Applied Soft Computing 13 (2013) 2823–2836

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A hierarchical particle swarm optimizer with latin sampling based memetic algorithm for numerical optimization

Yong Peng[a,*], Bao-Liang Lu[a,b]

[a] Center for Brain-like Computing and Machine Intelligence, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, PR China
[b] MoE-Microsoft Key Laboratory for Intelligent Computing and Intelligent Systems, Shanghai Jiao Tong University, Shanghai 200240, PR China

ARTICLE INFO

Article history:
Received 31 December 2011
Received in revised form 13 May 2012
Accepted 16 May 2012
Available online 6 June 2012

Keywords:
Memetic algorithm
Particle swarm optimizer
Latin hypercube sampling
Comprehensive learning
Cylindricity

ABSTRACT

Memetic algorithms, one type of algorithms inspired by nature, have been successfully applied to solve numerous optimization problems in diverse fields. In this paper, we propose a new memetic computing model, using a hierarchical particle swarm optimizer (HPSO) and latin hypercube sampling (LHS) method. In the bottom layer of hierarchical PSO, several swarms evolve in parallel to avoid being trapped in local optima. The learning strategy for each swarm is the well-known comprehensive learning method with a newly designed mutation operator. After the evolution process accomplished in bottom layer, one particle for each swarm is selected as candidate to construct the swarm in the top layer, which evolves by the same strategy employed in the bottom layer. The local search strategy based on LHS is imposed on particles in the top layer every specified number of generations. The new memetic computing model is extensively evaluated on a suite of 16 numerical optimization functions as well as the cylindricity error evaluation problem. Experimental results show that the proposed algorithm compares favorably with conventional PSO and several variants.

1. Introduction

Optimization has been a research hotspot for several decades. Many real-world optimization problems in engineering are becoming increasingly complicated, so optimization algorithms with high performance are needed [1,2]. Unconstrained optimization problems can be formulated as *D*-dimensional optimization problems over continuous space

$$min f(\mathbf{x}),\ \mathbf{x} = [x_1, x_2, \ldots, x_D] \tag{1}$$

Evolutionary algorithms, inspired by natural evolution, have been widely used as effective tools to solve optimization problems. One class of nature inspired algorithms are swarm intelligent algorithms. Particle swarm optimizer (PSO) [3,4] has attracted attention in the academic and industrial community. Although PSO shares many similarities with evolutionary algorithms, the original PSO does not use the traditional evolution operators such as crossover and mutation. PSO draws on the swarm behavior of birds flocking where they search for food in a collaborative way. Each member, in the swarm, called a *particle*, represents a potential solution to the target problem and it adapts its search patterns by learning from its own experience and other members' experience. The particle is a point in the search space and it aims at finding the global optimum which is regarded as the location of food. Each particle has two attributes called *position* and *velocity* and its direction of flight is adjusted according to the experiences of the swarm. The swarm as a whole searches for the global optimum in *D*-dimensional feasibility space.

The PSO algorithm is easy to understand and implement, and has been proved to perform well on many optimization problems. However, it may easily get trapped in a local optimum for many reasons, such as the lack of diversity among particles and overlearning from the best particle found so far. To improve PSO's performance on complex numerical optimization problems, we propose a hierarchical PSO framework, in which several swarms evolve in parallel towards the global optimum and we design a new mutation operator to increase the diversity of swarms. After evolving for a specified number of generations, a latin hypercube sampling method is used to execute the local search.

This paper is organized as follows. Section 2 introduces the original PSO and some variants. Section 3 describes the proposed hierarchical PSO with latin sampling based memetic algorithm, including four subsections: hierarchical PSO framework, mutation strategy, latin hypercube sampling based local search strategy and the overall framework of the proposed memetic algorithm. Section 4 gives the experimental results, describes the related parameter tuning process and compares the performance of the proposed algorithm on a suite of test problems to that of other PSO variants. Section 5 gives conclusions and describes future work.

* Corresponding author.
E-mail address: StanY.Peng@gmail.com (Y. Peng).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.05.020

```
//Initialize swarm S
for i := 1 to swarmsize do
    for d := 1 to D do
        V_i^d := rand[V_min, V_max]; X_i^d := rand[X_min, X_max];
    end for
end for
Compute the fitness value of each particle F = (f_1, f_2, ..., f_ps);
Set the pbest = (pbest_1, pbest_2, ..., pbest_ps) and the gbest;
Set the acceleration constants c_1 and c_2;
Set the iteration counter t:=0;
while t ≤ Gen do
    for i := 1 to swarmsize do
        for d := 1 to D do
            //Update the velocity V_i^d of particle X_i using Eq.2
            V_i^d := V_i^d + c_1 · rand1_i^d · (pbest_i^d − X_i^d) + c_2 · rand2_i^d · (gbest^d − X_i^d);
            //Update the position X_i^d of particle X_i using Eq.3
            X_i^d := X_i^d + V_i^d;
        end for
        Evaluate the fitness value f_i of the new particle X_i;
        if f_i is better than the fitness value of pbest_i then
            Set X_i to be pbest_i;
        end if
        if f_i is better than the fitness value of gbest then
            Set X_i to be gbest;
        end if
    end for
    if termination condition is met then
        break;
    else
        t := t + 1;
    end if
end while
```

Fig. 1. Original particle swarm optimizer.

2. Particle swarm optimizers

2.1. Original PSO

PSO is a stochastic optimization algorithm which simulates swarm behavior. The individuals move over a specified D-dimensional feasible space. As in a the genetic algorithm, the particles in PSO are initialized with random velocities and positions. The algorithm adaptively updates the velocity and position of each particle in the swarm by learning from the good experiences. In the original PSO [3], the velocity $V_i{}^d$ and position $X_i{}^d$ of the dth dimension of the ith particle are updated as follows.

$$\begin{aligned} V_i^d :&= V_i^d + c_1 \cdot rand1_i^d \cdot (pbest_i^d - X_i^d) \\ &\quad + c_2 \cdot rand2_i^d \cdot (gbest^d - X_i^d) \\ X_i^d &:= X_i^d + V_i^d \end{aligned} \tag{2}$$

where $X_i = (X_i^1, X_i^2, \ldots, X_i^D)$ is the position of the ith particle and $V_i = (V_i^1, V_i^2, \ldots, V_i^D)$ represents velocity of particle i, $\mathbf{pbest}_i = (pbest_i^1, pbest_i^2, \ldots, pbest_i^D)$ is the best previous position yielding the best fitness value for the ith particle, $\boldsymbol{gbest} = (gbest^1, gbest^2, \ldots, gbest^D)$ is the best position found so far over the whole swarm, c_1 and c_2 are the acceleration constants, reflecting the weighting of stochastic acceleration terms that pull each particle towards *pbest* and *gbest* positions, respectively. $rand1_i^d$ and $rand2_i^d$ are two random numbers in the range [0,1].

A particle's velocity on each dimension is confined to a maximum magnitude V_{max}. If $|V_i^d|$ exceeds a pre-specified positive constant value V_{max}^d, then the velocity on the dimension is assigned to $sign(|V_i^d|)V_{max}^d$.

The framework of the original PSO is shown in Fig. 1. From the flow of the iterative process, we can find that each particle flies to the global best particle in the swarm; this leads to a severe drawback of overlearning from the best particle. Consequently, the diversity of the whole swarm will drop down dramatically. If the best particle does not share the same niche with the global optimum, the particles may easily get trapped in a local optimum. Since PSO's introduction in 1995, many researchers have worked on improving its performance in various ways and many more effective variants have been proposed; these will be discussed in next subsection.

2.2. Some variants of PSO

This section gives a brief survey of several PSO variants proposed in recent years. Shi and Eberhart [5] introduced inertia weight w into the original PSO algorithm, so the criterion for updating the velocity was changed to

$$V_i^d := w \cdot V_i^d + c_1 \cdot rand1_i^d \cdot (pbest_i^d - X_i^d) + c_2 \cdot rand2_i^d \cdot (gbest^d - X_i^d). \tag{3}$$

They indicated that the inertia weight plays an important role in balancing the global and local search abilities; a large inertia weight encourages global search while a small inertia weight encourages local search. Based on this idea, the inertia weight is usually set to decrease linearly over iterations.

Different types of topologies have been designed to improve PSO's performance in solving different optimization problems. Kennedy [6,7] claimed that PSO with a small neighborhood might perform better on complex problems, while PSO with large neighborhood would perform better on simple problems. Suganthan [8] defined the neighborhood of a particle as the several nearest particles in each iteration so that a dynamic neighborhood is computationally intensive. Jian et al. [9] examined several