International Journal of Innovative Computing, Information and Control
Volume **7**, Number **4**, April **2011**

 ISSN 1349-4198
pp. 1–09-1160

SELF-MOTIVATED AND TASK-ORIENTED, MULTI-DIMENSIONAL LEARNING IN A DYNAMIC AND UNCERTAIN ENVIRONMENT WITHOUT HUMAN INTERVENTION

Hasan Mujtaba, Rauf Baig, Zahid Halim and Amjad Iqbal

Department of Computer Science
National University of Computer and Emerging Sciences
A.K. Brohi Road, H-11/4, Islamabad, Pakistan
{ hasan.mujtaba; rauf.baig; zahid.halim; muhammad.amjad }@nu.edu.pk

Received November 2010; revised March 2010

Abstract. *The abilities to accept new information from the environment and use it to update our existing knowledge thus adapting to the changes of our environment have played a crucial role in the success of human beings as a species. Incorporating these abilities in machines has been an age long desire of artificial intelligence. In this paper, we present a learning technique based on evolutionary approaches that enables artificial agents to detect changes in their environment and adapt accordingly. Our focus is on enabling the agents to learn new tasks without any human intervention, relying only on stimulus from their environment. We argue that learning in such a dynamic environment should be a continuous process and past experiences must be retained for future scenarios. The learning method itself provides a mechanism where the decrease in performance, forced by the change in goals, triggers new learning. We conduct experimentation to show how this approach works and results from these experiments are very encouraging.*
Keywords: Artificial intelligence, Evolutionary computation, Individual and social learning, Continuous learning, Learning in a dynamic environment, Particle swarm optimization, Artificial neural network, Learning in imperfect information, Evolutionary games

1. **Introduction.** An intelligent system is a system that acts intelligently, i.e., it reacts according to its surroundings and needs. It must cater for changes in its environment and objectives. Ideally the abilities of such a system depend upon the information available to it and its perceptual and computational limitations. Such an intelligent system must be able to update its existing knowledge space when new information is available to it. In other words, intelligence can be defined as *the ability to learn new abilities*. This ability has been pivotal to the success of humans as a species. We (humans) adapt according to the changes occurring in our environment, allowing us to not only survive but thrive in (almost) any environment. It must be noted that these (real-world) environments are not static. On the contrary, they are uncertain and constantly changing. These changes maybe subtle and slow or discreet and obvious. In this paper, we present an algorithm based on evolutionary approaches (Particle Swarm optimization and Artificial Neural Networks) to enable the agents residing in it to learn from and behave according to changes occurring in their surroundings. These intelligent agents build a dynamic relationship with their environment enabling them to monitor changes and explore new possibilities. This is crucial for a successful evolution of the species. To the best of our knowledge, such a dynamic bond has been missing in most of the research work conducted in artificial intelligence (AI) research. To test our hypothesis and our learning algorithm, we developed a game like environment to conduct experimentation.

Using gaming environments, researchers have tried to test limits of automated (non-human) players. One of the earliest achievements was made when *Deep Blue* [1], a chess playing computer developed by IBM, won against the reining chess world champion, Gary Kasporov. Later on, *Chinook*, developed by Jonathan Schaeffer, won the world checkers title back in 1994 [2,3]. However, both these systems relied heavily on a *brute force* approach and this led to an argument about whether such systems can be called intelligent or not. Chellipela and Fogel, developed *Blondie24* [4,5] which learned to play checkers without relying on any human expertise. *Blondie24* used an Artificial Neural Network (ANN) based co-evolutionary approach, i.e., in order to teach itself the game, it played games against itself. Although this was a significant improvement on previous learning mechanisms, it was still far from natural learning mechanism. *Blondie24* evolved an ANN during its learning phase which remains the same during its evaluation. This limitation restricts the system from detecting and adapting to any changes that have occurred in its surroundings. As mentioned earlier, response to external stimuli is crucial in the success of natural intelligence.

Stanley et al. [6] designed "Neuro-Evolving Robotic Operatives" (NERO) in which agents were trained to perform specific tasks. Once trained, teams of agents were deployed to fight off against other agents. The shortcoming of NERO agents however, is the lack of ability to learn from their surrounding environment at runtime. In a dynamic environment (where new inputs are being generated and old ones discarded) NERO agents remain oblivious to the changes around them and fail to see a change in task assigned to them, e.g., agents will not recognize the change if a "*capture the flag*" (where the objective it to capture the enemy flag while defending your own) scenario changed into a "*deathmatch"* (the objective is it to kill as many opponents agents as you can). In our opinion, the most significant work to address this failure to adaptive intelligence and respond to external stimulus has been of Kendall [7]. In their research on an Imperfect Evolutionary System (IES) they point out that this problem is due to the lack of a relationship between agents and their surroundings. Their work only focuses on dealing a change in the environmental parameters and does not deal with a change in objective of the agents.

From *Deep Blue to IES,* AI and AI based systems have come a long way in terms of learning. However, we believe that so far AI research has ignored the role of a changing environment in the evolutionary process. In this work, we present a new learning mechanism that deals with the imperfect and changing nature of the environment. In our approach, the environment not only acts a medium for evolution but is itself a part of it. Intelligent agents in this environment must be able to respond to change(s) at any time and modify their behavior according to the nature of the change (a change can be availability of new input parameters, introduction of new entities or species in the environment, change in agent's objectives, etc.). This self motivated and self directed evolutionary process allows the agents to develop better strategies to meet the demands of their environment. Humans play no part in this process, using the algorithm it learns everything on its own. Hence it is possible to avail opportunities that might have been ignored by a human performing training of the agents.

Our learning algorithm allows the agents to deal with the imperfection of information by:

- Accepting new information from the environment and updating the existing knowledge space.
- Exploration and exploitation of new opportunities provided by the environment.
- Training for a task (or objective) depending solely on the information provided by the environment.

- Handling multiple objectives without knowing a priori what those objectives might be.

Major contributions of the learning approach presented in this paper are as follow:

- Using the proposed learning algorithm, agents can acquire new skills that will help them survive the demands of their environment. These skills are acquired and updated without relying upon any human intervention.
- This technique allows agents to learn and adapt as per the requirements of their environment (these requirements change overtime without any warning).
- It is possible for agents to use this algorithm to learn multiple objectives (or skills) without any a priori information about those objectives (or skills).
- Our learning technique uses evolutionary approaches, a Particle Swarm Optimization (PSO) variant & Artificial Neural Network (ANN) to detect change, based on this change, new information can be added or obsolete information removed from current the knowledge space.

Although current learning approaches have been very successful in some domains, we believe their application is limited. Consider, for example, a co-evolutionary approach to game playing. Players compete against one another to find the best one. Once a strategy is successful against an opponent the learning methodology will force the remaining strategies to be similar to it. All current approaches will ignore any new development in the opponent. If the opponent devised a way to cheat the learning methodology will remain oblivious to this. Furthermore learning by observation or exposure is common to humans. We can learn new *things* by observing others; a child can learn to play checkers by observing others play the game. Keeping this in focus, our learning algorithm is not designed to evolve in a certain environment or a particular task; rather it builds a relationship with its surrounding environment and learns new abilities from it. The environment can be changed to a new state at any given time and the evolution process will incorporate the new changes within the learning process on its own.

The rest of the paper is structured in the following way: in Section 2, we present other learning methodologies and compare them with our own work. Section 3 presents the framework. In Section 4, we detail the environment we created to test the learning ability of our algorithm and present the results of these tests. Conclusions drawn from these results and recommendations for future work are given in Section 5.

2. **Problem Statement and Preliminaries.**

Learning in a dynamic environment can be compared with dynamic function optimization (DFO). In dynamic function optimization, all the information about the environment remains constant and the optimum point relocates its position. Two representative swarm based algorithms for dynamic function optimization are [8,9]. Cultural algorithms based approaches have been used by [10-12] to emulate a cultural evolutionary process. Reynolds and Saleem in [13,14] and Reynolds and Peng [15] describe application of cultural algorithm for function optimization problems in dynamic environments. However, the dynamic environment presented by them only changed values of existing environmental variables and does not deal with introduction of new variables. The dimension of the problems observational search space also remains unchanged.

Other studies that have investigated dynamic environments in AI research are fuzzy logic [16,17], qualitative reasoning [18], commonsense reasoning [19,20]. Bayesian networks were used in [21]. Evolutionary computation (EC) [22-25] has been used to study adaptive intelligence. Evolutionary learning has resulted in generating behaviors and solutions which were, in most cases, unexpected or previously unknown. Fogel et al. in [4,5]

present advances in developing adaptive intelligence in computer games using EC techniques. Uncertain and nonlinear systems were addressed by [26,27]. However, we believe that in [7], Kendall and Su presented the revolutionary idea of learning with imperfection. They evolved ANN based agents to deal with an environment in which all the input variables are not available at the onset and are incrementally added during learning. The main focus of their work was to introduce the idea of an environment in which information may change at later time slots. In this paper we introduce the idea of continuity of learning. As the Section 4 explains, we deal with an environment where change can occur at any level of the environment at any time. Agents are never informed about any changes. They must detect and self-adjust their behavior to deal with environmental changes.

J. C. R. Tseng et al. [28] developed *META-ANALYZER*, a web-search learning environment based on the meta-index approach. Their environment, however, does not deal with addition of new environment attributes or acquisition and learning of new abilities based on these new attributes. Fryan has used an evolutionary approach for evolving game playing strategies for Monopoly [29]. Another interesting experiment with evolutionary approach and games is presented in [30]. ANN based controllers have been used by Chellapilla and Fogel in [4], Fogel in [5] and the NERO game developed by Stanley et al. [6]. ANN controllers were also used by Yannakakis et al. [31] in a simulated world called *Flatland*. Yannakakis and Hallam have also evolved ANN controllers for the games of Pac-Man and Dead End [32]. Yet another effort in this field has been made by Lucas for the game of Cellz [33]. ANN based agents were also evolved by Messerschmidt and Engelbrecht for playing Tic-tac-toe [34] and by Franken and Engelbrecht for playing checkers [35]. Z. Zhang et al. [36] used NN based approach for the Boolean Series Prediction Question (BSPQ) problem.

Archives in games have been used for various purposes. One of its major uses has been to store cases for cultural algorithms. One representative work is [37]. Another use has been to overcome the problem of "forgetting" in co-evolution algorithms. An example of this use can be found in [38]. S.-Z. Zhao and P. N. Suganthan also used archives in [39].

In contrast to these traditional approaches, our learning algorithm is different from any previous work. Our focus is on an environment where change is essential part of the evolutionary process. Here anything can change (i.e., the nature of the problem, the goal of evolution or an evolutionary parameter(s) or even the environment itself can change) at any time. New information may become available and previously known information may become obsolete. A change in the environment will usually render the current learning void, resulting in not only a change in the shape of the search space but the search space itself. Traditional learning mechanisms restricted learning by forcing agents to move towards best solutions without considering the change in environmental status. Agents evolve strategies over generations to deal with their environment. These strategies ignore the new information available, rendering these strategies useless. Our learning algorithm allows the agents to discard these obsolete and outdated strategies in a changed environment but archiving them in case environmental changes revert back.

Such changing behavior is common to our natural world. An example of such a change can be the climatic changes occurring around us, forcing humans to adapt accordingly. In an agricultural society, strategies deployed for a warm weather are different from a colder one. A rapid change in the climate can force a change in strategy earlier than expected. If the people living within that area fail to recognize this their survival can be jeopardized. Keeping these factors in view, we define an imperfect evolutionary world as a system constituent of different entities, as shown in (1). Each of these entities plays a

vital role in the evolution of the environment and its inhabitants.

$$E^T = \{i^T\} \cup \{I^T\} \cup \{R^T\} \cup \{O^T\} \cup \{U^T\} \tag{1}$$

At every time step T, the environment E^T is a union of set of inputs i^T, set of intelligent (evolving) individuals (or agents) I^T, set of rules applicable to the environment R^T, set of tasks or objectives that agents must perform O^T and U^T is the set of all other environmental entities, e.g., non-evolving agents present in the environment, environmental features like walls, forts, etc. With a change in T any of these sets can change. This change in E^T will affect the strategy S of an individual i (2).

$$S_i^T = \sum a_i^T - \sum \gamma_i^T \tag{2}$$

S_i^T is the strategy evolved by an individual i at time T. This strategy is evolved using the current set of inputs available to the agent a_i^T minus the set of obsolete or irrelevant inputs γ_i^T. Note that since each of the individuals present in the environment views it from its own perspective, each of these sets can vary between individuals for the same time T.

Thus learning in an unpredictable and changing environment should be a continuous process. This continuity in learning enables the agents from latter generations to avoid mistakes committed by previous generations. In the scenario mentioned above traditional learning techniques would force the evolution to move in the direction of the fittest strategy found so far. Ignoring the fact that after a change in environment the fittest strategy is now obsolete and following it would lead to disastrous consequences. Hence the evolutionary process stops itself from improving when the environment changes. Continuous Learning enables the learning process to continue by exploring new avenues of evolution and avoid this state of stalemate.

Agents are evaluated based on their individual fitness values according to the role played by them, keeping in view the current status of environment. These individual fitness values cumulate as the social fitness. Just as in the natural world, humans try to improve their own existence. This individual improvement ultimately improves the overall society. Our algorithm allows for a social evolution which is driven by individual evolution. However, individual choices may also be affected by the direction of social evolution. Similar to natural world, individuals in our algorithm sacrifice themselves for the benefit of their society. This interdependent social and individual learning mechanism allows for a better understanding and exploration of the environment by the agents. Information is disseminated between agents via the social learning mechanism, while the exploitation of environmental factors is primarily targeted by the individual learning.

3. **Experience Based Learning Algorithm for Unpredictable and Dynamic Environment.**

Our proposed learning algorithm enables agents to adjust according to their surroundings. We call it Experience based *Learning Algorithm for Unpredictable and Dynamic Environment* (ELUDE). This learning algorithm trains the agents using a swarm based evolutionary approach. Abilities of the individual agents can be represented as neural networks or rule-sets etc. We have used an *lbest* PSO based approach to train ANNs. Each of these ANNs represents a certain skill or ability acquired by the agents from the inputs available from the environment. An agent may acquire more than one skill or it may focus on learning one skill. This direction of learning is dictated by the current state and requirement of the surrounding environment. The learning algorithm acquires all the information from the environment and at no point requires any human intervention to guide its learning process.

PSO is a heuristic search algorithm modeled on behavior patterns of flock of birds [40]. A population of solutions, called particles, is spawned randomly and then "flown" in the search space towards areas of higher fitness. A record of the best particle found so far is maintained and is called global best. Information about global best position is available to all particles. Each particle maintains a personal best position. Movement of particle is determined by taking into account the global best, personal best and a random component. We use an *lbest* PSO variant in our learning algorithm. We believe that *lbest* PSO provides more diversity at the cost of slower convergence. In *lbest* PSO, the population of particles is grouped into several sub swarms. All the particles in a sub-swarm are completely connected to one another. Each particle forms a sub-swarm by being completely connected to its n neighboring particles. Positions of the particles are updated using the Equation (3):

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{3}$$

where

$$v_i(t+1) = wv_i(t) + c_1r_1(t)(y_i(t) - x_i(t)) + c_2r_2(t)(\hat{y}_i(t) - x_i(t)) \tag{4}$$

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 $\hat{y}_j$ 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 $[-v_{\max}, v_{\max}]$.

In our approach, instead of creating sub-swarms in a sliding window fashion (a particle is connected to its n neighbors and the next particle is connected to its n neighbors, and so on). We have several completely connected sub-swarms which are isolated from one another except through their corner particles. Each corner particle is completely connected with all the particles of two sub-swarms (Figure 1). These corner particles act as a source of information transfer.

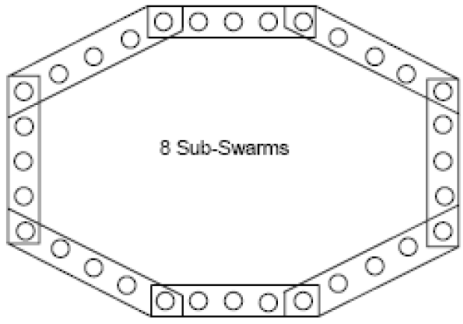

FIGURE 1. The architecture used for defining sub-swarms. The 32 particles are divided into 8 sub-swarms based on their index numbers. The corner particles are members of two sub-swarms and follow the better of the two local bests available to them.

In our initial experimentation with learning in a dynamic environment, we discovered that the classical PSO algorithm tends to fail in dynamic environments. This is 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. To overcome these limitations of

the PSO, we have introduced re-initialization of one of the worst performing sub-swarms in the *lbest* version of PSO. 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. We believe 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 learning algorithm are presented below:

1) Randomly initialize S overlapping swarms with P particles
2) Initialize c_1, c_2, r_1, r_2 and w
3) Create an archive A with x slots
4) Set a randomly selected task K for learning
5) Repeat this step for T iterations
   a. Evaluate fitness of each particle
      i. For every particle create N clones in the environment
      ii. Clones are allowed to move in the environment for E time steps
      iii. Fitness of the particle is determined by the performance of its clones
   b. Update personal best of the particles
   c. Determine the *lbest* particle of each sub-swarms
   d. After every T_a iterations
      i. Determine the *gbest* particle
      ii. Insert *gbest* particle into an empty slot of A
   e. Determine exploration boundary for each sub-swarm
   f. Use *lbest* PSO update equation to find the new positions of the particles
   g. After every T_r iterations force exploration of new opportunities
      i. Sort sub-swarms on the basis of their accumulated fitness F_a
      ii. Reinitialize the worst sub-swarm, i.e., sub-swarm with the lowest F_a value
   h. Update *age* of strategies present in A
6) Go to Step 5

The algorithm starts creating P particles, initialized randomly in the range $[-5, 5]$. Each of these particles represents a strategy to deal with the demands of the surrounding environment. These particles (or strategies) are grouped into S sub-swarms. Each particle is fully connected to all other particles within its sub-swarm, i.e., each particle can share information about its position with other particles of its sub-swarm. As shown in Figure 1, particles present at the corners of the sub-swarm share information between members of two different sub-swarms. Next parameters for PSO equations are initialized and an archive is created to store strategies. This archive acts as a pool for storing the best strategies that have been developed by the population of particles. The environment sets a task K for the agents to learn. Task is manifested in the fitness function being used for evaluating the particles. A task can be an objective the agents have to learn or a skill they must acquire. An agent of the system is represented by an ANN. Based on the inputs available from the environment; each agent tries to learn a task. During the evolutionary process, particles must choose whether they should improve their current strategy or should they abandon it and try a new one. As the agents are not given any explicit knowledge about the task that they have to learn; only their fitness evaluation is made available to them. Hence the decision to either refine current strategy or look for a

new one is based upon the fitness of their current strategy. Using this fitness value agents assess whether they are learning something useful or unnecessary (or obsolete, as the case maybe).

This fitness evaluation is performed by creating copies of an agent and deploying them in the environment. These copies (or clones as we call them) are allowed to perform the task they have chosen to learn based on the inputs for a certain number of time steps E. Their interactions with the environment from these E steps are monitored. These are passed on to the fitness function, which calculates their fitness based on current state of the environment. After fitness of all the particles has been calculated, the personal best (*pbest*) and the best particle of the sub-swarm (*lbest*) are determined. An *lbest* particle is the particle with the highest fitness value within a sub-swarm (if we have n sub-swarms, we will have n lbest particles). In contrast global best (*gbest*) particle is the best particle out of the entire population of particles (or the best of all *lbest* particles). *pbest* position of a particle is the best position it has ever achieved. *pbest* and *lbest* position of its sub-swarm are used to determine the next position of a particle using the PSO equations mentioned ealier.

While updating the position of the particle we apply a limit upon how far it can jump from its current position. We call this the exploration boundary. The rationale behind this boundary is to allow particles with good fitness to search around their current position in hopes of finding an even better solution. While weaker solutions are allowed to jump farther away from their current positions so they can find a better area to explore. This exploration boundary is set dynamically and is different for each sub-swarm (depending upon the fitness of its *lbest* particle). For our PSO based approach, we apply this movement restriction using velocity clamping, as following:

$$\nu^i_{\max} = \max\left\{\left(\mu - \left(lbest^i / gbest\right)\right), 1\right\} \tag{5}$$

$\nu^i_{\max}$ is the maximum velocity of the sub-swarm i and $lbext^i$ is its *lbest* particle. μ is set to 1.05 to avoid stagnation of the sub-swarm with the *gbest* particle. Using this velocity clamping the sub-swarm having the best fitness will have a low $\nu_{\max}$ allowing for better exploration of it current locality without wandering about too much. While the sub-swarm with poorest fitness will have highest velocity allowing it to escape the low fitness area in which it is currently searching. After every T_a iterations, the *gbest* particle is chosen as a candidate for addition to the archive. If the fitness value of this candidate particle is higher than a threshold value then it is added to archive. Duplication of particles is not allowed in the archive. Here it must be noted that threshold value too low or too high, based on experimentation we choose 0.7 – 0.8 out of a normalized fitness value (fitness is between 0 – 1, 1 being maximum fitness). This threshold seems to be dependent upon the complexity of the problem and the demands of the environment. If there is no empty slot then the most ancient strategy is deleted and the new strategy occupies its slot (first in first out).

In order to explore new opportunities in the environment, the worst sub-swarm is reinitialized after T_r iterations. This sub-swarm is chosen on the basis of its accumulated fitness F_a given in (6).

$$F^i_a = \sum_{T_a}(F^i_1 + F^i_2 + \ldots + F^i_n) \tag{6}$$

F^i_a is the accumulated fitness of sub-swarm i and it is the combined fitness values of all the particles of this sub-swarm summed over the past T_r iterations. It is possible for a sub-swarm to achieve high fitness in one state of the environment while the same sub-swarm main perform poorly after the change in environment. In such a case, monitoring only the previous T_r allows the learning algorithm to observe only the recent performance of a

sub-swarm. Ideally sub-swarms which have not shown any improvement in these recent iterations should be used for further exploration. On the other hand, sub-swarms that are currently performing well should be allowed to refine their strategies rather than explore new ones. For this reason, we not only check the accumulated fitness of a sub-swarm but also check if it has found its *lbest* particle within these last T_r iterations. A sub-swarm that has found its best particle within these iterations is probably improving itself and it should be allowed to improve. In such a case, we perform the same steps for the 2^{nd} worst sub-swarm. In case this 2^{nd} worst sub-swarm is also improving itself no further sub-swarms are checked. We only check the bottom two sub-swarms for exploration to avoid the whole algorithm becoming a random search in the search space. Once a sub-swarm is chosen for exploration, its particles can be reinitialized to random values or they may choose a strategy present in the archive (overlapping members of the sub-swarm are reinitialized along with the rest of the sub-swarm). If the strategy chosen from the archive is suitable for the current state of the environment, it will lead to better fitness value else its performance will be similar to a random re-initialization. In some of the experiments we chose the worst performing sub-swarm for random re-initialization while the second worst sub-swarm was replaced by a strategy from the archive, results of these experiments are presented in next section. Strategies present in the archive are tried in a fixed cyclic order. If the strategy present in a slot has been tried in the last iteration, then it is the turn of the next few strategies found in the next slots. Each member of the sub-swarm is initialized with one strategy. Hence we can try as many strategies as there are members in a sub-swarm.

This performance based learning enables the agents to explore different possible tasks (based on inputs it choose) to find the one best suited. After fitness evaluation of all particles of each sub-swarm, best sub-swarm is chosen. The best sub-swarm acts like a leader and all other sub-swarms try to adjust their behavior in coherence with this leader. No sub-swarm is told exactly what the leader is doing. The information shared is through the PSO update equation to find the new positions of the particles.

4. **Experimentation.**

We now describe our experimental setup for our experimentation. In order to test the learning ability of ELUDE we have created an uncertain and changing environment. Although this environment has similarities with *Flatland* [31], Cellz [33] and Dead End [32] but it serves a different purpose. This environment was developed to model uncertainty and imperfection of information similar to our real-world scenarios. Serving as the test bed for our experiments, it is a multi-agent environment in which agent can interact with each other and other artifacts. These interactions are monitored to calculate the learning ability of an agent present in the environment. It is a torus (the agents can reappear on the opposite side once they go over the boundary) 80×80 pixel, 2-D, environment in which a number of learning agents can be present along with non-learning artifacts. Agents have to learn to survive the different states of the environment. The demand of the environment can change at any time without any explicit warning. The change in task is manifested by the changed fitness function. The only information available to the agent is in the form of the fitness evaluation of their current strategy. There are two different kinds of agents present in the environment, artifacts (non-learning agents) and agents (learning agents).

4.1. **Artifacts.**

We have created two types of artifacts:

a. Food particles (target points) which act as goals for the agent,
b. Carnivores (animals) which can act both as predators and prey.

Even though several other artifacts can be conceived and used (e.g., the artifact called gun [41,42]), we think that these two artifacts are sufficient for our current set of experiments. Food particles are static, while the carnivores can move.

These carnivores play an interesting, dual role in the environment. They act as predators when they encounter a single agent and become prey when two (or more) agents cooperate to hunt a carnivore. Artifacts are randomly placed in the environment and their quantity is kept constant, i.e., if a food artifact is eaten up by an agent another one is spawned at some random location.

4.2. **Agents.** Agents have the ability to learn using the learning algorithm. For the current experiments we have only one type (or species) of agents which have the capability to observe their surrounding environment and act accordingly. However, the environment can support a multiple species. Each of these species can have unique abilities or characteristics that differentiate it from others.

For our current experimentation, each agent is an ANN with fixed architecture shown in Figure 2.

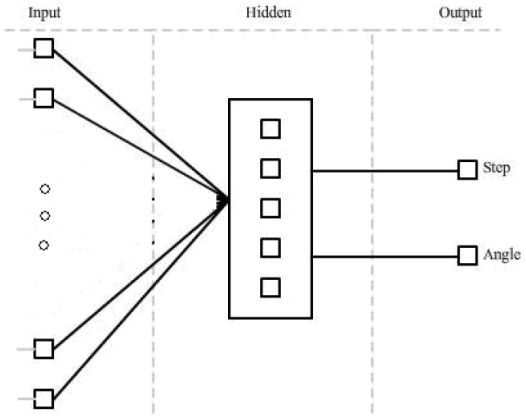

FIGURE 2. The architecture of the ANN controller.

Each of the agents receives different inputs from the environment. These are distance and angle from $z\,(z = 2)$ closest food targets, agents and carnivores. The distance is Euclidean distance and the angle is absolute angle taken from the perspective of the agent. There are two outputs of the ANN, direction and size of step of the agent. The hidden layer consists of five neurons with sigmoid activation function. The purpose of the ANN is to provide a strategy for learning of any task rather than mastering a given task, this learning need not be optimal and this architecture serves its purpose well, a similar been used by Yannakakis [27].

4.3. **Environment phases.** The environment we have developed can change its phase at any time without any warning to the agents. Each phase change demands the agents must learn a new ability. The objective of each phase is to train the agents to learn a specific task. We have used four different environment phases in our experimentation, these are.

4.3.1. *Grazing phase (food target achievement).* In this phase of the environment, agents have to collect food. Ideally each agent should approach its closest food point. Thus 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 fitness function for this phase is:

$$f^p = \min\left(1, t_a^p / t_a^m\right) \tag{7}$$

f_p is the fitness of the particle p and t_a^p are the total food points collected by it. t_a^m is the maximum number of food points allowed for collection.

4.3.2. *Awareness phase (collision avoidance).* In this phase, agents have to avoid collision with other agents. Agents are motivated to move by using the number of steps taken by them. Fitness is calculated using

$$f^p = \max\left(0, (t_c^m - t_c^p)/t_c^m\right) \tag{8}$$

f^p is the fitness of the particle p, while t_c^p are the number of collisions made by it. t_c^m are maximum collisions allowed by a particle.

During experimentation, we noted that agents came-up with some very interesting strategies to coup with the needs of the environment. In order to avoid collisions agents stopped moving. If they do not move they will not collide. To motivate agents to move we placed a threshold on the number of steps. Agents must move at least 5000 steps else their fitness will not be calculated and will remain 0. As a reaction to this, agents started continuously jumping back and forth between two points. Although we placed another to check to avoid this behavior, it highlights the algorithms ability to exploit any weakness in their surrounding environment.

4.3.3. *Survival phase (predator avoidance).* During this phase carnivores are introduced into the environment. Agent must avoid being eaten up by these carnivores. A carnivore actively pursues an agent according to the following rule:

At each time step, take one step towards the nearest agent.

In order to balance the odds, the maximum moving speed of the predator is half the maximum speed of the agents. If the predator catches an agent, agents deaths are incremented and it agent is randomly re-spawned. Fitness function for this phase is as follows:

$$f^p = \max\left(0, (t_d^m - t_d^p)/t_d^m\right) \tag{9}$$

f^p is the fitness of the particle p, while t_d^p are the number of times the agent died at the hands of the carnivores. t_d^m is the maximum number of deaths allowed for a particle.

4.3.4. *Hunting phase (carnivore consumption).* In this phase, agents have to cooperate with other agents to capture a carnivore. If the carnivore is caught, then it is randomly placed somewhere else and a successful capture is counted for all agents that took place in this capture. The carnivores move according to the following rule during this task:

At each time step, take one step away from the nearest agent.

The fitness function for this task is the number of carnivores captured by agents.

$$f^p = \min\left(1, t_h^p / t_h^m\right) \tag{10}$$

f^p is the fitness of the particle p and t_a^p are the number of carnivores successfully hunted by it. t_h^m is the maximum number of hunts allowed.

Difference between *Survival* and *Hunting phases* is, in the *Survival Phase* agents are not allowed to capture carnivores. While in the *Hunting Phase* this information is added to the environment.

It must be noted the activation of a new phase usually means the deactivation of pervious phase. Consider the example of the environment switching from *Grazing* to *Hunting Phase*. Fitness function of *Hunting Phase* does not take into account the number of food points captured by the particle, hence all food collection strategies will fail in the new phase. It is possible to form new phases using different combinations of these four phases. For example, *Grazing plus Awareness Phase*, where agents would have to collect food points while avoiding collisions. Fitness in such a case would be calculated using all the required task parameters, i.e., mean value of targets achieved and collisions avoided.

We normalize our fitness functions so that the fitness from one phase is comparable to the fitness achieved in another phase. For determining the max values agents are trained to learn a task using the learning algorithm. During this process the task remains unchanged. The best strategy found after this training phase is then tested for 100 simulations and the maximum, minimum, median and average values are noted. These values are then used for normalization and serve as threshold values (Table 1).

TABLE 1. Parameters for fitness normalization

t_a^m	220
t_c^m	50
t_d^m	300
t_h^m	100

TABLE 2. Parameters of environment for experiments

Agents	20
Targets	30
Carnivores	10

TABLE 3. Parameters for learning algorithm

P	32
c_1, c_2 & w	1
r_1, r_2	0–1
$v_{\max}$	0.05–1
S	8
T	20
N	20
E	300

4.4. **Learning experimentation.** We now test the ability of the learning algorithm. We create agents and place them in the environment. These agents must develop strategies that will improve their chances of survival. However, the environment adds a new layer of difficult by changing itself from one phase to another. Agents must detect this phase change and adapt accordingly. We do not claim that these agents have found the optimal strategies for each phase. Our focus is on their ability to learn new skills and acquire new objectives on their own. Tables 2 and 3 present parameters used in these experiments.

For our first experiment, we test the ability of the standard PSO approach. The environment changes from *Grazing* phase to *Hunting* results are shown in Figure 3. The agents were allowed to learn for 1,000 iterations in the environment according to the fitness function of the first task and then the task is changed. Since the PSO particles have their local best and personal best particle set according to the old task 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. Agent learning is plotted on y-axis and learning iterations are shown on x-axis.

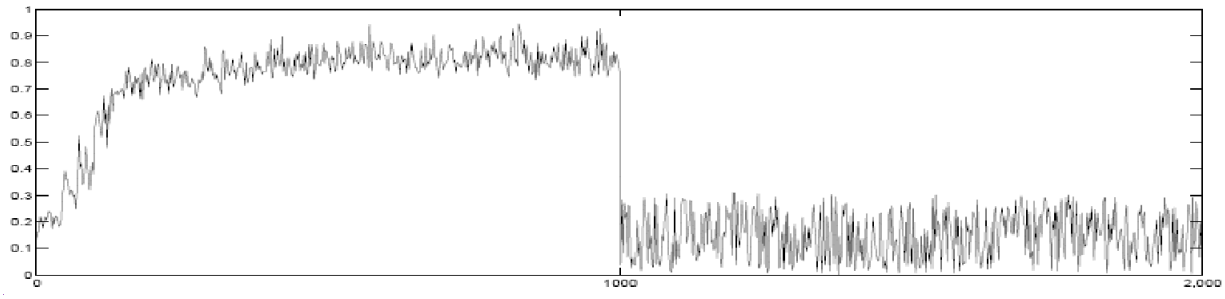

FIGURE 3. The results of non re-initialization of sub-swarms.

Figure 4 shows what happened if we explicitly inform the learners that the task has changed. The explicit information allows the learner to learn faster and achieve high fitness in relatively less number of iterations (compared to the previous case when no such information was available).

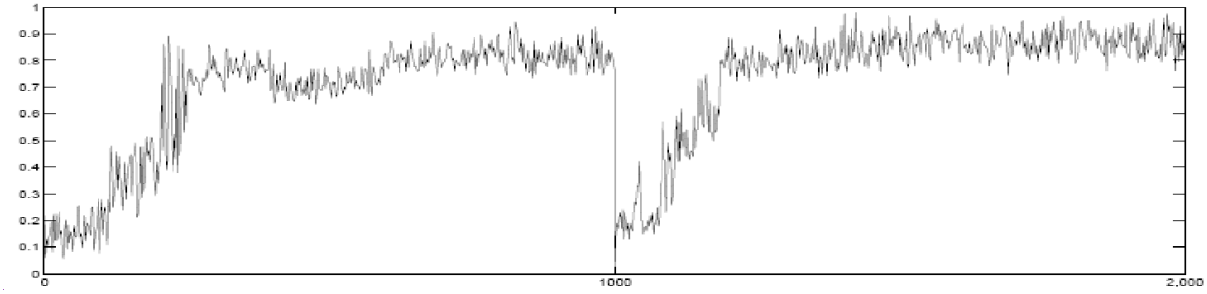

FIGURE 4. The results of explicit information about change.

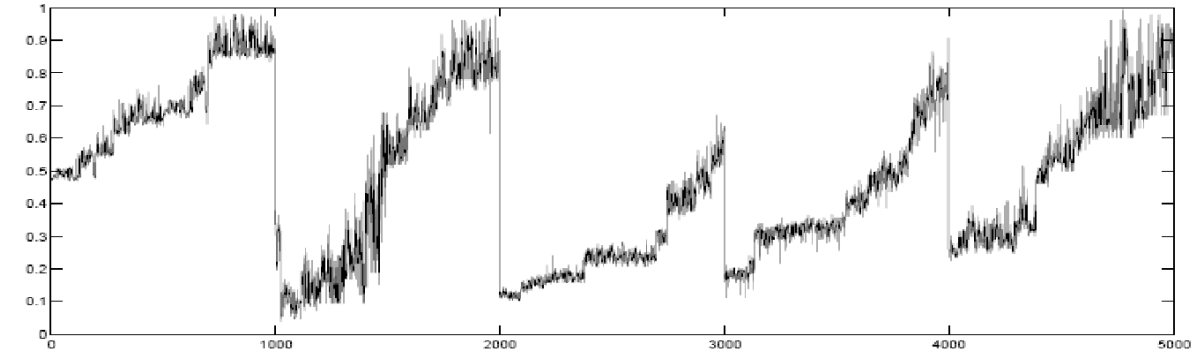

FIGURE 5. Learning in an uncertain and changing environment (without retaining historical learning).

We now test the ability of our learning algorithm in a changing and uncertain environment. For this experiment, we forced the environment to change phase after every 1000 iterations (agents are not aware of this fact). Environment starts off from *Grazing phase*; agents are allowed to learn food gathering strategies for 1,000 iterations. After this the environment switches to *Hunting phase*. After this the environment enters a new *Grazing plus Survival phase* where agents have to avoid being eaten by carnivores while

capturing food points. At the fourth phase change, the environment reverts to a *Hunting phase*. The final phase change enforces a *Grazing plus Awareness Phase*. Results of this experiment are presented in Figure 5. Iterations allowed for learning are on x-axis while y-axis shows the learning of the agents at that iteration.

It is evident from Figure 5 that after every phase change, agents have to abandon their previous strategies and learn new ones. These new strategies are developed to handle the new requirements of the environment. Figure 5 proves that our learning algorithm is able to learn in an uncertain and dynamic environment of imperfect information. Agents had to readjust their strategies according to the changes in their environment. The only information made available to the agents was the inputs available in the environment and the fitness evaluation of their strategy. Although we changed the iterations at a regular interval of 1000 iterations, the results for irregular intervals show a similar pattern. It must be noted that agents must be given enough time for learning to occur, e.g., in mentioned scenario if the environment changed after every 50 iterations the agents would not be able to explore enough of the search space hence failing to learn the task currently active in the environment. How much time (or iterations) is sufficient for an agent to perform successful learning is an open question and we believe its answer depends upon the complexity of the task.

We these results summarize that our learning algorithm is able to learn new strategies according to changes in environment and can handle task changes. It performs well where the classical PSO fails to do so showing 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. From our experimentation we observed that although the agents were successful in learning the new tasks assigned to them, in our approach they discarded the previously learned strategies [36]. This led to "re-inventing the wheel" like scenario where the agent had to learn a strategy forget it when the task changed and relearn it if the environment reverted back to its original state. In order to deal such a scenario we introduced the concept of retaining historical information (using archives).

Using this experiment, we now show the use of retaining historical information in an archive. These stored strategies improve the learning time needed in the changing task environment. To see how the system behaves with and without this historical information we test the learning ability of the algorithm with and without an archive. Table 4 present parameters of the archive used in this experiment.

TABLE 4. Parameters of archive

N_a	5
T_a	300
T_b	20

N_a are the total number of slots present in the archive, T_a is the number of iterations after which a strategy was selected to be stored in the archive and T_b is the number of iteration for determination of second-worst sub-swarm for trying a strategy from the archive.

We use the same environment settings and task sequence used for Figur 5. Note in the approach used in Figure 5 the archive was disabled.

To shorten the length of the experiment we filled the archive with the best strategies already found from previous experiments. At the start of the experiment, four of these slots have already been filled with strategies. The results of the experiment are shown in Figure 6.

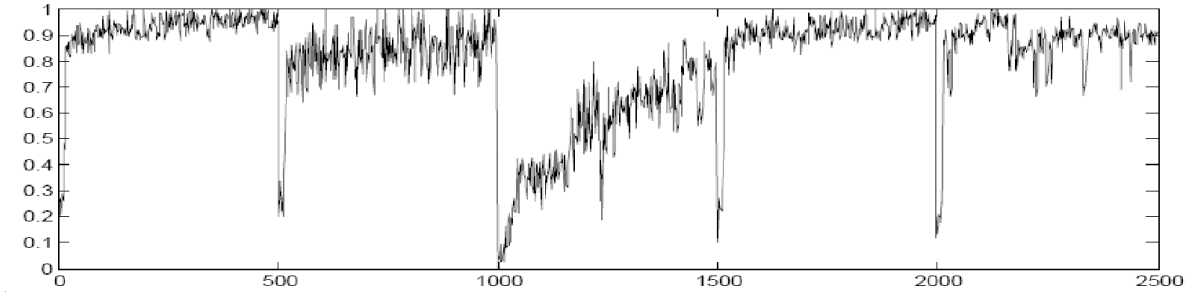

FIGURE 6. Learning algorithm with retaining historical learning.

Whenever a sub-swarm tries a strategy from the archive that matches the current state of the environment, the learning performance of the agents goes very high (evident from last two phase changes). This higher fitness position is disseminated to other sub-swarms and the fitness of the entire population increases in a fairly less time frame. However, If there are no strategies present in that archive that match a particular phase then learning resumes its normal course (evident from third phase change). We tested several random sequences of tasks (of arbitrary length) and results showed that whenever the environment reverts back to an earlier phase (i.e., a phase familiar to the agents due to their past experience). The presence of archive improves fitness dramatically and the learning spreads like wild-fire to all other sub-swarms.

5. **Conclusions.** Humans live in a dynamic world and have to cope with a series of challenges which are sometimes new and sometimes previous ones visiting again. They are forced to change their strategy as soon as the current strategy stops producing results. The changed strategy is sometimes a previously discovered one and sometimes a new one has to be learnt. 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 developed a PSO based learning algorithm which incorporates an archive, and test it in a gaming environment. 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. Half of the newly created agents are randomly initialized and half of them are resurrected by strategies taken from an archive. This continual learning is more coherent with the learning that takes place in our everyday environment.

Our experimental results show that our learning algorithm becomes more effective with the incorporation of an archive. The archive idea can be supplemented by finding a method for automated adjustment of parameters in three areas: the policy for placing strategies in the archive, the optimum size of the archive and the policy for retrieving strategies from the archive.

The gaming environment may also be made more interesting with 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] D. Goodman and R. Keene, *Man versus Machine: Kasparov versus Deep Blue*, MA: H3 Publications, Cambridge, 1997.

[2] J. Schaeffer, R. Lake and P. Lu, CHINOOK the world man-machine checkers champion, *AI Magazine*, vol.17, no.1, pp.21-30, 1996.
[3] J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers*, Springer-Verlag, New York, 1997.
[4] K. Chellapilla and D. B. Fogel, Evolving an expert checkers playing program without using human expertise, *IEEE Trans. Evol. Comput.*, vol.5, no.4, pp.422-428, 2001.
[5] D. B. Fogel, *Blondie24, Playing at the Edge of AI*, Morgan Kaufmann, San Mateo, 2002.
[6] K. O. Stanley, B. D. Bryant and R. Miikkulainen, Real-time neuroevolution in the NERO video game, *IEEE Trans. Evol. Comput.*, vol.9, no.6, pp.653-668, 2005.
[7] G. Kendall and Y. Su, Imperfect evolutionary systems, *IEEE Trans. Evol. Comput.*, vol.11, no.3, pp.294-307, 2007.
[8] T. Blackwell and J. Branke, Multiswarms, exclusion, and anti-convergence in dynamic environments, *IEEE Trans. Evol. Comput.*, vol.10, no.4, pp.459-472, 2006.
[9] D. Parrott and X. Li, Locating and tracking multiple dynamic optima by a particle swarm model using speciation, *IEEE Trans. Evol. Comput.*, vol.10, no.4, pp.440-458, 2006.
[10] R. G. Reynolds, *An Adaptive Computer Model of the Evolution of Agriculture for Hunter-Gatherers in the Valley of Oaxaca*, Ph.D. Thesis, Univ. Michigan, Ann Arbor, MI, 1979.
[11] R. G. Reynolds, An introduction to cultural algorithms, *Proc. of the 3rd Annu. Conf. Evol. Program.*, pp.131-139, 1994.
[12] M. Sternberg and R. G. Reynolds, Using cultural algorithms to support re-engineering of rule-based expert systems in dynamic environments: A case study in fraud detection, *IEEE Trans. Evol. Comput.*, vol.1, no.4, pp.225-243, 1997.
[13] R. G. Reynolds and S. Saleem, Function optimization with cultural algorithms in dynamic environments, *Proc. of IEEE Particle Swarm Optimization Workshop*, pp.63-79, 2001.
[14] R. G. Reynolds and S. Saleem, The impact of environmental dynamics on cultural emergence, in *Perspectives on Adaptation in Natural and Artificial Systems – Essays in Honor of John Holland*, L. Booker, S. Forrest, M. Mitchell and R. Riolo (eds.), London, Oxford Univ. Press, 2004.
[15] R. G. Reynolds and B. Peng, Cultural algorithms: Knowledge learning in dynamic environments, *Proc. of IEEE Int. Congr. Evol. Comput.*, pp.1751-1758, 2004.
[16] L. A. Zadeh, Fuzzy sets, *Information and Control*, vol.8, no.3, pp.338-353, 1965.
[17] L. A. Zadeh, Fuzzy sets as a basis for a theory of possibility, *Fuzzy Sets and Systems*, vol.1, pp.3-28, 1978.
[18] D. S. Weld and J. de Kleer, *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann, San Mateo, 1988.
[19] A. Newell and H. A. Simon, GPS, a program that simulates human thought, in *Computers and Thought*, E. A. Feigenbaum and J. Feldman (eds.), New York, McGraw-Hill, 1963.
[20] M. L. Minsky, *The Society of Mind*, Simon & Schuster, New York, 1986.
[21] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, 1988.
[22] X. Yao, *Evolutionary Computation – Theory and Applications*, World Scientific, Singapore, 1999.
[23] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd Edition, IEEE Press, Piscataway, 2000.
[24] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computation (Natural Computing Series)*, Springer, New York, 2003.
[25] K. Weng and L. Yao, Fuzzy modeling based on self-learning of adaptive ellipsoids, *ICIC Express Letters*, vol.3, no.4(A), pp.1043-1048, 2009.
[26] R. Mei, Q. Wu and C. Jiang, Neural network robust adaptive control for a class of time uncertain nonlinear systems, *International Journal of Innovative Computing, Information and Control*, vol.6, no.3(A), pp.931-940, 2010.
[27] C. Chen and P. Chen, GA-based adaptive neural network controllers for nonlinear systems, *International Journal of Innovative Computing, Information and Control*, vol.6, no.4, pp.1793-1803, 2010.
[28] J. C. R. Tseng, G. Hwang, P. Tsai and C. Tsai, Meta-analyzer: A web-based learning environment for analyzing student information searching behaviors, *International Journal of Innovative Computing, Information and Control*, vol.5, no.3, pp.567-580, 2009.
[29] C. M. Frayn, An evolutionary approach to strategies for the game of monopoly®, *Proc. of IEEE Symposium on Computational Intelligence and Games*, 2005.

[30] 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. Evol. Comput.*, vol.9, no.6, pp.707-720, 2005.
[31] 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, pp.382-396, 2007.
[32] 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, 2005.
[33] S. M. Lucas, Cellz: A simple dynamical game for testing evolutionary algorithms, *IEEE CEC*, vol.1, pp.1007-1014, 2004.
[34] 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, pp.280-288, 2004.
[35] N. Franken and A. Engelbrecht, Comparing PSO structures to learn the game of checkers from zero knowledge, *IEEE Congress on Evolutionary Computation*, 2003.
[36] Z. Zhang, S. Gao, G. Yang, F. Li and Z. Tang, An algorithm of supervised learning for elman neural network, *International Journal of Innovative Computing, Information and Control*, vol.5, no.10(A), pp.2997-3012, 2009.
[37] S. J. Louis and C. Miles, Playing to learn: Case-injected genetic algorithms for learning to play computer games, *IEEE Trans. on Evolutionary Computation*, vol.9, no.6, 2005.
[38] P. M. Avery, Z. Michalewicz and M. Schmidt, A historical population in a coevolutionary system, *Proc. of IEEE Symposium on Computational Intelligence and Games*, 2007.
[39] S. Zhao and P. N. Suganthan, Multi-objective evolutionary algorithm with ensemble of external archives, *International Journal of Innovative Computing, Information and Control*, vol.6, no.4, pp.1713-1726, 2010.
[40] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, John Willey & Sons, 2005.
[41] H. Mujtaba and A. R. Baig, Survival by continuous learning in a dynamic multiple task environment, *Proc. of IEEE Symposium on Computational Intelligence and Games*, 2008.
[42] H. Mujtaba and A. R. Baig, Retaining the lessons from past for better performance in a dynamic multiple task environment, *IEEE Congress on Evolutionary Computation*, 2008.