CLAN: An Algorithm for Mining Closed Cliques from Large Dense Graph Databases *

Jianyong Wang[†], Zhiping Zeng[‡], Lizhu Zhou[§]
Department of Computer Science and Technology
Tsinghua University, Beijing, 100084, China
{[†]jianyong, [§]dcszlz}@tsinghua.edu.cn, [‡]cengzp03@mails.tsinghua.edu.cn

Abstract

Most previously proposed frequent graph mining algorithms are intended to find the complete set of all frequent, closed subgraphs. However, in many cases only a subset of the frequent subgraphs with a certain topology is of special interest. Thus, the method of mining the complete set of all frequent subgraphs is not suitable for mining these frequent subgraphs of special interest as it wastes considerable computing power and space on uninteresting subgraphs. In this paper we develop a new algorithm, CLAN, to mine the frequent closed cliques, the most coherent structures in the graph setting. By exploring some properties of the clique pattern, we can simplify the canonical label design and the corresponding clique (or subclique) isomorphism testing. Several effective pruning methods are proposed to prune the search space, while the clique closure checking scheme is used to remove the non-closed clique patterns. Our empirical results show that CLAN is very efficient for large dense graph databases with which the traditional graph mining algorithms fail. The novelty of our method is further demonstrated by the application of CLAN in mining highly correlated stocks from large stock market data.

1 Introduction

In recent years, frequent graph mining has attracted much attention in the data mining community, due to the ability of a graph to naturally model more generic relationships among different objects and its wide applications, such as chemical compound classification, and the discovery of activity-related groups of chemical compounds or contrast fragment structures [4]. Various frequent subgraph discovery algorithms have been developed in recent years [6, 13, 16, 4, 19, 10, 20, 17].

Most of the above frequent graph mining algorithms mine the complete set of frequent (or closed) subgraphs without considering the underlying topology of these subgraphs. However, in some cases people may only show special interest to a subset of the subgraphs with a certain topology, like cliques, bipartites, or rings. In fact, computing cliques from a graph has long been studied [7, 9], and identifying the size of the largest clique in a graph is one of the first problems shown NP-Hard. In recent years, researchers have found more applications in computing some specially chosen subgraphs, such as enumerating and organizing all Web occurrences of cliques, bipartite cores, and webrings in order to build Web knowledge bases [12, 8], detecting massive cliques or quasi-cliques from telephone call detail database to identify some communities of interest [1], finding cliques from stock market graphs in order to identify groups of highly-correlated stocks [3], and mining cross-graph quasi-cliques in gene-expression and protein interaction data in order to identify the clusters of genes that are co-expressed and also their proteins interact [15].

The clique subgraph is the most coherent and dense substructure among all kinds of subgraphs if we assume there is at most one edge between any two vertices: each of its vertices has a connection to all the others. The frequently occurring clique patterns can be used as the dense cores to cluster (or classify) the graph transactions, to condense the graph databases by treating each clique as a supernode, or to give some insights about the underlying structure or relationships among different objects in the graph transactions (e.g., chemical compounds or protein molecules). In [11], the authors show the usefulness of finding cliques in searching

*The work was supported by National Natural Science Foundation of China (NSFC) under Grant No. 60573061 and 60520130299.

the maximal common structural features among protein molecular graphs. Although discovering frequent cliques from a set of graph transactions is very useful, to the best of our knowledge it has not been well studied. One solution to this problem is to first mine the complete set of all kinds of frequent subgraphs, from which we can further identify the clique patterns. However, this is inefficient due to the enumeration of a large number of unwanted subgraphs, and once the input graph database becomes a little dense, mining the complete set of frequent subgraphs from it will be infeasible. A more efficient way should be to explore some properties of the clique subgraph, design new methods, and directly mine the set of frequent clique patterns. As the previous studies have shown that mining closed patterns can generate more concise result set, from which the complete set of frequent patterns can be easily derived [14, 18, 20, 5], as a result, mining the set of frequent closed clique patterns should be the default task of mining frequent cliques.

In this paper we develop a new algorithm, CLAN[1], to mine the frequent closed cliques from large dense graph transaction databases. In the CLAN algorithm, we make full use of the properties of the clique graph and propose a simple canonical label to uniquely represent a clique, which facilitates clique (or subclique) isomorphism testing. Several effective pruning methods are also proposed to prune search space and speed up the mining process, while the clique closure checking scheme is used to remove the non-closed clique patterns. We have also performed an extensive performance study in order to evaluate the CLAN algorithm.

The rest of this paper is organized as follows. Section 2 defines the problem. The related work is discussed in Section 3. Section 4 describes the algorithm design of CLAN and Section 5 presents the empirical results. Finally Section 6 gives the conclusion.

2 Problem Definition

A graph transaction database, D, consists of a set of undirected graph transactions, and the cardinality of D is denoted by $|D|$. Each graph transaction, G, is defined as a tuple $G = \{V, E, L_V, F_V\}$, where V is the set of vertices of G, $E \subseteq V \times V$ is the set of edges of G, L_V is the set of vertex labels, and $F_V : V \rightarrow L_V$ maps the vertices to their labels.

Figure 1 shows a graph database that contains two

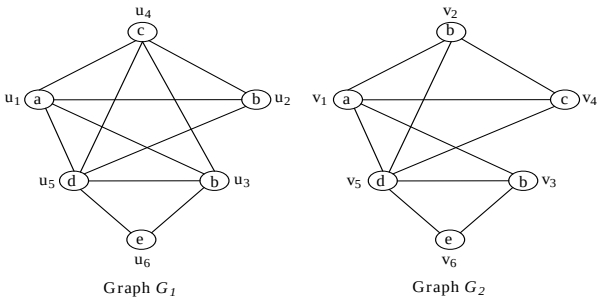


Figure 1. An example graph database D.

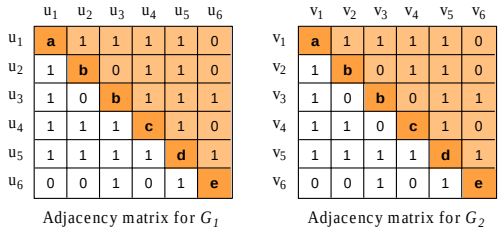


Figure 2. Adjacency matrix representation of G_1 and G_2 in our running example.

graph transactions G_1 and G_2 (i.e., $|D|$=2), which will be used as our running example. As in [13, 10], we use an adjacency matrix M to represent a graph G: Each row (or column) represents a vertex, each diagonal element $M[i,i](for\ 1 \leq i \leq |V|)$ is the corresponding label of the ith vertex v_i, and each non-diagonal element $M[i,j](for\ i \neq j)$ indicates the presence (denoted by 1) or absence (denoted by 0) of edge e_{ij} between vertices v_i and v_j. Figure 2 shows the adjacency matrix representation of G_1 and G_2 in our running example. Note that the graphs we study here are undirected, thus the lower triangle of any adjacency matrix is symmetric to its upper triangle.

A clique, C, is defined as a set of fully connected labeled vertices and denoted by a tuple $C = \{V, L_V, F_V\}$, where V is the set of vertices, L_V is the set of vertex labels, and $F_V : V \rightarrow L_V$ maps the vertices to their labels. The size of a clique is defined as the number of vertices it contains, i.e., $|V|$. A clique of size n is also called a n-clique and contains exactly $\frac{n \times (n-1)}{2}$ edges. A clique $C_1 = \{V_1, L_{V1}, F_{V1}\}$ is *isomorphic* to another clique $C_2 = \{V_2, L_{V2}, F_{V2}\}$ iff $|V_1| = |V_2|$ and there exists a bijection $f : V_1 \rightarrow V_2$ such that $\forall v \in V_1$, $F_{V1}(v) = F_{V2}(f(v))$. If a clique $C = \{V, L_V, F_V\}$ is *isomorphic* to a subgraph of another clique $C^{'} = \{V^{'}, L_V^{'}, F_V^{'}\}$, C is called a subclique of $C^{'}$, while $C^{'}$ is called a *superclique* of C. We use

[1] In English, the word 'clan' has a very close meaning to the word 'clique', that is 'a group united by a common interest or common characteristics'. Here we use CLAN to stand for 'frequent closed **CL**ique p**A**tter**N** mining'.

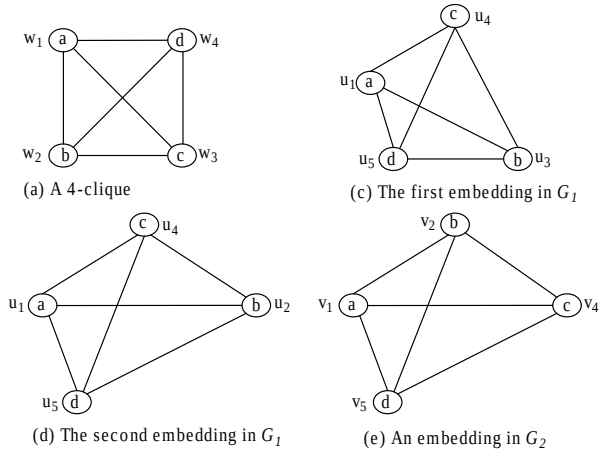


Figure 3. A 4-clique structure with vertex labels a, b, c, and d, and its embeddings in G_1 and G_2 of our running example.

$C \sqsubseteq C'$ or $C \sqsubset C'$ (i.e., $C \sqsubseteq C'$ but $C \neq C'$) to denote the subclique or proper subclique relationship.

A fully connected subgraph g of a graph G can form a clique and if g is isomorphic to a clique C, we call g an embedding of C in G. C is supported by a graph G, if there is at least one embedding of C in G. The number of graph transactions in graph database D that support C is called the absolute support (or simply called *support*) of C in D, denoted by $sup^D(C)$. Given an absolute support threshold min_sup, if $sup^D(C) \geq min\_sup$, C is called a *frequent* clique in D. If there does not exist another clique C' such that $C \sqsubset C'$ and $sup^D(C) = sup^D(C')$, C is called a *closed* clique in D.

Given a graph transaction database D and a minimum support threshold min_sup, the problem of *frequent closed clique mining* is to find the complete set of cliques in D that are both frequent and closed.

Example 2.1 Given min_sup=2, the 4-clique shown in Figure 3(a) with vertex labels a, b, c, and d is a frequent clique structure in the database D shown in Figure 1, as it is supported by both G_1 and G_2: it has two embeddings in G_1 (see Figure 3(c)-(d)) and one embedding in G_2(see Figure 3(e)). And we cannot find any proper superclique of it with the same support, thus it is also closed. From Figure 1 we can get the complete set of frequent closed cliques: one 4-clique with vertex labels a, b, c, and d, and one 3-clique with vertex labels b, d, and e. ALL the other cliques are frequent but not closed. □

Notice that in the above example we have not considered the edge label in computing a frequent closed clique. We made this decision due to the following observations. First, many real-life databases do not have edge labels. Second, this means we mainly concern about the clique topology and its associated vertex labels, which enables us to find some hidden clique patterns that may not be found when we require the exact match of the edge labels.

3 Related Work

Computing the maximum clique (or maximal cliques) or enumerating all the cliques from a single graph have been very well studied for many years [7, 9], and become active again in recent years since the advent of the Web, digital libraries, data mining, and bio-informatics [12, 11, 1]. However, most of this research only studies mining clique patterns from a single graph, which is different from the problem studied in this paper, that is, mining frequent closed cliques from a set of graph transactions. In [15], cross-graph quasi-clique mining in gene-expression and protein interaction data was proposed and shows an application in identifying the clusters of genes that are co-expressed and also their proteins interact. Their problem formulation is also different from ours.

Because the set of frequent closed patterns is usually more concise than the set of all frequent patterns while retaining completeness, since the introduction of the closed itemset mining in [14], mining 'closed patterns' has received much interest. Several algorithms have been proposed to mine closed sequences, trees, and graphs [18, 5, 20]. Due to the complexity of the structural pattern mining, designing some efficient pattern closure checking scheme or search space pruning methods is especially challenging for closed structural pattern mining. Like the existing methods used in [20, 5], the closure checking scheme of CLAN is straightforward and is directly based on the definition of a closed clique, i.e., check every possible extension to see if there is any extension with the same support. Because the same subgraph may have more than one embedding in the same graph transaction, designing some pruning methods for closed structure mining needs special caution. As the pruning methods previously proposed for closed structure mining cannot work in closed clique mining, in CLAN we proposed a new pruning method.

4 CLAN: An Efficient Algorithm to Mine Frequent Closed Cliques

In this section, we will introduce the CLAN algorithm in detail. First we describe the *canonical form*

of a clique graph that we choose in order to uniquely represent a clique and simplify the *clique (or subclique) isomorphism* testing. Then we focus on how to efficiently enumerate the complete set of frequent cliques by applying the *structural redundancy pruning* and *pseudo low-degree vertex pruning* methods. We also propose the *clique closure checking scheme* and *non-closed prefix pruning* methods in order to mine the set of frequent closed clique structures efficiently.

4.1 Canonical Representation of a Clique

Due to the high complexity of graph isomorphism problem, one of the key issues in frequent graph mining is how to choose a good canonical form that can uniquely represent a graph. Usually the 'goodness' of a canonical form is judged by whether it has low computational complexity or not, in order to facilitate the graph isomorphism test. Currently there exist two popular classes of solutions to this problem. In the first category, the canonical form of a graph is defined as the minimum (or maximum) adjacency matrix code among all the possible codes in terms of a certain global ordering (e.g., lexicographic ordering) for the corresponding graph [13, 10], where an adjacency matrix code is the sequence of the upper (or lower) triangular entries in the matrix. In the second class of solutions, some kind of DFS (abbreviated for Depth First Search) codes (e.g., the minimum DFS code in lexicographic ordering [19]) are used as the canonical form [21].

Although both the adjacency matrix code and DFS code can be used as the canonical form of a clique, they have very high computational complexity and are not efficient representations of a clique graph. Because any clique graph is completely connected, if two cliques with the same size have the same bag of vertex labels, their topology will be identical according to our *clique isomorphism* definition. Given a clique graph of a size k, C_k, we call any sequence of all its vertex labels a clique *string* (Note here we allow the same vertex label to appear in the same *string* multiple times as in the traditional sequence definition [18]). There will be totally $k!$ different *string*s if each of C_k's vertex labels is distinct. Assume there is a lexicographic ordering on the vertex labels, we define the following global order of any two strings p and q with the same size k. We use p_i to denote the i-th vertex label in *string* p, then p is said to be smaller than q if $\exists l$ $(0 < l \leq k)$ such that $\forall i$ $(0 < i < l)$, $p_i = q_i$ and $p_l < q_l$, otherwise p is said to be greater than or equal to q.

Definition 4.1 (Canonical form) Given a clique C, its canonical form is defined as the minimum *string* among all its possible *string*s and denoted by CF_C.

According to the above definition, the canonical form of a clique C_i is aac (i.e., $CF_{C_i} = aac$), if it has three vertices, among which two vertices have a label a, the other has a label c. Once we know the canonical forms of two cliques, it will be straightforward to test if they are *isomorphic*: if their canonical forms are identical, they are isomorphic, otherwise, they are not. The definition of the canonical form as the minimum string also facilitates the subclique relationship test, which is very helpful in clique closure checking. A string $S_a{=}a_1a_2\ldots a_n$ is called a *substring* of another string $S_b{=}b_1b_2\ldots b_m$ (denoted by $S_a \sqsubseteq S_b$), if there exist integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $a_1{=}b_{i_1}$, $a_2{=}b_{i_2}$, $\ldots$, $a_n{=}b_{i_n}$. Given the definition of *substring*, we have the following lemma that can be used to test if a clique is a subclique of another clique.

Lemma 4.1 (subclique relationship test) *Given any two cliques, C_a and C_b, with canonical forms CF_{C_a} and CF_{C_b}, respectively. $C_a \sqsubseteq C_b$ holds iff $CF_{C_a} \sqsubseteq CF_{C_b}$ holds.* □

This lemma can be directly obtained from the definitions of the *subclique*, *canonical form* of a clique and the *substring*, and it provides an efficient way to check if a clique is a subclique of another clique by simple substring relationship test among their corresponding canonical forms.

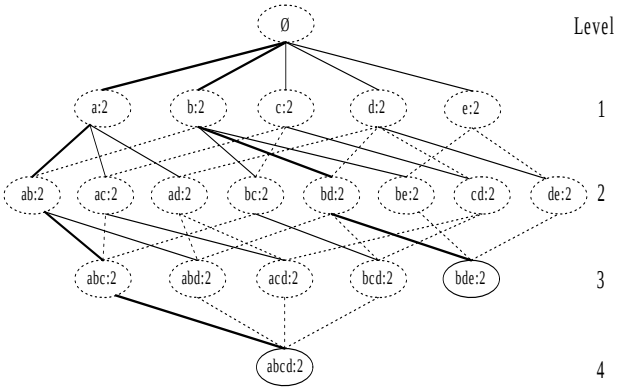


Figure 4. A lattice-like structure built from the frequent cliques in our running example.

4.2 Frequent Clique Enumeration

From Lemma 4.1 we know that the *subclique* relationship between two cliques can be converted to a *substring* relationship between their corresponding canonical forms. Thus once given a global lexicographic ordering on the vertex labels, we can represent

the frequent cliques by their corresponding canonical forms and conceptually organize them into a lattice-like structure: each node represents a clique in the form of '*canonical form:support*' (Note: for simplicity, in the following we will denote a clique by its canonical form plus its support), and each edge between two nodes represents a *direct subclique* (that is, with exactly one fewer vertex) relationship between the two corresponding cliques. All the frequent k-cliques are at level k and are arranged according to the lexicographic ordering of their canonical forms. In our running example in Figure 1, assume the lexicographic order of the vertex labels is $a \leq b \leq c \leq d \leq e$, all the 19 frequent cliques can be organized into a lattice-like structure as shown in Figure 4. For example, the 4-clique in Figure 3(a) is represented by the node with a label '*abcd*:2' and it has four 3-cliques as its direct subcliques, i.e., *abc*:2, *abd*:2, *acd*:2, and *bcd*:2. In addition, all the nodes with the dotted ellipses are the non-closed cliques. Figure 4 shows that among the 19 frequent cliques, only *abcd*:2 and *bde*:2 are closed.

By conceptually organizing the frequent cliques into a lattice-like structure, the problem of mining frequent cliques becomes how to traverse the lattice-like structure to enumerate frequent cliques. Previous studies have shown two popular search strategies: *breadth-first search* [13] and *depth-first search* [4, 19, 10]. In CLAN, we adopt the depth-first search strategy. However, our method is slightly different from the previous ones. The previous depth-first search methods like the *right most extension* [2, 21, 19], grow the current graph of size k by one edge in order to get the graph of size $(k+1)$, while CLAN grows the current k-clique by one vertex plus the corresponding k edges in one step to generate a $(k+1)$-clique.

Structural redundancy pruning. By depth-first traversing the lattice-like structure, we can mine a list of frequent cliques in the following order: *a*:2, *ab*:2, *abc*:2, *abcd*:2, *abd*:2, *abcd*:2, *ac*:2, *abc*:2, *abcd*:2, *acd*:2, *abcd*:2, ... From this list we can see there exist a lot of redundancy in the sense that some cliques are generated multiple times. For example, the clique *abc*:2 can grown from *ab*:2 by adding a new vertex *c*, or from *ac*:2 by adding vertex *b*, or from *bc*:2 by adding vertex *a*. A simple way to remove the redundant cliques can be implemented by maintaining the set of already mined cliques. Upon getting a new clique, we check if there is any already mined clique that has the same canonical form as the new one, if so, we just throw away the newly generated clique. However, this does not help in removing the redundant computing. A more efficient way can be based on the following property of the canonical form of a clique.

Lemma 4.2 (Prefix closure property of canonical form) *Given any clique C and its canonical form CF_C, any non-empty prefix of CF_C represents the canonical form of a certain clique.*

Proof. *We will prove it by contradiction. Let $P(CF_C)$ denote any prefix of CF_C, and $S(CF_C)$ denote the suffix by removing $P(CF_C)$ from CF_C. Assume $P(CF_C)$ is not in canonical form, then there must exist another string, X, which contains the same bag of vertex labels as $P(CF_C)$ and $X < P(CF_C)$. This means the concatenation of X and $S(CF_C)$ is smaller than CF_C, that is, $X \diamond S(CF_C) < CF_C$. This contradicts with the fact that CF_C is the canonical form of clique C.* □

From this lemma we propose a clique enumeration method which can eliminate the structural redundancy while maintaining the completeness of the result set. For any clique C with canonical form CF_C, we can get the *direct prefix* of CF_C by removing the last vertex label from CF_C. In CLAN, we require a clique C be mined by only growing its direct subclique with C's direct prefix as its canonical form. Because the canonical form uniquely represents a clique, thus this method eliminates the redundancy in clique enumeration. This structural redundancy pruning method can be implemented in another equivalent way. For the current prefix clique C with canonical form CF_C, let the last vertex label of CF_C be c. When we want to grow C in order to mine larger cliques, we require C be extended only with vertices whose labels are lexicographically no smaller than c. For example, if the current prefix clique is *ac*:2, we can only grow it with vertices whose labels are from $\{c, d, e\}$ in our example. In such a way clique *abc*:2 will not be generated from *ac*:2. In Figure 4, we use the dotted edges and solid edges to denote the redundant extensions and the valid extensions, respectively.

For any a current prefix k-clique, C_p (initially $k = 0$ and $C_p = \varnothing$), which has a canonical form CF_{C_p}, the CLAN algorithm first finds all the 'valid' vertex labels which are lexicographically no smaller than the last label of CF_{C_p} by scanning the database, and records their corresponding embeddings in each graph transaction. Here by a 'valid' vertex label, we mean it frequently fully connects to C_p and is lexicographically no smaller than the last label of CF_{C_p}. The list of valid vertex labels are sorted in lexicographic ascending order, denoted by $<l_1, l_2, ..., l_n>$. Each valid vertex label will be used to extend C_p in order to get a frequent $(k+1)$-clique and before we mine all the frequent cliques with prefix $C_p \diamond l_{i+1}$ we should first mine all the frequent cliques with prefix clique $C_p \diamond l_i$ in a recursive way according to the depth-first search order. Under

the depth-first search strategy and the structural redundancy pruning method, the 19 frequent cliques in our running example will be mined in such an order: a:2, ab:2, abc:2, $abcd$:2, abd:2, ac:2, acd:2, ad:2, b:2, bc:2, bcd:2, bd:2, bde:2, be:2, c:2, cd:2, d:2, de:2, e:2.

Pseudo low-degree vertex pruning. One of the most costly operations in clique enumeration is the database scan in order to find the valid extensions. A nice property of a k-clique is that any of its vertex degrees is at least $k-1$. The following simple observation has been used extensively in clique enumeration from a single graph and can be also adopted to reduce the overhead of database scan in frequent clique enumeration.

Observation 4.1 (Low-degree vertex pruning) *No vertex with a degree lower than $(k-1)$ can be contained in a k-clique.* □

From Observation 4.1 we know that when we want to extend a k-clique, we only need to scan the database in which the vertices whose degrees are lower than k have been pruned. The pruning of low degree vertices can be done in a recursive way, this is because the pruning of some low degree vertices may make more vertices have a low degree. For example, when we want to mine the frequent 4-cliques, the removal of node v_6 in the graph G_2 of Figure 1 will make v_3 have a degree 2, which can be pruned further. To accelerate the clique enumeration, one method is that we maintain a set of graph databases with various low-degree vertices physically pruned, and just scan the database whose vertices all have a degree no lower than k upon extending a k-clique in order to mine the $(k+1)$-cliques. Let the largest frequent clique contains N vertices, in the worst case we need to maintain $(N-1)$ such databases, which will consume much memory. To save space usage, in CLAN algorithm, we do not physically construct all such databases, instead, we only need to record the indices of the left high-degree vertices, by which we can locate the needed matrix entries in the original database.

4.3 Frequent Closed Clique Discovery

Using the above frequent clique enumeration method, we can mine the complete set of frequent cliques in the lexicographic order in terms of their corresponding canonical form. Upon getting a new frequent clique, we need to check if it is closed or not based on the following clique closure checking scheme, in order to generate the set of closed cliques.

Clique closure checking scheme. Assume the current prefix clique, C, contains k vertices with a canonical form $CF_C = \alpha_1\alpha_2...\alpha_k$. According to the definition of a closed clique, if there exists any vertex with a label β, which can be used to grow C to get a $(k+1)$-clique, C', such that $sup^D(C) = sup^D(C')$, C must be non-closed. In this case β is called a *new extension vertex* if $\beta \geq \alpha_k$, or an *old extension vertex* if $\beta < \alpha_k$. According to the following lemma, to check if a clique is closed or not we only need to check if there exists any new (or old) extension vertex.

Lemma 4.3 (Clique closure checking) *If there exists no new extension vertex nor old extension vertex w.r.t. a prefix k-clique C, C must be closed, otherwise C must be non-closed.*

Proof. *In considering the downward-closure property of a clique, this lemma can be easily derived from the definition of a closed clique.*□

Because the canonical form of a clique is a special type of sequence, the clique closure checking in Lemma 4.3 is very similar to the BI-Directional Extension sequence closure checking scheme used in the BIDE algorithm [18]. Both the new extension vertex and the old extension vertex checking can be combined with the clique enumeration phase by scanning the database and see if there is any extension vertex that has the same support as the current prefix clique. While Lemma 4.1 offers another viable way for old extension vertex checking, that is, if there is any already mined frequent clique which is a superclique of the current clique C and has the same support, there must exist at least one old extension vertex, and thus C is not closed. In addition, all the canonical forms of the already mined cliques can be organized into a hash structure in order to accelerate the checking.

Non-closed prefix pruning. Mining only closed cliques provides more chance to prune some unpromising search space. For example, as shown in Figure 4 there is no hope to grow the prefix clique c:2 to get any closed cliques, and thus should be pruned as quickly as possible. However, designing some efficient pruning methods in graph mining is especially challenging. Because a subgraph may have more than one embedding in a graph transaction, the support-based pruning methods used in closed itemset mining algorithms no longer hold. Some carefully designed methods like the *Early Termination* method seem straightforward and intuitively correct, but may still face several failure cases [20]. Chi *et al* proposed a nice occurrence-matched based pruning methods in [5], which works well for mining frequent closed rooted unordered trees. However, when we extend it to prune search space in mining closed cliques, it still encounters difficulties. For

example, let bd:2 be the current prefix clique, it has totally four occurrences in the example graph database D as shown in Figure 1, and for each occurrence of bd:2, we can always find one corresponding occurrence of another clique abd:2. As a result, bd:2 and its superclique abd:2 are occurrence-matched. In addition, from Figure 4 we can see that clique abd:2 is generated earlier than bd:2, which means abd:2 is not a rightmost extension of bd:2. According to [5], bd:2 can be pruned. However, as Figure 4 shows that bd:2 is on the *critical path* (i.e., the darker solid path in Figure 4, '$\varnothing \rightarrow b$:2 $\rightarrow bd$:2 $\rightarrow bde$:2') leading to the closed clique bde:2. If we prune clique bd:2, bde:2 will be never discovered. As a result, we need to design some new pruning method(s) for mining closed cliques.

Let the current prefix clique be C, the number of vertices in C be k, its canonical form be $CF_C = \alpha_1\alpha_2...\alpha_k$. Assume there are totally m embeddings of C in the graph database G and the set of embeddings is denoted by $EMB(C)$=$\{emb_1^C, emb_2^C, ..., emb_m^C\}$. For any embedding emb_i^C, a vertex v is an extension to emb_i^C, if there exists an edge between v and every vertex in emb_i^C. Let V_i^C denote the set of extension vertices of the ith embedding emb_i^C. A vertex $v' \in V_i^C$ is called a *fully connected vertex* in V_i^C, if there is an edge between vertex v' and any vertex in $V_i^C - \{v'\}$. The set of fully connected vertex labels in V_i^C is denoted by FV_i^C, and let FV^C=$FV_1^C \cap FV_2^C ... \cap ...FV_m^C$. If $FV^C \neq \phi$, and $\exists\beta$, $\beta \in FV^C$ and $\beta < \alpha_k$, we call β a *non-closed extension vertex label* w.r.t. clique C.

Lemma 4.4 (Non-closed prefix pruning) *If there exists at least one non-closed extension vertex label, β, w.r.t. a prefix k-clique C, C can be safely pruned.*

Proof. *Let the canonical form of C be $CF_C = \alpha_1\alpha_2...\alpha_k$, and use $CF_{C'} = \alpha_1\alpha_2...\alpha_k\gamma_1...\gamma_j$ ($j \geq 1$) to denote any proper superclique C' with C as the prefix clique. Under the structural redundancy pruning strategy, $\forall i$, $\gamma_i \geq \alpha_k$ must hold. Because C' is a clique, all the vertices corresponding to vertex labels $\gamma_1, ..., \gamma_j$ must be fully connected vertices w.r.t. clique C. As a result, vertex label β fully connects to each vertex in C' and can be used to extend C' to get a superclique of C', denoted by C''. Because $\beta < \alpha_k$, C'' must have been mined before C'. Also, according to the definition of the non-closed extension vertex label, we know $sup^D(C') = sup^D(C'')$. This means that C' is always non-closed. As a result, we cannot generate any closed cliques from prefix clique C and thus can be pruned.* □

We use an example to illustrate how to use Lemma 4.4. Assume the current prefix clique C contains one vertex with a label c. In our running example shown in Figure 1, there are two embeddings of C in database D. The embedding in G_1 (i.e., vertex u_4) has four neighbors, u_1, u_2, u_3, and u_5, among which u_1 has a label a and connects to all the other neighbors of u_4. Similarly, the embedding in G_2 (i.e., vertex v_4) has three neighbors, v_1, v_2, and v_5, and v_1 also fully connects to all other neighbors and has a label a. Because $a < c$, a is a *non-closed extension vertex label* w.r.t. clique C, and clique C with canonical form c can be pruned. Similarly, from Figure 1 we can see that both vertex labels b and d are *non-closed extension vertex label*s w.r.t. prefix clique e:2, thus e:2 can be pruned. However, if the current prefix clique is b:2, we will not be able to find any *non-closed extension vertex label*, as a result, we cannot prune b:2. In fact, if we prune b:2, we will no longer find the closed clique bde:2.

ALGORITHM 1: **CLAN**(*D, CF_C, EMB_C, min_sup*)

INPUT: (1) D: the pruned graph transaction database, (2) CF_C: *the canonical form of the current prefix clique C*, (3) EMB_C: *the set of embeddings of clique C in the database D*, and (4) *min_sup*: the minimum support threshold.
OUTPUT: (1) *SFCC*: *the set of frequent closed cliques.*
BEGIN
01. *Scan D according to EMB_C to find the set of frequent valid vertex labels, S_{fv};*
02. *Sort the labels in S_{fv} in lexicographic order;*
03. *Scan D again to find the embeddings for vertices in S_{fv};*
04. *If there exists any non-closed extension vertex label*
05. *return;*
06. *If there exists no extension vertex*
07. *SFCC=SFCC $\cup$ $\{CF_C\}$;*
08. *For each $l \in S_{fv}$ and $l \geq$ the last label of CF_C*
09. *Call CLAN(D, $CF_C \diamond l$, $EMB_{C \diamond l}$, min_sup);*
10. *return;*
END

4.4 The Algorithm

The CLAN algorithm is shown in Algorithm 1. Before we run CLAN, we apply the pseudo low-degree vertex pruning methods to generate a series of pseudo databases corresponding to cliques of different size. By scanning the original database, we can find the set of frequent 1-cliques and their corresponding embeddings. For each 1-clique, we can then use Algorithm 1 to mine all the frequent closed cliques with this 1-clique as prefix. For a prefix k-clique C with canonical form CF_C, we scan the pruned database corresponding to size k, and find the frequent valid vertices (by valid, we mean the vertex fully connects to C) and their corresponding embeddings (lines 01 and 03). The non-closed prefix pruning method(lines 04-05) and the closure checking scheme(line 06) are then applied. If C is closed, the algorithm output it (line 07). Each frequent valid vertex label l which is lexicographically

no smaller than the last label of CF_C will be used to extend C. The CLAN algorithm will be called with the new prefix clique whose canonical form is $CF_C \diamond l$(lines 08-09).

5 Empirical Results

In this section, we present empirical results. We will first show one application of the CLAN algorithm using the US stock market database. We mainly discuss how to convert the original US stock market database to the graph representation, and use CLAN to discover some highly correlated stocks and present some empirical results. We will also evaluate the CLAN algorithm's efficiency and scalability. All the experiments were performed on a 1GHz AMD PC with 256MB memory and linux installed.

5.1 Application - Mining Correlated Stocks

Several previous studies have shown some applications of clique graph mining [11, 3, 15]. Here we illustrate one of the applications: discover sets of correlated stocks by mining frequent closed cliques. According to [3], the stock market data w.r.t. a certain period of time can be converted to a graph based on the cross correlations of price fluctuations. More specifically speaking, each stock is represented by a vertex whose label is the corresponding stock index, and two vertices are connected by an edge if the correlation coefficient of the corresponding pair of stocks exceeds a specified threshold θ, $-1 \leq \theta \leq 1$. As pointed out in [3], analyzing cliques in market graph can give a very valuable knowledge about the internal structure of the stock market. This is because a clique in the graph represents a set of stocks whose prices evolve synchronously over time and a change of the price of any stock in a clique implies a similar change of the prices of all other stocks in the same clique.

As in [3], we used 11 sets of US stock market data, each of which consists of daily prices of a set of stocks over a different period of 500 consecutive trading days. The number of distinct stocks in the 11 sets of stock market data is 6556, 6399, 6262, 6104, 6013, 5866, 5768, 5666, 5593, 5507, and 5430 stocks, respectively. Although the number of stocks in each of the 11 sets of stock market data is different, most of the stocks are common in all the 11 sets of stock market data. We converted the 11 sets of stock market data to 11 market stock graphs based on the correlations of the stock prices. The correlation coefficient $C_{S_1}^{S_2}$ between two stocks, S_1 and S_2, w.r.t. a certain period of time T is defined as follows.

$$C_{S_1}^{S_2} = \frac{\frac{1}{|T|}\sum_{i=1}^{|T|}(S_1^i \times S_2^i - \overline{S_1} \times \overline{S_2})}{\sigma_{S_1} \times \sigma_{S_2}} \quad (1)$$

In Equation 1, $|T|$ denotes the number of days in period T, S_1^i and S_2^i denote the ith day's price of stock S_1 and S_2 respectively, and $\overline{S_1} = \frac{1}{|T|}\sum_{i=1}^{|T|} S_1^i$, $\overline{S_2} = \frac{1}{|T|}\sum_{i=1}^{|T|} S_2^i$, $\sigma_{S_1} = \sqrt{\frac{1}{|T|}\sum_{i=1}^{|T|}(S_1^i)^2 - \overline{S_1}^2}$, and $\sigma_{S_2} = \sqrt{\frac{1}{|T|}\sum_{i=1}^{|T|}(S_2^i)^2 - \overline{S_2}^2}$.

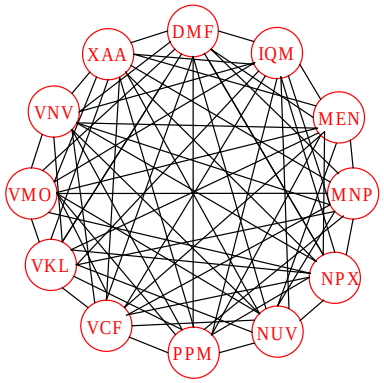


Figure 5. The maximum frequent closed clique in the *stock market* database with correlation coefficient threshold 0.9 and minimum relative support threshold 100%.

By setting the correlation coefficient threshold at 0.9 and using Equation 1, we converted the original 11 sets of US stock market data into a graph database containing 11 graphs (We will denote this graph database by the *stock market-0.9* from now on). There are totally 6018 distinct vertex labels in this graph database, on average there are 206,747 edges and 3,636 vertices in each graph, the maximal number of edges in a graph is 408,135, the maximal number of vertices in a graph is 4,085, and the maximal vertex degree is 823. It is obvious that the *stock market-0.9* graph database is large and dense, as it contains a large number of vertex labels, vertices, and edges, and the average vertex degree is also large. Due to the combinatorial explosion problem, mining frequent subgraphs from such kind of large dense graph databases is especially challenging. As we will show later that even the state of the art algorithm like ADI-Mine [17] cannot run with the *stock market-0.9* database at the highest possible minimum relative support threshold, 100%.

We set the minimum support threshold at 100% in order to find sets of stocks that are always highly correlated with each other over all the 11 periods (i.e.,

11×500 days). CLAN found totally 327 frequent closed cliques with a size no smaller than three, among which the maximum clique has a size 12. As shown in Figure 5, the maximum clique contains 12 stocks whose index names are DMF, IQM, MEN, MNP, NPX, NUV, PPM, VCF, VKL, VMO, VNV, and XAA, respectively. Because we have adopted a very high correlation coefficient threshold, 0.9, and a minimum support threshold, 100%, it is quite safe to say that the prices of these 12 stocks evolve in a similar way and a price change of any stock in the clique can be used to predict a similar change of the prices of all other 11 stocks.

5.2 Algorithm Evaluation

We evaluated the performance of CLAN using several real databases. The CA database in Table 1 is a chemical compound database which can be derived from the DTP AIDS Antiviral Screen database from National Cancer Institute and was used in previous studies [20, 10]. It was provided by the author of [13]. The other real databases can be derived from the *stock market* data by setting the correlation coefficient threshold at different values from 0.9 to 0.95. Table 1 depicts some characteristics of the real databases, such as the number of the graph transactions (column 2), the average number of vertices (column 3), and the average number of edges (column 4).

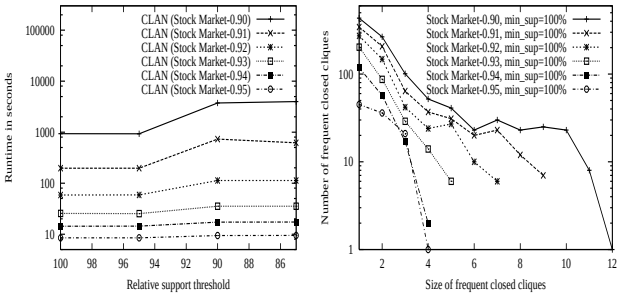
5.2.1 Efficiency Test

To evaluate the efficiency of the algorithm, we compared CLAN with ADI-Mine, which is one of the latest frequent subgraph mining algorithm [17]. The result set of CLAN is a subset of the result set of ADI-Mine, their performance is not directly comparable. However, our purpose here is to show that mining the complete set of frequent subgraphs as a first step to generate the set of frequent closed cliques is inefficient, and is in fact not a feasible way for large dense graph databases.

We first compared the two algorithms using the dense *stock market* series of databases. However, ADI-Mine could not complete after running for several days for these databases even with 100% minimum support, thus no result of ADI-Mine is shown in Figure 6. Figure 6(a) shows the runtime of CLAN for the six *stock market* databases by varying the support threshold from 100% to 85%, while Figure 6(b) shows the number of closed cliques against the size of closed cliques w.r.t. the six *stock market* databases at the support threshold of 100%. These results illustrate that although on average each graph of the *stock market* databases is very large and dense (e.g., for *stock market-0.9* database, on average each graph contains several thousand vertices and more than 200K edges), CLAN can still handle it efficiently.

Database	# graphs	Avg. # vertices	Avg. # edges
CA	422	39	42
Stock Market-0.95	11	1683	20074
Stock Market-0.94	11	2182	40008
Stock Market-0.93	11	2618	68608
Stock Market-0.92	11	2999	106156
Stock Market-0.91	11	3331	152356
Stock Market-0.90	11	3636	206747

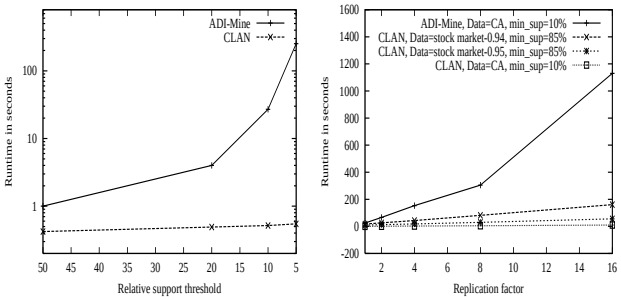
Table 1. Real database characteristics.



a) Runtime b) Distribution

Figure 6. Varying the support and correlation thresholds (*Stock Market*).

As Table 1 shows, the CA database contains some small and sparse graphs. From Figure 7(a) we can see that even for such a sparse graph database, CLAN outperforms ADI-Mine significantly. The performance gain for CLAN stems from the fact that CLAN only mines frequent closed cliques, and the newly proposed search space pruning methods can effectively remove some unpromising search space from consideration.



a) Efficiency (CA) b) Scalability

Figure 7. Efficiency and Scalability test.

5.2.2 Scalability Study

We also evaluated CLAN's scalability using several real databases in terms of the base size. We first chose two dense graph databases, *stock market-0.95* and *stock market-0.94*, and fixed the minimum support at 85%. As ADI-Mine cannot run with these two dense databases, we also used the sparse CA database in order to compare CLAN's scalability against ADI-Mine by fixing the minimum support at 10%. In Figure 7(b) we replicated the graphs from 2 to 16 times. It is evident that CLAN shows a linear scalability in runtime against the number of graphs in the database.

6 Discussions and Conclusion

In this paper we introduced the CLAN algorithm, which mines closed cliques from large dense graph databases. By focusing on the underlying clique topology and vertex labels, we first proposed a simple canonical form to uniquely represent a clique graph. By fully exploring some nice properties of the clique structure, we also proposed several pruning methods, *pseudo low-degree vertex pruning*, *structure redundancy pruning*, and *non-closed prefix pruning*, which can be used to accelerate the frequent closed clique mining. As in some cases people may also be interested in mining quasi-cliques besides the exact cliques, in future, we plan to explore how to extend the techniques proposed in this paper to mine closed quasi-cliques.

Acknowledgements.

The authors are grateful to Vladimir L. Boginski, Panos M. Pardalos, and Sergiy Butenko for providing us the US stock market database. We thank Michihiro Kuramochi for sending us several real graph databases, Wei Wang and Chen Wang for sending us their ADI-Mine algorithm, and Beng Chin Ooi for his valuable comments to this paper. Thanks also go to the anonymous ICDE'06 conference reviewers for their invaluable suggestions and detailed comments which helped a lot in improving the paper.

References

[1] J. Abello, et al. Massive quasi-clique detection. LATIN'02.

[2] T. Asai, et al. Efficient substructure discovery from large semi-structured data. SDM'02.

[3] V. Boginski, et al. On structural properties of the market graph. In *A. Nagurney (editor), Innovations in Financial and Economic Networks*, Edward Elgar Publishers, Apr. 2004.

[4] C. Borgelt and M. Berthold. Mining molecular fragments: finding relevant substructures of molecules. ICDM'02.

[5] Y. Chi, et al. CMTreeMiner: mining both closed and maximal frequent subtrees. PAKDD'04.

[6] L. Dehaspe, et al. Finding frequent substructures in chemical compounds. KDD'98.

[7] U. Feige, et al. Approximating Clique is Almost NP-Complete. FOCS'91.

[8] M. Garofalakis, et al. Data mining and the Web: past, present and future. WIDM'99.

[9] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. FOCS'96.

[10] J. Huan, et al. Efficient mining of frequent subgraphs in the presence of isomorphism. ICDM'03.

[11] H. Kato, and Y. Takahashi. Automated identification of three-dimensional common structural features of proteins. In *Journal of Chemical Software*, Vol. 7, No. 4, 2001.

[12] R. Kumar, et al. Extracting large-scale knowledge bases from the Web. VLDB'99.

[13] M. Kuramochi and G. Karypis. Frequent subgraph discovery. ICDM'01.

[14] N. Pasquier, et al. Discovering frequent closed itemsets for association rules. ICDT'99.

[15] J. Pei, et al. Mining Cross-graph Quasi-cliques in Gene Expression and Protein Interaction Data. ICDE'05.

[16] N. Vanetik, et al. Computing frequent graph patterns from semistructured data. ICDM'02.

[17] C. Wang, et al. Scalable mining of large disk-based graph databases. KDD'04.

[18] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. ICDE'04.

[19] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. ICDM'02.

[20] X. Yan and J. Han. CloseGraph: Mining closed frequent graph patterns. KDD'03.

[21] M. Zaki. Efficiently mining frequent trees in a forest. KDD'02.