AN APPROACH OF EFFECTIVE e-LEARNING ORGANIZATION

S. Sargsyan

A. Hovakimyan

N. Karapetyan

S. Barkhudaryan

Yerevan State University (YSU), Armenia
alglan@ysu.am

Abstract

An approach for the problem of building such a component of e-learning system that gives the user a chance to get the desired knowledge of teaching course in a user adaptable manner is suggested in this article. This approach is based on so-called "teaching scenarios" (sequence of teaching units) being constructed during the process of learning. We will introduce the GeneticChooser Algorithm (GCA) where Genetic Algorithm (GA) is used in the process of creating user adaptable teaching scenarios. Via the quality and quantity characteristics of the teaching units and user's knowledge, GA creates the appropriate sequence of teaching units from the course. The considered method is realized and introduced in the TeachArm e-learning system developed at the Department of Algorithmic Languages of YSU [1, 2].

Key Words

Genetic Algorithm, Fitness-function, teaching course, teaching scenarios, dynamic scenario, effective learning.

INTRODUCTION

We have a teaching course and are going to construct such a system that would organize the whole material in such a way that will make the process of learning easy and effective for the user.

The teaching process in learning systems is presented as separate teaching cycles, and each ends with testing the user's knowledge. Several types of teaching scenarios are distinguished: free, fixed and manageable via the teaching system. The third one is generated by the teaching system during the teaching process. The teaching material is divided into logical units that a teaching scenario is constructed of. Having the users' base of knowledge after a test, user is offered a scenario to study. A scenario is being constructed of teaching units sequenced in a way that provides a learner an effective way to get new knowledge.

An approach is suggested to solve this problem. It is worth mentioning that a solution will be an acceptable one rather than an exact one because Genetic Algorithm is an approximation method. In particular an exact solution for this kind of problems is almost impossible to find because they have a very large set of possible solutions where we will have to search the exact solution from. Besides no two learners are the same. They are of different backgrounds and different knowledge-base, which makes difficult to define exactly what does a good solution mean for a particular learner.

To solve a problem via GA means to represent the parameters of the real problem into parameters of GA, solve the newly created problem by running through the basic steps of GA, evolve the output and represent back the final output into the form acceptable for the initial problem, and finally declare the solution. What does parameter representation mean here? We know that GA operates on chromosomes. Thus a possible solution for the initial problem is a chromosome for the GA. The optimization function of the initial problem can be used as the fitness function of the GA. There is, however, a nicely serious question, what kind of representation is suitable for a chromosome. This and other questions that refer to internal concepts of GA are detailed and explained in [3,4].

So we find a user adaptable scenario of a teaching process phase using GA.

THE PROBLEM AND THE SOLUTION MODEL

The problem is the following:
Organize an effective teaching system having the teaching material and information of learner's knowledge base. We will define such concepts as teaching scenario, dependency matrix, teaching stage for a learner, acceptable scenario. Saying effective teaching we understand giving the next best teaching scenario to the learner in the given teaching stage taking into account the level of knowledge that the learner has gained studying the past teaching scenario.

Let's define the mathematical model for this problem.

Let's mark the teaching course as D. The course D is divided into *teaching units* that are numbered as $d_1, d_2, ..., d_n$, where $d_i \in N$. Each teaching unit contains a particular set of *key concepts* that a learner should master. Each such set we will mark as U_i, *where* $1 \le i \le n$. For each teaching unit d_i, *where* $1 \le i \le n$ we define the value $C_i \in N$ and is limited by $\theta \in N$, which describes its *complexity*. If the teaching unit d_i requires more time and efforts to be mastered than the teaching unit d_j then $C_i > C_j$.

Let's take $U = U_1 \cup U_2 \cup ... \cup U_n$, that is U is the whole set of key concepts in the course D. Now let's define dependency matrix $K = \left\{K_{i,j}\right\}$ of teaching units, where $1 \le i, j \le n$. If d_j depends on d_i then $K_{ij} = 1$, else $K_{ij} = 0$.

A *teaching scenario* is a sequence of *teaching units* $X = X_1, X_2, ..., X_m$, where $m < n$, $X_i \in D$ and $(\forall i \neq j) \Rightarrow X_i \neq X_j$.

The scenario X is *acceptable*, if every unit appears in it only once and a unit appears in the scenario sooner than those that depend on it.

Formally, the scenario X is *acceptable* if $(\forall i \neq j) \Rightarrow X_i \neq X_j$, and if for any pair (X_p, X_q), where $X_p, X_q \in X$, if $p < q$, then X_p teaching unit does not depend on X_q teaching unit (as matrix K states), $1 \le p, q \le m$.

Let's define the *complexity* of the scenario X as $\sum_{X_i \in X} C_i(X_i)$, where $C_i(X_i)$ is the complexity of the teaching unit X_i.

For each (X_i, X_{i+1}) pair of the scenario X let's define the concept of *distance* – $r_i(X_i, X_{i+1}), 1 \le i \le m-1$. $r_i(X_i, X_{i+1})$ is the number of those teaching units that the teaching unit X_{i+1} depends on after the learner has mastered the teaching unit X_i.

$$\text{Let's define } F(X) = \sum_{i=1}^{m-1} C_i(X_i) \bullet r_i(X_i, X_{i+1}) + C_m(X_m).$$

We will say that $X = (X_1, ..., X_m)$ acceptable scenario is better than $Y = (Y_1, ..., Y_m)$ acceptable scenario if $F(X) < F(Y)$.

$X = (X_1, ..., X_m)$ acceptable scenario is *the best* if<<Eqn039.eps>>. There may be several best scenarios; in that case we will just take one of them.

The sequence of knowledge of a learner we will denote as $Z = Z_1, Z_2, ..., Z_n$, where Z_i is a sequence itself and reflects which key concepts are learnt and which of them not in the teaching unit d_i. It can be either mastered $(Z_{ip} = 1)$ or not $(Z_{ip} = 0)$, Z_{ip} refers to the pth key concept of ith teaching unit.

Let's define the concept of *teaching phase* of a learner. It is defined in two steps:

- The best X scenario is being given to the learner. Testing is held after the learner has studied the units in the scenario X.
- Taking into account the results of testing Z vector and the matrix K are being updated to reflect the knowledge of the learner and the dependency between teaching units respectively.

Note here that each time when a learner passes a teaching phase the units, which have been included in the provided scenario and have been completely learnt by the learner (the corresponding Z_i sequence consists of ones only), are being removed from the whole teaching material thus removing the corresponding dependencies from K.

So, having defined the data domain of our problem let's state it formally.

Build such a scenario X for each teaching phase, that $F(X) \to \min$.

Now let's depict the parameters of our problem into parameters that are meaningful in the GA.

Teaching units will be the *genes* of the GA.

Teaching scenarios will be the *chromosomes* of the GA.

The set of teaching scenarios will be the *population* of the GA.

If a teaching scenario is allowable then the corresponding chromosome is *allowable* as well.

If the teaching scenario X is better than the scenario Y then the chromosome X is better than the chromosome Y in the GA.

There are many ways to solve this problem, but we have chosen the way that is provided by Genetic Algorithm. So, you may ask why GA? If GA can be used here then a regular algorithm may exist and possibly be better. Here we can answer, that the GA is used to optimize the problem. It is not used to find an exact solution. The problem itself is an optimization problem and it is hard to find a regular method for it. The GA guarantees that it will eventually find a good solution, because the GA has a "jumping" nature [4]. Mutating a chromosome or crossing over two chromosomes can yield much better chromosomes. Of course these operations can yield bad chromosomes as well. But the point to be considered here is that good chromosomes *are* constructed during the execution of the GA. And if we modify the GA so that it saved the best chromosomes after each cycle of its execution, we can be sure that in the end the GA will output the best result that it has found during its run.

THE GA PROBLEM AND ITS PARAMETERS

A gene is represented as a whole number, namely it is the number of the teaching unit. Whole numbers are preferable here because it makes the logic of the GA easier [3, 4, 5].

Now let's choose the operations of the GA [4, pp.476-477]. They are:

- Order crossover
- Double mutation

Note that simple crossover is not applicable here.

Let's define the main concept of the GA, namely the fitness function for our problem. We will take $F(X)$ as fitness function for our problem. GA states that if X chromosome is better than Y chromosome if $F(X) < F(Y)$.

Here we can say that the chromosome X is *the best* if $F(X) \to \min$.

THE GENETICCHOOSER ALGORITHM (GCA) DESCRIPTION

Well, we have got a teaching material. Author provides such information as the dependencies between the key concepts and the complexities of the key concepts. This kind of information will let us compute all the parameters of our problem.

For the beginning we assume that a learner knows nothing about the teaching material, and all the teaching units are still unstudied. Or the learner can take a test to find out his/her knowledge. If the learner knows nothing his/her vector Z is a zero vector. If the learner has taken a test and proved to know something, the corresponding elements of the vector Z will be set to ones.

Now let's describe the current step of GCA algorithm:

- Update the vector Z setting the members that correspond to learnt units to ones;
- Remove all the teaching units that are mastered. In this case some dependencies between the key concepts of the whole teaching material will disappear. If there is a key concept in a teaching unit which is not mastered that unit is not removed. Instead, its complexity value is decreased. The new complexity value is calculated by the following formula:

$$C_i'(X_i) = C_i(X_i) \bullet \frac{|U_i| - M}{|U_i|} + a \text{, where}$$

  - $C_i'(X_i)$ is the new complexity value, and $C_i'(X_i) < \theta$,
  - $C_i(X_i)$ is the old complexity value,
  - $|U_i|$ is the total number of key concepts in the teaching unit,
  - M is the number of key concepts in the teaching unit that are mastered,
  - a is a constant corrector value, $a < \min_i C_i(X_i)$. Its meaning is the following. If the learner has passed this teaching unit and the testing discovered that no key concepts are mastered, $M = 0$, it means that this teaching unit may be more difficult for this particular learner, and it is

    worth to have a higher complexity value.
    - Having the updated vector Z, the new complexity values, the possibly decreased set of teaching units, run the GA for these input values.
- After the GA has completed its job and returned the best scenario, forward that scenario to the learner.
- Repeat all the steps mentioned above while the set of the teaching units is not empty.

Now let's detail what GCA does.

- Construct the matrix K taking into account the vector Z after testing,
- Construct the initial population of chromosomes,
- While the end-condition is not true do the following:
    - Select some chromosomes (usually 1% of the entire population) using the roulette wheel method [3,4],
    - Order crossover the big part of the selected chromosomes,
    - Double mutate the rest of the chromosomes,
    - Check the offsprings against acceptability,
    - Compute the fitness values for all acceptable chromosomes,
    - Add theses chromosomes to the population,
    - Save the best ten (or other number of) chromosomes in a separate place.
    - Throw away the worst chromosomes so that the entire number of chromosomes in the whole population remained the same.

That's all for the GCA's steps. Of course, some aspects can be optimized. For example, a bad chromosome can contain a good "sub-chromosome" that is worth keeping around. For this purpose a "bucket brigade" algorithm is suggested to use [4].

The "end-condition" can be

- Completing some fixed number of cycles, or
- Testing if the difference between the average fitness of the previous population (the population before selection, order crossover, double mutation, and offspring addition) and the average fitness of the new population is small enough, or
- Testing whether the number of good chromosomes is big enough, etc.

Pay attention to one important aspect. Every time when testing is made the set of the teaching units are decreased. So we will eventually reach a situation when GA is neither good nor applicable because the set of teaching units is very small. In this case the "simple choosing" algorithm can be quite good (the fitness function is used here anyway to compare the chromosomes).

Having a set of chromosomes the GCA system gives another set of chromosomes where we can choose the chromosome which has the highest fitness value. The flow chart of the realized algorithm is illustrated in **Figure 1**. The GCA steps that have been modified to fit the needs of our system are marked with an "*".

The modifications are made to provide the system with acceptable chromosomes as we have defined above. We have also used the bucket brigade algorithm [4] to save those chromosomes that have good parts.

The GCA has been realized in the Java programming language. Experiments gave encouraging results. The entire GCA has been tested on hypothetical course with randomly generated dependencies between the key concepts. The

Figure 2 illustrates the effectiveness of GCA over the effectiveness of the simple choosing algorithm. The test showed that the larger is the problem domain the better the GCA behaves comparing to the simple choosing algorithm.

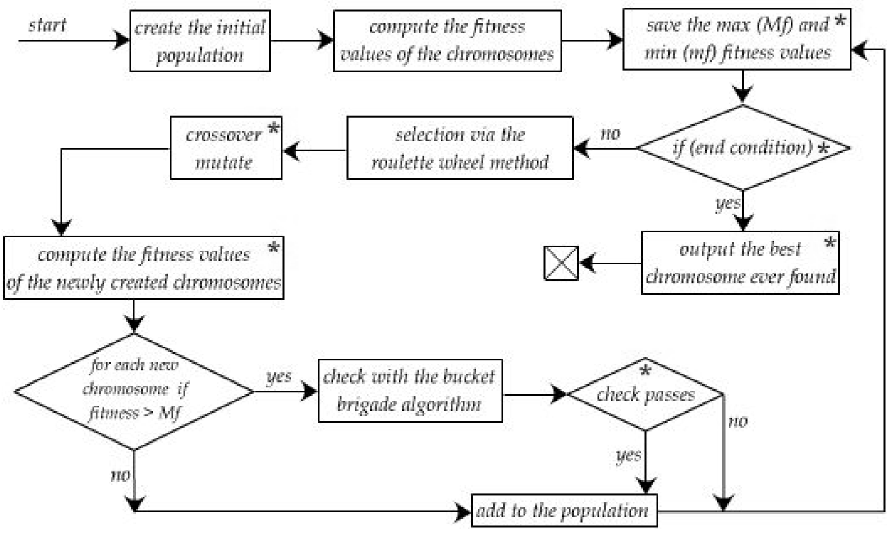
start
create the initial population
compute the fitness values of the chromosomes
save the max (Mf) and min (mf) fitness values *
if (end condition) *
no
selection via the roulette wheel method
crossover mutate *
yes
output the best chromosome ever found *
compute the fitness values of the newly created chromosomes *
for each new chromosome if fitness > Mf
yes
check with the bucket brigade algorithm
check passes *
yes
no
no
add to the population

Figure 1.

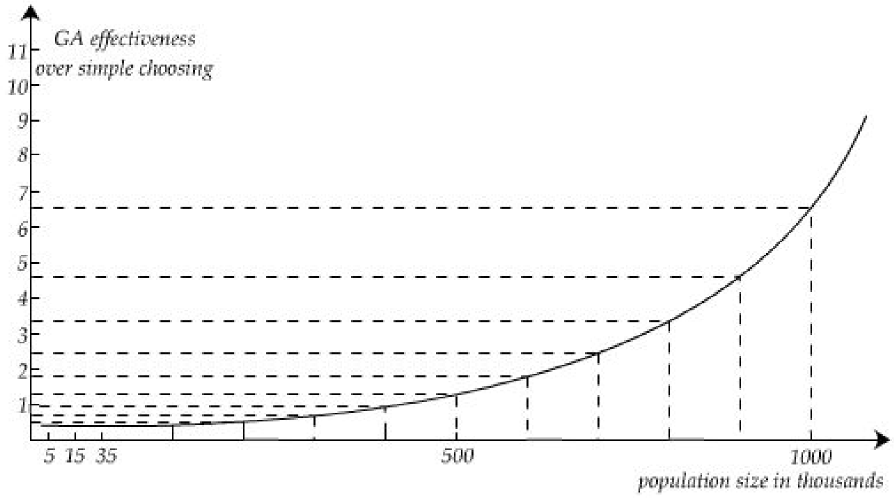

Figure 2.

The effectiveness of GCA also depends on the size of the population that the system operates on. The tests have shown that the larger is the population the better is the final fitness values. We can also see that the fitness value shrinks faster and faster but from some point it starts to shrink slower and slower. And there is some point where it stops changing. This also means that the best chromosome in of the final population is very close to the best chromosome entirely. See **Figure 3**.

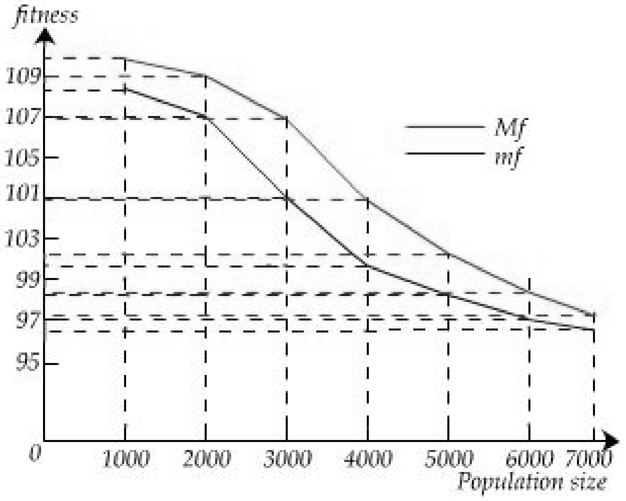

Figure 3.

CONCLUSIONS

In the presented article we discussed a problem dealing with the issues of the optimal management of the teaching process in e-learning systems. This problem's aim is to construct such user adaptable teaching scenarios, based on the given teaching resources, which will allow the student to get knowledge effectively. We developed the GeneticChooser Algorithm (GCA) using the Genetic Algorithms' concepts to solve this problem. We continue our researches in using GA-s in problems of developing e-learning systems and managing teaching process in them. The results we plan to present in our further papers.

REFERENCES

S.G. Sargsyan, A.S. Hovakimyan, K.S. Darbinyan, N. Ispiryan, E. Petrosyan, TeachArm Toolset for e-learning support, Proc. of International Workshop. Information Technologies in education, Yerevan, Armenia, 2005.

A.S. Hovakimyan, S.G. Sargsyan, About Building Teaching Systems for E-Learning, Proc. of International Conf. on Advanced Learning Technologies, Kazan, Russia, 2002.

Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs (Springer, 1999).

George F. Ludger, Artificial Intelligence. Structures and Strategies for Complex Problem Solving. Addison Wesley, 2003.

A.S. Hovakimyan, S.G. Sargsyan, The Genetic Algorithms (GA) in Web-based Learning Systems. Proc. of IASTED International Conference on ACIT-Software Engineering (ACIT-SE 2005), Novosibirsk, Russia, 2005.