February 20, 2002
RT0448
10 pages

Research Report

A Fast Algorithm for Mining Frequent Connected Subgraphs

Akihiro Inokuchi, Takshi Washio(Osaka Univ), Kunio Nishimura(Osaka Univ) and Hiroshi Motoda(Osaka Univ)

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A Fast Algorithm for Mining Frequent Connected Subgraphs

Akihiro Inokuchi INOKUCHI@JP.IBM.COM
Tokyo Research Laboratory, IBM Japan, 1623-14, Shimotsuruma, Yamato, Kanagawa, 242-8502, Japan

Takashi Washio WASHIO@AR.SANKEN.OSAKA-U.AC.JP
Kunio Nishimura NISHIMURA@AR.SANKEN.OSAKA-U.AC.JP
Hiroshi Motoda MOTODA@AR.SANKEN.OSAKA-U.AC.JP
Institute of Scientific and Industrial Research, Osaka University, 8-1, Mihogaoka, Ibaraki, Osaka, 567-0047, Japan

Abstract

Graph structured data mining to derive frequent subgraphs from a graph dataset is difficult because the search for subgraphs is combinatorially explosive, and includes subgraph isomorphism matching. The past research on this issue has not show the sufficient performance for practical use. In this paper, we propose a new approach called AcGM which achieves the complete search of frequent connected (induced) subgraphs in a massive labeled graph dataset within highly practical time. The power of AcGM comes from the algebraic representation of graphs, its associated operations and well-organized constraints to limit the search space efficiently. Its performance has been evaluated through some artificial simulation data and real world data. The high scalability of AcGM has been confirmed for the amount of data, the complexity of graphs and the computation time.

1. Introduction

Graph structured data mining which discovers frequent subgraph patterns embedded in a graph dataset is an important problem having broad applications. This problem is very difficult to solve in practical time because the search of possible subgraph patterns is combinatorially explosive, and includes subgraph isomorphism matching which is known to be NP-complete. For example, WARMR is a system to mine frequent subgraphs based on inductive logic programming (ILP) [Dehaspe 98]. It could derive all subgraphs consisting of only a few vertices within tractable time under its direct application to the graph dataset without data preprocessing.

To alleviate the difficulty of the computation time, some other approaches have introduced approximations. The most well known approximation is to apply the greedy search as SUBDUE [Cook 94] and GBI [Yoshida 95, Motoda 97]. However, this is not very suitable to many applications requiring the complete search of the results. Another approach is to limit the graphs to be searched within simpler classes. The levelwise version space algorithm proposed by De Raedt limits the search space to frequent paths for the derivation of the result in tractable time [De Raedt 01]. However, this is also weak for some practical use since the class of the structure to be mined is paths but not subgraphs.

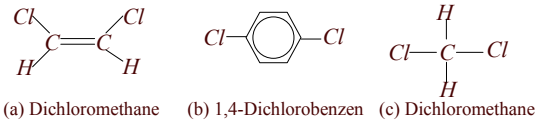
To overcome these limitations, we proposed an approach named AGM (Apriori-based Graph Mining) in which the knowledge representation and the search operations are highly dedicated to the graph structure mining [Inokuchi 00]. It can efficiently discover all frequent patterns in terms of induced subgraphs contained in a dataset of labeled graphs. The induced subgraph of a graph G has a subset of the vertices of G and the same edges between pair of vertices as in G. An induced subgraph can be an unconnected graph consisting of some isolated graph fragments. Though AGM is designed to work efficiently in comparison with the aforementioned work, it still requires intractable computation time for many practical scale problems. However, since the patterns under interested in many applications are connected graphs, the limitation of the search to connected subgraphs does not reduce the applicability of the graph structured data mining significantly. Recently, FSG which derives the complete set of frequent connected subgraphs included in a given graph dataset has been proposed [Kuramochi 01]. Because this uses a data structure consisting of many address pointers to represent the graph structure and many data ID pointers on working memory in a com-

puter, the scalability on graph size, number of data and data processing speed is quite limited.

In this paper, we propose a new approach called AcGM (Apriori-based connected Graph Mining) to discover only frequent connected graphs in terms of subgraphs or induced subgraphs in a graph dataset. The basic principles and the graph representation used in this approach follow these of AGM. However, some important changes and improvements have been introduced to search only connected (induced) subgraphs very efficiently. The algorithm of this approach works much faster than our AGM and FSG.

2. Graph and Problem Definitions

The input to AcGM is graphs in which each vertex and each edge have a vertex label and an edge label respectively, and any self-looped vertex is not included as shown in Fig. 1.



(a) Dichloromethane (b) 1,4-Dichlorobenzen (c) Dichloromethane

Figure 1. Example of Graph.

Definition 1 (Labeled Graph without Self-loops) *When a set $V(G)$ of vertices, a set $E(G)$ of edges, a set $L_V(V(G))$ of vertex labels and a set $L_E(E(G))$ of edge labels are provided as*

$$V(G) = \{v_1, v_2, \ldots, v_k\},$$

$$E(G) = \{e_h = (v_i, v_j) | v_i, v_j \in V(G), i \neq j\},$$

$$L_V(V(G)) = \{lb(v_i) | \forall v_i \in V(G)\},$$

$$L_E(E(G)) = \{lb(e_h) | \forall e_h \in E(G)\}$$

respectively, graph G is expressed as

$$G = (V(G), E(G), L_V(V(G)), L_E(E(G))).$$

If e_h is undirected, both (v_i, v_j) and (v_j, v_i) belong to $E(G)$. The number of vertices, $|V(G)|$, is called the size of graph G.

Though the frequent patterns to be derived in the final result of AcGM are connected graphs, the following semi-connected graphs are required to be searched for to perform the complete search in the intermediate process of AcGM.

Definition 2 (Connected Graph) *Given an undirected graph G, if a path exists between any two vertices, G is called a connected graph. In the case that G is a directed graph, the above definition applies to the undirected graph obtained by ignoring directions of edges in G.*

Definition 3 (Semi-connected Graph) *G is called a semi-connected graph, if G is a connected graph or a graph consisting of a connected subgraph and an isolated vertex.*

Graph structure can be expressed using an adjacency matrix.

Definition 4 (Adjacency Matrix) *Given a graph $G = (V, E, L_V, L_E)$, (i,j)-element $x_{i,j}$ of an adjacency matrix X_k of graph G whose size is k is represented as follows.*

$$x_{i,j} = \begin{cases} num(lb(e_h)) & \text{if } e_h = (v_i, v_j) \in E(G) \\ 0 & \text{if } (v_i, v_j) \notin E(G) \end{cases},$$

where $i, j \in \{1, \cdots, k\}$, $num(lb(e_h))$ and $num(lb(v_i))$ are natural numbers assigned to an edge label $lb(e_h)$ and a vertex label $lb(v_i)$ respectively for calculation efficiency.

The vertex corresponding to the i-th row (i-th column) of an adjacency matrix is called the i-th vertex, and the graph structure of an adjacency matrix X_k is represented as $G(X_k)$. The adjacency matrix differs depending on the assignment of rows and columns to vertices of the graph. In other words, an identical graph structure can be represented by multiple adjacency matrices. Such adjacency matrices are mutually convertible using the following transformation matrix.

Definition 5 (Transformation Matrix) *When adjacency matrices X_k and Y_k representing an identical graph structure of size k are given, a transformation matrix W_k is defined as follows.*

$$w_{ij} = \begin{cases} 1 & i\text{-th vertex of } G(X_k) \text{ corresponds} \\ & \text{to } j\text{-th vertex of } G(Y_k) \\ 0 & \text{otherwise} \end{cases}.$$

Y_k is expressed as $Y_k = W_k^T X_k W_k$.

In order to reduce the candidates of the frequent (induced) subgraphs which will be mentioned later, the code of an adjacency matrix is defined as follows.

Definition 6 (CODE) *In case of an undirected graph, the code of an adjacency matrix X_k is defined as*

$$code(X_k) = x_{1,2} x_{1,3} x_{2,3} x_{1,4} \cdots x_{k-2,k} x_{k-1,k}$$

by using (i,j)-element $x_{i,j}$. In case of a directed graph, it is defined as

$$code(X_k) = c_1 c_2 c_3 \cdots c_{\frac{k(k-1)}{2}}$$

$c_{\frac{j(j-1)}{2}-(j-i-1)} = (|L_E(E(G))|+1)x_{j,i} + x_{i,j} \quad (i<j)$

Furthermore, CODE including the vertex labels is defined as

$$CODE(X_k) = num(v_1)\cdots num(v_k)code(X_k).$$

The definition of the *code* is identical with that of AGM, whereas AcGM uses the extended *CODE*. This extension enables the matching of vertices within this *CODE* since it includes the information of vertices, and increases the search efficiency. The canonical form of the adjacency matrices is defined to uniquely choose an adjacency matrix among many matrices that represent an identical semi-connected graph.

Definition 7 (Canonical Form) *Representing the upper left $i \times i$ submatrix of adjacency matrix X_k as X_i $(1 \leq i \leq k)$, a set $\Gamma(G)$ of adjacency matrices representing an identical graph G will be defined.*

$$\Gamma(G) = \{X_k | G(X_i) \text{ is connected, } \forall i = 1,..,k-1, G \equiv G(X_k)\}.$$

The adjacency matrix C_k whose CODE is the largest in $\Gamma(G)$ is called the canonical form.

$$C_k \quad w.r.t \quad CODE(C_k) = \max_{X_k \in \Gamma(G)} CODE(X_k)$$

This definition is also changed from that of AGM where the canonical form is the matrix having the minimum code. This is due to another change on the definition of *normal form* as described later.

When graph shown in Fig. 1(a) is given, where 1, 2 and 3 are assigned to vertex labels H(hydrogen), C(carbon) and Cl(chlorine) respectively, and 0, 1, 2 and 3 are assigned to no bond, a single bond, a double bond and an aromatic bond, its canonical form is expressed as

$$C_6 = \begin{array}{c} \\ Cl \\ C \\ Cl \\ C \\ Cl \\ H \end{array} \begin{array}{c} \begin{array}{cccccc} Cl & C & Cl & C & Cl & H \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}.$$

The method to find the canonical form efficiently is mentioned latter. CODE of C_6 is represented as

$$CODE(C_6) = 323231101020000100010,$$

and this is the maximum number among all *CODE*s for the graph of Fig. 1(a).

Definition 8 (Subgraph) *Given a graph $G = (V(G), E(G), L_V(V(G)), L_E(E(G)))$, subgraph $G_s = (V(G_s), E(G_s), L_V(V(G_s)), L_E(E(G_s)))$ of G fulfills the following conditions.*

$$V(G_s) \subseteq V(G), E(G_s) \subseteq E(G).$$

Definition 9 (Induced Subgraph) *Given a graph $G = (V(G), E(G), L_V(V(G)), L_E(E(G)))$, subgraph $G_s = (V(G_s), E(G_s), L_V(V(G_s)), L_E(E(G_s)))$ of G fulfills the following conditions.*

$$V(G_s) \subseteq V(G), E(G_s) \subseteq E(G),$$

$$\forall u, v \in V(G_s), (u,v) \in E(G_s) \Leftrightarrow (u,v) \in E(G)).$$

When G_s is a subgraph or an induced subgraph of G, it is said that G_s is *included* in G, *i.e.*, $G_s \in G$.

Definition 10 (Support) *Given a set GD of graph structured data, the support $sup(G_s)$ of an (induced) subgraph G_s is defined as a ratio of the number of graph data including G_s to the total number of graph data in the dataset GD.*

The derived (induced) subgraph having the support more than the *minimum support* specified by a user is called a *frequent (induced) subgraph*. A frequent (induced) subgraph whose size is k is called a k-frequent (induced) subgraph.

Definition 11 (Problems) Connected Subgraph Derivation Problem *When the dataset which consists of graph structured data and the minimum support are given, the problem is to derive all frequent connected subgraphs that have the support more than the minimum support value in the dataset.*
Connected Induced Subgraph Derivation Problem *When the dataset which consists of graph structured data and the minimum support are given, the problem is to derive all frequent connected induced subgraphs that have the support more than the minimum support value in the dataset.*

For example, in the connected subgraph derivation problem for the dataset shown in Fig. 1, $sup(Cl\text{–}C) = \frac{3}{3}$ and $sup(Cl\text{–}C\ C) = \frac{3}{3}$. On the other hand, in the connected induced subgraph derivation problem, $sup(Cl\text{–}C) = \frac{3}{3}$ and $sup(Cl\text{–}C\ C) = \frac{2}{3}$.

3. Algorithm

3.1 Outline

AcGM algorithm proposed in this paper is obtained through the changes of AGM to search for connected

(induced) subgraphs only and the improvements of AGM to enhance the efficiency where AGM is the extension of Apriori algorithm [Agrawal 94] to graph structured data. Apriori algorithm derives all frequent itemsets in ascending order of the size of the itemset based on the monotonic property that the support of an itemset is less or equal to the supports of its subsets.

Similarly,

$$sup(G) \leq sup(G_s) \tag{1}$$

holds on the connected (induced) subgraph derivation problem. For example, $sup(Cl\text{–}C\ C) < sup(Cl\text{–}C)$ is fulfilled in both problems for Fig. 1. By using this property, frequent semi-connected (induced) subgraphs are derived stepwisely in ascending order of their sizes beginning with 1-frequent subgraphs. Figure 2 is the outline of our proposed method. First, a 1×1 adjacency matrix representing a vertex is generated for every vertex, and they are substituted for C_1. Next, the support of the each candidate frequent semi-connected (induced) subgraph is calculated by accessing the database. Subsequently, Generate-Candidate function generates the candidate frequent semi-connected (induced) subgraphs of size $\{k+1\}$ from k-frequent semi-connected (induced) subgraphs in F_k, and they are substituted for C_{k+1}. Above processes are repeated until C_k becomes empty. Finally, all frequent connected (induced) subgraphs are chosen from the frequent semi-connected (induced) subgraphs and returned.

Table 1 shows frequent semi-connected (induced) subgraphs and candidate frequent semi-connected (induced) subgraphs derived on each stage, when graph dataset shown in Fig. 1 is used, and the minimum support is set to 50%. i-candidate and i-frequent represent the candidate frequent semi-connected (induced) subgraphs and the frequent semi-connected (induced) subgraphs of size i respectively. "–", "=" and "∼" correspond to a single bond, a double bond and an aromatic bond respectively. In case of connected induced subgraph derivation problem, the graphs which have "†" are not generated. Only the frequent connected (induced) subgraphs are chosen in the output.

3.2 Details of Algorithm

3.2.1 Generation of Candidate Subgraphs

$\{k+1\}$-candidate frequent semi-connected (induced) subgraphs are generated from k-frequent semi-connected (induced) subgraphs through *join* operation in Generate-Candidate function. Given two adjacency matrices X_k and Y_k representing frequent semi-connected (induced) subgraphs, they are joinable if

```
// GD is a database consist of graph structured data.
// F_k is a set of adjacency matrix of k-frequent graphs
// C_k is a set of adjacency matrix of k-candidate graphs
// minsup is minimum support.
 0)  Main(GD, minsup){
 1)    C_1 ←{all adjacency matrices consisting
 2)          of one element};
 3)    k ← 1;
 4)    while(C_k ≠ ∅) {
 5)      Count(GD, C_k);
 6)      F_k ← {c_k ∈ C_k | sup(G(c_k)) ≥ minsup};
 7)      C_{k+1} ← Generate-Candidate(F_k);
 8)      k ← k + 1;
 9)    }
10)    return ⋃_k {f_k ∈ F_k | f_k is canonical,
11)                          G(f_k) is connected};
12)  }
```

Figure 2. Outline of Algorithm

Table 1. Extracted Frequent and Candidate Subgraphs

1-candidate	*H, C, Cl*
1-frequent	*H, C, Cl*
2-candidate	*Cl–Cl, Cl=Cl, Cl ∼ Cl, Cl Cl, Cl–C, Cl=C, Cl ∼ C, Cl C, Cl–H, Cl=H, Cl ∼ C, Cl H, C–C, C=C, C ∼ C, C C, C–H, C=H, C ∼ H, C H, H–H, H=H, H ∼ H, H H*
2-frequent	*Cl Cl, Cl–C, Cl C, Cl H, C–H, C H, H H, C C†*
3-candidate	*Cl–C–Cl, Cl–C Cl, Cl–C–H, Cl–C H, H–C–H, C–H H, C–Cl–C†, Cl–C C†, C–H–C†, C–H C†*
3-frequent	*Cl–C–Cl, Cl–C Cl, Cl–C–H, Cl–C H, C–Cl C†, C–H C†, C–H H†*
4-candidate	*Cl–C–Cl$_2$, Cl–C–Cl Cl, Cl$_2$–C–H, Cl–C–Cl H, Cl–C–H Cl, Cl–C–Cl C†, Cl–C–H C†, Cl–C–H H†*
4-frequent	*Cl–C–Cl H†, Cl–C–H Cl†*
output	*H, C, Cl, Cl–C, C–H, Cl–C–Cl, Cl–C–H*

and only if all of the following conditions are fulfilled.

Condition 1 X_k and Y_k are identical except the k-th row and the k-th column, *i.e.*,

$$X_k = \begin{pmatrix} X_{k-1} & \boldsymbol{x}_1 \\ \boldsymbol{x}_2^T & 0 \end{pmatrix}, Y_k = \begin{pmatrix} X_{k-1} & \boldsymbol{y}_1 \\ \boldsymbol{y}_2^T & 0 \end{pmatrix},$$

and $lb(v_i \in V(G(X_k))) = lb(v_i \in V(G(Y_k)))\ (i = 1, \cdots, k-1)$.

Condition 2 X_k is the canonical form of $G(X_k)$.
Condition 3 $G(X_k)$ is a connected graph.
Condition 4 In case that labels of k-th vertices of $G(X_k)$ and $G(Y_k)$ are identical,

$$code(X_k) \geq code(Y_k). \tag{2}$$

In case that they are not identical,

$$num(lb(v_k \in V(G(X_k)))) > num(lb(v_k \in V(G(Y_k)))), \tag{3}$$

or $G(Y_k)$ is not a connected graph.

If X_k and Y_k are joinable, their *join* operation is defined as follows.

$$Z_{k+1} = \begin{pmatrix} X_{k-1} & \boldsymbol{x}_1 & \boldsymbol{y}_1 \\ \boldsymbol{x}_2^T & 0 & z_{k,k+1} \\ \boldsymbol{y}_2^T & z_{k+1,k} & 0 \end{pmatrix},$$

$$lb(v_i \in V(G(Z_{k+1}))) = lb(v_i \in V(G(X_k))) \ (i = 1, \cdots, k-1),$$

$$lb(v_k \in V(G(Z_{k+1}))) = lb(v_k \in V(G(X_k))),$$

$$\text{and } lb(v_{k+1} \in V(G(Z_{k+1}))) = lb(v_k \in V(G(Y_k))).$$

X_k and Y_k are called the first generator matrix and the second generator matrix of Z_{k+1} respectively. Two elements $z_{k,k+1}$ and $z_{k+1,k}$ of Z_{k+1} are not determined by X_k and Y_k. In the mining of undirected graph data, the possible graph structures for $G(Z_{k+1})$ are those wherein there is a labeled edge or wherein there is no edge between k-th vertex and $\{k+1\}$-th vertex. Then, $(|L_E|+1)$ adjacency matrices under $z_{k,k+1} = z_{k+1,k}$ are generated, where $|L_E|$ is the number of edge labels. In the mining of directed graph data, $(|L_E|+1)^2$ adjacency matrices are generated since $z_{k,k+1} = z_{k+1,k}$ does not apply. The adjacency matrix generated under the above conditions is called a *normal form*.

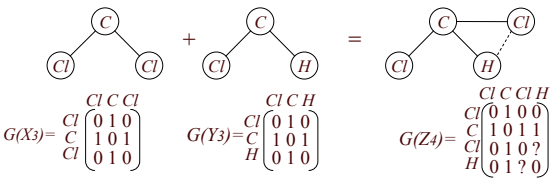


Figure 3. Example of Join Operation

Figure 3 shows an example of join operation. The (3,4)-element and (4,3)-element of Z_{k+1} corresponding to the dashed line in Fig. 3 can be either 0, 1, 2 or 3 representing no bond, a single bond, a double bond and an aromatic bond respectively.

The condition 1 of AcGM is common with that of AGM. It ensures that $G(X_k)$ and $G(Y_k)$ shares the common structure except each vertex corresponding to the last row and the last column of each adjacency matrix. The join operation depicted in Fig. 3 is principally enabled by this condition. The condition 2 is newly introduced to AcGM. As the canonical form uniquely corresponds to a graph structure by the definition of the extended $CODE$ including the vertex label information, this condition efficiently avoids the redundant join operation with the graph represented by the canonical form. The condition 3 plays an important role in conjunction with the use of the frequent connected (induced) subgraphs to limit the generated graph to the semi-connected graph in AcGM. When $G(X_k)$ consists of a connected part as Cl–C and a vertex as C and $G(Y_k)$ also consists as Cl–C H, their join operation derives only unconnected graphs which are not semi-connected as Cl–C C H and Cl–C C–H. However, if $G(X_k)$ is connected as Cl–C–C, then the generated graph can be connected depending on the assignment of the edge label number to $z_{k,k+1}$ and $z_{k+1,k}$ of Z_{k+1} as Cl–C–C–H. This consideration also explains the necessity to search frequent semi-connected (induced) subgraphs in AcGM. If only the frequent connected (induced) subgraphs are searched, then this will miss the possibility to generate a connected graph as Cl–C–C–H which is not derived by the join operation of two frequent connected (induced) subgraphs. The condition 4 is also newly introduced to AcGM, and efficiently avoids the join operation of the first generator matrix Y_k and the second X_k which is redundant with the join of X_k as the first and Y_k as the second. The $CODE(X_k)$ should be larger than the $CODE(Y_k)$ since $G(X_k)$ having larger $CODE$ has higher possibility to have more edges which correspond to non-zero elements in the matrix and then to be a connected graph. This increases the possibility to satisfy the condition 3. Because of the inequality in Eq. (2) and Eq. (3), the canonical form must have the maximum $CODE$ as defined in Definition 7.

```
0)  Normalize(X_k) {
1)    i ← 1;
2)    while(i ≠ k + 1){
3)      if(X_i is normal form and G(X_i) is connected){
4)        X_k ← P'_k^T X_k P'_k;
5)        i ← i + 1;
6)      }else{
7)        X_k ← Q_k^T X_k Q_k;
8)        i ← i − 1;
9)      }
10)   }
11)   return X_k;
12) }
```

Figure 4. Algorithm to Normalize

For the necessary condition of $G(Z_{k+1})$ being a frequent subgraph, all induced subgraphs of $G(Z_{k+1})$ must be frequent subgraphs. The application of this condition reduces the candidates. If all induced subgraphs $G(Z_k)$s being one size smaller than $G(Z_{k+1})$ are frequent, then all induced subgraphs of $G(Z_{k+1})$ are frequent since any induced subgraph of each $G(Z_k)$ is also frequent according to Eq. (1). Such a $G(Z_k)$ is obtained by removing the elements in the m-th row and the m-th column $(1 \leq m \leq k+1)$ of Z_{k+1}.

Then, if Z_k represents the semi-connected graph, it is transformed into the normal form by applying the algorithm as shown in Fig. 4 since its frequency is easily checked by using the normal form matrices obtained

in the former steps. Let the upper left $i \times i$ submatrix of the adjacency matrix X_k be X_i, the matrix to transform X_i into the canonical form be P_i and the unit matrix of size k be I_k. Transformation matrices P'_k and Q_k in Fig. 4 are as follows. The line 4 in Fig. 4 is the operation to transform X_i into its canonical form, and the line 7 is one to change $\{i-1\}$-th vertex into k-th vertex and j-th vertex into $\{j-1\}$-th vertex $(j = i, \cdots, k)$.

$$P'_k = \begin{pmatrix} P_i & 0 \\ 0 & I_{k-i} \end{pmatrix}, Q_k = \begin{pmatrix} I_{i-2} & 0 & 0 \\ 0 & 0 & 1 \\ 0 & I_{k-i+1} & 0 \end{pmatrix}.$$

Figure 5 is an example of the normalization. When $i = 1$, X_i is a normal form, and $G(X_i)$ is connected. Thus, the line 4 of Fig. 4 is performed. $P'_3 = I_3$ because X_i is the canonical form. Next, i is increased by 1. When $i = 2$, X_i is not a normal form, because Eq. (2) is not fulfilled. So the line 7 and the line 8 of Fig. 4 are performed. Thus $i = 1$, and X_i is a normal form again, and $G(X_i)$ is connected. So the line 4 and the line 5 of Fig. 4 are performed. Then $i = 2$, X_i is a normal form, and $G(X_i)$ is connected. So the line 4 of Fig. 4 is performed where $P'_3 = I_3$. Next, i is increased by 1, and i becomes 3. $G(X_i)$ is connected, and X_k is normal form since its second generator matrix is not a connected graph. Therefore normalized X_k is returned.

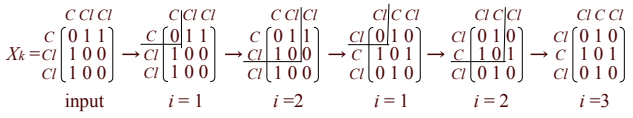


Figure 5. Example of Normalization

3.2.2 Finding Canonical Form

After generating the matrices of candidate subgraphs, a database is accessed to calculate their supports. However, since multiple normal form matrices can represent an identical graph, the canonical form of these matrices must be identified to collect all counts of the graph.

When the canonical form of X_k and its associated transformation matrix are looked for, it is assumed that all transformation matrices into the canonical forms of the frequent subgraphs of size $k-1$ are known. Let the transformation matrix of X^m_{k-1} to its normal from be T^m_{k-1} where X^m_{k-1} is the adjacency matrix obtained by removing the elements of the m-th row and m-th column of X_k. Also let the matrix to transform the normalized matrix $(T^m_{k-1})^T X^m_k T^m_{k-1}$ into its canonical form be S^m_{k-1}. Transformation matrix S^m_k and T^m_k of X_k are derived by the following equations from S^m_{k-1} and T^m_{k-1}.

$$S^m_k = \begin{pmatrix} S^m_{k-1} & 0 \\ 0 & 1 \end{pmatrix},$$

$$T^m_k = \begin{pmatrix} I_{m-1} & 0 & 0 \\ 0 & 0 & 1 \\ 0 & I_{k-m} & 0 \end{pmatrix} \begin{pmatrix} T^m_{k-1} & 0 \\ 0 & 1 \end{pmatrix}.$$

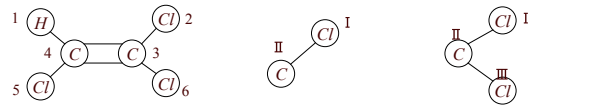
Candidate of canonical form for X_k is given by

$$X_{ck} = arg \max_{m=1,\cdots,k} CODE((T^m_k S^m_k)^T X_k (T^m_k S^m_k)). \quad (4)$$

The matrix to transform X_k into its candidate canonical form is represented as $T^m_k S^m_k$ which makes Eq. (4) the maximum. If the transformation matrix S'_k of a $(T^m_k S^m_k)^T X_k (T^m_k S^m_k)$ into the canonical form has already been found, the canonical form of X_k is provided as $S'^T_k (T^m_k S^m_k)^T X_k (T^m_k S^m_k) S'_k$, and thus calculations for all m in Eq. (4) are not required. It should be noted, however, that the canonical form may not be found by the above method in some cases where the canonical form and its transformation matrix must be searched for in accordance with the permutations. The principles of this canonical form finding are described in detail in the literature [Inokuchi 00].

3.2.3 Frequency Counting

After all canonical forms of candidate subgraphs are obtained, the database is accessed, and the frequency of each candidate subgraph is calculated. The simplest subisomorphism matching between a graph of size k and another graph of size K can be achieved by mapping of the vertices from the subgraph to the graph directly. This brute force search can be conducted by the K^k search tree. The worst time complexity of the brute force method turns into exponential time.



(a) Graph in structure G_K in a database (b) Candidate Gphra$G(X_{k-1})$ (c) Candidate graph$G(X_k)$

Figure 6. Graph Data and Candidate Subgraphs

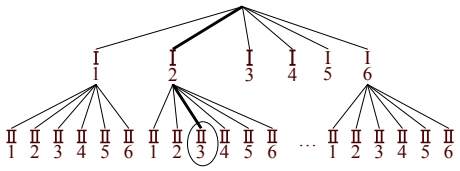


Figure 7. Search Tree for G_K and $G(X_{k-1})$

We propose the following method. Let the canonical form of a k-candidate subgraph be X_k, its first generator matrix be X_{k-1}, and a graph in a database whose

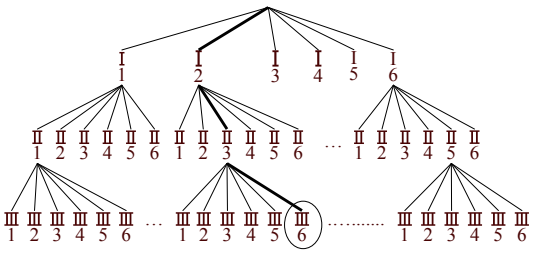


Figure 8. Search Tree for G_K and $G(X_k)$

size is K be G_K. For example, let G_K, $G(X_{k-1})$ and $G(X_k)$ be the graphs of Fig. 6(a), (b) and (c) respectively. The canonical forms of Fig. 6(b) and (c) are expressed as

$$X_{k-1} = \begin{array}{c} \\ Cl \\ C \end{array} \begin{array}{c} \begin{array}{cc} Cl & C \end{array} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{array}, X_k = \begin{array}{c} \\ Cl \\ C \\ Cl \end{array} \begin{array}{c} \begin{array}{ccc} Cl & C & Cl \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array},$$

where $k = 3$. The numbers assigned to the vertices in Fig. 6 are vertex IDs. If the brute force method checks whether G_K includes the graph $G(X_{k-1})$ by the depth first search in ascending order of vertex IDs when $G(X_{k-1})$ is the candidate subgraph, it turns out that the graph G_K includes the graph $G(X_{k-1})$, and the correspondences of the vertices between G_K and $G(X_{k-1})$ are 2=I and 3=II, where 2=I shows that vertex whose ID is 2 is mapped to I. The search tree of this case is shown in Fig. 7. On the other hand, when $G(X_k)$ is the candidate subgraph, it turns out that graph $G(X_k)$ is included, and the correspondences of the vertices are 2=I, 3=II and 6=III as shown in Fig. 8. In this case, the search in the part on the left side of the path root-I2-II3 in Fig. 8 is not necessary since this part has already been checked in Fig. 7. Therefore, if the correspondence relation of the vertex of G_K and $G(X_{k-1})$ is recorded, the G_K's inclusion of the graph structure which has X_{k-1} as the first generator matrix can be efficiently checked. This idea to reuse the matching result in an earlier level enables to calculate frequency much more efficiently than the simple brute force method and some other modified methods of subisomorphism matching [Ullman 76]. The operation of AcGM is different between the connected subgraph derivation problem and the connected induced subgraph derivation problem at this frequency counting only. In the former problem, the graphs including the candidate as a subgraph are counted, whereas the graphs including the candidate as an induced subgraph is counted in the latter.

4. Experiments

The method proposed in this paper was implemented by C++, and IBM PC 300PL with Windows 2000 was used for the evaluation experiments where PentiumIII-667MHz and 192MB of main memory are installed.

4.1 Performance Evaluation

Tables 2 summarizes the parameters to generate the test data and their default used in the simulation experiments. The size of each graph is determined by the Gaussian distribution having the average of $|T|$ and the standard deviation of 1. The edges are attached randomly with the probability of p. The vertex labels and edge labels are randomly determined with equal probability. Similarly, L basic patterns of connected induced subgraphs having the average size of $|I|$ are generated. One of them is chosen by equal probability, *i.e.*, $1/L$, and overlaid on each graph. In the overlay, the vertex to be mapped to each vertex in the basic pattern is randomly selected in the graph. Once the vertices in the graph for the mapping are determined, all edges among them are removed from the graph, and all edges in the basic pattern are attached. The test data are generated for both cases of directed and undirected graphs.

Table 2. Definitions of parameters of test data.

Parameter	Definition	Default
D	Number of graph data in DB	10,000
$\|T\|$	Average size of graph data	15
L	Number of basic patterns	10
$\|I\|$	Average basic patterns size	7
N_v	Number of vertex labels	5
N_e	Number of edge labels	5
p	edge existence probability	20%
$minsup$	Minimum support	10%

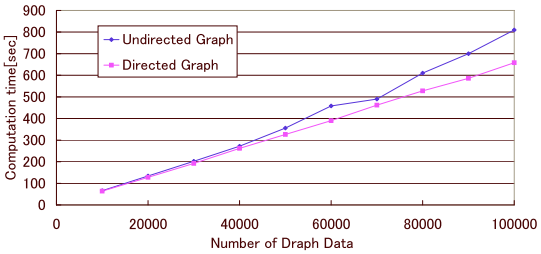


Figure 9. Num. of graph data v.s. Comp. time

Figures 9, 10, 11 12 and 13 show the computation time variations when $|T|$, N_v and p are changed respectively in the connected induced subgraph problem. The default parameter values were used in each figure except the parameter that was changed. Though the result for the connected subgraph problem is omitted, it shows

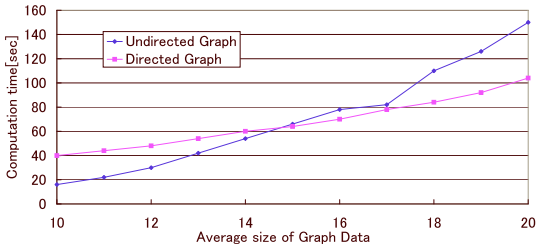


Figure 10. Average Size of Graph Data v.s. Comp. Time

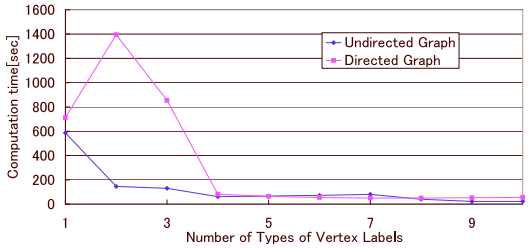


Figure 11. Num. of Vertex Labels v.s. Comp. Time

similar tendency. The computation time is observed to be proportional to D in Fig. 9. Computation time increase gradually with $|T|$ in Fig. 10. Theoretically, AcGM has the NP-time complexity in terms of $|T|$. Thus the proposed algorithm is so efficient that the exponential increase of computation time is sufficiently slow in reality. Figure 11 shows that computation time rapidly decreases as N_v increases. This is natural since larger number of vertex labels increases the possible frequent patterns, and the frequency of each pattern is reduced by the variety of the patterns. We observe the decrease of the computation time at $N_v = 1$ in the directed graph case because the unique label of vertices reduces the number of possible candidates by the factor of $(|L_E| + 1)^2 = (N_e + 1)^2 = 36$ comparing with the case of $N_v = 2$. This effect wins the effect to increase the frequency of each pattern. On the other hand, the decrease is not seen in the undirected graph case since its factor is only $|L_E|+1 = N_e+1 = 6$. Similar tendency is observed for N_e. Figure 13 indicates that the computation time decreases as p is reduced. This is trivial since less connected graphs are under less p.

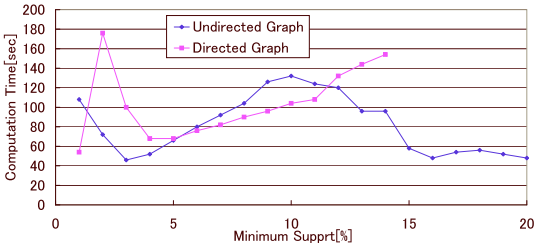


Figure 12. Num. of types of edge labels v.s. Comp. time

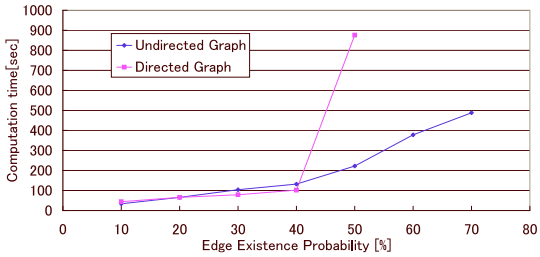


Figure 13. Edge Existence Probability v.s. Comp. Time

4.2 Application

The molecular structure data of carcinogenic compounds were analyzed. This data was provided by Predictive Toxicology Evaluation [PTE], and contains information on 340 chemical compounds. The number of types of the atoms which constitute chemical compounds is 24. In addition, the atoms take some different states, and thus the total number of atom types is 66. The atomic bonds which correspond to edges in a graph have 4 types. The average size of the graph data is around 27, and the maximum size is 214. The edge existence probability is around 4.7%.

Figure 14 shows the result of computation time for various minimum support values. It includes the results of AcGM for both a connected subgraph derivation problem and a connected induced subgraph problem, FSG and AGM. AGM is not directly comparable with the others since its search is not limited to the connected graphs. AcGM for the both problems far outperforms the other approaches. Furthermore, Fig. 15 shows the computation time per a discovered frequent subgraphs. In both figures, AcGM far outperforms the other approaches.

Table 3 shows the detailed computation time and the number of discovered frequent subgraphs for the both problem. In general, the computation time for Generation-Candidate function increased more sensitively than that of Count function when the minimum support is decreased. The number of discovered frequent subgraphs is also increased exponentially.

Figure 16 shows two examples of derived frequent subgraphs in the connected induced subgraph derivation problem. 6 chemical compounds with carcinogenic activity and 19 compounds without the activity contain the induced subgraph depicted in Fig. 16(a). Similarly, 17 compounds with carcinogenic activity and 4 compounds without the activity contain the induced subgraph in Fig. 16(b). The former molecular substructure does not induce significant activity whereas the latter induces quite high activity.

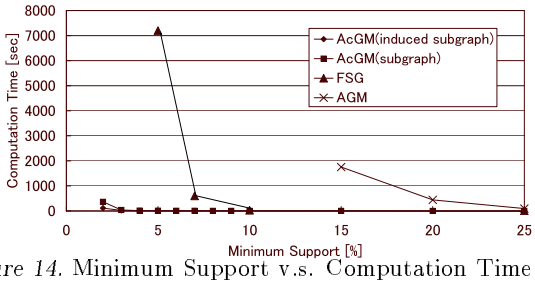


Figure 14. Minimum Support v.s. Computation Time

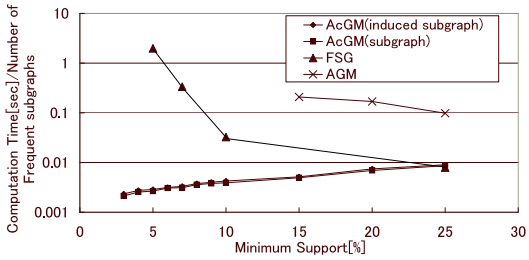


Figure 15. minimum support v.s. computation time

5. Discussion and Related Works

Some heuristic based approaches, *e.g.* SUBDUE [Cook 94] and GBI [Yoshida 95, Motoda 97], or some limitation on the class of subgraphs to be searched [De Raedt 01] have been introduced to alleviate the complexity issue. SUBDUE derives characteristic patterns based on Minimum Description Length of subgraphs. The recent version of GBI derives characteristic patterns in a dataset by chunking pair of connected vertices having a high score [Matsuda 00]. The advantage of these methods is the ability to search typical patterns under various criteria in rapid manner. However, their greedy search may miss some important patterns. De Raedt et al. proposed an approach to derive frequent paths in graphs [De Raedt 01]. Although its running speed is comparative to AcGM, it can not derive complex substructures having branches and loops.

In contract, WARMR [Dehaspe 98], AGM [Inokuchi 00] and FSG [Kuramochi 01] use the complete search. WARMR derives frequent patterns in a graph dataset by using level-wise search. The patterns are represented by the first order predicates. AGM derives frequent induced subgraphs contained in a graph dataset, where every graph structure is represented by an adjacency matrix. The basic algorithm structure of FSG is similar to AcGM. However, the knowledge representation of graphs and its associated operations are not tuned to graphs. As shown in Fig. 14, AcGM for the connected (induced) subgraph problem can derive the complete result within a few minutes where this performance is far faster than the other approaches. This is because the representation of graphs and the associated operations are well-organized and dedicated to the mining of graphs. This performance level is considered to be highly practical in many applications.

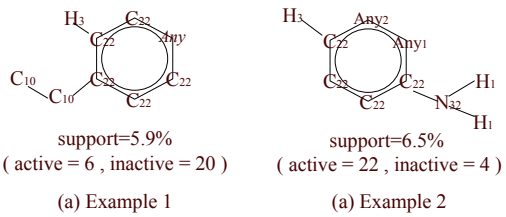


Figure 16. Extracted Frequent Connected Graphs

Table 3. Comp. time and Num. of frequent subgraphs

induced subgraph			
minsup [%]	computation time[sec]		Num. of frequent subgraphs
	Generate-Candidate	Count	
25	0.01	0.91	90
20	0.03	1.13	148
15	0.05	1.49	287
10	0.12	2.24	553
9	0.11	2.55	669
8	0.15	2.78	793
7	0.22	3.33	1,106
6	0.31	3.92	1,375
5	0.52	5.36	2,178
4	0.93	8.86	3,388
3	3.11	16.42	8,566
2	39.99	71.57	50,416

induced subgraph			
minsup [%]	computation time[sec]		Num. of frequent subgraphs
	Generate-Candidate	Count	
25	0.02	1.15	126
20	0.05	1.49	199
15	0.08	2.09	437
10	0.16	3.22	860
9	0.21	3.54	993
8	0.28	4.05	1,240
7	0.45	4.94	1,786
6	0.51	5.71	2,138
5	0.97	8.33	3,627
4	1.93	12.67	5,955
3	8.09	28.82	18,146
2	204.35	149.86	136,981

6. Conclusion

We proposed a new algorithm called AcGM which derives all frequent connected graphs from database. The algorithm can handle labeled graphs and perform the search highly efficiently. We evaluated its performance in terms of the required computation time for various data. Computational efficiency which is superior to the other approaches has been confirmed. In

addition, frequent subgraph patterns to predict carcinogenicity were successfully discovered from PTE database and its high practicality has been confirmed.

References

[Agrawal 94] Agrawal, R., & Srikant, R. (1994). Fast Algorithm for Mining Association Rules in Large Databases. *Proc. of the 20th Very Large Data Bases Conference*, (pp. 487–499).

[Cook 94] Cook, D. J., & Holder, L. B. (1994). Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*, Vol.1, (pp. 231–255).

[Dehaspe 98] Dehaspe, L., Toivonen, H., & King, R. D. (1998). Finding frequent substructures in chemical compounds. *Proc. of the 4th International Conference on Knowledge Discovery and Data Mining*, (pp. 30–36).

[De Raedt 01] De Raedt, L., & Kramer, S. (2001). The Levelwise Version Space Algorithm and its Application to Molecular Fragment Finding. *Proc. of the 17th International Joint Conference on Artificial Intelligence*, (pp. 853–859).

[Inokuchi 00] Inokuchi, I., Washio, T., & Motoda, H. (2000). An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data. *Proc. of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, (pp. 13–23).

[Kuramochi 01] Kuramochi, M., & Karypis, G. (2001) Frequent Subgraph Discovery. *Proc. of the 1st IEEE International Conference on Data Mining.*

[Matsuda 00] Matsuda, T., Horiuchi, T., Motoda, H., & Washio, T. (2000). Extension of Graph-Based Induction for General Graph Structured Data. *Proc. of the 4th Pacific- Asia Conference on Knowledge Discovery and Data Mining*, (pp. 420–431).

[Motoda 97] Motoda, H., & Yoshida, K. (1997). Machine Learning Techniques to Make Computers Easier to Use. *Proc. of the 15th International Joint Conference on Artificial Intelligence*, Vol. 2, (pp. 1622–1631).

[PTE] PTE
http://oldwww.comlab.ox.ac.uk/oucl/groups/machlearn/PTE

[Ullman 76] Ullman, J. R. (1976). An algorithm for subgraph isomorphism, *Journal of the ACM*, Vol. 23, no. 1, pp. 31–2.

[Yoshida 95] Yoshida, K., & Motoda, H. (1995). CLIP: Concept Learning from Inference Patterns. *Artificial Intelligence*, Vol. 75, No. 1 pp. 63–92.