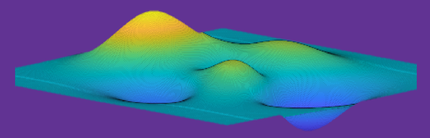
Swarm Behaviors and Optimization Methods

Dane Dunn and Dr. Jonathan Mitchell

Department of Mathematics, Stephen F. Austin State University, Nacogdoches, TX

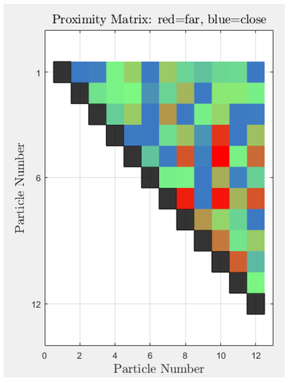


Abstract

Animals of all varieties in nature will interact in large groups, and exhibit swarming behaviors. These swarming techniques are done for many various reasons including avoiding predation, collection of resources, and the search of food. In all of these instances, each individual in the group follows only a few simple rules, and the result is a swarm that can complete complicated tasks and lead towards interesting cyclic behavior. This study will show how we model these swarming behaviors by giving simple instructions to particles that represent different animals, and how this swarm intelligence can be used to solve optimization problems. In particular, we close out this study with an in depth exposition of Particle Swarm Optimization (PSO), which is a numerical algorithm that minimizes a function of n variables.

Objectives of the Study

1. Simulate a variety of swarming behaviors



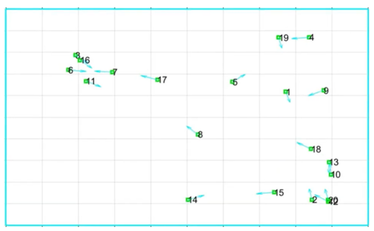
(a) Random & Inertia

(b) Proximity Matrix

(c) Bumping

Figure 1. (a) Particles can move with established velocity (inertia) or with random changes in velocity. If the rule for accel. involves the distance between particles, then a (b) proximity matrix is utilized accordingly. We also include (c) particles bumping into each other, which is truer to reality.

2. Predator-prey interactions
3. Fish chasing each other



(a) Sharks & Minnows

(b) Favorite Fish

Figure 2. A variety of behaviors can be simulated such as (a) Sharks and Minnows or (b) Chase an assigned fish

4. Implement the Particle Swarm Optimization (PSO) algorithm to optimize functions of multiple variables.

Essentials of Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) gives simple instructions to unintelligent particles in such a way that when each particle follows these rules and records what it knows, the entire swarm can obtain information no one particle could. To do this, we give each particle these rules to follow.

- 1.) Record the best COST value you have found, call this "best personal", B_P.
- 2.) Of all best personal values one will be the best, call this "best global", B_G.
- 3.) Draw position vectors to best personal, best global, and a velocity called "$\vec{v}$".
- 4.) Create a new vector that takes a step parallel to the three previous vectors to update new position.

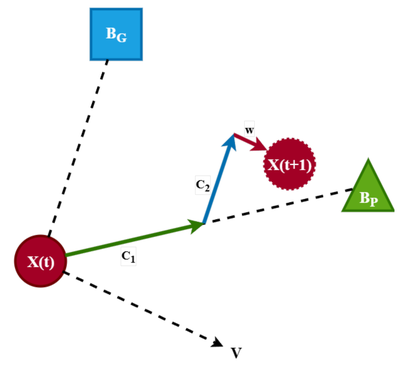


Figure 3. Notice how each particle updates its position Based on v, B_P, and B_G based on corresponding weights w, c_1, and c_2.

By repeating these 4 steps over many iterations, at least some of the particles will begin consolidating onto a local best cost value or the global (true) best cost value. As you might imagine, the more particles that are in a swarm means that the likelihood that any one particle approaches or exactly finds the best cost value increases. This can be easily shown in an example with very little particles.

$$\vec{v}(t+1) = w * \vec{v}(t) + c_1 * (\vec{p}(t) - \vec{x}(t)) + c_2 * (\vec{g}(t) - \vec{x}(t))$$

$$\vec{x}(t+1) = \vec{x}(t) + \vec{v}(t+1)$$

- $\vec{v}(t)$: particle's current velocity
- $\vec{x}(t)$: particle's current position
- $\vec{p}(t)$: particle's best position (best personal)
- $\vec{g}(t)$: swarm's best position (best global)
- w: inertia weight
- c_1: personal acceleration coefficient
- c_2: social acceleration coefficient
- $\vec{v}(t+1)$: particle's updated velocity
- $\vec{x}(t+1)$: particle's updated position

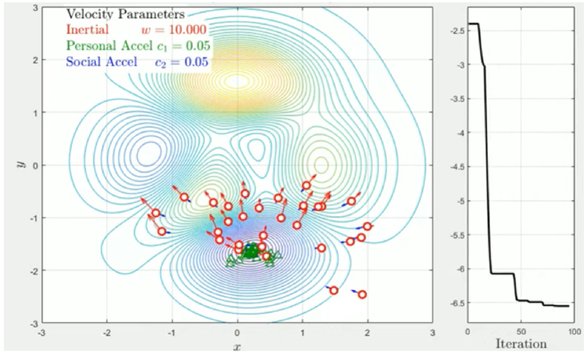
Traditional PSO Model

In the traditional PSO setup we introduce a dampening term called w_{damp} which reduces the size of the component, w, each iteration by some percentage. This is to help particles "fine tune" their final assessment of the best cost value by ensuring they do not overstep the true best cost value when they are close to it.

- PSO of One Particle
  With only one particle, it cannot move from its original starting position since it has no vector, $\vec{v}$, and its only known cost value is exactly what it is currently. This means that the best global and the best personal are equivalent, so the vector sum (acceleration) is the zero vector.
- PSO of Two Particles
  With the addition of only one more particle, the particles are now able to move. Because the amount of data two particles can collect is limited, it becomes very easy for the particles to give a "false positive". This false positive occurs when the starting points of the two particles are near a local best cost value but are too far from the true best cost value, and the component steps w, c_1, and c_2 are not sufficiently large enough to allow adequate exploration.

Cons of Traditional PSO's

- Can become stuck in a local best cost especially if the search space is complex or of high dimensions
- Can have slow convergence speed to the best cost value
- Can become computationally expensive with a large swarm
- Can be sensitive to initial parameters of the PSO



(a) Inertia (b) Global (c) Personal (d) Descent

Figure 4. PSO animations illustrating acceleration weights (a) inertia, (b) social, and (c) personal; contrasted with the (d) Steepest Descent Algorithm

Gradient Descent Model

In the "Steepest Descent" approach instead of each particle communicating with one another to find the best cost value, each particle instead moves independently of one another but all following the same rule. This rule is to move in the opposite direction to the gradient of the function at the current position of the particle, $-\nabla f(\vec{x})$. In this version of the PSO instead of having each particle look for the true best cost value, each particle will "flow downhill" to the local best cost value. This means that this version is exceptional not only at finding a true best cost value, but all local best cost values within a given search space.

Cons of Gradient Descent

- Must approximate partial derivatives in all directions.
- Can miss local optima depending on starting parameters.
- Can become stuck in saddle points.

```
%%%%%%% using steepest descent direction
if nVar ~= 2
    error('Steepest Descent only works with n=2 at the moment')
else
    S = particle(i).Position;
    % center difference approximation for partial derivatives
    fx = (ObjFunc([S(1)+h,S(2)])-ObjFunc([S(1)-h,S(2)])) / (2*h);
    fy = (ObjFunc([S(1),S(2)+h])-ObjFunc([S(1),S(2)-h])) / (2*h);
    gradient_approx = [fx,fy];
    particle(i).Velocity = - dt*gradient_approx;
end
```

Figure 5. Here is snippet of MATLAB code showing how the gradient is computed

Conclusion

Simulating dynamic swarming behaviors are not only helpful for modeling complex systems that are true to life like in animal swarms, but can also be used for complex optimization problems and machine learning. This process involves giving each particle in a swarm the same set of simple rules that will then result in an intelligent swarm. The traditional PSO model shown here becomes better at optimizing with the more particles it has in a swarm, but this results in an expensive computation. The gradient descent model is a better model for finding many optima within a search space, but it can become difficult to approximate partial derivatives of a function of many variables. We explored two different PSO's that each come with their own pros and cons, but there are many more PSO models each with their own usefulness and restrictions.

Future Research

- Simulations in three dimensions
- Incorporate bumping (elbow room) into the sharks and minnows (and other) simulations
- Using Particle Swarm Optimization to minimize functions of many variables

CONTACT
Dane Dunn Dunnda@jacks.sfasu.edu
Department of Physics, Engineering, and Astronomy

ACKNOWLEDGMENTS
This research was made possible by the Summer Undergraduate Research Experience (SURE) program of the College of Sciences and Mathematics and the Department of Physics, Engineering, and Astronomy at Stephen F. Austin State University.

REFERENCES
1. J. Kennedy et al., "Particle swarm optimization," Proceedings of ICNN'95, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4,
2. Alam, Mahamad Nabab. "(PDF) Particle Swarm Optimization: Algorithm and Its Codes in MATLAB." ResearchGate, 7 Mar. 2016, (link here)
3. 3Blue1Brown. "Gradient Descent, How Neural Networks Learn | Chapter 2, Deep Learning," (link here).
4.Yarpiz. "Particle Swarm Optimization in MATLAB - Yarpiz Video Tutorial - Part 1/3." (link here)