# Out-of-Core Coherent Closed Quasi-Clique Mining from Large Dense Graph Databases

ZHIPING ZENG, JIANYONG WANG, and LIZHU ZHOU
Tsinghua University
and
GEORGE KARYPIS
University of Minnesota

Due to the ability of graphs to represent more generic and more complicated relationships among different objects, graph mining has played a significant role in data mining, attracting increasing attention in the data mining community. In addition, frequent coherent subgraphs can provide valuable knowledge about the underlying internal structure of a graph database, and mining frequently occurring coherent subgraphs from large dense graph databases has witnessed several applications and received considerable attention in the graph mining community recently. In this article, we study how to efficiently mine the complete set of coherent closed quasi-cliques from large dense graph databases, which is an especially challenging task due to the fact that the downward-closure property no longer holds. By fully exploring some properties of quasi-cliques, we propose several novel optimization techniques which can prune the unpromising and redundant subsearch spaces effectively. Meanwhile, we devise an efficient closure checking scheme to facilitate the discovery of closed quasi-cliques only. Since large databasescannot be held in main memory, we also design an

A preliminary conference version of this article entitled "Coherent Closed Quasi-Clique Discovery from Large Dense Graph Databases" appeared in the 2006 *Proceedings of the ACM SIGKDD International Conference on Management of Data.* The work of Z. Zeng, J. Wang, and L. Zhou was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 60573061, the 973 Program under Grant 2006CB303103, the Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList), and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry of China. The research of G. Karypis was supported in part by NSF Grants EIA-9986042, ACI-0133464, IIS-0431135, and NIH RLM008713A; the Digital Technology Center at the University of Minnesota, and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement no. DAAD19-01-2-0014. The content of this article does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

Authors' addresses: Z. Zeng, J. Wang, and L. Zhou, Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, PR China; email: clipse.zeng@gmail.com; {jianyong, dcszlz}@tsinghua.edu.cn; G. Karypis, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455; email: karypis@cs.umn.edu.

out-of-core solution with efficient index structures for mining coherent closed quasi-cliques from large dense graph databases. We call this Cocain*. Thorough performance study shows that Cocain* is very efficient and scalable for large dense graph databases.

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

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Graph mining, quasi-clique, coherent subgraph, frequent closed subgraph, out-of-core algorithm

**ACM Reference Format:**

Zeng, Z., Wang, J., Zhou, L., and Karypis, G. 2007. Out-of-Core coherent closed quasi-clique mining from large dense graph databases. ACM Trans. Datab. Syst. 32, 2, Article 13 (June 2007), 40 pages. DOI = 10.1145/1242524.1242530 http://doi.acm.org/10.1145/1242524.1242530

## 1. INTRODUCTION

Data mining, whose goal is to discover implicit, previously unknown, and potentially useful information from massive data [Frawley et al. 1992], is expanding rapidly both in theory and in applications. A recent trend in data mining research is to consider more complex cases than those representable by single-table relational databases such as XML databases [Chen et al. 2003; Yang et al. 2003], spatial databases [Zhang et al. 2005; Papadias et al. 2005], microarray databases [Hu et al. 2005], graph databases [Wang et al. 2006b; Horvath et al. 2006], and so on. Both practical and theoretical problems should be solved for the databases involving these complex data types due to the relationships between entities that are thereby introduced.

As one of the central problems considered in the data mining community, the discovery of frequent patterns in a database, that is, patterns occuring in at least a certain specified number of elements of the database, has attracted much attention and made great progress in recent years. In addition to be interesting in their own right, frequent patterns can be used in mining association rules [Agrawal and Srikant 1994; Klemettinen et al. 1994], correlations [Brin et al. 1997; Wang et al. 2006b], causality [Silverstein et al. 2000], sequence patterns [Agrawal and Srikant 1995; Zhang et al. 2005], episodes [Mannila et al. 1997; Laxman and Unnikrishnan 2005] and emerging patterns [Dong and Li 1999], and also can be used as building blocks or features for clustering [Hu et al. 2002; Wang et al. 2002], classification [Matsuda et al. 1999; Hu et al. 2005; Yan et al. 2004], and predictive data mining tasks [Borgelt and Berthold 2002; Deshpande et al. 2005].

Previous studies on frequent pattern discovery have concentrated on relatively simple notions of patterns and elements in the database, as they are typically used for discovering association rules [Agrawal et al. 1993]. However, in a wide array of disciplines, data can be intuitively cast into graph patterns [Chakrabarti and Faloutsos 2006]. Meanwhile, due to the significance of application areas such as the analysis of chemical molecules [Borgelt and Berthold 2002] or graph structures in the World Wide Web [Broder et al. 2000], there has been increased interest in algorithms that can perform frequent pattern discovery in databases of structured objects, such as trees and arbitrary graphs.

While the frequent connected subgraph mining problem for tree datasets can be solved in incremental polynomial time (see Chi et al. [2005] for an overview of frequent subtree mining), it becomes intractable for arbitrary graph databases. However, one of the most general forms for modeling complex, structured data is that of the graph. Furthermore, graphs can naturally model increasingly generic and complicated relationships among different objects. Frequent subgraph mining has a wide range of applications, such as chemical compound classification [Dehaspe et al. 1998], functional annotation [Hu et al. 2005], and the discovery of activity-related groups of chemical compounds, contrast fragment structures, and functional modules among proteins. For all of these reasons, the latter has become more and more significant, attracting much attention in the data mining community. In past years, many frequent substructure mining algorithms have also been proposed, typical examples including AGM [Inokuchi et al. 2000], TreeMiner [Zaki 2002], FSG [Kuramochi and Karypis 2001], gSpan [Yan and Han 2002], PB [Vanetik et al. 2002], FFSM [Huan et al. 2003], CloseGraph [Yan and Han 2003], ADI-Mine [Wang et al. 2004], FPGrowth [Buehrer et al. 2006], EM [Hashimoto et al. 2006], Cocain [Zeng et al. 2006] and so on.

However, graphs in general have undesirable theoretical properties with regard to algorithmic complexity. In terms of complexity theory, currently no efficient algorithms are known to determine if one graph is isomorphic to a subgraph of another. Furthermore, no efficient algorithm is known to perform systematic enumeration of the subgraphs of a given graph, a common facet of data mining algorithms. Although several algorithms for mining frequent connected subgraphs from datasets of arbitrary graphs have demonstrated their performance empirically, we note that this general problem cannot be solved in output polynomial time, unless $P = NP$ [Horvath et al. 2006].

Meanwhile, several recent studies have shown that mining frequent coherent subgraphs is especially useful [Hu et al. 2005; Pei et al. 2005; Yan et al. 2005; Wang et al. 2006b], where a *coherent* subgraph can be informally defined as a subgraph that satisfies a minimum cut bound (the formal definition can be found in Section 2.1), since the set of frequent coherent subgraphs mined from a graph database usually reflects the density distribution of the relationships among objects in the database, and can provide valuable knowledge about the internal structure of the graph database. Note that, in contrast to traditional frequent subgraph mining, in this new problem setting we do not require each embedding of a given frequent coherent subgraph to have exactly the same edge topology, but rather to have a sufficient number of embeddings in the graph database, each having a highly connected set of vertices. Frequent coherent subgraph mining has also experienced several applications [Hu et al. 2005; Wang et al. 2006b], and we will show two examples in the following.

*Example* 1.1 (*Highly Correlated Stock Discovery*). A stock market dataset corresponding to a certain period can be converted to a graph in the following way. A stock is represented by a vertex whose label is the stock name,

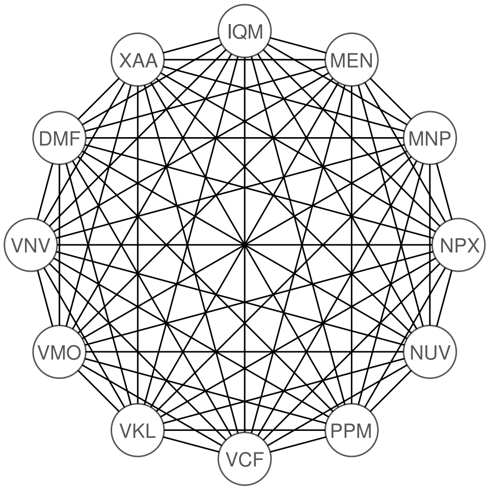


Fig. 1. The maximum closed clique in the stock market database with correlation coefficient threshold 0.90 and minimum relative support threshold 100%.

and an edge is used to connect two vertices if their correlation coefficient is no smaller than a user-specified threshold. A coherent subgraph like a clique is very meaningful from the application point of view, as it implies that the prices of the stocks contained in a coherent subgraph usually evolve synchronously over time, and a change of one stock's price can be used to predict a similar change for the prices of all other stocks in the same subgraph [Boginski et al. 2004; Wang et al. 2006b]. Figure 1 shows the maximum frequent closed clique mined from 11 sets of US stock market data with correlation coefficient threshold 0.9 and minimum relative support threshold 100% (more details can be found in Wang et al. [2006b]).

*Example* 1.2 (*Functional Annotation*). Similarly, a set of microarray data can be converted to a graph in which each node represents a unique gene and an edge represents a strong similarity between the expression data of the two genes corresponding to its two end-nodes. The similarity can also be measured by a Pearson correlation coefficient. By mining coherent dense subgraphs from a massive microarray database, functional modules can be mined and used to predict functions for uncharacterized genes. For example, Figure 2 shows a coherent subgraph discovered from the yeast microarray database by the Codense algorithm [Hu et al. 2005]. All five genes except $ASC1$ are known to be involved in protein biosynthesis. $ASC1$ can therefore be predicted to have the same function, as well.

In addition, coherent subgraph mining has been shown useful in identifying the clusters of genes that are coexpressed, as well as their protein interacts [Pei et al. 2005], and in searching the maximal common structural features among protein molecular graphs [Kato and Takahashi 2001].

During past years, several coherent subgraph discovery algorithms have been proposed [Hu et al. 2005; Pei et al. 2005; Yan et al. 2005; Wang et al.

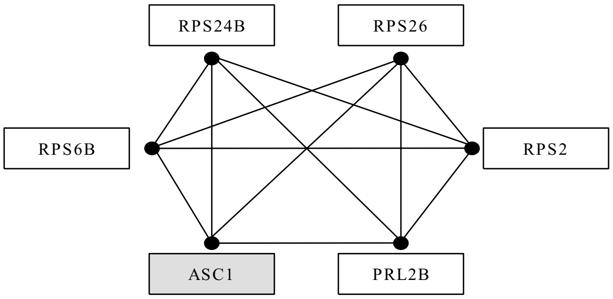


Fig. 2. Coherent dense subgraph containing six genes mined from the yeast microarray database using Codense.

2006b]. However, each of the previously proposed algorithms has its own limitations. For example, the algorithm proposed in Pei et al. [2005] can only mine quasi-cliques with exact 100% support threshold from a relational graph database, where a *relational graph* is defined as a special type of graph structure whose vertices are uniquely labeled [Yan et al. 2005]; similarly, the Clan algorithm [Wang et al. 2006b] can only mine fully connected frequent subgraphs (i.e., frequent cliques). In addition, all of the aforementioned algorithms are based on the assumption that either the entire database or its majority can fit into main memory. They cannot adapt to large dense databases. In this article, we study a more general problem formulation: mining frequent quasi-cliques from large dense graph databases, that is, we neither limit the minimum support to 100% nor require the input graphs to be relational. While the problem becomes more general, it gets more difficult. As we will see later, the downward-closure property [Agrawal and Srikant 1994] no longer holds (the same problem is faced by Pei et al. [2005] and Zhang et al. [2005]), thus devising some effective search-space pruning techniques is especially challenging. By fully exploring some properties of quasi-cliques, we propose several novel optimization techniques, and also an efficient closure checking scheme in order to facilitate the discovery of closed quasi-cliques only. Meanwhile, we develop an efficient coherent closed quasi-clique mining algorithm, Cocain*, and some efficient index structures for mining out-of-core structures.

## 2. PROBLEM FORMULATION

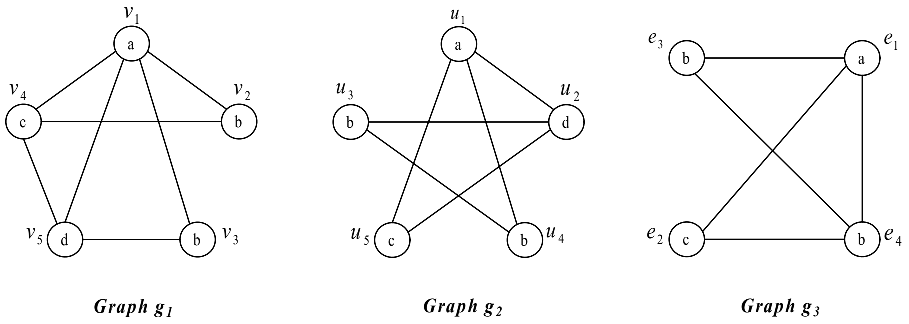
In this section, we introduce some preliminary concepts, notation, and terms in order to simplify our following discussion. We also formulate the problem of mining frequent closed coherent quasi-cliques from graph transaction databases. Table I summarizes some common notations used in graph theory and their meanings.

### 2.1 Preliminaries

In this article, we consider a *simple graph* only, which does not contain self-loops, multiedges, or edge labels. An *undirected vertex-labeled graph transaction* $G$ can be represented by a 4-tuple $G = (V, E, L, F)$. If $|G| = k$, $G$ is called

Table I. Notations Used in This Article

| Notations | Description |
|---|---|
| $V$ | $V = \{v_1, v_2, ..., v_k\}$, the set of vertices |
| $E$ | $E \subseteq V \times V$, the set of edges |
| $L$ | the set of vertex labels |
| $F$ | $F : V \rightarrow L$, the mapping function from labels to vertices |
| $G$ | $G = (V, E, L, F)$, an undirected vertex-labeled graph transaction |
| $\|G\|$ | $\|G\| = \|V\|$, the cardinality of $G$ |
| $\mathcal{L}(v)$ | the label of vertex $v$ |
| $G(S)$ | the induced subgraph on $S$ from $G$, where $S \subseteq V(G)$ |
| $N^G(v)$ | $N^G(v) = \{u\|(v, u) \in E(G)\}$ |
| $deg^G(v)$ | $deg^G(v) = \|N^G(v)\|$ |
| $dis^G(u, v)$ | the number of edges in the shortest path between $u$ and $v$ in $G$ |
| $V^G_{cad}(g)$ | the set of extensible candidate vertices with respect to $g$ in $G$ |
| $V^G_{vad}(g)$ | the set of valid extensible candidate vertices with respect to $g$ in $G$ |



*Graph $g_1$*

*Graph $g_2$*

*Graph $g_3$*

Fig. 3. Examples of a 0.5-quasi-clique.

a $k$-graph. A graph $G$ is said to be *connected* if $\forall u, v \in V(G), dis^G(u, v) < +\infty$ (i.e., there is a path from any vertex to any other vertex in the graph $G$). A graph that is not connected is said to be *disconnected*. Moreover, an *induced subgraph* of a graph $G$ is a subset of the vertices of $V(G)$, together with any edges whose endpoints are all in this subset. In the following discussions, the term "graph" means the undirected vertex-labeled graph, unless otherwise stated. Next we will introduce the formal definition of quasi-cliques.

*Definition* 2.1 (*$\gamma$-Quasi-clique*). A $k$-graph($k \geq 1$) $G$ is a $\gamma$-quasiclique ($0 \leq \gamma \leq 1$) if $\forall v \in V(G)$, $deg^G(v) \geq \lceil \gamma \cdot (k-1) \rceil$.

From the preceding definition we can see that quasi-cliques are subgraphs that satisfy a user-specified minimum vertex degree bound $\lceil \gamma \cdot (k-1) \rceil$. Apparently, a $\gamma$-quasi-clique must be a fully connected graph when $\gamma = 1$. This definition also means that singleton graphs are considered as $\gamma$-quasi-cliques. Given a $k$-graph $G$, if $\exists v \in V(G)$ such that $deg^G(v) = \lceil \gamma \cdot (k-1) \rceil$ and $\lceil \gamma \cdot (k-1) \rceil \neq \lceil \gamma \cdot k \rceil$, $v$ is called a *critical vertex* of $G$ with respect to $\gamma$.

As shown in Figure 3, $\forall v \in V(g_1)$, $deg^{g_1}(v) \geq 2 = \lceil 0.5 \times (5-1) \rceil$, so $g_1$ is a 0.5-quasi-clique. Since $deg^{g_1}(v_3) = \lceil 0.5 \times 4 \rceil$ and $\lceil 0.5 \times 4 \rceil \neq$

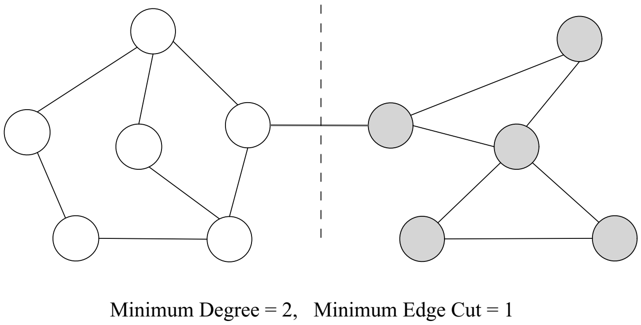


Fig. 4. A sample graph whose edge connectivity is smaller than minimum vertex degree.

$\lceil 0.5 \times 5 \rceil$, $v_3$ is a critical vertex of $g_1$ with respect to 0.5. However, $g_1$ is not a 0.6-quasi-clique, as there exists a vertex $v$ (e.g., $v_2$ and $v_3$) such that $deg^{g_1}(v) < \lceil 0.6 \times (5-1) \rceil$.

Most existing frequent subgraph mining algorithms are based on the downward-closure property [Agrawal and Srikant 1994]. Unfortunately, this nice property does not hold for quasi-clique patterns. An induced subgraph of a $\gamma$-quasi-clique may not be a $\gamma$-quasi-clique. For instance, in Figure 3, $g_2$ is a 0.5-quasi-clique, but one of its induced subgraphs, $g_2(\{u_3, u_4, u_5\})$, is not.

*Definition* 2.2 (*Edge Cut and Edge Connectivity*). Given a connected graph $G = (V, E)$, an edge cut is a set of edges $E_c$ such that $G' = (V, E - E_c)$ is disconnected. A minimum cut is the smallest set among all edge cuts. The edge connectivity of $G$, denoted by $\kappa(G)$, is the size of the minimum cut.

As shown in Yan et al. [2005], although the minimum vertex degree can reflect the level of connectivity of a graph to some extent, it cannot guarantee that the graph is connected in a balanced way, as the edge connectivity might be much smaller than the minimum vertex degree. Figure 4 shows such an example. However, the following lemma gives a lower bound on the edge connectivity of a $\gamma$-quasi-clique with $\gamma \geq 0.5$, which guarantees the coherency of the $\gamma$-quasi-clique when $\gamma \geq 0.5$ holds.

LEMMA 2.1 (MINIMUM EDGE CONNECTIVITY). *Let n-graph $Q = (V, E)$ be a $\gamma$-quasi-clique* ($0.5 \leq \gamma \leq 1, n \geq 2$). *The edge connectivity of $Q$ cannot be smaller than $\lfloor \frac{n}{2} \rfloor$, namely, $\kappa(Q) \geq \lfloor \frac{n}{2} \rfloor$.*

PROOF. Let us divide $V$ into two nonempty sets $V_1$ and $V_2$, and suppose $V_1 \leq V_2$ and $|V_1| = k$, then $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$ must hold.

Since $Q$ is a $\gamma$-quasi-clique, $\forall v \in V_1$, $deg^Q(v) \geq \lceil \gamma \cdot (n-1) \rceil$. However, $v$ is at most adjacent to other $k-1$ vertices in $V_1$, and $k-1 \leq \frac{n}{2}-1 < \gamma(n-1)$. Therefore, $v$ must be adjacent to vertices belonging to $V_2$, and the number of edges which connect $v$ and vertices in $V_2$ must be no smaller than $\lceil \gamma \cdot (n-1) \rceil - (k-1)$. There are $k$ vertices in $V_1$, so there exist at least $k \cdot (\lceil \gamma \cdot (n-1) \rceil - (k-1))$ edges connecting $V_1$ and $V_2$. Let $f(k) = k \cdot (\lceil \gamma \cdot (n-1) \rceil - (k-1))$, then

$$f(k) = -k^2 + k \cdot (\lceil \gamma \cdot (n-1) \rceil + 1). \tag{1}$$

The second-order derivative of $f(k)$ (i.e., $\frac{d^2 f(k)}{dk^2} = -2$) is negative, and $f(k)$ achieves the maximum at its sole stationary point $k = \frac{\lceil \gamma \cdot (n-1) \rceil + 1}{2}$. As there exists only one stationary point for quadratic polynomial function $f(k)$ and $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$, $f(k)$ must get its minimum either at $k = 1$ or at $k = \lfloor \frac{n}{2} \rfloor$. When $k = 1$, $f(1) = \lceil \gamma \cdot (n-1) \rceil \geq \lceil \frac{n-1}{2} \rceil = \lfloor \frac{n}{2} \rfloor$, while when $k = \lfloor \frac{n}{2} \rfloor$, each vertex in $V_1$ must be adjacent to at least one vertex in $V_2$, thus $f(\lfloor \frac{n}{2} \rfloor) \geq |V_2| \geq \lfloor \frac{n}{2} \rfloor$. From the previous result, we can get that $\forall k \in [1, \lfloor \frac{n}{2} \rfloor]$, $f(k) \geq \lfloor \frac{n}{2} \rfloor$. According to the definition of edge connectivity, $\kappa(Q) \geq \lfloor \frac{n}{2} \rfloor$. □

Because we are more interested in mining tightly connected subgraphs, we do not expect that the edge connectivity of a subgraph pattern is too small in comparison with the minimum vertex degree. From Lemma 2.1 we know that if $\gamma \geq 0.5$, the minimum cut of a $\gamma$-quasi-clique is no smaller than half the size of the corresponding quasi-clique, which assures that the vertices in the $\gamma$-quasi-clique are connected tightly and relatively evenly. In this article, a $\gamma$-quasi-clique is said to be *coherent* if $\gamma \geq 0.5$, and if not explicitly stated, the parameter $\gamma$ by default has a value no smaller than 0.5.

*Definition* 2.3 (*$\gamma$-Isomorphism*). A graph $G_1 = \{V_1, E_1, L_1, F_1\}$ is $\gamma$-isomorphic to another graph $G_2 = \{V_2, E_2, L_2, F_2\}$ iff both are $\gamma$-quasi-cliques, $|G_1| = |G_2|$, and there exists a bijection $f : V_1 \rightarrow V_2$ such that $\forall v \in V_1, F_1(v) = F_2(f(v))$.

According to the definition of $\gamma$-isomorphism, we know that $\gamma$-isomorphism is quite different from the graph isomorphism in graph theory, which is defined as a bijection $f : V(G_1) \rightarrow V(G_2)$ from a graph $G_1$ to another graph $G_2$ such that $(u, v) \in E(G_1)$ iff $(f(u), f(v)) \in E(G_2)$. The $\gamma$-isomorphism between two $\gamma$-quasi-cliques does not imply an exact bijective edge mapping. For example, in Figure 3, $g_1$ and $g_2$ are 0.5-isomorphic to each other, although they are not graph-isomorphic to each other.

A *multiset* is defined as a bag of vertex labels in which the order is ignored, but multiplicity is explicitly significant, for example, multisets $\{1, 2, 3\}$ and $\{3, 1, 2\}$ are equivalent, but $\{1, 2, 3\}$ and $\{3, 1, 1, 2\}$ differ. Let $M(G)$ indicate the multiset of labels of a graph $G$. Hence, in Figure 3, $M(g_1) = \{a, b, b, c, d\}$ and $M(g_3) = \{a, b, b, c\}$. From the $\gamma$-isomorphism definition, we can derive the following lemma.

LEMMA 2.2. *Two $\gamma$-quasi-cliques $Q_1$ and $Q_2$ are $\gamma$-isomorphic to each other iff $M(Q_1) = M(Q_2)$.*

Lemma 2.2 can be used to detect the existence of $\gamma$-isomorphism between two graphs, that is, we just need to check if they are both $\gamma$-quasi-cliques and their multisets are equivalent.

For two $\gamma$-quasi-cliques $Q$ and $Q'$, if $M(Q) \subseteq M(Q')$, $Q$ is called a *subquasi-clique* of $Q'$, while $Q'$ is called a *superquasi-clique* of $Q$. We use $Q \sqsubseteq Q'$ or

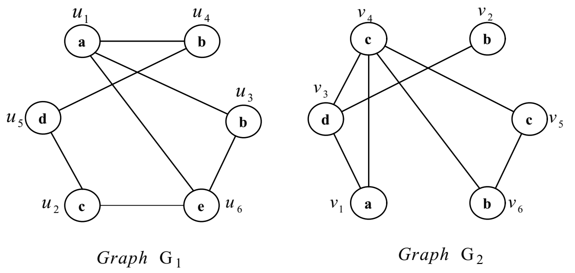


Fig. 5. An example of a graph transaction database D.

$Q \sqsubset Q'$ (i.e., $Q \sqsubseteq Q'$ but $Q \neq Q'$), respectively, to denote the subquasi-clique or proper subquasi-clique relationship.

## 2.2 Problem Definition

A *graph transaction database D* consists of a set of input graphs, and the cardinality of D is denoted by $|D|$. Figure 5 shows an example of a graph transaction database $D$ which consists of two input graphs $G_1$ and $G_2$, and $|D| = 2$. For simplicity, in the rest of this article we sometimes omit the notation of database $D$ when the context is clear.

For two graphs $G$ and $G'$, let $g$ be an induced subgraph of $G$, if $M(g) = M(G')$, then we call $g$ an *instance* of $G'$ in $G$. If there exists at least one instance of $G'$ in $G$, we say that graph $G$ *roughly supports* $G'$, while graph $G'$ is roughly supported by $G$. Meanwhile, if $g$ is $\gamma$-isomorphic to another $\gamma$-quasi-clique $Q$, we call $g$ an *embedding* of $Q$ in $G$. If there exists at least one embedding of $Q$ in $G$, then $G$ is said to *strictly support* $Q$, while $Q$ is said to be strictly supported by $G$.

The number of input graphs in graph database $D$ that strictly (or roughly) support a $\gamma$-quasi-clique $Q$ (or a graph $G$) is called the *absolute strict-support* (or *absolute rough-support*) of $Q$ (or $G$) in $D$, denoted by $sup_s^D(Q)$ (or $sup_r^D(G)$), while the *relative strict-support* (or *relative rough-support*) is the percentage of input graphs which strictly (or roughly) support $Q$ (or $G$), denoted by $rsup_s^D(Q)$ (or $rsup_r^D(G)$). Obviously, $rsup_s^D(Q) = sup_s^D(Q)/|D|$, and $rsup_r^D(G) = sup_r^D(G)/|D|$.

For an absolute support threshold *min_sup* and graph transaction database $D$, a quasi-clique $Q$ (or a subgraph $g$) is called a *frequent* quasi-clique (or a *vice-frequent* graph) if $sup_s^D(Q) \geq min\_sup$ (or $sup_r^D(g) \geq min\_sup$). If there does not exist any other quasi-clique $Q'$ such that $Q \sqsubset Q'$ and $sup_s^D(Q) \leq sup_s^D(Q')$, then $Q$ is called a *closed* quasi-clique in $D$.[1]

*Problem Definition.* Given a graph transaction database $D$ and a minimum strict-support threshold *min_sup*, we study the problem of mining the complete set of $\gamma$-quasi-cliques in $D$ that are *frequent*, *closed*, and *coherent* (i.e., $\gamma \geq 0.5$).

[1] Here the definition of a "closed" pattern is a little different from the traditional because the downward-closure property does not hold for frequent quasi-clique mining, and it is possible that a quasi-clique's strict-support is greater than that of its subquasi-cliques. Also, in the case where $Q \sqsubset Q'$ and $sup_s^D(Q) \leq sup_s^D(Q')$, we say $Q'$ can *subsume* $Q$.

## 3. RELATED WORK

Computing clique or quasi-clique structures from a single graph has long been studied [Karp 1972; Bron and Kerbosch 1973; Feige et al. 1991; Hastad 1996; Abello et al. 2002; Ostergard 2002] and identifying the size of the largest clique in a graph was one of the first problems shown NP-hard [Karp 1972]. However, these previous works did not study the problem of mining quasi-cliques that frequently occur across a set of input graphs. Recently, several algorithms were proposed to mine frequent dense subgraphs from large graph databases, such as Codense [Hu et al. 2005], Crochet [Pei et al. 2005], CloseCut and SPLAT [Yan et al. 2005], Clan [Wang et al. 2006b], and so on. In addition, according to our definition of a quasi-clique, we do not require that there exists an edge for each pair of vertices in a quasi-clique, thus mining closed quasi-cliques is similar to some extent to some variations of formal concept analysis, such as $\delta$-free sets [Selmaoui et al. 2006] and biclustering of categorical data [Pensa et al. 2005], which also allow for a certain number of zeros in a rectangle of ones.

In Hu et al. [2005], the Codense algorithm was devised to mine frequent coherent dense subgraphs across massive biological networks and to discover functionally homogenous clusters. The problem formulation in Hu et al. [2005] is quite different from ours, and a frequent coherent dense subgraph mined by Codense might not necessarily be a frequent quasi-clique. In Yan et al. [2005], two approaches, CloseCut and Splat, were proposed to mine frequent closed subgraphs with connectivity constraints. They can only be applied to relational graph databases. The problem studied in Yan et al. [2005] also differs from ours.

Probably the most related research is that in Pei et al. [2005] and [Wang et al. 2006b], which have their own limitations compared with this work. The Crochet algorithm proposed in Pei et al. [2005] can mine quasi-cliques, but requires each mined quasi-clique to have a perfect 100% support threshold, and also only works for relational graph databases. Meanwhile, the Clan algorithm proposed in Wang et al. [2006b] neither requires the input graphs to be relational nor limits the support threshold to be exactly 100%, but can only mine frequent closed cliques, that is, fully connected subgraphs.

In a preliminary version of this work [Zeng et al. 2006], we studied a more general problem formulation, that is, mining frequent closed quasi-cliques from large dense graph databases, and designed an efficient closed coherent quasi-clique discovery algorithm, Cocain. In Cocain, we neither limit the minimum support to be 100% nor require input graphs to be relational. Moreover, the downward-closure property holds for clique patterns, thus designing some pruning techniques for frequent clique mining is much easier compared with quasi-clique mining. However, by fully exploring some nice properties of quasi-clique patterns, we proposed several novel apriori-like pruning techniques which can be pushed deeply into the mining process. Furthermore, we also devised an efficient closure checking scheme in order to facilitate the discovery of closed quasi-cliques only.

As a major value-added version of our preliminary work [Zeng et al. 2006], we here summarize the major extensions as follows. First, Cocain and all the aforementioned algorithms are in-memory solutions which are based on the assumption that the entire graph database or at least its majority can fit into the main memory. However, in most real applications, the graph databases are usually too large to be held in memory. In order to adapt to the huge graph database setting, we devise an effective out-of-core solution based on several newly proposed efficient index structures. Our scalability study shows that the extended algorithm, Cocain*, can scale to very large databases. Second, some of the basic operations for calculating valid extensible vertex candidates in the algorithm are very time consuming, thus, we propose some new optimization techniques and devise a new subalgorithm, Valid+. Our performance study shows that these optimization techniques are very effective in improving the algorithm performance. Finally, we have conducted extensive new performance studies, including an effectiveness test of the newly proposed optimization techniques, effectiveness comparisons among various pruning methods, an efficiency test of Cocain*, a new scalability test on much larger databases, and illustrations of some coherent closed quasi-clique examples discovered from some real databases.

## 4. ENUMERATION STRATEGY

In this section, we describe our enumeration strategy to discover the complete set of frequent coherent quasi-cliques, including an efficient canonical representation of coherent quasi-clique patterns, a vice-frequent subgraph enumeration framework, and the structural redundancy pruning technique.

### 4.1 Canonical Representation of Subgraphs

One of the key issues in graph mining is how to choose an efficient canonical form that can uniquely represent a graph and has low computational complexity in order to facilitate the graph isomorphism testing. Currently, there are two popular classes of solutions to this problem: minimum(or maximum) adjacency matrix code [Kuramochi and Karypis 2001; Huan et al. 2003] and some kind of DFS (abbreviated for depth-first search) code [Zaki 2002; Yan and Han 2002]. However, both have high computational complexity and are not the most efficient representations for quasi-clique patterns.

From Lemma 2.2, we see that a vertex label multiset preserves much information for a quasi-clique. Given a $k$-graph $g$, we call any sequence of all elements in $M(g)$ a graph *string*, and there are $k!$ different strings for $g$ if each element in $M(g)$ is distinct. Assume there is a total order (e.g., lexicographical order) on the vertex labels, we define the following total order of any two strings $p$ and $q$ with size $|p|$ and $|q|$, respectively. Let $p_i$ denote the $i$th vertex label in string $p$, we define $p < q$ if either of the following two conditions holds: (1) $\exists t (0 < t \leq min\{|p|, |q|\})$ such that $\forall i \in [1, t-1]$, $p_i = q_i$ and $p_t < q_t$; and (2) $(|p| < |q|)$ and $\forall i \in [1, |p|]$, $p_i = q_i$; otherwise $p \geq q$. A string $S_a = a_1 a_2 ... a_n$ is called a *substring* of another string $S_b = b_1 b_2 ... b_m$, denoted by $S_a \sqsubseteq S_b$ (or $S_a \sqsubset S_b$ if $m > n$), if there exist $n$ integers $1 \leq i_1 < i_2 < ... i_n \leq m$ such that $a_1 = b_{i_1}$,$a_2 = b_{i_2}$, ..., $a_n = b_{i_n}$.

*Definition* 4.1. The canonical form of a graph $G$ is defined as the minimum string among all its strings and denoted by $CF(G)$.

As we ignore the exact topology of a quasi-clique, it is evident that the preceding definition is a unique representation of a quasi-clique. However, we note that it does not hold for a general graph. After giving the definition of the canonical form of a graph and the substring relationship, we can derive the following two lemmas to facilitate $\gamma$-isomorphism checking and subquasi-clique relationship checking.

LEMMA 4.1. *Two $\gamma$-quasi-cliques $Q_1$ and $Q_2$ are $\gamma$-isomorphic to each other iff $CF(Q_1) = CF(Q_2)$.*

LEMMA 4.2. *Given two $\gamma$-quasi-cliques $Q_1$ and $Q_2$, $Q_1 \sqsubseteq Q_2$ (or $Q_1 \sqsubset Q_2$) iff $CF(Q_1) \sqsubseteq CF(Q_2)$ (or $CF(Q_1) \sqsubset CF(Q_2)$).*

The previous two lemmas can be derived easily from the definition of $\gamma$-isomorphism and the subquasi-clique relationship, respectively; here we omit the proof.

## 4.2 Vice-Frequent Subgraph Enumeration

According to the definition of a vice-frequent graph, it is evident that any induced subgraph of a vice-frequent graph must be also vice-frequent. Thus, this downward-closure property can be exploited for vice-frequent subgraph enumeration.

Once given a total order on the vertex labels, we can represent the vice-frequent subgraphs by their corresponding canonical forms and conceptually organize them into a lattice-like structure in the same way as Wang et al. [2006b]: Each node represents a subgraph in the form of "canonical form:rough-support:strict-support". (Note that for simplicity, in the following we will denote a subgraph by its canonical form plus its rough-support and strict-support). Furthermore, each edge between two nodes represents a *direct subgraph* (i.e., with exactly one less vertex) relationship between the two corresponding subgraphs. All the vice-frequent $k$-subgraphs are at level $k$ and arranged according to the order of their canonical forms. For our running example in Figure 5, assume the total order among vertex labels is $a \leq b \leq c \leq d \leq e$, all the vice-frequent subgraphs are organized into a structure like the one shown in Figure 6. For example, $g_3$ in Figure 3 has instances (but not embeddings) in both $G_1$ and $G_2$ in Figure 5, and can be represented by the node with a label abbc:2:0, which has three 3-subgraphs (i.e., abb:2:1, abc:2:1, bbc:2:0) as its direct subgraphs. In addition, all the nodes with dashed ellipses are vice-frequent subgraphs but not frequent quasi-cliques, nodes with light grey color are nonclosed frequent quasi-cliques, and nodes with dark grey color are closed quasi-cliques. Figure 6 shows that among all the vice-frequent subgraphs, only abd:2:2 and bcd:2:2 are closed quasi-cliques.

Based on the definition of embedding and instance of a subgraph, we can see that the embedding is a special type of the instance, and an embedding of a graph $g$ must be an instance of $g$. Given a $\gamma$-quasi-clique $Q$, since $sup_s^D(Q) \leq sup_r^D(Q)$ holds, if $Q$ is frequent, then $Q$ must be vice-frequent. Consequently, we can discover the complete set of frequent $\gamma$-quasi-cliques from the set of vice-frequent subgraphs. By conceptually organizing the vice-frequent

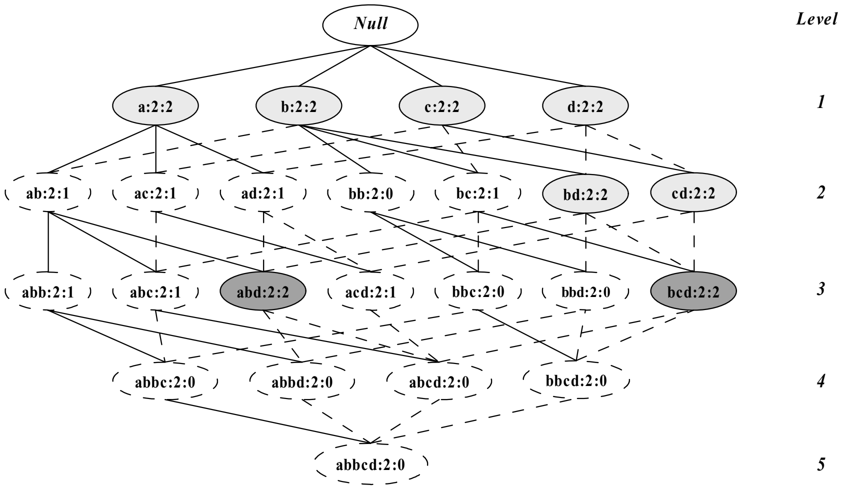


Fig. 6. A lattice-like structure built from our running example ($\gamma = 0.5$, $min\_sup = 2$).

subgraphs into a lattice-like structure, the problem of mining frequent quasi-cliques becomes that of how to traverse the lattice-like structure to enumerate vice-frequent subgraphs and discover frequent $\gamma$-quasi-cliques. Previous studies have shown two popular search strategies: DFS [Huan et al. 2003; Borgelt and Berthold 2002; Yan and Han 2002] and BFS [Kuramochi and Karypis 2001]. We adopt the DFS strategy, however, our method is slightly different from previous ones. The previous depth-first search methods like rightmost extension [Zaki 2002; Yan and Han 2002] grow the current graph of size k by one edge in each step in order to get a graph of size $(k + 1)$, while we grow the current k-subgraph by one vertex plus the corresponding edges in one step to generate a $(k+1)$-subgraph. In this way, we can get a rudimentary algorithm to discover frequent closed $\gamma$-quasi-cliques. However, this rudimentary algorithm costs too much in terms of space and runtime and is too expensive, as it incurs a lot of redundancy during the quasi-clique enumeration.

## 4.3 Structural Redundancy Pruning

By depth-first traversing the lattice-like structure, we find that some nodes might be generated more than once, thus there exists much redundancy. For example, abd:2:2 in Figure 6 can be generated from ab:2:1, ad:2:1, or bd:2:2. A simple way to remove the redundant subgraphs can be implemented by maintaining the set of already-mined subgraphs. Upon getting a new subgraph, we check whether there is any already-mined subgraph that has the same canonical form as the new one, and if so, we just throw away the newly generated subgraph. However, this does not help in removing the redundant computing.

In order to eliminate structural redundancy while maintaining the completeness of the result set, we also adopt the structural redundancy pruning technique used in Wang et al. [2006b] and propose an efficient vice-frequent subgraph enumeration method. Given an $m$-graph $G$ and $CF(G) = a_0a_1...a_m$, we require that $G$ can only be generated by growing the subgraph $g$ with canonical form $CF(g) = a_0a_1...a_{m-1}$. In this way, except for the node $\phi$, each node in the lattice-like structure has only one parent, and this lattice-like structure will turn into a tree structure. Let $LAS(g)$ be the last element in $CF(g)$ (i.e., $LAS(g) = a_m$), obviously, in the enumeration tree structure all descendants of the node $g$ would be in the form $CF(g) \diamond b_0b_1...b_k$, where $b_0 \geq LAS(g)$ and $\forall\, i \in [1, k], b_{i-1} \leq b_i$. Taking Figure 6 for example, node abd:2:2 can only be generated from node ab:2:1, and not ad:2:1. The solid lines among different nodes in Figure 6 are valid enumeration paths, while the dashed lines are invalid. Obviously, excluding the dashed lines, Figure 6 is a tree structure.

## 5. SEARCH SPACE PRUNING

In the previous section, we proposed an enumeration strategy and have already discovered coherent frequent quasi-clique patterns. However, this is a rudimentary and costly method. In this section, we propose several novel optimization techniques to prune futile parts of the search space based on some nice properties of quasi-clique patterns, including *diameter pruning, combination pruning, vertex connectivity pruning, failed-vertex pruning, and subgraph connectivity pruning*. At the end of this section, we will introduce an efficient algorithm for discovering valid extensible vertices of a subgraph and also its improvement.

### 5.1 Diameter Pruning

Before we elaborate on the optimization techniques, let us first introduce the following important lemma which plays a foundational role in the subsequent discussion.

LEMMA 5.1 (MAXIMAL DIAMETER). *If a graph $Q$ is a $\gamma$-quasi-clique, then $\forall\, u, v \in V(Q)$, $dis^Q(u, v) \leq 2$.*

PROOF. This property is a special case of Theorem 1 that appeared in Pei et al. [2005] on the diameter of a quasi-complete graph when $\gamma \geq 0.5$ holds. □

Lemma 5.1 shows that the distance between two vertices in a coherent quasi-clique must be no greater than 2. Based on this, we can derive the following lemma, which will help us prune many unpromising vertices.

LEMMA 5.2 (DIAMETER PRUNING). *Let $G$ be a graph and $S \subseteq V(G)$, if $G(S)$ is a $\gamma$-quasi-clique, then $\forall\, u, v \in S, dis^G(u, v) \leq 2$.*

PROOF. Since $G(S)$ is a subgraph of $G$, $dis^{G(S)}(u, v) \geq dis^G(u, v)$ holds. In addition, according to Lemma 5.1, we have $dis^{G(S)}(u, v) \leq 2$. Therefore, $dis^G(u, v) \leq 2$ must hold. □

Given a graph $G$, let $S \subset V(G)$ and $v \in V(G) - S$, from Lemma 5.2 we know that if $G(S)$ and $v$ can form a "bigger" quasi-clique, then $\forall\, u \in S, dis^G(v, u) \leq 2$ must hold. Accordingly, a straightforward and reasonable application of Lemma 5.2 is to discover the extensible vertices of a subgraph which can later be used to form quasi-cliques. First, with $\forall\, u \in S$ we calculate the *extensible* *vertex set* $K(u)$ ($K(u) = \{v \mid dis^G(u, v) \leq 2\}$), then conjoin all $K(u)$'s to obtain the global extensible vertex set of the subgraph $G(S)$. Obviously, $K(u)$ can be generated in three steps. The first is generating $D(u)$, which consists of the vertices that are adjacent to $u$, namely, $D(u) = \{v \mid dis^G(u, v) = 1\}$. The second is generating $I(u)$, where $I(u) = \{v \mid \exists\, u' \in D(u), dis^G(u', v) = 1\}$. Finally, we get $K(u)$ through the union of $D(u)$ and $I(u)$. In this way, we can discover an extensible vertex set for an instance of a subgraph $G(S)$, and thus pruning many unpromising vertices.

*Example* 5.1 (*Diameter Pruning*). In Figure 6, for node b:2:2, there are two instances in $G_2$, namely $G_2(\{v_2\})$ and $G_2(\{v_6\})$. Because $dis^{G_2}(v_2, v_6) = 3$, we can state that there does not exist a quasi-clique $Q$ in $G_2$ such that $V(Q) \supseteq \{v_2, v_6\}$ . Accordingly, we can remove $G_2(\{v_2, v_6\})$ from the instance set of bb, and thereby there exists no instance of bb in $G_2$. Though there exists an instance of bb in $G_1$, we can infer that the strict-support of bb is no greater than 1, which is less than *min_sup*. Thus, although bb is vice-frequent, it is unnecessary to generate node bb:2:0. Accordingly, nodes bbc:2:0, bbd:2:0, and bbcd:2:0 will also not be generated. This example shows that the *diameter pruning* method can be used to prune a lot of futile parts of the search space.

## 5.2 Combination Pruning

Furthermore, we can combine structural redundancy pruning with diameter pruning to further shrink the extensible vertex set. As stated in structural redundancy pruning, only those vertices whose labels are no smaller than $LAS(g)$ can be used to grow $g$, where $g$ is the current prefix subgraph. Therefore, when calculating $K(u)$ we can remove vertices whose labels are smaller than $LAS(g)$, and the removal of some vertices in $K(u)$ may make some of the vertices left in $K(u)$ violate the condition of an extensible vertex introduced in Lemma 5.1. For example, suppose $\exists\, v_0 \in I(u)$ and there exists only one vertex $v'$ such that $v' \in D(u)$ and $dis^G(v_0, v') = 1$. If $v' \notin V(g)$ and the label of $v'$ is smaller than $LAS(g)$, then $v'$ will never appear in all the descendants of $g$, thus a descendant $g'$ of $g$ which contains vertex $v_0$ cannot be a quasi-clique, as $dis^{g'}(u, v_0) > 2$ must hold. Therefore, we can remove the vertex $v_0$ from $K(u)$ safely.

In order to eliminate these vertices efficiently, after getting $D(u)$, we remove those vertices in $D(u)$ whose labels are both smaller than $LAS(g)$ and do not belong to $V(g)$. Let $\bar{\mathcal{D}}^g(u)$ denote the remaining of $D(u)$ after the aforementioned process, that is, $\bar{\mathcal{D}}^g(u) = \{v \mid dis^G(u, v) = 1 \wedge (v \in V(g) \vee \mathcal{L}(v) \geq LAS(g))\}$, and accordingly $\bar{\mathcal{I}}^g(u) = \{v \mid \exists\, u' \in \bar{\mathcal{D}}^g(u), dis^G(u', v) = 1\}$, $\bar{\mathcal{K}}^g(u) = \{v \mid v \in \bar{\mathcal{D}}^g(u) \cup \bar{\mathcal{I}}^g(u), \mathcal{L}(v) \geq LAS(g)\}$. After we compute the final set of extensible vertices $\bar{\mathcal{K}}^g(u)$, for each vertex $u$ in subgraph $g$, we can then conjoin all the $\bar{\mathcal{K}}^g(u)$'s to get the global extensible vertex set wih respect to $g$. We call an element in the global extensible vertex set an *extensible candidate* with respect to $g$, and use $V^G_{cad}(g)$ to denote the set of extensible candidates with respect to $g$ in $G$. Apparently, $V^G_{cad}(g) = \cap_{\forall v \in V(g)} \bar{\mathcal{K}}^g(v)$.

*Example* 5.2 (*Combination Pruning*). Given an instance $g_0 = G_1(\{u_1, u_2\})$ in Figure 5, we have $D(u_1) = \{u_3, u_4, u_6\}$. Because the labels of $u_3$ and $u_4$ are smaller than $LAS(g_0) = c$ and both are not in $V(g_0)$, we can remove them from $D(u_1)$. Thus $\bar{\mathcal{D}}(u_1) = \{u_6\}$, from which we get $\bar{\mathcal{I}}(u_1) = \{u_2, u_3\}$. Then $u_3$ in $\bar{\mathcal{I}}(u_1)$ can be pruned due to $\mathcal{L}(u_3) < \mathcal{L}(u_2)$, and we finally get $\bar{\mathcal{K}}(u_1) = \{u_2, u_6\}$. Similarly, we can get $\bar{\mathcal{K}}(u_2) = \{u_5, u_6\}$, and compute the set of extensible candidates with respect to $g_0$ in $G_1$ as $V_{cad}^{G_1}(g_0) = \{u_6\}$. From the preceding analysis we can see that although $dis^{G_1}(u_1, u_5) = 2$ and $dis^{G_1}(u_2, u_5) = 1$, $u_5$ is not an extensible candidate with respect to $g_0$, since for any descendant $g'$ of $g_0$ in the enumeration tree, $dis^{g'}(u_1, u_5) > 2$.

## 5.3 Pruning Techniques Based on Vertex Degree

Lemma 5.3. *If* $m + u < \lceil \gamma \cdot (k + u) \rceil$ *(where* $m, u, k \geq 0$ *and* $0.5 \leq \gamma \leq 1$*), then* $m < \lceil \gamma \cdot k \rceil$ *and* $\forall i \in [0, u]$, $m + i < \lceil \gamma \cdot (k + i) \rceil$.

Proof. First, we assume $m \geq \lceil \gamma \cdot k \rceil$, then $m+u \geq \lceil \gamma \cdot k \rceil + u \geq \lceil \gamma \cdot k \rceil + \lceil \gamma \cdot u \rceil \geq \lceil \gamma \cdot (k+u) \rceil$, which contradicts with the fact $m+u < \lceil \gamma \cdot (k+u) \rceil$. Thus $m < \lceil \gamma \cdot k \rceil$ holds. Second, let $t = u - i$, then $m + i = m + u - t < \lceil \gamma \cdot (k + u) \rceil - t \leq \lceil \gamma \cdot (k + u) \rceil - \lceil \gamma \cdot t \rceil \leq \lceil \gamma \cdot ((k + u) - t) \rceil = \lceil \gamma \cdot (k + i) \rceil$. □

In the following, we propose an additional three optimization techniques based on Lemma 5.3 which can be used to prune the unpromising search-space effectively.

For $v \in V_{cad}^{G}(g)$, we define the internal set $V_{in}^{g}(v) = N^{G}(v) \cap V(g)$ and external set $V_{ex}^{g}(v) = N^{G}(v) \cap V_{cad}^{G}(g)$. Let $indeg^{g}(v) = |V_{in}^{g}(v)|$ be the *inner degree* of $v$, and $exdeg^{g}(v) = |V_{ex}^{g}|$ be the *external degree*. Taking Figure 5 for example, assume $g = G_2(\{v_5\})$, $V_{cad}^{G_2}(g) = \{v_3, v_4\}$ (note $v_1$ and $v_6$ have been pruned by combination pruning), then $indeg^{g}(v_3) = 0$ and $exdeg^{g}(v_3) = 1$.

Lemma 5.4 (Vertex Connectivity Pruning). *Suppose* $g$ *is a* $k$*-subgraph of* $G$*, if* $\exists v \in V_{cad}^{G}(g)$ *such that* $indeg(v) < \lceil \gamma \cdot k \rceil$ *and* $indeg^{g}(v) + exdeg^{g}(v) < \lceil \gamma \cdot (k + exdeg^{g}(v)) \rceil$*, there does not exist a quasi-clique* $Q$ *in* $G$ *such that* $V(Q) \supseteq V(g) \cup \{v\}$.

Proof. Assume there exists a quasi-clique $Q$ such that $V(Q) \supseteq V(g) \cup \{v\}$, and let $|Q| = l$. Since $Q$ is a quasiclique, $V(Q) \subseteq V(g) \cup V_{cad}^{G}(g)$. We define $R = V(Q) \cap V_{ex}^{g}(v)$, and denoting $|R|$ by $m$, then $l - k - 1 \geq m$ and $m \leq |V_{ex}^{g}(v)| = exdeg^{g}(v)$. Because $indeg^{g}(v) < \lceil \gamma \cdot k \rceil$ and $indeg^{g}(v) + exdeg^{g}(v) < \lceil \gamma \cdot (k + exdeg^{g}(v)) \rceil$, according to Lemma 5.3, we get that $\forall i \in [0, exdeg^{g}(v)]$, $indeg^{g}(v) + i < \lceil \gamma \cdot (k + i) \rceil$. Thus, $deg^{Q}(v) = indeg^{g}(v) + m < \lceil \gamma \cdot (k + m) \rceil \leq \lceil \gamma \cdot (l - 1) \rceil$, that is, $deg^{Q}(v) < \lceil \gamma \cdot (|Q| - 1) \rceil$. This contradicts the assumption that $Q$ is a quasiclique. □

In the case of $indeg^{g}(v) + exdeg^{g}(v) < \lceil \gamma \cdot (k + exdeg^{g}(v)) \rceil$, $v$ is called an *invalid extensible candidate*, otherwise it is a *valid extensible candidate*. Obviously, invalid extensible candidates do not make any contribution to the generation of "bigger" quasi-cliques. Therefore, after getting the extensible candidate set $V_{cad}^{G}(g)$, we could remove all invalid extensible candidates from $V_{cad}^{G}(g)$. Due to the removal of these vertices, some originally valid extensible candidates may turn invalid, so we can do this pruning iteratively until no vertex can be removed from $V_{cad}^{G}(g)$. We denote the remaining set by $V_{vad}^{G}(g)$. Hence, if $G(S)$ is a quasi-clique in $G$ and $S \supset V(g)$, then $S \subseteq V(g) \cup V^G_{vad}(g)$. Consequently, we only need to use the vertices in $V^G_{vad}(g)$ to grow $g$, which can further improve the algorithm's efficiency.

Assume $g$ is a subgraph of a graph $G$, for a vertex $u \in V(g)$, let $V^g_{ext}(u) = N^G(u) \cap V^G_{vad}(g)$. Moreover, let the *extensible degree* $extdeg^g(u)$ be the cardinality of $V^g_{ext}(u)$, namely, $extdeg^g(u) = |V^g_{ext}(u)|$. If there exists a critical vertex $v$ in $g$ such that $extdeg^g(v) = 0$, we call $v$ a *failed vertex* of $g$.

Lemma 5.5 (Failed Vertex Pruning). *If there exists a failed vertex $v$ in a $k$-subgraph $g$ of $G$, there will be no such an induced subgraph $Q$ of $G$ such that $V(Q) \supset V(g)$ and $Q$ is a quasi-clique.*

Proof. We prove this by contradiction. Let $Q$ be such an induced subgraph of $G$ and $|Q| = m$. Obviously $m > k$ and $V(Q) \subseteq V(g) \cup V^G_{vad}(g)$. Since $v$ is a critical vertex in $g$, $deg^g(v) = \lceil \gamma \cdot (k-1) \rceil$ and $\lceil \gamma \cdot (k-1) \rceil < \lceil \gamma \cdot k \rceil$. Furthermore, because $deg^Q(v) \leq deg^g(v) + extdeg^g(v)$ and $extdeg^g(v) = 0$, $deg^Q(v) < \lceil \gamma \cdot k \rceil \leq \lceil \gamma \cdot (m-1) \rceil$, which contradicts the assumption that $Q$ is a quasi-clique. Thus, there must exist no such an induced subgraph. □

Given a $k$-subgraph $g$ of a graph $G$, if $\exists\, u \in V(g)$ such that $deg^g(u) < \lceil \gamma \cdot (k-1) \rceil$ and $deg^g(u) + extdeg^g(u) < \lceil \gamma \cdot (k - 1 + extdeg^g(u)) \rceil$, then we call $u$ an *unpromising vertex* in $g$.

Lemma 5.6 (Subgraph Connectivity Pruning). *If a $k$-subgraph $g$ of $G$ contains an unpromising vertex $u$, there will be no induced subgraph $Q$ of $G$ such that $V(Q) \supset V(g)$ and $Q$ is a quasi-clique.*

Proof. Let $Q$ be such an induced subgraph and $|Q| = l$. Since $Q$ is a $\gamma$-quasi-clique, $V(Q) \subseteq V(g) \cup V^G_{vad}(g)$ holds. Let $V' = V(Q) \cap V^g_{ext}(u)$ and $|V'| = m$, then $l \geq m + |V(g)| = m + k$. Because $deg^g(u) < \lceil \gamma \cdot (k-1) \rceil$ and $deg^g(u) + extdeg^g(u) < \lceil \gamma \cdot (k - 1 + exdeg^g(u)) \rceil$, from Lemma 5.3 we get that $\forall i \in [0, exdeg^g(u)]$, $deg^g(u) + i < \lceil \gamma \cdot (k - 1 + i) \rceil$. Thus $deg^Q(u) = deg^g(u) + |V'| = deg^g(u) + m < \lceil \gamma \cdot (k - 1 + m) \rceil \leq \lceil \gamma \cdot (l-1) \rceil$, that is, $deg^Q(u) < \lceil \gamma \cdot (|Q| - 1) \rceil$. This contradicts the assumption that $Q$ is a $\gamma$-quasi-clique. □

According to Lemmas 5.5 and 5.6, if a subgraph contains a failed or unpromising vertex, it has no hope to form a quasi-clique in the future. Therefore, after getting the valid extensible candidate set $V^G_{vad}(g)$, once we inspect the existence of failed or unpromising vertices in $g$, then there is no hope to grow instance $g$ to generate quasi-cliques, and thus we can set $V^G_{vad}(g) = \phi$ and stop growing $g$.

### 5.4 The Algorithm Valid

By integrating the previous pruning techniques, we can devise Subalgorithm 1, Valid, to calculate the valid extensible candidate set for a subgraph $g$. For each vertex $u$ in the current instance $g$, we scan the graph $G$ in which $g$ resides to find the set $\bar{\mathcal{D}}(u)$ (line 06). Then we generate the set $\bar{\mathcal{I}}(u)$ from $\bar{\mathcal{D}}(u)$, obtain the extensible candidate set $T$ of $u$ (lines 07–08), and conjoin each discovered extensible candidate set to get the global extensible candidates $rs$ (line 09).

**Subalgorithm 1** Valid ($g$)

```
INPUT:
  (1) $g$: an instance subgraph.
OUTPUT:
  (1) $rs$: the set of valid extensible candidates with respect to g.
BEGIN
01. set $rs = V(G)$($G$ is the graph in which g resides);
02. For each vertex $u$ in $V(g)$
03.     If $rs$ is empty
04.         break;
05.     set $\bar{\mathcal{K}} = \bar{\mathcal{D}} = \bar{\mathcal{I}} = \phi$;
06.     $\bar{\mathcal{D}} = \{v | dis^G(u, v) = 1 \wedge (v \in V(g) \vee \mathcal{L}(v) \geq LAS(g))\}$;
07.     $\bar{\mathcal{I}} = \{v \mid \exists t \in \bar{\mathcal{D}}, dis^G(t, v) = 1\}$;
08.     $\bar{\mathcal{K}} = \{v | (v \in (\bar{\mathcal{D}} \cup \bar{\mathcal{I}})) \wedge (\mathcal{L}(v) \geq LAS(g))\}$;
09.     $rs = rs \cap \bar{\mathcal{K}}$;
10. Remove invalid extensible candidates from $rs$;
11. If there exists a failed or an unpromising vertex in $g$
12.     $rs = \phi$;
13. return $rs$;
END
```

Finally, we apply vertex connectivity pruning (line 10), failed-vertex pruning, and subgraph connectivity pruning (lines 11–12) to $rs$ to generate the final set of valid extensible candidates with respect to $g$.

### 5.5 Improvement of Valid

Although the pruning techniques proposed in this article are supposed to be very effective in enhancing the efficiency of quasi-clique enumeration, the operations corresponding to lines 02–11 in subalgorithm valid are rather costly, as we have to calculate the valid extensible candidates for each vertex in the current subgraph. When the subgraph is extremely large, these loop operations will be very expensive in both space and runtime, and would potentially become bottlenecks of the pattern discovery process. In this subsection we will introduce some methods in order to improve these costly operations.

As we defined earlier, $\bar{\mathcal{K}}^g(u)$ is the extensible vertex set of $u$ through diameter and combination pruning. Since different subgraphs would have different last vertex labels, $u$ will have different $\bar{\mathcal{K}}^g(u)$'s for different $g$'s. The following lemma shows the relationship between $\bar{\mathcal{K}}^g(u)$ and $\bar{\mathcal{K}}^{g'}(u)$, where $g'$ is the direct subgraph of $g$ (i.e., $g'$ is obtained by removing from $g$ that vertex having the largest label among vertices in $g$).

Lemma 5.7. *Assume $g'$ is the direct subgraph of $g$ and both are subgraphs of $G$, then $\forall u \in V(g'), \bar{\mathcal{K}}^g(u) \subseteq \bar{\mathcal{K}}^{g'}(u)$.*

Proof. First of all, we prove that $\forall u \in V(g'), \bar{\mathcal{D}}^{g'}(u) \supseteq \bar{\mathcal{D}}^g(u)$. Let $V(g) - V(g') = \{a_m\}$. Since $\bar{\mathcal{D}}^g(u) = \{v | dis^G(u, v) = 1 \wedge (v \in V(g) \vee \mathcal{L}(v) \geq LAS(g))\}$.

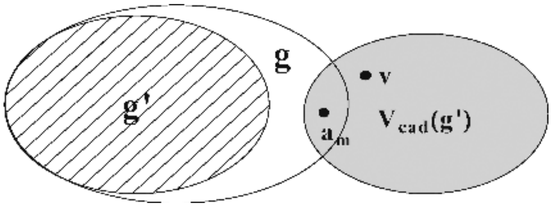


Fig. 7. Relationships of $v$, $a_m$, and $V_{cad}(g')$.

Let $Z_1^g(u) = \{v | dis^G(u, v) = 1, v \in V(g)\}$ and $Z_2^g(u) = \{v | dis^G(u, v) = 1, \mathcal{L}(v) \geq LAS(g)\}$. Accordingly, $\bar{\mathcal{D}}^g(u) = Z_1^g(u) \cup Z_2^g(u)$. As $LAS(g) \geq LAS(g')$, $Z_2^g(u) \subseteq Z_2^{g'}(u)$ holds.

If $dis^G(u, a_m) \neq 1$, $V(g) = V(g') \cup \{a_m\}$, then $Z_1^g(u) = Z_1^{g'}(u)$. In this case, $Z_1^g(u) \cup Z_2^g(u) \subseteq Z_1^{g'}(u) \cup Z_2^{g'}(u)$, that is, $\bar{\mathcal{D}}^g(u) \subseteq \bar{\mathcal{D}}^{g'}(u)$. Otherwise, if $dis^G(u, a_m) = 1$, then $a_m \in Z_1^g(u)$ and $Z_1^g(u) - Z_1^{g'}(u) = \{a_m\}$. Also, since $\mathcal{L}(a_m) = LAS(g) \geq LAS(g')$, $a_m \in Z_2^{g'}(u)$. Thus, $Z_1^g(u) \cup Z_2^g(u) = Z_1^{g'}(u) \cup \{a_m\} \cup Z_2^g(u) \subseteq Z_1^{g'}(u) \cup \{a_m\} \cup Z_2^{g'}(u) = Z_1^{g'}(u) \cup Z_2^{g'}(u)$, namely, $\bar{\mathcal{D}}^g(u) \subseteq \bar{\mathcal{D}}^{g'}(u)$. Therefore, $\forall u \in V(g')$, $\bar{\mathcal{D}}^{g'}(u) \supseteq \bar{\mathcal{D}}^g(u)$.

According to the definition of $\bar{\mathcal{I}}^g(u)$, we get that $\bar{\mathcal{I}}^g(u) \subseteq \bar{\mathcal{I}}^{g'}(u)$, $\bar{\mathcal{D}}^{g'}(u) \cup \bar{\mathcal{I}}^{g'}(u) \supseteq \bar{\mathcal{D}}^g(u) \cup \bar{\mathcal{I}}^g(u)$, that is, $\forall u \in V(g')$, $\bar{\mathcal{K}}^g(u) \subseteq \bar{\mathcal{K}}^{g'}(u)$. □

Assume $CF(g) = a_1a_2...a_m$, then $V_{cad}(g) = \cap_{i=1}^{m} \bar{\mathcal{K}}^g(a_i)$ and $V_{cad}(g') = \cap_{i=1}^{m-1} \bar{\mathcal{K}}^{g'}(a_i)$. From Lemma 5.7 we know that $\cap_{i=1}^{m-1} \bar{\mathcal{K}}^{g'}(a_i) \supseteq \cap_{i=1}^{m-1} \bar{\mathcal{K}}^g(a_i)$, thus $V_{cad}(g) \subseteq V_{cad}(g') \cap \bar{\mathcal{K}}^g(a_m)$. Therefore, if we know the information of $g'$'s extensible vertices, we can get a superset of the extensible vertices of $g$. In addition, after getting the extensible vertices, we should apply some more pruning techniques to get the valid extensible candidates $V_{vad}(g)$. But how can we get the valid extensible candidates of $g$ from those of $g'$?

Obviously, $g'$ does not have any failed vertex or unpromising vertex. Otherwise, it will not have a valid extensible candidate $a_m$ to generate a new subgraph $g$. Thus, we need only apply the vertex connectivity pruning method in order to get the valid extensible candidates. Let P(g) denote the vertex set which was pruned from the extensible vertex set of g by vertex connectivity pruning. In the following we will introduce two lemmas which will help us facilitate the computation of valid extensible candidates.

LEMMA 5.8. *Given a subgraph $g(CF() = a_1a_2...a_m)$ and its direct subgraph $g$ with $CR(g) = g'$, $\forall v \in P(g')$, if $v \in V_{cad}(g)$, then $v \in P(g)$.*

PROOF. Since $v \in P(g')$, $indeg^{g'}(v) < \lceil \gamma \cdot (m-2) \rceil$ and $indeg^{g'}(v) + exdeg^{g'}(v) < \lceil \gamma \cdot (m - 2 + exdeg^{g'}(v)) \rceil$ must hold. Moreover, as $a_m \in V_{cad}(g')$ and $V_{cad}(g) \subseteq V_{cad}(g')$, we have $exdeg^{g'}(v) \geq exdeg^g(v)$. The relationships of $v$, $a_m$, and $V_{cad}(g')$ are shown in Figure 7. We will prove the lemma in two cases.

In the first case, $v$ is not adjacent to $a_m$, so we have $indeg^{g'}(v) = indeg^g(v)$, $indeg^g(v) + exdeg^g(v) \leq indeg^{g'}(v) + exdeg^{g'}(v)$. As $indeg^{g'}(v) + exdeg^{g'}(v) < \lceil \gamma \cdot (m - 2 + exdeg^{g'}(v)) \rceil$ and $exdeg^g(v) \leq exdeg^{g'}(v)$, according to Lemma 5.3

we have $indeg^{g'}(v) + exdeg^{g}(v) < \lceil \gamma \cdot (m - 2 + exdeg^{g}(v)) \rceil$. Then $indeg^{g}(v) + exdeg^{g}(v) < \lceil \gamma \cdot (m - 2 + exdeg^{g}(v)) \rceil \leq \lceil \gamma \cdot (m - 1 + exdeg^{g}(v)) \rceil$ must hold.

In the second case, $v$ is adjacent to $a_m$, so we have $indeg^{g}(v) = indeg^{g'}(v) + 1$ and $exdeg^{g}(v) \leq exdeg^{g'}(v) - 1$. As $exdeg^{g}(v) + 1 \leq exdeg^{g'}(v)$ and $indeg^{g'}(v) + exdeg^{g'}(v) < \lceil \gamma \cdot (m - 2 + exdeg^{g'}(v)) \rceil$, from Lemma 5.3 we have $indeg^{g'}(v) + exdeg^{g}(v) + 1 \leq \lceil \gamma \cdot (m - 2 + exdeg^{g}(v) + 1) \rceil = \lceil \gamma \cdot (m - 1 + exdeg^{g}(v)) \rceil$, that is, $indeg^{g}(v) + exdeg^{g}(v) \leq \lceil \gamma \cdot (m - 1 + exdeg^{g}(v)) \rceil$.

In conclusion, in any case $indeg^{g}(v) + exdeg^{g}(v) \leq \lceil \gamma \cdot (m - 1 + exdeg^{g}(v)) \rceil$ must hold and $v$ should be pruned from $V_{cad}(g)$ based on the vertex connectivity pruning technique. Therefore $v \in P(g)$. □

LEMMA 5.9. *Given a subgraph $g$ with $CF(g) = a_1 a_2 ... a_m$, and its direct subgraph $g'$, $V_{vad}(g) \subseteq V_{vad}(g') \cap \bar{\mathcal{K}}^{g}(a_m)$.*

PROOF. For the sake of accomplishing this proof, we need to prove that $\forall v \in V_{vad}(g)$, $v \in V_{vad}(g') \cap \bar{\mathcal{K}}^{g}(a_m)$. Assume that $v \in V_{vad}(g)$, as $V_{vad}(g) = V_{cad}(g) - P(g)$, $v \in V_{cad}(g)$ and $v \notin P(g)$. From Lemma 5.8 we have $v \notin P(g')$. Otherwise, if $v \in P(g')$, since $v \in V_{cad}(g)$, we have $v \in P(g)$. This contradicts the fact that $v \notin P(g)$. Therefore, $v \notin P(g')$ and $v \in V_{cad}(g')$, $v \in V_{cad}(g') - P(g') = V_{vad}(g')$. In addition, $v \in V_{cad}(g)$ and $v \in \bar{\mathcal{K}}^{g}(a_m)$, thus $v \in V_{vad}(g') \cap \bar{\mathcal{K}}^{g}(a_m)$. In conclusion, we have that $\forall v \in V_{vad}(g)$, $v \in V_{vad}(g') \cap \bar{\mathcal{K}}^{g}(a_m)$, namely, $V_{vad}(g) \subseteq V_{vad}(g') \cap \bar{\mathcal{K}}^{g}(a_m)$. □

Based on Lemma 5.9, we can get a superset of $V_{vad}(g)$ from $V_{vad}(g') \cap \bar{\mathcal{K}}(a_m)$ on which we can then apply vertex connectivity pruning. Meanwhile, it is convenient for us to maintain $V_{vad}(g')$. Accordingly, based on Lemma 5.9 we can devise a more efficient algorithm, Subalgorithm Valid+, to replace Subalgorithm Valid. If $g$ has only one vertex $u$, then the set of valid extensible candidates $rs$ is equal to $\bar{\mathcal{K}}^{g}(u)$ (lines 16–18). Otherwise, assume $u$ and $g'$ are the last vertex and direct subgraph of $g$, respectively. Based on Lemma 5.9, we can get an extensible candidate set (line 20). Then we can apply pruning techniques like those used in Valid to further prune the search space (lines 21–23).

**Subalgorithm 2** VALID+ ($g$)

```
INPUT:
  (1) $g$: an instance subgraph.
OUTPUT:
  (1) $rs$: the set of valid extensible candidates with respect to g.
BEGIN
15. $u$ is the last vertex of $g$;
16. if $|g| = 1$;
17.     $rs = \bar{\mathcal{K}}(u)$;
18.     return $rs$;
19. $g'$ is the direct subgraph of $g$;
20. $rs = V_{vad}(g') \cap \bar{\mathcal{K}}(u)$;
21. Remove invalid extensible candidates from $rs$;
```

```
If there exists a failed or an unpromising vertex in $g$
    $rs = \phi$;
return rs;
END
```

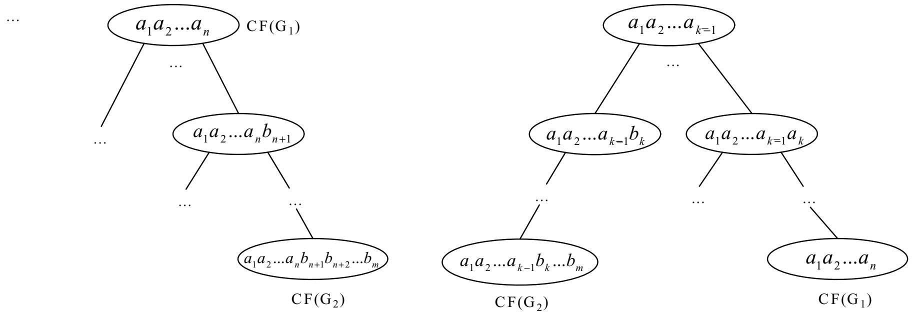
It is obvious from Subalgorithm Valid+ that we need to maintain $V_{vad}(g)$ for each node corresponding to an instance subgraph $g$ in the enumeration tree. It hence seems likely that these extensible candidate vertex sets may occupy a lot of space. We must remark here that, usually, maintaining $V_{vad}(g)$ for a given node with respect to $g$ has a very short lifecycle. In other words, we only need to maintain $V_{vad}(g)$ after the node corresponding to $g$ is generated and before all its descendants are enumerated. Thus, we usually do not need too much space to maintain the intermediate sets of extensible candidate vertex sets.

## 6. CLOSURE CHECKING SCHEME

By integrating the pruning techniques proposed in this article with the vice-frequent subgraph enumeration framework, we can discover the complete set of frequent quasi-cliques. However, how do we compute the set of closed quasi-cliques? A straightforward approach is to store all the frequent quasi-cliques that we have found, check the subquasi-clique relationships among the quasi-cliques, and remove the nonclosed ones. While inserting a frequent quasi-clique $q$ into the result set, we need to perform two types of checking. The first is *superclique detecting*, which checks whether there already exists an element $q'$ such that $q \sqsubset q'$ and $sup_s(q) \leq sup_s(q')$. If there exists such a $q'$, then $q$ is nonclosed and will not be inserted into the result set. The second is *subclique detecting*, which checks whether there exists any already-mined quasi-clique $q'$ such that $q' \sqsubset q$ and $sup_s(q) \geq sup_s(q')$. All such $q'$'s are nonclosed and should be removed. Apparently, this naïve approach is very costly. In the following, we introduce a more efficient closure checking scheme to facilitate the discovery of closed quasi-clique patterns.

Given two subgraphs $G_1$ and $G_2$ with canonical forms $CF(G_1) = a_1a_2...a_n$ and $CF(G_2)=b_1b_2...b_m$ (where $n<m$), respectively, if $CF(G_1) \sqsubset CF(G_2)$, there must exist $n$ integers $1 \leq i_1 < i_2 < ... < i_n \leq m$ such that $a_1=b_{i_1}$, $a_2=b_{i_2}$, ..., $a_n=b_{i_n}$. If $i_n=n$, the relationship of $CF(G_1)$ and $CF(G_2)$ in the enumeration tree can be as illustrated in Figure 8(a), that is, the node corresponding to $CF(G_1)$ is an ancestor of the node corresponding to $CF(G_2)$. Otherwise, let $k = min\{j \mid i_j \neq j\}$, then $CF(G_1)$ and $CF(G_2)$ have the same prefix $a_1a_2...a_{k-1}$ (if $k = 1$, the prefix is empty) and $a_k > b_k$ (as $a_k = b_{i_k}, i_k > k$, and $b_k < b_{i_k}$ hold). Thus $CF(G_1) > CF(G_2)$ holds, and this relationship in the enumeration tree is shown in Figure 8(b).

One strategy for speeding up pattern closure checking is to postpone the closure checking for $G_1$, as well as the insertion of $G_1$ into the result set (in case $G_1$ is closed) until all its descendants in the enumeration tree have been processed. In this way, $G_2$ must be discovered before $G_1$ for the first case, shown in Figure 8(a), as node $CF(G_2)$ is a descendant of $CF(G_1)$. In the second case, shown in Figure 8(b), it is evident that $G_2$ is also discovered before $G_1$ according



(a) descendant

(b) nondescendant

Fig. 8. Two cases of $CF(G_1) \sqsubset CF(G_2)$ in a vice-frequent subgraph enumeration tree.

to the DFS traversal strategy. In summary, if the current quasi-clique $G_1$ can be subsumed by another frequent closed $\gamma$-quasi-clique $G_2$ (i.e., $CF(G_1) \sqsubset CF(G_2)$ and $sup_s^D(G_1) \leq sup_s^D(G_2)$), then the insertion of $G_2$ into the result set will occur before the closure checking of $G_1$. Accordingly, when we check whether a frequent quasi-clique $q$ is closed, there is no need to perform subclique detection, as there does not exist any quasi-clique $q'$ in the result set such that $CF(q') \sqsubset CF(q)$ and $sup_s^D(q') \leq sup_s^D(q)$.

Although there is no need for subclique detecting, we still have to perform superclique detection. As shown in Figure 8, there are two cases for superclique detecting. In the first case, we need to check whether there exists a descendant quasi-clique of the current quasi-clique $G_1$ that can subsume $G_1$ in the enumeration tree. According to our strategy described before, we know that $G_1$ must be mined after all its descendants, and superclique detection in this case becomes relatively simple. After processing all the descendants of $G_1$, we let the recursive mining procedure return the maximum strict-support (denoted by $r$) of all frequent quasi-clique nodes under the subtree rooted with $CF(G_1)$ (if there does not exist any frequent quasi-clique, then it returns value zero). If $sup_s^D(G_1) \leq r$, we know that $G_1$ is nonclosed and will not insert $G_1$ into the result set. However, if $sup_s^D(G_1)>r$, we still need to check whether there exists any nondescendant superclique of $G_1$ that can subsume $G_1$ (i.e., the second case, shown in Figure 8(b)).

In order to accelerate the nondescendant superclique detecting process, we divide the elements in the result set into different groups according to their absolute strict-support.[2] In each group, we first order the elements by size of quasi-clique in descending order, and among quasi-cliques with the same size in the same group, we then order them by canonical form in ascending order. The idea is illustrated in Figure 9. In the support vector, we store the strict-support and a pointer that links the list of frequent quasi-cliques having the same strict-support. In Figure 9, we omit the canonical forms of frequent quasi-cliques, and depict only their size in the list. This data structure can

[2] We do not need to maintain the groups corresponding to strict-supports which are smaller than *min_sup*. Also, if the maximum support of the closed quasi-cliques is very large, we can use a *binning* technique [Wang et al. 2006a] to reduce the number of groups.

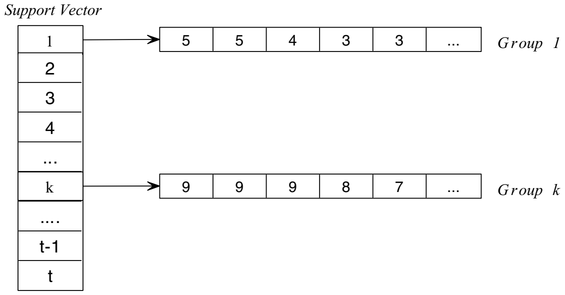


Fig. 9. Data structure for result set.

facilitate closure checking effectively. For example, when we try to perform nondescendant superclique detection for a frequent 8-quasi-clique $q$ which has a strict-support $k$, we need only check those elements in the groups whose corresponding strict-supports are no smaller than $k$. While checking group $k$, we only need to check whether the first three elements in this group are proper superquasi-cliques of $q$. In addition, once we find that the canonical form of the current 9-quasi-clique is greater than $CF(q)$, we can stop the comparison of the quasi-cliques of size 9 in group $k$ and guarantee that there exists no 9-quasi-clique of $q$ in group $k$ that can subsume $q$. We then continue to check the group $(k + 1)$. After checking group $t$, if we still cannot find any superquasi-clique of $q$ that has no smaller strict-support than $q$, then we can safely say $q$ is closed and insert it into the group $k$ according to the corresponding ordering scheme.

## 7. THE COCAIN* ALGORITHM

By introducing the vice-frequent enumeration framework and structural redundancy pruning technique, we can discover the complete set of coherent frequent quasi-clique patterns efficiently. With the help of several advanced pruning techniques, we can prune futile and unpromising search-space effectively. Furthermore, with the proposal of the closure checking scheme, the closed quasi-clique patterns could be mined. Therefore, by integrating them we get the final solution, Cocain*, for discovering the complete set of coherent frequent quasi-clique patterns.

Before running Cocain* as shown in Algorithm 1, we first compute the set of vice-frequent vertex labels and their corresponding instances by scanning the original database, and remove from the graph database those vertices with nonvice-frequent vertex labels. This procedure can reduce the size of input graphs significantly, especially when $min\_sup$ is high. After this preprocessing, we use Cocain* to mine the complete set of frequent closed coherent quasi-cliques. For the current prefix vice-frequent subgraph $g$, we first use Subalgorithm Valid+ to get the set of valid extensible candidates $V_{vad}$ for each instance of $g$ (lines 26–27), from which we can further calculate the vice-frequent extensible labels (line 28). For each vice-frequent extensible label, we invoke Cocain* to discover descendants of $g$ (lines 30–32). After all recursive invocations have returned, we can use the closure checking scheme to determine whether to

**Algorithm 1** Cocain*($D$, $CF(g)$,$INS(g)$,$min\_sup$,$\gamma$)

```
INPUT:
    (1) $D$: the input graph database,
    (2) $CF(g)$: the canonical form of g,
    (3) $INS(g)$: the set of instances of g in the database D,
    (4) $min\_sup$: the minimum support threshold,
    (5) $\gamma$: the edge density coefficient.
OUTPUT:
    (1) $rs$: the set of frequent closed $\gamma$-quasicliques,
    (2) $max$: the maximum strict-support of all descendant quasi-cliques of $g$.
BEGIN
25. glbsup=0, rv=0;
26. For each instance $ins \in INS(g)$
27.     $V_{vad}(ins) =$ VALID+$(ins)$;
28. Calculate vice-frequent valid candidate label set $VEX(g)$ according to each $V_{vad}(ins)$;
29. Sort the labels in $VEX(g)$ in a certain order;
30. For each label $l \in VEX(g)$
31.     $rv = Cocain{*}(D, CF(g) \diamond l, INS(g \diamond l), min\_sup, \gamma)$;
32.     $glbsup = max\{glbsup, rv\}$;
33. If $(sup_s(g) \geq min\_sup)$ and $(sup_s(g) > glbsup)$
34.     Insert $CF(g)$ into $RS$ if $g$ passes the nondescendant superclique detecting;
35. return $max\{sup_s(g), glbsup\}$;
END
```

insert $g$ into the final result set according to the strict-support of $g$ and the returned values of the recursive invocations (lines 33–34).

## 8. THE OUT-OF-CORE SOLUTION

Like most existing studies, all of our previous discussions are based on the assumption that either the whole database or at least its majority can fit into main memory, the number of possible labels in databases is not large, and the major data structures are designed for being held in main memory, too. Are there any real-life applications that need to mine large graph databases? The answer is yes. For example, in data integration of XML documents or mining semantic web, it is often required to find the common substructures from a large collection of XML documents. It is easy to see applications with collections of millions of XML documents. As another example, stock market data can also be converted to graphs with thousands of vertices. These large databases often cannot be held in main memory. Thus, it is necessary to devise out-of-core solutions for these large databases.

Why is mining large disk-based graph databases so challenging? In most previous studies, the major data structures (the adjacency list or adjacency matrix) are designed for being held in main memory. Moreover, most of the previous methods are based on efficient random accesses to elements (e.g., edges and their adjacent edges) in graphs. However, if the adjacency list or adjacency matrix representations cannot be held in main memory, the random accesses to them become very expensive. For disk-based data without any index, random accesses can be extremely costly, the computation becoming I/O bounded. In this section, we will devise some effective index structures to facilitate scalable mining of coherent closed quasi-cliques from disk-based large graph databases.

### 8.1 Critical Operations

Algorithm Cocain*, shown in Algorithm 1, is efficient when the database can be held in main memory, since this algorithm focuses on effective heuristics to prune the search space. However, it may encounter difficulties while mining large databases. The major overhead involved is that Cocain* has to randomly access elements (e.g., edges and vertices) in the graph database, as well as the projections of the graph database, many times. For databases that cannot be held in main memory, the mining becomes I/O bounded and thus costly.

Random access to elements in graph databases and isomorphism checking are not unique to Cocain*. In fact, this is a common problem for many graph pattern mining algorithms, such as FSG [Kuramochi and Karypis 2001], gSpan [Yan and Han 2002], and so on. In our solution for mining coherent closed quasi-cliques, the major data access operations are as follows.

**OP1**: *Vertex label support checking*, which calculates the support of a vertex label in the graph database;

**OP2**: *Label-host graph checking*, which finds for a vertex label $l$ those graphs in the databases where $l$ appears and also its relative positions in these graphs, and

**OP3**: *Adjacent vertex checking*, which finds all adjacent vertices for a vertex $v$ in the graphs where $v$ resides.

Each of the aforementioned operations is usually called numerous times during the pattern discovery process. Without appropriate indices, each of these operations may require scanning the database or its projections. If the database or its project cannot fit into main memory, the scanning and checking may result in a great deal of I/O overhead.

*Can we design some effective index structures so that the related information can be maintained and all the aforementioned operations can be executed efficiently using the indices only, thus without scanning the original database and checking the graphs?* This motivates the design of the index structures described in the next section.

### 8.2 The Data Structures

8.2.1 *Adjacency Information.* The first problem is how to store the graph databases, especially the adjacency information among different vertices. We adopt the adjacency list as our storage data structure. Figure 10 shows the adjacency list for our running example in Figure 5.

For each vertex, we store all its neighbors' indices in the current graph. For instance, the vertex $u_5$ in $G_1$ in Figure 5 has two neighbors, $u_2$ and $u_4$, whose

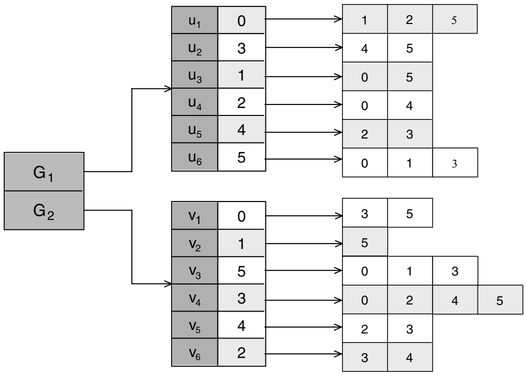


Fig. 10. Adjacency data structure.

indices are 2 and 3, respectively. Therefore, in the adjacent data structure, the vertex $u_5$ of $G_1$ has two elements, 2 and 3.

Apparently, if the graph database has $m$ edges, the space complexity of this adjacent data structure is $2m$. There exists redundancy in the structure we adopted, as each edge is stored twice. For example, from either the adjacency list of $u_2$ or the adjacency list of $u_5$ in $G_1$ of Figure 10, we know that $u_2$ is adjacent to $u_5$. Indeed, this is redundant, however, it will speed-up our pattern discovery process. For finding all the neighbors of a vertex $v$, we only need to check the adjacency list of $v$. Without this redundancy, we have to access more than one adjacency list. This redundancy will make our pattern discovery more timely and effective. Moreover, when we process large graph databases, what we are most concerned with is the runtime, and not the disk space which the indices occupy.

Consequently, with the help of the preceding adjacent data structure, the critical operation **OP3**, adjacent vertex checking, can be accomplished efficiently.

8.2.2 *Label Index.* Except for the critical operation **OP3**, we have to deal with another two critical operations efficiently. Thus, we design a label index data structure to facilitate these two operations. Figure 11 shows the label index data structure built for our running example in Figure 5.

The label index data structure has three layers. The first stores the list of sorted vertex labels. For each label, there is a link pointing to a list of graph-ids in the second layer, in which this label appears. For each graph in the second layer, it points to a 2-tuple list which keeps the starting and end positions of the corresponding vertex label. For instance, the vertex label $c$ appears in $G_1$ and $G_2$ in our running example. Hence, it points to a list containing two graph-ids $G_1$ and $G_2$. Meanwhile, in $G_2$ there are two vertices, $v_4$ and $v_5$, each having a vertex label of $c$, and their position indices are 3 and 4, respectively. Therefore, the 2-tuple is $(3, 4)$.

As we can see, for each vertex label $l$, if its rough-support is $Sup(l)$, its corresponding index will occupy $3Sup(l) + 1$ cells (note here that a cell may correspond to one space unit for storing an integer data type or a long integer

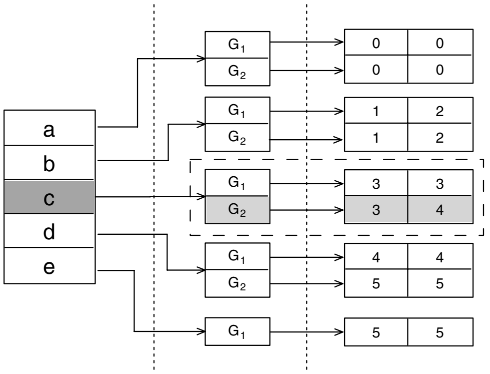


Fig. 11. Label index data structure.

data type, depending on the concrete implementation). Therefore, the space complexity of the label index data structure is $\sum_{\forall l \in L(G)}(3Sup(l) + 1)$.

Once the label index data structure is constructed, it will greatly facilitate the operations **OP1** and **OP2**. For **OP1** (i.e., vertex label support checking), we need only get the length of the corresponding graph-ids list in the second layer. And we can also maintain this information permanently after we construct the index. For **OP2** (i.e., label-host graph checking), we can get all the graph-ids from the corresponding list and also their corresponding starting and end positions in each graph.

### 8.3 Construction and Usage of the Index Structures

Given a specific minimum support threshold *min_sup*, we can scan the original graph database and remove those vertices with infrequent vertex labels. Then we can construct the index structures, adjacency data structure, and label index structure, and keep them in the disk for future use. For all minimum support thresholds no smaller than *min_sup*, we can always use these index structures constructed with a minimum support *min_sup*. With the label index structure, before running Cocain* we can directly get the frequency of each vertex label, and then get the frequent vertex labels and their corresponding instances. For adjacency information of the vertices, it is very easy to use the adjacency data structure.

The storage of the index structures is relatively flexible. If the graph database is small enough, all the index structures can be held in main memory. On the other hand, if the graph database is too large for the index structures to be held in main memory, some layers of the index structures can be stored on disk. The adjacency information is the most detailed and can be put on disk first. If the main memory is still too small to hold the label index structure, part of it can be switched to disk, too. In the extreme case, the entire set of vertex labels can be held on disk and a B$^+$-tree or hash index can be built on them.

In Wang et al. [2004], the authors also proposed an index structure called ADI in order to accelerate frequent subgraph mining from disk-based graph databases. However, differently from the ADI structure, we create indices for the vertex labels in the graph database, rather than for the edges, as the critical operations in Cocain* and gSpan are different. In addition, from the preceding data structure design we know that the efficiency of the index structures is mainly affected by the number of graphs in the dataset and the number of distinct frequent vertex labels. We will evaluate the construction time of the index structures and their disk space consumption empirically in Section 9.

## 9. DISCUSSIONS AND EMPIRICAL RESULTS

We conducted an extensive performance study to evaluate various aspects of the algorithm. We implemented the algorithm in C++, and all the experiments were performed on a PC running Fedora Core 4 Linux and with a 1.8GHz AMD Sempron CPU and 1GB of main memory installed. In the following we use *min_sup* and $\gamma$ to denote the relative strict-support threshold and edge density parameter, respectively. For simplicity, we use "$min\_sup - \gamma$" to indicate the pair of values of *min_sup* and $\gamma$ that we used in the experiments. For instance, "100%-1.0" indicates that *min_sup* and $\gamma$ are set at 100% and 1.0, respectively.

### 9.1 Databases

In the experiments we used both real and synthetic databases to evaluate the algorithm. Next we describe these databases in detail.

9.1.1 *Real Databases.* The real databases we used in our testing consist of a US stock market series database, the KEGG database (abbreviated for Kyoto Encyclopedia of Genes and Genomes database), and the yeast microarray database.

- —*Stock market databases*. According to Boginski et al. [2004] and Wang et al. [2006b], stock market data with respect to a certain period of time can be converted to a graph based on the cross-relationship of price fluctuations. We used the same method as Wang et al. [2006b] to generate the stock market databases, each of which was generated according to a certain correlation coefficient and comprises 11 graphs. The characteristics of these databases are depicted in Table II. Note that "*x-stock*" means this database was generated according to a correlation coefficient "*x*". As we can see, all these stock market databases except *0.99-stock* are dense and large, with thousands of vertices and hundreds of thousands of edges.
- —*KEGG databases*. The second real database is the *KEGG* database, which can be downloaded from http://www.genome.jp/kegg/. It contains 129 pathways, each specifying one graph object with the entry elements as its nodes and the relation and reaction elements as its edges. The relation and reaction elements indicate the connection patterns of rectangles (gene products) and of circles (chemical compounds), respectively. In the experiments, we only considered the relations among genes, and ignored the reaction elements. In this way, we got 129 input graphs, each of which, on average, containing 90 vertices and 104 edges (please refer to Table II to get more statistical information).

Table II. Stock Market Database Characteristics

| Database | # Graphs | # Vertices | # Distinct Labels | # Edges |
|---|---|---|---|---|
| *0.99-Stock* | 11 | 439 | 143 | 318 |
| *0.96-Stock* | 11 | 12643 | 3511 | 88310 |
| *0.95-Stock* | 11 | 18517 | 3998 | 220822 |
| *0.94-Stock* | 11 | 24002 | 4952 | 440090 |
| *0.93-Stock* | 11 | 28807 | 5363 | 754696 |
| *0.92-Stock* | 11 | 32993 | 5648 | 1167722 |
| *0.91-Stock* | 11 | 36647 | 5849 | 1675918 |
| *0.90-Stock* | 11 | 39998 | 6018 | 2274223 |
| *KEGG* | 129 | 11629 | 5720 | 13437 |
| *Yeast* | 39 | 88093 | 6349 | 940289 |

—*Yeast microarray databases*. The third real database we used is comprised of the coexpression networks derived from 39 sets of yeast microarray data provided by the authors of Hu et al. [2005], each consisting of the expression profiles of 6,661 genes in at least 8 experiments. For each set of yeast microarray data, we constructed an input relational graph where two genes are connected if they show strong similarity in their expression patterns as measured by Pearson's correlation. The statistical information about this database is also shown in Table II.

9.1.2 *Synthetic Databases.* Furthermore, we also used a synthetic data generator provided by the authors of Kuramochi and Karypis [2001] to generate the synthetic databases. The generator takes the following six parameters as input (for more details, please refer to Kuramochi and Karypis [2001]):
$D$: total number of input graphs
$E$: number of edge labels used
$V$: number of vertex labels used
$I$: average size of frequent subgraphs
$L$: number of potentially frequent subgraphs
$T$: average size of input graphs

Since we do not consider the difference of edge labels, the parameter $E$, namely, the number of edge labels, will not affect the algorithm's performance. In the experiments, we set a fixed value of 200 for $E$, that is, $E = 200$ (in the experiments we just randomly selected a value for $E$, and we must note that it has no impact on the algorithm performance). For the other parameters, we set $V = 50$, $I = 50$, and $L = 1{,}000$ as the default values if not explicitly stated. The value of $L$ is much bigger than $L = 200$ that was used in the experiments of Kuramochi and Karypis [2001] and Wang et al. [2004]. In the following we use $D100kT20$ to denote a synthetic database that contains $100k$ graphs with average size 20, 200 edge labels, and 1,000 potentially frequent subgraphs with average size 50.

## 9.2 Efficiency Comparison of Cocain* and Cocain

We first conducted some experiments to evaluate the newly proposed approach, Cocain*, in comparison with Cocain. The result is shown in Figure 12. In the

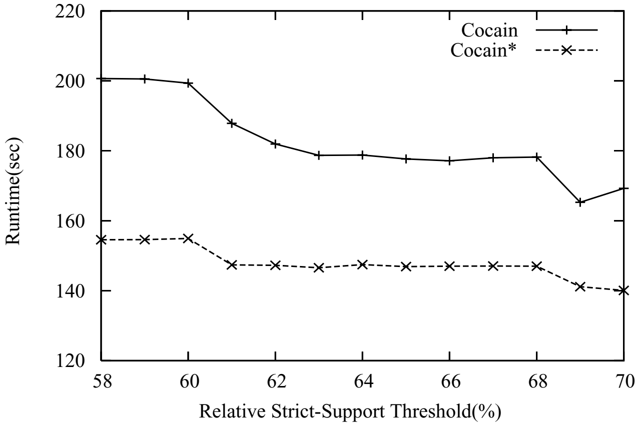


Fig. 12. Efficiency comparison of Cocain and Cocain*.

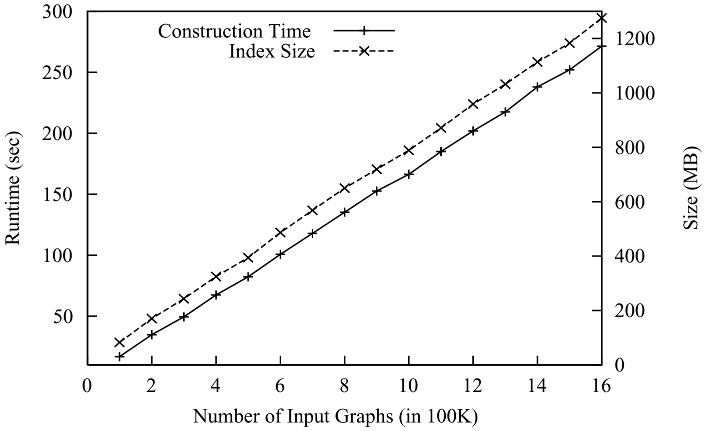


Fig. 13. Sensitivity analysis of the index structures with respect to base size.

experiments, we used the synthetic dataset $D1000kT40$, which contains one million input graphs. From Figure 12 we see that Cocain* is more efficient than Cocain, especially when the support is not very high. This illustrates that the newly proposed techniques are effective in improving the algorithm efficiency.

### 9.3 Sensitivity Analysis of the Index Data Structures

Meanwhile, we also evaluated the efficiency of the index structures proposed in Section 8 by testing their construction time and space consumption in the disk. Our purpose here is to identify which parameters have a big impact on the efficiency of the index structures. In the experiments we fixed the minimum relative strict-support at 1%. Figure 13 shows the results on the synthetic datasets $T10I50V50$ with a base size ranging from $100K$ to $1600K$. The left $y$ axis shows the runtime in seconds to construct the index structures, and the right $y$ axis shows the space in megabytes consumed by the index structures. From the

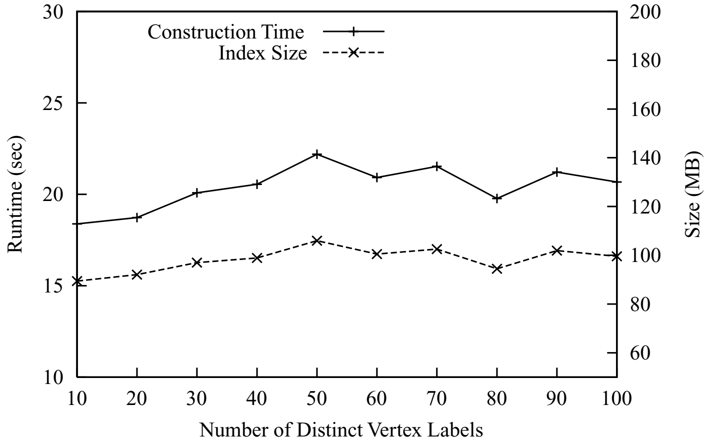


Fig. 14. Sensitivity analysis of the index structures with respect to number of distinct vertex labels.

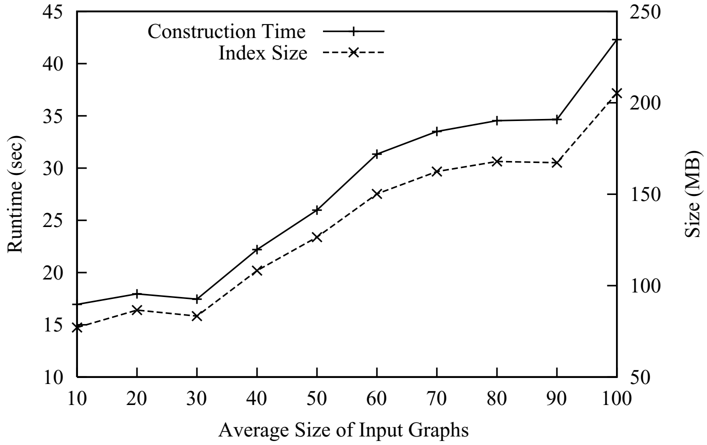
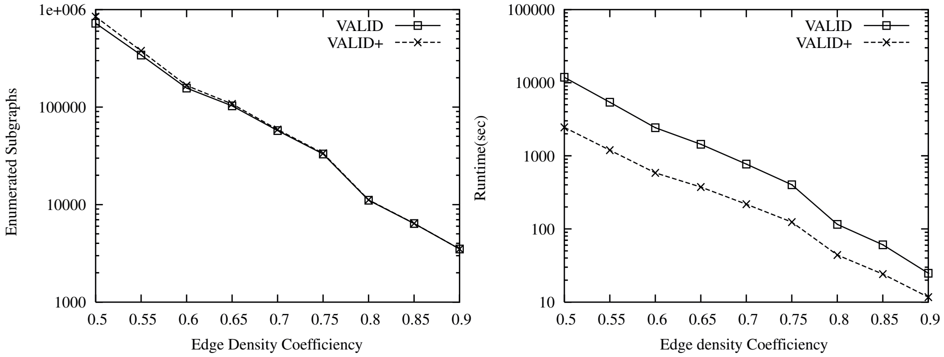


Fig. 15. Sensitivity analysis of the index structures with respect to average size of input graphs.

figure, we see that both the construction time and space consumed by the index structures have a linear relationship with the base size. Figure 14 shows the results on synthetic datasets $D100kT40I50$ with the number of distinct vertex labels ranging from 10 to 100. We see that the number of distinct vertex labels has little impact on both the construction time and space consumed by the index structures. Similarly, Figure 15 shows the results on synthetic datasets $D100kI50V50$ with the average size of input graphs ranging from 10 to 100. It is obvious that the average size of input graphs has a relatively big impact on the efficiency of the index data structures.

### 9.4 Comparison of Valid and Valid+

As we mentioned in Section 5.5, since $V_{vad}(g) \subseteq V_{vad}(g') \cap \bar{\mathcal{K}}^g(a_m)$, the adoption of Valid+ in Cocain* may cause more subgraphs to be enumerated in



(a) enumerated subgraphs (b) runtime

Fig. 16. Comparison of Valid and Valid+ (0.95-stock, $min\_sup = 60\%$).

the discovery process. Can we still benefit from Cocain*? Figure 16 depicts the evaluation results. In our test, we chose the 0.95-stock database and set $min\_sup = 60\%$. As shown in Figure 16(a), Valid+ only enumerates marginally more subgraphs than Valid. But Figure 16(b) shows that Valid+ costs much less in terms of time than Valid. This illustrates that Valid+ is more effective and efficient than Valid.

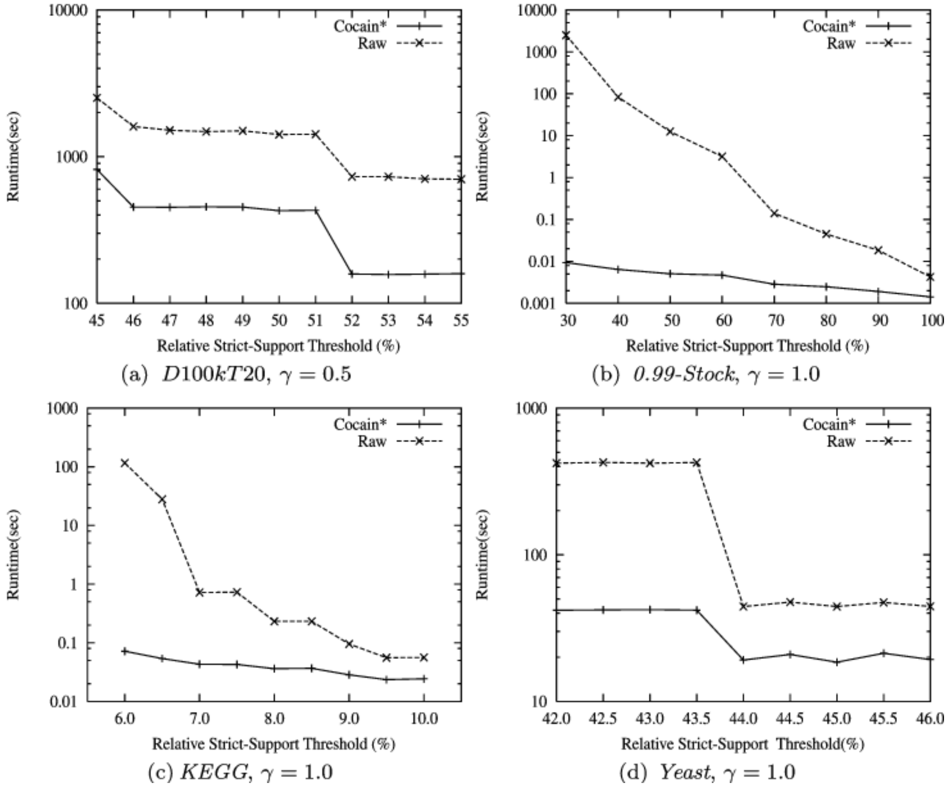
### 9.5 Efficiency Test

We also implemented one baseline algorithm, Raw, which excludes all the pruning techniques. By comparing the runtime efficiency of Cocain* and Raw using both synthetic and real databases, we can get an idea about the effectiveness of the pruning techniques proposed in this article.

Figures 17(a)–(d) show the runtime comparison between Cocain* and Raw with databases *D100kT20*, *0.99-Stock*, *KEGG*, and *Yeast*, respectively. From Figure 17 we see that Cocain* scales better in terms of support threshold, and is also much faster than Raw with all four databases. For example, for database *0.99-Stock*, Cocain* only takes 0.01 seconds to finish, while Raw takes about 2,511 seconds with $\gamma = 1.0$ and $min\_sup = 30\%$. The high performance of Cocain* in comparison with Raw also demonstrates that the pruning techniques proposed for Cocain* are extremely effective.

We then tested the runtime efficiency of Cocain* and Raw by fixing $min\_sup$ at a certain value and varying the value of $\gamma$ from 0.5 to 1.0. Figures 18(a) and (b) show the results for synthetic database $D100kT20$ with $min\_sup = 1\%$, and real database *0.99-Stock* with $min\_sup = 40\%$, respectively. We can see that Cocain* is always much faster than Raw for different values of $\gamma$. For example, for database *0.99-Stock*, Cocain* is several orders of magnitude faster than Raw with $min\_sup = 40\%$ and $\gamma$=0.7. The results in Figure 18 demonstrate that Cocain* is very efficient, even with a relatively low value for $\gamma$.

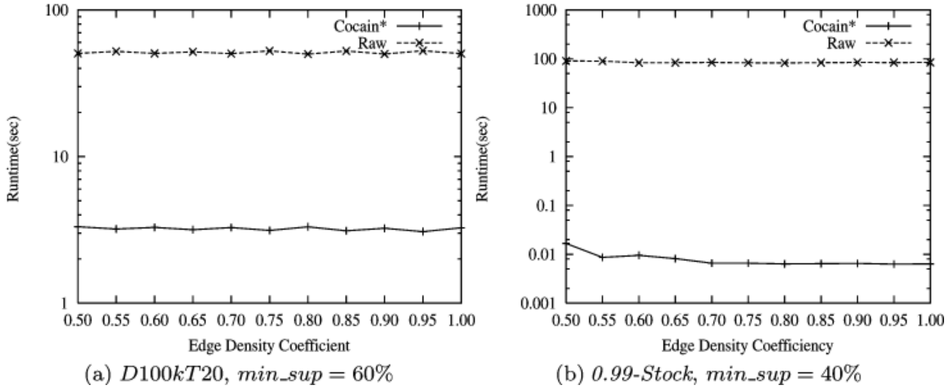
Moreover, in the efficiency test we oberserved that we cannot use the huge dense graph databases (e.g., $D100k20T$ rather than $D1000kT20$ and 0.99-stock rather than 0.90-stock) or a very low value for $min\_sup$. This is because Raw is too slow and takes too much time, even with a very small or sparse database.



(a) $D100kT20$, $\gamma = 0.5$ (b) *0.99-Stock*, $\gamma = 1.0$

(c) $KEGG$, $\gamma = 1.0$ (d) *Yeast*, $\gamma = 1.0$

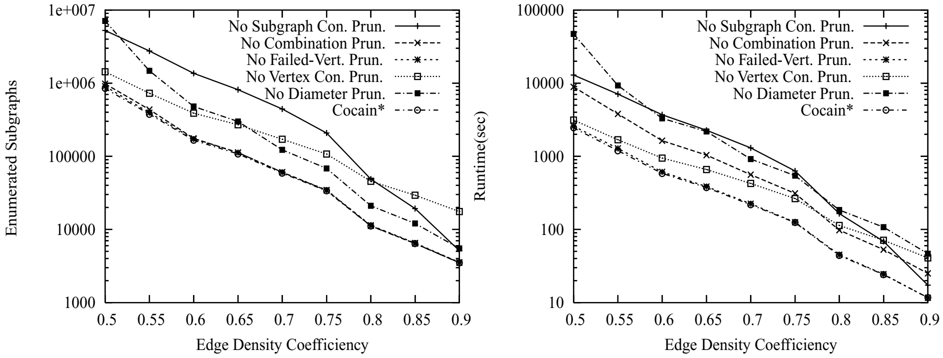
Fig. 17. Efficiency comparison (varying *min_sup*).



(a) $D100kT20$, $min\_sup = 60\%$ (b) *0.99-Stock*, $min\_sup = 40\%$

Fig. 18. Efficiency comparison (varying $\gamma$).

For example, even with the 0.99-stock and $min\_sup = 30\%$, Raw takes 2,511 seconds while Cocain* takes only about 0.01 seconds. When using 0.90-stock or a lower value for *min_sup*, Raw will take millions of seconds, which is a too long time. In order to get the testing results of Raw in a relatively short period of time, we have to use small sparse databases and low values for *min_sup*.



(a) enumerated subgraphs (b) time cost

Fig. 19. Comparison of pruning techniques (0.95-stock, *min_sup* = 60%).

### 9.6 Comparison of Different Pruning Techniques

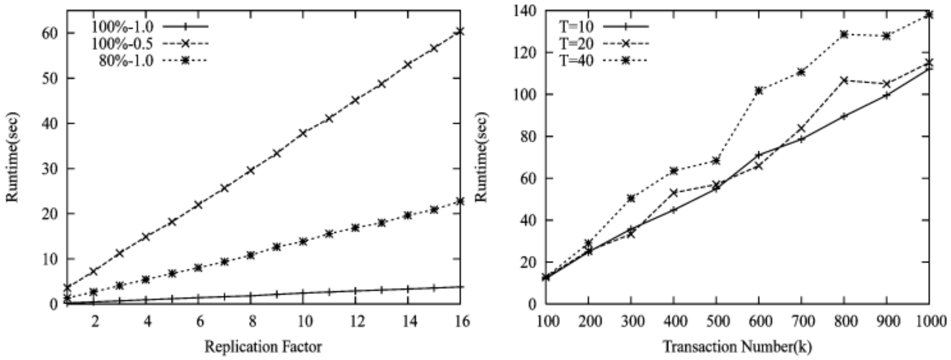
We proposed several pruning techniques in this study, including diameter pruning, combination pruning, vertex connectivity pruning, failed-vertex pruning, and subgraph connectivity pruning. In this section, we discuss and compare their effectiveness. Figure 19 shows the runtime and number of enumerated subgraphs in various conditions with combinations of different pruning techniques. For instance, the caption "No Vertex Con. Prun." denotes a variant of Cocain* without applying vertex connectivity pruning. From the results shown in Figure 19(a) we can observe that diameter pruning, vertex connectivity pruning, and subgraph connectivity pruning are rather effective in pruning unpromising search-spaces and play key roles in our pruning strategy. Based on the results in Figure 19(b), we observe that each pruning technique we proposed is effective while failed, vertex-pruning, diameter pruning, and subgraph connectivity pruning play more important roles in improving the efficiency of the algorithm.

### 9.7 Scalability Test

Meanwhile, a comprehensive scalability study was conducted in terms of base size on both real and synthetic databases. We first replicated the 0.95-stock database from 2 to 16 times and ran Cocain* with three different combined parameter values: 100%-1.0, 100%-0.5, and 80%-1.0. As shown in Figure 20(a), it is evident that Cocain* shows linear scalability in runtime against the number of input graphs in database. For the synthetic database, we set $min\_sup = 10\%$, $\gamma = 0.5$, and varied parameter $D$ from $100k$ to $1{,}000k$, parameter $T$ from 10 to 40, and got the runtime of Cocain* as shown in Figure 20(b). Similarly, Cocain* also shows good scalability in runtime against base size for synthetic databases.

### 9.8 Coherent Closed Quasi-Clique Examples

By applying Cocain* to stock market and KEGG databases, we found a lot of nontrivial coherent closed quasi-cliques. Figure 21 shows the distribution



(a) *0.95-stock* (b) synthetic databases (*min_sup*=70%,$\gamma$=0.5)

Fig. 20. Scalability on database size.

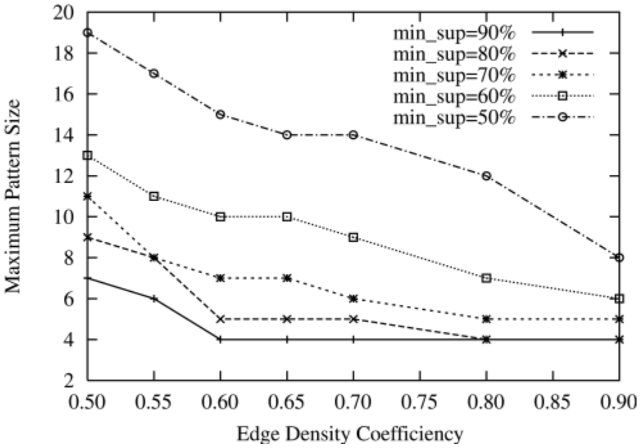
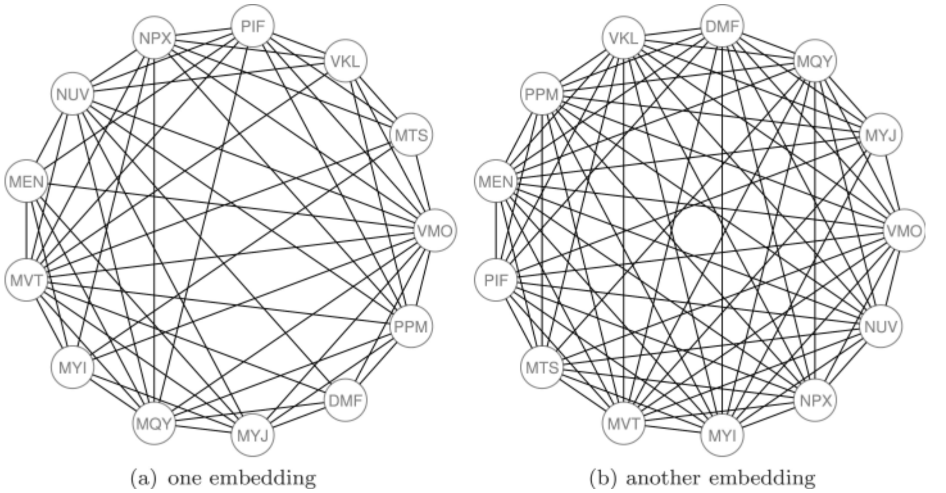


Fig. 21. Maximum pattern-size distribution (*0.95-stock*).

of the maximum closed quasi-clique size with respect to edge density coefficient $\gamma$ for 0.95-stock by fixing *min_sup* at 90%, 80%, 70%, 60%, and 50%, respectively. For example, by setting $\gamma = 0.5$ and *min_sup* $= 60\%$, Cocain* discovered 39 frequent closed quasi-cliques of size 13. Figure 22 shows two embeddings of one maximum quasi-clique of size 13, which corresponds to the set of stock index names DMF, MEN, MQY, MTS, MVT, MYI, MYJ, NPX, NUV, PIF, PPM, VKL, and VMO. At the same time, we plotted their price fluctuations in one of the 500-day periods, as shown in Figure 23. Obviously, in these 500 days, these 13 stocks' prices evolved synchronously. This illustrates that the closed quasi-cliques discovered by Cocain* from the US stock market data are very helpful in revealing the internal structure and providing valuable insights.

For the KEGG database, we also found some interesting coherent subgraphs. For example, by setting *min_sup* $= 1\%$ and $\gamma = 0.5$, we got a total of 1,759 frequent coherent closed quasi-cliques, among which the maximum quasi-clique contains 11 highly correlated genes: *ec:1.1.1.222*, *ec:1.1.1.237*,



(a) one embedding (b) another embedding

Fig. 22. Two example embeddings of one maximum closed quasi-clique of size 13 (*0.95-stock*, $\gamma = 0.5$, $min\_sup = 60\%$).

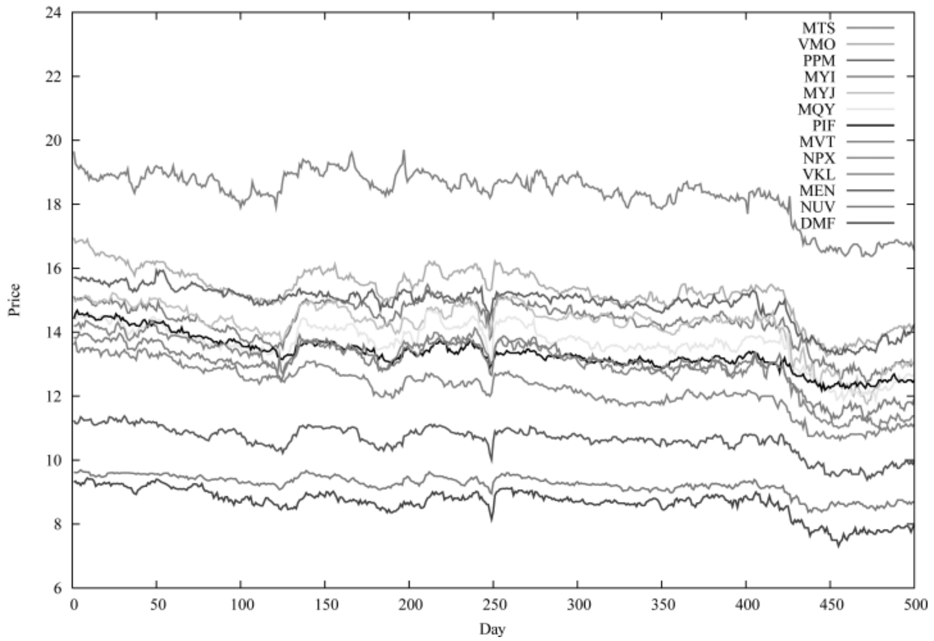
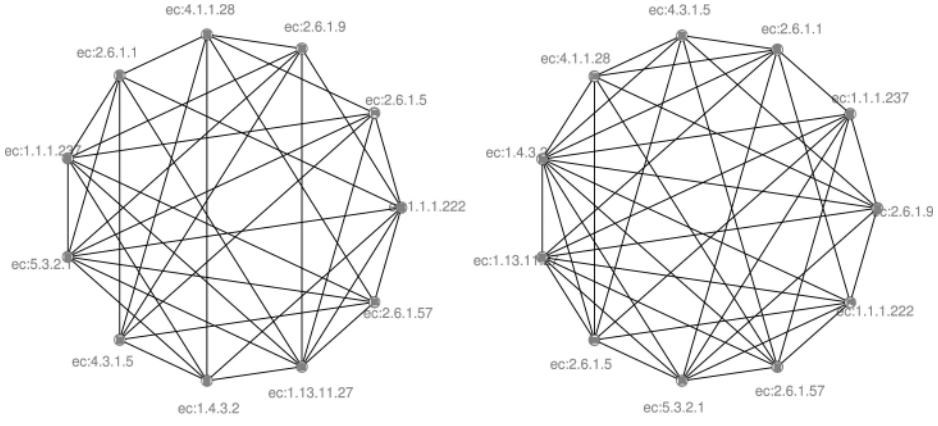


Fig. 23. Price fluctuation of the 13 stocks in one maximum closed quasi-clique of size 13.

*ec:1.4.3.2*, *ec:1.13.11.27*, *ec:2.6.1.1*, *ec:2.6.1.5*, *ec:2.6.1.9*, *ec:2.6.1.57*, *ec:4.1.1.28*, *ec:4.3.1.5*, and *ec:5.3.2.1*. This maximum quasi-clique was strictly supported by pathways *map00350* and *map00360*. Figure 24 shows the two corresponding embeddings.



(a) an embedding in *map00350* (b) an embedding in*map00360*

Fig. 24. Two example embeddings of the maximum closed quasi-clique of size 11 (*KEGG*, $\gamma = 0.5$, $min\_sup = 1\%$).

## 10. CONCLUSION

In this article we devised a novel algorithm, Cocain*, to mine frequent closed coherent quasi-cliques from large dense graph databases. We introduced a simple canonical form in order to uniquely represent a quasi-clique pattern, proposed several effective search-space pruning techniques by exploring some nice properties of quasi-clique patterns, diameter pruning, combination pruning, vertex connectivity pruning, failed-vertex pruning, and subgraph connectivity pruning, which can be used to accelerate the mining process, and also designed an efficient pattern closure checking scheme to speed-up the discovery of closed quasi-cliques. Finally, in order for Cocain* to scale to extremely large graph databases, we also proposed two effective index structures to support the out-of-core solution. An extensive performance study with both real and synthetic databases has demonstrated that the pruning techniques we proposed in this article are very effective, and that Cocain* is very efficient and scalable. Besides applications to highly correlated stock discovery and function prediction for uncharacterized genes, in the future we plan to explore potential applications of closed coherent quasi-clique mining in clustering/classifying graph-structured databases.

### ACKNOWLEDGMENTS

The authors would like to thank Vladimir L. Boginski, Panos M. Pardalos, and Sergiy Butenko for providing us the US stock market database, Jasmine Zhou and Haiyan Hu for sending us the yeast microarray database, Michihiro Kuramochi for sending us the KEGG database, and Beng Chin Ooi, Anthony K. H. Tung, and Xifeng Yan for their kind help. The authors are also grateful to editor Richard T. Snodgrass and Z. Meral Ożsoyoglu for their prompt and expert handling of this article, and the anonymous referees for their thorough and useful comments which helped much in improving this article.

## REFERENCES

Abello, J., Resende, M. G., and Sudarsky, S. 2002. Massive quasi-clique detection. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN)* (Cancun, Mexico). 598–612.

Agrawal, R., Imielinski, T., and Swami, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)* (Washington, D.C.). 207–216.

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)* (Santiago, Chile). 487–499.

Agrawal, R. and Srikant, R. 1995. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE)* (Taipei, Taiwan). 3–14.

Boginski, V., Butenko, S., and Pardalos, P. M. 2004. On structural properties of the market graph. In *Innovations in Financial and Economic Networks*, A. Nagurney ed. Edward Elgar. 29–45.

Borgelt, C. and Berthold, M. R. 2002. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)* (Washington, DC). 51–58.

Brin, S., Motwani, R., and Silverstein, C. 1997. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)* (Tucson, AZ). 265–276.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. 2000. Graph structure in the web: Experiments and models. In *Proceedings of the 9th International World Wide Web Conference (WWW)* (Amsterdam, the Netherlands). 309–320.

Bron, C. and Kerbosch, J. 1973. Finding all cliques of an undireced graph. *Commun. ACM 16*, 9, 575–576.

Buehrer, G., Parthasarathy, S., and Ghoting, A. 2006. Out-of-Core frequent pattern mining on a commodity PC. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. (Philadelphia, PA) 86–95.

Chakrabarti, D. and Faloutsos, C. 2006. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv. 38*, 1 (Mar.), Article 2.

Chen, Q., Lim, A., and Ong, K. W. 2003. D(k)-index: An adaptive structural summary for graph-structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)* (San Diego, CA). 134–144.

Chi, Y., Nijssen, S., Muntz, R., and Kok, J. 2005. Frequent subtree mining—An overview. *Fundam. Inf. 66*, 1-2, 161–198.

Dehaspe, L., Toivonen, H., and King, R. 1998. Finding frequent substructures in chemical compounds. In *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (New York). 30–36.

Deshpande, M., Kuramochi, M., and Wale, N. 2005. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. Knowl. Data Eng. 17*, 8, 1036–1050.

Dong, G. and Li, J. 1999. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (San Diego, CA). 43–52.

Feige, U., Goldwasser, S., Lovasz, L., Safra, S., and Szegedy, M. 1991. Approximating clique is almost NP-complete. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS)* (San Juan, PR). 2–12.

Frawley, W. J., Piatetsky-Shapiro, G., and Matheus, C. J. 1992. Knowledge discovery in databases—An overview. *AI Mag. 13*, 3, 57–70.

Hashimoto, K., Aoki-Kinoshita, K. F., Ueda, N., Kanehisa, M., and Mamitsuka, H. 2006. A new efficient probabilistic model for mining labeled ordered trees. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (Philadelphia, PA). 177–186.

Hastad, J. 1996. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)* (Burlington, VT). 627–636.

Horvath, T., Bringmann, B., and Raedt., L. D. 2006. Frequent hypergraph mining. In *Proceedings of the 16th International Conference on Inductive Logic Programming (ILP)* (Santiago, Spain).

HORVATH, T., RAMON, J., AND WROBEL, S. 2006. Frequent subgraph mining in outerplanar graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (Philadelphia, PA). 197–206.

HU, Y., OLMAN, V., AND XU, D. 2002. Clustering gene expression data using a graph-theoretic approach: An application of minimum spanning trees. *Bioinformatics 18*, 4, 536–545.

HU, H., YAN, X., HANG, Y., HAN, J., AND ZHOU, X. J. 2005. Mining coherent dense subgraphs across massive biological network for functional discovery. *Bioinformatics 21*, 213–221.

HUAN, J., WANG, W., AND PRINS, J. 2003. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)* (Melbourne, FL). 549–552.

INOKUCHI, A., WASHIO, T., AND MOTODA, H. 2000. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)* (Freiburg, Germany). 13–23.

KARP, R. 1972. Reducibility among combinational problems. In *Complexity of Computer Computations*, R. E. Miller and Thatcher eds. Plenum Press, New York. 85–103.

KATO, H. AND TAKAHASHI, Y. 2001. Automated identification of three-dimensional common structural features of proteins. *Genome Inf. 8*, 296–297.

KLEMETTINEN, M., MANNILA, H., RONKAINEN, P., TOIVONEN, H., AND VERKAMO, A. I. 1994. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM)* (Gaithersburg, MD). 401–407.

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

LAXMAN, S. AND UNNIKRISHNAN, K. P. 2005. Discovering frequent episodes and learning hidden Markov models: A formal connection. *IEEE Trans. Knowl. Data Eng. 17*, 11, 1505–1517.

MANNILA, H., TOIVONEN, H., AND VERKAMO, A. I. 1997. Discovery of frequent episodes in event sequences. *Data Mining Knowl. Discov. 1*, 3, 259–289.

MATSUDA, H., ISHIHARA, T., AND HASHIMOTO, A. 1999. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theor. Comput. Sci. 210*, 2, 305–320.

OSTERGARD, P. R. 2002. A fast algorithm for the maximum clique problem. *Discrete Appl. Math. 120*, 1-3, 197–207.

PAPADIAS, D., TAO, Y., MOURATIDIS, K., AND HUI, C. K. 2005. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst. 30*, 2, 529–576.

PEI, J., JIANG, D., AND ZHANG, A. 2005. On mining cross-graph quasi-cliques. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (Chicago, IL). 228–238.

PENSA, R.G., ROBARDET, C., AND BOULICAUT, J.F. 2005. A bi-clustering framework for categorical data. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)* (Porto, Portugal). 643–650.

SELMAOUI, N., LESCHI, C., GAY, D., AND BOULICAUT, J.F. 2006. Feature construction and delta-free sets in 0/1 samples. In *Proceedings of the 9th International Conference on Discovery Science (DS)* (Barcelona, Spain). 363–367.

SILVERSTEIN, C., BRIN, S., MOTWANI, R., AND ULLMAN, J. 2000. Scalable techniques for mining causal structures. *Data Mining Knowl. Discov. 4*, 2-3, 163–192.

VANETIK, N., GUDES, E., AND SHIMONY, S. E. 2002. Computing frequent graph patterns from semistructured data. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)* (Maebashi City, Japan). 458–465.

WANG, C., WANG, W., PEI, J., ZHU, Y., AND SHI, B. 2004. Scalable mining of large disk-based graph databases. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (Seattle, WA). 316–325.

WANG, H., WANG, W., YANG, J., AND YU, P. S. 2002. Clustering by pattern similarity in large data sets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. Madison, WI). 394–405.

WANG, J., HAN, J., AND PEI, J. 2006a. Closed constrained gradient mining in retail databases. *IEEE Trans. Knowl. Data Eng. 18*, 6, 764–769.

WANG, J., ZENG, Z., AND ZHOU, L. 2006b. Clan: An algorithm for mining closed cliques from large dense graph databases. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)* (Atlanta, GA). Article 73.

YAN, X. AND HAN, J. 2002. GSPAN: Graph-Based substructure pattern mining. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)* (Maebashi City, Japan). 721–724.

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 (SIGKDD)* (Washington, DC). 286–295.

YAN, X., YU, P. S., AND HAN, J. 2004. Graph indexing: A frequent structure-based approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)* (Paris). 335–346

YAN, X., ZHOU, X. J., AND HAN, J. 2005. Mining closed relational graphs with connectivity constraints. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (Chicago, IL). 324–333.

YANG, L., LEE, M. L., AND HSU, W. 2003. Efficient mining of XML query patterns for caching. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB)* (Berlin). 69–80.

ZAKI, M. J. 2002. Efficiently mining frequent trees in a forest. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (Edmonton, Alberta, Canada). 71–80.

ZENG, Z., WANG, J., ZHOU, L., AND KARYPIS, G. 2006. Coherent closed quasi-clique discovery from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (Philadelphia, PA). 797–802.

ZHANG, J., HSU, W., AND LEE, M. 2005. Clustering in dynamic spatial databases. *J. Intell. Inf. Syst. 24*, 1, 5–27.

ZHANG, M., KAO, B., CHEUNG, D. W., AND YIP, K. Y. 2005. Mining periodic patterns with gap requirement from sequences. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (Chicago, IL). 623–633.

Received November 2006; revised February 2007; accepted March 2007