Mining Frequent Cross-Graph Quasi-Cliques

16

DAXIN JIANG
Microsoft Research Asia
and
JIAN PEI
Simon Fraser University

Joint mining of multiple datasets can often discover interesting, novel, and reliable patterns which cannot be obtained solely from any single source. For example, in bioinformatics, jointly mining multiple gene expression datasets obtained by different labs or during various biological processes may overcome the heavy noise in the data. Moreover, by joint mining of gene expression data and protein-protein interaction data, we may discover clusters of genes which show coherent expression patterns and also produce interacting proteins. Such clusters may be potential pathways.

In this article, we investigate a novel data mining problem, *mining frequent cross-graph quasi-cliques*, which is generalized from several interesting applications in bioinformatics, cross-market customer segmentation, social network analysis, and Web mining. In a graph, a set of vertices S is a γ-quasi-clique ($0 < \gamma \leq 1$) if each vertex v in S directly connects to at least $\gamma \cdot (|S| - 1)$ other vertices in S. Given a set of graphs $G_1, \ldots, G_n$ and parameter min_sup ($0 < min\_sup \leq 1$), a set of vertices S is a frequent cross-graph quasi-clique if S is a γ-quasi-clique in at least $min\_sup \cdot n$ graphs, and there does not exist a proper superset of S having the property.

We build a general model, show why the complete set of frequent cross-graph quasi-cliques cannot be found by previous data mining methods, and study the complexity of the problem. While the problem is difficult, we develop practical algorithms which exploit several interesting and effective techniques and heuristics to efficaciously mine frequent cross-graph quasi-cliques. A systematic performance study is reported on both synthetic and real data sets. We demonstrate some interesting and meaningful frequent cross-graph quasi-cliques in bioinformatics. The experimental results also show that our algorithms are efficient and scalable.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms: Algorithms, Experimentation

A preliminary version of this paper appears as Pei et al. [2005].
This research is partly supported by NSF grant IIS-0308001, NSERC Discovery Grant, NSERCE CRD Grant, and IBM Eclipse Innovation Award. All opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.
Authors' addresses: D. Jiang, 4F Sigma Building, 49 Zhichun Road, Haidian District, Beijing, China, 100080; email: djiang@microsoft.com; J. Pei, School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, BC Canada V5A IS6; email: jpei@cs.sfu.ca.

Additional Key Words and Phrases: Graph mining, bioinformatics, clique, joint mining

ACM Reference Format:

Jiang, D. and Pei, J. 2009. Mining frequent cross-graph quasi-cliques. ACM Trans. Knowl. Discov. Data. 2, 4, Article 16 (January 2009), 42 pages. DOI = 10.1145/1460797.1460799 http://doi.acm.org/10.1145/1460797.1460799

1. INTRODUCTION

In many applications, data is often collected, organized and stored in multiple sources. Many hidden patterns crossing multiple pieces of data cannot be found by mining only one single data source. Therefore, advanced data analysis in practice calls for *joint mining of multiple datasets*, which can often discover interesting, novel, and reliable patterns that cannot be obtained solely from any single source.

Example 1 (*Joint Mining of Gene Expression Data*). The recent DNA microarray technology has made it possible to measure the expression levels of thousands of genes on a set of samples [Chu et al. 1998; DeRisi et al. 1997; Gasch et al. 2000; Spellman et al. 1998]. Technically, a microarray dataset (also called a *gene expression dataset*) is a matrix $W = \{w_{i,j}\}$ for a set of genes G and a set of samples S, where $w_{i,j}$ $(1 \leq i \leq |G|, 1 \leq j \leq |S|)$ is the expression level of gene g_i on sample s_j [Brazma and Vilo 2000]. From gene expression data, biologists are often interested in finding groups of genes which exhibit similar expression profiles on the samples [Alon et al. 1999; Cho et al. 1998; Eisen et al. 1998; Spellman et al. 1998; Tamayo et al. 1999; Tavazoie et al. 1999]. A group of such genes are called *co-expressed genes* and are likely to have similar functions [Eisen et al. 1998; Lee et al. 2004].

To find groups of coexpressed genes, we can represent a gene expression dataset $\{w_{i,j}\}$ using a *gene coexpression graph* G_E, where each vertex corresponds to one gene and two vertices are connected by an edge if the corresponding genes have similar expression profiles. After we construct the *gene coexpression graph* G_E, the problem of finding groups of coexpressed genes in $\{w_{ij}\}$ can then be converted into finding cliques in G_E.

Finding cliques in a graph is a problem that has been investigated for a long time. However, gene expression data derived from the microarray experiments is typically noisy [Beissbarth et al. 2000; Tseng et al. 2001]. Consequently, the mining results from a single dataset may not be reliable. To overcome this problem, we can collect gene expression data from various sources. For example, the expression levels of the yeast genes have been monitored by different labs and during various biological processes [Chu et al. 1998; DeRisi et al. 1997; Gasch et al. 2000; Spellman et al. 1998]. The measurements from each microarray experiment can be regarded as an independent data source and modeled by an individual gene coexpression graph. Although the noise ratio of each individual data source may be high, we can expect that the mining results consistently supported by various data sources are more reliable, because the errors in different data sources are usually independent of each other [Lee et al. 2004; Moreau et al. 2003; Stuart et al. 2003].

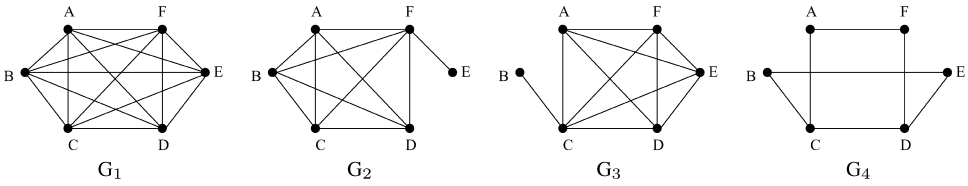


Fig. 1. $\{A, C, D, F\}$ is a clique conserved in graphs G_1, G_2, and G_3.

Figure 1 shows four example gene coexpression graphs G_1, G_2, G_3, and G_4. If we mine graph G_1 alone, we can find the whole set of genes $\{A, B, C, D, E, F\}$ as a cluster of coexpressed genes since it is a clique in G_1. Similarly, mining graphs G_2, G_3, and G_4 individually finds clusters $\{A, B, C, D, F\}$, $\{A, C, D, E, F\}$, and $\emptyset$, respectively. On the other hand, if we consider the four graphs altogether, we may recognize cluster $\{A, C, D, F\}$ as a reliable pattern since it is a clique in three out of the four graphs.

Example 1 shows that joint mining of multiple graphs can improve the reliability of the mining results. Moreover, joint mining of multiple graphs may also find novel patterns which cannot be obtained from any single data source. Let us consider another interesting application in bioinformatics—joint mining of gene expression data and protein-protein interaction data.

Example 2 (*Joint Mining of Gene Expression Data and Protein-Protein Interaction Data*). Several recent high-throughput biotechniques, such as yeast two-hybrid analysis [Ito et al. ; Uetz et al. 2000], mass spectrometry [Gavin et al. 2002; Ho et al. 2002] and synthetic lethality screen [Tong et al. 2001; Tong et al. 2004], have generated large-scale protein-protein interaction data. In general, a protein-protein interaction data set can be modeled as a *protein-protein interaction graph* where each vertex corresponds to one protein and two vertices are connected by an edge if the corresponding two proteins interact with each other. In biology, a cluster of proteins which often interact with each other are likely to be functionally related. In other words, a cluster of functional-related proteins are almost a clique (thus called quasi-clique) in the protein-protein interaction graph. To be specific, each protein in a quasi-clique is not necessarily connected to all the other proteins in the same quasi-clique; instead, it is connected to at least a portion γ $(0 < \gamma \leq 1)$ of the other vertices, where γ is a user-specified parameter. Clearly, cliques are special cases of quasi-cliques where $\gamma = 1$.

Naturally, protein-protein interaction data is complementary to gene expression data since proteins are products of genes. Joint mining of these two types of data can find novel patterns which cannot be obtained from any single data source. For example, many biological pathways exhibit two properties: their genes exhibit similar gene expression profiles, and the protein products of the genes often interact [Segal et al. 2003]. Such pathways cannot be found solely in either gene expression data or protein-protein interaction data since the membership of a pathway must be verified in both types of data.

To find biological pathways, we can combine gene coexpression graphs and protein-protein interaction graphs using a surjective (that is, onto) mapping

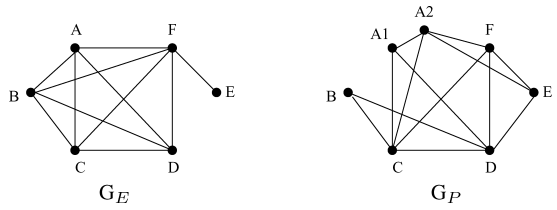


Fig. 2. $\{A, C, D, F\}$ is a cross-graph quasi-clique in graphs G_E and G_P.

$f : P \rightarrow G$, where P is the set of proteins, G is the set of genes, and $f(p) = g$ if gene $g \in G$ produces protein $p \in P$. A group of genes are likely to participate in the same biological pathway if they form a *cross-graph quasi-clique*, that is, they form a quasi-clique (with $\gamma = 1$) in the gene coexpression graph, and at the same time, their corresponding proteins also form a quasi-clique (with $\gamma \leq 1$) in the protein-protein interaction graph.

Figure 2 shows an example gene coexpression graph G_E and an example protein-protein interaction graph G_P. Both proteins A_1 and A_2 map to gene A, and the other proteins map to the genes with the same label. The group of genes $\{A, C, D, F\}$ is a cross-graph quasi-clique since it forms a quasi-clique (with $\gamma_E = 1$) in graph G_E and the corresponding group of proteins $\{A_1, A_2, C, D, F\}$ forms a quasi-clique (with $\gamma_P = 0.75$) in graph G_P.

Examples 1 and 2 motivate a novel problem of mining multiple graphs as follows. Consider a set of n graphs where the vertices of all graphs can be mapped to a common set of objects. We are interested in finding groups of objects such that the corresponding vertices form a quasi-clique in at least $(n \cdot min\_sup)$ graphs, where min_sup is a user-specified parameter. Such groups of objects are called *frequent cross-graph quasi-cliques*.

Mining frequent cross-graph quasi-cliques has important applications in various domains. For example, in social network analysis, joint mining of collaboration graphs and citation graphs can simultaneously explore the coauthor relationship and reference relationship and identify reliable research communities as well as papers on similar topics. As another example, in marketing and customer relation management, customer groups with consistent behaviors in multiple aspects, such as similar purchase patterns, saving patterns, and responses to marketing campaigns, are likely to be more coherent and reliable. Moreover, in the World-Wide Web, the relationship between Web sites can be described by both their content and their link structure. Joint mining of web content and Web structure may effectively distinguish real Web communities from malicious link farms. For more examples on joint mining of multiple sources, Page and Craven [Page and Craven 2003] surveyed the biological applications of mining multiple tables, such as pharmacophore discovery, gene regulation, information extraction from text and sequence analysis.

As shown, mining multiple graphs may discover reliable and novel patterns that cannot be found by conventional data mining approaches. Finding cliques in a graph is a problem that has been investigated for a long time. Computing quasi-cliques in one graph was also the topic of some previous studies, such as [Abello et al. 2002; Matsuda et al. 1999]. One may wonder, "*Can the complete*

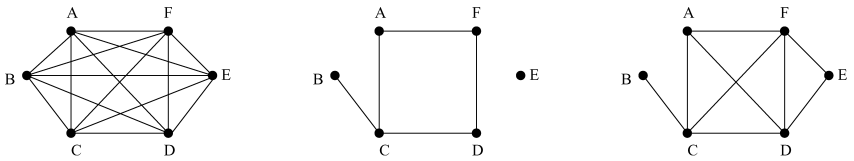


Fig. 3. The integrated graphs based on various integration schemas.

set of frequent cross-graph quasi-cliques be mined easily by extending the existing algorithms for finding cliques or quasi-cliques?"

Example 3 (*Mining Integrated Graph*). We can integrate multiple graphs into one by designing an integration function between data objects. The integration function combines the similarities between data objects in different data sets in some weighted manner. Then, we can find quasi-cliques in the integrated graph.

Consider the four graphs in Example 1 again. Figure 3 shows the integrated graphs based on various integration functions. In graphs $G_\cup$, $G_\cap$ and G_{freq}, respectively, two vertices u and v are connected by an edge in the integrated graph if edge (u, v) appears in (a) any one of the four graphs; (b) all of the four graphs; and (c) at least three out of the four graphs.

If we consider the integrated graphs $G_\cup$ and $G_\cap$, we will report clusters $\{A, B, C, D, E, F\}$ and $\emptyset$, respectively. In both cases, the real cluster $\{A, C, D, F\}$, which is conserved in three graphs, cannot be correctly identified. Finally, if we consider the integrated graph G_{freq}, although the cluster $\{A, C, D, F\}$ can be identified, another cluster $\{D, E, F\}$ will also be reported. However, $\{D, E, F\}$ is conserved in only two out of the four graphs.

In general, there could be two approaches to graph integration. In the first approach, an edge (u, v) in the integrated graph is only assigned with a weight derived from the integration function; it does not carry the information from which original graphs the edge was derived. For example, graphs $G_\cup$, $G_\cap$, and G_{freq} in Figure 3 can be considered as derived from functions $\max_{i=1}^{n} w_i(u, v)$, $\min_{i=1}^{n} w_i(u, v)$, and $\sum_{i=1}^{n} w_i(u, v) \geq \delta_{freq}$, respectively, where δ_{freq} is the threshold for frequent patterns, $w_i(u, v) = 1$ if (u, v) is an edge in graph G_i $(1 \leq i \leq n)$ and $w_i(u, v) = 0$ otherwise. Clearly, the integrated graphs derived from these functions are *unreconstructable* in the sense that we cannot reconstruct the original graphs from the integrated graph. For example, consider edge (A, B) in the integrated graph $G_\cup$. We cannot infer in which original graphs the edge appears. Similarly, although there is no edge between D and E in the integrated graph $G_\cap$, edge (D, E) still may appear in some original graphs.

We will show that, in general, frequent cross-graph quasi-cliques cannot be mined from an unreconstructable integrated graph. To elaborate the idea, consider two graphs G_1 and G_2 on the same set of vertices V. Let γ_1 and γ_2 be the user specified parameters for G_1 and G_2, respectively. Let f_{inter} be an unreconstructable integration function, that is, from $f_{inter}(w_1(u, v), w_2(u, v))$, we cannot tell the value of $w_1(u, v)$ or $w_2(u, v)$. Let G_{inter} be the integrated

graph derived from f_{inter}, that is, the vertex set $V(G_{inter}) = V$ and the edge set $E(G_{inter}) = \{(u, v) | f_{inter}(w_1(u, v), w_2(u, v)) = 1\}$. Without loss of generality, let $0 \leq \gamma_1 < \gamma_2 \leq 1$. Since f_{inter} is unreconstrutable, in general, a γ_{inter}-quasi-clique in G_{inter} cannot be guaranteed to be both a γ_1-quasi-clique in G_1 and a γ_2-quasi-clique in G_2. On the other hand, in some special cases as will be discussed in Section 4.2.2, the unreconstructable integration approach may be applied.

The second approach to graph integration is to record for each pair of vertices the edge information in the original graphs. This approach allows the reconstruction of the original graphs from the integrated one, and thus guarantees the correctness of the mined patterns. However, if we rely on such edge information when searching the integrated graph, the searching process will be equivalent to searching individual graphs in parallel. In this paper, we propose methods searching multiple graphs individually. These methods can also be viewed as searching a virtual integrated graph with minimal edge information in original graphs recorded. For example, we develop several pruning techniques (see Section 4.2) to remove futile edges from original graphs. This is equivalent to reducing redundant edge information in the virtual integrated graph. The pruned virtual integrated graph may be unreconstructable while it still retains all frequent cross-graph quasi-cliques.

Although there are extensive studies on cliques and quasi-cliques, to the best of our knowledge, our study is the first one to address the following two issues at the same time. First, we investigate *mining from multiple graphs* the quasi-cliques. We propose a general model on frequent cross-graph quasi-cliques and mining. Second, we compute the *complete set of frequent cross-graph quasi-cliques*. Many of the previous studies focused on finding one quasi-clique (from one graph) with an optimization goal, such as maximizing the number of vertices in the clique. However, many data mining applications require the completeness of the answers.

Mining the complete set of quasi-cliques from multiple graphs is challenging. A naïve method may have to examine a huge number of possible combinations of vertices and edges over the graphs, which is computationally expensive or even prohibitive on large graphs (e.g., graphs with thousands of vertices and tens of thousands of edges).

Bearing the above challenges, in this paper, we tackle the problem of mining frequent cross-graph quasi-cliques and make the following contributions.

(1) *We propose a novel and general model for the problem of mining frequent cross-graph quasi-cliques*. We show that frequent cross-graph quasi-cliques are interesting and meaningful in some applications.
(2) *We investigate the complexity of the problem and develop practical algorithms to tackle the problem*. We show that the problem is NP-hard. We develop two practical algorithms, *Crochet* and *Crochet*$^+$, to mine frequent cross-graph quasi-cliques. These two algorithms exploit several interesting and effective techniques and heuristics to prune the search space sharply.
(3) *We present a systematic performance study on* Crochet *and* Crochet$^+$ *to verify our design using both synthetic and real data sets*. The experimental

results show that frequent cross-graph quasi-cliques are interesting in real applications and the algorithms work well in practice.

The remainder of the article is organized as follows. In Section 2, we discuss related work. In Section 3, we present the general model of mining frequent cross-graph quasi-cliques, and also show the hardness of the problem. In Sections 4 and 5, we develop algorithms from mining cross-all-graphs quasi-cliques and general frequent cross-graph quasi-cliques, respectively. A systematic performance study is reported in Section 6. Section 7 concludes the article.

2. RELATED WORK

To the best of our knowledge, Abello et al. [2002] and Matsuda et al. [1999] are the two previous studies most related to this paper Abello et al. [2002] defined a γ-clique in a graph G as a subset of vertices $S \subseteq V(G)$ such that the induced graph on S is connected and $|E(G(S))| \geq \gamma \cdot \binom{|S|}{2}$. They also proposed a greedy randomized adaptive search algorithm, *GRASP*, to find a γ-clique with the maximum size. Matsuda et al. [1999] introduced a definition of quasi-clique similar to ours in this paper. However, instead of finding the complete set of quasi-cliques in the graph, they proposed an approximation algorithm to cover all the vertices in the graph G with a minimum number of p-quasi-complete subgraphs.

The critical difference between this paper and the above two is that both Abello et al. [2002] and Matsuda et al. [1999] neither find the complete set of quasi-cliques, nor address mining multiple graphs.

To a more general extent, graph mining has become an important topic in data mining. For example, mining frequent substructures and subgraph patterns from many graphs (that is, a graph database) has been studied intensively [Bayada et al. 1992; Takahashi et al. 1987; Holder et al. 1994; Inokuchi et al. 2000; Kuramochi and Karypis 2001; Yan and Han 2000, 2003; Wang et al. 2004]. Yan et al. [2004] used frequent graph patterns to index graphs. Frequent graph pattern mining in those previous studies focuses on finding the common embedded subgraphs that appear in many graphs, which is very different from the problem of mining cross-graph quasi-cliques investigated in this study. For a cross-graph quasi-clique, the induced graphs on the clique can be very different from graph to graph. Therefore, those frequent graph pattern mining algorithms cannot be extended to mine cross-graph quasi-cliques.

In addition to frequent graph pattern mining, Palmer et al. [2002] developed a fast and scalable tool *ANF* to answer various complex analytical queries from massive graphs that may not be able to fit into main memory. Faloutsos et al. [2004] investigated the problem of fast discovery of connection subgraphs which nicely capture the relationship between pairs of nodes in large social networks graphs. Jeh and Widom [2004] proposed the problem of mining the space of graph properties.

Besides graph mining, graph-based algorithms are applied to cluster large data sets. A data set can be modeled as a graph, and the problem of clustering can be converted to some traditional graph problems, such as finding (quasi-)

cliques or minimum cut in the graph. For example, the spectral clustering approach [Ding et al. 2001] can be viewed as finding the relaxed optimal normalized cuts in a weighted graph. A spectral clustering algorithm was presented in Ng et al. [2001] using k eigenvectors of the adjacency matrix simultaneously.

As another frontier of data mining research, mining from multiple sources has received more and more attention. [Dzeroski and Raedt 2003] is a collection of good examples of techniques and applications of mining multiple relational tables.

On the application side, recent technical advances have enabled collections of many different types of biological data at a genome-wide scale, such as DNA and protein sequences, gene expression measurements, and protein-protein interactions. Various clustering approaches, including the graph-based algorithms, have been developed to explore interesting patterns in those data sets. For example, Hartuv and Shamir [2000] proposed an algorithm *HCS* to find groups of genes that have similar expression patterns. *HCS* recursively splits the weighted gene graph G into a set of highly connected components along the minimum cut. Each highly connected component is considered as a gene cluster. Motivated by *HCS*, Shamir and Sharan [2000] developed an algorithm *CLICK*. Ben-dor et al. [1999] presented a heuristic algorithm *CAST* to iteratively identify maximal cliques once at a time. Xu et al. [2002] generated a *Minimum Spanning Tree* (MST) from the weighted gene graph G. By removing $(K - 1)$ edges from the generated MST, the data set is partitioned into K clusters. Moreover, Enright et al. [2002] developed *TRIBE-MCL* based on *MCL* [Dongen 2000], a graph-based algorithm using flow simulation, to efficiently detect protein families from large protein sequence databases. Bu et al. [2003] used the spectral clustering method [Ng et al. 2001] to analyze the topological structure in the protein interaction graphs. Moreover, Bader and Hogue [2003] proposed a heuristic approach, *MCODE*, based on the concept of scale-free networks [Barabási and Albert 1999] to find molecular complexes in large protein interaction graphs.

Recently, joint mining of multiple biological data sets has received intense interest. As a pioneer work, Segal et al. [2003] proposed a unified probabilistic model to learn the pathways from gene expression data and protein interaction data. However, their method requires the users to input the number of pathways that is usually unknown in advance.

3. MODEL AND PROBLEM DEFINITION

In this section, we propose a general model for mining frequent cross-graph quasi-cliques and show the complexity of the problem.

3.1 Quasi-Complete Graph

In this paper, we consider *simple graphs* only, that is, the graphs without self-loops or multiedges. Moreover, each vertex in a graph has a unique label. When mining multiple graphs, we associate vertices across graphs through surjective mappings.

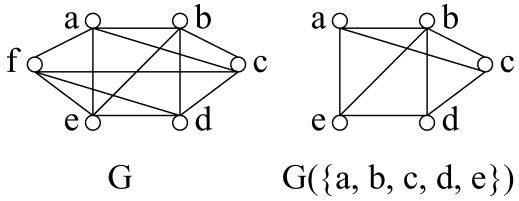


Fig. 4. A graph and an induced subgraph.

For graph G, let $V(G)$ and $E(G)$ denote the sets of vertices and edges of G, respectively. For vertices $u, v \in V(G)$, let $d(u, v)$ be the number of edges in the shortest path between u and v. For a vertex $u \in V(G)$, $N(u)$ is the set of neighbors of u, that is, $N(u) = \{v|(u, v) \in E(G)\}$. Moreover, we define $N^k(u) = \{v \mid d(u, v) \leq k\}$ for $(k \geq 1)$. Clearly, $N^{(|V(G)|-1)}(u)$ is the set of vertices that are connected to u. We also denote this set by $N^*(u)$.

In graph G, let $U \subseteq V(G)$ be a subset of vertices. The *subgraph induced on* U, denoted by $G(U)$, is the subgraph of G whose vertex-set is U and whose edge-set consists of all the edges in G that have both endpoints in U, that is, $G(U) = (U, E_U)$, where $E_U = \{(u, v)|(u, v) \in E(G) \wedge u, v \in U\}$.

A *complete graph* is a graph such that every pair of vertices is joined by an edge. In a graph G, a subset of vertices $S \subseteq V(G)$ is a *clique* if the subgraph induced on S, that is, $G(S)$, is a complete graph, and no proper superset of S has this property. Please note that, there can be more than one clique in a graph, and the cliques may not be exclusive. That is, two cliques may share some common vertices.

Definition 1 (*Quasi-Complete Qraph and Quasi-Clique*). A graph G is a *γ-quasi-complete graph* $(0 < \gamma \leq 1)$ if every vertex in the graph has a degree at least $\gamma \cdot (|V(G)| - 1)$.

In a graph G, a subset of vertices $S \subseteq V(G)$ is a *γ-quasi-clique* $(0 < \gamma \leq 1)$ if $G(S)$ is a γ-quasi-complete graph, and no proper superset of S has this property.

Clearly, a 1-quasi-complete graph is a complete graph, and a 1-quasi-clique is a clique. Complete graphs have the following property.

PROPERTY 1 (ANTI-MONOTONICITY OF COMPLETE QRAPHS). *In a graph G, let $S \subseteq V(G)$. If $G(S)$ is a complete graph, then, for any subset $S' \subset S$, $G(S)$ is also a complete graph.*

In general, the anti-monotonicity does not hold for quasi-complete graphs. That is, for a γ-quasi-complete graph G $(0 < \gamma < 1)$, $G(S)$ may not be a γ-quasi-complete graph for an $S \subset V(G)$.

Example 4 (*Quasi-Complete Qraph*). Consider graph G in Figure 4. It is a 0.8-quasi-complete graph, since every vertex has a degree of $4 = (6 - 1) \times 0.8$.

Interestingly, a subgraph induced on any subset of 5 vertices is not a 0.8-quasi-complete graph. As an example, the subgraph induced on $\{a, b, c, d, e\}$ is also shown in Figure 4. Vertices a, c, d and e in the induced subgraph have a degree of $3 < (5 - 1) \times 0.8 = 3.2$.

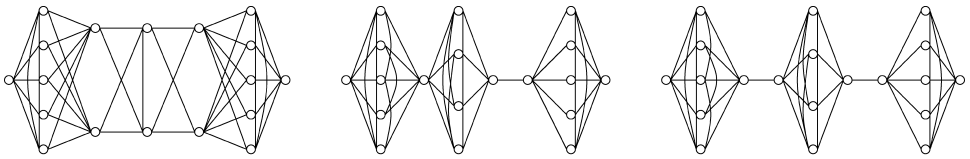


Fig. 5. The cases where $\frac{1}{2} > \gamma \geq \frac{2}{N-1}$.

For γ-quasi-complete graphs, the parameter γ controls the compactness of the graph. That is, with a larger γ, each vertex connects to more other vertices and thus the graph is more compact. One measure of the compactness of a graph is the diameter. The *diameter of* G, denoted by $diam(G)$, is defined as $diam(G) = \max_{u,v \in V(G)}\{d(u,v)\}$. It is interesting to examine the relationship between diameter of a γ-quasi-complete graph and γ.

THEOREM 1 (DIAMETER OF QUASI-COMPLETE GRAPH). *Let G be a γ-quasi-complete graph such that $N = |V(G)| > 1$.*

$$diam(G) \begin{cases} = 1 & \text{if } 1 \geq \gamma > \frac{N-2}{N-1} \\ \leq 2 & \text{if } \frac{N-2}{N-1} \geq \gamma \geq \frac{1}{2} \\ \leq 3\lfloor \frac{N}{\lceil \gamma(N-1) \rceil + 1} \rfloor - 3 & \text{if } \frac{1}{2} > \gamma > \frac{1}{N-1} \text{ and } N \bmod (\lceil \gamma(N-1) \rceil + 1) = 0 \\ \leq 3\lfloor \frac{N}{\lceil \gamma(N-1) \rceil + 1} \rfloor - 2 & \text{if } \frac{1}{2} > \gamma > \frac{1}{N-1} \text{ and } N \bmod (\lceil \gamma(N-1) \rceil + 1) = 1 \\ \leq 3\lfloor \frac{N}{\lceil \gamma(N-1) \rceil + 1} \rfloor - 1 & \text{if } \frac{1}{2} > \gamma > \frac{1}{N-1} \text{ and } N \bmod (\lceil \gamma(N-1) \rceil + 1) \geq 2 \\ \leq N - 1 & \text{if } \frac{1}{N-1} \geq \gamma > 0. \end{cases}$$

PROOF. When $1 \geq \gamma > \frac{N-2}{N-1}$, the degree of each vertex is greater than $(N-2)$, and hence must be $(N-1)$. Thus, G is a complete graph and $diam(G) = 1$.

When $\frac{N-1}{N-2} \geq \gamma \geq \frac{1}{2}$, for any vertices $u, v \in V(G)$, $|\{u,v\} \cup N(u) \cup N(v)| \geq N$. Thus, $diam(G) \leq 2$.

When $\frac{1}{2} > \gamma > \frac{1}{N-1}$, the situations are complicated. Consider vertices $u, v \in V(G)$ such that the shortest path between u and v has length $l = diam(G)$. $V(G)$ can be partitioned into $(l+1)$ exclusive groups $S_1, \ldots, S_{l+1}$: a vertex $w \in S_i$ $(1 \leq i \leq (l+1))$ if and only if $d(u,w) = (i-1)$. Trivially, $d(u,u) = 0$.

Clearly, $\cup_{1 \leq i \leq (l+1)} S_i = V(G)$, otherwise, $diam(G) > l$. A critical fact is that, for any vertex $w \in S_i$ $(1 \leq i \leq (l+1))$, $N(w) \subseteq S_{i-1} \cup S_i \cup S_{i+1}$. That means

$$|S_{i-1} \cup S_i \cup S_{i+1}| = |S_{i-1}| + |S_i| + |S_{i+1}| \geq deg(w) + 1 \geq \gamma(N-1) + 1.$$

Moreover, we have $|S_1| = 1$, $|S_2| = \gamma(N-1)$, and

$$|S_l| + |S_{l+1}| \geq \gamma(N-1) + 1.$$

Otherwise, there exists at least one vertex v such that $deg(v) < \gamma(N-1)$. The inequations in the theorem follow with the above inequations.

When $\frac{1}{N-1} \geq \gamma > 0$, some vertices may have degree 1. Since G is a γ-quasi-complete graph, G must be connected. Therefore, in the worst case, the graph can be a path, and thus $diam(G) \leq (N-1)$.

The bounds can be shown realizable. For example, Figure 5 shows three cases that the bounds are realized for $\frac{1}{2} > \gamma \geq \frac{2}{n-1}$. □

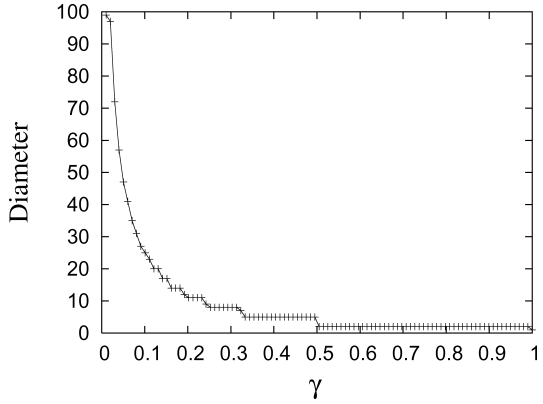


Fig. 6. The upper bound of the diameter of G decreases monotonically when the value of γ increases ($|V(G)| = 100$).

Theorem 1 gives a tight upper bound on the diameter of a quasi-complete graph G. From the formula, we can see that the upper bound is related to two variables, γ and N. The following three corollaries further disclose the relationship between the upper bound, γ, and N.

The first corollary follows with Theorem 1 immediately.

COROLLARY 1. *The realizable upper bound of diameter given in Theorem 1 is monotonically decreasing with respect to γ. That is, for $0 < \gamma_1 < \gamma_2 \leq 1$ and any γ_2-quasi-complete graph G_2 of N vertices, there exists a γ_1-quasi-complete graph G_1 of N vertices such that $diam(G_1) \geq diam(G_2)$.*

Intuitively, Corollary 1 says that the higher the γ value, the more compact the graph. Figure 6 shows the trend of diameter on γ, where the number of vertices is set to 100. We can observe the following.

—When γ is greater than or equal to 0.5, the γ-quasi-complete graph is compact, that is, the diameter is very small, no more than 2.

—When γ is small, the upper bound of the diameter of the quasi-complete graph is approximately in portion to $\frac{1}{\gamma}$, which is intuitive.

—When γ is small, the quasi-complete graph can be a series of small clusters. When $\gamma = \frac{1}{N-1}$, in the worst case, the quasi-complete graph can be a path of N vertices.

Based on the above analysis, a user may often be interested in quasi-complete graphs with a reasonably large γ value, such as $\gamma \approx 0.5$ or larger. In such cases, the diameter is bounded by a small integer.

COROLLARY 2. *The realizable upper bound of diameter given by Theorem 1 is monotonically increasing with respect to the number of vertices when $\gamma \geq 0.5$. That is, for any γ-quasi-complete graph G where $\gamma \geq 0.5$, there exists a γ-quasi-complete graph G' such that $|V(G')| > |V(G)|$ and $diam(G') \geq diam(G)$.*

PROOF. Consider $1 < N_1 \leq N_2$. Let d_1 and d_2 be the tight upper bound of γ-quasi-complete graphs of N_1 and N_2 vertices, respectively. Clearly, $\frac{N_1-2}{N_1-1} \leq \frac{N_2-2}{N_2-1}$. Then, γ must be in one of the following three cases.

—When $\frac{N_2-2}{N_2-1} < \gamma \leq 1$, $d_1 = d_2 = 1$;

—When $\frac{N_1-2}{N_1-1} < \gamma \leq \frac{N_2-2}{N_2-1}$, $d_1 = 1$ and $d_2 = 2$;

—When $\frac{1}{2} \leq \gamma \leq \frac{N_1-2}{N_1-1}$, $d_1 = d_2 = 2$.

In each of the three cases, $d_1 \leq d_2$ holds as long as $N_1 \leq N_2$. □

Corollary 2 indicates that when $\frac{1}{2} \leq \gamma \leq 1$, the upper bound of the diameter of a γ-quasi-complete graph does not decrease if we add more vertices to the graph. However, when $0 < \gamma < \frac{1}{2}$, this property may not hold. For example, according to Theorem 1, the realizable upper-bound of the diameter of 0.1-quasi-complete graphs of 31 vertices is 20, and the realizable upper-bound of the diameter of 0.1-quasi-complete graphs of 32 vertices is 17, which is smaller.

Intuitively, given a fixed γ, the diameter of a quasi-complete graph should increase when we add more vertices into the graph. However, when the number of vertices N increases, $\gamma(N-1)$ also increases. This means each vertex is required to be connected to more vertices. Consequently, the graph may become more compact, that is, the diameter of the graph may decrease.

Theorem 1 can be applied to induced subgraphs of γ-quasi-cliques.

COROLLARY 3. *In a graph G, let $S \subseteq V(G)$. If $C \subseteq S$ is a γ-quasi-clique of G, then*

$$diam(G(C)) \leq \begin{cases} u_{\gamma,|S|} & \text{if } \frac{1}{2} \leq \gamma \leq 1 \\ \max_{1<m\leq|S|}\{u_{\gamma,m}\} & \text{if } 0 < \gamma < \frac{1}{2} \end{cases}$$

where $u_{\gamma,m}$ is the realizable upper bound of diameter of γ-quasi-complete graphs of m vertices determined by Theorem 1.

PROOF. When $\frac{1}{2} \leq \gamma \leq 1$, from Corollary 2, the upper bound of the diameter is monotonic with respect to the number of vertices. Since $|C| \leq |S|$, we have

$$diam(G(C)) \leq u_{\gamma,|C|} \leq u_{\gamma,|S|}.$$

When $0 < \gamma < \frac{1}{2}$, the upper bound of diameter is not monotonic. Thus, we have

$$diam(G(C)) \leq \max_{1<m\leq|S|}\{u_{\gamma,m}\}. \qquad \square$$

3.2 Frequent Cross-Graph Quasi-Cliques

Intuitively, a frequent cross-graph quasi-clique is a maximal set of vertices whose induced subgraphs are frequent quasi-complete subgraphs in the given graphs.

Definition 2 (*Frequent Cross-Graph Quasi-Clique*). Let U be a set of vertices and $G_1, \ldots, G_n$ be n graphs such that $V(G_i) = U$ $(1 \leq i \leq n)$. For parameters $\gamma_1, \ldots, \gamma_n$ $(0 < \gamma_i \leq 1)$, a subset of vertices $S \subseteq U$ is *supported* by graph G_i if $G_i(S)$ is a γ_i-quasi-complete graph. Given a minimum support threshold *min_sup* $(0 < min\_sup \leq 1)$, S is called a *frequent cross-graph quasi-clique*

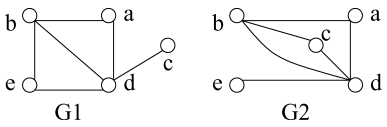


Fig. 7. A frequent cross-graph quasi-clique may not be a quasi-clique in an individual graph due to the maximality requirement.

(fCGQC for short) if it is supported by at least $(n \cdot min\_sup)$ graphs and there is no proper superset of S has the property.

When $min\_sup = 1$, the induced subgraph $G_i(S)$ $(1 \leq i \leq n)$ of a frequent cross-graph quasi-clique S is a γ_i-quasi-complete subgraph for every given graph G_i. In this case, S is called a *cross-all-graphs quasi-clique* (*CAGQC* for short). Clearly, in Definition 2, when $n = 1$ and $min\_sup = 1$, a frequent cross-graph quasi-clique is simply a quasi-clique in a single graph. Therefore, the concept of frequent cross-graph quasi-clique is a generalization of quasi-clique crossing multiple graphs. However, when $n > 1$, due to the maximality requirement, a frequent cross-graph quasi-clique may not be a quasi-clique in any graph even when $min\_sup = 1$.

Example 5 (*Maximality*). Consider graphs G_1 and G_2 in Figure 7. Suppose $\gamma_1 = \gamma_2 = 0.5$ and $min\_sup = 1$. Then, $S = \{a, b, d\}$ is a frequent cross-graph quasi-clique. However, S is not a 0.5-quasi-clique in either G_1 or G_2, since S is a proper subset of $\{a, b, d, e\}$ which is a 0.5-quasi-clique in G_1, and is also a proper subset of $\{a, b, c, d\}$ which is a 0.5-quasi-clique in G_2.

A frequent cross-graph quasi-clique can be insignificant in data analysis if it contains a very small number of vertices. For example, a single vertex itself is a (trivial) quasi-complete graph for any γ. To avoid such triviality, a user may specify a minimum number of vertices in the frequent cross-graph quasi-cliques. Only cross-graph quasi-cliques large enough should be returned.

Problem Definition (Mining Frequent Cross-Graph Quasi-Cliques). For a given set of graphs $G_1, \ldots, G_n$ on a set of vertices U (that is, $V(G_1) = \cdots = V(G_n) = U$), parameters $\gamma_1, \ldots, \gamma_n$ $(0 < \gamma_i \leq 1)$, a minimum support threshold min_sup, and a minimum size threshold min_size, the problem of *mining frequent cross-graph quasi-cliques* is to find the complete set of frequent cross-graph quasi-cliques that each has at least min_size vertices.

In some cases, such as the joint mining of gene expression data and protein interaction data (Example 1), the sets of vertices in different graphs are different, and there exist mappings between sets of vertices in different graphs. Our basic model of mining frequent cross-graph quasi-cliques can be extended to handle such cases.

Definition 3 (*Frequent Cross-Qraph Quasi-Clique with Mapping*). Let U be a set of objets, $G_1, \ldots, G_n$ be n graphs, and $f_1, \ldots, f_n$ be n functions such that f_i $(1 \leq i \leq n)$ is from U to $V(G_i)$. For parameters $\gamma_1, \ldots, \gamma_n$ $(0 < \gamma_i \leq 1)$ and a minimum support threshold min_sup $(0 < min\_sup \leq 1)$, a subset of objects $S \subseteq U$ is called a *frequent cross-graph quasi-clique with mapping* (fCGQC(M)

for short) if there exist at least $k = \lceil n \cdot min\_sup \rceil$ graphs $G_{j_1}, \ldots, G_{j_k}$ such that $G_{j_l}(f_{j_l}(S))$ $(1 \leq l \leq k)$ is a γ_{j_l}-quasi-complete graph, and there is no proper superset of S has this property, where $f_j(S) = \{f_j(u) | u \in S\}$.

Problem Definition (*Mining Frequent Cross-Graph Quasi-Cliques with Mapping*). For a given set of graphs $G_1, \ldots, G_n$ and functions $f_1, \ldots, f_n$ such that $f_i : U \rightarrow V(G_i)$ $(1 \leq i \leq n)$ where U is a set of common vertices, parameters $\gamma_1, \ldots, \gamma_n$ $(0 < \gamma_i \leq 1)$, a minimum support threshold min_sup $(0 < min\_sup \leq 1)$, and a minimum size threshold min_size $(min\_size \geq 1)$, the problem of *mining frequent cross-graph quasi-cliques with mapping* is to find the complete set of fCGQC(M)s that each has at least min_size vertices.

3.3 Complexity Analysis

To understand the complexity of the problem of mining frequent cross-graph quasi-cliques, we first consider the complexity of counting the number of frequent cross-graph quasi-cliques.

THEOREM 4 (COMPLEXITY). *The problem of counting the number of frequent cross-graph quasi-cliques is #P-Complete.*

PROOF. We prove by restriction. That is, we show that the problem of counting the number of frequent cross-graph quasi-cliques contains a #P-Complete problem as a special case. In fact, the problem of counting the number of cliques from one graph is in #P-Complete [Garey and Johnson 1979], and is a special case of the problem of counting the number of cross-graph quasi-cliques where $n = 1$, $\gamma = 1$, and $min\_sup = 1$. □

Since counting the number of frequent cross-graph quasi-cliques is #P-Complete, the problem of mining (that is, enumerating) the complete set of frequent cross-graph quasi-cliques is NP-hard.

4. MINING CROSS-ALL-GRAPHS QUASI-CLIQUES

In this section, we discuss mining cross-all-graphs quasi-cliques, a special case of mining frequent cross-graph quasi-cliques. We present two algorithms. The first algorithm is rudimentary. The second algorithm, *Crochet*, exploits several effective techniques to achieve efficient mining. These techniques will be applied or extended in the next section for mining general frequent cross-graph quasi-cliques.

To make our presentation clear and easy to follow, we assume all given graphs $G_1, \ldots, G_n$ share a common set of vertices, that is, $V(G_1) = \cdots = V(G_n) = U$. In this basic case, the mapping function f_i $(1 \leq i \leq n)$ for each graph G_i is the identity function $f(u) = u$ $(u \in U)$. The case of non-identity functions can be easily extended from the basic case.

4.1 A Rudimentary Algorithm

As illustrated in Examples 4 and 5, a cross-all-graphs quasi-clique may not be a quasi-clique in any graph, and an induced subgraph on a subset of a quasi-clique

Algorithm 1. The rudimentary algorithm.

Input: graphs $G_1, \ldots, G_n$ on vertices U; $\gamma_1, \ldots, \gamma_n$; minimum size threshold min_size;
Output: the complete set of cross-all-graphs quasi-cliques;
Method:

```
    // Phase 1: mining G_1
1:  in G_1, compute the complete set of γ_1-quasi-complete subgraphs H that have at
    least min_size vertices;
    // Phase 2: joint mining
2:  for each γ_1-quasi-complete subgraph H, in the |V(H)| descending order
3:      if H has a proper superset as a cross-graph quasi-clique then continue;
4:      let cross-graph-quasi-clique = true;
5:      for i = 2 to n
6:          if G_i(V(H)) is not a γ_i-quasi-complete graph
            then cross-graph-quasi-clique = false, break;
        end-for
7:      if (cross-graph-quasi-clique == true)
        then output V(H) as a cross-graph quasi-clique;
    end-for
```

may not even be a quasi-complete graph. Hence, we cannot find the quasi-cliques in each graph and take the intersection. Instead, we have to take a joint mining approach.

By definition, a cross-all-graphs quasi-clique must be a quasi-complete graph in every graph by the mapping. Thus, a rudimentary algorithm works in two steps: *mining* G_1 and *jointly mining*, as shown in Algorithm 1.

In the first step, we mine the complete set of γ_1-quasi-complete graphs in G_1 have at least min_size vertices. In other words, in graph G_1, we find the subsets of vertices S such that the subgraph induced on S is a γ_1-quasi-complete graph.

In the step of joint mining, for each subset of vertices S found in the first step, we check whether $G_i(S)$ is still a γ_i-quasi-complete graph for all $2 \leq i \leq n$. Only maximal subsets passing the tests are output as the cross-graph quasi-cliques.

Please note that the rudimentary algorithm is not naïve. It never searches the complete set of quasi-complete subgraphs in all the graphs. Instead, to prune the search space, it exploits the fact that a cross-all-graphs quasi-clique must be a maximal subset having the property .

Although the correctness of the rudimentary algorithm is straightforward, the rudimentary algorithm may not be efficient in mining large graphs due to the following two reasons.

First, the rudimentary algorithm still has to compute the complete set of quasi-complete subgraphs in G_1 that serve as the base in the joint mining in the second step. If G_1 is large and dense (that is, it has many edges), then there can be many quasi-complete subgraphs. Computing all of them can be expensive. Can we compute as few quasi-complete subgraphs in G_1 as possible?

Second, the rudimentary algorithm mines the whole graphs. If the graphs are large, it can be costly. A careful check may find that, some vertices and edges in the graphs, such as the vertices with low degrees, cannot be a part

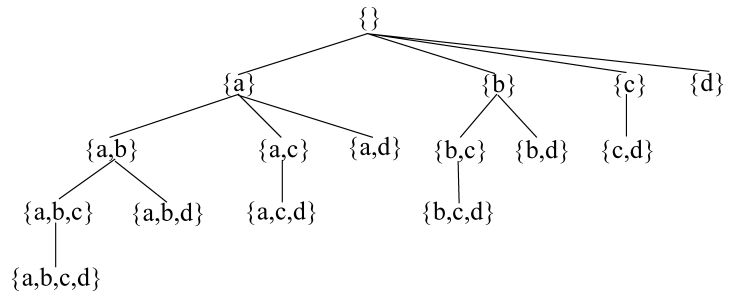


Fig. 8. A set enumeration tree.

of a cross-graph quasi-clique or the induced subgraphs. Such parts should be pruned as early as possible so that the graphs can be reduced. Mining smaller graphs can definitely improve the efficiency. In order to reduce the graphs, we may have to consider multiple graphs at the very beginning. Can we reduce the graphs aggressively to speed up the mining?

In summary, the major drawback of the rudimentary algorithm is that it conducts the joint mining late. Motivated by the above observations, in Section 4.2, we develop algorithm *Crochet*, which exploits "aggressive" joint mining of multiple graphs.

4.2 Algorithm *Crochet*

4.2.1 *General Idea and Framework.* In order to efficiently mine the complete set of cross-all-graphs quasi-cliques, we have to address two issues as follows.

—*Correctness and completeness.* That is, how can we develop a systematic way to find *all* the cross-all-graphs quasi-cliques *without duplicates*?

—*Efficiency.* That is, how can we find rules to prune futile search subspaces (that is, the subspaces do not contain any cross-all-graphs quasi-cliques), and find heuristics to speed up the mining?

To address the correctness and completeness issue, we compute the complete set of cross-all-graphs quasi-cliques by enumerating the subsets of vertices systematically and pruning the unfruitful subsets.

Given a set S of m elements and a total order $\prec$ on S, the complete set of various subsets of S (that is, the power set 2^S) can be enumerated systematically using a *set enumeration tree* [Rymon 1992]. For example, the set enumeration tree for set $\{a, b, c, d\}$ with respect to order $a \prec b \prec c \prec d$ is shown in Figure 8.

In a set enumeration tree of vertices, each node is a subset of vertices. Some nodes are cross-all-graphs quasi-cliques. Algorithm *Crochet* conducts a depth-first search on the set enumeration tree of vertices to find the cross-graph quasi-cliques, which can identify the complete set of answers.

To address the concern on efficiency, *Crochet* employs several techniques, as follows.

Algorithm 2. The framework of algorithm *Crochet*.

Framework of *Crochet*

1: conduct a depth-first search on the set enumeration tree of vertices, in the search of each node, do
2: conduct graph reduction if possible;
3: generate the children of the current node and prune the futile children;
4: dynamically order the surviving children and search them in depth-first manner;

—*Aggressively reducing graphs. Crochet* removes edges and vertices and even combines graphs as long as the correct mining results are retained. By reducing graphs, *Crochet* can work on smaller graphs and thus can mine cross-all-graphs quasi-cliques faster.

—*Carefully choosing the order of search.* In the depth-first search of a set enumeration tree, a node in the tree may have multiple children. The order that we use to search the children may have a substantial effect on the efficiency. *Crochet* uses an effective heuristic to dynamically determine the order of children based on the information from multiple graphs and accomplishes high performance.

—*Sharply pruning futile subtrees.* If a graph has many vertices, the set enumeration tree can be huge. *Crochet* actively detects whether a subtree has the potential to have some cross-all-graphs cross-cliques before it really searches the nodes in the subtree. If a subtree is futile, that is, it does not contain any node of cross-all-graphs quasi-cliques, then the subtree can be pruned early.

The framework of *Crochet* is shown in Algorithm 2. In the following sections, we will present the technical details.

4.2.2 *Reducing Graphs*

4.2.2.1. Reducing Vertices. Some vertices in the graphs can be removed if their degrees are too small or they are not connected to a sufficient number of other vertices under the current search to form a quasi-complete graph.

LEMMA 4.1 (REDUCING VERTICES). *Given a set of graphs $G_1, \ldots, G_n$ on vertices U, a vertex $u \in U$ can pruned if there exists any graph G_i ($1 \leq i \leq n$) such that*

$$deg_i(u) < \gamma_i \cdot (min\_size - 1)$$

or

$$\left|N_i^{k_i}(u)\right| < (min\_size - 1),$$

where $deg_i(u)$ is the degree of u in G_i, k_i is the upper bound of diameter determined by Corollary 3 (let $S = U$), and $N_i^{k_i}$ is the set of neighbors of u within k_i steps in graph G_i. Moreover, the edges having u as an endpoint can be removed from every graph G_i with all cross-all-graphs quasi-cliques retained.

PROOF. If $deg_i(u) < \gamma_i \cdot (min\_size - 1)$, u fails the requirement on the minimum degree in a quasi-complete graph in G_i. If $|N_i^{k_i}(u)| < (min\_size - 1)$, u

cannot be a vertex in any subgraph which has a diameter k_i and also has at least *min_size* vertices. Hence, in either of the two cases, u cannot be a vertex in any quasi-complete subgraph in G_i. That means u cannot be a vertex in any cross-all-graphs quasi-clique in graphs $G_1, \ldots, G_n$. Thus, u as well as the edges having u as an endpoint can be removed from each individual graph G_i $(1 \leq i \leq n)$ and the mining results will not be affected. □

Lemma 4.1 can be applied iteratively to reduce the graphs, until no vertex and edge can be removed. The checking of degrees is simple. However, dynamically maintaining $N_i^{k_i}(u)$ for every vertex u in each graph G_i can be costly if k is large. Fortunately, as shown in Theorem 1, the upper bound of the diameter of a γ-quasi-complete graph is pretty small if γ is not too small. Moreover, as discussed before, a user is often interested in only the cross-graph quasi-cliques with respect to reasonably large γ values, for example, $\gamma \approx 0.5$ or larger. In our experiments, when $\gamma \geq 0.5$, Lemma 4.1 often improves the efficiency by a factor of at least 20%.

4.2.2.2. Reducing Edges. Among graphs $G_1, \ldots, G_n$, if there is a $\gamma_i = 1$ $(1 \leq i \leq n)$, then we can remove some edges in the other graph according to the distribution of edges in G_i.

LEMMA 4.2 (REDUCING EDGES). *When $\gamma_i = 1$, any edge $(u, v) \in E(G_j)$ $(1 \leq j \neq i \leq n)$ can be removed if $(u, v) \notin E(G_i)$, and all cross-all-graphs quasi-cliques from $G_1, \ldots, G_n$ remain intact.*

PROOF. Since $\gamma_i = 1$, if $(u, v) \notin E(G_i)$, u and v cannot be in a complete subgraph in G_i, and thus cannot be in any cross-all-graphs quasi-clique in $G_1, \ldots, G_n$. Removing (u, v) from $E(G_j)$ $(1 \leq j \neq i \leq n)$ will not affect any cross-all-graphs quasi-cliques. □

We can apply Lemma 4.2 to reduce the graphs by scanning the edges of the graphs only once, if their edges are sorted consistently.

4.2.2.3. Combining Graphs. In the case $\gamma_{i_1} = \cdots = \gamma_{i_k} = 1$ $(1 \leq i_1 < \cdots < i_k \leq n)$, we can combine the k graphs G_{i_1} to G_{i_k} into one graph G' as follows. The vertices in G' remain the set of U; (u, v) is an edge in G' if and only if (u, v) is an edge in all graphs G_{i_1} to G_{i_k}, that is, $E(G') = E(G_{i_1}) \cap \ldots \cap E(G_{i_k})$. Then, the problem of mining cross-all-graphs quasi-cliques from the n graphs can be reduced to mining cross-all-graphs quasi-cliques in the combined graph G' and the remaining graphs.

LEMMA 4.3 (COMBINING GRAPHS). *When $\gamma_{i_1} = \cdots = \gamma_{i_k} = 1$ $(1 \leq i_1 < \cdots < i_k \leq n)$, let G' be a combined graph such that $V(G') = U$ and $E(G') = E(G_{i_1}) \cap \ldots \cap E(G_{i_k})$. A set of vertices S is a cross-all-graphs quasi-clique in $G_1, \ldots, G_n$ if and only if S is a cross-all-graphs quasi-clique in G' and the graphs G_i $(\gamma_i \neq 1)$.*

PROOF. We can apply Lemma 4.2 on each graph G_{i_j} $(1 \leq j \leq k)$ such that $\gamma_{i_j} = 1$. After the pruning, each graph G_{i_j} is identical to the combined graph G'. Therefore, S is supported by G_{i_j} if and only if it is supported by G'.

Hence, S is a cross-all-graphs quasi-clique in $G_1, \ldots, G_n$ is equivalent to S is a cross-all-graphs quasi-clique in G' and the graphs G_i ($\gamma_i \neq 1$). □

4.2.3 *Searching and Pruning.* As discussed in Section 4.2.1, *Crochet* conducts a depth-first search on a set enumeration tree of vertices. If a graph has many vertices, the set enumeration tree can be huge. Therefore, one critical part of *Crochet* is to prune the futile subtrees as early as possible.

Basically, three issues have to be addressed.

(1) At a node X in the set enumeration tree, what are the vertices $u \in (U - X)$ that should be used to extend X to its children and may lead to cross-all-graphs quasi-cliques?
(2) In what situation is a subtree rooted at the current node in the set enumeration tree futile (that is, there is no cross-all-graphs quasi-clique in the subtree) and thus can be pruned?
(3) A node in the set enumeration tree may have multiple children. In which order should the children be searched so that the efficiency is likely high?

We answer the above three questions one by one.

4.2.3.1. Generating Children. Let us consider a node $X \subseteq U$ in the set enumeration tree such that $X \neq \emptyset$. The following lemma indicates the set of vertices which can be used to generate the children of X that may lead to some cross-all-graphs quasi-cliques.

LEMMA 4.4 (CANDIDATE VERTICES). *Let $X \neq \emptyset$ be a subset of vertices. If $C \supset X$ is a cross-all-graphs quasi-clique, then for every vertex $u \in (C - X)$,*

$$u \in \bigcap_{v \in X, 1 \leq i \leq n} N_{G_i}^{k_i}(v),$$

where k_i is the upper bound of diameter of complete γ_i-quasi-complete subgraph in G_i given by Corollary 3 (let $S = U$).

PROOF. Following Corollary 3, for any $u \in (C - X)$ and $v \in X$, $d(u, v)$ in G_i is bounded by the upper bound of the diameter. Thus, we have the lemma. □

For a node X in the set enumeration tree, Lemma 4.4 gives the initial set of *candidate vertices*. Moreover, as required by the set enumeration tree, only the vertices behind the last vertex in X in the order of vertices should be taken.

4.2.3.2. Pruning Futile Children and Subtrees. The initial set of candidate vertices can be reduced further. The central idea is as follows. Let Y be the initial set of candidate vertices given by Lemma 4.4. If there is no cross-all-graphs quasi-clique in $G_1(X \cup Y), \ldots, G_n(X \cup Y)$, then X cannot be in any cross-all-graphs quasi-cliques. In other words, the vertices in Y should not be used to generate children of X since they are futile.

LEMMA 4.5 (PROJECTION). *Let $X \subset U$ be a node in the set enumeration tree and Y be the initial set of candidate vertices given by Lemma 4.4. For any C such that $X \subset C \subseteq (X \cup Y)$, C is a cross-all-graphs quasi-clique in graphs $G_1, \ldots, G_n$ if and only if C is a cross-all-graphs quasi-clique in $G_1(X \cup Y), \ldots, G_n(X \cup Y)$.*

PROOF. The necessity is straightforward. We show the sufficiency by contradiction.

Suppose C is a cross-all-graphs quasi-clique in $G_1(X \cup Y), \ldots, G_n(X \cup Y)$, but not in $G_1, \ldots, G_n$. Thus, there must be a $C' \supset C$ such that C' is a cross-all-graphs quasi-clique in $G_1, \ldots, G_n$. However, according to the construction of X and Y, $X \subset C' \subseteq (X \cup Y)$. Thus, C' must be a cross-all-graphs quasi-clique in $G_1(X \cup Y), \ldots, G_n(X \cup Y)$, which leads to a contradiction. □

$G_i(X \cup Y)$ is called the *projection* of G_i on X. Based on Lemma 4.5, we can recursively apply the vertex reduction (Lemma 4.1) to prune the projections. We denote the set of vertices in the projections after the pruning by $P(X)$. Clearly, in the set enumeration tree, only the children of X in the form of $X \cup \{u\}$ $u \in (P(X) - X)$ should be considered.

Moreover, in some situations, the whole subtree rooted at X cannot contain any cross-all-graphs quasi-cliques. The following lemma identifies three cases.

LEMMA 4.6 (PRUNING RULES). *Let X be a node in the set enumeration tree. The subtree rooted at X does not contain any cross-all-graphs quasi-clique if (1) $|P(X)| < min\_size$; (2) $X \not\subseteq P(X)$; or (3) $P(X) \subset C$ where C is a cross-all-graphs quasi-clique already found.*

PROOF. The first pruning rule follows the requirement on the minimum number of vertices in cross-all-graphs quasi-cliques. That is, if the sets of vertices in the subtree are too small, there is no hope to find significant cross-all-graphs quasi-clique and thus the subtree can be pruned.

The second rule follows the construction of the set enumeration tree. That is, some vertex in X can be pruned by edge reduction and vertex reduction. Then, there is no hope to get cross-all-graphs quasi-cliques from the subtree which are supersets of X.

The third rule follows the requirement of maximality for cross-all-graphs quasi-clique. In other words, the cross-all-graphs quasi-cliques containing X as a subset should be found in different branches of the set enumeration tree instead of the subtree rooted at X. □

If one of the three conditions specified in Lemma 4.6 happens, the subtree rooted at X can be pruned.

In some cases, we may find that any cross-all-graphs quasi-clique containing the current node X must also contain another subset of vertices S. In such situation, we can directly substitute the current node X with $(X \cup S)$ without enumerating the intermediate nodes between X and $(X \cup S)$ in the set enumeration tree.

LEMMA 4.7 (PARENT EQUIVALENCE SUBSTITUTION). *Let X be a node in a set enumeration tree.*

—*If there exists a graph G_i and an edge $(u, v) \in E(G_i)$ $(1 \leq i \leq n)$ such that $u \in X, v \in (P(X) - X)$, and the degree of u in the induced subgraph $G_i(P(X))$ is $\gamma_i \cdot (\max(|X|, min\_size) - 1)$, then node X can be replaced by $X' = X \cup \{v\}$ and all cross-all-graphs quasi-cliques are retained;*

—*If the induced subgraph of $P(X)$ is a quasi-complete graph for every graph G_i, X can be replaced by $P(X)$ and then reported as a cross-all-graphs quasi-clique.*

PROOF. For the first case, if the subtree of node X contains any cross-all-graphs quasi-clique S, then $|S| \geq max(|X|, min\_size)$. Since $G_i(S)$ must be a γ_i-quasi-complete graph, the degree of any vertex $u \in X$ in $G_i(S)$ must be greater than or equal to $\gamma_i \cdot (max(|X|, min\_size) - 1)$. According to the construction of $P(X)$, we have $S \subseteq P(X)$. Therefore, if the degree of any vertex $u \in X$ in $G_i(P(X))$ is exactly $\gamma_i \cdot (max(|X|, min\_size) - 1)$, then all the neighbors of u in $G_i(P(X))$ must belong to S. In other words, any cross-all-graphs quasi-clique S in the subtree of X must contain all the neighbors of u in $G_i(P(X))$. If a neighbor v of u in $G_i(P(X))$ does not belong to X, we can directly extend X with $X \cup \{v\}$.

For the second case, if the induced subgraph of $P(X)$ is a quasi-complete graph for every graph G_i, and X is not pruned by Lemma 4.6, then $P(X)$ must be a cross-all-graphs quasi-clique. Therefore, we can stop the enumeration and report $P(X)$ as a cross-all-graphs quasi-clique. □

4.2.3.3. Ordering Children and Identifying Cross-Graph Quasi-Cliques. Intuitively, a vertex with a high degree is likely a member of a quasi-complete subgraph. Heuristically, we can use $\theta(v) = \frac{ldeg(v)}{\gamma}$ to measure how well a vertex satisfies the γ-quasi-complete graph requirement, where $ldeg(v)$ is the degree of v in the induced graph on the current node X in the set enumeration tree. The vertices can be sorted in the $\theta(v)$ descending order. However, a vertex may have different θ values in different graphs. An observation is that how well a vertex is connected to the others crossing the graphs is bounded by the minimum $\theta(v)$ value in the graphs. Therefore, we have the following heuristic.

HEURISTIC 1 (ORDERING VERTICES). *Let $\theta_{\min}(v) = \min_{G_1,\ldots,G_n}\{\theta(v)\}$. The children vertices of a node in the set enumeration tree can be extended in the $\theta_{\min}(v)$ descending order.*

The experimental results (see Section 4.2 of [Pei 2005]) showed that the heuristic accomplishes good performance in practice. On the other hand, since it is a heuristic, there is no theoretical guarantee that the rule gives optimal efficiency.

After searching the subtree rooted at X, we can determine that X is a cross-all-graphs quasi-clique if the induced graph of X is a quasi-complete graph for every graph G_i $(1 \leq i \leq n)$ and there is no cross-graph quasi-clique in the subtree of X.

LEMMA 4.8 (DETERMINATION OF CROSS-ALL-QRAPHS QUASI-CLIQUE). *Let X be a node in the set enumeration tree. X is a cross-all-graphs quasi-clique if and only if $G_i(X)$ is a γ_i-quasi-complete graph for $1 \leq i \leq n$ and there exists no cross-graph quasi-clique C such that $X \subset C \subseteq P(X)$.*

4.2.4 *The Algorithm and an Example.* Algorithm 3 shows the algorithm of *Crochet*. The correctness of the algorithm is shown in the above discussion.

Algorithm 3. Algorithm *Crochet*: mining cross-all-graphs quasi-cliques.

Input: graphs $G_1, \ldots, G_n$ on vertices U; $\gamma_1, \ldots, \gamma_n$; minimum size threshold *min_size*;
Output: the complete set of cross-all-graphs quasi-cliques;
Method:

apply edge reduction (Lemma 4.2), vertex reduction (Lemma 4.1), and combine graphs (Lemma 4.3) if possible; // graph reduction
`if` all the graphs can be combined
`then` compute the complete set of cliques in the combined graph; exit;
let $G_1, \ldots, G_n$ denote the reduced graphs;
// depth-first search
`for each` vertex $v \in U$ in $\theta_{\min}(v)$ descending order // Heuristic 1
  let $X = \{x\}$; `call` *recursive-mine*$(X, G_1, \ldots, G_n)$;
`end for`

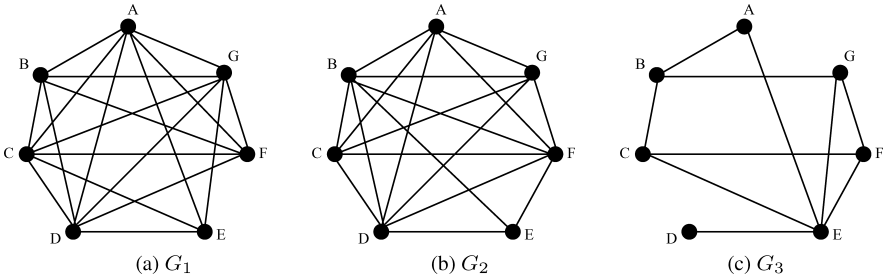
Function *recursive-mine*$(X, G_1, \ldots, G_n)$
compute $P(X)$ according to Lemma 4.4; // graph reduction
let $G_i = G_i(P(X))$ $(1 \leq i \leq n)$; // Lemma 4.5
apply vertex reduction (Lemma 4.1);
let $G_1, \ldots, G_n$ denote the reduced graphs;
`if` at least one condition in Lemma 4.6 holds `then return(0);`
// depth-first search
`if` the induced subgraph $G_i(P(X))$ is γ_i-quasi-complete for every graph G_i
`then` output $P(X)$ and `return`(1); // Lemma 4.7(2)
`if` X can be substituted with X' according to Lemma 4.7(1)
`then` let $X = X'$ and `goto` 7:
let *unsubsumed*= 1;
`for each` vertex $v \in P(X) - X$, in $\theta_{\min}(v)$ descending order // Heuristic 1
  `call` *recursive-mine*$(X \cup \{v\}, G_1, \ldots, G_n)$;
  `if` the returned value is 1 `then` *unsubsumed*= 0;
`end for`
`if` *unsubsumed* is 0 `then return`(1)
`else if` $G_i(X)$ is a γ_i-quasi-complete graph // Lemma 4.8
  `then` output X and `return`(1);
  `else return`(0);

THEOREM 3 (CORRECTNESS OF ALGORITHM *Crochet*). *Algorithm* Crochet *computes the complete set of cross-graph quasi-cliques without duplicate.*

We use the following example to demonstrate the execution of algorithm *Crochet*.

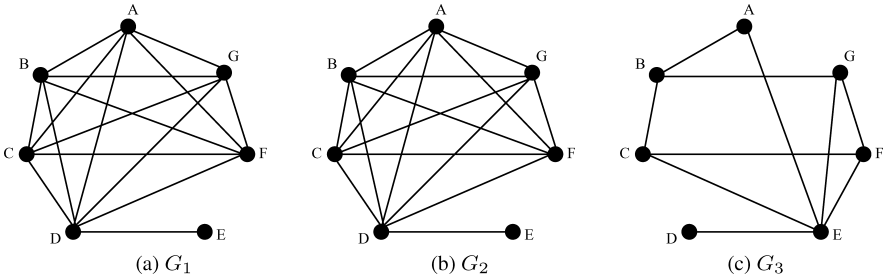
Example 6 (*Algorithm* Crochet). Figure 9 shows three graphs as the input of our example. Suppose $\gamma_1 = \gamma_2 = 1$, $\gamma_3 = 0.5$, and $min\_size = 3$.

The algorithm first iteratively applies edge reduction, vertex reduction, and graph combination. In the step of edge reduction, edges (A, E), (C, E), and (E, G) can be removed from G_1 according to Lemma 4.2. Similarly, edges (B, E) and (E, F) can be removed from G_2. Figure 10 shows the graphs after edge reduction.



(a) G_1 (b) G_2 (c) G_3

Fig. 9. The three graphs as the input of an example of *Crochet*.



(a) G_1 (b) G_2 (c) G_3

Fig. 10. The reduced graphs of Figure 9 after edge reduction.

In the next step, the *Crochet* algorithm applies vertex reduction to the graphs. According to Lemma 4.1, vertices D and E can be pruned. Moreover, G_1 and G_2 can be combined into one graph G_{12} according to Lemma 4.3. Figure 11 shows the final output of the graph reduction process.

As discussed in Section 4.2.3, *Crochet* conducts a depth-first search on a set enumeration tree of vertices and prunes the futile subtrees as early as possible. Figure 12 shows the set enumeration tree searched by *Crochet* for graphs G_{12} and G_3 in Figure 11. For each node on the tree, the vertices of the node and the candidates are separated by a vertical bar '|'. A comment is given under a node if a particular pruning rule is applicable at the node, or the node is reported as a cross-graph quasi-clique. Please note that with the pruning rules, *Crochet* only needs to search a small part of the whole search space, that is, the complete set enumeration tree.

After the search process of the set enumeration tree, *Crochet* terminates and outputs three cross-graph quasi-cliques: $\{A, B, C\}$, $\{A, B, G\}$, and $\{B, C, F, G\}$.

5. MINING FREQUENT CROSS-GRAPH QUASI-CLIQUES

In this section, we extend *Crochet* to *Crochet*$^+$ for the general case of mining frequent cross-graph quasi-cliques, that is, when $0 < min\_sup \leq 1$. Compared with mining cross-all-graphs quasi-cliques, mining the general case is more challenging because we have to handle not only combinations of vertices but also combinations of graphs. To be specific, to test whether a subset of vertices S is a cross-all-graphs quasi-clique, we only need to test whether it is supported by every given graph—if one graph says no, S is not a cross-all-graphs quasi-clique.

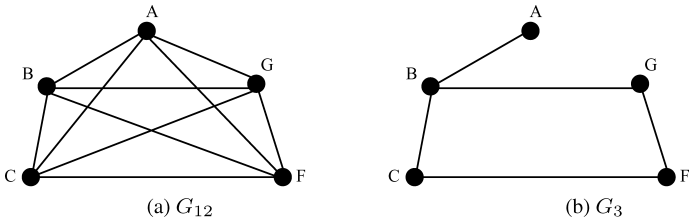


Fig. 11. The final reduced graphs of Figure 9 after the graph reduction process.

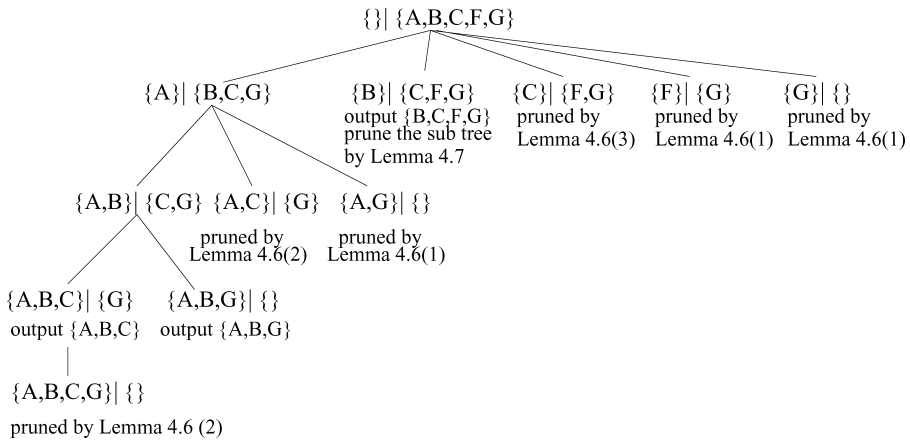


Fig. 12. The search and pruning process of *Crochet* for the reduced input graphs of Figure 11.

However, to test whether S is a general frequent cross-graph quasi-clique, we have to test whether it is supported by a subset of at least $(n \cdot min\_sup)$ graphs. How to conduct the tests efficiently is far from trivial.

The *Crochet*$^+$ algorithm adopts the framework of the *Crochet* algorithm (Algorithm 2). That is, it enumerates the subsets of vertices and prunes futile branches as much as possible. However, the technical details of *Crochet*$^+$ are substantially different from those of *Crochet*.

First, in *Crochet*, a vertex u can be pruned from the subtree T of node X if there exists any graph G_i such that no superset of $X \cup \{u\}$ in T is supported by G_i. However, a frequent cross-graph quasi-clique is not necessarily supported by every graph. Therefore, most graph reduction and pruning techniques, such as Lemmas 4.1, 4.2, 4.3, and 4.6, cannot be directly applied to *Crochet*$^+$. Moreover, we need to reconsider the issue of determining candidate vertices.

Second, unlike cross-all-graphs quasi-cliques, which are invariably supported by the whole set of given graphs, different frequent cross-graph quasi-cliques are usually supported by various combinations of graphs. Therefore, for a node X on the set enumeration tree, we need to record which graphs are likely to support the nodes in the subtree of X. These graphs are called *candidate graphs* of node X.

In the rest of this section, we will discuss how to determine candidate vertices and candidate graphs in the context of searching frequent cross-graph

quasi-cliques. Moreover, new pruning rules and searching heuristics will be explored.

5.1 Determining Candidate Vertices

In *Crochet*, the initial candidate vertices $candV$ for a node X on the set enumeration tree is determined by Lemma 4.4. The basic idea of Lemma 4.4 is to intersect the candidate vertices $cand\,V_i$ with respect to each individual graph G_i, where $cand\,V_i$ can be computed from $\bigcap_{v\in X} N^{k_i}_{G_i}(v)$ and k_i can be derived from Corollary 3. In other words, if a vertex u does not belong to the candidate vertex list $cand\,V_i$ with respect to any individual graph G_i, $X \cup \{u\}$ cannot lead to any cross-all-graphs quasi-clique, and thus u can be pruned from $cand\,V$. However, a frequent cross-graph quasi-clique is not necessarily supported by every graph. Therefore, a vertex u can be pruned from $candV$ only if it appears in less than $(n \cdot min\_sup)$ candidate vertex lists $candV_i$ with respect to the individual graphs G_i.

Based on the above analysis, we can use an idea similar to Lemma 4.4 to determine the initial candidate vertex list $cand\,V_i$ with respect to each individual graph G_i. We can then apply Lemma 4.1 to the induced subgraph $G_i(X \cup candV_i)$ to recursively reduce the size of the candidate list. Let $P_i(X)$ be the subset of vertices after reduction. If $P_i(X)$ satisfies any condition specified in Lemma 4.6, then $candV_i = null$, otherwise, $candV_i = P_i(X) - X$.

Example 7 (*Pruning Candidate Vertices*). Suppose we have six graphs $G_1, \ldots, G_6$ and nine vertices $v_1, \ldots, v_9$. At node $\{v_1, v_2\}$, the candidate vertex list $cand\,V_i$ with respect to each individual graph G_i is listed in Table I(a). To facilitate the pruning of unpromising vertices, for each vertex u, we use an *inverted list*, denoted by $inv(u)$, to record the graphs G_i with respect to which u appears in $cand\,V_i$. Table I (b) shows the inverted lists of the vertices. Suppose $min\_sup = 0.5$, v_3, v_7, and v_9 can be pruned since the size of their inverted lists is smaller than $n \cdot min\_sup = 6 \times 0.5 = 3$.

After pruning v_3, v_7, and v_9, the candidate vertex lists with respect to individual graphs are shown in Table I(c). The whole candidate list $cand\,V$ for node X is then the union of all the individual candidate lists. In this example,

$$candV = \bigcup_{1\le i\le 6} candV_i = \{v_4, v_5, v_6, v_8\}.$$

5.2 Determining Candidate Graphs

In Table I(a), $candV_4$ with respect to graph G_4 is $null$. As described in the previous subsection, this means that the subtree of node $X = \{v_1, v_2\}$ does not contain any node X' whose induced subgraph is a quasi-complete subgraph in G_4. Therefore, when searching the frequent cross-graph quasi-cliques in the subtree of X, we do not need to consider G_4 any more. Based on this idea, for each node X on the set enumeration tree, we maintain a list of *candidate graphs*, denoted by $cand\,G$, to record which graphs are possible to support the frequent cross-graph quasi-cliques in the subtree of X. In the running example, the initial candidate graph list $cand\,G$ for node X is $\{G_1, G_2, G_3, G_5, G_6\}$.

Table I. An Example of Determining and Pruning the Candidate Vertices and Candidate Graphs.

Graph	Candidate vertices
G_1	$\{v_4, v_5, v_6, v_9\}$
G_2	$\{v_4, v_5, v_6, v_8, v_9\}$
G_3	$\{v_4, v_7, v_8\}$
G_4	*null*
G_5	$\{v_4, v_5, v_6, v_7, v_8\}$
G_6	$\{v_3, v_5\}$

(a) The initial candidate vertices w.r.t. individual graphs

Vertex	The inverted list
v_3	$\{G_6\}$
v_4	$\{G_1, G_2, G_3, G_5\}$
v_5	$\{G_1, G_2, G_5, G_6\}$
v_6	$\{G_1, G_2, G_5\}$
v_7	$\{G_3, G_5\}$
v_8	$\{G_2, G_3, G_5\}$
v_9	$\{G_1, G_2\}$

(b) The initial inverted lists for candidate vertices

Graph	Candidate vertices
G_1	$\{v_4, v_5, v_6\}$
G_2	$\{v_4, v_5, v_6, v_8\}$
G_3	$\{v_4, v_8\}$
G_4	*null*
G_5	$\{v_4, v_5, v_6, v_8\}$
G_6	$\{v_5\}$

(c) The candidate vertices w.r.t. individual graphs after v_3, v_7, and v_9 are pruned

Vertex	The inverted list
v_4	$\{G_1, G_2, G_5\}$
v_5	$\{G_1, G_2, G_5\}$
v_6	$\{G_1, G_2, G_5\}$
v_8	$\{G_2, G_5\}$

(d) The inverted lists for candidate vertices after G_3, G_4, and G_6 are pruned

Graph	Candidate vertices
G_1	$\{v_4, v_5, v_6\}$
G_2	$\{v_4, v_5, v_6\}$
G_5	$\{v_4, v_5, v_6\}$

(e) The candidate vertices w.r.t. individual graphs after v_8 is pruned

Vertex	The inverted list
v_4	$\{G_1, G_2, G_5\}$
v_5	$\{G_1, G_2, G_5\}$
v_6	$\{G_1, G_2, G_5\}$

(f) The inverted lists for candidate vertices after v_8 is pruned

The initial candidate graph list can be further reduced. For example, after v_3 is pruned from the candidate vertex list *candV*, the size $|X| + |candV_3| = 4 < min\_size$. According to Lemma 4.6, the subtree of X cannot contain any X' whose induced subgraph is quasi-complete in G_3. Therefore, G_3 can be pruned from *candG*. Similarly, the size $|X| + |candV_6|$ is also smaller than min_size, therefore, G_6 is also pruned from *candG*. After G_3 and G_6 are pruned, the inverted list of the vertices should also be updated accordingly: G_3 and G_6 should be removed from the inverted lists as well (see Table I(d)).

The above analysis leads to the following result immediately.

LEMMA 5.1 (CANDIDATE VERTICES AND CANDIDATE GRAPHS). *Given a set of graphs $G_1, \ldots, G_n$, let X be a node on the set enumeration tree. A vertex u can be pruned from the candidate vertex list candV for X if $|inv(u)| < n \cdot min\_sup$, where $inv(u)$ is the inverted list of u. Moreover, a graph G_i $(1 \leq i \leq n)$ can be pruned from the candidate graph list candG for X if $|X| + |candV_i| < min\_size$, where $candV_i$ is the candidate vertex list of X with respect to graph G_i.*

Lemma 5.1 can be applied iteratively. Let us consider the running example in Table I again. After we prune the vertices v_3, v_7, v_9 and graphs G_3, G_4, G_6, the

size of $inv(v_8)$ drops below the threshold $n \cdot min\_sup = 3$, thus v_8 is removed from *candV* and also the candidate vertex lists with respect to graphs G_2 and G_5 (see Table I(e)). Till now, no more vertices or graphs can be pruned by Lemma 5.1. Then we can apply Lemmas 4.1 and 4.4 again to further prune the candidate vertex lists with respect to individual graphs. That is, Lemmas 4.1, 4.4, and 5.1 can be applied repeatedly until none of them can prune more vertices or graphs.

After the candidate vertices *candV* and candidate graphs *candG* for node X become stable, the whole subtree of X can be pruned if any of the following situations happens.

LEMMA 5.2 (PRUNING SUBTREES). *Let X be a node on the set enumeration tree, and candV and candG be the corresponding candidate vertices and graphs of X, respectively. The subtree rooted at X can be pruned if (1) $|X| + |candV| < min_s$; (2) $|candG| < n \cdot min\_sup$; or (3) $X \subset X'$, where X' is a frequent cross-graph quasi-clique already found.*

5.2.1 *Generating Children Nodes and Identifying Frequent Cross-Graph Quasi-Cliques.* Let *candV* be the candidate vertices of X. For each vertex $u \in candV$, we generate a child $X' = X \cup \{u\}$. Clearly, we can set the initial candidate graph list $candG'$ for X' as the inverted list of u. Moreover, the initial candidate vertex list $candV'$ for X consists of the vertices v such that $v \in candV$ and $inv(v) \supseteq candG'$.

Heuristic 1 can still be applied to choose the order to explore the children. The only modification is that $\theta_{min}(v)$ should be considered among the graphs in $inv(v)$, that is, $\theta_{min}(v) = min_{G_i \in inv(v)}\{\theta(v)\}$.

The process to identify whether a node X is a frequent cross-graph quasi-clique is similar to Lemma 4.8. After searching the subtree rooted at X, we can determine that X is a frequent cross-graph quasi-clique if there exist at least $n \cdot min\_sup$ graphs G_i in the candidate graph list *candG* for X such that $G_i(X)$ is a γ_i-quasi-complete graph and there is no frequent cross-graph quasi-clique in the subtree of X.

Algorithm 4 shows the algorithm of *Crochet*$^+$ (the subroutine is shown in Algorithm 5).

Algorithm 4. Algorithm *Crochet*$^+$: mining frequent cross-graph quasi-cliques.

Input: graphs $G_1, \ldots, G_n$ on a set of vertices U; $\gamma_1, \ldots, \gamma_n$;
minimum size threshold *min_size*; minimum support threshold *min_sup*
Output: the complete set of frequent cross-graph quasi-cliques;
Method:
// depth-first search
1: `for each` vertex $v \in U$ in $\theta_{min}(v)$ descending order //see Section 5.2.1
2: `let` $X = \{x\}$, $candV = \{u \mid u \in U$ and u is behind x in the order of vertices$\}$, $candG = \{G_1, \ldots, G_n\}$,
3: `call` *recursive-mine*$(X, candV, candG)$; // see Algorithm 5
`end for`

Algorithm 5. Subroutine *recursive-mine*() in Algorithm *Crochet*$^+$.

Function *recursive-mine*(X, *candV*, *candG*)
1: `for each` graph $G_i \in candG$ `do`
2: compute the initial candidate vertices $candV_i$ as described in Section 5.1;
3: `repeat` // recursively reduce *candV* and *candG*
4: `for each` graph $G_i \in candG$ // reduce $candV_i$ w.r.t. graph G_i
5: `let` $G_i = G_i(X \cup candV_i)$; // graph projection (Lemma 4.5)
6: iteratively reduce the vertices and edges in the projection according to Lemma 4.1 until there is no change;
7: `let` $P_i(X)$ be the set of vertices remaining in the projection;
8: `if` any of the three conditions in Lemma 4.6 is satisfied // prune G_i
9: `then let` $candG- = \{G_i\}$, $candV_i = null$;
10: `else let` $candV_i = P_i(X) - X$, $candV\cup = candV_i$;
`end for`
12: prune *candV* and *candG* using Lemma 5.1;
13: `until` no more vertices or graphs can be pruned from *candV* or *candG*;
14: `if` any condition in Lemma 5.2 is satisfied, `then` return 0;
`let` $unsubsumed = 1$;
15: `for each` vertex $u \in candV$ `do` $inv(u) = \{G_i \mid (1 \le i \le n) \vee (u \in candV_i)\}$;
16: `for each` vertex $u \in candV$ // depth-first search
17: `let` $candG' = inv(u)$, $candV' = \{v \mid inv(v) \supseteq candG'\}$;
18: `call` *recursive-mine*($X \cup \{u\}$, $candV'$, $candG'$);
19: `if` the returned value is 1 `then` $unsubsumed = 0$;
`end for`
20: `if` *unsubsumed* is 0 `then return`(1) `else`
21: `if` there exists at least $n \cdot min\_sup$ graphs G_i in *candG* such that $G_i(X)$ is a γ_i-quasi-complete graph
`then` output X as a cross-all-graphs quasi-clique, `return`(1);
`else return`(0);

6. EXPERIMENTAL RESULTS

We conducted an extensive performance study using both real data sets and synthetic data sets. The algorithms were implemented in Java and the experiments were run on a Sun Ultra 10 work station with a 440MHz CPU and 1G main memory. Since the performance of the rudimentary algorithm (Algorithm 1) and the *Crochet* algorithm (Algorithm 3) has been reported in Pei et al. [2005], here we focus on the results of the *Crochet*$^+$ algorithm.

6.1 The Datasets

We used both a real dataset and several synthetic datasets. We explain the configurations of the datasets in this section.

6.1.1 *The Real Dataset.* We used the gene expression data CDC28 [Cho et al. 1998] and the protein-protein interaction (PPI) data from GRID[1] as the

[1](`http://biodata.mshri.on.ca/grid/servlet/Index`)We did not use the DIP data as in [Pei et al. 2005] because the GRID database records the source of protein-protein interactions and thus we

real dataset. The CDC28 dataset [Cho et al. 1998] records the mRNA transcript levels of the budding yeast *S. cerevisiae* during the cell cycle. It contains the expression values of 6, 096 genes during a 17-point time-series. This dataset is publicly available.[2] The GRID database includes protein-protein interactions in *S. cerevisiae* from three major sources: yeast two hybrid, affinity precipitation, and synthetic lethality. We organized the protein-protein interactions from each source into an individual dataset.

We found 6,196 genes or proteins which appear in at least one data set. Genes and proteins were one-to-one mapped through their ORF (Open Reading Frame) identifications. For the CDC28 data set, we used the Pearson's correlation coefficient as the measure of coherence. A pair of genes were connected with an edge if their coherence is among top 10% of all pairs. As the result, the gene coexpression graph G_E contains 1, 928, 153 edges. For the protein-protein interaction datasets, two proteins were connected with an edge if they interact with each other. After removing the self-interacting protein pairs, the three protein-protein interaction graphs consist of 6,228, 7,244, and 5,329 edges, respectively,

In our experiments, we found the complete set of frequent CGQCs across the gene coexpression graph G_E and the three protein-protein interaction graphs G_{P1} (yeast two hybrid), G_{P2} (affinity precipitation), and G_{P3} (synthetic lethality). Unless particularly specified, we set $\gamma_E = 1$ for G_E, $\gamma_P = 0.5$ for G_{P1}, G_{P2}, G_{P3}. Moreover, we set $min\_size = 5$ and $min\_sup = 50\%$.

6.1.2 *Synthetic Datasets.* We wrote a data generator for synthetic datasets, which generates synthetic data sets as follows. Given the number of graphs n and a set of vertices U, the data generator first creates n graphs $G_1, \ldots, G_n$ such that for each graph G_i $(1 \leq i \leq n)$, $V(G_i) = U$ and $E(G_i) = \emptyset$. Then, given the expected number N_q of frequent cross-graph quasi-cliques and the parameters $\gamma_1, \ldots, \gamma_n$, the data generator randomly generates N_q frequent cross-graph quasi-cliques and embeds each one into a set of $n \cdot min\_sup$ randomly selected graphs. The sizes of the frequent cross-graph quasi-cliques are uniformly distributed between $qMin$ and $qMax$ that are specified by users. Finally, given the density value σ_i for graph G_i, the data generator keeps adding randomly generated edges into the graph G_i until the overall density of G_i reaches σ_i. Here, the density of a graph G is defined as

$$density(G) = \frac{|E(G)|}{\frac{(|V(G)|\cdot(|V(G)|-1))}{2}} = \frac{2|E(G)|}{(|V(G)| \cdot (|V(G)|-1))}.$$

In the experiments reported in this section, the default values for the parameters were as follows: $n = 4$, $\gamma_1 = \gamma_2 = 1$ for G_1 and G_2, $\gamma_3 = \gamma_4 = 0.5$ for G_3 and G_4, $density = 0.05$ for all graphs, $min\_size = 5$, $min\_sup = 75\%$, $qMin = 5$ and $qMax = 20$.

---

can construct separate interaction graphs for various sources. In fact, the GRID database contains most interactions in the DIP data.

[2]http://cellcycle-www.stanford.edu.

The experimental results on the real datasets and the synthetic datasets are consistent. To keep our presentation concise, we use the results from the real dataset to illustrate the effectiveness of the mining and the effect of the pruning techniques, and use both the real dataset and the synthetic datasets to examine the efficiency and the scalability.

6.2 Findings in the Real Data

We applied the *Crochet*$^+$ algorithm to the genomic data and obtained 2,667 frequent cross-graph quasi-cliques. Many frequent cross-graph quasi-cliques are very similar to each other. For example, one frequent CGQC S_1 across the gene expression data set CDC28 and the affinity precipitation PPI dataset contains five ORFs: *YGR145W, YCR057C, YGR090W, YHR196W*, and *YJL033W*; another frequent CGQC S_2 across the same two datasets contains *YNL132W, YCR057C, YGR090W, YHR196W*, and *YJL033W*. S_1 and S_2 share four ORFs. The only difference is that S_1 contains *YGR145W*, but S_2 contains *YNL132W*. In fact, all the six ORFs, *YGR145W, YNL132W, YCR057C, YGR090W, YHR196W*, and YJL033W, belong to a large known protein complex. This group of ORFs broke into two frequent CGQCs because the correlation information between *YGR145W* and *YNL132W* was missed in the highly noisy data. To handle this problem, we merged two frequent CGQCs C_1 and C_2 if they are frequent CGQCs across the same graphs and $\frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \geq \eta$. In the experiment, we set $\eta = 0.6$ and obtained 369 merged frequent CGQCs. In the following, we will not distinguish the original frequent CGQCs and the merged ones.

To evaluate the biological meaning of the reported frequent CGQCs, we compared them with the known complexes in MIPS (Munich Information Center for Protein Sequences, http://mips.gsf.de/). Clearly, the better the frequent CGQCs match the known complexes, the more meaningful they are. To measure the "matchingness" between the clusters and known complexes, we first defined the *matching coefficient* between a reported frequent CGQC C_i and a known complex X_j as $MC(C_i, X_j) = \frac{|C_i \cap X_j|}{|C_i|}$. We then matched each cluster C_i with the complex having the highest matching coefficient, that is, $MC(C_i) = max\{MC(C_i, X_j)|X_j \in \text{MIPS}\}$. Among the 369 frequent CGQCs, 252 have a matching coefficient higher than 0.8. Many matched complexes are involved in biological processes such as RNA metabolism, protein synthesis turnover, transcription, DNA maintenance, chromatin structure, and signal transduction. Since these biological processes are closely related to the cell cycle, the identified frequent CGQCs are biologically meaningful.

Figure 13 shows a frequent CGQC which is supported by three out of the four data sets. This pattern exactly matches the STE5-MAPK complex, which is a signal transduction complex [Choi et al. 1994; Lyons et al. 1996; Mewes et al. 2006]. However, when mining individual graphs, we found in the yeast two hybrid graph another pattern which substitutes YDR103w with YLR313c (Figure 14(b)). Interestingly, this pattern is not supported by other graphs (Figures 14(a), (c), and (d)). In fact, this pattern is a false positive since YLR313c does not form a complex with the other members in the

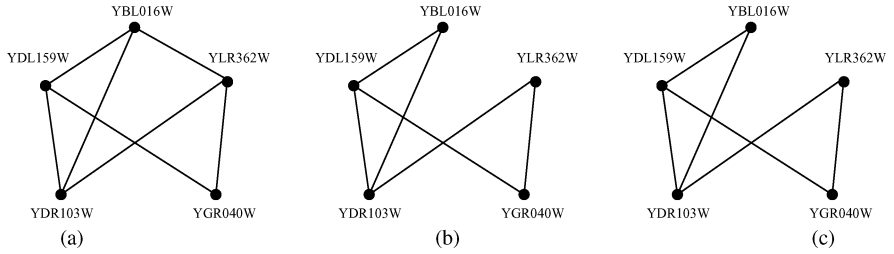


Fig. 13. The STE5-MAPK complex identified from the (a) yeast two hybrid, (b) affinity precipitation PPI data, and (c) synthetic lethality PPI data.

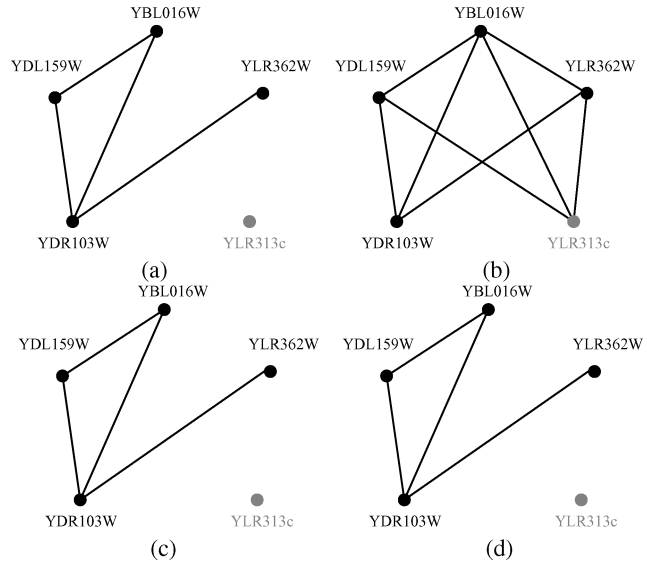


Fig. 14. A pattern which is only supported by yeast two hybrid data but not other data.

pattern [Mewes et al. 2006]. Some interactions in the yeast two hybrid data are possibly errors in the experiment. *Crochet*$^+$ can successfully prune such patterns since it searches across multiple graphs and filters out false positives using the minimum support threshold.

Figure 15 is a subcomplex of a huge complex with 83 ORFs involved in the RNA matabolism. It is not surprising that *Crochet*$^+$ only detected a part of the whole complex. The reason is that the complexes are often formed dynamically during the biological processes. In the cell-cycle process, it is possible that only 15 out of the whole 83 ORFs are expressed. Please note that none of the patterns in Figures 13 and 15 can be detected by the *Crochet* algorithm since they are not supported by all of the four graphs. Therefore, *Crochet*$^+$ is a relaxed version of the *Crochet* algorithm and can find more useful patterns.

6.3 Effect of Parameters

We now study the effect of the mining parameters (that is, min_sup, min_size and γ) in the *Crochet*$^+$ algorithm on the mining effectiveness and efficiency.

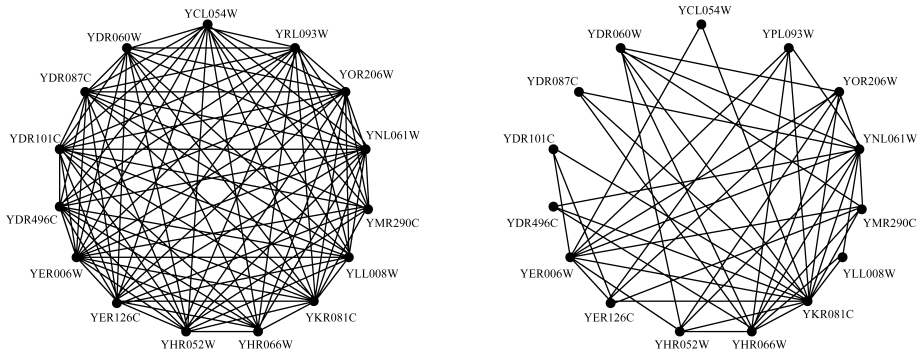


Fig. 15. A subcomplex involved in the RNA matabolism. Identified from (a) the cell-cycle gene expression data and (b) the affinity precipitation PPI data.

Figures 16(a) and (b) show the number of frequent CGQCs and runtime of *Crochet*$^+$ with respect to various *min_sup* values. When the support value increases, a frequent CGQC needs to be supported by more graphs and thus is less likely to be a false positive. In real applications such as mining biological data, since the datasets are often noisy, a relatively high support value, for example, above 50%, would result in more reliable patterns. From Figures 16(a) and (b), we can see when the support value increases, the number of frequent CGQCs decreases dramatically and the runtime also decreases accordingly. The two curves follow the same trend.

The effect of the minimum size threshold *min_size* is shown in Figures 16(c) and (d). As *min_size* grows, the number of frequent CGQCs decreases and the runtime also decrease sharply. This is because many small frequent CGQCs, for example, those of size 3 or 4, were formed by noise (false positives) in the data sets. When the requirement on the size became more stringent, those small frequent CGQCs were filtered out.

We also tested the effect of γ_P and γ_E, as shown in Figures 16(e)–(h). In the cell-cycle gene expression dataset, there exist several large groups of coexpressed genes that exhibit similar expression patterns [Spellman et al. 1998]. These coexpressed genes form highly dense areas in the gene coexpression graph G_E and generate a huge number of small quasi-complete subgraphs when γ_E is less than 1. To remove the influence of these small quasi-complete subgraphs, we set $min\_size = 12$ when we tested the effect of γ_E. As can be seen, the curves in Figures 16(e)–(h) follow the decreasing trend when the value of the values of γ_E and γ_P increase.

In fact, the curves in Figures 16(c)–(h) are consistent with those in Figures 12(a)–(f) in [Pei et al. 2005]. That means the effect of parameters *min_size*, γ_E, and γ_P is similar in both *Crochet* and *Crochet*$^+$.

Particularly, when γ_E decreases, the computational cost increases. However, as shown in Figure 16(h), the runtime increases mildly when $\gamma_E \leq 0.55$. When γ_E is smaller than or equal to 0.5, the number of CGCQs increases dramatically, and so does the computation time.

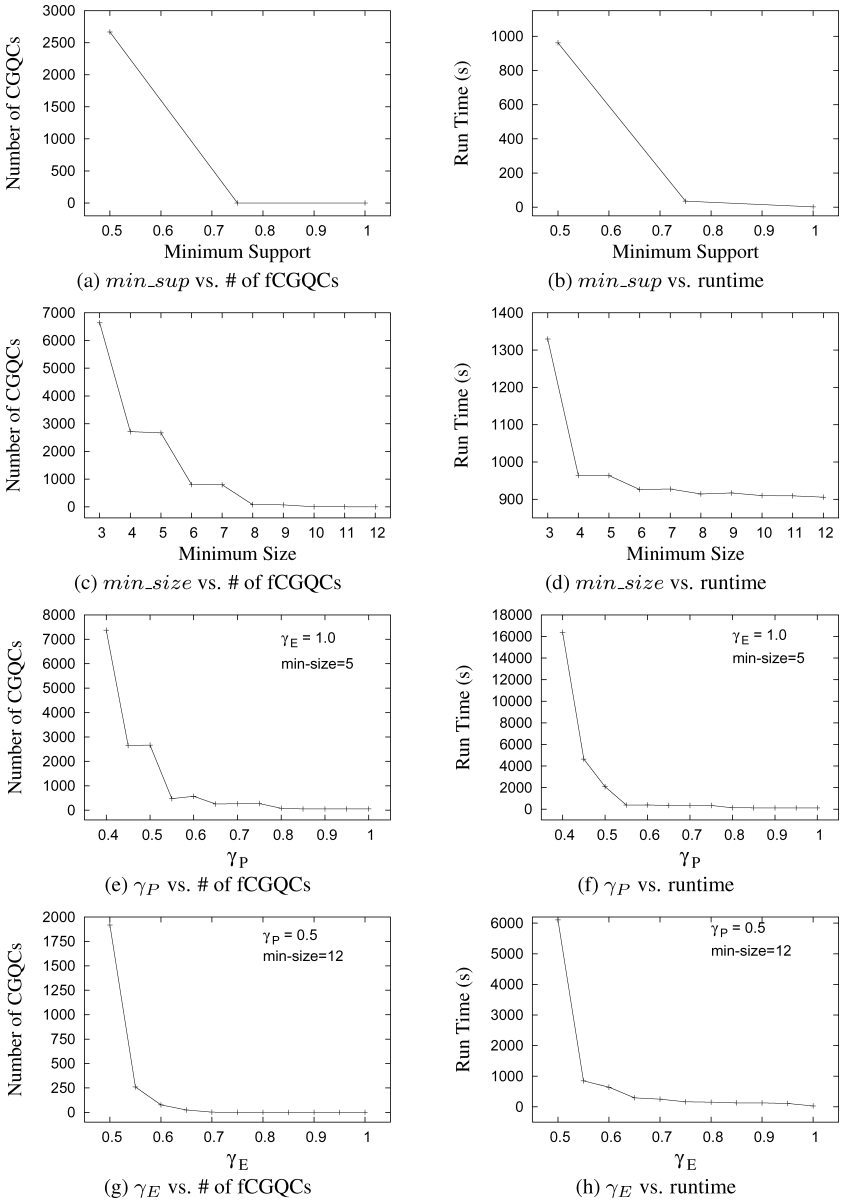


Fig. 16. The effect of parameters min_sup, min_size, γ_P and γ_E.

6.4 Effect of the Pruning Rules

In the *Crochet*$^+$ algorithm, we used a series of pruning rules and heuristics. The effect of some of these techniques, such as Lemmas 4.1, 4.5 and 4.6, as well as Heuristic 1, was tested and reported in Pie et al. [2005]. Here, we only report the effect of a newly developed technique, Lemma 4.7, and two pruning rules, Lemmas 5.1 and 5.2, which were devised specifically for mining frequent

Table II. The Effect of Various Techniques in *Crochet*$^+$ ($\gamma_E = 1.0$, $\gamma_P = 0.5$, $min\_size = 5$)

Technique	Runtime (sec) Without the Technique	Runtime (sec) with the Technique	Speedup
Lemma 4.7	4,668	2,551	1.830
Lemma 5.1	2,702	2,551	1.059
Lemma 5.2	1,588	2,551	0.623

cross-graph quasi-cliques. For each specific technique, we recorded the ratio of the runtime of *Crochet*$^+$ without the technique against using the technique. The results are shown in Table II.

From Table II, we can see that Lemma 4.7 is effective to improve the mining efficiency. By substituting the current node X with some appropriate superset X', we can skip the nodes between X and X' and go directly to the deeper level of the set enumeration tree. In our experiments, we found that this rule is particularly useful for big patterns. Once we find that the projection $P(X)$ of the current node X is a frequent cross-graph quasi-clique, we can immediately return $P(X)$ without enumerating the whole subtree of X. When the pattern is big, this rule can often save the enumeration of a large subtree and thus improve the mining efficiency substantially.

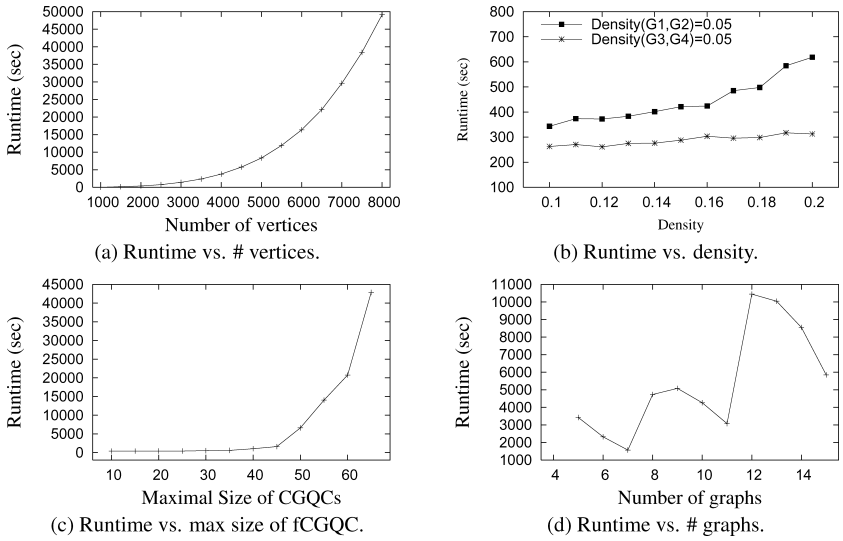
However, the improvement by Lemma 5.1 is not substantial, which means that the computation cost to apply this lemma almost offsets the the benefit brought by the lemma. Finally, Lemma 5.2 is expensive and costs more than it can save. The reason is that rule (3) of this lemma, checking whether a potential frequent CGQC is a subset of a frequent cross-graph quasi-clique, is time-consuming. However, to guarantee the maximality of frequent CGQCs, this step cannot be avoided from the algorithm. Interestingly, rule (3) of Lemma 5.2 for the *Crochet*$^+$ algorithm is analogous to rule (3) of Lemma 4.6 for the *Crochet* algorithm. By comparing Figure 13 in Pie et al. [2005] with Table II in this paper, we find that the speedup factors of these two rules are quite close: both are around 0.6.

6.5 Scalability on Synthetic Datasets

Using the synthetic datasets, we tested the scalability of *Crochet*$^+$ on four factors, namely, (1) the number of vertices of the graphs, (2) the density of the graphs, (3) the maximal size of frequent cross-graph quasi-cliques, and (4) the number of graphs.

To test the scalability with respect to the number of vertices, we fixed the density of the synthetic graphs to 1% and embedded 100 frequent CGQCs in the graphs. As the number of vertices increases, the size of the set enumeration tree increases exponentially. However, with the graph reduction, graph projection and other pruning techniques, *Crochet*$^+$ is able to handle large graphs in practical time (Figure 17(a)).

We explored the runtime of *Crochet*$^+$ with respect to the density of graphs. We first fixed the density of G_1 and G_2 (with $\gamma_1 = \gamma_2 = 1$) to 5% and increased the density of the graphs G_3 and G_4 (with $\gamma_3 = \gamma_4 = 0.5$) from 10% to 20%. The runtime is shown by the curve with filled squares in Figure 17(b). We then fixed the density of G_3 and G_4 and increased the density of G_1 and G_2 (shown



(a) Runtime vs. # vertices.
(b) Runtime vs. density.
(c) Runtime vs. max size of fCGQC.
(d) Runtime vs. # graphs.

Fig. 17. Scalability of the *Crochet*$^+$ algorithm.

by the curve with crosses in Figure 17(b)). The general trend of the curves is that the runtime of *Crochet*$^+$ increases as the density increases. Moreover, the increase of density in graphs with higher γ value brings more significant effect on the increase of *Crochet*$^+$ runtime. As can be seen in Figure 17(b), the runtime is longer and increases faster with respect to the density of graphs with higher γ values ($\gamma_1 = \gamma_2 = 1$ and $\gamma_3 = \gamma_4 = 0.5$). This is because, likely, the number of frequent cross-graph quasi-cliques is bounded by the graphs with higher γ values.

In our synthetic datasets, the noisy edges are randomly added to satisfy the density requirement. However, in real applications, the distribution of edges may not be uniform. To test the performance of *Crochet*$^+$ with skewed edge distribution, we increase the maximal size of frequent CGQCs embedded in the graphs. Intuitively, a large frequent CGQC forms a local dense area in the graphs, and the larger the size, the more skewed the data distribution. Since the local dense area contains potential frequent CGQCs, few vertices and edges can be pruned from the area, and thus the search is more costly. As can be seen in Figure 17(c), the runtime of *Crochet*$^+$ increases dramatically when the maximal size of frequent CGQCs increases.

Last, Figure 17(d) shows the scalability of *Crochet*$^+$ with respect to the number of graphs. The number of graphs influences the runtime in two aspects. First, given a fixed *min_sup* threshold, the more graphs, the less likely a subset of vertices form a frequent cross-graph quasi-clique. Therefore, more vertices can be pruned and the mining cost is lower. For example, when $n = 4, 5, 6$, the minimum frequencies are 3, 4, and 5, respectively. That means, a frequent cross-graph quasi-clique has to be supported by at least 3, 4, and 5 graphs, respectively. As a result, the runtime decreases when the number of graphs increases. Similarly, when $n = 8, 9, 10, 11$, the minimum frequencies are 6,7,8,9,

Table III. A Collection of Twenty Microarray Datasets

	Experiments	# Genes	# Time Points	Reference
1	Cdc28 block release	6178	17	[Spellman et al. 1998]
2	Cdc15 block release	6178	24	[Spellman et al. 1998]
3	Alpha factor release	6178	18	[Spellman et al. 1998]
4	Elutriation	6178	14	[Spellman et al. 1998]
5	Mec1 mutant irradiation	6129	11	[Gasch et al. 2001]
6	Mec1 mutant MMS	6129	7	[Gasch et al. 2001]
7	Wildtype irradiation	6129	12	[Gasch et al. 2001]
8	Wildtype MMS	6129	7	[Gasch et al. 2001]
9	Diamide exposure	6139	8	[Gasch et al. 2000]
10	Diauxic	6139	8	[Gasch et al. 2000]
11	DTT (exp. 1)	6139	8	[Gasch et al. 2000]
12	DTT (exp. 2)	6139	7	[Gasch et al. 2000]
13	H_2O_2 response	6139	10	[Gasch et al. 2000]
14	Heat shock (exp. 1)	6139	8	[Gasch et al. 2000]
15	Heat shock (exp. 2)	6139	7	[Gasch et al. 2000]
16	Menadione exposure	6139	9	[Gasch et al. 2000]
17	Nitrogen depletion	6139	10	[Gasch et al. 2000]
18	Sorbitol effects	6139	7	[Gasch et al. 2000]
19	YPD Stationary phase (exp. 1)	6139	12	[Gasch et al. 2000]
20	YPD Stationary phase (exp. 2)	6139	10	[Gasch et al. 2000]

and the runtime roughly show a decreasing trend. Again, when n increases from 12 to 15, the minimum frequency increases from 9 to 12, and the runtime shows a decreasing trend.

Second, given a fixed minimum frequency, when the number of graphs increases, the number of possible combinations of graphs increases exponentially. Therefore, it takes more time for the algorithm to test whether a subset of vertices is frequently supported by the given graphs. For example, with $min\_sup = 0.75$, when the number of graphs is 7 and 8, the minimum frequency is both 6. From Figure 17(d), we can see that the runtime of 8 is much longer than that of 7. As another example, when the number of graphs is 11 and 12, the minimum frequency is both 9. The runtime for 12 graphs is much longer than that for 11 graphs.

The results of the first three scalability tests are consistent with those of *Crochet*. That is, the runtime shows the similar trends with respect to the number of vertices, the density of graphs, and the maximal size of patterns. The major difference occurs when the number of graphs increases. The runtime of *Crochet* decreases monotonically when the number of graphs increases, while *Crochet*$^+$ shows a periodic trend due to the resulted support threshold value.

6.6 Scalability on Real Datasets

To further test the scalability of *Crochet*$^+$ on real datasets, we collected twenty gene-expression (cDNA microarray) datasets on yeast from Stanford Microarray Database (http://smd.stanford.edu/). Table III shows the description of the datasets.

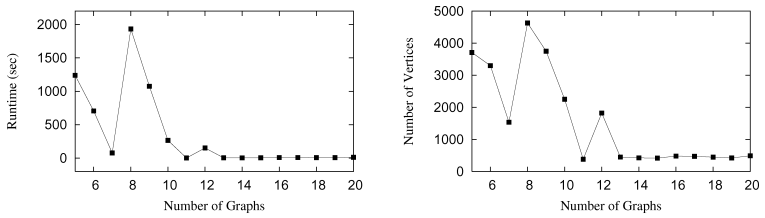


Fig. 18. The (a) runtime and (b) number of vertices after pruning with respect to the number of graphs ($\gamma_E = 1, min\_sup = 75\%$, and $min\_size = 5$).

Figure 18(a) shows the runtime of *Crochet*$^+$ with respect to the number of graphs ($\gamma_E = 1, min\_sup = 75\%$, and $min\_size = 5$). One might expect that the runtime should increase when the number of graphs increases, since the more graphs, the more data to process. However, the runtime only increases when the number of graphs is 7, 11, 15, and 19. In those situations, the number of graphs satisfies the following relation:

$$\lceil n \times min\_sup \rceil = \lceil (n+1) \times min\_sup \rceil. \tag{1}$$

For other cases, the runtime actually decreases when the number of graphs increases. This is because the more graphs, the less likely a frequent cross-graph quasi-cliques is formed. For example, when $min\_sup = 0.75$ and the number of graphs is 4, a frequent cross-graph quasi-cliques only needs to be supported by 3 graphs. However, when the number of graphs increases to 20, a frequent cross-graph quasi-clique should be supported by at least 15 graphs. As a result, the more the graphs, the sharper the cross graph pruning rules (edge reduction in Lemma 4.2 and vertex reduction in Lemma 4.1).

To further understand the pruning effect, Figure 18(b) shows the number of vertices remaining after the pruning process. As can be seen, generally, the larger the number of graphs, the more vertices are pruned. The exceptions happen when the number of graphs and the support threshold satisfy Equation (1). The trends in Figures 18(a) and 18(b) are similar, which well explain the scalability of *Crochet*$^+$.

6.7 Comparing *Crochet*$^+$ and the Integrated Graph Approach

In Example 3, we discuss the integrated graph approach, and show why it is insufficient in mining CGQCs. Comparing to the integrated graph approach, *Crochet*$^+$ has the following advantages.

First, *Crochet*$^+$ can mine CGQCs when $min\_sup < 1$, but the integrated graph approach only works when $min\_sup = 1$. This is because, after integration, we lose the information from which graphs an edge in the integrated graph originates.

Second, *Crochet*$^+$ can handle different graphs having various γ values, but the integrated graph approach can only work when all graphs sharing a uniform γ value. This is because, after integration, we have only a single integrated

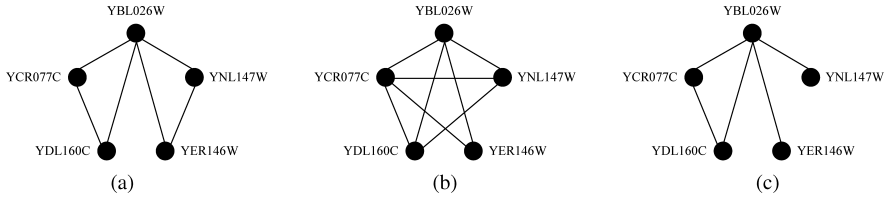


Fig. 19. The induced subgraphs of proteins S_1 = {YBL026W, YCR077C, YDL160C, YER146W, YNL147W} in (a) the yeast two hybrid graph G_{P1}, (b) the affinity precipitation graph G_{P2}, and (c) the intersected graph G_{inter}.

graph. Consequently, we can apply only one γ value when mining the single integrated graph.

Last, even in case of $min\_sup = 1$ and all graphs sharing a uniform γ value, *Crochet*$^+$ can find patterns which cannot be found by an integrated graph approach.

In this section, we report the experiments systematically compare *Crochet*$^+$ with the integrated approach, that is, intersecting the edges in all graphs on two real data sets: one is the yeast two hybrid data set, and the other is the affinity precipitation data set. We set $min\_sup = 1$, $\gamma = 0.5$, and $min\_size = 5$ in the experiments.

The *Crochet*$^+$ algorithm reports 51 patterns while the integrated graph approach identifies 53 patterns. We found 34 patterns appearing in both result sets. There are two cases for the inconsistent patterns, that is, the patterns found by one method but missing in the other method.

6.7.1 *Patterns Found by* Crochet$^+$ *but not by the Integrated Graph Approach.* For example, Figure 19 shows the induced subgraphs of the subset of proteins S_1 = {YBL026W, YCR077C, YDL160C, YER146W, YNL147W} in (a) the yeast two hybrid graph G_{P1}, (b) the affinity precipitation graph G_{P2}, and (c) the intersected graph G_{inter}, respectively. We can see that these proteins form a cross-graph quasi-clique across G_{P1} and G_{P2}, but do not form a quasi-clique in G_{inter}. We checked this subset of proteins with MIPS (http://mips.gsf.de/) complexes, and found all these proteins belong to a common complex related to mRNA metabolism. Out of the 51 patterns reported by *Crochet*$^+$, 13 patterns fall in this category.

6.7.2 *Patterns Found by* Crochet$^+$ *Broken into Smaller Patterns by the Integrated Graph Approach.* For example, Figure 20 shows the induced subgraphs of the subset of proteins S_2 = {YBL026W, YCR077C, YDL160C, YDR378C, YER112W, YJL124C, YOL149W} in (a) the yeast two hybrid graph G_{P1}, (b) the affinity precipitation graph G_{P2}, and (c) the intersected graph G_{inter}, respectively. Again, we can see that these proteins form a cross-graph quasi-clique across G_{P1} and G_{P2}. However, they do not form a single quasi-clique in G_{inter} because the degree of YDL160C is smaller than the threshold $(|S_2|-1)\times\gamma = 3$. Instead, they are broken into several smaller quasi-cliques by the integration graph method. We checked this subset of proteins with MIPS (http://mips.gsf.de/) complexes, and found that all but one protein, YOL149W,

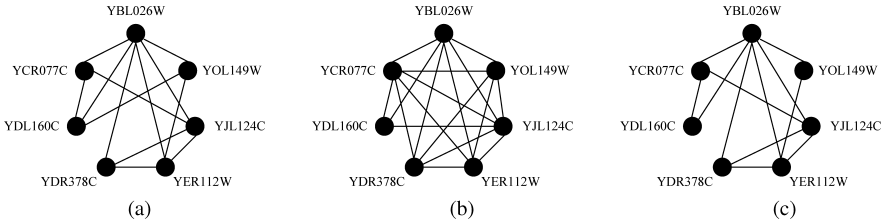


Fig. 20. The induced subgraphs of the subset of proteins S_2 = {YBL026W, YCR077C, YDL160C, YDR378C, YER112W, YJL124C, YOL149W} in (a) the yeast two hybrid graph G_{P1}, (b) the affinity precipitation graph G_{P2}, and (c) the intersected graph G_{inter}.

belong to a common complex related to mRNA metabolism. The finding may provide a good hypothesis for biologists that YOL149W share a similar function with other proteins in the pattern. Out of the 51 patterns reported by *Crochet*$^+$, 4 patterns fall in this category.

In summary, *Crochet*$^+$ can find frequent CGQC patterns which cannot be identified by the integrated graph approach and each quasi-clique pattern reported by the integrated graph approach is subsumed by some frequent CGQC found by *Crochet*$^+$. We further checked the MIPS complexes and found that all the frequent CGQC patterns reported by *Crochet*$^+$ have clear biological meaning.

7. CONCLUSIONS

In this paper, we proposed a novel and interesting problem, mining frequent cross-graph quasi-cliques from multiple graphs, and showed some application examples. The complexity analysis showed that the problem is difficult. We developed two efficient algorithms, *Crochet* and *Crochet*$^+$, which exploit several effective techniques to mine the complete set of cross-all-graphs quasi-cliques and general frequent cross-graph quasi-cliques, respectively. An extensive performance study using both real data sets and synthetic data sets illustrated that the mining results are interesting and algorithms *Crochet* and *Crochet*$^+$ are efficient and scalable.

As future work, mining frequent cross-graph quasi-cliques with various constraints is interesting and useful in applications. Moreover, efficient and scalable mining many large graphs is important. For example, it is interesting to explore efficient implementations and improvements of *Crochet* and *Crochet*$^+$ and investigate effective and efficient mining of disk-based graph databases.

REFERENCES

ABELLO, J., RESENDE, M., AND SUDARSKY, S. 2002. Massive quasi-clique detection. *Proceedings of the Latin-American Symposium on Theoretical Intormatics*, 598–612.

ALON, U., BARKAI, N., NOTTERMAN, D., GISH, K., YBARRA, S., MACK, D., AND LEVINE, A. 1999. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide array. *Proc. Natl. Acad. Sci. 96,* 12, 6745–6750.

BADER, G. AND HOGUE, C. 2003. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics 4*, 1, 2.

BARABÁSI, A. AND ALBERT, R. 1999. Emergence of scaling in random networks. *Science 286*.

Bayada, D., Simpson, R., and Johnson, A. 1992. An algorithm for the multiple common subgraph problem. *J. Chemi. Inform. Comput. Sci.*, 680–685.

Beissbarth, T., Fellenberg, K., Brors, B., et al. 2000. Processing and quality control of DNA array hybridization data. *Bioinformatics 16*, 1014–1022.

Ben-Dor, A., Shamir, R., and Yakhini, Z. 1999. Clustering gene expression patterns. *J. Computati. Biol. 6,* 3/4, 281–297.

Brazma, A. and Vilo, J. 2000. Minireview: Gene expression data analysis. *Federation European Biochemical Societies 480*, 17–24.

Bu, D., Zhao, Y., Cai, L., et al. 2003. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucl. Acids Research. 31,* 9, 2443–2450.

Cho, R., Campbell, M., Winzeler, E., et al. 1998. A genome-wide transcriptional analysis of the mitotic cell cycle. *Mole. Cell 2,* 1, 65–73.

Choi, K., Satterberg, B., Lyons, D., and Elion, E. 1994. Ste5 tethers multiple protein kinases in the map kinase cascade required for mating in *S. cerevisiae*. *Cell 78*, 499–C512.

Chu, S., DeRisi, J., Eisen, M., Mulholland, J., Botstein, D., Brown, P., and Herskowitz, I. 1998. The transcriptional program of sporulation in budding yeast. *Science 282,* 5389, 699–705.

DeRisi, J., Iyer, V., and Brown, P. 1997. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 680–686.

Ding, C., He, X., Zha, H., Gu, M., and Simon, H. 2001. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of the 1st IEEE International Conference on Data Mining*. 107–114.

Dongen, S. 2000. Graph clustering by flow simulation. PhD Thesis, University of Utrecht, The Netherlands.

Dzeroski, S. and Raedt, L. D. 2003. On multi relational data mining (mrdm). *SIGKDD Explorations 5,* 1.

Eisen, M., Spellman, P., Brown, P., and Botstein, D. 1998. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. 95*, 14863–14868.

Enright, A., Dongen, S., and Ouzounis, C. 2002. An efficient algorithm for large-scale detection of protein families. *Nucl. Acids Resear. 30,* 7, 1575–1584.

Faloutsos, C., McCurley, K., and Tomkins, A. 2004. Fast discovery of connection subgraphs. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'04)*.

Garey, M. and Johnson, D. 1979. *Computers and Intractability: a Guide to The Theory of NP-Completeness*. Freeman and Company, New York.

Gasch, A., Huang, M., Metzner, S., Botstein, D., Elledge, S., and Brown, P. 2001. Genomic expression responses to DNA-damaging agents and the regulatory role of the yeast ATR homolog Mec1p. *Mol. Biol. Cell 12,* 10, 2987–3003.

Gasch, A., Spellman, P., Kao, C., Carmel-Harel, O., Eisen, M., Storz, G., Botstein, D., and Brown, P. 2000. Genomic expression programs in the response of yeast cells to environmental changes. *Mol. Biol. Cell 11,* 12, 4241–4257.

Gavin, A., Bosche, M., Krause, R., and et. al. 2002. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature 415, 6868*, 123–124.

Hartuv, E. and Shamir, R. 2000. A clustering algorithm based on graph connectivity. *Inform. Process. Lett. 76,* 4-6, 175–181.

Ho, Y., GRUHLER, A., HEILBUT, A., et al. 2002. Systematic identification of protein complexes in saccharomyces cerevisiae by mass spectrometry. *Nature 415*, 180–183.

Holder, L., Cook, D., and Djoko, S. 1994. Substructure discovery in the subdue system. In *Proceedings of the AAAI'94 Workshop on Knowledge Discvoery in Databases (KDD'94)*. 359–370.

Inokuchi, A., Washio, T., and Motoda, H. 2000. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the European Symposium on Principle of Data Mining and Knolwedge Discovery (PKDD'00)*. 12–23.

Ito, T., Tashiro, K., Muta, S., Ozawa, R., Chiba, T., Nishizawa, M., Yamamoto, K., Kuhara, S., and Sakaki, Y. Toward a protein-protein interaction map of the budding yeast: A comprehensive system to examine twohybrid interactions in all possible combinations between the yeast proteins. *Proc. Natl. Acad. Sci*.

Jeh, G. and Widom, J. 2004. Mining the space of graph properties. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*.

KURAMOCHI, M. AND KARYPIS, G. 2001. Frequent subgraph discovery. In *Proceedings of the International Conference on Data Mining (ICDM'01)*. 313–320.

LEE, H., HSU, A., SAJDAK, J., QIN, J., AND PAVLIDIS, P. 2004. Coexpression analysis of human genes across many microarray data sets. *Genome Resear. 14*, 1085–1094.

LYONS, D., MAHANTY, S., CHOI, K., MANANDHAR, M., AND ELION, E. 1996. The Sh3-domain protein Bem1 coordinates mitogen-activated protein kinase cascade activation with cell cycle control in Saccharomyces cerevisiae. *Mol. Cell Biol. 16*, 4095–C4106.

MATSUDA, M., ISHIHARA, T., AND HASHIMOTO, A. 1999. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theor. Comput. Sci. 210, 2*, 305–325.

MEWES, H., FRISHMAN, D., MAYER, K., ET AL. 2006. MIPS: Analysis and annotation of proteins from whole genomes in 2005. *Nucl. Acids Resear. 34 (Supplement 1)*, D169–D172.

MOREAU, Y., AERTS, S., MOOR, B., STROOPER, B., AND DABROWSKI, M. 2003. Comparison and meta-analysis of microarray data: From the bench to the computer desk. *TRENDS Genetics 19*, 570–577.

NG, A., JORDAN, M., AND WEISS, Y. 2001. On spectral clustering: analysis and an algorithm. *Adv. Neural Inform. Process. Syst. 14*.

PAGE, D. AND CRAVEN, M. 2003. Biological applications of multi-relational data mining. *SIGKDD Explor. 5,* 1, 69–79.

PALMER, C., GIBBONS, P., AND FALOUTSOS, C. 2002. ANF: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'02)*.

PEI, J., JIANG, D., AND ZHANG, A. 2005. On mining cross-graph quasi-cliques. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)*.

RYMON, R. 1992. Search through systematic set enumeration. In *Proceedings of International Conference on Principles of Knowledge Representation Reasoning (RR'92)*.

SEGAL, E., WANG, H., AND KOLLER, D. 2003. Discovering molecular pathways from protein interaction and gene expression data. *Bioinformatics 19*, i264–i272.

SHAMIR, R. AND SHARAN, R. 2000. Click: A clustering algorithm for gene expression analysis. In *Proceedings of Intelligent Systems for Molecular Biology Conference (ISMB'00)*.

SPELLMAN, P., SHERLOCK, G., ZHANG, M., IYER, V., ANDERS, K., EISEN, M., BROWN, P., BOTSTEIN, D., AND FUTCHER, B. 1998. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Mol. Biol. Cell* 3273.

STUART, J., SEGAL, E., KOLLER, D., AND KIM, S. 2003. A gene-coexpression network for global discovery of conserved genetic modules. *Science 302*, 249–255.

TAKAHASHI, Y., SATOH, Y., AND SASAKI, S. 1987. Recognition of largest common fragment among a variety of chemical structures. *Analyt. Sci.*, 23–28.

TAMAYO, P., SOLNI, D., MESIROV, J., ZHU, Q., KITAREEWAN, S., DMITROVSKY, E., LANDER, E., AND GOLUB, T. 1999. Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation. *Proc. Natl. Acad. Sci.* 96, 6, 2907–2912.

TAVAZOIE, S., HUGHES, D., CAMPBELL, M., CHO, R., AND CHURCH, G. 1999. Systematic determination of genetic network architecture. *Nature Genet*. 281–285.

TONG, A. H., EVANGELISTA, M., PARSONS, A. B., AND ET AL. 2001. Systematic genetic analysis with ordered arrays of yeast deletion mutants. *Science 294*, 2364–2368.

TONG, A. H. Y., LESAGE, G., BADER, G. D., AND ET AL. 2004. Global mapping of the yeast genetic interaction network. *Science 303*, 808–813.

TSENG, G., OH, M., ROHLIN, L., LIAO, J., AND WONG, W. 2001. Issues in cDNA microarray analysis: Quality filtering, channel normalization, models of variations and assessment of gene effects. *Nucleic Acids Resear. 29*, 2549–2557.

UETZ, P., GIOT, L., CAGNEY, G., AND ET. AL. 2000. A comprehensive analysis of protein-protein interactions in Saccharomyces Cerevisiae. *Nature 403, 6770*, 601–603.

WANG, C., WANG, W., PEI, J., ZHU, Y., AND SHI, B. 2004. Scalable mining of large disk-based graph databases. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*.

XU, Y., OLMAN, V., AND XU, D. 2002. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics 18*, 536–545.

YAN, X. AND HAN, J. 2000. gspan: Graph-based substructure pattern mining. In *Proceedings of the International Conference on Data Mining (ICDM'02)*.

YAN, X. AND HAN, J. 2003. Closegraph: Mining closed frequent graph patterns. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*.

YAN, X., YU, P., AND HAN, J. 2004. Graph indexing: A frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*.

Received August 2006; revised March 2008; accepted July 2008