A Method of Case-Based Creative Design

Ning Zhang

School of Information, Central University of Finance and Economics, Beijing 100081, P.R.China

Abstract

Case-based reasoning can be used to explain many creative design processes since creativity stems from using old solutions in novel ways. However the process of case-based creative design is usually not very efficient and not easy to implement due to the complexity of creative design problems. This paper presents a method to support the whole case-based creative design process. Object-oriented techniques are used to represent cases. Fuzzy logic and self-organizing feature map are used to improve the accuracy and speed of case retrieval. Constraint rules and genetic algorithm are used to improve the validity and speed of case combination.

Keywords: Creative design, Case-based reasoning, Object-oriented, Self-organizing feature map, Genetic algorithm

1. Introduction

Creative design activities can be described by contrasting them to routine design activities. In general, routine design repeats old designs in obvious ways, adapting them by well-known and often-applied adaptation strategies. Routine design assumes a completely specified problem is given and little effort is applied to elaborating or designing a feasible specification. Creative design, on the other hand, includes a process of designing the design specification [1], going from an incomplete, contradictory, and under-constrained description of what needs to be designed to one with more detail, more concrete specifications, and more clearly defined constraints. Creative design also often includes a process of generating and considering several alternatives, weighing their advantages and disadvantages, and sometimes incorporating pieces of one into another. It involves using well-known design pieces in unusual ways or modifying well-known designs in unusual ways.

Since creativity stems from using old solutions in novel ways, case-based reasoning can be used to explain many creative design processes [2]. Research in case-based reasoning has provided extensive knowledge of how to reuse solutions to old problems in new situations, how to build and search case libraries (for exploration of design alternatives), and how to merge and adapt cases. Many of the activities of creative designers can be modeled by extending designed routine problem solving processes that exist in current case-based systems [3]-[5].

However, the process of case-based creative design, from case representation, case retrieval to case adaptation, is usually not very efficient and not easy to implement due to the complexity of creative design problems. There is not a complete and systematic method to support case-based creative design process. This paper aims to provide such a method to support the whole process effectively and efficiently.

2. Object-oriented case representation

Object-oriented case representation is particularly suitable for complex domains [6]. Cases are represented as collections of objects, each of which is described by a set of attribute-value pairs. We distinguish simple attributes, which have a simple type like Integer or Symbol, from composite attributes, which are objects themselves. Composite attribute represents a directed part-of relation between the object that defines the composite attribute and the object to which it refers. Part-of relations can be described in a hierarchy. A case object locates in the first level. The objects located in the middle levels are parts of the objects located in the higher level. The objects located in the lowest level contain only simple attributes.

We define a case basically as a tuple C=<N, O, B, R> where

N: case name used to mark the case uniquely;

O: a finite set of objects which can represent this case, $O = \{O_1, O_2, ..., O_l\}$, O_i is a part of the object O, i =1, 2, ..., *l*; $O_i = \{O_{i1}, O_{i2}, ..., O_{im}\}$, O_{ij} is a part of the object O_i, j =1, 2, ..., m; keep doing so until the objects only contain simple attributes, $O_x = \{O_x.A_1, O_x.A_2, ..., O_x.A_n\}$, x may be a subscript of one to several digits, A_k is a simple attribute of the object O_x, $O_x.A_k$ denotes the value of the attribute A_k in the object O_x.

B: background and environment information in text format of the case which cannot be quantified;

R: complete solution to the case and professionals' evaluations in text format of the case.

3. Intelligent case retrieval

The major task of case retrieval is to compare the new problem with each old case in the case base and to find the most suited one for the new problem.

3.1. Similarity measurement

We use the method of assessing similarity to measure the matching degree between the new problem and an old case. Suppose the new problem P is represented by the object O^*, $O^* = \{O^*_1, O^*_2, ..., O^*_l\}$, and an old case C in the case base is represented by the object O, $O= \{O_1, O_2, ..., O_l\}$. The similarity between O^* and O can be computed as Eq. (1).

$$S = SIM(O^*, O) = \sum_{i=1}^{l} W_i \times SIM(O^*_i, O_i) \quad (1)$$

where W_i denotes the weight factor given to the object O_i, $\sum_{i=1}^{l} W_i = 1$; $SIM(O^*_i, O_i)$ denotes the similarity between the two objects O^*_i and O_i, which can be computed as Eq. (2).

$$S_i = SIM(O^*_i, O_i) = \sum_{j=1}^{m} W_{ij} \times SIM(O^*_{ij}, O_{ij}) \quad (2)$$

where W_{ij} denotes the weight factor given to the object O_{ij}, $\sum_{j=1}^{m} W_{ij} = 1$; $SIM(O^*_{ij}, O_{ij})$ denotes the similarity between the two objects O^*_{ij} and O_{ij}.

We can keep doing so until the objects only contain simple attributes. Suppose O^*_x is an object of the new problem P, and O_x is an object of the old case C. The similarity between them can be computed as Eq. (3).

$$S_x = SIM(O^*_x, O_x) = \sum_{k=1}^{n} W_{xk} \times SIM(O^*_x.A_k, O_x.A_k) \quad (3)$$

where W_{xk} denotes the weight factor given to the simple attribute A_k, $\sum_{k=1}^{n} W_{xk} = 1$; $SIM(O^*_x.A_k, O_x.A_k)$ denotes the similarity between the two attribute values $O^*_x.A_k$ and $O_x.A_k$, which can be computed using fuzzy logic.

The basic idea of fuzzy logic is to fuzz absolute membership relations in cantor sets. The membership degree can be any value between zero and one rather than zero or one. Membership function plays a very important role in fuzzy logic. It uses classical mathematical methods to express the uncertainty in fuzzy sets.

Similar relation is a kind of fuzzy relation. The similar relation between two elements is not just similar or unsimilar; it should be measured by similarity. In this paper, we treat a similar relation between attribute values as a fuzzy relation characterized by fuzzy set named "similar". The similarity is the membership degree which can be computed using membership function as Eq. (4) shows.

$$SIM(V_1, V_2) = \mu_R(V_1, V_2) \quad (4)$$

where $SIM(V_1, V_2)$ denotes the similarity between the two attribute values; R denotes the fuzzy set "similar"; μ_R denotes membership function.

Simple attributes can have numeric type or symbolic type. For numeric type, some typical functions can be selected as membership functions. If the input of an attribute value of the new problem means that the corresponding attribute value of an old case should be close to it, we can choose a membership function which can describe the concept "close to" well, like normal distribution function.

The attribute value should first be normalized as Eq. (5).

$$V' = \frac{V}{V_{\max} - V_{\min}} \quad (5)$$

where V denotes the attribute value before normalization; $V_{\max}$ and $V_{\min}$ denote the maximum value and the minimum value of the attribute; V' denotes the attribute value after normalization.

If A_k is an attribute which has numeric type, the similarity between $O_x.A_k$ and $O^*_x.A_k$ can be computed as Eq. (6).

$$SIM(O_x.A_k, O^*_x.A_k) = \mu_R(O_x.A_k', O^*_x.A_k') \quad (6)$$

where $O_x.A_k'$ and $O^*_x.A_k'$ denote the values of $O_x.A_k$ and $O^*_x.A_k$ after normalization.

If normal distribution function is selected as the membership function, the similarity can be computed as Eq. (7).

$$SIM(O_x.A_k, O^*_x.A_k) = e^{-K(O^*_x.A_k' - O_x.A_k')^2} \quad (7)$$

The change curve of the similarity is illustrated in Fig. 1.

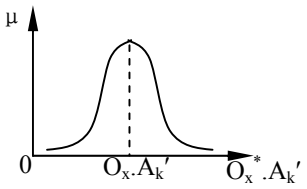

Fig.1: Normal distribution membership function.

For symbolic type, the attribute can be classified as name variable or sequence variable. The relationship between attribute values of name variable can be equal or unequal. If A_k is a name variable, the similarity between $O_x.A_k$ and $O^*_x.A_k$ can be computed as Eq. (8).

$$SIM(O_x.A_k, O_x^*.A_k) = \begin{cases} 0 & \text{if } O_x^*.A_k \neq O_x.A_k \\ 1 & \text{if } O_x^*.A_k = O_x.A_k \end{cases} \quad (8)$$

There is a sequential relation between the attribute values of sequence variable. The concepts like "greater than" or "less than" make sense. If an attribute value V_1 is in front of another attribute value V_2 sequentially, V_1 is "less than" V_2, which can be marked as $V_1<V_2$.

The attribute values of sequence variable are discrete, and the similarity between them can not be computed numerically. If the input of an attribute value of the new problem means that the corresponding attribute value of an old case should be close to it, the membership degree can be given by professionals according to their knowledge and experience.

If A_k is a sequence variable and there are *n* elements (attribute values) in the universe (V_i, i = 1, ..., n, $V_1<V_2<...<V_n$), the similarity between each two of them can be given by professionals. The similarity matrix of order *n* is as follows.

$$R = (r_{ij})_{n\times n} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} \end{pmatrix}$$

where r_{ij} denotes the similarity between V_i and V_j, $r_{ij} = \mu_R(V_i, V_j)$.

Once the similarity matrix is given, the similarity between $O_x.A_k$ and $O_x^*.A_k$ can be computed as Eq. (9).

$$SIM(O_x.A_k, O_x^*.A_k) = r_{ij}$$
$$\text{if } O_x^*.A_k = V_i \text{ and } O_x.A_k = V_j \quad (9)$$

Although our definition process of similarity measurement is top-down, the actual computation process of similarity measurement is bottom-up.

3.2. Hierarchical decomposition of case retrieval

It is possible that no case is totally similar with the new problem, i.e., the similarity between every old case and the new problem is less than a given threshold ε. But that does not mean all the cases are useless to the solution of the new problem. We can decompose the new problem hierarchically until the old cases partially similar with the new problem are retrieved, i.e., the similarity between some part of every retrieved case and the corresponding part of the new problem is greater than or equal to ε.

The collection of the old cases that are similar with an object of the new problem is called *similar set*, which is marked as G_x, where x is the subscript of the object, and may denote zero to several digits.

Fig. 2 shows an example of the decomposition of a new problem.

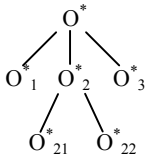

Fig. 2: Decomposition of a new problem.

There is no old case totally similar with the new problem, so the similar set G is empty. The new problem is decomposed to the second level. There are some cases similar with the object O^*_1 or O^*_3, so they are elements in the similar set G_1 or G_3. It is unnecessary to decompose these two parts. There is still no old case similar with the object O^*_2, so the similar set G_2 is empty. The new problem is decomposed to the third level. There are some cases similar with the object O^*_{21} or O^*_{22}, so they are elements in the similar set G_{21} or G_{22}.

If the new problem is decomposed, different cases in the similar sets similar with different parts of the new problem should be combined together to solve the new problem.

3.3. Intelligent clustering based on self-organizing feature map

The computation process of similarity measurement is complex and time-consuming. In order to increase the speed of case retrieval, the old cases in the case base are clustered based on intelligent techniques in this paper. The search process for old cases similar to the new problem is classified into two steps: the new problem is first clustered to identify its class; then similar cases are searched only from the cases in the same class with the new problem. Therefore the searching time can be saved greatly and the consistency between the retrieved old case and the new problem is further assured.

Before clustering creative design cases, which classes can be identified and which class a case should belong to are usually unknown. Therefore we use self-organizing feature map (SOFM) neural network to solve this kind of clustering problem. SOFM adopts unsupervised learning schemes. Given outputs are not necessary in the network. Based on the characteristics of input patterns, the network itself can continually modify the intensity of connection (weights) between neuron cells according to some judgement standards to make the distribution of weight vectors resemble the distribution of samples in the input vector space.

A well known type of SOFM is a Kohonen network [7]. The objective of a Kohonen network is to

map input vectors (patterns) of arbitrary dimension *n* onto a discrete map with 1 or 2 dimensions. Patterns close to one another in the input space should be close to one another in the map: they should be topologically ordered. A Kohonen network is composed of a grid of output units and *n* input units. The input pattern is fed to each output unit. The input lines to each output unit are weighted.

The old cases should be clustered on each level due to the possibility of decomposition of a new problem. Here we give the implementation of any SOFM network for each level, including the design of input level, output level and learning algorithm. Suppose the subscript of the objects of old cases clustered is x, which may denote zero to several digits, and the sample size is N.

(1) Input level

Input level represents all the simple attributes of the object O_x. In order to simplify the network structure, each attribute can be represented by one input unit. Suppose the number of input units is *n*.

The input vector is X, which is composed of input variables: x_1, x_2, ..., x_n. All the input data should be transferred into numbers because SOFM neural network can only deal with numbers.

(2) Output level

Each unit in the output level represents a type of sample. Generally, the units should be enough. Suppose the number of output units is *m*, $m >> n$. For example, m=4*4=16 (as shown in Fig. 3).

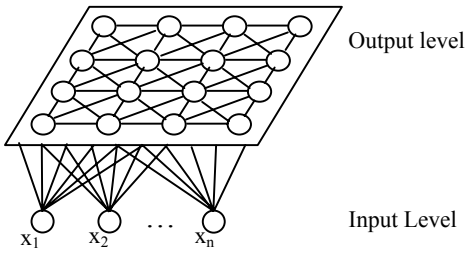

Fig. 3: The structure of Kohonen SOFM network.

The output variable of a unit is y_j (j = 1, …, m). The output value is the matching degree (similarity) between the input vector X and the corresponding weight vector ω_j of this output unit. The value of y_j can be computed as Eq. (10).

$$y_j = SIM(X, \omega_j) = \sum_{i=1}^{n} U_i \times SIM(x_i, \omega_{ij}) \quad (10)$$

where SIM(X, ω_j) denotes the similarity between the input vector X and the weight vector ω_j; SIM(x_i, ω_{ij}) denotes the similarity between the input variable x_i and the weight ω_{ij}; U_i is the weight factor denoting the importance of x_i, $0 \le U_i \le 1$, and $\sum_{i=1} U_i = 1$.

The computation of SIM(x_i, ω_{ij}) is the same as the computation of the similarity between attribute values presented in part 3.1.

(3) Learning algorithm

Suppose the output unit *c* is the best match with the input vector X as Eq. (11) shows.

$$y_c = SIM(X, \omega_c) = \max_j y_j = \max_j SIM(X, \omega_j) \quad (11)$$

The weight vectors ω_c and ω_j ($j \in N_c$, N_c is the adjacent area of *c*) are modified.

The learning algorithm is as follows:

- Initializing $\omega_{ij}(0)$ to small random numbers (i = 1, ..., n, j = 1, ..., m) and initializing learning rate $\alpha(0)$, adjacent area $N_c(0)$ and the times of learning T;
- Selecting a sample $X^p(t)$ from the samples (X^1, X^2, ..., X^N) as the input of the network;
- Computing the value of y_j (j = 1, ..., m) based on Eq. (10) to get the unit *c* as the winning unit whose output value is the maximum value of y_j (j = 1, ..., m);
- Modifying the weights ω_{ij} as Eq. (12);

$$\begin{cases} \omega_{ij}(t+1) = \omega_{ij}(t) + \alpha(t)(x_i^p - \omega_{ij}(t)) & j \in N_c(t) \\ \omega_{ij}(t+1) = \omega_{ij}(t) & j \notin N_c(t) \end{cases}$$
$$i = 1, 2, \ldots, n \quad (12)$$

- Updating $\alpha(t)$ and $N_c(t)$ as Eq. (13) and Eq. (14);

$$\alpha(t) = \alpha(0)(1 - \frac{t}{T}) \quad (13)$$

$$N_c(t) = INT[N_c(0)(1 - \frac{t}{T})] \quad (14)$$

- Going back to step 2 and continuing above steps until the iteration times come to T.

Each weight vector of the network represents the center of a class after the learning process. The network then can be used to classify the samples (old cases) and the new problem.

4. Intelligent case combination

Case combination is not simple adding of different parts of different cases because there are constraint relationships between them. The combined case which combines the most similar cases in different parts together may not satisfy constraint relationships. Therefore the objective of case combination is to generate combined cases totally similar with the new problem and satisfying constraint relationships.

4.1. Integrating constraint rules with object-oriented case

In many creative design activities specific domain knowledge contained in the cases is not sufficient to cope with all requirements of an application. General domain knowledge is often necessary. In object-oriented case representations there may be constraint relationships between different attributes of the same object or different objects. These constraint relationships are very important for verifying the validity of an old case, a new problem or a combined case.

Constraint rules can be used to represent constraint relationships. We attach constraint rules to the object classes. Within the scope of an object class, a constraint rule has direct access to the attributes which are defined for that class and indirect access to the attributes of those objects which are parts of the object the rule belongs to.

The way of defining constraint rules is: those representing constraint relationships between different attributes of the same object can be defined in the object class which the object belongs to; those representing constraint relationships between attributes of different objects can be defined in the object class which have access to these attributes and have access to least number of attributes among all such object classes. Thus constraint rules are defined in different object classes instead of the first-level object class which have access to all the attributes of the case object.

A constraint rule can be defined as follows:

Rule <rule name>
  If
    <pattern 1> [and <pattern 2> […[and <pattern *m*>]]]
  then
    <action 1> [and <action 2> […[and <action *n*>]]]
End [<rule name>]

where patterns contain attributes the rule has access to; actions maybe warning, etc.

In order to improve the speed of verifying the validity of an old case, a new problem or a combined case, we create a matching network of constraint rules for each object class based on Rete network [8]. The matching network is a rooted acyclic directed graph which consists of a root node, some one-input pattern nodes, and some two-input join nodes. First for each rule and each of its patterns we create a one-input pattern node which is connected to the root node and attached a matching state (true or false) identifying the pattern is satisfied or not. Then for each rule, if A_1, A_2, ..., A_n are the pattern nodes of the rule, we construct two-input join nodes B_2, B_3, ..., B_n where

B_2 has its left input from A_1 and its right input from A_2;

B_j has its left input from B_{j-1} and its right input from A_j, $2<j\leq n$.

A matching state is also attached to each join node identifying the patterns from the root node to the join node are satisfied or not. The matching state of B_n is the matching state of the rule. The collection of all the rules whose matching states are true is called conflict set.

Suppose there are two constraint rules R_1 and R_2 defined in an object class and four patterns P_1, P_2, P_3, and P_4 in the constraint rules. R_1 contains P_1, P_2 and P_3; R_2 contains P_1 and P_4. If P_1 and P_4 are satisfied, the matching network is shown in Fig. 4. Conflict set contains R_2.

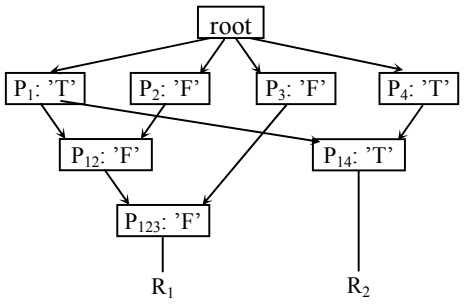

Fig. 4: A matching network of constraint rules.

The matching process based on a matching network is efficient. On the one hand, if a pattern is contained in several different rules, there is no need to examine the same pattern several times for different rules. On the other hand, if the values of some attributes change, there is no need to reexamine all the rules but to reexamine the patterns containing these attributes and propagate the changes to the leaf nodes.

4.2. Hierarchical combination

On the contrary to the top-down hierarchical decomposition of case retrieval, hierarchical combination is bottom-up. Suppose the matching results for a new problem P represented by an object O^* include similar sets G_1, G_{21}, G_{22}, and G_3. At first cases contained in G_{21} and G_{22} should be combined together to generate combined cases similar with O_2^* and stored in G_2. Then cases contained in G_1, G_2 and G_3 should be combined together to generate combined cases similar with O^* and stored in G. In order to verify the validity of combined cases, the matching network of constraint rules defined in the corresponding object class should be utilized. If the conflict set is empty, the combined case is valid and should be stored in the corresponding similar set.

Suppose O^*_x is an object of a new problem P, and $O^*_x = \{O^*_{x1}, O^*_{x2}, ..., O^*_{x\,m}\}$, where x is a subscript of zero to several digits. If G_x is empty and G_{xi} is not empty (i =1, 2, ..., m), the object $O'_x = \{O'_{x1}, O'_{x2}, ...,$

$O'_{xm}\}$, where $O'_{xi} \in G_{xi}$ (i =1, 2, ..., m), is called a combined case. If O'_x satisfies the constraint relationships, it is called a valid combined case which can be stored in G_x. If O'_x is most similar with O^*_x among all valid combined cases, it is called the best combined case.

All the cases in a similar set are ordered by the similarity. Suppose the number of cases in G_{xi} is marked as n_{xi}, each case in G_{xi} is marked as O^1_{xi}, O^2_{xi}, ..., or O^{nxi}_{xi} according to its order, and the similarity between O^j_{xi} and O^*_{xi} is marked as S^j_{xi} (j = 1, 2, ..., n_{xi}). In the process of case combination, each valid combined case is stored in G_x and is marked as O^1_x, O^2_x, ..., or O^{nx}_x according to its generation order, and the similarity between O^k_x and O^*_x is marked as S^k_x (k = 1, 2, ..., n_x).

4.3. Enumerative algorithm of case combination

The most simple and effective way to solve the problem of case combination is to try every possible combination to find all the valid combined cases and order them by the similarity to find the best combined case. Fig. 5 shows the enumerative algorithm which is a recursive process where O'_x is the temporary combined case created in the process of combination.

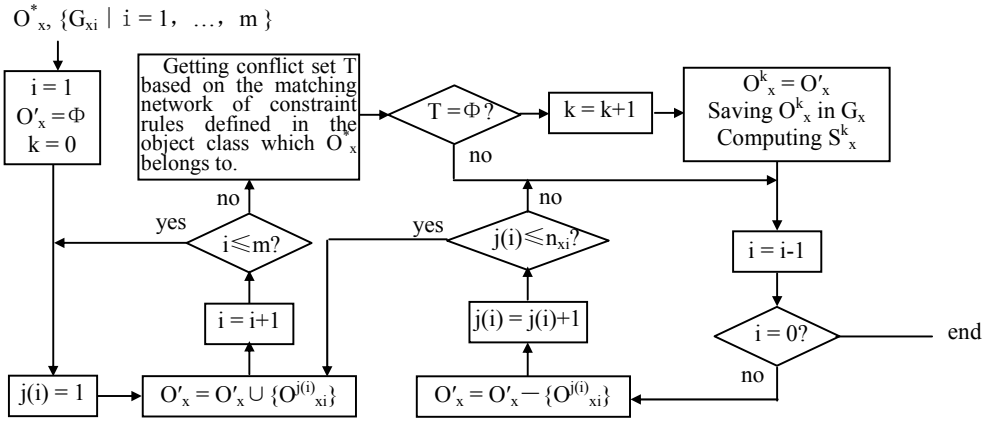

Fig. 5: Enumerative algorithm of case combination.

The advantage of the enumerative algorithm of case combination is that all the valid combined cases and the best combined case can be generated. But if there are many cases in the similar sets, the number of combination computed by Eq. (15) will be very large and the process will be low-efficient.

$$\prod C^1_{n_{xi}} = \prod n_{xi} \tag{15}$$

4.4. Genetic algorithm of case combination

The basic concept of genetic algorithms is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles of survival of the fittest such as reproduction, crossover, and mutation [9]-[10].

If we treat a combined case as a chromosome and treat parts of it as genes in the chromosome, the problem of case combination can be solved by genetic algorithm. The process is as follows.

(1) Defining the objective function and constraint relationships

The objective of case combination is to generate the best combined case, so the objective function is:

$$\max S_x = \sum_{i=1}^{m} W_{xi} \times S^{j(i)}_{xi} = \sum_{i=1}^{m} W_{xi} \times SIM(O^{j(i)}_{xi}, O^*_{xi})$$

s.t. $\bigcup_{i=1}^{m} \{O^{j(i)}_{xi}\}$ satisfies constraint relationships represented by constraint rules

$$j(i) = 1, 2, ..., n_{xi};\ i = 1, 2, ..., m \tag{16}$$

(2) Encoding

We encode the variables j(i) (i = 1, 2, ..., m) as two-digit strings in the range '01' to '99' representing the order in G_{xi}. The code for an individual (chromosome) is the concatenation of the codes for its different parts (genes). For example, the code for the combined case combining all the first cases in G_{xi} (i = 1, 2, ..., m) together is '0101…01'.

(3) Generating an initial population

Each individual in the initial population is generated randomly. The probability of generating any two-digit strings is equal to $1/n_{xi}$. The size of the population marked as pop_size can be defined as the maximum value of n_{xi} (i = 1, 2, ..., m). Each individual should be verified according to the matching network

of constraint rules defined in the object class which O^*_x belongs to. If the conflict set is empty, the corresponding combined case can be saved in G_x.

(4) Computing the fitness for each individual in the initial population

The fitness of each individual is computed by Eq. (16). The average fitness of the population is computed as Eq. (17).

$$\overline{S_x} = \sum_{h=1}^{pop\_size} S_x^h / pop\_size \qquad (17)$$

(5) Selection

The purpose of selection is to generate a new population with higher average fitness. Selection probability of an individual in the original population can be computed as Eq. (18).

$$P_h = S_x^h / \sum_{h=1}^{pop\_size} S_x^h \qquad h = 1, 2, ..., pop\_size \qquad (18)$$

(6) Crossover

Crossover is a process of exchanging the same parts of two selected individuals. We adopt two-point crossover method which selects two crossover points and exchanges the parts between the two points. The crossover rate marked as P_c can be assigned 0.8 which means about eighty percent of individuals should be crossed over to generate new individuals. All the new individuals should be verified.

(7) Mutation

Mutation is a process of randomly altering the individuals. The mutation rate marked as P_m can be assigned 0.1 which means about ten percent of parts in all individuals should be altered to generate new individuals. All the new individuals should be verified.

(8) Computing the fitness of each individual in the new population

The computing method is the same as (4). If the difference between the average fitness of the new population and the average fitness of the original population is less than a given threshold such as 0.005, the algorithm will be over. Otherwise the algorithm will return to (5).

Notice that the valid combined cases are saved in G_x instead of the final population. The advantage of the genetic algorithm of case combination is that many valid combined cases and the almost-best combined case can be generated efficiently. The shortcoming is that the best combined case and some valid combined cases may not be generated.

For most creative design activities, case adaptation is needed after the most similar old cases or combined cases are found. This step is usually executed by designers based on their knowledge and experience.

5. Application results

The method presented in this paper has already been applied to the domain of architectural design of information systems, and it works well.

Architectural design is an important stage in the process of information system (IS) development. It generates an enforceable technical scheme based on computer and communication systems according to the users' needs. Contemporary architectural design which is mostly based on distributed computer network environment involves various kinds of products and techniques.

We use unified modeling language (UML) to build an object-oriented model of architectural design of information systems, as shown in fig.6.

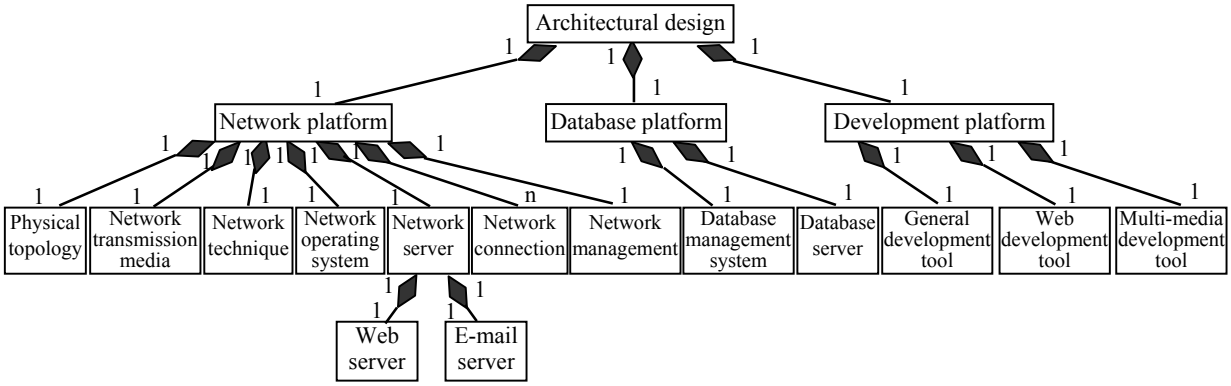

Fig. 6: Object-oriented model of architectural design of information systems.

A prototype system was developed in Chinese to implement above method in the domain. Initially more than 50 cases had been collected and stored in the system. They were clustered into six classes. We first analyzed users' needs of a new problem and inputted its descriptions into the system. The new problem was clustered to identify its class. Thirteen cases which were in the same class with the new problem were

matched. None of them was totally similar with the new problem. Therefore the new problem was decomposed to the second level which included network platform, database platform, and development platform. For each part, similar cases were found. After the process of case combination, two combined cases satisfying constraint rules were obtained. The solutions to these two combined cases were adapted to get the final solution to the new problem. It proved to be feasible and effective. The whole process was also efficient.

Apparently, the effectiveness and efficiency of this method would be more and more obvious with the development of the scale of the case base.

Acknowledgement

This work is partially supported by Key Discipline Foundation of Central University of Finance and Economics.

References

[1] C.H. Tong, *Knowledge-Based Circuit Design,* Ph.D. Thesis, Rutgers Technical Report LCSR-TR-108, Laboratory for Computer Science Research, Hill Center for the Mathematical Sciences Busch Campus, Rutgers University, 1988.

[2] J.L. Kolodner and L. M. Wills, Paying Attention to the Right Thing: Issues of Focus in Case-Based Creative Design. *AAAI-93, Case-Based Reasoning Workshop,* Washington D. C., pp. 19-25, 1993.

[3] P. Pu, *Special Issue: Case-Based Design Systems*, *Artificial Intelligence in Engineering, Design, and Manufacturing,* Cambridge University Press, 1990.

[4] M.L. Maher, M. B. Balachandran, and D. M. Zhang, *Case-Based Reasoning in Design*, Lawrence Erlbaum, 1997.

[5] B. S. Brigitte, L. Mario and H. Andre, Case-Based Reasoning: Survey and Future Directions. *Proceedings of 5th Biannual German Conference on Knowledge-Based Systems,* Würzburg, Germany, pp. 67-89, 1999.

[6] B.T. Juan José, A. G. C. Pedro and D. A. Belén, An Object-Oriented Framework for Building CBR Systems. *Proceedings of the 7th European Conference on Case-Based Reasoning*, pp. 32-46, 2004.

[7] http://www.cs.bham.ac.uk/~jlw/sem2a2/Web/Kohonen.htm.

[8] C.L. Forgy and Rete, A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence,* 19: 17-37, 1982.

[9] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT press, 1996.

[10] G. Alicia and M. Julie, Case-Base Injection Schemes to Case Adaptation Using Genetic Algorithms. *Proceedings of the 7th European Conference on Case-Based Reasoning*, pp. 198-210, 2004.