Machine Learning, 50, 321–354, 2003

Complete Mining of Frequent Patterns from Graphs: Mining Graph Data

AKIHIRO INOKUCHI*
TAKASHI WASHIO washio@sanken.osaka-u.ac.jp
HIROSHI MOTODA motoda@sanken.osaka-u.ac.jp
Institute for Scientific and Industrial Research, Osaka University, Osaka 567-0047, Japan

Editors: Ryszard S. Michalski and Pavel Brazdil

Abstract. Basket Analysis, which is a standard method for data mining, derives frequent itemsets from database. However, its mining ability is limited to transaction data consisting of items. In reality, there are many applications where data are described in a more structural way, e.g. chemical compounds and Web browsing history. There are a few approaches that can discover characteristic patterns from graph-structured data in the field of machine learning. However, almost all of them are not suitable for such applications that require a complete search for all frequent subgraph patterns in the data. In this paper, we propose a novel principle and its algorithm that derive the characteristic patterns which frequently appear in graph-structured data. Our algorithm can derive all frequent induced subgraphs from both directed and undirected graph structured data having loops (including self-loops) with labeled or unlabeled nodes and links. Its performance is evaluated through the applications to Web browsing pattern analysis and chemical carcinogenesis analysis.

Keywords: data mining, graph data, Apriori algorithm, adjacency matrix, Web browsing analysis, chemical carcinogenesis analysis

1. Introduction

Data mining aims to discover interesting and/or useful patterns that are hidden in a given data set and to use them as explicit knowledge. This approach heavily relies on computational power. Generally speaking, knowledge that can explain many cases is considered to be useful, and frequency and confidence of the discovered patterns are used very often as measures of goodness of knowledge. Along this line, Basket Analysis using Apriori algorithm (Agrawal & Srikant, 1994) that seeks for the co-occurrence of items in transactions under these two measures has been researched actively.

Apriori algorithm can extract co-occurrence of items in an efficient manner, but the data structure, which can be handled, is limited to a set of items. The extensions to treat structured data in form of sequences and/or taxonomy have been reported in recent studies. The approach proposed by Agrawal and Srikant for mining sequential patterns was an initiating work in this field (Agrawal & Srikant, 1995). Since then several approaches have been proposed from different angles for sequential or structural data. Mannila et al. proposed an approach to mine frequent episodes from sequences (Mannila, Toivonen, &

*Present address: Tokyo Research Laboratory, Japan IBM, Japan.

Verkamo, 1997). Srikant et al. used taxonomy hierarchy as background knowledge to mine association rules (Srikant, Vu, & Agrawal, 1997). Our previous work also suggested that Apriori algorithm can be applied to graph structured data after a simple preprocessing of the data (Inokuchi, Washio, & Motoda, 1999). However, this technique works only for a limited case where all nodes in a graph are distinct, i.e., all nodes have different labels, and thus it is not applicable to mine frequent patterns in graphs where multiple nodes of a same label are contained, e.g., chemical molecule structures.

The isomorphism problem of a "*general subgraph*" has been proved to be in NP-complete (Garey & Johnson, 1979), where a general subgraph of G consists of a subset of the nodes of G and a subset of the links connecting node pairs in the node subset. Accordingly, some heuristic based incomplete search or some limitation on the class of subgraph must be introduced to alleviate the complexity issue. GBI (Graph Based Induction) quite efficiently extracts typical subgraph patterns that appear frequently in a set of graph data. It applies recursive pair wise chunking operations (Yoshida & Motoda, 1995). SUBDUE is another approach to seek the characteristic subgraph patterns by efficiently compressing the original graphs under MDL principle (Cook & Holder, 1994). These approaches do not face the severe computational complexity. However, they may miss some significant subgraph patterns, since the search strategies are greedy or heuristic. Some other methods have been proposed to perform non-greedy searches which completely find some limited classes of subgraph patterns. de Raedt and Kramer proposed the level-wise version space algorithm to mine sequence patterns of the connections of nodes and links characterized by the combination of monotonic and anti-monotonic measures such as frequency and generality (de Raedt & Kramer, 2001). Though this method can apply a variety of measures and avoid the computational complexity issue, the class of the pattern to be mined is limited to sequence patterns embedded in the graphs. Geibel and Wysotzki proposed a method to derive induced subgraphs of graphs given in data and to use the induced subgraphs as attributes to classify the graphs in the data under decision tree approaches (Geibel & Wysotzki, 1996). The "*induced subgraph*" of a graph G has a subset of the nodes of G and the same links between pairs of nodes as in G. Their method can be used to find frequent induced subgraphs in the set of graph data. However, the upper limit number of the nodes to be included in the subgraph must be initially specified to avoid the exponential explosion of the computational time, and thus the search is not complete. Liquiere and Sallantin proposed a method to completely search homomorphically equivalent subgraphs which are the least general over a given set of graphs and do not include any identical triplet of the labels of two nodes and the link direction between the nodes within each subgraph (Liquiere & Sallantin, 1998). They show that the computational complexity to find this class of subgraphs is polynomial for 1/2 locally injective graphs where the labels of any two nodes pointing to another common node or pointed from another common node are not identical. However, many graphs appearing in the real-world problems such as chemical compound analysis are more general, and hence the polynomial characteristics of this approach do not apply. In addition, this approach may miss many interesting and/or useful subgraph patterns since the homomorphically equivalent subgraph is a small subclass of the general subgraph.

To our knowledge, a first system to perform complete search for the wider class of frequent substructure in graphs is WARMR proposed by Dehaspe, Toivonen, and King (1998).

They applied ILP (Inductive Logic Programming) in concert with Apriori-like level wise search to a problem of carcinogenesis prediction of chemical compounds. The structures of chemical compounds are represented by the first order predicates such as $atomel(C, A1, c)$, $bond(C, A1, A2, BT)$, $aromatic - ring(C, S1)$ and $alcohol(C, S2)$. The first two state that $A1$ which is a carbon atom bonds to $A2$ where the bond type is BT in a chemical compound C. The third represents that substructure $S1$ is an aromatic ring in a chemical compound C, and the last represents that $S2$ is an alcohol base in C. Because this approach allows variables to be introduced in the arguments of the predicates, the class of the structure which can be searched is more general than the graph. However, this approach easily faces the high computational complexity due to the equivalence checking under θ-subsumption on atoms and the generality of the problem class to be solved. To alleviate this difficulty, a new system called FARMAR has been recently proposed by Nijssen and Kok (2001). It also uses the level wise search, but applied less strict equivalence relation under substitution to reduced atom sets. FARMAR can also find the predicates including variables and runs faster in the second order of magnitudes of time. However, its result includes some predicates having different forms but equivalent in the sense of the Θ-subsumption due to the weaker equivalence criterion. Though these two systems can discover the frequent structures in high level of descriptions, the class of the structures to be searched is limited to *connected* structures, i.e., all variables and constants are related with predicates in a clause. For instance, a frequent clause $atomel(C, A1, c) \wedge bond(C, A1, A2, BT) \wedge atomel(C, A2, c) \wedge atomel(C, A3, cl)$ which represents two carbons connected by a bond BT and an isolated chlorine in a chemical substance C is not searched in the systems. Because many context dependent data in the real-world problems can be represented by a set of grounded first order predicates which is represented by graphs, the approach to mine each frequent pattern consisting of some subgraphs in a set of graph data will address many practical problems.

Among the aforementioned classes of the subgraph, the general subgraph is the most general. However, the general subgraph does not represent the topological substructure of the original graph sufficiently, because many general subgraphs lack the information of links among the nodes. For example, a set of 6 carbon atoms without any links is a general subgraph of an aromatic ring, but it does not represent any topology of the ring. In addition, the complete search of the general subgraphs has very high computational complexity. When the original graph consists of many nodes and links, the number of the combinations of the nodes and the links to form its general subgraphs become huge. In contrast, an induced subgraph represents the generic substructure of a general graph while reflecting the topology of the links among the nodes in the graph. The search of the induced subgraphs has relatively low computational complexity due to the constraints on the link topology, though the complexity of the isomorphism problem of the induced subgraphs including the ones consisting of multiple subgraph fragments is not clear yet. Some mathematicians proved that the *connected induced subgraph* isomorphism is a type of PB (Polynomially lower Bounded)-complete problem (Kann, 1995). Hence the induced subgraph isomorphism problem is likely to lie between NP-complete and PB-complete problems, and is clear not to be computed within low order polynomial time in terms of the problem scale. Accordingly the complete search of the frequent induced subgraphs must

be designed in highly efficient manner, and such search method is expected to provide a powerful measure to mine structural knowledge in many problems.

In this paper, we propose a novel principle and an algorithm that can capture all "*frequent induced subgraphs*" and "*association rules among induced subgraphs*" which are observed in many transactions where each is a general graph. The graph can be either directed or undirected. It can have loops (including self-loops), and the nodes and links can have labels. Further, it is not limited to a connected graph but also a set of isolated graphs. Our method can extract topological information involved in the graph in addition to the features of nodes and links. The paper is organized as follows. The proposed algorithm, which is obtained by the extension of Apriori algorithm, is outlined in Section 2. The details of the extension are described in Section 3. Its performance evaluation on test datasets is reported in Section 4, followed by two application results, one to Web browsing pattern analysis and the other to chemical carcinogenesis analysis, in Section 5. The related work is discussed in Section 6, and the paper ends with the concluding remarks in Section 7.

2. Outline of proposed algorithm

The aforementioned Apriori algorithm can handle only sets of items, i.e., "*itemsets*". For the extension to the graph structured data, a data representation called "*adjacency matrix*" is introduced. The proposed algorithm mines induced subgraphs frequently included in graph structured transactions. Similarly to Basket Analysis, the association rule to relate a body induced subgraph with another head induced subgraph is defined under almost same definitions of the support and the confidence of Apriori algorithm.

Because the adjacency matrix and the matrix coding, which will be mentioned later, are only suitable to represent a graph which includes labeled nodes but not suitable to represent labeled links and self-loop links, a preprocessing is applied to convert the general graph transaction data to the graphs having labeled nodes, unlabeled links and no self-loop links. Moreover, the rows and the columns corresponding to the graph nodes in the adjacency matrix are lexicographically ordered in terms of the node labels to reduce the ambiguity of the graph representation. Since the graphs represented by the adjacency matrices also needs to be uniquely ordered in the later mining process, a code for each adjacency matrix is introduced. The code is defined by the non-diagonal elements of the matrix. This coding also has an advantage that it reduces the memory requirements as a matrix representation needs much memory, while a coding does not.

Once the graphs are represented by the codes of their adjacency matrices, the codes of the frequent induced subgraphs are derived in the bottom up manner similarly to Apriori algorithm. The adjacency matrix of a graph is not unique. There are more than one adjacency matrix that represent the same graph. Accordingly, the matrix to represent a frequent induced subgraph is limited to a specific form called a "*normal form*". Based on the representation of the normal form matrix, the "*join operation*" to generate candidate frequent induced subgraphs having a larger size is conducted. After the code generation of all candidate frequent induced subgraphs, their frequencies are counted by accessing to the graph transaction data. However, since the normal form representation still has ambiguity that multiple normal forms represent a graph, the "*canonical form*" representation of the

adjacency matrix and its code are introduced to collect all counts for each frequent induced subgraph.

As noted in this outline, the extension of Apriori algorithm to the application to the graph structured data is not straightforward in terms of the data representation and the associated operations. Some novel principles have been introduced for the extension.

3. Details of extension to graph structured data

3.1. Representation of graph structured data

While an itemset is a transaction for Apriori algorithm, a graph constitutes a transaction in our extension. The graph-structured data can be transformed without much computational effort into an "*adjacency matrix*", which is a well-known representation of a graph in mathematical graph theory (Biggs, 1974). Each row and column of the matrix correspond to a node that appears in the graph, and if there is a link from the i-th node to the j-th node in the transaction, the value "1" is assigned to the (i, j)-element of the matrix, otherwise the value "0" is assigned. We call a node which corresponds to the i-th row (the i-th column) the i-th node v_i and the number of nodes contained in a graph its "*size*". We note a graph whose size is k as a k-graph, an adjacency matrix of a k-graph as X_k, the (i, j)-element of X_k as x_{ij} and the graph corresponding to X_k as $G(X_k)$. We further represent the node label as N_g (g-th label and $1 \leq g \leq p$) and the link label as L_h (h-th label and $1 \leq h \leq q$). "*support*", "*confidence*" and an "*association rule*" for induced subgraphs are defined as follows.

$$sup(G) = \frac{number\ of\ transactions\ including\ an\ induced\ subgraph\ G}{total\ number\ of\ transactions},$$

$$sup(B \Rightarrow H) = sup(B \cup H), \quad conf(B \Rightarrow H) = \frac{sup(B \cup H)}{sup(B)},$$

where B and H are induced subgraphs of some graph transactions in the given data, and $B \cup H$ is the graph constituted by the unions of the sets of nodes and the sets of links involved in B and H. $B \Rightarrow H$ indicates that H is included in a graph transaction under the confidence $conf(B \Rightarrow H)$ if B is included in it.

Here, the formal definition of the "*induced subgraph*" is given. Let $V(G)$ be the set of all nodes in the graph G, and let $E(G)$ be the set of all links $\{u, v\}$ connecting u and v in the graph G where $u, v \in V(G)$. G' is an induced subgraph of G if and only if

$$\begin{aligned} &V(G') \subseteq V(G), \quad E(G') \subseteq E(G) \quad \text{and} \\ &\forall u, v \in V(G')\ \{u, v\} \in E(G) \Leftrightarrow \{u, v\} \in E(G'). \end{aligned} \tag{1}$$

More intuitively speaking, an induced subgraph of a graph G has a subset of the nodes of G and the same links between pairs of nodes as in G. For example, the subgraph (b) in figure 1 is an induced subgraph of the graph (a), but the subgraph (c) is a general subgraph

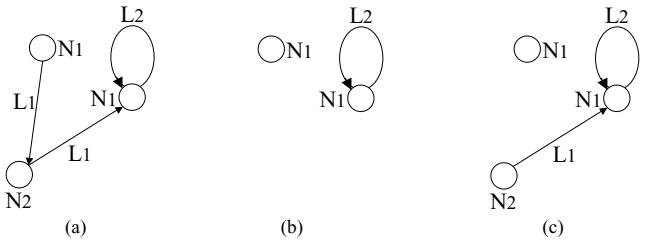
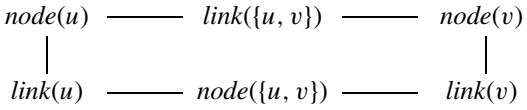


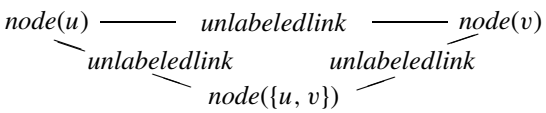
Figure 1. An induced subgraph and a general graph.

but not an induced subgraph since the incoming links of the node labeled as N_2 is not retained while the node labeled as N_2 is included in (c). "*minimum support*" and "*minimum confidence*" are defined in the same way as in Apriori algorithm. They provide threshold levels where the mined induced subgraphs and their association rules must have higher support and confidence values. We call the graph whose frequency exceeds the minimum support as a "*frequent induced subgraph*".

If a graph has labeled links, the following conversion of the graph to a new graph that has labels at its nodes only, is applied. This reduces the ambiguity in the adjacency matrix representation and the candidate generation of the frequent induced subgraph. Given a node pair u, v and a directed or undirected link $\{u, v\}$ between the nodes, i.e., $node(u) - link(\{u, v\}) - node(v)$ where $node()$ and $link()$ stand for the labels of a node and a link respectively, its dual expression, where the link becomes a node, and the nodes become links, is represented as $link(u) - node(\{u, v\}) - link(v)$. Because the dual expressions are mutually equivalent and thus redundant when they are combined together as follows,



the $link()$ label information can be removed from the combined graph, and the following triangular graph is deduced where the original information of the node pair and the link between them is preserved.



This operation on each link in the original graph can convert the graph into another graph representation having no labels on the links while preserving the topological information of the original graph. Accordingly, as shown in figure 2(a) and (b), when there is a link labeled as L_h $(1 \leq h \leq q)$ between two nodes labeled as N_f and N_g $(1 \leq f, g \leq p)$ where N_f and N_g can be a same label, a new node with the label L_h is inserted between the two nodes, and the original link between the nodes labeled as N_f and N_g is retained as an unlabeled link. For example, the link labeled as L_1 directing to a node labeled as N_2 from a node labeled as

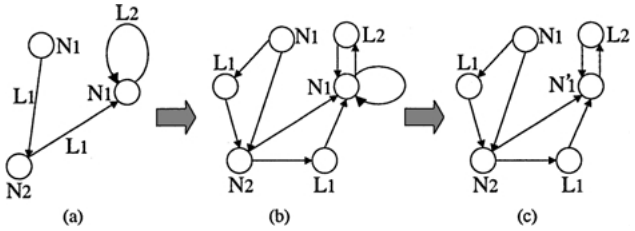


Figure 2. Preprocessing of labeled links and self-looped nodes in a graph.

N_1 becomes a node labeled as L_1 between the nodes N_1 and N_2 while the link which label is removed is retained. Similarly, the link labeled as L_1 directing to a node labeled as N_1 having a self-looped link from a node labeled as N_2 becomes a node labeled as L_1 between the nodes while the original but unlabeled link is retained. Moreover, the self-looped link labeled as L_2 becomes a node labeled as L_2 having the incoming and outgoing unlabeled links with N_1, and the original self-looped but unlabeled link is retained. Accordingly, the adjacency matrix of figure 2(a) is converted to that of (b) as follows.

$$\begin{array}{c} \\ N_1 \\ N_1 \\ N_2 \end{array} \begin{array}{c} \begin{array}{ccc} N_1 & N_1 & N_2 \end{array} \\ \begin{pmatrix} L_2 & 0 & 0 \\ 0 & 0 & L_1 \\ L_1 & 0 & 0 \end{pmatrix} \end{array} \Rightarrow \begin{array}{c} \\ N_1 \\ N_1 \\ N_2 \\ L_1 \\ L_1 \\ L_2 \end{array} \begin{array}{c} \begin{array}{cccccc} N_1 & N_1 & N_2 & L_1 & L_1 & L_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

The need to retain the direct link from the node N_f to the node N_g is also demonstrated in figure 3. According to the definition of an induced subgraph, graph2 is not an induced

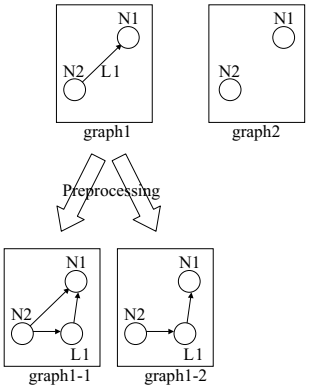


Figure 3. The need to retain the direct link from N_2 to N_1.

subgraph of graph1. When graph1 is transformed into graph1-1, then graph2 is not an included subgraph of graph1-1. However, if preprocessing which deletes the direct link from N_2 to N_1 is applied, then graph1 is transformed into graph1-2 in which graph2 is an induced subgraph. Accordingly, the latter preprocessing causes some spurious induced subgraphs.

When there is a self-looped node labeled as N_1 as shown in figure 2(b), its label is changed to a different and distinct label N_1' as depicted in figure 2(c) for the ease and efficiency of matrix coding as described later. Hence, the adjacency matrix of this graph is represented by

$$\begin{array}{c} \\ N_1' \\ N_1 \\ N_2 \\ L_1 \\ L_1 \\ L_2 \end{array} \begin{array}{c} \begin{array}{cccccc} N_1' & N_1 & N_2 & L_1 & L_1 & L_2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

The fact that N_1' has a self-loop is recorded, and is backed up again in the final result of the mined frequent induced subgraphs. This preprocessing increases the number of the label types of the rows and the columns of the adjacency matrix of this graph, and reduces the ambiguity of the possible lexicographic sort patterns of the rows and the columns needed in the subsequent stage. This also provides another advantage that the nodes having more than one self-loop link can be represented by different and distinct labels according to the number of the self-loop links. Hereafter, we consider only the graphs that have only labeled nodes without any labeled links and self-looped nodes.

The representation of a graph structure by an adjacency matrix can have some variety depending on the way to assign each node to a row (a column) of the matrix. To avoid the ambiguity of the representation and the inefficiency in the graph pattern search, the node labels are ordered according to some rule such as the lexicographical order.

$$N_1 < \cdots < N_{p-r} < \cdots < N_1' < \cdots < N_r' < \cdots < L_1 < \cdots < L_q,$$

where r is the number of the node labels representing self-looped nodes. By applying this ordering, the graph in figure 2(c) is represented by

$$\begin{array}{c} \\ N_1 \\ N_2 \\ N_1' \\ L_1 \\ L_1 \\ L_2 \end{array} \begin{array}{c} \begin{array}{cccccc} N_1 & N_2 & N_1' & L_1 & L_1 & L_2 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}, \tag{2}$$

where the order of nodes labels is $N_1 < N_2 < N_1' < L_1 < L_2$.

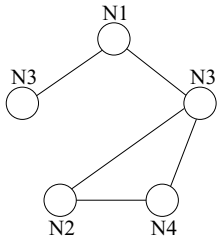


Figure 4. An undirected graph.

The aforementioned explanation is for directed graphs. But the identical preprocessing is applied to undirected graphs. The difference from the case of directed graphs is that the adjacency matrix becomes diagonally symmetric. The adjacency matrix of figure 4 is represented by

$$\begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_3 \\ N_4 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_2 & N_3 & N_3 & N_4 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}. \tag{3}$$

For the sake of the efficiency in memory consumption, we define an efficient code representation of an adjacency matrix as follows. In case of an undirected graph, the code of an adjacency matrix X_k, i.e., $code(X_k)$, is represented by a binary number as shown in Eq. (4). It is obtained by scanning the upper triangular elements of the matrix along each column.

$$X_k = \begin{pmatrix} 0 & x_{1,2} & x_{1,3} & \cdots & x_{1,k} \\ x_{2,1} & 0 & x_{2,3} & \cdots & x_{2,k} \\ x_{3,1} & x_{3,2} & 0 & \cdots & x_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{k,1} & x_{k,2} & x_{k,3} & \cdots & 0 \end{pmatrix},$$

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

For example, the code of the adjacency matrix (3) is given by

$$code(X_k) = 0101100101.$$

In case of a directed graph, $code(X_k)$ is represented by a base 4 number. If both (i, j)-element and (j, i)-element are 0, "0" is assigned to the digit of the code. If (i, j)-element is

1 and (j, i)-element is 0, "1" is assigned. In case that (i, j)-element is 0 and (j, i)-element is 1, "2" is assigned, and finally in case that both (i, j)-element and (j, i)-element are 1, "3" is assigned. Thus $code(X_k)$ for a directed graph is defined by

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

where

$$c_{i,j} = 2x_{j,i} + x_{i,j}.$$

For example, the code of the adjacency matrix (2) is given by

$$code(X_k) = 101012120000300.$$

3.2. *Extraction algorithm of frequent graph*

3.2.1. Candidate generation of frequent graph. The Apriori algorithm generates frequent itemsets very efficiently by the use of the constraint on the orders among items to generate a candidate frequent $k+1$-itemset from the frequent k-itemsets. Similarly, two frequent induced subgraphs are joined only when the following constraints are satisfied to generate a candidate frequent induced subgraph of size $k+1$, i.e., induced $k+1$-subgraph. Let X_k and Y_k be adjacency matrices of two frequent induced k-subgraphs $G(X_k)$ and $G(Y_k)$. If both $G(X_k)$ and $G(Y_k)$ have equal elements of the matrices except for the elements of the k-th row and the k-th column, then they are joined to generate Z_{k+1}.

Constraint 1.

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

$$Z_{k+1} = \begin{pmatrix} X_{k-1} & \boldsymbol{x}_1 & \boldsymbol{y}_1 \\ \boldsymbol{x}_2^T & 0 & z_{k,k+1} \\ \boldsymbol{y}_2^T & z_{k+1,k} & 0 \end{pmatrix} = \left(\begin{array}{cc|c} & & \boldsymbol{y}_1 \\ \multicolumn{2}{c|}{X_k} & z_{k,k+1} \\ \hline \boldsymbol{y}_2^T & z_{k+1,k} & 0 \end{array}\right),$$

where X_{k-1} is the adjacency matrix representing the graph whose size is $k-1$, $\boldsymbol{x}_i$ and $\boldsymbol{y}_i (i = 1, 2)$ are $(k-1) \times 1$ column vectors. Let the label of the i-th node of the adjacency matrix X_k be $N(X_k, i)$, then there must be the following relations among the adjacency matrices X_k, Y_k and Z_{k+1}.

Constraint 2.

$$N(X_k, i) = N(Y_k, i) = N(Z_{k+1}, i) \quad \text{and}$$
$$N(X_k, i) \leq N(X_k, i+1) \quad \text{for } i = 1, \ldots, k-1.$$
$$N(X_k, k) = N(Z_{k+1}, k), \quad N(Y_k, k) = N(Z_{k+1}, k+1), \quad \text{and } N(X_k, k) \leq N(Y_k, k).$$

Here, the $(k, k+1)$ and the $(k+1, k)$ elements of the adjacency matrix Z_{k+1} are not determined by X_k and Y_k. In case of an undirected graph, two possible cases are considered in which 1) there is a link between the k-th node and the $(k+1)$-th node of $G(Z_{k+1})$ and 2) there is no link between them. Corresponding to these two cases, two adjacency matrices whose $(k, k+1)$-element and $(k+1, k)$-element are "1" and "0" are generated respectively. In case of a directed graph, four possible cases are considered where 1) there is a directed link from the k-th node to the $k+1$-th node of $G(Z_{k+1})$, 2) there is a directed link from $k+1$-th node to the k-th node, 3) there are bi-directional links between them and 4) there is no link between them. Accordingly four different adjacency matrices are generated. We call X_k and Y_k the "*first matrix*" and the "*second matrix*" to generate Z_{k+1} respectively. Note that when the labels of the k-th nodes of X_k and Y_k are the same, switching X_k and Y_k, i.e., taking Y_k as the first matrix and X_k as the second matrix, produces another adjacency matrices representing an identical graph. In order to avoid this redundant candidate generation, the two adjacency matrices are joined only when the following constraint is satisfied.

Constraint 3.

code(the first matrix) $\leq$ *code*(the second matrix).

We call the join of two matrices under these three constraints the "*join operation*" and the adjacency matrix generated by this operation a "*normal form*" matrix.

The join operation is demonstrated through the example depicted in figure 5. The examples of the adjacency matrices representing the graphs (a) and (b) are given as follows.

$$X_4 = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_3 \end{array} \begin{array}{c} \begin{array}{cccc} N_1 & N_2 & N_3 & N_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}, \quad Y_4 = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_3 \end{array} \begin{array}{c} \begin{array}{cccc} N_1 & N_2 & N_3 & N_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array}.$$

In this case, since the upper left 3×3 submatrix parts, i.e., X_3 in both X_4 and Y_4 are identical, and the $(4, 4)$-elements of both matrices are 0, the conditions for X_4 and Y_4 in the Constraint 1 are met. For $i = 1, \ldots, 3$, the label of every i-th node is identical between X_4 and Y_4, and their order is lexicographic. Accordingly, the conditions for X_4 and Y_4

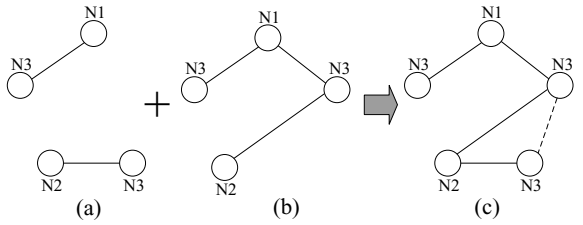


Figure 5. An example of join.

in the Constraint 2 are also met. Moreover, because the labels of the 4th nodes in X_4 and Y_4 are identical, the Constraint 3 must be checked. This constraint is also met since $code(X_4) = 01001 < code(Y_4) = 01011$. In summary, these two matrices are joinable, and the following Z_5 which satisfies all constraints is obtained.

$$Z_5 = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_3 \\ N_3 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_2 & N_3 & N_3 & N_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & ? \\ 1 & 1 & 0 & ? & 0 \end{pmatrix} \end{array}.$$

The (3, 4)-element and the (4, 3)-element which correspond to the dashed line in figure 5(c) can be either 0 or 1.

In Apriori algorithm, the $(k + 1)$-itemset becomes a candidate frequent itemset only when all the k-sub-itemsets of the $(k + 1)$-itemset are confirmed to be the frequent itemsets according to the monotonicity of the support. Similarly, in case of graph structured data, the $k + 1$-graph $G(Z_{k+1})$ is considered to be a candidate of frequent induced subgraphs only when the adjacency matrices of all induced k-subgraphs of $G(Z_{k+1})$ are confirmed to represent frequent induced subgraphs. Because our proposed algorithm generates only adjacency matrices of the normal form in the earlier k-stages, if the adjacency matrix of the induced subgraph generated by removing the i-th node, i.e., i-th row and column, $(1 \leq i \leq k+1)$ is a non-normal form matrix, it is not easy to confirm whether it is a frequent induced subgraph or not. Thus, a method to transform a matrix of non-normal form to that of normal form is needed. An adjacency matrix of non-normal form is transformed into a normal form matrix by reconstructing it. Figure 6 shows an example of the transformation of an adjacency matrix X_4 which is a non-normal form matrix. The numbers below the

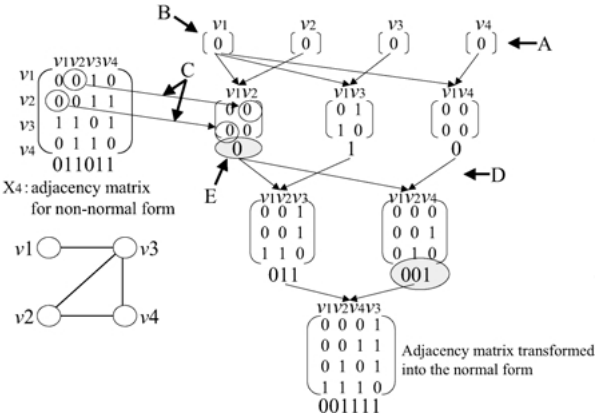


Figure 6. Transformation into the normal form.

adjacency matrices in this figure show the corresponding codes, and v_i denotes the i-th node of the adjacency matrix X_4 to be normalized. It starts with the adjacency matrices representing the induced subgraphs of X_4 consisting of one node (see figure 6(A)). As it is necessary to find a normal form matrix of $G(X_4)$ among many normal forms, the combination for joining should be restricted. We choose a matrix consisting of v_1 as the first matrix and the others the second matrix (see figure 6(B)). The values which can not be determined by join operation, for example (1, 2)-element and (2, 1)-element of a matrix consisting of v_1 and v_2, are taken from x_{12} and x_{21} of X_4 to reflect the graph structure of X_4 (see figure 6(C)). Next, matrices whose graphs have two nodes are joined (see figure 6(D)). Here, the matrix whose code is the least becomes the first matrix, and the others become the second matrices (see figure 6(E)). If there are adjacency matrices whose codes are tie, we simply choose one randomly. This process continues until an adjacency matrix having the same size with the original adjacency matrix is found. Because the rows and the columns of the original adjacency matrix are re-ordered by following the aforementioned constraints to generate a normal form matrix while retaining the graph structure, the resultant adjacency matrix becomes a normal form matrix.

3.2.2. Canonical form. After all candidate graphs are derived, their support values are counted by scanning the database. However, as mentioned earlier, multiple normal form matrices can represent an identical graph. For example, the following two matrices, which are all normal forms, represent an identical graph where Y_5 is the "*canonical form*" of the graph representation as explained later.

$$X_5 = \begin{array}{c} \\ N_1 \\ N_1 \\ N_1 \\ N_1 \\ N_1 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_1 & N_1 & N_1 & N_1 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}, \quad Y_5 = \begin{array}{c} \\ N_1 \\ N_1 \\ N_1 \\ N_1 \\ N_1 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_1 & N_1 & N_1 & N_1 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}. \tag{5}$$

Increasing the counter of X_5 for a transaction and increasing the counter of Y_5 for another transaction by treating them as different candidate graphs lead to a wrong result of the support value of the graph. Therefore, before counting the support, the identification of the normal form adjacency matrices representing an identical graph must be conducted. For doing so, we introduce a unique normal form matrix called the "*canonical form*" among all normal form matrices representing an identical graph G, and set pointers from the normal form matrices to the canonical form matrix. Let k be the size of G and $\Gamma(G) = \{X_k \mid X_k \text{ is a normal form matrix and } G \equiv G(X_k)\}$. Then the canonical form C_k of G is defined as follows.

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

The canonical form matrix C_k of $G(\equiv G(X_k))$ is obtained through the permutations of the rows and columns of the normal form matrix X_k. The permutations are made through the operation $C_k = W_k^T X_k W_k$ where each element of W_k, i.e., w_{ij}, is 1, when the i-th node of X_k should be permuted with the j-th node, otherwise 0.

We consider to derive W_k to transform a normal form matrix X_k to the canonical form matrix C_k under the assumption that the permutation matrix to transform a normal form matrix to its canonical form matrix is already known for every frequent induced $k-1$-subgraph. As shown by Theorem 1 in Appendix, the first matrix of the canonical form matrix C_k of a normal form matrix X_k, i.e., the submatrix obtained by removing the last row and the last column of C_k, is also the canonical form matrix C_{k-1} of an induced $k-1$-subgraph of $G(C_k) \equiv G(X_k)$. Because $G(C_{k-1})$ is an induced subgraph obtained from $G(C_k) \equiv G(X_k)$ by removing a node in $G(X_k)$, the code of C_{k-1} is equal to the code of the canonical form matrix $can(X_{k-1}^m)$ of the matrix X_{k-1}^m which is produced by removing one of k nodes, i.e., the m-th node for a number m $(1 \leq m \leq k)$, from $G(X_k)$. Moreover, according to Theorem 2 in Appendix, the canonical form $can(X_{k-1}^m)$ having the minimum code over $1 \leq m \leq k$ is C_{k-1}. Combination of this fact and Theorem 1 deduces Corollary 1 which ensures the existence of a permutation matrix to transform X_k to its canonical form C_k in which the part C_{k-1} is $can(X_{k-1}^m)$ for a m. Here, the following new definition of a matrix C_k^p named "*pseudo-canonical form*" is introduced.

$$C_k^p = \begin{pmatrix} C_{k-1} & \boldsymbol{c^p}_1 \\ \boldsymbol{c^p}_2^T & 0 \end{pmatrix},$$

where $G(C_k^p) \equiv G(C_k) \equiv G(X_k)$ and C_{k-1} is identical with the first matrix of C_k whereas the last row and the last column do not have to be identical with those of C_k. C_k^p has the same code digits with C_k except its lower $k-1$ digits. The pseudo-canonical form matrix C_k^p is derived quite easily.

First, the matrix X_{k-1}^m $(1 \leq m \leq k)$ is transformed into the normal form by the procedure explained in figure 6. The permutation matrix for this transformation is expressed as T_{k-1}^m, and the matrix transformed into a normal form is given by $(T_{k-1}^m)^T X_{k-1}^m T_{k-1}^m$. The matrix to transform the normal form $(T_{k-1}^m)^T X_{k-1}^m T_{k-1}^m$ into its canonical form $can(X_{k-1}^m)$ is expressed as S_{k-1}^m, and $can(X_{k-1}^m)$ is given by $(T_{k-1}^m S_{k-1}^m)^T X_{k-1}^m T_{k-1}^m S_{k-1}^m$. The matrices S_k^m and T_k^m to transform X_k are defined from S_{k-1}^m and T_{k-1}^m through Eqs. (6) and (7).

$$s_{ij} = \begin{cases} s_{ij}^m & 0 \leq i \leq k-1 \text{ and } 0 \leq j \leq k-1, \\ 1 & i = k \text{ and } j = k, \\ 0 & \text{otherwise}, \end{cases} \tag{6}$$

$$t_{ij} = \begin{cases} t_{ij}^m & i < m \text{ and } j \neq k, \\ t_{i-1,j}^m & i > m \text{ and } j \neq k, \\ 1 & i = m \text{ and } j = k, \\ 0 & \text{otherwise}, \end{cases} \tag{7}$$

where s_{ij}, s^m_{ij}, t_{ij} and t^m_{ij} are the (i, j)-elements of matrix S^m_k, S^m_{k-1}, T^m_k and T^m_{k-1} respectively. According to Theorem 3 in Appendix, a pseudo-canonical form C^p_k for X_k is given by

$$C^p_k \quad w.r.t. \quad code\big(C^p_k\big) = \min_m code\Big(\big(T^m_k S^m_k\big)^T X_k\big(T^m_k S^m_k\big)\Big) \tag{8}$$

for all $m = 1, \ldots, k$ satisfying $N(X_k, m) = N(X_k, k)$. As the last row and the last column of C^p_k can be different from those of C_k and $code(C_k) \leq code(C^p_k)$, some rows and columns of C^p_k must be further permuted to obtain C_k from C^p_k while maintaining the first matrix C_{k-1}. This can be done by introducing a transformation matrix U_k which permutes the nodes of C^p_k having an identical node labels, the identical rows and the identical columns in C_{k-1}. Given the set of all U_k for C^p_k as $\Lambda(C^p_k)$, the canonical form C_k of X_k is given by the following search.

$$C_k \quad w.r.t \quad code(C_k) = \min_{U_k \in \Lambda(C^p_k)} code\big(U^T_k C^p_k U_k\big) \tag{9}$$

The matrix $T^m_k S^m_k U_k$ minimizing the code is stored as S_k for the next $(K + 1)$-th step calculation to derive the canonical form of X_{k+1}. The computational complexity of the search of C_k is $O(k!)$ in the worst case. However, because of the variety of node labels and the heterogeneity of the graph topology in the real data, the required search space is far less in many real applications. In addition, during the computation of Eq. (8), if a matrix $(T^m_k S^m_k)^T X_k (T^m_k S^m_k)$ is derived where one of the matrices S_k to transform it into a canonical form have already been found, the canonical form of X_k is given by the following transformation, and the further computation of Eq. (8) for each m is stopped.

$$C_k = S^T_k \big(T^m_k S^m_k\big)^T X_k \big(T^m_k S^m_k\big) S_k. \tag{10}$$

In the example of Eq. (5), X_5 is not the canonical form since $code(Y_5) = 0000111011 < code(X_5) = 0001011101$. The X^m_4s for $m = 1, 2, 3, 4$ and 5 are as follows.

$$X^1_4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad X^2_4 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad X^3_4 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix},$$

$$X^4_4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \quad X^5_4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

Since X^1_4 is already a normal form matrix, the nodes is not permuted in the procedure of figure 6. Hence T^1_4 is the 4×4 unit matrix, and $(T^1_4)^T X^1_4 T^1_4$ is equal to X^1_4. In addition, S^1_4 is the 4×4 unit matrix, because X^1_4 is already the canonical form matrix where $code(X^1_4) =$

001101. Accordingly, the following T_5^1 and S_5^1 are obtained through Eqs. (6) and (7).

$$T_5^1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad S_5^1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Then the following $(T_5^1 S_5^1)^T X_5 (T_5^1 S_5^1)$ is derived where the code is 0011010011.

$$\left(T_5^1 S_5^1\right)^T X_5 \left(T_5^1 S_5^1\right) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}. \tag{11}$$

For $m = 2, 3, 4$ and 5, similar transformations are applied, and their resultant codes of $(T_5^m S_5^m)^T X_5 (T_5^m S_5^m)$ are 0011110010, 0011110010, 0000111011 and 0000111101 respectively. Among these codes, the one for $m = 4$ is the minimum, and thus the pseudo-canonical form C_5^p of X_5 is known to be equal to Y_5 in Eq. (5). Furthermore by applying Eq. (9), the canonical form matrix C_5 is searched. In this example, the code of Eq. (5) is already the minimum, and hence Y_5 is known to be the canonical form C_5.

The algorithm shown in figure 7 performs the aforementioned operations. In the loop from (3) to (14), the operations defined by Eqs. (8) and (10) are performed. S_k^m and T_k^m required in (5) and (6) are obtained by Eqs. (6) and (7). At the step (8), the operation of Eq. (10) is conducted, if the appropriate transformation matrices have been obtained before. At the step (16), the permutation search of Eq. (9) is conducted to derive the canonical form matrix C_k.

3.2.3. Frequency calculation. Frequency of each candidate frequent induced subgraph is counted by scanning the database after generating all the candidates of frequent induced subgraphs and obtaining their canonical forms. Here, the counting method is explained through an example to count frequent induced subgraphs of size 3 in a directed and unlabeled graph shown in figure 8. We assume that the candidates of frequent induced subgraphs of size 3 have already been obtained. The codes of the adjacency matrices are base 4 numbers, and v_i denotes the i-th node of the transaction. Further we assume that all the graphs of size 2 except for the one whose code is 3 are known to be frequent induced subgraphs. Starting with the adjacency matrices of size 1, the larger submatrices of the given matrix ($code = 301011$) are sequentially generated. Similarly to the procedure of figure 6, $(k, k-1)$ and $(k-1, k)$-elements of the joined matrix are taken from the given transaction matrix. Since the adjacency matrix consisting of v_1 and v_2 has code 3, and is not the frequent induced subgraph, this is not joined with any other matrices. In addition, the right most matrix consisting of v_3 and v_4 is not joined with the others since the node

```
1) forall X_k in a set of candidate frequent induced subgraphs
2)   X'_k = X_k,   m = 1
3)  while  m ≤ k do begin
4)   if(N(X_k, m) = N(X_k, k)) then do begin
5)    if(code(X'_k) > code((T_k^m S_k^m)^T X_k (T_k^m S_k^m)) then do begin
6)     X'_k = (T_k^m S_k^m)^T X_k (T_k^m S_k^m);
7)     if(the transformation matrix of X'_k to the canonical form
       is known) then do begin
8)      X'_k = S_k^T X'_k S_k;
        //where S_k is the matrix to transform X'_k
          in r.h.s. to its canonical form.//
9)      break;
10)    end
11)   end
12)  end
13)  m=m+1
14) end
15) if(the canonical form of X_k has not been derived in the step 8)
16)  X'_k=permutation(X'_k);
17) end
18) Canonical form of X_k is X'_k;
19) end
```

Figure 7. An algorithm to transform X_k to the canonical form.

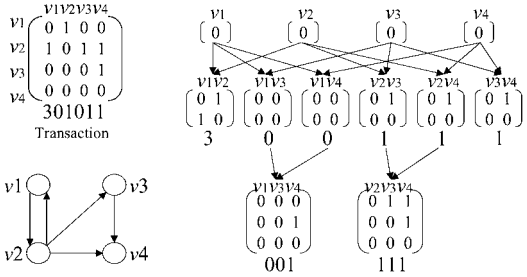


Figure 8. An example of frequency calculation.

label in its first matrix part, i.e., v_3, is not shared with any other matrices of size 2. The other matrices of size 2 are mutually joined into size 3 if the node label is shared in their first matrix part. Subsequently the joined matrices are checked if they correspond to any canonical forms of candidate frequent induced subgraphs. In this example, both normal form matrices of the codes 001 and 111 generated in the join operation are sought in the set of normal forms of candidate frequent graphs derived in the previous sections. Then, the counter of each canonical form is incremented by 1. Even when there are more than one isomorphic induced subgraphs in a transaction, the default is that we increase the counter by 1. This is based on the definition of support that is the fraction of the graph transactions containing a certain subgraph pattern in all transactions. As a second option we

can increase the count by the number of occurrences of the isomorphic subgraph within a transaction.

3.3. Implementation

The algorithm explained in the previous subsections is implemented using a "*trie*" data structure. The trie is one of basic and popular data structures used in the search domain, and is a specific case of the tree data structure. The trie is used to represent and retrieve sequences of symbols. Starting from the top root where the null symbol is assigned, a branching point representing a symbol sequence is added downward in a recursive manner. If an identical symbol subsequence appears at the beginning of multiple symbol sequences, the sequences share the path from the top root to a branching point corresponding to the identical symbol subsequence, and the point has branches to connect to another point for each symbol sequence. Thus the trie represents many symbol sequences hierarchically in a compact fashion. Figure 9 is an example of undirected graphs that have two node labels N_1 and N_2. A branching point of the trie corresponds to the normal form of a matrix. The symbol sequence consisting of literal (above) and the numeral (below) in each point show respectively the node label order and the code of the adjacency matrix. The depth of the trie corresponds to the size of the graphs, and the parent matrix of each matrix in the trie is the first matrix to generate the matrix.

Assuming all frequent induced subgraphs of size i and their matrix codes have been already found, and they are located in the $i + 1$-th level of the trie, we consider to search the frequent induced subgraphs of size $i + 1$ and their matrix codes. Because the matrices that satisfy the join constraints with a given matrix is located only on the right hand side of the given matrix, the matrix can be efficiently joined within this trie data structure. After the join operation, each node of the constructed graph is removed in sequence, and the resulting graphs having one smaller size i are checked if all of them have been found in the $i + 1$-th level of the trie as frequent induced subgraphs. If they are, the graph becomes a candidate frequent induced subgraph, and the canonical form is obtained for the candidate. After generating all candidates, the database is accessed, and all the induced subgraphs corresponding to the candidates are derived from each transaction, and the frequency for each candidate is counted. If it exceeds the minimum support threshold, the graph is found

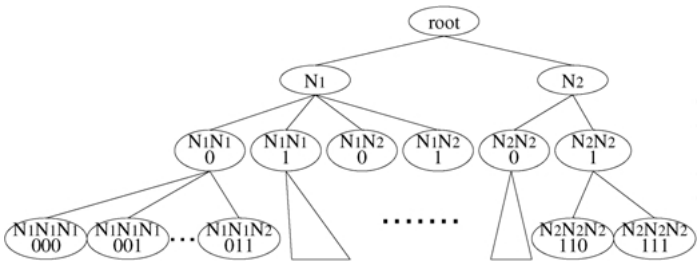


Figure 9. Trie data structure.

to be a frequent induced subgraph, and a new point of its normal form matrix is added at the $i+2$-th level under the point of its first matrix. The above process is repeated from the root to downward in a stepwise manner. Note that the candidate generation and the frequency calculation interleave, and thus the further expansion from the matrix whose frequency is below the minimum support threshold is terminated.

4. Performance evaluation

The basic performance of the proposing method was examined using the graph-structured transactions that were artificially generated in a random manner. An representative random graph generation method named "$G_{|T|,p}$ *model*" is applied in this evaluation where graphs having an expected number of $|T|$ nodes and an expected number of $p|T|(|T|-1)/2$ links are generated where each of the $|T|(|T|-1)/2$ possible links is set with fixed probability p. The set of graphs generated by this model is known to have a similar topology over the entire set (Walsh, 2001), and thus contains many and various frequent induced subgraphs. On the other hand, the graphs commonly seen in real world problems are known to have non-uniform topology such as the link network of Web pages which includes some nodes having very many links and the other having a few links. Their several models, such as small world model, ultrametric model and power law model, have been proposed (Watts & Strogatz, 1998; Hogg, 1996; Barabasi & Albert, 1999). Recently, Walsh showed that the graphs generated by the random models, e.g., $G_{|T|,p}$ model, induce far higher computational complexity than the above models of real world data to search some specific topological patterns such as graph coloring (Walsh, 2001). Accordingly the evaluation of our method provides almost the lower bound of the performance.

Table 1 summarizes the parameters to generate the test data and their default values that are used in the experiments. The size of each transaction, i.e., the number of nodes in a graph, is determined by the gaussian distribution having the average of $|T|$ and the standard deviation of 1. The node labels are randomly determined with equal probability. The links are attached randomly with the probability of p according to the $G_{|T|,p}$ model. Similarly, L basic patterns of 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 transaction. In the

Table 1. Definitions of parameters of test data.

Parameter	Definition	Default value
D	Number of transactions	10,000
$\|T\|$	Average transaction size	10
L	Number of basic patterns	10
$\|I\|$	Average basic patterns size	4
N	Number of node labels	5
p	Link existence probability	50%
minsup	Minimum support	10%

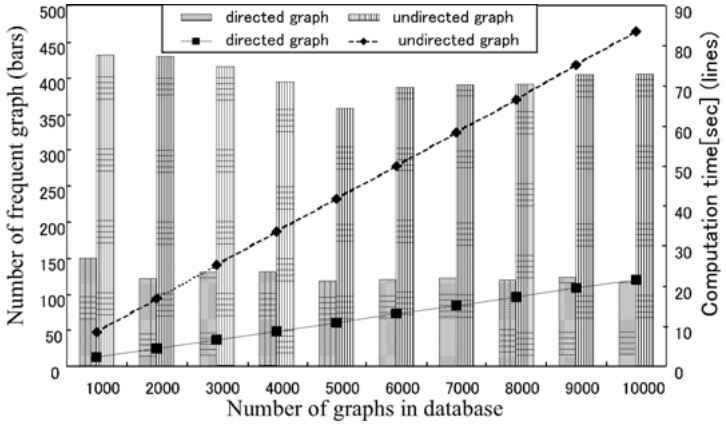


Figure 10. Computation time vs. number of transactions.

overlay, the node to be mapped to each node in the basic pattern for every node label is randomly selected in the transaction. Once the nodes in the transaction for the mapping are determined, all links among them are removed from the transaction, and all links in the basic pattern are attached. By this preprocessing, the L basic patterns are embedded as induced subgraphs in the transaction data. The test data are generated for both cases of directed and undirected graphs. The machine used for the evaluation is a PC having a Pentium CPU of 400 MHz and main memory of 128 MB.

Figures 10–13 are the results of how computation time varies when the number of transactions, average transaction size, number of node labels and minimum support threshold are changed respectively. The default parameter values were used in each figure except the value of the parameter that was changed. Figure 10 shows that the number of different frequent induced subgraphs is nearly constant, and the computation time is proportional to the number of transactions. Computation time and the number of different frequent induced subgraphs increase exponentially with the transaction size in figure 11. Figure 12 shows that computation time rapidly decreases as the number of node labels increases. Figure 13 indicates that the computation time and the number of different frequent induced subgraph increase as the minimum support is reduced. The observed tendencies in figures 10, 11 and 13 are almost identical with the properties of Apriori algorithm (Agrawal & Srikant, 1994). Figure 12 also shows the tendency similar to Apriori algorithm. Because the increase of the number of node labels increases the variety of graph transactions, the frequency to observe isomorphic subgraphs decreases in the transaction data, and this effect reduces the search space of the frequent induced subgraphs. The complexity of the problem which Apriori algorithm solves is known to be in NP-complete (Agrawal & Srikant, 1994). On the other hand, the problem to search induced subgraph isomorphism is also known to be between NP-complete and PB-complete problems as mentioned in the first section. Hence this experimental consequences are very natural. Further, we note that analysis of directed graphs generally requires less computation time than the analysis of undirected graphs. This is because the number of different graphs that are represented by the same number of nodes

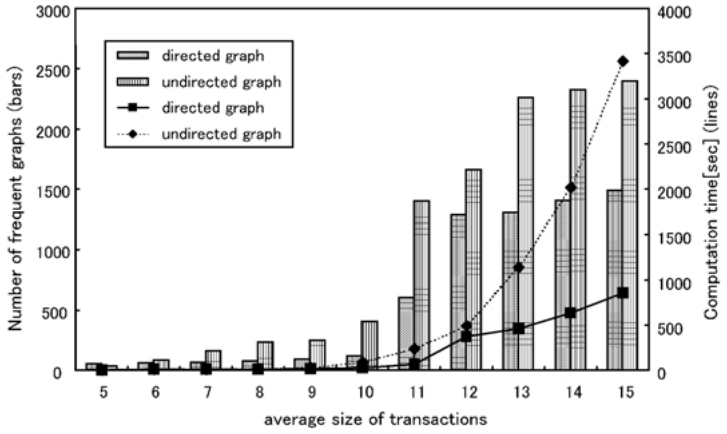


Figure 11. Computation time vs. transaction size.

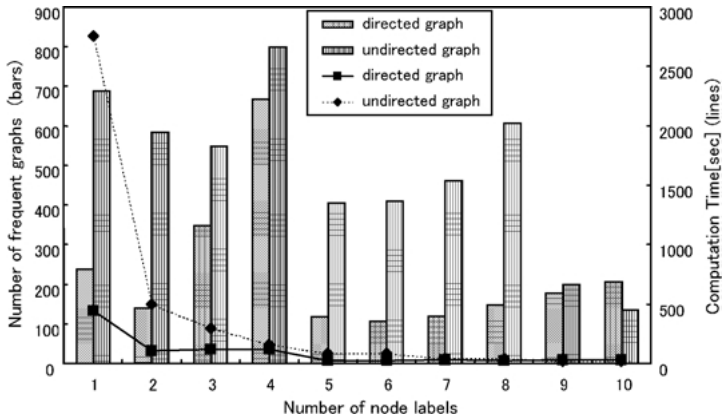


Figure 12. Computation time vs. number of node labels.

is much larger for the directed graphs, and thus their frequencies become smaller. This fact effectively prunes the candidate graphs at early stages.

Figure 14 is the results when the link existence probability p between any two nodes is changed in the generation of the directed and undirected graph transaction data. All nodes in the graphs have an identical node label in these transaction data, and thus they are equivalent to unlabeled graphs. Each computation time for a p was evaluated for a data set generated under the p. The decrease of the computation time is observed as the probability increases. This can be explained by the following reason. Most of the non-normal form matrices, if generated, would be on the right hand side of the trie in figure 9, and they are not generated in our algorithm. Accordingly the number of matrices in the right hand side of the trie is relatively small. On the other hand, the graphs in this region have relatively many links. If the average number of links of all transactions is larger, the number of matrix generation in the

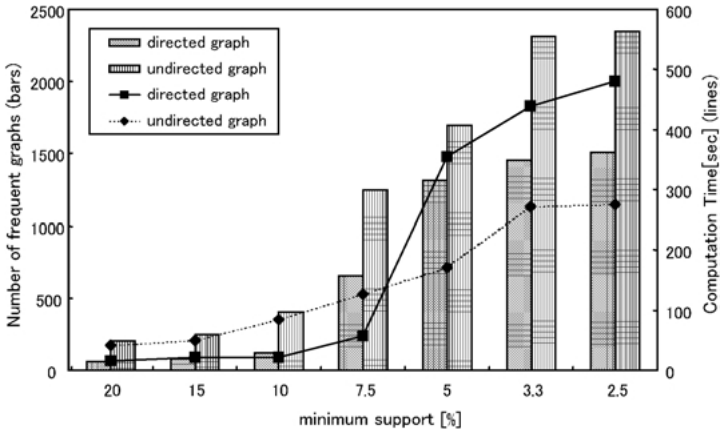


Figure 13. Computation time vs. minimum support threshold.

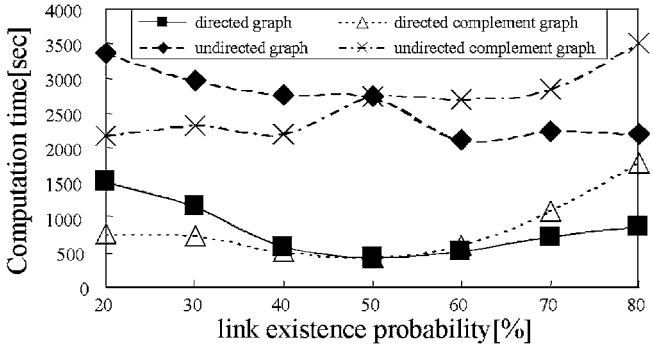


Figure 14. Computation time vs. link existence probability.

trie will be less. Figure 15 shows examples of unlabeled and undirected graphs. The graph of the right hand side is the one complementary on the left hand side to the complete graph. A complete graph is a graph where there exists a link for every node pair. Let the graph G' be a subgraph of a complete graph G. G''s complementary graph to G is the graph that results by removing all the links of G' from G. The numbers below each graph in figure 15 are the codes of all normal form matrices corresponding to the graph. The graph of the right hand side has more links, and has less normal forms. Thus the number of normal forms can be reduced by transforming the graph of the left to the one of the right. Consequently, if the average number of links of all graph transactions is less than half of that of their complete graphs, it is worth to convert all transactions to the graph complementary to their complete graphs as shown in figure 16, where the upper represents the case of an undirected graph and the lower the case of a directed graph. Because the proposed method searches the induced subgraph defined by Eq. (1), the subgraphs satisfying the contrapositive of Eq. (1) are also searched. Therefore, the identical results on the frequent induced subgraphs

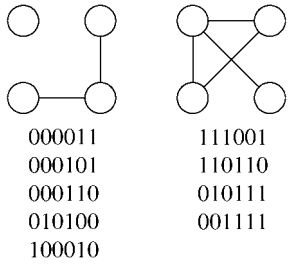


Figure 15. Comparison of codes for two graph structures.

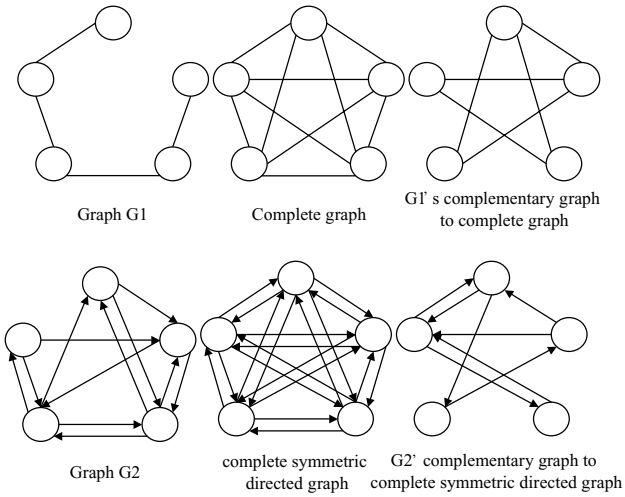


Figure 16. Graphs and their complementary graphs.

are obtained under the application of this preprocessing, if the re-conversion of the found frequent induced subgraphs to their complements are applied after the mining. In figure 14, the computation time for the directed and undirected complement graphs is also indicated. As supposed by the above consideration, the reduction of the computation time for $p < 50\%$ is observed.

5. Applications

In this section, two applications of our proposing method are described. One is Web browsing pattern analysis, and the other is chemical carcinogenesis analysis.

5.1. Web browsing analysis

Figure 17 is an imaginary example of a Website where there are 5 URLs that are hyper linked. The user of this Web site visits the URLs by following the hyperlinks. A sequence

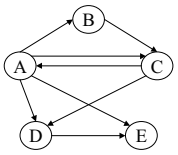


Figure 17. Graphs of URL.

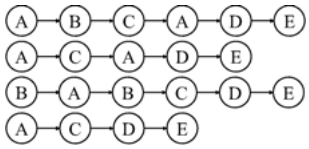


Figure 18. Sequence data.

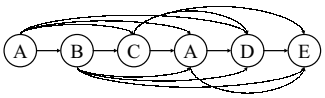


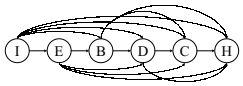
Figure 19. Conversion of sequence data to graph structure ($n = 4$).

in figure 18 represents an access history of a user recorded in the WWW server log file. It is the order of access whereas the arrows in figure 17 shows the existence of hyperlinks. Many users visit A, then go to C via some URLs, and finally visit to E via some URLs. Since there is no direct hyperlink from C to E, the user must have visited other URLs before reaching E. If such a sequence starting from A to E via C is mined as a frequent sequence, the potential need of clients to directly link from C to E is suggested. Accordingly, we have added extra links labeled as "*virtual*" up to the n-th next URL from each URL in each history in order to explicitly indicate the order of the discontinuous access to URLs as shown in figure 19.

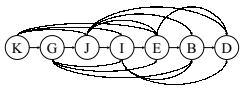
The access history of one-day log of a commercial WWW site called "Achara NAVI" of Recruit Co. Ltd. is analyzed in this study. The log is a text file that contains each user's IP address, access time stamp and a visited URL in each line. The number of URL registered in this site is about 25,000. The size of this file is about 400 MB and includes about 8,700 URLs and the associated links. If there is no access for more than 5 minutes from the same IP address, the access history is separated from one another assuming that they are independent browsing activities. An access history forms a transaction. The value of n was set to 5 in this analysis. The conversion of the access log into a set of transactions, which includes the operations to remove IP addresses and time stamps, was applied. Moreover, the conversion of the links to the nodes is conducted as described in Section 3.1. The size of the resultant data file became nearly 800 MB. The number of transactions was 50,666, the average size of one transaction 3.2, the maximum size of the transactions 118 and the number of node labels, i.e., URLs, 8,655. For the efficient analysis, each URL label was replaced by its short index symbol. This data compression reduced the size of the data file

Table 2. Mapping from alphabet to URL.

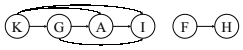
A	/NAVI/mscategory/Sub/s01.html
B	/NAVI/mscategory/Sub/s03.html
C	/NAVI/mscategory/Sub/s04.html
D	/NAVI/mscategory/Sub/s05.html
E	/NAVI/mscategory/Sub/s06.html
F	/NAVI/mscategory/Sub/s08.html
G	/NAVI/mscategory/Sub/s09.html
H	/NAVI/mscategory/Sub/s12.html
I	/NAVI/mscategory/Sub/s13.html
J	/NAVI/mscategory/Sub/s14.html
K	/NAVI/mscategory/Sub/s16.html



(1) Extracted pattern (Frequency 76).



(2) Extracted pattern (Frequency 80).



(3) Extracted pattern (Frequency 77).

Figure 20. Extracted patterns from Web browsing data.

significantly to 79 MB. For the minimum support of 0.15%, the number of frequent patterns extracted was 1,898, and the maximum size was 7. The computation time was 52.3 sec.

Three examples of the extracted patterns are shown in figure 20. The mapping between each alphabet symbol to each URL is given in Table 2. Figure 20(1) is a case where the user followed the 6 URLs in order. Since there is a link for every ordered pair of URLs, these URLs are known to be indeed visited in sequence. Figure 20(2) is a case where the user visited 7 URLs. In this case since there is no virtual link from K to B, the user has visited some other URLs on the way from K to B in some of the 80 cases. Figure 20(3) shows two patterns that are isolated from each other. It shows that many clients visit K, G, A and I in sequence while visiting F and H independently within the same browsing session. This case suggested to the Webmaster that some new links are needed between these two groups of URLs.

5.2. Chemical carcinogenesis analysis

The biological experiments on the toxicity of chemical compounds are quite expensive and very time consuming, and thus it is prohibitive to rely solely on the experiments from

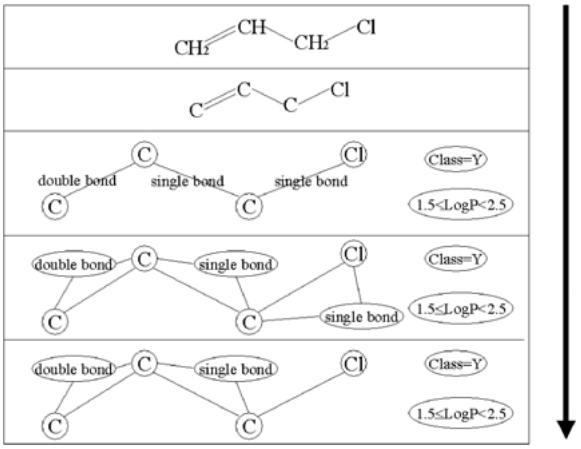


Figure 21. Preprocessing applied to an organic chloride.

both economical and time efficiency views. The prediction of some properties based on the molecular structures is highly required to search useful chemical compounds. Upon this background, the possibility to predict chemical carcinogenesis has been explored using our method. The task is to find structures typical to carcinogen of organic chlorides comprising C, H and Cl. The data were taken from the benchmark data site of Oxford University (Srinivasan et al., 1997). The data consists of 41 organic chlorides out of which 31 are carcinogenic (positive examples) and 10 non-carcinogenic (negative examples). There are three kinds of links: single bond, double bond and aromatic bond.

Figure 21 shows the preprocessing to convert an organic chloride to the corresponding undirected labeled graph. The hydrogen H was removed from the graph. This simplification is commonly applied in the chemistry, and is known not to affect the mining result significantly because the removed atoms are always known to be hydrogen, and the original structure can be easily reconstructed. The three labeled links are preprocessed by adding three new nodes as described in the Section 3.1. *Log P* value that was given for each substance was discretized into 4 intervals. This value is related with the chemical activity of a compound, and its boundaries are set at 1.5, 2.5 and 3.5 based on the chemist's expertise. The *Log P* value and the *class* value (positive or negative) were added as two isolated nodes to the graph representing the molecular structure. Noting that there exists only single bonding between C and Cl, the node representing the link label between these two nodes was removed as depicted in the bottom line in figure 21. With this preprocessing, the average node number of the graph was reduced to 16.8, and the computation time to extract all the frequent induced subgraphs was 1.92 sec when the minimum support was set to 50%.

For the minimum support of 30%, 200 different frequent induced subgraphs were discovered in the computation time 216.1 sec. Two examples of the extracted rules are shown in figure 22. The support of the association rule of example 1 is 31.7%, and its confidence is 86.7%. This rule implies that if there is a structure CCl_2 and a Cl that is not directly connected to CCl_2, the substance is carcinogenic. The support of the association rule of

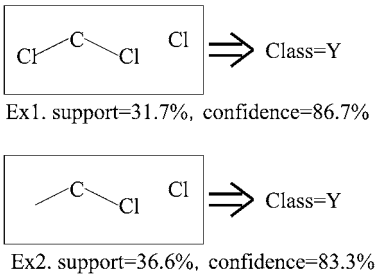


Figure 22. Extracted association rules.

example 2 is 33.6%, and its confidence is 83.3%. This rule implies that if there is a structure CCl and an atom that is single bonded to its C, and further there is a Cl that is not directly connected to C, the substance is carcinogenic.

The performance of our proposing method on the computational efficiency was also evaluated through the mining experiment for a larger data set. The objective data were obtained from the Website of National Toxicology Program (NTP) (Srinivasan et al., 1997). In total, the 300 compounds whose classes are known were selected for the analysis of which 185 compounds have positive carcinogenesis and the rests are negative. The types of atoms involved in the compounds are C, H, O, Cl, F, S and some cations, and the types of bonds are single, double, aromatic and cation bonds. The minimum support threshold was changed in the range from 10% to 20%. In each minimum support case, all frequent induced subgraphs were exhaustively discovered. The computation time required to complete the search was far longer for the smaller value of minimum support, and was almost 8 days for 10%, while it was only about 40 minutes for 20%. The size of the largest frequent induced subgraph discovered in the case of 10% was 13.

6. Discussion and related work

As described in the first section, an algorithm for the complete search of the structure which belongs to a more general class than the graph has been proposed by Dehaspe, Toivonen, and King (1998). They tried to mine carcinogenic substructures in the chemical molecules in form of the first order predicates using PROGOL which is a system used in inductive logic programming (ILP). They combined ILP principle with a level wise search technique to improve the search efficiency. However, the search space is still large and so only the substructure descriptions of short length consisting of six predicates at maximum were found within a tractable computation time even by the improved search technique. In the direct representation mode in which the molecule structure is described by the set of individual atoms and bonds such as $atomel(C, A1, c)$ and $bond(C, A1, A2, BT)$, the small frequent substructures consisting of only three atoms or so could be found since each predicate is consumed to represent either an atom or a bond. In the synthetic representation mode in which the descriptors of some characteristic molecule substructures based on the chemical background knowledge such as $aromatic - ring(C, S1)$ and $alcohol(C, S2)$ are

initially given in the data, the frequent substructures consisting of more than ten atoms could be found. The subsequent work of FARMAR proposed by Nijssen and Kok significantly increased the search efficiency by relaxing the equivalence criterion among clauses (Nijssen & Kok, 2001). However, their result includes some predicates having different forms but equivalent in the sense of the Θ-subsumption, and the class of the structures to be searched is limited to *connected* structures.

Though these approaches allow the introduction of variables in the predicates, most of the results of the carcinogenesis analysis were the grounded first order predicates which could be represented by graphs. In addition, our results shown in the previous section suggest many important structures consisting of some isolated subgraphs to characterize the data. In this regard, our approach showed the strength when analyzing complex structures, since the adjacency matrices have a strong and highly flexible expressiveness of the graph structures. Our approach is similar to the direct representation mode of Dehaspe et al. and is considered to be very powerful because it can exhaustively mine large and complex substructures including the substructures consisting of 13 atoms, while the past approaches could not achieve the exhaustive search because of the limit on the tractable computation time. If the synthetic nodes describing some characteristic substructures such as an aromatic ring and an alcohol base are introduced to the graph transaction, our approach is expected to mine far larger frequent substructures though this mode has not been assessed.

In our study, the association between the chemical substructures and the carcinogenesis was investigated exhaustively, but significant association was not observed. This fact is consistent with the claim made by Dehaspe et al. that the causation of chemical carcinogenesis is highly complex with many separate mechanisms involved (Dehaspe, Toivonen, & King, 1998). We also applied our approach to the chemical compound data of mutagenesis (Debnath et al., 1991). The chemical mechanism to cause the mutagenesis is considered among chemists to be more direct and simpler than the case of carcinogenesis. In fact, our approach found quite significant and complex substructures in this case (Inokuchi et al., 2000).

GBI and SUBDUE mentioned in the first section can use powerful measures such as information entropy, gini-index and description length to figure out important graph structures (Yoshida & Motoda, 1995; Cook & Holder, 1994). Chen et al. proposed a method to derive the longest access sequence and/or tree patterns among URLs (Chen, Park, & Yu, 1998). The method proposed by Liquiere and Sallantin completely searches homomorphically equivalent subgraphs (Liquiere & Sallantin, 1998). The method proposed by de Raedt and Kramer can mine sequence patterns in graphs characterized by the combination of monotonic and anti-monotonic criteria (de Raedt & Kramer, 2001). The method of Geibel and Wysotzki can derive frequent induced subgraphs of a limited size in graph data (Geibel & Wysotzki, 1996). Many of them work efficiently under some mining criteria which can be set in flexible manner to some extent. However these works take some measures of greedy search, heuristic search, limitation on graph size and/or strong limitation on the class of graphs. Thus, they are not suitable for applications that require the complete search of generic graph substructures.

In the area of the mathematics on the isomorphism and the subisomorphism of graphs, a great deal of research has been done. The most representative work on the graph

subisomorphism checking is the algorithm proposed by Ullman (1976). It checks if a given smaller graph is the general subgraph of another given larger graph. Its basic approach is to match the nodes and the links in a smaller graph to those in a larger graph by the depth first search (DFS). For the isomorphism checking between given two graphs, NAUTY (Mckay, 1990) would be the most well known program. It uses graph invariants to decompose the given graph to a set of subgraphs and to efficiently verify the isomorphism of these smaller subgraphs by node matching. However, since the invariants of the given graph such as degree of each node are not preserved in the subgraph, NAUTY is not suitable to check the existence of a subgraph in a given graph efficiently. Though these algorithms check the graph isomorphism or subisomorphism between given two graphs within tractable computation time, they have no capability to search for frequent subgraph patterns from a huge dataset.

7. Conclusion

We proposed a new algorithm by extending the Apriori algorithm to mine frequent induced subgraph structures in graph transaction data. The algorithm can handle general graphs and perform a complete search efficiently. A graph can be either directed or undirected, and have any number of labels for nodes and links and any loops including self-loops. Its performance was evaluated in terms of the required computation time for number of transactions, size of transactions, number of node labels, values of minimum support and values for link probability. The problem to search induced subgraph isomorphism is known to be between NP-complete and PB-complete problems. Though the computational complexity shown in the experiments through simulated graph structured datasets is consistent with the theoretical insight, the efficiency is confirmed to be practical for many real world purposes. Two real world problems were chosen for applications, and some characteristic frequent subgraph patterns were discovered.

For the future work, the following two issues are currently under development.

(1) The generalization of the adjacency matrix representation and the improvement of the efficiency to search frequent induced subgraphs.
(2) The extension to the other class of subgraphs and mining criteria.

Currently, a new representation of the adjacency matrix which can directly handle the link information is sought for the first issue. For the second issue, new approaches to focus on the mining of the connected subgraphs and the mining under non-monotonic criteria are expected to have contributions to real-world applications.

Appendix

Theorem 1. *The first matrix of a canonical form matrix is also the canonical form matrix.*

Proof: Let C_k and C_{k-1} be as follows.

$$C_k = \begin{pmatrix} C_{k-1} & \boldsymbol{c}_1 \\ \boldsymbol{c}_2^T & 0 \end{pmatrix} = \begin{pmatrix} 0 & c_{1,2} & c_{1,3} & \cdots & c_{1,k} \\ c_{2,1} & 0 & c_{2,3} & \cdots & c_{2,k} \\ c_{3,1} & c_{3,2} & 0 & \cdots & c_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{k,1} & c_{k,2} & c_{k,3} & \cdots & 0 \end{pmatrix}$$

$$\begin{aligned} code(C_{k-1}) &= c_{1,2}c_{1,3}c_{2,3}c_{1,4}c_{2,4}\cdots c_{k-4,k-1}c_{k-3,k-1}c_{k-2,k-1} \\ code(C_k) &= c_{1,2}c_{1,3}c_{2,3}c_{1,4}c_{2,4}\cdots c_{k-3,k}c_{k-2,k}c_{k-1,k} \\ &= code(C_{k-1})c_{1,k}\cdots c_{k-1,k} \end{aligned}$$

If C_k is the canonical form and C_{k-1} is not the canonical form, adjacency matrices of normal forms C'_k and C'_{k-1} which meet the following conditions exit.

$$C'_k = \begin{pmatrix} C'_{k-1} & \boldsymbol{c'}_1 \\ \boldsymbol{c'}_2^T & 0 \end{pmatrix}, \quad G(C_k) = G(C'_k)$$

$$G(C_{k-1}) = G(C'_{k-1}), \quad code(C'_{k-1}) < code(C_{k-1})$$

Thus,

$$code(C_k) = code(C_{k-1})c_{1,k}\cdots c_{k-1,k} > code(C'_k) = code(C'_{k-1})c_{1,k}\cdots c_{k-1,k}.$$

This is contradictory to the fact that C_k is the canonical form matrix. Thus, the first matrix of the canonical form matrix is also the canonical form. □

Theorem 2. *Let the adjacency matrix for which the m-th node of X_k ($1 \leq m \leq k$, $N(X_k, m) = N(X_k, k)$) is deleted be X^m_{k-1}. Also let the canonical forms of X_k and X^m_{k-1} be C_k and $can(X^m_{k-1})$ respectively, then*

$$code(C_{k-1}) \leq code\big(can\big(X^m_{k-1}\big)\big).$$

The equality holds when $G(C_{k-1})$ and $G(X^m_{k-1})$ are isomorphic.

Proof: C_k can be represented as

$$C_k = \begin{pmatrix} C_{k-1} & \boldsymbol{c}_1 \\ \boldsymbol{c}_2^T & 0 \end{pmatrix}.$$

As X_k and C_k are isomorphic, $G(C_{k-1})$ is one of induced subgraphs of X_k. If A_{k-1} represents an induced subgraph of X_k and meet

$$code(A_{k-1}) < code(C_{k-1}), \tag{12}$$

adjacency matrix A_k which meets

$$G(X_k) = G(C_k) = G(A_k) \quad \text{where } A_k = \begin{pmatrix} A_{k-1} & \boldsymbol{a}_1 \\ \boldsymbol{a}_2^T & 0 \end{pmatrix}$$

exists. By Eq. (12),

$$code(A_k) < code(C_k).$$

It is contradictory to the fact that C_k is canonical form. Thus,

$$code(C_{k-1}) \leq code(A_{k-1}).$$

Above derivation shows that the codes of any induced subgraphs of X_k whose sizes are $k-1$ are greater than or equal to $code(C_{k-1})$. Thus,

$$code(C_{k-1}) \leq code\big(can\big(X_{k-1}^m\big)\big). \tag{13}$$

As C_{k-1} is the canonical form based on Theorem 1, *l.h.s* of Eq. (13) equals its *r.h.s* in the case that $G(C_{k-1})$ and $G(X_{k-1}^m)$ are isomorphic. □

Corollary 1. *An m and a transformation matrix W_k exist, where C_k is given by*

$$C_k = \begin{pmatrix} C_{k-1} & \boldsymbol{c}_1 \\ \boldsymbol{c}_2^T & 0 \end{pmatrix} = \begin{pmatrix} can\big(X_{k-1}^m\big) & \boldsymbol{c}_1 \\ \boldsymbol{c}_2^T & 0 \end{pmatrix} = W_k^T X_k W_k.$$

Theorem 3. *Eq.* (8) *gives a pseudo-canonical form of X_k.*

Proof: A matrix S_k^m for X_k is represented by

$$S_k^m = \begin{pmatrix} S_{k-1}^m & 0 \\ 0 & 1 \end{pmatrix}.$$

And a matrix T_k^m for X_k is represented by

$$T_k^m = P_k^m Q_k^m,$$

where

$$P_k^m = \begin{array}{c} \\ 1 \\ \vdots \\ m-1 \\ m \\ m+1 \\ \vdots \\ k \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & \cdots & m-1 & m & \cdots & k-1 & k \end{array} \\ \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & 0 & & \\ & & 1 & & & & \\ & & & & & & 1 \\ & & & 1 & & & \\ & 0 & & & \ddots & & \\ & & & & & 1 & \end{pmatrix} \end{array}$$

and

$$Q_k^m = \begin{pmatrix} T_{k-1}^m & 0 \\ 0 & 1 \end{pmatrix}.$$

Thus,

$$\begin{aligned} \left(T_k^m S_k^m\right)^T X_k \left(T_k^m S_k^m\right) &= \left(P_k^m Q_k^m S_k^m\right)^T X_k \left(P_k^m Q_k^m S_k^m\right) \\ &= \left(Q_k^m S_k^m\right)^T P_k^{mT} X_k P_k^m \left(Q_k^m S_k^m\right) \\ &= \left(S_k^m\right)^T Q_k^{mT} \begin{pmatrix} X_{k-1}^m & \boldsymbol{x}_1' \\ \boldsymbol{x}_2'^T & 0 \end{pmatrix} Q_k^m S_k^m \\ &= (S_k^m)^T \begin{pmatrix} \left(T_{k-1}^m\right)^T X_{k-1}^m T_{k-1}^m & \left(T_{k-1}^m\right)^T \boldsymbol{x}_1' \\ \left(\left(T_{k-1}^m\right)^T \boldsymbol{x}_2'\right)^T & 0 \end{pmatrix} S_k^m \\ &= \begin{pmatrix} \left(T_{k-1}^m S_{k-1}^m\right)^T X_{k-1}^m T_{k-1}^m S_{k-1}^m & \left(T_{k-1}^m S_{k-1}^m\right)^T \boldsymbol{x}_1' \\ \left(\left(T_{k-1}^m S_{k-1}^m\right)^T \boldsymbol{x}_2'\right)^T & 0 \end{pmatrix} \end{aligned}$$

Because the canonical form matrix of X_k is a normal form matrix, its rows and columns must be lexicographically ordered, and so is the pseudo-canonical form matrix. From this constraint and the above expression, the label of the m-th node must be the last one in the lexicographical order. Thus, $N(X_k, m) = N(X_k, k)$ must be hold. If $m = m_{min}$ minimizes $code((T_{k-1}^m S_{k-1}^m)^T X_{k-1}^m T_{k-1}^m S_{k-1}^m)$, the following relation holds from Theorem 2.

$$\begin{aligned} &\left(T_k^{m_{min}} S_k^{m_{min}}\right)^T X_k \left(T_k^{m_{min}} S_k^{m_{min}}\right) \\ &= \begin{pmatrix} \left(T_{k-1}^{m_{min}} S_{k-1}^{m_{min}}\right)^T X_{k-1}^{m_{min}} T_{k-1}^{m_{min}} S_{k-1}^{m_{min}} & \left(T_{k-1}^{m_{min}} S_{k-1}^{m_{min}}\right)^T \boldsymbol{x}_1' \\ \left(\left(T_{k-1}^{m_{min}} S_{k-1}^{m_{min}}\right)^T \boldsymbol{x}_2'\right)^T & 0 \end{pmatrix} \\ &= \begin{pmatrix} can\left(X_{k-1}^{m_{min}}\right) & \boldsymbol{c}^{\boldsymbol{p}}{}_1 \\ \boldsymbol{c}^{\boldsymbol{p}}{}_2^T & 0 \end{pmatrix} = \begin{pmatrix} C_{k-1} & \boldsymbol{c}^{\boldsymbol{p}}{}_1 \\ \boldsymbol{c}^{\boldsymbol{p}}{}_2^T & 0 \end{pmatrix} \end{aligned}$$

From this fact and Corollary 1, Eq. (8) gives a pseudo-canonical form of X_k, i.e., C_k^p. □

Acknowledgments

The authors acknowledge the support provided by Mr. Kumazawa and Mr. Arai at Recuit Co. Ltd. and Prof. Okada at Kwansei Gakuin University. Furthermore, we extend our thanks to anonymous referees for their valuable suggestions.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. of Twentyth Very Large Dada Base Conference: VLDB'94* (pp. 487–499).

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In *Proc. of Eleventh International Conference on Data Engineering: ICDE'95* (pp. 3–14).

Barabasi, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science, 286*, 509–512.

Biggs, N. (1974). *Algebraic graph theory*. Cambridge: Cambridge University Press.

Chen, M. S., Park, J. S., & Yu, P. S. (1998). Efficient data mining for path traversal patterns. *IEEE Transaction on Knowledge and Data Engineering, 10:2*, 209–221.

Cook, D. J., & Holder, L. B. (1994). Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research, 1*, 231–255.

Debnath, A. K. et al. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. *J. Med. Chem, 34*, 786–797.

Dehaspe, L., Toivonen, H., & King, R. D. (1998). Finding frequent substructures in chemical compound. In *Proc. of Fourth International Conference on Knowledge Discovery and Data Mining: KDD'98* (pp. 30–36).

de Raedt, L., & Kramer, S. (2001). The levelwise version space algorithm and its application to molecular fragment finding. In *Proc. of Seventeenth International Joint Conference on Artificial Intelligence: IJCAI'01* (Vol. 2) (pp. 853–859).

Fortin, S. (1996). The graph isomorphism problem. Technical Report 96-20, University of Alberta, Edmonton, Alberta, Canada.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York: W. H. Freeman.

Geibel, P., & Wysotzki, F. (1996). Learning relational concepts with decision trees. In *Proc. of Thirteenth International Conference on Machine Learning: ICML'96* (pp. 166–174).

Hogg, T. (1996). Refining the phase transition in combinatorial search. *Artificial Intelligence, 81:1/2*, 127–154.

Inokuchi, A. Washio, T., & Motoda, H. (1999). Basket analysis for graph structured data. In *Proc. of Third Pacific-Asia Conference on Knowledge Discovery and Data Mining: PAKDD'99* (pp. 420–431).

Inokuchi, A. et al. (2000). Application of frequent substructure mining to mutagenesis data analysis. *Working notes of International Workshop of KDD Challenge on Real-world Data*, PAKDD2000.

Kann, V. (1995). Strong lower bounds on the approximability of some NPO PB-complete maximization problem. In *MFCS 95*, LNCS (Vol. 969) (pp. 227–236).

Liquiere, M., & Sallantin, J. (1998). Structural machine learning with Galois lattice and graphs. In *Proc. of Fifteenth International Conference on Machine Learning: ICML'98* (pp. 305–313).

Mannila, H., Toivonen, H., & Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery, 1:3*, 259–289.

Mckay, B. D. (1990). NAUTY users guide (version 1.5). Technical Report, TR-CS-90-02, Department of Computer Science, Australian National University.

Nijssen, S., & Kok, J. N. (2001). Faster association rules for multiple relations. In *Proc. of Seventeenth International Joint Conference on Artificial Intelligence: IJCAI'01* (Vol. 2) (pp. 891–896).

Read, R., & Corneil, D. (1977). The graph isomorphism disease. *Journal of Graph Theory, 1*, 339–363.

Srikant, R., Vu, Q., & Agrawal, R. (1997). Mining association rules with item constraints. In *Proc. of Third International Conference on Knowledge Discovery and Data Mining: KDD'97* (pp. 67–73).

Srinivasan, A., King, R. D., Muggleton, S. H., & Sternberg, M. J. E. (1997). The predictive toxicology evaluation challenge. In *Proc. of Fifteenth International Joint Conference on Artificial Intelligence: IJCAI'97* (pp. 4–9).

Ullman, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM, 23:1*, 31–32.

Walsh, T. (2001). Search on high degree graphs. In *Proc. of Seventeenth International Conference on Artificial Intelligence: IJCAI'2001* (pp. 266–271).

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of small-world networks. *Nature, 393*, 440–442.

Yoshida, K., & Motoda, H. (1995). Clip: Concept learning from inference pattern. *Artificial Intelligence, 75:1*, 63–92.

Received October 20, 2000
Revised June 24, 2002
Accepted June 24, 2002
Final manuscript September 1, 2002