Particle Swarm Optimization: A Tutorial

James Blondin

September 4, 2009

1 Introduction

Particle Swarm Optimization (PSO) is a technique used to explore the search space of a given problem to find the settings or parameters required to maximize a particular objective. This technique, first described by James Kennedy and Russell C. Eberhart in 1995 [1], originates from two separate concepts: the idea of swarm intelligence based off the observation of swarming habits by certain kinds of animals (such as birds and fish); and the field of evolutionary computation.

This short tutorial first discusses optimization in general terms, then describes the basics of the particle swarm optimization algorithm.

2 Optimization

Optimization is the mechanism by which one finds the maximum or minimum value of a function or process. This mechanism is used in fields such as physics, chemistry, economics, and engineering where the goal is to maximize efficiency, production, or some other measure. Optimization can refer to either minimization or maximization; maximization of a function f is equivalent to minimization of the opposite of this function, $-f$ [4].

Mathematically, a minimization task is defined as:

$$\begin{aligned} &\text{Given } f : \mathbb{R}^n \to \mathbb{R} \\ &\text{Find } \hat{\mathbf{x}} \in \mathbb{R}^n \text{ such that } f(\hat{\mathbf{x}}) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathbb{R}^n \end{aligned}$$

Similarly, a maximization task is defined as:

$$\begin{aligned} &\text{Given } f : \mathbb{R}^n \to \mathbb{R} \\ &\text{Find } \hat{\mathbf{x}} \in \mathbb{R}^n \text{ such that } f(\hat{\mathbf{x}}) \geq f(\mathbf{x}), \forall \mathbf{x} \in \mathbb{R}^n \end{aligned}$$

The domain $\mathbb{R}^n$ of f is referred to as the *search space* (or *parameter space* [2]). Each element of $\mathbb{R}^n$ is called a *candidate solution* in the search space, with $\hat{\mathbf{x}}$ being the optimal solution. The value n denotes the number of dimensions of the search space, and thus the number of parameters involved in the optimization problem. The function f is called the *objective function*, which maps the search space to the *function space*. Since a function has only one output, this function space is usually one-dimensional. The function space is then mapped to the one-dimensional *fitness space*, providing a single fitness value for each set of parameters. This single fitness value determines the optimality of the set of parameters for the desired task. In most cases, including all the cases discussed in this paper, the function space can be directly mapped to the fitness space. However, the distinction between function space and fitness space is important in cases such as multiobjective optimization tasks, which include several objective functions drawing input from the same parameter space [2, 5].

For a known (differentiable) function f, calculus can fairly easily provide us with the minima and maxima of f. However, in real-life optimization tasks, this objective function f is often not directly known. Instead,

the objective function is a "black box" to which we apply parameters (the candidate solution) and receive an output value. The result of this evaluation of a candidate solution becomes the solution's fitness. The final goal of an optimization task is to find the parameters in the search space that maximize or minimize this fitness [2].

In some optimization tasks, called *constrained optimization* tasks, the elements in a candidate solution can be subject to certain constraints (such as being greater than or less than zero) [4]. For the purposes of this paper, we will focus on unconstrained optimization tasks.

A simple example of function optimization can be seen in Figure 1. This figure shows a selected region the function f, demonstrated as the curve seen in the diagram. This function maps from a one-dimensional parameter space—the set of real numbers $\mathbb{R}$ on the horizontal x-axis—to a one-dimensional function space—the set of real numbers $\mathbb{R}$ on the vertical y-axis. The x-axis represents the candidate solutions, and the y-axis represents the results of the objective function when applied to these candidate solutions. This type of diagram demonstrates what is called the *fitness landscape* of an optimization problem [2]. The fitness landscape plots the n-dimensional parameter space against the one-dimensional fitness for each of these parameters.

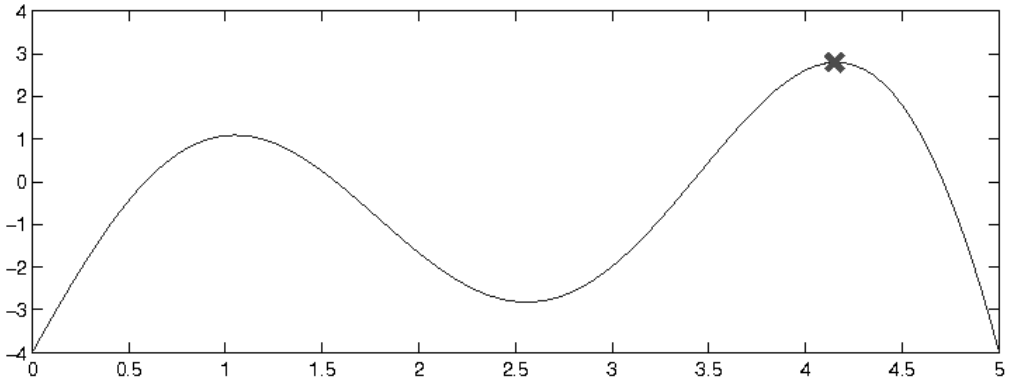

Figure 1: Function Maximum

Figure 1 also shows the presence of a *local maximum* in addition to the marked *global maximum*. A local maximum is a candidate solution that has a higher value from the objective function than any candidate solution in a particular region of the search space. For example, if we choose the interval [0,2.5] in Figure 1, the objective function has a local maximum located at the approximate value $x = 1.05$. Many optimization algorithms are only designed to find the local maximum, ignoring other local maxima and the global maximum. However, the PSO algorithm as described in this paper is intended to find the global maximum.

3 PSO Algorithm

The PSO algorithm works by simultaneously maintaining several candidate solutions in the search space. During each iteration of the algorithm, each candidate solution is evaluated by the objective function being optimized, determining the fitness of that solution. Each candidate solution can be thought of as a *particle* "flying" through the fitness landscape finding the maximum or minimum of the objective function.

Initially, the PSO algorithm chooses candidate solutions randomly within the search space. Figure 2 shows the initial state of a four-particle PSO algorithm seeking the global maximum in a one-dimensional search space. The search space is composed of all the possible solutions along the x-axis; the curve denotes the objective function. It should be noted that the PSO algorithm has no knowledge of the underlying objective function, and thus has no way of knowing if any of the candidate solutions are near to or far away

from a local or global maximum. The PSO algorithm simply uses the objective function to evaluate its candidate solutions, and operates upon the resultant fitness values.

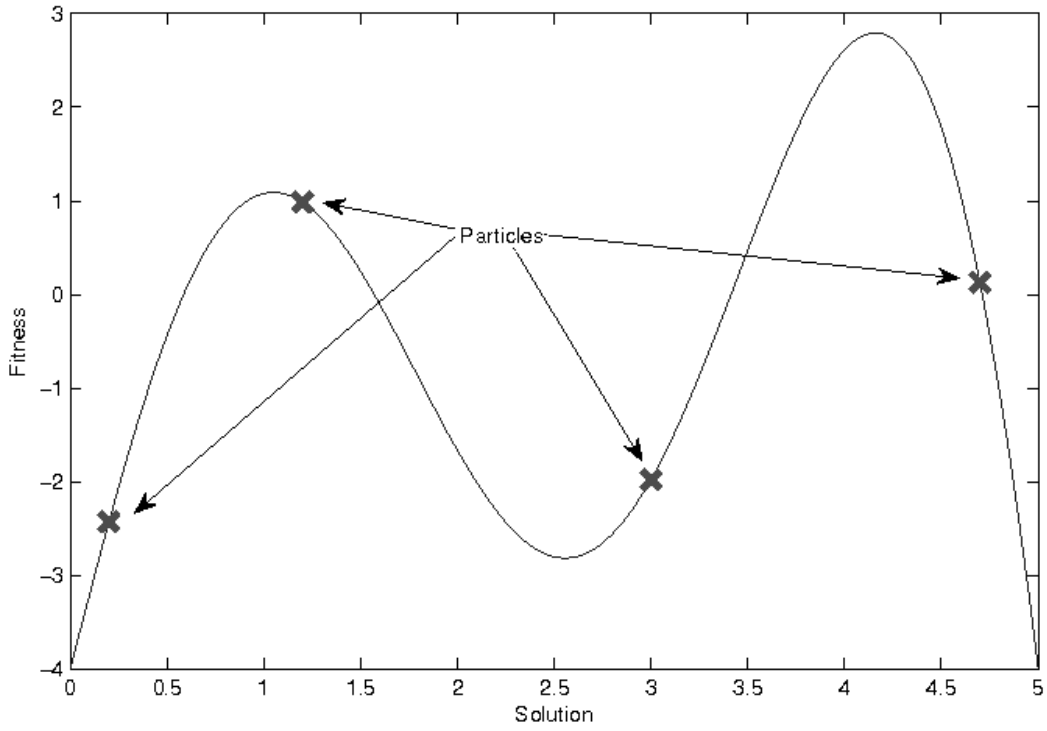

Figure 2: Initial PSO State

Each particle maintains its position, composed of the candidate solution and its evaluated fitness, and its velocity. Additionally, it remembers the best fitness value it has achieved thus far during the operation of the algorithm, referred to as the *individual best fitness*, and the candidate solution that achieved this fitness, referred to as the *individual best position* or *individual best candidate solution*. Finally, the PSO algorithm maintains the best fitness value achieved among all particles in the swarm, called the *global best fitness*, and the candidate solution that achieved this fitness, called the *global best position* or *global best candidate solution*.

The PSO algorithm consists of just three steps, which are repeated until some stopping condition is met [4]:

1. Evaluate the fitness of each particle
2. Update individual and global best fitnesses and positions
3. Update velocity and position of each particle

The first two steps are fairly trivial. Fitness evaluation is conducted by supplying the candidate solution to the objective function. Individual and global best fitnesses and positions are updated by comparing the newly evaluated fitnesses against the previous individual and global best fitnesses, and replacing the best fitnesses and positions as necessary.

The velocity and position update step is responsible for the optimization ability of the PSO algorithm. The velocity of each particle in the swarm is updated using the following equation:

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

The index of the particle is represented by i. Thus, $v_i(t)$ is the velocity of particle i at time t and $x_i(t)$ is the position of particle i at time t. The parameters w, c_1, and c_2 ($0 \leq w \leq 1.2$, $0 \leq c_1 \leq 2$, and $0 \leq c_2 \leq 2$) are user-supplied coefficients. The values r_1 and r_2 ($0 \leq r_1 \leq 1$ and $0 \leq r_2 \leq 1$) are random values regenerated for each velocity update. The value $\hat{x}_i(t)$ is the individual best candidate solution for particle i at time t, and $g(t)$ is the swarm's global best candidate solution at time t.

Each of the three terms of the velocity update equation have different roles in the PSO algorithm. The first term $wv_i(t)$ is the *inertia component*, responsible for keeping the particle moving in the same direction it was originally heading. The value of the *inertial coefficient* w is typically between 0.8 and 1.2, which can either dampen the particle's inertia or accelerate the particle in its original direction [3]. Generally, lower values of the inertial coefficient speed up the convergence of the swarm to optima, and higher values of the inertial coefficient encourage exploration of the entire search space.

The second term $c_1 r_1[\hat{x}_i(t) - x_i(t)]$, called the *cognitive component*, acts as the particle's memory, causing it to tend to return to the regions of the search space in which it has experienced high individual fitness. The *cognitive coefficient* c_1 is usually close to 2, and affects the size of the step the particle takes toward its individual best candidate solution $\hat{x}_i$.

The third term $c_2 r_2[g(t) - x_i(t)]$, called the *social component*, causes the particle to move to the best region the swarm has found so far. The *social coefficient* c_2 is typically close to 2, and represents the size of the step the particle takes toward the global best candidate solution $g(x)$ the swarm has found up until that point.

The random values r_1 in the cognitive component and r_2 in the social component cause these components to have a stochastic influence on the velocity update. This stochastic nature causes each particle to move in a semi-random manner heavily influenced in the directions of the individual best solution of the particle and global best solution of the swarm.

In order to keep the particles from moving too far beyond the search space, we use a technique called *velocity clamping* to limit the maximum velocity of each particle [4]. For a search space bounded by the range $[-x_{max}, x_{max}]$, velocity clamping limits the velocity to the range $[-v_{max}, v_{max}]$, where $v_{max} = k \times x_{max}$. The value k represents a user-supplied velocity clamping factor, $0.1 \leq k \leq 1.0$. In many optimization tasks, such as the ones discussed in the paper, the search space is not centered around 0 and thus the range $[-x_{max}, x_{max}]$ is not an adequate definition of the search space. In such a case where the search space is bounded by $[x_{min}, x_{max}]$, we define $v_{max} = k \times (x_{max} - x_{min})/2$.

Once the velocity for each particle is calculated, each particle's position is updated by applying the new velocity to the particle's previous position:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

This process is repeated until some stopping condition is met. Some common stopping conditions include: a preset number of iterations of the PSO algorithm, a number of iterations since the last update of the global best candidate solution, or a predefined target fitness value.

4 PSO Variations

Apart from the canonical PSO algorithm described in Section 3, many variations of the PSO algorithm exist. For instance, the inertia weight coefficient was originally not a part of the PSO algorithm, but was a later modification that became generally accepted. Additionally, some variations of the PSO do not include a single global best aspect of the algorithm, and instead use multiple global best that are shared by separate subpopulations of the particles.

Many more variations exist. For a full review of many of these modifications to the PSO algorithm, please see [4].

References

[1] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Piscataway, NJ, 1995. IEEE Press.

[2] James Kennedy, Russell Eberhart, and Yuhui Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[3] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 69–73, 1998.

[4] Frans van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, 2001.

[5] E. Zitzler, M. Laumanns, and S. Bleuler. A tutorial on evolutionary multiobjective optimization, 2002.