Solving the Rastrigin Function using Particle Swarm Optimization

Ralph Pereira 4554879

Abstract—This is an outline for the findings of running a vanilla PSO to solve the Rastrigin Function. Different particle sizes and dimensions were used through out the experiment, along with different velocity updating variables.

1 Introduction

Particle Swarm Optimization, in computer science, mimics a swarm of birds looking for an objective. The particles within the swarm look for an optimal solution, by checking local and global best solutions. The particles rely on a big enough swarm to help find an optimal solution.

2 Findings

Through the experiment different swarm sizes were used, between 30-40 were the fastest, but 200 particles got the best results. For a swarm size of 35 the average convergence was about 70, but with a swarm size of 200 the average convergence was around 30. The parameters are w = 0.729844, c1 = c2 = 1.496180 were used for the "best" testing, as those numbers are empirically proven to be the best.
The function used to calculate the fitness is the Rastrigin function, which goes as follows:

$$f(x_1 \cdots x_n) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$$

$$-5.12 \leq x_i \leq 5.12$$

Since this is a minimization problem, the resulting fitness needs to be as close to zero as possible. The lower the fitness, the better. Normally the fitness with 30 dimensions is around 400. Due to the summation the fitness will change based on the number of dimensions used.

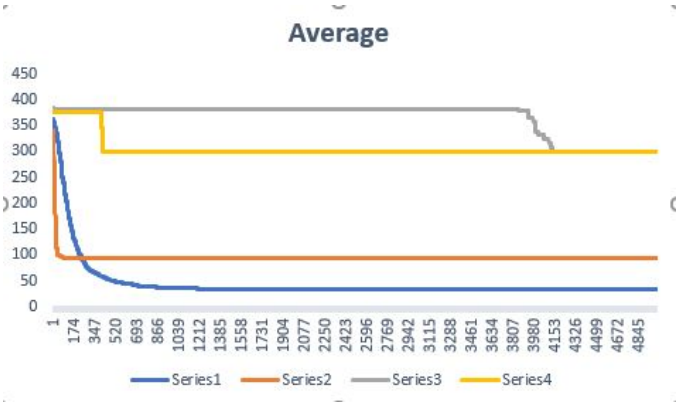
Summary of Fitness Values over 10 Tests

Configuration	Mean	Standard Dev	Median
Random Search	0.0	0.0	0.0
PSO-1	34.77	7.75	34.87
PSO-2	94.14	16.06	97.31
PSO-3	300	0.0	300
PSO-4	300	0.0	300

- PSO-1: w = 0.729844, c1 = c2 = 1.496180
- PSO-2: w = 0.4, c1 = c2 = 1.2
- PSO-3: w = 1.0, c1 = c2 = 2.0
- PSO-4: w = 1.0, c1 = c2 = 2.0

As the table shows the random search massively under-performed compared to the first two. PSO-3 and PSO-4 results in a constant output because most of the particles leave the search space and never converge on a new result. The algorithm had to use try and catch blocks in order to ensure the fitness value did not go too high, as the parameters did throw some exceptions.
The best results came out of PSO-1, with it normally settling down to around 35. The global best was updated after each particle was moved, which slowed down the program a bit. I believe this found the best global minimum slightly faster because it was constantly trying to search for it.

3 FIGURE



(a) Average

Fig. 1. The average of 10 tests

4 CONCLUSION

This was is interesting problem to solve because technically the fitness just needs to be zero, so it's possible to reverse function and just get the x values.

REFERENCES

[1] S. Russell & P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd Edition. Prentice Hall, 2010.