CONCLUSION AND FUTURE WORK

8.1 Summing up

The main aim of this thesis namely, increasing the efficiency of software architecture recovery by graph mining techniques has been achieved in the preceding chapters. These software architecture recovery concepts have been developed for the first time in the field of software architecture, reverse engineering and many other areas with pattern matching.

In the third chapter I have used the Bipartite graph in minimizing the complexity involved in Software Architecture recovery. There is A* algorithm for solving the problem. But this algorithm suffers from exponential time complexity because it uses the Breadth First Search method for the temporary storage. In this, our pattern matches in the nth phase, we backtracking to the root n times in this algorithm. So its time complexity increases by exponential order. It is very useful only at very first phase, if the pattern matches then this algorithm is best suited for Software Architecture Recovery So, to change this complexity to linear, I present a Bipartite algorithm. Here, we use Depth First Search algorithm to evaluate minimum distance. There is no backtracking, so time complexity is O (n).

In this way this algorithm runs in linear time which improves the performance in terms of iterative when compared to A* algorithm.

In the fourth chapter I used Spanning Tree based maximal Graph mining technique. Most of the Architecture Recovery techniques in the previous papers,
they have used Apriori algorithm to find the maximal association between the graphs. It was proved that Apriori algorithm as a drawback in two areas.

(i) Competition overhead

(ii) Candidate Step

In order to overcome, I used Spanning Tree based Graph mining technique (SPIN) based on pattern growth approach. The main concentration of SPIN algorithm is finding the isomorphic graphs based on Canonical graphs. Experimental results have been proved that this algorithm has been useful for finding maximal subsequent graphs and improves the recovery process.

In the fifth chapter, I used the error tolerance and penalty cost in graph mining to recover the Software Architecture. Generally the graph matching is not complete in its context. For that Architecture which is recovered through graph matching will not be efficient one. So for improving the matching process, I used error tolerance graph matching technique called Graph Isomorphism by Decision Tree. In this matching process is improved by finding the adjacency matrix, permutation matrix and a Decision Tree. Experimental results proved that Architecture recovered from above process is slightly improved.

The penultimate Chapter concentrates on improvements in clustering techniques. Several papers are proposed based on ordinary clustering techniques to recover the components present in the software, depends on Cohesion and Coupling. But it is observed that some of the components are not recovered during the recovery, because they are showing partial degree of membership in multiple nearby clusters. Because of this components are partly in one set and partly in another set. Architecture which is recovered from this ordinary clustering will not be complete. For that I am using a concept called Fuzzy
Clustering to consider the components whose degree of joining the cluster is measured and it makes them to join into any one of the cluster. The Software Architecture Recovery which is recovered from fuzzy clustering technique is more efficient than ordinary clustering.

In the last chapter for evaluating design patterns clustering methods and the concept of fielder vector are used. Software Architecture recovery includes the extraction of design patterns. Patterns may be found using many techniques such as fielder vectors, clustering methods, query languages etc. A Software system is comprised of modules (which includes procedures, files, functions etc.). First these modules should be classified into subsystems. To classify this, construct a graph $G = (V, E)$ such that each vertex is consisting of the modules and each edge represents the relationship between these modules. Then we decide to classify the nodes into subsets such that cohesion between the nodes of class is maximized and the coupling between the nodes of different classes is minimized.

With respect to graph matching there is exponential complexity, however I have proven the complexity is linear in certain situations not for all the problems. Due to this problem, to decompose the matching problem into subunits (smaller graphs). On this subunits investigate using edit distance method and use the Fiedler matrix for the partition of graph. This process may be a hierarchical framework which is suitable for parallel computation. Thus all the algorithms and representations presented in this thesis are efficient in terms of memory and time. However properties of graph matching helps to design more efficient algorithms for software architecture recovery.
8.2 Recommendations for future work

In this connection, it may not be out of order to make recommendations for further study and research in the field of software architecture recovery, reverse engineering & Software Engineering.

The source code of the system needs to be analyzed in every step. Particularly in this Thesis for analyzing system should be represented in the form of a tree using two models, one is AQL and another is Attributed Relational Graph (ARG). We can also use some other graph mining techniques for improving the graph matching process.

Secondly it has been detailed in the third chapter how to minimize the complexity of Software Architecture Recovery by using Bipartite Graph. The moot point is, Recovery is of immense importance it uses A* Algorithm which run in Exponential time. I proposed an algorithm which runs in linear time, which improves the performance in terms of iterations when compared to A* algorithm.

Thirdly, by using the SPIN mining technique we have recovered the associated graph very efficiently compared to Apriori algorithm. Mostly we reduced the time complexity in finding the most frequent sub graphs from given source graph. The efficiency of recovery can be improved further by using genetic algorithms and neural networks.

Next I have taken the measures in Graph matching by adopting new graph tolerance matching techniques by which error in the matching process is rectified to an extent. In our error tolerance algorithm suffers from space complexity. More efficient error tolerance matching techniques can be adopted for solving this draw back.
At last it is parallelize this operation and get the $O(\log(n)^2)$ and so the process would be fast even for large software and some patterns directory and sub graph patterns not be used in my concept, by using these, to get better result.