# Particle Swarm Optimization*

Jason Brownlee
jasonb@CleverAlgorithms.com
The Clever Algorithms Project
http://www.CleverAlgorithms.com

November 14, 2010
Technical Report: CA-TR-20101114b-1

## Abstract

The Clever Algorithms project aims to describe a large number of Artificial Intelligence algorithms in a complete, consistent, and centralized manner, to improve their general accessibility. The project makes use of a standardized algorithm description template that uses well-defined topics that motivate the collection of specific and useful information about each algorithm described. This report describes the Particle Swarm Optimization algorithm.

**Keywords:** `Clever, Algorithms, Description, Particle, Swarm, Optimization`

## 1 Introduction

The Clever Algorithms project aims to describe a large number of algorithms from the fields of Computational Intelligence, Biologically Inspired Computation, and Metaheuristics in a complete, consistent and centralized manner [1]. The project requires all algorithms to be described using a standardized template that includes a fixed number of sections, each of which is motivated by the presentation of specific information about the technique [2]. This report describes the Particle Swarm Optimization algorithm.

## 2 Name

Particle Swarm Optimization, PSO

## 3 Taxonomy

Particle Swarm Optimization belongs to the field of Swarm Intelligence and Collective Intelligence and is a sub-field of Computational Intelligence. Particle Swarm Optimization is related to other Swarm Intelligence algorithms such as Ant Colony Optimization and like the Genetic Algorithm it is a baseline algorithm for many variations, too numerous to list.

*© Copyright 2010 Jason Brownlee. Some Rights Reserved. This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 2.5 Australia License.

## 4 Inspiration

Particle Swarm Optimization is inspired by the social foraging behavior of some animals such as flocking behavior of birds and the schooling behavior of fish.

## 5 Metaphor

Particles in the swarm fly through an environment following the fitter members of the swarm and generally biasing their movement toward historically good areas of their environment.

## 6 Strategy

The goal of the algorithm is to have all the particles locate the optima in a multi-dimensional hyper-volume. This is achieved by assigning initially random positions to all particles in the space and small initial random velocities. The algorithm is executed like a simulation, advancing the position of each particle in turn based on its velocity, the best known global position in the problem space and the best known position known to a particle. The objective function is sampled after each position update. Over time, through a combination of exploration and exploitation of known good positions in the search space, the particles cluster or converge together around an optima.

## 7 Procedure

Algorithm 1 provides a pseudo-code listing of the Particle Swarm Optimization algorithm for minimizing a cost function.

## 8 Heuristics

- The number of particles should be low, around 20-40
- The speed a particle can move (maximum change in its position per iteration) should be bounded, such as to a percentage of the size of the domain.
- The learning factors (biases towards global and personal best positions) should be between 0 and 4, typically 2.
- A local bias (local neighborhood) factor can be introduced where neighbors are determined based on Euclidean distance between particle positions.
- Particles may leave the boundary of the problem space and may be penalized, be reflected back into the domain or biased to return back toward a position in the problem domain.
- An inertia or momentum coefficient can be introduced to limit the change in velocity.

## 9 Code Listing

Listing 1 provides an example of the Particle Swarm Optimization algorithm implemented in the Ruby Programming Language. The demonstration problem is an instance of a continuous function optimization that seeks $min f(x)$ where $f = \sum_{i=1}^{n} x_i^2$, $-5.0 \leq x_i \leq 5.0$ and $n = 3$. The optimal solution for this basin function is $(v_0, \ldots, v_{n-1}) = 0.0$. The algorithm is a conservative version of Particle Swarm Optimization based on the seminal papers. The implementation limits the velocity at a pre-defined maximum, and bounds particles to the search space, reflecting

**Algorithm 1**: Pseudo Code for the Particle Swarm Optimization algorithm.

**Input**: ProblemSize, $Population_{size}$
**Output**: $P_{g\_best}$

Population $\leftarrow 0$;
$P_{g\_best} \leftarrow 0$;
**for** $i = 1$ **to** $Population_{size}$ **do**
    $P_{position} \leftarrow$ `RandomPosition`($Population_{size}$);
    $P_{velocity} \leftarrow$ `RandomVelocity()`;
    $P_{cost} \leftarrow$ `Cost`($P_{position}$);
    $P_{p\_best} \leftarrow P_{position}$;
    **if** $P_{cost} \leq P_{g\_best}$ **then**
        $P_{g\_best} \leftarrow P_{p\_best}$;
    **end**
**end**
**while** $\neg$`StopCondition()` **do**
    **foreach** $P \in$ Population **do**
        $P_{velocity} \leftarrow$ `UpdateVelocity`($P_{velocity}$, $P_{g\_best}$, $P_{p\_best}$);
        $P_{position} \leftarrow$ `UpdatePosition`($P_{position}$, $P_{velocity}$);
        $P_{cost} \leftarrow$ `Cost`($P_{position}$);
        **if** $P_{cost} \leq P_{p\_best}$ **then**
            $P_{p\_best} \leftarrow P_{position}$;
            **if** $P_{cost} \leq P_{g\_best}$ **then**
                $P_{g\_best} \leftarrow P_{p\_best}$;
            **end**
        **end**
    **end**
**end**
**return** $P_{g\_best}$;

their movement and velocity if the bounds of the space are exceeded. Particles are influenced by the best position found as well as their own personal best position. Natural extensions may consider limiting velocity with an inertia coefficient and including a neighborhood function for the particles.

```ruby
def objective_function(vector)
  return vector.inject(0.0) {|sum, x| sum + (x ** 2.0)}
end

def random_vector(problem_size, search_space)
  return Array.new(problem_size) do |i|
    search_space[i][0] + ((search_space[i][1] - search_space[i][0]) * rand())
  end
end

def create_particle(problem_size, search_space, vel_space)
  particle = {}
  particle[:position] = random_vector(problem_size, search_space)
  particle[:cost] = objective_function(particle[:position])
  particle[:b_position] = Array.new(particle[:position])
  particle[:b_cost] = particle[:cost]
  particle[:velocity] = random_vector(problem_size, vel_space)
  return particle
end

```

```ruby
def get_global_best(population, current_best=nil)
  population.sort{|x,y| x[:cost] <=> y[:cost]}
  best = population.first
  if current_best.nil? or best[:cost] <= current_best[:cost]
    current_best = {}
    current_best[:position] = Array.new(best[:position])
    current_best[:cost] = best[:cost]
  end
  return current_best
end

def update_velocity(particle, gbest, max_v, c1, c2)
  particle[:velocity].each_with_index do |v,i|
    v1 = c1 * rand() * (particle[:b_position][i] - particle[:position][i])
    v2 = c2 * rand() * (gbest[:position][i] - particle[:position][i])
    particle[:velocity][i] = v + v1 + v2
    particle[:velocity][i] = max_v if particle[:velocity][i] > max_v
    particle[:velocity][i] = -max_v if particle[:velocity][i] < -max_v
  end
end

def update_position(particle, search_space)
  particle[:position].each_with_index do |v,i|
    particle[:position][i] = v + particle[:velocity][i]
    if particle[:position][i] > search_space[i][1]
      particle[:position][i] = search_space[i][1] -
          (particle[:position][i]-search_space[i][1]).abs
      particle[:velocity][i] *= -1.0
    elsif particle[:position][i] < search_space[i][0]
      particle[:position][i] = search_space[i][0] +
          (particle[:position][i]-search_space[i][0]).abs
      particle[:velocity][i] *= -1.0
    end
  end
end

def update_best_position(particle)
  if particle[:cost] <= particle[:b_cost]
    particle[:b_cost] = particle[:cost]
    particle[:b_position] = Array.new(particle[:position])
  end
end

def search(max_gens, problem_size, search_space, vel_space, pop_size, max_vel, c1, c2)
  pop = Array.new(pop_size) {create_particle(problem_size, search_space, vel_space)}
  gbest = get_global_best(pop, gbest)
  max_gens.times do |gen|
    pop.each do |particle|
      update_velocity(particle, gbest, max_vel, c1, c2)
      update_position(particle, search_space)
      particle[:cost] = objective_function(particle[:position])
      update_best_position(particle)
    end
    gbest = get_global_best(pop, gbest)
    puts " > gen #{gen+1}, fitness=#{gbest[:cost]}"
  end
  return gbest
end

if __FILE__ == $0
  problem_size = 3
  search_space = Array.new(problem_size) {|i| [-5, 5]}
  vel_space = Array.new(problem_size) {|i| [-1, 1]}
```

```
  max_gens = 200
  pop_size = 15
  max_vel = 5.0
  c1, c2 = 2.0, 2.0

  best = search(max_gens, problem_size, search_space, vel_space, pop_size, max_vel, c1, c2)
  puts "done! Solution: f=#{best[:cost]}, s=#{best[:position].inspect}"
end
```

Listing 1: Particle Swarm Optimization in the Ruby Programming Language

# 10 References

## 10.1 Primary Sources

Particle Swarm Optimization was described as a stochastic global optimization method for continuous functions in 1995 by Eberhart and Kennedy [3, 5]. It motivated as an optimization based on the flocking behavioral models of Reynolds [9]. Early works included the introduction of inertia [10] and early study of social topologies in the swarm by Kennedy [4].

## 10.2 Learn More

Poli, Kennedy, and Blackwell provide a modern overview of the field of PSO with detailed coverage of extensions to the baseline technique [8]. Poli provides a meta-analysis of PSO publications that focus on the application the technique, providing a systematic breakdown on application areas [7]. An excellent book on Swarm Intelligence in general with detailed coverage of Particle Swarm Optimization is "Swarm Intelligence" by Kennedy, Eberhart, and Shi [6].

# 11 Conclusions

This report described the Particle Swarm Optimization algorithm using the standardized template.

# 12 Contribute

Found a typo in the content or a bug in the source code? Are you an expert in this technique and know some facts that could improve the algorithm description for all? Do you want to get that warm feeling from contributing to an open source project? Do you want to see your name as an acknowledgment in print?

Two pillars of this effort are i) that the best domain experts are people outside of the project, and ii) that this work is subjected to continuous improvement. Please help to make this work less wrong by emailing the author 'Jason Brownlee' at `jasonb@CleverAlgorithms.com` or visit the project website at `http://www.CleverAlgorithms.com`.

# References


[1] Jason Brownlee. The clever algorithms project: Overview. Technical Report CA-TR-20100105-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[2] Jason Brownlee. A template for standardized algorithm descriptions. Technical Report CA-TR-20100107-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[3] 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*, pages 39–43, 1995.

[4] J. Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999.

[5] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proc. IEEE int'l conf. on neural networks Vol. IV*, pages 1942–1948, 1995.

[6] James Kennedy, Russell C. Eberhart, and Yuhui Shi, editors. *Swarm intelligence.* Morgan Kaufmann, 2001.

[7] R. Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 1:1–10, 2008.

[8] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization an overview. *Swarm Intelligence*, 1:33–57, 2007.

[9] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, 1987.

[10] Y. Shi and R. C. Eberhart. A modified particle swarm optimizers. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 69–73, 1998.