Association for Information Systems
AIS Electronic Library (AISeL)

PACIS 2004 Proceedings

Pacific Asia Conference on Information Systems (PACIS)

December 2004

A Case Adaptation Method Integrated with Genetic Algorithms for E-Commerce Product Configuration

Langtao Chen
Fudan University

Limin Lin
Fudan University

Hong Ling
Fudan University

A Case Adaptation Method Integrated with Genetic Algorithms for E-Commerce Product Configuration

Langtao Chen
School of Management,
Fudan University,
Shanghai 200433, China
langtaochen@hotmail.com

Limin Lin
School of Management,
Fudan University,
Shanghai 200433, China
lmlin@fudan.edu.cn

Hong Ling
School of Management,
Fudan University,
Shanghai 200433, China
hling@fudan.edu.cn

Abstract

When CBR is applied to E-Commerce product configuration, the case adaptation method plays a central role. For many years an automatic case adaptation method has always been an object of CBR study. Genetic algorithms can be employed in case adaptation process. This paper presented a new case adaptation method using genetic algorithms for E-Commerce product configuration. The new adaptation method has two features. First, it uses crossover and mutation mechanisms of genetic algorithms (GAs) to adapt similar case(s) retrieved. Second, it uses case-similarity for optimization while constraint knowledge is utilized to construct feasible solution space. To evaluate the presented method, a test was carried out on a real world personal computers configuration dataset, and promising results show the efficiency and effectiveness of the new method.

Keywords: Product Configuration, Case-Based Reasoning, Case Adaptation, Genetic Algorithms

1. Introduction

E-Commerce (electronic commerce) is playing a more and more important role for an organization's survival and growth. In an E-Commerce environment, manufactures must supply customized products with respect to the customers' requirements. As a result, product configuration paradigms are changed from mass production to mass customization (Sabin and Weigel 1998). The way in which case-based reasoning (CBR) solves a problem is much similar to the way in which a company uses similar, previously solved configurations and adapts them if necessary to suggest a solution. Therefore CBR has become an important technique for realizing product configuration on E-Commerce especially when there are only a few previous successfully-used configuration cases and traditional rule-based configuration methods are difficult to apply.

A major challenge to case-based product configuration is case adaptation which needs specific domain knowledge while CBR is much suitable to solve problems in poorly-understood domains. Many commercial CBR systems have been successfully used without performing adaptation or just passively leaving adaptation tasks to people, i.e. they are primarily case retrieval systems (Watson 1997).

As an excellent problem-solving strategy itself, CBR can be strengthened when combined with other problem solving paradigms such as rule-based systems (Lopez and Plaza 1993), genetic algorithms (Masher 1994), artificial neural networks (Thrift 1989), constraint satisfaction (Purvis and Pu 1995), and model-based reasoning (Karamouzis and Feyock 1992). Ramsey and Grefenstette (1993) presented a method to generate initial population of GAs based on past execution information stored in cases for the purpose of continuous learning in

a changing environment. Gómez de Silva Garza and Masher (1999) discussed an evolutionary case adaptation approach in which past solutions are adapted by evolving different combinations of their features in parallel and continuously, until a feasible combination is found. However, when evaluating candidate solutions generated in the evolutionary process, this case adaptation approach uses constraint satisfaction techniques to construct fitness function without the notion of finding the best solution which is more valuable for problem-solving applications.

The objective of this paper is to present a new case adaptation method integrated with genetic algorithms for E-Commerce product configuration. The rest of the paper is organized as follows: the next section introduces background information of our research. Section 3 presents the integrated case adaptation method in detail. In this section attention is especially directed to the definition of fitness function and a dynamic case representation is introduced to improve the performance of our method. Then experiments are carried out on real world personal computers (PCs) configuration dataset to evaluate the efficiency and effectiveness of the presented method. The last section provides conclusions of our work.

2. Background

2.1 Case Adaptation

At the highest level of generality, CBR can be typically described as a cyclical process comprising the four REs, i.e. REtrieve, REuse, REvise, and REtain (Watson 1997; Aamodt and Plaza 1994). There are two kinds of reuse: (1) copy solution directly to solve the new problem; (2) adapt solutions of similar cases according to the new problem and suggest a solution.

In case retrieval module unless at least one fully-matching case has been found, case adaptation must be executed to generate a solution to meet the new situation. CBR assumes that similar problems have similar solutions (Leake 1996), i.e. there exists a one-to-one mapping between the problem space and the solution space. It is noticed that most computing techniques especially decision support technologies also depend on this assumption about the real world (Watson 1997). Figure 1 gives an illustration of the process of case adaptation.

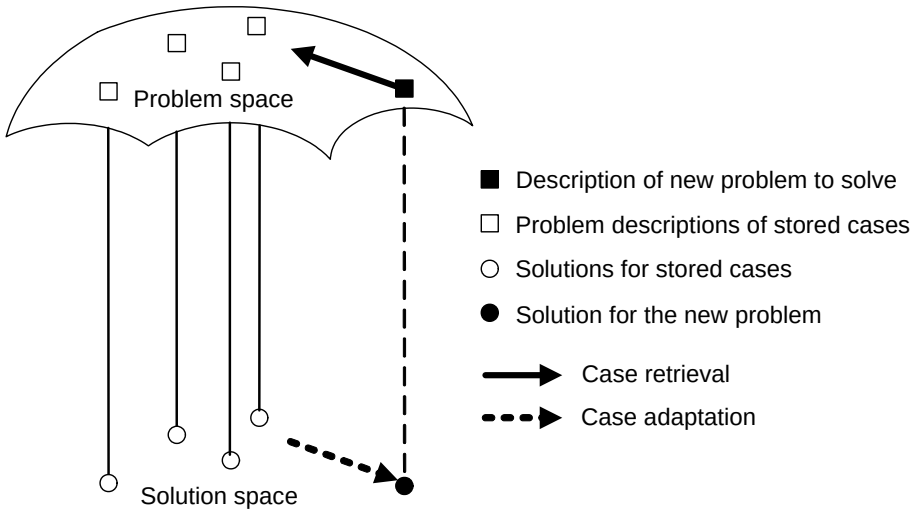

Figure 1. CBR adapts similar case(s) to fit new problem

Although many commercial CBR systems have been successfully used without case adaptation which has been a final challenge to CBR study, there are systems such as PLEXUS (Alterman 1986), CHEF (Hammond 1986), and COACH (Collins 1987) which especially focus on case adaptation (Watson 1997). Since adaptation is differently handled in different CBR systems, Hanney et al. (1998) introduce four dimensions to classify CBR systems with respect to adaptation:

- *Presence or Absence of Adaptation*: Systems clearly divide into those that do or do not use adaptation;
- *Single or Multiple Cases*: Solutions may be based on single or multiple cases. Single-case systems base their solutions on just one case (even if successive cases are tested before a solution is reached). Multiple-case systems usually have problem subparts that can be identified and separately modified to compose a solution;
- *Complexity of Case-solutions*: Solutions may be either atomic-valued solutions that are primitive indecomposable or compound solutions that have subparts that can be modified by adaptation;
- *Interactions within Case-solutions*: Compound solutions may have subparts that are independent or interacting (e.g., where the adaptation of one part of a solution requires further modification of other solution parts).

This paper deals with adaptation of both single and multiple cases whose solutions are compound while the subparts of solutions are interactive.

As far as case adaptation is concerned, there are three central questions, i.e. which parts of a solution to adapt, which changes are reasonable for adapting them, and how to control the adaptation process (Leake 1996). To answer these questions, CBR systems usually execute adaptation mainly through two ways: (1) reuse the solution stored in retrieved case(s) (transformational adaptation), and (2) reuse the rules or formulas that generate the solution (derivational adaptation). Traditional case adaptation is usually performed by rule-based systems, which lead to confront the knowledge elicitation bottleneck again. The difficulty of case adaptation lies in that effective adaptation generally needs the injection of specific domain knowledge while the philosophy of CBR is to solve problem with little domain knowledge available by reusing similar previous cases stored in case base as past experience of problem solving.

2.2 Case-Based Product Configuration on E-Commerce

The central task for product configuration is to choose appropriate components and assemble them into a product. Since traditional product configuration systems are knowledge-based, a major challenge is the elicitation of the rules or constraints on which the configuration process depends (Sabin and Weigel 1998).

Here we consider the configuration of Personal Computers (PCs) according to customers' requirements as an example, but the same principles apply to configurations of other highly complex products as well. A PC typically comprises such components as: motherboard, processor, hard disk, RAM, sound card, machine case, monitor, mouse, keyboard, floppy drive, and CD/DVD etc.. Among these components there exist compatibility issues. For instance, some types of motherboards only support AMD processors while others support Intel processors. In addition to the compatibilities, a more important issue is configuration optimization which mainly addresses the configuration of high performance products.

Case-based configuration is to reuse previous configuration knowledge stored in cases to solve the new configuration problem avoiding knowledge elicitation bottleneck, and it has become an important technique for product configuration application. System architecture of case-based product configuration on E-Commerce is shown in figure 2. The target of configuration system is to find a compatible combination of components which satisfies the requirements of the customer.

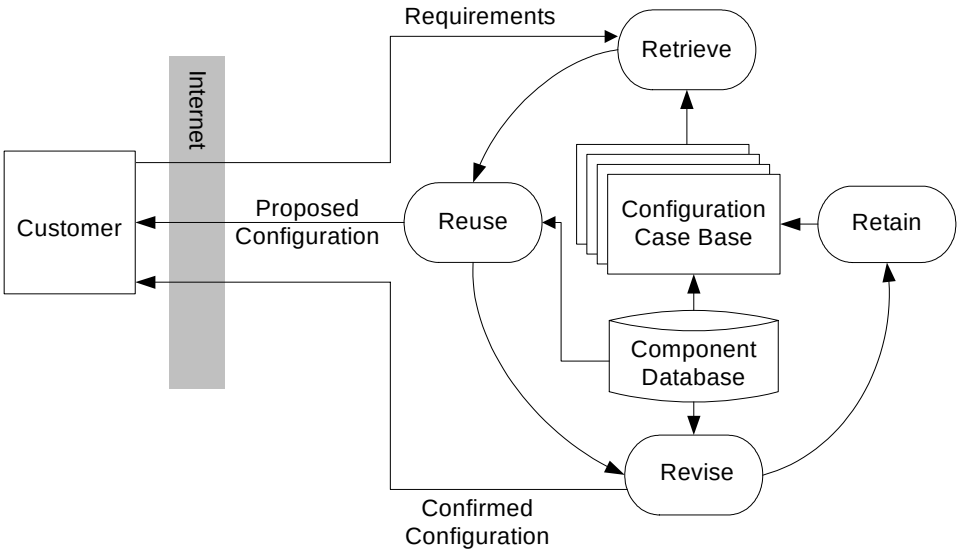

Figure 2. Case-based product configuration on E-Commerce

The cycle of the configuration system is outlined below.

- First customer requirements are input into the system.
- A set of similar configuration cases is then obtained from case base in retrieve module, using nearest neighbor (NN) as a simple retrieval algorithm.
- In the reuse phrase, the similar configuration cases are combined with respect to the customer's requirements and a solution is suggested through copy or adaptation. Traditional adaptation for configuration is performed by rule-based systems which lead to confront the knowledge elicitation bottleneck again.
- Through the revise process this solution is tested for success and repaired if failed. As a result, a confirmed configuration is given to customer.
- The revised configuration is then retained in configuration case base. This is a learning process through which useful experience is retained for future configuration reuse.

In consideration of the rapid update of components and the tremendous fluctuation of price, detailed information on components is stored in component database, while case base only stores information about general requirements and component identifiers corresponding to configuration cases, so that information inconsistency can be avoided.

2.3 Genetic Algorithms

Genetic algorithms were invented by John Holland in the 1960s. They are parallel, stochastic and adaptive search strategies based on natural selection and evolution. Genetic algorithms are especially useful when solving problems with large non-linear search space and poorly-understood domain knowledge which traditional optimization methods find difficult.

Given a clearly defined problem to be solved and a bit string representation for candidate solutions, a simple GA can be described as follows (Mitchell 1996):

Step1. Start with a randomly generated population P of n l−bit chromosomes (candidate solutions to a problem);
Step2. Calculate the fitness f(x) of each chromosome x in the population;
Step3. Repeat the following sub steps until a new population P' with n chromosomes has been generated:
a. Select a pair of parent chromosomes from population P. The higher the fitness is, the more likely a chromosome is selected;
b. With probability P_c (crossover probability), cross over the pair at a randomly chosen point (one point crossover) to form two offspring. If no crossover happens, just copy the pair to get two offspring respectively;
c. Mutate the two offspring at each locus with probability P_m (mutation probability), and place the resulting chromosomes in the new population P';
If n is odd, one offspring can be randomly rejected to keep population size unchanged;
Step4. Replace P with P';
Step5. If the condition of termination is satisfied, end the algorithm;
Else go to step2.

3. Method

When integrated with genetic algorithms, case adaptation method can not only synthesize subparts of old solutions to a new one, but also generate a fully novel solution through mechanisms of crossover and mutation. Case adaptation is performed incrementally through GAs' evolutionary process until a satisfactory solution is found. This section presents the case adaptation method in detail.

3.1 Algorithm Process

Figure 3 illustrates the way in which case adaptation guided with the aid of GAs is executed.

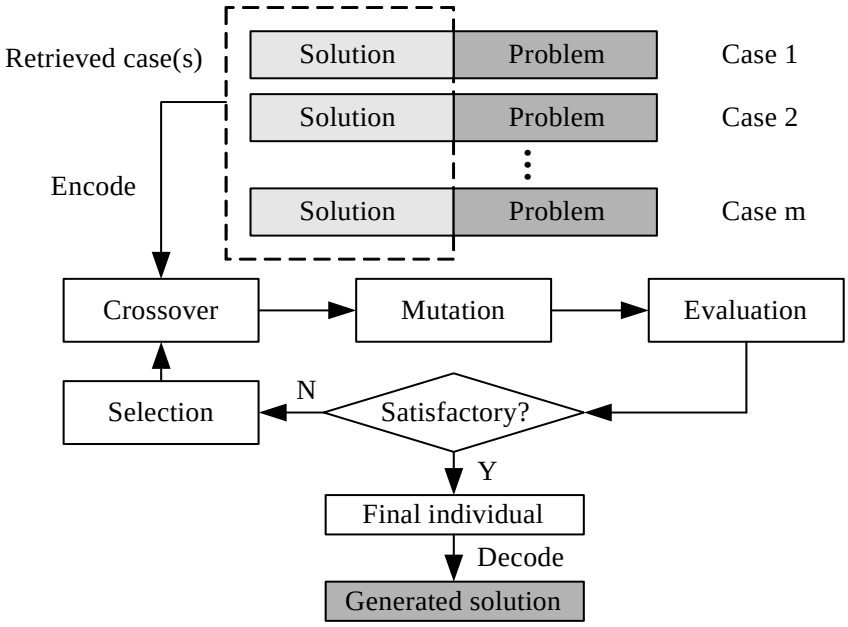

Figure 3. Case adaptation combined with GAs

After case retrieval, most similar cases from the case base are encoded to form the initial population of GAs. Then crossover and mutation are implemented generating new individuals which are genotype versions of candidate solutions to the given problem. To evaluate these individuals, fitness value is calculated for each one. If an individual's fitness satisfies the optimal condition, it is the final expected one which can be converted into the solution to the given problem by decoding. Otherwise these individuals are selected to form the new generation for the next evolutionary cycle. The higher the fitness is, the more likely an individual is selected to the next generation. This process runs incrementally until the algorithm converges to a satisfactory solution.

The role of GAs in the case adaptation process is to perform adaptation actions, i.e. crossover and mutation of case-solutions. Evolution processes through these two mechanisms, continuously generating new solutions which may be more suitable to the new situation than the old ones. Case adaptation is embedded in the evolutionary process which means the old solutions evolve to fit the new problem requirements. When convergence occurs in the GA cycles, the case adaptation is terminated with a solution being proposed to solve the new problem. The advantage of this kind of evolutionary adaptation lies in that it does not need adaptation rules or procedures which are difficult to elicit in real CBR applications.

3.2 Algorithm Pseudocode

The pseudocode for the case adaptation algorithm is shown as follows:

Algorithm GA_Adaptation(*P,CB, R_threshold,*P_c*,*P_m*,F_threshold,Maxgen*)

/**P*: the new problem, *CB*: case base, *R_threshold*: threshold of case retrieval */
/* P_c: probability of crossover, P_m: probability of mutation*/
/**F_threshold*: threshold of evolution termination*/
/* *Maxgen*: maximum generation of evolution*/
1. {C_1,C_2,…,C_m}=FindSimilarCase (*P*,*CB*); //*m* is the number of similar case(s)
2. for each case C_k in {C_1,C_2,…,C_m}
3. if *Sim*(C_k, *P*)>= *R_threshold* //C_k is very similar to the new problem
4. S_k = SolutionExtract (C_k); //copy solution of C_k to solve problem *P*
5. return (S_k); // it is unnecessary to perform case adaptation
6. {S_1,S_2,…,S_m}=SolutionExtract (C_1,C_2,…,C_m); //extract solution parts of cases
7. *pop*={S_1,S_2,…,S_m}; //initiate population
8. if *m*= =1
9. *m*=*m*+1;
10. S_m=S_{m-1}; //ensure the size of population not less than 2
11. *gen*=1;
12. repeat
13. *popsize*=*m*; //initiate the size of *pop*
14. for *i*=1 to *m*
15. {*parent1*,*parent2*}=ChooseParent (*pop*); //choose two parent chromosomes randomly from *pop*
16. {*offspring1*,*offspring2*}=Crossover (*parent1*,*parent2*,P_c); //with probability P_c, cross over the pair at a randomly chosen point (one point crossover) to form two offspring
17. if crossover happens
18. $S_{popsize+1}$= *offspring1*;
19. $S_{popsize+2}$= *offspring2*;
20. *popsize*=*popsize*+2;

21. for each individual S_j (j=1,2, …,*popsize*) in *pop*
22. S_j =Mutate (S_j,P_m); //mutate the individual at each locus with probability P_m
23. f_j=Fitness (S_j); //calculate the fitness value
24. S_l=MaxFitnessIndividual (*pop*); //find the individual S_l that has the maximum fitness
25. if f_l >= *F_threshold*
26. return(S_l); //get the satisfactory solution S_l to the problem *P* and terminate algorithm
27. for each individual S_j (j=1,2, …,*popsize*) in *pop*
28. SP_j =CalculateSelectionProbability (f_j); //selection probability SP_j is an increasing function of fitness f_j
29. SP=(SP_1, SP_2, …, $SP_{popsize}$);
30. for i=1 to m
31. S_i =Select (*pop*,*SP*); //select an individual from *pop* in which each individual S_j is selected with probability SP_j (j=1,2, …,*popsize*)
32. *pop*={S_1,S_2,…,S_m}; //get the new population *pop* for the next evolutionary process
33. *gen*=*gen*+1;
34. until *gen*>*Maxgen*

End GA_Adaptation

Note that when only one similar case is retrieved from case base, it can be copied to get another same case with the aim of keeping population size not less than 2. So this approach can be applied to the adaptation of both single and multiple cases.

3.3 Definition of Fitness Function

In the case adaptation method, since CBR usually is applied to domains with little knowledge available, a major difficulty of adaptation algorithm presented above lies in the definition of fitness function which is used to evaluate candidate solutions generated in evolutionary process of GAs. Gómez de Silva Garza and Masher (1999) presented a fitness function *F=C* which employs ideas based on constraint satisfaction techniques without the notion of finding the best solution. Effectively defining fitness function generally needs the availability of specific domain knowledge while CBR is often used to solve problems with little domain knowledge available. However, since the fundamental principle of CBR itself is to reuse previous experience which is contained in cases, this experience can also be employed to construct the fitness function.

This paper considers the evaluation of suggested solutions as an optimization problem. First, constraint satisfaction techniques are used to construct feasible region. Then the case-similarity mechanism can be applied to optimize the feasible region to get optimal or near-optimal solution. To express this idea, first we define the similarity function between the suggested solution T and the solution part of the k-th similar case retrieved from case base as the following equation:

$$Sim(T,S_k)=\frac{\sum_{j=1}^{n}(w_j \times localsim_j(T,S_k))}{\sum_{j=1}^{n} w_j} \quad (1)$$

where

T is the target solution which needs to be evaluated

S_k is the solution part of the k-th similar cases retrieved from case base

n is the number of solution attributes in each case

$localsim_j$ (T, S_k) is local similarity normalized on (0,1] for the j-th attribute in T and S_k

w_j is the importance weighting for the j-th attribute in T and S_k
$Sim(T,S_k)$ □(0,1] is the similarity between T and S_k

In the above equation, the nearest neighbor (NN) algorithm is considered as the criterion of similarity measuring. Then we get the weighted similarity between the target solution T and the solution parts of m similar cases retrieved, using $Sim(T,S_k)$ itself as the weighting of the similarity between T and S_k. Therefore the weighted similarity is computed as the following equation:

$$WS(T,S) = \frac{\sum_{i=1}^{m} (Sim(T,S_k))^2}{\sum_{i=1}^{m} Sim(T,S_k)} \quad (2)$$

where
S is the set of m similar retrieved cases (through S_1 to S_m)
$WS(T,S)$ □(0,1] is the weighted similarity between T and S

Given k constraints ($C_1(T)$, $C_2(T)$,..., $C_k(T)$) that must be satisfied for the target solution T, the total fitness function of T is constructed as the following equation:

$$Fitness(T) = \sum_{h=1}^{k} C_h(T) + WS(T,S) \quad (3)$$

If the h-th constraint is satisfied, C_h (T)=1, otherwise C_h (T)=0. The above fitness function can be simply expressed as $F=C+WS$. It is clear that $Fitness(T)$ is a value between 0 and k+1, i.e. $Fitness(T)$□(0,k+1]. If $Fitness(T)$□(k,k+1], which means that all constraints have been satisfied, T is a feasible solution. Furthermore, when $Fitness(T)$ is equal or very close to k+1, the target solution T can be considered as the optimal or sub-optimal solution to the given problem.

In real application, the constraints is varied for different given problems. For example, in a PCs configuration scenario, a customer may constrain her/his favorite PC as the following:

"My PC must comprise an Intel Pentium-4 2.0 GHz processor and one IBM hard disk with storage capacity not less than 60 GB; the total price should be between $800 and $1000."

In the evolutionary process of GAs, even a suggested solution with a fitness value less than k should not be eliminated from the GA cycle. The reason is that the suggested solution which has violated some constraints, i.e. it is not a feasible solution, may have optimal or near-optimal subparts which may yield to a whole optimal solution through crossover and mutation in the later GA cycles.

3.4 Dynamic Case Representation

A case is made up of two components, i.e. a problem description and a stored solution. It is assumed that there exits a one-to-one mapping between the problem space and the solution space (see section 2.1). However, since the difference between these two spaces exists, the similarity between the optimal solution and those ones of similar cases may not be the same with the similarity between the given problem and problem descriptions of similar retrieved cases. To reduce the difference between problem space and solution space, this paper introduces a dynamic case representation schema which can better support the definition of fitness function (see section 3.3).

In problem-solving domains such as PCs configuration, some local subparts of a solution is known or required, which reflects people's knowledge about how the solution should be. With the purpose of utilizing this kind of knowledge, we define problem description as a set of attributes which is known to the new problem-solving task while solution is a set of attributes not known. In this way, the original problem and known solution subparts of a case all comprise the new problem of the same case, and other subparts of solution recombine the new solution part. For example, a PC configuration case is a set of values of attributes which include PC type, performance, price, processor, motherboard, RAM, RAM number, hard disk, hard disk number, video card, monitor, sound card, and sound box etc.. General requirements are PC type, performance, and price. If the customer selects a processor component and a hard disk component (for instance, Intel Pentium-4 2.4GHz and Maxtor 120GB UDMA 133) as her/his preferred components, then these two known components comprise component requirements subpart. Other components belong to configuration subpart.

We suppose that all features in the original solution have something in common, which is true in many domains at least in the product configuration applications. The more attributes are known, which means more original solution subparts are transformed to the new problem part of the solution, the more similar the new problem space will be to the new solution space. As for what and how many attributes are known, it is dynamically determined by the specific problem-solving task.

The essence of dynamic case representation is to reduce the original solution space; therefore the adaptation difficulty is partly diminished in contrast with traditional static case representation. Figure 4 illustrates a hierarchical representation of configuration case employing the idea of dynamic case representation described above.

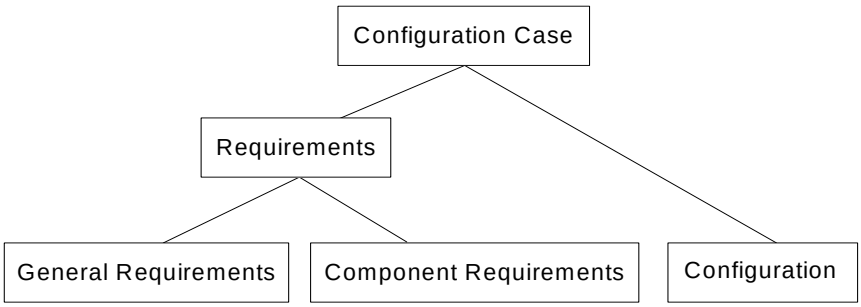

Figure 4. Case representation of product configuration

There are two different views on a configuration case: requirements (problem description) and configuration (solution). Requirements are divided into two subparts, i.e. general requirements and component requirements. General requirements are generic configuration descriptions such as product type, performance, and expected price etc. which are submitted by the customer. Component requirements are those components the customer selects. This set of components reflects the customer's preference for the product at component level. Other components excluding those customer selects comprise configuration subpart. Note that the component requirements subpart and configuration subpart are dynamically determined by specific configuration task. These two subparts have no intersection. The component requirements are treated as constraints; other constraints act on the whole configuration case such as compatibility of different product components and other rules elicited from domain experts.

3.5 Selection

There are several selection mechanisms available such as roulette wheel selection, elitist selection, and scaling. In the evolutionary process of GAs, as described in section 3.3, even an unsatisfactory solution which has violated the constraints should also have the opportunity to be selected into the next GA cycle to reserve excellent subparts that may be contains in the solution. In this paper we choose roulette wheel selection mechanism which is similar to spinning a roulette wheel where the slot for each individual is proportional to its fitness value in size. The higher the fitness is, the more likely an individual is selected into the next generation, which is the notion of survival of the fittest.

The cumulative selection probability P_j for each individual is calculated by the following expression:

$$P_j = \frac{\sum_{l=1}^{j} Fitness(l)}{\sum_{i=1}^{Popsize} Fitness(i)} \tag{4}$$

To begin with the selection, a random number p between 0 and 1 ($p \in (0,1]$) is generated. If $P_{j-1}<p<=P_j$ (j=1, 2,…, *popsize*; P_0=0), the j-th individual is selected.

4. Experiments

We have tested the case adaptation method presented in this paper on PCs configuration domain. The experiments aim to evaluate the proposed case adaptation method mainly in two aspects: time efficiency and accuracy of suggestion solutions. We selected 80 PCs configuration cases from *www.pconline.com.cn*. Each case has varied numbers of features. To simplify the problem, this experiment only considers 13 main features, i.e. PC type, performance, price, processor, motherboard, RAM, RAM number, hard disk, hard disk number, video card, monitor, sound card, and sound box.

In order to simulate the process of product configuration, the original dataset is divided randomly into two parts: a training set of 60 cases and a test set of 20 cases. The training set is stored in case base as configuration cases which serve as memory organization for case-based reasoning. The problem description of test case is input as customer general requirements and the corresponding solution subpart is used as evaluation benchmark for solution suggested by adaptation process. The constraints a feasible configuration must satisfy simply include two types in the experiments. The first type is compatibility between PC components; and the second one is component requirements of the customer.

4.1 Experiment 1 - Computational Time Efficiency

Evaluating the time performance of the suggested case adaptation method is not so important from the research point of view. However, it is important for real E-Commerce applications. In the experiment, the generations required before convergence are tested by comparing the proposed case adaptation method with a pure GA. The case adaptation method integrated with GAs uses similar cases to initiate the first generation of GAs while not only constraint satisfaction techniques but also case-similarity between the suggested solution and solutions of similar cases is utilized to construct the fitness function. For the pure GA, the first generation is initiated randomly and the fitness function definition only considers constraint satisfaction techniques.

The case base for the proposed case adaptation method contains all the 60 training cases, and the 20 test cases are used to evaluate the computational time efficiencies of the proposed algorithm and the pure GA method respectively. We do the experiment through totally 120 trials, with each test case 6 times. The experiment is given a limit of 200 generations of evolution in which to get a satisfactory solution, which means that if the algorithm does not converge till the 200 evolution generation, the trial will be redo until a convergence occurs. Table 1 shows the average generations before convergence for each algorithm.

Table 1. Average generations before convergence

Algorithm	Average Generations
Pure GA	21.57
Case Adaptation integrated with GA	12.32

It is obvious that case adaptation method combined with GAs has a higher convergence rate than pure GAs. The reason is that initial population injected by similar cases is served as the starting points for problem solving, while pure GAs' initial population is randomly selected, i.e. pure GAs solve a problem from scratch. In addition, the initial population injected by cases retrieved from case base may contains customer's tacit requirements which sometimes are key factors to enhance the customer's satisfaction.

4.2 Experiment 2 - Accuracy Competence

The second experiment comprises the variance of the case-base size to construct the case base, using cases from training dataset. We have used ten different case-base sizes: 15, 20, 25, 30, 35, 40, 45, 50, 55 and 60. The last one comprises all the possible training cases. At each case base size, 20 test cases are input to evaluate the accuracy competence of the proposed case adaptation algorithm compared with other methods.

The term accuracy is a criterion to measure how far the result obtained is close to the result expected. In the product configuration application, it is difficult to directly compute the accuracy of the adaptation methods since the adaptation results are product configuration instances which may not be characterized by a simple numeric value. In fact, the accuracy is a kind of similarity which is a key factor in case-based reasoning. Therefore, in this experiment, we use the nearest neighbor similarity between suggested solution according to problem part of test case and the original solution part of the same test case to measure the accuracy of the adaptation method.

Figure 5 compares average accuracy results of two different definitions of fitness function using static case representation with the case base size varying from 15 to 60. As expected we see that fitness function $F=C+WS$ is always more competent than $F=C$ which only considers constraint satisfaction without the notion of finding the optimal solution. As the case base size increases, both accuracy features rise and their difference is enhanced. However, the accuracy drops as case base size goes from 40 to 50. One reason might be the randomness of genetic algorithms. Another reason may lies in the organizing order of case base.

To compare accuracy results of dynamic case representation and static case representation which is the original case representation schema contrasted with the dynamic one, in this experiment we suppose that the customer has selected her/his favorite brand of processor. Moreover, we just consider $F=C+WS$ as the fitness function definition. From the results presented in figure 6, it can be inferred that the dynamic case representation has better performance than static case representation. The reason is that in dynamic case representation

the original solution space has been reduced to a new one so that the adaptation difficulty is partly diminished.

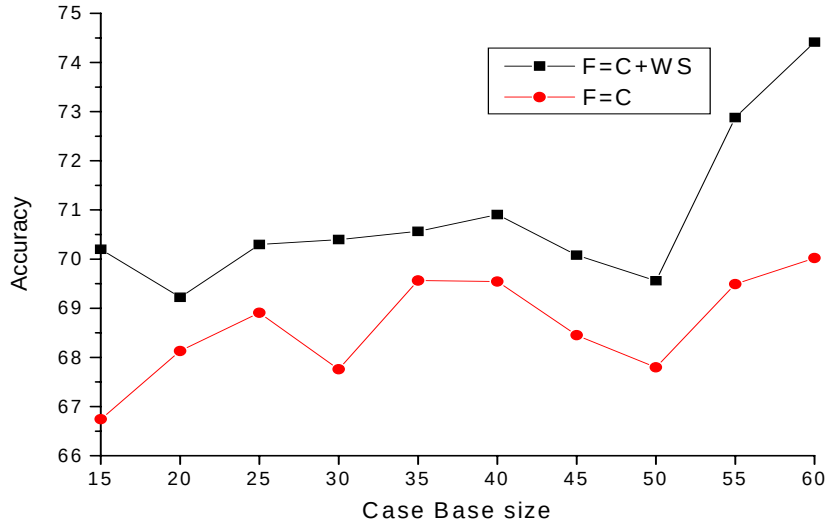

Figure 5. The accuracy of F=C+WS vs. the accuracy of F=C

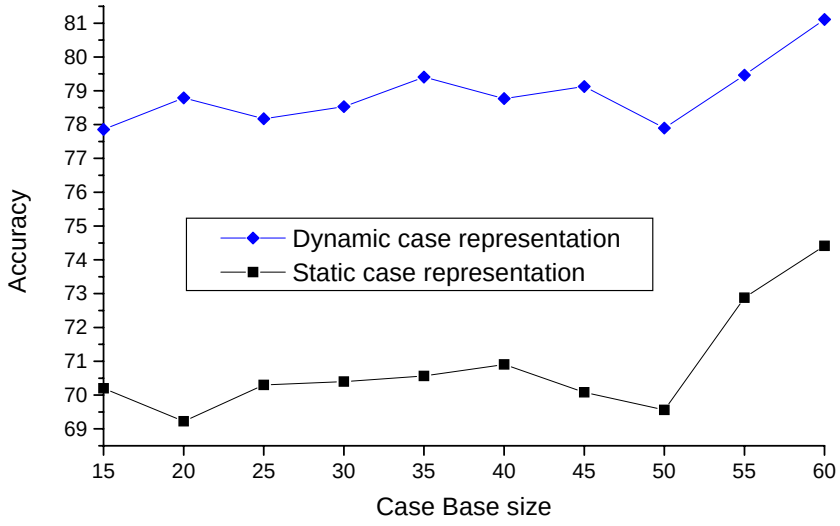

Figure 6. Accuracy results of dynamic case representation and static case representation

5. Conclusions

In this paper, we have presented a new case adaptation method combined with genetic algorithms for the application of E-Commerce product configuration. A significant advantage of this case adaptation method is that the entire adaptation process does not need specific domain knowledge apart from evaluation module, and since the knowledge for evaluation is relatively easy to acquire, this method could be very efficient for problem-solving tasks with little domain knowledge available.

The new algorithm recombines similar cases by using crossover mechanism, and through mutation mechanism existing cases could be extended to cover as much domain knowledge

as possible. Moreover, in the evolutionary process of GAs, this method employs case-similarity to find an optimal or sub-optimal solution with constraint knowledge being used to construct feasible solution space.

Experiments have been carried out on a real world E-Commerce PCs configuration dataset. The preliminary results showed that the proposed method is appropriate for product configuration applications with a high performance especially in E-Commerce environment. The presented case adaptation method can be generally used for product configuration applications both at on-line and off-line settings. Reasoning tasks other than product configuration may also benefit.

For further research, some more interesting work could be done on the dynamic case representation introduced in this paper. Dynamic case representation could reduce the difference between original problem space and solution space and then could better support the fitness function definition in this paper. But the resulting performance depends on how much the difference is, which is usually determined by how much knowledge people have about the solution to the new situation. Further work could be done on precisely evaluating the resulting solution when the difference between problem space and solution space is inevitably significant.

Acknowledgement

The research in this paper has been funded by the NSFC No. 70201009.

References

Aamodt, A., and Plaza, E. "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *AI Communications* (7:1), 1994, pp. 39-59.

Alterman, R. "An Adaptive Planner," *Proceedings of AAAI-86*, Cambridge, MA: AAAI Press /MIT Press, 1986.

Collins, G. "Plan Creation: Using Strategies as Blueprints," *Ph.D. thesis*, Department of Computer Science, Yale University, 1987.

Gómez de Silva Garza, A., and Masher, M.L. "An Evolutionary Approach to Case Adaptation," *Proceedings of the 3rd International Conference on CBR*, 1999.

Hammond, K.J. "CHEF: A Model of Case-Based Planning," *Proceedings of AAAI-86*, Cambridge, MA: AAAI Press /MIT Press, 1986.

Hanney, K., Keane, M.T., Smyth, B., and Cunningham, P. "When Do You Need Adaptation?: A Review of Current Practice," *AAAI-95 Fall Symposium on Adaptation in Knowledge Reuse*, Cambridge, MA, USA, 1995.

Holland, J. "Adaptation in Natural and Artificial Systems," The University of Michigan Press, Ann Arbour, 1975.

Karamouzis, T., and Feyock, S. "An Integration of Case-Based and Model-Based Reasoning and its Application to Physical System Faults," in *Lecture Notes in Computer Science 604*, Berlin: Springer-Verlag, 1992, pp. 100-108.

Kelbassa, H.-W. "Optimal Case-Based Refinement of Adaptation Rule Bases for Engineering Design," *Proceedings of the 53rd International Conference on CBR*, 2003.

Leake, D. "Case-Based Reasoning: Experience, Lessons, and Future Directions," AAAI Press, Menlo Park, California, 1996.

Lopez, B., and Plaza, E. "Case-Based Planning for Medical Diagnosis," *Methodologies for Intelligent Systems, 7th International Symposium, ISMIS-93*, Lecture Notes in Artificial Intelligence 689, Berlin: Springer-Verlag, 1993.

Louis, S.J., and Johnson, J. “Robustness of Case-Initialized Genetic Algorithms,” *Proceedings of FLAIRS (Florida Artificial Intelligence Conference)’99*, 1999, pp. 129-133.

Louis, S.J., and Rawlins, G.J.E. “Designer genetic algorithms: Genetic algorithms in structure design,” *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kauffmann Publishers, 1991, pp. 53-60.

Maher, M.L. “Creative Design Using a Genetic Algorithm,” *ASCE*, 1994, pp. 2014-2021

Mitchell, M. “An Introduction to Genetic Algorithms,” The MIT Press, Cambridge MA, 1996.

Purvis, L., and Pu, P. “Adaptation Using Constraint Satisfaction Techniques,” in *Case-Based Reasoning Research and Development*, edited by M. Veloso and A. Aamodt., Lecture Notes in Artificial Intelligence 1010. Berlin: Springer-Verlag, 1995.

Ramsey, C.L., and Grefenstette, J.J. “Case-Based Initialization of Genetic Algorithms,” *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, 1993, pp. 84-91.

Sabin, D., and Weigel, R. “Product Configuration Frameworks-A Survey,” *Intelligent Systems*, IEEE[see also IEEE Expert](13: 4), 1998, pp. 42-33.

Shin, K. and Han, I. “Case-Based Reasoning Supported by Genetic Algorithms for Corporate Bond Rating,” *Expert Systems with Applications* (16:2), 1999, pp. 85-95.

Thrift, P. “A Neural Network Model for Case-Based Reasoning,” *Proceedings of the DARPA Case-Based Reasoning Workshop*, edited by K.J. Hammond, San Francisco: Morgan Kaufman Publishers, 1989.

Watson, I. “Applying Case-Based Reasoning: Techniques for Enterprise Systems,” Morgan Kaufman, Inc., 1997.

Wiratunga, N., Craw, S., and Rowe, R. “Learning to Adapt for Case-Based Design,” *6th European Conference on Case-Based Reasoning*, Springer Verlag, 2002, pp. 421-435.