Particle Swarm Optimization with Flocking and Genetic Programming

Seth Chatterton and Professor Stephen Majercik

Computer Science Department, Bowdoin College, Brunswick, Maine

Introduction

Particle swarm optimization, or PSO, is a widely used optimization algorithm used to find the optimum of multidimensional functions. For example, if we wanted to find the lowest point in a landscape, PSO could help us find it. PSO works by sending out multiple particles that fly across the function space, which in our example is the landscape, and measure the function value at that point, in this case the height of the landscape. Each particle then looks at what function value its neighbors have, and adjusts its velocity at each iteration to head towards its neighbors if the neighbors have found a better spot. Much of PSO research is determining what a particle's neighborhood should be. In our research, we try to create good neighborhoods for our PSO particles using flocking behavior.

Flocking in nature determined mostly through three parameters: alignment, cohesion, and separation. Alignment means that all of the creatures in a flock tend to align their direction with one another, cohesion means they tend to move toward the average position of the group, and separation means they tend to not get too close to their neighbors. These three parameters create flocks of particles within a separate flocking space. Particles within a certain radius of one another are added to that particle's neighborhood, which is then used as the neighborhood in the PSO function space.

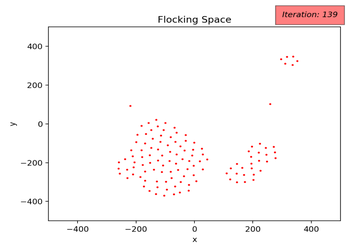
The next question is then how we determine what values of flocking parameters we use for our flocking space. For this, we used genetic programming. Essentially, our genetic program mimics evolution in nature. Our genetic program creates a population of random programs that set flocking parameters. These programs are then evaluated on how good the neighborhoods they produce through flocking behavior are. The best programs are saved for the next generation, are mixed with other good programs to create new programs, and occasionally mutated. This hopefully produces a program that can dynamically set flocking parameters to create very good neighborhoods for PSO.

Research Goals & Questions

- Find a more effective and efficient particle swarm optimization algorithm (PSO).
- Determine if flocking behavior is a useful mechanism for determining particle neighborhoods
- Determine if genetic programming is a useful mechanism for optimizing flocking behavior

Methods & Materials

The algorithm consists of three components: a particle swarm optimizer, a flocking systems that controls the neighborhood of each particle, and a genetic program, which determines how the flocking parameters change. A neighborhood is a set of particles that PSO uses to determine the velocity of any given particle. If a neighbor in a particle's neighborhood has found a more optimal position, the particle will move toward that neighbor. In this algorithm, flocking behavior determines a particle's neighborhood by using a separate flocking space. Each particle has a corresponding particle in the flocking space. In the flocking space, particle move according to the alignment, cohesion, and separation parameters described in the introduction, as well as maximum speed, normal speed, pace keeping, neighborhood radius, and random motion probability parameters. If a particle X in the flocking space is within the neighborhood radius of particle Y, X is added to Y's neighborhood. Below is a visualization of the flocking behavior.



Right: An example of flocking behavior exhibited by this algorithm. The red particles form flocks, and nearby particles become neighbors in the PSO algorithm.

To control the flocking behavior, we decided to use programs created through a genetic algorithm. These programs are trees consisting of nodes, and each node performs some action. Some of the important nodes are SEQUENCE, IF, ASSIGN, and VAR nodes. The program performs a traversal of the tree, performing the actions specified in the tree such as "assign cohesion a value of 0.5". PSO then runs some number of times to get an average for how well the tree performs, with each individual in the population receiving a fitness value according to how well it did. The fitness function used was

$$\frac{\mu_N - \mu_F}{\mu_F} \cdot I_{Fit}$$

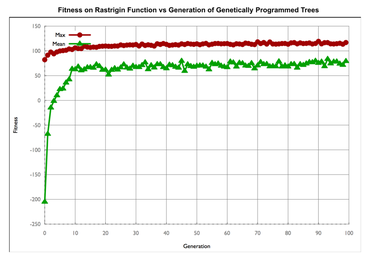
where μ_N is the average value found through PSO of the standard, non-flocking algorithm, μ_F is the average value found through PSO of our flocking algorithm, and I_{fit} is an arbitrary scaling factor called the fitness interval. Trees are then selected using tournament selection, such that trees that performed better in PSO and thus have a higher fitness are more likely to reproduce in the next generation. A large number of generations are performed, which ideally produce better and better flocking programs.

Results & Discussion

To test the algorithm, I used Bowdoin's High Performance Computing (HPC) Grid, which allowed me to test dozens of settings at a time by running tests in parallel. Parameters that I tested were the size of the genetic program, the crossover rate (how often to genetic programs get mixed together), the genetic program mutation rate, and the function to optimize.

In the final round of testing performed this summer, a population size of 1000 was used, and the programs were evolved for 100 generations. The crossover rate was set to 0.7, while the mutation rates tested were 0.001, 0.01, and 0.5. Each tree was tested 50 times before assigning a fitness value to it, in order to determine how good that particular program was on average. The baseline algorithm used for comparison was the standard PSO algorithm with a global best neighborhood topology. The PSO algorithm ran for 1000 iterations per test. Three different functions standard for testing PSO were used: Ackley, Rosenbrock, and Rastrigin. Tree sizes of depth 9 and 11 were tried, and various sized flocking spaces of 250, 500, and 1000 length squares were tried. Testing on the HPC Grid took 14 days to complete.

As a whole, the population of genetic programs evolved over time. Initially, most of the programs performed worse than the baseline PSO algorithm. Over time, however, they surpassed the fitness value of zero signaling that they performed equally as well, and the average fitness always stabilized to a positive value, indicating that on average the flock programs were performing better than the baseline algorithm.



Right: An example of the evolution of a population of genetic programs which control flocking behavior. Both maximum and mean fitness increase and then level off. This graph is of the population that produced the best Rastrigin value found described below.

However, even though these results surpassed our baseline, does not mean they are up to the state of the art. Compared to our best PSO values of 29.430 for Rastrigin, 19.258 for Rosenbrock, and 3.58E-4 for Ackley, Mohais et al. report values of 12.860, 26.560, and 1.32E-7 respectively[1]. Since lower is more optimal, the comparison of top algorithms in Mohais et al. shows that this algorithm is not up to par, with possibly one exception. Our algorithm actually performed better on Rosenbrock, but it should be noted that our result is the best of thousands of tests, and could be due to statistical variation. Further testing is needed to verify the repeatability of these results.

Acknowledgements

Sophia Ardell, '17, for her previous work on this project in the summer of 2017
Professor Stephen Majercik, for his previous work on this project
Funded by the Maine Space Grant Consortium. Bowdoin College is an affiliate of the Maine Space Grant Consortium.

References

[1] Mohais, Arvind S., et al. "Neighborhood Re-Structuring in Particle Swarm Optimization." *AI 2005: Advances in Artificial Intelligence Lecture Notes in Computer Science*, Dec. 2005, pp. 776–785., doi:10.1007/11589990_80.