Exploration of the Particle Swarm Optimization Algorithm

Ronald Randolph
CS420: Biologically-Inspired Computation
April 26, 2019

Introduction

Particle Swarm Optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution towards a given level of quality. This is achieved through the use of a population of candidate solutions – also referred to as a swarm of particles. Using the records if their best-known position, each of these particles iteratively moves toward the best solutions. This algorithm is similar to genetic algorithms in that it uses an evolutionary population strategy to mimic to simulate biological processes. The creation of the particle swarm optimization is accredited to Kennedy and Eberhart in the early 1990's. The strategy of the PSO was inspired by the social behavior of animal species such as birds, ants, and fish that live in large colonies.

The PSO parameters have a significant impact on optimization performance. The goal of this experiment is to create an implementation of the particle swarm optimization algorithm and try to draw conclusions on how the parameters affect the performance of the algorithm. To achieve this, multiple executions will be done with several different values and combinations of the input parameters.

Theory and Methodology

The algorithm behind particle swarm optimization is relatively straightforward. A PSO implementation is composed of a population of N randomly initialized particles (solutions). These particles are usually stored in a list or array-like data structure.

After the initialization of the solution particles, it then undergoes the process of updating generations. With every generation, the particle position is updated using the global best position

and its personal best position. Using these two values the particles' velocity and position values are updated using the following equations (**Figure 1**).

$$x_{k+1}^i = x_k^i + v_{k+1}^i$$
$$v_{k+1}^i = w_k v_k^i + c_1 r_1 \left(p_k^i - x_k^i\right) + c_2 r_2 \left(p_k^g - x_k^i\right)$$

Figure 1: Particle Position and Velocity Update Functions

Where x_k is particle position, v_k is particle velocity, w_k is inertia constant, $p^i{}_k$ is best personal position, where $p^g{}_k$ is best global position, c_1 is the cognitive parameter, c_2 is the social parameter, and r_1 and r_2 are random numbers in [0,1].

The population of particles is iteratively updated until one of two stopping conditions is reached. The first condition checks that the error in the x and y coordinates drops below a threshold of 0.01. The x and y coordinate error can be calculated using the equations below (**Figure 2**). The second condition stops the process after the maximum number of iterations has been reached.

$$e_x \mathrel{+}= \left(x_k^i - p_k^g\right)^2 \qquad e_y \mathrel{+}= \left(y_k^i - p_k^g\right)^2$$
$$e_x = \sqrt{\left(\frac{1}{2N}\right) * e_x} \qquad e_y = \sqrt{\left(\frac{1}{2N}\right) * e_x}$$

Figure 2: Equations for Calculated Coordinate Error

The implementation created for the use of this exploration takes seven parameters: number of particles, inertia, cognition parameter, social parameter, world width, world height, and maximum velocity. During initialization, each particle's position is randomly initiated within the range of the maximum height and width of the world. The particle's velocity is initialized to 0. For the beginning of the process, each particle's personal best is set to their initialized position and the global best is set to the most correct particle's position. With each generation, the population of particles is updated as described above.

In this exploration, the aforementioned implementation will be tested on two separate problems (**Figure 3**). The variables within these problems – *mdist, pdist, and ndist* – is also described below (**Figure 4**).

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

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

Figure 3: Problems 1 and 2

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

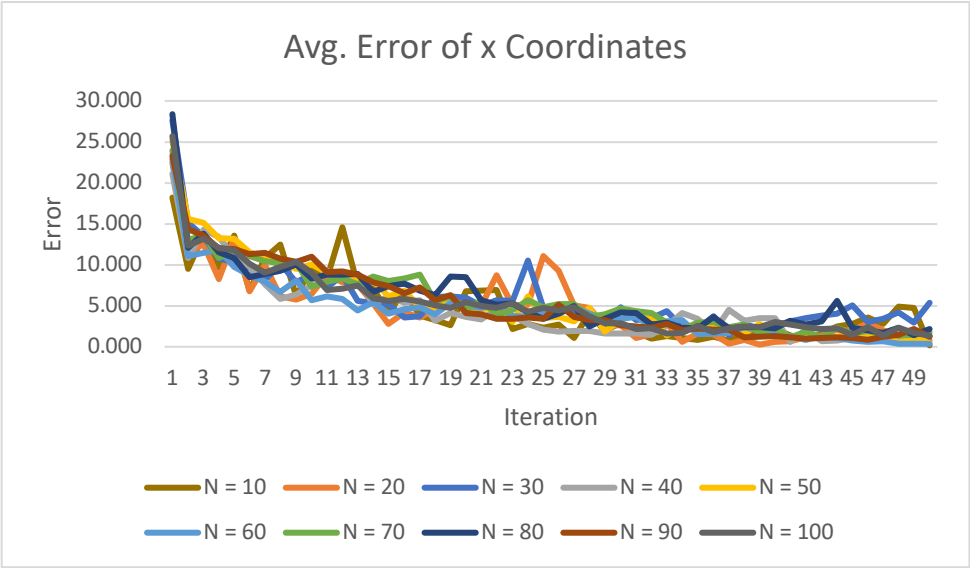
$$pdlist = \sqrt{(p_x - 20)^2 + (p_y - 7)^2}$$

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

Figure 4: Problem Variable Definitions

Results and Findings

To measure the performance of the particle swarm optimization algorithm, this experiment will be looking at the calculated error, percentage of converged particles, and the number epochs required for convergence. Throughout this observation, the input parameters – number or particles, inertia, cognitive, and social – will be varied and thoroughly assessed to find any correlations between variables. During these runs, the number of max epochs allowed, and the world size will remain the same.



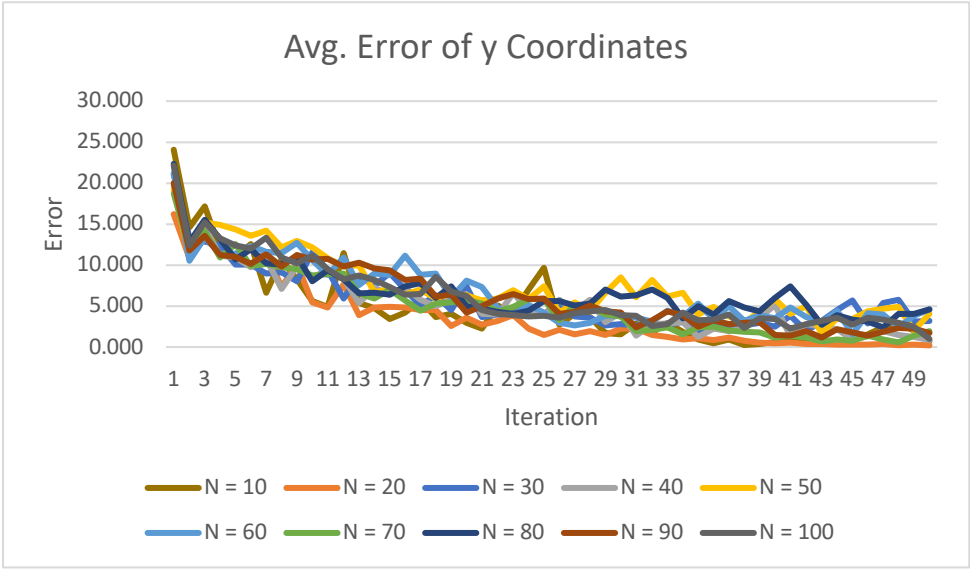


Figure 5: Average Error of x and y Coordinates by N

To explore the level of influence from the number of particles, the parameters W, C1, and C2 were kept at their default values. The graphs above (**Figure 5**) show an expected trend of ten runs – each with a varying value for the number of particles. As each iteration occurs, the coordinate error approaches 0. For the most part, it appears that the number of particles does not have an extremely large effect on the error measurements for problem 1.

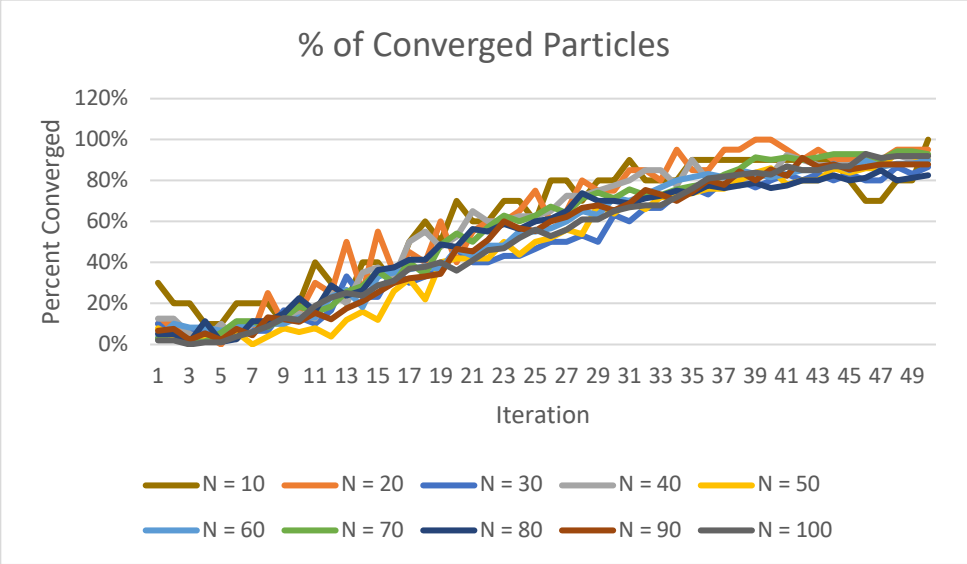
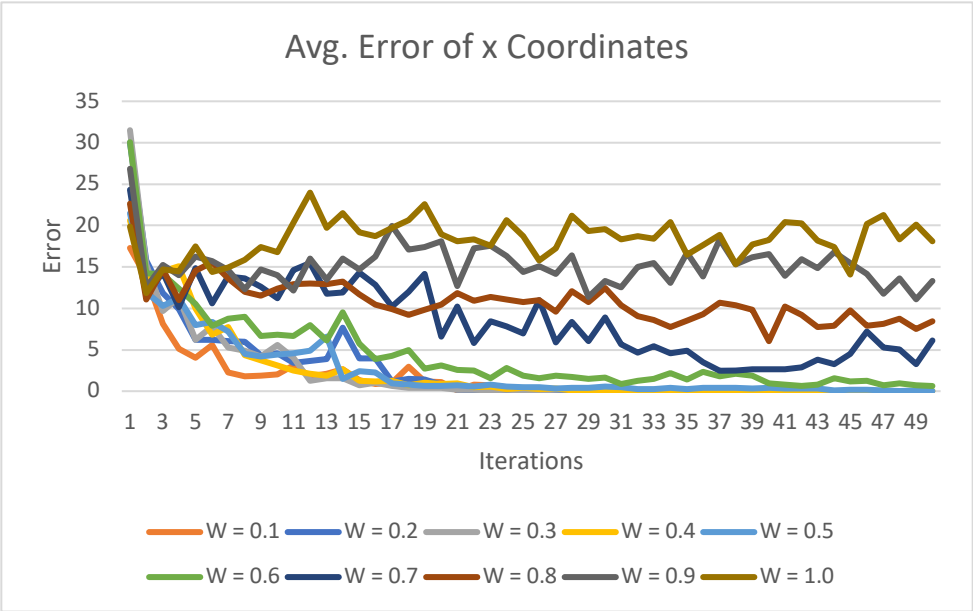


Figure 6: Percentage of Converged Particles by N

Furthermore, the number of converged particles was recorded for each of the iterations throughout each run (**Figure 6**). Once again, it appears that each run – regardless of number of particles – follows a similar increasing path of convergence.



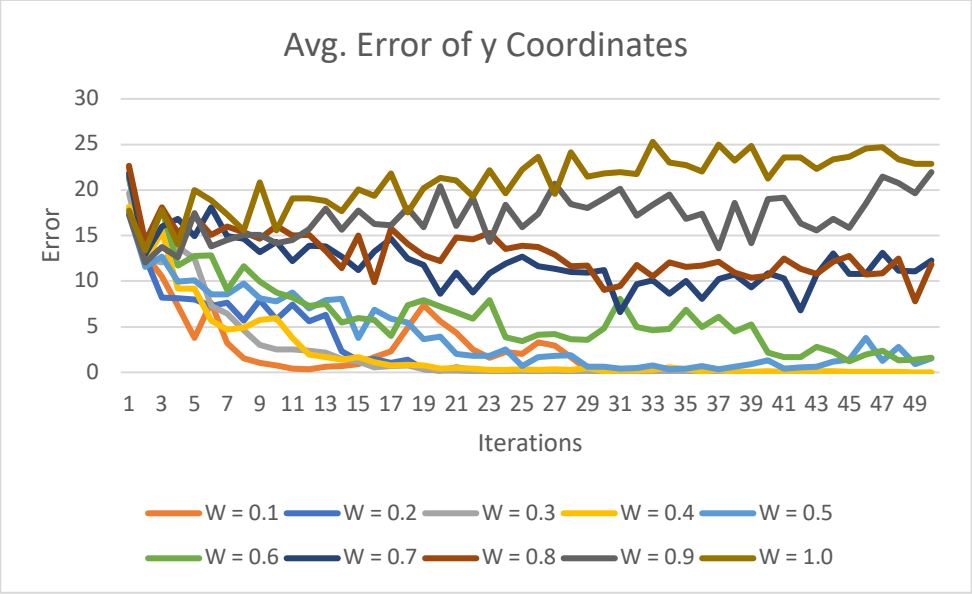


Figure 7: Avg. Error of x and y Coordinates by W

To explore the level of influence from inertia, the parameters N, C1, and C2 were kept at their default values. The graphs above (**Figure 7**) utilize data from ten separate runs to show a clear affect caused by inertia. As inertia begins to increase the average coordinate error also increases. From the recorded data, it appears that the most accurate values for inertia fall in the range [0.1, 0.5]. As each iteration occurs, one would expect to observe a general decrease in the coordinate error. However, for the higher levels of inertia ([0.5, 1]), the error seems to either increase over iterations or remain roughly the same.

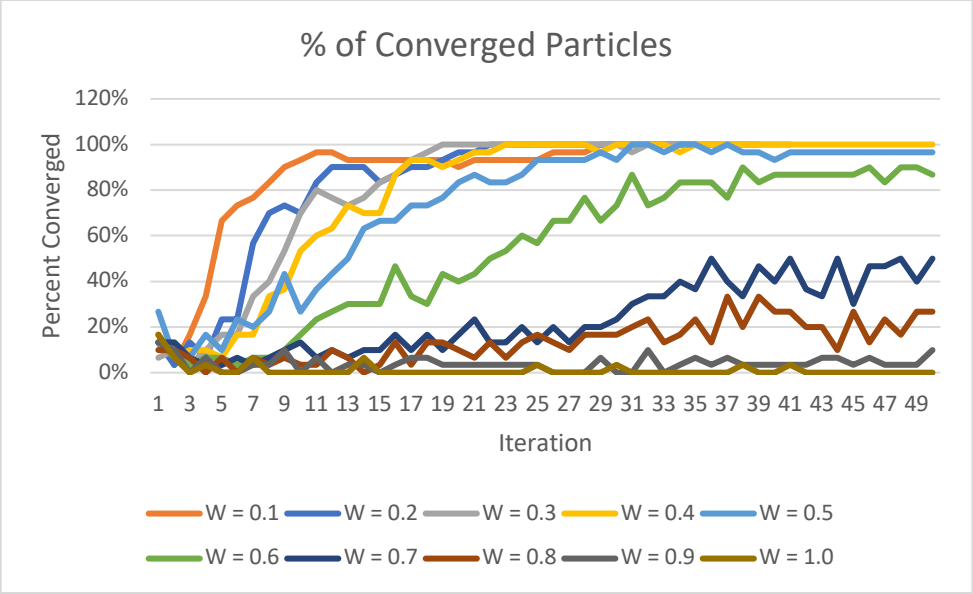


Figure 8: Percentage of Converged Particles by W

The chart above shows the percentage of converged particles with each iteration. As mentioned earlier, the lower values of inertia result in a higher rate of particle convergence. Inertia within the range of [0.1, 0.5] quickly begins to total convergence. However, higher levels of inertia result in extremely low levels of convergence (below 50%).

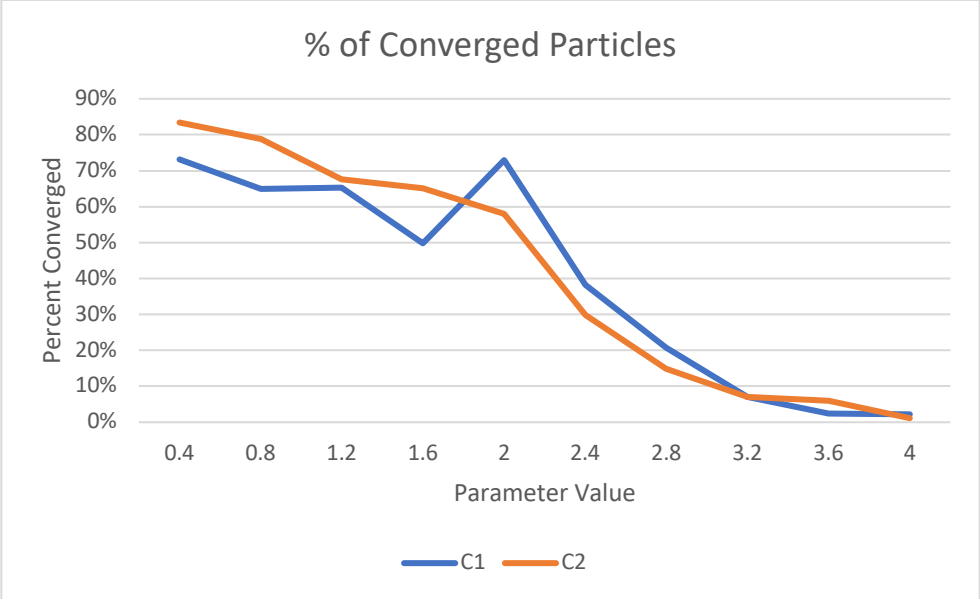
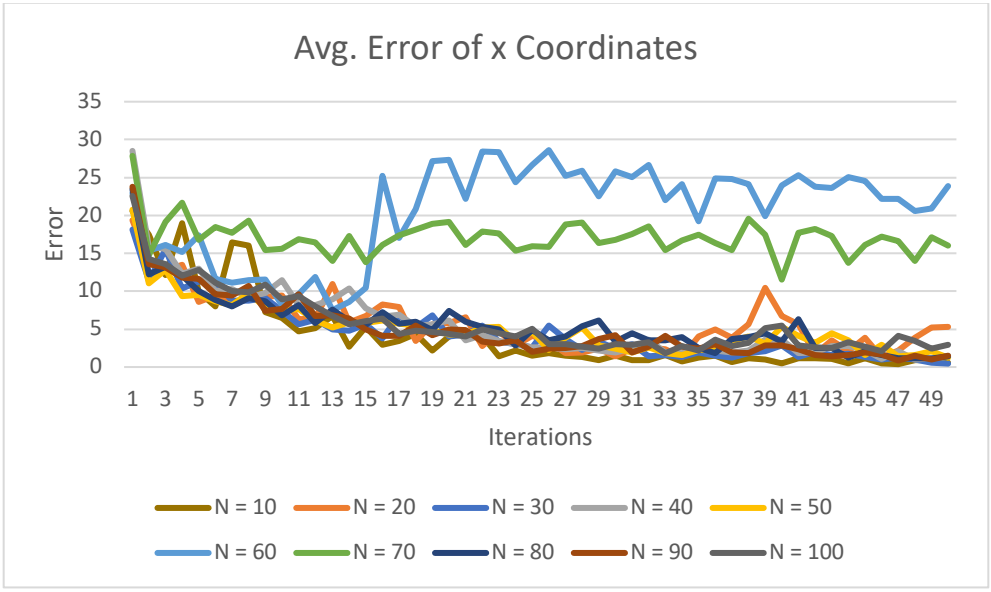


Figure 9: Percentage of Converged Particles by C1 and C2

The final observation for the first problem is the influence of the parameters C_1 and C_2. To observe this relationship, the other parameters – N and W – are kept at their default values. The chart above shows the average convergence over 20 runs each with 50 iterations. 10 runs were executed to observe and isolate each parameter. For the most part, the average convergence decreases as both C_1 and C_2 increases. However, there is a slight increase in average convergence when C_1 and C_2 is 2. This corresponds with the default values suggested to use for those parameters.



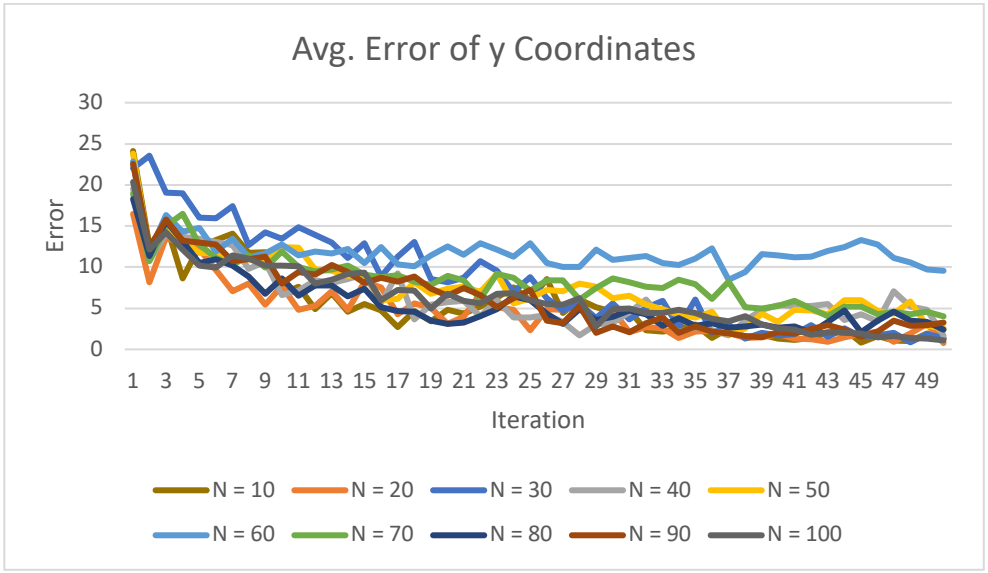


Figure 10: Average Error of x and y Coordinates by N

As with the first problem, it appears that the number of particles does not have a significant effect on the coordinate error. In the chart containing the x coordinate error, it appears that N=60 and 70 are outliers in the data.

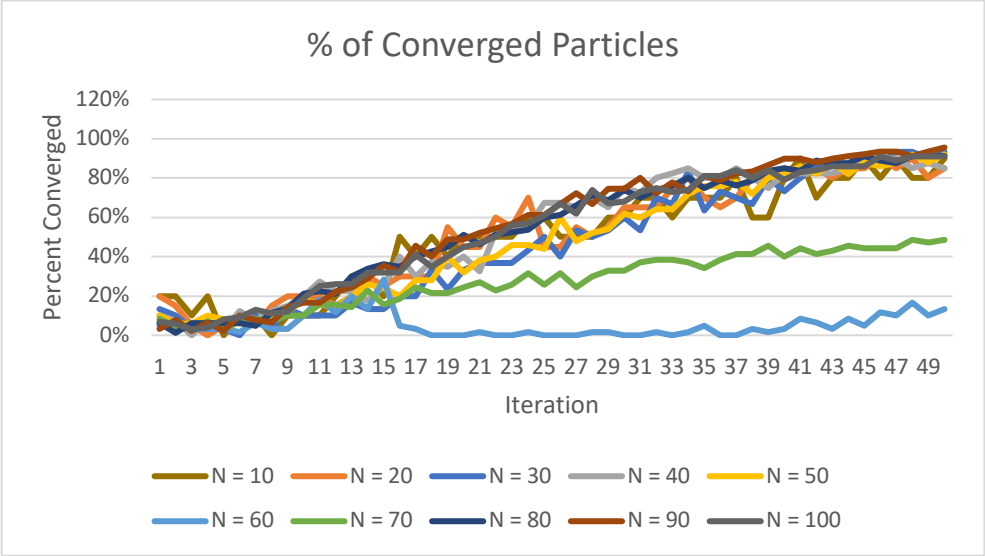
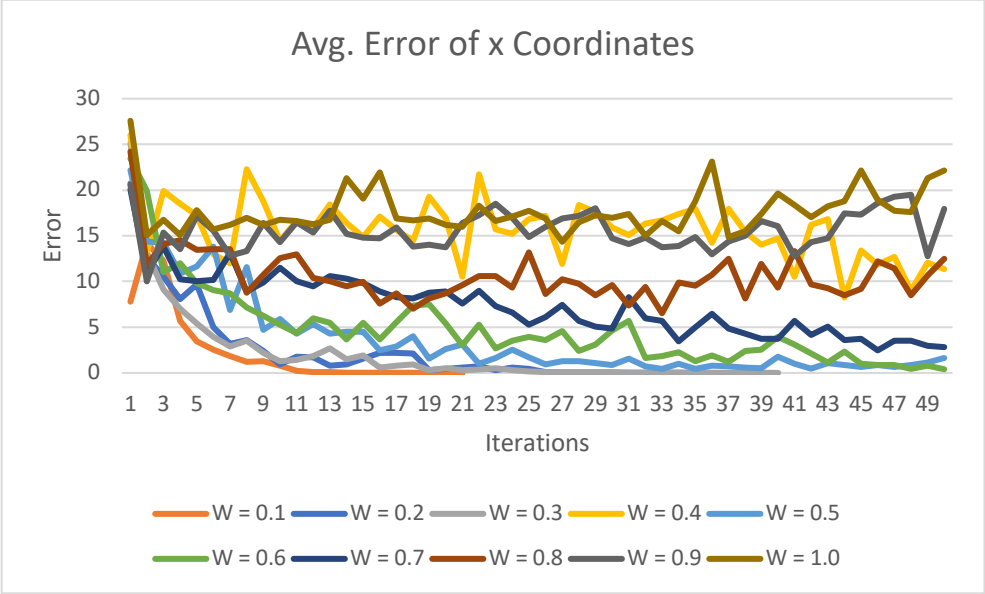


Figure 11: Percentage of Converged Particles by N

Figure 11 above reinforces the idea that the number of particles has only a slight influence on the PSO. With the exception of the two aforementioned outliers, convergence appears to increase similarly throughout each of the runs.



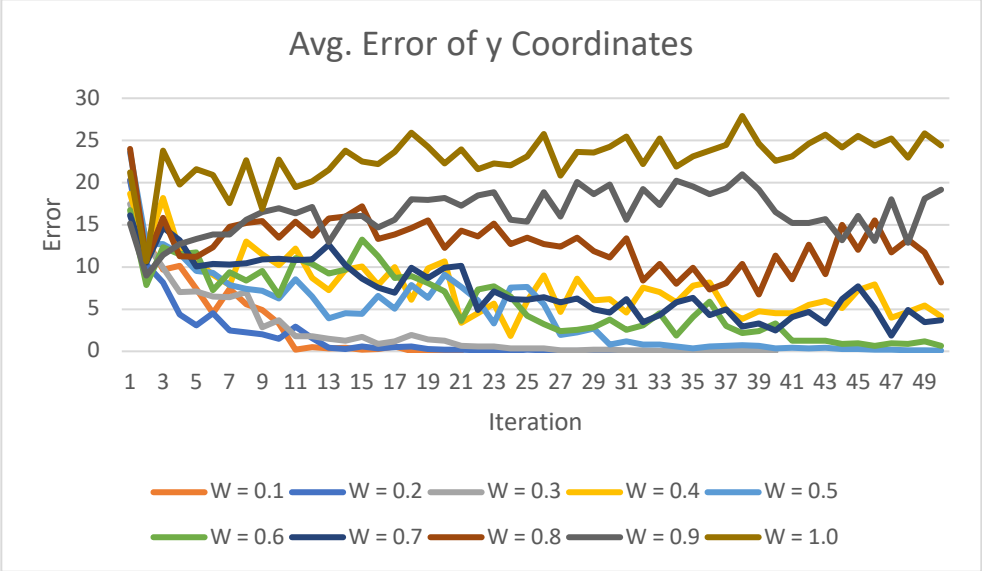


Figure 12: Average Error of x and y Coordinates by W

The chart above shows the percentage of converged particles with each iteration in relation to inertia. Once again, the lower values of inertia result in an overall lower coordinate error. Inertia within the range of [0.1, 0.4] results in a relatively low coordinate error. However, higher levels of inertia result in no change of coordinate error.

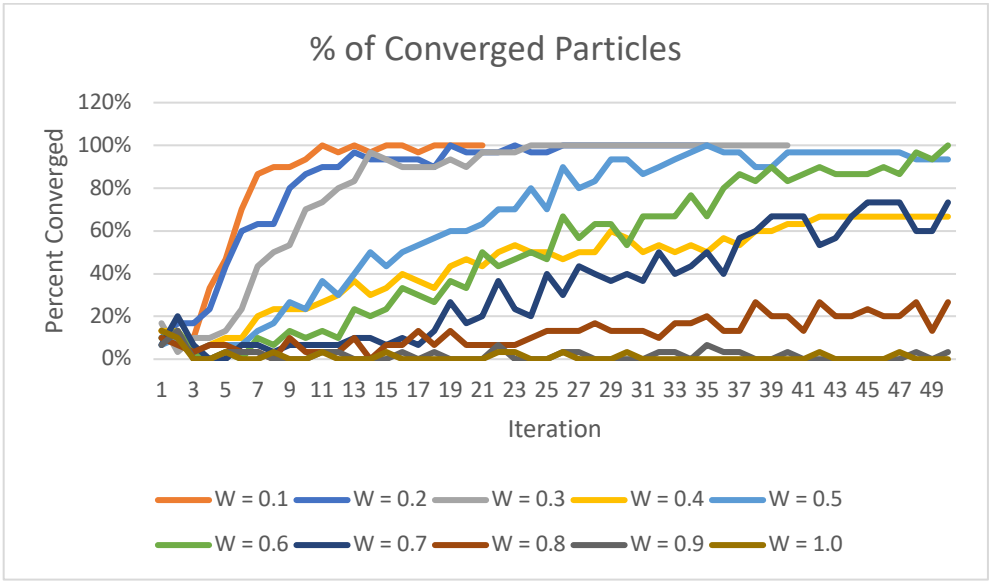


Figure 13: Percentage of Converged Particles by W

The chart above shows the percentage of converged particles with each iteration in relation to inertia. The lower values of inertia result in a quicker more complete convergence while the higher levels of inertia above 0.5 result in a much lower increase of convergence. Another note to be made is that the lower bounds of inertia [0.1, 0.3] result in compete convergence early and drop below the error threshold allowing an early termination of the PSO.

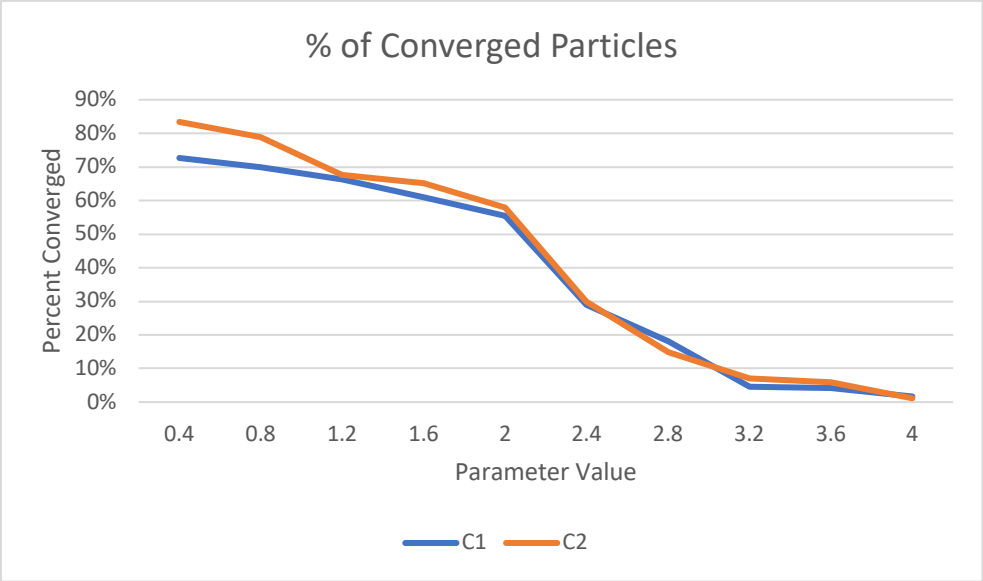


Figure 14: Percentage of Converged Particles by C1 and C2

The chart above (**Figure 14**) is much the same as the one for problem 1. However, the only difference in this chart is the lack of the notable spike when C_1 and C_2 = 2.

Conclusion

In conclusion, biological algorithms provide a unique way to minimize and optimize complex problems using the natural occurrences found in swarming animals and insects such as birds and ants. From the exploration, it is clear that the parameters N, W, C_1, and C_2 all have a distinct effect on the performance of the PSO algorithm. However, recorded data has shown that the most influential parameter by far is inertia. This is followed with C_1 and C_2 and the number of particles being the least influential in the algorithm's performance.