Efficient Correlation Search from Graph Databases

Yiping Ke, James Cheng, and Wilfred Ng

***Abstract*— Correlation mining has gained great success in many application domains for its ability to capture underlying dependencies between objects. However, research on correlation mining from graph databases is still lacking despite that graph data, especially in scientific domains, proliferate in recent years.**

We propose a new problem of correlation mining from graph databases, called Correlated Graph Search (CGS). CGS adopts Pearson's correlation coefficient as the correlation measure to take into account the occurrence distributions of graphs. However, the CGS 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 that set bounds on the occurrence probability of a candidate in the database. With this result, we devise an efficient algorithm that mines the candidate set from a much smaller projected database and thus we are able to obtain a significantly smaller set of candidates. Three heuristic rules are further developed to refine the candidate set. We also make use of the bounds to directly answer high-support queries without mining the candidates. Our experimental results demonstrate the efficiency of our algorithm. Finally, we show that our algorithm provides a general solution when most of the commonly used correlation measures are used to generalize the CGS problem.

***Index Terms*— Correlation, Graph Databases, Pearson's Correlation Coefficient.**

I. Introduction

Correlation mining is recognized as one of the most important data mining tasks for its capability to identify underlying dependencies between objects. It has a wide range of application domains and has been studied extensively in market-basket databases [1], [2], [3], [4], [5], [6], quantitative databases [7], multimedia databases [8], data streams [9], and many others. However, little attention has been paid to mining correlations from graph databases, in spite of the popularity of graph data models pertaining to various domains, such as biology [10], [11], chemistry [12], social science [13], the Web [14] and XML [15].

In this paper, we study a new problem of mining correlations from graph databases [16]. We propose to use *Pearson's correlation coefficient* [17] 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 with respect to* q *is no less than* θ.

Our problem of CGS has a close connection to graph similarity search. There are two types of similarity in graph databases: *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 [18], [19], [20], [21], [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 the database 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 to 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 and the discovered correlated graphs are very useful in many real applications such as drug design, anomalous detection, etc.

We use Pearson's correlation coefficient to define CGS since it is shown to be one of the most desirable correlation measures in [17] 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 [17], [5], [6]. 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 [23]). 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 any subgraph of a graph in $\mathcal{D}$ can be a candidate graph. Thus, there are great challenges in tackling the problem of CGS.

How can we reduce the large search space of CGS to avoid expensive graph operations as much as possible? We investigate the properties 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 from the set of all subgraphs of all graphs in $\mathcal{D}$ to be the set of *Frequent subGraphs* (*FGs*) [24] 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 small or when $\mathcal{D}$ is large. Moreover, we still have a large number of candidates and the solution is not scalable. Thus, we need to reduce further the number of candidates and address the scalability problem. The underlying idea of our solution, named *CGSearch*, 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 $\frac{lowerbound(supp(g))}{supp(q)}$ as the minimum support threshold is *complete* with respect to the answer

The authors are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. E-mail: {keyiping, csjames, wilfred}@cse.ust.hk.

Corresponding Author: Yiping Ke. Phone: 852-2358-8836. Fax: 852-2358-1477. Email: keyiping@cse.ust.hk.

set. Since $\mathcal{D}_q$ is much smaller than $\mathcal{D}$ while $\frac{lowerbound(supp(g))}{supp(q)}$ is greater than $lowerbound(supp(g))$, our findings not only save the computational cost for generating the candidate set, but also significantly reduce the number of candidates. Furthermore, we develop three heuristic rules to be applied on the candidate set generated from the projected database 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.

Since candidate generation involves a mining operation, which can still be expensive, we further improve the CGSearch algorithm to avoid performing this mining operation. More specifically, we maintain a set of FGs at a minimum support threshold σ. Given a query whose corresponding $lowerbound(supp(g))$ is no less than σ, we propose to generate its candidate set by querying from the set of FGs. We name this process *FGQuery*. To reduce the number of candidate verifications, we further develop another set of three heuristic rules to be applied on the candidate set produced by FGQuery. By integrating CGSearch and FGQuery, we present a more efficient solution to the CGS problem, named *CGSearch**.

Our extensive experiments on both real and synthetic datasets show that our algorithm CGSearch processes a wide range of queries with short response time and small memory consumption. Compared with the approach that generates candidate sets by mining the entire database with a support range, CGSearch is orders of magnitude faster and consumes up to 40 times less memory. The effectiveness of the candidate generation from the projected database and that of the three heuristic rules are also demonstrated. The results also show that the algorithm CGSearch* further improves the response time of CGSearch by an order of magnitude, with comparable memory consumption, for queries that are of high support.

Finally, considering that there are also many other well-established correlation measures [17], we generalize the CGS problem to adopt other correlation measures. In order to find a general solution, we model the generalized CGS problem as a system of inequalities. By solving this inequality system, we prove that our solution for Pearson's correlation coefficient also serves as an effective and efficient solution for the majority of the correlation measures.

Contributions. We make the following specific contributions.

- 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 present an efficient algorithm, CGSearch, to solve the problem of CGS. We propose to generate the candidate set by mining FGs from the projected database of the query graph. We develop three heuristic rules to further reduce the size of the candidate set. We also prove the soundness and completeness of the query results returned by CGSearch.
- We present an improved algorithm, CGSearch*, which is able to avoid performing the mining process of candidate generation for queries of high support. Three more heuristic rules are presented to be applied on this candidate set to further reduce the search space.
- We conduct a comprehensive set of experiments to verify the efficiency of the algorithm and the effectiveness of the candidate set generation and the heuristic rules.
- We generalize the CGS problem to adopt other correlation measures and show that our algorithm provides a general solution for most of the commonly used measures.

Organization. We give preliminaries in Section II. We define the CGS problem in Section III. We propose effective candidate generation from a projected database in Section IV. We present the CGSearch algorithm in Section V. We present the improved algorithm, CGSearch*, together with FGQuery, in Section VI. We analyze the performance in Section VII. Then, we generalize the CGS problem and discuss its solution in Section VIII. Finally, we discuss related work in Section IX and conclude our paper in Section X.

II. Preliminaries

In this paper, we restrict our discussion to *undirected, labelled connected graphs* (or simply *graphs* hereinafter), but 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 $g' = (V', E', L', l')$, g is called 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 $\forall(u, v) \in E$, $(f(u), f(v)) \in E'$, $l(u) = l'(f(u))$, $l(v) = l'(f(v))$, and $l(u, v) = l'(f(u), f(v))$. The injective function f is called a *subgraph isomorphism* from g to g'. Testing subgraph isomorphism is known to be *NP*-complete [23].

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*) [25], [24], [26] 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 properties: $supp(g_1, g_2) \leq supp(g_1)$ 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 simplicity of illustration, all the nodes have 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 $supp(g_8) = \frac{freq(g_8)}{|\mathcal{D}|} = \frac{5}{10} = 0.5$. 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 $freq(g_8, g_9) = |\mathcal{D}_{g_8} \cap \mathcal{D}_{g_9}| = |\{g_6, g_7\}| = 2$. Therefore, the joint support of g_8 and g_9 is computed as $supp(g_8, g_9) = \frac{freq(g_8,g_9)}{|\mathcal{D}|} = 0.2$. ■

III. The CGS Problem

We first define *Pearson's correlation coefficient* [27] for two graphs. Pearson's correlation coefficient for boolean variables is also known as the "ϕ *correlation coefficient*" [28].

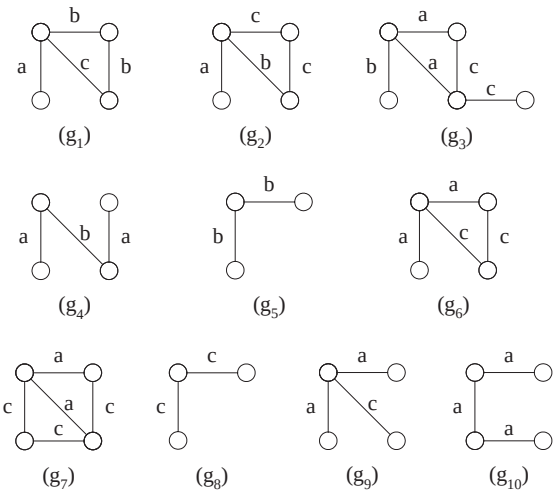


Fig. 1. A Graph Database, $\mathcal{D} = \{g_1, \ldots, g_{10}\}$

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; if $\phi(g_1, g_2)$ is zero, then g_1 and g_2 are independent; 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 associate $\mathcal{D}_g$ with g to form a pair $(g, \mathcal{D}_g)$ 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 I presents the notation used throughout the paper.

TABLE I
NOTATION USED THROUGHOUT THE PAPER

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 $\mathcal{D}_q$
$lower_{supp(g)}$, $upper_{supp(g)}$	the lower/upper bound of $supp(g)$
$lower_{supp(q,g)}$, $upper_{supp(q,g)}$	the lower/upper bound of $supp(q, g)$

IV. 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 generate a small set of candidates for a given query.

A. 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 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 obtain the lower bound as follows:

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

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

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

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

■

For simplicity, we use $lower_{supp(g)}$ and $upper_{supp(g)}$ 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, that is, a candidate graph should have support within the range of $[lower_{supp(g)}, upper_{supp(g)}]$.

With the result in Lemma 1, we are able to obtain the candidate set by mining the set of FGs [24], [26], [29] from $\mathcal{D}$ using $lower_{supp(g)}$ as the minimum support threshold and $upper_{supp(g)}$ as the maximum support threshold. However, according to the anti-monotone property of the support measure, the graphs with higher support values are always generated before those with lower support values, no matter whether a breadth-first or a depth-first strategy is adopted. As a result, the maximum threshold $upper_{supp(g)}$ is not able to speed up the mining process. Therefore, generating the candidate set by mining the FGs from $\mathcal{D}$ with a support range is still not efficient enough, especially when $lower_{supp(g)}$ is small or $\mathcal{D}$ is large. This motivates us to devise a more efficient and effective approach to generating the candidates.

B. Candidate Generation From a Projected Database

From Definition 1, it follows that if $\phi(q, g) > 0$, 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 from $\mathcal{D}$?

The challenge is that 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 to mine all FGs since it is too expensive. In this subsection, we derive a minimum support threshold that enables us to compute the candidates from $\mathcal{D}_q$ efficiently. Our solution is inspired by an 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, it follows that all graphs in $\mathcal{D}_q$ contain q. Therefore, each 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 the projected database $\mathcal{D}_q$ 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 joint support, we obtain the upper bound of $supp(q,g)$ as follows:

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

Then, we derive a lower bound for $supp(q,g)$. Given $\phi(q,g) \geq \theta$, the following inequality can be obtained from Definition 1.

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

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 want to get by using $supp(q,g)$. However, since we have obtained the range of $supp(g)$, i.e., $[lower_{supp(g)}, upper_{supp(g)}]$ as stated in Lemma 1, we now show that this range can be used in Inequality (2) to obtain the lower bound of $supp(q,g)$, which is independent of g.

By investigating the property of the function f, we find that f is monotonically increasing with $supp(g)$ in the range of $[lower_{supp(g)}, upper_{supp(g)}]$. Therefore, by substituting $supp(g)$ with $lower_{supp(g)}$ in Inequality (2), we are then able to 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 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 the range of $[lower_{supp(g)}, upper_{supp(g)}]$, $f'(supp(g)) \geq 0$ 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 $supp(g) \leq upper_{supp(g)} \leq 0.5$, then $(1-2\cdot supp(g)) \geq 0$ and hence $f'(supp(g)) \geq 0$.

Now, we consider the case when $upper_{supp(g)} \geq supp(g) > 0.5$. Since the left hand side of Inequality (3) is less than 0, we square 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 $0.5 < supp(g) \leq upper_{supp(g)}$, we replace $supp(g)$ with $upper_{supp(g)}$ in this quadratic function:

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

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

Thus, f is monotonically increasing within the range of $[lower_{supp(g)}, upper_{supp(g)}]$. The lower bound of $supp(q,g)$ follows by substituting $supp(g)$ with $lower_{supp(g)}$ in Inequality (2):

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

∎

From now on, we use $lower_{supp(q,g)}$ and $upper_{supp(q,g)}$ 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 $\frac{lower_{supp(q,g)}}{supp(q)}$ 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 $\frac{lower_{supp(q,g)}}{supp(q)}$. 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 $lower_{supp(q,g)} \leq supp(q,g) \leq upper_{supp(q,g)}$ by Lemma 3. Dividing these expressions by $supp(q)$, we have $\frac{lower_{supp(q,g)}}{supp(q)} \leq \frac{supp(q,g)}{supp(q)} \leq 1$. By Lemma 2, we have $\frac{lower_{supp(q,g)}}{supp(q)} \leq supp(g;\mathcal{D}_q) \leq 1$. The result $g \in \mathcal{C}$ follows, since $\mathcal{C}$ is the set of FGs mined from $\mathcal{D}_q$ using $\frac{lower_{supp(q,g)}}{supp(q)}$ 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 $\frac{lower_{supp(q,g)}}{supp(q)}$ (compared with $lower_{supp(g)}$ which is equal to $lower_{supp(q,g)}$, as shown in Lemmas 1 and 3).

V. CGSearch Algorithm

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 $\frac{lower_{supp(q,g)}}{supp(q)}$ as the minimum support threshold.
3) Refine $\mathcal{C}$ by using 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 (e.g., [30], [31]) 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 [24], [26], [29]. 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 to $\mathcal{C}$ 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 $(supp(g;\mathcal{D}_q) \cdot supp(q))$ according to Lemma 2.

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

A. Heuristic Rules

To check whether each graph g in $\mathcal{C}$ is correlated with q, a query operation is needed to obtain $\mathcal{D}_g$ for each candidate g (Step 4(a)). This 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 $supp(g,q;\mathcal{D}_q) \geq \frac{lower_{supp(q,g)}}{supp(q)}$. By Lemma 2, we further have $supp(q,g) \geq lower_{supp(q,g)}$.

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

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

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 for free 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 x with 0 and 1 in 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 $\phi(q,g_1) < \theta$. Therefore, $\phi(q,g_2) \leq \phi(q,g_1) < \theta$. Thus, we have $g_2 \notin base(\mathcal{A}_q)$. ■

By Lemma 2, if $supp(g_1,q)=supp(g_2,q)$, then $supp(g_1;\mathcal{D}_q) = supp(g_2;\mathcal{D}_q)$. 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 in $\mathcal{C}$ 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 $[lower_{supp(g)}, upper_{supp(g)}]$. As shown in the proof of Lemma 3, the function f is monotonically increasing within the range $[lower_{supp(g)}, upper_{supp(g)}]$. 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 derive the following expressions:

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

It thus follows the result that $g_2 \notin base(\mathcal{A}_q)$. ■

Note that in Heuristic 3, $supp(g_2, q) < f(supp(g_1))$ 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 in $\mathcal{C}$ that have support less than $\frac{f(supp(g))}{supp(q)}$ in $\mathcal{D}_q$.

B. 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 apply 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 $\mathcal{C}$, we first obtain their projected databases (Line 7). 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 $\mathcal{D}_q$;
2. Mine FGs from $\mathcal{D}_q$ using $\frac{lower_{supp(q,g)}}{supp(q)}$ as the minimum support threshold and add the FGs to $\mathcal{C}$;
3. **for each** graph $g \in \mathcal{C}$ in *size-descending order* **do**
4. **if** $(g \supseteq q)$
5. Add $(g, \mathcal{D}_g)$ to $\mathcal{A}_q$;
6. **else**
7. Obtain $\mathcal{D}_g$;
8. **if** $(\phi(q, g) \geq \theta)$
9. Add $(g, \mathcal{D}_g)$ to $\mathcal{A}_q$;
10. **else**
11. $H_2 \leftarrow \{g' \in \mathcal{C} : g' \subseteq g, supp(g'; \mathcal{D}_q) = supp(g; \mathcal{D}_q)\}$;
12. $\mathcal{C} \leftarrow \mathcal{C} - H_2$;
13. $H_3 \leftarrow \{g' \in \mathcal{C} : g' \subseteq g, supp(g'; \mathcal{D}_q) < \frac{f(supp(g))}{supp(q)}\}$;
14. $\mathcal{C} \leftarrow \mathcal{C} - H_3$;
15. **return** $\mathcal{A}_q$;

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

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.

We now prove 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 these 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 again 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 $lower_{supp(q,g)} = 0.19$. 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 nine candidates, which are shown in Figure 2(b). The number following the colon ":" 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 supergraph of q, $(c_1, \mathcal{D}_{c_1})$ is directly included in $\mathcal{A}_q$ 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 directly includes $(c_3, \mathcal{D}_{c_3})$ into $\mathcal{A}_q$ since c_3 is a supergraph of q. 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 to $\mathcal{A}_q$. 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 $(c_8, \mathcal{D}_{c_8})$ to $\mathcal{A}_q$.

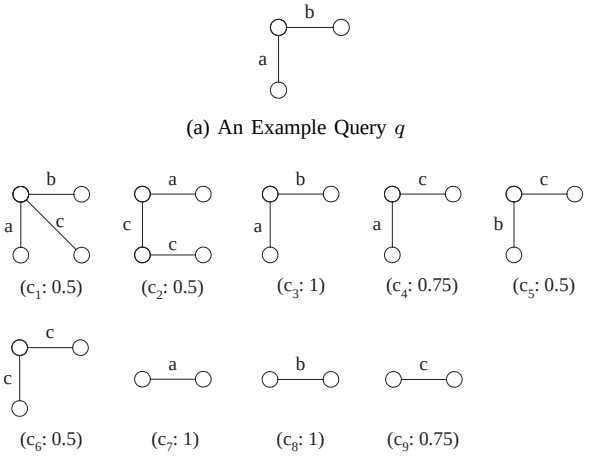
Therefore, $\mathcal{A}_q = \{(c_1, \mathcal{D}_{c_1}), (c_3, \mathcal{D}_{c_3}), (c_5, \mathcal{D}_{c_5}), (c_8, \mathcal{D}_{c_8})\}$. Among the nine candidates, five of them do not need to perform correlation verification by applying Heuristics 1 to 3.

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 $lower_{supp(g)} = 0.19$ and $upper_{supp(g)} = 0.64$ as support thresholds, there are still 16 graphs in the candidate set. This clearly illustrates that the candidate generation from the projected database significantly reduces the search space indeed. ■

C. Discussion

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 expensive and should be avoided as much as possible. We find that the number of subgraph isomorphism tests can be effectively reduced by using a depth-first FG mining algorithm (such as *gSpan* [26]) for the candidate generation. In a depth-first mining process, the FGs generated can be organized in a prefix tree, in which a child is a supergraph of its parent. Thus, by following the *root-to-leaf paths* 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



(a) An Example Query q

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

Fig. 2. An Example Query and Its Candidate Set

are not able to identify all the graphs in H_2 and H_3 of Algorithm 1 and all the supergraphs of q. However, we observe 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 tests have to be performed but fewer candidates are needed for verification of the 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 test is needed but more candidates are required for verification. We further demonstrate this trade-off in our experiments.

VI. CGSEARCH* ALGORITHM

An essential step in the CGSearch algorithm introduced in Section V is to mine the set of candidates from the projected database (Line 2 of Algorithm 1). Although mining from the projected database is much cheaper than mining from the whole database, the mining operation is still expensive. In this section, we further improve the CGSearch algorithm to avoid performing the mining process for queries that are of high support.

More specifically, we organize the set of FGs (as well as their corresponding projected databases) with a minimum support threshold σ according to the support values of FGs. In this way, the set of FGs within the support range of $[lower_{supp(g)}, upper_{supp(g)}]$, i.e., the set of candidate graphs, can be obtained directly if $lower_{supp(g)} \geq \sigma$. We call this process of obtaining the candidate set *FGQuery*. Since FGQuery is much cheaper than the mining operation, we can significantly reduce the response time for processing query graphs whose corresponding $lower_{supp(g)}$ is no less than σ.

Let $\mathcal{C}_Q$ be the candidate set returned by FGQuery and $\mathcal{C}$ be the candidate set generated from the projected database by CGSearch. The cost of correlation verification for a candidate graph $g \in \mathcal{C}_Q$ is significantly lower that for $g \in \mathcal{C}$. To check the correlation condition, i.e., $\phi(q, g) \geq \theta$, we need the values of $supp(g)$ and $supp(q, g)$. For $g \in \mathcal{C}$, the value of $supp(q, g)$ is obtained when mining the projected database of q. To check whether g is an answer, we should obtain the projected database of g, $\mathcal{D}_g$, (Line 7 of Algorithm 1) to get $supp(g)$, which is an expensive operation. On the other hand, for $g \in \mathcal{C}_Q$, the value of $supp(g)$ and the projected database $\mathcal{D}_g$ are indexed in FGQuery. Thus, to verify whether g is an answer, we only need to intersect $\mathcal{D}_q$ and $\mathcal{D}_g$ to compute $supp(q, g)$, which is much cheaper than the operation to obtain $\mathcal{D}_g$ in the case when $g \in \mathcal{C}$.

Although FGQuery can save candidate generation time and candidate verification in $\mathcal{C}_Q$ is also cheaper than that in $\mathcal{C}$, $\mathcal{C}_Q$ is usually much larger than $\mathcal{C}$. To reduce the number of candidate verifications required for the candidates in $\mathcal{C}_Q$, we develop another set of heuristic rules to be applied to $\mathcal{C}_Q$.

In the remainder of this section, we first discuss the heuristic rules. Then, we describe how the heuristic rules are applied. Finally, we present the improved algorithm, named *CGSearch**, which uses FGQuery to avoid mining for queries of high support.

A. Heuristic Rules

Similar to Heuristic 1, the following heuristic is to identify the candidate graphs in $\mathcal{C}_Q$ that are guaranteed to be answer graphs.

HEURISTIC *4:* Given a graph g, if $g \in \mathcal{C}_Q$ and $g \supseteq q$, then $g \in base(\mathcal{A}_q)$.

Proof: Since $g \supseteq q$, we have $supp(q, g) = supp(g)$. Furthermore, since $g \in \mathcal{C}_Q$, we have $lower_{supp(g)} \leq supp(g) \leq upper_{supp(g)}$.

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

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

which is monotonically increasing with $supp(g)$. We further replace $supp(g)$ with its lower bound of $lower_{supp(g)}$ in the above $\phi(q, g)$ and obtain the following expression:

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

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

By Heuristic 4, if we find that a graph in $\mathcal{C}_Q$ is a supergraph of q, we can directly add it to the answer set without any verification.

The following two heuristic rules are not only useful for identifying answer graphs without performing correlation checking, but also effective for eliminating false-positives.

HEURISTIC *5:* Given two graphs g_1 and g_2, where $g_1 \supseteq g_2$ and $supp(g_1) = supp(g_2)$, the following two statements are true:

(a) $g_1 \in base(\mathcal{A}_q)$ if and only if $g_2 \in base(\mathcal{A}_q)$.
(b) $g_1 \notin base(\mathcal{A}_q)$ if and only if $g_2 \notin base(\mathcal{A}_q)$.

Proof: Since $g_1 \supseteq g_2$ and $supp(g_1) = supp(g_2)$, we know that g_1 appears in every graph in which g_2 appears. Thus, we have $supp(q, g_1) = supp(q, g_2)$. By Definition 1, we have $\phi(q, g_1) = \phi(q, g_2)$ and hence the results of 5(a) and 5(b) follow. ■

By Heuristic 5, when a candidate graph g in $\mathcal{C}_Q$ is included in or excluded from the answer set, the same result also applies to g's supergraphs or subgraphs that have the same support.

HEURISTIC *6:* Given two graphs g_1 and g_2, where $g_1 \supseteq g_2$, the following two statements are true:

(a) If $supp(g_2) \leq h(supp(q, g_1))$, then $g_2 \in base(\mathcal{A}_q)$.
(b) If $supp(g_1) > h(supp(q, g_2))$, then $g_1 \notin base(\mathcal{A}_q)$.

The function h is defined as follows:

$$h(supp(q, g)) = \frac{a(2 \cdot supp(q, g) - a) + c - b\sqrt{c - (2 \cdot supp(q, g) - a)^2}}{2c},$$

where $a = supp(q)$; $b = \theta\sqrt{a(1 - a)}$ and $c = a^2 + b^2$.

Proof: We first prove that the function h is monotonically increasing with $supp(q, g)$. The derivative of h is given as follows:

$$h'(supp(q,g)) = \frac{a\sqrt{c-(2\cdot supp(q,g)-a)^2}+b(2\cdot supp(q,g)-a)}{c\sqrt{c-(2\cdot supp(q,g)-a)^2}}. \quad (5)$$

If $supp(q,g)\geq\frac{supp(q)}{2}$, that is, $(2\cdot supp(q,g)-a)\geq 0$, Equation (5) is no less than 0. This proves that the function h monotonically increases with $supp(q,g)$ within the range of $[\frac{supp(q)}{2},1]$.

We now consider the case when $supp(q,g)<\frac{supp(q)}{2}$. We have $0\leq(a-2\cdot supp(q,g))\leq a$. Thus, it follows from Equation (5) that:

$$h'(supp(q,g)) = \frac{a\sqrt{c-(a-2\cdot supp(q,g))^2}-b(a-2\cdot supp(q,g))}{c\sqrt{c-(2\cdot supp(q,g)-a)^2}} \quad (6)$$

$$= \frac{a\sqrt{a^2+b^2-(a-2\cdot supp(q,g))^2}-b(a-2\cdot supp(q,g))}{c\sqrt{c-(2\cdot supp(q,g)-a)^2}} \quad (7)$$

$$\geq \frac{ab-b(a-2\cdot supp(q,g))}{c\sqrt{c-(2\cdot supp(q,g)-a)^2}} \quad (8)$$

$$\geq 0. \quad (9)$$

Equation (7) is obtained by replacing c with (a^2+b^2) in the numerator of Equation (6). The last two inequalities follow, since $a\geq(a-2\cdot supp(q,g))$.

Therefore, we have proved that the function h is monotonically increasing with $supp(q,g)$.

Now, we prove Heuristic 6(a). Since $g_1\supseteq g_2$, we have $supp(q,g_1)\leq supp(q,g_2)$. Since the function h is monotonically increasing with $supp(q,g)$, we have $supp(g_2)\leq h(supp(q,g_1))\leq h(supp(q,g_2))$. By Lemma 4, $\phi(q,g_2)$ is monotonically decreasing with $supp(g_2)$ when other parameters are fixed. Therefore, by replacing $supp(g_2)$ with $h(supp(q,g_2))$ in $\phi(q,g_2)$, we derive the following expression:

$$\phi(q,g_2) \geq \frac{supp(q,g_2)-supp(q)h(supp(q,g_2))}{\sqrt{supp(q)(1-supp(q))h(supp(q,g_2))(1-h(supp(q,g_2)))}} = \theta.$$

The result $g_2\in base(\mathcal{A}_q)$ follows.

Heuristic 6(b) can be proved similarly as Heuristic 6(a). ■

Note that in Heuristic 6(a), $supp(g_2)\leq h(supp(q,g_1))$ also implies $g_1\in base(\mathcal{A}_q)$. This is because $g_1\supseteq g_2$ implies $supp(g_1)\leq supp(g_2)\leq h(supp(q,g_1))$. As similar proof to that for Heuristic 6(a) applies, $g_1\in base(\mathcal{A}_q)$ follows. Similarly, in Heuristic 6(b), $supp(g_1)>h(supp(q,g_2))$ also implies $g_2\notin base(\mathcal{A}_q)$.

By Heuristic 6, if we find that a candidate graph g is an answer, we can directly include its subgraph in $\mathcal{C}_Q$ whose support value is no greater than $h(supp(q,g))$. On the other hand, if we find that a candidate graph g is not an answer, we can prune its supergraph in $\mathcal{C}_Q$ whose support value is greater than $h(supp(q,g))$.

B. Application of Heuristic Rules in FGQuery

In this section, we show how Heuristics 4 to 6 can be effectively applied in FGQuery. Since a set of FGs is usually indexed by the indexing technique (such as FG-index [31]) for obtaining the projected database, we implement FGQuery on this set of FGs.

Let $\mathcal{F}$ be the set of FGs indexed by FG-index. To apply the heuristics, we construct a lattice on $\mathcal{F}$, called the *FG-lattice*. To build the lattice, we associate a *children* list and a *parents* list for each $g\in\mathcal{F}$, where the children list keeps all subgraphs of g that have one less edge than g and the parents list keeps all supergraphs of g that have one more edge than g. The FG-lattice can be constructed during the construction of FG-index, without incurring too much extra cost. The only change to the algorithm for the construction of FG-index is by deleting Line 9 of Algorithm 1 in [31] and computing the children list and the parents list of each FG.

Algorithm 2 *FGQuery*

Input: A query graph q and a set of FGs $\mathcal{F}$.
Output: The answer set $\mathcal{A}_q$.

```
1.  Obtain C_Q from F;
2.  Initialize two empty queues, Q_Y and Q_N;
3.  for each g ⊇ q, where g ∈ C_Q, do
4.    Add (g, D_g) to A_q and mark g;
5.    for each unmarked child, c, of g do
6.      Child_Y(c, g, Q_Y, Q_N, A_q);
7.  while (Q_N is not empty)
8.    Pop g out of Q_N;
9.    for each unmarked parent, p, of g do
10.     Parent_N(p, g, Q_Y, Q_N, A_q);
11.   for each unmarked child, c, of g do
12.     Child_N(c, g, Q_Y, Q_N, A_q);
13. while (Q_Y is not empty)
14.   Pop g out of Q_Y;
15.   for each unmarked parent, p, of g do
16.     Parent_Y(p, g, Q_Y, Q_N, A_q);
17.   for each unmarked child, c, of g do
18.     Child_Y(c, g, Q_Y, Q_N, A_q);
19. if (Q_N is not empty)    goto Line 7;
20. else if (Q_Y is not empty)    goto Line 13;
21. else    /* both Q_N and Q_Y are empty */
22.   Scan C_Q until an unmarked graph g is found;
23.   Mark g;
24.   if (Check(g))
25.     Add (g, D_g) to A_q, push g into Q_Y, and goto Line 13;
26.   else
27.     Push g into Q_N and goto Line 7;
28. return A_q;
```

Since many graphs share a large number of supergraphs and subgraphs, we need an effective strategy to apply Heuristics 4 to 6, so that the graphs will not be processed duplicately. We devise an efficient algorithm, *FGQuery*, as shown in Algorithm 2, to apply the three heuristics to compute $\mathcal{A}_q$.

FGQuery first obtains the candidate set $\mathcal{C}_Q$ from $\mathcal{F}$. Since whether or not a graph belongs to $\mathcal{C}_Q$ is determined by its support, $\mathcal{F}$ can be pre-sorted in ascending order of the support of the FGs. Thus, $\mathcal{C}_Q$ is simply the sub-array $\mathcal{F}[lower_{supp(g)}, upper_{supp(g)}]$, where $lower_{supp(g)}$ and $upper_{supp(g)}$ of a given q can be computed by using Lemma 1.

According to Heuristics 5 and 6, given the knowledge of whether or not a graph g is in the answer set, we can directly determine the inclusion/exclusion of many supergraphs and subgraphs of g into/from the answer set. Thus, in Algorithm 2, we use two queues, Q_Y and Q_N, for keeping graphs that have been determined to be in and not in the answer set, respectively. We mark a candidate whenever we push it into a queue to avoid it being processed repeatedly.

There are four cases when Heuristics 5 and 6 can be applied. We express these four cases in Procedures 1 to 4. If a candidate graph g is determined to be an answer, i.e., $g\in Q_Y$, we can process g's child, c, by calling *Child_Y*() as given in Procedure 1. Heuristics 5(a) and 6(a) are applied in Line 1 of Procedure 1 to include the qualified subgraph of g into the answer set. When

the graph c cannot be determined by the heuristics (Lines 3-6 of Procedure 1), we check the correlation condition of c using the boolean operation *Check*(), which is true when the correlation condition is true. The graph c is then included in either Q_Y (and $\mathcal{A}_q$) or Q_N. Procedure 2 applies Heuristic 5(a) to include the qualified supergraph of $g \in Q_Y$ into the answer set. Similarly, Procedure 3 applies Heuristic 5(b) to prune the subgraph of $g \in Q_N$ from the candidate set; while Procedure 4 applies Heuristics 5(b) and 6(b) to prune the supergraph of $g \in Q_N$.

Procedure 1 *Child_Y*(c, g, Q_Y, Q_N, $\mathcal{A}_q$)

1. **if** ($supp(c) = supp(g)$ or $supp(c) \leq h(supp(q, g))$)
2. Add $(c, \mathcal{D}_c)$ to $\mathcal{A}_q$ and push c into Q_Y;
3. **else if** (*Check*(c))
4. Add $(c, \mathcal{D}_c)$ to $\mathcal{A}_q$ and push c into Q_Y;
5. **else**
6. Push c into Q_N;
7. Mark c;

Procedure 2 *Parent_Y*(p, g, Q_Y, Q_N, $\mathcal{A}_q$)

1. **if** ($supp(p) = supp(g)$)
2. Add $(p, \mathcal{D}_p)$ to $\mathcal{A}_q$ and push p into Q_Y;
3. **else if** (*Check*(p))
4. Add $(p, \mathcal{D}_p)$ to $\mathcal{A}_q$ and push p into Q_Y;
5. **else**
6. Push p into Q_N;
7. Mark p;

Procedure 3 *Child_N*(c, g, Q_Y, Q_N, $\mathcal{A}_q$)

1. **if** ($supp(c) = supp(g)$)
2. Push c into Q_N;
3. **else if** (*Check*(c))
4. Add $(c, \mathcal{D}_c)$ to $\mathcal{A}_q$ and push c into Q_Y;
5. **else**
6. Push c into Q_N;
7. Mark c;

Procedure 4 *Parent_N*(p, g, Q_Y, Q_N, $\mathcal{A}_q$)

1. **if** ($supp(p) = supp(g)$ or $supp(p) > h(supp(q, g))$)
2. Push p into Q_N;
3. **else if** (*Check*(p))
4. Add $(p, \mathcal{D}_p)$ to $\mathcal{A}_q$ and push p into Q_Y;
5. **else**
6. Push p into Q_N;
7. Mark p;

Algorithm 2 consists of four main parts. First, Lines 3-6 process the supergraphs of q by Heuristic 4, using the parents list of the candidate graphs in the FG-lattice. The algorithm also uses the children list of the graphs to include the children of q's supergraphs in either Q_Y (and $\mathcal{A}_q$) or Q_N by calling *Child_Y*().

Next, Lines 7-12 process the subgraphs and supergraphs of the graphs in Q_N by calling *Child_N*() and *Parent_N*(), respectively. Then, Lines 13-18 process the subgraphs and supergraphs of the graphs in Q_Y by calling *Child_Y*() and *Parent_Y*(), respectively.

When both Q_Y and Q_N become empty (Lines 21-27), we linearly scan $\mathcal{C}_Q$. When an *unmarked* candidate g is found, we check whether g is an answer, push it into Q_Y or Q_N, and then continue to apply Heuristics 5 and 6 to process the candidates that are g's supergraphs and subgraphs.

Finally, the algorithm returns $\mathcal{A}_q$ when all candidates in $\mathcal{C}_Q$ are marked.

C. CGSearch* Algorithm

We now present the overall algorithm, *CGSearch**, which is a more efficient solution to the CGS problem by integrating the CGSearch algorithm and the FGQuery algorithm.

Algorithm 3 *CGSearch**

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

1. Obtain $\mathcal{D}_q$;
2. **if**($lower_{supp(g)} \geq \sigma$)
3. *FGQuery*;
4. **else**
5. *CGSearch*;
6. **return** $\mathcal{A}_q$;

As shown in Algorithm 3, the first step is to obtain the projected database of q using the indexing technique. Then, we compute the lower bound of a candidate graph, $lower_{supp(g)}$, as given in Lemma 1. If $lower_{supp(g)}$ is no less than the minimum support threshold σ used in the indexing technique, *FGQuery* is invoked to avoid the mining operation for candidate generation; otherwise, *CGSearch* is invoked to generate the candidates from the projected database. We remark that, the calling of *CGSearch* in Algorithm 3 skips processing Line 1 of Algorithm 1 since it has been executed by Line 1 of *CGSearch**.

VII. Performance Evaluation

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

A. 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. After 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. The real dataset is used in the experiments in Sections VII-B, VII-C, and VII-D.

Since the graphs in the real dataset are generally small and of low density, we use synthetic datasets to evaluate the performance of the algorithms on graphs with different sizes and densities in Sections VII-E and VII-F. We develop a synthetic graph generator (see details in GraphGen[2]) for our experiments. We first vary the average number of edges in a graph from 40 to 100, by fixing the average graph density to 0.15. Then, we fix the average number of edges in a graph to 60, and vary the average graph density from 0.05 (50 nodes) to 0.2 (25 nodes). Each synthetic dataset has 100K graphs and the number of distinct labels is 30.

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, for each of the datasets tested. Each F_i 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. We set the minimum correlation threshold θ to 0.8 for all experiments, except for Sections VII-B and VII-D, where

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

we test the effect of heuristic rules and the performance of algorithms when varying θ.

The efficiency of CGSearch is based on the effective candidate generation from the projected database and the application of Heuristics 1 to 3. 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. First, to show the efficiency gained by using the projected database for candidate generation, we compare with the approach, called *Range*, for which the candidates are mined from the original database with a support range. Second, to show the effect of Heuristics 1 to 3 on refining the candidate set, we implement 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 applying Heuristics 1 to 3 as discussed in Section V-C. We also test the CGSearch* algorithm to assess the efficiency improvement by using FGQuery. Table II summarizes the algorithms tested in this experiment.

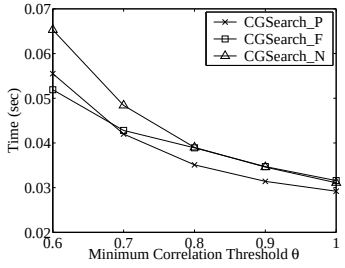
TABLE II
ALGORITHMS TESTED

Name	Description
Range	Generate the candidate set from $\mathcal{D}$ using $[lower_{supp(g)}, upper supp(g)]$ as a support range.
CGSearch_P	**P**artially apply Heuristics 1 to 3 in CGSearch.
CGSearch_F	**F**ully apply Heuristics 1 to 3 in CGSearch.
CGSearch_N	Do **N**ot apply Heuristics 1 to 3 in CGSearch.
*CGSearch**	A hybrid approach: invoke FGQuery for high-support queries and CGSearch_P for low-support queries.

We use *FG-index* [31] to obtain the projected database of a graph. In all experiments, we set the minimum support threshold σ and the frequency tolerance factor δ in FG-index to 0.03 and 0.05, respectively. The same value of σ is also used for our CGSearch* algorithm. We use *gSpan* [26] to mine the FGs for generating the set of candidates. All experiments are run on a linux machine with an AMD Opteron 248 CPU and 1 GB RAM.

B. Effect of Heuristic Rules

We first show the effect of applying Heuristics 1 to 3 presented in Section V-A. Figure 3 shows the running time on F_4 for the three variants of CGSearch at different values of θ. In order to focus on the effect of the heuristic rules, we do not include the time taken by the candidate generation and only present the time for querying the candidates and checking the correlation condition. The time for processing other query sets follows similar trends and is hence omitted for brevity.



Running Time on F_4

Fig. 3. Effect of Heuristic Rules

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 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 the 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 the 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 tests that need to be performed when fully applying the heuristic rules. Therefore, it can be seen from Figure 3 that the running time of CGSearch_F is almost the same as that of CGSearch_N when θ is high. However, in general, CGSearch_P outperforms CGSearch_N, since the partial application of the heuristic rules requires no subgraph isomorphism test due to the prefix tree, as discussed in Section V-C.

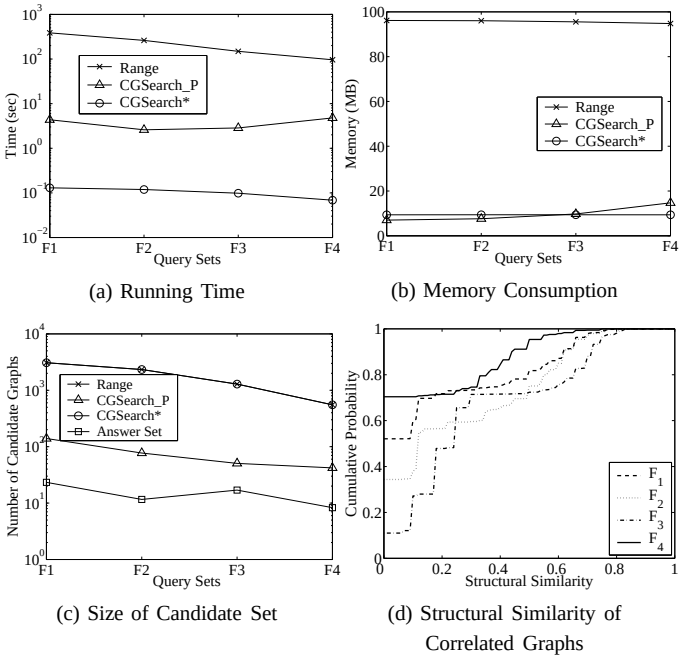
In the rest of the experiments, we use CGSearch_P when we compare the algorithm CGSearch with CGSearch* and Range, since CGSearch_P on average achieves the best performance among all the variants.

C. Performance on Varying Query Support

We now assess the performance of our algorithm on queries with different support ranges. Figure 4 presents the results for CGSearch*, CGSearch_P and Range on the query sets F_1 to F_4.

Figures 4(a-b) show the average running time per query and the peak memory consumption. From these two figures, we can see that CGSearch_P is almost two orders of magnitude faster and consumes ten times less memory than Range. The results also show that CGSearch* is even over an order of magnitude faster than CGSearch_P, with comparable memory consumption. For both CGSearch_P and Range, the dominating factor in the running time is the candidate generation process, which involves mining the projected database for CGSearch_P and mining the entire database for Range. On the other hand, the cost of candidate generation is minimal for CGSearch* since most of the queries are processed directly using FGQuery.

We observe that CGSearch_P is slightly slower for processing F_1 and F_4. This is because the cost of candidate generation not only depends on the size of the projected database (i.e., $supp(q)$), but also on the minimum support threshold (i.e., $\frac{lower_{supp(q,g)}}{supp(q)}$). Although the minimum support threshold for F_4 is the largest among all the query sets, its projected database is also the largest, which increases the mining time; 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 greatly influenced by the support of the queries. With the decrease in the support of the queries, the running time of Range increases rapidly from 100 seconds to 400 seconds. For CGSearch*, since



(a) Running Time (b) Memory Consumption

(c) Size of Candidate Set (d) Structural Similarity of Correlated Graphs

Fig. 4. Performance on Varying Query Support

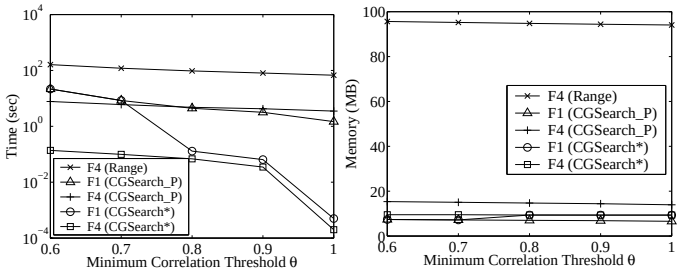
only a small number of queries performs the mining operation for candidate generation, the performance is very stable and significantly better than the other two algorithms.

We show the sizes of the candidate sets of CGSearch*, CGSearch_P and Range in Figure 4(c). The size of the answer set is also shown as a reference. The result shows that the size of the candidate set produced by CGSearch_P is over an order of magnitude smaller than Range and is close to that of the answer set. Note that the set of candidates of CGSearch* is the same as that of Range in this experiment; however, CGSearch* obtains the candidates from the FGQuery rather than mines them from the database as does Range.

We further study the structural similarity of correlated graphs. We compute the *Maximum Common Subgraph (MCS)* of a query q and each of its correlated answer graph g. The structural similarity of q and g is then computed as $\frac{|MCS(q,g)|}{max(|q|,|g|)}$, where $|g|$ denotes the size of a graph g. Figure 4(d) presents the cumulative probability distributions of the structural similarity of correlated graphs in F_1 to F_4. The result shows that most of the answer graphs are structurally dissimilar to query graphs. About 70% of the answer graphs have a structural similarity of less than 0.12 to the query graphs in F_1 and F_4, while for the query graphs in F_2 and F_3, about 60% of the answer graphs have a structural similarity of less than 0.24. The result indicates that the high correlation between a query graph and its answer graph is mostly due to their co-occurrences rather than their structural similarity. This demonstrates the contribution of our new proposal of correlated graphs since correlated graphs are not able to be discovered by existing approaches for structural similarity search.

D. Performance on Varying θ

Figure 5 shows the performance of CGSearch*, 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 but for clarity of presentation, we only present the results for F_1 and F_4. We also do not present F_1 for Range because its running time is too long.



(a) Running Time (b) Memory Consumption

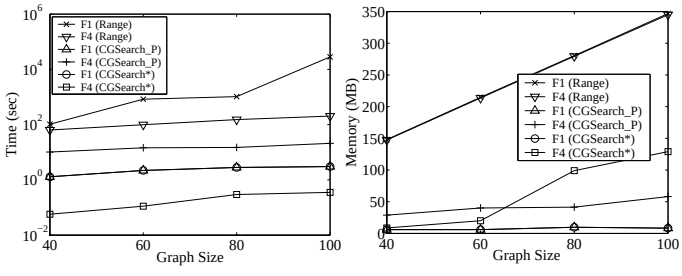
Fig. 5. Performance on Varying θ

As shown in Figure 5, 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, while CGSearch* is near an order of magnitude faster than CGSearch_P. In processing F_1, when $\theta < 0.8$, CGSearch* invokes CGSearch_P to process the queries since their corresponding $lower_{supp(g)}$ is less than σ, while for $\theta \geq 0.8$, with the use of the FGQuery, CGSearch* is significantly faster than CGSearch_P.

E. Performance on Varying Graph Size

Since the graphs in the real dataset are of small size (on average 23 edges per graph), we use synthetic datasets to assess the performance of the algorithms on different graph sizes.

We report the results for F_1 and F_4, which are of the largest and the smallest support ranges, respectively. Figure 6 shows the performance of CGSearch*, CGSearch_P and Range. For F_1, CGSearch_P is up to four orders of magnitude faster and consumes 40 times less memory than Range. Note that CGSearch* invokes CGSearch_P to process F_1. Therefore, the running time of CGSearch* and CGSearch for F_1 is the same. For F_4, CGSearch_P is still over an order of magnitude faster than Range, while CGSearch* is even an order of magnitude faster than CGSearch_P. 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). However, we can use the more efficient algorithm CGSearch* instead of CGSearch_P for processing F_4. The memory consumption of both CGSearch* and CGSearch_P for F_4 is significantly less than that of Range. CGSearch* for graph sizes of 80 and 100 consumes slightly more memory, since the number of FGs for larger graph sizes is larger and consequently the space needed to build the FG-lattice is larger.



(a) Running Time (b) Memory Consumption

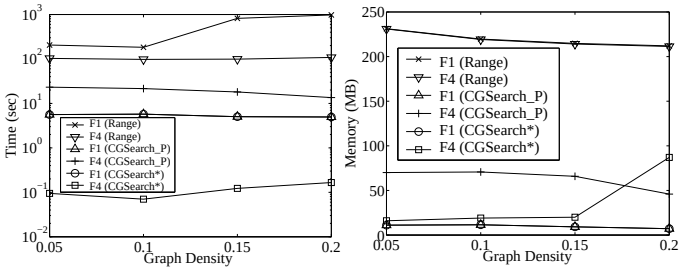
Fig. 6. Performance on Varying Graph Size

Overall, the results in Figure 6 show that our algorithms, both CGSearch* and CGSearch_P, are efficient for all graph sizes tested and their performance is also much more stable than that of Range.

F. Performance on Varying Graph Density

We assess the performance of the algorithms on different graph densities. We test the average densities of 0.05, 0.1, 0.15 and 0.2, which correspond to 50, 35, 30 and 25 nodes per graph, respectively. The average number of edges in a graph is 60.

Again, we present the results for F_1 and F_4 in Figure 7. For F_1, CGSearch_P is more than two orders of magnitude faster than Range. Note that CGSearch* invokes CGSearch_P to process F_1 and hence their running time for F_1 is the same. For F_4, CGSearch_P is almost an order of magnitude faster than Range, but CGSearch* is three orders of magnitude faster than Range. The longer running time of CGSearch_P in processing F_4 is mainly due to the large projected databases of the queries in F_4, since large projected databases result in a more costly candidate generation. In fact, over 99% of the running time is used to mine the candidates in CGSearch_P for F_4. However, in this case, CGSearch* can be used instead of CGSearch_P to process F_4. The memory consumption of both CGSearch* and CGSearch_P is significantly less than that of Range for all densities. The slightly more memory consumption of CGSearch* for F_4 is due to larger number of FGs for building the FG-lattice in FGQuery.



(a) Running Time (b) Memory Consumption

Fig. 7. Performance on Varying Graph Density

For all the densities tested, the results also show that both CGSearch* and CGSearch_P are very stable in processing both F_1 and F_4.

VIII. Generalization of the CGS Problem

In the previous sections, we study an efficient solution to the CGS problem. The CGS problem adopts Pearson's correlation coefficient as the correlation measure; however, there are many other well-established correlation measures proposed in the literature [17]. *Does our method work only for Pearson's correlation coefficient? Or does it work for other correlation measures as well?* In this section, we generalize our problem definition to adopt other measures and show that our method is a general solution for a majority of correlation measures being used.

We first define the generalized CGS problem as follows.

Definition *4: (*Generalized Correlated Graph Search*)* Given a graph database $\mathcal{D}$, a correlation query graph q and a minimum correlation threshold θ, the generalized problem of correlated graph search is *to find the set of all graphs that are correlated with q as defined by a correlation measure $\mathcal{M}$.*

It is challenging to find a general solution for the above generalized CGS problem since the various correlation measures are not only defined differently but also carry very different semantic meanings. We set up the following system of inequalities to model the generalized CGS problem. By solving this system of inequalities, we show how our solution developed for the CGS problem applies to the generalized CGS problem.

$$\begin{cases} supp(q,g) \le supp(g) \\ supp(q,g) \le supp(q) \\ supp(q,g) \ge 0 \\ supp(g) \le 1 \\ \mathcal{M}(supp(q,g), supp(g)) \ge \theta \end{cases}$$

where the first two inequalities are the properties of joint support, the third and fourth inequalities represent the bounds of the support measure, and the last inequality expresses the correlation condition of the generalized CGS problem.

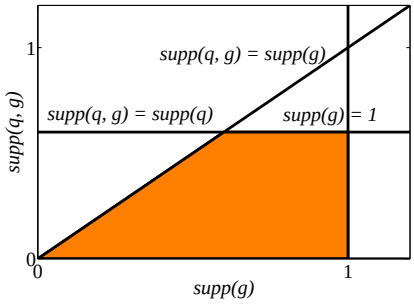


Fig. 8. Graph of the Inequality System

If a graph g is an answer to the generalized CGS problem, the corresponding pair $(supp(g), supp(q,g))$ must satisfy the above inequality system. Thus, the inequality system defines a necessary condition and we can find the set of candidate graphs by solving the inequality system without missing any answer graph. Note that a pair $(supp(g), supp(q,g))$ that satisfies the inequality system is not necessary to correspond to an answer graph, because there may not be such a graph with these support and joint support values in the database.

The solution to the above inequality system can be better visualized by graphing the inequalities and shading the solution region. Figure 8 shows four thick lines that represent the corresponding equalities of the first four inequalities. The shaded trapezoid represents the region where the first four inequalities are true, i.e., the solution to these four inequalities. Therefore, the solution to the inequality system is the overlap of this trapezoid and the region defined by the last inequality.

Now the problem is: how do we plot the last inequality, i.e., the correlation condition, in the graph? To do this, we need to first investigate the properties of a correlation measure. According to Piatetsky-Shapiro [32], a good measure M of two variables A and B should satisfy the following three key properties:

- P1: $M = 0$ if A and B are statistically independent;
- P2: M monotonically increases with $p(A,B)$ when $p(A)$ and $p(B)$ are fixed;
- P3: M monotonically decreases with $p(A)$ (or $p(B)$) when $p(A,B)$ and $p(B)$ (or $p(A)$) are fixed.

Here, $p(A)$ represents the probability of A and $p(A,B)$ represents the joint probability of A and B. In our problem, $p(A)$ is equivalent to $supp(A)$ and $p(A,B)$ is equivalent to $supp(A,B)$.

Now, we state and prove an important property of the correlation condition ($\mathcal{M}(supp(q,g), supp(g)) \ge \theta$) in the following lemma.

Lemma *5:* If a correlation measure $\mathcal{M}$ satisfies P2 and P3, then $supp(q,g)$ is monotonically increasing with $supp(g)$ in the function $\mathcal{M}(supp(q,g), supp(g)) = \theta$.

Proof: Let g_1 and g_2 be two graphs, where $supp(g_1) \ge supp(g_2)$. We show that $supp(q,g_1) \ge supp(q,g_2)$, given that $\mathcal{M}(supp(q,g_1), supp(g_1)) = \mathcal{M}(supp(q,g_2), supp(g_2)) = \theta$.

First, since $\mathcal{M}$ satisfies P3, by fixing $supp(q, g_1)$, it follows that $\mathcal{M}(supp(q, g_1), supp(g_1)) \leq \mathcal{M}(supp(q, g_1), supp(g_2))$. Then, since $\mathcal{M}(supp(q, g_1), supp(g_1)) = \mathcal{M}(supp(q, g_2), supp(g_2))$, it follows that $\mathcal{M}(supp(q, g_2), supp(g_2)) \leq \mathcal{M}(supp(q, g_1), supp(g_2))$. Finally, since $\mathcal{M}$ satisfies P2, it follows that $supp(q, g)$ monotonically increases with $\mathcal{M}$ when $supp(g)$ and $supp(q)$ are fixed. Therefore, we have $supp(q, g_2) \leq supp(q, g_1)$ and the result follows. ■

According to Lemma 5, the curve of the correlation condition should be plotted from the lower left to the upper right in Figure 8. Moreover, according to P2, the region where the inequality of the correlation condition is true should be located in the upper left of the figure. Therefore, the overlap of this region and the shaded trapezoid, i.e., the solution to the inequality system, depends on the intersection points of the curve of the correlation condition and the four sides of the trapezoid.

We now investigate the cases of the intersection points to derive the solution to the inequality system. We first find that the correlation curve has no intersection point with the line "$supp(q, g) = 0$". This is because, when the joint support of two variables is zero, the correlation of these two variables defined by any correlation measure is no greater than zero; while the correlation of two variables represented by a point in the curve is θ, which is greater than zero. Therefore, there are two cases when the correlation curve intersects with the other three sides of the trapezoid as follows:

Case 1: The correlation curve has an intersection point with the line "$supp(q, g) = supp(g)$".

Case 2: The correlation curve has no intersection point with the line "$supp(q, g) = supp(g)$".

We now discuss these two cases in detail.

In Case 1, there is a lower bound for $supp(q, g)$, i.e., the value of $supp(q, g)$ of the intersection point. It is a lower bound since the region where the correlation condition is true is located above the curve. Therefore, we can solve the generalized CGS problem efficiently by generating the candidates from the projected database as in the CGS problem. Moreover, there is also a lower bound for $supp(g)$, i.e., the value of $supp(g)$ of the intersection point. This lower bound is the same as the lower bound for $supp(q, g)$ since the intersection point is on the line "$supp(q, g) = supp(g)$". Therefore, if the lower bound of $supp(g)$ is no less than the minimum support threshold for building the index (as discussed in Section VI), we can compute the query results efficiently using FGQuery and avoid candidate generation through a more costly mining process.

By investigating the commonly used measures introduced in [17], we find that, among all the fourteen measures that possess both properties of P2 and P3, ten of them fall under Case 1. They are as follows:

- Pearson's correlation coefficient;
- Cohen's kappa coefficient;
- Mutual information;
- Cosine measure;
- Piatetsky-Shapiro's measure;
- Certainty factor;
- Added value;
- Collective strength;
- Jaccard index;
- Klosgen's evaluation function.

Therefore, the generalized CGS problem defined by most correlation measures can be efficiently solved by our current solution. The only difference is that the expressions of the bounds in Lemmas 1 and 3 vary for different measures. The bounds can be obtained by computing the intersection point of the correlation condition with the line "$supp(q, g) = supp(g)$". The corresponding heuristic rules for further reducing the search space can also be obtained in a similar way as in the case of Pearson's correlation coefficient.

Figure 9 shows the graph of the inequality system when Pearson's correlation coefficient is applied as the correlation measure. The thick curve represents the cases when the Pearson's correlation coefficient of two graphs equals θ. The shaded region, which is the overlap of the trapezoid and the region above the thick curve, is the solution to the inequality system. According to this solution, we can identify the lower and upper bounds for both $supp(g)$ and $supp(q, g)$ as indicated in the axes of the figure. These bounds are the key to the design of our efficient solution: candidate generation from projected databases and effective use of the index for answering high-support queries.

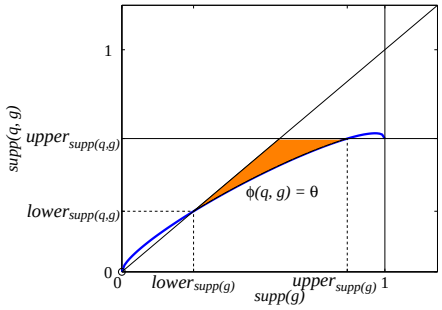


Fig. 9. Graph of the Inequality System when $\mathcal{M}$ is Pearson's Correlation Coefficient

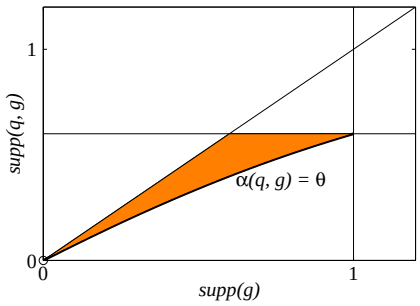


Fig. 10. Graph of the Inequality System when $\mathcal{M}$ is Odds Ratio

In Case 2, the correlation curve intersects with either the line "$supp(q, g) = supp(q)$" or the line "$supp(g) = 1$". Thus, there is no non-trivial lower bound for both $supp(q, g)$ and $supp(g)$. To generate the candidate set from either the projected database or the whole database, the trivial minimum support threshold of 0 has to be used. In many real applications, mining all FGs from the whole database is infeasible. Moreover, mining FGs from the projected database is always cheaper than from the whole database using the same minimum support threshold. Therefore, candidate generation from the projected database is the only existing solution, although it can still be costly when the projected database is large or dense. Fortunately, the number of correlation measures that fall within Case 2 is very small, including odds ratio and its two normalizations (Yule's Q and Yule's Y) and the interest measure. Figure 10 gives the graph of the inequality

system when odds ratio is used as the correlation measure. It can be seen from the figure that there is no non-trivial lower bounds for the support values in the solution region.

In conclusion, the discussions in this section show that our algorithm CGSearch* still serves as an efficient and effective solution when most of the correlation measures are used to generalize the CGS problem.

IX. 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, the ϕ correlation coefficient, are prevalently used as a correlation measure. Sakurai et al. [9] use Pearson's correlation coefficient to define the lag correlation between two time sequences. Xiong et al. [5] apply the ϕ correlation coefficient to define the strongly correlated pairs in transaction databases. An upper bound of ϕ, as well as monotonic properties of the upper bound, are identified to facilitate the efficient mining process. Recently, Zhang and Feigenbaum [6] also adopt the ϕ correlation coefficient to measure 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 the 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 [1], interest [1], all-confidence [2], [3], bond [3], h-confidence [4], and so on. For multimedia data, Pan et al. [8] use random walks 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. [7] utilize mutual information and all-confidence to define the correlated patterns.

X. Conclusions

We formulate the problem of CGS, which finds the set of graphs that exhibit high correlation to a given query graph, as measured by Pearson's correlation coefficient. The search space of the problem is exponential; however, by deriving the theoretic bounds for the support of a candidate graph, we effectively reduce the search space to a small set of candidates mined from the projected database of the query graph. We develop three effective heuristic rules to further reduce the size of the candidate set. Using the above results, we devise 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. We further eliminate the mining process of candidate generation for queries of high support by FGQuery. Combining FGQuery and CGSearch, we present an improved algorithm CGSearch*. The experimental results justify the efficiency and effectiveness of our candidate generation and heuristic rules. Compared with the approach that mines the candidates from the whole database by a support range, our solution is orders of magnitude faster and consumes much less memory. More importantly, our algorithm achieves very stable performance when varying the support of the queries, the minimum correlation threshold, the graph size, as well as the graph density.

A significant finding of this paper is that, when the CGS problem is generalized to adopt other correlation measures, our algorithm still serves as an efficient solution for most of the existing measures [17].

Acknowledgement. We thank Dr. Xifeng Yan and Prof. Jiawei Han for providing us the executable of gSpan.

References

[1] S. Brin, R. Motwani, and C. Silverstein, "Beyond market baskets: generalizing association rules to correlations," in *SIGMOD*, 1997, pp. 265–276.
[2] S. Ma and J. L. Hellerstein, "Mining mutually dependent patterns," in *ICDM*, 2001, pp. 409–416.
[3] E. R. Omiecinski, "Alternative interest measures for mining associations in databases," *IEEE TKDE*, vol. 15, no. 1, pp. 57–69, 2003.
[4] H. Xiong, P.-N. Tan, and V. Kumar, "Hyperclique pattern discovery," *DMKD*, vol. 13, no. 2, pp. 219–242, 2006.
[5] 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*, vol. 18, no. 4, pp. 493–508, 2006.
[6] J. Zhang and J. Feigenbaum, "Finding highly correlated pairs efficiently with powerful pruning," in *CIKM*, 2006, pp. 152–161.
[7] Y. Ke, J. Cheng, and W. Ng, "Mining quantitative correlated patterns using an information-theoretic approach," in *KDD*, 2006, pp. 227–236.
[8] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu, "Automatic multimedia cross-modal correlation discovery," in *KDD*, 2004, pp. 653–658.
[9] Y. Sakurai, S. Papadimitriou, and C. Faloutsos, "AutoLag: Automatic discovery of lag correlations in stream data," in *ICDE*, 2005, pp. 159–160.
[10] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne, "The protein data bank," *Nucleic Acids Research*, vol. 28, pp. 235–242, 2000.
[11] M. Kanehisa and S. Goto, "KEGG: Kyoto encyclopedia of genes and genomes," *Nucleic Acids Research*, vol. 28, pp. 27–30, 2000.
[12] "National library of medicine," *http://chem.sis.nlm.nih.gov/chemidplus*.
[13] "The International Network for Social Network Analysis," *http://www.insna.org/*.
[14] S. Raghavan and H. Garcia-Molina, "Representing Web graphs," in *ICDE*, 2003, pp. 405–416.
[15] "DBLP Dataset," *http://dblp.uni-trier.de/xml/*.
[16] Y. Ke, J. Cheng, and W. Ng, "Correlation search in graph databases," in *KDD*, 2007, pp. 390–399.
[17] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the right interestingness measure for association patterns," in *KDD*, 2002, pp. 32–41.
[18] L. Holder, D. Cook, and S. Djoko, "Substructure discovery in the subdue system," in *KDD*, 1994, pp. 169–180.
[19] J. W. Raymond, E. J. Gardiner, and P. Willett, "RASCAL: calculation of graph similarity using maximum common edge subgraphs." *Comput. J.*, vol. 45, no. 6, pp. 631–644, 2002.
[20] X. Yan, F. Zhu, P. S. Yu, and J. Han, "Feature-based similarity search in graph structures," *ACM TODS*, vol. 31, no. 4, pp. 1418–1453, 2006.
[21] H. He and A. K. Singh, "Closure-tree: An index structure for graph queries," in *ICDE*, 2006, p. 38.
[22] D. Williams, J. Huan, and W. Wang, "Graph database indexing using structured graph decomposition," in *ICDE*, 2007, pp. 976–985.
[23] S. A. Cook, "The complexity of theorem-proving procedures," in *STOC*, 1971, pp. 151–158.
[24] M. Kuramochi and G. Karypis, "Frequent subgraph discovery." in *ICDM*, 2001, pp. 313–320.
[25] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *PKDD*, 2000, pp. 13–23.
[26] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *ICDM*, 2002, p. 721.
[27] H. Reynolds, *The analysis of cross-classifications*. New York: The Free Press, 1977.
[28] G. U. Yule, "On the methods of measuring association between two attributes," *Journal of the Royal Statistical Society*, vol. 75, no. 6, pp. 579–652, 1912.
[29] S. Nijssen and J. N. Kok, "A quickstart in frequent structure mining can make a difference," in *KDD*, 2004, pp. 647–652.
[30] X. Yan, P. S. Yu, and J. Han, "Graph indexing based on discriminative frequent structure analysis," *ACM TODS*, vol. 30, no. 4, pp. 960–993, 2005.
[31] J. Cheng, Y. Ke, and W. Ng, "FG-Index: Towards verification-free query processing on graph databses," in *SIGMOD*, 2007, pp. 857–872.
[32] G. Piatetsky-Shapiro, "Discovery, analysis, and presentation of strong rules," in *Knowledge Discovery in Databases*, 1991, pp. 229–248.