Natural Computing Assignment

Genetic Programming

Problem 1: PSO

Problem 1.1

[**Result**] In this problem, the dimension of the **Sphere** function is set to 50 and population size is set to 40. $w = 0.82$, $a1 = 1.2$, $a2 = 0.9$, which averagely converge in 700 steps. The dimension of the **Sphere** function is set to 4 and population size is set to 40. $w = 0.82$, $a1 = 1.1$, $a2 = 1.1$, which averagely converge in 200 steps.

[**Experiment**] Since there are seven variables to decide the performance of the PSO function, it is not possible to run through all the possible combinations of each parameter. One of the methods is to independently choose random values for each parameter in a reasonable range, then look at the fitness after 5000 moves (*time_steps* = 5001) of the Particles for this set of parameters. Repeat above step several times and observe how much time it converges under different parameter settings.

Termination criterion is strictly set to *best_fitness* < 1e-7 for both functions which stands for zero distance to the target since the optimization goal is to approach the optimum approximation. What's more, computer also has limited precision when dealing with numbers during the calculation process.

The experiment process iterated 50 times by **test**(). As *dim* = 50, **Sphere**'s performance is sensitive to the population size. It failed to converge for 40 times within 50 iterations when *population_size* = 20 but converge for all iterations when *population _size* = 40 [See Table 1]. **Rastrigin** Function is much more complex. Large increase on N only pushes a tiny improvement on the performance. To ensure the effectiveness of **Rastrigin** function, *population_size* is fixed to 120. The algorithm converged in all 50/50 cases when *dim* = 4 [See Table 2] but failed in 4/50 cases when *dim* = 5.

Problem 1.2

[**Result**] **Sphere** function performance is sensitive to *population_size* and **Rastrigin** function is sensitive to the *dimension*.

[**Experiment**] The goal of the optimization is to solve as high dimension with as few particles as it can. The experiment looks at the average converge time and converge rate when varying the value of *dim* and *population_size*. For **Sphere**, we look at $dim \in [30,70]$ and $population_{size} \in [40,20]$ [See Table 3]; For **Rastrigin**, we look at $dim \in [30,70]$ and $population_{size} \in [40,20]$[See Table 4]. The result is carried out by **test1_2_sph**() and **test1_2_ras**().

Problem 1.3

[**Modification 1**] Made the inertia w to be dynamic that depends on previous values.

$$w = w_{end} + (w_{start} - w_{end}) * \frac{(total\_Steps - current\_Step)}{total\_Steps}$$

[**Modification 2**] Added the constriction factor K to improve exploitation which applies to particle velocities [1].

$$K = \frac{2}{|2 - a - \sqrt{a^2 - 4 * a}|}$$

[**Experiment**] Compared 50 pairs of two optimization algorithms by **compare**() [See Fig1]. For **Rastrigin** (*dim* = 50, *population_size* = 40), **DynamicPSO** is averagely 366.14 steps faster to converge than **PSO** algorithm. **DynamicPSO** wins 49(within 50) times and has averagely converge in 618.45 steps while original one converges in 984.59 steps. Same for **Sphere** (*dim* = 4, *population _size* = 120), mean converge time for **DynamicPSO** = 337.82 while time for **PSO** = 472.46. These gives evidence that above modifications effectively improve the algorithm. Stabilization is also under consideration. Similar test in Problem1.1 has done to **DynamicPSO**(). Under current parameter settings, both functions are not likely to diverge since all test results converged. [See Table 5 and 6]

For Sphere: w_s = 0.86, w_e = 0.82, a1 = 1.2, a2 = 0.9

For Rastrigin: w_s = 0.86, w_e = 0.82, a1 = a2 = 2.04.

Problem 2: GP

[**Result**] For **Sphere**:

```
add
    pow_2
        sub
            0
            y
    pow_2
        x
```

For **Rastrigin**: Function is hard to optimize since it is a tree with height of 9 which has too much probable combinations of functions. It failed to converge under any approxi-mation with proofs in next section. Example divergent result under a reasonable parameter setting which give a fitness of 0.13 is provided in Appendix. [See Fig4]

[**Experiment**] **test_gp_sph**() runs genetic programming on **Sphere** for 100 times. Table of 50 times test are given in appendix for brief reading [See Table 7]. The result is a matrix of parameter setting with the NO. of best-generation found. For this function I import *sklearn.linear_model* to investigate whether there is a linear relationship between each parameter settings with the best-generation for best fitness value found rather than the fitness value itself as in all cases it can easily converge [See Fig2]. Crossover rate has very significantly large coefficient (435.919), so I tend to set XI_RATE as large as possible(0.9). Per-node mutation probability has significantly small coefficient (-250.946) to the result. So smaller PROB_MUTATION (0.2) is preferred.

```
POP_SIZE = 120                          # population size
MIN_DEPTH = 2                           # min initial random tree depth
MAX_DEPTH = 3                           # max initial random tree depth
GENERATIONS = 500 # max num of generations to run evolution
TOURNAMENT_SIZE = 5 # size of tournament for tourn-selection
XO_RATE = 0.9                           # crossover rate
PROB_MUTATION = 0.2                     # per-node mutation probability
```

Other parameter setting seems not too dependent to the convergence property of the algorithm. They are set randomly in a reasonable range.

test_gp_ras() runs genetic programming on **Rastrigin** for 20 times [See Table 8]. It takes very long time on **Rastrigin** and not likely to converge. In this situation, we need to look at the relationship between parameters with the fitness value of best fit generation to adjust parameter values of Genetic Programming on **Rastrigin** function. As the table shown, parameter setting is almost independent to the convergence property of the algorithm [See Fig3].

```
POP_SIZE = 80                           # population size
MIN_DEPTH = 2                           # min initial random tree depth
MAX_DEPTH = 6                           # max initial random tree depth
GENERATIONS = 400 # max num of generations to run evolution
TOURNAMENT_SIZE = 6 # size of tournament for tourn-selection
XO_RATE = 0.9                           # crossover rate
PROB_MUTATION = 0.4                     # per-node mutation probability
```

Hence, the conclusion is, the convergency of GP on **Rastrigin** program cannot be guaranteed and not likely to converge. So, they are set randomly in a reasonable range in result.

In the implementation, trees which are higher that *max_h* has been deleted to reduce computational complexity. In order to be convenient to get the best result tree, hall-of-fame parameters are set (changeable size).

Additionally, functions are splat into FUNCTIONS1 and FUNCTIONS2, which are unary operators and binary operators respectively. For FUNCTIONS1, only the left subtree is generated when initializing, and we only compute the value of the left subtree. Blank is left in right subtree when printing tree unary functions.

Problem 3: PSO + GP

[**Result**] Smaller average converge step for combined algorithm **GPSO** compared to **PSODynamic**. Longer runtime for **GPSO** compared to **GP.**

The original **PSO** particle's velocity has been replaced by expression trees. The other operations such as position setting, crossover, mutation, etc. are kept the same as classical **GP** process. This idea is based on *Geometric PSO + GP =Particle Swarm Programming* by Julian Togelius et al [2]. In this work, a new type of crossover is introduced, which is named "weighted homologous crossover" in Julian's paper. It is implemented by the function **homo_crossover2**(). The binary function is function takes 2 inputs values then outputs 1 value and unary function takes 1 input and gives 1 output.

The parameters are parent1, parent2, a1 and a2.

1. If parent1 and parent2 are both binary functions, we randomly choose a node among a set of nodes based on the weight.
2. If parent1 is a binary function and parent2 is a unary function, parent1 is chosen. If parent2 is unary function and parent1 is a binary function, parent2 should be chosen.
3. Otherwise, we randomly choose a subtree of the two nodes.

We update the position by a randomized on convex combination.

$$x_i = CX((x_i, \omega), (\hat{g}, \varphi_1), (\hat{x_1}, \varphi_2))$$

Normalized structural hamming distance (SHD) is a distance measurement tool on GP expression tree's topology. Applying the definition of the SHD, the distance relationship between the parents and the offspring is:

$$SHD(p1, offspring) \leq SHD(p1, p2)$$
$$SHD(p2, off) \leq SHD(parent1, parent2)$$

So the algorithm can be guaranteed at least not to diverge because of the cross over process. Recover that the particle has no velocity, so the position will be updated in **particle_newpos**() by applying **homo_crossover2**() three times to calculate the location of local optimum, global optimum, and the location of its own, with the weights set manually.

[**Experiment**]

As same as previous testing strategy, a random-parameter-chosen test by function **test_gpso**() is carried out. It generate dataset based on **Sphere** by **generate_dataset**(). For each parameter used by the algorithm, the test assigns a random value within a prefix reasonable range. Then we run **GPSO**() and record at how many steps the algorithm needs to find the correct expression – converge at the global optimum. Above processes repeat 50 times.

From the result, it can be generally concluded that other settings except POP_SIZE has no strong relationship to the fatal error that function failed to converge [See Table 9]. Parameter settings with larger POP_SIZE is more likely to diverge. According to these findings, the next comparison test is that to compare the run-steps between **GPSO** and **PSODynamic** and run-times between **GPSO** and **GP**.

POP_SIZE = **70**	*# population size*
MIN_DEPTH = **2**	*# min initial random tree depth*
MAX_DEPTH = **3**	*# max initial random tree depth*
a_gpso = **0.6**	*# inertia compliments for GPSO*
a_pso = 4.08	*# a_pso = a1 + a2 for PSODynamic*
TOURNAMENT_SIZE = **5**	*# size of tournament for tourn-selection*
PROB_MUTATION = **0.2**	*# per-node mutation probability*
Dataset_generate = **30**	# Number of data generated

Test runs for 50 turns and each turn run through 3 algorithms once. GPSO algorithm win 48 times within 50 tests. For GPSO, the fitness finds its global optimum in averagely 209.64 time_steps while **PSODynamic** use 440.8 time_steps. The combined GPSO algorithm is 231.16 steps quicker. Stabilization of GPSO need to be noticed. Comparing the number of converged **GPSO** results to the number of converged **PSODynamic** results, 96% of the test passed for **GPSO** but **PSODynamic** can maintain 100% pass rate. The runtime is not ideal as well. Average runtime for **GPSO** is 45.98s and for **GP** is 3.36s. As one of a factor to evaluate algorithm performance, GPSO need more improvements.

Problem 4: Symbolic regression

[**Result**]

```
add
    pow2
        x
    pow2
        y
```

As My function is

```
def myfunc(x,dim):
    return sum((2* _x ** 2 + 10) for _x in x[:dim])
```

[**Experiment**] The changes made to Problem 4 compared to Problem 2 is the additional set of functions and terminal set.
The parameter settings are shown below:

```
POP_SIZE = 120                          # population size
MIN_DEPTH = 2                   # min initial random tree depth
MAX_DEPTH = 3                   # max initial random tree depth
GENERATIONS = 300 # max num of generations to run evolution
TOURNAMENT_SIZE = 7 # size of tournament for tourn-selection
XO_RATE = 0.8                           # crossover rate
PROB_MUTATION = 0.2         # per-node mutation probability
```

I change the dataset of Sphere into txt form and read it as the dataset file for late use and data consistency. **protected_div**() is added to the function. A small number 1e-5 will add to the denominator to ensure that the arithmetic exceptation won't occur because of the division by zero error. Other than **pow2**(), higher power functions are added to the function set such as **pow3**() and **pow4**(). Root functions are also added such as **pow12**(x) which is the square root of x. They are classified as 2 types of functions: power-up functions **pow_type1**() and root functions **pow_type2**() in order to decrease the complexity of expression tree node choosing. Because the operations in a same class have same scaling effects even with different magnitude. Need to be notice that, root functions suffers from invalid input such that negative x will get a complex number which is not acceptable, so absolute value of x is transferred into the function.

New exponential function **exp**() suffers from value overflow. The overflow error will be caught in this implementation and return 'inf' of type 'float' to avoid accidentally SystemExit.

Appendix

Converge	Converge Time	Dim	w	a1	a2	Population	Time_step	Range
True	868.0	50	0.81	1.3	1.1	30	5001	5.12
True	875.0	50	0.82	1.2	1.0	30	5001	5.12
True	896.0	50	0.8	1.3	1.1	30	5001	5.12
True	920.0	50	0.82	1.0	1.0	30	5001	5.12
True	934.0	50	0.8	1.1	1.0	30	5001	5.12
True	971.0	50	0.81	1.2	1.0	30	5001	5.12
True	1025.0	50	0.82	1.0	1.1	30	5001	5.12
True	1034.0	50	0.83	1.1	0.9	30	5001	5.12
True	1036.0	50	0.81	1.2	1.2	30	5001	5.12
True	1039.0	50	0.8	1.2	1.0	30	5001	5.12
True	1080.0	50	0.8	1.0	1.0	30	5001	5.12
True	1144.0	50	0.8	1.0	1.2	30	5001	5.12
True	1153.0	50	0.8	1.0	1.2	30	5001	5.12
True	1163.0	50	0.8	1.2	1.1	30	5001	5.12
True	1164.0	50	0.82	1.2	1.0	30	5001	5.12
True	1192.0	50	0.81	1.3	1.1	30	5001	5.12
True	1224.0	50	0.8	1.0	1.0	30	5001	5.12
True	1226.0	50	0.82	1.2	1.0	30	5001	5.12
True	1229.0	50	0.8	1.0	1.0	30	5001	5.12
True	1255.0	50	0.81	1.0	1.1	30	5001	5.12
True	1261.0	50	0.81	1.3	1.0	30	5001	5.12
True	1272.0	50	0.8	1.1	1.0	30	5001	5.12
True	1280.0	50	0.8	1.0	1.0	30	5001	5.12
True	1299.0	50	0.8	1.0	1.1	30	5001	5.12
True	1353.0	50	0.83	1.0	1.1	30	5001	5.12
True	1380.0	50	0.83	1.2	1.0	30	5001	5.12
True	1390.0	50	0.81	0.9	1.1	30	5001	5.12
True	1398.0	50	0.83	1.3	1.1	30	5001	5.12
True	1437.0	50	0.81	1.2	1.2	30	5001	5.12
True	1439.0	50	0.81	1.2	1.2	30	5001	5.12
True	1454.0	50	0.83	1.0	1.2	30	5001	5.12
True	1460.0	50	0.81	1.2	0.9	30	5001	5.12
True	1490.0	50	0.83	1.1	1.2	30	5001	5.12
True	1493.0	50	0.8	1.0	1.1	30	5001	5.12
True	1502.0	50	0.81	1.1	1.2	30	5001	5.12
True	1521.0	50	0.8	1.1	1.1	30	5001	5.12
True	1536.0	50	0.81	1.0	1.3	30	5001	5.12
True	1540.0	50	0.82	1.0	1.2	30	5001	5.12
True	1546.0	50	0.83	1.2	1.2	30	5001	5.12
True	1555.0	50	0.83	1.0	1.3	30	5001	5.12
True	1604.0	50	0.83	1.2	1.1	30	5001	5.12
True	1616.0	50	0.81	1.0	1.1	30	5001	5.12
True	1714.0	50	0.81	1.0	1.2	30	5001	5.12
True	1917.0	50	0.81	1.1	0.9	30	5001	5.12
True	1923.0	50	0.83	1.0	1.2	30	5001	5.12
True	1978.0	50	0.83	1.0	1.3	30	5001	5.12
True	1988.0	50	0.8	1.1	1.2	30	5001	5.12
True	1994.0	50	0.81	1.0	1.0	30	5001	5.12
True	2027.0	50	0.8	1.0	1.2	30	5001	5.12
True	2429.0	50	0.83	1.0	1.0	30	5001	5.12

Table 1: Sphere dim=50, pop=30

Converge	Converge Time	Dim	w	a1	a2	Population	Time_step	Range
True	120.0	4	0.8	1.0	1.1	120	5001	5.12
True	141.0	4	0.8	1.2	0.9	120	5001	5.12
True	144.0	4	0.8	1.0	1.2	120	5001	5.12
True	149.0	4	0.81	1.0	1.0	120	5001	5.12
True	152.0	4	0.8	1.2	1.2	120	5001	5.12
True	153.0	4	0.82	1.0	1.1	120	5001	5.12
True	154.0	4	0.8	1.0	1.0	120	5001	5.12
True	155.0	4	0.81	1.0	1.1	120	5001	5.12
True	156.0	4	0.81	1.2	1.0	120	5001	5.12
True	161.0	4	0.81	1.0	1.0	120	5001	5.12
True	161.0	4	0.81	1.0	1.2	120	5001	5.12
True	166.0	4	0.81	1.0	1.2	120	5001	5.12
True	168.0	4	0.81	1.2	0.9	120	5001	5.12
True	170.0	4	0.8	1.0	1.0	120	5001	5.12
True	172.0	4	0.83	1.1	0.9	120	5001	5.12
True	176.0	4	0.81	1.0	1.0	120	5001	5.12
True	177.0	4	0.82	1.0	1.2	120	5001	5.12
True	178.0	4	0.82	1.2	1.2	120	5001	5.12
True	178.0	4	0.83	1.3	1.1	120	5001	5.12
True	180.0	4	0.83	1.1	1.0	120	5001	5.12
True	181.0	4	0.81	1.2	0.9	120	5001	5.12
True	182.0	4	0.82	1.0	1.2	120	5001	5.12
True	183.0	4	0.82	1.0	1.0	120	5001	5.12
True	183.0	4	0.82	1.1	0.9	120	5001	5.12
True	187.0	4	0.8	1.1	1.1	120	5001	5.12
True	189.0	4	0.83	1.0	1.0	120	5001	5.12
True	190.0	4	0.82	1.0	1.3	120	5001	5.12
True	196.0	4	0.82	1.0	1.1	120	5001	5.12
True	197.0	4	0.82	1.1	1.2	120	5001	5.12
True	201.0	4	0.8	1.0	1.1	120	5001	5.12
True	202.0	4	0.83	1.1	1.0	120	5001	5.12
True	202.0	4	0.83	1.3	1.1	120	5001	5.12
True	203.0	4	0.81	1.3	1.1	120	5001	5.12
True	204.0	4	0.83	1.0	1.0	120	5001	5.12
True	206.0	4	0.81	1.1	1.0	120	5001	5.12
True	207.0	4	0.83	1.0	1.2	120	5001	5.12
True	208.0	4	0.82	1.1	1.1	120	5001	5.12
True	211.0	4	0.83	1.2	1.2	120	5001	5.12
True	212.0	4	0.83	1.2	1.2	120	5001	5.12
True	214.0	4	0.83	1.0	1.2	120	5001	5.12
True	216.0	4	0.82	1.1	1.3	120	5001	5.12
True	217.0	4	0.8	1.1	1.3	120	5001	5.12
True	217.0	4	0.82	1.3	1.0	120	5001	5.12
True	222.0	4	0.8	1.1	1.3	120	5001	5.12
True	225.0	4	0.83	1.1	1.3	120	5001	5.12
True	226.0	4	0.83	1.0	1.3	120	5001	5.12
True	245.0	4	0.82	1.2	1.1	120	5001	5.12
True	247.0	4	0.81	1.2	1.1	120	5001	5.12
True	262.0	4	0.83	1.0	1.2	120	5001	5.12
True	302.0	4	0.81	1.1	1.1	120	5001	5.12

Table 2: Rastrigin dim=4, pop=120

Dim	Population	Average Converge Time	Converge Rate
30.0	20.0	746.52	1.0
30.0	30.0	480.66	1.0
30.0	40.0	403.56	1.0
30.0	50.0	377.88	1.0
40.0	20.0	1505.5	1.0
40.0	30.0	788.34	1.0
40.0	40.0	642.12	1.0
40.0	50.0	524.36	1.0
50.0	20.0	2448.96	1.0
50.0	30.0	1280.24	1.0
50.0	40.0	963.36	1.0
50.0	50.0	780.04	1.0
60.0	20.0	3319.02	0.84
60.0	30.0	2234.22	1.0
60.0	40.0	1431.54	1.0
60.0	50.0	1188.14	1.0
70.0	20.0	4452.78	0.18
70.0	30.0	3323.61	0.92
70.0	40.0	2252.02	1.0
70.0	50.0	1671.12	1.0

Table 3: Sphere parameter-performance Table

Dim	Population	Average Converge Time	Converge Rate
3.0	40.0	172.19	0.94
3.0	80.0	142.34	1.0
3.0	120.0	127.9	1.0
3.0	160.0	123.98	1.0
4.0	40.0	264.36	0.88
4.0	80.0	217.4	0.9
4.0	120.0	189.64	1.0
4.0	160.0	160.48	1.0

Table 4: Rastrigin parameter-performance Table

Converge	Converge Time	Dim	w_end	a1	a2	Population	Time_step	Range
True	1591.0	50	0.8	2.01	2.01	40	5001	5.12
True	1759.0	50	0.8	2.01	2.01	40	5001	5.12
True	1836.0	50	0.8	2.01	2.01	40	5001	5.12
True	1906.0	50	0.81	2.01	2.01	40	5001	5.12
True	2017.0	50	0.82	2.01	2.01	40	5001	5.12
True	2032.0	50	0.82	2.01	2.01	40	5001	5.12
True	2037.0	50	0.81	2.01	2.01	40	5001	5.12
True	2075.0	50	0.81	2.01	2.01	40	5001	5.12
True	2082.0	50	0.81	2.01	2.01	40	5001	5.12
True	2100.0	50	0.82	2.01	2.01	40	5001	5.12
True	2153.0	50	0.82	2.01	2.01	40	5001	5.12
True	2155.0	50	0.82	2.01	2.01	40	5001	5.12
True	2207.0	50	0.82	2.01	2.01	40	5001	5.12
True	2237.0	50	0.82	2.01	2.01	40	5001	5.12
True	2330.0	50	0.82	2.01	2.01	40	5001	5.12
True	2336.0	50	0.82	2.01	2.01	40	5001	5.12
True	2352.0	50	0.82	2.01	2.01	40	5001	5.12
True	2485.0	50	0.83	2.01	2.01	40	5001	5.12
True	2529.0	50	0.83	2.01	2.01	40	5001	5.12
True	2545.0	50	0.83	2.01	2.01	40	5001	5.12
True	2594.0	50	0.83	2.01	2.01	40	5001	5.12
True	2632.0	50	0.83	2.01	2.01	40	5001	5.12
True	2669.0	50	0.84	2.01	2.01	40	5001	5.12
True	2755.0	50	0.83	2.01	2.01	40	5001	5.12
True	2773.0	50	0.83	2.01	2.01	40	5001	5.12
True	2889.0	50	0.84	2.01	2.01	40	5001	5.12
True	2900.0	50	0.84	2.01	2.01	40	5001	5.12
True	2933.0	50	0.84	2.01	2.01	40	5001	5.12
True	2952.0	50	0.83	2.01	2.01	40	5001	5.12
True	2983.0	50	0.83	2.01	2.01	40	5001	5.12
True	2992.0	50	0.84	2.01	2.01	40	5001	5.12
True	3001.0	50	0.84	2.01	2.01	40	5001	5.12
True	3090.0	50	0.83	2.01	2.01	40	5001	5.12
True	3146.0	50	0.84	2.01	2.01	40	5001	5.12
True	3149.0	50	0.84	2.01	2.01	40	5001	5.12
True	3165.0	50	0.85	2.01	2.01	40	5001	5.12
True	3170.0	50	0.84	2.01	2.01	40	5001	5.12
True	3251.0	50	0.84	2.01	2.01	40	5001	5.12
True	3355.0	50	0.84	2.01	2.01	40	5001	5.12
True	3357.0	50	0.85	2.01	2.01	40	5001	5.12
True	3476.0	50	0.84	2.01	2.01	40	5001	5.12
True	3695.0	50	0.85	2.01	2.01	40	5001	5.12
True	3741.0	50	0.85	2.01	2.01	40	5001	5.12
True	3834.0	50	0.85	2.01	2.01	40	5001	5.12
True	3873.0	50	0.85	2.01	2.01	40	5001	5.12
True	4096.0	50	0.85	2.01	2.01	40	5001	5.12
True	4289.0	50	0.85	2.01	2.01	40	5001	5.12
True	4340.0	50	0.85	2.01	2.01	40	5001	5.12
True	4556.0	50	0.85	2.01	2.01	40	5001	5.12
True	4908.0	50	0.85	2.01	2.01	40	5001	5.12

Table 5: Sphere DynamicPSO test all passed

Converge	Converge Time	Dim	w_end	a1	a2	Population	Time_step	Range
True	144.0	4	0.81	2.01	2.01	120	5001	5.12
True	145.0	4	0.8	2.01	2.01	120	5001	5.12
True	151.0	4	0.8	2.01	2.01	120	5001	5.12
True	164.0	4	0.8	2.01	2.01	120	5001	5.12
True	168.0	4	0.8	2.01	2.01	120	5001	5.12
True	173.0	4	0.84	2.01	2.01	120	5001	5.12
True	181.0	4	0.81	2.01	2.01	120	5001	5.12
True	186.0	4	0.82	2.01	2.01	120	5001	5.12
True	188.0	4	0.83	2.01	2.01	120	5001	5.12
True	190.0	4	0.81	2.01	2.01	120	5001	5.12
True	190.0	4	0.84	2.01	2.01	120	5001	5.12
True	191.0	4	0.83	2.01	2.01	120	5001	5.12
True	192.0	4	0.81	2.01	2.01	120	5001	5.12
True	194.0	4	0.8	2.01	2.01	120	5001	5.12
True	194.0	4	0.8	2.01	2.01	120	5001	5.12
True	195.0	4	0.83	2.01	2.01	120	5001	5.12
True	197.0	4	0.8	2.01	2.01	120	5001	5.12
True	198.0	4	0.84	2.01	2.01	120	5001	5.12
True	199.0	4	0.8	2.01	2.01	120	5001	5.12
True	200.0	4	0.85	2.01	2.01	120	5001	5.12
True	202.0	4	0.81	2.01	2.01	120	5001	5.12
True	203.0	4	0.84	2.01	2.01	120	5001	5.12
True	204.0	4	0.82	2.01	2.01	120	5001	5.12
True	210.0	4	0.82	2.01	2.01	120	5001	5.12
True	220.0	4	0.82	2.01	2.01	120	5001	5.12
True	220.0	4	0.82	2.01	2.01	120	5001	5.12
True	221.0	4	0.82	2.01	2.01	120	5001	5.12
True	221.0	4	0.85	2.01	2.01	120	5001	5.12
True	222.0	4	0.83	2.01	2.01	120	5001	5.12
True	222.0	4	0.85	2.01	2.01	120	5001	5.12
True	224.0	4	0.83	2.01	2.01	120	5001	5.12
True	224.0	4	0.84	2.01	2.01	120	5001	5.12
True	227.0	4	0.82	2.01	2.01	120	5001	5.12
True	229.0	4	0.83	2.01	2.01	120	5001	5.12
True	234.0	4	0.82	2.01	2.01	120	5001	5.12
True	234.0	4	0.83	2.01	2.01	120	5001	5.12
True	235.0	4	0.81	2.01	2.01	120	5001	5.12
True	235.0	4	0.83	2.01	2.01	120	5001	5.12
True	240.0	4	0.85	2.01	2.01	120	5001	5.12
True	241.0	4	0.8	2.01	2.01	120	5001	5.12
True	242.0	4	0.81	2.01	2.01	120	5001	5.12
True	243.0	4	0.84	2.01	2.01	120	5001	5.12
True	247.0	4	0.83	2.01	2.01	120	5001	5.12
True	257.0	4	0.83	2.01	2.01	120	5001	5.12
True	259.0	4	0.85	2.01	2.01	120	5001	5.12
True	261.0	4	0.84	2.01	2.01	120	5001	5.12
True	276.0	4	0.84	2.01	2.01	120	5001	5.12
True	280.0	4	0.84	2.01	2.01	120	5001	5.12
True	284.0	4	0.82	2.01	2.01	120	5001	5.12
True	291.0	4	0.85	2.01	2.01	120	5001	5.12

Table 6: Rastrigin DynamicPSO all pass

Data	POP_SIZE	MIN_D	MAX_D	GENERATIONS	TOUR_SIZE	XO	PROB	B_fitness	B_gen
50	100	2	6	250	5	0.7	0.2	1.0	0
50	100	2	6	200	5	0.75	0.3	1.0	1
50	130	2	6	200	4	0.7	0.15	1.0	1
50	100	2	6	200	5	0.75	0.3	1.0	2
50	110	2	6	100	5	0.7	0.2	1.0	2
50	110	2	6	150	5	0.8	0.25	1.0	2
50	120	2	6	100	5	0.7	0.2	1.0	2
50	120	2	6	250	3	0.8	0.25	1.0	2
50	100	2	6	100	4	0.8	0.2	1.0	3
50	100	2	6	150	4	0.75	0.2	1.0	3
50	100	2	6	150	5	0.75	0.15	1.0	3
50	140	2	6	150	3	0.7	0.15	1.0	3
50	100	2	6	250	3	0.7	0.25	1.0	4
50	100	2	6	250	5	0.7	0.25	1.0	4
50	110	2	6	150	4	0.75	0.2	1.0	4
50	120	2	6	100	5	0.7	0.15	1.0	4
50	120	2	6	150	5	0.8	0.2	1.0	4
50	140	2	6	100	4	0.75	0.25	1.0	4
50	140	2	6	100	5	0.8	0.2	1.0	4
50	100	2	6	200	5	0.8	0.15	1.0	5
50	110	2	6	100	5	0.75	0.3	1.0	5
50	120	2	6	100	5	0.7	0.25	1.0	5
50	120	2	6	150	3	0.7	0.2	1.0	5
50	140	2	6	150	3	0.7	0.15	1.0	6
50	120	2	6	150	4	0.7	0.15	1.0	7
50	120	2	6	150	5	0.8	0.3	1.0	7
50	100	2	6	100	3	0.7	0.15	1.0	8
50	100	2	6	100	4	0.75	0.2	1.0	8
50	110	2	6	150	3	0.75	0.3	1.0	8
50	130	2	6	100	3	0.8	0.3	1.0	8
50	130	2	6	150	3	0.8	0.25	1.0	10
50	140	2	6	250	3	0.7	0.2	1.0	10
50	100	2	6	100	3	0.8	0.25	1.0	11
50	140	2	6	250	5	0.75	0.15	1.0	11
50	100	2	6	200	5	0.7	0.2	1.0	13
50	120	2	6	100	5	0.8	0.15	1.0	14
50	120	2	6	100	4	0.8	0.2	1.0	16
50	110	2	6	150	5	0.75	0.2	1.0	18
50	120	2	6	100	5	0.8	0.25	1.0	18
50	120	2	6	200	5	0.8	0.15	1.0	18
50	130	2	6	100	5	0.75	0.15	1.0	18
50	120	2	6	250	4	0.8	0.2	1.0	22
50	140	2	6	200	3	0.8	0.3	1.0	25
50	130	2	6	100	3	0.75	0.3	1.0	26
50	140	2	6	250	3	0.7	0.15	1.0	36
50	100	2	6	100	3	0.8	0.2	1.0	38
50	140	2	6	250	3	0.8	0.2	1.0	44
50	130	2	6	150	5	0.75	0.3	1.0	56
50	130	2	6	250	3	0.75	0.25	1.0	69
50	110	2	6	250	4	0.8	0.15	1.0	478

Table 7: 50 tests to the parameter values on Sphere

Data	POP_SIZE	MIN_D	MAX_D	GENERATIONS	TOUR_SIZE	XO	PROB	B_fitness	B_gen
50	50	2	5	200	7	0.9	0.25	0.14	266
50	80	2	5	300	7	0.7	0.15	0.12	308
50	70	2	5	200	7	0.95	0.15	0.13	317
50	80	2	5	300	7	0.7	0.15	0.12	323
50	70	2	5	200	5	0.75	0.3	0.12	325
50	80	2	5	300	5	0.85	0.25	0.13	396
50	70	2	5	200	7	0.7	0.25	0.2	403
50	80	2	5	200	7	0.85	0.2	0.13	431
50	80	2	5	300	6	0.8	0.2	0.31	443
50	80	2	5	240	5	0.95	0.15	0.13	447
50	50	2	5	300	5	0.75	0.25	0.11	463
50	50	2	5	220	5	0.8	0.3	0.18	466
50	70	2	5	280	6	0.9	0.3	0.17	478
50	80	2	5	200	5	0.7	0.2	0.12	484
50	70	2	5	220	4	0.8	0.15	0.14	491
50	70	2	5	260	7	0.9	0.25	0.16	492
50	50	2	5	240	4	0.95	0.3	0.12	493
50	80	2	5	280	6	0.75	0.25	0.13	494
50	60	2	5	300	5	0.9	0.2	0.14	497
50	80	2	5	240	4	0.75	0.25	0.15	499

Table 8: 20 tests to the parameter values on Rastrigin

Convergence	Converge Time	a	POP_SIZE	PROB_MUTATION	Best_fitness
False	5001	0.1	90	0.2	0.11408227870233723
False	5001	0.2	110	0.3	0.11880946143523102
False	5001	0.5	120	0.4	0.12616348968914481
False	5001	0.1	130	0.3	0.2953244503802418
False	5001	0.7	140	0.3	0.5104842555345649
True	10	0.6	130	0.3	0.9999999999999987
True	10	0.7	100	0.4	0.9999999999999996
True	10	0.1	60	0.5	1.0
True	10	0.1	70	0.4	1.0
True	10	0.1	80	0.1	1.0
True	10	0.1	80	0.4	1.0
True	10	0.1	120	0.2	1.0
True	10	0.2	70	0.2	1.0
True	10	0.2	100	0.1	1.0
True	10	0.3	60	0.2	1.0
True	10	0.3	60	0.4	1.0
True	10	0.3	60	0.4	1.0
True	10	0.3	80	0.1	1.0
True	10	0.3	80	0.3	1.0
True	10	0.3	130	0.2	1.0
True	10	0.3	140	0.5	1.0
True	10	0.4	80	0.3	1.0
True	10	0.4	80	0.5	1.0
True	10	0.4	130	0.5	1.0
True	10	0.5	60	0.1	1.0
True	10	0.5	60	0.5	1.0
True	10	0.5	100	0.5	1.0
True	10	0.5	120	0.1	1.0
True	10	0.5	120	0.3	1.0
True	10	0.5	140	0.4	1.0
True	10	0.6	60	0.1	1.0
True	10	0.6	60	0.4	1.0
True	10	0.6	60	0.5	1.0
True	10	0.6	70	0.5	1.0
True	10	0.6	80	0.2	1.0
True	10	0.6	80	0.4	1.0
True	10	0.6	110	0.5	1.0
True	10	0.6	120	0.3	1.0
True	10	0.6	120	0.3	1.0
True	10	0.6	120	0.5	1.0
True	10	0.6	140	0.5	1.0
True	10	0.6	140	0.5	1.0
True	10	0.7	70	0.3	1.0
True	10	0.7	100	0.3	1.0
True	10	0.7	100	0.4	1.0
True	10	0.7	110	0.3	1.0
True	10	0.7	120	0.4	1.0
True	10	0.7	130	0.2	1.0
True	10	0.8	80	0.1	1.0
True	10	0.8	80	0.5	1.0

Table 9: GPSO 50 random tests

```
In [464]: 1 compare(sphere,50,40,50)

Dynamic PSO algorithm win * 49 * times/ 50 times
 Average Converge time for DynamicPSO is  618.45
 Average Converge time for PSO is  984.59
 which is  366.14 steps quicker!

In [465]: 1 compare(rastrigin,4,120,50)

Dynamic PSO algorithm win * 50 * times/ 50 times
 Average Converge time for DynamicPSO is  337.82
 Average Converge time for PSO is  472.46
 which is  134.64 steps quicker!
```

Fig1: Evaluate the behaviours of DynamicPSO and PSO

coef_Data	coe_POPSIZE	cMIN_D	cMAX_D	coef_GENS	coef_TOURSIZE	coef_XO	coef_PROB
0.0	0.001	0.0	-0.0	0.0	0.007	0.016	0.185

Fig2: Coefficient on GP parameter for Sphere function

coef_Data	coe_POPSIZE	cMIN_D	cMAX_D	coef_GENS	coef_TOURSIZE	coef_XO	coef_PROB	coef_B_f
0.0	1.161	0.0	0.0	0.243	-40.016	115.26	236.256	329.086

Fig3: Coefficient on GP parameter for Rastrigin function

```
add
   sub
      pow_2
         y
      add
         1
         10
   add
      mul
         sin
            mul
               sin
                  cos
                     10
                  mul
                     sin
                        mul
                           sin
                              cos
                                 10
                           mul
                              pi
                              pow_2
                                 x
                        mul
                           pi
                           2
         mul
            pi
            2
      add
         mul
            10
            pi
         pow_2
            x
```

Fig4: Best_of_run of RASTRIGIN attained at gen391, has f=0.116

Bibliography

[1] S. Y. M. M. a. H. N. Lim, A constriction factor based particle swarm optimization for economic dispatch., Bristol, 2009.

[2] J. D. N. R. &. M. A. Togelius, Geometric pso+ gp= particle swarm programming., IEEE, Ed., 2008 IEEE Congress on Evolutionary Computation, 2008, pp. pp. 3594-3600.