Particle Swarm Optimization

Article · May 2011

CITATIONS
8

READS
8,241

1 author:

Gonçalo Pereira
INESC-ID / IST / Universidade de Lisboa
21 PUBLICATIONS **246** CITATIONS

SEE PROFILE

Particle Swarm Optimization

Gonçalo Pereira
INESC-ID and Instituto Superior Técnico
Porto Salvo, Portugal
gpereira@gaips.inesc-id.pt

April 15, 2011

1 What is it?

Particle Swarm Optimization is an algorithm capable of optimizing a non-linear and multidimensional problem which usually reaches good solutions efficiently while requiring minimal parameterization.

The algorithm and its concept of "Particle Swarm Optimization"(PSO) were introduced by James Kennedy and Russel Ebhart in 1995 [4]. However, its origins go further backwards since the basic principle of optimization by swarm is inspired in previous attempts at reproducing observed behaviors of animals in their natural habitat, such as bird flocking or fish schooling, and thus ultimately its origins are nature itself. These roots in natural processes of swarms lead to the categorization of the algorithm as one of Swarm Intelligence and Artificial Life.

The basic concept of the algorithm is to create a swarm of particles which move in the space around them (the problem space) searching for their goal or the place which best suits their needs given by a fitness function. A nature analogy with birds is the following: a bird flock flies in its environment looking for the best place to rest (the best place can be a combination of characteristics like space for all the flock, food access, water access or any other relevant characteristic).

Based on this simple concept there are two main ideas behind its optimization properties:

- a single particle (which can be seen as a potential solution to the problem) can determine "how good" its current position is. It benefits not only from its problem space exploration knowledge but also from the knowledge obtained and shared by the other particles.

- a stochastic factor in each particle's velocity makes them move through unknown problem space regions. This property combined with a good initial distribution of the swarm enable an extensive exploration of the

problem space and gives a very high chance of finding the best solutions efficiently.

2 How does it work?

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, 6, 3]. 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.

Even though many variations exist usually all have a fitness function. The specification of this function depends on the problem being optimized (especially in its dimensions) and as such we will simply refer to it as $f(x_i)$ being the short for $f(x_{i,0}, ..., f(x_{i,d})$. This function represents how good the i particle's position in the multidimensional space is relatively to the desired goal. With this it weights and binds the D dimensions to be optimized, given a problem modeled as an optimization one of d dimensions. By being a multi-dimensional algorithm, the positions and velocities of the particles we manipulated will then have d components, so we have positions as $x_i(x_{i,0}, ..., x_{i,d})$ and velocities as $v_i(v_{i,0}, ..., v_{i,d})$.

2.1 Global Best (original version)

In this algorithm we have a completely connected swarm, meaning that all the particles share information, any particle knows what is the best position ever visited by any particle in the swarm. Each particle has a position (1) and a velocity (2) which are calculated as follows:

$$x_{i,d}(it+1) = x_{i,d}(it) + v_{i,d}(it+1) \tag{1}$$

$$\begin{aligned} v_{i,d}(it+1) &= v_{i,d}(it) \\ &+ C_1 * Rnd(0,1) * [pb_{i,d}(it) - x_{i,d}(it)] \\ &+ C_2 * Rnd(0,1) * [gb_d(it) - x_{i,d}(it)] \end{aligned} \tag{2}$$

Caption:

i particle's index, used as a particle identifier;

d dimension being considered, each particle has a position and a velocity for each dimension;

$\boldsymbol{it}$ iteration number, the algorithm is iterative;

$\boldsymbol{x_{i,d}}$ position of particle i in dimension d;

$\boldsymbol{v_{i,d}}$ velocity of particle i in dimension d;

$\boldsymbol{C_1}$ acceleration constant for the cognitive component;

$\boldsymbol{Rnd}$ stochastic component of the algorithm, a random value between 0 and 1;

$\boldsymbol{pb_{i,d}}$ the location in dimension d with the best fitness of all the visited locations in that dimension of particle i;

$\boldsymbol{C_2}$ acceleration constant for the social component;

$\boldsymbol{gb_d}$ the location in dimension d with the best fitness among all the visited locations in that dimension of all the particles.

Regarding velocity update (equation 2) notice that it results from the sum of different components, each having a specific meaning. On the the first line we have the momentum component, which is the previous velocity. On the second line we have the cognitive component, which depends heavily on the particle's current distance to the best position it has ever visited. Finally on the third line we have the social component which depends heavily 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 these values are initialized can have an important impact on the algorithm's performance[3].

Regarding position, all the particles in the Swarm are positioned in the n-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 2 we present the pseudo-code of the basic PSO algorithm. In some parts which weren't fully detailed in the original presentation of PSO we give a simple solution, but several alternatives exist each having different impacts on the performance of the algorithm[3]. Such parts are the initialization of the particle´s positions and velocities in algorithm 1, and the stopping condition of the actual PSO 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]. 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[6].

Algorithm 1 Initialize

```
1: for each particle i in S do
2:    for each dimension d in D do
3:       //intialize all particles' position and velocity
4:       x_{i,d} = Rnd(x_min, x_max)
5:       v_{i,d} = Rnd(−v_max/3, v_max/3)
6:    end for
7:
8:    //intialize particle's best position
9:    pb_i = x_i
10:   //update the global best position
11:   if f(pb_i) < f(gb) then
12:      gb = pb_i
13:   end if
14: end for
```

Algorithm 2 Particle Swarm Optimization (Global Best)

```
1: //initialize all particles
2: Initialize
3: repeat
4:    for each particle i in S do
5:       //update the particle's best position
6:       if f(x_i) < f(pb_i) then
7:          pb_i = x_i
8:       end if
9:       //update the global best position
10:      if f(pb_i) < f(gb) then
11:         gb = pb_i
12:      end if
13:   end for
14:
15:   //update particle's velocity and position
16:   for each particle i in S do
17:      for each dimension d in D do
18:         v_{i,d} = v_{i,d} + C_1 * Rnd(0,1) * [pb_{i,d} − x_{i,d}] + C_2 * Rnd(0,1) * [gb_d − x_{i,d}]
19:         x_{i,d} = x_{i,d} + v_{i,d}
20:      end for
21:   end for
22:
23:   //advance iteration
24:   it = it + 1
25: until it < MAX_ITERATIONS
```

2.2 Local Best

When compared with the original algorithm this variation reduces the sharing of information between particles to a smaller neighborhood. Instead of each particle knowing the global best value, the swarm is divided into neighborhoods where particles only share their best value with their neighbors. Nonetheless it is important that neighborhoods overlap to enable convergence to the global best. This difference means that this version of the algorithm is slower to converge on a solution but it has a better coverage of the search space while also being less susceptible to local minima[1]. In the context of PSO a better coverage of the search space represents a higher probability that the best solution will be found, in this case at the cost of a lower convergence speed and therefore it might need more iterations to actually converge on a solution than the original version.

The neighborhoods can be easily created in two ways: statically by particle index or dynamically by spatial distance (computationally expensive). However, there are several social network topologies which specify concrete distributions of particles in neighborhoods with documented effects on the algorithm's performance [3].

2.3 Inertia Weight

This variation of the algorithm aims to balance two possible PSO tendencies (dependent on parameterization) of either exploiting areas around known solutions or explore new areas of the search space. To do so this variation focuses on the momentum component of the particles' velocity equation 2. Notice that if you remove this component the movement of the particle has no memory of the previous direction of movement and it will always explore close to a found solution. On the other hand if the velocity component is used, or even multiplied by a w (inertial weight, balances the importance of the momentum component) factor the particle will tend to explore new areas of the search space since it cannot easily change its velocity towards the best solutions. It must first "counteract" the momentum previously gained, in doing so it enables the exploration of new areas with the time "spend counteracting" the previous momentum. This variation is achieved by multiplying the previous velocity component with a weight value, w. Based on the author's experiments it has been documented that in order to obtain a good ratio between performance improvement and the algorithm's success in finding a the desired solution, the w value should be between [0.9 , 1.2]. Further testing showed that a variable w value increased even further this ratio, the tested case was with a linear decrease of w [6].

3 Applications

Since its initial development PSO has had an exponential increase in applications (either in the algorithm's original form or in an improved or differently parameterized fashion). A first look into the applications of this algorithm was presented by one of its creators Eberhart and Shi in 2001[2] 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[5] reported the exponential growth in applications and identified around 700 hundred documented works on PSO. He also categorized the areas of applications in 26 categories being those most researched (with around or more than 6% of the works reviewed by Poli):

- Antennas - especially in its optimal control and array design. Aside from there there are many others like failure correction and miniaturization.
- Control - especially in PI (Proportional Integral) and PID (Proportional Integral Derivative) controllers. These systems control a process based on its current output and its desired value (the difference between this two is its current error, P), past errors (I) and a prediction of future errors (D);
- Distribution Networks - especially in design/restructuring and load dispatching in electricity networks.
- Electronics and Electromagnetics - here the application are very disperse, but some of the main applications are: on-chip inductors, fuel cells, generic design and optimization in electomagnetics, semi-conductors optimization.
- Image and Video - this area is the one with most documented works in a wide range of applications, some examples are: face detection and recognition, image segmentation, image retrieval, image fusion, microwave imaging, contrast enhancement, body posture tracking.
- Power Systems and Plants - especially focused are power control and optimization. However, other specific applications are: load forecasting, photovoltaic systems control and power loss minimization.
- Scheduling - especially focused are flow shop scheduling, task scheduling in distributed computer systems, job-shop scheduling and holonic manufacturing systems. But other scheduling problems are addressed such as assembly, production, train and project.

The other categories identified are: Biomedical, Communication Networks, Clustering and Classification, Combinatorial Optimization, Design, Engines and Motors, Entertainment, Faults, Financial, Fuzzy and Neuro-fuzzy, Graphics and Visualization, Metallurgy, Modeling, Neural Networks, Prediction and Forecasting, Robotics, Security and Military, Sensor Networks, Signal Processing.

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] 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.

[3] A.P. Engelbrecht. *Computational intelligence: An introduction*. Wiley, 2007.

[4] J. Kennedy, R.C. Eberhart, et al. Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, volume 4, pages 1942–1948. Perth, Australia, 1995.

[5] R. Poli. An analysis of publications on particle swarm optimization applications. *Essex, UK: Department of Computer Science, University of Essex*, 2007.

[6] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 2002.