International Journal of Computer Applications (0975 – 8887)
Volume 85 – No 7, January 2014

Design Pattern Mining by Product of Sum (POS) Expression for Graphs

Manjari Gupta
Department of Computer Science
Banaras Hindu University, Varanasi

Rajwant Singh Rao
Department of Computer Science & Information Technology (CSIT), Gurughasidas Vishwavidyalaya

ABSTRACT

There are many recurring patterns of classes which exist in several object oriented software as an experience of developers. Design Pattern Mining is an important part of many solutions to Software Reuse practices.

Design pattern instances are highly important and useful for program understanding and software maintenance. Hence an automatic and reliable design pattern mining capability is required. Here we are proposing a new method for design pattern detection based on Boolean functions.

Keywords

Design pattern, UML, Boolean function, POS form.

1. INTRODUCTION

Design patterns [1] are increasingly being applied in object oriented software design processes as a part of many solutions to Software Engineering difficulties and thus are extensively used by software industries. Each design pattern denotes a high level abstraction, and contains expert knowledge and thus a software developed using design patterns have many desired properties. Design pattern detection is a part of reengineering process and thus gives important information to the designer. To understand a software system and to modify it, it is necessary to recover pattern instances. It would be useful for reengineers to have an automatic design pattern detection tool that can detect design pattern from the system without need of thorough/manual analysis of it. There are number of pattern detection techniques, some of them have been discussed in section 5. In this paper we are proposing a new method for design pattern detection based on boolean functions.

We will convert the UML diagrams of design patterns and model graph into boolean function in POS form. Then by comparing boolean functions of both it can be decided whether the pattern exists in system design or not. We are implementing this approach to get a design pattern detection tool so that reengineers need not to manually analyze the design of the system to identify used design patterns, if any, to understand the design of the system.

Here we are taking two graphs, one is corresponding to the system design (i.e. system under study) and other is corresponding to the design pattern graph. A particular example is shown in figure 1.

The advantage of this approach is that it reduces time complexity of matching two graphs. Using this approach we can also detect instances of design patterns that is not possible by many other approaches proposed in the literature. Related works are discussed in section 2. In section 3 boolean function in POS form representation of the system design and design patterns are explained. The proposed design pattern detection is described in section 4. Section 5 describes the issues in design pattern detection based on boolean functions. Lastly we concluded in section 6.

2. RELATED WORK

Brown [6] proposed a method for automatically detection of design patterns. In his work Smalltalk code was reverse-engineered to facilitate the detection of four well-known patterns from the catalog by Gamma et al. [1]. Nikolaos Tsantalis [3], proposed a methodology for design pattern detection using similarity scoring between graph vertices. It can detect variants of design patterns also. 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 [4] gave another approach called template matching, which calculates the similarity between sub-graphs of two graphs instead of vertices. They detected design patterns from software by using normalized cross correlation. Stencel and Wegrzynowicz [5] proposed a method for automatic design pattern detection that is able to detect many nonstandard implementation variants of design pattern. Their method was customizable because a new pattern retrieval query can be introduced along with modifying an existing one and then repeat the detection using the results of earlier source code analysis stored in a relationaldatabase. Drawback was that the method was not general enough to identify all design patterns. Further the translation of first order logic formulae as SQL queries is very laborious and error-prone.

In our earlier work, we used the klenberg approach and fuzzy graph algorithms for design pattern detection [9]. The drawback of these two methods is that they are only concerned about node similarity not the whole graph. We used sub-graph isomorphism detection approach that overcomes this drawback [9]. We have used these and other approaches for design pattern detection in GIS application [10]. To reduce complexity of design pattern detecting algorithm we used the graph decomposition technique [11]. The order of complexity of this decomposition algorithm is O(n3), 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. In another work we find out whether design pattern matches to any sub-graph of system design by using decision tree [12]. A decision tree is developed with the help of row-column elements, and then it is traversed to identify patterns. By applying the decision tree approach, the complexity is reduced. We proposed a new approach ‘DNIT’ (Depth-Node-Input Table) [13]. It is based on the concept of depths from the randomly chosen initial node (also called root node which has depth zero) in directed graph. In another work we applied state space representation of graph matching algorithm to detect design patterns [14]. State space representation easily describes the graph matching process.

The advantage of this method used for design pattern detection was that the memory requirement was quite lower than from other similar algorithms. Another advantage is that it detects variants as well as any occurrence of each design patterns.

3. BOOLEAN FUNCTION REPRESENTATION OF SYSTEM DESIGN AND DESIGN PATTERN

UML diagrams of system design and design patterns are converted into graphs. Before converting a UML diagram into graphs we first modify the UML diagram in such a way so that variant of design patterns can also be detected. The reason for the design pattern variant problems is the fact that the inheritance and aggregation relationship have the property of transitiveness [2]. Thus if there is an inheritance (or aggregation) relationship between classes c1 and c2 and the same relationship between c2 and c3, we will introduce the same relationship between c1 and c3 also. 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(a), 2(b) and 2(c) respectively.

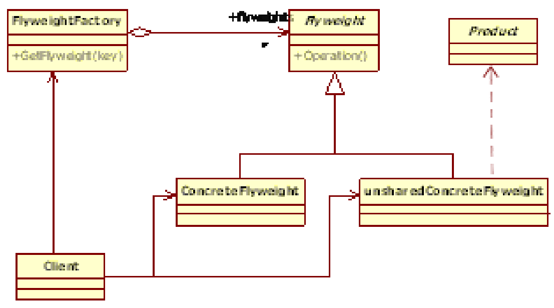

Fig. 1 UML Diagram of System Design [8]

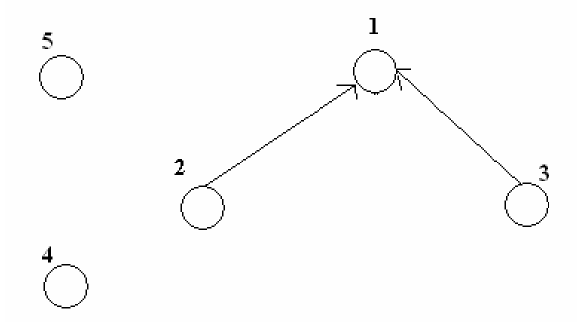

Fig. 2(a) Generalization Relationship Graph for System Design

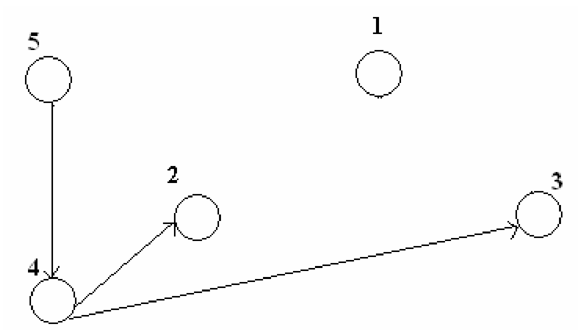

Fig. 2(b) Direct Association Relationship Graph for System Design

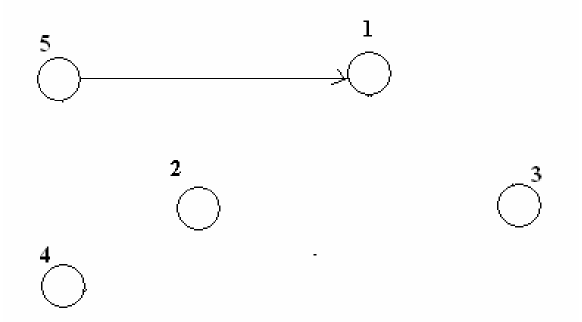

Fig. 2(c) Aggregation Relationship Graph for System Design

The relationship graphs for design patterns can be extracted in similar manner as for system design. For example singleton, façade and strategy design pattern relationship graphs are shown in section 4.

All relationship graphs can now be converted into Boolean function form in product of sum (POS) form [7]. Algorithm for converting a directed graph into sum of product form (POS) is as follows.

```
*Directed_Graph2POS*

*Aij is the adjacency matrix of order nxm corresponding to relationship directed graph*

*count=0 // count is a flag whose value becomes 1 when 1 is found in a row in the adjacency matrix*

**step1. for** j=1 to m **begin**

**step2.  if** $a_{1j} \neq 0$ **then**

      position[count] = j; // position counts column number which has value 1

      count= count +1;

       j= j + 1;

*//end of  if condition step 2*

**else** j= j+1

*//end of  for loop staep 1*

**Step3. for** l=1 to k (where k<count) **begin**

      i = 1;

      j = position[count];

      print i;

    **for** j = 1 to m **begin**

        **if** aij ≠ 0 **then**
```

```
            if j<i then print j break
            end if
        print j
         i = j; j = 0;
         end if
     j= j + 1;
   end for
   m = m + 1;
end for
END Directed_Graph2POS
```

Now by applying Directed_Graph2POS algorithm on relationship adjacency matrices, corresponding POSs can be written.

On applying the above algorithm on adjacency matrices of relationship graphs of system design (figure 5, 6, 7), three POS form will be generated: for generalization, for direct association and for aggregation, as shown in equation 1.

POS(gen) = (2+1) *(3+1)… (1-a)

POS (d.a.) = (5+4+2) * (5+4+3)… (1-b)

POS(agg) = (5+1)… (1-c)

The relationship graphs of design patterns can also be converted into POS form by applying above algorithm in similar manner as done for system design. Singleton facade and strategy design pattern relationship graphs are converted into POS form in section 4.

4. PROPOSED METHOD FOR DESIGN PATTERN MINING

There are 23 GoF (Gang of Four) [1] design patterns. UML diagrams can be drawn for each of the corresponding design patterns. After checking sub isomorphism between the relationship graphs of a design pattern and the model graph, there may be three

cases [14]:

i) Relationship graph of a design pattern is (sub) isomorphic to the model graph. This is the case of existence of design pattern.

ii) Relationship graph of a design pattern is partially (sub) isomorphic to the model graph. This is the case of existence of variant of a design pattern.

iii) Relationship graph of a design pattern is not sub isomorphic to the model graph. This is the case of non-existence of design pattern.

In the 23 GoF (Gang of Four) [1] design patterns. Generally design patterns are used to reuse design. Thus there exists a UML diagram corresponding it each design pattern. Our method can be used for all those design patterns in which there is no relationship from a class to itself. Here we are considering some of them. We will first convert the design pattern into relationship graphs and that into Boolean function in POS form. This string (of design patter) represented by Boolean function will be searched into corresponding string of system design.

Substring search, is a kind of sub-graph isomorphism detection, where the string to be searched is corresponding to sub-graph and the bigger string, in which we search the smaller one, is corresponding to whole graph. 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. Using the discussed algorithm one can find whether a design pattern or its variants exist in the system design or not. In the first case design pattern relationship graph exist in corresponding relationship graph of system design. In the second case design pattern relationship graph partially exists in the corresponding relationship graph of system design. Further more than one mapping we may get for one relationship graph. That shows possibility of multiple occurrence of a design pattern (complete/variant) in the system design.

It is important to note that even if POS expressions of all the relationships of design pattern exist in corresponding POS expressions of system design, design patterns may not fully match with subgraph of the system design. Reason is that the incidence relationship preservation condition may not satisfy. In the following subsections we demonstrate the proposed method of design pattern detection by some examples.

4.1 Design Pattern Mining: Exact Matching

Design pattern is said to be fully matched to a subpart of system design if it matches to each relationship which present in design pattern. To demonstrate this case, let us chose façade design pattern Which UML diagram is shown in Figure 3. It is having a single direct association relationship. Graph for this relationship is shown in Figure 4(a).

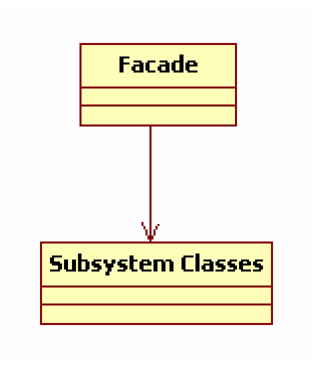

Fig. 3 Façade Design Pattern [8]

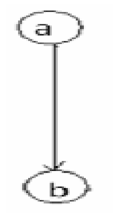

Fig. 4(a) Direct Association Relationship Graph for Façade Design Pattern

To search façade design pattern in the system design first we find out Boolean function for façade design pattern. By applying Directed_Graph2POS algorithm on adjacency matrix for graph shown in figure 4(a), we get the corresponding POS expressions as follows:

POS(d.a.) = a+b … (2)

To find out whether it exists in system design or not, we have to consider equation 2 and equation 1(b) (Boolean function for system design for direct association). Since façade design pattern has one term with two components only, so in system design we will search for a term having more than two components. In this way we get three distinct mappings {[(a, 5) (b, 4)], [(a, 4), (b, 2)], [(a, 4), (b, 3)]}. There is no other relationship in the design pattern thus no need to check for other POS expressions of system design. Only one relationship in design pattern also relieves us from checking the incidence relationship preservation. Thus, there are three occurrences of façade design patterns in system design.

Please use a 9-point Times Roman font, or other Roman font with serifs, as close as possible in appearance to Times Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

4.2 Design Pattern Detection: Partial Matching

Design pattern is said to be partially matched to a subpart of system design if it does not match for each relationship present in the design pattern or some of the incidence relationship preservation done not hold. The example of partial match in our case is abstract factory design pattern. Relationship graph of direct association is a sub-graph of corresponding graph of system design. But for rest of the relationships it is not the case. Generalization relationship graph of design pattern is disconnected while for design pattern it is connected thus we have to check for partial existence.

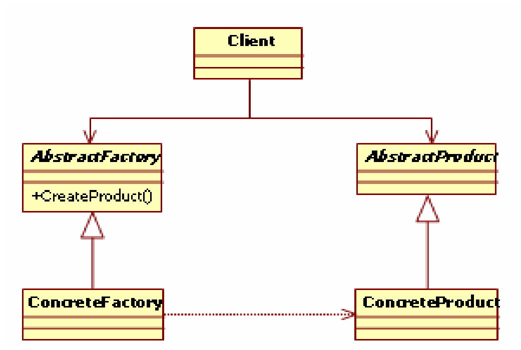

Fig. 5 Abstract Factory Design Pattern [8]

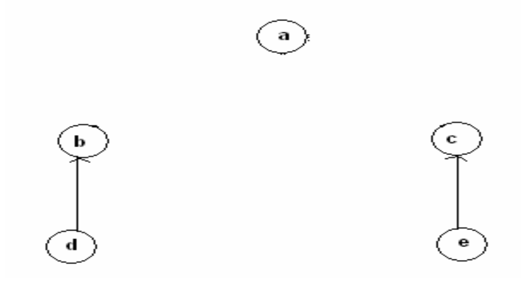

Fig. 6(a) Generalization Relationship Graph for Abstract Factory Design Pattern

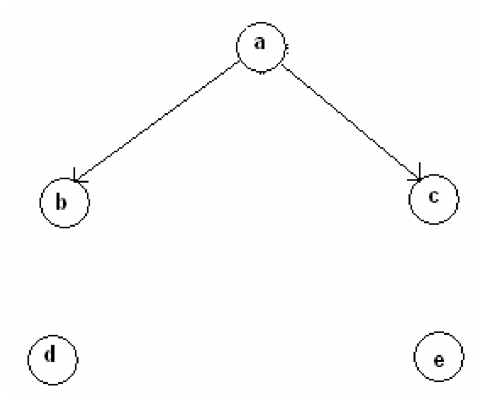

Fig.6 (b) Direct Association Relationship Graph for Abstract Factory Design Pattern

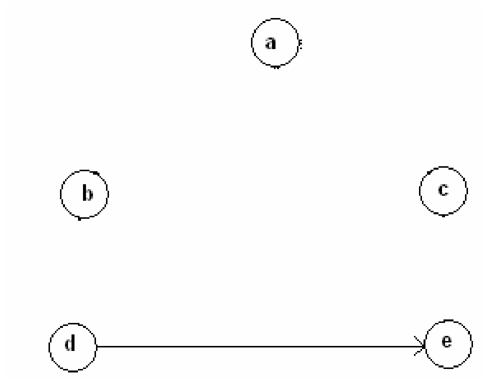

Fig. 6(c) Dependency Relationship Graph for Abstract Factory Design Pattern

To search abstract factory design pattern (figure 5) in the system design first we find out Boolean function for abstract factory design pattern. By applying Directed_Graph2POS algorithm on adjacency matrix for figure 6(a), 6(b) and 6(c) we get the orresponding

POS expressions as follows:

POS(gen1) = d+b POS(gen 2) = e+c 3(b)

POS(d.a.) = a.b + a.c 3(a)

POS(dep) = d+e 3(c)

To find out whether it exists in system design or not, we will check for each relationship one by one. Consider equation 2(a) and equation 3(a). Both POS expressions for generalization relationship of this pattern match with two terms present in corresponding POS expression of system design independently. Thus two mappings {[(d,2), (b,1)], [(e,3), (c,1)]}. Since both of these two mappings do not cover all vertices of generalization relationship graph of design pattern to which edges are incident, it is partial mapping.

Now we will check for direct association relationship. By comparing equation 2(b) and 3(b) we get two partial mappings {(a,4), (b,2), (c,3)], [(a,4), (b,3), (c,2)]}. Since both these mappings cover all elements of direct association relationship graph of design pattern to which edges are incident, it is full mapping for direct association. No other relationship is common. Now we check incidence relationship preservation. After checking incidence relationship we get two final mappings that consider all relationships of design patterns that are common with relationships present in design pattern. These mappings are {[(a,1), (b,2), (c,3), (d,4)], [(a,1), (b,2), (c,3), (e,4)]. Since both these two final mappings do not cover all vertices of design pattern, there are two variants of abstract factory pattern in the system design.

Please use a 9-point Times Roman font, or other Roman font with serifs, as close as possible in appearance to Times Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

4.3 Design Pattern Detection: may not exist

The limitation of this approach is that it cannot be used to identify some design patterns where relationship exists from a class to itself for example singleton design pattern. Its UML diagram and relationship graph is shown in figure 7 and figure 7(a) respectively.

Since there is a self-loop, one cannot write Boolean function for the graph shown in figure 7(a) and thus this method cannot be used for these types of design patterns.

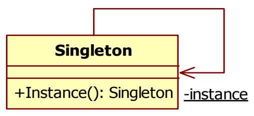

Figure 7. Singleton Design Pattern [8]

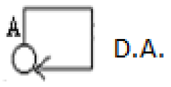

Figure 7(a) DPG of UML Diagram of Singleton Design Pattern

Please use a 9-point Times Roman font, or other Roman font with serifs, as close as possible in appearance to Times Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

5. CONCLUSIONS

In this paper we proposed an approach to design pattern detection using an algorithm for converting of directed graphs (for relationships) into sum of product (POS) form of boolean functions. By taking the combination of terms (substring) present in design patternwe tried to detect design pattern in system design. The limitation of this approach is that few patterns involving self-relationship on a class cannot be identified. We are developing a prototype that allows the implementation of the approach discussed.

6. REFERENCES

[1] Gamma E., Helm R., Johnson R., Vlissides J. (1995): Design Patterns Elements of Reusable Object-Oriented Software, Addison- Wesley.

[2] Xhang Z.X., Li Q.H. and Ben K.R. (2004): A New Method for Design Pattern Mining, IEEE Explore.

[3] Tsantalis N., Chatzigeorgiou A., Stephanides G., and Halkidis S. (2006): Design Pattern Detection Using Similarity Scoring, IEEE transaction on software engineering, 32(11).

[4] Dong J., Sun Y., Zhao Y. (2008): Design Pattern DetectionBy Template Matching , the Proceedings of the 23rd AnnualACM, Symposium on Applied Computing (SAC), pages 765- 769,Ceará, Brazil,March.

[5] Stencel K. and Wegrzynowicz P. (2008): Detection of Diverse Design Pattern Variants, 15th Asia-Pacific Software Engineering Conference, IEEE Computer Society.

[6] Brown K. (1996): Design Reverse-Engineering and Automated Design Pattern Detection in Smalltalk, Technical Report TR-96-07, Dept. of Computer Science, NorthCarolina State Univ.

[7] Cortadella J. and Valiente G. (2000): A Relational View of Sub Graph Isomorphism, In Proc. Fifth Int. Seminar on Relational Methods in Computer Science.

[8] StarUML, The Open Source UML/MDA Platform. http://staruml.sourceforge.net/en/

[9] Pande A., Gupta M. (2010): Design Pattern Detection Using Graph Matching. International Journal of ComuterEngineering and Information Technology(IJCEIT), Vol 15, No 20, Special Edition 2010, pp 59-64.

[10] Pande A., Gupta M., Tripathi A.K. (2010): Design Pattern Mining for GIS Application using Graph Matching Techniques. 3rd IEEE International Conference on Computer Science and Information Technology . 09-11 July, 2010, Chengdu, China.

[11] Pande A., Gupta M., Tripathi A.K. (2010): A New Approach for Detecting Design Patterns by Graph Decomposition and Graph Isomorphism. In Proc. Of Third International Conference on Contemporary Computing(IC3), published by Springer. 09-11 August, 2010, Noida, India.

[12] Pande A., Gupta M., Tripathi A.K. (2010): A Decision Tree Approach for Design Patterns Detection by Subgraph Isomorphism, International Conference on Advances inInformation and Communication Technologies, ICT 2010, Kochi, Kerala, published by Springer.

[13] Pande A., Gupta M., Tripathi A.K. (2010): DNIT – A New Approach for Design Pattern Detection, International Conference on Computer and Communication Technology (ICCCT-2010), proceedings to be published by the IEEE.

[14] Gupta M., Singh R.R., Pande A., Tripathi A.K.(2011): Design pattern Mining Using State Space Representation of Graph Matching, 1st International Conference on Computer Science and Information Technology, Banglore, 2011, LNCS, Springer