A New Learning Mechanism for Resolving Inconsistencies in Using Cooperative Co-evolution Model*

Yongrui Xu and Peng Liang**

State Key Lab of Software Engineering
School of Computer, Wuhan University, Wuhan, China
{xuyongrui,liangp}@whu.edu.cn

Abstract. Many aspects of Software Engineering problems lend themselves to a coevolutionary model of optimization because software systems are complex and rich in potential population that could be productively coevolved. Most of these aspects can be coevolved to work better together in a cooperative manner. Compared with the simple and common used predator-prey co-evolution model, cooperative co-evolution model has more challenges that need to be addressed. One of these challenges is how to resolve the inconsistencies between two populations in order to make them work together with no conflict. In this position paper, we propose a new learning mechanism based on *Baldwin effect*, and introduce the *learning* genetic operators to address the inconsistency issues. A toy example in the field of automated architectural synthesis is provided to describe the use of our proposed approach.

Keywords: Cooperative co-evolution, Baldwin effect, automated architectural synthesis.

1 Introduction

In many software engineering problems, one aspect of a problem is often related to other aspects [1]. In order to acquire better solutions for these problems, co-evolution mechanism is used to model these problems, and each aspect of the problems corresponds to an independent population. In co-evolutionary computation, there are mainly two different evolution models: one is *predator-prey* model, and the other is *cooperative* model [1]. The main difference between the two models is that each evolving population in predator-prey model evolves to acquire better solutions only for their own populations (e.g., test case population evolves in order to generate better test case only) and the relationship between different populations is competitive. On the contrary, in cooperative co-evolution model, all the populations evolve to acquire better solutions for the whole problem (e.g., in [2], one population represents developers' team staffing, and the other population is responsible for work package scheduling. The two populations co-evolve to achieve minimum completion time for projects).

* This work is partially sponsored by the NSFC under Grant No. 61170025.
** Corresponding author.

C. Le Goues and S. Yoo (Eds.): SSBSE 2014, LNCS 8636, pp. 215–221, 2014.

There are many existing work about predator-prey evolution model, especially in the area of testing, such as [3][4]. However, cooperative co-evolution model is not well explored in Search-Based Software Engineering (SBSE) until very recently [2].

Compared with predator-prey co-evolution model, cooperative co-evolution model has much more challenges when using it. One of these challenges is how to avoid the conflicts between populations that work together to generate the final solutions for the software engineering problems. Here, we take an example to illustrate this challenge briefly. Xu and Liang proposed a cooperative coevolution approach for automated architectural synthesis using patterns [5]. In their approach, there are two populations: one is responsibility population which is used for responsibility synthesis (i.e., how to assign different methods and attributes from requirement specifications to different classes in object-oriented architectural synthesis), and the other is pattern population which is used for architectural pattern synthesis (i.e., how to implement a given pattern in architecture level). When synthesizing the candidate architectural solutions with the individuals from the two cooperative populations, the conflicts may appear. For example, method *A* and method *B* belong to the same class in an individual of responsibility population, whilst these two methods belong to different layers in an individual of pattern population (we simply suppose that Layer pattern is used). As methods in the same class should not belong to different layers in Layer pattern, this inconsistency should be resolved before a candidate architectural solution is synthesized with these two individuals. In the above example, the inconsistency occurs when two populations interact cooperatively, and this kind of inconsistency is specific to cooperative co-evolution model. As a consequence, special attention should be paid to resolve inconsistencies in using cooperative co-evolution model.

Recently, the community of SBSE has realized the importance of using Artificial Intelligence (AI) techniques (e.g., machine learning) to solve software engineering problems [6]. In this paper we propose a new learning mechanism, which is based on the Baldwin effect [7] original from the biological evolution field, to address the inconsistency issue in the cooperative co-evolution computation. In our approach, we extend the steps in each generation of evolution procedure with a new kind of genetic operator called **learning operator**, and we define four specific types of learning operators. We further use a specific type of learning operator to resolve the inconsistency issue in the automated pattern-based architectural synthesis as a toy example to show the use and effectiveness of our proposed approach. The contributions of this work are: (1) introduce a new genetic operator for individual learning in each generation, which extends the traditional genetic operators (e.g., selection operator, crossover operator, and mutation operator). This new operator is generic in cooperative co-evolution computation, and AI techniques can be integrated in the search process with this operator; (2) propose a new learning mechanism based on Baldwin effect for cooperative co-evolution computation, which can be used to resolve the inconsistencies between different populations. To our knowledge, it is the first attempt to investigate the learning relationship between different populations in the field of SBSE.

2 Approach

In evolutionary developmental biology, a character or trait change occurs in an organism as a result of its interaction with its environment. In [7], Baldwin proposed a mechanism for specific selection of offspring for general learning ability. Selected offspring would tend to have an increased capacity for learning new skills rather than being confined to genetically coded and relatively fixed abilities. This is a theory of evolutionary process known as Baldwin effect.

In [7], Baldwin observed that there are three different sorts of modifications to organisms which should be distinguished. The first one is rooted in the physical agencies and influences in the environment, which is called "physic-genetic". In nature, physical agencies and influences in the environment include all chemical agents, temperature changes, and so on. This kind of agencies works upon the organism to produce modifications of its form and functions. As far as these forces change the organism peremptorily, they may be considered **accidental**. One of the examples in biology is genetic mutation, and in the field of evolution computation, we can map this kind of modifications into the mutation operators, which are defined to introduce relatively small changes to individual solutions. Second, some "neuro-genetic" modifications arise from the spontaneous activities of the organism itself when it is carrying out of its normal congenital functions [7]. In plants, in unicellular creatures, and in very young children, we can see these variations and adaptations in a remarkable way. The commonality of these changes is that all of them have the **selective** property of the nervous system. In the field of evolution computation, we can map this kind of modifications into the selection operators and crossover operators. In addition, there are a set of "psycho-genetic" modifications which come from the conscious agency of the organism itself [7]. For instance, gregarious influences, maternal instruction, the lessons of pleasure and pain, and experience in the life may change the organism. This kind of modifications has the **intelligent** property, and has great influence on organisms in nature. However, in the field of evolution computation, there is no kind of genetic operators corresponds to this kind of modifications for individuals, which is widespread in nature.

On one hand, for co-evolution computation, each population acts as an external environment for other populations and individuals in one population can learn the experience from individuals of other populations, consequently other populations play a "conscious agency" role [7] of individuals in each population, which further leads to appearance of "psycho-genetic" modifications for individuals. On the other hand, in cooperative co-evolution, close relationships exist between populations, which are the root cause of appeared inconsistencies. Hence we introduce a new genetic operator called learning operator for intelligent learning of individuals, and this new genetic operator can be used to address the inconsistency issue. Fig. 1 illustrates an improved cooperative coevolution procedure, which introduces the new learning mechanism based on Baldwin effect.

Due to space limitation, we omit the details from Step 1 to Step 5, and Step 9, which are widely used in existing SBSE research. In the improved cooperative coevolution procedure, we add Step 6 to Step 8 to implement the learning mechanism for

each population in cooperative co-evolution, and the inconsistencies between populations can be resolved in these steps. We introduce these steps (i.e., Step 6, Step 7, and Step 8) in detail in following sub-sections. A toy example in the field of automated architectural synthesis is provided in Section 3.

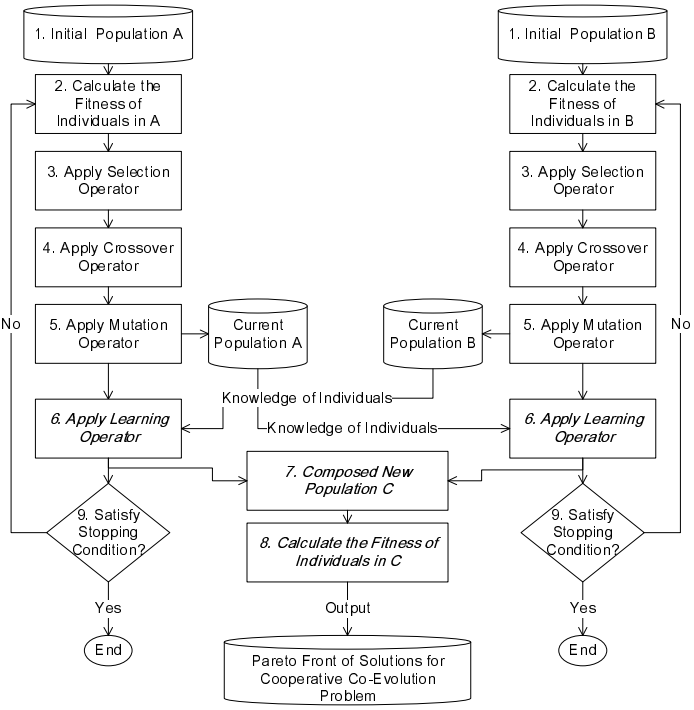

Fig. 1. The improved cooperative co-evolution procedure of our approach

2.1 Apply Learning Operator (Step 6)

In traditional population evolution, when Step 5 is completed, the next generation of each population is produced. But in our approach, the new produced generation of each population is regarded as a change impact for other populations, and each individual in any population should execute certain learning operations with a type of learning operator in Step 6. In this paper, we define four types of learning operators, which are similar to the types of traditional genetic operator (e.g., roulette wheel selection and tournament selection for the selection operator, and one-point and two-point crossover for the crossover operator): (1) *Lazy learning operator*. Similar to human beings that not all the people like learning, some individuals in a population are "lazy" in this step, which means they don't update their genes and remain the same representation. When using this learning operator, it follows the same procedure with traditional evolution for these individuals since no extra learning operation to be conducted; (2) *Localized learning operator*. In a society, people have different

interests (e.g., the readers of this paper may be interested in SBSE, but may not be interested in software architecture), and they only care about the knowledge of their interested fields. In this step, some individuals in a population may do localized learning with this operator, which means they update their genes according to the knowledge of only a few individuals in other populations that they are interested. We may use AI techniques (e.g., classification and clustering algorithms) in this learning operator to decide what the "interesting knowledge" that an individual cares about is (e.g., we can define a similarity function to choose the most similar individuals, which possess the "interesting knowledge", in other populations for an individual); (3) *One-to-one learning operator*. Similar to the human communication that some people prefer one-to-one communication, individuals in a population may update their genes according to the knowledge of an individual in other populations, which can also be decided by AI techniques; (4) *Global learning operator*. Some individuals in a population may update their genes according to the knowledge contained in all the individuals in other populations. Statistics or probabilistic methods might be used with this type of learning operator to acquire the collective knowledge of the whole population. In summary, these learning operators are used to adjust and update the genes of individuals according to the knowledge from individuals in other populations except *Lazy learning operator*. Inconsistencies between individuals in different populations can be resolved during this learning step (one exception is that when some "lazy" individuals are composed in the next step, they will make a mistake for their "laziness" because inconsistencies appear. In this situation, these "lazy" individuals only need to re-apply another type of learning operator to resolve the inconsistencies).

2.2 Composed New Population and Calculate the Fitness (Step 7 & 8)

In Step 7, individuals from two population respectively should be composed to a set of new individuals, and these new individuals establish a new population (e.g., for automated pattern-based architectural synthesis problem, it is needed to compose the individuals from responsibility population and pattern population respectively in order to acquire the final solutions. The details can be found in [5]). When this step is completed, we employ the defined fitness function to evaluate the individuals in the new population in Step 8. In our improved cooperative co-evolution procedure, we distinguish the fitness functions in Step 2 and Step 8 explicitly. The former only evaluates an independent aspect of the software engineering problem, but the latter evaluates the problem itself in a cooperative way. For example, in automated pattern-based architectural synthesis [5], both responsibility and pattern populations have an independent fitness function for evaluating their individuals respectively, in which one fitness function measures the quality of responsibility synthesis results (i.e., individuals in responsibility population) and the other measures the quality of pattern synthesis results (i.e., individuals in pattern population), while there is a third fitness function that measures the quality of pattern-based architectural solutions (i.e., individuals in composed population).

3 Example

In this section, we show how to resolve inconsistencies in cooperative co-evolution using our approach with a toy example (a more detailed example can be found in [5] due to space limitation). We use automated pattern-based architectural synthesis that mentioned in the Introduction Section as the application domain of the problem, but our approach is generic for nearly all the software engineering problems using cooperative co-evolution, which is not limited to this specific problem.

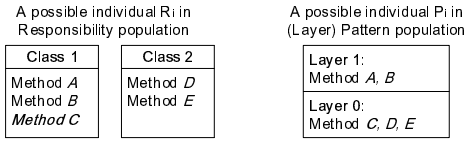

Fig. 2. An example of conflict in automated pattern-based architectural synthesis

As shown in Fig. 2, methods *A*, *B*, and *C* belong to the same class (Class 1) but are not allocated in the same layer (*A*, *B* are in Layer 1, and *C*, *D*, *E* are in Layer 0), and consequently the two individuals in responsibility and pattern populations are conflicting. We use the *One-to-one learning operator* to update the genes of the individual in responsibility population according to the knowledge of the individual in pattern population. For instance, in order to generate the next generation solution, R_i should learn the layer allocation information from P_i to update itself (or the other way round). After learning the genes of P_i, R_i knows that methods *A* and *B* should not be allocated in a class which contains any of methods *C*, *D*, and *E*. It is then found that the three methods in Class 1 (A, B, and C) of R_i have conflicts. As method *C* conflicts with both *A* and *B*, R_i decides to move *C* to another class (e.g., Class 2). The result of applying this learning operator to the individuals in responsibility and pattern population is shown in Fig. 3. The conflict is resolved between the two individuals after the learning operation, and we can compose them to an individual for the design problem.

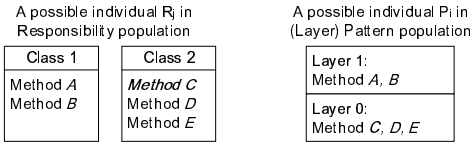

Fig. 3. The result of resolving an inconsistency between individuals using learning operator

4 Conclusions

In this paper, we propose a new learning mechanism for resolving inconsistencies in using cooperative co-evolution model, which is based on Baldwin effect. We extend the traditional genetic operators with the concept of learning operator which includes four types of learning operators. We describe the use of our approach using a toy

example in automated pattern-based architecture synthesis. The approach can be applied to various software engineering problems that use cooperative co-evolution.

References

1. Harman, M., Mansouri, S.A., Zhang, Y.: Search-Based software engineering: Trends, techniques and applications. ACM Comput. Surv. 45(1), 1–61 (2012)
2. Ren, J., Harman, M., Di Penta, M.: Cooperative co-evolutionary optimization of software project staff assignments and job scheduling. In: Cohen, M.B., Ó Cinnéide, M. (eds.) SSBSE 2011. LNCS, vol. 6956, pp. 127–141. Springer, Heidelberg (2011)
3. Adamopoulos, K., Harman, M., Hierons, R.M.: How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In: Deb, K., Tari, Z. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 1338–1349. Springer, Heidelberg (2004)
4. Arcuri, A.: On the automation of fixing software bugs. In: ICSE, pp. 1003–1006 (2008)
5. Xu, Y., Liang, P.: Co-evolving pattern synthesis and class responsibility assignment in architectural synthesis. In: ECSA (2014)
6. Harman, M.: The role of artificial intelligence in software engineering. In: RAISE, pp. 1–6 (2012)
7. Baldwin, J.M.: A new factor in evolution. The American Naturalist 30(354), 441–451 (1896)