Modification of the Particle Swarm Optimizer for locating all the global minima

K.E. Parsopoulos, M.N. Vrahatis*[1]

*Department of Mathematics, University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras, GR-261.10 Patras, Greece

Abstract

In many optimization applications, escaping from the local minima as well as computing all the global minima of an objective function is of vital importance. In this paper the Particle Swarm Optimization method is modified in order to locate and evaluate all the global minima of an objective function. The new approach separates the swarm properly when a candidate minimizer is detected. This technique can also be used for escaping from the local minima which is very important in neural network training.

1 Introduction

Many recent advances in science, economics and engineering rely on numerical techniques for computing globally optimal solutions to corresponding optimization problems. These problems are extremely diverse and include economic modeling, neural networks training, image processing and engineering design and control [3]. Due to the existence of multiple local and global optima all these problems cannot be solved by classical nonlinear programming techniques.

During the past three decades, however, many new algorithms have been developed and new approaches have been implemented, resulting to powerful optimization algorithms such as the Evolutionary Algorithms [6]. In contrast to other adaptive algorithms, evolutionary techniques work on a set of potential solutions, which is called *population*, and find the optimal solution through cooperation and competition among the potential solutions. These techniques can often find optima in complicated optimization problems faster than traditional optimization methods. The most commonly used population–based evolutionary computation techniques, such as Genetic Algorithms and Artificial Life methods, are motivated from the evolution of nature and the social behavior.

It is worth noting that, in general, Global Optimization (GO) strategies possess strong theoretical convergence properties, and, at least in principle, are straightforward to implement and apply. Issues related to their numerical efficiency are considered by equipping GO algorithms with a "traditional" local optimization phase. Global convergence, however, needs to be guaranteed by the global–scope algorithm component which, theoretically, should be used in a complete, "exhaustive" fashion. These remarks indicate the inherent computational demand of the GO algorithms, which increases non–polynomially, as a function of problem–size, even in the simplest cases.

In practical applications, most of the aforementioned methods can detect just *sub–optimal solutions* of the objective function. In many cases these sub–optimal solutions are acceptable but there are applications where an optimal solution is not only desirable but also indispensable. Moreover, in many applications there are many global minima that have to be computed quickly and reliably. Therefore, the development of robust and efficient GO methods is a subject of considerable ongoing research.

Recently, Eberhart and Kennedy (1995) proposed the *Particle Swarm Optimization* (PSO) algorithm: a new, simple evolutionary algorithm, which differs from other evolution–motivated evolutionary computation techniques in that it is motivated from the simulation of social behavior [2, 4]. Although, in general, PSO results in global solutions even in high–dimensional spaces, there are some problems whenever the objective function has many global and few (or not at all) local minima.

In this paper we propose a strategy that finds all global minima (or some of them if their number is infinite) of an objective function using a modification of the PSO technique and show, through simulation experiments, that this strategy is efficient and effective.

The paper is organized as follows: the background of the PSO is presented in Section 2. The proposed

[1]e–mail: {kostasp, vrahatis}@math.upatras.gr

strategy is derived in Section 3. In Section 4 some results are presented and discussed, and finally conclusions are drawn in Section 5.

2 The Particle Swarm Optimizer

As it is already mentioned, PSO is different from other evolutionary algorithms. Indeed, in PSO the population dynamics simulates a "bird flock's" behavior where social sharing of information takes place and individuals can profit from the discoveries and previous experience of all other companions during the search for food. Thus, each companion, called *particle*, in the population, which is now called *swarm*, is assumed to "fly" over the search space in order to find promising regions of the landscape. For example, in the minimization case, such regions possess lower functional values than other visited previously. In this context, each particle is treated as a point in a D–dimensional space which adjusts its own "flying" according to its flying experience as well as the flying experience of other particles (companions). There are many variants of the PSO proposed so far. In our experiments we used a new version of this algorithm, which is derived by adding a new inertia weight to the original PSO dynamics [1]. This version is described in the following paragraphs.

First, let us define the notation adopted in this paper: the i-th particle of the swarm is represented by the D–dimensional vector $X_i = (x_{i1}, x_{i2}, \dots, x_{iD})$ and the best particle in the swarm, i.e. the particle with the smallest function value, is denoted by the index g. The best previous position (the position giving the best function value) of the i-th particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, \dots, p_{iD})$, and the position change (velocity) of the i-th particle is $V_i = (v_{i1}, v_{i2}, \dots, v_{iD})$.

The particles evolve according to the equations

$$\begin{aligned} v_{id} &= w\, v_{id} + c_1\, r_1\, (p_{id} - x_{id}) + \\ &\quad + c_2\, r_2\, (p_{gd} - x_{id}), \qquad (1) \\ x_{id} &= x_{id} + v_{id}, \qquad (2) \end{aligned}$$

where $d = 1, 2, \dots, D$; $i = 1, 2, \dots, N$, and N is the size of population; w is the inertia weight; c_1 and c_2 are two positive constants; r_1 and r_2 are two random values in the range $[0, 1]$.

The first equation is used to calculate i-th particle's new velocity by taking into consideration three terms: the particle's previous velocity, the distance between the particle's best previous and current position, and, finally, the distance between swarm's best experience (the position of the best particle in the swarm) and i-th particle's current position. Then, following the second equation, the i-th particle flies toward a new position. In general, the performance of each particle is measured according to a predefined fitness function, which is problem–dependent.

The role of the inertia weight w is considered very important in PSO convergence behavior. The inertia weight is employed to control the impact of the previous history of velocities on the current velocity. In this way, the parameter w regulates the trade–off between the global (wide–ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine–tuning the current search area. A suitable value for the inertia weight w usually provides balance between global and local exploration abilities and, consequently, a reduction on the number of iterations required to locate the optimal solution. A general rule of thumb suggests that it is better to initially set the inertia to a large value, in order to make better global exploration of the search space, and gradually decrease it to get more refined solutions, thus a time decreasing inertia weight value is used. The initial population can be generated either randomly or by using a Sobol sequence generator [8] which ensures that the D-dimensional vectors will be uniformly distributed into the search space.

From the above discussion it is obvious that PSO, to some extent, resembles evolutionary programming. However, in PSO, instead of using genetic operators, each individual (particle) updates its own position based on its own search experience and other individuals' (companions) experience and discoveries. Adding the velocity term to the current position, in order to generate the next position, resembles the mutation operation in evolutionary programming. Note that in PSO, however, the "mutation" operator is guided by particle's own "flying" experience and benefits by the swarm's "flying" experience. In another words, PSO is considered as performing mutation with a "conscience", as pointed out by Eberhart and Shi [1].

3 Locating all the global minima of an objective function using the PSO method

Let $f\colon \mathcal{B} \to \mathbb{R}$ be an objective function that has many global minima inside a hypercube $\mathcal{B}$. If we use the plain PSO algorithm to compute just one global minimizer, i.e. a point $\bar{x} \in \mathcal{B}$ such that $f(\bar{x}) \leq f(x)$, for all $x \in \mathcal{B}$, there are two things that might happen: either the PSO will find one global minimum (but we don't foreknow which one) or the swarm will ramble over the search space failing to decide where to land. This last behavior is due to the equal "good" information that each global minimizer has. Each particle moves toward a global minimizer and influences the swarm in order to move toward that direction, but it is also affected by the rest of the particles in order to move toward the other global minimizer that they target. The result of this interaction between particles is a cyclic movement over the search space and disability to detect a minimum. A strategy to overcome these problems and find all global minimizers of f is described in the rest of this section.

In many applications, such as neural networks training, the goal is to find a global minimizer of a nonnegative function. The global minimum value is a priori known and is equal to zero, but there is a finite (or infinite in neural networks case) number of global minimizers. In order to avoid the problem mentioned in the previous paragraph, we can do as follows: we determine a not–so–small threshold $\epsilon > 0$ (e.g. if the desired accuracy is 10^{-5}, a threshold around 0.01 or 0.001 will work) and whenever a particle has a functional value that is smaller than ϵ, we pull this particle away from the population and isolate it. Simultaneously, we apply deflation or *stretching* [7] to the original objective function f at that point, in order to repel the rest of the swarm from moving toward it and add a new particle (randomly generated) in the swarm.

"Stretching" is a new technique that provides a way of escape from the local minima when PSO's convergence stalls. It consists of a two–stage transformation to the form of the original function f and can be applied soon after a local minimum $\bar{x}$ of the function f has been detected:

$$G(x) = f(x) + \gamma_1 \frac{\|x-\bar{x}\|(\mathrm{sign}(f(x)-f(\bar{x}))+1)}{2}, \qquad (3)$$

$$H(x) = G(x) + \gamma_2 \frac{\mathrm{sign}\,(f(x)-f(\bar{x}))+1}{2\,\tanh\,(\mu(G(x)-G(\bar{x})))}, \qquad (4)$$

where γ_1, γ_2 and μ are arbitrary chosen positive constants, and $\mathrm{sign}(\cdot)$ defines the well known three–valued sign function [7]:

$$\mathrm{sign}(x) = \begin{cases} +1, & x > 0, \\ 0, & x = 0, \\ -1, & x < 0. \end{cases} \qquad (5)$$

Thus, after isolating a particle, we check its functional value. If the functional value is far from the desired accuracy, we can generate a small population of particles around it and constrain this small swarm in the isolated neighborhood of f to perform a finer search while the big swarm continues searching the rest of the search space for other minimizers. If we set the threshold to a slightly higher value, then the isolated particle is probably a local minimizer and during the local search, a global minimum will not be detected but we have already helped PSO to avoid it by deflating or stretching it. If we know how many global minimizers of f exist in $\mathcal{B}$ then, after some cycles, we will find all of them. In case we do not know the number of global minimizers, we can ask for a specific number of them or let the PSO run until it reaches the maximum allowable number of iterations in a cycle. This will imply that no other minimizers can be detected by PSO. The whole algorithm can be parallelized and run the two procedures (for the big and the small swarm) simultaneously, saving this way a lot of time.

In the next section we will discuss a simulation of a simple yet difficult problem that can be solved using the presented algorithm.

4 Some experimental results

Let f be the 2-dimensional function

$$f(x_1, x_2) = \cos(x_1)^2 + \sin(x_2)^2, \qquad (6)$$

where $(x_1, x_2) \in \mathbb{R}^2$. This function has infinite number of minima in $\mathbb{R}^2$, at the points $(\kappa \frac{\pi}{2}, \lambda \pi)$, where $\kappa = \pm 1, \pm 3, \pm 5, \ldots$ and $\lambda = 0, \pm 1, \pm 2, \pm 3, \ldots$. We assume that we are interested only in the subset $[-5, 5]^2$ of $\mathbb{R}^2$. Into this hypercube, the function f has 12 global (equal to zero) minima.

If we try to find a single minimizer of f then we find out that the swarm moves back and forth as described in the previous section, until the maximum number of iterations is reached, failing to detect the minimizer. As already mentioned, this happens due to the same information (i.e. functional value) that each minimizer has. Thus, we could say that the swarm is so excited that it cannot decide where to land. Applying the algorithm given above, after 12 cycles of the method, we found all global minimizers with accuracy 10^{-5} and there even was no need for further local search.

In a second experiment we tried to find all minima of a notorious two dimensional test function, called the *Levy No. 5*:

$$f(x) = \sum_{i=1}^{5} i \cos[(i+1)x_1 + i] \times$$

$$\times \sum_{j=1}^{5} j \cos[(j+1)x_2 + j] + +(x_1+1.42)^2 + (x_2+0.80)^2, \quad (7)$$

where $-10 \leq x_i \leq 10, i = 1,2$. There are about 760 local minima and one global minimum with function value $f(x^*) = -176.1375$ located at $x^* = (-1.3068, -1.4248)^\top$. The large number of local optimizers makes it extremely difficult for any method to locate the global minimizer. Using the presented algorithm, we are able to compute all minima (global and local) of this function in cpu time that does not surpass 760 times the mean cpu time needed to compute each minimizer separately.

In another experiment a neural network has been trained using the PSO to learn the XOR Boolean classification problem. The XOR function maps two binary inputs to a single binary output and the network that was trained to solve the problem had two linear input nodes, two hidden nodes with logistic activations and one linear output node. Training the network corresponds to the minimization of a 9-dimensional objective function [5, 9]. It is well known from the neural networks literature that successful training in this case, i.e. reaching a global minimizer, strongly depends on the initial weight values and that the error function of the network presents a multitude of local minima. To solve this problem, we use the new algorithm as follows: we set a threshold of 0.1 and start the algorithm as above but if the standard deviation of the population in an iteration is too close to zero without having functional value close to the threshold (e.g. if the error value of the network is around 0.5, where there is a well known local minimum of the function), we pull the best particle of the population away and isolate it. This particle is probably a local minimizer (or near one) and thus we provide to it some new particles (in our simulation the size of the population was 40 thus we were adding 10 particles to the isolated one) and perform a local search in the vicinity of it (we took an area of radius 0.01 around it) while the rest of the big swarm continues searching the rest of the space. If the local search yields a global minimizer, we add it to our list of found global minima, otherwise it is a local minimizer and we have already avoided it. In this way, we are able to detect an arbitrarily large number of global minimizers while simultaneously avoiding local ones.

5 Conclusions

A new strategy for locating efficiently and effectively all the global minimizers of a function with many global and local minima has been introduced. Experimental results indicate that the proposed modification of the PSO method is able to detect effectively all the global minimizers instead of rambling over the search space or attracting by the local minima.

The algorithm provides stable and robust convergence and thus a better probability of success for the PSO. Also, it can be straightforwardly parallelized.

Extensive testing on higher–dimensional and more complicate real–life optimization hard tasks is necessary to fully investigate the properties of the proposed algorithm, as well as give some hints of modifications that will probably improve its performance.

References

[1] R.C. Eberhart and Y.H. Shi, "Evolving Artificial Neural Networks", *Proc. Int. Conf. on N.N. and Brain*, Beijing, P.R. China, 1998.

[2] R.C. Eberhart, P.K. Simpson and R.W. Dobbins, "Computational Intelligence PC Tools", Academic Press Professional, Boston, 1996.

[3] R. Horst, P.M. Pardalos and N.V. Thoai, "Introduction to Global Optimization", Kluwer Academic Publishers, 1995.

[4] J. Kennedy and R.C. Eberhart, "Particle Swarm Optimization", *Proc. IEEE Int. Conf. on N.N.*, Piscataway, NJ, pp. 1942–1948, 1995.

[5] G.D. Magoulas, M.N. Vrahatis and G.S. Androulakis, "Effective back–propagation with variable stepsize", *Neural Networks*, vol. 10, pp. 69–82, 1997.

[6] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", Springer, New York, 1996.

[7] K.P. Parsopoulos, V.P. Plagianakos, G.D. Magoulas and M.N. Vrahatis, "Objective function "stretching" to alleviate convergence to local minima", *Nonlinear Analysis, T.M.A.*, 2001, to appear.

[8] W.H. Press, W.T. Vetterling, S.A. Teukolsky and B.P. Flannery, "Numerical Recipes in Fortran 77", Cambridge University Press, 1992.

[9] M.N. Vrahatis, G.S. Androulakis, J.N. Lambrinos and G.D. Magoulas, "A class of gradient unconstrained minimization algorithms with adaptive stepsize", *J. of Comp. and App. Math.*, vol. 114, pp. 367–386, 2000.