Journal of Software Engineering and Applications, 2012, 5, 574-582
http://dx.doi.org/10.4236/jsea.2012.58066 Published Online August 2012 (http://www.SciRP.org/journal/jsea)

Solving the Balance Problem of On-Line Role-Playing Games Using Evolutionary Algorithms

Haoyang Chen*, Yasukuni Mori, Ikuo Matsuba

Advanced Integration Science, Graduate School of Chiba University, Chiba, Japan.
Email: *chenhaoyang@graduate.chiba-u.jp, yasukuni@faculty.chiba-u.jp, matsuba@faculty.chiba-u.jp

Received May 6th, 2012; revised June 14th, 2012; accepted June 22nd, 2012

ABSTRACT

In on-line role-playing games (RPG), each race holds some attributes and skills. Each skill contains several abilities such as physical damage, hit rate, etc. Parts of the attributes and all the abilities are a function of the character's level, which are called Ability-Increasing Functions (AIFs). A well-balanced on-line RPG is characterized by having a set of well-balanced AIFs. In this paper, we propose an evolutionary design method, including integration with an improved Probabilistic Incremental Program Evolution (PIPE) and a Cooperative Coevolutionary Algorithm (CCEA), for on-line RPGs to maintain the game balance. Moreover, we construct a simplest turn-based game model and perform a series of experiments based on it. The results indicate that the proposed method is able to obtain a set of well-balanced AIFs efficiently. They also show that in this case the CCEA outperforms the simple genetic algorithm, and that the capability of PIPE has been significantly improved through the improvement work.

Keywords: Game Design; Game Balance; Cooperative Coevolutionary Algorithm; Probabilistic Incremental Program Evolution

1. Introduction

In recent years, on-line Role-playing games (RPGs) become more and more popular all over the world. An on-line RPG is a multi-player virtual world which consists of several distinct races. Unlike most of board games (such as Chess, Shogi, Go, etc.), which are symmetrical systems, while on-line RPGs are regarded as asymmetrical ones since each race has some unique capabilities. For example: wizard can perform spells, while knight specializes in close fighting. Before starting the game, players should first create one character belonging to a certain race, then play the game by both exploring the virtual world and fighting with others.

To be enjoyable, the game must be balanced well [1]. In other words, for the individual player, it must be neither too easy nor too hard, and for players competing against each other, fairness is crucial. A game without balance is untidy and ugly, flaws are reflected in the experience of playing it. An unbalanced game is less satisfying. More seriously, from the designer's viewpoint, not only time but also effort has been wasted. If parts of the game are not well-balanced, there will be a risk that some of them will be used rarely, if at all, in which case the time spent developing them has gone to waste.

*Corresponding author.

The concept of balance can be divided into two parts, balance of Player-versus-Environment (PvE) and balance of Player-versus-Player (PvP) [2]. In this paper, specifically, the term PvP refers to "1 versus 1" because it is the core of on-line RPG's combat systems. Generally, balance of PvE mainly means the difficulty control of the game, while balance of PvP, existing only in asymmetrical games, refers to the power balancing between races. When designing an on-line RPG, the power of a race will be represented by some attributes and a set of skills each of which consists of several abilities. All the abilities and parts of the attributes are a function of the character's level, called the Ability-Increasing Functions (AIFs). So the main task of balancing PvP is to construct a set of AIFs by which there are no dominating strategies in the game world and each race should have the same probability to win unless designers have some special design purposes.

Traditionally, game companies solve the balance problem by designer's talent and human testing. In the case of single-player games and some multi-player games which do not allow players to fight with each other, called PvE games, such approach can handle the problem very well, even though it is expensive in both time and resources. The challenges of PvE games come from the environment rather than other players, so there is only one aim of the tuning process, namely to satisfy the human side. That is why the traditional way succeeded. However, in the case of on-line RPG, the traditional method causes two problems. Firstly, since designers can not ensure the inexistence of dominating strategies, testers have to take a lot of time to play the games and even human testers can not explore all strategies. Secondly, once a dominating strategy is identified, all the AIFs need to be modified to make the game in balance, which is too complicated for human testing to handle. For instance(refer to **Figure 1**), supposing there are three races (A, B, C) in the game world, designers may modify B's AIFs to get a balanced relationship between A and B, then similarly modify C's AIFs to get the balanced relationship between B and C. After that, the relationship between C and A becomes determinate because all the AIFs have already been constructed, which means that the last relationship is uncontrollable. This is a multi-objective optimization problem which will become hard to solve with the number of races increasing. The number of uncontrollable relationships, denoted by NUR, can be calculated by the following formula:

$$\mathrm{NUR} = \binom{n}{2} - n + 1 \qquad (1)$$

where n is the number of races.

Leigh, *et al.* [3] have presented a solution for balancing a two-player, real-time action game called CaST. They use the competitive coevolutionary algorithm to search the dominating strategies and once found, tune the game rules and parameters. This solution highlights game imbalance as well as provides intuition towards balancing the game. It can be applied to on-line RPGs with some improvements. Alternatively, Olsen [4] has described the outline of an automated testing framework for on-line RPGs. In such framework, game designers firstly create all AIFs empirically. Then the in-game combat data, recorded in game log, is used to construct a set of decision-making models by which a set of complex AI systems can be built. Based on those AIs, combats are simulated to verify whether one race is superior to another. If so, the AIFs will be continually tuned, until the game is in balance. Moreover, the decision-making models are refined periodically and after each refinement, the combat simulations as well as the tuning tasks should be redone (refer to **Figure 2**). This approach can be regarded as a kind of dynamic balance adjustment in which the dominating strategies are identified dynamically.

However, in those two solutions, the AIFs still will be adjusted by hand-tuning which is inefficient and time-consuming. Moreover, since strengthening one race will definitely weaken the others, the result of tuning operations may, somehow, become worse. In this paper, we propose an evolutionary method for on-line RPGs to obtain a set of well-balanced AIFs. The method includes integration with an improved Probabilistic Incremental Program Evolution (PIPE) and a Cooperative Coevolutionary Algorithm (CCEA).

The remainder of this paper is organized as follows. Section 2 presents a brief but necessary overview of genetic algorithm and genetic programming. Section 3 explains the CCEA technology, followed by the details of PIPE in Section 4. Section 5 introduces the proposed

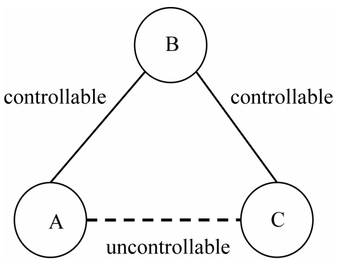

Figure 1. There is one uncontrollable relationship in the game having three races.

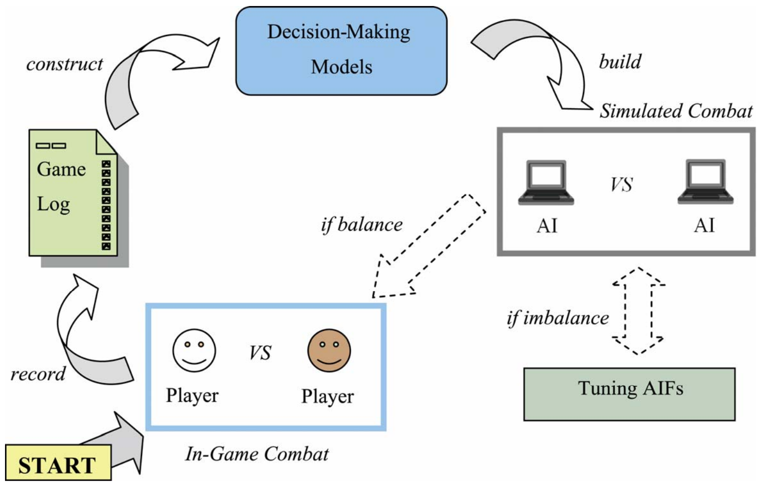

Figure 2. Working process of the automated testing framework.

design method as well as the simplest turn-based on-line RPG model. Section 6 demonstrates the experiments and their results. Conclusions and possible future research directions are placed in the last section.

2. Genetic Algorithm and Genetic Programming

Genetic Algorithm (GA) [5] is a probabilistic search algorithm based on the mechanics of natural selection and natural genetics. It iteratively transforms a population of objects, usually a fixed-length binary string and each with an associated fitness value, into a new population of offspring objects by using three operators that are selection, crossover and mutation.

The theoretical basis of GA, the schema theorem, formalized by Holland [5] and popularized by Goldberg [6]. A schema, $H = [*11*0**]$, is a template for the 7-length individuals in which 2nd and 3rd locus are "1" while 5th locus is "0". The asterisk "*" is the wild card symbol which matches either a "0" or a "1" at a particular position. Schema order, denoted by $o(H)$, is the number of non "*" genes in schema *H*. Moreover, schema defining length, denoted by $\delta(H)$, is the distance between first and last non "*" gene in schema *H*. The schema theorem is given as follow:

Theorem 1. *Given a simple GA with proportional selection, single point crossover and gene wise mutation. Then the expected number of schema H at generation* $t+1$ *is*,

$$E\left[m(H,t+1)\right] = m(H,t)\cdot\frac{f(H)}{\bar{f}}\left[1-\mu\right]$$

where

$$0 \le \mu \le p_c \cdot \frac{\delta(H)}{l-1} + o(H)\,p_m$$

Here, $m(H,t)$ is the number of individuals matching schema *H* at generation *t*, $f(H)$ denotes the mean fitness of individuals matching schema *H*, $\bar{f}$ is the mean fitness of individuals in the population of generation *t*, p_c and p_m denote the crossover rate and the mutation rate, respectively.

It means that short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm. If we consider the algorithm without variational operators, then $\mu = 0$.

Genetic programming (GP) [7-9] is a specialization of GA in which each individual of the population is a tree-structured program. It is a machine learning technique used to automatically solve problems without requiring the user to know or specify the form or structure of the solution in advance. At the most abstract level, GP is a systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done.

Figure 3 shows the tree structure of the program: $\cos x + \sin(x * y)$, where $\{x, y\}$ is called the terminal set and $\{\cos, \sin, +, *\}$ is the function set. Such two sets together form the solutions of the target problem. While GA is usually concerned with modifying fixed-length strings, which associated with parameters of a function, GP is concerned with actually creating and manipulating the (non-fixed length) structure of the program (or function). Consequently, GP is a much more complex and difficult topic.

3. Cooperative Coevolutionary Algorithm

CCEAs [10,11] have been applied to solve large and complex problems, such as multiagent systems [12-14], rule learning [15,16], fuzzy modeling [17], and neural network training [18]. It models an ecosystem which consists of two or more species. Mating restrictions are enforced simply by evolving the species in separate populations which interact with one another within a shared domain model and have a cooperative relationship. The original architecture of the CCEA for optimization can be summarized as follows:

1) Problem Decomposition: Decompose the target problem into smaller subcomponents and assign each of the subcomponents to a population.

2) Subcomponents Interaction: Combine the individual of a particular population with representatives selected from others to form a complete solution, then evaluate the solution and attribute the score to the individual for fitness.

3) Subcomponent Optimization: Evolve each population separately by using a different evolutionary algorithm, in turn.

The empirical analyses have shown that the power of CCEAs depends on the decomposition work as well as separate evolving of these populations resulting in significant speedups over simple GAs [19-21]. Here, we give the theoretical evidence of such results with the following two assumptions.

1) The elitists of CCEA populations are chosen as the representatives.

2) There are no variational operators in both the simple

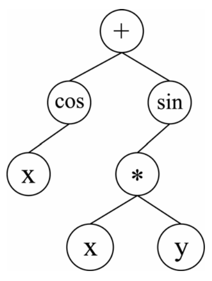

Figure 3. The tree structure of cos *x* + sin(*x*y*).**

GA and CCEA.

Let's begin with some definitions.

Definition 1. Given schemata: $H_1, H_2, \cdots, H_n$, where H_i denotes a schema of the *i*th CCEA population, the n-expanded schema, denoted by H_1^n, is the sequential concatenation of the *n* schemata. For example, let $H_1 = [1*0*]$, $H_2 = [*1*1]$, then $H_1^2 = [1*0**1*1]$.

Definition 2. Let there be *n* populations in the CCEA. A complete genotype is the sequential concatenation of *n* individuals selected from *n* different populations. If all the individuals are representatives, then the complete genotype is the best one.

Definition 3. Given an individual *I* of the *i*th CCEA population, the expanded genotype of *I* is the best complete genotype in which the *i*th representative is replaced by *I*.

Definition 4. Given a target problem *f*, the two algorithms, simple GA and CCEA, are comparable if the population of the simple GA consists of all the expanded genotypes in the CCEA.

Theorem 2. *Let a target problem f be decomposed into n subcomponents, r_i be the increasing rate of the individuals matching H_i in the ith CCEA population, r_{CCEA} be the increasing rate of the complete genotypes matching H_1^n in the CCEA, and r_{SGA} be the increasing rate of the individuals matching H_1^n in the simple GA. If the two algorithms are comparable, then,*

1) $r_{\text{SGA}} < \sum_{i=1}^{n} r_i$

2) $r_{\text{CCEA}} = k \cdot \left(\min\{r_1, r_2, \cdots, r_n\}\right)^n, \quad k \ge 1$

Proof. Since the two algorithms are comparable, in the simple GA, the number of individuals matching H_1^n at generation t can be calculated by:

$$m\left(H_1^n, t\right) = \sum_{i=1}^{n} M\left(H_i, t\right)$$

where $M\left(H_i, t\right)$ denotes the number of individuals matching H_i at generation *t*, in the *i*th CCEA population. Then according to the schema theorem (refer to Theorem 1), we have,

$$E\left[m\left(H_1^n, t+1\right)\right] \le m\left(H_1^n, t\right) \cdot \frac{\sum_{i=1}^{n} \overline{f\left(H_i, t\right)}}{\sum_{i=1}^{n} \overline{F\left(i, t\right)}}$$
$$< m\left(H_1^n, t\right) \cdot \sum_{i=1}^{n} r_i$$

where in the *i*th CCEA population, $\overline{f\left(H_i, t\right)}$ and $\overline{F\left(i, t\right)}$ denote the mean fitness of individuals matching H_i and the mean fitness of all individuals, at generation *t*, respectively.

In the case of CCEA, because H_1^n is the conjunction of H_i, the number of the complete genotypes matching H_1^n at generation *t* is given by:

$$M\left(H_1^n, t\right) = \prod_{i=1}^{n} M\left(H_i, t\right)$$

Again, according to the schema theorem, we obtain the following equation:

$$E\left[M\left(H_i, t+1\right)\right] = M\left(H_i, t\right) \cdot r_i$$

Then,

$$\begin{aligned} E\left[M\left(H_1^n, t+1\right)\right] &= \prod_{i=1}^{n} E\left[M\left(H_i, t+1\right)\right] \\ &= \prod_{i=1}^{n} M\left(H_i, t\right) \cdot r_i \\ &= M\left(H_1^n, t\right) \cdot k \cdot \left(\min\{r_1, r_2, \cdots, r_n\}\right)^n \end{aligned}$$

where $k = \prod_{i=1}^{n} \frac{r_i}{\min\{r_1, r_2, \cdots, r_n\}} \ge 1$.

Hence, obviously, with the increasing of the $\min\{r_1, \cdots, r_n\}$, H_1^n will receive a much higher increasing rate in the CCEA.

However, Theorem 2 does not mean that CCEAs is superior to simple GAs, which depends on the target problems. Actually, since the representatives are necessary in calculating the fitness of the individual of an arbitrary population, the relationships between populations impose a great influence on the efficiency of CCEAs [22, 23]. It has been proved that even with prefect information, infinite population size and no variational operators, CCEAs can be expected to converge to suboptimal solution [23], while simple GAs does not suffer from such affliction [24-26]. However, Liviu [27] has emphasized that CCEAs will settle in the globally optimal solution with arbitrarily high probability, when properly set and if given enough resources.

4. Probabilistic Incremental Program Evolution

PIPE [28] is a technology for synthesizing programs. Unlike traditional GP, It creates population according to an adaptive probability distribution over all possible programs with respect to a predefined instruction set. In each generation, the distribution is refined by using the information learned from the best program or the best program found so far (elitist) to guide the evolutionary search [29]. It has been shown that PIPE achieved better results than traditional GP in solving function regression problem and 6-bit parity problem [28].

PIPE stores the probability distribution in the Probabilistic Prototype Tree (PPT), which is generally a complete n-ary tree with infinitely many nodes, where n is the maximal number of function arguments. Each node N_j in a PPT consists of a value R_j which is randomly taken from a problem-dependent set of constants and a variable probability vector $\vec{P}_j$ for instruction set, which is initialized as follows.

$$\begin{aligned} P_j(I) &= \frac{P_T}{l}, \forall I : I \in T \\ P_j(I) &= \frac{1-P_T}{l}, \forall I : I \in F \end{aligned} \qquad (2)$$

where P_T is a predefined, constant probability for selecting an instruction from T (the terminal set), l is the total number of terminals in T and k is the total number of functions in F (the function set).

As we mentioned above, there are two kinds of learning mechanisms in the PIPE, generation-based learning (GBL) and elitist learning (EL). GBL is the main learning algorithm and the purpose of EL is to use the best program found so far as an attractor. The pseudo-code of PIPE follows:

```
1) GBL.
2) REPEAT.
3) With probability $P_{el}$ DO EL.
4) Otherwise DO GBL.
5) UNTIL termination criterion is reached.
```

where P_{el} is a user-defined constant in $[0,1]$.

The process of GBL can be divided into five distinct phases:

1) Creation of program population. A population of programs P_{ROG_j} ($0 < j < PS$; PS is population size) is generated using the PPT.

2) Population evaluation. Each program P_{ROG_j} of the current population is evaluated on the given task and assigned a fitness value $FIT\left(P_{\mathrm{ROG}_j}\right)$ according to the predefined fitness function. The best program of the current population is denoted by P_{ROG_b}. The elitist is preserved in $P_{\mathrm{ROG}^{el}}$.

3) Learning from population. Prototype tree probabilities are modified such that the probability $P\left(P_{\mathrm{ROG}_b}\right)$ of creating P_{ROG_b} increases. This procedure is called adapt_PPT_towards $\left(P_{\mathrm{ROG}_b}\right)$ which is implemented as follows.

First $P\left(P_{\mathrm{ROG}_b}\right)$ is computed by looking at all PPT nodes N_j used to generate P_{ROG_b}:

$$P\left(P_{\mathrm{ROG}_b}\right) = \prod_{j:N_j \text{ used to generate } P_{\mathrm{ROG}_b}} P_j\left(I_j\left(P_{\mathrm{ROG}_b}\right)\right) \qquad (3)$$

where $P\left(P_{\mathrm{ROG}_b}\right)$ denotes the instruction of program P_{ROG_b} at node position j.

Then a target probability P_{TARGET} for P_{ROG_b} is calculated:

$$\begin{aligned} P_{\mathrm{TARGET}} &= P\left(P_{\mathrm{ROG}_b}\right) \\ &+ \left(1 - P\left(P_{\mathrm{ROG}_b}\right)\right) \cdot lr \cdot \frac{\varepsilon + FIT\left(P_{\mathrm{ROG}^{el}}\right)}{\varepsilon + FIT\left(P_{\mathrm{ROG}_b}\right)} \end{aligned} \qquad (4)$$

where lr is a constant learning rate and ε is a positive user-defined constant. Given P_{TARGET}, all single node probabilities $P_j\left(I_j\left(P_{\mathrm{ROG}_b}\right)\right)$ are increased iteratively:

```
a) REPEAT
b) $P_j\left(I_j\left(P_{\mathrm{ROG}_b}\right)\right) = P_j\left(I_j\left(P_{\mathrm{ROG}_b}\right)\right) + c^{lr} \cdot lr \cdot \left(1 - P_j\left(I_j\left(P_{\mathrm{ROG}_b}\right)\right)\right)$
c) UNTIL $P\left(P_{\mathrm{ROG}_b}\right) \geq P_{\mathrm{TARGET}}$
```

where c^{lr} is a constant influencing the number of iterations.

4) Mutation of prototype tree. All probabilities $P_j(I)$ stored in nodes N_j that were accessed to generate program P_{ROG_b} are mutated with probability P_{M_p}:

$$P_{M_p} = \frac{P_M}{z \cdot \sqrt{\left|P_{\mathrm{ROG}_b}\right|}} \qquad (5)$$

where the user-defined parameter P_M denotes the overall mutation probability, z is the number of instructions in instruction set S and $\left|P_{\mathrm{ROG}_b}\right|$ denotes the number of nodes in program P_{ROG_b}. Selected probability vector components are then mutated as follows:

$$P_j(I) = P_j(I) + mr \cdot \left(1 - P_j(I)\right) \qquad (6)$$

where mr is the mutation rate, another user-defined parameter. All mutated vectors $\vec{P}_j$ are finally renormalized:

$$P_j(I) = \frac{P_j(I)}{\sum_{I^* \in S} P_j\left(I^*\right)} \quad \forall P_j(I) : I \in S \qquad (7)$$

5) Prototype tree pruning. At the end of each generation the prototype tree is pruned. PPT subtrees attached to nodes that contain at least one probability vector component above a threshold T_p can be pruned. The pruning operation results in more concise population than traditional GP.

In the EL, on the other hand, adapt_PPT_towards

$\left(P_{\mathrm{ROG}_b}\right)$ is modified to adapt_PPT_towards $\left(P_{\mathrm{ROG}^{el}}\right)$, that is, PPT is adapted towards the elitist program without creating and evaluating population.

We notice that PIPE does not use the Elitist-Preserving Strategy (EPS) in which the individual with the highest fitness survives to be an individual of next generation. The EPS is a useful option by which, in some case, the performance of evolutionary algorithms can be improved [30-32]. In fact, there is a serious problem in applying it to PIPE, that is, if a low-quality elitist hasn't been replaced for many generations, PPT will converge to the suboptimal solution. So, in order to enable the EPS option, an improvement has been made by adding a factor, named satisfied fitness value (FIT_c), into Formula (4). The improved one is given by:

$$P_{\mathrm{TARGET}} = P\left(P_{\mathrm{ROG}_b}\right) + \left(1 - P\left(P_{\mathrm{ROG}_b}\right)\right) \cdot lr \cdot \frac{\varepsilon + FIT\left(P_{\mathrm{ROG}^{el}}\right)}{\varepsilon + FIT\left(P_{\mathrm{ROG}_b}\right)} \cdot \exp\left(-\frac{\left|FIT_c - FIT\left(P_{\mathrm{ROG}_b}\right)\right|}{K}\right) \tag{8}$$

where K is a positive user-defined constant which used to control the pressure of learning. It is important to note that once PIPE employs the EPS, the EL will no longer be necessary.

5. Design Method

With the assumption that all the dominating strategies have been identified, the task of balancing PvP is reduced to constructing a set of AIFs by which each race has the same probability to win. In order to solve this multi-objective optimization problem, we build a single aggregate objective function by weighted linear sum of the objectives, which means the relationships between the AIFs become cooperative. Therefore, we propose to use the CCEA to search the optimal solution, where each population is evolved by the improved PIPE. The pseudo-code of the proposed method follows:

```
1) Create a (population, PPT) pair for each AIF.
2) FOR population $p_i \in P$, all population.
3) Initialize $PPT_i$.
4) Construct $p_i$ by $PPT_i$.
5) Select representative $P^i_{ROG^{rep}}$ randomly.
6) $FIT\left(P^i_{ROG^{rep}}\right) = +\infty$.
7) END FOR.
8) REPEAT.
9) FOR population $p_i \in P$, all population.
10) Evaluate individual of $p_i$ with representatives from the others.
11) IF $FIT\left(P_{\mathrm{ROG}_b}\right) < FIT\left(P^i_{\mathrm{ROG}^{rep}}\right)$.
12) Replace $P^i_{\mathrm{ROG}^{rep}}$ with $P_{\mathrm{ROG}_b}$.
13) END IF.
14) Learn from $P_{ROG_b}$ and refine $PPT_i$.
15) Reproduce $p_i$ with EPS.
16) END FOR.
17) UNTIL the termination criteria are met.
```

In an arbitrary group, the win probability of combatant 1 is regarded as a discrete random variable all outcomes of which are equally likely. So, similar to the portfolio theory, we define the fitness function mainly by using the mean and variance of the win probability of the combatant. The better individuals will return lower values in evaluation process. The function is given by:

$$FIT\left(P_{\mathrm{ROG}}\right) = \sum_{g=1}^{m}\left\{w_1 \cdot \left[E\left(P_g\right) - \xi_g\right]^2 + V\left(P_g\right)\right\} + w_2 \cdot f_{\mathrm{rule}}\left(S\right) \tag{9}$$

where $m = \binom{n}{2}$ denotes the number of "1 versus 1" groups, n is the number of races, $P_g = \left\{p_{1g}\cdots, p_{lg}\cdots, p_{Lg}\right\}$, a random variable, is the win probability of combatant 1 of group g, p_{lg} is it's value at level l, and L is the maximum level. $\xi_g \in (0,1)$, called balance factor, is a user-defined constant, $f_{rule}(S) \geq 0$ denotes how good the complete solution S conforms to the designed rules.

In order to test the proposed method, we construct a simplest turn-based on-line RPG model as the platform for experiments. The contents of design are listed as follows:

1) There are three distinct races ($R1$, $R2$, $R3$) in the game world with the maximum level of L.

2) Each race has only one skill and two attributes which are health and dodge rate.

3) Each skill includes only one ability, the physical damage.

so, p_{lg} can be calculated by the following formula approximately.

$$p_{lg} = \sum_{i=1}^{T}\frac{1}{2} \cdot p_l\left(F_i\right)\left[p_l\left(F'_{i-1}\right) + p_l\left(F'_i\right)\right] \tag{10}$$

where

$$p_l\left(F_i\right) = \begin{cases} \binom{i-1}{LT_2(l)-1}\left(1-D_2\right)^{LT_2(l)} D_2^{i-LT_2(l)} & i \geq LT_2(l) \\ 0 & i < LT_2(l) \end{cases}$$

$$p_l\left(F'_i\right) = \begin{cases} \sum_{j=0}^{LT_1(l)-1}\binom{i}{j}\left(1-D_1\right)^j D_1^{i-j} & i \geq LT_1(l) \\ 1 & i < LT_1(l) \end{cases}$$

$$LT_2(l)=\left\lceil \frac{h_2(l)}{d_1(l)} \right\rceil \quad LT_1(l)=\left\lceil \frac{h_1(l)}{d_2(l)} \right\rceil$$

Here, we suppose that combat ends within T rounds, $p_l(F_i)$ denotes the probability that combatant 1 launch the lethal attack at round i, $p_l(F_i')$ is the probability that combatant 2 hasn't launched the lethal attack before round i + 1, D_k, LT_k, h_k, d_k refer to the dodge rate, lifetime, health value and damage value of combatant k, respectively, where $k=\{1,2\}$.

6. Experiments and Results

In this section, we conduct the comparison of the proposed algorithm, denoted by CCEA_I, with the proposed algorithm using original PIPE instead of the improved one, denoted by CCEA_O, and the proposed algorithm using the simple GA instead of the CCEA, denoted by SGA_I. The environments are set as follows.

The Game Model:

1) The maximum level: $L=10$.

2) The dodge rate: $D_{R1}=0.2$, $D_{R2}=0.6$, $D_{R3}=0.1$.

3) Rules of health value: $h_{R1}(l)>h_{R2}(l)>h_{R3}(l)$.

4) Rules of damage value: $d_{R3}(l)>d_{R1}(l)>d_{R2}(l)$.

5) All the AIFs must be monotone increasing and $LT_k(l)\le 20$, $k=\{1,2\}$.

The Fitness Function: [(1)]

1) Balance factor: $\xi_1=\xi_2=\xi_3=0.5$.

2) $w_1=10$, $w_2=20$, $T=30$.

The Improved and Original PIPE: [(1)]

1) Function and terminal sets have been used: $F=\{+,-,*,\sin,\cos,\exp\}$ and $T=\{R,l\}$, $R\in[0,50]$ denotes the generic random constant.

2) The termination criteria: maximal generation = 300 and $FIT_c=0.075$.

3) $lr=0.01$, $c^{lr}=0.1$, $P_M=0.4$, $mr=0.3$, $T_P=0.999999$, $K=2$.

The CCEA:

1) population size = 100, the number of populations = 6.

The Simple GA:

1) population size = 600.

Considering the stochastic nature of evolutionary algorithms, each of the algorithms is repeated 100 times. **Figure 4** depicts a set of AIFs corresponding to the smallest fitness obtained by CCEA_I. It is important to note that all the designed rules are well obeyed. To verify whether the AIFs are well-balanced, according to Formula 10, we calculate p_{lg} and show the mean and variance of P_g in **Table 1**, which indicates that $E(P_g)$ is close to 0.5 and $V(P_g)$ is close to 0, for $\forall g\in\{1,2,3\}$. Hence, the AIFs can be accepted as well-balanced.

We compare the algorithms both in terms of final best-ever fitness and computation time. The comparison results are plotted in **Figure 5**. In this case, it is obvious that both CCEA_I and CCEA_O are superior to SGA_I. By counting the number of executions in which the final best-ever fitness is larger than 10, we obtain 6 for CCEA_I while 26 for CCEA_O. Moreover, the average computation time of 100 executions is 35.6991 for CCEA_I, whereas the value is 56.2348 with respect to CCEA_O. Hence, in case that PIPE employs the EPS, the capability of PIPE has been improved significantly by the improvement work.

7. Conclusion and Future Work

This paper presented an evolutionary design method which contains integration with a CCEA and an improved PIPE for solving the balance problem of on-line RPGs. We demonstrated the theoretical evidence of why CCEAs, in some case, is faster than simple GAs, and in order to enable the EPS option, we improved the learning mechanism of PIPE. Through a series of experiments, we have shown that the proposed method is the most efficient way to obtain a set of well-balanced AIFs for the

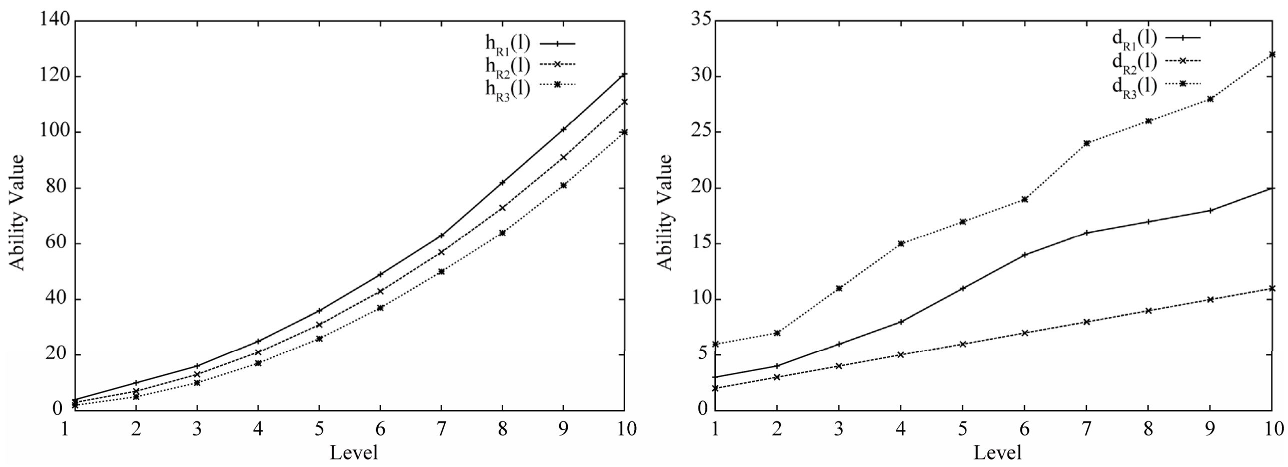

Figure 4. The best AIFs obtained by the proposed algorithm.

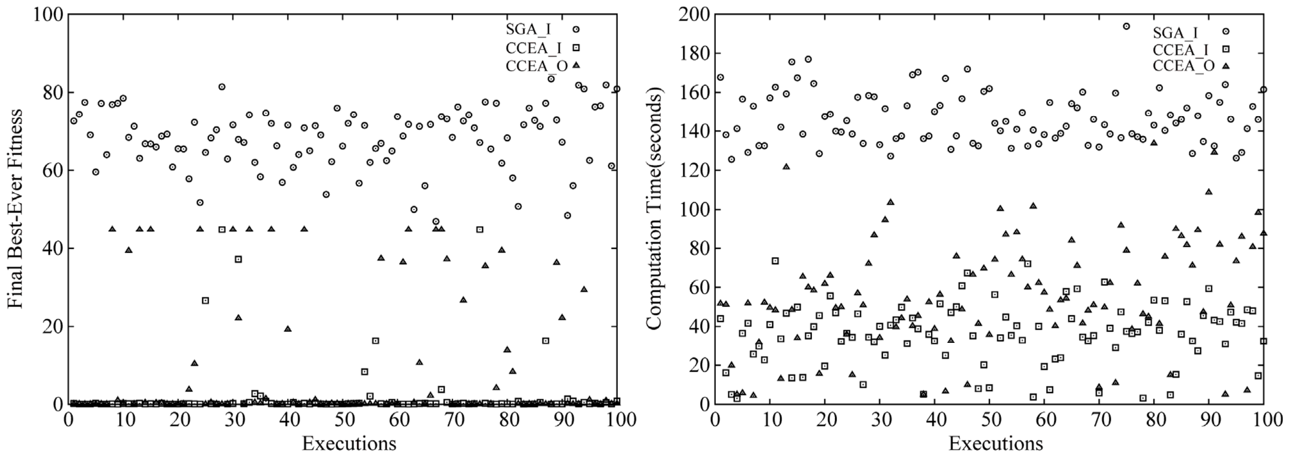

Figure 5. Performance comparison of the three algorithms in terms of final best-ever fitness as well as computation time with 100 executions.

Table 1. Verification data.

Group	$g = 1$ (A vs B)	$g = 2$ (B vs C)	$g = 3$ (C vs A)
$E(P_g)$	0.4803	0.4734	0.532
$V(P_g)$	0.0115	0.0233	0.0323

target game model.

The game model presented above is a turn-based game in which each race has only one skill with only one ability. While this may seem an extreme simplification, it should be noted that the original intent of this work was to lay a foundation of study, and that foundation would begin with simpler systems for which behavioral understanding is more tractable. In future works, we will construct an action-based game model, which is much more complicated than the turn-based one. In such game model, since there are no formulas for calculating the win probability, we have to use the Monte Carlo method which costs too much for evolutionary algorithms. Therefore, we plan to model the combat using the discrete competitive Markov decision process by which the win probability can be obtained quickly.

REFERENCES

[1] A. Rollings and D. Morris, "Game Architecture and Design: A New Edition," New Readers, Indianapolis, 2004.

[2] E. Adams, "Fundamentals of Game Design," 2nd Edition, New Readers, Berkeley, 2009.

[3] R. Leigh, J. Schonfeld and S. Louis, "Using Coevolution to Understand and Validate Game Balance in Continuous Games," *Proceedings of the* 10*th Annual Conference on Genetic and Evolutionary Computation*, Atlanta, 12-16 July 2008, pp. 1563-1570. doi:10.1145/1389095.1389394

[4] J. M. Olsen, "Game Balance and AI Using Payoff Matrices," In: B. S. Jones and T. Alexander, Eds., *Massively Multiplayer Game Development*, Charles River Media Inc., Hingham, 2002, pp. 38-48.

[5] J. H. Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology," *Control and Artificial Intelligence*, MIT Press, Cambridge, 1975.

[6] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning," Addison Wesley, San Francisco, 1989.

[7] N. L. Cramer, "A Representation for the Adaptive Generation of Simple Sequential Programs," *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, 24-26 July 1985, pp. 183-187.

[8] J. R. Koza, "Genetic Programming—On the Programming of Computers by Means of Natural Selection," MIT Press, Cambridge, 1992.

[9] R. Poli, W. B. Langdon and N. F. Mcphee, "A Field Guide to Genetic Programming," Lulu, Morrisville, 2008.

[10] P. Husbands and F. Mill, "Simulated Coevolution as the Mechanism for Emergent Planning and Scheduling," *Proceedings of the* 4*th International Conference on Genetic Algorithms*, San Diego, 13-16 July 1991, pp. 264-270.

[11] M. Potter, "The Design and Analysis of a Computational Model of Cooperative Coevolution," Ph.D. Thesis, George Mason University, Fairfax, 1997.

[12] L. Bull, T. C. Fogarty and M. Snaith, "Evolution in Multi-Agent Systems: Evolving Communicating Classifier Systems for Gait in a Quadrupedal Robot," *Proceedings of the* 6*th International Conference on Genetic Algorithms*, Pittsburgh, 15-19 July 1995, pp. 382-388.

[13] M. Potter, L. Meeden and A. Schultz, "Heterogeneity in the Coevolved Behaviors of Mobile Robots: The Emergence of Specialists," *Proceedings of the* 17*th International Conference on Artificial Intelligence*, Seattle, 4 August 2001, pp. 1337-1343.

[14] K. S. Hwang, J. L. Lin and H. L. Huang, "Dynamic Patrol Planning in a Cooperative Multi-Robot System," *Communications in Computer and Information Science*, Vol. 212, 2011, pp. 116-123.

doi:10.1007/978-3-642-23147-6_14

[15] M. Potter and K. De Jong, "The Coevolution of Antibodies for Concept Learning," *Proceedings of the* 5th *International Conference on Parallel Problem Solving from Nature*, Kraków, 11-15 September 1998, pp. 530-539. doi:10.1007/BFb0056895

[16] Y. Wen and H. Xu, "A Cooperative Coevolution-Based Pittsburgh Learning Classifier System Embedded with Memetic Feature Selection," *Proceedings of IEEE Congress on Evolutionary Computation*, New Orleans, 5-8 June 2011, pp. 2415-2422.

[17] A. Carlos and S. Moshe, "Fuzzy CoCo: A Cooperative-Coevolutionary Approach to Fuzzy Modeling," *IEEE Transactions on Fuzzy Systems*, Vol. 9, No. 5, 2001, pp. 727-737. doi:10.1109/91.963759

[18] R. Chandra and M. Zhang, "Cooperative Coevolution of Elman Recurrent Neural Networks for Chaotic Time Series Prediction," *Neurocomputing*, Vol. 86, 2012, pp. 116-123. doi:10.1016/j.neucom.2012.01.014

[19] M. Potter and K. De Jong, "A Cooperative Coevolutionary Approach to Function Optimization," *Proceedings of the* 3rd *International Conference on Parallel Problem Solving from Nature*, Jerusalem, 9-14 October 1994, pp. 249-257. doi:10.1007/3-540-58484-6_269

[20] M. Potter and K. De Jong, "Cooperative Coevolution: An Architecture for Evolving Co-Adapted Subcomponents," *Evolutionary Computation*, Vol. 1, No. 8, 2000, pp. 1-29. doi:10.1162/106365600568086

[21] R. P. Wiegand, W. Liles and K. De Jong, "An Empirical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms," *Proceedings of the Genetic and Evolutionary Computation Conference*, Dublin, 12-16 July 2001, pp. 1235-1242.

[22] T. Jansen and R. P. Wiegand, "Exploring the Explorative Advantage of the CC (1+1) EA," *Proceedings of the Genetic and Evolutionary Computation Conference*, Chicago, 12-16 July 2003, pp. 310-321.

[23] R. P. Wiegand, "An Analysis of Cooperative Coevolutionary Algorithms," Ph.D. Thesis, George Mason University, Fairfax, 2004.

[24] C. Reeves and J. Rowe, "Genetic Algorithms Principles and Perspectives: A Guide to GA Theory," Kluwer, New York, 2002.

[25] L. Schmitt, "Theory of Genetic Algorithms," *Theoretical Computer Science*, Vol. 259, No. 1-2, 2001, pp. 1-61. doi:10.1016/S0304-3975(00)00406-0

[26] M. Vose, "The Simple Genetic Algorithm," MIT Press, Cambridge, 1999.

[27] P. Liviu, "Theoretical Convergence Guarantees for Cooperative Coevolutionary Algorithms," *Evolution Computation*, Vol. 18, No. 4, 2010, pp. 581-615.

[28] R. P. Salustowicz and J. Schmidhuber, "Probabilistic Incremental Program Evolution," *Evolutionary Computation*, Vol. 5, No. 2, 1997, pp. 123-141. doi:10.1162/evco.1997.5.2.123

[29] S. Baluja, "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning," *Technical Report*, CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, 1994.

[30] K. De Jong, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. Thesis, University of Michigan, Ann Arbor, 1975.

[31] A. Fujino, T. Tobita, K. Segawa, K. Yoneda and A. Togawa, "An Elevator Group Control System with Floor-Attribute Control Method and System Optimization Using Genetic Algorithms," *IEEE Transactions on Industrial Electronics*, Vol. 44, No. 4, 1997, pp. 546-552. doi:10.1109/41.605632

[32] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *Evolutionary Computation*, Vol. 6, No. 2, 2002, pp. 182-197. doi:10.1109/4235.996017