Survey of Update Rules for Particle Swarm Optimization

Alex Bechanko, Jason Hiebel, Jack Kelly

1 Introduction

Particle Swarm Optimization (PSO) is a numerical optimization technique which rose out of the simulation of social behavior is swarm-like settings. PSO involves the modeling of a set of particles existing within a bounded subset of our function domain, with each particle making decisions as to how to move in the domain based on a number of factors. These factors could include local influences such as the gradient at a particles position, global influences such as the best position found thus far from any particle in the swarm, and neighborhood influences which fall in-between.

These factors are encoded in to an *Update Rule*, which encodes the behavior of a particle based on its local, neighborhood, and global influences. Let p be a particle with update rule $U(p)$. A simple update rule might consist of just the gradient term $U(p) = \nabla f$, in which case each particle would simply follow a steepest descent path to a local minimum. A slightly more complicated update rule might take use of the *best-thus-far* information, with the best position as g, $U(p) = \nabla f + (g - x)$, in which case the particle has an attractive force to draw it away from a worse local minima.

Here we survey a few update rules for particle swarm optimization, including the basic update rule and some update rules defined by the authors.

2 Update Rules

Here we present the following update rules for use in testing:

- Classic and Random-Restart Swarms
- Gradient Shotgun
- Gravity Marbles

The two later methods take advantage metaphoric elements for providing useful and (hopefully) advantageous terms for use in update rules. Note that while we introduce each rule as a complete update rule, or a set of complete update rules, there are countless modifications and combinations which could be made.

2.1 Classic and Random-Restart Swarm

Let x, v be the particle's current position and velocity, respectively. Let b_l, b_g be the best-thus-far for the particle and for the whole swarm, respectively. Let $0 < r_p, r_g < 1$ weights randomized per each call to the update rule, and let ϕ_p, ϕ_g, ω be fixed weights. We can then define the update rule by the following differential equation:

$$v = U(p) = \omega v + \phi_p r_p(b_p - x) + \phi_g r_g(b_g - x).$$

In this way, the particle is influenced towards both its own best position, and towards the swarm's best position. These two influences are then averaged together with weights ϕ_p, ϕ_g and are randomized slightly by r_p, r_g.

This is the standard implementation of a particle swarm optimization update rule. However, this rule can be modified easily to include more or less decision making mechanisms. The desired complexity of an update rule is a highly contested point of inquiry.

Random-Restarts are a mechanism to prevent the convergence to a poor local minima by randomly, with a small probability, re-initializing a particle to a new point. In this way, there is a chance of initializing a particle in a valley which is lower than the swarm's current best, thus influencing the swarm to this new valley.

2.2 Gradient Shotgun

Gradient shotgun is an update rule which metaphorically takes advantage of the spray pattern one might observe after shooting a shotgun shell. Essentially, the particle advances in the direction of its gradient, but randomness allows for an amount of 'spread'. We have two different methods of defining such a direction: Pick d such that

1. $-\nabla f \cdot d > 0$, or
2. $d = -\nabla f * r$ where $*$ is the piecewise multiplication, and r is a random vector with components $0 < r_i < 1$.

Note that method 1 produces a significantly larger spread than method 2.

For the purposes of a full update rule, we combine this term with an attractive force similar to the classic algorithm. The gradient shotgun method we used for our tests is defined as

$$v = U(p) = \omega v + \phi_p r_p(b_p - x) + \phi_g r_g(b_g - x) + \phi_a d.$$

2.3 Gravity Marbles

Gravity marbles is a method which directly uses the gradient of a particles position to alter its velocity and acceleration. This has the effect of slowing a particle which is going uphill in order to get to a point which it is being attracted to, minimizing the possibility a particle will 'jump' from its current valley to a valley which is only a local minimum. We have two different methods defined:

1. $v = -\nabla f + \phi(f(b_g) - f(b_p))(x - b_g)$,
2. $v' = -\nabla f - \omega v - \phi(x - b_g)$.

The primary difference between gravity marbles and gradient shotgun is that update rules based on the gravity marbles ideology utilize ∇f directly, where as gradient shotgun introduce some method or randomness to their use.

3 Test Functions

We have chosen a set of functions which will allow for the progressive testing of our PSO update rule perform ace. The based functions chosen are Rosenbrock, Himmelblau, and Siam. In the case of Rosenbrock and Himmelblau, there exist multi-dimensional variants which we use in addition to the classic two-dimensional definitions.

Rosenbrock

The classic, two-dimensional Rosenbrock function can be defined as

$$R_2(\vec{x}) = (1 - x_0)^2 + 100(x_1 - x_0^2)^2.$$

The Rosenbrock function is useful in terms of testing as it has a single local minima.

Himmelblau

The classic two-dimensional Himmelblau function can be defined as

$$H_2(\vec{x}) = (x_0^2 + x_1 - 11)^2 + (x_0 + x_1^2 - 7)^2.$$

The Himmelblau function is useful in terms of testing as it has multiple local minima of the same value. Alternatively, we can alter the Himmelblau function with the addition of the quadratic centered at one of the local minima so that it becomes a distinct global minima. Additionally, we can define a higher dimensional variant by embedding a quadratic in higher dimensional space. We define this as

$$H_{27}(\vec{x}) = H_2((x_0, x_1)) + \frac{1}{2}\bar{x}A\bar{x} + b\bar{x},$$

where A is a randomly generated positive definite matrix, b is a randomly generated vector, and $\bar{x} = (x_2, x_3, \cdots, x_{26})$. Note that we generate A with uniformly distributed eigenvalues.

SIAM

The Siam function can be defined as

$$S(\vec{x}) = e^{\sin(50x_0)} + \sin(10e^{x_1}) + \sin(70\sin(x_0)) + \sin(\sin(80x_1)) - \sin(10(x_0 + x_1)) + \frac{1}{4}(x_0^2 + x_1^2).$$

and a higher dimensional variant

$$S_4(\vec{x}) = S_2((x_0, x_1)) + S_2((x_2, x_3))$$

The Siam function is useful as it has several local minimum.

4 Methodology

We use the *Mathematica* command `NMinimize` as a benchmark for comparison with our methods. Specifically, we compare the results of the optimization methods Nelder-Mead, Differential Evolution, Simulated Annealing, and Random Search (as provided by `NMinimize`) with each of the update rules we provide. See figure 1 for benchmark solutions provided by `NMinimize`. Note that for each of the methods provided by `NMinimize`, we use the default parameters provided by *Mathematica*.

Method	True Min.	Nelder-Mead	Diff. Evo.	Annealing	Random
Rosenbrock	0.0000	0.0000	0.0000	0.0000	0.0000
Himmelblau	0.0000	99.0462	99.0462	56.8024	0.0000
Himmelblau 27D	-36.0000	20.5465	-36.2559	-36.2559	-36.2559
SIAM	-3.3069	-2.2026	-3.2081	-1.8375	-2.3871
SIAM 4D	-6.6137	-3.1677	-5.8542	-3.7040	-4.3151

(*) Observing the $\vec{x}$ for these methods shows that they did not find the minimum and instead were in a very shallow valley.

Figure 1: Benchmarking Data

The PSO update rules will be tested using a series of 12 runs for each update rule, test function pair. We will then use the average and best values for each method in our analysis. For our PSO parameters, we will use 500 iterations with a swarm size of $32d$ where d is the dimension of the solution space for the test function. Other parameters for these methods were not the focus of our inquiry and thus we set them to sane and constant values for the purposes of our experimental procedure.

We focus on the optimality of a solution as opposed to the run time of the algorithm, as we are developing these update rules for the purpose of minimizing very difficult functions where standard methods find poor local minima often.

5 Results

Note, there is an attached appendix which contains the experimental results. Here we discuss high level comparisons of the methods based on these results.

The SIAM problem is a difficult function for the purposes of optimization. However, all of our update rules were able to find the correct global minima and they found this minima reliably. The SIAM function does expose a critical

weakness of the gradient based update rules (gradient shotgun, gravity marbles) however. These methods have an issue converging to the found minimum and instead oscillate in close proximity. This suggests that future work should investigate modifying the update rule and/or its parameters to detect such oscillation and work to reduce it.

Similar results appear in the Rosenbrock and Himmelblau functions. In general, smooth functions appeared to be more difficult for our algorithm. Counterintuitively the update rules which accounted for the gradient performed worse on smooth functions.

Every method performed poorly on the SIAM 4D problem, as the quantity of local minima is significantly large. Our consideration for success in regards to this test problem is a final function value of -6.0000, where the true global minimum is -6.6137. While none of our update rules were able to find the global minimum in any of our runs, most satisfied our criterion for success (with the exception of gravity shotgun method 2) at least once in the runs provided.