Particle Swarm Optimization

Conner Brinkley

04.26.2020

Biologically-Inspired Computing

The University of Tennessee

INTRODUCTION

The purpose here is to investigate the performance of the particle swarm optimization (PSO) algorithm by implementing it into a simulator and measuring the results based on varying number of particles, inertia, cognitive, and social parameters. The resulting raw data for each variation is was outputted as CSV files to make it easier to graph the results in Excel.

PROCESS (& THEORY)

The PSO algorithm relies on the use of a fitness function to determine the best particle position both locally and globally within the entire population. The two fitness functions used in this experiment use the following parameters to calculate the quality of a particle's position, where p_x and p_y are the *x* and *y* coordinates of its position, and max_x and max_y are the world width and world height, respectively. For this project, the world width and world height both stay constant at 100.

$$mdist = \frac{\sqrt{{max_x}^2 + {max_y}^2}}{2}$$

$$pdist = \sqrt{(p_x - 20)^2 + (p_y - 7)^2}$$

$$ndist = \sqrt{(p_x + 20)^2 + (p_y + 7)^2}$$

Figure 1. Parameters for the fitness functions.

These values help define the two following fitness functions – two different ones were given to evaluate and compare the performance of each. The first fitness function in Figure 2 is associated with Problem 1, and the second function given in Figure 3 is associated with Problem 2.

$$Q(p_x, p_y) = 100 \cdot \left(1 - \frac{pdist}{mdist}\right)$$

Figure 2. Fitness function for Problem 1 with one maximum at (20, 7).

The above fitness function for Problem 1 has one maximum located at (20, 7), whereas the following fitness function for Problem 2 has two maxima: a global maximum at (20, 7) and a local maximum at (-20, -7).

$$Q(p_x, p_y) = 9 \cdot \max(0, 10 - pdist^2) + 10 \cdot \left(1 - \frac{pdist}{mdist}\right) + 70 \cdot \left(1 - \frac{ndist}{mdist}\right)$$

Figure 3. Fitness function for Problem 2 with maxima at (20, 7) (global) and (-20, -7) (local).

Now that the fitness functions are defined, the simulator starts out by initializing the global best with the best initial particle by using the selected fitness function to evaluate each one in the population. The *x* position and *y* position for each particle and global best are calculated and stored separately for each of the equations to follow.

Once the global best values are initialized, the PSO algorithm starts out by going through and updating each particle in the population. The update function updates the velocity, position, personal best, and global best. The following equation is used for updating the velocity, where v' is the new velocity, v is the previous velocity, c_1 is the cognition parameter, c_2 is the social parameter, $rand[0, 1]$ is a random number between 0 and 1 inclusive, $best_{local}$ and $best_{global}$ are the best overall particle positions locally and globally, respectively, and p is the particle's current position. In the actual implementation, this was broken up into *x* velocity and *y* velocity.

$$v' = inertia \cdot v + (c_1 \cdot rand[0, 1] \cdot (best_{local} - p)) + (c_2 \cdot rand[0, 1] \cdot (best_{global} - p))$$

Figure 4. Update rule for particle velocity.

Once the new velocity is set, it needs to be scaled to keep it within range of the world parameters. If it is not scaled, the positions will become too large and cause the simulator to have undefined behavior. The following equation is how the program accomplishes this, where v_{max} is the maximum velocity. This value is constant and was set at 1.0 for the duration of this experiment. Again, this scaling equation was broken up into *x* velocity and *y* velocity.

$$v = \frac{v_{max}}{\sqrt{{v_x}^2 + {v_y}^2}} \cdot v$$

Figure 5. Scaling velocity to keep it in range.

Once the algorithm finishes updating velocity, it adds the new velocity to the current position to get the new updated position. Then, it checks if the fitness of the new position is better than the fitness of the current local best position. If it is, then the new position is set as the local best position. The same applies for the global best position.

Now that the entire population is updated, the simulator calculates the average error in the *x* and *y* coordinates of the particles, given by the following equation, where *N* is the population size and p_i is the position of the current particle.

$$error = \sqrt{\frac{1}{2 \cdot N} \cdot \sum_{i=0}^{N} p_i - best_{global}{}^2}$$

Figure 6. Average error calculated as part of the stopping condition.

The algorithm continues to repeat all the above steps until a stopping condition has been met. The average error calculated above is used in the stopping condition to determine when the algorithm has converged. If the *x* or *y* error values drop below the threshold (in this experiment, the threshold is 0.01), then the state is considered to be converged and the algorithm stops. Otherwise it continues until the number of iterations hits the epoch limit, which is set at 300 for this project.

SIMULATOR

The program was written in C++ with the help of Dr. Van Hornweder's write up, and the source code can be found in the pso.cpp file in the source directory. The simulator takes five command line arguments with an optional sixth. The command line arguments are as follows: epochs (number of iterations), number of particles, inertia, cognition parameter, social parameter, and an optional problem number that determines the fitness function used. By default, the problem number is 1 if one is not specified. The simulator is executed by the following command after compiling with the given makefile:

USAGE: ./pso epochs number_of_particles inertia cognition social (problem_number)

Once the program finished calculating everything, it outputted all of the data as CSV files, which were then saved as Excel workbooks to graph the results using Excel's plotting tool.

RESULTS

The purpose of this was to measure the performance of the algorithm given a variety of parameters. The methods used to measure this can broken down into two simple categories: qualitatively and quantitatively. For each set of parameters given, the simulator generated two files: a scatterplot that shows where all of the particles converged (qualitative) and a graph of the average error over time (quantitative).

The first notable observation is that, on average, the first fitness function (Problem 1) performs better than the second (Problem 2). Generally, there was a higher amount of runs that did not converge for Problem 2 until the epoch limit was reached, given the same parameters as Problem 1. This was an interesting observation that did not really have an obvious answer. Initially, it was suspected to be because Problem 2 has two maxima instead of only one to focus on, so the next logical thing to do was to increase the number of particles to see if that would help converge faster. Unfortunately, it did not even when the number of particles was increased from 30 to 200 as the data below shows.

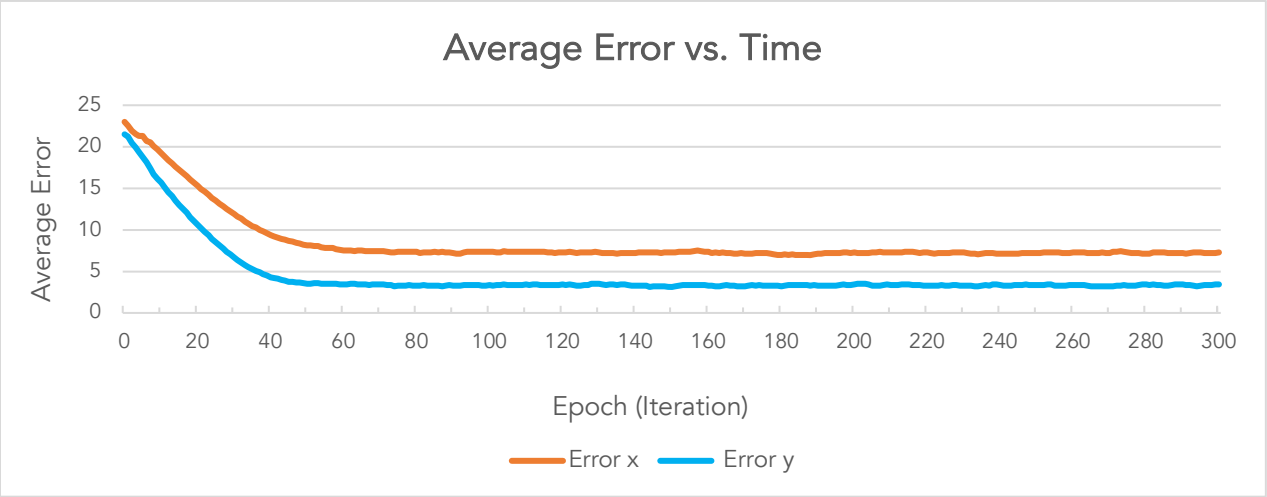

Figure 7. Quantitative graph for Problem 2 with parameters: 200 particles, 0.6 inertia, 2.0 social and cognitive.

It is still unclear why Problem 2 does not perform as optimally as Problem 1. Looking at the data for Problem 1, given the same parameters, it converges relatively quickly around epoch 102.

Figure 8. Quantitative graph for Problem 1 with parameters: 200 particles, 0.6 inertia, 2.0 social and cognitive.

The next notable observation is that neither Problem 1 nor Problem 2 normally converged when the cognitive or social parameters increased too much from the value 2.0 for each of them. It would be interesting to find out if they eventually converged given the maximum epoch was increased from 300, but for the scope of this experiment, it did not converge by that limit. It is hard to say why, but it was an interesting observation. However, based on a couple of test experiments, they seem to converge quicker when either the social or cognitive parameter was set to 1.0.

Finally, the last notable observation is that neither Problem 1 nor Problem 2 normally converged when the inertia was increased to 0.9. The only two inertia values that were tested across the two problems were 0.6 and 0.9, but there seemed to be a correlation with the higher inertia value and the lack of convergence. The following two graphs show the qualitative difference between the two inertia values for Problem 1.

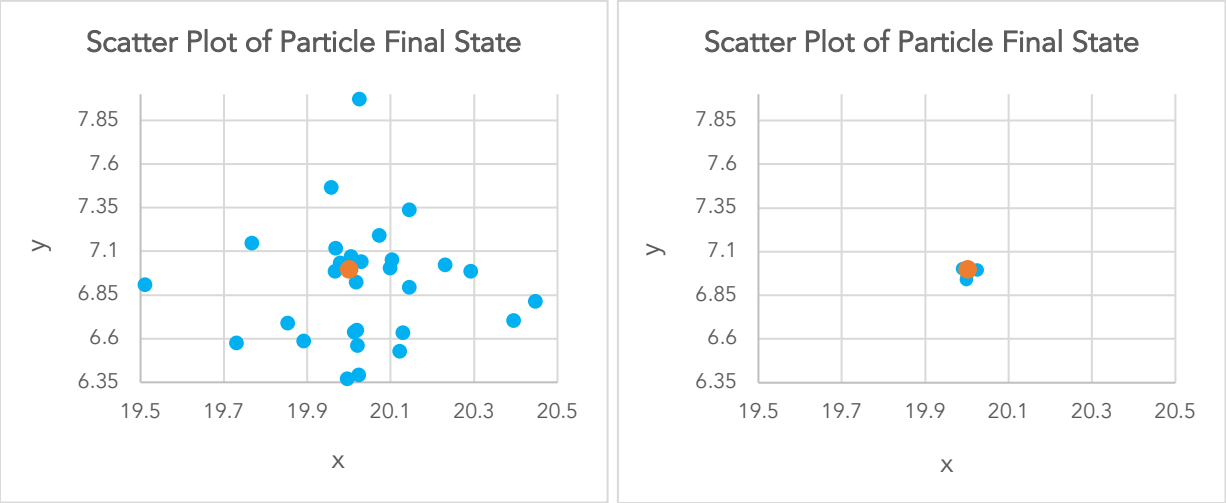

Figure 9. Qualitative graphs for Problem 1. Left has inertia set to 0.9, while right is set to 0.6.

As it is probably easy to see, the simulation on the right did a much better job of converging on the maxima (20, 7) (the red plot). Aside from inertia, the other parameters stayed constant with: 300 epoch limit, 30 particles, 2.0 cognition parameter, and 2.0 social parameter.

CONCLUSION

After analyzing the data and taking the most notable observations into consideration, it appears like the most optimal parameters for performance in the PSO algorithm are as follows:

- 30 particles
- $\leq$ 2.0 cognitive parameter
- $\leq$ 2.0 social parameter

It is important to note that these parameters are only valid for the given fitness functions. Also, the data recorded was not an average over several runs, so these may not be definitive given more data. In the future it would be interesting to write a shell script to average the data for each variation over 100 runs or so to see if these conclusions still hold up.