A New Approach for Detecting Design Patterns by Graph Decomposition and Graph Isomorphism

Akshara Pande, Manjari Gupta, and A.K. Tripathi

DST- Centre for Interdisciplinary Mathematical Sciences,
Banaras Hindu University, Varanasi-221005, India
pandeakshara@gmail.com ,manjari@bhu.ac.in,
aktripathi.cse@itbhu.ac.in

Abstract. Design Pattern Detection is a part of many solutions to Software Engineering difficulties. It is a part of reengineering process and thus gives important information to the designer. Design Pattern existence improve the program understanding and software maintenance. With the help of these patterns specific design problem can be solved and object oriented design become more flexible and reusable. Hence a reliable design pattern mining is required. Here we are applying graph decomposition followed by graph isomorphism technique for design pattern detection.

Keywords: design pattern, graph decomposition, UML, adjacency matrix, graph isomorphism.

1 Introduction

In many object oriented software, there are recurring patterns of classes. Design Patterns are defined as explanation of corresponding classes that forms a common solution to frequent design problem. To reuse expert design experiences, the design patterns [1] have been extensively used by software industry. When patterns are implemented in a system, the pattern-related information is generally no longer available. It is tough to trace out such design information. To understand the systems and to modifications in them it is necessary to recover pattern instances. There are number of pattern detection techniques, some of them have been discussed in section 5. In this paper we are proposing an algorithm for graph decomposition (directed graph) and then trying to find out whether design pattern exists by applying graph isomorphism technique.

Here we are taking two graphs, one is corresponding to the model graph (i.e. system under study) and other is corresponding to the design pattern graph. Graph Decomposition is applied on model graph, and then try to find out is there any isomorphism between the decomposed graphs (corresponding to model graph) and design patterns. The advantage of this approach is that it reduces time complexity of matching two graphs. The detailed methods are given in following sections. In section 2 graph decomposition algorithm is proposed. Condition of graph isomorphism is explained in section 3. In section 4 we have proved the algorithm with examples. Related works are discussed in section 5. Lastly we concluded in section 6.

S. Ranka et al. (Eds.): IC3 2010, Part II, CCIS 95, pp. 108–119, 2010.

2 Graph Decomposition

The main idea of decomposing the directed graph is that, the smaller the graph structure, the easier the matching process and this will result in lower complexity than matching the original graph.

Let G= (V, E) be a graph, where V is the set of vertices present in the graph and E is the set of edges present between the vertices. Suppose V= {n_1, n_2, n_3…n_n| n_i is the nodes, i=1, 2….n}, E= {e_1, e_2, e_3…e_n| e_i is the edges, i=1, 2….n}, the graph G can be decomposed on the basis of the edges present between the nodes. Here we are applying decomposition algorithm on the graph (having more than one edge) for obtaining the decomposed graphs (which have one or two edges).

2.1 Relationship Graphs Representation

The system under study or the system for which we have the source code is taken first, the corresponding the class diagram of UML of that code (object oriented system) is drawn. After that the relationship graphs (that exists in UML diagram) is extracted. We have taken the UML Diagram of system design as shown in Figure 1. There are three relationships (i.e. generalization, direct association and aggregation), the corresponding relationship graphs (i.e. directed graph) are shown in Figure 2, 3, 4.

Generalization relationship graph (i.e. fig 2) has relationship between only three of the nodes, Direct Association relationship graph (i.e. fig 3) has relationship between 4 of the nodes so we will apply decomposition algorithm only on figure 2 and figure 3. There is no need to apply the decomposition algorithm on figure 4 (i.e. aggregation relationship graph) as it has relationship between only two nodes.

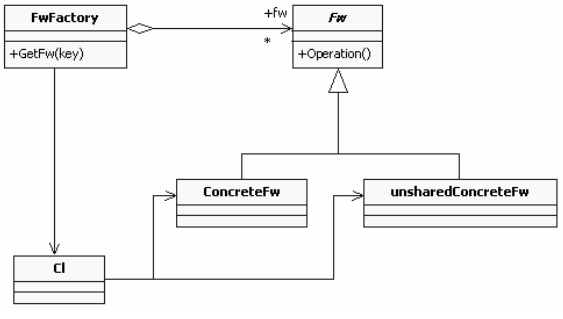

Fig. 1. UML Diagram of System Design

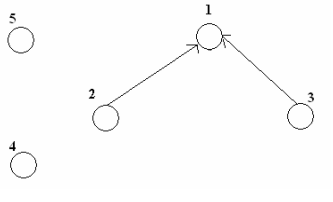

Fig. 2. Generalization Relationship Graph of UML Diagram of System Design

Decomposition algorithm uses adjacency matrix representation of the corresponding relationship graphs. The adjacency matrices for figure 2 and figure 3 have been shown in figure 5 and figure 6. In adjacency matrix the entry should be 1 if there exists an edge between the two nodes, otherwise 0.

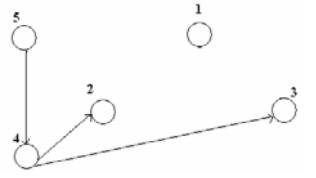

Fig. 3. Direct Association Relationship Graph of UML Diagram of System Design

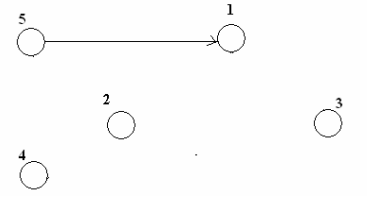

Fig. 4. Aggregation Relationship Graph of UML Diagram of System Design

	1	2	3	4	5
1	0	0	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Fig. 5. Adjacency matrix corresponding to Generalization Relationship Graph

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	1	1	0	0
5	0	0	0	1	0

Fig. 6. Adjacency matrix corresponding to Direct Association Relationship Graph

2.2 Graph Decomposition Algorithm

The graph decomposition can be done using the following algorithm:

```
DECOMPOS (Graph)
For i = 1 to Number of Nodes
   For j = 1 to Number of Nodes
     if aij ≠ 0  then
      DECOMPOS2 {i, j} = DECOMPOS2 {i, j} ⋃ {i, j, aij}
      For k = i+1 to Number of Nodes
       if  ajk ≠ 0 then
        DECOMPOS3{i, j, k } = DECOMPOS3{i, j, k}⋃{i,
                                        j, j, k, aijk}
       end if
       if  akj ≠ 0   then
       DECOMPOS3{i, j, k } = DECOMPOS3{i, j, k} ⋃ {i,
                                        j, k, j, aijk}
       end if
      For k = j+1 to Number of Nodes
       if  aik ≠ 0   then
       DECOMPOS3{i, j, k } = DECOMPOS3{i, j, k} ⋃ {i,
                                        j, i, k, aijk}
       end if
  end if
End DECOMPOS
```

The example of decomposing a graph has been shown in figure 7. Firstly, take the original graph, and apply the DECOMPOS algorithm on it. As a result, we found the 3 two order decomposition (graphs having two nodes) and 3 three order decomposition (graphs having three nodes) as shown in figure 7.

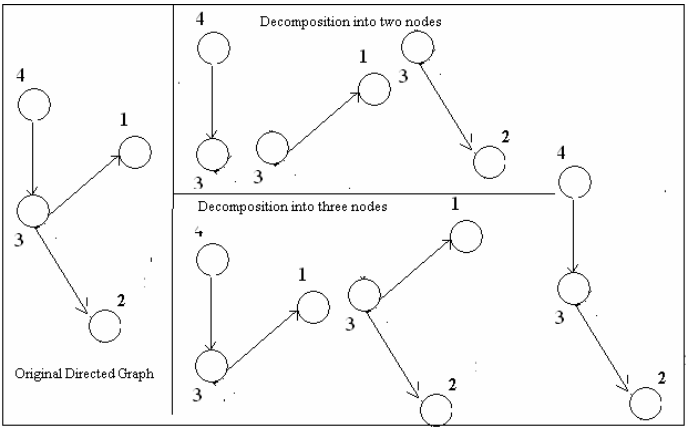

Fig. 7. Original Directed Graph; and its decomposed graphs

The order of complexity of this decomposition algorithm is $O(n^3)$, where n is the number of nodes present in the graph. This algorithm works for only those design patterns having similar relationships among at most three classes in its UML class diagram. However this condition may not hold for only few of the design patterns. Thus this approach can be applied for almost all of the design patterns.

Now this algorithm is applied on adjacency matrices of Generalization relationship and Direct Association relationship of system design. The decomposed graphs have been shown in figure 8 and figure 9 (corresponding to Generalization relationship) and figure 10 and figure 11 (corresponding to Direct Association relationship).

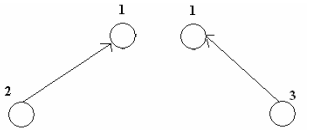

Fig. 8. After Decomposition of Generalization Relationship Graph (for two nodes)

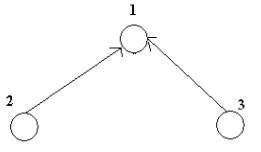

Fig. 9. After Decomposition of Generalization Relationship Graph (for three nodes)

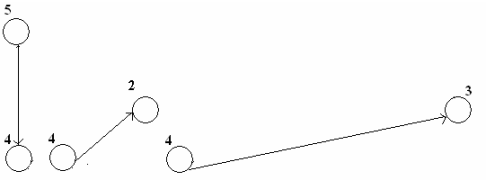

Fig. 10. After Decomposition of Direct Association Relationship Graph (for two nodes)

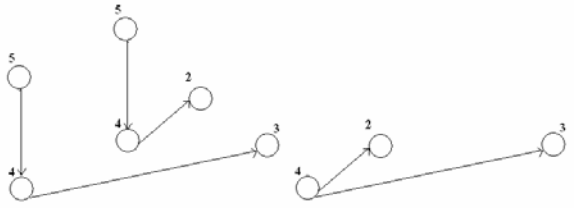

Fig. 11. After Decomposition of Direct Association Relationship Graph (for three nodes)

The graphs are stored after decomposition in different orders for example to store Generalization relationship graph we have order 2 (since edge is present between two nodes). There are two graphs of order 2 and one graph of order 3 (edge present among

three nodes). But for storing Direct Association relationship graph after decomposition the order must be 2 and 3 (since edge is present for two nodes and three nodes both). There are three graphs of order 2 and three graphs of order 3.

3 Graph Isomorphism: A Graph Matching Technique

Graph Matching techniques are important and very general form of pattern matching that finds realistic use in areas such as image processing, pattern recognition and computer vision, graph grammars, graph transformation, bio computing, search operation in chemical structural formulae database, etc. Graph isomorphism is a graph matching technique. In graph isomorphism, one to one correspondence between two nodes and their edges is searched out. If there is one to one correspondence, then two of the graphs are said to be isomorphic to each other.

Let [5] G1 (V1, E1) and G2 (V2, E2) be two graphs, where V1, V2 are the set of vertices and E1, E2 are the set of edges. Let M1 and M2 be the adjacency matrices corresponding to G1 and G2 respectively. A permutation matrix is a square (0, 1)-matrix that has exactly one entry 1 in each row and each column and 0's elsewhere. Two graphs G1 (M1, L_v, L_e) and G2 (M2, L_v, L_e) are said to be isomorphic [5], here the term L_v and L_e represents the labeling of vertices and edges respectively. If there exist a permutation matrix P such that

$$M2 = P\ M1\ P^T \tag{1}$$

In this paper we are assuming M1 is the matrix corresponding to the system design (i.e. the adjacency matrix of relationship after decomposition) and M2 as the design pattern adjacency matrix which should have the same order as of system design. Since we are decomposing only up to 3 order (lastly graph has only relationship between three nodes) so design pattern must have the order 2 or 3. Permutation matrix should also have the same order as of the relationship adjacency matrix of system design and design pattern.

4 Design Pattern Detection Using Graph Isomorphism

There are 23 GoF (Fang of Four) [1] design patterns, corresponding UML diagrams can be drawn for them, but we are only considering those which have relationship graph of order 2 or 3.

4.1 Design Pattern Detection as Façade Design Pattern

Firstly we are considering Façade design pattern, the UML diagram corresponding to it, is shown in figure 12. There is only one relationship, i.e. Direct Association relationship, relationship graph has been shown in figure 13.

Now adjacency matrices corresponding to Direct Association relationship of system design (of order 2 i.e. consisting two nodes and an edge between them) and design pattern should be drawn (shown in figure 14 and 15). A 2×2 order permutation matrix is taken as shown in figure 16.

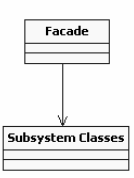

Fig. 12. Façade Design Pattern

Fig. 13. Direct Association Relationship Graph of UML Diagram of Façade Design Pattern

$$\begin{array}{c|cc} & 4 & 5 \\ \hline 4 & 0 & 0 \\ 5 & 1 & 0 \end{array} \qquad \begin{array}{c|cc} & 2 & 4 \\ \hline 2 & 0 & 0 \\ 4 & 1 & 0 \end{array} \qquad \begin{array}{c|cc} & 3 & 4 \\ \hline 3 & 0 & 0 \\ 4 & 1 & 0 \end{array}$$

Fig. 14. Adjacency matrices for Direct Association Relationship Graph of fig. 8 (M1)

Since permutation matrix is same as an identity matrix so transpose of permutation will be the same as of permutation matrix.

$$\begin{array}{c|cc} & b & a \\ \hline b & 0 & 0 \\ a & 1 & 0 \end{array}$$

Fig. 15. Adjacency matrices for Direct Association Relationship Graph of fig. 11(M2)

$$\begin{array}{|cc} \hline 1 & 0 \\ 0 & 1 \end{array}$$

Fig. 16. Permutation matrix (P or P^T)

Now it could be easily seen that, equation 1 is satisfied. So Façade design pattern exists in system design or in model graph.

4.2 Design Pattern Detection as Strategy Design Pattern

Now consider strategy design pattern, UML as shown in figure 17. The corresponding Generalization relationship and Aggregation relationship has been shown in figure 18 and figure 19 respectively.

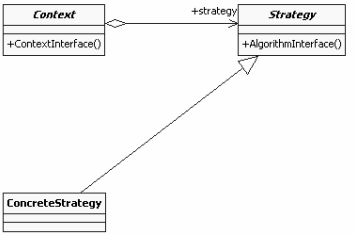

Fig. 17. Strategy Design Pattern

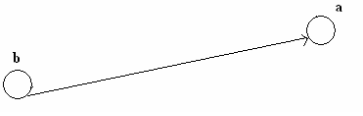

Fig. 18. Generalization Relationship Graph of UML Diagram of Strategy Design Pattern

Fig. 19. Aggregation Relationship Graph of UML Diagram of Strategy Design Pattern

	1	2
1	0	0
2	1	0

	1	3
1	0	0
3	1	0

Fig. 20. Adjacency matrices for Generalization Relationship Graph of fig. 8 (M1)

	a	b
a	0	0
b	1	0

	a	c
a	0	0
c	1	0

Fig. 21. Adjacency matrices for Generalization Relationship Graph and Aggregation Relationship graph of fig. 17 and fig. 18 (either can be taken as M2)

Matrix M1 and M2 are shown in figure 20 and figure 21 respectively. If permutation matrix is of same as in figure 16, then by applying equation 1, we can say that strategy design pattern exists in system design for Generalization relationship. Similarly could be seen for aggregation relationship also.

4.3 Design Pattern Detection as Strategy Design Pattern

Now consider composite design pattern, UML as shown in figure 22. The corresponding Generalization relationship, Direct Association relationship and Aggregation relationship has been shown in figure 23, figure 24 and figure 25 respectively.

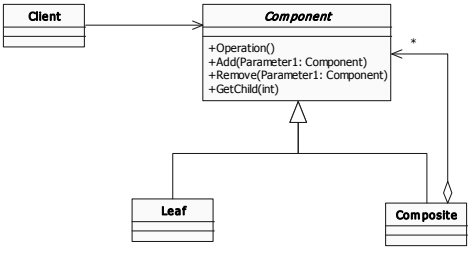

Fig. 22. Composite Design Pattern

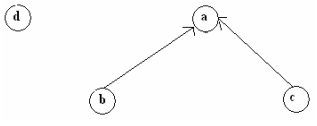

Fig. 23. Generalization Relationship Graph of UML Diagram of Composite Design Pattern

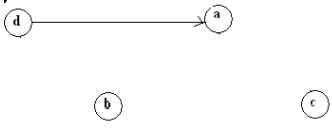

Fig. 24. Direct Association Relationship Graph of UML Diagram of Composite Design Pattern

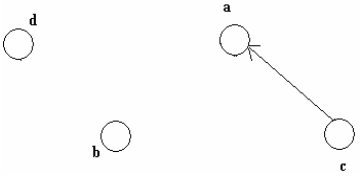

Fig. 25. Aggregation Relationship Graph of UML Diagram of Composite Design Pattern

If Generalization relationship of system design is taken as 3 order as shown in figure 8, adjacency matrix can be written as shown in figure 26. And Generalization relationship of composite design pattern is shown in figure 27. Permutation matrix and its transpose are taken as shown in figure 28.

	1	2	3
1	0	0	0
2	1	0	0
3	1	0	0

Fig. 26. Adjacency matrix for Generalization Relationship Graph of fig. 9(M1)

	a	b	c
a	0	0	0
b	1	0	0
c	1	0	0

Fig. 27. Adjacency matrix for Generalization Relationship Graph of fig. 23(M2)

1	0	0
0	1	0
0	0	1

Fig. 28. Permutation Matrix (P or P^T)

Now if equation 1 is applied for composite design pattern detection for Generalization relationship, it exists. We found that both the graphs are isomorphic for generalization relationship. Similar is for Direct Association and for Aggregation relationship also.

4.4 Particular Design Pattern May or May Not Exist

Above we see the examples of design pattern existence but it can be possible that a particular design pattern does not exist in system design. In this there will be no permutation matrix for which we can find out an isomorphism between adjacency matrix of system design and adjacency matrix of design pattern.

5 Related Work

The first effort towards automatically detect design pattern was achieved by Brown [6]. In this work, Smalltalk code was reverse-engineered to facilitate the detection of four well-known patterns from the catalog by Gamma et al. [1].

Antoniol et al. [7] gave a technique to identify structural patterns in a system with the purpose to observe how useful a design pattern recovery tool could be in program understanding and maintenance.

Nikolaos Tsantalis [2], proposed a methodology for design pattern detection using similarity scoring. But the limitation of similarity algorithm is that it only calculates the similarity between two vertices, not the similarity between two graphs. To solve this Jing Dong [3] gave another approach called template matching, which calculates the similarity between subgraphs of two graphs instead of vertices.

Wenzel [4] gave the difference calculation method works on UML models. The advantage of difference calculation method on other design pattern detecting technique is that it detects the incomplete pattern instances also.

In our earlier work [8-11], we have shown how to detect design patterns using various techniques. The complexity of previous proposed techniques are very high, we are trying to reduce this complexity by introducing graph decomposition.

6 Conclusion

In this paper we took the model graph and a data graph (corresponding to design pattern), and tried to find out whether design pattern exists or not in model graph. Firstly, we decompose the model graph into different orders, and then take same order design pattern. There are 23 GoF (Fang of Four) [1] design patterns. By applying the decomposition algorithm, the complexity is reduced. For isomorphism detection, only one permutation matrix (i.e. identity matrix) is required in our case. Hence the time complexity of choosing suitable permutation matrix has also been removed.

References

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
2. Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., Halkidis, S.: Design Pattern Detection Using Similarity Scoring. IEEE transaction on software engineering 32(11) (2006)
3. Dong, J., Sun, Y., Zhao, Y.: Design Pattern Detection By Template Matching. In: The Proceedings of the 23rd Annual ACM, Symposium on Applied Computing (SAC), Ceará, Brazil, pp. 765–769 (March 2008)
4. Wenzel, S., Kelter, U.: Model-driven design pattern detection using difference calculation. In: Proc. of the 1st International Workshop on Pattern Detection For Reverse Engineering (DPD4RE), Benevento, Italy (October 2006)
5. Messmer, B.T., Bunke, H.: Subgraph isomorphism detection in polynomial time on preprocessed model graphs. In: Second Asian Conference on Computer Vision, pp. 151–155 (1995)

6. Brown, K.: Design Reverse-Engineering and Automated Design Pattern Detection in Smalltalk, Technical Report TR-96-07, Dept. of Computer Science, North Carolina State Univ. (1996)
7. Antoniol, G., Casazza, G., Penta, M.D., Fiutem, R.: Object- Oriented Design Patterns Recovery. J. Systems and Software 59(2), 181–196 (2001)
8. Pande, A., Gupta, M.: Design Pattern Detection Using Graph Matching. International Journal of Computer Engineering and Information Technology (IJCEIT) 15(20), 59–64 (2010); Special Edition 2010
9. Pande, A., Gupta, M.: A New Approach for Design Pattern Detection Using Subgraph Isomorphism. In: Proc. of National Conference on Mathematical Techniques: Emerging Paradigm for Electronics and IT Industries (MATEIT 2010) (2010)
10. Gupta, M., Pande, A.: A New Approach for Detecting Design Patterns Using Genetic Algorithm. Presented in International Conference on Optimization and its Application, organized by Deptt. of Mathematics, Banaras Hindu University (2010)
11. Pande, A., Gupta, M., Tripathi, A.K.: Design Pattern Mining for GIS Application using Graph Matching Techniques. In: 3rd IEEE International Conference on Computer Science and Information Technology, Chengdu, China, July 9-11 (accepted, 2010)