Fundamenta Informaticae 66 (2005) 53–82
IOS Press

A General Framework for Mining Frequent Subgraphs from Labeled Graphs

Akihiro Inokuchi*
Tokyo Research Laboratory
IBM Japan
1623-14, Shimotsuruma, Yamato, Kanagawa, 242-8502, Japan
inokuchi@jp.ibm.com

Takashi Washio and Hiroshi Motoda
The Institute of Scientific and Industrial Research
Osaka University
8-1, Mihogaoka, Ibaraki, Osaka, 567-0047, Japan
washio@ar.sanken.osaka-u.ac.jp
motoda@ar.sanken.osaka-u.ac.jp

Abstract. The derivation of frequent subgraphs from a dataset of labeled graphs has high computational complexity because the hard problems of isomorphism and subgraph isomorphism have to be solved as part of this derivation. To deal with this computational complexity, all previous approaches have focused on one particular kind of graph. In this paper, we propose an approach to conduct a complete search for various classes of frequent subgraphs in a massive dataset of labeled graphs within a practical time. The power of our approach comes from the algebraic representation of graphs, its associated operations and well-organized bias constraints to limit the search space efficiently. The performance has been evaluated using real world datasets, and the high scalability and flexibility of our approach have been confirmed with respect to the amount of data and the computation time.

Keywords: Data Mining, Graph Mining, Frequent Subgraph, Bias, Canonical Form, Subgraph Isomorphism

* Address for correspondence: Tokyo Research Laboratory, IBM Japan, 1623-14, Shimotsuruma, Yamato, Kanagawa, 242-8502, Japan

1. Introduction

Graph mining algorithms that discover characteristic subgraph patterns embedded in a dataset of labeled graphs have a broad range of applications. However it is hard to develop methods with practical run times because the search for candidate frequent subgraphs has exponential complexity and includes the subgraph isomorphism problem, which is known to be NP-complete.

To address these issues, various approaches to mine a complete set of frequent patterns from massive datasets of labeled graphs or labeled trees have been proposed. Although each method can efficiently discover the patterns, the subgraphs to be searched are limited within a specific class. For example, MolFea efficiently mines frequent paths from labeled graphs [7]. TreeMiner [27] and FREQT [2] can quickly discover all frequent patterns from ordered trees. However, they cannot mine more complex substructures such as labeled subgraphs. On the other hand, the AGM algorithm [10, 12], FSG [17], and gSpan [24] can mine frequent subgraphs from a set of labeled graphs. However, they cannot efficiently discover frequent patterns of paths and trees, because their data structures and their search operations are not dedicated to path and tree structure mining.

In this paper, we propose a generic and efficient framework to mine various classes of substructures. By introducing a bias for each class of substructure, *e.g.*, connected subgraphs, ordered subtrees, and path structures, to the AGM algorithm, a complete search for the frequent substructures of each class is achieved. The bias includes restrictions on the search space of the frequent patterns, on the ambiguity of the structural representation, and on the criteria used for subgraph isomorphism checking. We call this new framework *Biased Apriori-based Graph Mining (B-AGM)*. We evaluate its performance in terms of the required computation time for real world datasets of various sizes.

The rest of this paper is organized as follows. Section 2 defines the frequent subgraph pattern mining problem, and describes the basic concepts of the Apriori-based Graph Mining algorithm used for mining frequent patterns in a dataset consisting of labeled graphs. Section 3 defines some additional specific biases to derive various types of patterns, *e.g.*, general subgraphs, connected subgraphs, ordered subtrees and path patterns. Section 4 provides an experimental evaluation of our algorithm on some real datasets consisting of chemical compounds, Web access logs, and XML data. In Section 6, we discuss future extension of our framework. We provide a discussion and some related work in Section 6, and finally conclude in Section 7.

2. Problem Definitions and the AGM algorithm

We use the basic principles of the AGM algorithm in our extended framework. By applying some specific biases to the algorithm, our B-AGM discovers frequent subgraphs of various classes. In this section, we define the problem and explain the AGM algorithm. In the next section, we propose biases to enable graph mining of various classes.

2.1. Problem Definition

The input for frequent subgraph mining is a set of labeled graphs in which each vertex and each edge have a vertex label and an edge label respectively. The label of each vertex (edge) does not need to be unique, and it is possible that the same label can be used for several vertices (edges). Each graph in the dataset is represented as $G = (V, E, L_V, E_V, lb)$, where $V = \{v_1, v_2, \ldots, v_k\}$, $E = \{(v_i, v_j) | v_i, v_j \in V(G)\}$,

$L_V(V) = \{lb(v_i)|\forall v_i \in V(G)\}$, $L_E(E) = \{lb((v_i, v_j))|\forall (v_i, v_j) \in E(G)\}$, and $lb : (V \to L_V) \cup (E \to L_E)$ are sets of vertices, edges, vertex labels, edge labels, and a function to assign a label to a vertex or to an edge, respectively. To the convenience of the description, the sets of vertices, edges, vertex labels, and edge labels of the labeled graph G are represented as $V(G)$, $E(G)$, $L_V(G)$, and $L_E(G)$, respectively. The number of vertices, $|V(G)|$, is called the "size" of the graph G.

A graph can be represented by using an "adjacency matrix". For calculation efficiency, let $num(lb(v_i))$ and $num(lb((v_i, v_j)))$ be natural numbers assigned to a vertex label $lb(v_i)$ and an edge label $lb((v_i, v_j))$, respectively. Given a labeled graph G, the (i, j)-element $x_{i,j}$ of an adjacency matrix X_k of the graph G whose size is k is represented as follows.

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

where $i, j \in \{1, 2, 3, \cdots, k\}$. 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 $G(X_k)$.

By choosing different assignments of rows and columns to vertices in a graph, multiple adjacency matrix representations for a single graph can be obtained. To remove this ambiguity, we use a "canonical form" of the adjacency matrices to represent a graph. To mathematically define the canonical form of a graph and to deal efficiently with matrices, the code of an adjacency matrix is defined as follows. For an undirected graph, the function $code$ of an adjacency matrix X_k is defined as

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

which is a concatenation of the (i, j)-element with $x_{i,j}$ as shown in Figure 1. For a directed graph, it is defined as

$$code(X_k) = c_1c_2c_3c_4 \cdots c_n, \quad (n = \frac{k(k-1)}{2}), \text{ and}$$

$$c_l = (|L_E| + 1)x_{j,i} + x_{i,j}, \quad (l = i + \frac{(j-1)(j-2)}{2}),$$

where $i < j$. Furthermore, a function $CODE$ including the vertex labels is defined as

$$CODE(X_k) = num(X_k)code(X_k),$$

which is a concatenation of $num(X_k)$ and $code(X_k)$, and

$$num(X_k) = num(lb(v_1)) \cdots num(lb(v_k)).$$

The canonical form of the adjacency matrices representing a graph is the unique matrix having the maximum (or minimum) $CODE$. The choice of the maximum or minimum $CODE$ depends on a class of substructure patterns to be mined, which is included in the definition of each bias. For example, when connected subgraphs are mined, the maximum $CODE$ is used to define the canonical form, while the minimum $CODE$ is used for subtree mining. Both are applicable to the conventional AGM algorithm.

$$X_k = \begin{array}{c} \\ lb(v_1) \\ lb(v_2) \\ lb(v_3) \\ \vdots \\ lb(v_k) \end{array} \begin{array}{c} \begin{array}{ccccc} lb(v_1) & lb(v_2) & lb(v_3) & \cdots & lb(v_k) \end{array} \\ \begin{pmatrix} 0 & x_{1,2} & x_{1,3} & \cdots & x_{1,k} \\ x_{2,1} & 0 & x_{2,3} & \cdots & x_{2,k} \\ x_{3,1} & x_{3,2} & 0 & \cdots & x_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{k,1} & x_{k,2} & x_{k,3} & \cdots & 0 \end{pmatrix} \end{array}$$

Figure 1. Order of Matrix Elements to Define a Function $code$ for an Undirected Graph.

$$X_k = \begin{array}{c} \\ lb(v_1) \\ lb(v_2) \\ lb(v_3) \\ \vdots \\ lb(v_k) \end{array} \begin{array}{c} \begin{array}{ccccc} lb(v_1) & lb(v_2) & lb(v_3) & \cdots & lb(v_k) \end{array} \\ \begin{pmatrix} 0 & x_{1,2} & x_{1,3} & \cdots & x_{1,k} \\ x_{2,1} & 0 & x_{2,3} & \cdots & x_{2,k} \\ x_{3,1} & x_{3,2} & 0 & \cdots & x_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{k,1} & x_{k,2} & x_{k,3} & \cdots & 0 \end{pmatrix} \end{array}$$

Figure 2. Order of Matrix Elements to Define a Function $code$ for a Directed Graph.

Adjacency matrices corresponding to an identical graph are mutually convertible using the following "transformation matrix" (permutation matrix). When adjacency matrices X_k and Y_k representing an identical graph of size k are given, each element $t_{i,j}$ of a transformation matrix T_k is defined as follows.

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

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

Given graphs G and G_s, if there is a function $\phi : V(G_s) \to V(G)$ that satisfies

1. $\forall v \in V(G_s),\ \ \phi(v) \in V(G),\ \ lb(v) = lb(\phi(v))$, and
2. $\forall (v_i, v_j) \in E(G_s),\ \ (\phi(v_i), \phi(v_j)) \in E(G),\ \ lb((v_i, v_j)) = lb((\phi(v_i), \phi(v_j)))$.

G_s is a "subgraph" of G, which is represented as $G_s \sqsubseteq G$. Additionally, if the function satisfies

3. $(v_i, v_j) \in E(G_s) \Leftrightarrow (\phi(v_i), \phi(v_j)) \in E(G)$,

then G_s is an "induced subgraph" of G, which is represented as $G_s \sqsubseteq_i G$.

A "path" is a sequence of consecutive vertices and edges in a graph. Given an undirected graph G, if a path exists between any two vertices of the graph, then G is called a "connected graph". In the case of a directed graph G, G is called a connected graph if a path exists between any two vertices in the undirected graph obtained by ignoring the directions of the edges in G. An "unordered tree" is a directed

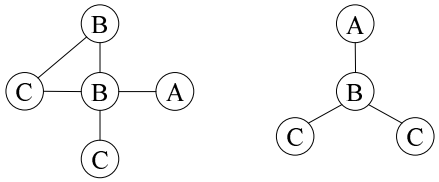


Figure 3. Examples of Labeled Graphs.

acyclic graph with a root vertex and where every other vertex has one entering edge. An "ordered tree" is a tree with a left-to-right ordering among the children of each vertex.

Given a set of labeled graphs GD, the "support" $sup(G_s)$ of an induced subgraph pattern G_s is defined as

$$sup(G_s) = \frac{|\{G|G \in GD, G_s \sqsubseteq_i G\}|}{|GD|},$$

where $\sqsubseteq_i$ stands for inclusion of an induced subgraph in a graph. When a user would like to derive all of the frequent patterns that are contained as subgraphs, the support is defined as

$$sup(G_s) = \frac{|\{G|G \in GD, G_s \sqsubseteq G\}|}{|GD|}.$$

There is an induced subgraph derivation and a general subgraph derivation for each class of structure except for a subtree. These derivations are introduced independently of any bias for each class of structure which is defined in Section 3. By combining an induced or general subgraph derivation with a bias, the B-AGM algorithm can mine the frequent induced subgraphs separately from the frequent general subgraphs. Any derived subgraph having support greater than or equal to the "minimum support" specified by a user is called a "frequent subgraph". A frequent subgraph with k vertices is called a frequent k-subgraph. When a dataset which consists of labeled graphs and the minimum support are given as input, the frequent subgraph mining problem is to derive all frequent subgraphs in the dataset that have support greater than or equal to the minimum support value [11].

For example, two labeled graphs as shown in Figure 3 are given as an input dataset GD, where the numbers 1, 2, and 3 are assigned to A, B, and C, respectively, and 1 is assigned to an edge label. The canonical form of the graph G_1 in Figure 3 is expressed as

$$X_5 = \begin{array}{c} \\ C \\ C \\ B \\ B \\ A \end{array} \begin{array}{c} \begin{array}{ccccc} C & C & B & B & A \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}.$$

The $CODE$ of X_5 is represented as

$$CODE(X_5) = 33221\,0111010010,$$

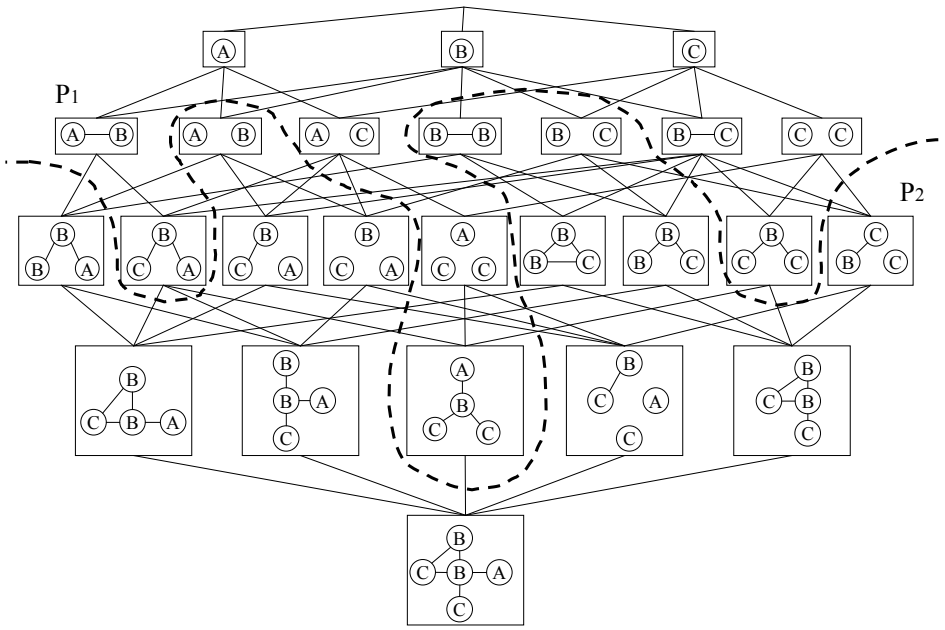


Figure 4. Search Space to Mine Subgraphs in Data in Figure 3.

where the italic characters represent $num(X_5)$. When the dataset GD is given as input and the minimum support is set to 100%, the search space for mining frequent induced subgraphs is as represented in Figure 4, where each graph in a rectangle corresponds to a subgraph pattern which has support greater than 0%. (In Figure 4, any pattern whose support is 0% is omitted due to space limitations.) A rectangle for a subgraph pattern is linked to the rectangles of its induced subgraphs. The subgraph patterns above the dashed line in Figure 4 are the frequent subgraphs. The support value of the subgraph P_1 in Figure 4 is $sup(P_1) = \frac{2}{2} = 100\%$. The support value of the subgraph P_2 is $sup(P_2) = 50\%$, and the subgraph is not added to the set of frequent subgraphs.

2.2. Apriori-based Graph Mining Algorithm

In our previous work, we proposed an approach named AGM (Apriori-based Graph Mining) algorithm in which the knowledge representation and the search operations are highly dedicated to the graph structured data mining [10, 12]. The AGM algorithm is so generic that it can discover not only connected frequent subgraphs, but also disconnected frequent subgraphs. We use the basic concept of the AGM algorithm as the framework for frequent subgraph mining. By adding some additional biases, the AGM framework can discover various types of subgraphs, such as connected subgraphs, subtree structures, and path structures.

The AGM algorithm derives all frequent subgraphs in ascending order of the size of the graph based on the anti-monotonic property of the support measure. Frequent subgraphs are derived stepwise from the top in the lattice search space as depicted in Figure 4. Figure 5 is the outline of our AGM algorithm. First, a 1×1 adjacency matrix representing a vertex is generated for every vertex, and they are substituted for C_1 (Line 1). Next, the support for the each candidate frequent subgraph is calculated by scanning

the database (Lines 4 and 5). Next, the Generate-Candidate function generates the candidate frequent subgraphs of size $k+1$ from the frequent k-subgraphs in F_k, and they are substituted for C_{k+1} (Line 6). These steps are repeated until C_k becomes empty. Finally, all of the frequent subgraphs are returned (Line 9).

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

Figure 5. Outline of the Apriori-based Graph Mining Algorithm.

2.2.1. Join Operation

The Generate-Candidate function referenced in Figure 5 consists of three parts: the join operation, the subgraph-check operation, and the canonicalize operation. In the join operation, the adjacency matrices of the candidate frequent subgraphs of size $k+1$ are generated by joining the two adjacency matrices of the frequent k-subgraphs in F_k. Given two adjacency matrices X_k and Y_k representing the frequent subgraphs, they are joinable if and only if all of the conditions to join are satisfied.

Condition 1. Let $V(G(X_k))$ and $V(G(Y_k))$ be $\{x_1, x_2, \cdots, x_k\}$ and $\{y_1, y_2, \cdots, y_k\}$ respectively, where x_i is the i-th vertex of $G(X_k)$ and y_i is the i-th vertex of $G(Y_k)$. X_k and Y_k are identical except for the k-th row and the k-th column, *i.e.*,

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

$$lb(x_i) = lb(y_i) \text{ for } i = 1, \cdots, k-1.$$

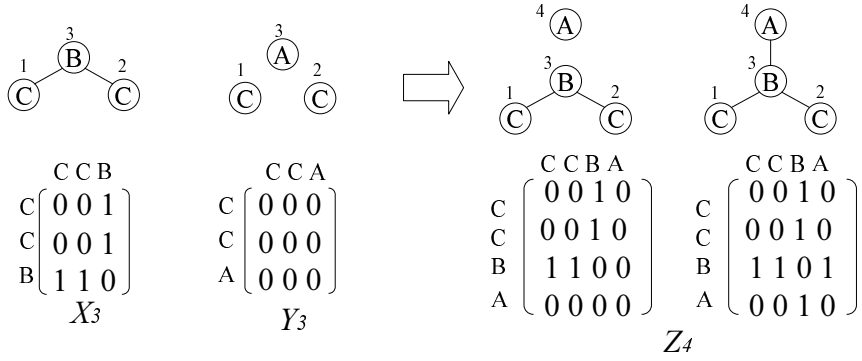


Figure 6. Example of Join Operation.

Condition 2. X_k is the canonical form of $G(X_k)$.
Condition 3. $CODE(X_k) \geq CODE(Y_k)$ is fulfilled.[1]

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

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

$$lb(x_i) = lb(z_i) \text{ for } i = 1, \cdots, k-1, \;\; lb(x_k) = lb(z_k), \text{ and } lb(y_k) = lb(z_{k+1}).$$

These matrices X_k and Y_k are called the "first generator matrix" and the "second generator matrix" of Z_{k+1}, respectively. The two elements $z_{k,k+1}$ and $z_{k+1,k}$ of Z_{k+1} are not determined by X_k and Y_k. For the undirected graph, the possible graph structures for $G(Z_{k+1})$ are those where there is a labeled edge or where there is no edge between k-th vertex and $(k+1)$-th vertex. For these undirected graphs, the $(|L_E|+1)$ adjacency matrices under $z_{k,k+1} = z_{k+1,k}$ are then generated, where $|L_E|$ is the number of edge labels, while the $(|L_E|+1)^2$ adjacency matrices are generated for directed graphs. The adjacency matrix generated under the above conditions is called a "normal form".

Figure 6 shows an example of the join operation when there is only one edge label in the undirected graphs and $num(A) < num(B) < num(C)$. Since X_k and Y_k are joinable, the two adjacency matrices Z_{k+1} are generated, where the difference is the pair consisting of the $(3,4)$-element and the $(4,3)$-element. In the two matrices, each pair consists of 0s or 1s.

2.2.2. Subgraph-Check Operation

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 according to the anti-monotonic property of the support. This condition reduces the candidates. When the subgraph-check operation for a graph of size $k+1$ is done, it can be assumed that one of transformation matrices from every normal form matrix to its canonical form matrix whose size is less than $k+1$ is known, since the complete search was done in the previous k steps.

[1] In the case that the canonical form is defined as the unique matrix having the *minimum* $CODE$, the condition 3 is $CODE(X_k) \leq CODE(Y_k)$.

```
0)  Normalize(Z_k) {
1)    i ← 1;
2)    while(i ≠ k + 1){
3)      if(Z_i is a normal form and can become the first generator matrix){
4)        Z_k ← P'_k^T Z_k P'_k;
5)        i ← i + 1;
6)      }else{
7)        Z_k ← Q_k^T Z_k Q_k;
8)        i ← i − 1;
9)      }
10)   }
11)   return Z_k;
12) }
```

Figure 7. Normalization Algorithm.

An adjacency matrix Z_k of an induced subgraph of size k is obtained by removing the elements in the i-th row and the i-th column ($1 \leq i \leq k+1$) of Z_{k+1}. Then Z_k is transformed into the normal form by applying the algorithm shown in Figure 7. This is necessary because the AGM algorithm generates only normal form matrices, and support of Z_k is easily checked by using the normal form matrices obtained in the earlier steps. Let the upper left $i \times i$ submatrix of the adjacency matrix Z_k be Z_i, the matrix to transform Z_i into the canonical form be P_i and the unit matrix of size k be I_k. The transformation matrices P'_k and Q_k in Figure 7 are generated as follows.

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

Line 4 in Figure 7 is the operation to transform Z_i into its canonical form, and Line 7 changes the $(i-1)$-th vertex into the k-th vertex and the j-th vertex into the $(j-1)$-th vertex ($j = i, i+1, \cdots, k$).

2.2.3. Canonicalization Operation

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

When the canonical form of X_k and its associated transformation matrix are searched for, it can be assumed that one of the transformation matrices from each normal form matrix into its canonical form matrix of size $k-1$ is known, because of the stepwise extension of the graph size in the search. Let the transformation matrix of X^i_{k-1} to its normal form be T^i_{k-1} where X^i_{k-1} is the adjacency matrix obtained by removing the elements in the i-th row and i-th column of X_k. Also let the matrix to transform the

normalized matrix $(T^i_{k-1})^T X^i_k T^i_{k-1}$ into its canonical form be S^i_{k-1}. The transformation matrices S^i_k and T^i_k for X_k are generated by using the following equations from S^i_{k-1} and T^i_{k-1}.

$$S^i_k = \begin{pmatrix} S^i_{k-1} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{pmatrix}, \text{ and } T^i_k = \begin{pmatrix} I_{i-1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & 1 \\ \boldsymbol{0} & I_{k-i} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} T^i_{k-1} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{pmatrix}.$$

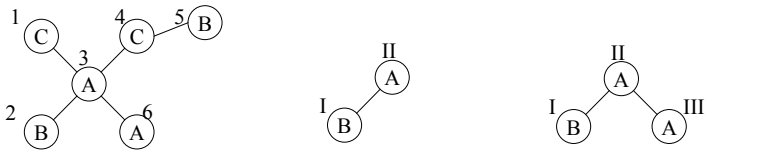
A canonical form for X_k is given by

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

The matrix to transform X_k into its canonical form is represented as $T^i_k S^i_k$, which makes Equation (1) the maximum. If the transformation matrix S'_k which transforms X_k through a $(T^i_k S^i_k)^T X_k (T^i_k S^i_k)$ into the canonical form has already been found, the canonical form of X_k is provided as $S'^T_k (T^i_k S^i_k)^T X_k (T^i_k S^i_k) S'_k$, and thus the calculations for all of the is in Equation (1) are not required. It should be noted, however, that the canonical form might 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 method are described in detail in the literature [12].

2.2.4. Counting the Frequency of Each Candidate

After all of the canonical forms of the candidate subgraphs are obtained, the database is accessed, and the frequency of each candidate subgraph is calculated. It is known that subgraph isomorphism [8] is NP-complete, and ordered subtree isomorphism matching and subtree isomorphism matching require $O(|C||T|)$ time and space where $|C|$ and $|T|$ are the sizes of the two graphs for isomorphism [14].



(a) Graph G_K (b) Candidate Subgraph $G(X_{k-1})$ (c) Candidate Subgraph $G(X_k)$

Figure 8. Graph Data and Candidate Subgraphs.

We now explain the counting in the case of frequent induced subgraph derivation. Let the canonical form of the candidate k-subgraph be X_k, its first generator matrix be X_{k-1}, and the graph in the database of size K be G_K. For example, let G_K, $G(X_{k-1})$, and $G(X_k)$ be the graphs in Figure 8 (a), (b), and (c) respectively. The canonical forms of Figure 8 (b) and (c) are expressed as

$$X_{k-1} = \begin{matrix} \\ B \\ A \end{matrix} \begin{matrix} \begin{matrix} B & A \end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{matrix}, \text{ and } X_k = \begin{matrix} \\ B \\ A \\ A \end{matrix} \begin{matrix} \begin{matrix} B & A & A \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{matrix},$$

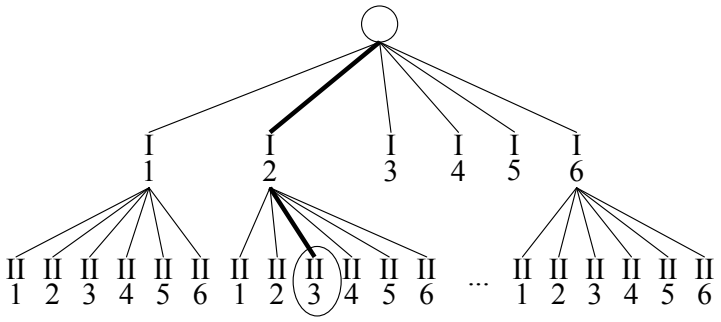


Figure 9. Search Tree for G_K and $G(X_{k-1})$.

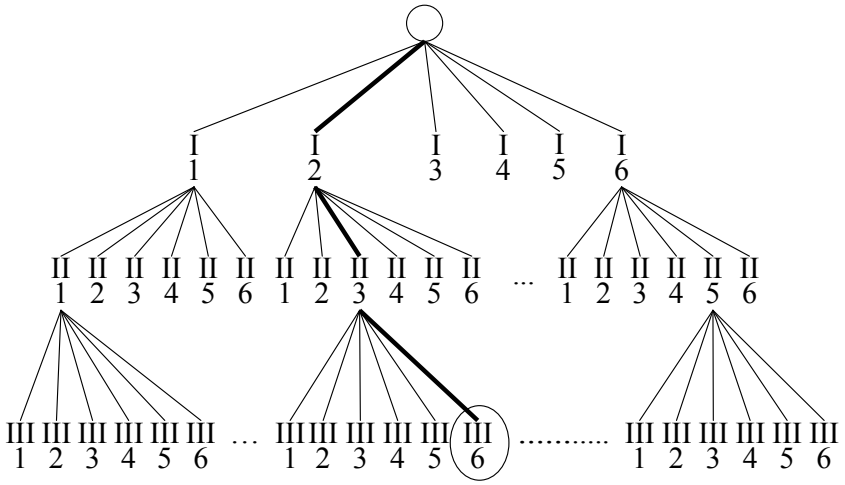


Figure 10. Search Tree for G_K and $G(X_k)$.

where $k = 3$ and $num(B) > num(A)$. The numbers assigned to the vertices in Figure 8 are vertex IDs. If the brute force method checks whether G_K includes the graph $G(X_{k-1})$ by a depth first search in the ascending order of the 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 for this case is shown in Figure 9. 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 Figure 10. In this case, the search in the part on the left side of the path root-I2-II3 in Figure 10 is not necessary since this part has already been checked in Figure 9. Therefore, if the correspondence relation of the vertex of G_K and $G(X_{k-1})$ is recorded, G_K's inclusion of a graph structure which has X_{k-1} as the first generator matrix can be efficiently checked.

We use this method as the default method to count the frequency. However, this method is implemented so that it can be overwritten. As mentioned later, the method is modified to compare B-AGM with other tree mining methods.

2.3. Completeness of Search of Frequent Subgraphs

The completeness of the search of frequent subgraphs in this join operation is proven as follows:

Theorem 2.1. Given a canonical form matrix X_{k+1} of an undirected graph and its $CODE(X_{k+1}) = num(X_{k-1})num(lb(x_k))num(lb(x_{k+1}))code(X_{k-1})x_{1,k}\cdots x_{k-1,k}x_{1,k+1}\cdots x_{k,k+1}$. Then

$$\begin{array}{ll} num(lb(x_k)) > num(lb(x_{k+1})), & \text{or} \\ num(lb(x_k)) = num(lb(x_{k+1})) & \text{and} \quad x_{1,k}\cdots x_{k-1,k} \geq x_{1,k+1}\cdots x_{k-1,k+1} \end{array}$$

holds. Similarly,

$$\begin{array}{ll} num(lb(x_k)) > num(lb(x_{k+1})), & \text{or} \\ num(lb(x_k)) = num(lb(x_{k+1})) & \text{and} \quad c_{n-k+2}\cdots c_n \geq c_{n+1}\cdots c_{n+k-1} \end{array}$$

holds for a directed graph, where $n = \frac{k(k-1)}{2}$.

Proof:
Consider a matrix X'_{k+1} obtained by permutation of the k-th and $(k+1)$-th rows and columns of the matrix X_{k+1}.

$$CODE(X'_{k+1})$$

$$= num(X_{k-1})num(lb(x_{k+1}))num(lb(x_k))code(X_{k-1})x_{1,k+1}\cdots x_{k-1,k+1}x_{1,k}\cdots x_{k-1,k}x_{k+1,k}.$$

Accordingly, $CODE(X'_{k+1}) > CODE(X_{k+1})$ when

$$\begin{array}{ll} num(lb(x_k)) < num(lb(x_{k+1})), & \text{or} \\ num(lb(x_k)) = num(lb(x_{k+1})) & \text{and} \quad x_{1,k}\cdots x_{k-1,k} < x_{1,k+1}\cdots x_{k-1,k+1}. \end{array}$$

On the other hand, $G(X'_{k+1}) \equiv G(X_{k+1})$, because the graph represented by an adjacency matrix is invariant over the permutation of rows and columns. This contradicts the assumption that X_{k+1} is a canonical form matrix. The same argument applies to the directed graph. □

Theorem 2.2. The first generator matrix X_{k-1} of a canonical form matrix X_k is also a canonical form matrix.

Proof:
If X_k is a canonical form matrix, but X_{k-1} is not, then the matrices X'_k and its first generator matrix X'_{k-1} meeting the following conditions must exist:

$$\begin{array}{ll} num(X'_{k-1}) > num(X_{k-1}), & \text{or} \\ num(X'_{k-1}) = num(X_{k-1}) & \text{and} \quad code(X'_{k-1}) > code(X_{k-1}), \end{array}$$

where

$$G(X'_k) \equiv G(X_k) \text{ and } G(X'_{k-1}) \equiv G(X_{k-1}).$$

In the latter condition, the labels of the vertices corresponding to the last rows and columns of X'_k and X_k are identical, *i.e.*, $num(lb(x'_k)) = num(lb(x_k))$, because $G(X'_k) \equiv G(X_k)$. Accordingly, the following relation satisfied:

$$\begin{aligned} & CODE(X'_k) = num(X'_{k-1})num(lb(x'_k))code(X'_{k-1})x'_{1,k} \cdots x'_{k-1,k} \\ > \quad & CODE(X_k) = num(X_{k-1})num(lb(x_k))code(X_{k-1})x_{1,k} \cdots x_{k-1,k}. \end{aligned}$$

This contradicts the assumption that X_k is a canonical form matrix. Thus, X_{k-1} is a canonical form matrix. □

Theorem 2.3. Given $F_k = \{$all frequent subgraphs of size $k\}$ and $FX_k = \{X_k | G(X_k) \in F_k$, and X_k is the canonical form$\}$, for a given $X_k \in FX_k$ then let $FY_k(X_k) = \{Y_k | G(Y_k) \in F_k, where Y_k$ shares its first generator matrix X_{k-1} with X_k, and $CODE(X_k) \geq CODE(Y_k)\}$, $FY_k = \cup_{X_k \in FX_k} FY_k(X_k)$, $SZ_{k+1}(X_k) = \{Z_{k+1} | Z_{k+1}$ is derived by the join operation between X_k and $Y_k \in FY_k(X_k)\}$, and $SZ_{k+1} = \cup_{X_k \in FX_k} SZ_{k+1}(X_k)$. Then SZ_{k+1} includes all X_{k+1}s in FX_{k+1}.

Proof:
Each $X_k \in FX_k$ meets Condition 2. The codes of X_k and Y_k for undirected graphs are represented as follows from Condition 1:

$$CODE(X_k) = num(X_{k-1})num(lb(x_k))code(X_{k-1})x_{1,k} \cdots x_{k-1,k},$$

$$CODE(Y_k) = num(X_{k-1})num(lb(y_k))code(X_{k-1})y_{1,k} \cdots y_{k-1,k}.$$

Hence, Condition 3 can be rewritten as follows (#1):

$$\begin{aligned} & num(lb(x_k)) > num(lb(y_k)), \quad \text{or} \\ & num(lb(x_k)) = num(lb(y_k)) \quad \text{and} \quad x_{1,k} \cdots x_{k-1,k} \geq y_{1,k} \cdots y_{k-1,k}. \end{aligned}$$

Also, the label $lb(z_{k+1})$ and the element values $z_{1,k+1} \cdots z_{k-1,k+1}$ of Z_{k+1} corresponds to $lb(y_k)$ and $y_{1,k} \cdots y_{k-1,k}$, respectively. These constraints on $CODE$s are identical with those of Theorem 2.1 when Z_{k+1} is considered as X_{k+1}. The elements $z_{k,k+1}$ and $z_{k+1,k}$ in Z_{k+1} take any values in $num(L_E) = \{num(lb) | \forall lb \in L_E\}$ (#2). $G(X_k)$ and $G(Y_k)$ are frequent. Through the join operations of any X_k and Y_ks satisfying these constraints, all canonical form matrices X_{k+1}s representing frequent subgraphs and having its first generator matrix X_k are derived in $SZ_{k+1}(X_k)$. The corresponding discussion applies to the case of directed graphs. From this observation and the fact that the join operations are applied to all X_ks in FX_k, we conclude that every canonical form matrix X_{k+1} where the first generator matrix is one of X_ks in FX_k is completely derived in SZ_{k+1}. On the other hand, every canonical form matrix X_{k+1} has the first generator matrix X_k which is a canonical form from Theorem 2.1. Since FX_k is complete, every X_{k+1} has the first generator matrix X_k in FX_k. Therefore, SZ_{k+1} includes the complete set of X_{k+1}s in FX_{k+1}. □

After deriving SZ_{k+1}, complete pruning of infrequent Z_{k+1}s and frequency counts of Z_{k+1}s in the objective data are used to derive F_{k+1} and FX_{k+1} as described later. At the level $k = 1$, all complete sets of F_1, FX_1 and FY_1 are derived, since all frequent single vertices and their 1×1 matrix notions are completely enumerated at the initial search. Accordingly, the complete F_k and FX_k are found in every step k from Theorem 2.3.

3. Extension to Mine Various Classes of Structures

The original AGM performs the complete mining of the frequent subgraphs. However, the variation of AGM that we introduce here contains a bias to derive only the frequent induced subgraphs [10, 12]. An induced subgraph of a graph G has a subset of the vertices of G and the same edges between pairs of vertices as in G. To limit the search of the frequent subgraphs within this class, the following bias has been applied in the past work. When counting the frequency of each candidate frequent subgraph, the AGM algorithm checks whether it is contained in each graph in a database as an induced subgraph.

In the following subsections, we propose further biases that allow for the graph mining of various classes based on the AGM framework as depicted in Figure 11. We call this framework *B-AGM* (Biased-Apriori based Graph Mining). A bias for a specific class of the graph structure consists of the dedicated definitions of the canonical form and the join operation. By choosing an appropriate bias on the platform of the AGM framework, the complete mining for the frequent subgraphs of the objective class we are seeking for is defined.

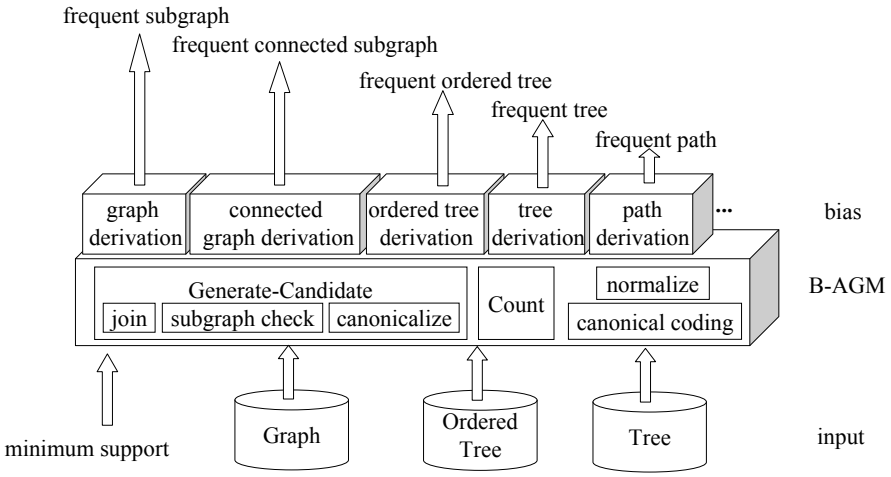


Figure 11. B-AGM Framework.

3.1. Bias for Connected Subgraph Derivation

For calculation efficiency, the B-AGM algorithm with this bias mines all of the frequent connected subgraphs and some semi-connected subgraphs which consist of a connected subgraph and an isolated vertex. The semi-connected graphs are not added to the output of the frequent subgraphs[2].

Canonical Form

The definition of canonical form is altered from the original. Given the upper left $i \times i$ submatrix of the adjacency matrix X_k as X_i ($1 \leq i \leq k$), the following set $\Gamma(G)$ of adjacency matrices representing an identical graph G is defined.

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

[2]B-AGM with this bias is available from http://www.alphaworks.ibm.com/tech/fsm.

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

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

Join Operation

The original Conditions 1 and 2 are retained, and the following Conditions 3 and 4 are introduced.

Condition 3. $G(X_k)$ is a connected graph.

Condition 4. $G(Y_k)$ is not a connected graph, otherwise $CODE(X_k) \geq CODE(Y_k)$.

Each frequent subgraph of $G(X_k)$ has a flag to represent whether or not it is connected. The flag is determined from the flags of its first and second generator matrices. If both flags are connected, then the flag of a graph which is made from the first and second matrices becomes connected.

If the second generator matrix corresponds to an disconnected graph, $CODE(X_k) \geq CODE(Y_k)$ does not have to be satisfied to join the adjacency matrices. For example, let $num(A)$ and $num(B)$ be 1 and 2, respectively, in Figure 12. The canonical form of $G(Z_3)$ in Figure 12 is $CODE(Z_3) = 212101$, and it is generated by joining two adjacency matrices, such as $CODE(X_2) = 211$ and $CODE(Y_2) = 220$. Therefore, the condition that the second generator matrix corresponds to an disconnected graph is needed[3].

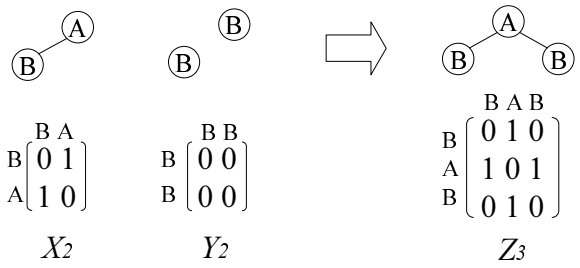


Figure 12. Example of Join Operation Bias for Connected Subgraph Derivation.

The completeness of the search by this join operation is proven. However, due to the space limitation, only the points to be altered in the aforementioned proof for the standard join operation are explained. Theorem 2.1 is altered as follows.

Theorem 3.1. Given a canonical form matrix X_{k+1} of an undirected graph and its $CODE(X_{k+1}) = num(X_{k-1})num(lb(x_k))num(lb(x_{k+1}))code(X_{k-1})x_{1,k} \cdots x_{k-1,k}x_{1,k+1} \cdots x_{k,k+1}$. For X_{k+1} representing a connected graph,

1. $num(lb(x_k)) > num(lb(x_{k+1}))$ or
2. $num(lb(x_k)) = num(lb(x_{k+1}))$ and $x_{1,k} \cdots x_{k-1,k} \geq x_{1,k+1} \cdots x_{k-1,k+1}$, or
3. $num(lb(x_k)) < num(lb(x_{k+1}))$, $num(lb(x_1)) > num(lb(x_{k+1}))$, $x_{1,k+1} = \cdots = x_{k-1,k+1} = 0$, and $x_{k,k+1} \neq 0$,

[3] In the case of the conventional AGM algorithm which finds not only connected subgraphs but also disconnected subgraph, the canonical form of $G(Z_3)$ in Figure 12 becomes $CODE(Z_3) = 221011$.

and for X_{k+1} representing a disconnected graph,

$$x_{i,k+1} = 0 \text{ for all } i = 1, \cdots, k$$

holds.

A similar alternation is made for directed graphs. This theorem is similarly proven by the reduction to absurdity. Theorem 2.2 holds, since X_{k-1} of X_k represents a connected graph, and the identical definition of the canonical form applies to the adjacency matrix of a connected graph. Theorem 2.3 also holds since Condition 3 and 4 alter the constraints of (#1) in the manner consistent with Theorem 3.1.

3.2. Bias for Ordered Subtree Derivation

The B-AGM algorithm with this bias efficiently mines all frequent ordered subtrees included in a forest in the same way as the connected subgraph derivation.

Canonical Form

When the total order of the rows and columns of the adjacency matrix X_k representing an ordered tree $G(X_k)$ matches the preorder numbers assigned to the vertices in the ordered tree, the matrix is defined as the canonical form of $G(X_k)$ as shown in Figure 13. More strictly, let the set of the total order numbers of the columns and rows of X_k be $I = \{i | i = 1 \cdots k\}$. When $G(X_k)$ is an ordered tree, let the set of the preorder numbers assigned to the vertices of $G(X_k)$ be $J = \{j | j = 1 \cdots k\}$. When $G(X_k)$ consists of an ordered tree and an isolated vertex, let the set of the preorder numbers $1 \cdots k-1$ assigned to the vertices of the ordered tree and the last k assigned to the isolated vertex in $G(X_k)$ be $J = \{j | j = 1 \cdots k\}$. Then, X_k is the canonical form adjacency matrix of $G(X_k)$ if an identity mapping $j = ID(i)$ between I and J exists. Under this identity mapping, $CODE(X_k)$ is represented as

$$CODE(X_k) = num(lb(x_1)) \cdots num(lb(x_k)) code(X_k).$$

Since the identity mapping ID is unique, the total order numbers assigned to the vertices in $G(X_k)$ uniquely specifies X_k. Under this definition, the AGM algorithm can search only canonical form matrices. This greatly enhances the search efficiency [27, 2].

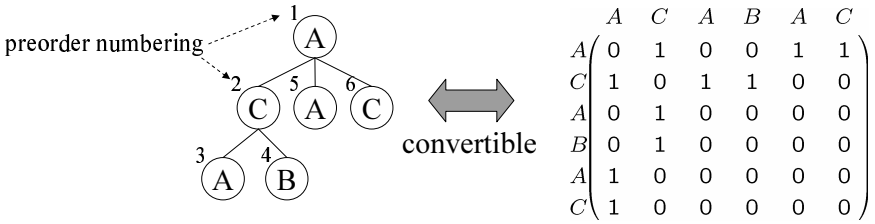


Figure 13. $CODE$ of Ordered Tree.

Join Operation

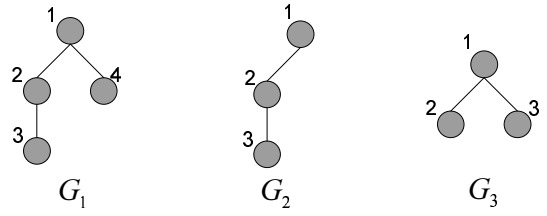
The original Conditions 1 and 2 are retained, and the following Conditions 3 and 4 are introduced.

Condition 3. $G(Y_k)$ is disconnected, otherwise $code(X_k) \leq code(Y_k)$.

Condition 4. $G(X_k)$ is a connected graph.

The Condition 3 related to $code$ for this bias is different from the conditions for the standard and the

connected subgraph derivation in terms of the part of the code and the direction of the inequality used in the condition. The difference comes from the definition of the canonical form for the ordered tree. The rows and the columns of the adjacency matrix are ordered by following the left and depth first preorder numbering in the ordered tree. Accordingly, the matrix having the last row and the last column representing a vertex located at a deeper level has a small $code$ under a given first generator matrix X_{k-1}. For example, X_3 of a graph G_1 having the third vertex at depth 3 has a small $code$ than Y_3 of G_2 having the third vertex at depth 2. Because the graph G_3 resulting from the join operation of two graphs having their final vertices at different depths is uniquely determined independent from the labels of the vertices, redundants joins are avoided by introducing the condition $code(X_k) < code(Y_k)$. When X_k and Y_k represent graphs whose last vertices are at the same depth, *i.e.*, $code(X_k) = code(Y_k)$, the join operation for the graphs result in two candidates depending on the order of the join of X_k and Y_k as depicted in Figure 15. Since these candidates represent different ordered trees, the join operations in the both orders should be admitted when $code(X_k) = code(Y_k)$. Thus, $code(X_k) \leq code(Y_k)$ in the Condition 3 should be applied to the case where $G(X_k)$ and $G(Y_k)$ are both connected graphs. When $G(Y_k)$ is not connected, the analysis of Section 3.1 is used.



$$Z_4 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad X_3 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad Y_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$code(Z_4) = 101100 \qquad code(X_3) = 101 \qquad code(Y_3) = 110$$

Figure 14. Examples of Join Operation for Ordered Subtree having Different Depth.

The completeness for the search by this join operation is now proven. Theorem 2.1 is altered as follows.

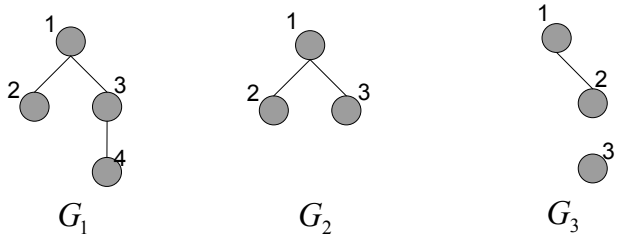
Theorem 3.2. Given a canonical form matrix X_{k+1} of an ordered tree or an ordered tree with an isolated vertex and its

$$CODE(X_{k+1}) = num(X_{k-1})num(lb(x_k))num(lb(x_{k+1}))code(X_{k-1})x_{1,k}\cdots x_{k-1,k}x_{1,k+1}\cdots x_{k,k+1}.$$

Then,

1. $x_{1,k}\cdots x_{k-1,k} \leq x_{1,k+1}\cdots x_{k-1,k+1}$, or
2. $x_{1,k+1} = \cdots = x_{k-1,k+1} = 0$ and $x_{k,k+1} \neq 0$

holds.



$$Z_4 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad X_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad Y_3 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$code(Z_4) = 110001 \qquad code(X_3) = 110 \qquad code(Y_3) = 100$$

Figure 15. Examples of Join Operation for Ordered Subtree having Identical Depth.

Theorem 2.2 is also easily proven by the reduction to absurdity. Theorem 2.3 also holds since Condition 3 and 4 alter the constraints of (#1) in a manner consistent with Theorem 3.2.

3.3. Bias for Path Derivation

This bias derives frequent subgraphs included as paths that have no loops or branches. The definition of a canonical form is identical with that of the connected graph.

Join Operation

The original Conditions 1 and 2 are retained, and the following Conditions 3, 4 and 5 are introduced.

$\underline{Condition\ 3.}$ $G(X_k)$ is a connected graph.

$\underline{Condition\ 4.}$ $G(Y_k)$ is not a connected graph, otherwise $CODE(X_k) \geq CODE(Y_k)$.

$\underline{Condition\ 5.}$ When $G(Y_k)$ is connected, $z_{k+1,k}$ and $z_{k,k+1}$ of Z_{k+1} are set to zero to prevent making a pattern which has loops or branches.

The completeness of the search by this join operation is proven in almost identical manner with Section 3.1 except the part (#2) in the proof of Theorem 2.3. $z_{k,k+1}$ and $z_{k+1,k}$ in Z_{k+1} are always zero to avoid the generation of loops and branches. These constraints do not break the completeness of the search as far as only frequent paths are searched. Hence, the search by this join operation is complete.

4. Experiments

An IBM PC 300PL with Windows 2000 was used for the experiments. The test machine has a Pentium III-667 MHz CPU and 192MB of main memory installed.

4.1. Mining Connected Subgraphs

Molecular structure data of carcinogenic compounds was analyzed. This data was provided from a Predictive Toxicology Evaluation database [23], and contains information on 340 chemical compounds.

There are 24 types of atoms making up these 340 chemical compounds. Since the atoms have different states, the total number of atom types is 66. There are four types of atomic bonds corresponding to edges in a graph. The molecular compounds contained an average of 27 atoms, with the largest compound having 214.

Figure 16 shows the results for the computation times for various minimum support values. It includes the results of B-AGM for both connected subgraphs and connected induced subgraphs and FSG [18][4] and gSpan [24][5] for connected subgraphs. When the minimum support decreases, the computation time increases because the number of discovered patterns increases. The performance of B-AGM is better than FSG and is comparable with gSpan for general connected subgraph derivation.

Figure 17 shows two examples derived in the connected induced subgraph derivation. The subgraph of (a) is contained in 6 chemical compounds with carcinogenic activity and 19 compounds without such activity. In contrast, the graph of (b) is contained in 17 compounds with carcinogenic activity and 4 inactive compounds. The first molecular substructure does not exhibit significant activity, whereas the second one exhibits quite high activity.

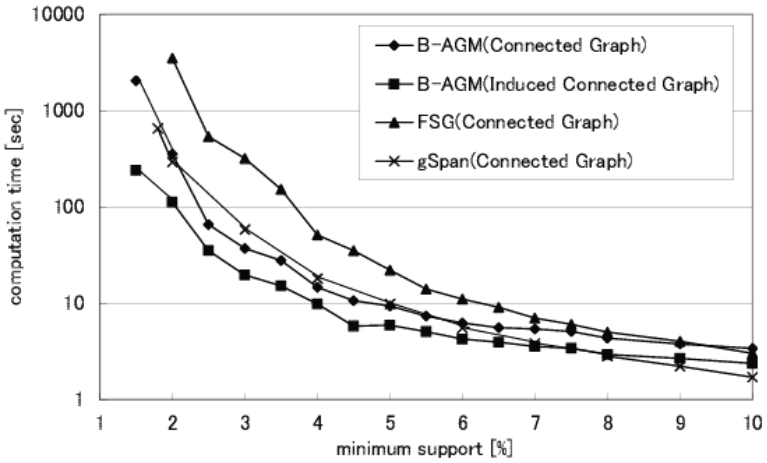
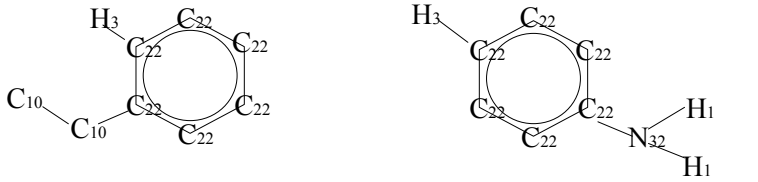


Figure 16. Minimum Support vs. Computation Time.



support=7.3%
(positive = 6, negative = 19)

(a) Example 1

support=6.2%
(positive = 17, negative = 4)

(b) Example 2

Figure 17. Discovered Frequent Connected Subgraphs.

[4]The experiments were done on a Linux PC with dual AMD Athlon MP 1800+ and 2GB main memory.
[5]The experiments were done on a Linux PC with a Pentium III 500 MHz processor and 448MB main memory.

4.2. Mining Ordered Subtrees

4.2.1. Web browsing data

Zaki presented experimental results of substructure discovery from a set of logs files over one month at the RPI computer science department Web site [27]. After the preprocessing, the dataset had 595,691 user browsing subtrees with 13,361 unique labels (Web pages). We used the same data provided by Zaki in our experiment and recalculated with the same machine to compare our approach with TreeMiner. TreeMiner can find frequent patterns embedded in a dataset of ordered trees. It is defined that pattern P discovered by TreeMiner is embedded in tree data D if and only if two vertices in a branch in P are on the same path from the root to a leaf in D. For example in Figure 18, an ordered tree in the right is embedded in the data. We used the same definition in the comparison of the results to obtain equivalent results. The function to count the frequency is changed in B-AGM to adjust to the definition of frequent patterns of TreeMiner. Figure 19 shows the computation times for various minimum support values. As shown in Figure 19, B-AGM is comparable to TreeMiner.

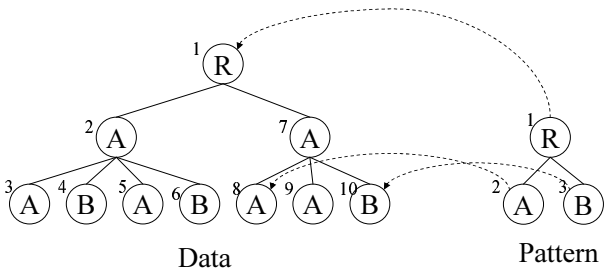


Figure 18. Examples (1) of Ordered Tree Data and a Pattern.

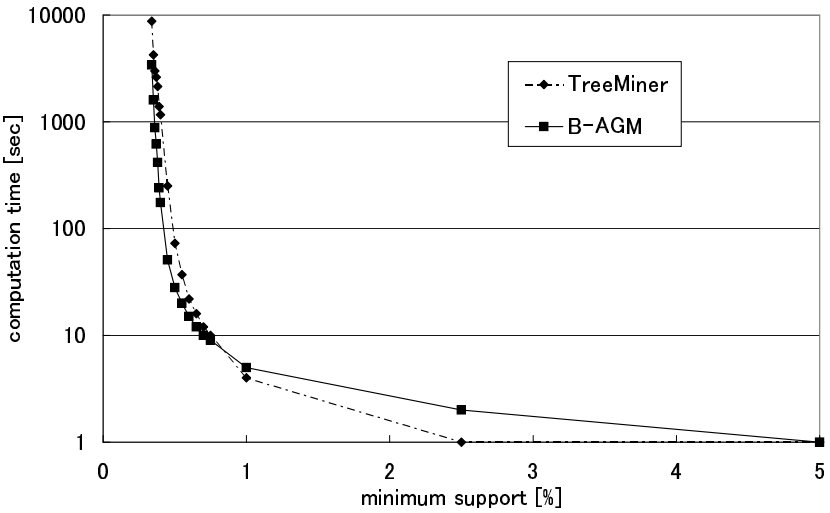


Figure 19. Minimum Support vs. Computation Time.

4.2.2. Semi-structured data

Asai et al. presented experimental results of substructure discovery from a collection of Web pages gathered from the Internet [2]. The data was collected from the online bibliographic archive Citeseer, and was parsed to create DOM trees. After the preprocessing, the tree for the data had 196,247 vertices with 7,125 unique labels (tags). We used the same data provided by Asai in our experiment. Since FREQT can find frequent patterns embedded in *one* ordered tree, the definition of support of FREQT is slightly different from that defined in Section 2. Given an ordered tree D, the support $sup(P)$ of a pattern P is defined as a ratio of the number of occurrences of the root of P to the total number of vertices in D. For example, given the ordered data tree D in Figure 20, the $sup(P) = 2/10$, because the root of the pattern P occurs at nodes 2 and 7 in the tree data that contains a total of 10 nodes. We used the same support definition as in FREQT in our comparison of the results to obtain equivalent results.

Figure 21 shows the results of computation times and the numbers of derived frequent patterns for various minimum support values. FREQT was implemented in JAVA and our B-AGM is implemented in C++. This experiment was done in the same computational environment. As shown in Figure 21, B-AGM is comparable to or faster than FREQT.

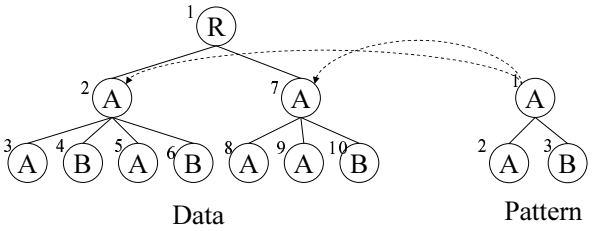


Figure 20. Examples (2) of Ordered Tree Data and a Pattern.

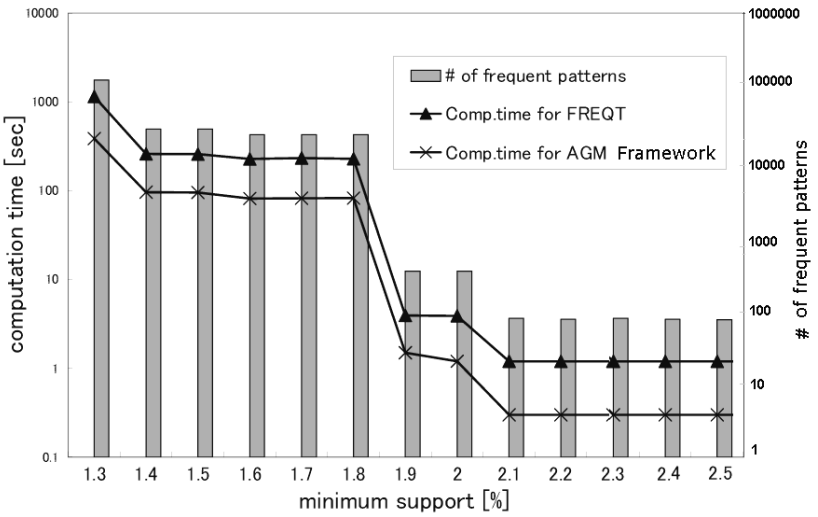


Figure 21. Minimum Support vs. Computation Time and the Number of Derived Frequent Patterns.

4.3. Mining Path Patterns

The Developmental Therapeutics Program's AIDS Antiviral Screening has checked tens of thousands of compounds for evidence of anti-HIV activity [9]. Available data are screening results and chemical structural data on compounds that are not covered by any confidentiality agreements. The dataset contains the structures of 42,687 chemical compounds and the related screening data. The data is categorized into one of the three classes: active (CA), moderately active (CM) and inactive (CI). There are 422 CA compounds, 1,081 CM compounds, and 41,184 CI compounds in the dataset. Kramer et al. applied MolFea to 41,768 of these compounds to discover the characteristic path patterns (called fragments) [16][6]. MolFea can mine path patterns with a minimum support for one class in the data and a maximum support for another class in the data based on the search algorithm in the version space. For the first task, MolFea mined fragments that were contained in CA compounds more than 13 times which corresponds to the minimum support on the CA dataset is 3% and in CI less than 516 times corresponding to the maximum support of 1.282%. The total computation time was about five hours and twenty minutes, and more than 1,600 fragments were discovered. For the second task, it also mined fragments that were contained in more than 13 CA compounds and in less than 8 CM compounds (0.75%). The total computation time was about 34 minutes, and more than 680 fragments were discovered in this case.

We compared the performance of B-AGM to derive paths with MolFea on this HIV dataset. Because our framework cannot apply the minimum support and the maximum support simultaneously, the mining equivalent to that of MolFea is conducted in two steps. First, our approach finds all frequent subgraphs having supports greater than or equal to the minimum support in the data having a class. Second, the patterns less than or equal to the maximum support for another class are deleted. For the first task, we set the minimum support for the CA dataset and the maximum support for CI to 3% (13 compounds) and 1.265% (521 compounds), respectively, which were decided on the same criteria as used for MolFea. The B-AGM algorithm took around 12 minutes to derive a set of path patterns which is almost identical with that MolFea found. Under the identical conditions with the second task of MolFea, our approach took around one minute, and identical patterns were obtained. Our algorithm mines all of the fragments under the minimum and maximum support constraints more efficiently than MolFea.

4.4. Usefulness of Discovered Subgraph Patterns

To evaluate the ability of the B-AGM algorithm to discover characteristic patterns in wider classes not limited to paths, we also mined characteristic connected subgraphs by applying B-AGM with a connected subgraph derivation bias to CA and CI compounds of the HIV dataset. Figure 22 shows the results for the computation times and the numbers of discovered patterns for various minimum support and maximum support values based on the same criteria used for MolFea. When the minimum support decreases, the computation time increases because the number of discovered patterns increases. Most of the computation time was required to find all frequent connected subgraphs whose supports are greater than or equal to the specified minimum support threshold. For example, when the minimum support for the CA dataset and the maximum support for CI were set to 16 compounds and 699 compounds, respectively, the B-AGM algorithm took 3 hours to find all of the frequent patterns from the CA compounds and thirty minutes to delete the patterns which do not have the maximum support for CI compounds. Figure 23 is one of the discovered patterns which has the maximum chi-squared value in a test introduced

[6]The experiments were done on a Linux PC with a Pentium III 600 MHz processor

by Brin [4]. It is contained in 64 CA compounds including azidothymidine depicted in Figure 24 and in 16 CI compounds.

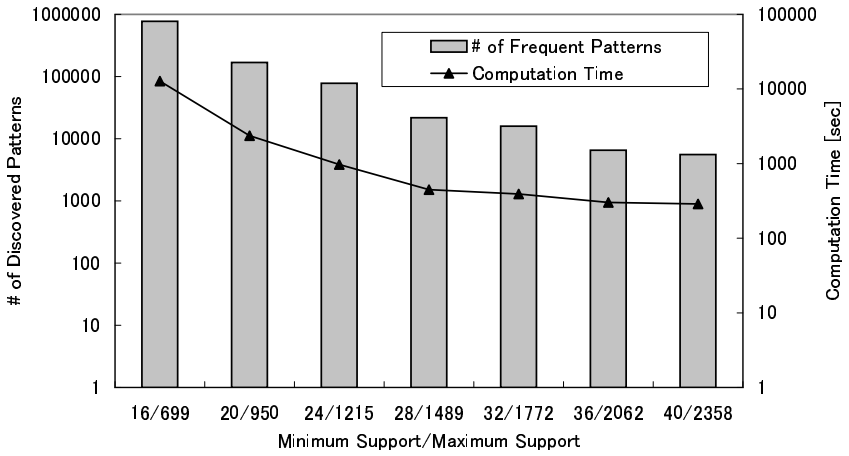


Figure 22. Computation Time and the Number of Discovered Patterns.

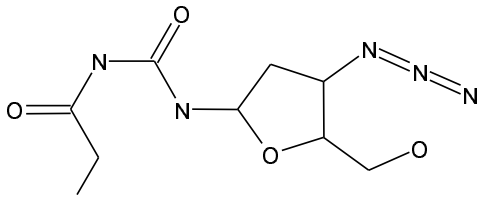


Figure 23. A Discovered Pattern having a Maximal Chi-squared Value.

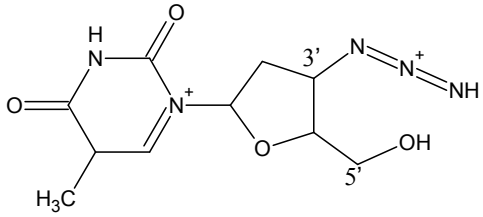


Figure 24. Molecular Structure of Azidothymidine (AZT).

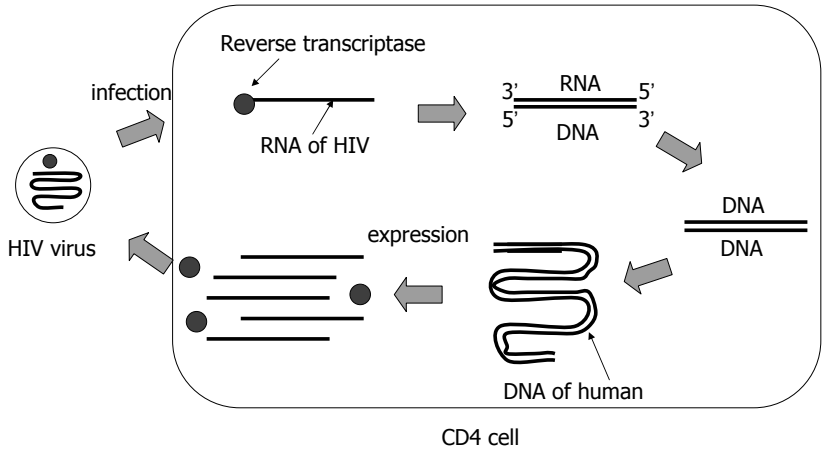


Figure 25. Mechanism of HIV Infection (1).

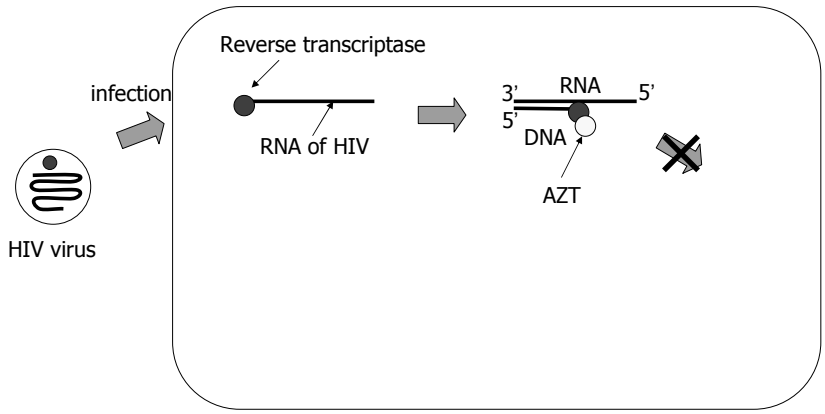


Figure 26. Mechanism of HIV Infection (2).

Figures 25 and 26 show the mechanism of HIV Infection. When viruses invade from the outside, the immune system works to eliminate them in the human body. The CD4 cells play a central role in the immune system. HIV invades the CD4 cells, and destroys them. After infection by HIV, within the CD4 cell, the RNA of HIV is replicated into DNA by reverse transcriptase, and is included in the host chromosome. After the inclusion, the DNA is activated to produce new HIV as shown in Figure 25. Medical treatment is difficult because of HIV's capability of hiding in the host chromosome, but since the reverse transcriptase of HIV is unnecessary for the host cell, the reverse transcriptase has become the target in the development of anti-HIV medicines [1]. Azidothymidine (AZT) depicted in Figure 24 is known to be an anti-HIV medicine. The reasons why AZT can inhibit the growth of the DNA chain are that

- the structure of ATZ is similar to Thymine as depicted in Figure 27, and

- AZT does not have a hydroxyl group (-OH) at its 3' end as shown in Figure 24.

The reverse transcriptase reads the sequence of the HIV's RNA and transcribes it into a DNA sequence consisting of Adenine, Thymine, Guanine, and Cytosine. Since AZT is similar to Thymine, it binds to the reverse transcriptase and is added to the DNA chain during extension as shown in Figure 26. In addition, the RNA of HIV is replicated into the DNA by the reverse transcriptase from the 5' end to the 3' end indicated in Figure 27. Since there is no hydroxyl group at the 3' end of AZT, it stops the further production of the DNA. B-AGM was able to discover the significant subgraph pattern shown in Figure 23, which is included in many CA compounds but fewer CI compounds without using this background knowledge. The compounds including this substructure are promising candidates for developing new anti-HIV medicines.

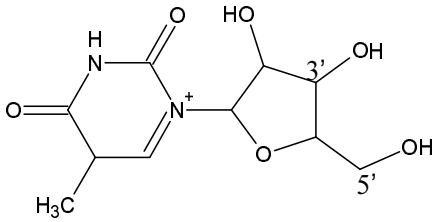


Figure 27. Molecular Structure of Thymine.

5. Future Extension

Since the bias of B-AGM has high generality, it can be applied to mine frequent patterns from a set of unordered trees by introducing the following bias for the unordered subtree derivation. We plan to expand our B-AGM by adding the bias for the unordered subtree derivation, and to compare with other existing methods [22, 3].

Canonical Form

From an unordered tree G, we can generate many order trees G' by reordering children of one node in the unordered tree. Let an adjacency matrix of the ordered tree G' be X_k. Let the set of the total order numbers of the columns and rows of X_k be $I = \{i | i = 1 \cdots k\}$. When $G(X_k)$ is an ordered tree without an isolated vertex, let the set of the preorder numbers of assigned to the vertices of $G(X_k)$ be $J = \{j | j = 1 \cdots k\}$. When $G(X_k)$ consists of an ordered tree and an isolated vertex, let the set of the preorder numbers $1 \cdots k-1$ assigned to the vertices of the ordered tree and the last k assigned to the isolated vertex in $G(X_k)$ be $J = \{j | j = 1 \cdots k\}$. Let $\Gamma(G)$ be a set of the adjacency matrices representing the identical unordered tree G. The adjacency matrix C_k whose $CODE$ is the smallest in $\Gamma(G)$ is called the canonical form.

$$C_k \quad s.t. \quad CODE(C_k) = \min_{X_k \in \Gamma(G)} CODE(X_k).^{7}$$

[7]The canonical form must be the minimum $CODE$ among the adjacency matrices representing an identical graph according to the definition of the condition 3.

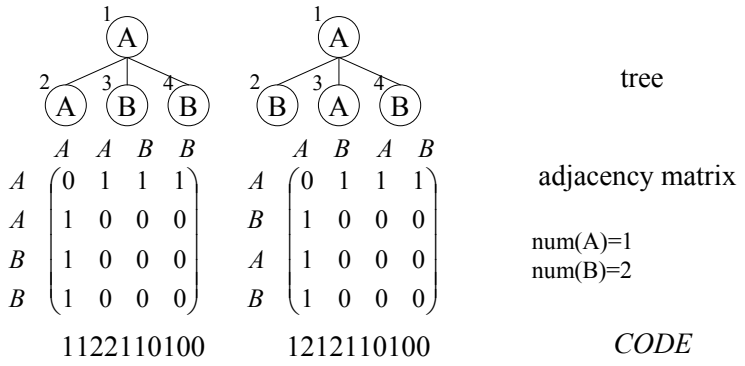


Figure 28. Tree Representation.

For example, the trees in Figure 28 are not isomorphic as ordered trees, but they are isomorphic as unordered trees. Since the $CODE$ of the adjacency matrix corresponding to the left tree is at a minimum for the matrices that represent the same tree, the matrix is the canonical form. The conditions of the join operation for this bias are the same as those for the ordered tree bias. The completeness of the search by this join operation is proven similarly to the bias for the ordered subtree derivation.

6. Discussion and Related Work

Heuristic-based approaches, *e.g.*, SUBDUE [5] and GBI [26, 20] have been introduced to alleviate the complexity issue. SUBDUE derives characteristic patterns based on Minimum Description Length of subgraphs. The latest version of GBI derives characteristic patterns in a dataset by chunking pair of connected vertices having various types of high scores [19]. The advantage of these methods is their ability to search for typical patterns under various criteria in a rapid manner. However, their greedy search may miss some important patterns.

In contrast, MolFea [7], TreeMiner [27], and FREQT [2] use complete search strategies similarly to the AGM algorithm. They require $O(k)$ memory to store the trees or paths, where k is the number of vertices in the tree. Although our approach requires $O(k^2)$ memory for the storage, other methods which focused on one particular class of graph cannot be applied to mine more complex substructures. Our proposed method can conduct a complete search of various classes of frequent subgraphs. As shown in Section 4, the AGM algorithm with a bias for the ordered subtree or path derivation can derive the complete result within a practical time period for each class of problem, where the relative performance is better than or comparable to the other approaches.

In addition, the B-AGM algorithm with a bias for the connected subgraph derivation can efficiently discover all of the frequent subgraph patterns, with relative performance better than FSG [17] and comparable to gSpan [24]. The first reason for the efficiency of B-AGM is that it can quickly generate adjacency matrices of candidate frequent subgraphs through the join operation finding a common substructure shared by two graph patterns under the adjacency matrix representation. If a well-defined data structure such as an adjacency matrix or the DFS canonical code of gSpan is not used, exponential time

is needed to find the common substructures from two graphs. In fact, B-AGM can find one of the common structures in $O(k^2)$ from two adjacency matrices of size k. Although gSpan does not generate any candidates, it can find one of common substructures in $O(v+e)$ if it generates them, where v and e are the numbers of vertices and edges in a graph pattern, respectively. In addition, B-AGM and gSpan can find it in $O(1)$ by using the implementation used in [12, 24]. On the other hand, the first version of FSG needs exponential time to find the common subgraphs [17], because it has to do subgraph isomorphism matching between a pattern with e edges and subgraphs with $e-1$ edges of another pattern.

The second reason is frequency counting. The B-AGM algorithm stores a set of the correspondences of the vertices between a subgraph pattern and each graph in a database, as mentioned in Section 2.2.4. Figure 29 shows the computation times under various minimum support thresholds for cases where the B-AGM algorithm stores the correspondences of the vertices and does not store the relations[8]. The latter counting method is referred to Naive Counting in Figure 29. The computation times of the former counting method are smaller than those of the latter counting method. Accordingly, the B-AGM algorithm can quickly count support values. A drawback of the B-AGM algorithm is that it requires memory to store the relations as shown in Figure 30 while no memory is required in the latter counting method. However, the required memory space for moderate amounts of data remains small. These features of the B-AGM algorithm result in high efficiency since the load of the subgraph isomorphism matching required to count the frequency of each pattern is quite small. FSG needs more computation time for the subgraph isomorphism matching required in the join operation, because of the code representation of graphs without an algebraic background. The gSpan algorithm uses an efficient depth-first search based on DFS canonical codes for graphs. Although it can also be extended to mine directed subgraphs, subtrees, and paths, it is not easily extended to mine general subgraphs including unconnected subgraphs [25]. Table 1 summarizes the combinations of the given datasets, the objective classes of substructures, and the applicable approaches. This shows that our B-AGM can be applied to various problems.

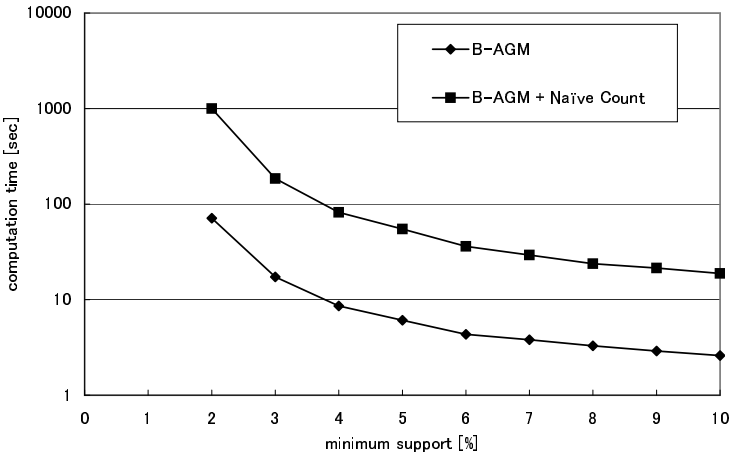


Figure 29. Computation Times to Count Support.

[8]The B-AGM algorithm with a bias for the connected subgraph derivation was used in Figures 29, 30. The derived subgraphs are induced subgraphs of each graph data.

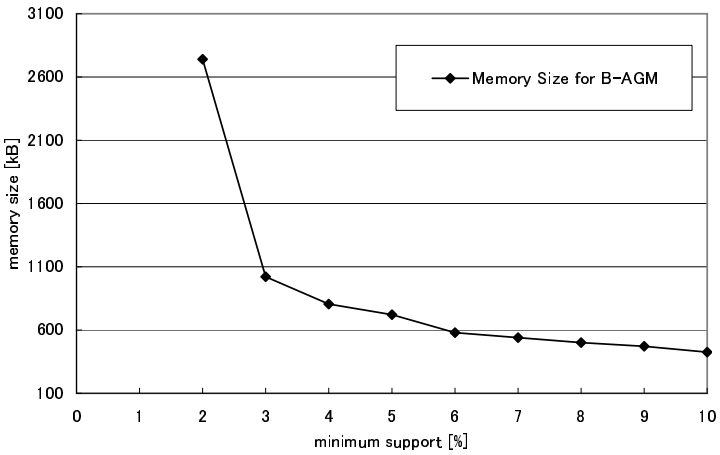


Figure 30. Required Memory Space to Store Correspondences of Vertices.

Table 1. Summary of Graph Mining Methods.

Dataset	Substructure Class	Approach
Graph	General subgraph	AGM, B-AGM
Graph	Connected subgraph	FSG, B-AGM, gSpan
Tree	Tree	[22], UNOT [3], B-AGM, gSpan
Ordered tree	Ordered tree	FREQT, TreeMiner,B-AGM, gSpan
Graph	Path	MolFea, B-AGM, gSpan

WARMR [6] and FARMER [21] for the complete search of the structures that belong to a more general class than the graph have been proposed. They tried to mine characteristic patterns in the form of first order predicates using PROGOL, which is a system used in inductive logic programming (ILP). They combined ILP principles with a levelwise search technique to improve the search efficiency. However, their results include some predicates having different forms but equivalent in the sense of θ-subsumption, and the class of the substructures to be searched was limited to connected structures.

7. Conclusion

We proposed a generic framework for the data mining of graph structures. By introducing additional biases, our approach can easily derive various types of frequent substructures. We evaluated its performance in terms of the required computation time for some real world datasets. The wide coverage for various problem classes and the computational efficiency were confirmed and our method appears to outperform or be comparable with other approaches.

Acknowledgement

We would like to thank Prof. Luc De Raedt of the University of Freiburg, Prof. Stefan Kramer of the Technical University of Munich, Prof. Mohammed Zaki of Rensselaer Polytechnic Institute, Prof. Takashi Okada of Kwansei University, Prof. Hiroki Arimura and Dr. Tatsuya Asai of Kyushu University, and Shannon Jacobs of IBM Japan HRS for their help and advice.

References

[1] Alberts, B., Bray, D., Johnson, A., Lewis, J., Raff, M., Roberts, M., and Walter, P.: Essential Cell Biology: An Introduction to the Molecular Biology of the Cell, *Garland Science Publishing*, 1998.

[2] Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., and Arikawa, S.: Efficient Substructure Discovery from Large Semi-Structured Data, *Proc. of the 2nd SIAM International Conference on Data Mining*, pp. 158–174, 2002.

[3] Asai, T., Arimura, H., Uno, U., and Nakano, S.: Discovering Frequent Substructures in Large Unordered Trees, *Proc. of the 6th International Conference on Discovery Science*, pp. 47–61. 2003.

[4] Brin, S., Motwani, R., and Silverstein, C: Beyond market baskets: Generalizing association rules to correlations, *Proc. of the SIGMOD International Conference on Management of Data*, pp. 265–276. 1997.

[5] Cook, D. and Holder, L.: Substructure Discovery Using Minimum Description Length and Background Knowledge, *Journal of Artificial Intelligence Research*, Vol. 1, pp. 231–255, 1994.

[6] Dehaspe, L., Toivonen, H., and King, R.: Finding Frequent Substructures in Chemical Compounds, *Proc. of the 4th International Conference on Knowledge Discovery and Data Mining*, pp. 30–36, 1998.

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

[8] Garey, M. and Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979.

[9] AIDS Antiviral Screen, http://dtp.nci.nih.gov/docs/aids/aids_data.html

[10] Inokuchi, A., Washio, T., and Motoda, H.: An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data, *Proc. of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 13–23, 2000.

[11] Inokuchi, A., Washio, T., Nishimura, Y., and Motoda, H.: A Fast Algorithm for Mining Frequent Connected Graph, *IBM Research Report*, RT0448, 2002.

[12] Inokuchi, A., Washio, T., and Motoda, H.: Complete Mining of Frequent Patterns from Graphs: Mining Graph Data, *Machine Learning*, Vol.50, pp. 321-354, 2003.

[13] Inokuchi, A., Washio, T., and Motoda, H.: A General Framework for Mining Frequent Patterns in Structures, *IBM Research Report*, RT0513, 2003.

[14] Kilpelainen, P. and Mannila, H.: Ordered and Unordered Tree Inclusion, *SIAM Journal on Computing*, Vol.24, pp. 340–356, 1995.

[15] Kramer, S., and De Raedt, L.: Feature Construction with Version Space for Biochemical Applications, *Proc. of the 18th International Conference on Machine Learning*, pp. 258–265, 2001.

[16] Kramer, S., De Raedt, L., and Helma, C.: Molecular Feature Mining in HIV data, *Proc. of the 7th International Conference on Knowledge Discovery and Data Mining*, pp. 136–143, 2001.

[17] Kuramochi, M. and Karypis, G.: Frequent Subgraph Discovery, *Proc. of the 1st International Conference on Data Mining*, pp. 313–320, 2001.

[18] Kuramochi, M. and Karypis, G.: An Efficient Algorithm for Discovering Frequent Subgraphs, *Technical Report*, 02-026, 2002.

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

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

[21] Nijiissen, S. and Kok, J.: Faster Association Rules for Multiple Relations, *Proc. of the 17th International Joint Conference on Artificial Intelligence*, pp. 891–896, 2001.

[22] Nijiissen, S. and Kok, J.: Efficient Discovery of Frequent Unordered Trees *Proc. of the 1st First International Workshop on Mining Graphs, Trees and Sequences* , 2003.

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

[24] Yan, X. and Han, J.: gSpan: Graph-Based Substructure Pattern Mining, *Proc. of the 2nd International Conference on Data Mining*, pp. 721–724, 2002.

[25] Yan, X. and Han, J.: CloseGraph: Mining Closed Frequent Graph Patterns, *Proc. of the 9th International Conference on Knowledge Discovery and Data Mining*, pp. 527–532, 2003.

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

[27] Zaki, M.: Efficiently Mining Frequent Trees in a Forest, *Proc. of the 8th International Conference on Knowledge Discovery and Data Mining*, pp. 71–80, 2002.