2011 International Conference on Computer and Software Modeling
IPCSIT vol.14 (2011) © (2011) IACSIT Press, Singapore

Enhancing Niching Method: A Learning Automata Based Approach

Mohsen Jahanshahi [1+], Mohammad Sadegh Kordafshari [2], Majid Gholipour [2]

[1]Department of Computer Engineering, Central Tehran Branch, Islamic Azad University, Tehran, Iran
[2]Department of Computer Engineering, Qazvin branch, Islamic Azad University, Qazvin, Iran

Abstract. In some problems of mathematics there are many functions which have several optimum points which all, should be computed. The method NichePSO is previously designed to accomplish this. This paper presents a new memetic-based scheme to enhance the NichePSO. This work utilizes a powerful soft computing tool namely Learning Automata as the local search algorithm in our proposed memetic approach. Numerical results demonstrate the superiority of the proposed method over the original NichePSO in terms of convergence and diversity.

Keywords: PSO, Niching, Learning automata

1. Introduction

Optimization is the minimization or maximization of an objective function that normally is done with consideration of limitations identifying conditions of a problem. In other words, it means the finding of the best solution for a given problem. Some problems have several local optimums which may be computed using Niching method. In this paper, memetic method is used for increase of convergence speed and also the increasing rate of particles' diversity in Niching method, as two assessment factors of search methods. In this research, learning automata is used as a local search algorithm in memetic method.

The rest of the paper is organized as follow: in section 2, PSO method is introduced. In section 3, NichePSO is briefly discussed. Section 4 explains the local search methods which are used and then proposed methods are discussed in section 5. The results of simulations are included in section 6. Section 7 is conclusion.

2. Particle Swarm Optimization

Particle Swarm Optimization (PSO) was discussed in [1][2][3][4] and then has been improved for many years (e. g. [5][6]). It can be also used for several applications such as numerical optimization. Its main idea had been taken from Birds or fishes swarm behaviour which are searching for meal. Some birds are searching for meal randomly. Only a piece of meal may be found in the mentioned space. None of them knows about the real place of the meal. One of the best strategies is to follow a bird that is placed in minimum distance to a meal. In fact, this strategy is basis of PSO algorithm. This method is an effective technique for solution of optimization problems based on a swarm behaviour. In this method, each member of a swarm is called a particle who attempts to achieve a final solution with adjustment of its route and movement to the best personal and swarm experiences. In PSO algorithm, a solution is called a particle the

+ Corresponding author.
E-mail address: mjahanshahi@iauctb.ac.ir

same as a bird in swarm scheme. Each particle is described using a quality factor is given by a fitness function. The more nearness to the goal, the more qualification is obtained. Each particle also moves with a specified speed which conducts its movement. Each particle that follows the optimum particles in current position will continue its movement in the space of problem.

PSO starts in this way that a swarm of particles (solutions) are generated randomly and are updated during the generations and attempt to find an optimum solution. In each step, every particle is updated using two best values. First, is the best situation that a particle has ever been reached. It is known and kept by pbest (personal best). Another best value is used by the algorithm is the best position that has ever been obtained by a swarm of particles. It is known by gbest (global best). In some PSO editions, a particle chooses parts of populations that are its topological neighbors and only involves those in its behavior. In this case, the best local solution is shown by lbest (local best) and is used instead of gbest. After finding the best values, the velocity and position of particles are updated by formula (1) and (2) given below:

$$v[] = v[] + c1 * rand\ () * (\ pbest[] - position[]\ ) + c2 * rand\ () * (\ gbest[] - position[]\ ) \quad (1)$$

$$position\ [] = \ position[] + \ v[] \quad (2)$$

In (1) and (2) equations, V [] is the particle velocity and $position$ [] is current position of a particle. Both are arrays as long as the number of problem dimensions. $Rand$ () is a random variable in the random domain $(0, 1)$. $C1$ and $C2$ are learning parameters, normally both are the same values as $c1 = c2 = c$. In each dimension, velocity of particles is limited to a $Vmax$. In case, the sum of accelerations cause that the velocity in one dimension exceeds the maximum value, it is considered as $Vmax$.The right hand side of equation (1) is composed of three components, the first part, is the current velocity of a particle, the second and third parts take responsibility for variation of the velocity and its rotation towards the best personal and swarm experiences. Combining these two factors in equation (1) helps create a balance between local and global searches. Let us we don't consider the first part of the equation then the particles velocity is determined only with consideration of current position and the best single and swarm experiences of particles. Therefore, the best particle is fixed in its position and the rest of the particles move towards it. In fact, if we ignore the first part of equation (1), PSO will be a process in which, search space gradually becomes smaller and local search is performed around the best particle. In contrast, if we consider only first part of the equation (1), then the particles will continue their normal route up to border and it is said those are doing global search. Since in this algorithm, particles gradually tend to current optimum solution so if this is a local optimum solution then whole particles move towards it consequently PSO is not a practical way to leave this local optimization. Meanwhile, some problems have more than one general optimum solution that all have to be computed. These are the greatest problems of PSO algorithm that make it unable to solve multi peak problems particularly with a large state space.

3. Niche PSO

NichePSO is presented to find all the solutions of problems with more than one general optimum solution [2]. In this algorithm, niches are parts of the environment and main operation of each niche is to self-organize the particles to independent sub-swarm. Each sub-swarm determines the position of one niche and keeps it. The task of each sub-swarm is to find one of the optimum solutions. No information is exchanged amongst sub-swarms. This independency lets sub-swarms to keep niches. In summary, it can be said that performance of sub-swarms is stable and independent of the other swarms.

NichePSO begins its operation with one swarm that is called main swarm that includes whole particles. As soon as, a particle gets close to an optimum solution, one swarm will be formed with classification of particles. Then these particles are put out of the main swarm and continue the operations for finding the solution in their own sub swarms. In fact the main swarm is broken into some sub-swarms. NichePSO is convergent when smaller sub-swarms improve their presented solutions, and then in continue, the best global position for each sub-swarms is accepted as a solution.

3.1. Identification of Niches

When a particle is getting close to a local optimum, a swarm is formed. If the acceptability rate of a particle indicates tiny variation of some repetitions, then the swarm will be formed by this particle and its nearest

neighbors. In simpler word, standard deviation (σ_i) occurs in some repetition in objective function ($f(x_i)$) for each particle. If ($\sigma_i < \varepsilon$), the swarm is formed. To prevent the dependency problem,(σ_i) is normalized based on domain. The nearest neighbor of l for position of (x_i) of particle i, is given by Euclidean distance as follow:

$$\boxed{l = \arg\min_a \{\|x_i - x_a\|\}} \tag{3}$$

The process of swarm generation or niche identification is summarized in Fig. 1. In this algorithm, the symbol Q indicates set of sub-swarms ($Q = \{S_1, \ldots, S_K\}$) where, ($|Q| = K$). Each sub-swarm has the number of ($S_k.n_s$) particles. At the time of NichePSO generation, the value of K is zero and Q is an empty set.

```
if    σ_i < ε    then
    k = k + 1
    create    sub − swarm    S_k = {x_i, x_l};
    Let    Q ⟵ Q ∪ S_k;
    Let    S ⟵ S \ S_k;
end
```

Fig. 1: Generation algorithm of sub-swarms in Niche PSO

3.2. Absorbing particles in sub-swarms

Particles of main swarm move towards an area that is covered by the sub-swarms (S_k). These particles combine with sub-swarm for reasons below:

- The particles move in search area for creation of a sub-swarm may improve the diversity of sub-swarms.
- These particles in a sub-swarm increase the dispersion of those to an optimum space via increase of general information.

If for particle *i* we have:

$$\boxed{\|x_i - S_k.\hat{y}\| \le S_k.R} \tag{4}$$

Then the particle *i* is absorbed to sub-swarm (S_k) in such a way

$$\boxed{\begin{aligned} S_k &\longleftarrow S_k \cup \{x_i\} \\ S &\longleftarrow S \setminus \{x_i\} \end{aligned}} \tag{5}$$

In equation above, ($S_k.R$) points to radius of sub-swarm (S_k) and is defined as follow:

$$\boxed{S_k.R = \max\{\|S_k.\hat{y} - S_k.x_i\|\} \quad \forall_i = 1, \ldots, S_k.n_s} \tag{6}$$

Here, ($S_k.\hat{y}$) is the best general position of sub-swarm (S_k).

3.3. Merging of sub-swarms

Perhaps, more than one sub-swarms points to an optimum point. Here, it is possible that a particle moves to a solution is not absorbed to a sub-swarms. As a result, a new sub-swarm will be generated and it leads to a problem that a solution is followed by several sub-swarms in form of redundancy. For solving this problem, similar sub-swarms must merge together. The sub-swarms are the same if their space is considered in such a way that radiuses of particles converge in sub-swarms. The new merged sub-swarm has more general information and uses experiences of both old sub-swarms. The results of new sub-swarm are normally more accurate than the smaller old sub-swarm. In simpler word, two sub-swarms (S_k) and (S_{k+1}) merge together if:

$$\|S_{k1}.\hat{y} - S_{k2}.\hat{y}\| < (S_{k1}.R + S_{k2}.R) \tag{7}$$

If ($S_{k1}.R = S_{k2}.R = 0$) the following formula will be replaced with equation 8.

$$\boxed{\|S_{k1}.\hat{y} - S_{k2}.\hat{y}\| < \mu} \tag{8}$$

Where μ is a tiny value tends to zero (e.g. $\mu = 10^{-3}$). If μ is a very large value, perhaps unsuitable sub-swarms merge together and lead to failure of finding some solutions. In order to keep μ in the domain, ($\|S_{k1}.y - S_{k2}.y\|$) is normalized in (0, 1).

3.4. Stop Conditions

Several stop conditions may be used to finish the search of solutions. Note that each sub-swarm has founded a unique solution.

4. Learning Automata Theory

In this section the Learning Automata for the proposed framework will be briefly reviewed;

Cellular Automata (CA) are mathematical models for systems consisting of large numbers of simple identical components with local interactions. The simple components act together to produce complicated patterns of behavior with high degree of efficiency and robustness. CA are non-linear dynamical systems in which space and time are discrete and consists of a finite dimensional lattice of cells whose states are restricted to a finite set of integers. The state of each cell at any time instant is determined by a rule from states of neighboring cells at the previous time instant. Given a finite set and a finite dimension *d,* CA can be considered as a d-dimensional lattice [7][8].

Learning Automata (LA) is an abstract model that chooses an action from a finite set of its actions randomly and takes it [8][9][10] . In this case, environment evaluates this taken action and responses by a reinforcement signal. Then, learning automata updates its internal information regarding both the taken action and received reinforcement signal. After that, learning automata chooses another action again. Every environment is represented by $E = \{\alpha, \beta, c\}$, where $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ is a set of inputs, $\beta = \{\beta_1, \beta_2, ..., \beta_r\}$ is a set of outputs, and $c = \{c_1, c_2, ..., c_r\}$ is a set of penalty probabilities. Whenever set β has just two members, model of environment is *P*-model. In this environment $\beta_1 = 1$, $\beta_2 = 0$ are considered as penalty and reward respectively. Similarly, *Q*-model of environment contains a finite set of members. Also, *S*-model of environment has infinite number of members. c_i is the penalty probability of taken action α_i. Learning automata is classified into fixed structure and variable structure. Learning automata with variable structure is introduced as follows; Learning automata with variable structure is represented by $\{\alpha, \beta, p, T\}$, where $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_r\}$ is a set of actions, $\beta = \{\beta_1, \beta_2, \cdots, \beta_r\}$ is a set of inputs, $p = \{p_1, p_2, \cdots, p_r\}$ is the action probability vector, and $p(n+1) = T[\alpha(n), \beta(n), p(n)]$ is learning algorithm. Learning automata operates as follows; learning automata chooses an action from its probability vector randomly (P_i) and takes it. Suppose that the chosen action is α_i . Learning automata after receiving reinforcement signal from environment updates its action probability vector according to formulas 9 and 10 in case of desirable and undesirable received signals respectively. In formulas 1 and 2, *a* and *b* are reward and penalty parameters respectively. If *a* = *b* then algorithm is named L_{R-P} . Also, if $b \ll a$ then the algorithm is named $L_{R\varepsilon P}$. Similarly, if *b* = 0 then the algorithm is called L_{R-I} .

$$\begin{aligned} p_i(n+1) &= p_i(n) + a.(1 - p_i(n)) \\ p_j(n+1) &= p_j(n) - a.p_j(n) \qquad \forall j \ \ j \neq i \\ p_i(n+1) &= (1-b).p_i(n) \\ p_j(n+1) &= \frac{b}{r-1} + (1-b)p_j(n) \qquad \forall j \ \ j \neq i \end{aligned} \tag{9}$$

(10)

5. Proposed methods

In this paper, to make NichePSO, more effective, memetic method has been used for training the particles in each sub-swarm. This paper introduces a different method resulting from combination of learning automata by NichePSO for improvement of this method.

In proposed methods in each sub-swarm, a set of particles is chosen and improved by the local search method. Improved particles are added to set of whole particles and from those, enough sub-swarms will be chosen. In what follows, details of used algorithms used as a local search in memetic method are presented.

5.1. LA (L_{R-P}) –based method

This method uses learning automata to update the velocity of particles. In this method two functions are considered to update particles velocity by learning automata. The first operation means to consider the effect of global best of subgroup for updating a particle and the second operation means to update velocity of the

particle without consideration of the global best of subgroup. After applying these functions to whole particles of subgroup if the total values of particles is less than the old total value of the particles then automata is rewarded otherwise is penalized.

5.2 LA (L_{R-I}) – Based Method

This method is similar to method above. The only difference is that after doing Automata functions on whole particles of subgroup, if the total values of current particles is less than the old particles then the automata is rewarded but if the total value of particles is more than old total values the automata is not penalized.

6. SimulationResults

In this section, the results of simulations are presented compared with original NichePSO method to be run in Matlab 7.04. The aim of running all the methods in all simulations is to find optimum points of a following standard function.

$$y = (exp((-2 * log2(2)) * ((x - 0.08)/(0.854))\text{^}2)) * (sin(5 * pi * x))\text{^}6 \tag{11}$$

In all simulations, PSO parameters are initially valued as follow:

$$pop = 40 \quad c1 = 1.2 \quad a = 0.01 \quad r1 = a + (b - a) * rand$$
$$c2 = 1.2 \quad b = 1 \quad r2 = a + (b - a) * rand$$

Also the value of inertia weight w is given by the following formula where, T is the maximum repetition and t is current time of simulation.

$$w = 0.5 - (0.3 * (1 - t)/(1 - T)) \tag{12}$$

Figure 2, indicates the diversity rate of particles during different simulations. This results are gained from 10 times repetition of an algorithm that its accurate results is listed in Table 1 in which the values are gained by taking the mean of variance for whole particles during the period of each step of simulation. As depicted in this figure, the method LA (L_{R-P})-based has the most diversity rate and hence outperforms others.

To compare the convergence of above algorithms, each algorithm is repeated 50 times and the number of their divergences is mentioned as Table 2. From the results, LA (L_{R-I})-based method has the least divergence rate and hence outperforms the other methods in terms of convergence rate.

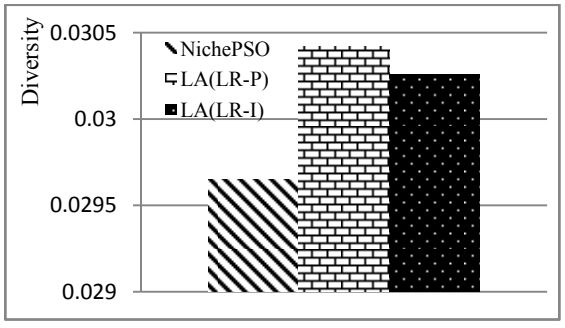

Fig. 2: Diversity of particles in various algorithms

Table 2: Divergence rate of different algorithms

NichePSO	**LA(L_{R-P})**	**LA(L_{R-I})**
15	*13*	*5*

Table 1. Diversity rate of particles in various algorithms for 10 times of simulations

NichePSO	LA(L_{R-P})	LA(L_{R-I})
0.0296	0.0331	0.0331
0.0284	0.0315	0.0315
0.0221	0.0397	0.0297
0.0357	0.0314	0.0314
0.0265	0.0205	0.0205
0.0268	0.0301	0.0301
0.0256	0.0336	0.0336
0.0397	0.0311	0.0311
0.0292	0.0299	0.0299
0.0329	0.0233	0.0317
0.02965	0.03042	0.03026

7. Conclusions

In this paper, two combinational methods were presented to raise the diversity of particles and improvement of convergence of NichePSO as two assessment factors of search methods. From the results, we found that the LA(L_{R-P})-based method is preferred when diversity is the main objective. On the other hand, applying, LA(L_{R-I})-based method leads to most convergence rate among the other methods.

8. References

[1] J. Kennedy and R. C. Eberhart. Particle swarm optimization. *Proc. IEEE International Conf. Neural Networks*. Piscataway, NJ, 1995: 1942-1948.

[2] R. Brits, A.P. Engelbrecht and F. van den Bergh. A niching particle swarm optimizer. *Proc. IEEE swarm intelligence symposium*. Indianapolis, Indiana, USA, 2003: 54-59.

[3] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithm. *Technical report*, Caltech, Pasadena, 1989, Concurrent computation program report 826.

[4] A. Stacey, M. Jancic, and I. Grundy. Particle swarm optimization with mutation. *Proc. Congress on Evolutionary Computation.* Piscataway, NJ, 2003:1425-1430

[5] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer, *Proc. IEEE International Conf. Evolutionary Computation*. Anchorage, Alaska, USA, 1998: 69-73.

[6] F. v. d. Bergh, A. P. Engelbrecht. A new locally convergent particle swarm optimizer, *Proc. IEEE Conf. Systems Man and Cybernetics*. Tunisia. 2002: 96-101

[7] H. Beigy and M. R. Meybodi. A mathematical framework for cellular learning automata. *Advances on Complex Systems*. 2004, 7(3): 295-320

[8] M. Jahanshahi, M. R. Meybodi, and M. Dehghan. Cellular learning automata based scheduling method for wireless sensor networks. *Proc. 14th International Computer Conference*, Amirkabir University of Technology, Tehran, Iran. 2009: 646-651

[9] M. A. L. Thathachar and P. S. Sastry, Varieties of learning automata: an overview. *IEEE Transaction on Systems, Man, and Cybernetics-Part B: Cybernetics*. 2002, 32(6), 711–722