Using a Hash-based Method for Apriori-based Graph Mining

Phu Chien Nguyen, Takashi Washio, Kouzou Ohara and Hiroshi Motoda

The Institute of Scientific and Industrial Research, Osaka University
8-1 Mihogaoka, Ibaraki, Osaka, 567-0047, Japan
Phone: +81-6-6879-8541 Fax: +81-6-6879-8544

Abstract. The problem of discovering frequent subgraphs of graph data can be solved by constructing a candidate set of subgraphs first, and then, identifying within this candidate set those subgraphs that meet the frequent subgraph requirement. In Apriori-based graph mining, to determine candidate subgraphs from a huge number of generated adjacency matrices is usually the dominating factor for the overall graph mining performance since it requires to perform many graph isomorphism tests. To address this issue, we develop an effective algorithm for the candidate set generation. It is a hash-based algorithm and was confirmed effective through experiments on both real-world and synthetic graph data.

1 Introduction

Discovering frequent patterns of graph data, *i.e.*, frequent subgraph mining or simply graph mining, has attracted much research interest in recent years because of its broad application areas such as cheminformatics and bioinformatics [2, 7, 9, 11]. The kernel of frequent subgraph mining is subgraph isomorphism test, which is known to be in NP-complete. The earliest studies to find subgraph patterns characterized by some measures from massive graph data, SUBDUE [3] and GBI [13], use greedy search to avoid high complexity of the subgraph isomorphism problem, resulting in an incomplete set of characteristic subgraphs.

Recent graph mining algorithms can mine a complete set of frequent subgraphs given a minimum support threshold. They can be roughly classified into two categories. Algorithms in the first category employ the same level-by-level expansion adopted in Apriori [1] to enumerate frequent subgraphs. Representative algorithms in this category include AGM [7] and FSG [9]. AGM finds all frequent induced subgraphs, which will be explained later, with a vertex-growth strategy. FSG, on the other hand, finds all frequent connected subgraphs based on an edge-growth strategy and was claimed to run faster than AGM. Algorithms in the second category use a depth-first search for finding candidate frequent subgraphs. A typical algorithm in this category is gSpan [12], which was reported to outperform both AGM and FSG in terms of computation time. However, a variant of AGM focusing on mining frequent connected subgraphs, AcGM [6], was claimed to be superior to FSG and comparable with gSpan.

In the context of mining frequent itemsets by Apriori, the heuristic to construct the candidate set of frequent itemsets is crucial to the performance of frequent pattern mining algorithms [10]. Clearly, in order to be efficient, the heuristic should generate candidates with high likelihood of being frequent itemsets because for each candidate, we need to count its occurrences in all transactions. The smaller the candidate set, the less the processing cost required to enumerate frequent itemsets. This is even more decisive in the case of graph mining since to count support as well as to check the downward closure property for each candidate, we need to perform many graph and subgraph isomorphism tests.

To address this issue in mining frequent itemsets, an effective hash-based method called DHP [10] was introduced to Apriori. This algorithm uses hash counts to filter the candidates. Also, the generation of smaller candidate sets enables DHP to effectively trim the transaction database. In this paper, we propose using the hash-based approach combined with the transaction identifier (TID) list [1] for the AGM algorithm to filter the generated adjacency matrices before checking the downward closure property. The reason for applying this approach to AGM is that AGM is the only method so far which can mine general frequent subgraphs, including unconnected ones, where the complexity of candidate generation is huge comparing with the other Apriori-like graph mining algorithms. Under this condition, our hash-based approach is expected to be very effective. This approach is not limited to AGM, rather it can be applied to any Apriori-like graph mining algorithms such as AcGM or FSG. It is also expected to be applied to the Apriori-like algorithms for mining itemsets. As this paper aims at providing precise information about the way these modifications are implemented, we will start with presenting a rather detailed summary of the original AGM method as far as needed for describing the modifications.

2 Graph and Problem Definitions

A graph in which all of its vertices and edges have labels is mathematically defined as follows.

Definition 1 (Labeled Graph) *A labeled graph*

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

is made of four sets, the set of vertices $V(G) = \{v_1, v_2, ..., v_k\}$, the set of edges connecting some vertex pairs in $V(G)$: $E(G) = \{e_h = (v_i, v_j) | v_i, v_j \in V(G)\}$, the set of vertex labels $L_V(V(G)) = \{lb(v_i) | \forall v_i \in V(G)\}$, and the set of edge labels $L_E(E(G)) = \{lb(e_h) | \forall e_h \in E(G)\}$. If e_h is undirected, both (v_i, v_j) and (v_j, v_i) belong to $E(G)$. The number of vertices, $\mid V(G) \mid$, is called the size of graph G.

If all the vertices and edges of a graph have the same vertex and edge label associated with them, we will call it an *unlabeled* graph.

Definition 2 (Subgraph and Induced Subgraph) *Given a graph $G = (V(G), E(G), L_V(V(G)), L_E(E(G)))$, a subgraph of G, $G_s = (V(G_s), E(G_s),$*

$L_V(V(G_s)), L_E(E(G_s)))$, is a graph which fulfills the following conditions.

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

A subgraph whose size is k is called a k-subgraph.

If $V(G_s) \subset V(G)$ and $E(G_s)$ contains all the edges of $E(G)$ that connect vertices in $V(G_s)$, we call G_s an induced subgraph of G. In this case, G_s is said to be included in G and denoted as $G_s \in G$.

The aforementioned graph G is represented by an adjacency matrix X which is a very well-known representation in mathematical graph theory [4, 5]. The transformation from G to X does not require much computational effort [6, 7].

Definition 3 (Adjacency Matrix) *Given a graph $G = (V(G), E(G), L_V(V(G)), L_E(E(G)))$, the adjacency matrix X is a square matrix which has the following (i, j)-element, x_{ij},*

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

where $i, j \in \{1, \ldots, |V(G)|\}$ and $num(lb(e_h))$ is a natural number assigned to an edge label $lb(e_h)$.

Each element of an adjacency matrix in the standard definition is either '0' or '1', whereas each element in Definition 3 can be the number of an edge label.

The vertex corresponding to the i-th row (i-th column) of an adjacency matrix is called the i-th vertex. The adjacency matrix of a graph of size k and the corresponding graph are denoted as X_k and $G(X_k)$, respectively. An identical graph structure can be represented by multiple adjacency matrices depending on the assignment of its rows and columns to its vertices. Such adjacency matrices are mutually convertible using the so-called transformation matrix [6, 7].

In order to reduce the number of candidates of frequent induced subgraphs, a coding method of the adjacency matrices needs to be introduced.

Definition 4 (Code of Adjacency Matrix) *In case of an undirected graph, the code of a vertex-sorted adjacency matrix X_k is defined as*

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

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

$$code(X_k) = c_1c_2 \ldots c_{\frac{k(k-1)}{2}},$$

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

A canonical form of adjacency matrices representing an identical graph is introduced to remove this ambiguity and to handle the matrices efficiently.

Definition 5 (Canonical Form) *The canonical form of adjacency matrices representing a graph is the unique matrix having a minimum (maximum) code. This minimum (maximum) code is referred to as the canonical label of the graph.*

The canonical label is the unique code that is invariant on the ordering of the vertices and edges in the graph.

Definition 6 (Graph and Subgraph Isomorphism) *Two graphs $G_1 = (V(G_1), E(G_1), L_V(V(G_1)), L_E(E(G_1)))$ and $G_2 = (V(G_2), E(G_2), L_V(V(G_2)), L_E(E(G_2)))$ are isomorphic if there is a bijection mapping between $V(G_1)$ and $V(G_2)$ such that G_1 and G_2 are identical, i.e., each edge of $E(G_1)$ is mapped to a single edge in $E(G_2)$ and vice versa. This mapping must also preserve the labels of vertices and edges.*

The problem of graph isomorphism is to determine an isomorphism between G_1 and G_2. Meanwhile, the problem of subgraph isomorphism is to find an isomorphism between G_2 and a subgraph of G_1.

Two graphs are isomorphic if and only if they have the same canonical label. Canonical labels therefore are extremely useful as they allow us to quickly compare two graphs regardless of the original vertex and edge ordering. However, the problem of determining the canonical label of a graph is equivalent to determining the isomorphism between two graphs [9]. Graph isomorphism is not known to be in P or NP-complete and subgraph isomorphism is known to be in NP-complete [5].

Definition 7 (Graph Transaction and Graph Database) *A graph G is referred to as a graph transaction or simply a transaction, and a set of graph transactions GD, where $GD = \{G_1, G_2, ..., G_n\}$, is referred to as a graph database.*

Definition 8 (Support) *Given a graph database GD and a graph G_s, the support of G_s is defined as*

$$sup(G_s) = \frac{\textit{number of graph transactions } G \textit{ where } G_s \in G \in GD}{\textit{total number of graph transactions } G \in GD}.$$

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

Definition 9 (Frequent Induced Subgraph Mining Problem) *Given a graph database GD and a minimum support (minsup), the problem is to derive all frequent induced subgraphs in GD.*

3 Algorithm of AGM-Hash

3.1 Candidate Generation

The generation of candidate frequent induced subgraphs in the AGM method is made by the levelwise search in terms of the size of the subgraphs [7]. Let X_k and Y_k be vertex-sorted adjacency matrices of two frequent induced subgraphs $G(X_k)$ and $G(Y_k)$ of size k. They can be joined if and only if the following conditions are met.

1. X_k and Y_k are identical except the k-th row and the k-th column, and the vertex labels of the first $(k-1)$ vertices are the same between X_k and Y_k.
2. In case that labels of the k-th vertex of $G(X_k)$ and the k-th vertex of $G(Y_k)$ are identical, $code(X_k) \leq code(Y_k)$. In case that they are not identical, $num(lb(v_k \in V(G(X_k)) < lb(v_k \in V(G(Y_k)))$.

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

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

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

$$lb(v_i \in V(G(Z_{k+1}))) = lb(v_i \in V(G(X_k))) = lb(v_i \in V(G(Y_k)))(i = 1, \ldots, k-1),$$
$$lb(v_k \in V(G(Z_{k+1}))) = lb(v_k \in V(G(X_k))),$$
$$lb(v_{k+1} \in V(G(Z_{k+1}))) = lb(v_{k+1} \in V(G(Y_k))).$$

X_k and Y_k are called the first and second generator matrix, respectively. The aforementioned condition 1 ensures that $G(X_k)$ and $G(Y_k)$ share the common structure except the vertex corresponding to the last row and column of each adjacency matrix. Meanwhile, condition 2 is required to avoid producing redundant adjacency matrices by exchanging X_k and Y_k, *i.e.*, taking Y_k as the first generator matrix and X_k as the second generator matrix. Unlike the joining of itemsets in which two frequent k-itemsets lead to a unique $(k+1)$-itemset [1, 10], the joining of two adjacency matrices of two frequent induced subgraphs of size k can lead to multiple distinct adjacency matrices of size $k+1$. Two elements $z_{k,k+1}$ and $z_{k+1,k}$ of Z_{k+1} are not determined by X_k and Y_k. In the case of mining of undirected graph data, the possible graph structures for $G(Z_{k+1})$ are those wherein there is a labeled edge and those wherein there is no edge between the k-th vertex and the $(k+1)$-th vertex. Therefore, $(|L_E|+1)$ adjacency matrices satisfying $z_{k,k+1} = z_{k+1,k}$ are generated, where $|L_E|$ is the number of edge labels. In the case of mining directed graph data, $(|L_E|+1)^2$ adjacency matrices are generated since $z_{k,k+1} = z_{k+1,k}$ does not apply. The vertex-sorted adjacency matrix generated under the above conditions is called a *normal form.*

In the standard basket analysis, a $(k+1)$-itemset becomes a candidate frequent itemset only when all of its k-sub-itemsets are confirmed to be frequent itemsets [1], *i.e.*, the downward closure property is fulfilled. Similarly, the graph G of size $k+1$ is a candidate of frequent induced subgraphs only when all adjacency matrices generated by removing from the graph G the i-th vertex v_i $(1 \leq i \leq k+1)$ and all of its connected links are confirmed to be adjacency matrices of frequent induced subgraphs of size k. As this algorithm generates only normal-form adjacency matrices in the earlier k-levels, if the adjacency matrix of the graph generated by removing the i-th vertex v_i is a non-normal form, it must be transformed to a normal form to check if it matches one of the normal form matrices found earlier. An adjacency matrix X_k of a non-normal form is transformed into a normal form X'_k by reconstructing the matrix structure in a bottom up manner. The reader is referred to [7] for a detailed mathematical treatment of normalization.

Though this heuristic helps prune redundant candidates efficiently, it consumes much computational effort due to the large number of adjacency matrices produced. Moreover, the task of checking if a graph of size $(k+1)$ can be a candidate, *i.e.*, checking its downward closure property, is very expensive since we need to perform many graph isomorphism tests. In Section 3.3, we will introduce a hash-based method for AGM, called AGM-Hash, to reduce the number of adjacency matrices of one size larger before checking the downward closure property. This technique helps prune a significant number of redundant candidates right after the joining operation has been performed.

3.2 Frequency Counting

Once the candidate frequent induced subgraphs have been generated, we need to count their frequency. The naïve strategy to achieve this end is for each candidate to scan each one of the graph transactions in the graph database and determine if it is contained or not using subgraph isomorphism. However, having to compute these isomorphisms is particularly expensive and this approach is not feasible for large datasets [9].

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. Every subgraph of each graph transaction G in the graph database can be represented by an adjacency matrix X_k, but it may not be a normal form in most cases. Since the candidates of frequent induced subgraphs in the AGM method are normal forms, it is required to derive the normal forms of every induced subgraph of G at each level. The frequency of each candidate is counted based on all normal forms of the induced subgraphs of G of same size and the canonical label of that candidate. When the value of the count exceeds the minimum support threshold, the subgraph is a frequent induced subgraph. Since some normal forms may represent the same graph and thus, the number of normal forms of induced subgraphs present in each transaction can be extremely large, we need to reduce that number of normal forms before counting support. Note that a normal form of size k is generated from two normal forms of size $k-1$. Therefore, if one of the two normal forms of size $k-1$ does not represent a frequent induced subgraph, then the normal form of size k cannot represent any frequent induced subgraph, and thus should not be considered for support counting. For this reason, AGM does not use normal forms of infrequent induced subgraphs to produce normal forms of any size larger.

In the AprioriTid algorithm [1], the transaction database is not used for counting support after the first pass. Rather, the time needed for the support counting procedure is reduced by replacing every transaction in the database by the set of candidate itemsets that occur in that transaction. This is done repeatedly at every iteration k. If a transaction does not contain any candidate

k-itemsets, then the adapted transaction database will not have an entry for this transaction. We employ this idea of ApriotiTid for counting support. The adapted graph database at level k is denoted as $\overline{C}^k$. Each member of the set $\overline{C}^k$ is of the form $< TID, \{X_k\} >$, where each X_k is a normal form of a potentially k-frequent induced subgraph present in the transaction with identifier TID. The member of $\overline{C}^k$ corresponding to transaction t is $< t.TID, C_t^k >$ with $C_t^k = \{c \in C^k \mid c \text{ contained in } t\}$, where C^k is the set of normal forms of candidate k-frequent induced subgraphs. If transaction t does not contain any candidate k-frequent induced subgraph, *i.e.*, $C_t^k = \emptyset$, then $\overline{C}^k$ will not have an entry for this transaction. As $\overline{C}^k$ will be used to generate normal forms of one size larger in order to count support for candidate $(k+1)$-frequent induced subgraphs, if $c \in C^k$ and c is not a normal form of any frequent induced subgraph, then c is removed from C_t^k.

3.3 Details of the AGM-Hash Algorithm

Fig. 1 gives the algorithmic form of AGM-Hash which is divided into two parts. Part 1 detects the set of 1-frequent induced subgraphs, *i.e.*, individual vertices that frequently occur in the graph database GD, and the set of their normal forms (*i.e.*, N^1). In addition, this part computes $\overline{C}^1$ which contains normal forms of 1-frequent induced subgraphs present in each graph transaction. Finally, AGM-Hash makes a hash table for 2-subgraphs (*i.e.*, H^2) using $\overline{C}^1$.

Part 2 consists of two phases. The first phase is to generate candidates of k-frequent induced subgraph C^k based on the normal forms of $(k-1)$-frequent induced subgraphs (*i.e.*, N^{k-1}) made in the previous pass and the hash table (*i.e.*, H^k), and calculates the canonical labels for each candidate. In the second phase, AGM-Hash generates the normal forms of induced subgraphs contained in each graph transaction $t \in \overline{C}^{k-1}$. Based on those normal forms, AGM-Hash counts the support for the candidates according to their canonical labels. If a candidate appears in transaction t, it will be added to C_t^k which is the set of candidates contained in t. These steps are repeated for every $t \in \overline{C}^{k-1}$. If $C_t^k \neq \emptyset$, it will be used for making the hash table for $(k+1)$-subgraphs (*i.e.*, H^{k+1}) and added to $\overline{C}^k$. In the following, the normal forms of k-frequent induced subgraphs (*i.e.*, N^k) and the set of k-frequent induced subgraphs are realized. Finally, AGM-Hash removes from C_t^k all normal forms of infrequent induced subgraphs, and from $\overline{C}^k$ all transactions t which do not contain any normal forms (*i.e.*, $C_t^k = \emptyset$). $\overline{C}^k$ is cached on the hard disk due to its huge size.

The subprocedures for the AGM-Hash algorithm are shown in Fig. 2. Same as AGM [7], AGM-Hash also generates adjacency matrices of size k based on N^{k-1}. However, similar to DHP [10], AGM-Hash employs the hash table (can be replaced by a bit vector) to test the validity of each adjacency matrix. A generated adjacency matrix of size k will be checked the downward closure property to be a candidate of k-frequent induced subgraphs only if that matrix is hashed into a hash entry whose value is larger than or equal to the minimum support

Part 1

```
1.  $s = minsup|GD|$; //a minimum support
2.  $F^1$ = { 1-frequent induced subgraphs };
3.  $N^1$ = { normal forms of 1-frequent induced subgraphs };
4.  $\overline{C}^1 = \{< t.TID, C_t^1 >| C_t^1$ are normal forms of 1-frequent induced subgraphs present in $t\}$;
5.  set all the buckets of $H^2$ to zero;
6.  forall entries $t \in \overline{C}^1$ do
7.      if $C_t^1 \neq \emptyset$ then do begin
8.          $N_t^2$ = gen-norm ($C_t^1$); //generate normal forms of induced subgraphs in $t$
9.          forall normal forms of size 2, $X \in N_t^2$, do
10.             $H^2[h_2(X)] + +$;
11.     end
```

Part 2

```
1.  $k = 2$;
2.  while $C^k$ = gen-candidate($N^{k-1}, H^k$) $\neq \emptyset$ do begin
3.      forall entries $t \in \overline{C}^{k-1}$ do begin
4.          $N_t^k$=gen-norm ($C_t^{k-1}$); //generate normal forms of induced subgraphs in $t$
5.          forall candidates $c \in C^k$ do
6.              if $c \in N_t^k$ then do begin
7.                  $c.canonical\_label.count + +$; //increase count of corresponding can. label
8.                  $C_t^k$ += $\{c\}$
9.              end
10.         if $C_t^k \neq \emptyset$ then do begin
11.             $\overline{C}^k$ += $< t.TID, C_t^k >$;
12.             makehasht($C_t^k, H^{k+1}$);
13.         end
14.     end
15.     $N^k = \{c \in C^k \mid c.canonical\_label.count \geq s\}$;
16.     $F^k$ ={k-frequent induced subgraphs extracted from $N^k$};
17.     forall entries $t \in \overline{C}^k$ do begin
18.         forall $c \in C_t^k$ do
19.             if $c \notin N^k$ then $C_t^k$ -= $\{c\}$;
20.         if $C_t^k = \emptyset$ then $\overline{C}^k$-=$< t.TID, C_t^k >$;
21.     end
22.     k++;
23. end
```

Fig. 1. Main program of the AGM-Hash algorithm

minsup. AGM-Hash is unique in that it employs graph invariants to make the hash function. In this paper, the hash function value of an induced subgraph is calculated based on its number of edges having the edge label of each type as well as that of vertices, with different scales between edge labels and vertex labels due to the need of diversity of hash values reflecting the graph topology. For example, given an induced subgraph X of size 4 which has two and four edges having the labels numbered 1 and 2, respectively, three and one vertices having the labels numbered 1 and 2, respectively, and a hash table whose size is 80, then $h_4(X) = (2 \times 10^1 + 4 \times 10^2) + (3 \times 1 + 1 \times 2)$ mod 80=25. Note that the number of k-candidate subgraphs depends on the number of k-subsets of vertices which has the largest value at $k = [N/2]$, where N is the number of vertices. It also depends on the number of edge labels. Therefore, the hash table size should be chosen reflecting this observation. In this paper the hash table size was empirically chosen as $2^{|L_E|} \times \binom{|T|}{k}$, where $|L_E|$ is the number of edge labels, $|T|$ is the average graph size in the graph database and $k = [|T|/2]$.

```
Procedure $C^k$ = gen-candidate($N^{k-1}$, $H^k$)
1. $C^k = \emptyset$
2. foreach pair of normal forms, $N_i^{k-1}$ and $N_j^{k-1}$, do
3.     if $N_i^{k-1}$,$N_j^{k-1}$ can be joined then
4.         $C^k$ += { normal forms of size $k$, $X$, generated from $N_i^{k-1}$, $N_j^{k-1}$ | $H^k[h_k(X)] \geq s$};
5. forall $c \in C^k$ do
6.     if $c$ holds the downward closure property then
7.         calculate $c.canonical\_label$;//calculate canonical label for $c$
8.     else $C^k$-={$c$};

Procedure makehasht($C_t^k$, $H^{k+1}$)
1. $N_t^{k+1}$ = gen-norm($C_t^k$);
2. forall normal forms of size $(k+1)$, $X \in N_t^{k+1}$, do
3.     $H^{k+1}[h_{k+1}(X)] + +$;

Procedure $N_t^{k+1}$ = gen-norm($C_t^k$)
1. $N_t^{k+1} = \emptyset$;
2. foreach pairs of normal forms of size $k$, $C_i^k$ and $C_j^k \in C_t^k$, do
3.     if $C_i^k$ and $C_j^k$ can be joined then do begin
4.         construct $X$, a normal form of the $(k+1)$-induced subgraph in $t$ whose vertices are
   $k+1$ vertices of $C_i^k$ and $C_j^k$;
5.         $N_t^{k+1} + = \{X\}$;
6.     end
```

Fig. 2. Subprocedures for the AGM-Hash algorithm.

Moreover, unlike the DHP method, AGM-Hash tests the validity of adjacency matrices before checking if they fulfill the downward closure property since graph isomorphism tests are particularly expensive.

4 Experiments

To assess the performance of the AGM-Hash algorithm, we conducted several experiments on both synthetic and real-world graph databases. The AGM-Hash method proposed in this paper was implemented by C++. All experiments were done on a Pentium 2.8 GHz machine with 2 GB main memory, running the Windows XP operating system. All the times reported are in seconds.

4.1 Simulation Results

The basic performance of the proposing method was examined using the graph-structured transactions that were artificially generated in a random manner. The number of vertices in a graph, is determined by the gaussian distribution having the average of $|T|$ and the standard deviation of 1. The edges are attached randomly with the probability of p. The vertex labels and edge labels are randomly determined with equal probability. The number of vertex labels and the number of edge labels are denoted as $|L_V|$ and $|L_E|$, respectively. 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. The test data are generated for both cases of directed and undirected graphs.

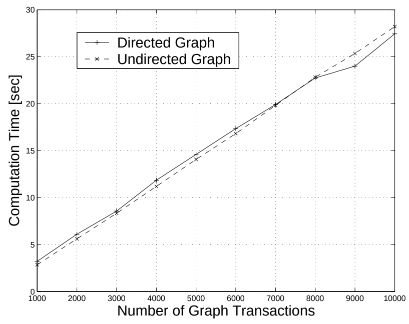

Fig. 3. Computation time v.s. number of transactions.

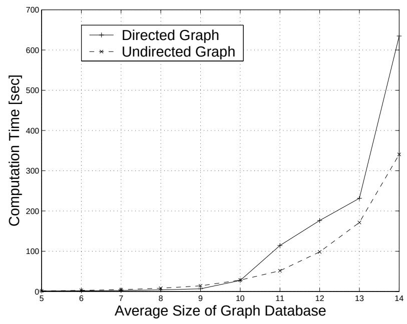

Fig. 4. Computation time v.s. average transaction size.

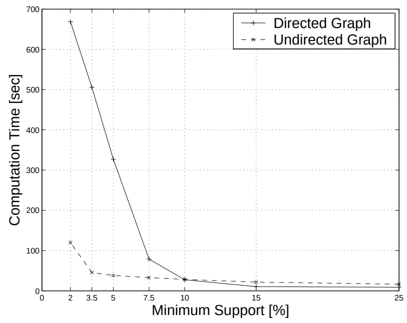

Fig. 5. Computation time v.s. minimum support.

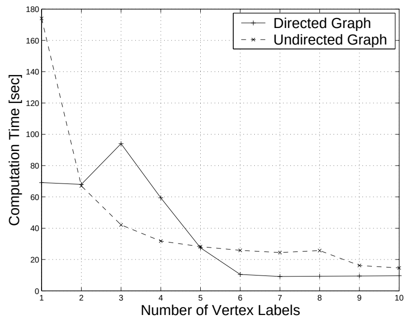

Fig. 6. Computation time v.s. number of vertex labels.

The default parameter values are $|GD| = 10000$, $|T| = 10$, $L = 10$, $|I| = 7$, $|L_V| = 5$, $|L_E| = 5$, $p = 50\%$ and *minsup* (minimum support) = 10%.

Figs. 3, 4, 5 and 6 show the computation time variations when $|GD|$, $|T|$, *minsup* and $|L_V|$ are changed respectively in the frequent induced subgraph problem. The default parameter values were used in each figure except the parameter that was changed. The computation time is observed to increase almost linearly with $|GD|$ in Fig. 3 and to be exponential to $|T|$ in Fig. 4. Fig. 5 indicates that computation time decreases rapidly as *minsup* increases. This is trivial since less *minsup* results in larger number of frequent patterns. The observed tendencies in Figs. 3, 4 and 5 are almost identical with the properties of the Apriori algorithm [1]. Fig. 6 shows that computation time mostly decreases as $|L_V|$ increases, which also has the tendency similar to Apriori. Because the larger number of vertices increases the variety of subgraph patterns, the frequency of each pattern is decreased, and this effect reduces the search space of the frequent induced subgraphs.

4.2 Performance Evaluation

The performance of the proposed AGM-Hash algorithm was compared with that of the original AGM method using a real-world graph database. This graph database consists of 82 mutagenesis compounds [8] whose average size is about

12.5. There are 7 graph transactions having the maximum size of 15 and 1 graph transaction having the minimum size of 9. The parameters evaluated include the number of adjacency matrices being tested the downward closure property, the number of candidates and computation time. It should be noted that unlike recently proposed methods, AGM and AGM-Hash find frequent induced subgraphs, *i.e.*, they can be either connected or unconnected. It is also worth noting that AcGM, which is a variant of AGM to mine frequent connected subgraphs, is fast enough [6] and it is AGM that needs improvement.

Table 1 shows the experimental results obtained from both AGM-Hash and AGM given a minimum support of 10%. The number of frequent induced subgraphs and the number of their normal forms are also shown together for reference. The maximum size of frequent induced subgraphs discovered by AGM and AGM-Hash is 12. As can be seen from Table 1, AGM-Hash efficiently reduces the number of adjacency matrices generated by the joining operation before passing them to the step of checking the downward closure property. This task, as shown earlier, is particularly expensive since it requires many graph isomorphism tests. As a result, computation time required by AGM-Hash is reduced significantly compared with that required by AGM. Meanwhile, the number of candidate matrices in the case of AGM-Hash is reduced slightly compared with that in AGM, which confirms the efficiency of the pruning method based on the Apriori-like principle and employed in AGM.

It can also be seen from Table 1 that computation time is less in the case of AGM-Hash since level 8 though the number of adjacency matrices being checked the downward closure property has been reduced significantly in earlier levels. This is mainly attributed to the additional cost required for making the hash table as well as for testing the validity of generated adjacency matrices.

5 Conclusion and Future Work

A hash-based approach to the AGM algorithm has been introduced to improve the method in terms of efficiency. The use of graph invariants for calculating the hash values and the combination of the hashing technique and the TID list can be highlighted as the main features of the proposed AGM-Hash method. Its performance has been evaluated on both the synthetic graph data and the real-world chemical mutagenesis data. The powerful performance of this approach under some practical conditions has been confirmed through these evaluations. Future work includes a study of the graph invariants to be used for making the hash table to further enhance the proposed method. Also, a design of the hash table for the purpose of deriving characterized frequent subgraphs is being considered. The AGM-Hash approach can be extended to any Apriori-like graph mining algorithm.

References

1. Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. In Proceedings of the 20th VLDB Conference, pp.487–499.

Table 1. Comparisons of AGM and AGM-Hash.

	# kept matrices		Elapsed time (sec)		# candidate matrices			
Level	AGM	AGM-Hash	AGM	AGM-Hash	AGM	AGM-Hash	# NF	# FSG
1	-	-	0	0	8	8	6	6
2	84	36	0.016	0.187	84	36	28	28
3	492	223	0.062	0.422	172	129	103	86
4	2440	1084	0.266	1.047	436	406	371	214
5	12380	5826	1.156	2.781	1516	1507	1351	443
6	54040	23923	4.812	7.203	5050	4956	4595	758
7	180872	89916	18.922	19.031	14473	14384	12334	992
8	316796	157107	51.922	41.531	24713	24673	22377	904
9	310668	130337	94.969	67.359	26947	26947	25833	522
10	212740	47434	121.297	73.609	17576	17562	17562	174
11	94604	10732	133.984	75.328	5832	5832	5832	30
12	25432	526	137.375	75.516	526	526	526	2
Total	1210548	467144	137.375	75.516	97353	96966	90918	4159

# kept matrices: number of matrices being tested the downward closure property.
# NF: number of normal forms of frequent induced subgraphs for each level.
# FSG: number of frequent induced subgraphs derived at each level.

2. Borgelt, C. and Berthold, M.R. 2002. Mining molecular fragments: Finding relevant substructures of molecules. In Proceedings of the 2nd IEEE International Conference on Data Mining (ICDM'02), pp.51–58.
3. Cook, D.J. and Holder, L.B. 1994. Substructure discovery using minimum description length and background knowledge, Journal of Artificial Intelligence Research, Vol.1, pp.231–255.
4. Diestel, R. 2000. Graph Theory. Springer-Verlag, New York.
5. Fortin, S. 1996. The graph isomorphism problem. Technical Report 96-20, University of Alberta, Edmonton, Alberta, Canada.
6. Inokuchi, A., Washio, T., Nishimura, K. and Motoda, H. 2002. A fast algorithm for mining frequent connected subgraphs. IBM Research Report RT0448, Tokyo Research Laboratory, IBM Japan.
7. Inokuchi, A., Washio, T. and Motoda, H. 2003. Complete mining of frequent patterns from graphs: Mining graph data. Machine Learning, Vol.50, pp.321–354.
8. Debnath, A.K., *et al.* 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. Journal of Medical Chemistry, Vol. 34, pp.786–797.
9. Kuramochi, M. and Karypis, G. 2001. Frequent subgraph discovery. In Proceedings of the 1st IEEE International Conference on Data Mining (ICDM'01), pp.313–320.
10. Park, J.S., Chen, M-.S. and Yu, P.S. 1997. Using a hash-based method with transaction trimming for mining association rules. IEEE Trans. Knowledge and Data Engineering, Vol.9, No.5, pp.813–825.
11. Washio, T. and Motoda, H. 2003. State of the art of graph-based data mining. SIGKDD Explorations, Vol.5, No.1, pp.59–68.
12. Yan, X. and Han, J. 2002. gSpan: Graph-based structure pattern mining. In Proceedings of the 2nd IEEE International Conference on Data Mining (ICDM'02), pp.721–724.
13. Yoshida, K. and Motoda, H. CLIP: Concept learning from inference patterns. Artificial Intelligence, Vol.75, No.1, pp.63–92.