CS 471 - Project 2

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

April 22, 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 two different search methods in various dimension sizes. 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 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.

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	$10, 20, 30$	$[-100, 100]^n$

2 Methods

Our method was to test two different main search algorithms, blind search and local search. Local search is executed a certain number of times, resulting in a method called iterative local search. We executed both of these search algorithms 30 times for

each of the 18 functions, using populations of 30 vectors in three different dimensions: 10, 20, and 30. The population vectors were generated using a uniformly stochastic random number generator. Each function population is generated between certain bounds as seen in Table 1. Our goal is to produce a fitness value from a generated population that is close as possible to the optimal f(x*) for each function. All fitness data is recorded and ran through a few different statistical analysis functions: average, standard deviation, range, and median. We also recorded the total amount of time in milliseconds that it takes to run each iteration of the search algorithms. An important note regarding local search is that we used a fixed alpha value of 0.11 for all functions. Further testing could improve results by tailoring this value for each of the 18 functions.

3 Execution Times

Table 2: Total Execution Times - 30 Iterations - All Functions

	10 Dim. (sec)	20 Dim. (sec)	30 Dim. (sec)
blind search	0.00474	0.00580	0.00660
local search	24.00245	89.02809	209.24550

Table 3: Single Iteration Execution Times - Blind Search

	10 Dimensions (ms)		20 Dimensions (ms)		30 Dimensions (ms)	
f	Min	Max	Min	Max	Min	Max
1	0.03	1.74	0.04	0.23	0.06	0.19
2	0.02	0.03	0.01	0.02	0.02	0.03
3	0.01	0.02	0.01	0.02	0.03	0.03
4	0.03	0.05	0.03	0.08	0.04	0.06
5	0.04	0.05	0.04	0.10	0.05	0.07
6	0.03	0.05	0.03	0.07	0.04	0.06
7	0.07	0.10	0.08	0.10	0.10	0.14
8	0.04	0.07	0.04	0.09	0.07	0.08
9	0.09	0.13	0.09	0.13	0.15	0.22
10	0.05	0.50	0.06	0.07	0.08	0.77
11	0.09	0.11	0.11	0.12	0.12	0.58
12	0.04	0.04	0.04	0.05	0.05	0.06
13	0.07	0.15	0.09	0.09	0.09	0.12
14	0.06	0.06	0.07	0.09	0.07	0.10
15	0.02	0.02	0.02	0.02	0.02	0.03
16	0.04	0.06	0.04	0.04	0.04	0.06
17	0.01	0.05	0.02	0.02	0.02	0.33
18	0.02	0.02	0.03	0.04	0.04	0.06

The first thing we are going to look at is the execution times for our tests. Looking at Table 2, it is clear that local search takes considerably more execution time than blind search. In addition, the local search times rapidly scale up with increased dimension sizes, while blind search does not seem to be affected much, if at all. Note that the total execution times in Table 2 are the sums for all 18 functions, with 30 iterations of each. Also note that the these times are from multi-threaded runs, utilizing 8 threads. Single threaded performance would be far worse.

Table 3 and Table 4 break these execution times down into the individual functions, comparing the difference between the min and max execution times for a single iteration. Table 3 shows that the blind search algorithm produces very consistent execution times for all 18 functions. Despite the one outlier in function 1 for 10 dimensions, there isn't much variation here. For the most part blind search completed all 30 iterations for each function in less than a millisecond. Local search however tells a different story.

Table 4: Single Iteration Execution Times - Local Search

	10 Dimensions (ms)		20 Dimensions (ms)		30 Dimensions (ms)	
f	Min	Max	Min	Max	Min	Max
1	5.95	30.90	27.13	92.46	54.83	226.69
2	0.06	0.20	0.15	0.52	0.30	1.04
3	0.01	0.01	0.02	0.02	0.03	0.04
4	0.02	0.04	0.05	0.05	0.08	0.08
5	4572.48	6115.94	17607.70	24396.40	40820.40	57527.70
6	0.02	0.03	0.05	0.06	0.08	0.11
7	0.07	338.32	0.57	2484.65	2.21	4803.42
8	0.11	4.45	0.48	3.92	1.12	9.33
9	0.12	54.10	0.31	167.10	0.88	380.48
10	0.21	204.44	27.11	2769.17	1.44	2367.77
11	0.06	405.90	0.14	4875.01	0.34	7853.03
12	0.02	0.03	0.05	0.08	0.09	0.12
13	0.05	0.07	0.11	0.15	0.18	0.27
14	0.04	4403.19	0.09	6540.68	0.15	17906.60
15	0.01	0.01	0.02	0.02	0.02	0.03
16	0.04	11.48	0.05	41.74	6.16	91.13
17	0.04	0.07	0.12	0.18	0.22	0.31
18	0.02	0.04	0.04	0.09	0.07	0.19

Table 4 is a lot more interesting. There is a huge variance in execution times here between the different functions. The first thing to note is the uniform scaling of function 1 - Schwefel. Both the min and max execution times for this function consistently increase as the dimensions increase. Function 5 exhibits similar behavior, except that the execution times are many multiples higher. From this observation we can deduce that the local search algorithm is consistently able to improve the resulting fitness from function 1 and 5 over many iterations. Looking at the min times from all other functions, it is clear that for most of them local search is not able to consistently find significant

improvements during a single iteration. This could be a result of a non-optimal alpha value, bad luck, bad floating point precision, or some other issue. More testing is needed on this front in order to improve the consistency of local search fitness improvements. Times colored in red show this discrepancy between the min and max times at it's worst.

4 Project 1 Results Review

To recall the results from project 1, you can find the them in this section.

Table 5: Stochastic Experiment Results - 10 Dimensions

f	Average	Std. Deviation	Range	Median	Time (ms)
1	4,019.88	492.55	1,809.48	3,873.74	0.0551
2	34,579.22	9,706.26	37,931.35	33,815.30	0.0012
3	1.79E+10	8.98E+09	3.54E+10	1.74E+10	0.0015
4	321,550.56	72,051.13	280,613.40	306,918.72	0.0221
5	219.16	64.16	288.32	207.19	0.0388
6	-6.86	0.73	2.49	-6.89	0.0195
7	31.53	4.09	15.71	31.76	0.1106
8	175.07	31.66	171.88	175.27	0.0334
9	182.15	8.39	39.62	181.82	0.1066
10	8.65	847.98	4,033.26	-20.69	0.0387
11	24.96	603.32	2,656.54	-145.63	0.0794
12	4.52	0.20	0.92	4.50	0.0215
13	-1.04	0.50	2.03	-1.01	0.0900
14	-0.01	0.12	0.94	0.00	0.0541
15	1.05E+09	5.44E+08	2.39E+09	1.02E+09	0.0012
16	140.85	59.85	221.74	146.29	0.0229
17	32,044.73	9,942.17	38,758.74	33,330.50	0.0009
18	338.92	98.25	354.77	329.99	0.0210

The results for our experiments in 10, 20, and 30 dimensions can be seen in Table 5, Table 6, and Table 7, respectively. Table entries colored in red are column maximums, green entries are those that are closest to zero for that column, and blue entries are the best results, i.e. closest to their optimal value f(x*) for that function.

Table 6: Stochastic Experiment Results - 20 Dimensions

f	Average	Std. Deviation	Range	Median	Time (ms)
1	8,204.07	1,032.94	5,548.10	8,238.37	0.0866
2	68,629.45	14,068.63	58,755.97	68,627.08	0.0013
3	3.69E+10	1.07E+10	4.84E+10	3.72E+10	0.0025
4	1.24E+06	2.02E+05	9.48E+05	1.18E+06	0.0414
5	440.08	90.04	361.81	457.95	0.0485
6	-14.10	1.02	4.63	-14.11	0.0400
7	64.41	7.40	29.30	65.59	0.2183
8	380.79	48.38	198.92	382.92	0.1011
9	383.01	10.20	45.69	383.26	0.2785
10	117.29	1,050.96	4,397.03	-17.11	0.0845
11	-142.35	808.78	3,288.88	-22.48	0.1686
12	9.59	0.32	1.41	9.54	0.0404
13	-2.24	1.17	5.15	-1.91	0.2283
14	-0.01	0.06	0.33	0.00	0.1132
15	4.38E+09	1.44E+09	7.29E+09	4.17E+09	0.0020
16	331.53	72.68	311.47	331.26	0.0432
17	66,477.11	13,505.11	71,380.45	66,014.58	0.0013
18	610.66	113.03	437.06	617.00	0.0412

Table 7: Stochastic Experiment Results - 30 Dimensions

f	Average	Std. Deviation	Range	Median	Time (ms)
1	12,301.16	985.68	4,887.74	12,355.14	0.0810
2	103,117.42	17,520.46	86,154.48	104,561.49	0.0018
3	6.22E+10	1.36E+10	6.27E+10	6.07E+10	0.0036
4	2.62E+06	3.68E+05	1.24E+06	2.65E+06	0.0618
5	662.14	117.59	448.00	666.04	0.0721
6	-20.78	1.03	4.37	-20.69	0.0609
7	98.51	10.18	44.33	100.10	0.2932
8	582.41	64.82	248.54	587.76	0.1068
9	585.29	16.41	66.94	584.79	0.3169
10	-249.12	1,672.73	8,184.98	-287.59	0.1220
11	-345.75	818.74	3,679.20	-427.49	0.2809
12	14.50	0.41	2.22	14.55	0.0650
13	-3.59	1.03	4.40	-3.72	0.2573
14	-0.02	0.17	0.96	0.00	0.1703
15	9.18E+09	2.21E+09	1.00E+10	9.17E+09	0.0027
16	437.67	119.36	546.59	424.54	0.0617
17	99,729.83	14,470.03	61,171.12	101,060.69	0. 0018
18	974.59	164.92	553.69	938.47	0.0603

5 New Results

In this section you will find all results and statistical analysis for both the blind search and local search algorithms in 10, 20, and 30 dimensions. Note that the statistics for each function f_n is over the course of 30 iterations. We did not differentiate between local search ($1\ iteration$) and iterative local search ($> 1\ iterations$) because single iteration local search is just a subset of iterative local search. Table entries colored in red are column maximums, green entries are those that are closest to zero for that column, and blue entries highlight functions that produce the best results, i.e. closest to their optimal value f(x*) for that function.

Table 8: Blind Search - 10 Dimensions - 30 Iterations

f	Average	Std. Deviation	Range	Median	Avg. Time (ms)
1	2944.83	315.52	1240.76	2978.93	0.12
2	15267.40	3084.60	13221.69	15705.50	0.02
3	4.08E+09	2.32E+09	9.19E+09	4.26E+09	0.02
4	137205.87	38162.19	153540.40	141574.50	0.04
5	89.45	27.83	96.75	98.09	0.05
6	-8.10	0.37	1.66	-8.12	0.04
7	19.25	2.03	9.79	19.41	0.09
8	119.02	15.60	65.23	117.01	0.05
9	152.41	11.33	41.26	154.08	0.11
10	-1913.65	518.21	2133.55	-1784.95	0.07
11	-1242.26	266.56	871.74	-1175.56	0.09
12	3.99	0.19	0.79	4.02	0.04
13	-2.82	0.41	1.74	-2.72	0.08
14	-0.36	0.23	0.84	-0.30	0.06
15	2.47E+08	1.27E+08	5.90E+08	2.48E+08	0.02
16	52.17	18.41	64.81	55.06	0.04
17	15915.49	3299.22	12953.92	15373.15	0.02
18	174.08	37.28	169.03	181.30	0.02

Table 9: Local Search - 10 Dimensions - 30 Iterations

f	Average	Std. Deviation	Range	Median	Avg. Time (ms)
1	1581.19	330.48	1207.94	1656.35	13.35
2	0.03	0.00	0.01	0.03	0.13
3	4.52E+09	2.19E+09	8.61E+09	4.30E+09	0.01
4	142966.90	29072.33	91297.00	136259.00	0.02
5	1.00	0.00	0.00	1.00	5306.73
6	-8.29	0.49	1.83	-8.35	0.02
7	9.80	1.14	5.26	9.34	46.19
8	80.58	17.82	73.20	82.25	0.62
9	145.66	8.00	35.45	146.06	3.05
10	-4039.61	859.35	3161.73	-3685.56	52.00
11	-2497.59	574.14	2273.17	-2541.98	94.86
12	3.94	0.24	1.13	4.00	0.02
13	-2.90	0.52	2.08	-2.96	0.05
14	-0.51	0.38	1.45	-0.30	158.49
15	2.81E+08	1.34E+08	5.11E+08	2.39E+08	0.01
16	15.92	10.46	43.75	13.05	7.02
17	3.05	0.03	0.10	3.04	0.05
18	129.79	35.98	142.37	130.33	0.02

Table 10: Blind Search - 20 Dimensions - 30 Iterations

f	Average	Std. Deviation	Range	Median	Avg. Time (ms)
1	6521.16	422.89	1773.63	6466.02	0.05
2	40544.85	5837.89	23767.40	42004.35	0.02
3	1.63E+10	4.27E+09	1.71E+10	1.64E+10	0.02
4	740094.07	118027.76	563741.00	752434.00	0.04
5	266.87	28.48	110.68	265.89	0.05
6	-16.09	0.71	3.16	-15.98	0.04
7	47.23	3.96	17.91	47.11	0.09
8	279.03	31.83	121.92	283.36	0.06
9	347.03	11.79	46.77	350.47	0.12
10	-2814.89	719.53	3093.46	-2660.70	0.07
11	-1741.43	468.21	2349.50	-1670.74	0.11
12	8.73	0.21	0.93	8.73	0.05
13	-4.41	0.69	3.38	-4.35	0.09
14	-0.49	0.23	0.85	-0.51	0.08
15	1.59E+09	3.50E+08	1.33E+09	1.60E+09	0.02
16	146.01	27.80	93.33	149.22	0.04
17	40922.22	5770.62	21896.10	41484.85	0.02
18	417.85	48.20	195.83	422.98	0.04

Table 11: Local Search - 20 Dimensions - 30 Iterations

f	Average	Std. Deviation	Range	Median	Avg. Time (ms)
1	3447.45	497.15	1806.62	3489.39	51.48
2	0.06	0.00	0.01	0.06	0.38
3	1.74E+10	4.09E+09	1.93E+10	1.74E+10	0.02
4	707591.87	106009.17	502076.00	708505.50	0.05
5	1.00	0.00	0.00	1.00	20950.33
6	-16.05	0.52	2.23	-15.93	0.05
7	22.12	3.14	13.96	21.01	260.96
8	213.08	25.45	87.55	220.22	1.91
9	325.94	11.93	44.88	324.67	6.42
10	-7788.24	853.15	3587.89	-7802.68	422.17
11	-4371.91	1482.73	4700.34	-4920.22	452.04
12	8.72	0.22	0.98	8.78	0.05
13	-4.48	0.49	1.81	-4.45	0.11
14	-0.62	0.34	1.46	-0.72	364.61
15	1.57E+09	4.02E+08	1.67E+09	1.52E+09	0.02
16	43.06	29.82	132.31	28.36	30.49
17	6.14	0.04	0.18	6.16	0.15
18	317.45	86.28	305.32	313.46	0.06

Table 12: Blind Search - 30 Dimensions - 30 Iterations

f	Average	Std. Deviation	Range	Median	Avg. Time (ms)
1	10502.84	383.22	1592.69	10465.35	0.08
2	68757.26	7670.73	33798.20	68587.80	0.03
3	3.17E+10	5.59E+09	2.46E+10	3.09E+10	0.03
4	1758367.33	250518.28	1065510.00	1768355.00	0.06
5	425.79	44.73	209.79	432.34	0.07
6	-24.05	0.90	3.83	-23.90	0.06
7	76.07	3.76	14.00	76.78	0.13
8	460.59	32.39	139.24	467.33	0.08
9	541.21	12.85	62.85	543.50	0.18
10	-3449.60	793.66	3781.37	-3229.55	0.11
11	-2231.85	569.79	2746.48	-2228.94	0.18
12	13.66	0.21	0.77	13.66	0.06
13	-6.13	0.70	3.26	-5.95	0.12
14	-0.57	0.33	1.67	-0.62	0.09
15	4.32E+09	8.37E+08	4.11E+09	4.42E+09	0.03
16	258.74	37.28	184.23	257.53	0.06
17	69049.46	8105.32	36845.40	70051.20	0.03
18	675.36	60.63	316.87	673.45	0.05

Table 13: Local Search - 30 Dimensions - 30 Iterations

f	Average	Std. Deviation	Range	Median	Avg. Time (ms)
1	5134.00	521.29	2144.33	5011.16	108.88
2	0.09	0.00	0.01	0.09	0.74
3	3.15E+10	3.90E+09	1.62E+10	3.17E+10	0.03
4	1903896.00	227903.55	880340.00	1960160.00	0.08
5	1.00	0.00	0.00	1.00	48424.05
6	-23.93	0.74	3.00	-23.67	0.08
7	34.58	4.26	20.06	33.13	607.07
8	352.28	24.91	108.66	352.02	3.97
9	498.68	15.16	63.40	500.62	14.34
10	-11490.52	1844.84	9890.88	-11958.25	644.98
11	-6731.40	1768.29	7128.38	-7097.64	1773.02
12	13.66	0.26	1.04	13.72	0.09
13	-5.99	0.68	3.01	-5.93	0.19
14	-0.87	0.37	1.35	-0.74	1206.63
15	4.38E+09	1.12E+09	4.52E+09	4.28E+09	0.03
16	51.85	17.96	55.98	41.92	68.71
17	9.20	0.06	0.32	9.21	0.25
18	514.43	105.54	402.18	502.35	0.12

6 Analysis

When comparing our new results to the old results, immediately we can see see a few things. First of all, both blind search and local search produce consistently better results than the original pure stochastic method from project 1. Blind search in particular produces better results with minimal execution time overhead. Local search produces the best results, but with this comes a considerable execution time overhead for some functions, in particular functions 5, 11, and 14. Function 5 is by far the most time consuming function. With 30 dimensions, local search took an average of 48 seconds per single iteration on function 5.

During project 1 we noted that function 6 performed relatively well and scaled with the increasing dimensions. While this is still true, there is not a big improvement when using blind search or local search over the previous method.

Where local search really shines is functions 2 and 5. In all dimensions our average fitness values are very close to 0 for both functions. Function 5 has a standard deviation and range of 0 in every dimension, which means every one of the 30 iterations of local search produced the same fitness value. Function 5 also happens to take the longest execution time. Function 2 produces the best average fitness values as it is the closest to it's optimal value of 0. When you compare the local search results for these functions to the blind search results, the difference is huge. In 30 dimensions, local search produced an average of 0.09 fitness for function 2 while the best blind search could produce is over 68,000.

Unfortunately however, local search did not perform well for all functions. In particular, functions 3, 4, and 15 saw little to no improvements with local search over the other methods despite having extremely high fitness values. These functions also happen to have some of the smallest local search execution times out of all the other functions, which tells us that local search is failing to improve the neighboring fitness values for these functions. Either a different method is needed, or further experimentation is required. One thing that we did not try is to adjust the alpha value to try and get better performance on a per-function basis. So far we have only tried using a global constant alpha value of 0.11 that is shared across all functions.

7 Conclusions

As you can see from our analysis of this experiment's results, the blind search and local search methods are definitely an improvement over the previous purely stochastic method. Functions 2 and 5 produced very good results with local search, getting extremely close to the optimal values for both functions. Blind search is a slight improvement over the stochastic method with minimal overhead. Local search can produce very long execution times, but only for the functions where it is most effective. As we noted before, increasing the population dimensions tends to worsen the results, but local search helped mitigate this with mixed results. While local search is currently the best method we have, it is still a ways off from producing optimal results from all functions. There is still some room for improvement with local search, specifically with testing various alpha values for specific functions.