Central Washington University

CS 471 OPTIMIZATION

Spring 2019

Project 4 Report

Author:
Hermann Yepdjio

Supervisor:
Dr. Donald Davendra

May 22, 2019

Contents

1 Introduction

Project 4 was about optimizing 18 standard benchmark functions namely Schwefel, De Jong 1, Rosenbrock's Saddle, Rastrigin, Griewangk, Sine Envelope Sine Wave, Stretch V Sine Wave, Ackley One, Ackley Two, Egg Holder, Rana, Pathological, Michalewicz, Master's Cosine Wave, Quartic, Levy, Step and Alpine. For this purpose, 3 optimization algorithms were to be implemented then applied to those functions. Those algorithms are: Particle Swarm Optimization (PSO), Firefly Algorithm (FA) and Harmony Search (HS) . After they have been implemented, those algorithms were run on each of the 18 benchmark functions using randomly generated data. Statistics for each algorithm were computed and stored in a tabular format and they will be discussed then analyzed later on in this report. However, in order to observe how the fitness of each objective function evolves across iterations, the first experiment was picked for each algorithm and its results were plotted. Those plots are also analyzed later on in this report.

2 Results

2.1 Particle Swarm Optimization

2.1.1 Values used for the parameters

The PSO program was run using the following values:

- dimension: 30
- population size: 500
- number of iterations: 500
- number of experiments: 30
- c1: 0.3
- c2: 0.8
- k: 0.00001(for First De Jong, Rosenbrock and Quartic) and 0.1 (for the other functions)

After experimenting with different values for c1, c2 and k, it appeared that only varying k could make a significant impact on the outcome of the objective functions. Therefore, 0.00001 was chosen for First De Jong, Rosenbrock and Quartic and 0.1 was chosen for the rest as they seemed to work pretty well for their respective functions.

2.1.2 Statistics

Table 1: Statistics for PSO (30 experiments)

	Average	**Std_Dev**	**Range**	**Median**	**Avg_Time**	**Avg #calls\exp**
f1 Schwefel	0.20	0.51	2.65	0.05	396.60	250500
f2 De Jong 1	0.00	0.00	0.00	0.00	261.70	250500
f3 Rosenbrock	4.31	5.66	24.22	1.80	380.10	250500
f4 Rastrigin	220176.30	47554.73	208458.00	226013.00	441.23	250500
f5 Griewangk	1.01	0.00	0.00	1.01	499.87	250500
f6 Sine Envelope	-20.49	0.65	2.41	-20.57	741.33	250500
f7 Stretch V Sine	10.15	0.00	0.00	10.15	747.03	250500
f8 Ackley One	5.89	3.69	12.01	6.69	633.63	250500
f9 Ackley Two	0.32	0.35	1.30	0.25	881.27	250500
f10 Egg Holder	-4880.27	458.96	1927.40	-4899.37	629.73	250500
f11 Rana	-3310.65	302.34	1235.29	-3313.05	926.03	250500
f12 Pathological	0.00	0.00	0.00	0.00	721.10	250500
f13 Michalewicz	-7.88	0.55	2.32	-7.90	1105.03	250500
f14 Masters' Cosine	-15.67	0.00	0.00	-15.67	693.50	250500
f15 Quartic	0.00	0.00	0.01	0.00	681.30	250500
f16 Levy	0.00	0.00	0.00	0.00	721.77	250500
f17 Step	7.39	0.11	0.40	7.36	271.03	250500
f18 Alpine	0.03	0.03	0.15	0.02	376.23	250500

2.1.3 Statistics analysis

As table 1 above shows, PSO produced pretty good results for all the objective functions. Those results are even better (except for the Rastrigin function (220176.30 vs -89999.93)) than those produced by the Differential Evolution Algorithm in the previous assignment.

2.1.4 plots

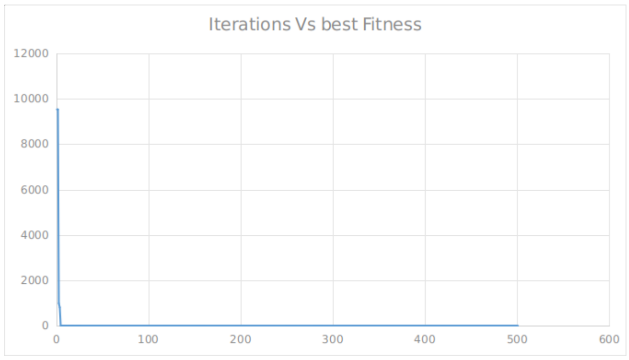

Figure 1: Schwefel (Experiment #1)

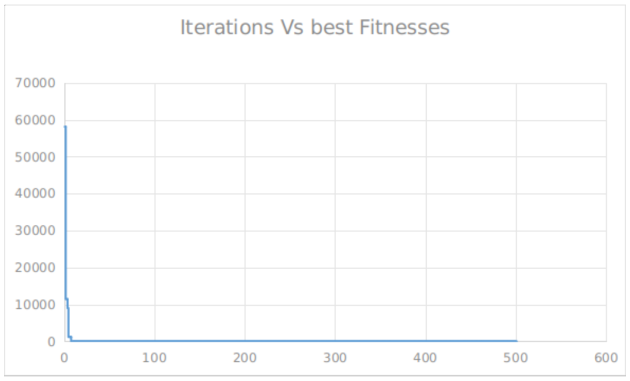

Figure 2: De Jong 1 (Experiment #1)

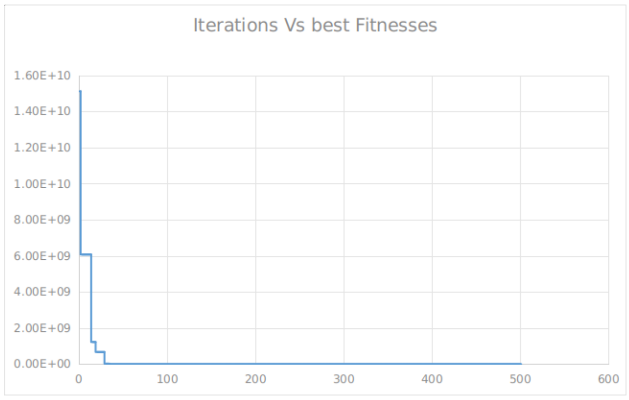

Figure 3: Rosenbrock (Experiment #1)

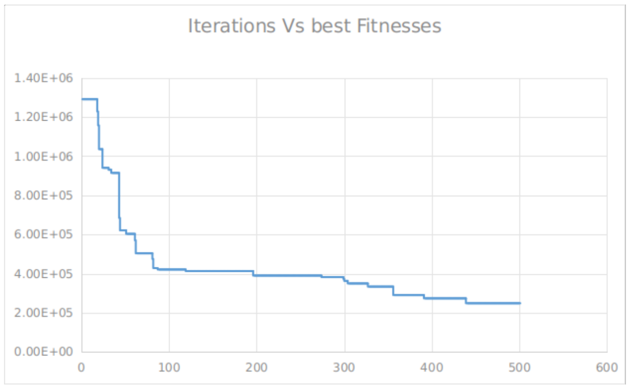

Figure 4: Rastrigin (Experiment #1)

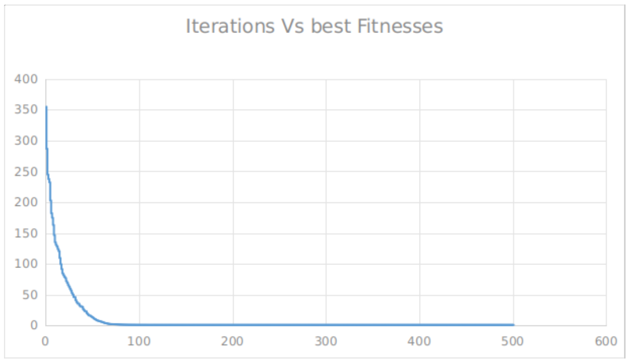

Figure 5: Griewangk (Experiment #1)

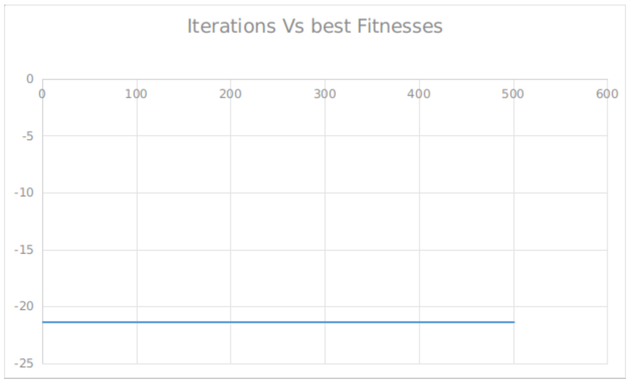

Figure 6: Sine Envelope (Experiment #1)

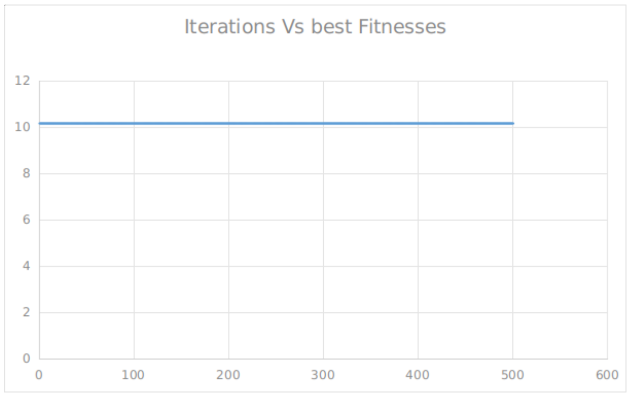

Figure 7: Stretch V Sine (Experiment #1)

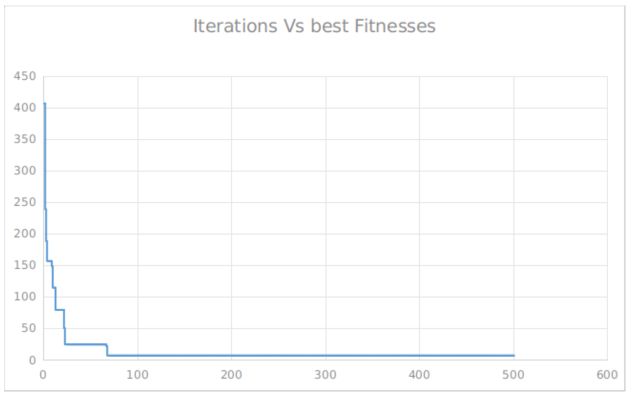

Figure 8: Ackley One (Experiment #1)

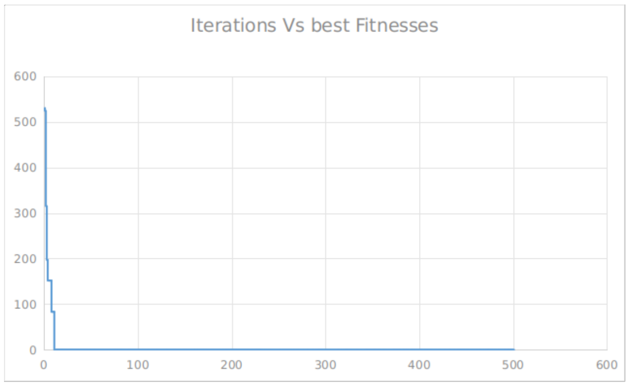

Figure 9: Ackley Two (Experiment #1)

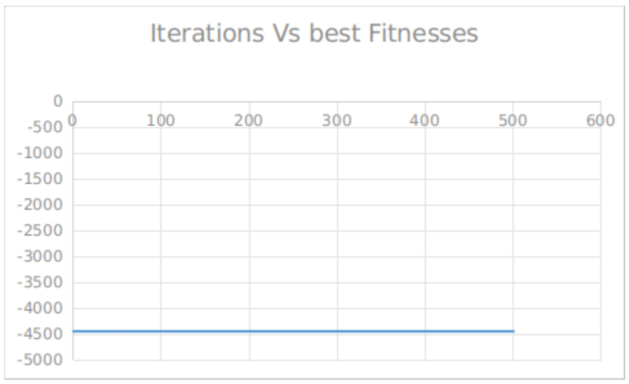

Figure 10: Egg Holder (Experiment #1)

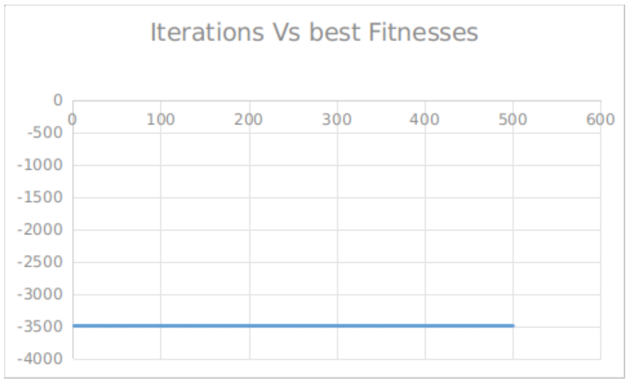

Figure 11: Rana (Experiment #1)

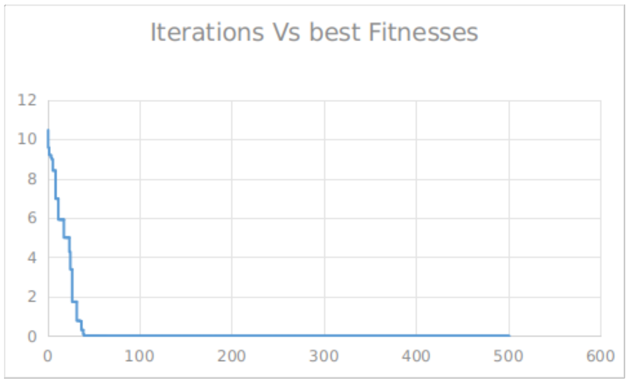

Figure 12: Pathological (Experiment #1)

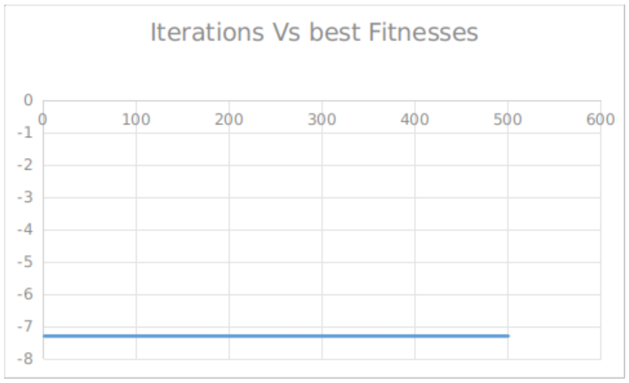

Figure 13: Michalewicz (Experiment #1)

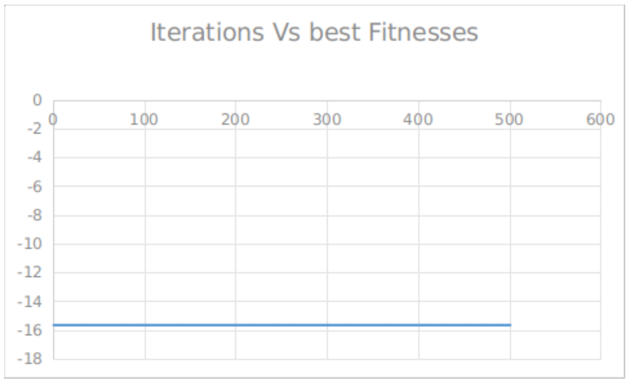

Figure 14: Masters' Cosine (Experiment #1)

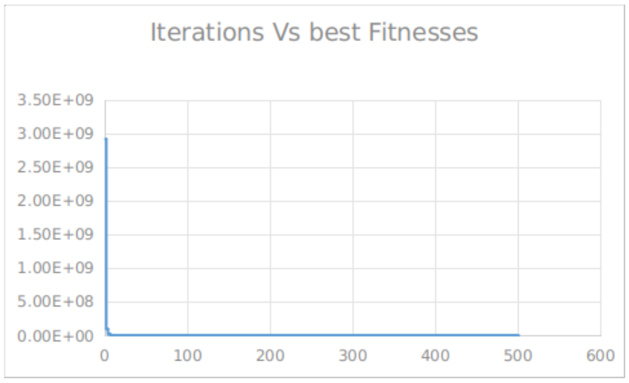

Figure 15: Quartic (Experiment #1)

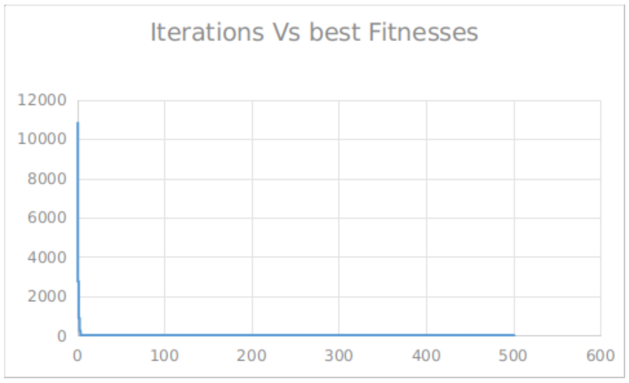

Figure 16: Levy (Experiment #1)

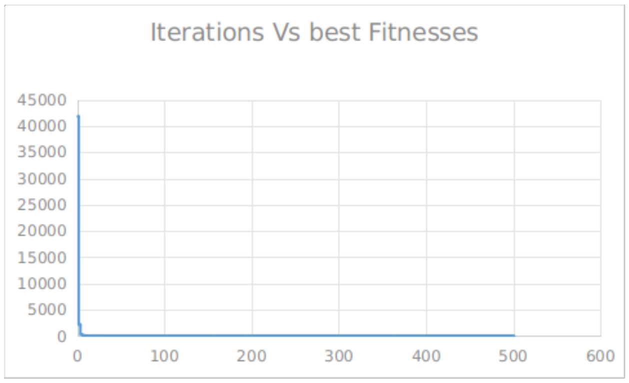

Figure 17: Step (Experiment #1)

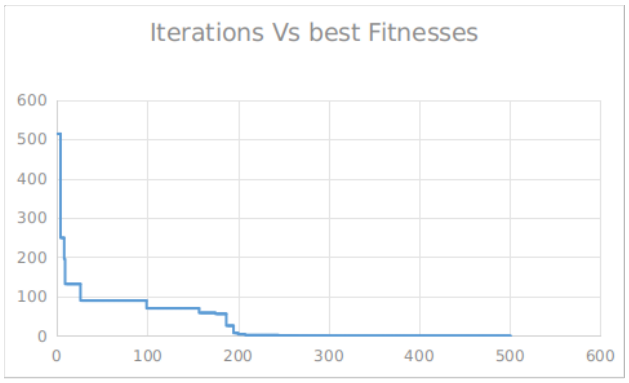

Figure 18: Alpine (Experiment #1)

2.1.5 Plots analysis

When looking at the figures above, we can see that PSO improves the quality of the solutions as it goes through iterations. The reason to this improvement is that after each iteration, the less fitted particles try to adjust their speeds in order to get closer to the best fitted ones. However, Figures 6, 7,10, 11, 13 and 14 corresponding to functions Sine Envelope, Stretch V Sine, Egg Holder, Rana, Michalewicz and Master's Cosine respectively also show that PSO does not work well for all the functions. In this case, the best solutions do not improve nor they deteriorate.

2.1.6 Stagnation iterations

Table 2: Iteration at which population stagnates (0 means no stagnation observed)

	f1	**f2**	**f3**	**f4**	**f5**	**f6**	**f7**	**f8**	**f9**	**f10**	**f11**	**f12**	**f13**	**f14**	**f15**	**f16**	**f17**	**f18**
iter 1	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 2	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
fiter 3	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 4	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 5	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 6	0	0	35	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
iter 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 9	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 10	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 11	0	0	16	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
iter 12	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 13	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 14	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 15	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 16	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 17	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 18	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 19	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 20	0	0	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 21	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 22	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 23	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0
iter 24	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 25	0	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 26	0	3	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 27	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 28	0	4	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iter 29	0	0	16	0	0	0	0	0	0	0	0	0	0	0	2	0	3	0
iter 30	0	0	8	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0

2.2 Firefly Algorithm

2.2.1 Values used for the parameters

The Firefly Algorithm was run using the following values:

- dimension: 30
- population size: 500
- number of generations: 500
- number of experiments: 30
- γ: 1
- β_0: 0.2
- α: 0.5

Experiments were performed while varying the values for γ, β_0 and α but all the results looked similar.

2.2.2 Statistics

Table 3: Statistics for FA (30 experiments)

	Average	**Std_Dev**	**Range**	**Median**	**Avg_Time**	**Avg #calls\exp**
f1 Schwefel	9190.78	237.84	985.20	9232.69	167793.93	4.93E+07
f2 De Jong 1	1.67	0.16	0.67	1.69	100650.17	3.43E+07
f3 Rosenbrock	131.14	9.40	35.02	132.95	167647.10	4.96E+07
f4 Rastrigin	25.91	55.34	282.34	7.84	211564.90	5.73E+07
f5 Griewangk	1.00	0.00	0.00	1.00	36595.37	9.13E+06
f6 Sine Envelope	-20.22	0.70	2.57	-20.15	303557.50	6.16E+07
f7 Stretch V Sine	10.15	0.00	0.00	10.15	371.17	500
f8 Ackley One	3.77	1.59	6.36	3.81	264987.13	5.96E+07
f9 Ackley Two	62.15	3.26	15.05	63.11	341409.83	6.17E+07
f10 Egg Holder	-5092.90	578.42	1992.42	-5196.45	219777.37	5.29E+07
f11 Rana	-3399.74	408.34	1885.43	-3429.27	307177.87	5.71E+07
f12 Pathological	0.00	0.00	0.00	0.00	293725.87	5.76E+07
f13 Michalewicz	-8.02	0.59	2.32	-7.85	387429.70	6.16E+07
f14 Masters' Cosine	-15.67	0.00	0.00	-15.67	366.80	500
f15 Quartic	2.67	0.57	2.15	2.65	212631.63	4.71E+07
f16 Levy	0.59	0.07	0.33	0.59	299312.63	6.13E+07
f17 Step	14.45	0.50	1.97	14.58	101594.60	3.41E+07
f18 Alpine	2.16	0.17	0.58	2.21	204291.83	6.13E+07

2.2.3 Statistics Analysis

As table 3 above shows, the Firefly Algorithm performed a little less better than PSO for most of the objective functions. However, for some functions such as Rastrigin, it produced really good results compared to the corresponding results for PSO(25.91 vs 220176.30). Table 3 also shows that, FA takes a lot of time to process compared to all the other algorithms seen so far.

2.2.4 plots

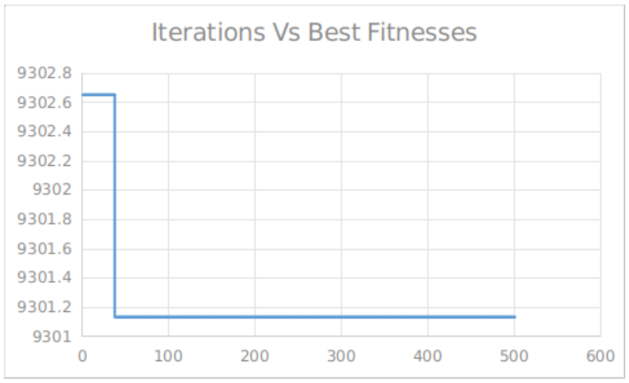

Figure 19: Schwefel (Experiment #1)

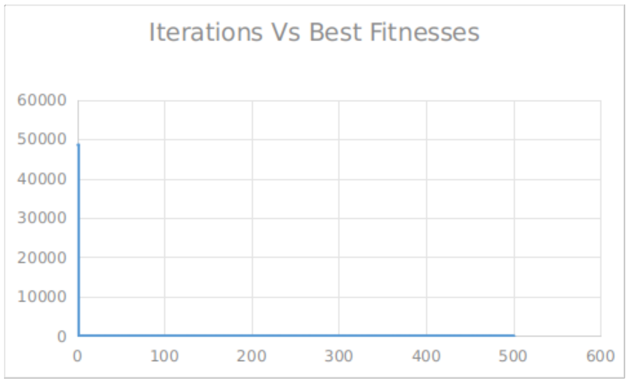

Figure 20: De Jong 1 (Experiment #1)

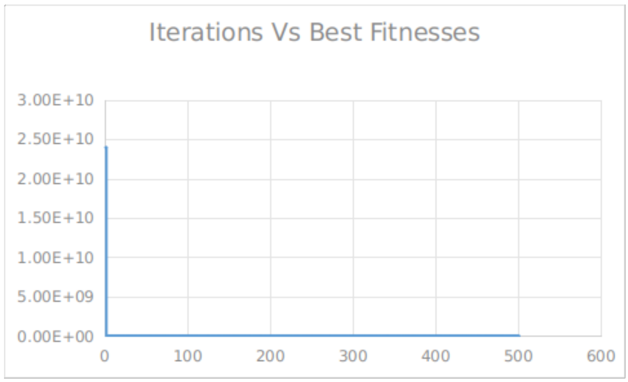

Figure 21: Rosenbrock (Experiment #1)

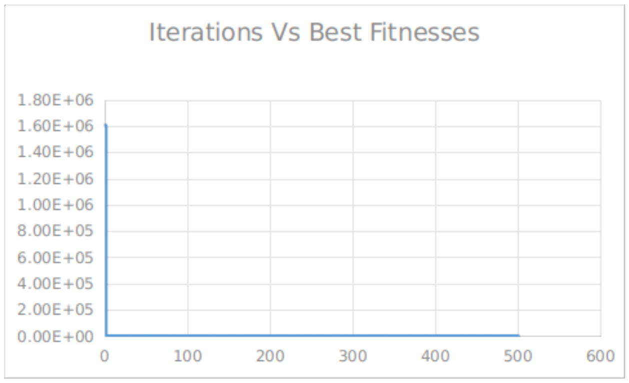

Figure 22: Rastrigin (Experiment #1)

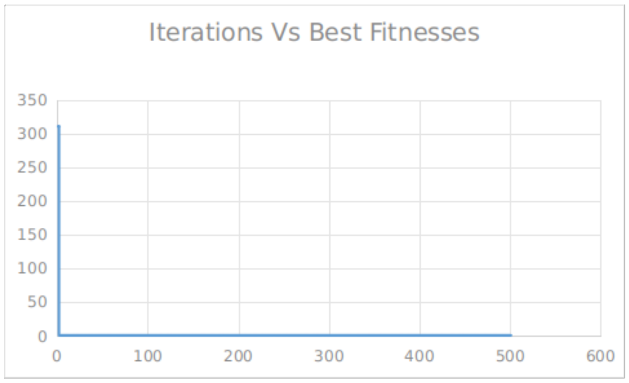

Figure 23: Griewangk (Experiment #1)

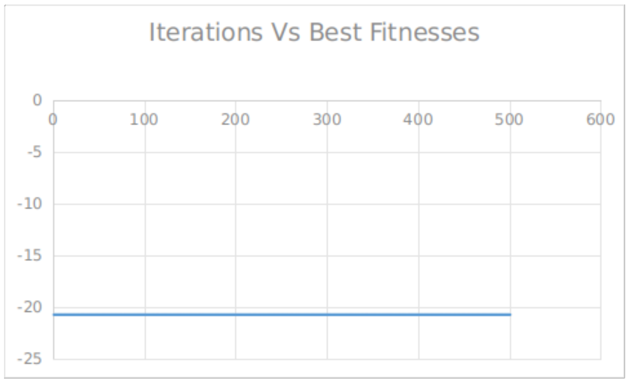

Figure 24: Sine Envelope (Experiment #1)

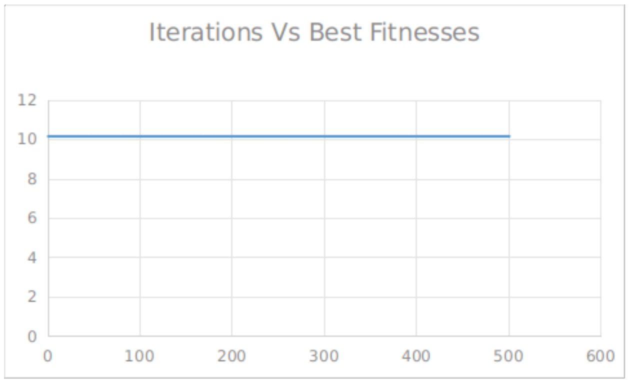

Figure 25: Stretch V Sine (Experiment #1)

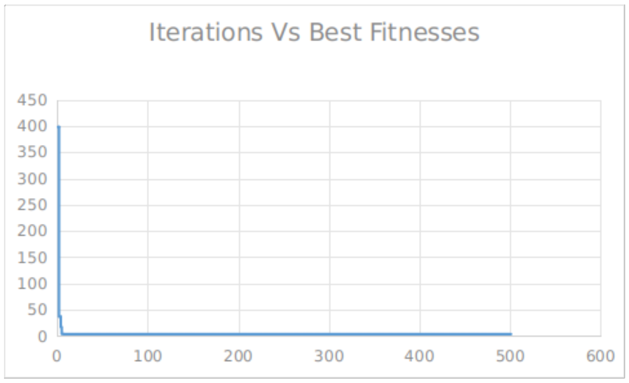

Figure 26: Ackley One (Experiment #1)

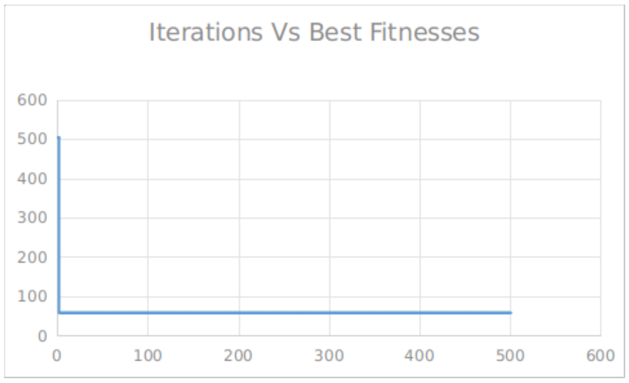

Figure 27: Ackley Two (Experiment #1)

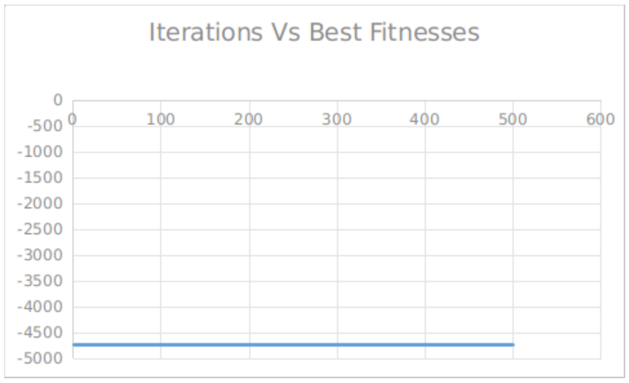

Figure 28: Egg Holder (Experiment #1)

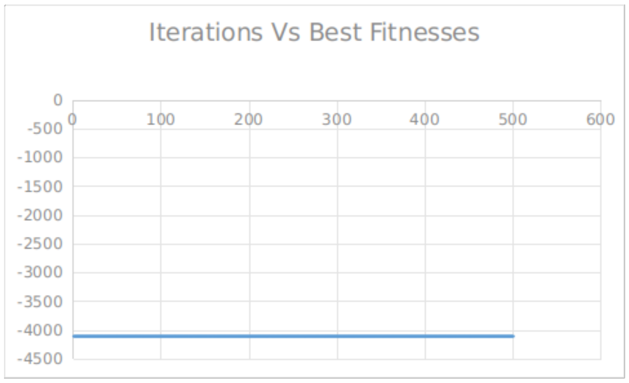

Figure 29: Rana (Experiment #1)

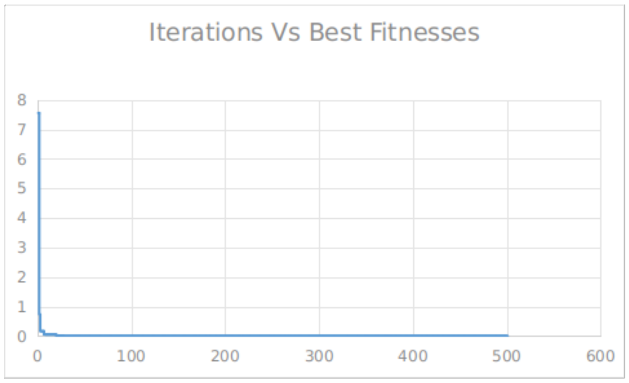

Figure 30: Pathological (Experiment #1)

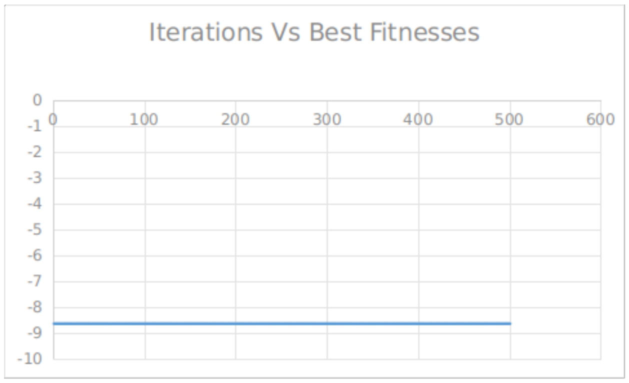

Figure 31: Michalewicz (Experiment #1)

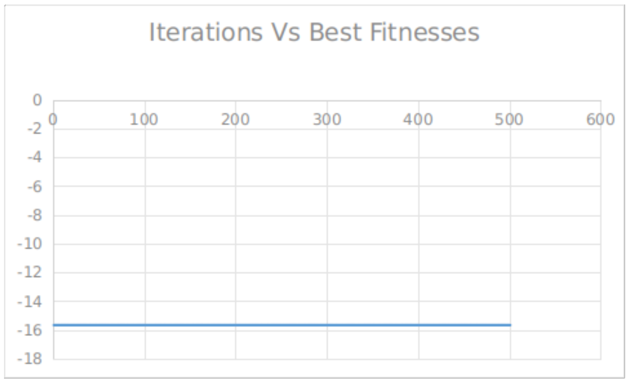

Figure 32: Masters' Cosine (Experiment #1)

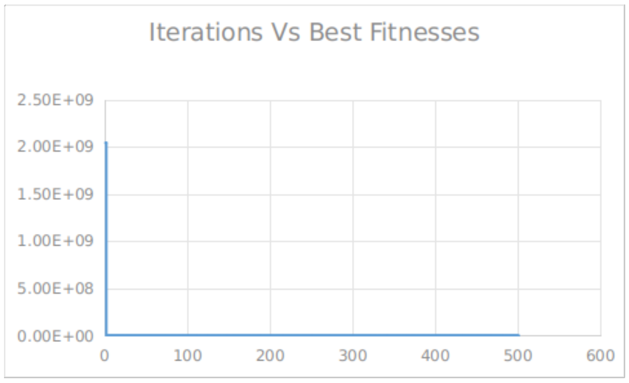

Figure 33: Quartic (Experiment #1)

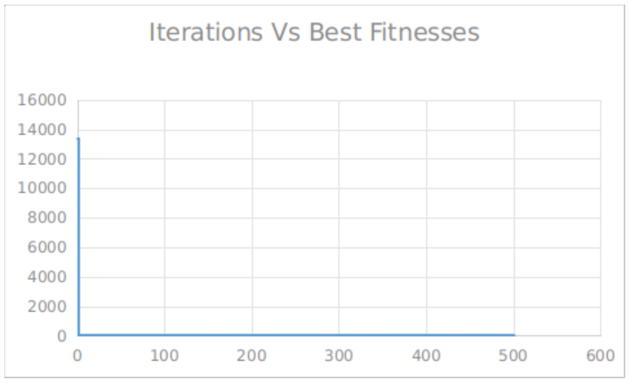

Figure 34: Levy (Experiment #1)

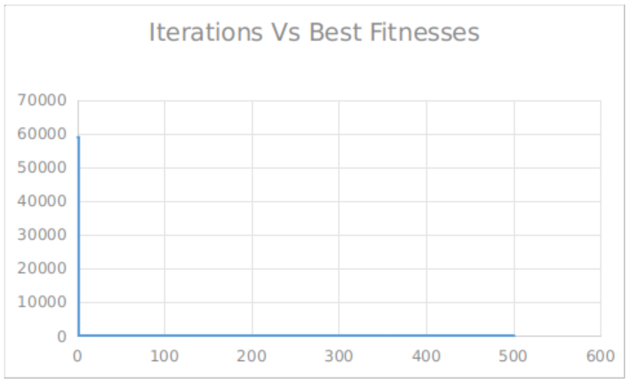

Figure 35: Step (Experiment #1)

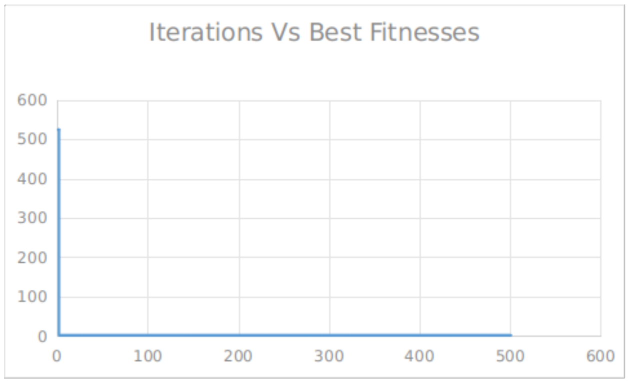

Figure 36: Alpine (Experiment #1)

2.2.5 Plots analysis

When looking at the figures above, we can see that FA improves the quality of the solutions as it goes through iterations. The reason to this improvement is that after each iteration, the less bright firefly try to adjust their paths in order to move towards the brightest ones. However, Figures 24, 25, 28, 29, 31 and 32 corresponding to functions Sine Envelope, Stretch V Sine, Egg Holder, Rana, Michalewicz and Master's Cosine respectively also show that FA does not work well for all the functions. In this case, the best solutions remain stable across iterations.

2.2.6 Stagnation iterations

Table 4: Iteration at which population stagnates (0 means no stagnation observed)

	f1	**f2**	**f3**	**f4**	**f5**	**f6**	**f7**	**f8**	**f9**	**f10**	**f11**	**f12**	**f13**	**f14**	**f15**	**f16**	**f17**	**f18**
iter 1	0	0	0	0	0	500	500	500	493	0	499	0	499	500	0	500	0	498
iter 2	498	0	0	469	0	500	500	481	500	0	462	0	498	500	0	500	0	500
fiter 3	0	0	0	0	0	496	500	435	499	494	341	0	499	500	0	500	0	498
iter 4	0	0	0	492	0	495	500	475	499	0	500	0	499	500	0	500	0	500
iter 5	488	0	0	450	0	500	500	428	498	0	495	0	500	500	0	500	0	496
iter 6	0	0	0	396	0	494	500	423	500	0	499	0	498	500	0	500	0	500
iter 7	476	0	0	490	0	499	500	500	499	494	497	0	500	500	0	500	0	498
iter 8	499	0	0	491	0	497	500	426	500	377	0	0	500	500	0	499	0	500
iter 9	0	0	0	0	0	491	500	429	497	405	500	0	498	500	0	500	0	499
iter 10	476	0	0	0	0	500	500	498	498	405	495	0	500	500	0	500	0	500
iter 11	498	0	0	0	0	500	500	451	500	0	495	0	500	500	0	499	0	495
iter 12	495	0	0	482	0	499	500	494	500	500	500	0	499	500	0	499	0	500
iter 13	431	0	0	381	0	495	500	481	500	0	413	0	500	500	0	500	0	500
iter 14	0	0	0	483	0	496	500	493	500	0	493	0	500	500	0	500	0	500
iter 15	500	0	0	497	0	498	500	496	500	0	486	0	500	500	0	499	0	500
iter 16	496	0	0	489	0	500	500	497	500	0	496	0	499	500	0	499	0	500
iter 17	0	0	0	0	0	496	500	496	500	0	469	0	498	500	0	499	0	500
iter 18	471	0	0	440	0	499	500	434	500	0	451	0	500	500	0	500	0	500
iter 19	491	0	0	0	0	497	500	399	499	0	455	0	500	500	0	500	0	498
iter 20	0	0	0	499	0	494	500	0	500	0	475	0	498	500	0	500	0	500
iter 21	485	0	0	442	0	500	500	448	499	495	471	0	500	500	0	499	0	500
iter 22	0	0	0	255	0	500	500	470	500	0	493	0	499	500	0	498	0	499
iter 23	487	0	0	0	0	500	500	0	498	479	497	0	499	500	0	498	0	495
iter 24	497	0	0	0	0	498	500	489	492	0	497	0	500	500	0	500	0	500
iter 25	492	0	0	352	0	498	500	477	499	0	500	0	500	500	0	499	0	500
iter 26	494	0	0	488	0	499	500	457	500	0	493	0	497	500	0	498	0	500
iter 27	494	0	0	483	0	494	500	488	500	464	0	0	500	500	0	500	0	499
iter 28	0	0	0	383	0	498	500	454	498	473	497	0	500	500	0	500	0	500
iter 29	0	0	0	0	0	493	500	423	500	0	497	0	500	500	0	500	0	500
iter 30	0	0	0	500	0	500	500	437	500	308	498	0	498	500	0	500	0	500

2.3 Harmony Search

2.3.1 Values used for the parameters

The Harmony Search Algorithm was run using the following values:

- dimension: 30
- population size: 500
- number of generations: 500
- number of experiments: 30
- Harmony Memory Considering rate (HMCR): 0.1
- Pitch Adjusting rate (PAR): 0.1
- Bandwidth(bw): 0.2

Experiments were performed while varying the values for HMCR, PAR and bw but all the results looked similar.

2.3.2 Statistics

2.3.3 Statistics Analysis

As table 5 below shows, the Harmony Search algorithm performed a little less better than FA for most of the objective functions even though for some functions such as Rastrigin, it produced better (0.14 vs 5.91).

Table 5: Statistics for HS (30 experiments)

	Average	**Std_Dev**	**Range**	**Median**	**Avg_Time**	**Avg #calls\exp**
f1 Schwefel	9227.24	605.30	2184.23	9388.01	808.43	250500
f2 De Jong 1	3.53	0.32	1.15	3.59	694.80	250500
f3 Rosenbrock	175.82	34.69	137.79	188.62	809.77	250500
f4 Rastrigin	0.14	0.12	0.58	0.11	884.80	250500
f5 Griewangk	1.00	0.00	0.00	1.00	910.13	250500
f6 Sine Envelope	-20.36	0.73	3.19	-20.28	1096.10	250500
f7 Stretch V Sine	10.15	0.00	0.00	10.15	1190.03	250500
f8 Ackley One	7.92	3.67	14.44	7.94	1020.73	250500
f9 Ackley Two	69.13	1.85	6.46	69.17	1364.03	250500
f10 Egg Holder	-5090.64	674.13	2613.92	-5107.24	1016.50	250500
f11 Rana	-3319.87	393.35	1705.87	-3288.47	1270.10	250500
f12 Pathological	0.00	0.00	0.00	0.00	1259.40	250500
f13 Michalewicz	-8.03	0.59	2.74	-7.99	1519.70	250500
f14 Masters' Cosine	-15.67	0.00	0.00	-15.67	1108.60	250500
f15 Quartic	12.42	1.92	8.53	12.75	1119.37	250500
f16 Levy	0.60	0.09	0.39	0.60	1180.10	250500
f17 Step	18.08	0.72	3.13	18.26	701.83	250500
f18 Alpine	3.78	0.70	3.88	3.98	773.77	250500

2.3.4 plots

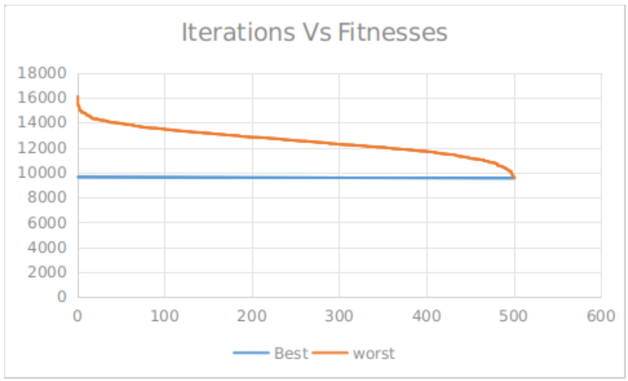

Figure 37: Schwefel (Experiment #1)

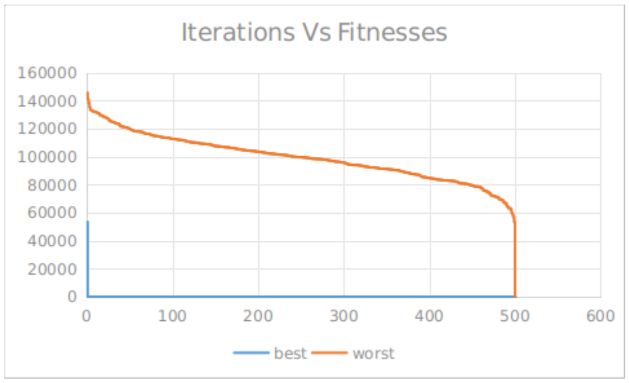

Figure 38: De Jong 1 (Experiment #1)

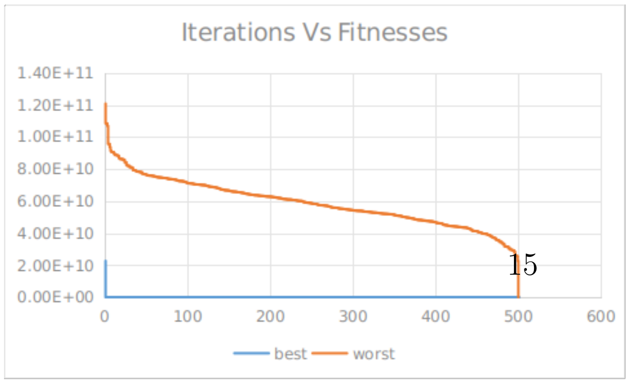

Figure 39: Rosenbrock (Experiment #1)

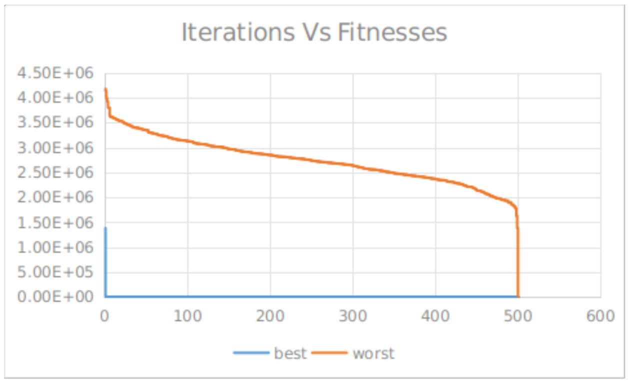

Figure 40: Rastrigin (Experiment #1)

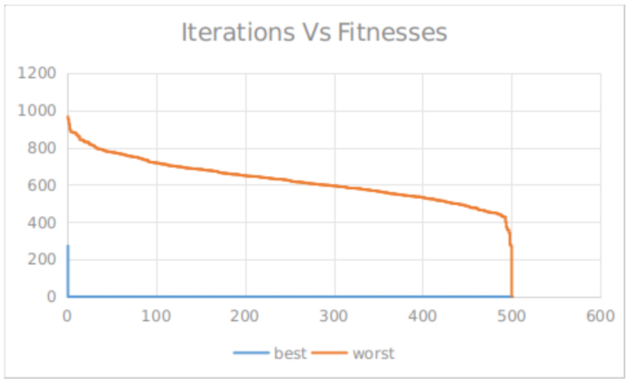

Figure 41: Griewangk (Experiment #1)

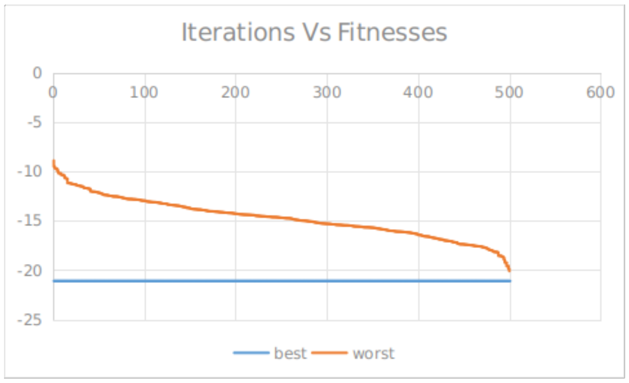

Figure 42: Sine Envelope (Experiment #1)

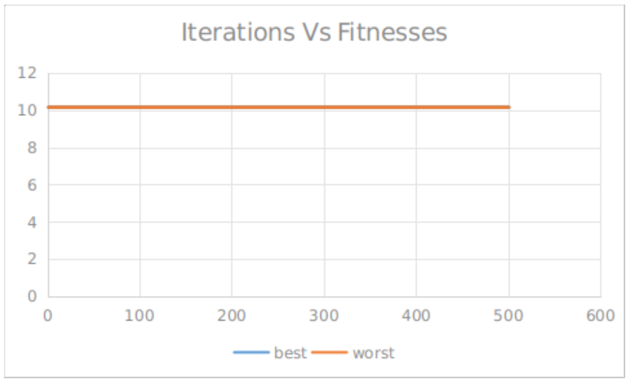

Figure 43: Stretch V Sine (Experiment #1)

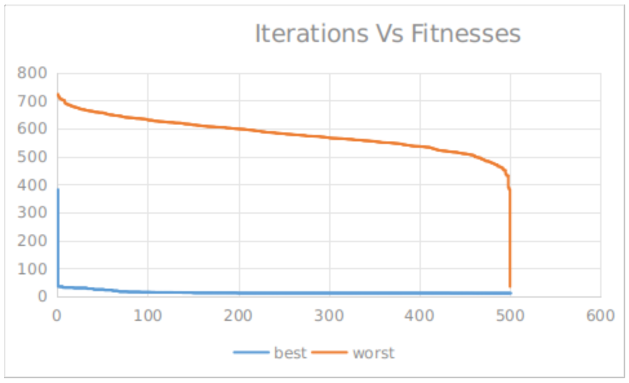

Figure 44: Ackley One (Experiment #1)

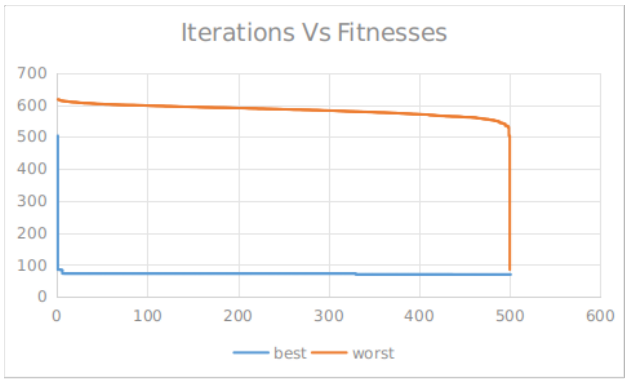

Figure 45: Ackley Two (Experiment #1)

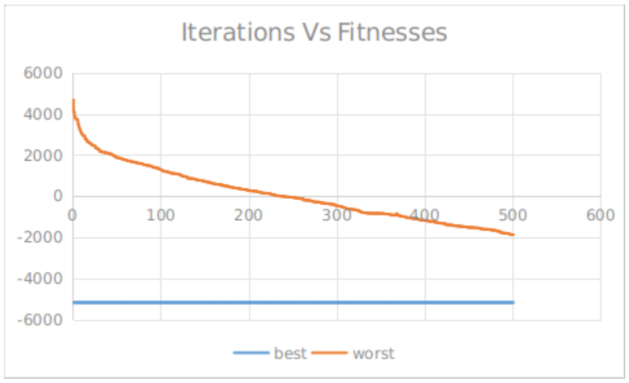

Figure 46: Egg Holder (Experiment #1)

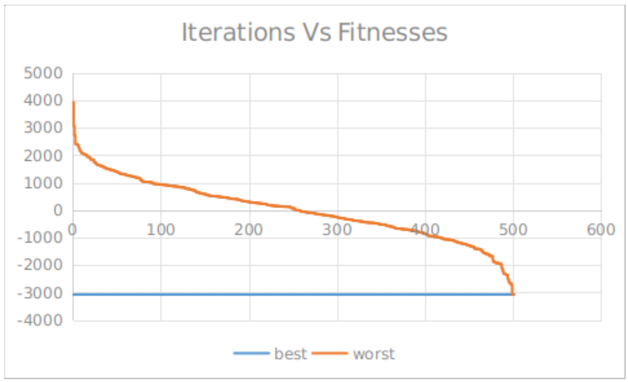

Figure 47: Rana (Experiment #1)

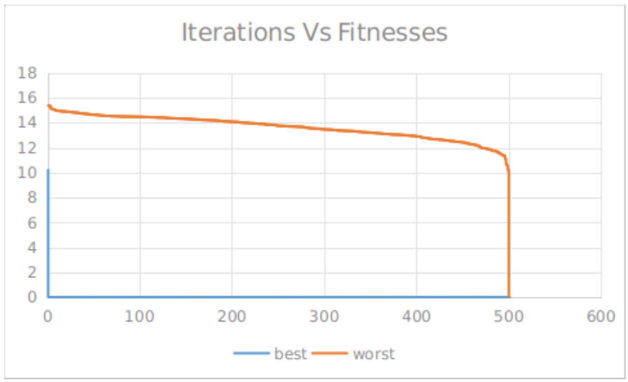

Figure 48: Pathological (Experiment #1)

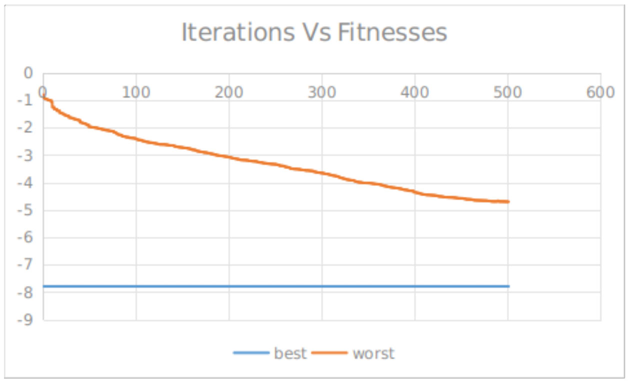

Figure 49: Michalewicz (Experiment #1)

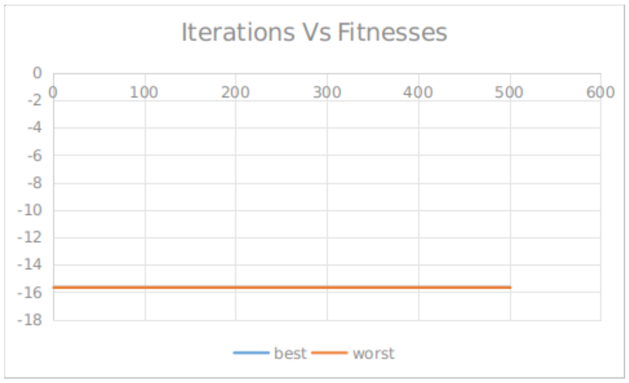

Figure 50: Masters' Cosine (Experiment #1)

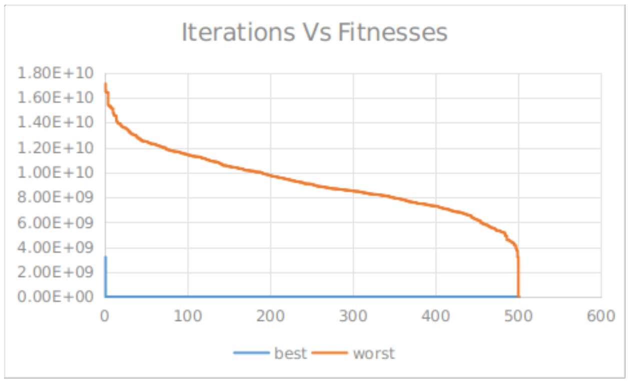

Figure 51: Quartic (Experiment #1)

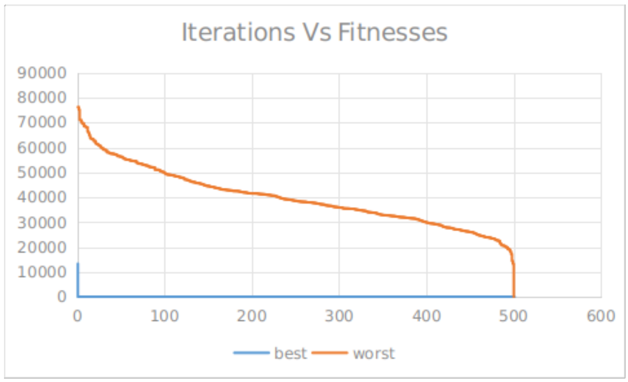

Figure 52: Levy (Experiment #1)

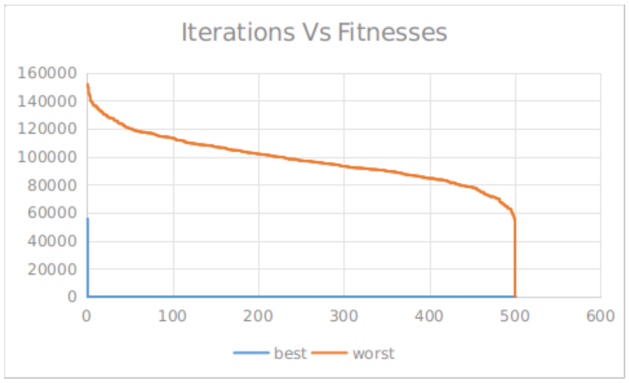

Figure 53: Step (Experiment #1)

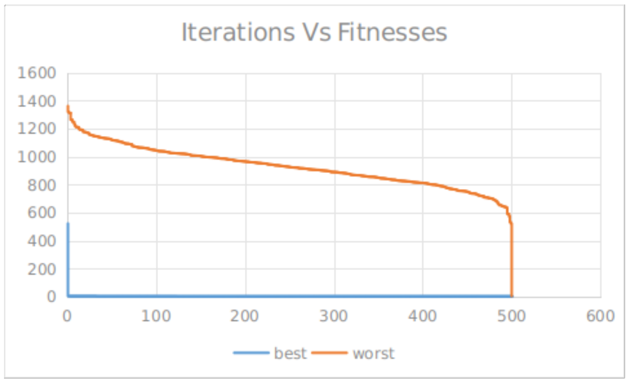

Figure 54: Alpine (Experiment #1)

2.3.5 Plots analysis

When looking at the figures above, we can see that HS improves the quality of the solutions as it goes through iterations. The reason to this improvement is that after each iteration, the pitch of the bad harmonies are adjusted either randomly or using a specific function in order to get them closer the best ones. However, Figures 42, 43, 46, 47, 49 and 50 corresponding to functions Sine Envelope, Stretch V Sine, Egg Holder, Rana, Michalewicz and Master's Cosine respectively also show that HS does not work well for all the functions. In this case, the best solutions remain stable across iterations.

2.3.6 Stagnation iterations

Table 6: Iteration at which population stagnates (0 means no stagnation observed)

	f1	**f2**	**f3**	**f4**	**f5**	**f6**	**f7**	**f8**	**f9**	**f10**	**f11**	**f12**	**f13**	**f14**	**f15**	**f16**	**f17**	**f18**
iter 1	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 2	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
fiter 3	0	0	0	0	0	500	500	0	0	0	500	0	0	500	0	0	0	0
iter 4	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 5	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 6	0	0	0	0	0	500	500	0	0	0	500	0	0	500	0	0	0	0
iter 7	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 8	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 9	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 10	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 11	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 12	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 13	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 14	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 15	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 16	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 17	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 18	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 19	0	0	0	0	0	500	500	0	0	0	500	0	0	500	0	0	0	0
iter 20	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 21	0	0	0	0	0	500	500	0	0	0	500	0	0	500	0	0	0	0
iter 22	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 23	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 24	0	0	0	0	0	500	500	0	0	0	500	0	0	500	0	0	0	0
iter 25	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 26	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 27	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 28	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 29	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0
iter 30	0	0	0	0	0	0	500	0	0	0	500	0	0	500	0	0	0	0

3 Conclusion

After implementing, experimenting and analyzing three different instances of the particle swarm intelligence algorithms, it can be determined that particle swarm intelligence algorithms not only are easier to implement than evolutionary algorithms but also produce better results even though they might take more time to process in some cases (FA). However, analysis of the results of PSO, FA and HS showed that none of them works for functions such as

Sine Envelope, Stretch V Sine, Egg Holder, Rana, Michalewicz and Master's Cosine