An Efficient Algorithm for Solving Pseudo Clique Enumeration Problem

Takeaki Uno

National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan, `uno@nii.jp`

Abstract. The problem of finding dense structures in a given graph is quite basic in informatics including data mining and data engineering. Clique is a popular model to represent dense structures, and widely used because of its simplicity and ease in handling. Pseudo cliques are natural extension of cliques which are subgraphs obtained by removing small number of edges from cliques. We here define a pseudo clique by a subgraph such that the ratio of the number of its edges compared to that of the clique with the same number of vertices is no less than a given threshold value. In this paper, we address the problem of enumerating all pseudo cliques for a given graph and a threshold value. We first show that it seems to be difficult to obtain polynomial time algorithms using straightforward divide and conquer approaches. Then, we propose a polynomial time, polynomial delay in precise, algorithm based on reverse search. The time complexity for each pseudo clique is $O(\Delta \log |V| + \min\{\Delta^2, |V| + |E|\})$. Computational experiments show the efficiency of our algorithm for both randomly generated graphs and practical graphs[1].

1 Introduction

Let us consider the problem of finding dense structures from a given graph, i.e., finding vertex induced subgraphs including many edges. In graphs representing similarity or relation, dense structures can be considered to represent groups of similar objects or deeply related objects. Thus, the problem is important in many scientific areas, such as bio-informatics[13], and Web science[11]. This is especially true in data engineering and data mining, where it is one of basic problems, and has many applications such as clustering, community discovering, machine learning, Web search, etc. [2, 8, 11–14, 18, 26].

A clique is a subgraph that is a complete graph. It is a fundamental structure for representing dense structures. It has several good mathematical properties, and can be easily handled. As a result, clique detection has been used in many researches[15, 1]. In particular, frequent pattern mining, which is a fundamental problem in data mining, is equivalent to the bipartite clique enumeration in the graph representing inclusion relation between transactions and items. Cliques are

[1] An extend abstract version is in [23]

considered as dense structures, and also considered as a part of dense structures, or seeds of dense structures. The clique enumeration can be efficiently carried out thanks to the increase of computation power, and new algorithms[16, 20]. Currently, the bottle neck of the computation in the practice is usually the output process or the post processing using the output, thus the algorithm can be said to be almost optimal.

As a next step, people wanted to use a richer model than cliques. In very sparse graphs, a subgraph containing only small cliques can be considered as a dense structure if it has many edges when compared to the others. If the data is incorrect so that some edges are missing, then a vertex set which we consider it should be a clique will not be a clique. For robust computation, "pseudo cliques" should be used. For example, such pseudo cliques are used for web page clustering[12].

We can consider several models to represent pseudo cliques. A possible model is a subgraph that is obtained from a clique by removing at most θ edges, where θ is a given threshold constant number. An advantage of this model is that in this definition any subset of a pseudo clique is also a pseudo clique in the sense of the vertex subset, thus the family of pseudo cliques satisfies the monotone property. This is a useful property, and we can use many techniques in the literatures to develop an efficient algorithm. However, a disadvantage of the model is that for graphs of any size the threshold is the same, thus larger subgraphs can lose only a small portion of its edges. This is contrary to the intuitions. Moreover, so many small vertex sets will become pseudo cliques.

Another model of a pseudo clique is a subgraph that has at least a constant ratio of edges compared to a clique of the same size. Precisely, we define the *density* of a subgraph by the number of edges over the number of all its vertex pairs. A subgraph is a pseudo clique if its density is no less than the given threshold value. In this definition, the family of pseudo cliques no longer satisfies the monotone property. It is a disadvantage of this definition. On the other hand, small subgraphs are pseudo cliques only if they are cliques, since the limitation of the number of edge removals changes as the size of subgraphs. This is an advantage of this definition.

In the literatures, a subgraph having many edges compared to the number of its vertices is often called *dense subgraph*, *heavy graph*, or *maximum edge subgraph*. However, all these terms are used to represent the subgraph having the highest density, thus it is mainly used in optimization. On the other hand, in many areas in informatics, a subgraph having many edges thereby similar to a clique is often called a "pseudo clique" or "quasi clique". In this paper, we use the term pseudo clique.

The problem of maximizing the density among subgraphs of a given size k is NP-complete since it includes the maximum clique problem as its special case[6]. However, if $k = \Theta(|V|)$ holds, there is a PTAS algorithm[3]. For the edge weighted version, an $O(|V|^{1/3-\epsilon})$ approximation algorithm is known[6]. However, finding an induced subgraph maximizing the average degree in it can be solved in polynomial time[9].

In this paper, we address the enumeration problem of all pseudo cliques in a given graph. We choose the latter definition for pseudo cliques. We first show that the existence of polynomial delay algorithm is non-trivial, by proving that straightforward back-tracking (branch-and-bound) approaches involve an NP-complete problem in each iteration. Note that it does not assure the non-existence of output polynomial time algorithm even when $P \neq NP$.

In contrast, a reverse search works well for this problem. We will show that for any pseudo clique, one of its vertex satisfies that its removal is also a pseudo clique. From this, we can obtain an adjacency relation on pseudo cliques spanning all the pseudo cliques. Then we define a tree-shaped traversal route on all pseudo cliques by this adjacency. Our algorithm traverses the route in a depth-first manner with taking polynomial time on each pseudo clique, thus it is polynomial delay algorithm. We then introduce a method to decrease the time complexity and practical computation time. The algorithm runs in $O(\Delta \log |V| + \min\{\Delta^2, |V| + |E|\})$ time for each pseudo clique with $O(|V| + |E|)$ memory. It also works for the edge-weighted version of the problem, in the same time complexity and space complexity. By computational experiments we show the efficiency of our algorithm in practice. An extended abstract version of this paper is published in [23]. Actually, one claim of the paper has an error, so the time complexity of the algorithm in the paper is slightly different. We give a correct statement and proof in this paper.

The organization of this paper is as follows. Section 2 is for preliminaries. In Section 3, we present the hardness result for straightforward approaches. Section 4 describes the adjacency relation of pseudo cliques and the enumeration algorithm. In Section 5, we present the results of our computational experiments, and conclude the paper in Section 6.

2 Preliminary

Let $G = (V, E)$ be a graph with a vertex set V and an edge set $E \subseteq V \times V$. In this paper we consider graphs with no multiple edge. We say a vertex v is *adjacent* to a vertex u if there is an edge $e = \{u, v\}$ in E. We denote the degree of v by $deg(v)$, and the maximum degree by Δ.

For a vertex set $U \subseteq V$, $E[U]$ denotes the set of edges both whose endpoints are in U, and the *vertex induced subgraph* by U, denoted by $G[U] = (U, E[U])$, is a graph composed of edges of G whose endpoints are both in U. $G[U]$ is also called an *induced subgraph.* In Fig. 1, the subgraph induced by vertex set $\{3, 5, 6, 9, 11\}$ is the graph inside the gray circle without edges one of its endpoint outside the circle. If $U = \emptyset$, we define $G[U]$ by the empty graph $(\emptyset, \emptyset)$. If any two vertices in U are connected by an edge, U and $G[U]$ are called a *clique.* In Fig 1, vertices $\{5, 7, 9, 11\}$ form a clique. For a vertex v and a vertex set U, we denote the number of edges connecting v and a vertex in U by $deg_U(v)$.

Let $clq(n) = \frac{n(n-1)}{2}$, where $clq(n)$ is the number of edges in the clique of n vertices. For a vertex subset $U \subseteq V$ at a size of at least 2, the *density* of $G[U]$ is defined by $|E[U]|/clq(|U|)$. The density is the ratio of the edges in $G[U]$

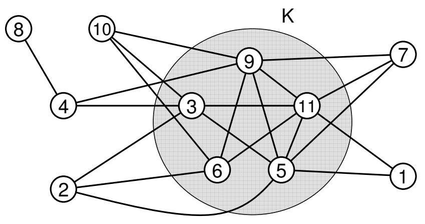


Fig. 1. An example of pseudo clique and other vertices

compared to the complete graph of $|U|$ vertices. In Fig 1, the density of the subgraph induced by $K = \{3, 5, 6, 9, 11\}$ is $7/10$. We define the density of $G[\emptyset]$ and graphs with one vertex by 1. Suppose that $\theta, 0 \leq \theta \leq 1$ be a constant number called *threshold*. Then, an induced subgraph $G[U]$ is called a *pseudo clique* if its density is no less than θ. We note that $G[\emptyset]$ and $G[\{v\}]$ for any vertex v are pseudo cliques for any threshold.

Let w be an edge weight function $w : E \rightarrow \Re$. For an edge subset $F \subseteq E$, we define the weight of F by the sum of weights of edges in F, and denote it by $w(F)$. We also define the weight of $G[U]$ by $w(E[U])$. For a given edge weight function $w : E \rightarrow \Re$, we define the *weighted density* of an induced subgraph $G[U]$ by $w(E[U])/clq(|U|)$. We define the weighted density of the graphs with at most one vertex by $+\infty$. For a threshold $\theta \in \Re$, an induced subgraph $G[U]$ is a *weighted pseudo clique* if its weighted density is no less than θ.

We define the (weighted) density of a vertex set by that of the subgraph induced by the vertex set. We often say that U is a (weighted) pseudo clique if $G[U]$ is a (weighted) pseudo clique. We here address the following enumeration problem.

(Weighted) Pseudo Clique Enumeration Problem:
For given a graph $G = (V, E)$ and a density threshold θ, (and edge weight function w), output all vertex sets whose induced subgraphs are (weighted) pseudo cliques of G, i.e., all vertex sets whose (weighted) densities are no less than θ

If an enumeration algorithm which outputs a set of solutions terminates in time polynomial of the sum of the input and output sizes, the algorithm is said to be *output polynomial time.* The longest computation time between the output of any two consecutive solutions is called the *delay.* An algorithm is *polynomial delay* if the delay is polynomial in the input size. The computation time of a polynomial delay algorithm is linear in the output size, thus it is optimal for the output size and considered to be practically efficient. Our goal in this paper is to develop a polynomial delay algorithm for the (weighted) pseudo clique enumeration problem.

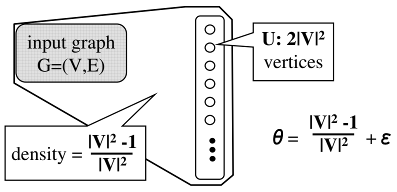


Fig. 2. Construction of the graph for reduction

3 Hardness Result for Straightforward Approaches

A popular approach for solving enumeration problems is so called binary partition, which is a variant of branch and bound method. For the problem of enumerating all the elements of an implicitly given subset family $\mathcal{F} \subseteq 2^E$, binary partition algorithm works in the following way. If $|\mathcal{F}|$ is less than a certain constant number, it enumerates elements in $\mathcal{F}$ directly. Otherwise, it chooses an element $e \in E$ and divide $\mathcal{F}$ into two sets $\mathcal{F}^+$ and $\mathcal{F}^-$ so that $\mathcal{F}^+$ consists of all the elements of $\mathcal{F}$ including e, and $\mathcal{F}^-$ consists of those not including e. Then, we check whether each of $\mathcal{F}^+$ and $\mathcal{F}^-$ is the emptyset or not, and if it is not empty, we enumerate the elements recursively, by dividing $\mathcal{F}^+$ or $\mathcal{F}^-$ by choosing another element e'.

The algorithms so called "depth first search" or "back tracking" are also considered to be a variant of binary partition algorithms. Binary partition works for many problems, such as spanning trees, paths, and cycles[19], cliques and independent sets[16], perfect matchings in bipartite graphs[7], frequent itemsets[24], frequent trees[4], etc.

The number of iterations of a binary partition algorithm is linear in $|\mathcal{F}|$, which is the size of output, thanks to the check whether either $\mathcal{F}^+$ or $\mathcal{F}^-$ is the emptyset. Thus, if the check can be done in polynomial time of the input size, the binary partition algorithm is polynomial delay. When we want to enumerate the pseudo cliques by binary partition, the check problem is defined as follows.

Problem: Pseudo Clique Existence
Given a graph G, a vertex subset U, and a threshold θ, check whether a pseudo clique K including U exists.

However, as we prove below, this problem is NP-complete. Thus, a straightforward binary partition algorithm will possibly take exponential time in an iteration. Note that this result does not assure that binary partition algorithms will never be output polynomial time unless $P = NP$. For example, a good ordering of vertices in V can exist so that the existence of pseudo cliques in any iteration can be checked in polynomial time. Moreover, even if one iteration takes exponential time in the input size, the total computation time can be polynomial in the output size since the output size can be also exponential in the input size. In general, it is not easy to prove such statements, but also not impossible.

Theorem 1. *The problem pseudo clique existence is NP-complete.*

Proof. The problem obviously belongs to NP, thus we prove NP-hardness by reducing the k clique problem which is to answer whether a given graph $G' = (V', E')$ includes a clique of a given size k. This problem is NP-complete[10].

We will construct a graph $G = (V, E)$ from G' as follows. Fig. 2 shows an example. Without loss of generality, we assume that G' has at least a certain number of vertices, say 10 vertices. Let $V = V' \cup U$ where U is a vertex set of size $2|V'|^2$. We choose two arbitrary vertices z_1 and z_2 from U. The edge set E is defined by

$$E = E' \cup \{(u, v) \mid v \in V', u \in U, u \neq z_1, z_2\} \cup E_U$$

where E_U is an arbitrary edge subset $E_U \subseteq U \times U$ such that $|E_U| = (2|V'|^2 - 1) \times (|V'|^2 - 1)$. Then, the density of $G[U]$ is equal to $(|V'|^2 - 1)/|V'|^2$.

Let K be a subset of V'. If and only if $G[K]$ is a clique, the density of $G[K \cup U]$ is greater than $(|V'|^2 - 1)/|V'|^2$, since the density of $G[\{v\} \cup U]$ is equal to $(|V'|^2 - 1)/|V'|^2$ for any $v \in V'$. For any $K \subset V$ such that $G[K]$ is a clique, the density of $G[K \cup U]$ is determined by the size of K. When K is the empty set or composed of one vertex, its density is $(|V'|^2 - 1)/|V'|^2$, and strictly increases as the increase of the size of K. If the size of K is k, the density is

$$\frac{clq(k) + (2|V'|^2 - 1) \times (|V'|^2 - 1) + k(2|V'|^2 - 2)}{clq(k + |V'|^2)}.$$

Thus, by setting θ to this value, the density of $G[U \cup K], K \subseteq V'$ is no less than θ if and only if K is a clique of at least size k in G'. Thus, the problem pseudo clique existence is NP-complete. □

4 Polynomial Delay Enumeration of Pseudo Cliques

In the previous section we showed that a straightforward approach to the pseudo clique enumeration may fail in the sense of the polynomiality. However, the problem can actually be solved in polynomial delay. In this section we describe a polynomial delay algorithm for the non-weighted version of our problem. A slight modifications adopt the weighted version of the problem. The key to an efficient enumeration is that we construct a tree-shaped traversal route on the set of pseudo cliques, and perform a depth first search on the tree, without having the traversal route explicitly on the memory. Such a transversal route can be obtained by looking at an adjacency relation on pseudo cliques spanning on all the pseudo cliques. This technique is called reverse search, which is originally developed by Avis and Fukuda[5].

The idea of reverse search is as follows. We first define a parent for each element to be enumerated, except for a specified element called the *root*. The definition of the parent has to be acyclic, that is, each element is not a proper ancestor of itself. Then, the parent-child relation induces a spanning tree rooted at the root on the set of elements to be enumerated. We call the tree the *enumeration tree*. The algorithm traverses the tree in a depth first manner. Reverse search does not need to memorize the previously visit elements in memory space to avoid duplications. It uses an algorithm for listing the children of an element. By finding a child of an element, we can go deeper on the enumeration tree by a recursive call. After we come back from the recursive call, we find the next child and make a recursive call for it. In this way, we can perform the depth first search with memory space linear in the height of the enumeration tree[2].

First we observe the following property to obtain an adjacency relation on pseudo cliques.

Lemma 1. *Let v be a vertex in $G[K]$ with the degree no greater than the average of the degrees of vertices in $G[K]$. The density of $K \setminus \{v\}$ is no less than the density of K.*

Proof. If $|K| = 1$, then $K \setminus \{v\} = \emptyset$, thus the statement holds. Hence, we assume that $|K| \geq 2$. Let F_1 and F_2 form a partition of the set of pairs of vertices in K such that F_1 is the set of pairs of v and another vertex, and F_2 is the set of the remaining pairs. Let $E_1 = E[K] \cap F_1$, and $E_2 = E[K] \cap F_2$. The density of K multiplied by $|K| - 1$ is equal to the average degree of vertices in K, and thus it is no less than $|E_1|/|F_1|$. Therefore, we can observe that the density of K is between $|E_1|/|F_1|$ and $|E_2|/|F_2|$. Since $E[K \setminus \{v\}] = E[K] \cap F_2$, the density of $K \setminus \{v\}$ is no less than the density of K. It concludes the proof. $\square$

From the lemma, we can see that for any pseudo clique K, $K \setminus \{v\}$ is also a pseudo clique for any vertex v whose degree is no greater than the average. This introduces an adjacency relation on pseudo cliques. Since any K has such a vertex v, we can remove the vertices of K iteratively until K will be the emptyset, passing through only pseudo cliques. Thus, the adjacency spans all the pseudo cliques. The graph induced by the adjacency is not a tree, thus we introduce parent-child relation to obtain a tree-shaped traverse route.

For a vertex set $K \neq \emptyset$, we define $v^*(K)$ by the minimum degree vertex in $G[K]$. If there are more than one minimum degree vertices, we choose the minimum index one. We define the parent $Prt(K)$ of $K \neq \emptyset$ by $K \setminus \{v^*(K)\}$. From Lemma 1, $Prt(K)$ is a pseudo clique if K is a pseudo clique. Since any K has a unique parent, the graph induced by the parent-child relation forms a tree. Let us see an example. In Fig 1, the degrees of vertices in $G[K]$ are $deg_K(3) = deg_K(6) = 2, deg_K(5) = deg_K(9) = 3, deg_K(11) = 4$, thus $v^*(K)$ is vertex 3. The parent of $K = \{3, 5, 6, 9, 11\}$ is $\{5, 6, 9, 11\}$. Note that the definition of the parent does not depend on the threshold value, thus the parent is identical for any threshold value.

[2] The space complexity of the original reverse search does not depend on the height of the enumeration tree.

Now the remaining task is how to find the children of a pseudo clique. For this task, we first observe the following property, immediately obtained from the definition of the parent.

Property 1. For a pseudo clique $K \subseteq V$, K' is a child of K if and only if $K' \setminus K = \{v^*(K')\}$.

From the property, we can see that K has at most $|V| - |K|$ children, each of which is obtained by adding a vertex v not in K to K. Thus, we can list the children of K by computing the density of $K \cup \{v\}$ and $v^*(K \cup \{v\})$ for each vertex $v \notin K$. This results a polynomial delay enumeration algorithm. We describe the algorithm as follows, which enumerates all pseudo cliques by calling with G and $K = \emptyset$.

```
**Algorithm** EnumPseudoClique ($G = (V, E)$, $K$)
1: **output** $K$
2: **for each** $v \notin K$ **do**
3:   **if** $K \cup \{v\}$ is a pseudo clique **then**
4:     **if** $v = v^*(K \cup \{v\})$ **then** EnumPseudoClique ($G = (V, E)$, $K \cup \{v\}$)
```

In a straightforward implementation, an iteration of the algorithm takes $O(|V|^2)$ time, thus pseudo cliques can be enumerated $O(|V|^2)$ time for each. The computation time can be reduced by a sophisticated process.

We characterize vertices generating children in terms of deg_K. Let $\theta(K) = \theta clq(|K|+1) - |E[K]|$. The density of $K \cup \{v\}$ is $(|E[K]| + deg_K(v))/clq(|K|+1)$, thus the following lemma holds.

Lemma 2. *Let $K \subseteq V$ be a pseudo clique and v be a vertex not in K. $K \cup \{v\}$ is a pseudo clique if and only if $deg_K(v) \geq \theta(K)$*

$\theta(K)$ is the threshold value for $deg_K(v)$ such that $K \cup \{v\}$ is a pseudo clique, thus efficient search of vertices v satisfying $deg_K(v) \geq \theta(K)$ is a key to efficient enumeration.

To check whether $v^*(K \cup \{v\}) = v$ or not, we introduce a total order $\prec_U$ for any vertex subset $U \neq \emptyset$, defined by

$$u \prec_U v \quad \Leftrightarrow \quad deg_U(u) < deg_U(v) \quad \text{or} \quad (deg_U(u) = deg_U(v) \quad \text{and} \quad u \leq v),$$

here $u \leq v$ means that the index of u is less than that of v. Then, $v^*(K)$ is the vertex of K satisfying $v^*(K) \prec_K u$ for any other vertex $u \in K$. Then, we have the following lemma.

Lemma 3. *For any pseudo clique K and a vertex v not in K, $K \cup \{v\}$ is a child of K if and only if*
(1) $K \cup \{v\}$ is a pseudo clique,
(2) the tuple $(deg_K(v^*(K)), v^*(K))$ is lexicographically larger than the tuple $(deg_K(v) - 1, v)$, and
(3) v is adjacent to any vertex $u \in K, u \prec_K v$.

Proof. We first observe that for any vertex u, $deg_{K\cup\{v\}}(u) = deg_K(u) + 1$ if v is adjacent to u, and $deg_{K\cup\{v\}}(u) = deg_K(u)$, otherwise. Then, the only if part of the statement is obvious; if (2) is violated, then $v^*(K) \prec_{K\cup\{v\}} v$. Hence we prove the if part. It is sufficient to prove that conditions (2) and (3) lead that $v^*(K \cup \{v\})$ is v. $v^*(K \cup \{v\})$ is v when any vertex $u \in K$ satisfies $v \prec_{K\cup\{v\}} u$.

Suppose that conditions (2) and (3) hold for v. Then, from condition (3), any vertex $u \in K$ satisfying $u \prec_K v$ is adjacent to v. Thus, the vertex u satisfies $deg_{K\cup\{v\}}(u) = deg_K(u) + 1$. Together with condition (2), the tuple $(deg_{K\cup\{v\}}(u), u)$ is lexicographically larger than $(deg_{K\cup\{v\}}(v), v) = (deg_K(v), v)$. Thus, the statement holds. □

We show an example in Fig. 1. $K \cup \{v\}$ is a child of K if v is either $1, 2$ or 4. Vertex 7 does not satisfy (3), and vertex 10 does not satisfy (2).

If a vertex v satisfies that $K \cup \{v\}$ is a pseudo clique and $v \prec_K v^*(K)$, then $K \cup \{v\}$ is always a child of K. To find such vertices efficiently, we maintain a binary tree which stores all vertices on its leaves in the order of $\prec_K$. We start at the leaf corresponding to $v^*(K)$, and trace the leaves in the decreasing order of $\prec_K$, until we meet a vertex v satisfying $deg_K(v) < \theta(K)$. In this way, we can find all vertices preceding to $v^*(K)$ and satisfying $deg_K(v) \geq \theta(K)$, in $O(\log |V|)$ time for each.

When we add/delete a vertex v to/from K and obtain a new vertex set K', we update the binary tree with keeping the order on leaves. By the addition/deletion of v, $(deg_K(u), u)$ and $(deg_{K'}(u), u)$ differ only for the vertices adjacent to v. Thus, the update can be done in $O(\Delta \log |V|)$ time. In summary, these operations take $O(\Delta \log |V|)$ time for each iteration, or equivalent to, for each child.

The remaining task is to find vertices v such that $K \cup \{v\}$ is a child and $v \prec_K v^*(K)$ does not hold. For each vertex $v \notin K$, we define $l(v, K)$ by the first vertex in K which is not adjacent to v, in the order of $\prec_K$. In the case that v is adjacent to all vertices, $l(v, K)$ is not defined. In Fig. 1, $l(2, K) = 9$ and $l(7, K) = 3$.

Lemma 4. *Let $v \notin K$ be a vertex satisfying that $v^*(K) \prec_K v$ and $K \cup \{v\}$ is a pseudo clique. Then, $K \cup \{v\}$ is a child of K if and only if $v \prec_K l(v, K)$, and the tuple $(deg_K(v^*(K)), v^*(K))$ is lexicographically larger than the tuple $(deg_K(v) - 1, v)$. Here we suppose that when $l(v, K)$ is not defined, $v \prec_K l(v, K)$ always holds. In particular, v is adjacent to $v^*(K)$.*

Proof. The condition $v \prec_K l(v, K)$ is equivalent to condition (3) of Lemma 3, thus the lemma holds. □

Among the vertices satisfying the conditions of Lemma 4, the vertices v satisfying $l(v, K) = v^*(K)$ can be found by the above operation, since $v \prec_K v^*(K)$. Thus, we are going to find all vertices $l(v, K) \neq v^*(K)$, among them. These vertices are adjacent to v^*, thus we have to check at most Δ vertices. For any vertex v, $l(v, K)$ is always in the first $\Delta + 1$ vertices of K, in the order of $\prec_K$. We keep the list of the vertices of K sorted in the order of $\prec_K$ and update it at each change of K, then we can compute $l(v, K)$ in $O(\Delta)$ time. Thus, the time to compute $l(v, K)$ for all vertices adjacent to $v^*(K)$ is bounded by $O(\Delta^2)$. Since in the computation an edge is accessed at most once, thus this time can also be bounded by $O(|V| + |E|)$. The update of the sorted list needs $O(\Delta \log |V|)$ time for an addition/deletion of a vertex of K, thus it is not the bottleneck part of the time complexity. Thus, we can see that the time complexity of the algorithm is $O(\Delta \log |V| + \min\{\Delta^2, |V| + |E|\})$ for each pseudo clique. Although the algorithm needs $O(|V| \log |V| + |E|)$ time for preprocessing, it is always smaller than the main computation time since the number of pseudo cliques is always larger than the number of vertices and edges.

The weighted version of the problem can be solved in the same way. For each vertex v and vertex set K, we define $w_K(v)$ by the sum of the weight of edges connecting v and vertices in K. Using this, we define $v^*(K)$ by the vertex in K which minimizes w_K. Ties are broken by choosing the minimum index one. The parent is defined in the same way, and we can construct an enumeration algorithm in the same framework. Thus we obtain the following theorem.

Theorem 2. *For a given graph G, (edge weight function w) and density threshold value θ, all (weighted) pseudo cliques can be enumerated in $O(\Delta \log |V| + \min\{\Delta^2, |V| + |E|\})$ time for each with $O(|V| + |E|)$ memory. In particular, the delay is $O(\Delta \log |V| + \min\{\Delta^2, |V| + |E|\})$.*

We note that the delay can be bounded by the computation time for one iteration by using so called "odd-even output method", described in [17, 22]. We modify the algorithm so that in each iteration, the algorithm outputs the solution before generating recursive calls if the depth of the recursion is odd, and after the recursive calls otherwise. By this, during the execution, any three consecutive iterations output at least one solution, thus the delay is reduced to be equal to the computation time for one iteration.

Considering the practice, the estimation of the computation time is too large, since we are usually given a possibly large but sparse graphs thereby the pseudo cliques are small in comparison to the graph sizes. Otherwise the number of pseudo cliques explodes so that we can not handle the output. Thus, there will be few candidates for children, and few vertices adjacent to $v^*(K)$. Thus the complexity we state here is possibly far from the practical computation time.

5 Computational Experiments

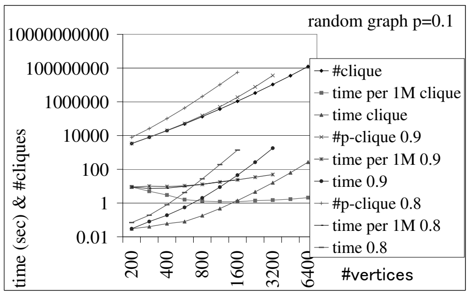
We here present the results of computational experiments to show the practical efficiency of our algorithm. We implemented the non-weighted version of our algorithm. The implementation is coded by C, compiled by gcc, and executed in a notebook PC with a Pentium M 1.1GHz processor with 256MB memory with cygwin which is an emulator of Linux environments on Windows. The implementation is a simpler version of our algorithm, which maintains only $deg_K(v)$ for each vertex v, and the sets of vertices having the same value of $deg_K(v)$, thus the worst computation time for an iteration is $O(|E| + |V|)$. The reason that we did not use the technique described in the previous section is that in practice we expected that only few vertices satisfy $deg_K(v) = deg_K(v^*(K))$ on average, and pseudo clique is not large on average. This was actually observed in the computational experiments.

We examined several types of graphs as inputs of the implementation, randomly generated graphs and graph taken from real world data. We had three groups of random graphs which are generated in the following three different ways. The first group consists of ordinary random graphs. For each pair of vertices, we connected them by an edge with the same probability 0.1. The second group is of so called locally dense graphs. Consider a necklace sequence obtained by connecting the head and the tail of the vertex sequence $(1, 2, \ldots, |V|)$. For each vertex v, we connected it to each of its neighboring vertices with probability $1/2$. Here a neighbor of v is a vertex u with $|u - v| \leq r$, or $|u - v| \geq |V| - r$, for given r. In the experiments we set $r = 20$.

The third was randomly generated scale free graphs. In a scale free graph, the probability that the degree of a vertex is λ is $1/\lambda^{\Gamma}$. Such a distribution is called zip distribution, and such data is said to satisfy power law. The graphs appearing in real world problems are often scale free graphs. Our scale free graphs are generated by starting from a clique of k vertices and adding vertices one-by-one to it, and then connect it to k vertices. The vertices to be connected are chosen randomly such that a vertex is chosen with a probability proportional to its degree. A graph generated in this way is known to be scale free. The graph tends to have few locally dense structures which we can see many in real world data, thus the average size of cliques in this graph is often small.

We run the implementations for these graphs with the thresholds $\theta = 0.8$ and $\theta = 0.9$. Since we could not find any implementation for the pseudo clique enumeration problem, we compare the performance to that of an ordinary backtracking clique enumeration algorithm, which recursively adds vertices larger than any vertex in the current clique. We maintain $|deg_K(v)|$ to find the vertices which can be added to the current clique, since $K \cup \{v\}$ is a clique if and only if $|deg_K(v)| = |K|$ holds. Since the clique enumeration is a special case of our problem, the performance of the clique enumeration algorithm can be considered as a kind of upper bound of the performance of the pseudo clique enumeration algorithm.

The results of the experiments are shown in Figure 3 and 4. The left side of Fig. 3 shows the results for ordinary random graphs, and the right side of Fig. 3 is for locally dense random graphs. The left side of Fig. 4 is for scale free graphs. The horizontal axis is the number of vertices in the input graphs, and the vertical axis is for the computation time, computation time for 1 million (pseudo) cliques, and the number of output (pseudo) cliques. All these are shown in log scales. The lines almost horizontal in the figures display the computation time per one million pseudo cliques. The computation time for each pseudo clique does not change with the change in the threshold value θ, and does not differ very much from that of the clique enumeration. This means that the performance of the pseudo clique enumeration is close to optimal, thus in the practice a high performance is expected.



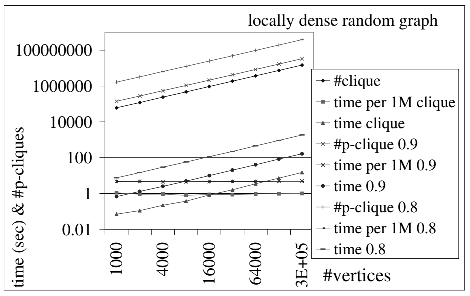
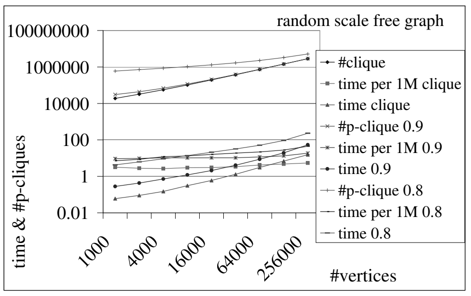


Fig. 3. Results for random graphs (left) and locally dense random graphs (right)



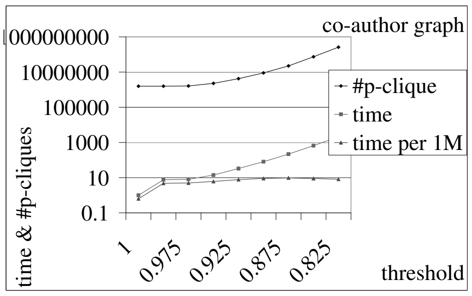


Fig. 4. Results for scale free graphs (left) and co-author graph (right)

The computation time is increasing with the increase of the graph sizes in the scale free graphs and slightly increasing in random graphs. This is possibly because of the increase in the average degree, and the decrease in the ratio of children in the candidates of children. When we set threshold value θ to a small value, the number of vertices v satisfying $deg_K(v^*(K)) \leq deg_K(v) \leq deg_K(v^*(K)) + 1$ increases. Since scale free graphs have a kind of locality, few of them will be the children of K, on average.

For all the results, the diagonal lines represent the numbers of pseudo cliques and cliques. For the random graphs, the ratio of these three values increases with the increase in graph size. This could be because of the increase in the average degree. It is interesting to note that the ratio does not change much in locally dense random graphs, and is reduced for the scale graphs.

The right side of Fig. 4 is the result for the graph taken from the real world data. The graph is a co-author graph[25] such that each vertex is a researcher and two researchers are connected if they have a joint paper. The number of vertices is about 30,000 and the number of edges is about 125,000. It is known that the graph is a scale free graph. For this graph we observe the results by changing the threshold θ. The leftmost point indicates the computation time of the clique enumeration, thus it is faster than the others, but not different much from that of the pseudo clique enumeration. The computation time per each pseudo clique does not seem to depend on the threshold value.

6 Conclusion

In this paper we addressed the problem of finding dense structures from a graph. The density is given by the ratio of the existing edges compared to a complete graph, and we define a pseudo clique as a dense structure by a subgraph with density no less than the given threshold value. In this term we define our problem of enumerating all pseudo cliques of given a graph and a threshold.

We first showed that it is not easy to get polynomial time algorithm by straightforward approaches since in this way we have to solve an NP-complete problem in each iteration. On the other hand, we showed that any pseudo clique has a proper subset being a pseudo clique with one fewer vertices. This induces a relation spanning all pseudo cliques. Using the relation we developed a reverse search algorithm whose delay is $O(\Delta \log |V| + \min\{\Delta^2, |V| + |E|\})$, thus computation time for each pseudo clique is $O(\Delta \log |V| + \min\{\Delta^2, |V| + |E|\})$. This also works for the weighted version.

Recently, it has become popular to use dense structures to represent related objects. One of the problems on practice is that the number of output solutions is often larger than that of the cliques. The "maximal" pseudo clique enumeration may help, but it is not straightforward to introduce maximality because the family of pseudo cliques does not satisfy the monotone property. Detailed characterizations of the dense structures which would allow us to develop efficient algorithms are important for applications, and an interesting open problem.

Acknowledgment

This research was partly supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Specially Promoted Research, 18017015 for developing high-speed high-quality algorithms for analyzing huge genome database.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, Fast Discovery of Association Rules, *In Advances in Knowledge Discovery and Data Mining,* MIT Press, pp. 307–328, 1996.
2. J. Aslam, K. Pelekhov and D. Rus, A Practical Clustering Algorithms for Static and Dynamic Information Organization, *Symposium on Discrete Algorithms (SODA) 99*, ACM Press, pp. 51–60, 1999.
3. S. Arora, D. Karger and M. Karpinski, Polynomial time approximation schemes for dense instances of NP-hard problems, *Proceedings of ACM Symposium on Theory of Computing*, pp. 284–293, 1995.
4. Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Sakamoto and Setsuo Arikawa, Efficient Substructure Discovery from Large Semi-structured Data, *In Proceedings of SDM 2002*, 2002.

5. D. Avis and K. Fukuda, Reverse Search for Enumeration, *Discrete Applied Mathematics* **65**, pp. 21–46, 1996.
6. U. Feige, D. Peleg and G. Kortsarz, The Dense k-subgraph Problem, *Algorithmica* **29**, pp. 410–421, 2001.
7. K. Fukuda and T. Matsui, Finding All Minimum-Cost Perfect Matchings in Bipartite Graphs, *Networks* **22**, pp. 461–468, 1992.
8. K. Fujisawa, Y. Hamuro, N. Katoh, T. Tokuyama and K. Yada, Approximation of Optimal Two-Dimensional Association Rules for Categorical Attributes Using Semidefinite Programming, *Lecture Notes in Computer Science* **1721**, pp. 148–159, 1999.
9. G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, A Fast Parametric Maximum Flow Algorithm and Applications, *SIAM Journal on Computing* **18**, pp. 30–55, 1989.
10. M. R. Garey, D. S. Johnson and L. Stockmeyer, Some Simplified NP-complete Problems, *Proceedings of ACM Symposium on Theory of Computing*, pp. 47–63, 1974.
11. D. Gibson, R. Kumar and A. Tomkins, Discovering Large Dense Subgraphs in Massive Graphs, *Proceedings of Very Large Data Bases Conference*, pp. 721–732, 2005.
12. M. Haraguchi, Y. Okubo, A Method for Clustering of Web Pages with Pseudo-Clique Search, *Lecture Notes in Artificial Intelligence* **3847**, pp. 59–78, 2006.
13. H. Hu, X. Yan, Y. Huang, J. Han and X. J. Zhou, Mining Coherent Dense Subgraphs Across Massive Biological Networks for Functional Discovery, *Bioinformatics* **21**, pp. 213–221, 2005.
14. R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins Extracting Large-Scale Knowledge Bases from the Web, *Proceedings of Very Large Data Bases Conference*, pp. 639–650, 1999.
15. S. R. Kumar, P. Raphavan, S. Rajagopalan and A. Tomkins, Trawling the Web for emerging cyber communities, *In Proceddings of 8th International WWW Conference*, pp. 1481–1493, 1999.
16. K. Makino and T. Uno, New Algorithms for Enumerating All Maximal Cliques, *Lecture Notes in Computer Science* **3111**, pp. 260–272, 2004.
17. S. Nakano and T. Uno, Constant Time Generation of Trees with Specified Diameter, *Lecture Notes in Computer Science* **3353**, pp. 33–45, 2004.
18. G. Palla, I. Derenyi, I. Farkas and T. Vicsek, Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society, *Nature*, **435** 7043, pp. 814–818, 2005.
19. R. C. Read and R. E. Tarjan, Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees, *Networks* **5**, pp. 237–252, 1975.
20. E. Tomita, A. Tanaka and H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments, *Theoretical Computer Science* **363**, pp.28–42, 2006.
21. T. Uno, Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs, *Lecture Notes in Computer Science* **1350**, pp. 92–101, 1997.
22. T. Uno, Two General Methods to Reduce Delay and Change of Enumeration Algorithms, *National Institute of Informatics (in Japan) Technical Report*, 004E, 2003.
23. T. Uno, An Efficient Algorithm for Enumerating Pseudo Cliques, *Lecture Notes in Computer Science* **4835**, pp. 402–414, 2007.
24. T. Uno, M. Kiyomi, H. Arimura, "LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets", In *Proceedings of IEEE*

ICDM'04 Workshop FIMI'04, available at http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-126/, 2004.

25. S. Warner, E-prints and the Open Archives Initiative, *Library Hi Tech* **21**, pp. 151–158, 2003.
26. Y. Zhang, C. H. Chu, X. Ji and H. Zha, Correlating Summarization of Multisource News with k Way Graph Biclustering, *ACM SIGKDD Explorations Newsletter* **6**, pp. 34–42, 2004.