ as follows:

$$f'(supp(g)) = \frac{\theta \cdot supp(q)(1 - supp(q))(1 - 2 \cdot supp(g))}{2\sqrt{supp(q)supp(g)(1 - supp(q))(1 - supp(g))}} + supp(q).$$

Thus, we need to prove that within $[lower(g), upper(g)]$, $f'(supp(g)) \geq 0$ or equivalently the following inequality:

$$\frac{1 - 2 \cdot supp(g)}{\sqrt{supp(g)(1 - supp(g))}} \geq -\frac{2}{\theta}\sqrt{\frac{supp(q)}{1 - supp(q)}}. \tag{3}$$

First, if $supp(g) \leq upper(g) \leq 0.5$, then $(1 - 2 \cdot supp(g)) \geq 0$ and hence $f'(supp(g)) \geq 0$.

Now we consider the case when $upper(g) \geq supp(g) > 0.5$. Since the left hand side of Inequality (3) is less than 0, we take square on both sides of Inequality (3) and obtain:

$$\frac{(1 - 2 \cdot supp(g))^2}{supp(g)(1 - supp(g))} \leq \frac{4 \cdot supp(q)}{\theta^2(1 - supp(q))}$$

$$\Leftrightarrow \quad a \cdot (supp(g))^2 - a \cdot supp(g) + \theta^2(1 - supp(q)) \leq 0, \tag{4}$$

where $a = 4\theta^2(1 - supp(q)) + 4supp(q)$.

The left hand side of Inequality (4) is a quadratic function, which is monotonically increasing within the range of $[0.5, \infty]$. Since $0.5 < supp(g) \leq upper(g)$, we replace $supp(g)$ with $upper(g)$ in this quadratic function:

$$\begin{aligned} & a \cdot (upper(g))^2 - a \cdot upper(g) + \theta^2(1 - supp(q)) \\ = \ & \theta^2(1 - supp(q))(-4 \cdot upper(g) + 1) \\ < \ & \theta^2(1 - supp(q))(-4 \times 0.5 + 1) \qquad (\text{Since } upper(g) > 0.5) \\ < \ & 0. \end{aligned}$$

Therefore, when $0.5 < supp(g) \leq upper(g)$, Inequality (4) holds and hence $f'(supp(g)) \geq 0$.

Thus, f is monotonically increasing within the range of $[lower(g), upper(g)]$. By substituting $supp(g)$ with $lower(g)$ in Inequality (2), the lower bound of $supp(q, g)$ thus follows:

$$\begin{aligned} supp(q, g) \ & \geq \ f(supp(g)) \\ & \geq \ f(\frac{supp(q)}{\theta^{-2}(1 - supp(q)) + supp(q)}) \\ & = \ \frac{supp(q)}{\theta^{-2}(1 - supp(q)) + supp(q)}. \end{aligned}$$

□

We use $lower(q, g)$ and $upper(q, g)$ to denote the lower and upper bounds of $supp(q, g)$ with respect to q, as given in Lemma 3.

With the results of Lemmas 2 and 3, we propose to generate the candidates by mining FGs from $\mathcal{D}_q$ using $\frac{lower(q,g)}{supp(q)}$ as the minimum support threshold. A generated candidate set, $\mathcal{C}$, is said to be *complete* with respect to q, if $\forall g \in base(\mathcal{A}_q)$, $g \in \mathcal{C}$. We establish the result of completeness by the following theorem.

THEOREM 1. *Let $\mathcal{C}$ be the set of FGs mined from $\mathcal{D}_q$ with the minimum support threshold of $\frac{lower(q,g)}{supp(q)}$. Then, $\mathcal{C}$ is* complete *with respect to q.*

PROOF. Let $g \in base(\mathcal{A}_q)$. Since $\phi(q, g) \geq \theta$, it follows that $lower(q, g) \leq supp(q, g) \leq upper(q, g)$ by Lemma 3. By dividing the inequality by $supp(q)$ on all the expressions, we have $\frac{lower(q,g)}{supp(q)} \leq \frac{supp(q,g)}{supp(q)} \leq 1$. By Lemma 2, we have $\frac{lower(q,g)}{supp(q)} \leq supp(g; \mathcal{D}_q) \leq 1$. The result $g \in \mathcal{C}$ follows, since $\mathcal{C}$ is the set of FGs mined from $\mathcal{D}_q$ using $\frac{lower(q,g)}{supp(q)}$ as the minimum support threshold. □

The result of Theorem 1 is significant, since it implies that we are now able to mine the set of candidate graphs from a much smaller projected database $\mathcal{D}_q$ (compared with $\mathcal{D}$) with a greater minimum support threshold $\frac{lower(q,g)}{supp(q)}$ (compared with $lower(g)$ which is equal to $lower(q, g)$, as shown in Lemmas 1 and 3).

5. CORRELATED GRAPH SEARCH

In this section, we present our solution to the CGS problem. The framework of the solution consists of the following four steps.

1. Obtain the projected database $\mathcal{D}_q$ of q.
2. Mine the set of candidate graphs $\mathcal{C}$ from $\mathcal{D}_q$, using $\frac{lower(q,g)}{supp(q)}$ as the minimum support threshold.
3. Refine $\mathcal{C}$ by three heuristic rules.
4. For each candidate graph $g \in \mathcal{C}$,
   (a) Obtain $\mathcal{D}_g$.
   (b) Add $(g, \mathcal{D}_g)$ to $\mathcal{A}_q$ if $\phi(q, g) \geq \theta$.

Step 1 obtains the projected database of q. This step can be efficiently performed using any existing graph indexing technique [26, 6] that can be used to obtain the projected database of a given graph.

Step 2 mines the set of FGs from $\mathcal{D}_q$ using some existing FG mining algorithm [12, 25, 14]. The minimum support threshold is determined by Theorem 1. The set of FGs forms the candidate set, $\mathcal{C}$. For each graph $g \in \mathcal{C}$, the set of graphs in $\mathcal{D}_q$ that contain g is also obtained by the FG mining process.

In Step 3, three heuristic rules are applied on $\mathcal{C}$ to further prune the graphs that are guaranteed to be false positives, as well as to identify the graphs that are guaranteed to be in the answer set.

Finally, for each remaining graph g in $\mathcal{C}$, Step 4(a) obtains $\mathcal{D}_g$ using the same indexing technique as in Step 1. Then Step 4(b) checks the correlation condition of g with respect to q to produce the answer set. Note that, the joint support of q and g, which is needed for computing $\phi(q, g)$, is computed as $(supp(g; \mathcal{D}_q) \cdot supp(q))$ by Lemma 2.

In the remainder of this section, we present three heuristic rules and our algorithm, *CGSearch*, to solve the problem of CGS.

5.1 Heuristic Rules

To check whether each graph g in $\mathcal{C}$ is correlated with q, a query operation to obtain $\mathcal{D}_g$ is needed for each candidate (Step 4(a)). The step can be expensive if the candidate set is large. Thus, we develop three heuristic rules to further refine the candidate set.

First, if we are able to identify the graphs that are guaranteed to be correlated with q before processing Step 4, we can save the cost of verifying the result. We achieve this goal by Heuristic 1.

HEURISTIC 1. *Given a graph g, if $g \in \mathcal{C}$ and $g \supseteq q$, then $g \in base(\mathcal{A}_q)$.*

PROOF. Since $g \supseteq q$, we have $supp(q, g) = supp(g)$. Moreover, since $g \in \mathcal{C}$, we have $supp(g, q; \mathcal{D}_q) \geq \frac{lower(q,g)}{supp(q)}$. By Lemma 2, we further have $supp(q, g) \geq lower(q, g)$.

By replacing $supp(q, g)$ with $supp(g)$ in $\phi(q, g)$, we have

$$\phi(q, g) = \sqrt{\frac{1 - supp(q)}{supp(q)}} \cdot \sqrt{\frac{supp(g)}{1 - supp(g)}}$$

Now, ϕ is monotonically increasing with $supp(g)$, and $supp(g) = supp(q, g) \geq lower(q, g)$. We replace $supp(g)$ with its lower bound of $lower(q, g) = \frac{supp(q)}{\theta^{-2}(1-supp(q))+supp(q)}$ in $\phi(q, g)$. Then, we have the following:

$$\begin{aligned} \phi(q, g) &\geq \sqrt{\frac{1 - supp(q)}{supp(q)}} \cdot \sqrt{\frac{\theta^2 supp(q)}{1 - supp(q)}} \\ &\geq \theta. \end{aligned}$$

Therefore, $g \in base(\mathcal{A}_q)$. □

Based on Heuristic 1, if we find that a graph g in the candidate set is a supergraph of q, we can add $(g, \mathcal{D}_g)$ into the answer set without checking the correlation condition. In addition, since g is a supergraph of q, $\mathcal{D}_g$ can be obtained when g is mined from the projected database $\mathcal{D}_q$.

We next seek to save the cost of unrewarding query operations by pruning those candidate graphs that are guaranteed to be uncorrelated with q. For this purpose, we develop the following two heuristic rules.

Before introducing Heuristic 2, we establish the following lemma, which describes a useful property of the function ϕ.

LEMMA 4. *If both $supp(q)$ and $supp(q, g)$ are fixed, then $\phi(q, g)$ is monotonically decreasing with $supp(g)$.*

PROOF. Since both $supp(q)$ and $supp(q, g)$ are fixed, we first simplify ϕ for clarity of presentation. Let $x = supp(g)$, $a = supp(q, g)$, $b = supp(q)$, and $c = supp(q)(1 - supp(q))$. Then we have

$$\phi(x) = \frac{a - b \cdot x}{\sqrt{c \cdot x(1 - x)}}.$$

The derivative of ϕ at x is given as follows:

$$\phi'(x) = \frac{1}{\sqrt{c}} \cdot \frac{(2a - b)x - a}{2x(1 - x)\sqrt{x(1 - x)}}.$$

Since $0 \leq x \leq 1$, we have $x(1-x) \geq 0$. Thus, the sign of $\phi'(x)$ depends on the sign of $((2a-b)x-a)$. Since $((2a-b)x-a)$ is a linear function, we can derive its extreme values by replacing 0 and 1 of x into the function. The two extreme values of $((2a-b)x-a)$ are $(-a)$ and $(a-b)$, both of which are non-positive, since $a \geq 0$ and $a \leq b$. Therefore, we have $((2a-b)x-a) \leq 0$ and $\phi'(x) \leq 0$. It follows that $\phi(q,g)$ is monotonically decreasing with $supp(g)$. □

HEURISTIC 2. *Given two graphs g_1 and g_2, where $g_1 \supseteq g_2$ and $supp(g_1,q) = supp(g_2,q)$, if $g_1 \notin base(\mathcal{A}_q)$, then $g_2 \notin base(\mathcal{A}_q)$.*

PROOF. Since $g_1 \supseteq g_2$, we have $supp(g_1) \leq supp(g_2)$. Since $supp(g_1,q) = supp(g_2,q)$ and $supp(q)$ is fixed, by Lemma 4, we have $\phi(q,g_1) \geq \phi(q,g_2)$. Since $g_1 \notin base(\mathcal{A}_q)$, we have $\phi(q,g_1) \leq \theta$. Therefore, $\phi(q,g_2) \leq \phi(q,g_1) \leq \theta$. Thus, we have $g_2 \notin base(\mathcal{A}_q)$. □

By Lemma 2, if $supp(g_1,q) = supp(g_2,q)$, then we have $supp(g_1;\mathcal{D}_q) = supp(g_2;\mathcal{D}_q)$. Thus, Heuristic 2 can be applied as follows: if we find that a graph g is uncorrelated with q, we can prune all the subgraphs of g that have the same support as g in $\mathcal{D}_q$.

We now use the function f again to present the third heuristic:

$$f(supp(g_1)) = \theta\sqrt{supp(q)(1-supp(q))supp(g_1)(1-supp(g_1))} + supp(q)supp(g_1).$$

HEURISTIC 3. *Given two graphs g_1 and g_2, where $g_1 \supseteq g_2$, if $supp(g_2,q) < f(supp(g_1))$, then $g_2 \notin base(\mathcal{A}_q)$.*

PROOF. Since $g_1 \supseteq g_2$, we have $supp(g_1) \leq supp(g_2)$. By Lemma 1, the necessary condition for $\phi(q,g_2) \geq \theta$ is that, $supp(g_2)$ should fall within the range $[lower(g_2), upper(g_2)]$. As shown in the proof of Lemma 3, the function f is monotonically increasing within the range $[lower(g_2), upper(g_2)]$. Therefore, we have $supp(g_2,q) < f(supp(g_1)) \leq f(supp(g_2))$. By replacing $supp(g_2,q)$ with $f(supp(g_2))$ in $\phi(q,g_2)$, we have the following derivations:

$$\begin{aligned}\phi(q,g_2) &< \frac{f(supp(g_2)) - supp(q)supp(g_2)}{\sqrt{supp(q)supp(g_2)(1-supp(q))(1-supp(g_2))}} \\ &= \frac{\theta\sqrt{supp(q)supp(g_2)(1-supp(q))(1-supp(g_2))}}{\sqrt{supp(q)supp(g_2)(1-supp(q))(1-supp(g_2))}} \\ &= \theta.\end{aligned}$$

Therefore, we have $g_2 \notin base(\mathcal{A}_q)$.

Note that, $supp(g_2,q) < f(supp(g_1))$ also implies $g_1 \notin base(\mathcal{A}_q)$. This is because $g_1 \supseteq g_2$ implies $supp(g_1,q) \leq supp(g_2,q)$. Therefore, we have $supp(g_1,q) < f(supp(g_1))$. Similarly, by replacing $supp(g_1,q)$ with $f(supp(g_1))$ in $\phi(q,g_1)$, we can have $\phi(q,g_1) < \theta$ and thus $g_1 \notin base(\mathcal{A}_q)$. □

By Lemma 2, we have $supp(g_2,q) = supp(g_2;\mathcal{D}_q) \cdot supp(q)$. Thus, if $supp(g_2,q) < f(supp(g_1))$, then $supp(g_2;\mathcal{D}_q) < \frac{f(supp(g_1))}{supp(q)}$. Thus, Heuristic 3 can be applied as follows: if we find that a graph g is uncorrelated with q, we can prune all the subgraphs of g that have support values less than $\frac{f(supp(g))}{supp(q)}$ in $\mathcal{D}_q$.

5.2 CGSearch Algorithm

Now, we present the CGSearch algorithm. As shown in Algorithm 1, after we obtain the candidate set $\mathcal{C}$ from the projected database $\mathcal{D}_q$ (Lines 1-2), we process each candidate graph in $\mathcal{C}$ according to the descending order of the graph sizes. Then, Lines 4-5 applies Heuristic 1 to include the supergraphs of $q \in \mathcal{C}$ directly into the answer set without performing the query operation (as in Line 7). For other graphs in $\mathcal{C}$, if they are verified to be correlated with q, we include them in the answer set (Lines 8-9); otherwise, Heuristic 2 (Lines 11-12) and Heuristic 3 (Lines 13-14) are applied to further reduce the search space so that the unrewarding query costs for false-positives are saved.

Algorithm 1 *CGSearch*

Input: A graph database $\mathcal{D}$, a query graph q, and a correlation threshold θ.
Output: The answer set $\mathcal{A}_q$.

```
1.  Obtain D_q;
2.  Mine FGs from D_q using lower(q,g)/supp(q) as the minimum
      support threshold and add the FGs to C;
3.  for each graph g ∈ C in size-descending order do
4.    if (g ⊇ q)
5.      Add (g, D_g) to A_q;
6.    else
7.      Obtain D_g;
8.      if (φ(q,g) ≥ θ)
9.        Add (g, D_g) to A_q;
10.     else
11.       H_2 ← {g' ∈ C : g' ⊆ g, supp(g';D_q) = supp(g;D_q)};
12.       C ← C − H_2;
13.       H_3 ← {g' ∈ C : g' ⊆ g, supp(g';D_q) < f(supp(g))/supp(q)};
14.       C ← C − H_3;
```

We now prove the soundness and completeness of the result returned by CGSearch algorithm. In other words, we prove that CGSearch is able to precisely return $\mathcal{A}_q$ with respect to a given q.

THEOREM 2. *The answer set, $\mathcal{A}_q$, returned by Algorithm 1, is* sound *and* complete *with respect to q.*

PROOF. We first prove the soundness. $\forall(g,\mathcal{D}_g) \in \mathcal{A}_q$, $(g,\mathcal{D}_g)$ is added to $\mathcal{A}_q$ in either Line 5 or Line 9. For the case of Line 5, we have proved in Heuristic 1 that g is correlated with q; while for the case of Line 9, the soundness is guaranteed in Line 8. Thus, the soundness of $\mathcal{A}_q$ follows.

It remains to show the completeness. By Theorem 1, the candidate set, $\mathcal{C}$, produced in Line 2 of Algorithm 1 is complete. $\forall g \in \mathcal{C}$, if g is not included in $\mathcal{A}_q$, then $\phi(q,g)$ is checked to be less than θ (Line 10) or g is pruned by Heuristics 2 or 3 (Lines 11-14). For all cases, g is proved to be uncorrelated with q and thus is not in $\mathcal{A}_q$. Therefore, the completeness of $\mathcal{A}_q$ follows. □

EXAMPLE 2. Consider the graph database in Figure 1 and the query q in Figure 2(a). Let $\theta = 0.6$. CGSearch (Line 1) first obtains $\mathcal{D}_q = \{g_1, g_2, g_3, g_4\}$. Thus, we have $supp(q) = 0.4$ and $lower(q,g) = 0.19$. Then, CGSearch (Line 2) mines FGs from $\mathcal{D}_q$ using $\frac{0.19}{0.4} = 0.475$ as the minimum support threshold and obtains 9 candidates, which are shown in Figure 2(b). The number following ":" in the figure is the support of each candidate in $\mathcal{D}_q$.

Since the candidates are sorted in descending order of their size, CGSearch first processes c_1. Since c_1 is a super-

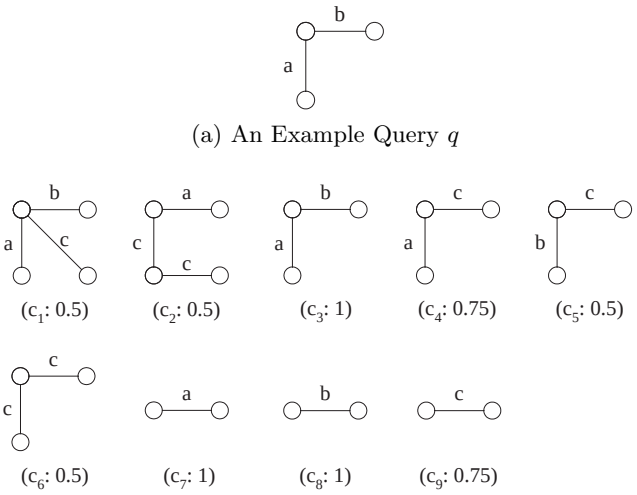
graph of q, $(c_1, \mathcal{D}_{c_1})$ is directly included into $\mathcal{A}_q$ by Heuristic 1. Note that $\mathcal{D}_{c_1} = \{g_1, g_2\}$ can be obtained in the process of mining the candidates from $\mathcal{D}_q$, since c_1 is a supergraph of q.

Then, CGSearch processes c_2 to obtain $\mathcal{D}_{c_2} = \{g_2, g_3, g_6, g_7\}$. Therefore, we have $\phi(q, c_2) = \frac{0.5\times0.4-0.4\times0.4}{\sqrt{0.4\times0.6\times0.4\times0.6}} = 0.17 < \theta$. Then, CGSearch computes $H_2 = \{c_6\}$ since $c_6 \subset c_2$ and $supp(c_6; \mathcal{D}_q) = supp(c_2; \mathcal{D}_q) = 0.5$. CGSearch further computes $H_3 = \{c_4, c_9\}$ since $c_4 \subset c_2$, $c_9 \subset c_2$, and $supp(c_4; \mathcal{D}_q) = supp(c_9; \mathcal{D}_q) = 0.75 < 0.76 = \frac{f(supp(c_2))}{supp(q)}$, as shown in Figure 2(b). Therefore, after processing c_2, $\mathcal{C} = \{c_3, c_5, c_7, c_8\}$.

Similar to c_1, CGSearch finds that c_3 is a supergraph of q and $(c_3, \mathcal{D}_{c_3})$ is directly included into $\mathcal{A}_q$ by Heuristic 1. For c_5, after obtaining $\mathcal{D}_{c_5}$, CGSearch computes $\phi(c_5, q) = 0.61 \geq \theta$, so $(c_5, \mathcal{D}_{c_5})$ is added into $\mathcal{A}_q$. Finally, by querying c_7 and c_8, since $\phi(c_7, q) = 0.4 < \theta$ and $\phi(c_8, q) = 0.82 \geq \theta$, CGSearch adds $(c_8, \mathcal{D}_{c_8})$ into $\mathcal{A}_q$.

Therefore, $\mathcal{A}_q = \{(c_1, \mathcal{D}_{c_1}), (c_3, \mathcal{D}_{c_3}), (c_5, \mathcal{D}_{c_5}), (c_8, \mathcal{D}_{c_8})\}$. Among the 9 candidates, 5 of them do not need to perform correlation verification by applying Heuristics 1 to 3.

When carrying out the exhaustive search, there are 40 subgraphs for such a small and simple graph database. If we generate the candidate set by mining FGs from $\mathcal{D}$ using $lower(g) = 0.19$ and $upper(g) = 0.64$ as support thresholds, there are still 16 graphs in the candidate set. This clearly illustrates that the candidate generation from the projected database indeed significantly reduces the search space. ■



(a) An Example Query q

(b) Candidate Set, $\mathcal{C}$, of q

Figure 2: An Example Query and Its Candidate Set

5.3 Discussions

To apply the three heuristic rules in our algorithm, we need to obtain supergraphs or subgraphs of a given graph (Lines 4, 11 and 13 of Algorithm 1) by testing subgraph isomorphism. However, subgraph isomorphism testing is an expensive operation and we want to avoid it as much as possible. We find that the number of subgraph isomorphism testings can be effectively reduced by using a depth-first FG mining algorithm (such as *gSpan* [25]) for the candidate generation. For a depth-first mining process, the FGs generated can be organized by a prefix tree, in which a child is a supergraph of its parent. Thus, by following the *root-to-leaf paths* (simply called *path*) in the prefix tree, we are able to determine the subgraph-supergraph relationship without performing subgraph isomorphism testing.

If we only follow a path in the prefix tree and do not check the relationship of the graphs that appear in different paths, we are not able to identify all the graphs in H_2 and H_3, as well as all the supergraphs of q. However, we note that there is a trade-off here. On the one hand, if we fully apply the three heuristic rules by cross-checking the graphs in different paths to find all the subgraph-supergraph relationships, more subgraph isomorphism testings have to be performed but less candidates are needed for verification of correlation condition. On the other hand, if we only partially apply the three heuristic rules by simply following the paths in the prefix tree, no subgraph isomorphism testing is needed but more candidates are required for verification. We demonstrate further this trade-off in our experiments.

6. PERFORMANCE EVALUATION

We evaluate the performance of our solution to the CGS problem on both real and synthetic datasets.

6.1 Experimental Settings

The real dataset contains the compound structures of cancer and AIDS data from the NCI Open Database Compounds[1]. The original dataset contains about 249K graphs. By preprocessing and removing the disconnected graphs, we randomly select 100K graphs for our experiments. On average, each graph in the dataset has 21 nodes and 23 edges. The number of distinct labels for nodes and edges is 88.

To test the scalability of CGSearch on graph size, we design a synthetic graph generator (see details in GraphGen[2]). We generate four synthetic datasets by varying the average number of edges in a graph from 40 to 100. Each synthetic dataset has 100K graphs. The number of distinct labels is 30 and the average graph density is 0.15.

We use *FG-index* [6] to obtain the projected database of a graph. In all experiments, we set the minimum support threshold and the frequency tolerance factor in FG-index to be 0.03 and 0.05, respectively. We use *gSpan* [25] to mine the FGs for generating the set of candidates. All experiments were run on a linux machine with an AMD Opteron 248 CPU and 1 GB RAM.

The efficiency of CGSearch is based on the effective candidate generation from the projected database and the three heuristic rules. Since there is no existing work on mining correlations from graph databases, we mainly assess the effects of the candidate generation method and the heuristic rules on the performance of our algorithm.

To justify the effect of the candidate generation from the projected database on speeding up the mining process and on reducing the search space, we implement the approach whose candidates are mined from the whole database with a support range. Furthermore, to show the effect of the heuristic rules on refining the candidate set, we make three variants of our algorithm: *CGSearch_P*, *CGSearch_F* and *CGSearch_N*. Among them, CGSearch_P and CGSearch_F are implemented based on the different strategies of apply-

[1]http://cactus.nci.nih.gov/ncidb2/download.html
[2]http://www.cse.ust.hk/graphgen

ing the heuristic rules as discussed in Section 5.3. Table 2 summarizes the algorithms tested in our experiments.

Table 2: Algorithms Tested

Name	Description
Range	Generate the candidate set from $\mathcal{D}$ using $[lower(g), upper(g)]$ as a support range.
CGSearch_P	Partially apply the heuristic rules in CGSearch.
CGSearch_F	Fully apply the heuristic rules in CGSearch.
CGSearch_N	Do not apply the heuristic rules in CGSearch.

6.2 Performance on Real Dataset

Since the complexity of the CGS problem mainly depends on the support of the query, we randomly generate four sets of queries, 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.

6.2.1 Effect of Candidate Generation

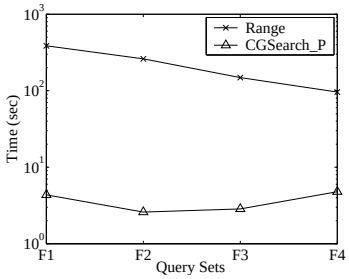
Figure 3 reports the performance of CGSearch_P and Range on the real dataset when varying the support of the queries. Figures 3(a-b) show the running time and memory consumption per query. On average, CGSearch_P is two orders of magnitude faster and consumes 10 times less memory than Range. For both CGSearch_P and Range, the time taken by the candidate generation dominates. We observe that, CGSearch_P is slightly slower for the query set F_1 and F_4. This is because the time for generating the candidates not only depends on the size of the projected database (i.e., $supp(q)$), but also depends on the minimum support threshold (i.e., $\frac{lower(q,g)}{supp(q)}$). Although the minimum support threshold of F_4 is the highest among all the query sets, its projected database is the largest, which increases the mining time slightly. While for F_1, its low minimum support threshold results in slightly longer processing time. Compared with Range, the running time of CGSearch_P is much more stable. For all support ranges, CGSearch_P takes 2 to 4 seconds for each query, while the running time of Range is largely influenced by the support of the query. With the decrease in the support of the query, the running time of Range increases rapidly from 100 seconds to 400 seconds.

We show the size of the candidate sets of CGSearch_P and Range in Figure 3(c). The size of the answer set is also shown as a reference. It is obvious that the size of the candidate set produced by CGSearch_P is much closer to that of the answer set. Compared with Range, the candidate set of CGSearch_P is over an order of magnitude smaller.

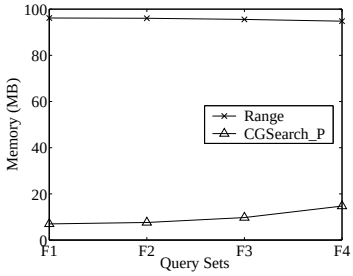
6.2.2 Effect of θ

Figure 4 reports the performance of CGSearch_P and Range when varying the minimum correlation threshold θ from 0.6 to 1. We test all query sets on the real dataset for both CGSearch_P and Range. For clarity of presentation, we only present F_4 for Range. But we remark that Range performs the best on F_4 among all the query sets, which means that the performance of Range on F_4 is the lower bound.

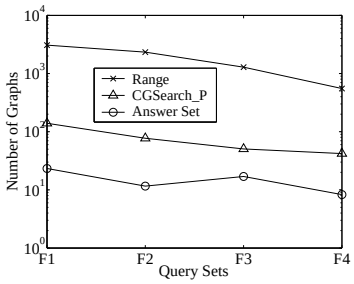
As shown in Figure 4, for all values of θ, CGSearch_P is over an order of magnitude faster and consumes 6.5 times less memory than Range on F_4. Given a query, with the decrease in θ, the minimum support threshold used to generate the candidates also decreases for both CGSearch_P and



(a) Running Time

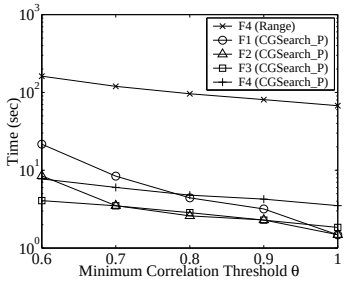


(b) Memory Consumption

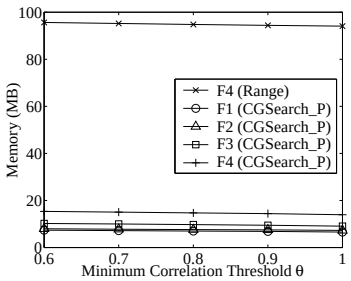


(c) Size of Candidate Set

Figure 3: Performance on Varying Query Support



(a) Running Time



(b) Memory Consumption

Figure 4: Performance on Varying θ

Range. Hence, the processing time of both CGSearch_P and Range increases with the decrease in θ. We find that, when varying θ, the running time of CGSearch_P on F_1 and F_2 is less stable than that on F_3 and F_4. We also observe similar phenomenon for Range on F_1 and F_2 (not reported in the figure). This is because the small minimum support threshold results in a large number of candidates. In this case, the time taken by candidate generation no longer dominates the total processing time. Instead, much more time is spent on querying the candidates by FG-index to verify the correlation condition. For example, for the query set F_1, when $\theta = 1$, only 3% of the total time is spent on querying the candidates; while when $\theta = 0.6$, more than 60% of the total time is spent on querying the candidates. This explains the trend of the running time for queries of low support.

6.2.3 Effect of Heuristic Rules

In order to show the effect of the heuristic rules clearly, we get rid of the time taken by the candidate generation and only present the time for querying candidates and checking the correlation condition.

Figure 5 shows the time on F_4 for the three variants of CGSearch at different values of θ. When $\theta = 0.6$, the number of candidates is large. Therefore, CGSearch_F performs the best, since the cost for querying the candidates is much larger than the cost for fully applying the heuristic rules. In this case, CGSearch_P is slower than CGSearch_F since partially applying the heuristic rules is not able to further reduce the number of candidates as effectively as does CGSearch_F. However, with the increase in θ, and hence the decrease in the size of candidate set, CGSearch_P outperforms CGSearch_F. This is because, given the smaller number of candidates, the full application of the heuristic rules which involves subgraph isomorphism testings is more costly than querying the candidates by FG-index. This suggests a good strategy for applying the heuristic rules: when the number of candidates is large, we can use CGSearch_F to reduce the search space as much as possible; when the number of candidates is relatively small, we can simply use CGSearch_P.

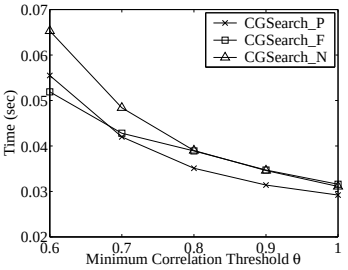
In most of the cases, CGSearch_N is the worst, since all the candidates need to go through the verification of correlation condition. However, if the number of candidates is small, it is possible that CGSearch_F is even slower than CGSearch_N due to too many subgraph isomorphism testings performed when fully applying the heuristic rules. Therefore, it can be seen from Figure 5 that the time of CGSearch_F is almost the same as that of CGSearch_N for high values of θ. However, in general, CGSearch_P outperforms CGSearch_N, since the partial application of the heuristic rules requires no subgraph isomorphism testing due to the prefix tree, as discussed in Section 5.3.

6.3 Performance on Synthetic Dataset

Since the graphs in the real dataset are of small size (averagely 23 edges per graph), we use the synthetic datasets to test the scalability of CGSearch and Range on different graph sizes.

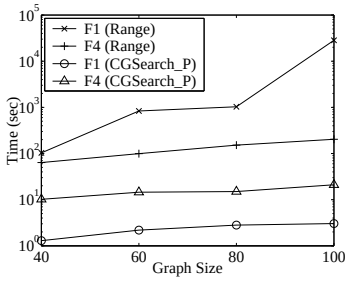
Similar to the experiments on the real dataset, we generate four sets of queries, F_1 to F_4, with the same setting of support ranges as in Section 6.2.

For clarity of presentation, we only show the results of F_1 and F_4, which are of the largest and the smallest support ranges, respectively. Figure 6 reports the performance of CGSearch_P and Range. For F_1, CGSearch_P is up to four orders of magnitude faster and consumes 41 times less memory than Range. While for F_4, CGSearch_P is still over an order of magnitude faster and consumes 6 times less memory than Range. The smaller improvement on the performance of CGSearch_P over Range for F_4 is because the average number of candidates of Range for F_4 is over three orders of magnitude smaller than that of Range for F_1 ($111,955$ for F_1 and 795 for F_4). Figure 6 also shows that, CGSearch_P is much more stable on resource usage than Range when varying graph sizes.

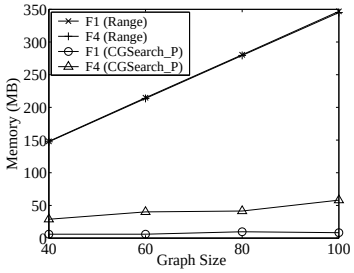


Running Time on F_4

Figure 5: Effect of Heuristic Rules



(a) Running Time



(b) Memory Consumption

Figure 6: Performance on Varying Graph Size

7. RELATED WORK

There have been a number of studies on mining correlations from various types of databases. Pearson's correlation coefficient, as well as its computation form for binary variables, ϕ correlation coefficient, are prevalently used as a correlation measure. Sakurai et al. [20] use Pearson's correlation coefficient to define the lag correlation between two time sequences. Xiong et al. [23] apply ϕ correlation coefficient to define the strongly correlated pairs in transaction databases. An upper bound of ϕ, as well as monotone

properties of the upper bound, are identified to facilitate the efficient mining process. Recently, Zhang and Feigenbaum [29] also adopt ϕ correlation coefficient to measure the correlated pairs in transaction databases. An efficient algorithm that uses min-hash functions as the pruning method is developed. To the best of our knowledge, our work is the first application of ϕ correlation coefficient in the context of graph databases.

In literature, many other correlation measures are proposed for different applications. For market-basket data, correlation measures include χ^2 [5], interest [5], all-confidence [13, 15], bond [15], h-confidence [24], and so on. For multimedia data, Pan et al. [16] use random walk with restart to define the correlation between the nodes in the graph that is constructed from a multimedia database. For quantitative databases, Ke et al. [11] utilize mutual information and all-confidence to define the correlated patterns.

For similarity searching techniques developed for general graph models, Holder et al. [8] use the minimum description length principle for inexact graph matching. Raymond et al. [18] propose an efficient algorithm, called *MCES*, to perform similarity search measured by maximum common subgraphs. Yan et al. [27] develop a structural filtering algorithm, called *Grafil*, to filter graphs without performing similarity computations. Recently, Williams et al. [22] propose an indexing technique that adopts graph decomposition methods to facilitate similarity search on graph databases. However, all of them focus on structural similarity search as indicated by their graph similarity measures, while our work captures statistical similarity defined by Pearson's correlation coefficient.

8. CONCLUSIONS

We formulate the problem of correlated graph search, which takes into account the occurrence distributions of graphs using Pearson's correlation coefficient. By deriving the theoretic bounds for the support of a candidate graph, we propose to efficiently generate the candidate set by mining FGs from the projected database of the query graph. We develop three effective heuristic rules to further reduce the size of the candidate set. We propose an efficient algorithm, CGSearch, to solve the problem of CGS. The soundness and completeness of the query results returned by CGSearch are also formally proved. The experimental results justify the efficiency and effectiveness of the candidate generation and the heuristic rules. Compared with the approach that generates the candidates directly from the database by a support range, our solution is orders of magnitude faster and consumes much less memory. More importantly, CGSearch achieves very stable performance when varying the support of the queries, the minimum correlation threshold, as well as the graph size.

Acknowledgement. This work is partially supported by RGC CERG under grant number HKUST6185/03E. We thank Dr. Xifeng Yan and Prof. Jiawei Han for providing us the executable of gSpan. We also thank the anonymous reviewers for their valuable comments.

9. REFERENCES

[1] DBLP Dataset. *http://dblp.uni-trier.de/xml/*.
[2] National library of medicine. *http://chem.sis.nlm.nih.gov/chemidplus*.
[3] The International Network for Social Network Analysis. *http://www.insna.org/*.
[4] 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.
[5] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. In *SIGMOD*, pages 265–276, 1997.
[6] J. Cheng, Y. Ke, W. Ng and A. Lu. FG-Index: Towards verification-free query processing on graph databses. In *SIGMOD*, 2007.
[7] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.
[8] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In *KDD*, pages 169–180, 1994.
[9] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 2000.
[10] M. Kanehisa and S. Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28:27–30, 2000.
[11] Y. Ke, J. Cheng, and W. Ng. Mining quantitative correlated patterns using an information-theoretic approach. In *KDD*, pages 227–236, 2006.
[12] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.
[13] S. Ma and J. L. Hellerstein. Mining mutually dependent patterns. In *ICDM*, pages 409–416, 2001.
[14] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *KDD*, pages 647–652, 2004.
[15] E. R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE TKDE*, 15(1):57–69, 2003.
[16] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.
[17] S. Raghavan and H. Garcia-Molina. Representing Web graphs. In *ICDE*, pages 405–416, 2003.
[18] J. W. Raymond, E. J. Gardiner, and P. Willett. RASCAL: calculation of graph similarity using maximum common edge subgraphs. *Comput. J.*, 45(6):631–644, 2002.
[19] H. Reynolds. *The analysis of cross-classifications*. The Free Press, New York, 1977.
[20] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. AutoLag: Automatic discovery of lag correlations in stream data. In *ICDE*, pages 159–160, 2005.
[21] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *KDD*, pages 32–41, 2002.
[22] D. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In *ICDE*, 2007.
[23] 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.
[24] H. Xiong, P.-N. Tan, and V. Kumar. Hyperclique pattern discovery. *DMKD*, 13(2):219–242, 2006.
[25] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, page 721, 2002.
[26] X. Yan, P. S. Yu, and J. Han. Graph indexing based on discriminative frequent structure analysis. *ACM TODS*, 30(4):960–993, 2005.
[27] X. Yan, F. Zhu, P. S. Yu, and J. Han. Feature-based similarity search in graph structures. *ACM TODS*, 31(4):1418–1453, 2006.
[28] G. U. Yule. On the methods of measuring association between two attributes. *Journal of the Royal Statistical Society*, 75(6):579–652, 1912.
[29] J. Zhang and J. Feigenbaum. Finding highly correlated pairs efficiently with powerful pruning. In *CIKM*, pages 152–161, 2006.