

# Survival by Continuous Learning in a Dynamic Multiple Task Environment

Hasan Mujtaba, and A. Rauf Baig

**Abstract**—The ability to adapt or not when challenged with a dynamic, changing environment is what differentiates between survival and extinction of species. In this paper we present a machine learning method that allows agents in an environment with changing tasks to adapt and modify their behavior thus ensuring their survival. These agents do not get explicit information about the change in tasks. The learning mechanism ensures the presence of enough diversity in the agents so that they can restart learning if the previous learning stops to be effective and enough continuity in the system so that the agents can keep on learning if their task has not changed.

## I. INTRODUCTION

The ability of humans to learn is complex and multifaceted, and researchers have been able to model only some of its characteristics and that too in one-aspect-at-a-time manner. One area of research is that of learning in dynamic or changing environments. Adapting to the changes in our environment ensure our survival. It has been an age long desire of humans to create artificial entities with learning ability that can cater for the changes in their environment. The environment change can be of many forms. In our current work we deal with the problem of learning in environments where the goal or objective changes suddenly without any explicit indication being given to the learner. The learning method is itself supposed to provide a mechanism where the decrease in performance forced by the change in goals triggers new learning.

As our test bed we have chosen an environment inspired by the video game field. Intelligent agents, propelled by an artificial neural network (ANN) based controller, are expected to perform a given task. The task can, however, change abruptly without any warning. The learning is in the form of search for appropriate weights of the ANN controller which can make the agents to perform their task. The learning method provides for continuous learning and allows the agents to keep on learning and improving their task performing strategies while the task is same. It has a provision which keeps on weeding the low performing agents and replacing them with randomly initialized ones. Since the agents are still following the strategy useful for the old task hence their performance decreases after a task change. The continuous weeding out provides an elegant

and smooth way to force the agents to find new strategies following the task change.

In a larger perspective, this research has the potential to be useful in the area of dynamic function optimization. More particularly, it can contribute in the research efforts for making the video games more interesting by their automatic evolution.

Rest of the paper is organized as follow. Section II covers some related work on this topic, and Section III presents the test bed environment, the agents and the tasks assigned to them. Section IV covers our continuous learning algorithm and elaborates on how the agents learn to deal with the changes occurring in their goals, and Section V covers experimentation and results respectively. Section VI presents our conclusions and future work.

## II. RELATED WORK

In the last few years agents that “learn” to play games have come in vogue replacing the earlier agents which used brute force approach. Computational intelligence techniques, due to their inherent mimicking of human and nature observed phenomenon, have been used for developing learning agents.

One representative work in the area of ANN based controllers is the NERO game developed by Stanley *et al.* [1]. In the learning mechanism associated with NERO, we have evolution of architecture as well as weights of artificial neural network (ANN) based agents. Each agent is developed and trained to perform a specific task in a game like environment. Different agents can have different tasks and act as a team. The task or goal of each agent is fixed and remains unchanged for the entire duration of the game. We, in our work, tackle the problem of learning in an environment where tasks for an agent change without any warning.

ANN controllers are also used by Yannakakis *et al.* [2] in a simulated world called *Flatland*. They experiment with agents controlled by ANN with fixed architecture and trainable weights. One of the major contributions of their work is the mechanism of finding the fitness of agents by making them act out their strategies in the environment. We utilize the same technique for finding the fitness of our agents, but our main focus is different from theirs.

Yannakakis and Hallam have also evolved ANN controllers for the games of Pac-Man and Dead End [3]. Yet another effort in this field has been made by Lucas for the game of Cellz [4].

Kendall and Su [5] have evolved ANN based agents to deal with an environment in which all the input variables are

Hasan Mujtaba is a PhD student at the Computer Science Department, National University of Computer & Emerging Sciences, Islamabad, Pakistan

Dr. Rauf Baig is a Professor at Department of Computer Science, National University of Computer & Emerging Sciences, Sector H-11/4, Islamabad, Pakistan (e-mail: rauf.baig@nu.edu.pk).

not available at the onset and are incrementally added during learning. The agents, especially the badly performing ones have the possibility to learn new strategies with a set of variables which includes some newly introduced variables. The task of the agents remains the same, however. Their focus is learning in the presence of changing variables, whereas our focus is learning in the presence of changing tasks. Our learning mechanism is also completely different from theirs.

ANN based agents are also evolved by Messerschmidt and Engelbrecht for playing Tic-tac-toe [6] and by Franken and Engelbrecht for playing checkers [7]. The agents act as the evaluation function for the leaf nodes in minim-max search of the game tree. Their work is related to ours only by the fact that they, like us, use PSO algorithm to train the weights of the ANN. We have however introduced innovation in the PSO algorithm to cater for dynamic fitness functions. Previously, Chellapilla and Fogel [8] used a co-evolutionary approach to evolve a population of ANN for playing the game of checkers.

Fryan has used an evolutionary approach for evolving game playing strategies for Monopoly [9]. Another interesting experiment with evolutionary approach and games is presented in [10].

Our learning mechanism can be also be used in the area of dynamic function optimization. Since we have neither tested our algorithm on the benchmark dynamic functions, nor compared it with other available algorithms, hence we cannot say whether it is better or worse than others. All we can say is that the algorithm works, as indicated by the results presented in Section VII. Two recent representative swarm based algorithms for dynamic functions are [11] and [12].

Now we move on to present the environment for our experimentation in continuous learning.

### III. DYNAMIC ENVIRONMENT WITH CHANGING TASKS

We have selected the environment called *Flatland* [2] as the test bed for our experimentation on learning in the presence of changing tasks. It is a multi-agent environment in which agent can interact with each other and with other artifacts and in which agent learning can be observed. However, the original *Flatland* is not sufficient for our experimentation and we had to modify it by including more tasks and artifacts and by increasing the inputs of our learning agents. To avoid confusion with the original *Flatland* we are calling our version *Flatland II*.

*Flatland II* is a spherical (the original *Flatland* is square) 80 x 80 pixel, 2-D, environment in which a number of learning agents can be present along with non-learning artifacts. The agents can interact with one another and with other artifacts, and the agents have some specific task to be achieved for which they have to develop (learn) their strategies. The task of the agents can change from time to time without any explicit warning or information given to the agents. The change in task is manifested by the changed fitness function. The only information available to the agent

is in the form of feedback from the fitness function. After a task change, the fitness of the agent usually decreases because it is still pursuing the old (and now unwanted) task. An inbuilt learning mechanism imparts the agents the ability to deal with the changes. The artifacts, agents, and tasks are described below.

#### A. Artifacts

We have created three types of artifacts: target points which act as goals for the agents, guns which hit anything which comes within a radius of 5 pixels, and double target points which need the convergence of two agents to be claimed as achieved. All artifacts are randomly placed in the environment and their quantity is kept constant. All of them are static.

#### B. Agents

The environment has the potential to support a multitude of agent types (or species), where each type can have some unique abilities or characteristics that differentiate it from others. However, for the current experimentation we have defined only one species of agents. Each agent is an ANN with fixed architecture shown in Fig. 1. Each of the agents receives 14 inputs: distance and angle from 2 closest targets, distance and angle from 2 closest agents, distance and angle from one closest firing gun, and distance and angle from two closest double targets. The distance is Euclidean distance and the angle is absolute angle taken from the perspective of the agent (as if the agent is positioned at the center of the coordinate system). Both outputs of the ANN give a number between 0 and 1. The first output is converted into number of steps to move (0 to 10) by multiplying the raw output by 10. The second output is converted into angle of movement, in degrees, by multiplying it by 360. Hidden layer consists of five neurons, and sigmoid activation function was used for all neurons. The architecture is not tested for optimality and has no other justification except that it seems reasonable and an almost similar architecture has been used by Yannakakis [2].

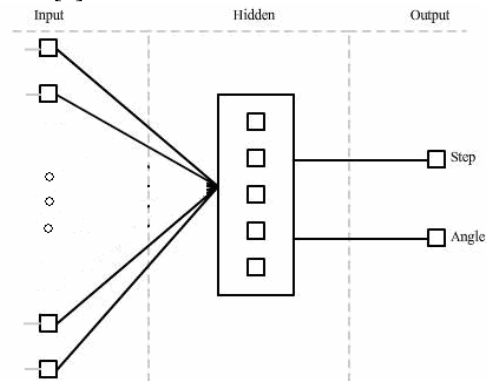


Fig. 1. The architecture of the ANN controller. There are 14 inputs, 5 hidden neurons and 2 outputs.

#### C. Tasks

**Target Achievement:** An agent has to reach a target point to achieve this task. An agent does not have a designated target point. It can try to reach any one of them. However, the fitness function is designed so that the agents are encouraged to try to reach their nearest target point in minimum number of steps. When a target is achieved by an agent it vanishes from the environment and a new target is randomly placed to keep the quantity of targets constant.

The fitness function for this task is simply the number of targets achieved by an agent in a fixed number of time steps. The number of targets achieved is divided by the maximum number of targets expected to be reached, so that the fitness function is normalized between 0 and 1. This number can be determined experimentally by first training the agents with the help of un-normalized fitness function and then using the best agent found to determine the maximum targets achieved. The normalized fitness function is:

$$\text{Min}(1, \text{targets achieved} / \text{maximum expected targets})$$

**Collision Avoidance:** An agent has to avoid collisions with other agents present in the environment. This task has to be combined with some other movement task so that such trivial learning may not take place where all agents stand still in order to avoid collisions.

The fitness function of this task is the same as used in [2]. For normalization, both the targets and collisions are divided by their maximum expected numbers. The normalized fitness function is:

$$0.5 * \text{Max}(1 - (\text{collisions made} / \text{allowed collisions}), 0) + 0.5 * \text{Min}(\text{targets achieved} / \text{maximum targets expected}, 0)$$

**Fire Avoidance:** An agent has to avoid coming in the range of guns which fire at everything within a specified radius around them. This task, also like the collision avoidance task, has to be combined with some other movement task so that the agents may not learn to stand still in order to avoid being fired upon.

For normalization, both the targets and hits are divided by their maximum expected numbers. The normalized fitness function is:

$$0.5 * \text{Max}(1 - (\text{hits made} / \text{allowed hits}), 0) + 0.5 * \text{Min}(\text{targets achieved} / \text{maximum targets expected}, 0)$$

Even though the collision avoidance and fire avoidance tasks seem the same there are two fundamental differences. First, in collision avoidance the agent to be avoided is moving while in fire avoidance the artifact to be avoided is static. Second, collisions occur when the two agents collide while the guns can hit from a distance.

**Double Target Achievement:** An agent has to cooperate with some other agent and both of them have to reach the double target point in order to achieve it. This implies that an agent has to learn to stop moving after reaching a double target point in order to allow another agent to reach it. Also, an agent has to learn to move towards a distant double target point instead of the nearest one if the distant one has already an agent near it.

The fitness function for this task is simply the number of targets achieved by an agent in a fixed number of time steps. For normalization, the number of double targets achieved is divided by the maximum number of double targets expected to be reached. The maximum expected double targets can be found in the same manner as described for target achievement task. The normalized fitness function is:

$$\text{Min}(1, \text{double targets achieved} / \text{maximum expected double targets})$$

#### IV. LEARNING ALGORITHM

In the above presented environment, the agents have to learn their assigned task. The task is then changed arbitrarily, and the learning algorithm has to be versatile enough to make the agents learn the new task.

Our learning algorithm is based upon the PSO algorithm. PSO is a heuristic search algorithm modeled on behavior patterns of flock of birds [13]. A population of solutions, called particles, is spawned randomly and then “flown” in the search space towards areas of higher fitness. Record of the particle which has found the best fitness since the start of the search is kept and called global best and this information is available to all particles. Each particle also remembers the position where it found its personal best fitness. The particle movement is determined by an equation which takes into account the global best as well as the personal best and which also has a random component. We have used *lbest* variation of PSO in which the population is grouped into several sub swarms. It is believed that *lbest* PSO provides more diversity at the cost of slower convergence. Positions of the particles are updated using the equation:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

where

$$v_i(t+1) = wv_i(t) + c_1r_1(t)(y_i(t) - x_i(t)) + c_2r_2(t)(\hat{y}_i(t) - x_i(t))$$

$x_i$  represents the current position of particle  $x$  in dimension  $i$ ,  $v_i$  represents current velocity,  $y_i$  is the personal best position of the particle and  $\hat{y}_i$  represents that sub-swarm’s global best (called neighborhood best) position to which the particle being updated belongs. The constants  $c_1$ ,  $c_2$ ,  $r_1$  and  $r_2$  are used to control the area in which search is to be conducted and  $w$  is inertia weight. The velocity updates are clamped in a range  $[-v_{\max}, v_{\max}]$ .

The PSO algorithm tends to fail in dynamic environments, because given an environment the particles converge on an optimum point in the fitness landscape. As soon as the optimum point changes due to changed fitness function the particles gathered in the old convergence area may not have enough diversity to find the new optimum. They are further impeded by the fact that the global best and personal bests are still at positions which may be now in an area with very low fitness, but the PSO update equation tries to keep the particles in that area.

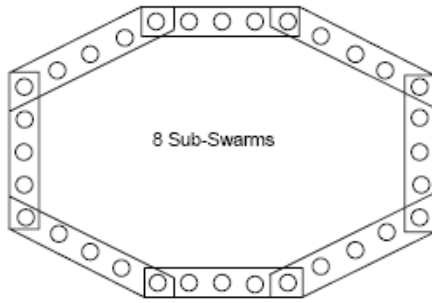


Fig. 2. The architecture used for defining sub-swarms. The 32 particles are divided into 8 sub-swarms based on their index numbers. The corner particles are members of two sub-swarms and follow the better of the two local bests available to them.

To overcome the limitations of the PSO, we have introduced re-initialization of one of the worst performing sub-swarms in the *lbest* version of PSO (Fig. 2). The re-initialization of a sub-swarm to random positions injects enough diversity in the particles so that they can overcome the disadvantages of previous convergence in the event of an environment change. On the other hand, if the environment is the same as that of the previous iteration, the other sub-swarms keep on learning according to the old global and personal bests. Hence learning is not disturbed if the environment is stable but there is an opportunity to learn new strategies if the environment changes. It also seems plausible that this balance between striving for convergence (exploitation) while maintaining enough diversity (exploration) is beneficial for avoiding sub-optimal and premature convergence even in static environments. The steps of our algorithm for learning in environment where tasks can change, are presented below:

- 1) Randomly initialize (in the range [-5, 5]) a PSO swarm with P particles each representing the weights of an ANN. Form S overlapping sub-swarms in the population of particles. Each ANN represents an agent which is trying to learn its own specific strategy to deal with the environment.
- 2) Initialize necessary parameters (i.e.  $c_1$ ,  $c_2$ ,  $r_1$ ,  $r_2$ , and  $w$ )
- 3) Set a randomly selected task to be learnt. Task is manifested in the fitness function being used for evaluating the particles.
- 4) Repeat this step for TT iterations
  - a. Evaluate fitness of each particle
    - i. For a particle create N clones in the environment.
    - ii. The clones are allowed to perform for E time steps in the environment and their performance is monitored.
    - iii. Using the current fitness function, fitness of the particle is calculated based upon the collective performance of its clones.

- b. Update personal best of the particles and the local bests of the sub-swarms.
- c. Determine  $v_{\max}$  dynamically for each sub-swarm.

$$v_{\max} = \max\{1.05 - (\text{local best} / \text{best of local bests}), 1\}$$

The sub-swarm having the best local-best will have a low  $v_{\max}$  of 0.05, so that it does not wander about too much, and the other sub-swarms will have a higher  $v_{\max}$ . The sub-swarm with poorest local best will have highest  $v_{\max}$ , so that it may have independence to leave the low fitness area in which it is currently finding itself.

- d. Use *lbest* PSO update equation to find the new positions of the particles.
- e. After every T iterations force new learning
  - i. Sort sub-swarms on the basis of accumulated fitness of the sub-swarm particles in the current iteration
  - ii. Reinitialize the worst (bottom most) sub-swarm to random positions. An exception is made if the neighborhood best of the sub-swarm is found to have improved in the previous T iterations. (This exception is made to discourage re-initialization of a sub-swarm which is improving). If the worst sub-swarm cannot be re-initialized due to this constraint, the next worst sub-swarm is reinitialized. The second worst sub-swarm is also subject to the same constraint and if we fail to re-initialize it then we do not have any re-initializations.

## 5. Go to Step 3

In the next section, we present the details of our experiments based on this algorithm and the results we got.

## V. EXPERIMENTS & RESULTS

Our main concern is to determine whether the agents using the continuous learning algorithm are able to adapt and learn new strategies when challenged with changing tasks. For this purpose, we create an environment (Table I) and test our learning algorithm whose parameters are shown in Table II.

The results for one sequence of changing tasks are presented in Fig. 3. On the x-axis are learning iterations and on the y-axis is the best found in the *current* iteration. The first task is target achievement and the agents are allowed to learn and develop their strategies for 1,000 iterations. At that point the task is changed to double target achievement. The agents perform poorly at first, but are reinitialized one by one and start to relearn and again achieve good performance. After 2,000 iterations (from the beginning of the

experiment), the task is again changed and this time the artifacts which fire upon anyone passing within a radius of 5 pixels are activated. The task for the agents is to achieve targets without getting hit. The double target achievement is deactivated. The agents again perform poorly and then start to relearn. After 3,000 iterations, the task is changed again. The new task is to achieve double targets. The single target achievement and hitting artifacts are deactivated. The agents are again able to learn to achieve the new task. After 4,000 iterations the task is changed to avoiding collisions with other agents while achieving targets.

TABLE I  
PARAMETERS OF ENVIRONMENT FOR EXPERIMENTS

Number of Agents (Clones)	20
Targets (this number is kept constant; as soon as a target is achieved another one appears at a random location)	30
Double Targets (this number is kept constant; as soon as a target is achieved another one appears at a random location)	10
Guns placed at (20, 20), (20, 60), (60, 20), (60,60) and (40, 40)	5
Guns Hit Range in all directions (in pixels)	5

TABLE II  
PARAMETERS FOR LEARNING ALGORITHM

Number of particles	32
Parameters $c_1, c_2$ & $w$	1
Parameters $r_1, r_2$	0 – 1
$v_{max}$ is variable according to fitness of sub-swarm	0.05–1
Sub-swarms	8
Iterations between re-initialization of sub-swarm	20
Clones/particle (for fitness evaluation of particle)	20
Time-steps/clone (for fitness evaluation of particle)	300
Number of maximum targets for normalization of fitness function of target achievement task	220
Number of maximum collisions for normalization of fitness function of collision avoidance task	50
Number of maximum hits for normalization of fitness function of hit avoidance task	300
Number of maximum double targets for normalization of fitness function of double target achievement task	100

Several random sequences of tasks are created and analyzed. Results of the experiments show that the continuous learning algorithm is effective in coping with the changes occurring in the environment. Sooner or later, after a task change all or most of the sub-swarms reinitialize and start learning again. How soon the new learning occurs and how widespread it is, depends upon the number of learning

iterations allowed before re-initialization of a sub-swarm. It is not an overly-sensitive parameter. However, we can have a lesser number if the two consecutive tasks are completely different and the strategies learnt for the first task are not better than random strategies when the task changes. If the tasks are somewhat similar and the strategies learnt for the first task are much better than purely random strategies for the second task, then we need a large number of learning iterations before attempting to re-initialize a sub-swarm. If the learning iterations are not sufficient the same sub-swarm may get re-initialized again and again. In such cases, the constraint that a worst sub-swarm cannot be re-initialized if its local best has changed during the learning period, helps us in preventing too frequent re-initializations.

A special case is that of second task being a sub-set of first task. An example is target achievement with collision avoidance as task followed by a task change to target achievement only. In such a case the learnt strategies are already good performers on the second task and the re-initialization does not contribute much.

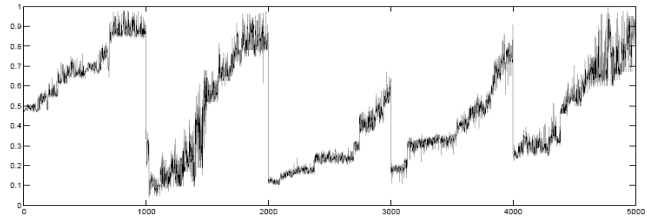


Fig. 3. The five tasks changes are in the following sequence: target achievement, double target achievement, gun fire avoidance with target achievement, double target achievement, and collision avoidance with target achievement. The particle-agents are allowed to learn for 1,000 iterations before each task change. The x-axis shows the iteration numbers and the best of all the local best fitness of the sub-swarms in the current iteration is plotted against the y-axis.

We have performed experiments without re-initialization and the results of one such experiment (for target achievement task followed by double target achievement task) are shown in Fig. 4. The agents are allowed to learn for 1,000 iterations in the environment according to the fitness function of the first task and then the fitness function is changed. Since the PSO particles have their local best and personal bests according to the old fitness function they are not able to explore the new fitness landscape. The same failure to learn is observed for most of the experimental runs that have been made for different task sequences.

The last experiment is to explicitly inform the learners that the tasks have changed. This is in the form of complete re-initialization of all agents to random values at the change of tasks. The results are presented in Fig. 5 for target achievement task followed by double target achievement task. The explicit information allows the learner to learn faster and achieve high fitness in relatively less number of iterations when compared with the case when no such information is available.

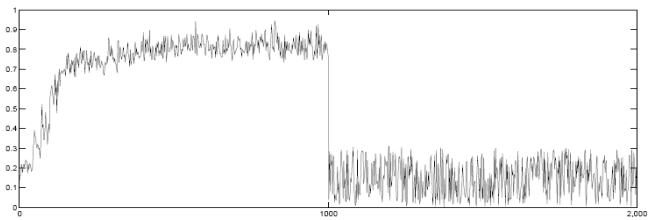


Fig. 4. The result of non re-initialization of sub-swarms. The task is target achievement for the first 1,000 iterations which the randomly initialized particle-agents learn to handle. As soon as the task changes to double target achievement after 1,000 iterations, the particle-agents perform poorly and fail to learn to cope with the new task.

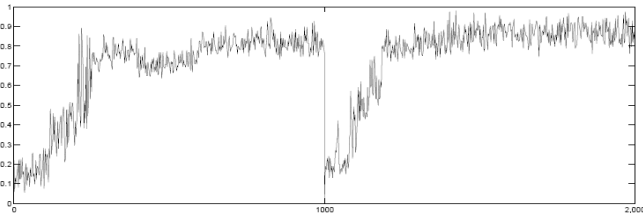


Fig. 5. The result of explicit re-initialization of sub-swarms at the onset of task change. The task is target achievement for the first 1,000 iterations. The task is then changed to double target achievement and all the particle-agents are re-initialized to random values. As expected, the particle-agents learn the new task much quickly with the help of this explicit intervention, as compared to the results obtained with worst sub-swarm re-initialization algorithm where no information about the task change is available to the particle-agents.

We summarize that our learning algorithm is able to learn new strategies according to changed tasks. It performs well while the classical PSO fails to do so. This shows that it is better if there is a portion of the population continuously trying to learn new strategies as compared to a population which has converged on optimum strategies.

## VI. CONCLUSION

The humans live in a dynamic world that presents new challenges in an arbitrary fashion and they are forced to develop new strategies in order to cope with new situations. In this work, we attempt to model and solve the problem of learning in an environment where the tasks change without any warning. For this purpose, we enhance an already existing gaming environment and use it to test our PSO based learning algorithm. Poorly performing agents are weeded out and new ones are created in their place to promote diversity, while average and above average agents continue to learn with efficient strategies being disseminated throughout the population. This continual learning is more coherent with the learning that takes place in our everyday environment. Our experimentation results show that our learning algorithm is effective.

One potential extension of this work is the incorporation of archives to retain previously developed reliable strategies so that we may not have to relearn them if the environment returns to one of its previous states. Another extension can be to have an environment where multiple tasks are present at the same time and each agent has a set of previously

learnt strategies for achieving these tasks. The agent learns strategies to achieve these tasks and keep them in its memory. It also learns to determine which task to pursue under the present conditions in which it finds itself.

Our learning algorithm may also be enhanced to deal with incomplete (imperfect) information. Each agent may determine whether new information is available, what new changes have occurred in the environment and whether already known information is now outdated. Each of the agents can have an archive, where it stores strategies that it has learnt so far. Agents can also be made to determine whether the new information is actually helping them improve their strategies or is it irrelevant to their needs. Outdated information may be discarded by the agents.

Even though we have enhanced the *Flatland* environment there is still room for more complex artifacts, agents and tasks. For example, we can have a multitude of agent types (or species), where each species has some unique abilities and architecture that differentiate it from others. We can have a flying species, swimming species, and yet others which can hear. The agents of the same species can all have the same task(s) or they can have different goals and act as a team to achieve some higher objective.

## REFERENCES

- [1]. K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-Time Neuroevolution in the NERO Video Game," *IEEE Trans. On Evolutionary Computation*, Vol. 9, No. 6, Dec. 2005.
- [2]. G. N. Yannakakis, J. Levine, and J. Hallam, "Emerging Cooperation with Minimal Effort: Rewarding Over Mimicking," *IEEE Trans. on Evolutionary Computation*, Vol. 11, No. 3, June 2007.
- [3]. G. N. Yannakakis and J. Hallam, "A Generic Approach for Obtaining Higher Entertainment in Predator/Prey Computer Games," *Journal of Game Development*, Vol. 1, No. 3, Dec. 2005
- [4]. S. M. Lucas, "Cellz: A Simple Dynamical Game for Testing Evolutionary Algorithms," in *IEEE CEC*, 2004.
- [5]. G. Kendall and Y. Su, "Imperfect Evolutionary Systems", *IEEE Trans. on Evolutionary Computation*, Vol. 11, No. 3, June 2007.
- [6]. L. Messerschmidt and A. P. Engelbrecht, "Learning to Play Games Using a PSO-Based Competitive Learning Approach", *IEEE Trans. on Evolutionary Computation*, Vol. 8, No. 3, June 2004.
- [7]. N. Franken and A. Engelbrecht, "Comparing PSO Structures to Learn the Game of Checkers from Zero Knowledge," in *IEEE Congress on Evolutionary Computation*, Canberra, Australia, Dec. 2003.
- [8]. K. Chellapilla and D. Fogel, "Evolving an Expert Checkers Playing Program without using Human Expertise," *IEEE Trans. on Evolutionary Computation*, Vol. 5, No. 4, Aug. 2001.
- [9]. C. M. Frayn, "An Evolutionary Approach to Strategies for the Game of Monopoly®", *Proceedings of IEEE Symposium on Computational Intelligence and Games*, 2005.
- [10]. R. G. Reynolds, Z. Kobti, T. A. Kohler, and L. Yap, "Unraveling Ancient Mysteries: Reimagining The Past Using Evolutionary Computation In A Complex Gaming Environment," *IEEE Trans. On Evolutionary Computation*, Vol. 9, No. 6, pp. 707–720, Dec. 2005.
- [11]. T. Blackwell and J. Branke, "Multiswarms, Exclusion, and Anti-Convergence in Dynamic Environments," *IEEE Trans. on Evolutionary Computation*, Vol. 10, No. 4, Aug 2006.
- [12]. D. Parrott and X. Li, "Locating and Tracking Multiple Dynamic Optima by a Particle Swarm Model Using Speciation", *IEEE Trans. on Evolutionary Computation*, Vol. 10, No. 4, Aug 2006.
- [13]. A. P. Engelbrecht, "Fundamentals of Computational Swarm Intelligence", John Wiley & Sons, 2005.