**Bulletin of Environment, Pharmacology and Life Sciences**
Bull. Env. Pharmacol. Life Sci., Vol 3 [6] May 2014: 115-127
©2014 Academy for Environment and Life Sciences, India
Online ISSN 2277-1808
Journal's URL:http://www.bepls.com
CODEN: BEPLAD
Global Impact Factor 0.533
Universal Impact Factor 0.9804

**ORIGINAL ARTICLE**

# New Approach to Optimization by simulating insect in wheat farms

**Fariborz Ahmadi[1], Hamid Salehi[2], seied Veria Hoseini[3]**
[1,2,3]Young Researcher and Elite Club, Ghorveh Branch, Islamic azad university, Ghorveh, Iran

**ABSTRACT**

*The aim of optimization is to determine the best-suited solution to a problem under a given set of constraints. Several researchers over the decades have come up with different solutions to linear and non-linear optimization problems. Mathematically an optimization problem involves a fitness function describing the problem, under a set of constraints representing the solution space for the problem. Unfortunately, most of the traditional optimization techniques are centered around evaluating the first derivatives to locate the optima on a given constrained surface. Because of the difficulties in evaluating the first derivatives, to locate the optima for many rough and discontinuous optimization surfaces, in recent times, several derivative free optimization algorithms have emerged. The optimization problem, now-a-days, is represented as an intelligent search problem, where one or more agents are employed to determine the optima on a search landscape, representing the constrained surface for the optimization problem. Almost all of the approach to solve NP-hard and NP-complete problem simulate artificial life. In this research, the behavior of eurygaster life is studied, so according to their life the new algorithm is introduced. In spite of PSO algorithm, that is used to solve continuous nonlinear functions, researchers' algorithm is so suitable to solve both continuous and discrete functions. Eurygasters attack to grain farms and distributed over them. It is worth to mention that these insects attack to farms in groups and each group colonize in one farm. It is observed that after periods of time all of the farms in a region are occupied by these groups of eurygasters. When each group of these insects are going to seek a farm to feed on it, they consider nearly all the farms and settles on a farm which have a lowest distance with them and doesn't have any group of eurygasters. It is clear that by distributing several groups of eurygasters, depending on the problem size, on search space of problem, the solution of the problem can be extracted. In this research, using the behavior of eurygasters, a new algorithm has been invented and has been tested on graph partitioning. The evaluation results show the advantage of researcher algorithm over ancient ones like genetic and PSO.*

***Keywords:*** *Evolutionary computation, genetic algorithm, particle swarm optimization, eurygaster algorithm, evolutionary programming.*

Received 10.01.2014 Revised 04.03.2014 Accepted 19.04. 2014

## INTRODUCTION

Nonlinear problems are important in many real life applications because of their intrinsic difficulty; recently the area has attracted much research with the advances in nature inspired heuristics and multi agent systems for these problems. The dramatic increase in the size of the search space and the need for real time solutions motivated research idea in solving NP-problems using nature inspired heuristic techniques.in order to solve NP-problems some method like genetic algorithm and particle swarm optimization has been invented. These algorithms are population based and use intelligent to converge to the solution of problems.

Genetic algorithms consist of a certain number of individuals in each generation. The individuals are called chromosomes and each one show one of the solutions of problem. In each generation the fitness of chromosomes are evaluated and crossover and mutation operators are applied to the selected chromosomes to generate new chromosomes or individuals (offspring) in the next generation.

Also particle swarm optimization [1, 2] was introduced by Kennedy and Eberthart. In PSO, Some social systems of natural species, such as bird flock and fish School, possess interesting collective behavior. In these systems, globally sophisticated behavior emerges from local, indirect communication among simple agents with only limited capabilities. In an attempt to simulate this flocking behavior by computers, Kennedy and Eberthart (1995) realized that an optimization problem can be formulated as that of a flock of bird's flying across an area seeking a location with abundant food. This observation, together with some abstraction and modification techniques, led to the development of a novel optimization technique – particle swarm optimization.

To solve NP-problem some other algorithm such as ant colony [3], simulated annealing [4], tabu search [5], honey bee algorithm [6], photosynthetic algorithm [7], enzyme algorithm [8], glowworm swarm optimization [9], monkey search [10], firefly algorithm[11] were also introduced. These algorithms simulate behavior of artificial life of species.

In this work, researchers have invented new method called eurygaster algorithm based on eurygaster life. This algorithm is suitable for both continuous and discrete NP-problems and also, special for divide and conquer problems. In researcher approach, to find the solution of problem, problem must be divided in to some categories. After that, each cattle of eurygasters scatters over one of these categories to find the solution in search space of the category. In case of not finding the solution, new eurygasters are produced and distributed on the other categories and in case of finding the solution, the algorithm is finished. This method is easy to implement and can be simulated by a few line of computer code and is computationally inexpensive in term of memory and speed. Despite PSO that is used for continuous NP-problem, this algorithm is proposed for both continuous and discrete NP-problems. In other hand, in PSO algorithm [1,2], particles go through the search space of problem at each time continuously, while in research approach a group of eurygasters change their presentations and locations according to new category in which they colonize. In researcher approach, the search space of problems must be split on several groups or categories so that a great number of eurygasters disperse on each group to exhaustedly consider its space to find the best solution in that group.

The reminder of this paper is divided into 5 sections. At first in section 2, the characteristics of previous work like PSO and GA are detailed. After that, in section 3 behavior of eurygaster is described. Then, in section 4, our method or eurygaster algorithm is elaborated. Finally, section 5, 6 shows the evolution results of our method on graph partitioning [12] and draw some conclusion, respectively.

## Previous works

PSO and GA have been interested by researchers in the area of heuristic and soft computing methods. In this paper the characteristics of both of them are described and finally, we compare our approach with these methods to show the efficiency of our approach over them.

### Overview of PSO

Particle swarm optimization (PSO) is an algorithm modeled on swarm intelligence, that finds a solution to an optimization problem in a search space, or model and predict social behavior in the presence of objectives. The PSO is a stochastic, population-based computer algorithm modeled on swarm intelligence. Swarm intelligence is based on social-psychological principles and provides insights into social behavior, as well as contributing to engineering applications. The particle swarm optimization algorithm was first described in 1995 by James Kennedy and Russell C. Eberhart [1, 2]. By adding a new inertia weight into PSO, a new version of PSO is introduced in [13].

The particle swarm simulates a kind of social optimization. A problem is given, and some way to evaluate a proposed solution to it exists in the form of a fitness function. A communication structure or social network is also defined, assigning neighbors for each individual to interact with. Then a population of individuals defined as random guesses at the problem solutions is initialized. These individuals are candidate solutions. They are also known as the particles, hence the name particle swarm. An iterative process to improve these candidate solutions is set in motion. The particles iteratively evaluate the fitness of the candidate solutions and remember the location where they had their best success. The individual's best solution is called the particle best or the local best. Each particle makes this information available to their neighbors [14]. They are also able to see where their neighbors have had success. Movements through the search space are guided by these successes, with the population usually converging, by the end of a trial, on a problem solution better than that of non-swarm approach using the same methods. Each particle represents a candidate solution to the optimization problem. The position of a particle is influenced by the best position visited by itself i.e. its own experience and the position of the best particle in its neighborhood i.e. the experience of neighboring particles. When the neighborhood of a particle is the entire swarm, the best position in the neighborhood is referred to as the global best particle, and the resulting algorithm is referred to as the gbest PSO. When smaller neighborhoods are used, the algorithm is generally referred to as the pbest PSO. The performance of each particle is measured using a fitness function that varies depending on the optimization problem. Each Particle in the swarm is represented by the following characteristics:

- The current position of the particle
- The current velocity of the particle

The particle swarm optimization which is one of the latest evolutionary optimization techniques conducts searches uses a population of particles. Each particle corresponds to individual in evolutionary algorithm.

Each particle has an updating position vector and updating velocity vector by moving through the problem space.

The PSO parameters

Since the introduction of the PSO several variations have been presented. Here we will only explain in detail how the original one works and then summarize two variants. For more in-depth information on the variants please see references [1, 13]. Remember that the main concept is that we have particles of a swarm moving in a problem space and evaluating their positions through a fitness function. Once a problem space is defined, a set of particles is spawned in it and their positions and velocities are updated iteratively according to the specific PSO algorithm. Even though PSO has been proven to be an efficient algorithm with good results, it is not by design one that guarantees that the best solution is found, since it relies on visiting and evaluating problem space positions.

In PSO, instead of using genetic operators, each particle (individual) adjusts its "flying" according to its own flying experience and its companions' flying experience. Each particle is treated as a point in a D-dimensional and m particles form the colony. The $i_{th}$ particle is represented by a *D*-dimensional $Xi$ ($i$ = 1, 2,..., $m$) vector which means that the particle locates at $Xi = (xi1, xi2, ..., xiD)$ ($i$ = 1, 2,..., $m$) in the search space. The rate of the position change (velocity) for particle $i$ is represented as $V_I = (v_{i1}, v_{i2}, ..., v_{iD})$ ($i$ = 1, 2,..., $m$). The best previous position (the position giving the best fitness value) of the $i_{th}$ particle is recorded and represented as $P_I = (p_{i1}, p_{i2}, ..., p_{iD})$ ($i$ = 1, 2,..., $m$). The index of the best particle among all the particles in the population is represented by the $Pg = (pg1, pg2, ..., p_{gD})$ ($i$ = 1, 2,..., $m$). The particles are manipulated according to the following equation:

$$v_{id}(t+1) = w * v_{id}(t) + (c_1 * rand() * (p_{id}(t) - x_{id}(t))) + (c_2 * Rand() * (p_{gd}(t) - x_{id}(t))) \qquad (1)$$

$$x_{id}(t) = x_{id}(t) + v_{id}(t+1) \qquad (2)$$

Where $c_1$ and $c_2$ are two positive constants, rand () and Rand () are two random functions in the range (0, 1), and w is the inertia weight. Besides, a maximum allowable velocity vector $V_{max}$ clamps velocities of particles on each dimension. Equation (1) is used to calculate the particle's new velocity according to its previous velocity and the distances of its current position from its own best experience (position) and the group's best experience. Then the particle flies toward a new position according to equation (2). The performance of each particle is measured according to a pre-defined fitness function, which is related to the problem to be solved. The inertia weight w is employed to control the impact of the previous history of velocities on the current velocity, thus to influence the trade-off between global (wide-ranging) and local (nearby) exploration abilities of the "flying points".

A larger inertia weight w facilitates global exploration (searching new areas) while a smaller inertia weight tends to facilitate local exploration to fine-tune the current search area. Suitable selection of the inertia weight w can provide a balance between global and local exploration abilities and thus require less iteration on average to find the optimum. In [13], an analysis of the impact of this inertia weight together with the maximum velocity allowed on the performance of PSO is given, followed by experiments that illustrate the analysis and provide some insights into optimal selection of the inertia weight and maximum velocity allowed.

**Global best algorithm (original version)**

In PSO algorithm, we have a completely connected swarm, meaning that all the particles share information, any particle knows what the best position is ever visited by any particle in the swarm. The basic pseudo code for the classic PSO algorithm taken from [15] is:

```
1. Initialize a population (array) or particle with random positions and velocities and on d
   dimensions in the problem space.

2. Evaluate – compute fitness of each particle in swarm

3. Do for each particle in swarm

  3.1. Find particle best (pbest) – compute fitness of particle.
           If (current pbest < pbest)
               Pbest = current pbest
                Pbest location = current location
```

```
            End if
    3.2. Find global best (gbest) – best fitness of all pbest    gbest location = location of min (all pbest)

    3.3. Update velocity of particle per equation (1)

    3.4. Update position of particle per equation (2)

 4. Repeat steps 3.1 through 3.4 until termination conditions are reached.
```

Algorithm. 1. PSO algorithm

Regarding velocity update (equation (1)) notice that t results from the sum of different components, each having a specific meaning. On the first point, we have the momentum component, which is the previous velocity. On the second point we have the cognitive component, which depends heavily on the particle's current distance to the best position t has ever visited. Finally on the third point we have the social component which depends entirely on the particle's distance to the best position where any of the swarm's particles has ever been.
Since these update functions are recursive in the sense that they need the previous values of velocity and position, it is important to understand how these values are initialized. Moreover, the way in which these values are initialized can have an important impact on the algorithm's performance.
Regarding position, all the particles in the Swarm are positioned in the D-dimensional space by distributing them as desired within the search space, two common ways to do it are randomly or uniformly position them. The velocity is usually set to a random value, but lower ones are usually more adequate to avoid large initial offsets. The social and cognitive component scale values are also determined at initialization time since they are constant throughout the optimization process.
In algorithm 1, it's presented the pseudo-code of the basic PSO algorithm. In some parts which aren't fully detailed in the original presentation of PSO, it's given a simple solution, but several alternatives exist and each of them does have different impacts on the performance of the algorithm.
With this algorithm the main parameterization needed regards the acceleration multipliers and the maximum velocity (velocity clamping) that even though not referenced in the original presentation of the algorithm, when the local version was introduced by the same author, he mentions it for both versions as a way to limit particles flying out of the search space [1,2]. You might notice that you also have to specify the bounds of the search space and the maximum number of iterations for the algorithm to run, but the bounds should be derived directly from the problem being modeled and the number of iterations can be easily set for any reasonable value presented in the literature like in the modified PSO [13].

## Analysis of PSO

PSO, to some extent, resembles evolutionary programming. The addition of velocity to the current position to generate the next position is similar to the mutation operation in evolutionary programming except that "mutation" in PSO is guided by a particle's own "flying" experience and the group's "flying" experience. In another words, PSO performs "mutation" with a "conscience". By looking at the personal best elements associated with each individual as additional population members, PSO also has a form of selection even though it is quite weak.
In evolutionary programming, the global and local exploration abilities are brought in by the mutation and controlled by the variances of the Gaussian random functions employed. In order to balance between the global and local exploration abilities and obtain a quick search, the variances can also be encoded into the individuals and therefore evolved simultaneously, as done in evolutionary strategy [16]. In PSO the balance between the global and local exploration abilities is mainly controlled by the inertia weights.
The parametric study on coefficients(w,c1 and c2) for terms $v_{id}(t)$,$(p_{id}(t) - x_{id}(t))$ and $(p_{gd}(t) - x_{id}(t))$ respectively, were conducted in [17,18] and empirical studies revealed that in equation (1), the w value should be about 0.7 to 0.8,and c1 and c2 around 1.5 to 1.7. Irrespective of the choice of w, c1 and c2, while working with the bounded spaces the velocity expression often causes particles to 'fly-out' of the search space. To control this problem, a velocity clamping mechanism was suggested in [19]. The clamping mechanism restricted the velocity component in $[-v_{max,i}, v_{max,i}]$ along each dimension (i). Usually, $v_{max,i}$ along $i_{th}$ dimension is taken as 0.1 to 1.0 times the maximum value of x along the dimension. However, the clamping does not necessarily ensure that particles remain in the search space. It is worth noting that in unbounded search spaces such a mechanism cannot be employed [20].
By looking at equation (1) more closely, it can be seen that the maximum velocity allowed actually serves as a constraint that controls the maximum global exploration ability that PSO can have. By setting a too small maximum velocity allowed, maximum global exploration ability is limited, and PSO will always favor a local search no matter what the inertia weight is. By setting a large maximum velocity allowed, the PSO can have

a large range of exploration ability by selecting the inertia weight. Since the maximum velocity allowed affects global exploration ability indirectly and the inertia weight affects it directly, it will generally be better to control global exploration ability through inertia weight only. A way to do that is to delete maximum velocity allowed in the algorithm implementation, and allow inertia weight itself to control exploration ability. But this should be done very carefully. It's not a good idea for PSO to do global exploration all the time because this will mean that the system will always be eager to explore new areas and consequently make the system lack local exploration ability, and fail to find the solution. From the above, it's clear that choosing a large inertia weight to facilitate more global exploration is not a good strategy, and a smaller inertia weight should be selected to achieve a balance between global and local exploration so that a faster search results.

**Advantages and Disadvantages of the PSO Algorithm**

Advantages of the basic particle swarm optimization algorithm [21]:

- PSO is based on the intelligence. It can be applied into both scientific research and engineering use.
- PSO have no overlapping and mutation calculation. The search can be carried out by the speed of the particle. During the development of several generations, only the most optimist particle can transmit information onto the other particles, and the speed of the researching is very fast.
- The calculation in PSO is very simple. Compared with the other developing calculations, it occupies the bigger optimization ability and it can be completed easily.
- PSO adopts the real number code, and it is decided directly by the solution. The number of the dimension is equal to the constant of the solution.

Disadvantages of the basic particle swarm optimization algorithm [21]:

The method easily suffers from the partial optimism, which causes the less exact at the regulation of its speed and the direction.

- The method cannot work out the problems of scattering and optimization.
- The method cannot work out the problems of non-coordinate system, such as the solution to the energy field and the moving rules of the particles in the energy field.

**PSO applications**

Particle swarm optimization can be and has been used across a wide range of applications. Areas where PSOs have shown particular promise include multimodal problems and problems for which there is no specialized method available or all specialized methods give unsatisfactory results. A first look into the applications of this algorithm was presented by one of its creators, Eberhart and Shi in 2001 [22] focusing on application areas such as: Artificial Neural Networks training (for Parkinson Diagnostic), Control Strategy determination (for electricity management) and Ingredient Mix Optimization (for microorganisms' strains growth). Later in 2007 a survey by Riccardo Poli [23] reported the exponential growth in applications and identified around 700 hundred documented works on PSO.

**Overview of genetic algorithm**

GA is inspired from the natural selection of Darwin to find the solution of the problems. It is worthwhile to introduce some of the biological terms before explaining algorithm in details.

**Biological terminology**

At this point it is useful to formally introduce some of the biological terminology that will be used throughout this research. In the context of genetic algorithms, these biological terms are used in the spirit of analogy with real biology, though the entities they refer to are much simpler than the real biological ones.

All living organisms consist of cells, and each cell contains the same set of one or more chromosomes—strings of DNA—that serve as a "blueprint" for the organism. A chromosome can be conceptually divided into genes— each of which encodes a particular protein. Very roughly, one can think of a gene as encoding a trait, such as eye color. The different possible "settings" for a trait (e.g., blue, brown, and hazel) are called alleles. Each gene is located at a particular locus (position) on the chromosome.

Many organisms have multiple chromosomes in each cell. The complete collection of genetic material (all chromosomes taken together) is called the organism's genome. The term genotype refers to the particular set of genes contained in a genome. Two individuals that have identical genomes are said to have the same genotype. The genotype gives rise, under fetal and later development, to the organism's phenotype—its physical and mental characteristics, such as eye color, height, brain size, and intelligence. Organisms whose chromosomes are arrayed in pairs are called diploid; organisms whose chromosomes are unpaired are called haploid. In nature, most sexually reproducing species are diploid, including human beings, who each have 23 pairs of chromosomes in each somatic (non– germ) cell in the body. During sexual reproduction, recombination (or crossover) occurs: in each parent, genes are exchanged between each pair of chromosomes to form a gamete (a single chromosome), and then gametes from the two parents pair up to create a full set of diploid chromosomes. In haploid sexual reproduction, genes are exchanged between the two parents' single- strand chromosomes. Offspring are subject to mutation, in which single nucleotides (elementary bits of DNA) are changed from parent to offspring, the changes often resulting from copying errors. The fitness of an organism is typically defined as the probability that the organism will live to reproduce (viability) or as a function of the number of offspring the organism has (fertility). In genetic algorithms, the term chromosome typically refers to a candidate solution to a problem, often encoded as a bit string. The "genes" are either single bits or short blocks of adjacent bits that encode a particular element of the candidate solution (e.g., in the context of multi parameter function optimization the bits encoding a particular parameter might be considered to be a gene). An allele in a bit string is either 0 or 1; for larger alphabets more alleles are possible at each locus. Crossover typically consists of exchanging genetic material between two single chromosome haploid parents. Mutation consists of flipping the bit at a randomly chosen locus (or, for larger alphabets, replacing the symbol at a randomly chosen locus with a randomly chosen new symbol). Most applications of genetic algorithms employ haploid individuals, particularly, single- chromosome individuals. The genotype of an individual in a GA using bit strings is simply the configuration of bits in that individual's chromosome. Often there is no notion of "phenotype" in the context of GAs, although more recently many workers have experimented with GAs in which there is both a genotypic level and a phenotypic level (e.g., the bit- string encoding of a neural network and the neural network itself).

**Search spaces and fitness landscapes**

The idea of searching among a collection of candidate solutions for a desired solution is so common in computer science that it has been given its own name: searching in a "search space." Here the term "search space" refers to some collection of candidate solutions to a problem and some notion of "distance" between candidate solutions. For an example, let us take one of the most important problems in computational bioengineering: the aforementioned problem of computational protein design. Suppose you want use a computer to search for a protein—a sequence of amino acids—that folds up to a particular three-dimensional shape so it can be used to fight a specific virus. The search space is the collection of all possible protein sequences—an infinite set of possibilities. To constrain it, let us restrict the search to all possible sequences of length 100 or less—still a huge search space, since there are 20 possible amino acids at each position in the sequence [24]. If we represent the 20 amino acids by letters of the alphabet, candidate solutions will look like this:

A G G M C G B L....

We will define the distance between two sequences as the number of positions in which the letters at corresponding positions differ. For example, the distance between A G G M C G B L and M G G M C G B L is 1, and the distance between A G G M C G B L and L B M P A F G A is 8. An algorithm for searching this space is a method for choosing which candidate solutions to test at each stage of the search. In most cases the next candidate solution(s) to be tested will depend on the results of testing previous sequences; most useful algorithms assume that there will be some correlation between the qualities of "neighboring" candidate solutions—those close in the space [25]. Genetic algorithms assume that high-quality "parent" candidate solutions from different regions in the space can be combined via crossover to, on occasion, produce high-quality "offspring" candidate solutions.

Another important concept is that of "fitness landscape." Originally defined by the biologist Sewell Wright (1931) in the context of population genetics, a fitness landscape is a representation of the space of all possible genotypes along with their fitnesses. Suppose, for the sake of simplicity, each genotype is a bit string of length $l$, and that the distance between two genotypes is their "Hamming distance"—the numbers of locations at which corresponding bits differ. Also suppose that each genotype can be assigned real- valued fitness. A fitness landscape can be pictured as an $(l + 1)$ -dimensional plot in which each genotype is a point in $l$ dimensions and its fitness is plotted along the (l +1) axis. Such plots are called landscapes because the plot of fitness values can form "hills," "peaks," "valleys," and other features analogous to those of physical landscapes. Under Wright's formulation, evolution causes populations to move along landscapes in particular ways, and "adaptation" can be seen as the movement toward local peaks [25]. (A "local peak," or "local optimum," is not necessarily the highest point in the landscape, but any small movement away from it goes downward in fitness.) Likewise, in GAs the operators of crossover and mutation can be seen as ways of moving a population around on the landscape defined by the fitness function. The idea of evolution moving populations around in unchanging landscapes is biologically unrealistic for several reasons. For example, an organism cannot be assigned a fitness value independent of the other organisms in its environment; thus, as the population changes, the fitnesses of particular genotypes will change as well. In other words, in the real world the "landscape" cannot be separated from the organisms that inhabit it. In spite of such caveats, the notion of fitness landscape has become central to the study of genetic algorithms.

## Elements of genetic algorithms

It turns out that there is no rigorous definition of "genetic algorithm" accepted by all in the evolutionary–computation community that differentiates GAs from other evolutionary computation methods. However, it can be said that most methods called "GAs" have at least the following elements in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring. Inversion—Holland's fourth element of GAs—is rarely used in today's implementations, and its advantages, if any, are not well established [26]. The chromosomes in a GA population typically take the form of bit strings. Each locus in the chromosome has two possible alleles: 0 and 1. Each chromosome can be thought of as a point in the search space of candidate solutions. The GA processes populations of chromosomes, successively replacing one such population with another. The GA most often requires a fitness function that assigns a score (fitness) to each chromosome in the current population. The fitness of a chromosome depends on how well that chromosome solves the problem at hand [26].

## Examples of Fitness Functions

One common application of GAs is function optimization, where the goal is to find a set of parameter values that maximize, say, a complex multiparameter function. As a simple example, one might want to maximize the real–valued one–dimensional function

$F(y) = y + |\sin(32y)| \quad 0<y<\prod$.

Here the candidate solutions are values of *y*, which can be encoded as bit strings representing real numbers. The fitness calculation translates a given bit string *x* into a real number *y* and then evaluates the function at that value. The fitness of a string is the function value at that point. As a non–numerical example, consider the problem of finding a sequence of 50 amino acids that will fold to a desired three–dimensional protein structure. A GA could be applied to this problem by searching a population of candidate solutions, each encoded as a 50–letter string such as IHCCVASASDMIKPVFTVASYLKNWTKAKGPNFEICISGRTPYWDNFPGI, where each letter represents one of 20 possible amino acids. One way to define the fitness of a candidate sequence is as the negative of the potential energy of the sequence with respect to the desired structure. The potential energy is a measure of how much physical resistance the sequence would put up if forced to be folded into the desired structure—the lower the potential energy, the higher the fitness. Of course one would not want to physically force every sequence in the population into the desired structure and measure its resistance—this would be very difficult, if not impossible. Instead, given a sequence and a desired structure (and knowing some of the relevant biophysics), one can estimate the potential energy by calculating some of the forces acting on each amino acid, so the whole fitness calculation can be done computationally. These examples show two different contexts in which candidate solutions to a problem are encoded as abstract chromosomes encoded as strings of symbols, with fitness functions defined on the resulting space of strings. A genetic algorithm is a method for searching such fitness landscapes for highly fit strings.

## GA Operators

The simplest form of genetic algorithm involves three types of operators: selection, crossover (single point), and mutation [26].

***Selection:*** This operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce [27].

***Crossover*** This operator randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring 10011111 and 11100100. The crossover operator roughly mimics biological recombination between two single–chromosome (haploid) organisms [29].

***Mutation*** This operator randomly flips some of the bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001) [27].

## Simple genetic algorithm

Given a clearly defined problem to be solved and a bit string representation for candidate solutions, a simple GA works as follows:

1. Start with a randomly generated population of *n l*–bit chromosomes (candidate solutions to a problem).
2. Calculate the fitness $f(x)$ of each chromosome *x* in the population.
3. Repeat the following steps until *n* offspring have been created:
   3.1 Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can be selected more than once to become a parent.

3.2 With probability *p*c (the "crossover probability" or "crossover rate"), cross over the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents. (Note that here the crossover rate is defined to be the probability that two parents will cross over in a single point. There are also "multi–point crossover" versions of the GA in which the crossover rate for a pair of parents is the number of points at which a crossover takes place.)

3.3 Mutate the two offspring at each locus with probability *p*m (the mutation probability or mutation rate), and place the resulting chromosomes in the new population. If *n* is odd, one new population member can be discarded at random.

4. Replace the current population with the new population.
5. Go to step 2.

Algorithm. 2. Genetic algorithm

Each iteration of this process is called a *generation*. A GA is typically iterated for anywhere from 50 to 500 or more generations. The entire set of generations is called a *run*. At the end of a run there are often one or more highly fit chromosomes in the population. Since randomness plays a large role in each run, two runs with different random–number seeds will generally produce different detailed behaviors. GA researchers often report statistics (such as the best fitness found in a run and the generation at which the individual with that best fitness was discovered) averaged over many different runs of the GA on the same problem [27].

The simple procedure just described is the basis for most applications of GAs. There are a number of details to fill in, such as the size of the population and the probabilities of crossover and mutation, and the success of the algorithm often depends greatly on these details [27]. There are also more complicated versions of GAs (e.g., Gas that work on representations other than strings or GAs that have different types of crossover and mutation operators).

As a more detailed example of a simple GA, suppose that *l* (string length) is 8, that *f*(*x*) is equal to the number of ones in bit string *x* (an extremely simple fitness function, used here only for illustrative purposes), that *n* (the population size) is 4, that *p*c = 0.7, and that *p*m = 0.001. (Like the fitness function, these values of *l* and *n* were chosen for simplicity. More typical values of *l* and *n* are in the range 50–1000. The values given for *p*c and *p*m are fairly typical.).

The initial (randomly generated) population might look like this:

Table1. Initial population of genetic algorithm

| *Chromosome label* | *Chromosome string* | *Fitness* |
|---|---|---|
| *A* | *00000110* | *2* |
| *B* | *11101110* | *6* |
| *C* | *00100000* | *1* |
| *D* | *00110100* | *3* |

A common selection method in GAs is fitness–proportionate selection, in which the number of times an individual is expected to reproduce is equal to its fitness divided by the average of finesses in the population. A simple method of implementing fitness–proportionate selection is "roulette–wheel sampling"[29], which is conceptually equivalent to giving each individual a slice of a circular roulette wheel equal in area to the individual's fitness. The roulette wheel is spun, the ball comes to rest on one wedge–shaped slice, and the corresponding individual is selected. In the *n* = 4 example above, the roulette wheel would be spun four times; the first two spins might choose chromosomes B and D to be parents, and the second two spins might choose chromosomes B and C to be parents. (The fact that A might not be selected is just the luck of the draw. If the roulette wheel were spun many times, the average results would be closer to the expected values.).

Once a pair of parents is selected, with probability *p*c they cross over to form two offspring. If they do not cross over, then the offspring are exact copies of each parent. Suppose, in the example above, that parents B and D cross over after the first bit position to form offspring E = 10110100 and F = 01101110, and parents B and C do not cross over, instead forming offspring that are exact copies of B and C. Next, each offspring is subject to mutation at each locus with probability *p*m. For example, suppose offspring E is mutated at the sixth locus to form E' = 10110000, offspring F and C are not mutated at all, and offspring B is mutated at the first locus to form B' = 01101110. The new population will be the following:

Table2. Intermediate population of genetic algorithm

| *Chromosome label* | *Chromosome string* | *fitness* |
|---|---|---|
| *E'* | *10110000* | *3* |
| *F* | *01101110* | *5* |
| *C* | *00100000* | *1* |
| *B'* | *01101110* | *5* |

Note that, in the new population, although the best string (the one with fitness 6) was lost, the average fitness rose from 12/4 to 14/4. Iterating this procedure will eventually result in a string with all ones.

**Genetic algorithm applications**

The GA literature describes a large number of successful applications, but there are also many cases in which GAs perform poorly. There is no rigorous answer, though many researchers share the intuitions that if the space to be searched is large, is known not to be perfectly smooth and unimodal (i.e., consists of a single smooth "hill"), or is not well understood, or if the fitness function is noisy, and if the task does not require a global optimum to be found—i.e., if quickly finding a sufficiently good solution is enough—a GA will have a good chance of being competitive with or surpassing other "weak" methods (methods that do not use domain–specific knowledge in their search procedure). If a space is not large, then it can be searched exhaustively, and one can be sure that the best possible solution has been found, whereas a GA might converge on a local optimum rather than on the globally best solution. If the space is smooth or unimodal, a gradient–ascent algorithm such as steep1est–ascent hill climbing will be much more efficient than a GA in exploiting the space's smoothness. If the space is well understood (as is the space for the well–known Traveling Salesman problem, for example), search methods using domain–specific heuristics can often be designed to outperform any general–purpose method such as a GA. If the fitness function is noisy (e.g., if it involves taking error–prone measurements from a real–world process such as the vision system of a robot), a one–candidate–solution–at–a–time search method such as simple hill climbing might be irrecoverably led astray by the noise, but GAs, since they work by accumulating fitness statistics over many generations, are thought to perform robustly in the presence of small amounts of noise. These intuitions, of course, do not rigorously predict when a GA will be an effective search procedure competitive with other procedures. A GA's performance will depend very much on details such as the method for encoding candidate solutions, the operators, the parameter settings, and the particular criterion for success [28].

**EURYGASTER BEHAVIORS**

Eurygaster integriceps is an insect pest that predominantly attacks grains, feeding on the leaves, stems and grains, reducing yield and injecting a toxin into the grains which adds a foul smell to the resulting flour, and substantially reduces the baking quality of the dough.

In winters eurygasters live under the plants and bushes in hillside, in several numbers and make a group. At the end of winter and at the beginning of spring when it gets warmer, these insects end their winter sleeps and get ready to move and fly to grain fields by moving over the high mountains and leaving the nests in groups. The first group by the use of its instinct finds the best and the nearest grain fields and stays there. Getting there, this group of insect sends signals to the air to show the other groups their being there. Based on the number of eurygasters in a place, the strength of signals will be different. If the number of eurygasters in a grain field is not great, the rate of diffused signals will be little and if the number of eurygasters in a grain fields is greater, the rate of diffused signals will be increased. They diffuse these signals to show the others that reside there. So that the other groups of eurygasters understand that they should not close to the grain field which contains the first group. Of course the other groups based on diffused signals by the first group and the strength of these signals they decide if they can land and stay there or not. If the power of diffused signals is low, it means that some of the other groups of eurygasters can land and stay by the other groups which are resident there and began to eat. While the strength of the signals in the sky is high, it means that the other groups cannot land on the field(s) containing eurygasters, and they must fly to other fields in which there are no eurygasters, to live and eat.

According to the passage mentioned above, the next group of eurygasters while flying from their nests to other fields to find the best grain fields searches the best and closest ones to land and eat based on the broadcasted signals by landed group(s). This process will continue until they will find a suitable and useful grain field to eat.

We conclude that all the grain fields in a wide area will be attacked by eurygasters, because they do not gather in a one place.so, when there is not enough food in a grain field in which the eurygasters have stayed for a time, they will fly to a new field with no eurygasters according to the process mentioned above.

## EURYGASTER ALGORITHMS

In this section, proposed approach or eurygaster algorithm is described. Solving non-linear functions are so necessary in real life today and recently researchers interested in inventing methods to solve them. Thus, our approach contributes to solve NP-class problems. The great advantage of this algorithm is that it's so easy to implement and is also inexpensive in term of memory and speed. The second advantage of this algorithm is its convergence speed compared to other methods like GA and PSO.

In spite of GA that traps in local optimum, the evaluation results of this approach show this method doesn't have this drawback. The major disadvantage of this method is that it can be used only for dividable problems. So the problems must be divided into some groups to be solved by researchers approach. Despite this problem, the algorithm advantage clearly outweighs over its disadvantage.

As it was mentioned, eurygasters attack wheat fields in groups. When the new generation is produced, they attack the fields which haven't been attacked before. After some period of times, it is revealed that all the fields in a region have been attacked totally by eurygasters. The main point inspired by eurygasters behaviors is that if we can divide the problems into some partitions by producing element named eurygasters in each partition all the space of the problem can be searched. The main point in solving problems by this algorithm is the way in which the problem can be divided into some partitions or how to the problem can be partitioned. That is, the better the partitioning is done, the more accurate the solutions to the problem can be concluded.

Now suppose that the problem has been divided into n partitions. First several eurygasters based on the size of the problems partitions are generated and distributed in the space of the first part to find the solution of the problem. It should be mentioned that if the number of eurygasters is not enough to exhaustedly cover all the space of the related partition, we can search all the related space by redistributing them and changing their position so that they can cover all the space mentioned. This process in genetic algorithms is done through mutation [24]. Using mutation in the proposed approach prevents local optimum. After searching for the first partition and in case of not finding the optimal solution to the problem, new kinds of eurygaster relevant to the second part are produced in order to be used to search in the space of the second partition. This operation is continued up to the n-part and in case of finding the expected solution in each part, the related algorithm is terminated and the solution of the problem is reported. The related semi-code of the proposed algorithm is as follows:

```
I← the number of clusters

   While I <> 0 do
  1.  Initialization:  produce euragasters or particles according to characteristic of one partition
  2.  Distribution: distribute eurygasters on the regions of the partition
  3.  Evaluation: evaluate suitability of each eurygaster or particle depend on the problem
    3.1  If the suitable result of the partition is not obtained
         3.1.1.  Change the position of Eurygasters in the partition
         3.1.2.  goto 3
    3.2  If the result of the problem is not obtained
         3.2.1.  I--
         3.2.2.  goto 1
        Else
         3.2.3.  Stop algorithm or break
  End while
  4.  Report the solution of the problem
```

Algorithm. 3 .Eurygaster algorithm

In proposed algorithm by researchers, namely eurygaster algorithm, the partitions must be created before execution of the algorithm. At first, in *initialization* phase a set of the eurygasters are produced and scattered on the search space of relevant partition by *distribution* phase. After that the suitability of each eurygaster is computed according to the problem kind as you can see in evaluation phase. In case of finding the solution of the problem in any of the partition, there may be two possibilities.

1. The found solution is optimal solution for that partition but not for the main problem. In this case line 3.1 is not executed but 3.2 is executed and the algorithm executes initialization phase to create new eurygasters for searching the solution in another part because of moving control to that phase in line 3.2.2 (goto 1).

2. The found solution is optimal solution for that partition and also for the main problem. In this case neither the sub instructions of line 3.1 nor the sub instructions of line 3.2 are executed and cause the algorithm be stopped. Therefore, this is a point that the absolute solution of the problem is reached.

In case of not finding the best solution of the partition or when it is hoped that the solution is in search space of the partition, line 3.1 is executed and changed the eurygasters location to fully cover the area of the partition search space. Finally, if the appropriated solution to the problem is obtained, the algorithm is terminated by line 3.2.3, otherwise a new partition is created and also new eurygasters are produced to be distributed over the partition.

## RESULTS

The results of the studies show the algorithm PSO is suitable for continuous problems and its using in discrete problems is very rare and expensive, while the proposed algorithm is suitable for both continuous and discrete problems. In the proposed algorithm the space of the problem is divided into several partitions and in each step of algorithm just one partition of the problem is investigated and in case of not reaching the expected solution, the other part is going to be searched. Also, in each part by changing the position of the eurygasters, we can prevent trapping in local optimum. On the one hand, our method has great advantages over genetic algorithms. In the genetic algorithms, the chromosomes of the first generation are distributed randomly all over the problem; this operation causes some problems as follows:

1. The probability of trapping in the local optimum increases.

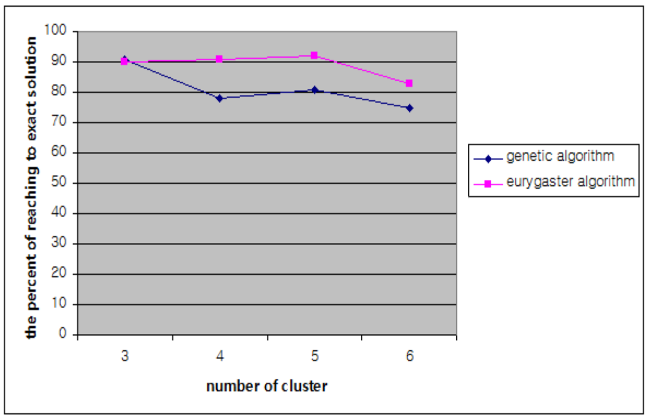

Figure1. The reaching rate to exact solution

2. Chromosomes that will be produced in the next generations can be placed in some locations in the problem space where the chromosomes of the previous generations had been placed. In other word, some regions of the problem space will be searched several times that causes execution time of the algorithm is increase and lead to decreasing in convergence speed.
3. Since in each generation all the space of the problems is searched for, the execution time of the algorithm will be increased. The proposed algorithm will not have the above-mentioned problems because of searching all the partition continuously, orderly and separately.

To evaluate the proposed algorithm we have used "Graph partitioning" [12] in which the nodes are split into some clusters in a way that the amount of interconnection is decreases to the least and intra connection increases to the most. To demonstrate the optimum of the proposed method, 100 nodes have been used to partitions into 3, 4, 5 and 6 clusters.

Before solving the problem by using the proposed algorithm, the related partitioning to solve the problems or the wheat fields must be determined. For example to solve the problem of "Graph Partitioning" to three clusters the following partitions can be done. Suppose n1, n2, and n3 are the number of the related nodes of clusters 1, 2 and 3 respectively.

n1>n2>n3 , n1>n2= n3, n1=n2>n3 , n1=n2=n3

Similarly, this kind of division can be done for the above-mentioned problem into 4, 5 and 6 clusters. The following figure1 shows the execution time of the above-mentioned problem in case of dividing into 3, 4, 5, and 6 clusters by using the genetic algorithm and the proposed algorithm.

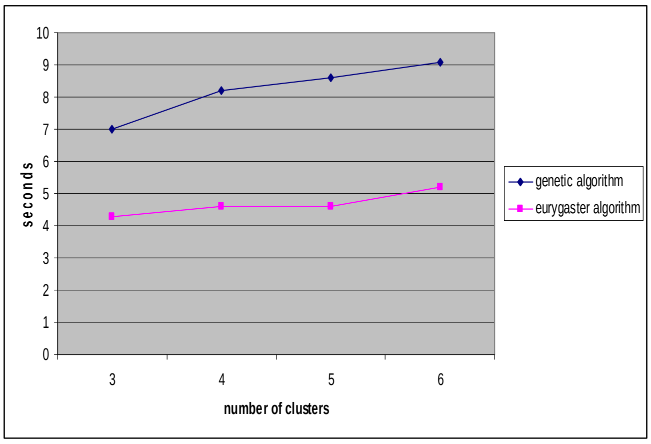

Fig 2. Execution time of the graph partitioning problem

In Fig. 1, the axis of X shows the number of the clusters and the axis of Y shows the execution time of the algorithm in seconds.
As the above Fig. 1 shows the execution time of the proposed algorithm is much less than the execution time of the genetic algorithm. Moreover, every program has been operated 100 times which Fig. 2 shows the percent of getting the exact solution to the problem in both of the algorithms.
Since the limitation that local optimum has created for the genetic algorithm, the percentage of successfulness by using this algorithm to get the exact solution is less than the proposed algorithm. Proposed algorithm lacks the limitations of local optimum and if the related fitness functions are defined properly, it can be claimed that almost 100 percent of cases getting the exact solution is possible. Also, Fig. 3 show that our approach has totally more performance than genetic algorithm. This figure has been computed by multiplying of Fig. 1 and Fig. 2 and denotes the ratio of the algorithm in reaching the exact solution of problems. As you can see in the Fig. 3, the total time of our approach is less than genetic algorithm that can be concluded researcher approach is faster and convergence speed is high.

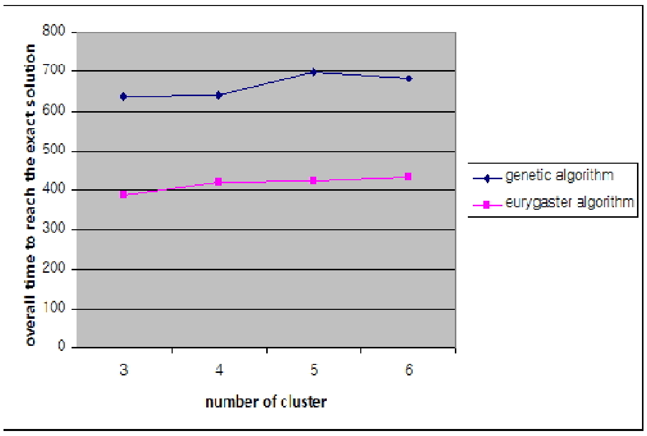

Fig. 3. Overall time to reach the exact solution

## CONCLUSIONS

In this article, a new method based on the behaviors of eurygasters has been presented to solve NP-class problems. Although PSO algorithms just can be used for continuous problems, this method can be used for both continuous and discrete problems. This approach unlike genetic algorithm lacks the local optimum so the probability of getting the more accurate solution in this method is much more than the genetic algorithm. Moreover, in this algorithm every space of the problem is searched for once while in the genetic algorithm every part of the problem space can be searched several times in different generations, so the rate of convergence in this algorithm is much more than the genetic algorithm. This algorithm is a suitable replacement for the genetic algorithm. Also, this algorithm is easy to implement by computer. If takes a few lines to programming and doesn't need a huge memory or CPU speed. In the proposed algorithm the space of the problem is divided to several partitions and every partition is searched for separately. The more efficient manner to reach the more convergence speed is that if the solution of a problem is found in a specific partition, the searching process be stopped in order to decrease the amount of execution time of the algorithm. This algorithm can be used for Divide and Congers problems properly. Since different parts of the problem have been separated, this method is very simple and efficient to be applied in parallel systems. The researchers hope that the explanation in this article can satisfy its readers about the operation of the algorithm.

## REFERENCES

1. R.C. Eberhart, and J. Kennedy, "A new optimizer using particle swarm theory". In Proceedings of the sixth international symposium on micro machine and human science, volume 43. New York, NY, USA: IEEE, 1995.
2. J. Kennedy, and R.C. Eberhart, "Particle swarm optimization". In Proceedings of IEEE international conference on neural networks, volume 4, pages 1942–1948. Perth, Australia, 1995.
3. M. Dorigo, "Optimization, Learning and Natural Algorithms", PhD thesis, Politecnico di Milano, Italie, 1992.
4. S. Kirkpatrick, C. D. JR. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing", IBM Research Report RC 9355.
5. F. Glover, and M. Laguna, "Tabu Search", Kluwer Academic Publishers, Boston, 1997.
6. Pham, DT. Ghanbarzadeh, A. Koc, E. Otri, S. Rahim, and M. Zaidi, "The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.
7. H. Murase, "Finite element analysis using a photosynthetic algorithm", Computers and Electronics in Agriculture, 29, pp. 115-123, 2000.
8. X. S. Yang, " New enzyme algorithm", Tikhonov regulation and inverse parabolic analysis, in Advances in Computational Methods in Science and Engineering, Lecture Series on Computer and Computer Sciences, ICCMSE, Eds. T. Simons and G. Maroulis, 4, 1880-1883, 2005.
9. K.N. Krishnan and, and D.Ghose, "Detection of multiple source locations using a glowworm metaphor with applications to collective robotics," IEEE Swarm Intelligence Symposium, Pasadena, California, USA, pp. 84–91, 2005.
10. A. Mucherino, and O. Seref, "Monkey Search: A Novel Meta-Heuristic Search for Global Optimization,"AIP Conference Proceedings 953, Data Mining, System Analysis and Optimization in Biomedicine, 162–173, 2007.
11. X.S. Yang, "Firefly algorithm," (chapter8) in: Nature-inspired Metaheuristic Algorithms, Luniver Press, 2008.
12. D. Doval, S. Mancoridis, and B. Mitchell,"Automatic clustering of software systems using a genetic algorithm," STEP '99, IEEE Computer Society, 1999.
13. Y. Shi, and R. Eberhart, "A Modified Particle Swarm Optimizer," IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9, 1998.
14. P. Mathiyalagan, U.R. Dhepthie, and S.N. Sivanandam, "Grid Scheduling Using Enhanced PSO Algorithm," (IJCSE) International Journal on Computer Science and Engineering, Vol. 02, No. 02, 140-145, 2010.
15. K. Derr, and M. Manic, "Multi-robot, multi-target particle swarm optimization search in noisy wireless environments," Proceedings of the 2nd Conference on Human System Interactions, Catania, Italy, 81–86, 2009.
16. Rechenberg, "Evolution Strategy," In Computational Intelligence: Imitating Life, J. M. Zurada, R. J. Marks II, and C. Robinson, Eds., IEEE Press, Piscataway, NJ, 1994.
17. Y. Shi, and R. Eberhart, "Parameter selection in particle swarm optimization," In Proceedings of the 7th International Conference on Evolutionary Programming VII, volume 1447, pages 591–600, 1998.
18. M. Clerc, "Particle Swarm Optimization," ISTE Ltd, UK/USA, 2006.
19. R. C. Eberhart, P. Simpson, and R. Dobbins, Chapter 6, pages 212–226. AP Professional, San iego, CA, 1996.
20. K. Deb, and N. Padhye, "Understanding and Improving Particle Swarm Optimization Using Evolutionary Algorithm Viewpoint," Under review. Evolutionary Computation Journal, published by MIT press, 2011.
21. Q. Bai, "Analysis of Particle Swarm Optimization Algorithm," Computer and Information Science, Vol. 3, No. 1, 2010.
22. R.C. Eberhart, and Y. Shi, "Particle swarm optimization: developments, applications and resources," In Proceedings of the 2001 congress on evolutionary computation, volume 1, pages 81–86. Piscataway, NJ, USA: IEEE, 2001.
23. R. Poli, "An analysis of publications on particle swarm optimization applications," Essex, UK: Department of Computer Science, University of Essex, 2007.
24. R. J. Collins, and D. R. Jefferson, "The evolution of sexual selection and female choice," In F. J. Varela and P. Bourgine, eds., Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life. MIT Press, 1992.
25. D. Ackley, and M. Littman, "Interactions between learning and evolution," In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, eds., Artificial Life II. Addison-Wesley, 1992.
26. L. Altenberg, "The evolution of evolvability in genetic programming," In K. E. Kinnear, Jr., ed.,Advances in Genetic Programming. MIT Press, 1994.
27. R. M. French, and A. Messinger, "Genes, phenes, and the Baldwin effect: Learning and evolution in a simulated population," In R. A. Brooks P. Maes, eds., Artificial Life IV. MIT Press, 1994.
28. J. R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection," MIT Press, 1992.
29. J. R. Koza, "Genetic Programming II: Automatic Discovery of Reusable Programs. MITPress, 1994.