Searching Optimal Combat Strategies of On-line Action Role-playing Game using Discrete Competitive Markov Decision Process

Chen Haoyang[†1] Yasukuni Mori[†1] Ikuo Matsuba[†1]

In the case of on-line Action Role-playing Game, the combat strategies can be divided into three distinct classes, Strategy of Motion(SM), Strategy of Attacking Occasion(SAO) and Strategy of Using Skill(SUS). In this paper, we analyze such strategies of a basic game model in which the combat is modeled by Discrete Competitive Markov Decision Process. By introducing the chase model and the combat assistant technology, we identify the optimal SM and the optimal SAO, successfully. Also, we propose an evolutionary framework, including integration with competitive and cooperative coevolution, to search the optimal SUS which is regarded as the Nash Equilibrium Point of the strategy space. Moreover, some experiments are made to demonstrate that the proposed framework has the ability to retrieve the optimal SUS. Furthermore, from the results, it is shown that using cooperative coevolution is much more efficient than using simple evolutionary algorithm.

1. Introduction

In recent years, on-line Action Role-playing Games (ARPGs, for short) become more and more popular all over the world. An on-line ARPG is a virtual world that consists of several distinct races. Player first creates a character of any race, then plays the game by both exploring the virtual world and fighting with others. During gaming, player has direct control over the created character. Usually, the on-line ARPG is regarded as an extension of the off-line one by the reason that it allows players to fight with each other besides AI opponent. Such new feature results in more complexity of the game balance.

What is the game balance? In the case of off-line ARPG, the game balance means the difficulty control of the game and can be reached simply by adjusting the power of AI opponent. On the other hand, however, in the on-line ARPG, game balance mainly refers to power balancing among the races. Currently, on-line ARPGs are mainly balanced by hand tuning, but this approach presents several problems. Human players are expensive in both time and resources, and even human players can not explore all strategies to find out whether a dominate one exists[1]. Moreover, since strengthening one race will definitely weaken the others, the result of tuning operations may, somehow, become worse, it is hard to control. Hence, a more theoretical and efficient design method is needed.

Chen, Mori and Matsuba[2),3)] have proposed an evolutionary design method for both turn-based and action-based on-line role-playing games. In such approach, they, successfully, constructed an automated testing framework to verify whether the game world is well-balanced. However, they just analyzed the case in which each race has only one skill. The situation of having multi-skills was ignored because they failed to retrieve the optimal Strategy of Using Skill(SUS, for short), which plays a crucial role in the automated testing framework.

In this paper, we analyze the optimal SUS of a basic action-based game model, where the combat is modeled by the Discrete Competitive Markov Decision Process(DCMDP, for short)[4]. Also, by introducing the Chase Model and the Combat Assistant Technology(CAT, for short), the optimal Strategy of Motion(SM, for short) and the optimal Strategy of Attacking Occasion(SAO, for short) are investigated. The CAT can help players to run towards the opponent automatically and attack if possible. Moreover, an evolutionary framework,

†1 Graduate School of Advanced Integration Science, Chiba University

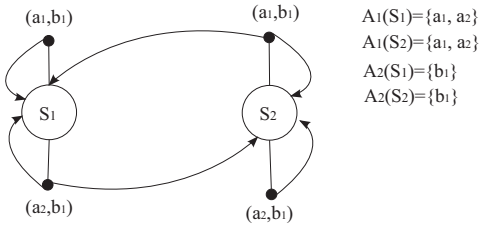

Fig. 1 A Discrete Competitive Markov Decision Process With Two States

including integration with competitive and cooperative coevolution, is proposed to search the optimal SUS for both sides of the combat. Furthermore, we make some experiments to verify the proposed framework as well as to compare the efficiency of Cooperative Coevolution Algorithm(CCEA, for short) and Simple Evolutionary Algorithm(SEA, for short).

2. Discrete Competitive Markov Decision Precess

A DCMDP is a multi-player dynamic system that evolves along discrete time points. At each time point t, the state of the system, denoted by S_t, is a random variable that can take on values from the finite set $S = 1,2,...,N$. At these discrete time points, called stages, both players have the possibility to influence the course of the system. In this paper, we only consider the case of two players, and we associate two finite action sets $A_1(s) = \{1,2,...,m_1(s)\}$ for player 1 and $A_2(s) = \{1,2,...,m_2(s)\}$ for player 2, then at any stage, the system is one of the states and both players are allowed to choose an action out of their respective action sets independently of one another. If in a state s, at some decision moment, player 1 chooses $a^1 \in A_1(s)$ and player 2 chooses $a^2 \in A_2(s)$, then two things happen:

(1) Player 1 earns the immediate reward $r^1(s,a^1,a^2)$ and player 2 earns $r^2(s,a^1,a^2)$.

(2) The dynamic of the system is influenced. The state at the next decision moment is determined in a stochastic sense by a transition vector which is denoted as:
$p(s,a^1,a^2) = (p(1|s,a^1,a^2),...,p(N|s,a^1,a^2))$

a simple example of the DCMDP is shown in **Fig. 1**.

2.1 Classes of Strategy

Strategies for players are rules that tell them what action to choose in any situation. The choice at a certain decision moment may depend upon the history of the play up to that moment. Furthermore, as is usual in game theory, the choice of an action may occur in randomized way, that is, the players can specify a probability vector over their action spaces and next the action is the result of a chance experiment according to this probability vector.

There are three classes of strategy exist, namely, the behavior strategy, the Markov strategy, and the stationary strategy.

Behavior strategy, denoted by F_B, being the most general type of strategy, can be represented by a sequence $\pi = (f_0, f_1, f_2, ...)$ where, for each$t = 0,1,2,...$, the decision rule $f_t = (f_t(1), f_t(2), ..., f_t(N))$ specifies for each state s a probability vector $f_t(s)$ on $A(s)$ as a function of history of the game up to decision moment t.

A Markov strategy, denoted by F_M, is a behavior strategy where, for every $t = 0,1,2,...$, the decision rule f_t is completely determined by the decision moment t and the current state s_t at moment t.

A stationary strategy is a Markov strategy where, for every $t = 0,1,2,...$, the decision rule f_t is completely determined by the current state s_t at moment t. Thus, a stationary strategy can be represented by a sequence $\pi = (f,f,f,...)$, where $f = (f(1), f(2), ..., f(N))$ specifies for each state $s \in S$ a probability vector $f(s)$ on $A(s)$. We will denote such stationary strategy by F_S. If the players use $\pi^1 \in F_S^1, \pi^2 \in F_S^2$ as their strategy respectively, there exist stationary transition probabilities:

$$p(s'|s,\pi^1,\pi^2) = \mathrm{P}\{S_{t+1} = s'|S_t = s,\pi^1,\pi^2\} = \sum_{a_1}^{m_1(s)} \sum_{a_2}^{m_2(s)} p(s'|s,a_1,a_2)\pi^1(s,a_1)\pi^2(s,a_2) \quad (1)$$

for all $t = 0,1,2,...$, and Formula 1 is called Stationary Markov Transition Property.

2.2 β-Discounted Competitive Markov Decision Model

The infinite stream of rewards that results during a particular implementation of a strategy pair $(\pi^1,\pi^2) \in F_S^1 \times F_S^2$ need to be evaluated in some manner. So, the β-Discounted Competitive Markov Decision Model, denoted by Γ_β, is introduced.

We have R_t^k denoting the reward at time t to player k, as well as

$$r^k(\pi^1,\pi^2) = (r^k(1,\pi^1,\pi^2),...,r^k(N,\pi^1,\pi^2)^T$$

denoting the immediate excepted reward vector to player k corresponding to a strategy pair (π^1,π^2) mentioned above, where $k = 1,2$, and $r^k(s,\pi^1,\pi^2)$, for each $s \in S$,

can be calculated by the following formula:

$$r^k(s,\pi^1,\pi^2)=\sum_{a_1}^{m_1(s)}\sum_{a_2}^{m_2(s)} r^k(s,a_1,a_2)\pi^1(s,a_1)\pi^2(s,a_2)$$

The expected reward at stage t to player k resulting from (π^1,π^2) and an initial state s is denoted by:

$$E_{s\pi^1\pi^2}(R_t^k)=[P^t(\pi^1,\pi^2)r^k(\pi^1,\pi^2)]_s$$

where $[u]_s$ denotes the sth entry of a vector u, $P^t(\pi^1,\pi^2)$ is the t-step Transition Probability Matrix(TPM, for short). Consequently, the sth entry of the overall discounted value vector of a strategy pair (π^1,π^2) to player k will be given by:

$$v_\beta{}^k(s,\pi^1,\pi^2)=\sum_{t=0}^{\infty}\beta^t E_{s\pi^1\pi^2}(R_t{}^k) \qquad (2)$$

where $\beta\in[0,1)$.

Theorem 1[4)]

Let $(\pi^{1*},\pi^{2*})\in F_S^1\times F_S^2$ be a optimal strategy pair of Γ_β, then (π^{1*},π^{2*}) is optimal in the entire class of behavior strategies. That is:

$$v_\beta{}^1(\pi^1,\pi^{2*})\le v_\beta{}^1(\pi^{1*},\pi^{2*})$$

and

$$v_\beta{}^2(\pi^{1*},\pi^2)\le v_\beta{}^2(\pi^{1*},\pi^{2*})$$

for all $\pi^1\in F_B^1,\pi^2\in F_B^2$.

In the game theory, we also call that $(\pi^{1*},\pi^{2*})\in F_S^1\times F_S^2$ is a Nash Equilibrium Point(EP, for short) of the space $F_B^1\times F_B^2$. This theorem is vary important because it suggests that we can retrieve the EP of $v_\beta{}^k(s,\pi^1,\pi^2)$ with $(\pi^1,\pi^2)\in F_B^1\times F_B^2$ by just searching the space of $F_S^1\times F_S^2$.

2.3 Zero-Sum Γ_β

A Γ_β will be called sum-zero if

$$r^1(s,a_1,a_2)+r^2(s,a_1,a_2)=0 \qquad (3)$$

for all $s\in S,a_1\in A_1(s),a_2\in A_2(s)$. Thus we may drop the superscript k by defining:

$$r(s,a_1,a_2)=r^1(s,a_1,a_2)=-r^2(s,a_1,a_2)$$

so, a extension of this definition lead to the following:

$$v_\beta(\pi^1,\pi^2)=v_\beta{}^1(\pi^1,\pi^2)=-v_\beta{}^2(\pi^1,\pi^2)$$

for all $(\pi^1,\pi^2)\in F_B^1\times F_B^2$.

Hence, in the case of zero-sum Γ_β, the two sets of inequalities defining an EP reduce to the single set of saddle-point inequality as follow:

$$v_\beta(\pi^1,\pi^{2*})\le v_\beta(\pi^{1*},\pi^{2*})\le v_\beta(\pi^{1*},\pi^2) \qquad (4)$$

Formula 4 leads to another important theorem, that is, if $(\theta^{1*},\theta^{2*})\in F_B^1\times F_B^2$ is another pair of optimal strategies, then we have the following equation:

$$v_\beta(\theta^{1*},\theta^{2*})=v_\beta(\pi^{1*},\pi^{2*}) \qquad (5)$$

this simply means that in the case of zero-sum Γ_β, the overall discounted value vectors of all optimal strategy pair coincide and can be denoted by:

$$v_\beta=(v_\beta(1),v_\beta(2),...v_\beta(N))^T$$

Thus, we can retrieve v_β by searching the space $F_S^1\times F_S^2$ instead of $F_B^1\times F_B^2$, and the result follows from **Theorem 1** and Formula 5.

3. Cooperative Coevolution Algorithm

Cooperative Coevolution Algorithm(CCEA, for short)[5)] has been proposed to solve large and complex problems through problem decomposition. It models an ecosystem which consists of two or more species. Mating restrictions are enforced simply by evolving the species in separate populations. The species interact with one another within a shared domain model and have a cooperative relationship. The original architecture of CCEA for optimization can be summarized as follows:

(1) Problem Decomposition.
Decompose a objective vector with high dimension into smaller subcomponents that can be handled by conventional Evolution Algorithms(EAs).

(2) Subcomponents Interaction.
Combine the individual of current species with representatives from other species to form a total solution.

(3) Subcomponent Optimization.
Evolve each subcomponent separately using a certain EA.

An ideal CCEA should decompose a large problem into subcomponents in which the no-linear relationship among different subcomponents are minimal. In such case, CCEA improves the performance significantly on traditional EAs[6)]. However, when the no-linear relationship becomes strong, CCEA provide lower performance than traditional EAs[7)]. In this paper, a comparison is made between Simple Evolutionary Algorithm(SEA, for short) and CCEA to varify the capability of CCEA.

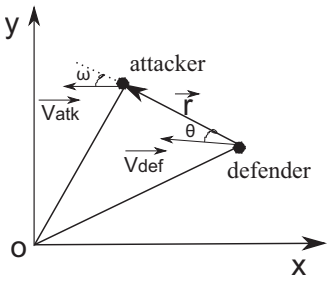

Fig. 2 The Chase Model

4. Game Model

In this paper, we construct a two-dimensional ARPG model in which three distinct races(race A, B and C) are designed with maximum level of L. Each race has seven properties, which are Health(H, for short), Dodge Rate(DR, for short), Skill Damage(SD, for short), Skill Critical Hit Rate(SCHR, for short), Skill Cool Down Time(SCDT, for short), Skill Range(SR, for short) and Velocity(V, for short), as well as two skills. Except H and SD which are monotone increasing functions of $l = \{1,2,...,L-1,L\}$, all the other properties are designed to be constant. SCHR denotes the probability that player outputs the double damage. SCDT, belonging to the time feature, means that how much time should the character rest after an attack, and SR, a kind of space feature, denotes the range in which the skill can be used. Specially, in the game world, Velocity is measured in pixels per frame instead of miles per second.

In order to evaluate the power of each race, a standard square map with width w is introduced, and all battles will be held in it. In this case, we only concern the "1 versus 1" fighting type which contains two players(the defender and the attacker), because it is the core of the combat system of on-line ARPG. As a principle, after being attacked, the defender will be able to launch at least one attack. And there is no constrain on battle time, that is , players are allowed to fight as long as they like. The basic design contents are as follows:

(1) The width of the standard map: $w = 400$

(2) Health: $H_A(l) > H_B(l) > H_C(l)$

(3) Skill Damage: $SD_A^{max}(l) > SD_B^{max}(l) > SD_C^{max}(l)$

(4) Dodge Rate: $DR_C > DR_B > DR_A$

(5) SCDT: $SCDT_C > SCDT_B > SCDT_A$

(6) Skill Range: $SR_C > SR_B > SR_A$

```
logic of attacker
if wait_time_atk <> 0
{
  wait_time_atk -= 1;
  if wait_time_atk == 0
    escape = True;
}
else
{
  if distance >= SR_atk
  {
    distance -= V_atk;
    escape = False;
  }
  else
  {
    if escape
      use optimal SM_atk;
    else
    {
      attack;
      wait_time_atk = SCDT_atk;
      escape = True;
    }
  }
}
```

```
logic of defender
if wait_time_def <> 0
  wait_time_def -= 1;
else
{
  if distance >= SR_atk
    distance -= V_def;
  else
  {
    attack;
    wait_time_def = SCDT_def;
  }
}
```

Fig. 3 The Attacking Logic of Attacker and Defender

(7) Velocity: $V_C > V_B > V_A$

where $SD_k^{max}(l)$ denotes the maximum damage among two skills of race k with level l.

In the combat system of on-line ARPG, the strategies can be divided into three types, Strategy of Using Skill(SUS, for short), Strategy of Motion(SM, for short) and Strategy of Attacking Occasion(SAO, for short). Also in the "1 versus 1" fighting type, we design the low-velocity race to play the role of defender, because if a high-velocity defender chose the strategy of keeping escaping, the battle will become meaningless and uninteresting. Based on these ideas, we model the motion of the players by using the Chase Model(refer to **Fig. 2**), and by the theory of vector differential, we acquire the following formula:

$$\frac{d|\overrightarrow{r}|}{dt} = |\overrightarrow{v_{atk}}|\cos\omega - |\overrightarrow{v_{def}}|\cos\theta \qquad (6)$$

where $\overrightarrow{r}$ is the distance vector of the two players, $\overrightarrow{v_{atk}}$ and $\overrightarrow{v_{def}}$ denote the velocity of attacker and defender.

According to Formula 6, the attacker should maximize $d|\overrightarrow{r}|$ by maximizing $\cos\omega$ and the defender should minimize $d|\overrightarrow{r}|$ by maximizing $\cos\theta$. These can be recognized as the optimal SMs of both players.

Before analyzing the optimal SAO, we introduce the Combat Assistant Technology(CAT, for short) which has been applied to most of on-line ARPGs(such as JXOnline, World of Warcraft, etc.). It helps players to run towards the opponent automatically and attack if possible, so the optimal SAO should be analyzed based on it. The pseudo-code of the attacking logic is shown in **Fig. 3**, and **Fig. 4** demonstrates the attacking process of combat.

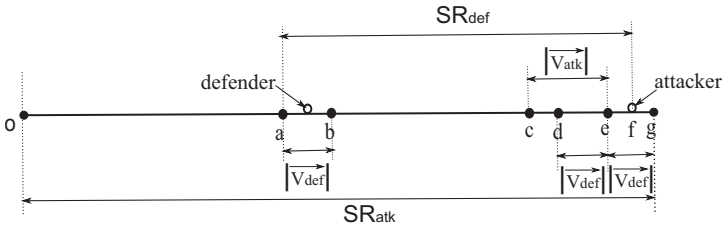

Fig. 4 Attacking Process of Using The Combat Assistant Technology

After making an attack at point f, the attacker waits at point f for $SCDT_{atk}$ frames, and the defender, by using the CAT, will strike back at a point within $(a,b]$ where he can output the maximum damage because $|ab| = |\overrightarrow{v_{def}}|$. So using the CAT to attack is the optimal SAO of the defender. On the other hand, using the CAT, the attacker could launch an attack in range $(c,d]$, $(d,e]$ and $(e,g]$ with the probability of P_1, P_2, P_3. We define (P_1,P_2,P_3) as the Action Probability Vector(APV, for short) , which represents the action feature and can be retrieved through statistical analyze. Hence, the optimal SAO of attacker is that making attack with the APV of the expert-level players.

Assuming the combat can be modeled by DCMDP, we shall use the maximum lifetime of defender(T_{def}) and attacker(T_{atk}) to form the state space

$$S = \{(0,1),(0,2),...,(T_{def},T_{atk}-1),(T_{def},T_{atk})\}$$

in which the $(0,0)$ state does not exist and T_{def},T_{atk} are functions of level l as follow:

$$T_{def}(l) = \left\lceil \frac{H_{def}(l)}{SD_{atk}^{min}(l)} \right\rceil \quad T_{atk}(l) = \left\lceil \frac{H_{atk}(l)}{SD_{def}^{min}(l)} \right\rceil \quad (7)$$

where $H_k(l)$ denotes the health value of k with level l, and $SD_k^{min}(l)$ is defined as the minimum damage among two skills of k with level l.

5. Competitive Coevolutionary Framework

In this section, first, we model the combat process by using the zero-sum Γ_β process, in this step, the fitness function will be constructed. Next, we will introduce the competitive framework as well as demonstrate how it works.

As the preceding section stated, there is no time constrain during fighting, that is, the combat can be regarded as an infinite process, where players keep changing their state from one to another by their actions. And this infinite process can be evaluated by Γ_β.

It is natural for any player that he(she) wants to maximize his(her) win probability during battles. Thus, we shall use the increment of win probability from stage t to $t+1$, as the expectation of immediate reward of the decision rule pair (f_t,g_t). Let $p^1(t),p^2(t)$ be the win probabilities of player 1 and player 2 at stage t respectively, then, by the fact that the larger t becomes, the higher probability of game over will be, we obtain the follows:

$$\sum_{t\to\infty} p^1(t) + p^2(t) = 1$$

from such formula, it can be inferred that an increment of $p^1(t)$ will bring a potential decrement to $p^2(t)$ in the future.

With these results, in this case, combat can be regarded as the zero-sum game, and can be modeled by zero-sum Γ_β process. The the immediate reward vector and the overall discounted value vector are defined as follows:

$$r(\pi^1,\pi^2) = (P^1(\pi^1,\pi^2) - I_N)Q$$
$$v_\beta(s,\pi^1,\pi^2) = \sum_{t=0}^{\infty} \beta^t \left[P^t(\pi^1,\pi^2) r(\pi^1,\pi^2)\right]_s \quad (8)$$

where

$$\pi^1 = (f,f,f,.....f,.....)$$
$$\pi^2 = (g,g,g,.....,g,.....)$$
$$Q = (q_1,q_1,...q_N)^T$$
$$q_i = \begin{cases} 1 & s_i \in S^T \\ 0 & s_i \notin S^T \end{cases} \quad (9)$$

S^T denotes the absorbing state set in which the lifetime of attacker is 0, and particularly, in this case, the $v_\beta(s,\pi^1,\pi^2)$ will always converge into a certain value, even though the discount factor, β, is set to 1.

Since the combat always starts from the state (T_{def},T_{atk}), denoted by s_N, it is natural that we shall use $v_1(s_N,\pi^1,\pi^2)$, denoting the win probability of defender, as the fitness function of our evolutionary algorithm.

The competitive framework consists of two symmetrical blocks(refer to **Fig. 5**), the left block denotes the evolutionary process of π^1 while the right block represents π^2. Both blocks use CCEA to search the solution space,

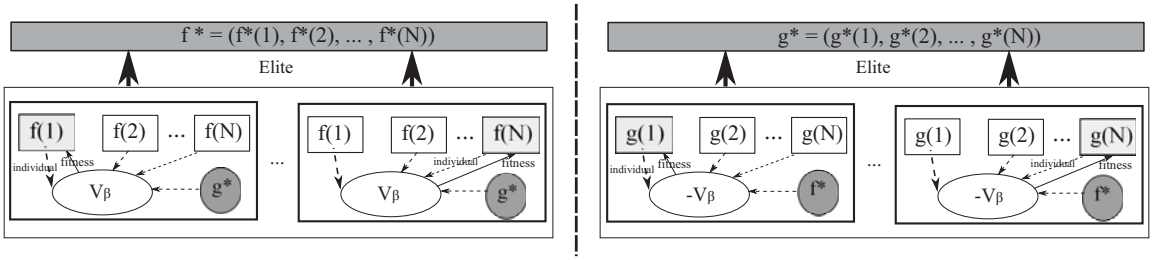

Fig. 5 The Competitive Coevolutionary Framework

that is, each decision rule, f or g, is divided into N groups which will be evaluated in turn. Such group, for example, group i, denotes the probability vector on actions of state i, that is, $f(i)$ or $g(i)$. Through the elite pair of current generation, (f^*, g^*), the two blocks of framework interact in a competitive way, and this enables the framework to lead such elite pair to converging in the decision rule pair of an EP.

The pseudo-code of the algorithm is shown as follows:

(1) Create species s_f^i for $f(i)$ and s_g^i for $g(i)$, where $i = \{1,2,...,N\}$.
(2) Initialize population for every species by using the uniform distribution, where each bit of chromosome has the same probability to be a 1.
(3) Select a individual from each $f(i)$ and $g(i)$, randomly, to form the initial elite decision rule pair (f^*, g^*).
(4) Set generation $t = 0$.
(5) Set $i = 1$.
(6) Evaluate the individuals of species s_f^i with g^* as well as $f^*(j)$, where $j \neq i$.(maximize v_β)
(7) Replace the $f^*(i)$ by the current elite, if necessary.
(8) Select the outstanding individuals, and construct the new probability distribution from them. Then new generation can be obtained by sampling this distribution and mutation.
(9) While($i <= N$) set $i = i + 1$, goto 6.
(10) Set $i = 1$,
(11) Evaluate the individuals of species s_g^i with f^* as well as $g^*(j)$, where $j \neq i$.(minimize v_β)
(12) Replace the $g^*(i)$ by the current elite, if necessary.
(13) Select the outstanding individuals, and construct the new probability distribution from them. Then new generation can be obtained by sampling this distribution and mutation.
(14) While($i <= N$) set $i = i + 1$, goto 11.

Table 1 The Attacking Times of Defender in a Round

	Range $(c,d]$	Range $(d,e]$	Range $(e,g]$
(SK_A^1, SK_C^1)	2	1	1
(SK_A^2, SK_C^1)	2	2	1
(SK_A^1, SK_C^2)	2	1	1
(SK_A^2, SK_C^2)	2	2	1

(15) If the termination criteria are not met, set $t = t + 1$, goto 5.

6. Experiments and Results

In this section, we will use the proposed competitive coevolutionary framework to retrieve the EP of a battle group. A character, from race A with level 3, is chosen as the defender, and with the same level, a character from race C as the attacker. Also, for convenience, we define SK_A^1, SK_A^2 as the two skills of race A, similarly, SK_C^1, SK_C^2 for race C. In accordance with the basic design contents mentioned before, we set the experiment environments as follows:

(1) $V_A = 10$, $V_C = 25$, $DR_A = 0.3$, $DR_C = 0.45$, $H_A(3) = 100$, $H_C(3) = 60$.
(2) Properties of SK_A^1: $\{SD(3) = 30, SR = 70, SCDT = 1, SCHR = 0.1\}$.
(3) Properties of SK_A^2: $\{SD(3) = 20, SR = 70, SCDT = 2, SCHR = 0.3\}$.
(4) Properties of SK_C^1: $\{SD(3) = 25, SR = 190, SCDT = 13, SCHR = 0.3\}$.
(5) Properties of SK_C^2: $\{SD(3) = 25, SR = 190, SCDT = 14, SCHR = 0.6\}$.
(6) The APV of attacker is $(0.2, 0.6, 0.2)$.
(7) In CCEA, population size of each species = 100, In SEA, population size = 500.
(8) Generation t = 600, Mutation Rate = 0.01.

We define the fighting round, a period of time, from attacker launching an attack to the ending of his(her) wait-

SK^1_A, SK^1_C	(0,1)	(0,2)	(0,3)	(1,0)	(2,0)	(3,0)	(4,0)	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	(4,1)	(4,2)	(4,3)
(0,1)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,2)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,3)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,0)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,0)	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,0)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
(1,1)	0.7	0	0	0.17985	0	0	0	0.12015	0	0	0	0	0	0	0	0	0	0	0
(1,2)	0	0.7	0	0.0326865	0	0	0	0.132165	0.1351485	0	0	0	0	0	0	0	0	0	0
(1,3)	0	0	0.7	0.0312015	0	0	0	0	0.14553	0.1232685	0	0	0	0	0	0	0	0	0
(2,1)	0.21	0	0	0.293755	0.17985	0	0	0.196245	0	0	0.12015	0	0	0	0	0	0	0	0
(2,2)	0	0.21	0	0.0509625	0.0312015	0	0	0.237699	0.196245	0	0.14553	0.1283621	0	0	0	0	0	0	0
(2,3)	0	0	0.21	0.02695	0.0165	0	0	0	0.237699	0.196245	0	0.14553	0.167076	0	0	0	0	0	0
(3,1)	0	0	0	0.125895	0.293755	0.17985	0	0.084105	0	0	0.196245	0	0	0.12015	0	0	0	0	0
(3,2)	0	0	0	0.0218411	0.0509625	0.0312015	0	0.101871	0.084105	0	0.237699	0.196245	0	0.14553	0.130545	0	0	0	0
(3,3)	0	0	0	0.0218411	0.0509625	0.0312015	0	0	0.101871	0.084105	0	0.237699	0.196245	0	0.14553	0.130545	0	0	0
(4,1)	0	0	0	0	0.125895	0.293755	0.17985	0	0	0	0.084105	0	0	0.196245	0	0	0.12015	0	0
(4,2)	0	0	0	0	0.0218411	0.0509625	0.0312015	0	0	0	0.101871	0.084105	0	0.237699	0.196245	0	0.14553	0.130545	0
(4,3)	0	0	0	0	0.0218411	0.0509625	0.0312015	0	0	0	0	0.101871	0.084105	0	0.237699	0.196245	0	0.14553	0.130545

SK^1_A, SK^2_C	(0,1)	(0,2)	(0,3)	(1,0)	(2,0)	(3,0)	(4,0)	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	(4,1)	(4,2)	(4,3)
(0,1)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,2)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,3)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,0)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,0)	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,0)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
(1,1)	0.7	0	0	0.17985	0	0	0	0.12015	0	0	0	0	0	0	0	0	0	0	0
(1,2)	0	0.7	0	0.0326865	0	0	0	0.132165	0.1351485	0	0	0	0	0	0	0	0	0	0
(1,3)	0	0	0.7	0.0312015	0	0	0	0	0.14553	0.1232685	0	0	0	0	0	0	0	0	0
(2,1)	0.42	0	0	0.16786	0.17985	0	0	0.11214	0	0	0.12015	0	0	0	0	0	0	0	0
(2,2)	0	0.42	0	0.0291214	0.0312015	0	0	0.135828	0.11214	0	0.14553	0.1261791	0	0	0	0	0	0	0
(2,3)	0	0	0.42	0.0154	0.0165	0	0	0	0.135828	0.11214	0	0.14553	0.154602	0	0	0	0	0	0
(3,1)	0	0	0	0.25179	0.16786	0.17985	0	0.16821	0	0	0.11214	0	0	0.12015	0	0	0	0	0
(3,2)	0	0	0	0.0436821	0.0291214	0.0312015	0	0.203742	0.16821	0	0.135828	0.11214	0	0.14553	0.130545	0	0	0	0
(3,3)	0	0	0	0.0436821	0.0291214	0.0312015	0	0	0.203742	0.16821	0	0.135828	0.11214	0	0.14553	0.130545	0	0	0
(4,1)	0	0	0	0	0.25179	0.16786	0.17985	0	0	0	0.16821	0	0	0.11214	0	0	0.12015	0	0
(4,2)	0	0	0	0	0.0436821	0.0291214	0.0312015	0	0	0	0.203742	0.16821	0	0.135828	0.11214	0	0.14553	0.130545	0
(4,3)	0	0	0	0	0.0436821	0.0291214	0.0312015	0	0	0	0	0.203742	0.16821	0	0.135828	0.11214	0	0.14553	0.130545

SK^2_A, SK^1_C	(0,1)	(0,2)	(0,3)	(1,0)	(2,0)	(3,0)	(4,0)	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	(4,1)	(4,2)	(4,3)
(0,1)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,2)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,3)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,0)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,0)	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,0)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
(1,1)	0.7	0	0	0.2244	0	0	0	0.0756	0	0	0	0	0	0	0	0	0	0	0
(1,2)	0	0.7	0	0.10428	0	0	0	0.06468	0.13104	0	0	0	0	0	0	0	0	0	0
(1,3)	0	0	0.7	0.04356	0	0	0	0.08646	0.10626	0.06372	0	0	0	0	0	0	0	0	0
(2,1)	0.21	0	0	0.35035	0.2244	0	0	0.12348	0	0	0.09177	0	0	0	0	0	0	0	0
(2,2)	0	0.21	0	0.141218	0.08646	0	0	0.173558	0.12348	0	0.10626	0.159024	0	0	0	0	0	0	0
(2,3)	0	0	0.21	0.035574	0.02178	0	0	0.11858	0.173558	0.12348	0.0825	0.10626	0.128268	0	0	0	0	0	0
(3,1)	0	0	0	0.15708	0.36652	0.2244	0	0.05292	0	0	0.12348	0	0	0.0756	0	0	0	0	0
(3,2)	0	0	0	0.0595518	0.1389542	0.085074	0	0.079002	0.05292	0	0.173558	0.12348	0	0.10626	0.1812	0	0	0	0
(3,3)	0	0	0	0.015246	0.035574	0.02178	0	0.04985	0.074382	0.05292	0.1324862	0.173558	0.12348	0.081114	0.10626	0.13335	0	0	0
(4,1)	0	0	0	0	0.15708	0.35882	0.2244	0	0	0	0.05292	0	0	0.12348	0	0	0.0833	0	0
(4,2)	0	0	0	0	0.0595518	0.1389542	0.085074	0	0	0	0.074382	0.05292	0	0.173558	0.12348	0	0.10626	0.18582	0
(4,3)	0	0	0	0	0.015246	0.035574	0.02178	0	0	0	0.0567798	0.074382	0.05292	0.132486	0.173558	0.12348	0.081114	0.10626	0.12642

SK^2_A, SK^2_C	(0,1)	(0,2)	(0,3)	(1,0)	(2,0)	(3,0)	(4,0)	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	(4,1)	(4,2)	(4,3)
(0,1)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,2)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,3)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,0)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,0)	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,0)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
(1,1)	0.7	0	0	0.2244	0	0	0	0.0756	0	0	0	0	0	0	0	0	0	0	0
(1,2)	0	0.7	0	0.10428	0	0	0	0.06468	0.13104	0	0	0	0	0	0	0	0	0	0
(1,3)	0	0	0.7	0.04356	0	0	0	0.08646	0.10626	0.06372	0	0	0	0	0	0	0	0	0
(2,1)	0.42	0	0	0.2002	0.2244	0	0	0.07056	0	0	0.08484	0	0	0	0	0	0	0	0
(2,2)	0	0.378	0	0.080696	0.08646	0	0	0.099176	0.07056	0	0.10626	0.178848	0	0	0	0	0	0	0
(2,3)	0	0	0.378	0.020328	0.02178	0	0	0.06776	0.099176	0.07056	0.0825	0.10626	0.153636	0	0	0	0	0	0
(3,1)	0	0	0	0.31416	0.20944	0.2244	0	0.10584	0	0	0.07056	0	0	0.0756	0	0	0	0	0
(3,2)	0	0	0	0.1191036	0.0794024	0.085074	0	0.158004	0.10584	0	0.099176	0.07056	0	0.10626	0.17658	0	0	0	0
(3,3)	0	0	0	0.030492	0.020328	0.02178	0	0.0997	0.148764	0.10584	0.0757064	0.099176	0.07056	0.081114	0.10626	0.14028	0	0	0
(4,1)	0	0	0	0	0.31416	0.20944	0.2244	0	0	0	0.10584	0	0	0.07056	0	0	0.0756	0	0
(4,2)	0	0	0	0	0.1191036	0.0794024	0.085074	0	0	0	0.148764	0.10584	0	0.099176	0.07056	0	0.10626	0.18582	0
(4,3)	0	0	0	0	0.030492	0.020328	0.02178	0	0	0	0.1135596	0.148764	0.10584	0.075706	0.099176	0.07056	0.081114	0.10626	0.12642

Fig. 6 The Probability Transition Matrix Defined by Skill Pair SK^i_A, SK^j_C

ing time. Such definition implies that attacker can attack only one time in a round. Also, based on the experiment environments, we obtain the attacking times of defender, when players chose a certain skill pair and attacker attacked in a certain range. The results are shown in **Table 1**.

By using these results, the probability transition matrix defined by skill pair, SK^i_A, SK^j_C where $i, j = \{1,2\}$, can be

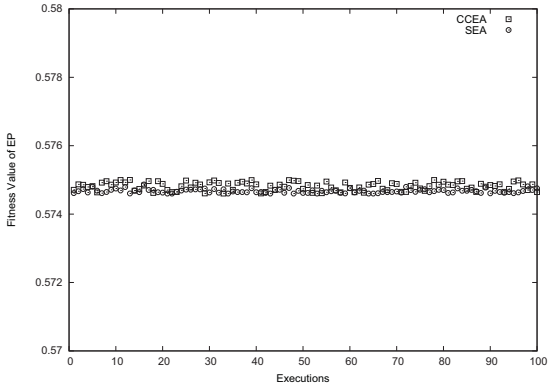

Fig. 7 The Fitness of EP Retrieved by Each Execution

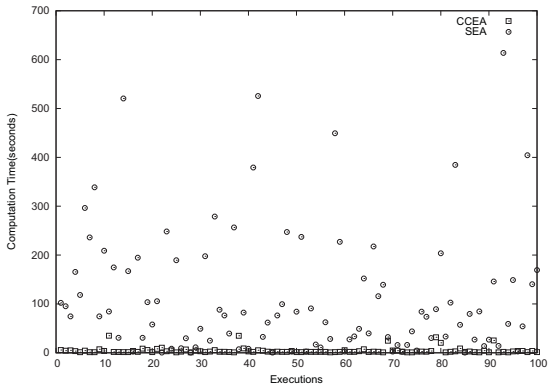

Fig. 8 The Computation Time of Retrieving EP in Each Execution

calculated(refer to **Fig. 6**), then according to Formula 1, we can retrieve the probability transition matrix defined by a stationary strategy pair (π^1, π^2). Consequently, such strategy pair can be evaluated by $v_1(s_N, \pi^1, \pi^2)$.

Besides the proposed algorithm, another one, whose blocks use SEA instead of CCEA, is introduced for comparison. Considering the stochastic nature of the algorithms, each of the programs is repeated 100 times. The results are shown in **Fig. 7** and **Fig. 8**.

According to the results in **Fig. 7**, both two algorithms can retrieve the EP, (π^{1*}, π^{2*}), and we can obtain the exception and the variance of $v_1(s_N, \pi^{1*}, \pi^{2*})$, as follows:

$$\begin{aligned} &E(v_1(s_N, \pi^{1*}, \pi^{2*})) = 0.57474595 \\ &\mathrm{Var}(v_1(s_N, \pi^{1*}, \pi^{2*})) = 1.41743336 \times 10^{-8} \end{aligned} \quad (10)$$

Such data are very important because we can judge whether the designed contents are well-balanced based on it. Also, it can be seen from **Fig. 8** that the CCEA is much faster than SEA in searching EP of the solution space, hence a better placement solution than using SEA.

7. Conclusion and Future Work

In this paper, we constructed a simple ARPG system and modeled it by using the zero-sum Γ_β process. We noted that in the case of zero-sum Γ_β, an EP of the entire strategy space can be retrieved just by searching the stationary strategy space, and all the overall discounted value vectors of the EPs are coincide. Also, an evolutionary framework, which includes integration with competitive coevolution and CCEA, has been proposed to search the EP of combat. Based on the framework, we made some experiments with certain environments. The results showed that the EP can be retrieved, and by using CCEA, the efficiency and capability of the framework have been improved significantly.

In the future work, we will investigate another combat type in which the battle time is constrained, and in such type the strategies of players will no longer be stationary.

References

1) Leigh R., Schonfeld J., and Louis S. Using coevolution to understand and validate game balance in continuous games. Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, 1563-1570, ACM, 2008.
2) Chen Haoyang, Y. Mori, and I. Matsuba. Design Method for Game Balance with Evolutionary Algorithms using Stochastic Model. Proc. International Conference on Computer Science and Engineering, Shanghai, Oct. 2011.
3) Chen Haoyang, Y. Mori, and I. Matsuba. Evolutionary Approach to The Balance Problem of On-line Action Role-playing Game. Proc. International Conference on Computational Intelligence and Software Engineering, Wuhan, Nov. 2011.
4) Filar J., K Vrieze. Competitive Markov Decision Processes, Springer-Verlag, 1996.
5) M, Potter. and K, De Jong. A Cooperative Coevolutionary Approach to Function Optimization, Proc. of the Third Conference on Parallel Problem Solving from Nature, pp.249-257, Jerusalem, Israel, 1994.
6) M, Potter. and K, De Jong. Cooperative Coevolution: An architecture for evolving coadapted subcomponents, Evolutionary Computation, 8(1):1-29, 2000.
7) R, P Wiegand. An Analysis of Cooperative Coevolutionary Algorithms, PhD thesis, George Mason University, Fairfax, Virginia, 2004.