

An Optimization of Closed Frequent Subgraph Mining Algorithm

*J. Demetrovics*¹, *H. M. Quang*², *N. V. Anh*², *V. D. Thi*³

¹*Computer and Automation Institute Hungarian Academy of Sciences, Hungary*

²*Institute of Information Technology, Vietnam Academy of Science and Technology, Viet Nam*

³*The Information Technology Institute (ITI), Vietnam National University, Hanoi, Viet Nam*

Emails: demetrovics@sztaki.mta.hu hoangquang@ioit.ac.vn anhmv@ioit.ac.vn vdthi@vnu.edu.vn

Abstract: *Graph mining is a major area of interest within the field of data mining in recent years. A key aspect of graph mining is frequent subgraph mining. Central to the entire discipline of frequent subgraph mining is the concept of subgraph isomorphism. One major issue in early subgraph isomorphism research concerns computational complexity. Normally, the subgraph isomorphism problem is NP-complete. Previous studies of frequent subgraph mining have not solved NP-complete problem in the subgraph isomorphism. In this paper, we propose a new algorithm which can deal with this problem. The proposed algorithm can solve the subgraph isomorphism in polynomial time in some settings. Moreover, the new algorithm is proved theoretically more effective than previous studies in closed frequent subgraph mining.*

Keywords: *Frequent patterns, closed frequent subgraph, frequent subgraphs, subgraph mining, subgraph isomorphism.*

1. Introduction

Data mining is a process for extracting knowledge from data. The data can be represented in many formats of structured data such as tables [1, 13], graphs [11], etc. Graph mining is recently a major area of interest within the field of data mining. A key aspect of graph mining is frequent subgraph mining. Frequent patterns are itemsets, subsequences, or substructures that appear in a data set with frequency no less than a user-specified threshold [4]. A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently in a graph database, it is called a frequent structural pattern. Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data indexing, classification, clustering, and other data mining tasks as well. Thus, frequent pattern

mining has become an important data mining task and a focused theme in data mining research.

Generally, Frequent Subgraph Mining (FSM) aims to identify all subgraph patterns whose occurrences within a graph data set are above a user defined threshold. These subgraph patterns are called frequent subgraphs. Theoretically, frequent subgraph mining can be formulated as a search in a search space, modelled by a lattice, consisting of all possible subgraph patterns. Because the number of possible frequent subgraphs increases exponentially with the size of the graph, traversing the search space completely is computationally intractable, because of a “combinatorial explosion”. Thus, a user specified support threshold is often used to prune this combinatorial search space, i.e., to separate infrequent subgraphs from the frequent ones. Frequent subgraph mining problem has received considerable critical attention [6, 8, 9, 17, 18]. It is widely accepted that FSM techniques can be divided into two categories: (i) the “A priori-based” approach (also called the BFS strategy based approach) [8] and (ii) the pattern growth approach [17]. Both approaches have advantages and disadvantages; however, they include generating candidate and isomorphism subgraph testing to decide which subgraph is frequent. In theoretical computer science, the subgraph isomorphism problem is a computational task in which two graphs G and H are given as input, and it has to be determined whether G contains a subgraph that is isomorphic to H . Subgraph isomorphism is a generalization of both the maximum clique problem and the problem of testing whether a graph contains a Hamiltonian cycle, and is therefore NP-complete [3]. However certain other cases of subgraph isomorphism may be solved in polynomial time [2, 5] for planar graphs.

A considerable amount of literature has been published on subgraph isomorphism in frequent subgraph mining problem [2, 5, 6, 10, 15]. These studies help to reduce time complexity of subgraph isomorphism. The use of adjacency matrices, although straightforward, does not lend itself to isomorphism detection, because the vertexes (and edges) can be enumerated in many different ways [16]. With respect to isomorphism testing, it is therefore desirable to adopt a consistent labelling strategy that ensures that any two identical graphs are labelled in the same way regardless of the order in which vertexes and edges are presented (i.e., a canonical labelling strategy). A canonical labelling strategy defines a unique code for a given graph [11]. Canonical labelling facilitates isomorphism checking because it ensures that if a pair of graphs is isomorphic, then their canonical labellings will be identical [9]. One simple way of generating a canonical labelling is to flatten the associated adjacency matrix by concatenating rows or columns to produce a code comprising a list of integers with a lexicographical ordering imposed. To further reduce the computation resulting from the permutations of the matrix, canonical labellings are usually compressed, using what is known as a vertex invariant scheme [11]; this allows the content of an adjacency matrix to be partitioned according to the vertex labels.

Alternative methods to reducing the search space include concentrating on the identification of a subset of the total set of frequent subgraphs, for example, closed frequent subgraph mining [18] or maximal frequent subgraph mining [7, 14].

Although these methods address the issue to some extent, the combinatorial explosion issue is still unresolved; significantly large numbers of closed frequent subgraphs and maximal frequent subgraphs are still generated.

In this paper, we propose new algorithm for closed frequent subgraph mining based on canonical labelling strategy, the Random Access Machine (RAM) model or von Neumann model [12] and the ‘‘A priori-based’’ approach. The RAM makes use of a random access memory, thus overcoming the limitation of Turing machines which use a sequential access tape as a memory component. The RAM that can access any field of their memory in one step has to know which cell to access, and each cell must be assigned an address. The subgraph isomorphism problem with canonical labelling strategy is defined by searching an element in an array according to code string and can utilize binary search on RAM. In the proposed algorithm, the subgraph isomorphism problem solve in polynomial time complexity. As far we know the subgraph isomorphism problem has not been considered in the random access machine model. This paper attempts to show that the new algorithm is faster than some algorithms in [6, 8, 9, 17, 18]. In addition, this paper also shows the correctness and gives the time complexity of the new algorithm.

2. Preliminaries

Definition 2.1. A labelled graph G is five element tuple $G = (V, E, \sum_V, \sum_E, l)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges. \sum_V and \sum_E are the set of vertex labels and edge labels respectively. The labelling function l defines the mappings $V \rightarrow \sum_V$ and $E \rightarrow \sum_E$.

Definition 2.2. Without loss of generality, we assume that there is a total order \geq on each label set \sum_V and \sum_E . A graph $G = (V, E, \sum_V, \sum_E, l)$ is a *subgraph of other graph* $G' = (V', E', \sum_{V'}, \sum_{E'}, l')$ if

- (1) $V \subseteq V'$,
- (2) $\forall u \in V, (l(u) = l'(u))$,
- (3) $E \subseteq E'$,
- (4) $\forall (u, v) \in E, (l(u, v) = l'(u, v))$,

G' is also referred to as a *supergraph of G* .

Definition 2.3. Two graphs $G = (V, E, \sum_V, \sum_E, l)$, $G' = (V', E', \sum_{V'}, \sum_{E'}, l')$ are *isomorphic* if there exists a bijection $f : V \rightarrow V'$ such that:

- (1) $\forall u \in V, (l(u) = l'(f(u)));$
- (2) $\forall u, v \in V, ((u, v) \in E) \Leftrightarrow (f(u), f(v)) \in E';$
- (3) $\forall (u, v) \in E, (l(u, v) = l'(f(u), f(v))).$

Definition 2.4. A labelled graph G is *subgraph isomorphism* to a labelled graph G' , denoted by $G \subseteq G'$, if there exists a subgraph G'' of G' such that G is isomorphic to G'' .

Definition 2.5. Given a set of graphs GD (referred as a Graph Database) and a *threshold* σ ($0 \leq \sigma \leq 1$), the support of a graph G , denoted by sup_G is defined as the fraction of graphs in GD to which G is subgraph isomorphic of G' :

$$\text{sup}_G = \frac{|\{G' \in \text{GD} \mid G \subseteq G'\}|}{|\text{GD}|}.$$

G is *frequent* if $\text{sup}_G \geq \sigma$.

Definition 2.6. The frequent subgraph mining problem is given a threshold σ and a graph database GD, finding all *frequent subgraphs* in GD.

Definition 2.7. A set consisting of all Frequent Subgraphs of graph g denoted as $\text{FS}(g)$.

Definition 2.8. If g is a *subgraph* of g' , then g' is a *supergraph* of g , denoted by $g \subseteq g'$ (*proper supergraph*, if $g \subset g'$). Let FS be the set of frequent subgraphs, the set of *closed frequent subgraphs*, CS, is defined $\text{CS} = \{g \mid g \in \text{FS}, \nexists g' \in \text{FS}: g \subset g' \text{ and } \text{sup}_g = \text{sup}_{g'}\}$.

Definition 2.9. A k -*subgraph* of graph g is a subgraph $g' \subseteq g$ such that $|V_{g'}|=k$.

Definition 2.10. Given an $n \times n$ adjacency matrix M of a graph G with n vertices, we define the code of M , denoted by $\text{code}(M)$, as the sequence formed by concatenating lower triangular entries of M (including entries on the diagonal) in the order: $m_{1,1}m_{2,1}m_{2,2} \dots m_{n,1}m_{n,2} \dots m_{n,n-1}m_{n,n}$ where $m_{i,j}$ is the entry at the i -th row and j -th column in M ($0 < j \leq i \leq n$). We assume that the rows in M are numbered 1 through n from top to bottom and the columns are numbered 1 through n from left to right.

3. The PSI-CFSM algorithm

In this paper, we propose a method about optimization computation for subgraph isomorphism in frequent subgraph mining. The proposed method is proved theoretically faster than gSpan [17] and FFSM [6]. A graph will be represented uniquely with a code by using canonical labelling, lexicographic order similar to MDFS-C [17, 18] and CAM [6, 8, 9]. The unique representation of a graph with a code of CAM or MDFS-C will avoid duplication generating of subgraphs and can help to construct ordered array that contain subgraphs in code of MDFS-C of CAM. Furthermore, the test subgraph isomorphism described as follow can utilize binary search in random access machine model. We consider subgraph isomorphism problem that is to match a pair of two subgraph. This problem is equivalence to comparing a pair of strings that are codes of MDFS-C of CAM representation of the

two subgraphs. The time complexity of binary search is $O(\log n)$ where n is the size of the input ordered array. The algorithm is called Polynomial Subgraph Isomorphism Closed Frequent Subgraph Mining (PSI-CFSM). It uses “A priori-based” approach and random access machine model. According to “A priori-based” approach, the frequent subgraph mining process starts from level one that generates a set C_2^i of 2-subgraph (subgraphs have two nodes or only one edge) candidates of each graph G_i in graph database GD. It then tests 2-subgraph isomorphism and counts support each of the set C_2^i to find a set of frequent 2-subgraphs of graph G_i , a set of frequent 2-subgraphs of GD, a set of closed frequent 2-subgraph of G_i and a set of closed frequent 2-subgraph of GD which are denoted as FS_2^i , FS_2 , CS_2^i and CS_2 , respectively. The next process is a loop with $k \geq 3$ that generates a set C_k^i of candidate k -subgraphs (subgraphs having k nodes can contain more than $k-1$ edges and less than $(k(k-1)/2)$ edges) of G_i from the set of frequent 2-subgraph FS_2^i combined with the set of frequent $(k-1)$ -subgraphs FS_{k-1}^i . It then tests k -subgraph isomorphism and counts support of k -subgraph of C_k^i to find a set of frequent k -subgraphs of G_i , a set of frequent k -subgraphs of GD, a set of closed frequent k -subgraphs of G_i and a set of closed frequent k -subgraphs of GD denoted as FS_k^i , FS_k , CS_k^i and CS_k , respectively. In this process at k , every k -subgraph of CS_k^i and CS_k is constructed with a linked list containing $(k-1)$ -subgraph of FS_{k-1}^i and FS_{k-1} , respectively. The algorithm checks each k -subgraph of CS_k^i and CS_k such that the k -subgraph contains $(k-1)$ -subgraph of CS_{k-1}^i and CS_{k-1} in their linked list and removes identified $(k-1)$ -subgraphs in CS_{k-1}^i and CS_{k-1} , respectively. The loop process stops when no candidate k -subgraph is generated. The set of union sets CS_1 , CS_2 , ..., CS_k is the set of closed frequent subgraphs. We can see that k -subgraph isomorphism testing process is to find a code of unique representation of k -subgraph in an ordered array in which each element containing a code of unique representation of k -subgraph in the k -subgraphs set. Subgraph isomorphism testing process uses binary search in random access machine to reduce time complexity to polynomial. Hence, the PSI-CFSM algorithm is faster than gSpan, FFSM and FSG algorithms in running time.

3.1. Canonical labelling strategy

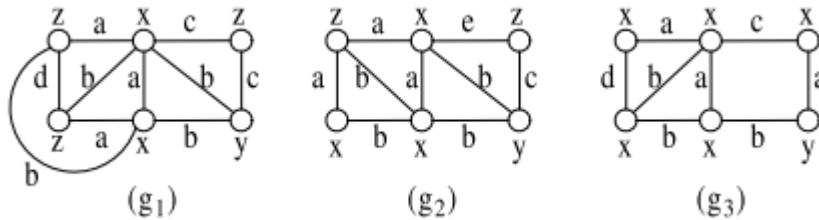


Fig. 1. A sample graph database GD

Minimum DFS Code (M-DFSC): There are a number of variants of the Depth First Search (DFS) code canonical labelling scheme; but essentially each vertex is given a unique identifier generated from a DFS traversal of the graph (DFS subscripting). Each constituent edge of the graph in the DFS code is then represented by a 5-tuple: (i, j, l_i, l_e, l_j) , where i and j are the vertex identifiers, l_i and l_j are the labels for the corresponding vertexes, and l_e is the label for the edge connecting the vertexes. Based on the DFS lexicographic order, the M-DFSC of a graph g is defined as the canonical labelling of g [17, 18]. The DFS codes for the left-most branch and the right-most branch of the example graph G given in Fig. 1. (g_1) are $\{(0, 1, x, a, x), (1, 2, x, a, z), (2, 3, z, d, z)\}$ and $\{(0, 1, x, a, x), (1, 4, x, b, y), (4, 5, y, c, z)\}$, respectively.

Canonical Adjacency Matrix (CAM): Given an adjacency matrix M of a graph g , an encoding of M can be obtained by the sequence of concatenating lower (or upper) triangular entries of M , including entries on the diagonal. Since different permutations of the set of vertexes correspond to different adjacency matrices, the canonical (CAM) form of g is defined as the maximal (or minimal) encoding. The adjacency matrix from which the canonical form is generated defines the Canonical Adjacency Matrix or CAM [6, 8, 9]. The encoding for the example graph G given in Fig. 1. (g_1), represented by the canonical adjacency matrix have $\text{code}(\text{CAM}(g_1)) = \text{xaxabzabdzb00yc000cz}$.

3.2. Generate subgraph candidates

gSpan [17] developed an efficient way to reduce the total number of nodes need to be considered. In gSpan, the extension operation is only performed to nodes on the “rightmost path” of a graph. Given a graph g and one of its depth first search trees T , the rightmost path of g with respect to T is the rightmost path of the tree T . gSpan chooses only one depth first search tree T which produces the canonical form of g for extension. gSpan extends one edge to right most path to receive $(k+1)$ -subgraph from k -subgraph (k -subgraph in gSpan means that the subgraph have k edges).

The FFSM [6] algorithm uses two procedures FFSM_Extension and FFSM_Join to generate candidate subgraphs. FFSM_Join combines two k -subgraphs to generate $(k+1)$ -subgraphs if the two k -subgraphs sharing a common $(k-1)$ -subgraph (k -subgraph in FFSM means that the subgraph have k edges) and the FFSM_Join does not generate unique $(k+1)$ -subgraphs from the two k -subgraphs. FFSM_Extension improves the efficient gSpan by always choosing a single fixed node in a CAM and attaches a newly introduced edge to it together with an additional node.

In the proposed algorithm, PSI-CFSM, we use an enumeration technique that is an extension operation to construct a $(k+1)$ -subgraph candidate g from a k -subgraph graph of G_i by adding additional edges (the k -subgraph means that the subgraph have k nodes). The newly introduced edge might connect two existing nodes or connect an existing node with a node introduced together with the edge. A simple way to perform the extension operation is to introduce every possible edge to every node in a graph g . This method has clearly a polynomial time complexity for the set of available vertex- and edge- labels for a graph g , respectively.

Procedure Combine(F_{k-1}^i, F_2^i)

Input: A set $F_{k-1}^i \subseteq FS(i)$, a set $F_2^i \subseteq FS(i)$

Output: a set of candidate k -subgraphs of graph g denote as C_k^i

1. $C_k^i \leftarrow \emptyset$
2. for each $u \in F_{k-1}^i$
3. $ng \leftarrow u$
4. $tg \leftarrow \text{diagonal}(u)$
5. for each $v \in F_2^i$
6. $ag \leftarrow \text{diagonal}(v) // ag = (x, y)$
7. if $((x \in ag \wedge x \in tg \wedge y \notin tg) \vee (y \in ag \wedge y \in tg \wedge x \notin tg))$ then
8. add new row in u
9. set $\text{location}_u(|\text{diagonal}(u)| + 1, \text{col}(x)) = \text{location}_v(2, 1)$
10. else if $((x \in ag \wedge x \in tg \wedge y \in tg) \vee (y \in ag \wedge y \in tg \wedge x \in tg))$ then
11. $\text{location}_u(\text{row}(y), \text{col}(x)) = \text{location}_v(2, 1)$
12. add ng into C_k^i
13. return C_k^i

Lemma 3.1. The procedure Combine(F_{k-1}^i, F_2^i) is correct.

Proof: We prove the correctness of Combine(F_{k-1}^i, F_2^i) by induction for all $k \geq 3$. In basis step at $k = 3$, Combine(F_{k-1}^i, F_2^i) generates a set C_3^i of candidate 3-subgraphs of graph $g_i \in \text{GD}$. Clearly, given u, v two graphs in F_2^i , $\text{diagonal}(u) = \{u_x, u_y\}$, $\text{diagonal}(v) = \{v_x, v_y\}$, if $\text{diagonal}(u) \neq \text{diagonal}(v)$ and $((u_x = v_x) \vee (u_x = v_y) \vee (u_y = v_x) \vee (u_y = v_y))$ then the combination of u, v will generate candidate 3-subgraphs $sg \subseteq g_i \in \text{GD}$. Inductive step: at $k > 3$ step, we suppose that Combine(F_{k-1}^i, F_2^i) is correct and generate C_k^i . The C_k^i after pruning $c \in C_k^i \wedge \text{sup}_c < \sigma$ obtains FS_k^i . We need prove that Combine(F_k^i, F_2^i) is correct and generates C_{k+1}^i . At $k+1$ step, Combine(F_k^i, F_2^i) generates C_{k+1}^i , let $sc \in C_{k+1}^i, sp \in C_k^i$, the $\text{diagonal}(sc)$ and $\text{diagonal}(sp)$ can be found and $|\text{diagonal}(sc)| - |\text{diagonal}(sp)| = 1$. By removing all edges containing node $nx = \text{diagonal}(sc) - \text{diagonal}(sp)$ to obtain subgraph nc then $nc \in F_k^i$. By hypothesis induction nc must be a member of C_k^i that is a candidate k -subgraph. Otherwise, $nc \in FS_k^i, FS_k^i \subseteq C_k^i \Rightarrow nc \in C_k^i$. Hence, at $k+1$ step, C_{k+1}^i is correct using Combine(F_k^i, F_2^i), and after pruning $rs \in C_{k+1}^i \wedge \text{sup}_{rs} < \sigma$ we will obtain FS_{k+1}^i . \square

Lemma 3.2. The procedure Combine(F_{k-1}^i, F_2^i) runs in polynomial time complexity.

Proof: Let m is the number of edges of a subgraph $g \in F_{k-1}^i$, $|\text{diagonal}(g)| = k-1$, $m \geq |\text{diagonal}(g)| - 1 = k-2$, n is cardinality of F_2^i (each subgraph in F_2^i contains only one edge), h is cardinality of F_{k-1}^i , before adding one node to $(k-1)$ -subgraph g to obtain k -subgraph g' then the number of maximal nodes can be add to g is $|V_{g_i}| - |\text{diagonal}(g)|$. By CAM representation, the number of edges of g maximum is $\frac{(k-2) \times (k-3)}{2}$. Assume that each subgraph $g \in F_{k-1}^i$ can be added maximum number of edges when generating candidate k -subgraph $g' \in C_k^i$ then the number of edges can be added to g is $(k-1)$. Thus, the procedure $\text{Combine}(F_{k-1}^i, F_2^i)$ has maximum number of computation step that is $h \times (|V_{g_i}| - (k-1)) \times (k-1)$. In our algorithm, we focus in closed frequent subgraph mining, we use the procedure $\text{Combine}(\text{CS}_{k-1}^i, \text{FS}_2^i)$ to generate the set of candidate k -subgraphs C_k^i and the cardinality of C_k^i is much less than using $\text{Combine}(\text{FS}_{k-1}^i, \text{FS}_2^i)$. \square

3.3. Test subgraph isomorphism

In algorithms [6, 8, 9, 17, 18] the subgraph isomorphism testing process runs in a sequence way. Therefore, a new candidate subgraph g generated by right most path extension (gSpan) [17] or by joining (FFSM) [6] must test subgraph isomorphism with every graph $g' \subseteq g_i \in \text{GD}$. The subgraph isomorphism testing process compares code of MDFS-C or CAM of one candidate subgraph g with every subgraph $g' \subseteq g_i \in \text{GD}$ in a set of very large subgraphs of graph $g_i \in \text{GD}$. Assume the number of subgraphs of graph $g_i \in \text{GD}$ is 2^n then the process implies 2^n comparison step. We can easily see why the process runs slow. Hence, subgraph isomorphism testing step of algorithms such as FFSM, gSpan, CloseGraph has the time complexity in NP class.

We improve subgraph isomorphism testing step of the above algorithms by using a random access machine model in binary search. In the complexity theory, the time complexity of binary search is $O(\log n)$ where n is number of candidate subgraphs. Assume the cardinality of candidate subgraphs is 2^n then number of computation steps of subgraph isomorphism by binary search on random access machine model is $\log_2 2^n = n$ and the time complexity is $O(n)$.

Procedure BinarySearch($L, x, \text{first}, \text{last}$)

Input: Array $L[\text{first}, \text{last}]$ and value x

Output: -1 if $x \notin L$ or $i, 0 \leq i < n$ if $L[i] = x$

Step 1. if ($\text{first} > \text{last}$) return -1

Step 2. else

Step 3. $\text{middle} \leftarrow \left\lfloor \frac{\text{first} + \text{last}}{2} \right\rfloor$

Step 4. if $(L[\text{middle}] = x)$ return middle

Step 5. else if $(L[\text{middle}] < x)$ return $\text{BinarySearch}(L, x, \text{middle} + 1, \text{last})$

Step 6. else return $\text{BinarySearch}(L, x, \text{first}, \text{middle} - 1)$

Lemma 3.3. The procedure $\text{BinarySearch}(L, x, \text{first}, \text{last})$ runs in time complexity $O(\log n)$.

Proof: Let $T(n)$ is the number of computation steps that algorithm BinarySearch needs to perform when the size of the input is n . At $n=0$ we have $T(0) = c'$, where c' is constant and $|L|=0$, the procedure just performs a constant number of computation steps. At $n > 0$, the procedure performs a constant number c of computation steps to find the element in the middle of L , compares that element with x and defines the range on the left half or on the right half of the array L for recursion. Assume that both halves of the array L have the same size, $(n-1)/2$. Hence, the total number of computation steps BinarySearch performs when $n > 0$ is $T(n) = c + T((n-1)/2)$. The recurrence equation as follow:

$$T(0) = c'$$

$$(3.3.1) \quad T(n) = c + T\left(\frac{n-1}{2}\right) \text{ if } n > 0,$$

$$T\left(\frac{n-1}{2}\right) = c + T\left(\frac{\frac{n-1}{2} - 1}{2}\right) = c + T\left(\frac{n-1-2}{2^2}\right) = c + T\left(\frac{n-2^0-2^1}{2^2}\right),$$

$$T\left(\frac{n-2^0-2^1-\dots-2^k}{2^{k+1}}\right) = c + T\left(\frac{n-2^0-2^1-\dots-2^{k+1}}{2^{k+2}}\right).$$

The procedure will stop whenever the argument is equal to zero:

$$(3.3.2) \quad \frac{n-2^0-2^1-\dots-2^{k+1}}{2^{k+2}} = 0;$$

$$(3.3.3) \quad T(n) = c + c + \dots + c + f\left(\frac{n-2^0-2^1-\dots-2^{k+1}}{2^{k+2}}\right).$$

The set of Equations (3.3.1) has $k+2$ equations, the number of c terms in (3.3.3) is $k+2$ and so $T(n) = (k+2)c + c'$. According to the fact that (3.3.2):

$$(3.3.4) \quad n = 2^0 + 2^1 + \dots + 2^{k+1} = \sum_{i=0}^{k+1} 2^i = 2^{k+2} - 1.$$

Therefore, taking logarithms on the both sides of the last equality (3.3.4) we obtain $k+2 = \log_2(n+1)$. Then, $T(n) = c \log_2(n+1) + c'$.

Ignoring constant terms, we finally conclude that $T(n) = O(\log n)$. \square

Lemma 3.4. The procedure $\text{BinarySearch}(L, x, \text{first}, \text{last})$ is correct

Proof: We need to prove that $\forall n \geq 0$, $\text{BinarySearch}(L, x, \text{first}, \text{last})$ returns a range in sorted array L with $0 \leq \text{first} \leq \text{last} \leq |L|$ if value x is in sorted array L .

Basis step: At $n=0$ step, sorted array L contains a range from 0 to $|L|-1$. BinarySearch($L, x, 0, |L|-1$) returns the range $[0, |L|-1] \subseteq [0, |L|-1]$ for searching x . Clearly, value x is in the range $[0, |L|-1]$ of L , $L[0] \leq x \leq L[|L|-1]$.

Inductive step: Suppose that at $n \geq 0$ step and BinarySearch($L, x, first_n, last_n$) returns the range $[first_n, last_n] \subseteq [0, |L|-1]$ for searching x , $L[first_n] \leq x \leq L[last_n]$.

We need to prove that at $n+1$ step BinarySearch($L, x, first_{(n+1)}, last_{(n+1)}$) must return the range $[first_{(n+1)}, last_{(n+1)}] \subseteq [0, |L|-1]$ for searching x , $L[first_{(n+1)}] \leq x \leq L[last_{(n+1)}]$.

BinarySearch($L, x, first_{(n+1)}, last_{(n+1)}$) returns $[first_{(n+1)}, last_{(n+1)}]$:
(3.4.1) $L[first_n] \leq x \leq L[last_n]$ (hypothesis induction)

In case 1:

$$(3.4.2) \quad L\left[\left\lceil \frac{first_n + last_n}{2} \right\rceil\right] < x,$$

$$[first_{(n+1)}, last_{(n+1)}] = \left[\left\lceil \frac{first_n + last_n}{2} \right\rceil + 1, last_n \right].$$

$$(3.4.3) \quad \text{By (3.4.1), (3.4.2) } L\left[\left\lceil \frac{first_n + last_n}{2} \right\rceil + 1\right] \leq x \leq L[last_n],$$

$$L[first_{(n+1)}] \leq x \leq L[last_{(n+1)}].$$

In case 2:

$$(3.4.4) \quad L\left[\left\lceil \frac{first_n + last_n}{2} \right\rceil\right] > x,$$

$$[first_{(n+1)}, last_{(n+1)}] = \left[first_n, \left\lfloor \frac{first_n + last_n}{2} \right\rfloor - 1 \right].$$

$$(3.4.5) \quad \text{By (3.4.1), (3.4.4) } L[first_n] \leq x \leq L\left[\left\lfloor \frac{first_n + last_n}{2} \right\rfloor - 1\right],$$

$$L[first_{(n+1)}] \leq x \leq L[last_{(n+1)}].$$

From equalities (3.4.3), (3.4.5) we have the lemma proved. \square

Procedure TestIsomorphism($g \in C_k^j, C_k^i$)

Input: $g \in C_k^j, C_k^i$

Output: true or false

1. $b \leftarrow$ BinarySearch(code(CAM($g' \in C_k^i$)), code(CAM(g)), 0, $|C_k^i|$)
2. if ($b > 0$) return true
3. else return false

Lemma 3.5. The procedure TestIsomorphism($g \in C_k^j, C_k^i$) runs in time complexity $O(\log |C_k^i|)$.

Proof: This is evident by Lemma 3.3.

Lemma 3.6. The procedure $\text{TestIsomorphism}(g \in C_k^j, C_k^i)$ is correct.

Proof: This is evident by Lemma 3.4.

3.5. The algorithm

In PSI-CFSM algorithm, the first step constructs a sorted array, in the order of code of Canonical Adjacency Matrix (CAM) of subgraphs with two nodes (2-subgraph) or only one edge of graph G_i in graph database GD. This sorted array is denoted as C_2^i , and we denote $C_2 = \{C_2^i\}$. With each element u in C_2^i , we compare $\text{codeCAM}(u)$ with $\text{codeCAM}(v)$, $v \in \{C_2^j = C_2 - C_2^i\}$. If $\text{code}(\text{CAM}(u)) = \text{code}(\text{CAM}(v))$ then we increase the count support of u by 1. If $\text{sup}_u \geq \sigma$ then we put u into FS_2 , FS_2^i . FS_2 (FS_2^D) is the set of frequent 2-subgraphs of graph database GD and FS_2^i is the set of frequent 2-subgraphs of graph G_i in graph database GD. We construct a loop with $k \geq 3$ to compute C_k^i , FS_k , FS_k^i , CS_k , CS_k^i based on the PSI-CFSM algorithm.

Algorithm PSI-CFSM(GD, $\sigma = \text{min_sup}$)

Input: graph database GD, $\sigma = \text{min_sup}$

Output: $\text{CS}_2, \text{CS}_3, \dots, \text{CS}_k$, closed frequent subgraph sets corresponding level

Step 1. Building ordered array according to $\text{code}(\text{CAM})$ of C_2^i

Step 2. for each $u \in C_2^i$

Step 3. $\text{TestIsomorphism}(u, C_2^j)$ and find $\text{sup}_u \geq \sigma$ to put u into $\text{FS}_2^i, \text{FS}_2^D, \text{CS}_2^i, \text{CS}_2$

Step 4. while ($\forall i$: Combine ($\text{CS}_{k-1}^i, \text{FS}_2^i$) is not null)

Step 5. Build ordered array according to $\text{code}(\text{CAM})$ of C_k^i

Step 6. for each $u \in C_k^i$

Step 7. $\text{TestIsomorphism}(u, C_k^j)$ and find $\text{sup}_u \geq \sigma$ to put u into $\text{FS}_k^i, \text{FS}_k^D, \text{CS}_k^i, \text{CS}_k$

Step 8. Test $v \in \text{CS}_{k-1}^i$ if $\text{sup}_v = \text{sup}_u$ then remove v out $\text{CS}_{k-1}^i, \text{CS}_{k-1}$

Step 9. $k \leftarrow k + 1$

Lemma 3.7. The algorithm PSI-CFSM is correct.

Proof: By induction on $k \geq 2$, we show that the set FS_k computed by the algorithm coincides with the set of frequent k -subgraphs. At basic step, the induction is initialized ($k=2$, corresponds to FS_2 , set of all frequent subgraphs with 2 vertex or one edge) that is easily tested. Let's assume that at step k FS_{k-1} is the set of frequent subgraphs with $k-1$ vertices, and FS_{k-1} coincides with a set of frequent subgraphs of size $k-1$. The maximum number of edges of subgraph with $k-1$ vertices is $(k-1)(k-2)/2$. We need to prove that FS_k coincides with the set of

frequent subgraphs of size k . For the inductive step, it is sufficient to prove that given an arbitrary frequent subgraphs X of size k , X is surely included in the set FS_k . Thus, X is also a member of set FS_k^i of k -subgraphs of one graph g_i in graph database GD. FS_k^i is obtained by pruning step that removes candidate k -subgraphs $r \in C_k^i \wedge \sup_r < \sigma$ where C_k^i is output of $\text{Combine}(\text{CS}_{k-1}^i, \text{FS}_2^i)$. Then $\text{Combine}(\text{CS}_{k-1}^i, \text{FS}_2^i)$ is correct by Lemma 3.1. Hence, FS_k contains X and coincides with the set of frequent subgraphs of size k .

We now consider the time complexity of PSI-CFSM algorithm. We suppose that the cardinality of graph database GD is n and each computation step is constant 1. At line 1, the number of computation steps is cardinality of all edges of

all graphs in graph database GD $\left(\sum_{g_i \in \text{GD}} |E_{g_i}| \right) \times |\text{CS}_{k-1}^i| \times (|V_{g_i}| - (k-1)) \times (k-1)$. At

line 2 and 3, the number of steps is $\left(\sum_{g_i \in \text{GD}} |E_{g_i}| \right) \times \left(\sum_{g_i \in \text{GD}} \log_2 |E_{g_i}| \right)$. From line 4 to

line 9 is a loop which has k steps. Line 4 runs the procedure Combine that has $|\text{CS}_{k-1}^i| \times (|V_{g_i}| - (k-1)) \times (k-1)$ steps. Thus, at line 5 the number of computation

steps is $|\text{CS}_{k-1}^i| \times (|V_{g_i}| - (k-1)) \times (k-1) \times \left(\sum_{g_i \in \text{GD}} |C_k^i| \right)$. In the similar way

2-subgraphs computation is initialized; at line 5, 6, 7, 8 and 9 the number of computation steps is

$$|\text{CS}_{k-1}^i| \times (|V_{g_i}| - (k-1)) \times (k-1) \times \left(\sum_{g_i \in \text{GD}} |C_k^i| \right) \times \left(\sum_{g_i \in \text{GD}} \log_2 |C_k^i| \right).$$

Assume that every graph g_i in graph database GD has $|V_{g_i}| \geq 3$ then the number of computation steps from line 1 to line 3 is too small in comparing from 4 to 9. Hence, the total number of computation steps in time complexity of the PSI-CFSM algorithm is

$$\sum_{k=1}^{\max(|V_{g_i}|)} |\text{CS}_{k-1}^i| \times (|V_{g_i}| - (k-1)) \times (k-1) \times \left(\sum_{g_i \in \text{GD}} |C_k^i| \right) \times \left(\sum_{g_i \in \text{GD}} \log_2 |C_k^i| \right).$$

4. Conclusion

In this paper, we introduce an efficient method to reduce subgraph isomorphism testing process. The proposed method obtains polynomial time complexity in closed frequent subgraph mining. The subgraph isomorphism problem has time complexity in NP class in the current state-of-the-art frequent subgraph mining algorithms. To obtain polynomial time complexity in subgraph isomorphism, we use binary search to find string code of unique representation of a k -subgraph in an ordered unique string code array of a k -subgraph set in a random access machine model. In future,

we will continue this study by applying parallel mining to increase the effectiveness and efficiency of frequent subgraph mining.

References

1. Demetrovics, J., V. D. Thi, N. L. Giang. On Finding All Reducts of Consistent Decision Tables. – *Cybernetics and Information Technologies*, Vol. **14**, 2014, No 4, pp. 3-10.
2. Eppstein, D. Subgraph Isomorphism in Planar Graphs and Related Problems. – In *SODA*, Vol. **95**, 1995, pp. 632-640.
3. Garey, M. R., D. S. Johnson. *Computers and Intractability: An Introduction to the Theory of Np-Completeness*. San Francisco, 1979.
4. Han, J., H. Cheng, D. Xin, X. Yan. Frequent Pattern Mining: Current Status and Future Directions. – *Data Mining and Knowledge Discovery*, Vol. **15**, 2007, No 1, pp. 55-86.
5. Hopcroft, J. E., R. E. Tarjan. Isomorphism of Planar Graphs. – In: *Complexity of Computer Computations*. Springer, 1972, pp. 131-152.
6. Huan, J., W. Wang, J. Prins. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. – In: *3rd IEEE International Conference on Data Mining, 2003 (ICDM'2003)*, IEEE, 2003, pp. 549-552.
7. Huan, J., W. Wang, J. Prins, J. Yang. Spin: Mining Maximal Frequent Subgraphs from Graph Databases. – In: *Proc. of 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2004, pp. 581-586.
8. Inokuchi, A., T. Washio, H. Motoda. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. – In: *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, 2000, pp. 13-23.
9. Kuramochi M., G. Karypis. Frequent Subgraph Discovery. – In: *Proc. of IEEE International Conference on Data Mining, 2001. ICDM 2001*, IEEE, 2001, pp. 313-320.
10. McKay, B. D., et al. *Practical Graph Isomorphism*. Department of Computer Science, Vanderbilt University Tennessee, US, 1981.
11. Read, R. C., D. G. Corneil. The Graph Isomorphism Disease. – *Journal of Graph Theory*, Vol. **1**, 1977, No 4, pp. 339-363.
12. Savage, J. E. *Models of Computation. Exploring the Power of Computing*, 1998.
13. Thi, V. D., N. L. Giang. A Method to Construct a Decision Table from a Relation Scheme. – *Cybernetics and Information Technologies*, Vol. **11**, 2011, No 3, pp. 32-41.
14. Thomas, L. T., S. R. Valluri, K. Karlapalem. Margin: Maximal Frequent Subgraph Mining. – *ACM Transactions on Knowledge Discovery from Data (TKDD)*, Vol. **4**, 2010, No 3, pp. 10.
15. Ullmann, J. R. An Algorithm for Subgraph Isomorphism. – *Journal of the ACM (JACM)*, Vol. **23**, 1976, No 1, pp. 31-42.
16. Washio, T., H. Motoda. State of the Art of Graph-Based Data Mining. – *ACM Sigkdd Explorations Newsletter*, Vol. **5**, 2003, No 1, pp. 59-68.
17. Yan, X., J. Han. GSPAN: Graph-Based Substructure Pattern Mining. – In: *Proc. of 2002 IEEE International Conference on Data Mining, 2002. ICDM 2003*, IEEE, 2002, pp. 721-724.
18. Yan, X., J. Han. Closegraph: Mining Closed Frequent Graph Patterns. – In: *Proc. of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2003, pp. 286-295.