Computer and Information Science
www.ccsenet.org/cis
Vol. 3, No. 1
February 2010

Recovery of Software Architecture Using Partitioning Approach by Fiedler Vector and Clustering

Shaheda Akthar
Sri Mittapalli College of engineering
Affiliated to JNTU, Kakinada, India
E-mail: shaheda.akthar@yahoo.com

Sk.MD.Rafi
Sri Mittapalli Institute of Technology for Women
Affiliated to JNTU, Kakinada, India
E-mail: rafi_527@rediffmail.com

Abstract

Software Architecture Recovery includes the extraction of design patterns. Patterns may be found using many techniques such as fielder vectors, using clustering methods, query languages etc. In this chapter, for evaluating design patterns clustering methods and the general notion of fielder vector are used.

Keywords: Fiedler Vector, Laplacian matrices, Eigen Vector, Cohesion, Coupling

1. Introduction

A Software system is comprised of modules (B.W. Kernighan, S.Lin, 1970)(R.J. Wilson, J.J. Watkins, Graphs, 1990) (which includes procedures, files, functions etc.). In the beginning these modules should be classified into subsystems. For this purpose, construct a graph G= (V, E) such that each vertex is consists of the modules and each edge shows the relationship between these modules. After that we decide the classifications of the nodes into subsets, there by the cohesion between the nodes of class may be maximized and the coupling between the nodes of different classes be minimized.

2. Modules Identification

We cannot construct a graph, without identifying modules and relations. In this way the first step is to find the modules and its relations among them.

The following ways can be adopted to find modules

1) The easiest way is to treat each file as a module because the functions in the file are semantically related.

2) We can consider groups of files as a module. But here the question is which files should be grouped?

3) We can also consider procedure as a module, as we are following the easier approach i.e. considering each file as a module.

3. Relations

There will be three types of relations among files.

f1 Useproc f2: shows that there is one function in f1 which calls the other function in f2.

f1 Usevar f2:shows that there is one function in f1 which uses a variable defined in f2.

f1 Implementby f2:shows that there is one header function in f1 which is implemented in f2.

In this way, we can construct a graph, (B.W. Kernighan, S.Lin, 1970)(R.J. Wilson, J.J. Watkins, Graphs, 1990) using these modules and relations among modules. The next step is to classify these modules into subsystems. So as to divide the graph into sub graphs for using the concept of Fiedler vector. First this we have to know what a Fiedler Vector (M. Fiedler) is.

For the input graph G= (V, E)

1) calculate the Adjacency matrix

2) Calculate the Degree matrix

3) Calculate the Laplacian matrix

The adjacency matrix is

$$\mathbf{A(i,j)=1 \text{ if } (i,j) \in E}$$
$$\mathbf{=0 \text{ otherwise}}$$

The degree matrix of the graph is the diagonal matrix of the row sum of the adjacency matrix

$$\mathbf{D= diag(deg(i);\quad i \in V)}$$
$$\mathbf{Deg(i) = \sum_{j \in V} A(i,j)}$$

The Laplacian matrix L is the difference between diagonal matrix and Adjacency matrix (B. Mohar, 1997)

$$\mathbf{L=D-A}$$
$$\mathbf{L(i,j) = \sum_{(i,k)\in E} A(i,k) \quad \text{if } i=j}$$
$$\mathbf{= -A(i,j) \text{ if } i \neq j \text{ and } (i,j) \in E}$$
$$\mathbf{= 0 \text{ otherwise}}$$

Now the matrix available is symmetric. The Eigen vector $(1, 1 \ldots.1)^T$ corresponds to trivial zero, Eigen values. With these Eigen values now, we get the Fiedler vector. We have to arrange first these Eigen values in the ascending order. The largest Eigen value and the second smallest Eigen value, whose corresponding Eigen vector is referred to as the Fiedler vector (M. Fiedler, 1975).

In this way Fiedler vector is known to us. Now decompose the graph into sub graphs. In this, the path sequence for the nodes has to be calculated using a permutation π. The sequence is that the elements of the edge weight matrix W decrease as the path is traversed **if** $\boldsymbol{\pi(i) < \pi(j) < \pi(k)}$ **then** $\mathbf{W(i,j) > W(i,k)}$ **and** $\mathbf{W(j,k) > W(i,k)}$

Consider a vector $x^- =(x_1, x_2, \ldots\ldots.x_{|v|})$ of continuous variables x_i . Calculate the penalty function

$$\mathbf{g(x)^- = \sum_{i=1}^{|V|} \sum_{j=1}^{|v|} W(i,j)(x_i - x_j)^2.}$$

The constraints of this function are

$$\mathbf{\sum_{i=1}^{|V|} (x_i)^2=1 \text{ and } \sum_{i=1}^{|V|} x_i=0.}$$

4. Decomposing a graph into sub graphs

By making use of Fiedler vector concept, the graph has to be divided into sub graphs (R.J. Wilson, J.J. Watkins, 1990). The neighbourhood of the node i consist of its center node together with its immediate neighbours connected by edges in the graph.

$$\mathbf{N_i^{\wedge} = \{i\} \cup \{u ;(i,u) \in E\}}$$

Assign each node measure of significance as the center of the neighbourhood. After assigning, it traverses the path defined by the Fiedler vector. Select center nodes on the basis of this measure to traverse the path. Assign weights to nodes based on the rank-order in the permutation. The weight assigned to the node is i€ V is wi=**Rank ($\boldsymbol{\pi}$ (i)).** After assigning weight to each node calculate the score function. The significance of the node will be known after giving score to each node. Score can be calculated by using a function

$$\mathbf{S_i = C_1(deg(i)+|N_i \Lambda P|)+C_2/ W_i}$$

Where C_1 and C_2 are threshold values that were detected heuristically. P is the set of nodes on the perimeter of the graph. The first term depends on the degree of the node and its proximity to the perimeter. In this way the nodes will be sorted according to their distance from the perimeter. This is proposed as it is better to decompose first from the outermost layer. The second term says that the first ranked nodes in the Fiedler vector are visited first. To locate the non overlapping neighbourhoods of the graph G, we use the scoring function. We traverse this list until we find a node K which is neither in the perimeter and also the calculated score should not exceed of its neighbours. If this condition gets satisfied then the node K together with its neighbours represents the first sub graph. This process will be repeated for all the nodes that satisfies the condition. Then we have to find out the sub graphs which are overlapping with its neighborhood. By doing this step the sub graphs are found. Thus Input the overlapping sub graphs to a clustering algorithm. There are two major approaches for subsystem classification.

1) Top down approach

2) Bottom up approach.

In the Top down approach, creation of subsystem includes all modules and then iteratively decomposes the current subsystem to create them at lower levels. In the bottom up approach, consider each module as a subsystem and then iteratively merge them to create those at higher levels.

Top down approaches suffer from exponential complexity as in A* algorithm. So, follow the bottom up approach.

1) For clustering, calculate the similarity between two nodes.
2) Identify a set of nodes that are pair wise & most similar. After identifying, create a cluster by taking the union of the most similar cluster or creation of more than one cluster is also possible by taking the union of some of pairs of this set.

5. Similarity/Dissimilarity measures

Two nodes are said to be similar if they have either the highest similarity measure or lowest dissimilarity.

If the component of the system is entirely connected to just other component, that connection should be computed as a lower dissimilarity than any other connection that is not complete. It is based on the percentage degree of vertices & common neighbours of the two vertices. That is, let p be the dissimilarity matrix and is defined by

$p(i,j)=deg(i)+deg(j)-2*b(i,j)/deg(i)+deg(j)-b(i,j)$

where deg(i)is the degree of the vertex i in the graph and b(i,j) is the number of common neighbour of vertices i and j. since deg(i)+deg(j)-b(i,j) is the number of all vertices connected to exactly one of the i and j. note that if deg(i)=deg(j)=b(i,j) then p(i,j)=0 and so i and j are completely similar. Note also if i and j have no common neighbor then p(i,j)=1 and so i and j are completely dissimilar.

After clustering now there is a need to optimize the solution. Then consider the measurement of intra-connectivity and inter-connectivity.

5.1 Intra-connectivity

It is a measure of connectivity between the two components that are grouped together in the same cluster. The degree of intra-connectivity should be high for good sub system partitioning , because many software level features are shared by the modules grouped with in a common subsystem.

$$\mathbf{A_i = m_i/N_i^2}$$

Where Ai is the intra-connectivity measurement
N_i is number of components
m_i intra-edge dependencies.

5.2 Inter-connectivity

It is a measure of connectivity between two distinct clusters. Inter connectivity should be very less. It is denoted by E_{ij}

$$\mathbf{E_{ij} = m_{ij}/2N_iN_j \quad \text{if } i\neq j}$$
$$\mathbf{= 0 \text{ if } i=j}$$

i and j are clusters consisting of N_i N_j components. m_{ij} is inter dependency.

The clusters will be derived. To apply the clustering techniques to software architecture recovery and reengineering, the object-attribute data matrix should be converted to object-object data matrix, so that the input reflects the interconnectivity of components. The clustering techniques are then used to minimize interconnections among components. Here we explain how the clustering technique could be used to support the identification of a pattern.

There are some client classes that are accessible to some subsystem classes. With the existing software architecture recovery assistants, especially file names based approaches; the result may look perfect for the subsystem. In other words, the architecture recovered through this type of technique is close to or the same as the modules that are partitioned by the designer.

Certainly, architecture capture is important and valuable. But we are also concerned with the ways to improve the architecture rather than simply capture it. Besides, in reality, the directory structures already often reveal the high level components of a system. Simply capturing software architecture at a higher level abstraction often has limited benefits. We get very different partitions by applying the clustering techniques to this example. In fact, the subsystem does not exist anymore, since many subsystem classes are directly accessed by or related to client classes. In other words, the clustering technique reveals that some classes in the subsystem are more closely related to client classes, which contradicts the design concept. Ideally, the subsystem classes should be grouped together as one unit. Clustering techniques could be used in this type of situations to enforce the architect to reason ways to keep the subsystem classes in a more cohesive manner. Facade pattern provides common interfaces to subsystem classes and facilitates separation of concerns. The subsystem classes in the new pattern-based design are grouped in the same unit according to the clustering method, which is consistent with the original design. In this example, the clustering technique helps the adoption of a design pattern to reduce the coupling between the subsystem and the clients.

6. Conclusion

With respect to graph matching there is exponential complexity, however we 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 suing 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.

References

A.Marzal, E. Vidal. (1995). Computation of normalized edit distance and applications, IEEE Trans. Syst. Man Cybern. 25 (1995) 202-206.

B. Mohar. (1997). Some applications of Laplace eigenvalues of graphs, Graph Symmetry: Algebraic Methods and Applications, NATO ASI Series C, vol. 497, 1997, pp. 227–275.

B.W. Kernighan, S.Lin. (1970). An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. (1970) 291-307.

M. Fiedler. (1975). A property of eigenvectors of non-negative symmetric matrices and its application to graph theory, Czech. Math. J. 25 (1975) 619–633.

R.J. Wilson, J.J. Watkins. (1990). Graphs: An Introductory Approach: A first Course in Discrete Mathematics, Wiley International Edition. New York, etc., Wiley, 1990.

W.H. Haemers. (1995). Interlacing Eigen values and graphs, Linear Algebra Appl. 1995, pp. 593–616.