Finding Frequent Subgraphs in a Single Graph based on Symmetry

D. Kavitha
Sr Assistant Professor,
PVPSIT, Vijayawada,
Andhra Pradesh, INDIA

V. Kamakshi Prasad
Professor & Head of the
Department of Computer Science
JNTUH College of Engineering,
Hyderabad

J. V. R. Murthy
Professor
Dept of Computer Science
JNTUK, Kakinada

ABSTRACT
Mining frequent subgraphs is a basic activity that plays an important role in mining graph data. In this paper an algorithm is proposed to find frequent subgraphs in a single large graph that has applications such as protein interactions, social networks, web interactions. One of the key operations required by any frequent subgraph discovery algorithm is to perform graph isomorphism. The proposed algorithm offers mining frequent subgraphs by avoiding the subgraph isomorphism problem through exploiting the symmetry properties present in the given graph.

General Terms
Graph mining, frequent subgraph mining, data mining

Keywords
Frequent subgraph, single graph, graph isomorphism, symmetry

1. INTRODUCTION
Over the years, frequent subgraph discovery algorithms have been used to find interesting patterns in various application areas[1-7]. The frequent subgraph discovery problem can be defined as the process of finding subgraphs from a single large graph or from a set of graphs in a graph database which have frequency greater than the specified threshold. Developing algorithms that discover all frequently occurring subgraphs in a large graph is particularly challenging and computationally intensive, as candidate generation and subgraph isomorphisms play a key role throughout the computations. In recent years, several algorithms have been developed to find patterns in a large graph [8-14]. These algorithms start with a frequent edge or vertex and extend the graph by adding a new edge or by adding a new vertex to the existing graph recursively. These algorithms are complete in the sense that they are guaranteed to discover all frequent subgraphs and were shown to scale to very large graphs. The difficulty in frequent subgraph mining algorithms is to calculate the support of subgraphs and to prune redundant candidate generation arises from the subgraph isomorphism which is itself NP-hard problem[15-17].

In this paper, a new algorithm is proposed FSSG (Finding **F**requent **S**ubgraphs in a **S**ingle **G**raph based on Symmetry), which generates frequent subgraphs by avoiding the subgraph isomorphism cost in the mining process. The algorithm uses symmetries present in a graph to generate candidate subgraphs. Unlike most algorithms, this algorithm extends subgraph by adding another frequent subgraph determined by the symmetry mapping of subgraph.

2. PRELIMINARIES
Definition1: A labelled graph is defined as $G = (V, E, L, l)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, L is a set of labels, and l is a function that gives a unique label to each vertex of G.

Definition2: Frequent Subgraph: Given a graph G and a frequency threshold T, a subgraph G_i, with an observed support f is frequent if and only if $f \geq T$

Definition3: Partitioning: The vertices of a labelled graph G are partitioned into p disjoint non empty sub sets denoted by V, an ordered partition $V(G) = \{V_1, V_2, \ldots, V_p\}$ if, satisfies the following properties

(1) For all vertex $u, v \in V_k(G)$, $deg(u) = deg(v)$, $\forall u, v \in V_k(G)$, $1 \leq k \leq p$.

(2) For all vertex $u, v \in V_k(G)$, $lbl(u) = lbl(v)$, $\forall u, v \in V_k(G)$, $1 \leq k \leq p$.

(3) For all vertex u ∈ Vk(G) and v ∈ Vl(G), deg (u) >deg (v) if k< l.

Definition 4: The symmetry group of G denoted by $Sym(G)$ is a set of symmetries of G forms a group under functional composition. An automorphism (a symmetry) of graph G is a permutation of G's vertices that preserves G's edge relation, i.e., $G = G$. The *automorphism group Aut(G)* of a graph G is the set of all automorphisms of G with permutation composition as group operation.

Lemma 1: A graph G is vertex-transitive if for every vertex pair $u,v \in V_G$ there is an automorphism that maps u to v.

Lemma 2: A graph G is edge-transitive if for every edge pair $d,e \in E_G$, there is an automorphism that maps d to e.

Vertex orbits are the equivalence classes of the vertices of a graph G identified by the automorphism. The equivalence classes of the edges are called edge orbits. All vertices in the same orbit have the exact same degree, label and neighbours. All edges in the same orbit have the same pair of degrees at their endpoints.

A set of adjacent vertices of a vertex v of a graph g is denoted as $adj(v)$.

Definition 5: Vertex *sign*: For a vertex v with m adjacent vertices $u_1, u_2, \ldots, u_m$, vertex sign $sig(v)$is a string defined as: $dsig_v + lsig_v + esig_v$where "+" is string concatenation, and which satisfies the following criteria:

(1) For each $v \in S$, $dsig_v = (d(u_1), d(u_2), \ldots d(u_m))$ where $deg(u_k) \geq deg(u_{(k+1)})$, for all k, $1 \leq k \leq m - 1$
(2) For each $v \in S$, $lsig_v = (l(u_1) || l(u_2) || \ldots || l(u_m))$.,where u_i is the in the order of $dsig_v$

(3) For each $v \in S$, a sequence $esig_v=(l(u_1, v)|| l(u_2, v)||... ||l(u_m, v))$ where u_i is the in the order of $dsig_v$

Therefore, the sign of vertex v with m adjacent vertices is composed in the order of adjacent vertices degree ($dsig_v$), adjacent vertices label ($lsig_v$) and adjacent vertices edge label ($esig_v$). These are ordered by vertices degree first, then by vertices labels and then by edge labels. Notice that the vertex sign is not obtained by calculating all possible permutations; it is attained by sorting the degrees of adjacent vertices as described in the above definition 5 rule1. And then adjacent vertices label and adjacent vertices edge label are obtained just by concatenating the labels in the order of vertices obtained with Rule 1.The intention behind the design of vertex sign is to combine the degree, label and the edge label of adjacent vertices information of a vertex into a string-based representation. Therefore, the lexicographic order of vertex sign is totally ordered as normal strings.

Lemma 1. Given two vertex signatures $sig(v_1)$ and $sig(v_2)$ of vertices v_1 and $v_2$of a labelled graph G, respectively, v_1 is not symmetric to v_2 if $sig(v_1) \neq sig(v_2)$.

Proof: This lemma was proved by showing that if $v_1$is symmetric to v_2 then $sig(v_1) = sig(v_2)$. Since, $v_1 \cong v_2$, there exists a bijection f that maps each adjacent vertex v_i of v_1 to a vertex $f(v_i)$, namely w_i, of v_2. Otherwise the adjacency relations preservations such as adjacent vertex degrees, vertex labels and edge labels must be violated between these two vertices. Since the lexicographic order of adjacent vertices relationships are totally ordered, and the signature of a vertex is the string obtained by concatenation, these two vertex signatures must be equal. Notice that Lemma 1states a necessary condition to identify symmetry between two vertices.

As these vertices are symmetrical, the subgraph of a vertex with its adjacent vertices is also symmetrical. This symmetry property of vertices can be used to enumerate subgraphs just by connecting an edge between subgraphs of vertices.

3. THE APPROACH

The proposed FSSG -Finding **F**requent **S**ubgraphs in a **S**ingle **G**raph based on Symmetry- algorithm presents a novel technique that addresses the frequent subgraph mining problem without subgraph isomorphism checking. It employs a novel method to enumerate candidate subgraphs. Able to mine in large graphs and at low frequency thresholds.

3.1 Mining Frequent Subgraphs

The overall flow of finding frequent subgraphs algorithm is as follows: The algorithm starts by partitioning the graph G into p parts according to the definition 3 so that the vertices in each part contain same degree and same label. Compute the vertex signature for each vertex of V_i if the size of the part is greater than the frequency threshold. These parts are further refined into classes based on the definition 4. Now the vertices in a class become the starting subgraphs. These subgraphs are further extended by checking the symmetry of their children.

3.2 Storage Structure of a Graph

Graph representation is an important aspect as it has direct and significant influence on memory usage and execution time of the algorithm. The most commonly used representations of graph are Adjacency Matrix, Adjacency List, Hash Tables, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. A more space-efficient way to implement a sparsely connected graph is to use an adjacency list. An adjacency list representation for a graph associates each vertex in the graph with the collection of its neighbouring vertices or edges.

3.2.1Adjacency List

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a labelled graph. The labels of edges can be stored in nodes of linked lists. This algorithm uses an ordered adjacency list representation to store graph. Adjacency list representation of the graph G(figure1) is shown in figure 2. The adjacent vertices of a vertex are ordered by the definition 3 when constructing the adjacency list itself making it computationally efficient.

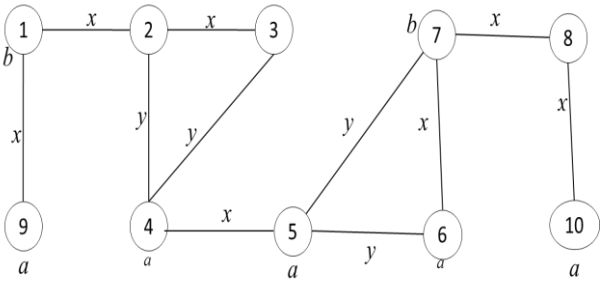

Fig1: Graph G

```
1(1)b 2 --> 2 b 3 x --> 9 a 1 x
2(2)b 3 --> 4 a 3 y --> 3 a 2 x --> 1 b 2 x
3(3)a 2 --> 4 a 3 y --> 2 b 3 x
4(4)a 3 --> 5 a 3 x --> 2 b 3 y --> 3 a 2 y
5(5)a 3 --> 4 a 3 x --> 7 b 3 y --> 6 a 2 y
6(6)a 2 --> 5 a 3 y --> 7 b 3 x
7(7)b 3 --> 5 a 3 y --> 6 a 2 x --> 8 b 2 x
8(8)b 2 --> 7 b 3 x --> 10 a 1 x
9(9)a 1 --> 1 b 2 x
10(10)a 1 --> 8 b 2 x
```

Fig 2: Adjacency list representation

3.3 Algorithm: Frequent Subgraph Mining

Input: A graph G and frequency threshold f

Output: The frequent subgraphs F of G

Begin

1. $S \leftarrow$ *makepartition()*
2. for each vertex orbit $S_k \in S$ do
3. if $|S_k| > f$ then
4. *result* $\leftarrow S_k$
5. *result* $\leftarrow$ *result U subgextnsn*(S_k)

End

Frequent subgraph mining starts by partitioning the vertices of graph G into vertex orbits by executing *makepartition()* algorithm. This algorithm takes set of vertices V of graph G as input and makes partitioning based on the signature of vertices that creates vertex orbits which have vertex transitive property. This algorithm is further explained. *Subgextnsn()* algorithm is executed for a vertex orbit S_k that supports frequency threshold to enumerate frequent subgraphs emerge from that orbit. Finally subgraph extension algorithm *Subgextnsn()* is recursively executed to find all the frequent subgraphs.

3.4 Making Partitioning

Algorithm: *makepartition()*
Input: A set of vertices *V* of graph G

Output: Refined partition *S*

Begin

1. Let $V \leftarrow V_1, V_2 .. V_n$ be the initial partition according to definition 3
2. repeat
3. for each cell $V_i \in V$ do s.t $|V_i| > f$
4. for each $v \in V_i$ do
5. compute sig_v
6. let V_{ik} be the sub partition of V_i
7. refine V_i into $V_i \leftarrow V_{i1}, V_{i2} .. V_{ij}$ s.t $\forall u, v \in V_i$ $sig(u) = sig(v)$
8. for each V_{ij} do s.t $|V_{ij}| > f$
9. $S \leftarrow S \cup V_{ij}$

End

Initially, the algorithm starts by forming an equitable ordered partition of vertices according to the definition 3, thereby extracting all the label and degree information. In order to find symmetrical vertices other graph theoretical information such as degree of adjacent vertices and labels of adjacent vertices and edges are exploited. For each vertex *v* in the cell V_i signature is calculated. These vertices are then split into groups of equal signatures, forming vertex orbits V_{ij}. The process is iterated for each cell V_i of the initial partition *V*. *Makepartition()* algorithm eliminates the cells that do not support the frequency threshold since according to the anti-monotone property, their extensions are also infrequent. Finally, the partition *S*, returned by this refinement procedure contains the resultant vertex orbits that will become basic subgraphs and can be further extendible. At this position a subgraph of a vertex i.e a vertex along with its adjacent vertices in each orbit will return a kind of subgraph.

3.5 Enumeration of Subgraphs

Algorithm: *subgextnsn(S_i)*

Input: A refined partition set S_i

Output: The frequent subgraphs of *G*

1. while $(S_i \neq \emptyset)$
2. for each vertex orbit S_i do
3. repeat
4. $ext \leftarrow sg(v)$ //a subgraph of vertex *v* in the vertex orbit S_i
5. for each vertex $v \in S_i$ do
6. let *adj(v)*and *adj(u)* are set of adjacent vertices of *u*, $v \in S_i$
7. compute sig $_{adj(vi)}$ for the Ith adjacent vertex
8. $ext \leftarrow ext <> sg(v)$ if $sig_{adj(vi)} = sig_{adj(ui)}$.
9. mark vertex *v and u visited*
10. until no more vertices to visit
11. return *ext*

Subgraph enumeration is the one of the important aspect in the process of frequent subgraph discovery. The mechanism used in this algorithm is a new approach. In this approach, the subgraph enumeration is carried out by extending the previously enumerated subgraph with newly identified extendible frequent subgraph. The input to this algorithm set S_i contains only the vertices which are symmetrical according to the definition 6 and frequent. So each vertex *v* of the set S_i (i.e the element of frequent set) can be associated to a small subgraph with its adjacent vertices. To extend the subgraph, now check the signature of its first adjacent vertex with other vertices first adjacent vertex. If they are same then connect the vertex *v* to the subgraph of first adjacent vertex v_1. Repeat this procedure until no more matching's found or all the vertices are visited. Execute this procedure recursively for the next level adjacent vertices also if the extensions are possible to further extend the subgraphs. This algorithm will return the frequent subgraphs that can be enumerated from each vertex orbit.

4. EXPERIMENTAL EVALUATION

Experiments are carried out on a Intel® Pentium® Dual CPU T3400 @2.17 GHz with 4GB RAM. To evaluate the performance of the algorithm, the following real graph data sets are used.

Aviation (ailab.wsu.edu/subdue). This dataset contains a list of records extracted from the aviation safety reporting system database [18]. Each record corresponds to an event and nodes represents the events(and are labelled with the ids of the event) while the information regarding event. Aviation consists of 100K nodes and 133K edges. The Aviation graph has on average one edge per node, thus, it is very sparse.

protein-protein interaction data of Saccharomyces cerevisiae obtained by Database of Interacting Protein (DIP) [19] for the experimental analysis. They provide a set of data that is experimentally determined protein interaction. It contains 1274 protein nodes and 3222 interactions between proteins.

The efficiency of the algorithm against frequency and time is shown in figure 3. It can be observed that the number of frequent subgraphs grows exponentially as the threshold decreases. So that the time required to execute also increases. The algorithm shows linear increase in execution time against threshold as it identifies the frequency at the time of partitioning.

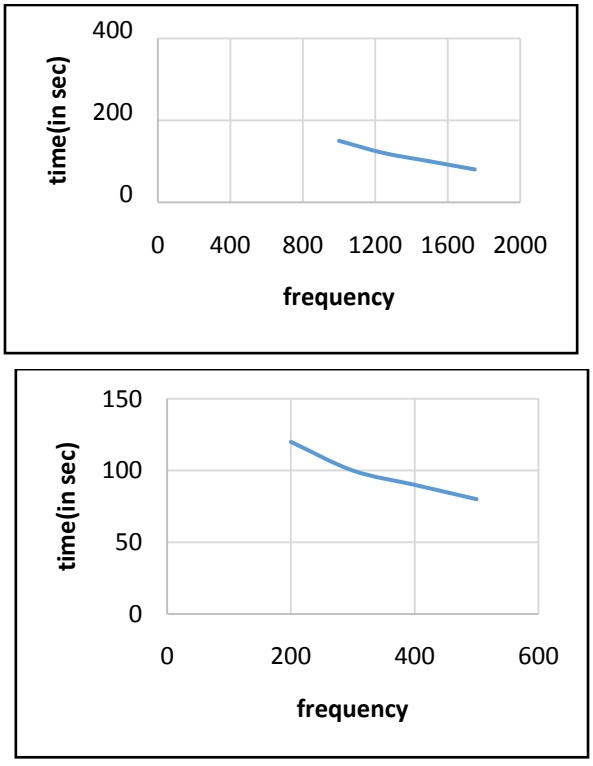

Fig 3: Performance evaluation for aviation data and protein interaction

5. CONCLUSION

The emphasis of proposed algorithm is to provide an efficient and fast computational approach to mine frequent subgraphs in a large graph without isomorphism testing by exploiting the symmetries present in a graph. The proposed partitioning based on signature identifies the symmetries present in a graph and make available to enumerate frequent subgraphs faster than existing techniques. The results obtained in the preliminary tests confirmed the effectiveness of the proposed algorithm. The proposed algorithm can also be applied to search structures in graph data sets for general applications. The algorithm can also be extended to find frequent subgraphs for unlabelled graphs.

6. REFERENCES

[1] D. Cook, L. Holder, Mining Graph Data, John Wiley & Sons Inc, 2007.

[2] R. Kumar, P Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, E. Upfal. The Web as a Graph. ACM PODS Conference, 2000.

[3] D. Maio and D. Maltoni. A structural approach to fingerprint classification. In Proceedings of 13th International Conference on Pattern Recognition, Vienna, Austria, pp. 578–585, 1996

[4] F. Eichinger, K. Bohm, M. Huber. Improved Software Fault Detection with Graph Mining. Workshop on Mining and Learning with Graphs, 2008

[5] M. S. Gupta, A. Pathak, S. Chakrabarti. Fast algorithms for top-k personalized pagerank queries. WWW Conference, 2008

[6] M. Koyuturk, A. Grama, W. Szpankowski. An Efficient Algorithm for Detecting Frequent subgraphs in Biological Networks. Bioinformatics, 20:I200–207, 2004.

[7] B. Zhou, J. Pei. Preserving Privacy in Social Networks Against Neighborhood Attacks. ICDE Conference, pp. 506-515, 2008

[8] Kuramochi, M. and Karypis, G., Finding frequent patterns in a large sparse graph. *Data Min. Knowledge Discovery*, 2005, **11**(3),243–271

[9] L. Zou, L. Chen, and M. T. ¨Ozsu. Distance-join: pattern match query in a large graph database. *PVLDB*, 2(1):886–897, 2009.

[10] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, May 2012.

[11] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *PVLDB*, 6(2):133–144, Dec. 2012

[12] Elseidy M., Abdelhamid E., Skiadopoulos S. and Kalnis P. (2014) GraMi: Frequent subgraph and pattern mining in a single large graph. PVLDB, 7,517–528.

[13] Fiedler M. and Borgelt C. (2007) Support Computation for Mining Frequent Subgraphs in a Single Graph. Proc. MLG 07, Firence, Italy, August 1–3. ACM, New York, NY, USA.

[14] Bringmann B. and Nijssen S. (2008) What is Frequent in a Single Graph?. Proc. PAKDD 08, Osaka, Japan, May 20–23, pp. 858–863. Springer, Berlin.

[15] McKay, B., *Practical Graph Isomorphism*, Citeseer, 1981.

[16] J.R. Ullmann, An algorithm for subgraph isomorphism, J. ACM 23 (1976) 31–42.

[17] J. Gross, J. Yellen (Eds.), Handbook of Graph Theory, CRC Press, Florida, 2004

[18] SUBDUE databases: http://ailab.wsu.edu/subdue/

[19] L. Salwinski, C.S. Miller, A.J. Smith, F.K. Pettit, J.U. Bowie and D. Eisenberg, The database of interacting proteins: 2004 update. Nucleic Acids Research. 2004, 32: D449-D451.