CS 471 - Project 4

Andrew Dunn
Instructor: Dr. Donald Davendra
Central Washington University
Ellensburg, Washington

May 20, 2019

Abstract

In this document we will first introduce you to the problems that were tested and our test methodology. We will then display the results we gathered, and explain our conclusions based on the data analysis.

1 Introduction

For this experiment we tested 18 different mathematical functions that are n-dimension scalable. More information about all 18 functions can be found in Table 1. The program developed to test these functions can be configured with several different parameters, such as specifying the population and dimension sizes, search algorithm, and number of test iterations. In our tests we ran multiple experiments utilizing three different search algorithms: Firefly, harmony search, and particle swarm. Tests were executed using a population size of 500, and the algorithms were ran for 500 iterations. The latest version of our program utilizes multi-threading to decrease execution times. The tests we ran used 8 different threads, and all functions computed values using 64-bit floating point data types.

The three search algorithms that we are testing are all based on real world patterns and behaviors. The firefly algorithm is based on the movement of fireflies in nature. The harmony search algorithm is based on the process for which a musician tunes their instrument. Finally, the particle swarm algorithm models swarm behaviors in nature.

The testing hardware we used for this experiment is a desktop computer running Ubuntu 19.10 with an Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz and 32 GB of DDR3 @ 1600 MHz. Our testing program was written in C++ and compiled with CMake version 3.12.1 and g++ version 8.2.0. We are using the C++11 standard libraries with the built in Mersenne Twister random number generator to generate the population data between the given ranges for each function in Table 1.

2 Methods

To help reach our goal of producing the global optimal minimum, for each of the three search algorithms we created four different strategies, each with a different combination of control parameters. Using these different strategies, we were able to find some parameter combinations that work better than others for certain objective functions. You can find the parameter values used for each strategy in tables 2, 3, and 4.

The firefly algorithm takes three different control parameters: alpha, beta, and gamma. The alpha value controls the degree of randomization when computing the movement of a firefly. We found that larger alpha values tended to push all fireflies to the outer bounds of the objective function. The beta and gamma values control the amount of influence other fireflies have when computing the current fireflies movement. When beta is larger and gamma is smaller, a firefly will be more attracted to other fireflies, and thus move towards them more. We found that if gamma is much larger than 0.001, the distances between fireflies would be so great that their brightness would have little to no influence on the movement of the current firefly.

Table 1: Experiment functions

f	Name	f(x*)	Dimensions	Range
1	Schwefel	0	$10,20,30$	$[-512,512]^n$
2	De Jong 1	0	$10,20,30$	$[-100,100]^n$
3	Rosenbrocks Saddle	0	$10,20,30$	$[-100,100]^n$
4	Rastrigin	0	$10,20,30$	$[-30,30]^n$
5	Griewangk	0	$10,20,30$	$[-500,500]^n$
6	Sine Envelope Sine Wave	$-1.4915(n-1)$	$10,20,30$	$[-30,30]^n$
7	Stretch V Sine Wave	0	$10,20,30$	$[-30,30]^n$
8	Ackley One	$-7.54276-2.91867(n-3)$	$10,20,30$	$[-32,32]^n$
9	Ackley Two	0	$10,20,30$	$[-32,32]^n$
10	Egg Holder	$-$	$10,20,30$	$[-500,500]^n$
11	Rana	$-$	$10,20,30$	$[-500,500]^n$
12	Pathological	$-$	$10,20,30$	$[-100,100]^n$
13	Michalewicz	$0.966n$	$10,20,30$	$[0,\pi]^n$
14	Masters Cosine Wave	$1-n$	$10,20,30$	$[-30,30]^n$
15	Quartic	0	$10,20,30$	$[-100,100]^n$
16	Levy	0	$10,20,30$	$[-10,10]^n$
17	Step	0	$10,20,30$	$[-100,100]^n$
18	Alpine	0	$10,20,30$	$[-100,100]^n$

The harmony search algorithm takes three control parameters as well: HMCR (Harmony Memory Considering Rate), PAR (Pitch Adjusting Rate), and BW (Bandwidth). The HMCR parameter influences the random chance that the next population iteration will use use an existing value in the population. The higher the HMCR value, the more likely the selected value will be from another in the existing population rather than just completely random. The PAR value controls how likely the value will be adjusted after passing the HMCR check. A higher PAR value leads to a higher chance that the value will be adjusted. Finally, the BW value affects the amount of adjustment that is made after passing the PAR check.

Finally, the particle swarm algorithm also takes three control parameters: c1, c2, and k. The c1 and c2 values influence a particles movement, such that a higher c1 value leads to the particle moving more towards it's personal best, while a higher c2 value moves the particle more to the global best. Finally, the k parameter is a dampening factor that slows down or speeds up a particle's movement between each iteration.

Table 2: Firefly Algorithm Parameter Strategies

	alpha	**beta**	**gamma**
FA Strategy 1	0.02	0.3	0.001
FA Strategy 2	0.02	0.3	0.01
FA Strategy 3	0.05	0.3	0.001
FA Strategy 4	0.05	0.3	0.01

Table 3: Harmony Search Parameter Strategies

	HMCR	**PAR**	**BW**
HS Strategy 1	0.7	0.1	0.1
HS Strategy 2	0.8	0.2	0.2
HS Strategy 3	0.9	0.4	0.2
HS Strategy 4	0.95	0.6	0.4

Table 4: Particle Swarm Parameter Strategies

	c1	**c2**	**k**
PSO Strategy 1	0.8	1.2	0.5
PSO Strategy 2	0.6	1.4	0.5
PSO Strategy 3	1.2	0.8	0.5
PSO Strategy 4	1.4	0.6	0.5

3 Execution Time Analysis

In this section you will find data detailing the execution times and total number of objective function call counts required when running the firefly. harmony search, and particle swarm algorithms.

Table 5: Objective Function Call Counts - 500 to 100,000 Iterations

	500 Iter.	**1,000 Iter.**	**10,000 Iter.**	**100,000 Iter.**
Firefly Algorithm	62,375,500	124,751,000 (est.)	1,247,510,000 (est.)	12,475,100,000 (est.)
Harmony Search	1,000	1,500	10,500	100,500
Particle Swarm	250,500	500,500	5,000,500	50,000,500

Table 6: Firefly Algorithm - Execution Times (ms)

f(x)	**Strategy 1**	**Strategy 2**	**Strategy 3**	**Strategy 4**
1	113567.91	116827.67	121534.89	123957.58
2	60009.38	62022.69	62863.52	62055.69
3	63436.60	67016.46	65863.66	67935.44
4	120203.93	126114.57	124536.43	124204.90
5	125528.23	131396.50	134305.74	136094.34
6	123084.35	128384.97	126730.74	127972.65
7	255408.10	269303.57	261388.36	264160.37
8	162020.74	171126.28	167991.83	169963.83
9	289329.95	301450.78	294128.79	293670.48
10	169005.13	181608.12	181619.97	183291.18
11	261199.78	277795.73	274464.89	276855.62
12	131281.25	142452.98	135635.68	134556.68
13	219904.12	228027.23	221654.36	218485.15
14	193564.72	203242.22	182157.59	200335.48
15	62603.04	69005.30	63201.64	63484.06
16	104228.79	107836.44	102713.52	109569.02
17	53964.83	61118.50	51862.51	56260.97
18	103691.65	102085.50	96870.44	93471.88

First, we will take a look at the total number of objective function calls in Table 5. The values contained in this table are when the algorithm is ran for a single objective function. Each algorithm produced the same number of objective function calls for each of the 18 functions with the given number of iterations. Immediately we can see that the firefly algorithm requires many more objective function calls than both harmony search and particle swarm. This is due to the pair-wise calculations done between each firefly in the population. Particle swarm also requires a fair number of objective function calls, again far more than harmony search. Based off these numbers, it is clear why the firefly algorithm is very slow while harmony search is very fast, and particle swarm is some where in the middle.

Looking at the execution times in tables 6, 7, and 8, you can see that they follow this same pattern. The firefly algorithm requires the most execution time, up to 300,000 milliseconds (about 5 minutes) in some cases for a single objective function. In contrast, harmony search finished in less than 200 milliseconds for every function, which is about 1,500 times faster than the firefly algorithm. The particle swarm algorithm is also relatively fast compared to firefly, never taking more than 1,300 milliseconds (about 1.3 seconds). Looking at these execution times, it is hard recommend the firefly algorithm purely from a performance stand-point.

Table 7: Harmony Search - Execution Times (ms)

f(x)	Strategy 1	Strategy 2	Strategy 3	Strategy 4
1	79.54	82.71	66.86	73.26
2	79.62	73.82	74.41	73.37
3	88.29	75.05	73.45	75.10
4	95.72	75.46	68.21	73.20
5	81.12	79.89	72.94	83.95
6	142.76	116.81	116.57	90.57
7	170.68	181.69	160.24	145.70
8	104.39	77.42	71.44	74.74
9	82.84	76.28	72.37	68.40
10	182.50	184.49	177.25	159.72
11	171.98	166.79	153.66	160.79
12	174.10	186.95	176.27	170.89
13	148.09	117.65	104.91	132.32
14	180.99	160.24	147.08	126.88
15	96.48	77.69	81.65	71.68
16	88.93	75.88	69.82	67.96
17	78.32	69.87	71.49	73.97
18	73.17	49.23	67.88	65.71

Table 8: Particle Swarm - Execution Times (ms)

f(x)	Strategy 1	Strategy 2	Strategy 3	Strategy 4
1	606.70	618.08	643.73	648.05
2	402.82	407.11	462.40	421.46
3	429.22	462.50	429.09	502.23
4	612.24	593.53	639.95	599.00
5	653.64	677.48	636.06	610.22
6	643.75	641.47	683.29	653.62
7	1229.71	1149.09	1256.08	1183.21
8	795.52	752.95	780.77	879.70
9	1258.28	1234.21	1270.26	1264.97
10	827.91	820.19	898.27	934.23
11	1169.11	1191.49	1190.59	1305.04
12	678.38	651.08	701.97	689.50
13	1038.33	1049.77	1089.31	1071.76
14	912.16	821.12	946.97	917.33
15	420.06	422.29	407.72	426.79
16	533.61	533.14	571.01	553.90
17	294.94	415.19	449.46	366.69
18	530.98	461.72	525.75	538.84

4 Project 3 Results Review

In this section you will find all experiment results from the genetic algorithm and differential evolutionary algorithm used in project 3. For each table we have also provided an example line graph showing the population improvement over time for the Schwefel function, function 1 over all 100 generations. Note that we have only included the differential evolution strategies that have provided the best result for at least one of the objective functions.

Figure 1: Example of Population Improvement

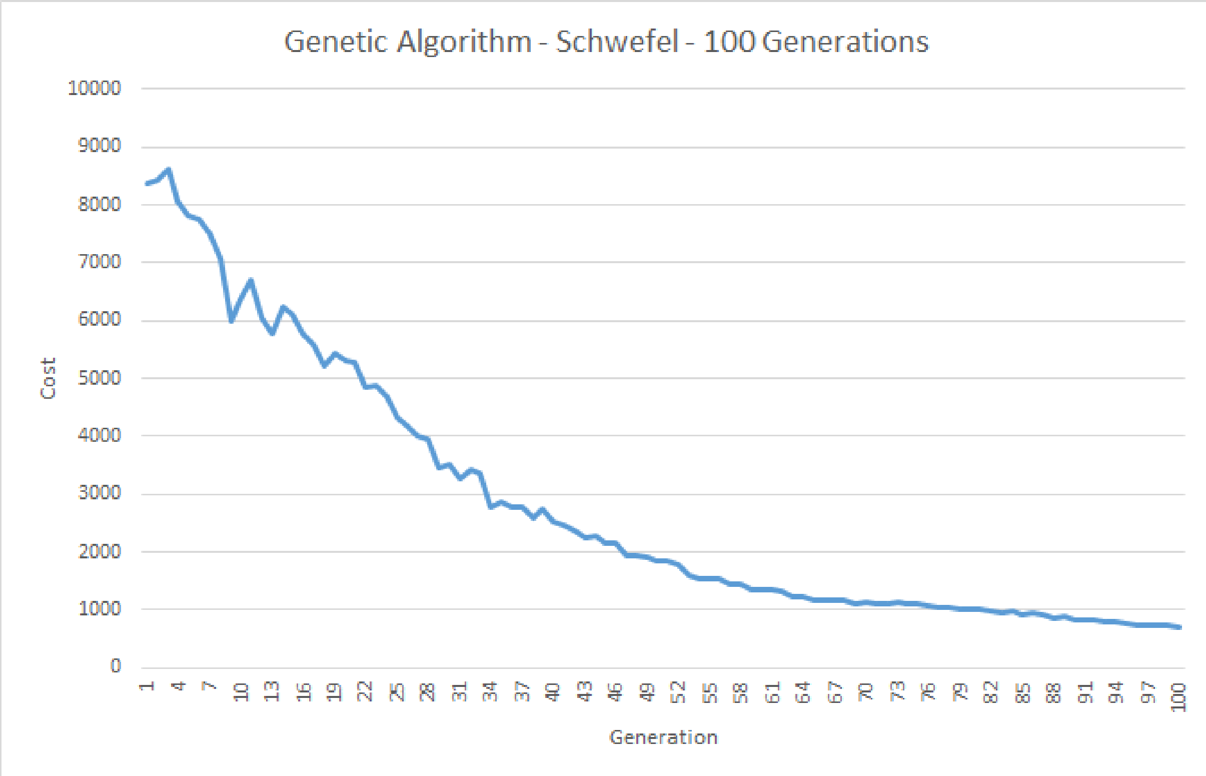


Table 9: Genetic Algorithm - Roulette Wheel - 50 Experiments

F(x)	Average	Std. Deviation	Range	Median	Min
1	1409.284653	358.375724	1451.37079	1591.5312	699.11931
2	857.807546	308.67808	1131.92014	512.86124	325.79366
3	40871057.69	34468711.68	181464667.1	34060619.5	5899992.9
4	-19799.06439	14723.4247	73989.943	-25740.2785	-41248.367
5	6.166095	2.486216	10.55429	5.16473	2.73055
6	-38.566841	0.863623	4.021618	-38.565934	-40.597703
7	31.309248	0.647397	3.640922	31.226281	30.018
8	-37.67213	11.877742	44.540614	-42.256097	-52.303887
9	117.773244	13.508431	49.593823	120.221655	97.118497
10	-12407.26667	1249.221689	6005.2914	-12333.5495	-15402.188
11	-7882.019022	598.526	3010.474	-7846.7365	-9722.9117
12	6.946521	0.466502	1.885526	6.965459	6.022934
13	-24.842781	1.011068	5.231098	-25.195884	-26.286702
14	-9.715474	1.867635	8.888868	-7.1306	-14.446877
15	6859130.134	5289334.142	28712241.8	4203369.7	1543819.2
16	5.771988	1.962127	8.886648	5.315028	1.971973
17	859.679765	353.307661	1470.88086	679.09807	335.08444
18	35.976585	8.599594	36.352284	35.721819	17.434286

Figure 2: Example of Population Improvement

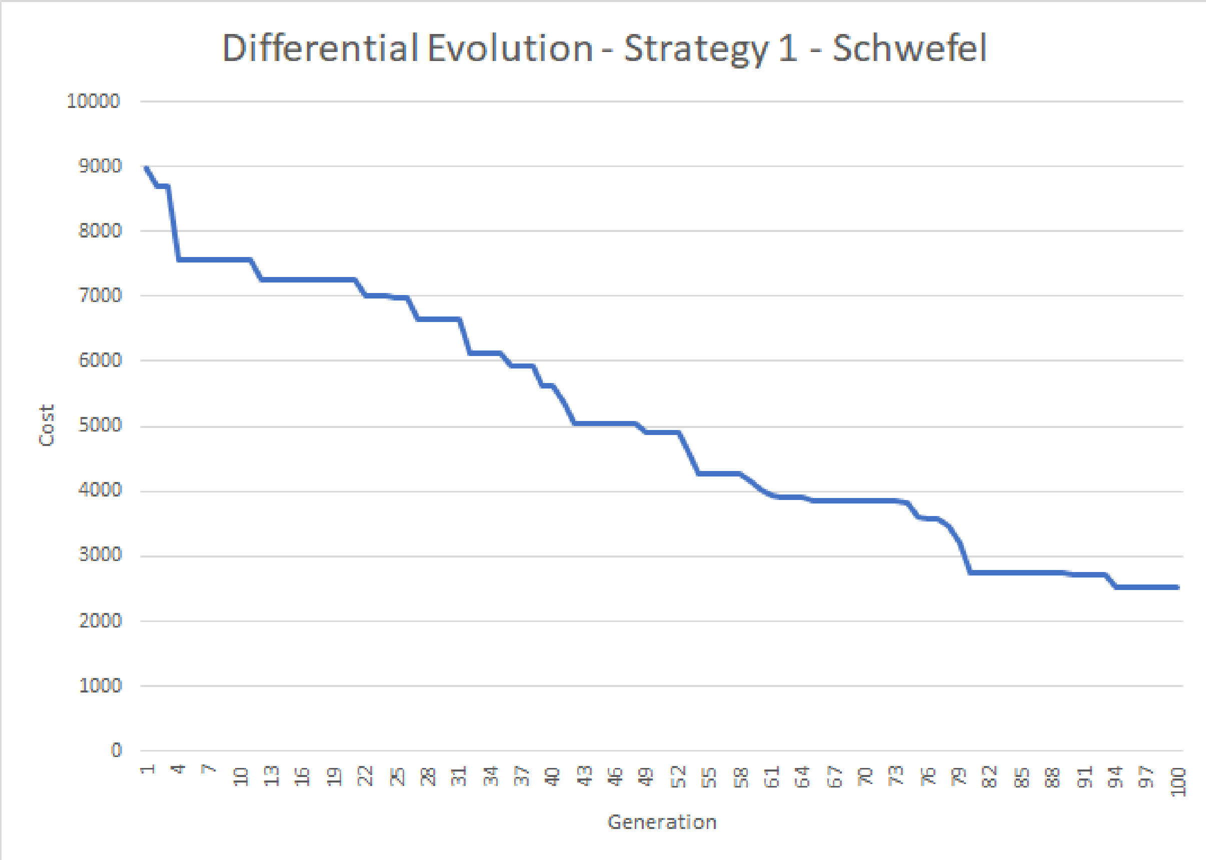


Table 10: Differential Evolution - Strategy 1 - 50 Experiments

F(x)	Average	Std. Deviation	Range	Median	Min
1	3572.96	339.41	1772.60	3571.04	2521.12
2	93.76	27.60	151.49	73.14	40.82
3	304950.86	193170.45	1029141.32	263284.03	73813.38
4	-28394.98	4154.84	19349.39	-28171.76	-39278.30
5	1.53	0.13	0.66	1.53	1.25
6	-35.46	0.43	2.06	-35.45	-36.53
7	42.23	1.23	6.36	42.44	38.13
8	-10.62	7.18	27.93	-17.66	-22.41
9	102.49	8.27	37.15	110.22	80.83
10	-12930.67	819.25	3979.40	-12899.64	-15614.14
11	-9375.61	854.25	3521.98	-8821.06	-11226.20
12	5.06	1.04	5.20	4.99	2.63
13	-15.93	0.67	3.14	-15.90	-17.56
14	-4.90	0.58	2.51	-4.83	-6.28
15	43635.68	23424.97	111182.31	40216.23	10654.49
16	1.44	0.33	1.30	1.43	0.88
17	131.86	30.43	145.10	131.08	62.11
18	82.80	13.61	66.82	82.36	41.27

Figure 3: Example of Population Improvement

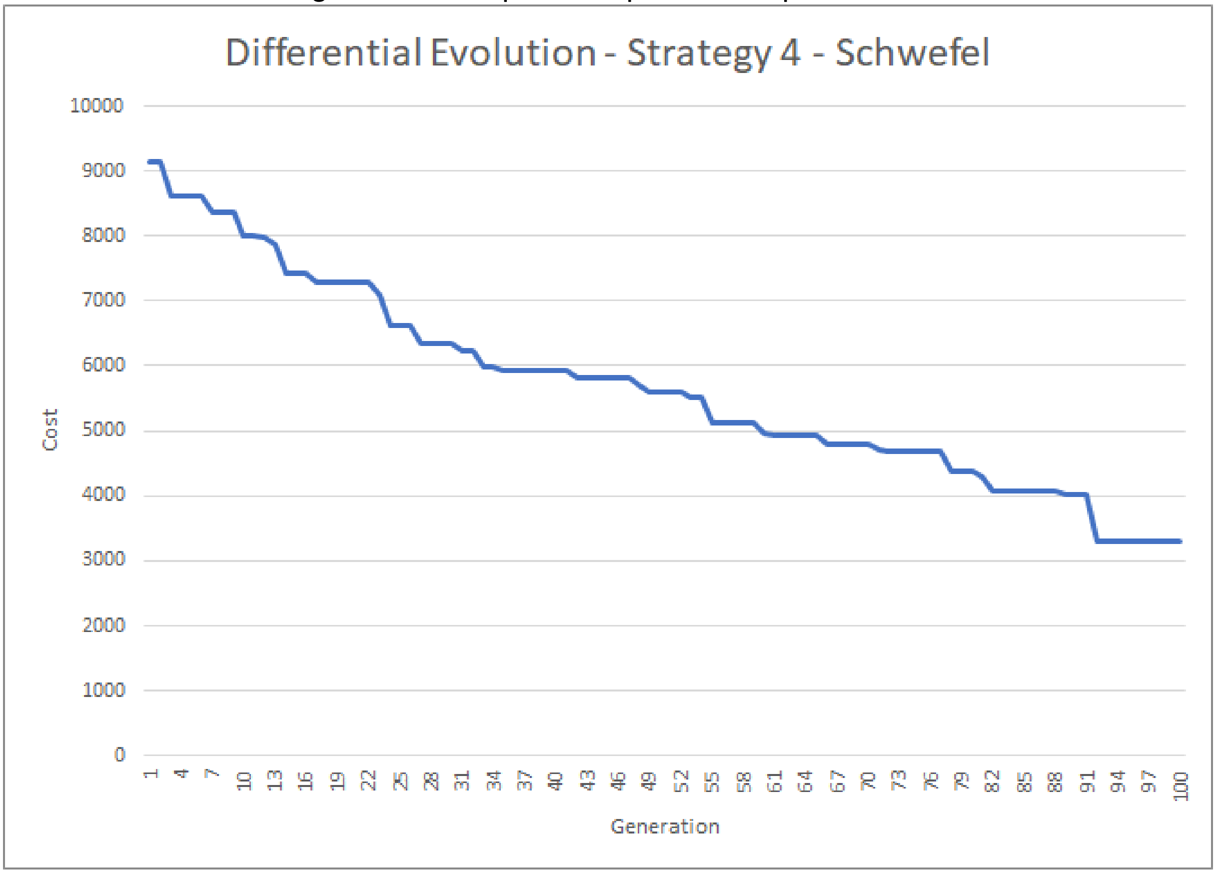


Table 11: Differential Evolution - Strategy 4 - 50 Experiments

F(x)	Average	Std. Deviation	Range	Median	Min
1	4192.13	392.61	1587.30	4243.86	3313.34
2	1939.71	407.22	1675.42	1990.82	1190.95
3	57577200.66	22134638.43	91138265.00	55376571.00	14681815.00
4	41970.15	12387.08	54190.87	42919.91	16036.49
5	13.18	2.59	12.07	14.62	8.10
6	-33.56	0.59	2.63	-33.55	-34.80
7	44.07	1.63	7.53	44.46	39.74
8	56.25	8.53	43.05	57.25	35.33
9	262.24	15.28	75.35	262.52	215.76
10	-12277.01	645.52	2731.21	-12285.22	-13844.73
11	-9995.47	922.60	3971.90	-11578.07	-12033.35
12	4.34	0.63	3.28	4.33	2.69
13	-14.65	0.61	3.17	-14.60	-16.70
14	-3.20	0.46	2.21	-3.15	-4.57
15	8658983.20	3325287.75	17543205.70	5876402.75	1773855.30
16	13.50	2.54	12.05	14.49	7.84
17	2137.31	343.10	1411.92	2181.63	1442.91
18	154.28	15.63	70.54	155.88	111.55

Figure 4: Example of Population Improvement

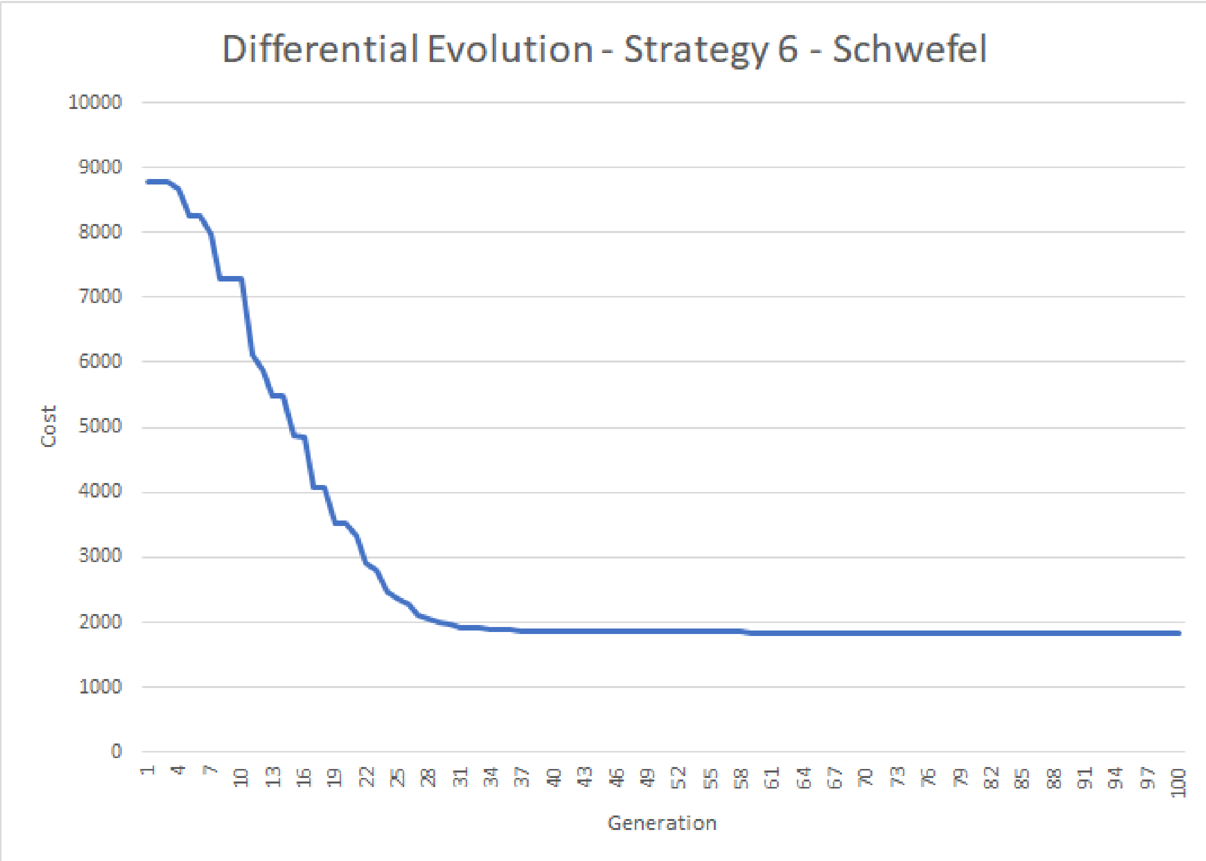


Table 12: Differential Evolution - Strategy 6 - 50 Experiments

F(x)	Average	Std. Deviation	Range	Median	Min
1	3029.23	477.49	2247.99	3060.95	1844.78
2	306.41	247.63	1151.03	319.69	25.03
3	7532626.55	9311763.05	42871038.98	18612839.35	122850.02
4	-32180.24	20721.20	96385.15	-42667.28	-65153.50
5	3.00	1.56	7.97	2.59	1.11
6	-31.35	0.95	4.16	-31.45	-33.55
7	53.02	4.18	19.68	53.09	43.76
8	-17.02	20.80	96.62	1.18	-58.33
9	176.72	41.49	192.04	191.38	69.28
10	-15502.46	1187.81	6224.95	-15660.11	-18780.43
11	-9276.44	1944.66	6890.17	-5520.44	-11769.03
12	7.66	1.29	6.11	7.41	4.88
13	-14.14	2.17	12.23	-13.94	-23.35
14	-4.60	0.81	4.02	-4.65	-6.41
15	1262775.76	1313474.84	5711614.82	302328.54	20456.68
16	8.09	3.52	13.69	6.03	1.46
17	373.91	351.80	1896.50	335.90	50.66
18	128.09	67.46	444.70	153.85	13.36

Figure 5: Example of Population Improvement

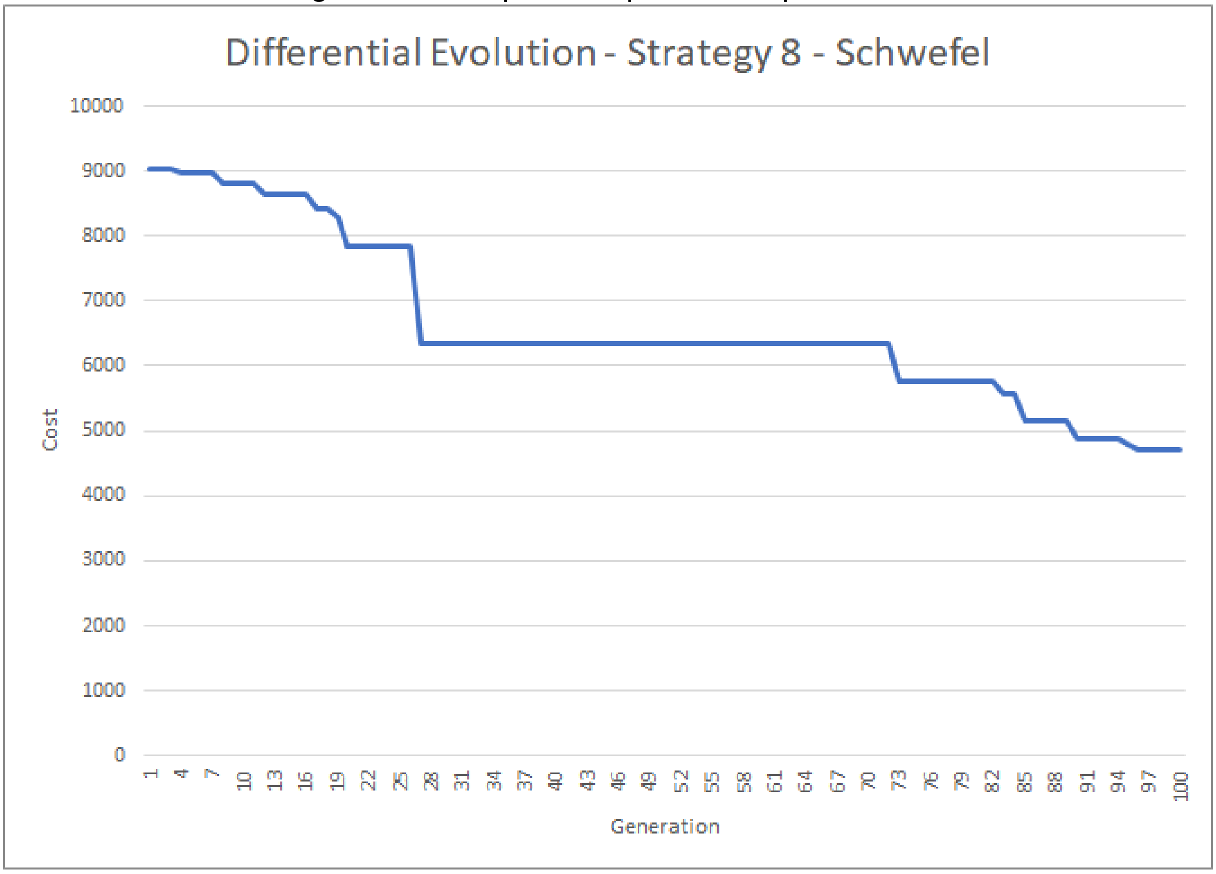


Table 13: Differential Evolution - Strategy 8 - 50 Experiments

F(x)	Average	Std. Deviation	Range	Median	Min
1	6531.14	908.86	3526.90	6750.97	4712.35
2	186.03	136.14	677.47	320.52	5.82
3	2103049.60	2857073.84	12960229.27	3009385.50	22633.73
4	-13408.29	11631.20	49777.96	-26707.32	-36971.04
5	2.34	1.01	4.05	2.10	1.08
6	-33.08	0.67	2.76	-33.00	-34.77
7	50.33	2.37	8.04	50.26	46.42
8	-41.49	24.77	109.72	-49.72	-72.52
9	79.28	32.67	156.69	63.62	25.79
10	-8492.33	1069.25	4132.24	-8078.64	-11109.90
11	-5818.92	995.01	4061.75	-5732.20	-8226.33
12	7.95	1.62	6.47	7.42	5.80
13	-12.33	0.59	2.45	-12.30	-13.49
14	-3.20	0.42	1.69	-3.13	-4.18
15	302240.38	401037.01	2165953.36	322340.66	16669.45
16	2.40	1.29	5.66	2.30	0.30
17	235.75	148.96	815.06	256.21	30.19
18	73.82	26.81	113.09	61.20	29.03

5 New Results

In this section you will find the results of our new experimentations with the firefly algorithm, harmony search, and particle swarm. Note that all results in this section were computed using a population size of 500 over the course of 500 iterations. We tested four different control parameter strategies for each of the three algorithms. Each algorithm includes an example graph which shows how the best and worst fitness values in the population improved over time for the Schwefel function.

In the fitness results tables, better values are colored green while worse values are colored red. For each function we computed the average, standard deviation, range, and median for the four strategies used.

Figure 6: Example of Population Improvement

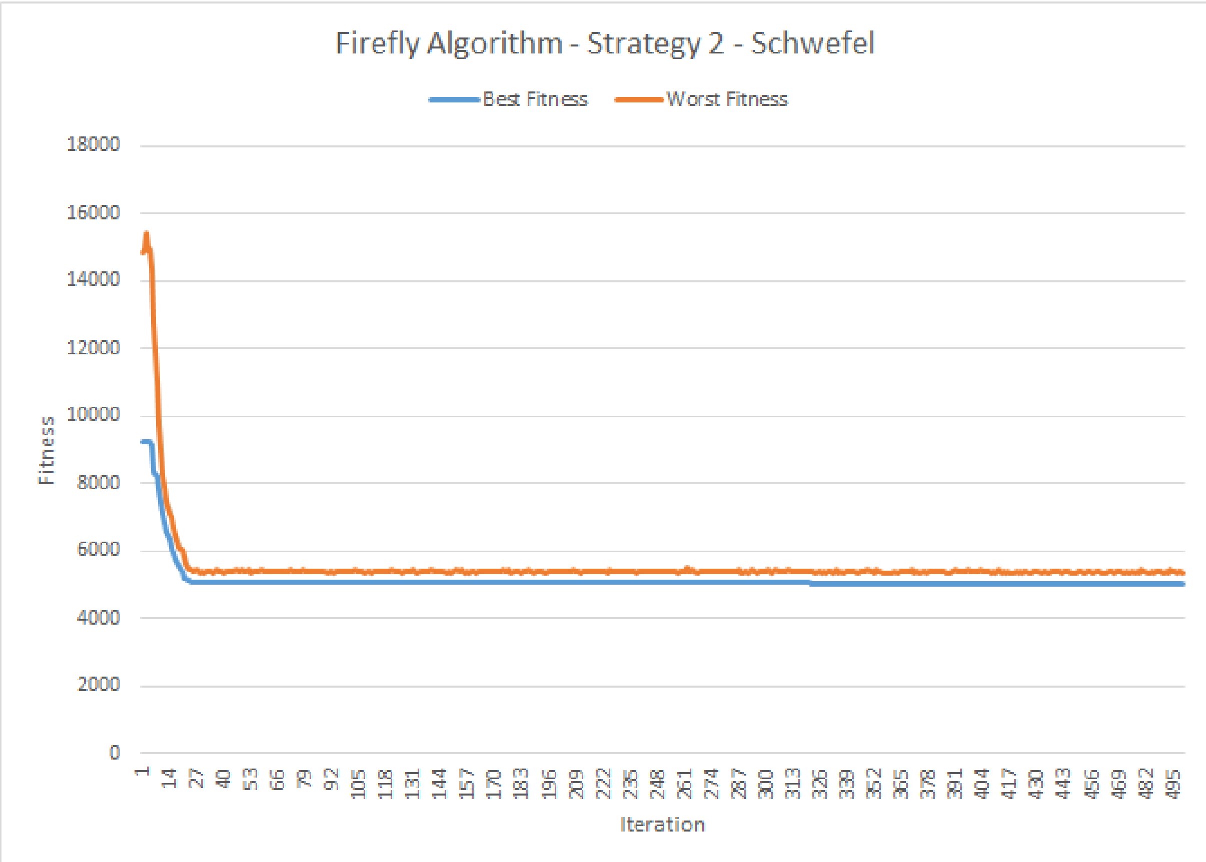


Table 14: Firefly Algorithm - Best Fitness - Functions 1 to 6

	f1	f2	f3	f4	f5	f6
FA Strategy 1	7284.187	17.521	6936.137	94044.346	1.107	-29.958
FA Strategy 2	5047.590	17.920	3713.891	60936.678	1.100	-31.281
FA Strategy 3	5516.222	113.818	112146.410	-23278.446	1.554	-29.089
FA Strategy 4	6897.773	88.209	94480.090	-16734.626	1.584	-33.107
Average	6186.443	59.367	54319.132	28741.988	1.336	-30.859
Standard Dev.	1073.501	49.213	57046.575	57951.543	0.269	1.749
Range	2236.597	96.297	108432.519	117322.792	0.484	4.018
Median	6206.998	53.064	50708.113	22101.026	1.331	-30.620

Table 15: Firefly Algorithm - Best Fitness - Functions 7 to 12

	f7	f8	f9	f10	f11	f12
FA Strategy 1	34.576	273.816	446.094	-11490.775	-7953.729	11.686
FA Strategy 2	40.996	83.669	311.474	-10127.822	-4196.777	11.185
FA Strategy 3	50.967	202.496	273.549	-12331.237	-5413.582	11.772
FA Strategy 4	47.532	126.104	110.613	-11491.573	-5603.625	11.257
Average	43.518	171.521	285.433	-11360.352	-5791.928	11.475
Standard Dev.	7.255	84.072	138.074	912.136	1570.190	0.297
Range	16.391	190.147	335.482	2203.415	3756.952	0.587
Median	44.264	164.300	292.512	-11491.174	-5508.603	11.472

Table 16: Firefly Algorithm - Best Fitness - Functions 13 to 18

	f13	f14	f15	f16	f17	f18
FA Strategy 1	-12.000	-1.484	324.563	46.240	40.154	230.918
FA Strategy 2	-13.571	-3.132	260.194	20.932	40.865	150.645
FA Strategy 3	-11.831	-3.544	9253.344	46.692	158.057	224.942
FA Strategy 4	-10.611	-3.181	7221.784	18.321	138.372	84.184
Average	-12.003	-2.835	4264.971	33.047	94.362	172.672
Standard Dev.	1.215	0.919	4661.604	15.534	62.701	69.378
Range	2.960	2.060	8993.150	28.371	117.903	146.734
Median	-11.915	-3.157	3773.173	33.586	89.618	187.793

Figure 7: Example of Population Improvement

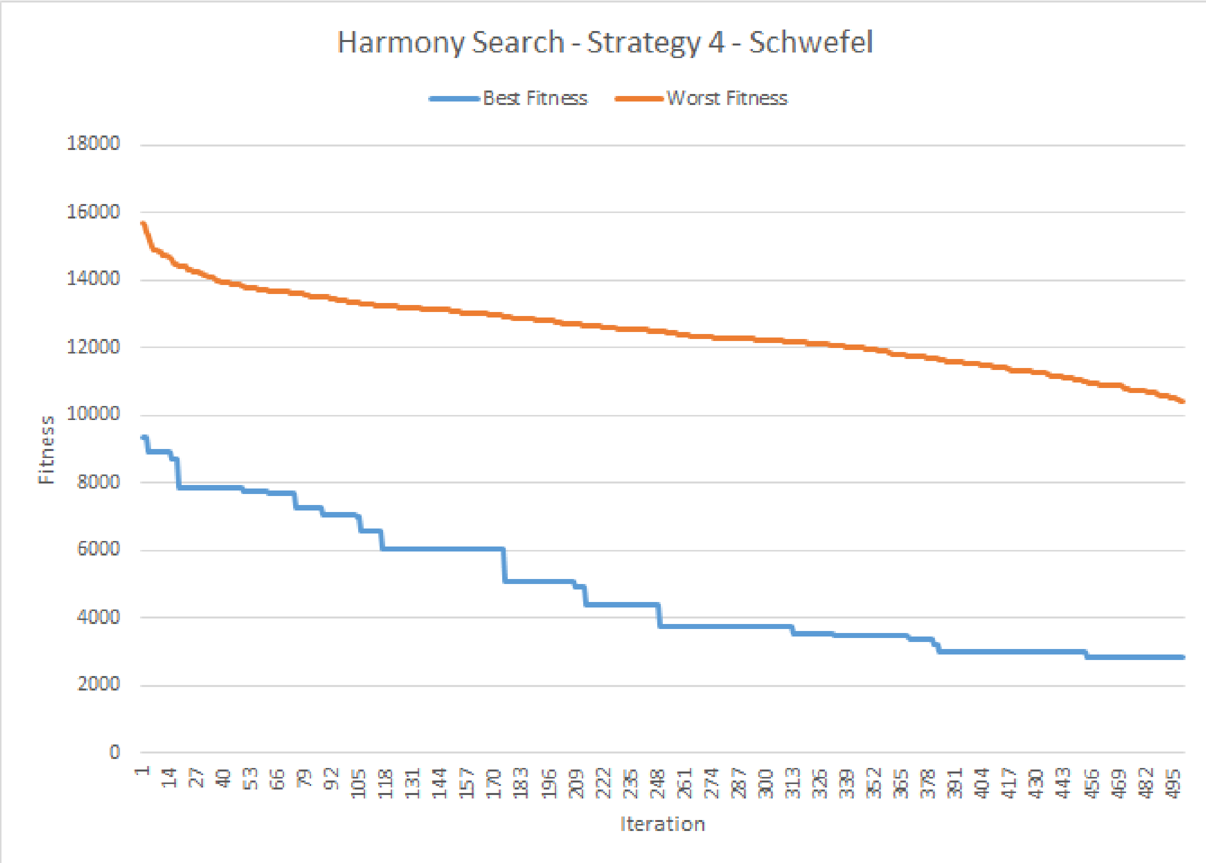


Table 17: Harmony Search - Best Fitness - Functions 1 to 6

	f1	f2	f3	f4	f5	f6
HS Strategy 1	6353.091	29225.010	7.89E+09	616821.520	153.893	-27.115
HS Strategy 2	5115.651	18591.831	4.10E+09	484632.910	119.973	-28.977
HS Strategy 3	3892.947	9722.603	1.44E+09	237710.830	80.683	-29.420
HS Strategy 4	2857.027	9550.773	1.14E+09	156563.330	52.995	-29.635
Average	4554.679	16772.554	3.64E+09	373932.148	101.886	-28.787
Standard Dev.	1513.152	9313.599	3.13E+09	213740.775	44.241	1.148
Range	3496.064	19674.237	6.75E+09	460258.190	100.899	2.520
Median	4504.299	14157.217	2.77E+09	361171.870	100.328	-29.199

Table 18: Harmony Search - Best Fitness - Functions 7 to 12

	f7	f8	f9	f10	f11	f12
HS Strategy 1	64.563	222.704	429.484	-4438.581	-3651.545	13.016
HS Strategy 2	64.667	121.686	378.710	-5299.875	-3512.070	13.044
HS Strategy 3	64.539	123.948	357.348	-5751.109	-4232.369	12.964
HS Strategy 4	57.376	93.597	243.271	-7053.026	-3666.806	12.492
Average	62.786	140.484	352.203	-5635.648	-3765.698	12.879
Standard Dev.	3.607	56.525	78.671	1090.567	318.810	0.260
Range	7.291	129.107	186.213	2614.446	720.299	0.552
Median	64.551	122.817	368.029	-5525.492	-3659.176	12.990

Table 19: Harmony Search - Best Fitness - Functions 13 to 18

	f13	f14	f15	f16	f17	f18
HS Strategy 1	-11.097	-2.319	1.02E+09	84.669	30267.877	345.118
HS Strategy 2	-11.380	-2.653	8.93E+08	72.491	16239.682	239.232
HS Strategy 3	-12.332	-4.241	2.74E+08	45.164	12627.795	196.705
HS Strategy 4	-10.184	-5.128	6.90E+07	22.406	6814.641	143.366
Average	-11.248	-3.585	5.65E+08	56.182	16487.499	231.105
Standard Dev.	0.884	1.327	4.65E+08	27.927	9973.661	85.531
Range	2.148	2.809	9.55E+08	62.262	23453.236	201.752
Median	-11.238	-3.447	5.83E+08	58.827	14433.739	217.968

Table 20: Particle Swarm - Best Fitness - Functions 1 to 6

	f1	f2	f3	f4	f5	f6
PSO Strategy 1	4336.292	9097.622	2.92E+08	158206.610	27.756	-37.204
PSO Strategy 2	4515.867	3149.339	9.11E+07	152689.770	16.067	-33.795
PSO Strategy 3	5102.389	8998.760	1.21E+08	91015.200	51.604	-36.730
PSO Strategy 4	6218.784	7876.113	2.10E+08	124426.480	47.819	-33.149
Average	5043.333	7280.459	1.79E+08	131584.515	35.812	-35.219
Standard Dev.	849.176	2809.246	9.11E+07	30828.961	16.816	2.044
Range	1882.492	5948.283	2.01E+08	67191.410	35.537	4.055
Median	4809.128	8437.436	1.65E+08	138558.125	37.787	-35.262

Figure 8: Example of Population Improvement

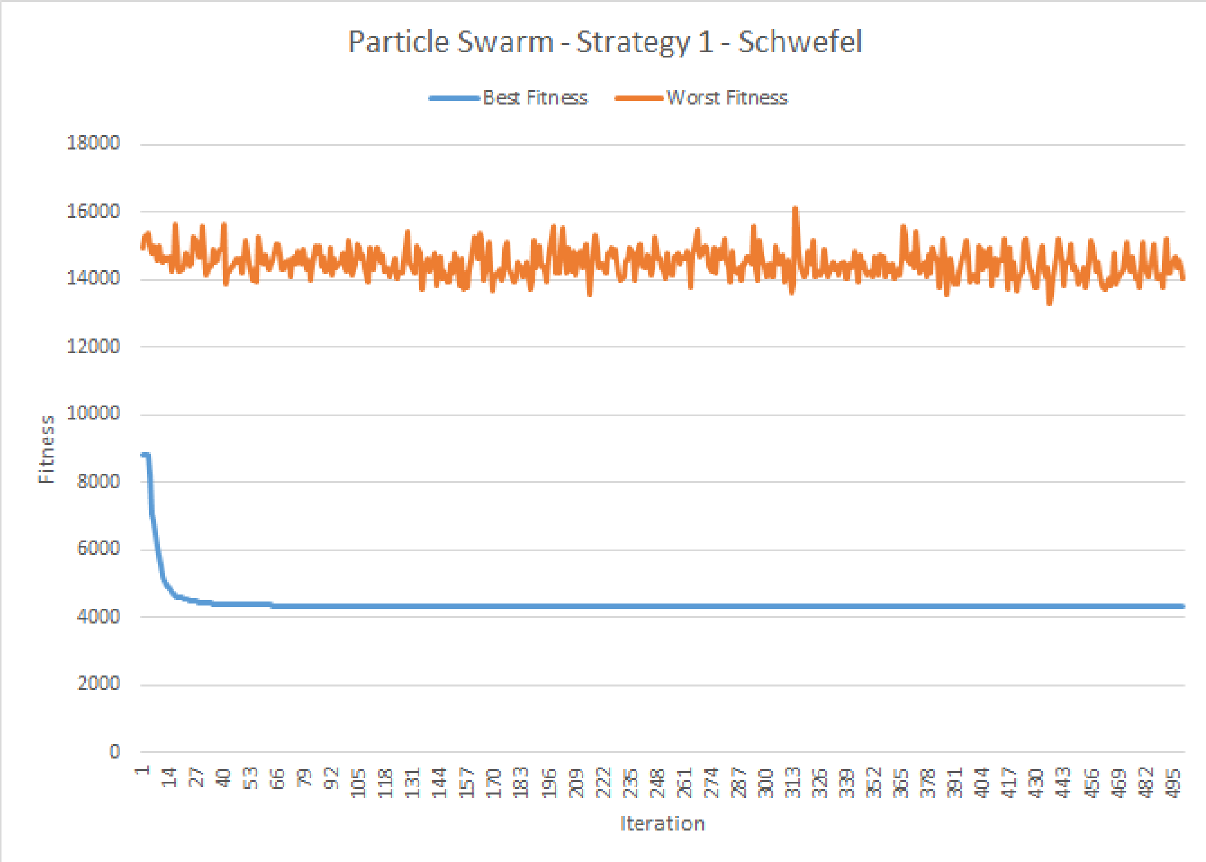


Table 21: Particle Swarm - Best Fitness - Functions 7 to 12

	f7	f8	f9	f10	f11	f12
PSO Strategy 1	32.815	82.699	299.883	-10260.042	-7383.529	11.092
PSO Strategy 2	33.990	91.114	337.475	-12393.610	-7289.826	10.890
PSO Strategy 3	30.209	101.396	220.678	-12858.561	-8539.278	10.623
PSO Strategy 4	31.441	76.522	269.361	-12602.878	-6261.012	11.062
Average	32.114	87.933	281.849	-12028.773	-7368.411	10.917
Standard Dev.	1.642	10.785	49.387	1194.384	931.605	0.215
Range	3.781	24.873	116.796	2598.519	2278.265	0.470
Median	32.128	86.906	284.622	-12498.244	-7336.678	10.976

Table 22: Particle Swarm - Best Fitness - Functions 13 to 18

	f13	f14	f15	f16	f17	f18
PSO Strategy 1	-19.692	-5.284	2.17E+07	28.107	8197.448	99.145
PSO Strategy 2	-19.443	-7.237	1.80E+07	15.499	3536.684	55.018
PSO Strategy 3	-20.925	-5.111	4.09E+07	5.045	7538.421	60.659
PSO Strategy 4	-21.683	-3.757	1.12E+07	8.711	4200.497	51.810
Average	-20.436	-5.347	2.29E+07	14.340	5868.262	66.658
Standard Dev.	1.054	1.433	1.27E+07	10.148	2340.387	21.965
Range	2.239	3.479	2.97E+07	23.062	4660.764	47.334
Median	-20.308	-5.198	1.98E+07	12.105	5869.459	57.838

6 Population Stagnation Results

In this section you will find example tables showing the degree of stagnation in populations for the firefly, harmony search, and particle swarm algorithms. For each algorithm we calculated and plotted the average standard deviation of each population iteration for the best performing strategy utilizing the Schwefel function. Note that larger values are better and indicate a diverse population while smaller values indicate stagnation. We calculated the average standard deviation across all 30 dimensions for each iteration because there did not seem to be any individual dimensions that stagnated more than others. In other words, the stagnation in our tests appeared to be more or less uniform across all 30 dimensions.

Figure 9: Example of Population Standard Deviation

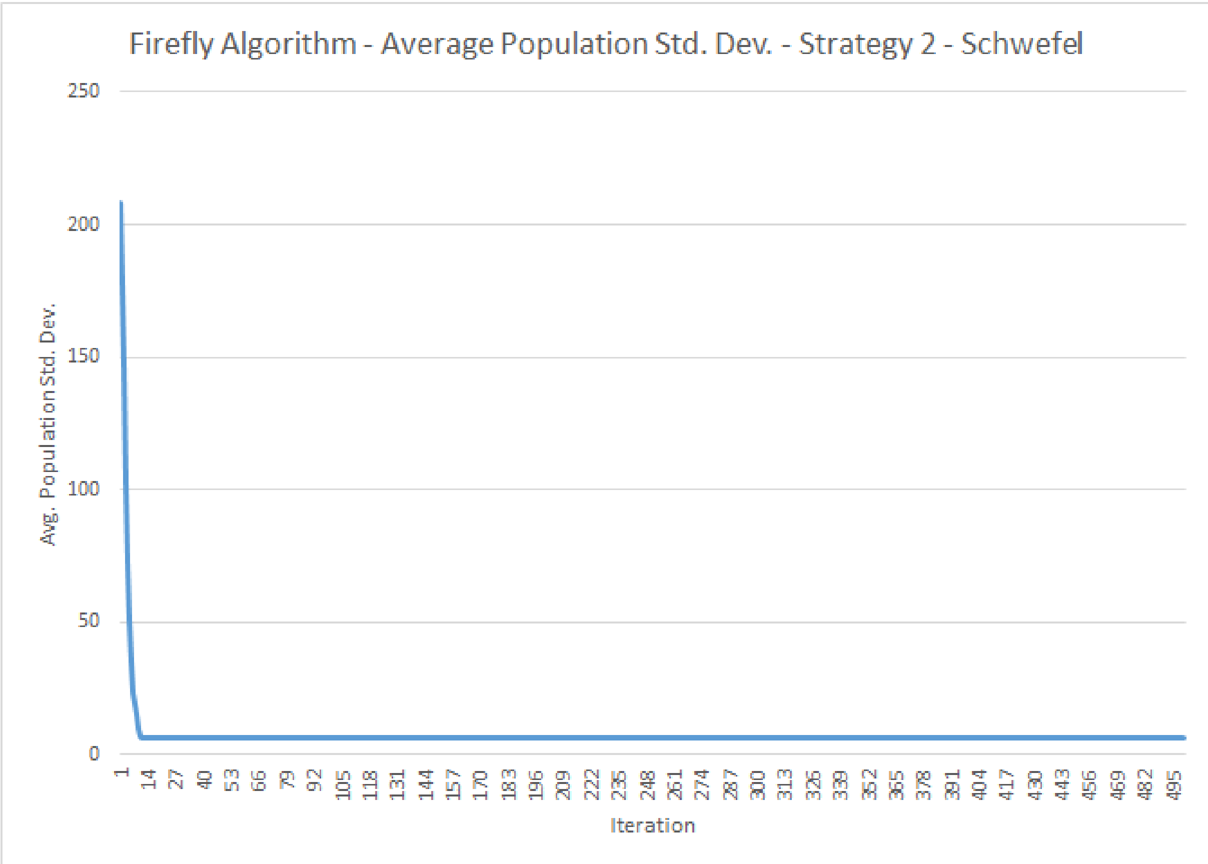


Figure 10: Example of Population Standard Deviation

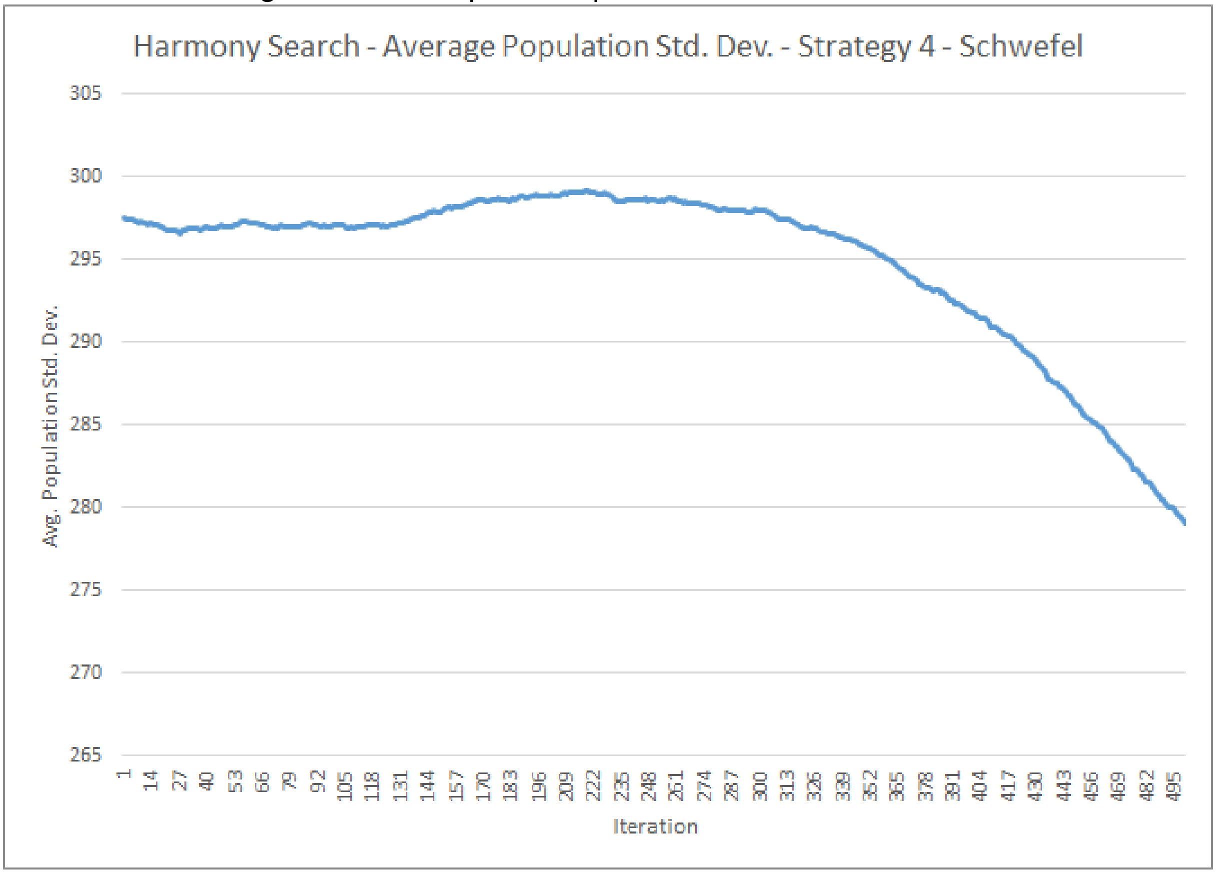
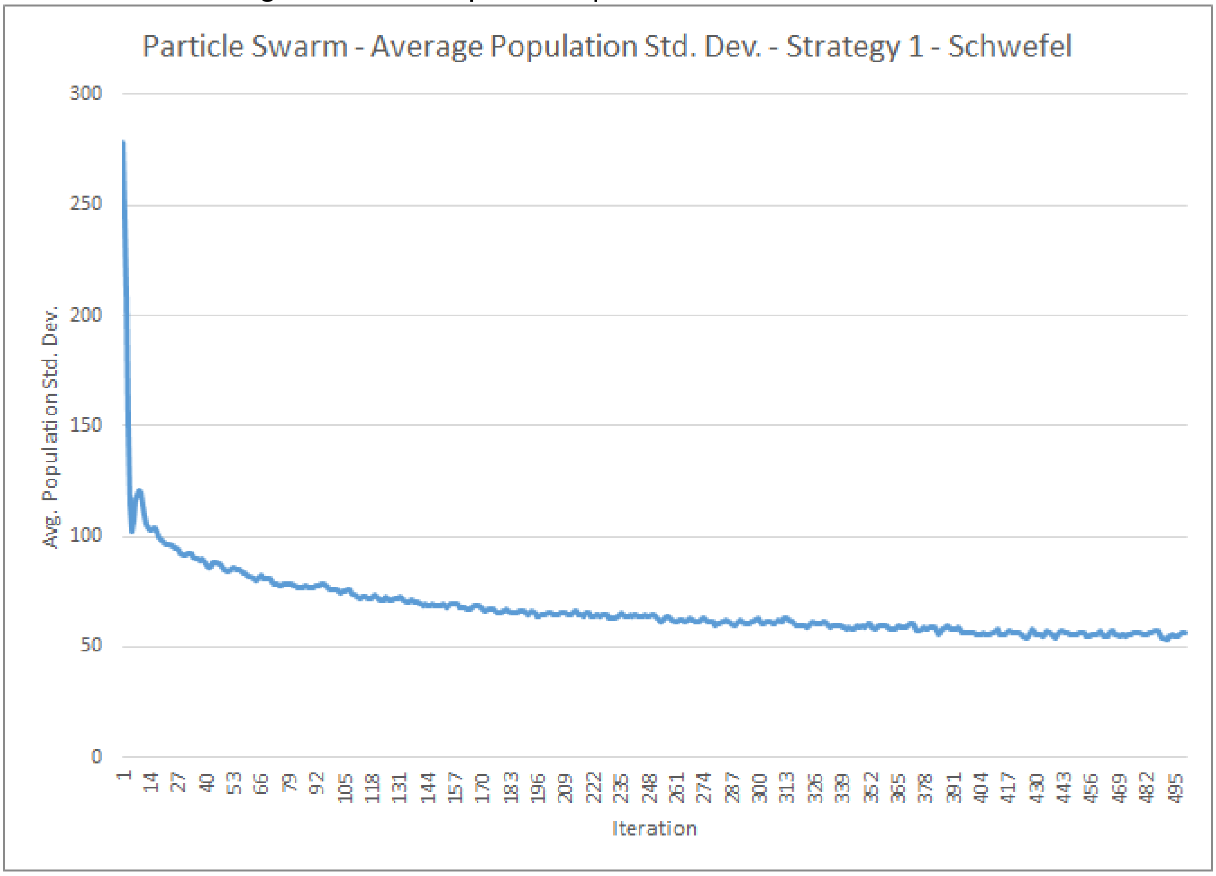


Figure 11: Example of Population Standard Deviation



7 Harmony Search Additional Experimentation

Due to the very good performance from harmony search, we have ran additional experiments using a higher number of iterations, from 1,000 up to 100,000 iterations. As you can see in tables 24, 25, and 25, harmony search tended to experience continued improvement all the way up to 100,000 iterations.

Table 23: Harmony Search - Execution Times (ms)

f(x)	500 Iter.	1,000 Iter.	10,000 Iter.	100,000 Iter.
1	73.26	148.72	4432.90	64236.01
2	73.37	169.96	4677.58	64124.71
3	75.10	211.93	4634.39	63401.66
4	73.20	175.96	4826.26	64580.36
5	83.95	172.24	4561.62	63173.30
6	90.57	255.22	5178.28	66160.70
7	145.70	226.78	4717.30	64667.21
8	74.74	160.72	4473.82	64954.81
9	68.40	153.20	4342.00	62866.16
10	159.72	305.10	4083.52	56709.30
11	160.79	267.91	4233.53	58288.05
12	170.89	365.65	5038.11	62911.49
13	132.32	290.70	5341.01	63814.01
14	126.88	257.63	4892.08	61979.26
15	71.68	202.47	4628.79	62608.39
16	67.96	184.75	4679.43	61942.38
17	73.97	159.25	3212.60	40331.34
18	65.71	109.55	2943.55	39323.47

Table 24: Harmony Search - Best Fitness - Strategy 4 - Functions 1 to 6

Iterations	f1	f2	f3	f4	f5	f6
500	2857.03	9550.77	1142371200.00	156563.33	52.99	-29.64
1,000	1922.58	3878.03	382300090.00	78153.76	17.69	-30.43
10,000	48.83	6.94	39049.00	-22681.58	1.41	-34.18
100,000	0.04	0.51	367.00	-51877.44	1.00	-35.72

Figure 12: Example of Population Improvement

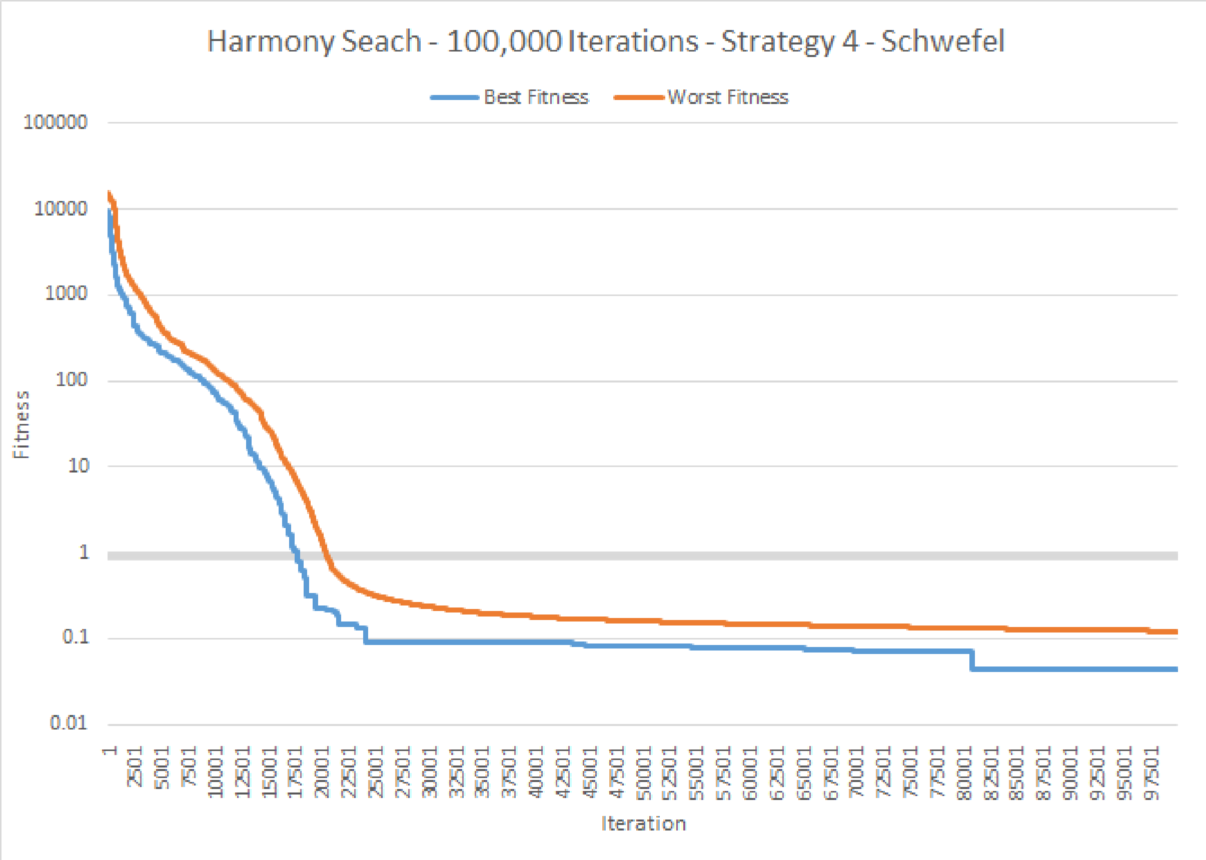


Table 25: Harmony Search - Strategy 4 - Best Fitness - Functions 7 to 12

Iterations	f7	f8	f9	f10	f11	f12
500	57.38	93.60	243.27	-7053.03	-3666.81	12.49
1,000	48.56	24.25	223.53	-9573.78	-7615.89	12.73
10,000	31.70	-72.38	49.47	-19809.99	-12434.33	8.49
100,000	31.19	-75.43	25.53	-22213.56	-13209.75	7.56

Table 26: Harmony Search - Strategy 4 - Best Fitness - Functions 13 to 18

Iterations	f13	f14	f15	f16	f17	f18
500	-10.18	-5.13	68994564.00	22.41	6814.64	143.37
1,000	-11.98	-4.35	43485740.00	10.21	3923.71	67.48
10,000	-14.47	-13.94	735.00	0.44	14.98	32.62
100,000	-17.13	-18.23	0.00	0.23	10.57	14.18

8 Analysis

First, let's look at the population stagnation results in figures 9, 10, and 11. The firefly algorithm stagnates extremely quickly around iteration 10, and stays that way all the way to iteration 500. The standard deviation for our population reduced down to a value of around 5. Considering that the bounds for the Schwefel function is set to -512 to 512, this shows that we have very little diversity in our populations. This most likely explains the poor results we are seeing from firefly, which we will discuss shortly. The particle swarm algorithm stagnates as well, but more slowly and to not the same degree. By iteration 500 the standard deviation of the population is a little over 50. Finally, harmony search performed the best when looking at population diversity, which did not start to stagnate until around iteration 300. Even so, by iteration 500 harmony search still had an average population standard deviation of nearly 280, which shows that it's population was far more diverse than the other two algorithms. Using this knowledge, we can deduce that harmony search is the best candidate for running more iterations. Since harmony search is also the fastest algorithm of the three, we tested iterations up to 100,000 in Section 7, which produced the best results by far.

Next, let's look at each algorithm and see which strategies performed the best for each. For the firefly algorithm, each strategy had at least one objective function where it worked the best. This shows that the firefly algorithm's performance is highly dependent on the objective function landscape, and thus the parameters need to be tuned specifically for each objective function to improve performance. The ranges of best fitness values can vary a lot between the different strategies. For example, function 3 and function 4 have a huge variance in fitness values between the different strategies. Firefly did perform well for certain functions, like function 2 where it's best result was 17.521. In contrast, harmony search's best result for function 2 with 500 iterations was 9,550.773, and particle swarm's best was 3149.339. Firefly also performed much better on function 15 than the other two.

Harmony search is interesting because over 500 iterations it's fitness optimization performance is not great for most functions. In fact, firefly out-performed harmony search in all but two functions, function 1 and 14. This however is not the strength of harmony search. As we said previously, harmony search was 1,500 times faster than firefly. Because of this, we are able to push harmony search to much higher iterations than firefly. Even at 100,000 iterations, harmony search still only required about one minute per objective function. Firefly in contrast used up to 5 minutes of execution time for just 500 iterations. Taking this into account, running harmnony search for 100,000 iterations is very reasonable and produces great results. Looking at tables 24, 25, and 26, you can see harmony search has produced the best results by far out of all other search algorithms we have tested while requiring similar execution times. We can also see in figure 12 that harmony search continued to find fitness improvements nearly all the way to 100,000 iterations. Finally, out of the four strategies we tested, strategy 4 performed the best for nearly all objective functions. This shows that harmony search is more robust than firefly, and requires less tuning to get good results.

Like harmony search, particle swarm also tended to perform worse than firefly over 500 iterations. The difference however is that particle swarm stagnated very quickly, thus leading to higher iterations not producing much better results. Particle swarm also requires more execution time than harmony search, but it is still much faster than firefly. Having said that, between the two firefly still performed better overall. Another similarity with firefly is particle swarm did not have a particular strategy that seemed to work best for all objective functions, again implying that particle swarm needs it's parameters to be hand tailored for each function to produce the best results.

The last thing we are going to look at is the change over time between the best and worst fitness values in the populations for each algorithm. First, when looking at figure 6, we can see how after a few iterations the worst fitness approaches very closely to the best fitness and stays there. This reinforces our previous observation that firefly stagnated very quickly and had little diversity between populations. Then, when we look at figure 7 we see the exact opposite. Harmony search maintains a worse fitness that is a fair distance away from it's best fitness all the way up to 500 iterations. This shows that harmony search tends to have a much more diverse population than firefly. Particle swarm is interesting in figure 8, where the worst fitness fluctuates but does not really improve over time despite some improvements to the best fitness. From this we can conclude that although particle swarm is more diverse than firefly, it does not prevent the algorithm from stagnating.

9 Conclusions

In this report we have consolidated our experiment results for three different search algorithms: Firefly, harmony search, and particle swarm. Immediately it was clear that the firefly algorithm performs poorly despite producing some good results due to the massive difference in execution times from harmony search and particle swarm. Harmony search performed very well, especially considering the comparatively small execution time versus the other two algorithms. This allowed us to run harmony search for many more iterations, up to 100,000 without exceeding the relative execution time needed by firefly. These experiments also produced the best results out of any algorithm we have tested so far. Therefore it is hard to recommend any search algorithm we have tested so far over harmony search. At 100,000 iterations harmony search got very close to the global minimums. For example, with the Schwefel function it got down to a fitness of just 0.04. In conclusion firefly and particle swarm require more work to reduce stagnation and improve viability for single objective optimization problems. Harmony search in contrast is much easier to work with, and produces far better results within similar execution times.