INCREASING THE EFFICIENCY OF THE SOFTWARE ARCHITECTURE RECOVERY THROUGH SPANNING TREE BASED MAXIMAL GRAPH MINING TECHNIQUE

Shaheda Akthar[1] , Sk.Md.Rafi [2]

1.Department of Computer science and Engineering,, Sri Mittapalli college of Engineering,A.P. INDIA.
2.Department of Computer science and Engineering,, Sri Mittapalli Institute of Technology for Women,A.P. INDIA.

Abstract This paper represents a technique for recovering the Software Architecture based on Graph Pattern Matching by the help of mining techniques. Generally Software Architecture is represented in terms of graphs with set of vertices and edges. Finding the frequent data sets is the major step in the software architecture recovery. Many algorithms are proposed in this context, for example Apriori based. In this paper to find the frequent data sets and a maximum association between the graphs we used an efficient algorithm called SPIN (Spanning tree based Maximal Graph Mining Technique). The results have shown that our proposed Graph mining technique is more efficient in recovering the Software Architecture.

1. Introduction

In the present Chapter, to increase the efficiency of the Software Architecture a sophisticated graph mining technique called spanning tree based maximal graph
is used. Query graph is first produced by making use of Architectural Query language [2, 8, 9, 10, 11]. A pattern graph is produced based on that query graph by using its Syntaxes and Semantics.

As the software grows the complexity also grows. Complexity of the big software system regarded as a major issue for the maintenance. Continuous usage of the software can make changes in its architecture. Reverse engineering process helps the project managers and code generators in understanding the software and components. Architecture recovery is one of the finest area of Software Engineering, generally it adopts the reverse engineering process. Software recovery starts with the legacy software and is interpreted through the graphs. Bulk of papers are presented in this context Many of them have used graph as the main source of interpretation and some of them used the cluster based technique. Finding the frequent sub-graphs in the given main graph[1] is one important point in the graph based architecture recovery. In this approach each component is represented as the vertex and the relation between them as the edge. Vast literature has shown that to find the frequent data sets from the main graph has used the Apriori based algorithm [1]. But there is a significant disadvantage with Apriori algorithm, first it needs the candidate generation step and secondly it creates the considerable overhead while joining two size-k and size-k+1 sub-graphs. In order to over come from the above two overheads, we used an algorithm SPIN (Spanning tree based maximal graph mining technique) based on the pattern growth approach to find maximum graphs with maximum association [3]. Maximal sub graph mining significantly reduces the total number of mined sub graphs and several pruning techniques can be efficiently integrated into the mining process and dramatically improve the performance of the mining algorithm. [12]

2. Pattern Growth Approach and Over view of Spanning tree based maximal graph mining

In this Section, once the graph is identified g^1, the recursion is continued until all the frequent graphs are identified. When there are no more graphs generated, the recursion stops.

2.1 Labeled Graph

A labeled graph G^1 is a graph where each node and edge has an associated label. We use V^1 and E^l to denote the set of node labels and edge labels respectively. Without loss of generality, we assume

a total order ≥ on V^1 and E^1. The labeling function θ defines the mappings $V \rightarrow V^1$ and $E \rightarrow E^1$
1. When we study recurring sub-graphs in graph databases, it is critical to know whether a graph occurs in another graph, as defined below

Definition: A labeled graph G^1 is **sub-graph isomorphic** to another graph G^{11}, if there exists an injection $f : V[G^1] \rightarrow V[G^{11}]$ such that
• $\forall u \in V[G^1]$, $(\theta(u) = \theta^1(f(u)))$,
• $\forall$ u, v ∈ V, $((u, v) \in E[G^1] \Rightarrow (f(u), f(v)) \in E[G^{11}])$, and
• $\forall (u, v) \in E[G^1]$, $(\theta(u, v) = \theta^1(f(u), f(v)))$.
Where $V[G^1]$ and $E[G^1]$ denote the node set and edge set of a graph G^1. The injection *s* is a *sub-graph isomorphism* from G^1 to G^{11}. By a slightly abused notation, we refer G^l as a "sub-graph" of G^{11}, denoted by $G^1 \subseteq G^{11}$ by omitting the word "isomorphic"; similarly G^{11} is referred to as a *super-graph* of G^1. A labeled graph G^l is defined to be *isomorphic* to another graph G^{11} if G^1 and G^{11} are mutually sub-graphs. Non- isomorphic sub-graph is referred to as a *proper sub-graph*, denoted by $G^1 \subseteq G^{11}$ and similarly G^{11} is referred to as a *proper super-graph* of G^l .Given a set G^1 of labeled graphs, the *support* of a graph G^1 is the fraction of graphs in G^1 in which G^1 occurs.

3. Maximal Sub-graph Mining

In the following discussion, we present a novel framework for mining maximal frequent sub-graphs. We show that we can unify tree mining and sub-graph mining into one process where we first find all frequent trees from a graph database and then construct frequent cyclic graphs from the mined trees. We developed two procedures to support the new framework. The first one is a graph partitioning method where we group all frequent sub-graphs into equivalence classes based on the spanning trees they contain. Primarily, tree related operations, such as isomorphism, normalization, and testing whether a tree is a sub-tree of another tree, or asymptotically simpler than the related operations for graphs, which are NP complete. Second, in certain applications such as chemical structure mining, most of the frequent sub-graphs are really trees. Last but not least, this framework adapts well to *maximal* frequent sub graph mining, which is the focus of this paper. Using a chemical structure benchmark, we show 99% of cyclic graph patterns and 60% of tree patterns can be eliminated by our optimization technique in searching for maximal sub-graphs [35].

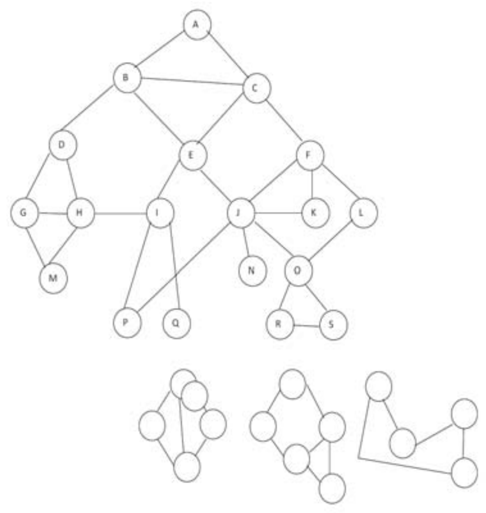

Fig:1 Source Graph and related Sub Graphs

4. Canonical Spanning Tree of a Graph

We define a *sub-tree* of an undirected graph G^l as an acyclic connected sub-graph of G^l. A sub-tree T^l is a *spanning tree* of G^l if T^l contains all nodes in G^l. There are many spanning trees for a given graph G^l. We define the maximal one according to a total order defined on trees as the *canonical spanning tree* of G^l, denoted by T^l (G^l).First picking up maximal labeled nodes in G^l as a group of single node trees. It iteratively grows those trees by attaching an additional node to each of them in all possible ways. The outcomes of these sub-trees are properly checked and sub-tree with maximum peer value taken for next iteration. This procedure is guaranteed to converge to the canonical spanning tree of a graph. Since every tree is a graph, the procedure can be applied to obtain canonical representations of trees.

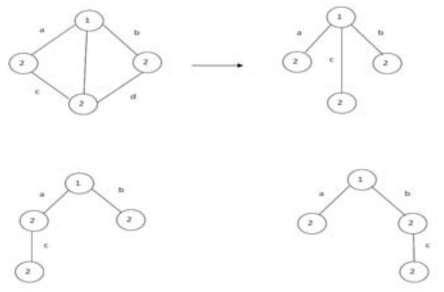

Fig:2 All Spanning tree of graph G with Canonical tree

5. Tree-based Equivalence Class

In this section based on the canonical spanning tree we introduced a graph partitioning, and outline a new frequent sub-graph search algorithm based on the graph partitioning method discussed herein before with two steps: (1) mine all the frequent trees from a graph database and (2) for each such frequent tree *T*, find all frequent sub-graphs whose canonical spanning trees are isomorphic to *T*. Maximal frequent sub-graphs can be found subsequently among frequent ones.

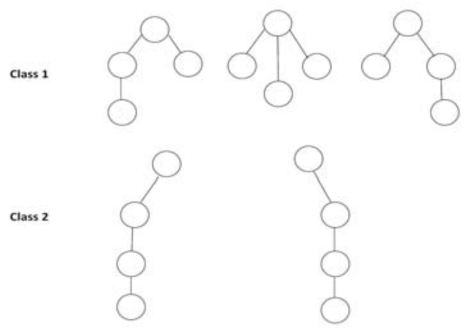

Fig 3 Divide the above trees into equivalent Classes

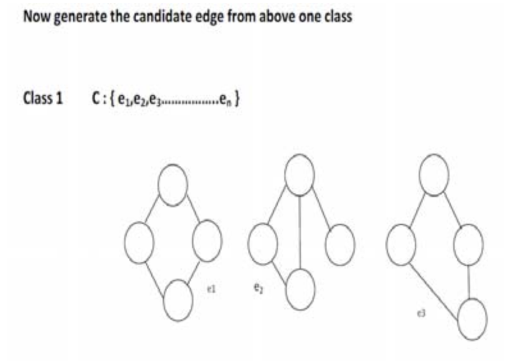
Now generate the candidate edge from above one class
Class 1
C : { e1,e2,e3.................en }
e1
e2
e3

Fig 4 Convert all trees of Class 1 into Graphs

6. Source Graph Decomposition

6.1 Decomposed Source Graph

From the above procedure we decompose a source graph into sub graphs of most associate relationship, generally called offline phase. For example $G_s = \{V_s, E_s\}$ be the source graph which has been divided into the number of frequent graphs with maximum association. Let the sub-graphs be $\sum G_s^{\,i}$ denotes the graphs of most frequent datasets. Based on the above graph mining, find the sub-graphs with same canonical form can be under one class. Let ∑ C_i be the set of classes of graphs which are obtained from the domain knowledge, system documents and other analysis tools, and is done under the online phase. Here we call $G_{C_i}^{g}$ be the sub-graph belong to the class.

7. Architecture Query Language (AQL)

A query containing a number of abstract components and connectors where each component imports a place holder and exports a matched place holder if any is known as an AQL [4, 5, 6, 7].

Abstract Component: A collection of place holder. Here a place holder is one which retrieves the searching mechanism is called Abstract component.

Abstract Connector: A connection among the abstract components is called an Abstract connector.

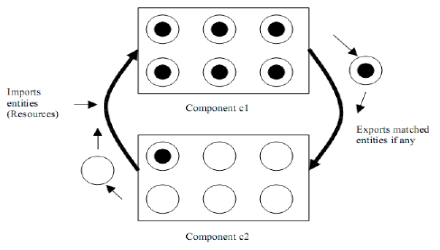

Fig 5. The notations of abstract component and abstract connector in AQL,

7.1. Query Generation

By applying any one of the following methods, we can generate an initial Pattern.

(i) Comparing the source code of the system with its architecture.

(ii) Applying a clustering technique

iii) Using the available system architecture document

The objective of these methods is to extract small groups of system entities that represent the core functionality of the system modules. These groups are used to generate an initial Query graph of an AQL query [4, 5, 6, 7] Now, we got an idea of the AQL specification.

Now, the textual specification of an AQL query should be transformed into a graph representation. The groups of graphs that are generated from the query-graph during the iterative matching process are defined. The generated graphs are related by recursive graph algebraic equations.

Now the query is generated. Generate a pattern graph based on that Query. Therefore the textual specification of an AQL query should be transformed into a graph representation. To generate a graph pattern we have to get a basic idea of source graph and a source region. Let the graph obtained from the AQL be G_p^g. Here the query graph is iteratively processed to get more and more graph generations.

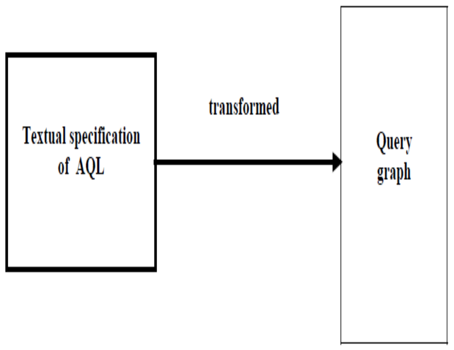

Fig 6. Transforming textual specification of AQL to a query graph

8. Graphing matching process

In this section we compare the sub-graph obtained from the section 3(source graph) with graph obtained from the AQL from the section 4. Here we used our previously proposed algorithm bipartite graph pattern matching. The experiments done on the C source code is observed that graph mining with pattern growth approach is more efficient than other.

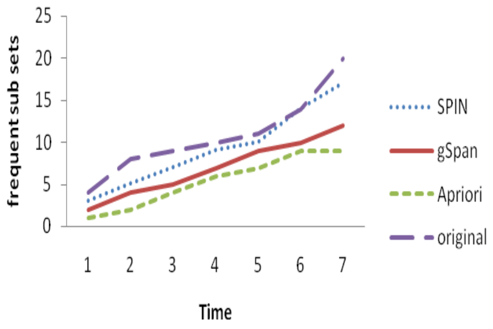

Fig7. Comparison Graph

Conclusion In this paper we used an graph mining technique to recover the Software Architecture. It is more efficient than Apriori algorithm. To avoid the overheads incurred in Apriori algorithm, we used a non Apriori based algorithm, SPIN to recover the Software Architecture and experimental results have shown that the recovered architecture from SPIN is more efficient than Apriori based.

References

[1] A.Inokuchi,T.Washio and H.Motoda. an Apori based algorithm for mining frequent substructures from graph data, In PKDD' 00,2000

[2] D.R. Harris, H. B. Reubenstein, and A. S. Yeh. Reverse engineering to the architectural level. In *Proceedings of the 17th ICSE,* pages 186–195, 1995.

[3] Jun Haun, Weiwang,Jan Prins,Jiong Yang. SPIN: Mining Maximal frequent Subgraphs from Graph Databases[35]

[4] Kamran Sartipi and Kostas Kontogiannis. A user-assisted approach to component clustering. *Accepted for the Journal of Software Maintenance: Research and Practice (JSM),* 2002.

[5] Kamran Sartipi and Kostas Kontogiannis. Interactive software architecture recovery: An incremental supervised clustering approach. Technical Report UWE&CE#2002-06, Dept. E&CE, University of Waterloo, Waterloo, Canada, April 2002.

[6] Kamran Sartipi, Kostas Kontogiannis, and Farhad Mavaddat. A pattern matching framework for software architecture recovery and restructuring. In *Proceedings of the IEEE IWPC,* pages 37–47, Limerick, Ireland, June 2000.

[7] Kostas Kontogiannis, R. DeMori, M. Bernstein, M. Galler, and E. Merlo. Pattern matching for design concept localization. In *Proceedings of the Working Conference on Reverse Engineering (WCRE'95),* pages 96–103, 1995.

[8] Mary Shaw and David Garlan. *Software Architecture.* Prentice-Hall, 1995.

[9] R. Fiutem, E. Merlo, G. Antoniol, and P. Tonella. Understanding the architecture of software systems. In *Proceedings of the 4th Workshop on Program Comprehension,* pages 187–196, 1996.

[10] Rick Kazman and Marcus Burth. Assessing architectural complexity. In *Proceedings of the CSMR,* pages 104–112, 1998.

[11] Rick Kazman and S. Jeromy Carriere. Playing detective: Reconstruction software architecture from available evidence. Technical Report CMU/SEI-97-TR-010, Carnegie Mellon University, 1997.

[12] SPIN: Mining Maximal Frequent Subgraphs from Graph Datanases by Jun Haun, Wei Wang, Jan Prins, Joing Yang

Shaheda Akthar received Bachelor of computer science & Master of Computer Science from Acharya Nagarjuna University, M.S (Software Systems) from BITS, Pilani, Pursuing PhD from Acharya Nagarjuna University. Presently working as Asscociate .Professor in Sri Mittapalli College of engineering, affiliated to J.N.T.U, Kakinada. My area of interest is Software Reliability, Software Architecture Recovery, Network Security, and Software Engineering

Sk.MD.Rafi received B.Tech (comp) from Jawaharlal Nehru Technological University, M.Tech (comp) from Acharya Nagarjuna University. Pursuing PhD from Jawaharlal Nehru Technological University. Presently working as Associate. Professor in Sri Mittapalli Institute of Technology for women, affiliated to J.N.T.U, Kakinada. My area of interest is Software Reliability, Software Architecture Recovery, Network Security, and Software Engineering.