

Compound Record Clustering Algorithm for Design Pattern Detection by Decision Tree Learning

Jing Dong
Computer Science Department
University of Texas at Dallas
Richardson, TX 75083, USA
jdong@utdallas.edu

Yongtao Sun
American Airlines
4333 Amon Carter Blvd
Fort Worth, TX 76155, USA
Yongtao.Sun@aa.com

Yajing Zhao
Computer Science Department
University of Texas at Dallas
Richardson, TX 75083, USA
yxz045100@utdallas.edu

Abstract

Recovering design patterns applied in a system can help refactoring the system. Machine learning algorithms have been successfully applied in mining data patterns. However, one of the main obstacles of applying them for design pattern detection is the difficulty of collecting training examples. Unlike other applications, a design pattern instance typically includes a group of classes with certain relationships. Thus, the possible combinations of the group of classes can be enormous which results in huge training sets making the application of machine learning algorithms impracticable. In this paper, we propose an innovative method using matrix transformations to cluster the training examples. Our method can significantly reduce the size of training examples, thus making it possible to be efficiently applied in machine learning algorithm.

KEYWORDS: design pattern, machine learning, decision tree, training example, detection

1. Introduction

Each design pattern typically includes a group of classes relating to each other in some structure and behavior in certain ways. Pattern-related information is manifested by the role each class plays in the pattern. However, such role information is normally lost when a design pattern is applied in a system. Recovering pattern-related information from system design or source code has several benefits. First, it may help to understand software systems based on the patterns. Second, it can assist the refactoring of the systems. Third, new design patterns may be discovered. Different approaches have been proposed to recover design patterns from software systems. However, there lack approaches based on machine learning techniques.

As a popular inductive machine learning algorithm, decision tree learning algorithm [8,9,11] has been successfully applied to many applications, such as pattern matching, weather forecast and virus classification. Decision tree algorithm extracts useful rules and pattern information from the training examples that must be pre-classified by an expert (or a supervisor). It is normally easy to collect the training examples by recording the related attribute values as one single record. For example,

whether a person will play tennis is decided by the outlook, temperature, humidity, and wind of a day. Thus, the outlook, temperature, humidity and wind attribute values can be collected together as one entry. Based on the classification of these collected entries, a decision tree prediction model can be constructed by applying the decision tree algorithms. However the learning problem of design pattern detection is more complicated which involves a group of classes with relationships. Each class corresponds to a record. The potential relationships among the group of classes (records) of a pattern can be a large number. Nevertheless, only one set of relationships is typically valid for the pattern. In other words, the training example for decision tree algorithm here is a combination of training records (classes) connected by some relationships/links, instead of a single record (class). This is a compound records learning problem involving multiple training records. It is not easy to classify all the potential combinations within a set of classes, especially when considering the roles of different relationship.

In this paper, we propose an innovative data preprocessing method to tackle the compound record learning problems. We simplify the compound records learning to some basic models by clustering the training records with the same attribute/relationship sequences. In this way, a decision tree can be built based on the clustered training examples. The computation complexity of training example classification is dramatically reduced.

The rest of this paper is organized as follows: Section 2 and 3 present our learning module architecture and the detail pre-processing algorithms. An example of design pattern detection is provided in Section 4. The last two sections cover related work and conclusions.

2. Learning Architecture

Our main goal is to apply decision tree algorithms to the compound records learning problem for design pattern detection by learning the classification rule for composite training examples from a set of collected records efficiently. Different from typical problems solved by decision tree algorithm, there are some characteristics of the problem of design pattern detection: first, there are vast varieties of implementations for each design pattern; second, pattern features are encoded in the entities (class/interface/object) and the relationships among these

entities; third, the relationships among the entities form a complicate graph while each entity may only has a few attributes to characterize a specific pattern in the graph.

While the decision tree scheme is a good approach to generalize the rules, it is hard to apply to pattern detection problem directly. Most design pattern information is encoded in the graph including both entity and relationship information, while the collected training records are usually the individual entity and the corresponding relationship information to others entities. We propose a learning architecture shown in Figure 1. Taking the original data set as input, our preprocessing algorithm clusters the training examples with the same set of attributes into a compound record and computes the number of its occurrences, which can be fed into any decision tree algorithms for concept learning. In this way, the decision tree algorithm can be applied for design pattern detection.



Figure 1 Structured data pattern learning architecture

3. Preprocessing Algorithm

One of the main obstacles of applying decision tree algorithm for design pattern detection is that the training examples include multiple records with some specific relationships. In this section, we propose a preprocessing algorithm, called compound record algorithm, which encodes the records and their relationships into matrix. Through matrix transformation, our approach can significantly reduce the size of a training set.

Section 3.1 introduces the base case of our algorithm that includes only a single attribute and single relationship in the training set. In this case, we assume each training record has only one attribute/feature and relates to other records via only a single relationship. Section 3.2.1 extends our algorithm to multiple-attribute case. Each record can have one or more attributes while each record still relates to other records via a single relationship. Section 3.2.2 takes one step further to consider multiple relationships, i.e., each record may relate to other records through more than one relationship. This is the case that can be applied to design pattern detection.

3.1. Basic algorithm

The training set of the decision tree learning algorithm may include attributes and their relationships. Let us first consider the basic case where there is only a single attribute and single relationship in each record of the training set. In this case, each record relates to other records via a single relationship. There is only one kind

of relationship. A pattern is a sequence of records, each of which with a specific attribute value, connected to each other by the relationship. The diagram in Table 1 shows an example which includes 7 records, each of which connects to each other by a relation. Consider the 7 records as 7 classes and the relationship as the association relationship in a class diagram. Each class has only one attribute specifying whether it is an abstract class or concrete class. For instance, Table 1 specifies that Class 1 is a concrete class and Class 2 is an abstract class. We only consider one kind of relationship, association, in this example. Thus, Class 1 associates with Classes 3, 4, 5.

Class No.	Attribute	Association Relationship	Diagram
1	C	3, 4, 5	
2	A	1, 4, 6, 7	
3	A	2, 4, 6	
4	A	5, 7	
5	C	2, 3, 6, 7	
6	C	1, 4, 7	
7	A	1, 3	

Table 1 Single Attribute and Single Relationship

Suppose the expert can classify a training set that includes any 3 classes related by the association relationship. In other words, for each directed path with three classes in the diagram of Table 1, the expert can classify it with either “Positive” or “Negative” value which is the basis for building a decision tree. However, it is hard to know how many such paths exist in an application, such as that in Table 1. If there are m classes, each of which associates to k classes in average, then the size of a n -class combination (path) is $O(m \times k \times k \times \dots \times k) = O(mk^{n-1})$. Consider the extreme case that every class associates with all other classes, then the complexity is $O(m \times (m-1) \times (m-1) \times \dots \times (m-1)) \approx O(m^n)$. Thus, the size of the classification can be very large.

We observe that the expert classification of training set only consider the combinations of attributes of the records, rather than the exact sequence order of the records. If we know the occurrence of a specific combination (such as “ACA” that represents 3 classes whose attributes are “Abstract” “Concrete” “Abstract”, respectively), together with its classification result (such as “Positive” or “Negative”), we have enough information to build the decision tree. In this case, there are a maximal of $2^3 = 8$ such kind of combinations. Thus, the computation complexity is dramatically reduced. One assumption is that the expert always classifies each specific combination with the same value no matter how many times it occurs. The key issue is how to get the occur-

rence of each combination when building the decision tree.

Definition 1 (Occurrence): V_i is the starting state vector for state i , where V_{ij} is the number of occurrence for class j in state i . While at the initial state, every class is qualified to be placed on state 1. Thus, the value is 1 for every class, i.e., $V_1 = (1 \ 1 \ . \ . \ . \ 1)$.

Definition 2 (Filter Matrix): The filter matrix F_i for state i is a diagonal matrix where $(F_i)_{jj} = 1$ means the j record remains in the transition on state i . If only the first record remains in the transition, then

$$F = \begin{pmatrix} 1 & 0 & . & . & 0 \\ 0 & 0 & & & \\ . & & & & \\ . & & & & \\ 0 & & & & 0 \end{pmatrix} \text{ and } F = I = \begin{pmatrix} 1 & 0 & . & . & 0 \\ 0 & 1 & & & \\ . & & & & \\ . & & & & \\ 0 & & & & 1 \end{pmatrix} \text{ if}$$

all records remain.

Definition 3 (Filtered Vector): V_i' is the filtered vector for state i , where $V_i' = V_i \times F_i$. If $F_i = I$, then $V_i' = V_i \times I = V_i$

Definition 4 (Transition

Matrix): M is the transition matrix where M_{jk} value means the record j can connect (transit) to record k . For instance, the transition matrix of the example in Table 1 is:

$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Based on the previous definitions, we introduce the following algorithm that is illustrated by two examples.

Algorithm:

Input:

- n is the record size of target combination (which is 3 for our example)
- Transition Matrix M
- the target combination (such as "ACA")

Output:

- the occurrence of the target combination.

Procedure:

```

init i = 1;
init  $V_1 = (1 \ 1 \ . \ . \ . \ 1)$ ;
repeat until  $i > n$ 
    if ( $i > 1$ )  $V_i = V_{i-1} \times M$ ;
    find  $F_i$  based on the attribute value of target combination in state  $i$ ;
     $V_i' = V_i \times F_i$ ;
     $i = i + 1$ ;
occurrence =  $V_n' \times (1 \ 1 \ . \ . \ 1)^T$ 

```

Example 1: Calculate the occurrence of three-class combinations with "ACA" value. "ACA" means the first and the third classes are abstract classes and the second class is a concrete class.

Following our algorithm, we start with state 1.

- State 1, $i = 1$: $V_1 = (1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$ represents that every class can be considered as a candidate of the first class (state) in the three-class combination at the initial state. Since classes 2,3,4,7 are abstract class ("A") and classes 1,5,6 are concrete class ("C"), the filter matrix is as follows

$$F_1 = F_3 = \begin{pmatrix} 0 & 0 & . & . & 0 \\ 0 & 1 & & & \\ . & & 1 & & \\ . & & & 1 & \\ & & & & 0 \\ & & & & & 0 \\ 0 & & & & & & 1 \end{pmatrix} \quad F_2 = \begin{pmatrix} 1 & 0 & . & . & 0 \\ 0 & 0 & & & \\ . & & 0 & & \\ . & & & 0 & \\ & & & & 1 \\ & & & & & 1 \\ 0 & & & & & & 0 \end{pmatrix}$$

Therefore, the filtered vector is $V_1' = V_1 \times F_1 = (0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1)$ which shows that only classes 2,3,4,7 can be the first class because the first class in three-class combination is an abstract class ("A"). Starting from classes 2,3,4,7, the following classes in state 2 can be reached through the association relationship: $V_2 = V_1' \times M = (2 \ 1 \ 1 \ 2 \ 1 \ 2 \ 2)$, where there are two class 1, one class 2, one class 3, two class 4, one class 5, two class 6 and two class 7.

- State 2 and State 3

Continuing with the algorithm, we can calculate the occurrence of "ACA" cluster as:

$$V_3' = V_3 \times F_3 = V_2' \times M \times F_3 = V_2 \times F_2 \times M \times F_3 = (0 \ 1 \ 3 \ 4 \ 0 \ 0 \ 3)$$

$$\text{Occurrence(ACA)} = V_3' \times (1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)^T = 11$$

No	First Class	Second Class	Third Class	Occurrence	Classification
1	A	A	A	8	Positive
2	A	A	C	7	Negative
3	A	C	A	11	Negative
4	A	C	C	5	Positive
5	C	A	A	10	Positive
6	C	A	C	8	Negative
7	C	C	A	7	Negative
8	C	C	C	3	Negative

Table 2 Classification of Clustered Training Examples

Example 2: Calculate the total number of occurrences of all three-class combinations.

In this case, all classes remain in the transition. The filter matrix is I .

$$V_3' = V_1 \times I \times M \times I \times M \times I = V_1 \times M^2 = (9 \ 5 \ 9 \ 11 \ 7 \ 7 \ 11)$$

Thus, the total number is $V_3' \times (1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)^T = 59$.

Based on our algorithm, we can easily get the list of all non-zero occurrences of the combinations of classes with the same set of attributes. The expert can classify based on these combinations (training examples). A decision tree can be built straightforwardly.

3.2. Extended Algorithm

In section 3.1, we presented the basic case of our algorithm. In this section, we study the more complicated cases involving multiple attributes and relationships.

Class No.	Abstract	Static	Association
1	C	N	3, 4, 5
2	A	N	1, 4, 6, 7
3	A	S	2, 4, 6
4	A	S	5, 7
5	C	N	2, 3, 6, 7
6	C	N	1, 4, 7
7	A	N	1, 3

Table 3 Multiple attributes training data.

3.2.1. Multiple Attributes

In the basic case of our algorithm, each class (training record) only has one attribute, e.g., “Abstract”. Our algorithm can be easily extended to multiple attributes per training record. Assume each class has a second attribute to specify whether it is a static class, where “S” stands for “Static Class” and “N” indicates it is not a static class. In this section, we investigate how the additional attributes of a class affect the classification. Table 3 shows the training data with both the “Abstract” and “Static” attributes information collected for each class.

We can still directly apply the basic algorithm for the multiple-attribute case. The only difference is that the multiple attributes need to be considered together when deciding the filter matrix. For example, to compute the occurrence of pattern “ANASCN”, we use “AN”, “AS”, and “CN” to decide the filter matrix F_1 , F_2 , and F_3 , $F_1 = \begin{pmatrix} 0 & 0 & . & & 0 \\ 0 & 1 & & & \\ . & & 0 & & \\ & & & 0 & \\ 0 & & & & 0 \\ 0 & & & & & 1 \end{pmatrix}$ respectively. In this example, since both class 2 and class 7 have the “AN” value, then the filter matrix for state 1 is

After applying the basic algorithm, our algorithm generates 25 non-zero occurrence combinations. A decision tree can be easily built based on these 25 combination patterns from expertise.

3.2.2. Multiple Relationships

When multiple relationships exist between the training records (classes in our example), we need to consider not only the attribute combinations, but also different relationships between these records in a combination. Let us extend the example in Section 3.1 with a new type of

relationship, delegation. When class A delegates to class B, one method of class A invokes another method in class B. Table 4 displays an example of the training set with one attribute and two relationships (association and delegation). When classifying the training set, the expert needs to consider both the attribute values of the classes and the specific relationships between the classes. In the final clustered training example combination, the attributes set should have “FirstClassAbstract”, “SecondClassAbstract”, “ThirdClassAbstract” and “RelationshipSequence”, where “RelationshipSequence” represents a unique relationship order to construct the target attribute combination.

Let the association and delegation relationships be represented by symbols r_1 and r_2 , respectively. There can be $2^2 = 4$ relationships between any two classes A and B: 0 means no relationship; r_1 means A has r_1 relationship with B; r_2 means A has r_2 relationship with B; and r_1r_2 means A has both r_1 and r_2 relationships to B. Let R denotes the four possible relationships between two classes, then a relationship sequence $R_1 \otimes R_2$ represents the relationships between three classes A, B, C where A connect to B via R_1 and B connect to C via R_2 . We only consider the forward relationship sequences in relationship combinations. The relationships sequence “ $r_1 \otimes r_2$ ” is different from “ $r_2 \otimes r_1$ ” and “ $r_2 \otimes r_1r_2$ ”, and thus may be classified differently by an expert.

Class No.	Abstract	Association r_1	Delegation r_2
1	N	3, 4, 5	2,3
2	Y	1, 4, 6, 7	1,3
3	Y	2, 4, 6	1,2
4	Y	5, 7	6
5	N	2, 3, 6, 7	7
6	N	1, 4, 7	4
7	Y	1, 3	5

Table 4 Training data of multiple-relationship

Now the key issue is how to cluster the attribute combinations including relationship sequence. We need to make a little change to the basic algorithm. Originally, the value of M_{ij} is either 0 or 1 in the transition matrix M where 1 means the relationship exists and 0 stands for not existing because we only consider one kind of relationship at that time. With multiple kinds of relationships, the value of M_{ij} should be able to represent all kinds of relationships between record i and record j . For the example in Table 4, the value of M_{ij} between i and j should be one of $\{0, r_1, r_2, r_1r_2\}$. Thus, the transition matrix turns into

$$M = \begin{pmatrix} 0 & r_2 & r_1r_2 & r_1 & r_1 & 0 & 0 \\ r_1r_2 & 0 & r_2 & r_1 & 0 & r_1 & r_1 \\ r_2 & r_1r_2 & 0 & r_1 & 0 & r_1 & 0 \\ 0 & 0 & 0 & 0 & r_1 & r_2 & r_1 \\ 0 & r_1 & r_1 & 0 & 0 & r_1 & r_1r_2 \\ r_1 & 0 & 0 & r_1r_2 & 0 & 0 & r_1 \\ r_1 & 0 & r_1 & 0 & r_2 & 0 & 0 \end{pmatrix}$$

Let us use the same ‘‘ACA’’ example in Section 3.1. We already know the filter matrix as:

$$F_1 = F_3 = \begin{pmatrix} 0 & 0 & . & . & 0 \\ 0 & 1 & & & \\ . & & 1 & & \\ . & & & 1 & \\ 0 & & & & 0 \\ 0 & & & & 1 \end{pmatrix}, F_2 = \begin{pmatrix} 1 & 0 & . & . & 0 \\ 0 & 0 & & & \\ . & & 0 & & \\ . & & & 0 & \\ 0 & & & & 1 \\ 0 & & & & 0 \end{pmatrix}$$

Following the algorithm, we can get:

$$\begin{aligned} V_1' &= V_1 \times F_1 = (0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1) \\ V_2 &= V_1' \times M = (r_1 r_2 + r_2 + r_1 \quad r_1 r_2 \quad r_2 + r_1 \quad 2r_1 \quad r_1 + r_2 \quad 2r_1 + r_2 \quad 2r_1) \\ V_2' &= V_2 \times F_2 = (r_1 r_2 + r_2 + r_1 \quad 0 \quad 0 \quad 0 \quad r_1 + r_2 \quad 2r_1 + r_2 \quad 2r_1) \\ (V_3')^T &= (V_3 \times F_3)^T = (V_2' \times M \times F_3) = \begin{pmatrix} 0 \\ (r_1 r_2 + r_2 + r_1) \otimes r_2 + (r_1 + r_2) \otimes r_1 \\ (r_1 r_2 + r_2 + r_1) \otimes r_2 + (r_1 + r_2) \otimes r_1 \\ (r_1 r_2 + r_2 + r_1) \otimes r_1 + (2r_1 + r_2) \otimes r_2 \\ 0 \\ 0 \\ (r_1 + r_2) \otimes r_1 r_2 + (2r_1 + r_2) \otimes r_1 \end{pmatrix} \end{aligned}$$

Therefore, the total number of relationships for combination ‘‘ACA’’ is

$$\begin{aligned} V_3' \times (1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)^T &= (r_1 r_2 \otimes r_2) + (r_2 \otimes r_2) + (r_1 \otimes r_2) \\ &+ 5(r_1 \otimes r_1) + 4(r_2 \otimes r_1) + (r_1 r_2 \otimes r_1 r_2) + 3(r_2 \otimes r_1 r_2) \\ &+ 4(r_1 \otimes r_1 r_2) + (r_1 r_2 \otimes r_1) \end{aligned} \quad \text{where}$$

each $R_1 \otimes R_2$ item is a valid instance of the ‘‘ACA’’.

Thus, the total number of the ‘‘ACA’’ is 21. The detail of the ‘‘ACA’’ with the relationships is shown in Table 5, which can be used for classification purpose.

No.	Attribute Pattern	Relationship First-Second (R_1)	Relationship Second-Third(R_2)	Occurrence
1	ACA	Assoc + Delegation	Delegation	1
2	ACA	Delegation	Delegation	1
3	ACA	Association	Delegation	1
4	ACA	Association	Association	5
5	ACA	Delegation	Association	4
6	ACA	Assoc + Delegation	Assoc + Delegation	1
7	ACA	Delegation	Assoc + Delegation	3
8	ACA	Association	Assoc + Delegation	4
9	ACA	Assoc + Delegation	Association	1

Table 5 Training Example for Multiple-Relationship

4 Experiment results

Design pattern is a recurring solution to a standard software problem. Each design pattern typically includes several classes structured with multiple relationships. Each class in the pattern may have multiple attributes. Thus, it corresponds to the multiple-attribute and multiple-relationship case of our algorithm for preprocessing the training examples.

Consider a software system with 1000 classes. The maximum size of training set of three-class combination could be $1000^3 = 1 \times 10^9$. By partitioning the training examples into the compound record combination groups, the training set size can be reduced greatly because it is not related to the size of the software anymore. When

more class attributes/relationships are considered, nevertheless, the pattern combination size is also increased, which may make the calculation of compound record algorithm less practical. For example, if taking two class attributes and five relationships into consideration, the total combination size could be $(2 \times 2)^3 \times 2^5 \times 2^5 \approx 65536$, which is not a small effort although improved from 1×10^9 . Most of these pattern groups have zero or only a few instances because some relationships cannot coexist. For instance, the association and generalization relationships cannot co-exist between two classes.

One way to get the training set more sufficiently is to apply a sampling approach on the top of compound record algorithm. We may consider only a small number of the critical class/relationships attributes to partition all training examples. For example, we can only take the ‘‘hasChildren’’ and ‘‘association’’ relationships into account if we feel these two relationships may contribute to the final design pattern detection. Totally it will be $4 \times 4 = 16$ pattern groups for any connected three-class set. In addition, we can calculate the occurrence of each group based on the compound record algorithm. For each group, it may contain both the positive and negative instances, so we cannot label the whole group as one pattern candidate. Instead we take samples based on the occurrence of each group to represent that group in the whole training population. There are totally 41717 connected three-classes set found in the AWT [13], and we can take approximately 1000 training examples as the final training set. We randomly pick up proportional number of instances in each group to build up the final training set. There are exceptions for the low occurrence groups, since their proportional sample number is close to 0 while their group may hold critical training examples. Because it won’t cause too much effort for classification due to its small size, we can either increase the sampling rate, or just take the whole group into the final training set.

Software systems	Test Set Size	False positive	True negative	Correct rate
AWT	1066	5	2	99.3%
JUnit	1052	4	0	99.6%
JHotDraw	995	8	1	99.1%

Table 6 Experimental results

After finalizing the training set, we retrieve the full class/relationships information for each instance in the final training set to feed into the decision tree algorithm. The size of training set is reduced significantly compared with the effort to classify any random three-class training examples from the software without losing too much generality for pattern distribution. Table 6 shows the test result for AWT, JUnit [15] and JHotdraw [14] based on

the decision tree model build from AWT final training set for detecting the Strategy pattern.

5 Related Work

A comprehensive review of different design pattern detection approaches has been presented in [5]. Machine learning algorithms, such as decision tree and neural network, have been applied to classify the potential design pattern candidates in [6,7]. Neither approach can be fully automated. Manual collection of some features and human intervention are required. In addition, the training set was obtained by another design pattern detection tool in [6], which used machine learning algorithm to fine-tune their results from that tool. The main goal of their application of machine learning techniques is to eliminate potential false positive results from their other tool, rather than to detect design patterns. Thus, their training example set is limited by their tool, which may miss some true pattern cases (true-negative). On the other hand, our approach presents a novel way to cluster the training examples that include all possible cases. We are able to present an approach to design pattern detection purely based on machine learning techniques.

Matrices have been used as intermediate representations in some approaches in design pattern detection [2,3,12], where both pattern information and system information are encoded into matrixes. Both [3] and [12] compare the similarity between the pattern matrix and potential candidate from the system. While a graph similarity theory is used to calculate the similarity between two classes in [12], a template matching method for the similarity comparison of two matrixes is applied in [3]. In contrast, we presented a new approach using machine learning techniques for design pattern detection.

6 Conclusions and Future Work

In this paper, we present a new compound records preprocessing algorithm for decision tree learning algorithm. It enables clustering the training records with the same attributes, which help the creation of final decision tree for design pattern detection. Our approach helps efficiently generate the training examples from software system designs. Three cases have been introduced to cover the single attribute single relationship, the multiple attributes single relationship and the multiple attributes multiple relationships scenarios. The possibility of the occurrences of the training examples with the same set of attributes can be efficiently calculated for decision tree construction purpose. An example of the detection of the Strategy pattern is presented. Our approach can detect design pattern instances from software system by learning from the training examples.

In the future, we plan to extend our research to cover more complicated scenarios and provide a large case study of design pattern detection. More different kinds of

relationships (not necessarily sequential connected) will be considered. In addition, other learning schemes, such as neural network and dynamic Bayesian network, could be applied. Numerical analysis and comparison will be implemented based on those theoretical models so that fully automatic design pattern learning cases will be addressed. The detection results can be visualized by our visualization techniques [4].

References

- 1 O. Cappe, E. Moulines, T. Ryden. Inference in Hidden Markov Models, Springer, 2005. ISBN 0-387-40264-0.
- 2 J. Dong, D.S. Lad and Y. Zhao, DP-Miner: Design Pattern Discovery Using Matrix, the Proceedings of the Fourteenth Annual IEEE International Conference on Engineering of Computer Based Systems, USA, March 2007.
- 3 Jing Dong, Yongtao Sun, Yajing Zhao, Design Pattern Detection By Template Matching, the Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC), Ceará, Brazil, March 2008.
- 4 Jing Dong, Sheng Yang and Kang Zhang, Visualizing Design Patterns in Their Applications and Compositions, IEEE Transaction on Software Engineering (TSE), Volume 33, Number 7, pp. 433-453, July 2007.
- 5 J. Dong, Y. Zhao, and T. Peng, Architecture and Design Pattern Discovery Techniques – A Review, Proceedings of International Conference on Software Engineering Research and Practice (SERP), USA, June 2007.
- 6 R. Ferenc, A. Beszedes, L. Fulop and J. Lele, Design Pattern Mining Enhanced by Machine Learning, 21st IEEE International Conference on Software Maintenance (ICSM'05) pp. 295-304.
- 7 Y.-G. Gueheneuc, H. Sahraoui, and F. Zaidi, Fingerprinting Design Patterns, Proc. 11th Working Conf. on Reverse Eng. (WCRE'04), Nov. 2004.
- 8 R. Kothari and M. Dong. "Decision Trees for Classification: A Review and some new results". University of Cincinnati, 2000.
- 9 T. M. Mitchell, "Machine Learning" Chapter 3, p52-p78. 1997, ISBN 0070428077
- 10 L. Pachter and B. Sturmfels. "Algebraic Statistics for Computational Biology". Cambridge University Press, 2005. ISBN 0-521-85700-7.
- 11 J. R. Quinlan, Induction of Decision Trees, 1986. ISBN 0885-6125.
- 12 N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis, "Design Pattern Detection Using Similarity Scoring" IEEE transaction on software engineering, Vol. 32, No.11, November 2006.
- 13 Java.awt resource information, September 2006, <http://java.sun.com/j2se/1.5.0/docs/guide/awt/index.html>.
- 14 JHotDraw Start Page. <http://www.jhotdraw.org/>
- 15 JUnit, Testing Resources for Extreme Programming. <http://www.junit.org/>