Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Solving the balance problem of massively multiplayer online role-playing games using coevolutionary programming

Haoyang Chen*, Yasukuni Mori, Ikuo Matsuba

Advanced Integration Science, Graduate School of Chiba University, Japan

A R T I C L E   I N F O

Article history:
Received 15 December 2011
Received in revised form 20 October 2013
Accepted 6 January 2014
Available online 24 January 2014

Keywords:
Game balance
Game design
Coevolutionary programming
Cooperative coevolutionary algorithm
Probabilistic incremental program evolution
Multi-objective optimization

A B S T R A C T

In massively multiplayer online role-playing games (MMORPGs), each race holds some attributes and skills. Each skill contains several abilities such as physical damage and hit rate. All those attributes and abilities are functions of the character's level, which are called Ability-Increasing Functions (AIFs). A well-balanced MMORPG is characterized by having a set of well-balanced AIFs. In this paper, we propose a coevolutionary design method, including integration with the modified probabilistic incremental program evolution (PIPE) and the cooperative coevolutionary algorithm (CCEA), to solve the balance problem of MMORPGs. 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 more efficiently, compared with the simple genetic algorithm (SGA), the simulated annealing algorithm (SAA) and the hybrid discrete particle swarm optimization (HDPSO) algorithm. The results also show that the performance of PIPE has been significantly improved through the modification works.

1. Introduction

In recent years, massively multiplayer online role-playing games (MMORPGs) become more and more popular all over the world. The global market has risen above $12 billion in 2012 and is forecasted to reach $17.5 billion in 2015. A MMORPG is a multi-player persistent virtual world which consists of several distinct races. Unlike the board games (such as Chess, Shogi, and Go), which are symmetrical systems, MMORPGs are regarded as asymmetrical since each race is designed to hold some unique features. For example: the wizard can perform spells, while the knight specializes in close fighting. Before starting the game, players should first create one character from a certain race, then play the game by either exploring the virtual world or interacting with the other players.

In MMORPGs, the concept of balance can be divided into two parts, the balance of Player-versus-Environment (PvE) and the balance of Player-versus-Player (PvP) [1]. In this paper, specifically, the term PvP refers to "1 versus 1" combat because it is the core of the MMORPG'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 among the races. When designing a MMORPG, the power of a race will be represented by some attributes and a set of skills each of which consists of several abilities. In most cases, the attributes and the abilities must be non-negative integers, and they are functions 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. For the balance of PvE, such approach can handle very well, even though it is expensive in both time and resources. The challenges of balancing PvE 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 balancing PvP, the traditional method causes two problems. Firstly, since designers cannot ensure the inexistence of dominating strategies, testers have to take a lot of time to play the games and even human testers cannot explore all strategies. Secondly, once a dominating strategy is identified, all the AIFs need to be modified to ensure the game is in balance, which is too complicated for human testing to handle. For instance (refer to Fig. 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

Abbreviations: AIF, Ability-Increasing Function; ACTR, Average Computation Time Ratio; NUR, Number of Uncontrollable Relationships.

* Corresponding author. Tel.: +81 080 4686 1413.
E-mail addresses: hust.chenhaoyang@gmail.com (H. Chen), yasukuni@faculty.chiba-u.jp (Y. Mori), matsuba@faculty.chiba-u.jp (I. Matsuba).

1568-4946/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2014.01.011

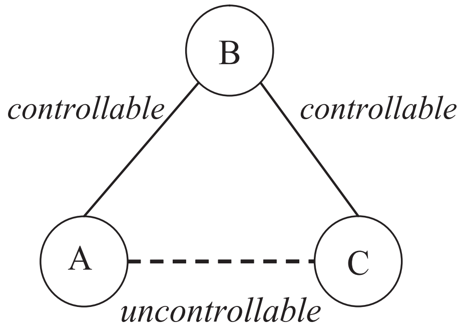


Fig. 1. There is one uncontrollable relationship in the game having three races.

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.

Tuning game balance 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.

Moreover, since the nonlinearity of the objective functions, it can also be regarded as a nonlinear integer programming problem, which belongs to the class of NP-hard problems [2]. Therefore, there does not exist an exact algorithm, which can solve the problem in time, depending polynomially on the problem input data length or on the problem size. In this circumstance, heuristic algorithms, such as genetic algorithm (GA), become the only viable option because they find approximate solutions but have acceptable time and space complexity.

In the game industry, there are two kinds of revenue models, profiting from the game software and profiting from the game services. For most traditional games (such as console-based games, etc.), game companies employ the former one, that is, they earn money by selling the game software. However, in the case of MMO games, the latter one is preferred. The game software of MMO games are free to install, but players should subscribe to the game before they can play. In this revenue model, game companies make money by selling either the game time or the virtual items. Fig. 2 shows the sequence diagrams of the revenue models. In the left part, the yellow activation box refers to the life cycle of the game which means the total game time that the player spent in playing the game. But in the right part, the life cycle equals to the summation of all the yellow activation boxes. Since the free game software, it's obvious that the life cycle is much more crucial to the MMO games. If the players stop the subscription, the incomes of the game company will definitely decrease. Therefore, having a long life cycle has become the most important feature of the high-quality MMO games.

In the case of MMORPG, beside the user interface and the story line, the game balance also directly affects the life cycle of the game. For example, if we let a crab and a duck to play the rock-paper-scissors game, undoubtedly, the duck will quit the game as quickly as possible because it always loses. Generally, to be enjoyable, the game must be balanced well [3]. 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.

Leigh et al. [4] 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 [5] has described the outline of an automated testing framework for MMORPGs. 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 Fig. 3). This approach can be regarded

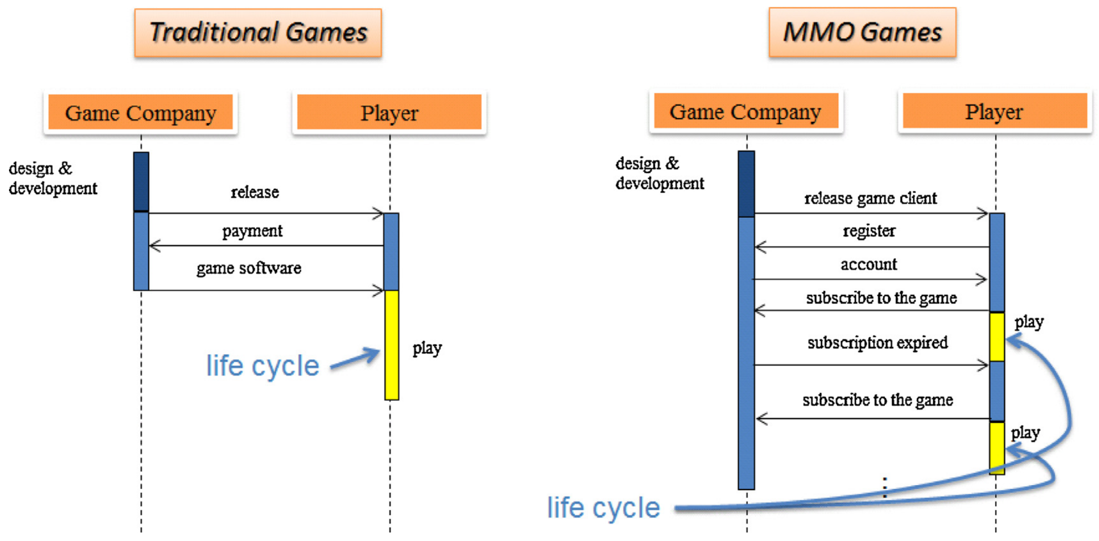


Fig. 2. Sequence diagrams of the two revenue models of the game industry.

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 indirectly weaken the others, the result of tuning operations may, somehow, become worse. Therefore, it is emergent to find a low-cost and efficient solution. In this paper, we propose a coevolutionary method to obtain a set of well-balanced AIFs. The method includes integration with the modified probabilistic incremental program evolution (PIPE) and the cooperative coevolutionary algorithm (CCEA). The merit of the evolutionary algorithms is that we simply need to define a proper fitness function to reach the near-optimal or optimal solution, which will liberate the development team from the endless testing, and eventually improve the work efficiency.

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 design method as well as the simplest turn-based MMORPG 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) [6] 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 [6] and popularized by Goldberg [7]. A schema, $H=[*11*0**]$, is a template for the 7-length individuals in which 2*nd* and 3*rd* 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,*

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

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) [8–10] 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.

Fig. 4 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

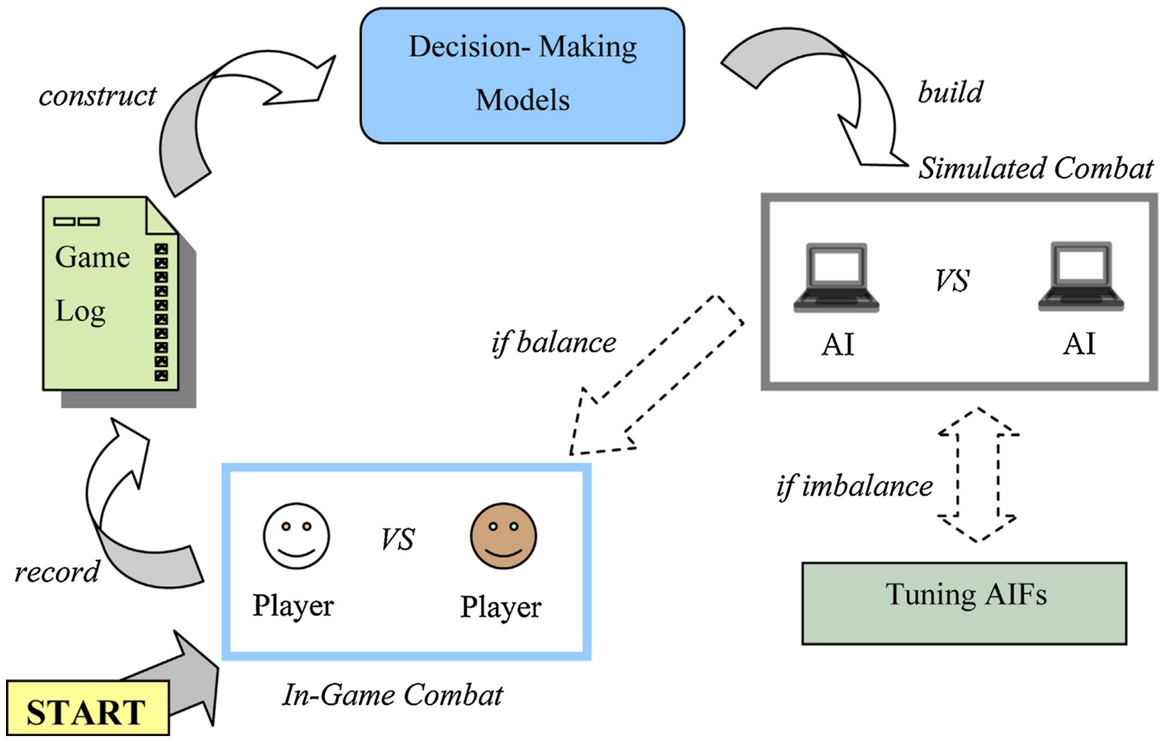


Fig. 3. Working process of the automated testing framework.

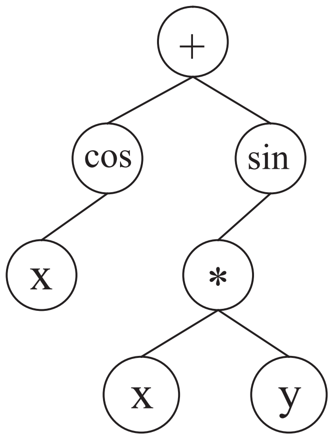


Fig. 4. The tree structure of $\cos x + \sin(x * y)$.

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 [11,12] have been applied to solve large and complex problems, multiagent systems [13–15], rule learning [16,17], fuzzy modeling [18], and inventory control systems [19]. 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 the Simple GA (SGA) [20–22]. 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.
(3) There are no variational operators in both the SGA and the CCEA.

Let us begin with some definitions.

Definition 1. Given schemata: $H_1, H_2, \ldots, H_n$, where H_i denotes a schema of the ith 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 ith CCEA population, the expanded genotype of I is the best complete genotype in which the ith representative is replaced by I.

Definition 4. Given a target problem f, the two algorithms, the SGA and the CCEA, are strictly comparable if the population of the SGA 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 SGA. If the two algorithms are strictly comparable, then,*

(1) $r_{SGA} < \sum_{i=1}^{n} r_i$
(2) $r_{CCEA} = k \cdot (\min\{r_1, r_2, \ldots, r_n\})^n$, $k \geq 1$

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

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

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

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

where $\overline{f(H_i, t)}$ and $\overline{F(i, t)}$ denote the mean fitness of individuals matching H_i and the mean fitness of all individuals, at generation t in the ith CCEA population, 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:

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

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

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

Then,

$$E[M(H_1^n, t+1)] = \prod_{i=1}^{n} E[M(H_i, t+1)] = \prod_{i=1}^{n} M(H_i, t) \cdot r_i = M(H_1^n, t) \cdot k \cdot (\min\{r_1, r_2, \ldots, r_n\})^n$$

where $k = \prod_{i=1}^{n} \frac{r_i}{\min\{r_1, r_2, \ldots, r_n\}} \geq 1$. This completes the proof. □

Therefore, if $\min\{r_1, r_2, \ldots, r_n\} \gg 1$, with the n increasing, H_1^n will receive a much higher increasing rate in the CCEA. However, Theorem 2 does not mean that the CCEA is superior to the SGA, 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 CCEA [23,24]. It has been proved that even with prefect information, infinite population size and no variational operators, the CCEA can be expected to converge to suboptimal solution [24], while the SGA does not suffer from such affliction [25–27]. However, Liviu [28] also has emphasized that the CCEA 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 [29] 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 [30]. It has been shown that PIPE achieved better results than traditional GP in solving function regression problem and 6-bit parity problem [29].

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.

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

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:

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

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

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

First $P(P_{ROG_b})$ is computed by looking at all PPT nodes N_j used to generate P_{ROG_b}:

$$P(P_{ROG_b}) = \prod_{j:N_j \text{ used to generate } P_{ROG_b}} P_j(I_j(P_{ROG_b})) \tag{3}$$

where $I_j(P_{ROG_b})$ denotes the instruction of program P_{ROG_b} at node position j.

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

$$P_{TARGET} = P(P_{ROG_b}) + (1 - P(P_{ROG_b})) \cdot lr \cdot \frac{\varepsilon + FIT(P_{ROG^{el}})}{\varepsilon + FIT(P_{ROG_b})} \tag{4}$$

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

```
(1) REPEAT
(2)    $P_j(I_j(P_{ROG_b})) = P_j(I_j(P_{ROG_b})) + c^{lr} \cdot lr \cdot (1 - P_j(I_j(P_{ROG_b})))$
(3) UNTIL $P(P_{ROG_b}) \geq P_{TARGET}$
```

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

(d) 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}:

$$P_{M_p} = \frac{P_M}{z \cdot \sqrt{|P_{ROG_b}|}} \tag{5}$$

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

$$P_j(I) = P_j(I) + mr \cdot (1 - P_j(I)) \tag{6}$$

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

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

(e) 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 (P_{ROG_b}) is modified to adapt_PPT_towards ($P_{ROG^{el}}$), 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 unconditionally. The EPS is a useful technic which guarantees that the best chromosome generated will not be lost from one generation to the next as a consequence of sampling effects or the application of genetic operators. In some case, it has been shown that the EPS can significantly improve the performance of evolutionary algorithms [31–33]. However, the EPS biases the distributor of trails in favor of those subspaces which have produced the best-performing individual, namely, it suggests that the effect of EPS on performance will improve local search at the expense of global search. Therefore, to avoid such premature problem, when applying the EPS, we make some modifications to the learning mechanism of the PIPE. Referring Formula (4), in the original PIPE, the learning rate is controlled by the following fraction dynamically:

$$v = \frac{\varepsilon + FIT(P_{ROG_{el}})}{\varepsilon + FIT(P_{ROG_b})} \tag{8}$$

Here, we construct a new fraction v' as follow:

$$v' = \begin{cases} \exp\left(-\dfrac{\left|FIT(P_{ROG_b}) - SF\right|}{K}\right) & FIT(P_{ROG_b}) > SF \\ 1 & FIT(P_{ROG_b}) \le SF \end{cases} \tag{9}$$

where SF denotes the satisfactory fitness, K is a positive user-defined constant which used to control the pressure of learning. The new fraction is able to control the learning rate according to the distance between $FIT(P_{ROG_b})$ and SF. It is important to note that once PIPE employs the EPS, the EL will no longer exist.

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 nearly 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. In the proposed design method, each of the AIFs is assigned to a population in which the individuals are created by the modified PIPE. All those populations are evolved by the CCEA. Fig. 5 shows the flow chart of the method.

For an arbitrary “1 versus 1” combat group i, we define the residual r_{ij} as $(p_{ij} - 0.5)$, where p_{ij} is the win probability of combatant 1 of group i at level j. To be in balance, both the mean the r_{ij} and the mean of $|r_{ij}|$ should be close to 0. Thus we build the single aggregate objective function as follows:

$$FIT(P_{ROG}) = w_1 e + w_2 v + w_3 f_{rule}(S)$$

where

$$e = \frac{1}{m*L}\sum_{i=1}^{m}\left|\sum_{j=1}^{L} r_{ij}\right| \qquad v = \frac{1}{m*L}\sum_{i=1}^{m}\sum_{j=1}^{L}|r_{ij}| \tag{10}$$

where w_1, w_2 and w_3 are weighting parameters, $m = \binom{n}{2}$ denotes the number of “1 versus 1” groups, n is the number of races, L is the maximum level, $f_{rule}(S) \ge 0$ denotes how many times the solution S disobeyed the designed rules of AIF. The better individuals will return lower values in evaluation process.

In order to test the proposed method, we construct a simplest turn-based MMORPG 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. So, we have three combat groups, ($R1$, $R2$), ($R2$, $R3$) and ($R3$, $R1$).
(2) Each race has only one skill and two attributes which are health and dodge rate. For simplicity, we let the dodge rate as a constant.
(3) Each skill includes only one ability, the physical damage.

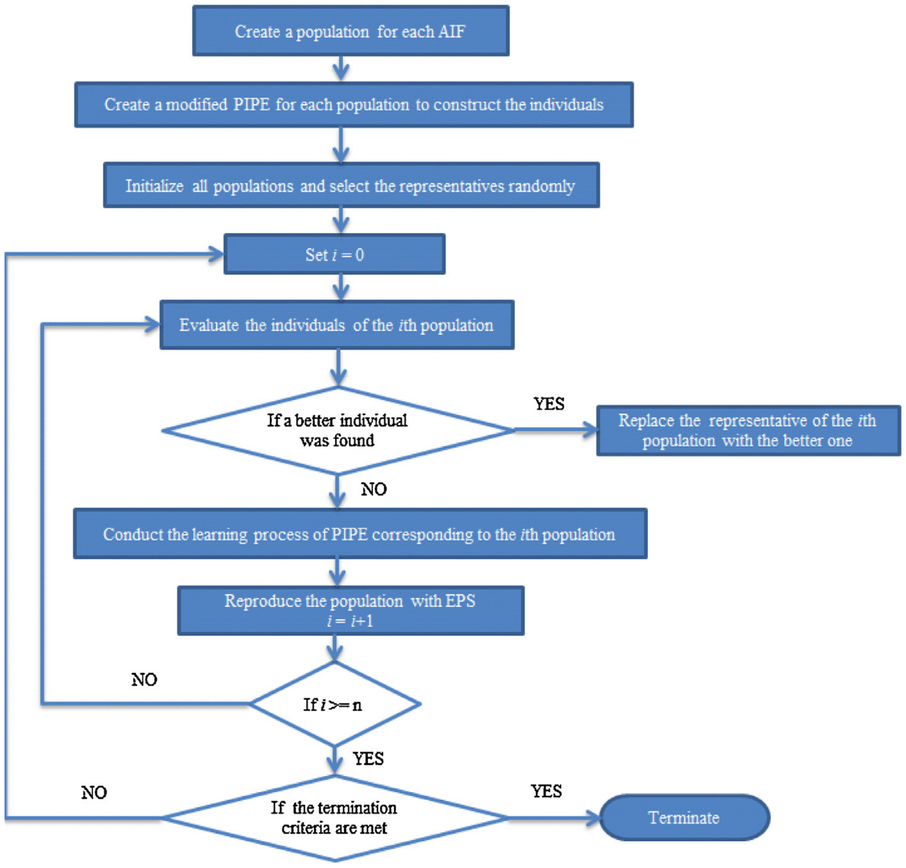

Create a population for each AIF
Create a modified PIPE for each population to construct the individuals
Initialize all populations and select the representatives randomly
Set i = 0
Evaluate the individuals of the ith population
If a better individual was found
YES
Replace the representative of the ith population with the better one
NO
Conduct the learning process of PIPE corresponding to the ith population
Reproduce the population with EPS i = i+1
NO
If i>=n
YES
NO
If the termination criteria are met
YES
Terminate

Fig. 5. The flow chart of the proposed design method.

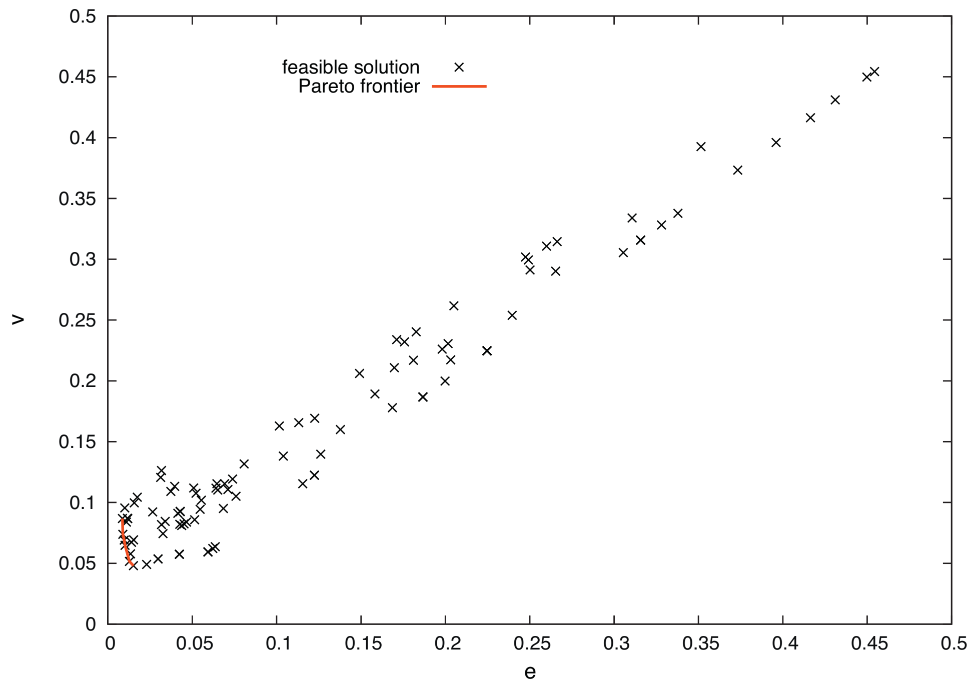


Fig. 6. The (e, v) values of the potential optimal solutions obtained by the algorithms.

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

$$p_{ij} = \sum_{g=1}^{T_N} \frac{1}{2} \cdot p_i(F_g) \left[p_i({F'}_{g-1}) + p_i({F'}_g)\right] \tag{11}$$

where

$$p_i(F_g) = \begin{cases} \begin{pmatrix} g-1 \\ LT_2(i)-1 \end{pmatrix} (1-D_2)^{LT_2(i)} D_2^{i-LT_2(i)} & g \geq LT_2(i) \\ 0 & g < LT_2(i) \end{cases}$$

$$p_i(F'_g) = \begin{cases} \sum_{k=0}^{LT_1(i)-1} \begin{pmatrix} g \\ k \end{pmatrix} (1-D_1)^k D_1^{g-k} & g \geq LT_1(i) \\ 1 & g < LT_1(i) \end{cases}$$

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

Here, we suppose that the combat ends within T_N rounds, $p_i(F_g)$ denotes the probability that combatant 1 launch the lethal attack at round g, $p_i({F'}_g)$ is the probability that combatant 2 has not launched the lethal attack before round $g+1$, D_l, LT_l, h_l, d_l refer to the dodge rate, lifetime, health value and damage value of combatant l, respectively, where $l = \{1, 2\}$.

6. Experiments and results

In this section, we conduct a series of experiments to verify the performance of the proposed algorithm. For a comprehensive comparison, beside the SGA, the simulated annealing algorithm (SAA) [34] as well as the hybrid discrete particle swarm optimization (HDPSO) algorithm [35] are also introduced.

The SAA is a computational stochastic technique for obtaining near global optimum solutions to combinatorial and function optimization problems. It is inspired from the thermodynamic process of cooling (annealing) of molten metals to attain the lowest free energy [34]. When molten metal is cooled slowly enough, it tends to solidify in a structure of minimum energy. Starting with an initial solution and armed with adequate perturbation and evaluation functions, the algorithm performs a stochastic partial search of the state space. Uphill moves are occasionally accepted with a probability controlled by a parameter called temperature. The probability of acceptance of uphill moves decreases as the temperature decreases. At high temperature, the search is almost random, while at low temperature the search becomes almost greedy, which can eventually help locate the neighborhood to the true (global) minimum. We will use the same SAA described in [34].

Particle swarm optimization (PSO), originally by Eberhart and Kennedy in 1995 [36], is a heuristic global optimization method and also an optimization algorithm. It comes from the research on the bird and fish flock movement behavior. The algorithm is widely used and rapidly developed for its easy implementation and few particles required to be tuned. However, the formulation of the original PSO requires the search space to be continuous and the individuals to be represented as vectors of real numbers, which becomes the major obstacle of applying a PSO algorithm to combinatorial problems. Fortunately, some variants have been proposed in recent literatures to remedy the drawback [35,37–39]. Here we will employ the HDPSO [35], one of the latest variants, for comparisons. In the HDPSO, particle will be updated by the following four steps at each generation.

(1) Applying the mutation operator to the current particle with the probability of w
(2) Applying the crossover operator to the result of step 1 and the current personal best solution with the probability of c_1
(3) Applying the crossover operator to the result of step 2 and the current global best solution with the probability of c_2
(4) Applying a local search method with simulation annealing type of a constant temperature T_{em} to the global best solution.

In the experiments, the environments are set as follows.
The Game Model:

(1) The maximum level: $L = 10$
(2) The dodge rate: $D_{R1} = 0.2$, $D_{R2} = 0.6$, $D_{R3} = 0.1$

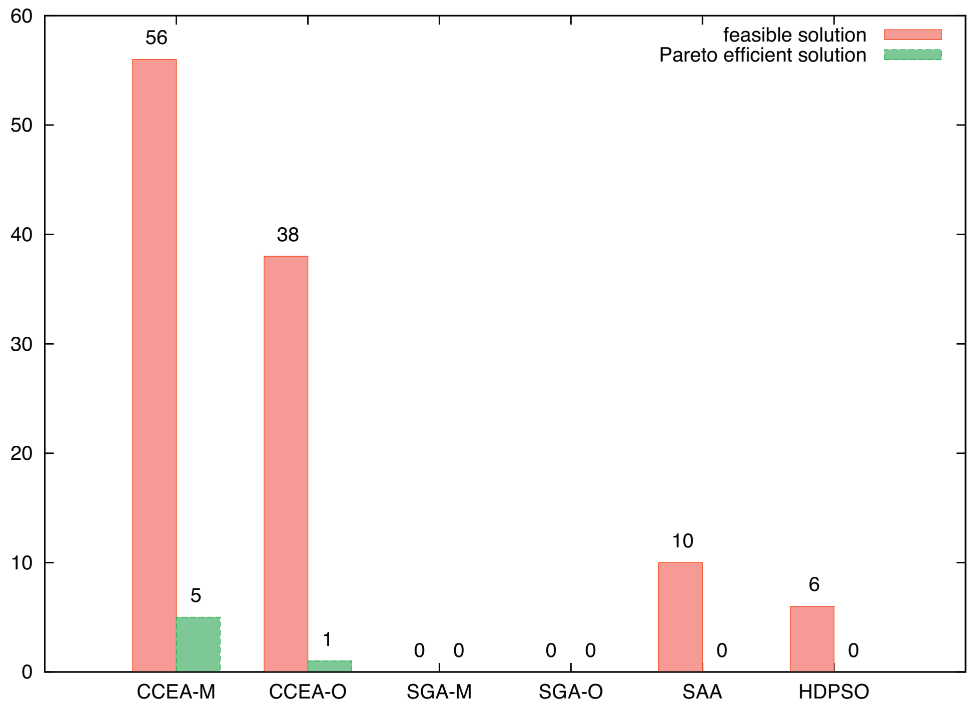


Fig. 7. The number of the potential optimal solutions and the Pareto efficient solutions obtained by each algorithm.

(3) Rules of health value: $h_{R3}(l)>h_{R1}(l)>h_{R2}(l)$
(4) Rules of damage value: $d_{R2}(l)>d_{R1}(l)>d_{R3}(l)$
(5) All the AIFs must be monotone increasing

Parameters of The Fitness Function:

(1) $w_1=1, w_2=2, w_3=100, T_N=30$

The Common Termination Criteria:

(1) The maximal number of evaluations = 120,000

Parameters of The Modified and The Original PIPE:

(1) Function and terminal sets have been used: $F=\{+, -, *, \sin, \cos, \exp\}$ and $T=\{R, l\}$, $R\in[0, 10]$ denotes the generic random constant
(2) $\varepsilon=0.00001$, $lr=0.1$, $c^{lr}=0.1$, $P_M=0.4$, $P_{el}=0.01$, $mr=0.3$, $T_P=0.999999$, $SF=0.03$, $K=0.12$
To decide the value of K, we define that, if $\left|FIT(P_{ROG_b})-SF\right|=0.27$, then $\nu'=0.1$. Thus we have $K=0.12$.

Parameters of The CCEA:

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

Parameters of The SGA:

(1) population size = 600
Since the generations and the number of evaluations in each generation are crucial to the performance of evolutionary algorithms, we set the population size of the SGA to 600 for a fair comparison.

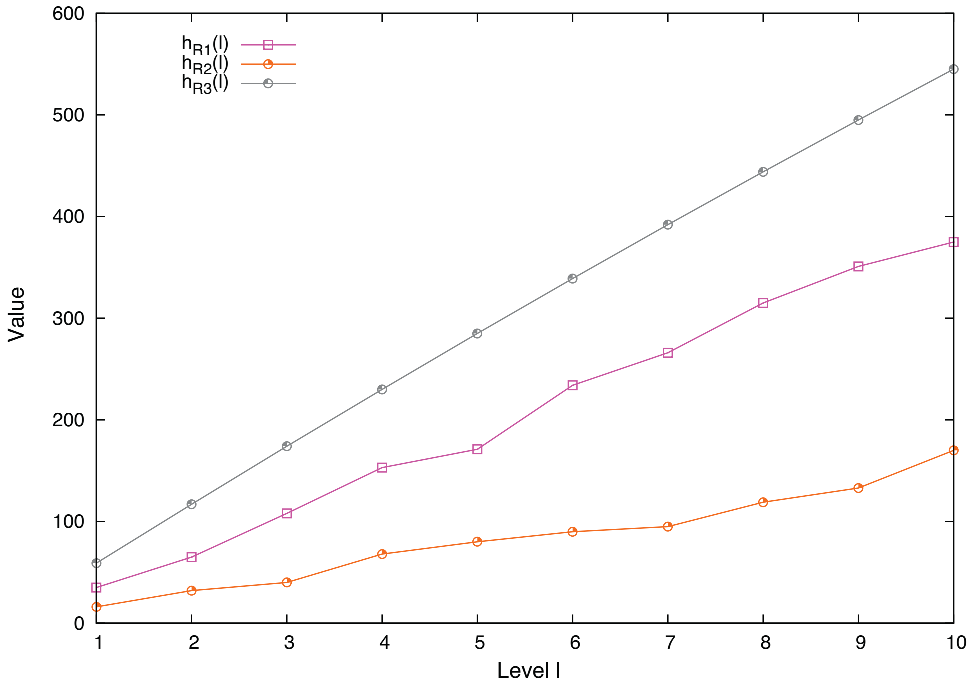
Parameters of The SAA:

(1) The initial temperature T_0
In the objective function, since the w_3 is set to a relative large number, the value of $f_{rule}(S)$ should be 0 for any feasible solution. Therefore, we obtain $\left|\Delta FIT(P_{ROG_f})\right|\le 1.5$, where P_{ROG_f} is a part of a feasible solution. At the beginning, the acceptance probability of the uphill moves should be close to 1(we choose 0.9 in this paper). So according to the Metropolis criterion, we have $T_0=15$.
(2) Equilibrium criteria
   (a) The maximal number of the iterations through the inner loop:
   $LP_{max}=600$
   (b) The maximal number of the accepted moves (either uphill or downhill) through the inner loop: $AM_{max}=60$
(3) Choosing the neighbors
We generate the neighbors by using 6 original PIPEs. It is because that the SAA allows the PIPEs to have only one individual in the population, where the EPS will become meaningless. If the generated neighbor is accepted, all the PIPEs will learn from it, otherwise, they will learn from the current solution.
(4) Reducing the temperature
We use the geometric schedule to reduce the temperature periodically, that is, $T_t=(0.9)^t T_0$, where t is the current stage number.

Parameters of The HDPSO:

(1) The number of particles is set to 20
Each particle is evolved by 6 original PIPEs. At each generation, the PPTs of those PIPEs firstly will be mutated with probability w, and then learn from the current personal best solution and the current global best solution, with the probability c_1 and c_2, respectively.
(2) $w=0.9$, $c_1=c_2=0.8$
(3) $T_{em}=15$

We denote the CCEA with the modified PIPE, the proposed one, as CCEA-M, and the CCEA with the original PIPE as CCEA-O. The



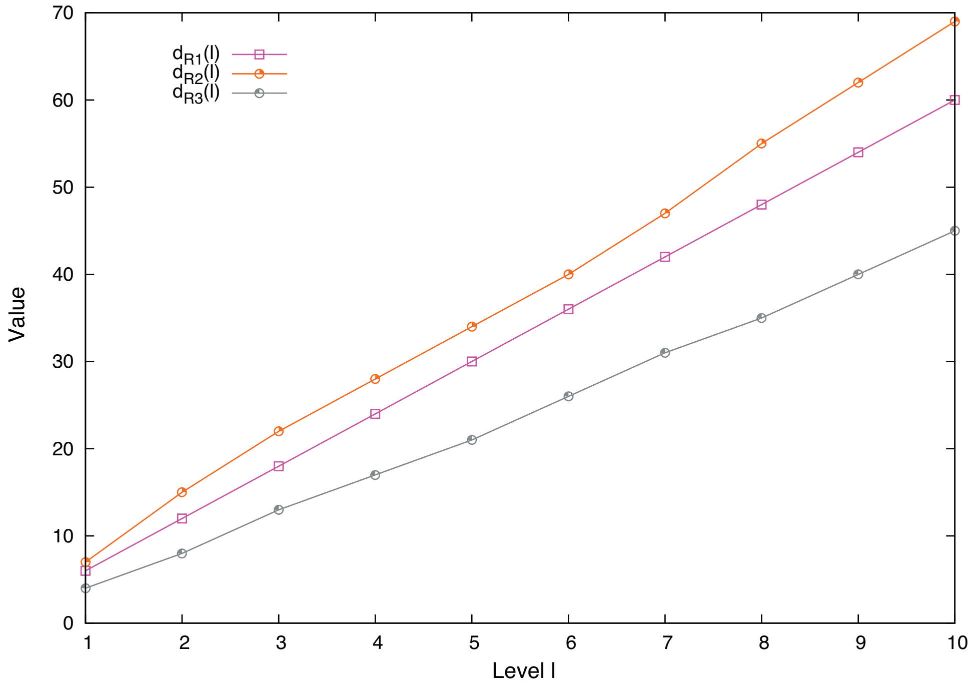


Fig. 8. The AIFs of the best solution.

same rules are applied to SGA. Thus we have six different algorithms which are CCEA-M, CCEA-O, SGA-M, SGA-O, SAA and HDPSO. Considering the stochastic nature of such algorithms, each of the algorithms is repeated 100 times and then 600 solutions will be obtained. In order to compare the computational speed, for each algorithm, we record the average time, spent for obtaining the final solution, over 100 runs, and then we use the Average Computation Time Ratio (ACTR), the ratio of the algorithm's average time to the CCEA-M's average time, as the measurement for comparison. Hence, the effect of hardware on computation time vanished.

As we described above, in the single aggregate objective function (refer to Formula (10)), the third objective, $f_{rule}(S)$, should be 0 in any feasible solution. We exclude the solutions with $f_{rule}(S) \neq 0$ obtained by the algorithms, and plot the (e, v) values of the rest solutions in Fig. 6, where the horizontal coordinate denotes the e value and the vertical coordinate denotes the v value. The red line shows the Pareto frontier of the two objectives. The solutions on that line are regraded as Pareto efficient. Actually, each Pareto efficient solution can be used as optimal, and the Formula (10), depending on the design purpose, simply provides a way to decide which Pareto efficient solution is better. Fig. 7 presents the number of the feasible solutions and the Pareto efficient solutions obtained by each algorithm. As we can see, over 85 percent of the feasible solutions and all the Pareto efficient solutions were obtained by

Table 1
The smallest fitness, the lower quartile, the median and the ACTR of each algorithm.

Algorithm	Smallest	Lower quartile	Median	ACTR
CCEA-M	0.111197	0.210727	0.650961	1
CCEA-O	0.136838	0.561036	2050.735716	1.303456
SGA-M	500.810954	2001.326758	3000.960972	1.409746
SGA-O	4001.102408	7001.192032	8001.063090	1.844213
SAA	0.947184	600.648115	1100.849107	2.197854
HDPSO	0.747449	201.150587	400.848935	2.585349

CCEA-M and CCEA-O. Thus, we can infer that CCEA is much more likely to obtain the optimal solutions than SGA and SAA as well as HDPSO. Moreover, Table 1 shows the comparison results for the smallest fitness, the lower quartile, the median and the ACTR. According to those data, we can further obtain the following four inferences:

(1) The quality of the solutions obtained by CCEA and SGA has been significantly improved with the modified PIPE.
(2) With the modified PIPE, CCEA and SGA can obtain the final solution faster.
(3) CCEA-M has obtained the best solution (fitness value is 0.111197).
(4) CCEA-M outperforms all the other algorithms.

Based on those exploratory inferences, we make the following six null hypothesis(H_0) and alternative hypotheses(H_1) pairs:

(1) H_0: CCEA-M outperforms CCEA-O; H_1: CCEA-M is not better than CCEA-O
(2) H_0: SGA-M outperforms SGA-O; H_1: SGA-M is not better than SGA-O
(3) H_0: CCEA-M outperforms SGA-M; H_1: CCEA-M is not better than SGA-M
(4) H_0: CCEA-M outperforms SAA; H_1: CCEA-M is not better than SAA
(5) H_0: CCEA-M outperforms HDPSO; H_1: CCEA-M is not better than HDPSO
(6) H_0: HDPSO outperforms SAA; H_1: HDPSO is not better than SAA

Then statistical significance is verified via sign tests which make very few assumptions about the nature of the distributions. Each such test is performed at a 5% significance level, so the p-value is 42, the lower limit of the left-tailed rejection region. According to the fitness values of the obtained solutions, we have that:

(1) CCEA-M obtained 64, 78, 67 and 65 better fitness values, compared with CCEA-O, SGA-M, SAA and HDPSO, respectively
(2) SGA-M obtained 100 better results, compared with SGA-O
(3) HDPSO obtained 79 better results, compared with SAA

Hence, none of the null hypotheses can be rejected.

Table 2
The value of p_{ij} with the best solution.

Level j	$i=1$ ($R1$ VS $R2$)	$i=2$ ($R2$ VS $R3$)	$i=3$ ($R3$ VS $R1$)
1	0.410674	0.439829	0.494629
2	0.410674	0.555165	0.494629
3	0.410674	0.555165	0.494629
4	0.559848	0.439829	0.494629
5	0.559848	0.439829	0.494629
6	0.559848	0.439829	0.494629
7	0.559848	0.439829	0.494629
8	0.559848	0.439829	0.494629
9	0.559848	0.555165	0.494629
10	0.559848	0.555165	0.494629

Fig. 8 depicts the AIFs corresponding to the best solution obtained by CCEA-M. It is important to note that all the designed rules are well obeyed. To verify whether the AIFs are well-balanced, according to Formula (11), we calculate p_{ij} and show the results in Table 2. The results indicate that the p_{ij} is close to 0.5, for $\forall i \in \{1, 2, 3\}$ and $\forall j \in \{1, 2, \ldots, 10\}$. Therefore, the AIFs can be accepted as well-balanced.

7. Conclusion and future work

This paper presented a co-evolutionary design method, containing integration with the CCEA and the modified PIPE, to solve the balance problem of the MMORPGs. We demonstrated the theoretical evidences of why the CCEA is usually faster than the SGA. Moreover, in order to enable the EPS option, we modified the learning mechanism of the PIPE. After a series of experiments, we have shown that the proposed method is the most efficient way to obtain a set of well-balanced AIFs, and the performance of PIPE has been significantly improved by the modification works.

The game model presented in this paper 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 more complicated than the turn-based case. In such game model, since there are no formulas for calculating the win probability, we have to employ the Monte Carlo method and that is 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 much more quickly.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.asoc.2014.01.011.

References

[1] E. Adams, Fundamentals of Game Design, 2nd ed., New Readers, Indiana, 2009.
[2] S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, New York, 2009.
[3] A. Rollings, D. Morris, Game Architecture and Design: A New Edition, New Readers, Indiana, 2004.
[4] R. Leigh, J. Schonfeld, S. Louis, Using coevolution to understand and validate game balance in continuous games, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, ACM, 2008, pp. 1563–1570.
[5] J.M. Olsen, Game balance and AI using payoff matrices, in: B.S. Jones, T. Alexander (Eds.), Massively Multiplayer Game Development, Charles River Media Inc., Hingham, MA, 2002, pp. 38–48.
[6] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, MIT Press, Cambridge, 1975.
[7] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, California, 1989.
[8] N.L. Cramer, A representation for the adaptive generation of simple sequential programs, in: Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985, pp. 183–187.
[9] J.R. Koza, Genetic Programming – On the Programming of Computers by Means of Natural Selection, MIT Press; Cambridge, 1992.
[10] R. Poli, W.B. Langdon, N.F. Mcphee, A Field Guide to Genetic Programming, Lulu, Morrisville, 2008.
[11] P. Husbands, F. Mill, Simulated coevolution as the mechanism for emergent planning and scheduling, in: R. Belew, L. Booker (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 264–270.
[12] M. Potter, The design and analysis of a computational model of cooperative coevolution (Ph.D. thesis), George Mason University, Fairfax, VA, 1997.
[13] L. Bull, T.C. Fogarty, Evolving cooperative communicating classifier systems, in: A.V. Sebald, L.J. Fogel (Eds.), Proceedings of the Third Annual Conference on Evolutionary Programming, 1994, pp. 308–315.
[14] L. Bull, T.C. Fogarty, M. Snaith, Evolution in multi-agent systems: Evolving communicating classifier systems for gait in a quadrupedal robot, in: L. Eshelman

(Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA), 1995, pp. 382–388.
[15] M. Potter, L. Meeden, A. Schultz, Heterogeneity in the coevolved behaviors of mobile robots: the emergence of specialists, in: Proceedings of the Seventeenth International Conference on Artificial Intelligence, 2001, pp. 1337–1343.
[16] M. Potter, K. De Jong, J. Grefenstette, A coevolutionary approach to learning sequential decision rules, in: Proceedings of the Sixth International Conference on Genetic Algorithms, 1995, pp. 366–372.
[17] M. Potter, K. De Jong, The coevolution of antibodies for concept learning, in: A.E. Eiben, et al. (Eds.), Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature, 1998, pp. 530–539.
[18] A. Carlos, S. Moshe, Fuzzy CoCo: a cooperative-coevolutionary approach to fuzzy modeling, IEEE Transactions on Fuzzy Systems 5 (9) (2001) 727–737.
[19] R. Eriksson, B. Olsson, Cooperative coevolution in inventory control optimisation, in: G. Smith, N. Steele, R. Albrecht (Eds.), Proceedings of the Third International Conference on Artificial Neural Networks and Genetic Algorithms, 1997, pp. 583–587.
[20] M. Potter, K. De Jong, A cooperative coevolutionary approach to function optimization, in: Proceedings of the Third International Conference on Parallel Problem Solving from Nature, 1994, pp. 249–257.
[21] M. Potter, K. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (1) (2000) 1–29.
[22] R.P. Wiegand, W. Liles, K. De Jong, An empirical analysis of collaboration methods in cooperative coevolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2001, pp. 1235–1242.
[23] T. Jansen, R.P. Wiegand, Exploring the explorative advantage of the CC (1+1) EA, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2003, pp. 310–321.
[24] R.P. Wiegand, An analysis of cooperative coevolutionary algorithms (Ph.D. thesis), George Mason University, Fairfax, VA, 2004.
[25] C. Reeves, J. Rowe, Genetic Algorithms Principles and Perspectives: A Guide to GA Theory, Kluwer, Norwell, 2002.
[26] L. Schmitt, Theory of genetic algorithms, Theoretical Computer Science 259 (2001) 1–61.
[27] M. Vose, The Simple Genetic Algorithm, MIT Press, Cambridge, 1999.
[28] P. Liviu, Theoretical convergence guarantees for cooperative coevolutionary algorithms, Evolution Computation 4 (18) (2010) 581–615.
[29] R.P. Salustowicz, J. Schmidhuber, Probabilistic incremental program evolution, Evolutionary Computation 2 (5) (1997) 123–141.
[30] 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.
[31] K. De Jong, An analysis of the behavior of a class of genetic adaptive systems (Ph.D. thesis), University of Michigan, 1975.
[32] A. Fujino, T. Tobita, K. Segawa, K. Yoneda, A. Togawa, An elevator group control system with floor-attribute control method and system optimization using genetic algorithms, IEEE Transactions on Industrial Electronics 4 (44) (1997) 546–552.
[33] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, Evolutionary Computation 2 (6) (2002) 182–197.
[34] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 20 (4598) (1983) 671–680.
[35] Q.K. Pan, L. Wang, M.F. Tasgetiren, B.H. Zhao, A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion, International Journal of Advanced Manufacturing Technology 38 (2008) 337–347.
[36] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995, pp. 39–43.
[37] Q.K. Pan, M.F. Tasgetiren, Y.C. Liang, A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem, Computers & Operations Research 35 (2008) 2839–2907.
[38] C.J. Liao, C.T. Tseng, P. Luran, A discrete version of particle swarm optimization for flowshop scheduling problems, Computers & Operations Research 34 (2007) 3099–3111.
[39] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, Proceedings of IEEE International Conference on Systems, Man and Cybernetics 5 (1997) 4104–4108.