Design Pattern Detection using Genetic Algorithm for Sub-graph Isomorphism to Enhance Software Reusability

Arti Chaturvedi
School of studies in Computer Science and Applications,
Jiwaji University, Gwalior (M.P.)

Manjari Gupta
Department of Computer Science,
Faculty of Science,
Banaras Hindu University, Varanasi (U.P.)

Sanjay Kumar Gupta
School of studies in Computer Science and Applications,
Jiwaji University, Gwalior (M.P.)

ABSTRACT
Design patterns have been proposed as a technique to introduce reuse in design phase. In industry, it is focused to reuse design patterns as a reusable part when designing a new application. Reusable Design Pattern, that are proven solutions to common design problems, to improves many qualities of applications like Reusability and its maintainability. If better reusability is required for an application where design patterns were used, then an automated tool that can detect the used design pattern in the application will be useful. Therefore, a reliable design pattern discovery is required to promote software reusability. The techniques of finding an isomorphic sub-graph were used to solve design pattern detection in past. Furthermore, we are applying a hybrid genetic algorithm for sub-graph isomorphism problem which uses an incremental approach to detect design patterns. Moreover, detection is done with increasing the size of sub-problem step by step. A hybrid GA is applied to each sub-problem, initialized with the evolved population of previous step. This proposed work of identifying and then later reusing of design pattern facilitate to bring software design in reduced time and consequently expedite software reusability.

General Terms
Software reusability, Graph representation, Algorithm.

Keywords
Design pattern, UML diagram, sub-graph isomorphism, genetic algorithm, incremental genetic algorithm.

1. INTRODUCTION
Now a day, software enters deep into our daily life .Therefore, large and complex applications have greater market demand for such kind of software development paradigms which improve the quality and productivity of software development. There are many possibilities exist in software reuse that make software development cheaper, easier, efficient and also lead to appropriate solution of the problem to quickly cater the market need of software development. Reuse is considered as main benchmark in software development to include quality characteristics in lesser development time. In many ways, reuse is implemented in software development. Design pattern, library files, classes, software component, modules, architectural style are the important techniques used to practice reuse in software development. Design pattern is one among many techniques to facilitate reuse to provide complete and more efficient solution of repeated problem during design of software development.

The main idea behind a good programming language is the concept of transformation of modules. If a program is written as a collection of many modules then it would be easy to understand and maintain it. Gamma et al [1] had given the birth of design patterns that can be treated as modules of a design. Now, one has an idea about design patterns and if they are used in a design of an application it would be easy to understand and reuse it in large context of various applications.

Work of detection design pattern from the existing design pattern repository will provide support to reuse design part multiple times if once design effectively. This may save large amount of time and effort incurred to redesign as solution same thing in many places in software development paradigm by the designers, developers and maintainers. Thus, this way of work makes software development paradigm simple and also easier to control complexity in new application.

2. RELATIONSHIP GRAPHS REPRESENTATION
UML diagram of system design as well as design pattern is converted into graphs. Classes of UML class diagram are represented as nodes and relationships among classes by edges. Each node and edge is labeled. The label of each node is abstract or concrete or concrete sub-class. Thus, represent it by three tuple (t_1, t_2, t_3) where t_1=1 if it is an abstract class, t_2=1 if it is a concrete subclass, and t_3=1 if it is a concrete class, otherwise 0. It can be modified to include more other attributes of a class. In this initial effort, we are considering only these labels of class. Each edge is corresponding to one of the relationships. We assign label 1 for dependency, 2 for generalization, 3 for direct association, and 4 for aggregation. For the system design represented by the UML diagram shown in figure 1, the corresponding graph (MG) is extracted and shown in figure 2.

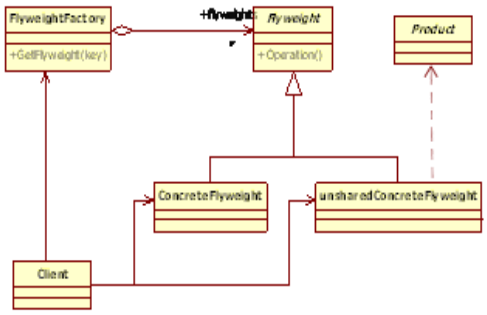

Fig 1 : UML diagram of system design

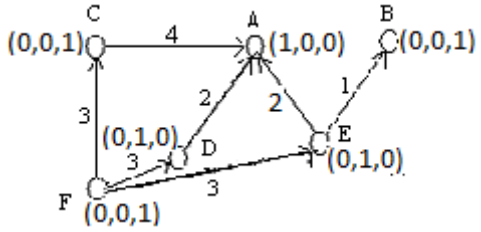

Fig 2 : Model Graph (MG) corresponding to system design

Similarly the class diagram and corresponding design pattern graphs (DPG) are shown in figure 3 and figure 4 for strategy and command design patterns respectively.

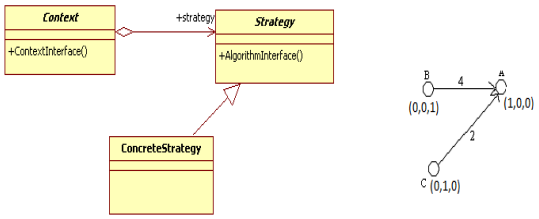

Fig 3: Strategy design pattern and its corresponding graph (DPG)

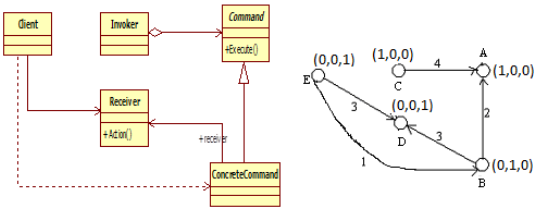

Fig 4: Command design pattern and its corresponding graph (DPG)

3. GRAPH MATCHING ALGORITHM

In this section first will describe how the design pattern detection issue can be solved by extending the solution of its sub-problem. Since, we are considering the evolutionary version of this problem; each sub-problem will have multiple possible solutions. Some of them will be extendable to the solution at the next level sub-problem and some are not.

3.1 Chromosome Structure

Each chromosome represents a particular mapping from sub-graph of G_1 to sub-graph of G_2. One of the chromosomes that give the mapping of G_1 to a sub-graph of G_2 will be the solution of the design patterns detection. Our technique uses a two dimensional array for a chromosome. And population is represented by three dimensional arrays for Cr. First index gives the l^{th} chromosome number, second index gives the vertex of graph G_1 and third index gives the vertex of graph G_2. For example, l^{th} chromosome gives the following matching as shown below:

Chromosome: Cr[l][1][1]=1 & Cr[l][1][2]=b, Cr[l][2][1]=3 & Cr[l][2][2]=a and Cr[l][3][1]=2 & Cr[l][3][2]=c

3.2 Fitness Function

At first level, to check matching between a two vertex (v_i, v_i') use the following fitness function f_i that computes the dissimilarity between vertex v_i of design pattern graph and the vertex v_i' of model graph:

f_i = | label of v_i- label of v_i' | + | label of incoming edges of v_i- label of incoming edges of v_i'| + |label of outgoing edges of v_i- label of outgoing edges of v_i'|.

One can enhance this fitness function f_i considering number of predecessors, number of successors, in-degree and out degree of nodes v_i and v_i'. At the next level, define the fitness function g that computes the dissimilarity of sub-graph of G_1 (G_1') with sub-graph of G_2 (G_2') where Chromosome at this level represents a mapping of vertices of G_1' to vertices of G_2'.

$g_i = g_{i-1} + f_i$ + |label of edges from v_i to all nodes that have been previously included in G_1' – labels of edges from v_i' to all corresponding nodes that have been previously included in G_2'|.

3.3 Crossover

Here, uses the two-point crossover with a repair function since after two-point crossover , may get an offspring (l^{th} individual) that may have Cr[l][i][1]= Cr[l][j][1]= a (say) i.e. mapping node 'a' twice or more. To repair this chromosome, change the repeated nodes in any chromosome with some other node that is not included in that chromosome. For example if Cr[l][i][1]=6 and Cr[l][j][1]=6 and Cr[l][k][1]=6 then will leave Cr[l][i][1]=6 as unchanged but Cr[l][j][1]=6 and Cr[l][k][1]=6 will change and get some other node number like 4 and 9 respectively, where node number 4 and 9 have not been assigned in any Cr[l][m][1] for any 1<=m<=n.

3.4 Mutation

We use exchange mutation with standard probability. The standard mutation frequency, defined as the probability of applying the mutation operator on a ga-gene, is 0.001. For example, to mutate the l^{th} chromosome one can exchange any two nodes Cr[l][i][1] and Cr[l][j][2] and thus interchange the mapping of these two nodes.

3.5 Algorithm Description

Given two graphs, design pattern graph $G_1 = (V_1, E_1, u_1, v_1)$ and model graph $G_2 = (V_2, E_2, u_2, v_2)$, where V_1, V_2 are set of nodes, E_1, E_2, are set of edges, u_1, u_2 are functions assigning labels to nodes and v_1, v_2 are functions assigning labels to edges in G_1 and G_2 respectively.

The algorithm is given below

Input: G_1 and G_2.

Output: Best matching between nodes in DPG and MG (An injective function g: $V_1 \rightarrow V_2$) having minimum fitness function among all chromosomes found in the last generation.

1. n ← the number of vertices in G_1

2. m ← the number of individuals in a population

3. For i=1 to n-1 do

3.1 For j=1 to m do

Cr[j][i][1] = random value from vertex of G_1 not Occurring in any previous i^{th} step

Cr[j][i][2] = random value from vertex of G_2 not Occurring in any previous i^{th} step

3.2 Apply genetic algorithm on all Cr[j][i] with fitness Function described in SinglePairFitness(Cr[j][i][1], Cr[j][i][2])

3.3 If (i==1)

g[j][i]=f[j][i]

Else

g[j][i]=calculate fitness of each individual Representing sub-graph of G_1described in SubSolFitness(g[j][i-1],f[j][i], Cr[j][i][1], Cr[j][i][2])

3.4 Store some fixed number of best individuals with their Fitness g[j][i].

3.5 If (i==n-1)

Apply crossover and mutation on the population of the current generation and select the best m individuals from all these chromosomes generated after crossover and mutation.

3.6 If termination condition is not satisfied

Go to 3.2

Else

Print the best individual with its fitness g[j][i].

End

Procedure: SinglePairFitness (Cr[j][i][1], Cr[j][i][2])

f[j][i] = | label of Cr[j][i][1]- label of Cr[j][i][2]| + | label of incoming edges of Cr[j][i][1]- label of incoming edges of Cr[j][i][2]| + |label of outgoing edges of Cr[j][i][1]- label of outgoing edges of Cr[j][i][2]|

Procedure: SubSolFitness(g[j][i-1],f[j][i],Cr[j][i][1],Cr[j][i][2])

g[j][i]= g[j][i-1]+f[j][i]+ | label of edges from Cr[j][i-1][1] to Cr[j][i][1] – labels of edges from Cr[j][i-1][2] to Cr[j][i][2] |

4. RELATED WORK

Many tools and techniques have been proposed by researchers working in the area of design pattern detection. Combination of static and dynamic analysis both have been used for analyzing the source code to identify design patterns. We emphasized more on those papers where static analysis is used. DP-Miner [2] is a very famous tool for design pattern detection. This tool discovers design pattern by defining the structural characteristics of each design pattern in terms of weight and matrix. The discovery of design patterns from source code becomes matching between the two matrices. Beside matrix, it uses weight to represent the attributes or operations of each class and its relationships with other classes. The work done by Wendehals [3] shows that design pattern detection process is based on an Abstract Syntax Graph (ASG) representation of the source code. Authors defined a structural and behavioral pattern for each design pattern to enable tool based recognition. The source code is parsed into the ASG representation. This ASG representation is searched for the structural patterns and if matched that structural pattern is annotated as a design pattern candidate. The author's further used dynamic analysis to get confidence on their results getting from static analysis. In [4] authors defined a set of matrices for describing specific features for system as well as design patterns. These features are association and generalization relationships, abstract classes etc. For each feature, a concrete matrix is created for pattern as well as for system. In [4], extended the work of [5] where all of the structural features are treated equally that may dilute the process of design pattern detection. Paper [4] finds the solution to this problem by applying weights of the structural parts of the pattern, so more important parts play more important role in the process of design pattern detection. Researchers proposed some other techniques for design patterns detection using sub-graph isomorphism in [6, 7, 8, 10,11,12,13 and 14]. We also propose a technique for design pattern detection in [9].

5. CONCLUSION

Till now many problems of software development have been solved using soft computing techniques. Design pattern detection is one of the most important problems in reverse engineering. Reusable design may reduce design, development and implementation time if detected and used properly .Further more ample possibility exit in research to improve software development paradigm to organize and harmonize the use of design pattern in more efficient manner to promote reuse and reusability in design phase, based on certain transformation rules and constraints, so that new software is not only easy to design from design point of view but also developed software is easy to enhance. The problem faced by software industry is that there are number of legacy software's with code only and without software requirement document and design. It becomes very difficult to maintain these types of legacy software if requirements are changed or some errors are found. If one has the idea of design patterns implementation in the source code then it would be easy to understand that code and also to maintain it. In this paper, efforts have been made to show the application of genetic algorithm in the design pattern detection. We also applied a hybrid genetic algorithm for sub-graph isomorphism problem which uses an incremental approach for design pattern detection. Moreover, this proposed algorithm is analyzed and discussed with respect to enhance selected software quality attributes related to software reuse and reusability. Previously, we solved this by traditional genetic algorithm as well as using genetic algorithm with neural network. Our future work includes the implementation of this hybrid genetic algorithm and comparison of results with previous genetic algorithms outcomes as well as with other work in this area.

6. REFERENCES

[1] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1994. Design patterns: elements of reusable object-oriented software. Pearson Education.

[2] Dong, J., Lad, D.S. and Zhao, Y., 2007. March. DP-Miner: Design pattern discovery using matrix. In Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the(pp. 371-380). IEEE.

[3] Wendehals, L., 2005. Dynamic Design Pattern Recognition. Disertation Proposal, Department Of Computer Science, University of Padeborn.

[4] Jakubík, J., 2009. Extension for Design Pattern Identification Using Similarity Scoring Algorithm. Information Sciences and Technologies Bulletin of the ACM Slovakia, *1*(1), pp.25-32.

[5] Tsantalis N., Chatzigeorgiou A., Stephanides G., Halkidis S. 2006. Design Pattern Using Similarity Scoring, IEEE transaction on software engineering, vol.32, no.11.

[6] Pande, A., Gupta, M. and Tripathi, A.K., 2010. September. DNIT—A new approach for design pattern detection. In Computer and Communication Technology (ICCCT), 2010 International Conference on (pp. 545-550). IEEE.

[7] Pande, A., Gupta, M. and Tripathi, A.K., 2010. July. Design pattern mining for GIS application using graph matching techniques. In Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on (Vol. 3, pp. 477-482). IEEE.

[8] Pande, A., Gupta, M. and Tripathi, A.K., 2010. A new approach for detecting design patterns by graph decomposition and graph isomorphism. In *Contemporary Computing* (pp. 108-119). Springer Berlin Heidelberg.

[9] Arti chaturvedi , Manjari Gupta ,Sanjay Kumar Gupta , "An Idea towards Improving Design Pattern Detection", International Journal of System and Software Engineering(Volume 3, Issue 2,December 2015), in press.

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

[11] Pande A.,.Gupta M., Tripathi A.K. 2010. A Decision Tree Approach for Design Patterns Detection by Subgraph Isomorphism, In Proc. Of International Conference on Advances in Information and Communication Technologies (ICT 2010) published by Springer, Kochi, Kerela, India. International Journal of Software Engineering and Its Applications Vol. 5 No. 2, April, 2011 55 .

[12] Gupta M., Singh R. R., Pande A., Tripathi A.K. 2011. Design Pattern Mining Using State Space Representation of graph matching, CCSIT, published by Springer,Banglore, India.

[13] Gupta M., 2011.Design Pattern Mining Using Greedy Algorithm for Multileveled Graphs, International Joint Conference on Information and Communication Technology, Bhubaneswar, IPM Pvt. Ltd, India.

[14] Gupta M., Singh R.R, Tripathi A.K. 2010. Design Pattern Detection using Inexact Graph Matching, International Conference on Communication and Computational Intelligence, Tamilnadu.