F21BC : Biologically Inspired Computation

Coursework 1 : Black-Box Optimization

Submitted by :
Azimeh Gharavi & Jules Detrie
ag72@hw.ac.uk / jd57@hw.ac.uk
Data Science Master students
Heriot Watt University, Edinburgh

Date and place :
November 10, 2016
Heriot Watt University, Edinburgh

To the attention of :
Patricia A. Vargas
Marta Vallejo

Contents

Chapter 1

Evolution of MLP with GA

1.1 Introduction

The aim of the experiment is using genetic algorithms to evolve a multi-layer perceptron (MLP) to approximate a set of 24 continuous noiseless functions which can be seen as models for known problems. To evaluate the success of the experiment, we used black-box optimisation framework "COCO".

1.2 Steps to create the algorithm

We wrote the algorithm in Java, used the Java version of COCO to evaluate it because it was our most familiar language, and generated the output graphs with COCO's Python post-processing script.

We elaborated our algorithm inside COCO environment but without using COCO features[1]. This allowed us later to use COCO features easily. We only used the first function of COCO ("Sphere function") in dimension 2. We created our input / output values table and then our MLP.

Each chromosome is a flat vector of weights representing one possible configuration of MLP. We generated initial population randomly and decided to have the same number of neurons for each hidden layer. Its size is fixed because links do not get deleted / added. The MLP is "static" because we do not move neurons but change the weights.

Hence :

$$sizeOfChromosome = dimension * NNPHL + (NHL - 1) * NNPHL^2 + NNPHL \tag{1.1}$$

- NHL : number of hidden layers
- NNPHL : number of neurons per hidden layer

[1] We erased everything that was in the third loop (the one incrementing "idx_instances" variable)

To find that formula, we designed our MLP so :

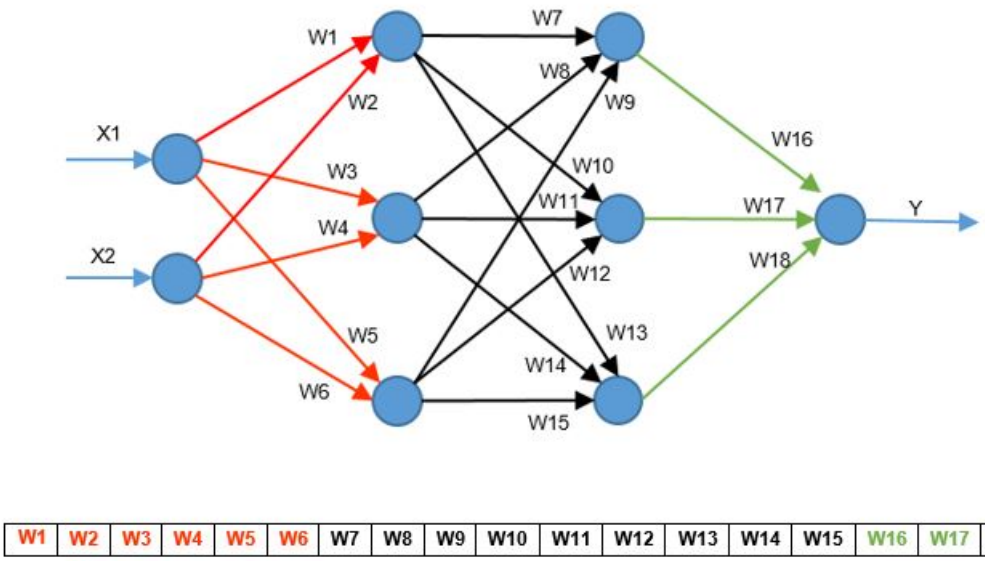

Figure 1.1: MLP with hidden layer

Weights are ordered by layer number and by neuron in the chromosome.

Hyperbolic tangent is the activation function. It performs better than threshold and piece-wise linear functions, and allows to have the output of neurons within [-1;1] as opposed to sigmoid of range [0;1].

We chose a small-size MLP, ran and manually tested it[2] with two inputs[3].

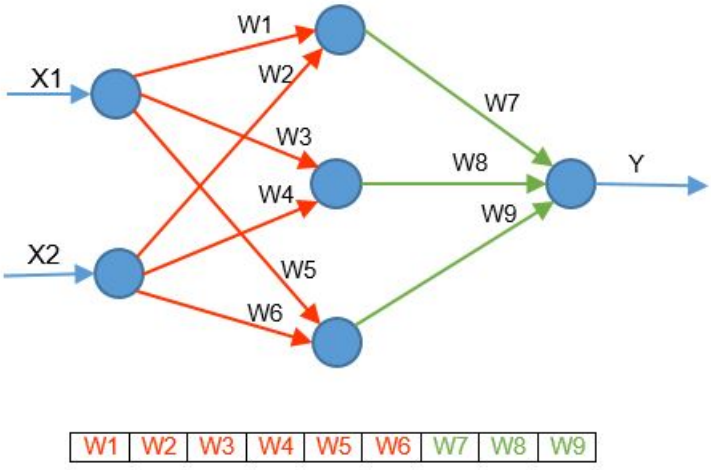

Figure 1.2: MLP without hidden layer

Then, we built the core part of our GA (step 3) and adapted it to use COCO features.

[2] with one hidden layer and then with two hidden layers

[3] because number of inputs equals to dimension, which is 2 here

1.3 Final version of our GA

Step 0 : generating sample input dataset to evolve our MLP

Each parameter of the functions can take multiples of 0.25 (chosen interval) as values within domain [-5,5]. Each input is a dimension-sized vector.

Step 1 : generating initial population

See section 1.2.

Step 2 : calculating output and custom fitness of initial population

For each chromosome we calculate:

- MLP output for each input.
- difference between outputs and real function outputs
- sum of absolute values of these differences

One accumulated error is a numeric representation of a chromosome's performance. The higher it is, the worse this configuration is[4].

During step 3, these values will be compared to determine the chromosome to kill[5].

Finally, we add values previously calculated to create initial population's fitness.

Step 3 : evolving MLP, determining best chromosome

While the population contains at least one chromosome, we do :

1. **Parents selection :** random
2. **Crossover :** single-point[6]
3. **Mutation :** single-gene[7] on both parents
4. **Calculation of modified generation's custom fitness**
5. **Choosing generation to keep :** comparison of custom fitness values of current and modified generations. Best population becomes next generation.
6. **Worst individual removal**[8]
7. **Calculation of new population's custom fitness**

This loop gives us the best chromosome of the population.

N.B : different GA methods are experimented in chapter 2.

[4] because the MLP output is further from real output
[5] our custom fitness function deletes the one with highest error
[6] with crossover point chosen randomly in second half of the chromosome
[7] modification of a randomly chosen gene
[8] see step 2

1.4 Set-up of experiments

We investigated the effect of increasing values of four parameters of MLP on our algorithm's performance.

We fixed our reference values, and changed the value of each parameter twice, one at a time.

In experiment 9 we wanted to test our algorithm with high-value parameters (NIV = 50, NHL = 50, NNPHL = 50, NC = 100). Unfortunately, because of our algorithm's complexity and lack of computer resources, it was impossible to perform, so we used lower values.[9]

	NIV	NHL	NNPHL	NC
Reference	**20**	**5**	**4**	**10**
Experiment 1	30	5	4	10
Experiment 2	50	5	4	10
Experiment 3	20	10	4	10
Experiment 4	20	50	4	10
Experiment 5	20	5	10	10
Experiment 6	20	5	50	10
Experiment 7	20	5	4	50
Experiment 8	20	5	4	100
Experiment 9	20	10	8	30

1. NIV : number of rows in input data set
2. NHL : number of hidden layers
3. NNPHL : number of neurons per hidden layer
4. NC : initial population size

NB : 20 inputs might be little to draw a function accurately. This is why we tested this non-MLP parameter.

1.5 Expectations

We expect that increasing these parameters will have following impact :

- NIV : it gives the algorithm more points to approximate functions accurately
- NHL & NNPHL : we think there could be an optimum MLP size. Since a simple XOR problem requires a two-layer MLP[10], we expect to need a larger MLP to approximate complex functions accurately. Yet, huge difference is not expected between experiments [3,4] and [5,6].
- NC : it gives the algorithm more chance to pass the local optima and reach global optima.

Because population is generated randomly each time MY_OPTIMIZER is called, and because the algorithm converges depending on the initial population, worse results after optimizing the parameters is possible.

Finally, we expect higher run-times by increasing these values.

[9] experiment 9 with parameter values in the table completed in 8 hours
[10] As seen in Lecture 3 & 4 of F21BC

1.6 Results

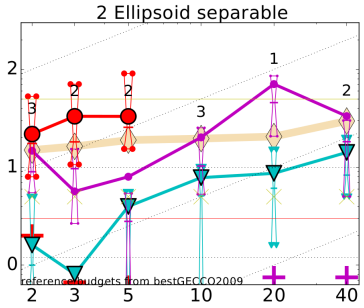

Figure 1.3: F2 Exp. 6

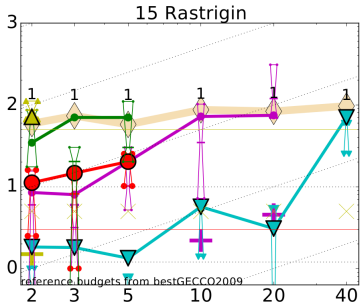

Figure 1.4: F15 Exp. 8

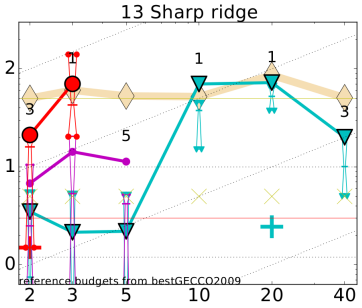

Figure 1.5: F13 Exp. 2

1.6.1 Global remarks

We cannot fully interpret graphs but by comparing them we noticed :

- Our algorithm performs better for some functions. Taking brown line as reference[11], sometimes other lines are close and follow a similar pattern as the reference.
- For most functions, changing parameters does not modify the shape of lines but alters values slightly. However, for function 11 curves change by increasing NNPHL.
- Our algorithm performs better as dimension increases for most functions (counter-example : function 1).
- Sometimes, increasing parameters helps the algorithm converge for specific dimension which could be optimal : increasing NIV (1) or NHL (2)

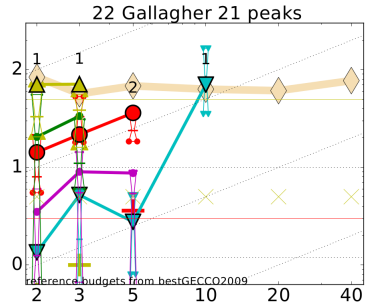

Figure 1.6: F22 Ref

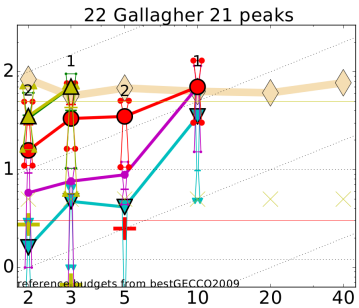

Figure 1.7: F22 Exp. 2 (1)

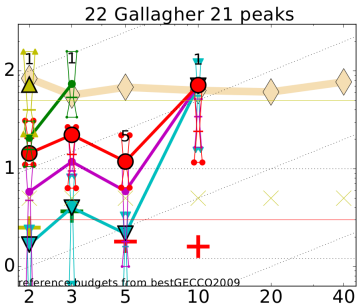

Figure 1.8: F22 Exp. 3 (2)

[11] the brown line represents "the respective best result from BBOB-2009 for the most difficult target"

We could not interpret some graphs :

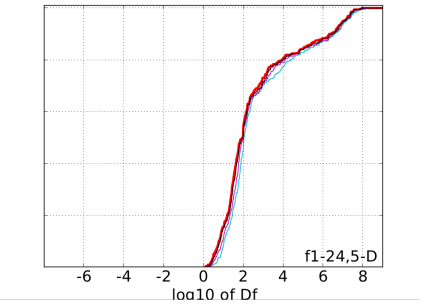

Figure 1.9: Exp. 8 "ppfvdistr_05D_all"

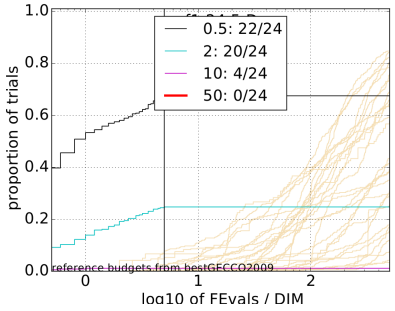

Figure 1.10: Exp. 9 "pprldistr_05D_all"

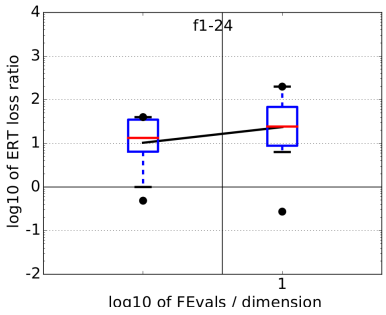

Figure 1.11: Exp. 9 "pplogloss_20D_noiselessall"

Finally, sometimes our algorithm outperformed COCO for some functions[12] :

ERT in number of function evaluations

5-D

#FEs/D	0.5	1.2	3	10	50	#succ
f_1	*2.5e+1*:4.8	*1.6e+1*:7.6	*1.0e-8*:12	*1.0e-8*:12	*1.0e-8*:12	15/15
	7.4 (6)	22 (19)	∞	∞	∞ *25*	0/15
f_2	*1.6e+6*:2.9	*4.0e+5*:11	*4.0e+4*:15	*6.3e+2*:58	*1.0e-8*:95	15/15
	7.0 (13)	3.7 (5)	23 (28)	∞	∞ *25*	0/15
f_3	*1.6e+2*:4.1	*1.0e+2*:15	*6.3e+1*:23	*2.5e+1*:73	*1.0e+1*:716	15/15
	13 (7)	24 (29)	∞	∞	∞ *25*	0/15
f_4	*2.5e+2*:2.6	*1.6e+2*:10	*1.0e+2*:19	*4.0e+1*:65	*1.6e+1*:434	15/15
	0.77 (0.7)	2.3 (3)	9.1 (9)	∞	∞ *25*	0/15
f_5	*6.3e+1*:4.0	*4.0e+1*:10	*1.0e-8*:10	*1.0e-8*:10	*1.0e-8*:10	15/15
	∞	∞	∞	∞	∞ *25*	0/15
f_6	*1.0e+5*:3.0	*2.5e+4*:8.4	*1.0e+2*:16	*2.5e+1*:54	*2.5e-1*:254	15/15
	6.2 (10)	6.7 (11)	23 (55)	∞	∞ *25*	0/15
f_7	*1.6e+2*:4.2	*1.0e+2*:6.2	*2.5e+1*:20	*4.0e+0*:54	*1.0e+0*:324	15/15
	3.5 (5)	4.1 (6)	3.6 (3)	∞	∞ *25*	0/15
f_8	*1.0e+4*:4.6	*6.3e+3*:6.8	*1.0e+3*:18	*6.3e+1*:54	*1.6e+0*:258	15/15
	1.7 (3)	3.3 (5)	9.2 (5)	∞	∞ *25*	0/15
f_9	*2.5e+1*:20	*1.6e+1*:26	*1.0e+1*:35	*4.0e+0*:62	*1.6e-2*:256	15/15
	0.97 (1)	1.8 (1)	4.8 (2)	∞	∞ *25*	0/15
f_{10}	*2.5e+6*:2.9	*6.3e+5*:7.0	*2.5e+5*:17	*6.3e+3*:54	*2.5e+1*:297	15/15
	35 (57)	50 (38)	21 (41)	∞	∞ *25*	0/15
f_{11}	*1.0e+6*:3.0	*6.3e+4*:6.2	*6.3e+2*:16	*6.3e+1*:74	*6.3e-1*:298	15/15
	8.8 (15)	9.1 (6)	7.1 (7)	1.5 (1)	∞ *25*	0/15
f_{12}	*4.0e+7*:3.6	*1.6e+7*:7.6	*4.0e+6*:19	*1.6e+4*:52	*1.0e+0*:268	15/15
	2.0 (9)	13 (13)	8.7 (13)	∞	∞ *25*	0/15
f_{13}	*1.0e+3*:2.8	*6.3e+2*:8.4	*4.0e+2*:17	*6.3e+1*:52	*6.3e-2*:264	15/15
	3.9 (9)	8.5 (12)	∞	∞	∞ *25*	0/15
f_{14}	*1.6e+1*:3.0	*1.0e+1*:10	*6.3e+0*:15	*2.5e-1*:53	*1.0e-5*:251	15/15
	4.8 (6)	2.4 (6)	3.4 (7)	∞	∞ *25*	0/15
f_{15}	*1.6e+2*:3.0	*1.0e+2*:13	*6.3e+1*:24	*4.0e+1*:55	*1.6e+1*:289	5/5
	2.2 (2)	7.8 (6)	4.2 (4)	6.3 (5)	∞ *25*	0/15
f_{16}	*4.0e+1*:4.8	*2.5e+1*:16	*1.6e+1*:46	*1.0e+1*:120	*4.0e+0*:334	15/15
	3.5 (5)	2.1 (2)	1.7 (2)	3.0 (5)	∞ *25*	0/15
f_{17}	*1.0e+1*:5.2	*6.3e+0*:26	*4.0e+0*:57	*2.5e+0*:110	*6.3e-1*:412	15/15
	2.8 (2)	3.1 (3)	6.2 (8)	∞	∞ *25*	0/15
f_{18}	*6.3e+1*:3.4	*4.0e+1*:7.2	*2.5e+1*:20	*1.6e+1*:58	*1.6e+0*:318	15/15
	2.6 (3)	3.2 (9)	5.5 (5)	6.4 (3)	∞ *25*	0/15
f_{19}	*1.6e-1*:172	*1.0e-1*:242	*6.3e-2*:675	*4.0e-2*:3078	*2.5e-2*:4946	15/15
	∞	∞	∞	∞	∞ *25*	0/15
f_{20}	*6.3e+3*:5.1	*4.0e+3*:8.4	*4.0e+1*:15	*2.5e+0*:69	*1.0e+0*:851	15/15
	0.42 (0.2)	**0.40** (0.3)$^{\downarrow}$	∞	∞	∞ *25*	0/15
f_{21}	*4.0e+1*:3.9	*2.5e+1*:11	*1.6e+1*:31	*6.3e+0*:73	*1.6e+0*:347	5/5
	2.5 (3)	4.4 (6)	3.6 (0.9)	∞	∞ *25*	0/15
f_{22}	*6.3e+1*:3.6	*4.0e+1*:15	*2.5e+1*:32	*1.0e+1*:71	*1.6e+0*:341	5/5
	3.4 (6)	2.1 (4)	2.4 (3)	∞	∞ *25*	0/15
f_{23}	*1.0e+1*:3.0	*6.3e+0*:9.0	*4.0e+0*:33	*2.5e+0*:84	*1.0e+0*:518	15/15
	2.6 (1)	1.9 (2)	2.4 (2)	4.3 (4)	∞ *25*	0/15
f_{24}	*6.3e+1*:15	*4.0e+1*:37	*4.0e+1*:37	*2.5e+1*:118	*1.6e+1*:692	15/15
	0.25 (0.1)$^{\downarrow 3}$	4.8 (5)	4.8 (6)	∞	∞ *25*	0/15

Figure 1.12: Exp. 8

[12] "Bold entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm in BBOB-2009"

1.6.2 Function 1 : Sphere function

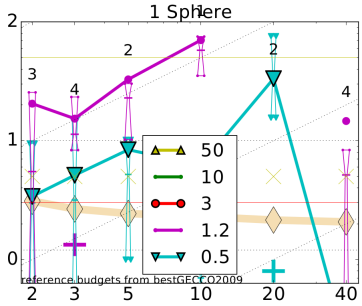

Figure 1.13: F1 Reference

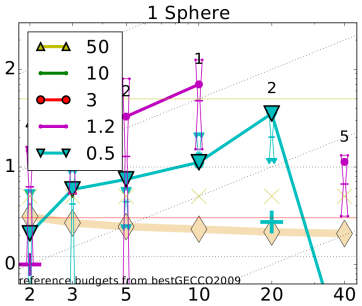

Figure 1.14: F1 Exp. 1

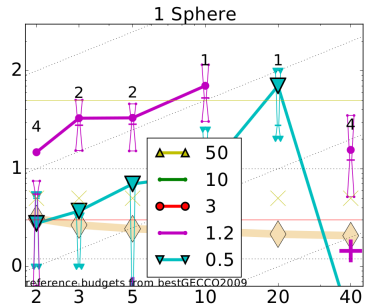

Figure 1.15: F1 Exp. 2

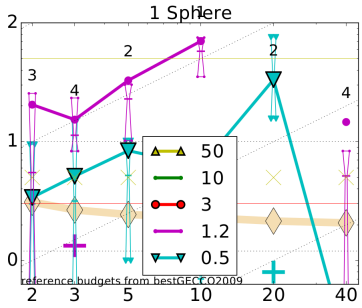

Figure 1.16: F1 Reference

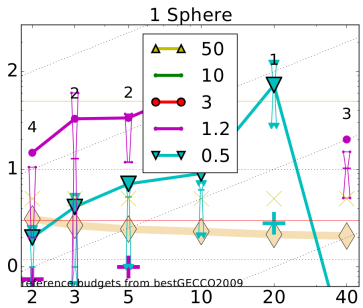

Figure 1.17: F1 Exp. 3

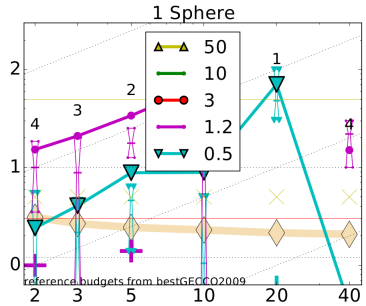

Figure 1.18: F1 Exp. 4

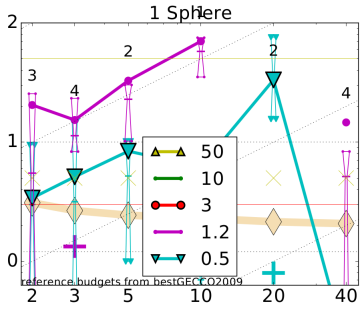

Figure 1.19: F1 Reference

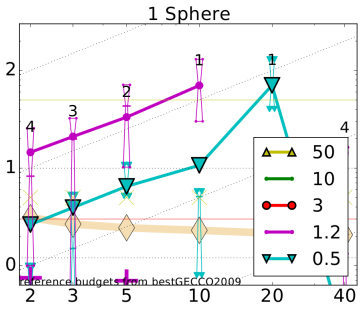

Figure 1.20: F1 Exp. 5

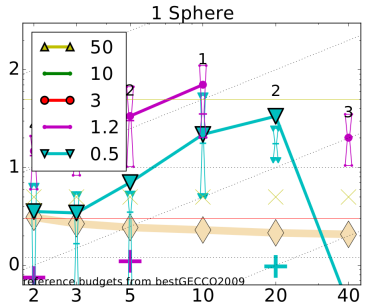

Figure 1.21: F1 Exp. 6

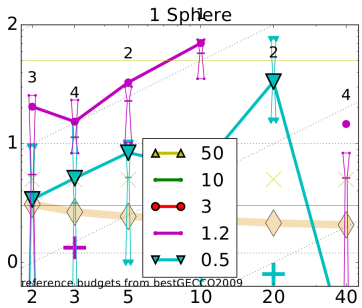

Figure 1.22: F1 Reference

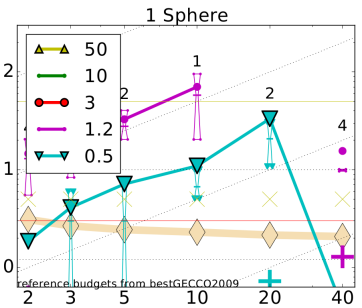

Figure 1.23: F1 Exp. 7

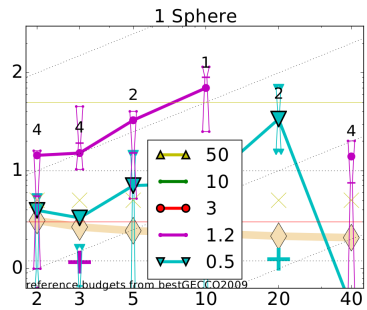

Figure 1.24: F1 Exp. 8

As expected, modifying values of parameters change results, but insignificantly.

1.6.3 Experiments 1 & 2

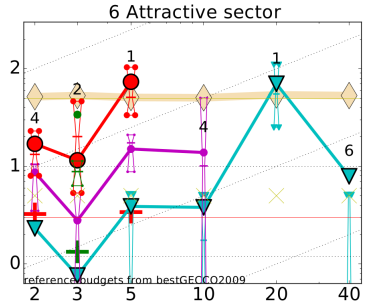

Figure 1.25: F6 Reference

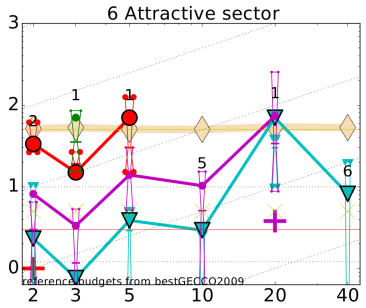

Figure 1.26: F6 Exp. 1

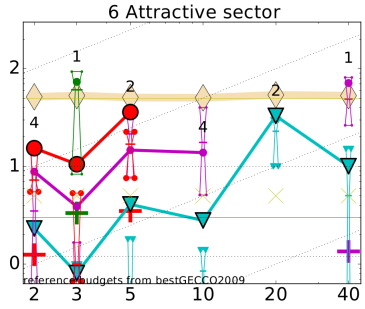

Figure 1.27: F6 Exp. 2

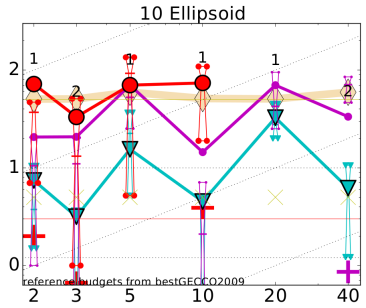

Figure 1.28: F10 Reference

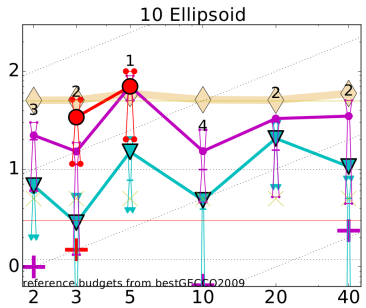

Figure 1.29: F10 Exp. 1

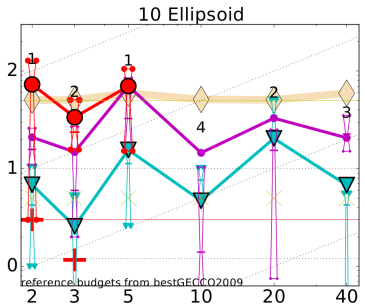

Figure 1.30: F10 Exp. 2

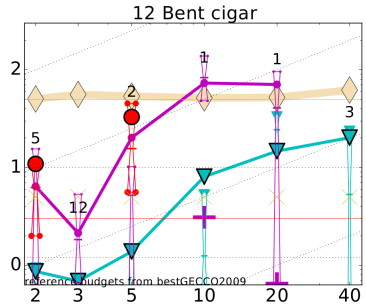

Figure 1.31: F12 Reference

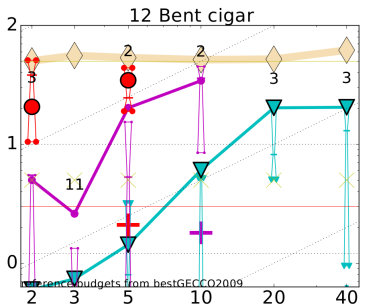

Figure 1.32: F12 Exp. 1

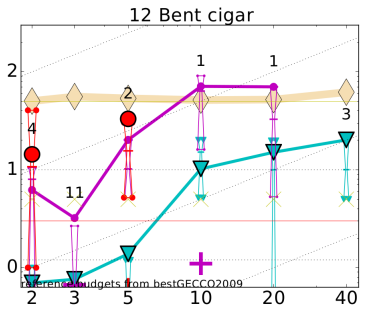

Figure 1.33: F12 Exp. 2

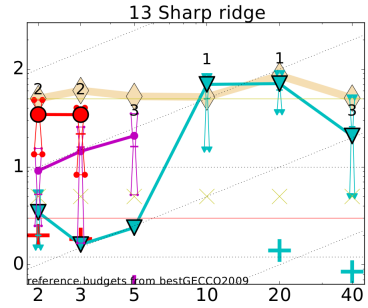

Figure 1.34: F13 Reference

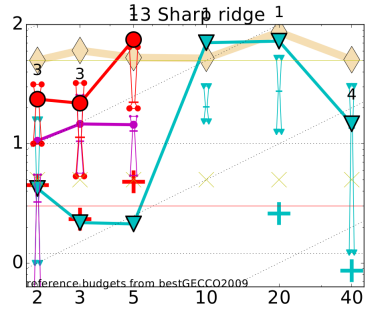

Figure 1.35: F13 Exp. 1

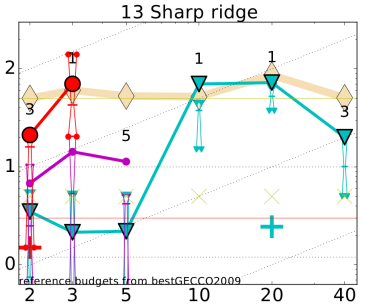

Figure 1.36: F13 Exp. 2

Goal : finding effect of increasing number of inputs

Results : this increase did not have expected impact : overall, shapes of graphs stayed the same. However, slight improvement is noticed for some functions (e.g F12 when comparing reference and experiment 2).

1.6.4 Experiments 3 & 4

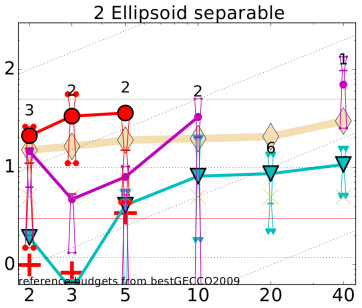

Figure 1.37: F2 Reference

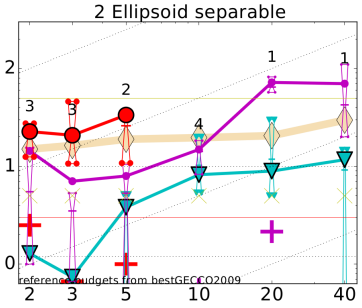

Figure 1.38: F2 Exp. 3

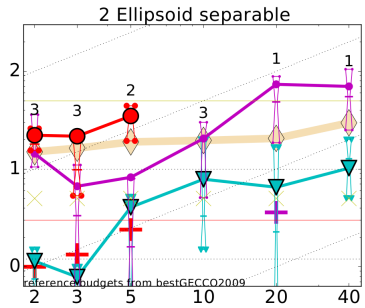

Figure 1.39: F2 Exp. 4

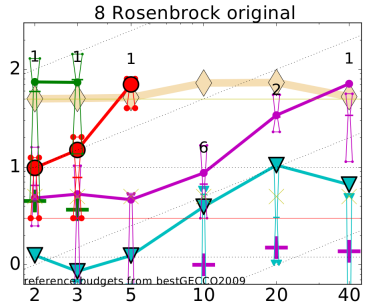

Figure 1.40: F8 Reference

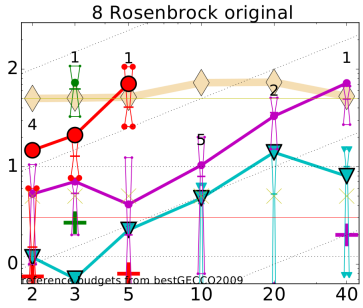

Figure 1.41: F8 Exp. 3

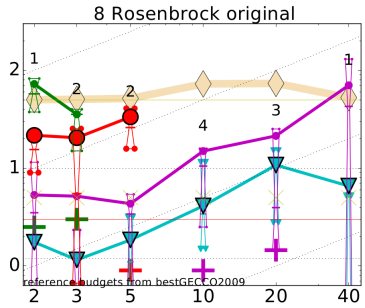

Figure 1.42: F8 Exp. 4

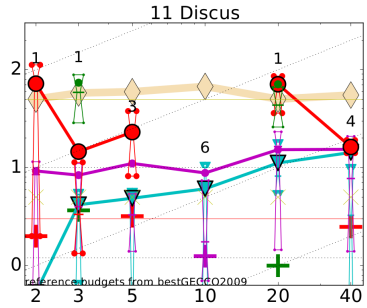

Figure 1.43: F11 Reference

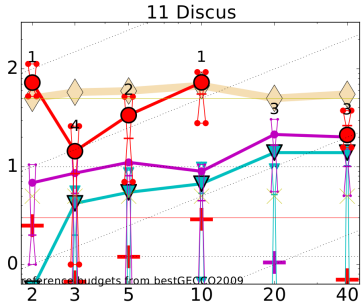

Figure 1.44: F11 Exp. 3

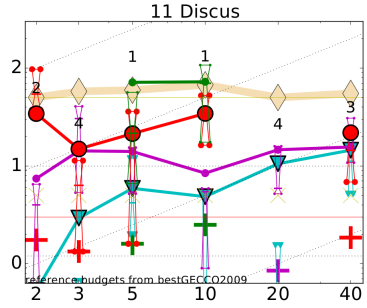

Figure 1.45: F11 Exp. 4

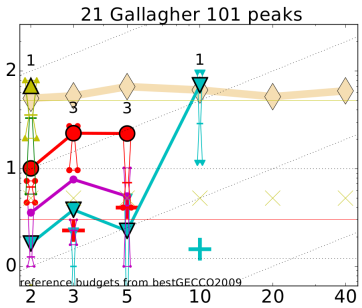

Figure 1.46: F21 Reference

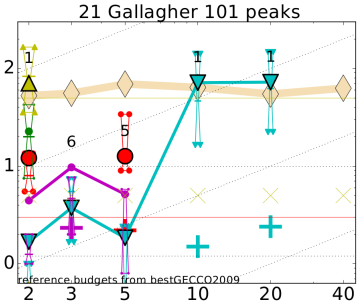

Figure 1.47: F21 Exp. 3

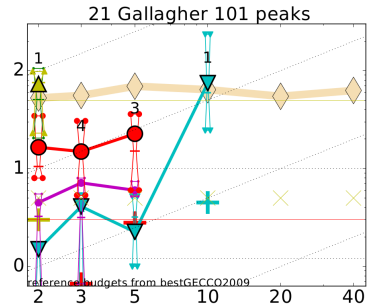

Figure 1.48: F21 Exp. 4

Goal : finding effect of increasing NHL

Results : we did not see expected improvement. Yet, behaviour of F2 (purple) and F21 (blue) changed.

1.6.5 Experiments 5 & 6

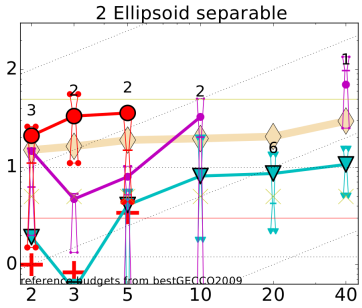

Figure 1.49: F2 Reference

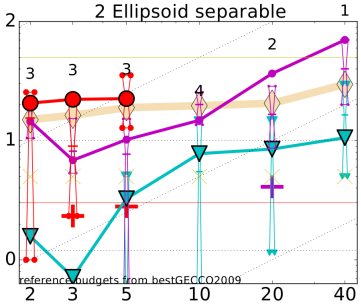

Figure 1.50: F2 Exp. 5

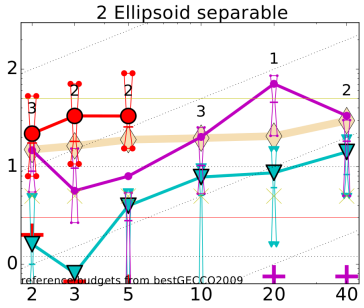

Figure 1.51: F2 Exp. 6

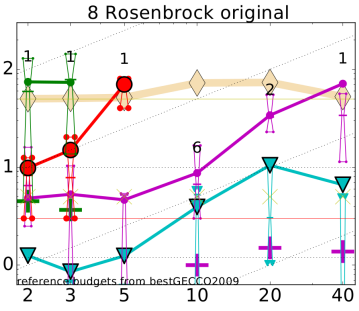

Figure 1.52: F8 Reference

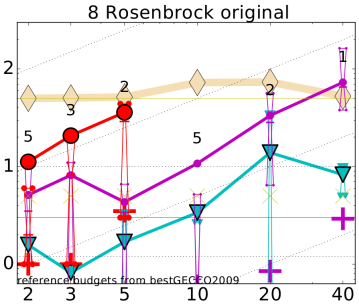

Figure 1.53: F8 Exp. 5

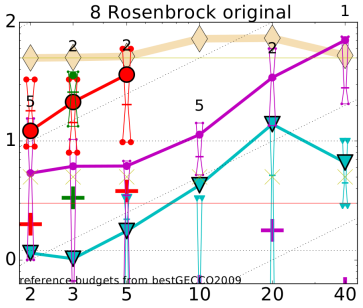

Figure 1.54: F8 Exp. 6

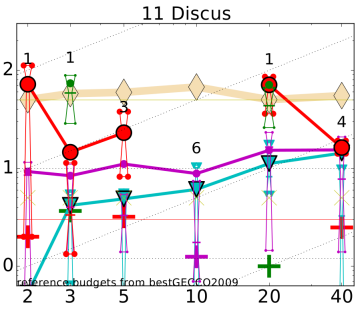

Figure 1.55: F11 Reference

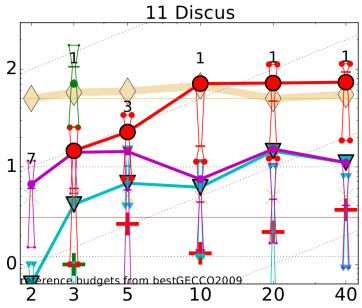

Figure 1.56: F11 Exp. 5

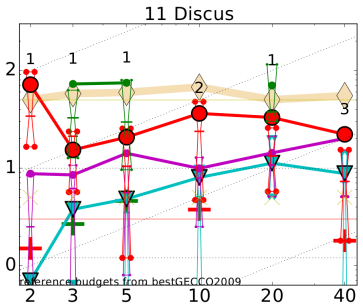

Figure 1.57: F11 Exp. 6

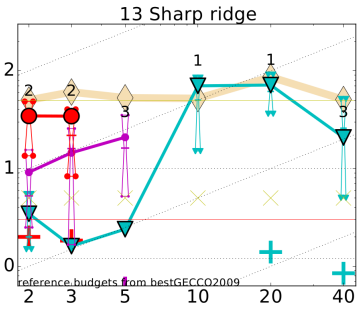

Figure 1.58: F13 Reference

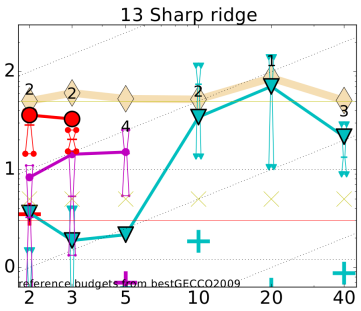

Figure 1.59: F13 Exp. 5

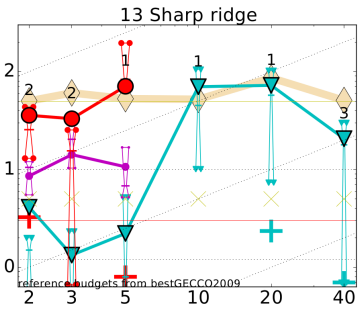

Figure 1.60: F13 Exp. 6

Goal : finding effect of increasing NNPHL

Results : we noticed some improvement for F2. However, changing NNPHL from 4 to 10 had more impact than from 10 to 50. Considering run-time difference between experiments 5 and 6, it might not be worth increasing NNPHL significantly.

1.6.6 Experiments 7 & 8

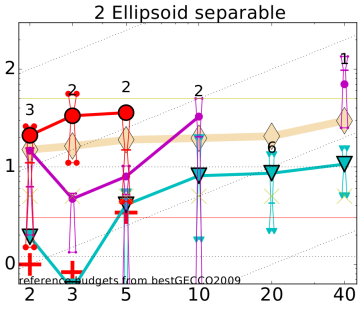

Figure 1.61: F2 Reference

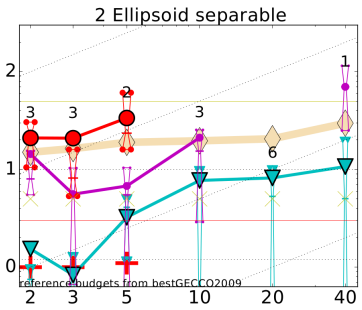

Figure 1.62: F2 Exp. 7

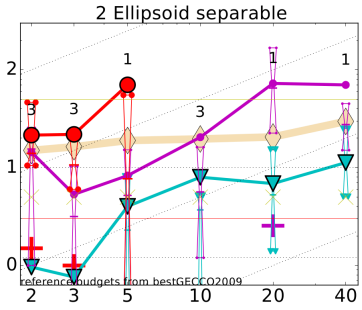

Figure 1.63: F2 Exp. 8

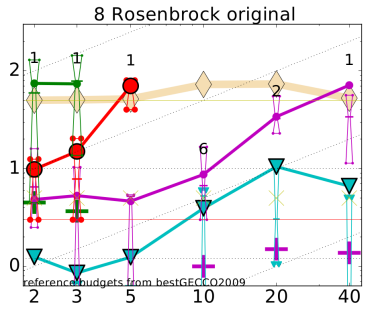

Figure 1.64: F8 Reference

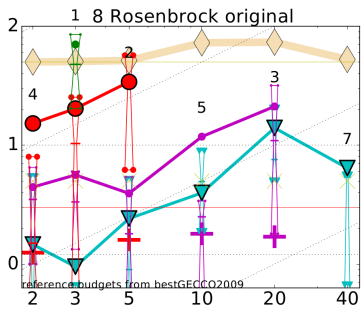

Figure 1.65: F8 Exp. 7

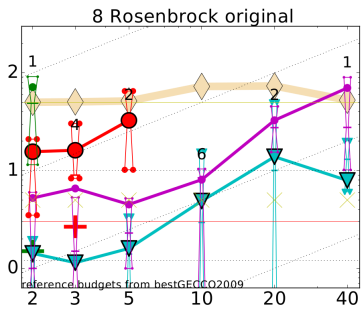

Figure 1.66: F8 Exp. 8

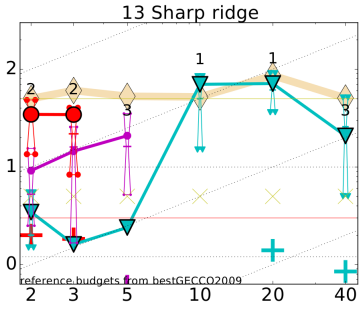

Figure 1.67: F13 Reference

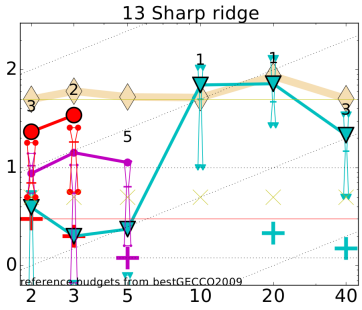

Figure 1.68: F13 Exp. 7

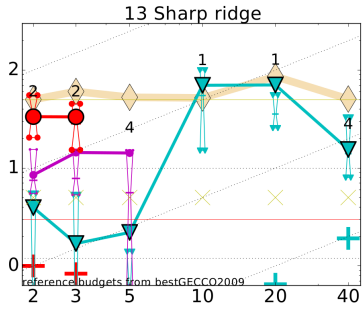

Figure 1.69: F13 Exp. 8

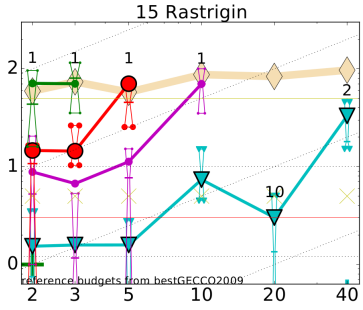

Figure 1.70: F15 Reference

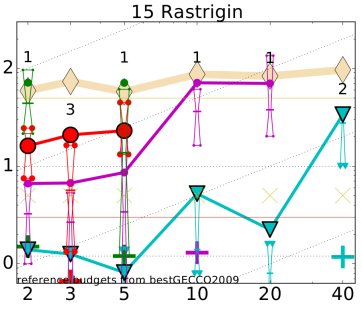

Figure 1.71: F15 Exp. 7

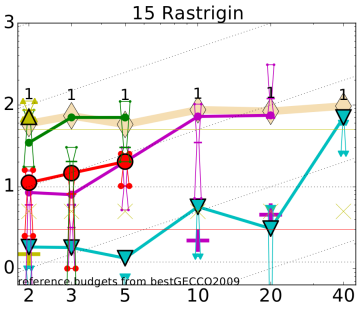

Figure 1.72: F15 Exp. 8

Goal : finding effect of increasing initial population size

Results : improvement can be noticed for F2 and especially F15, where our algorithm did as good as BBOB-2009 (blue) with 50 chromosomes.

1.6.7 Experiment 9

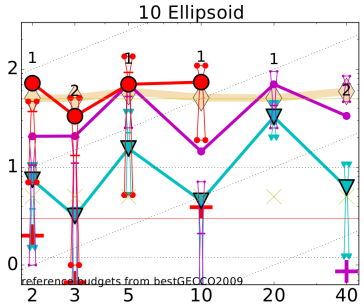

Figure 1.73: F10 Reference

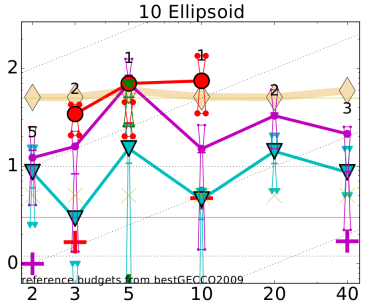

Figure 1.74: F10 Exp. 9

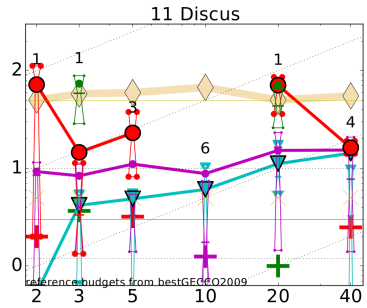

Figure 1.75: F11 Reference

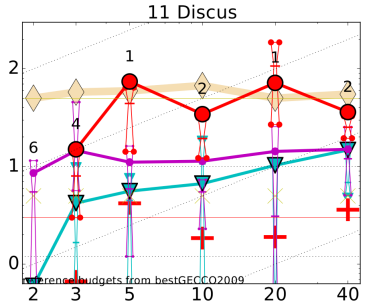

Figure 1.76: F11 Exp. 9

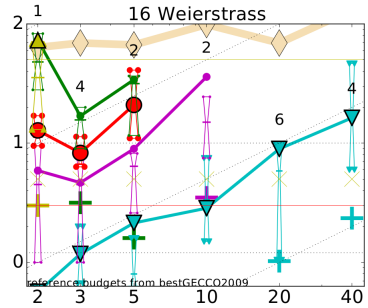

Figure 1.77: F16 Reference

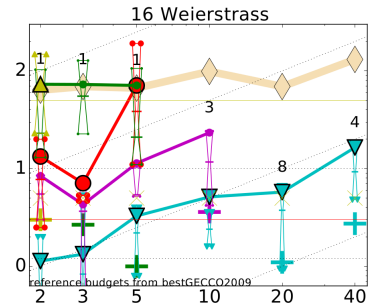

Figure 1.78: F16 Exp. 9

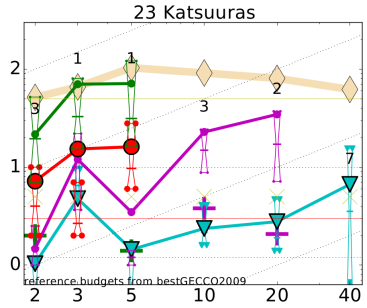

Figure 1.79: F23 Reference

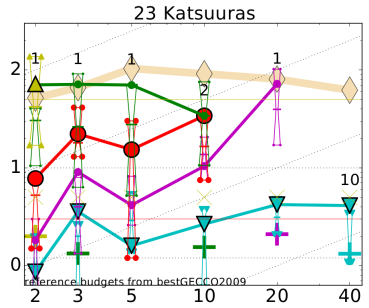

Figure 1.80: F23 Exp. 9

Goal : finding effect of increasing three parameters

Results : we barely see improvement, although we expected highest improvement here (maybe too many simultaneous changes or too small values).

1.7 Conclusion

We think it is not worth increasing values of parameters significantly because improvement / time rate decreases drastically as they grow. It might be feasible using high-performance computer. Yet, increasing each of these parameters seemed to improve results for some functions.

Chapter 2

Evolutionary Programming with GA

2.1 Introduction

The aim of the experiment is to see how changing GA methods affects its performance. We do not alter the structure of MLP.

2.2 Set-up of experiments

Tested methods :

- PS : parents selection
  - RS : random
  - WTCS : worst two chromosomes
- C : crossover
  - SP : single-point
  - TP : two-point
- M : mutation
  - SRCG : single randomly chosen gene
  - MRCG : multiple (five) randomly chosen genes
- CR : chromosome removing
  - WC : worst chromosome[1]
  - RC : random chromosome

	PS	C	M	CR
Reference	**R**	**SP**	**SRCG**	**WC**
Experiment 1	WTC	SP	SRCG	WC
Experiment 2	R	TP	SRCG	WC
Experiment 3	R	SP	MRCG	WC
Experiment 4	R	SP	SRCG	RC

We believe that changing GA methods will have more impact on results when MLP and population sizes are large. For example using a better parents selection method will make a bigger difference if initial population size is 100 and not 5.

[1] in term of custom fitness

Therefore, we decided to have bigger values for MLP :

	NIV	NHL	NNPHL	NC
MLP Structure	**20**	**10**	**8**	**30**

NB : due to lack of time[2], testing / creating additional GA methods and using larger MLP and population sizes were impossible.

Parents selection methods :

In most sources, researchers used best individuals as parents (MAGALHÃES-MENDES 2013; Pongcharoen et al. 2007), which seems a good practice. Anyhow, because we keep the best of the two populations after applying genetic operators, fitness can only be steady or better. It is therefore the case when choosing worst two individuals as parents, one of which is going to be deleted anyway. Applying genetic operators on them could improve population fitness drastically.[3] This is why we tested the effect of picking worst two individuals as parents.

Crossover and mutation methods :

We read about other crossover methods such as "cut-and-splice" (Wikipedia) which we could not test because of our fixed-length chromosomes.

We also read about arithmetic, heuristic (Leung et al. 2003), uniform, flat (MAGALHÃES-MENDES 2013) and three parents crossover (Wikipedia). Methods such as boundary, uniform and nonuniform mutation (Leung et al. 2003) exist, but we did not have time to test them.

[2]mainly due to computer power limitation

[3]Indeed, there can be a bigger improvement by selecting worst than best two (e.g it is easier to increase a mark from 10% to 20% than from 90% to 100%).

2.3 Results

2.3.1 Experiment 1

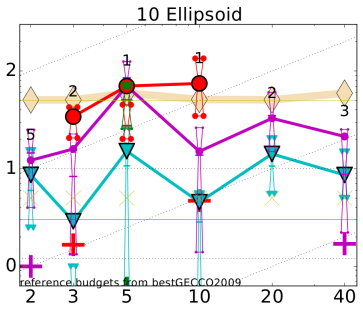

Figure 2.1: F10 Reference

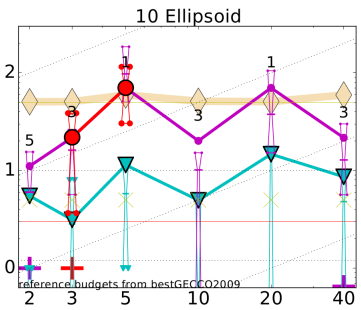

Figure 2.2: F10 Exp. 1

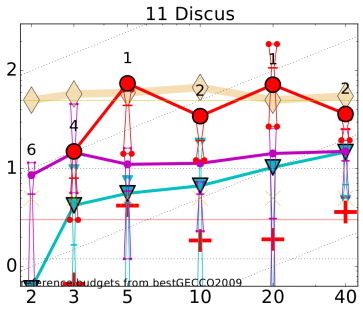

Figure 2.3: F11 Reference

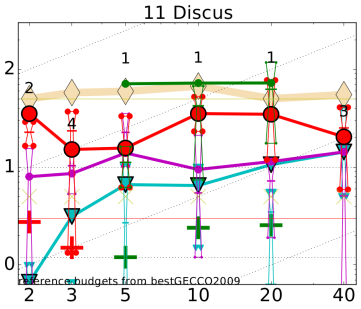

Figure 2.4: F11 Exp. 1

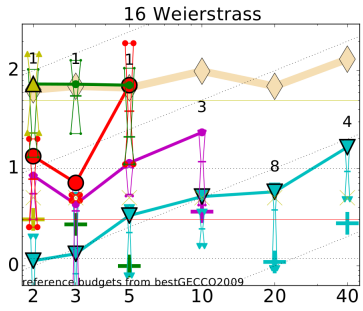

Figure 2.5: F16 Reference

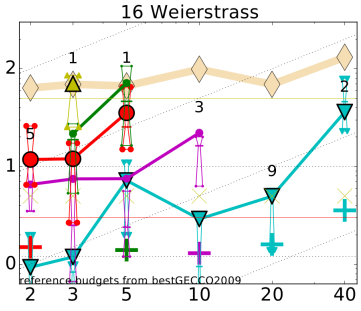

Figure 2.6: F16 Exp. 1

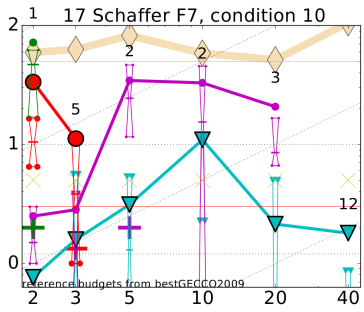

Figure 2.7: F17 Reference

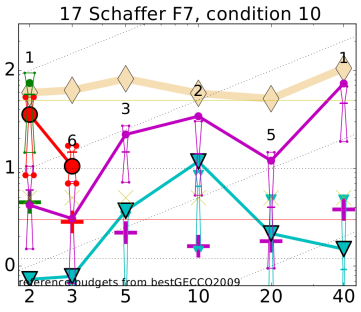

Figure 2.8: F17 Exp. 1

Expectations : improvement

Results : improvement for F10 (purple), F16 (blue) and F17 (purple), which is worth doing because it did not cost extra time.

2.3.2 Experiment 2

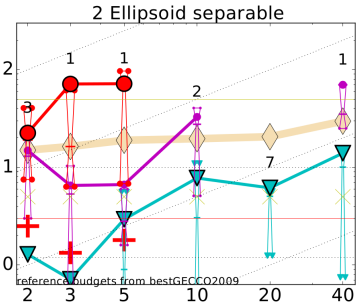

Figure 2.9: F2 Reference

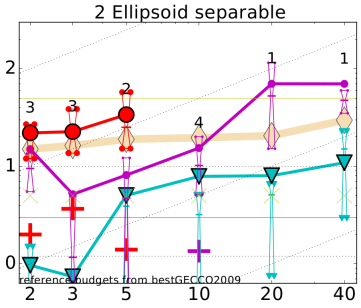

Figure 2.10: F2 Exp. 2

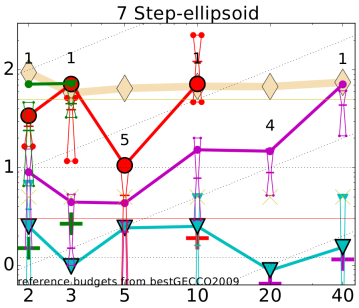

Figure 2.11: F7 Reference

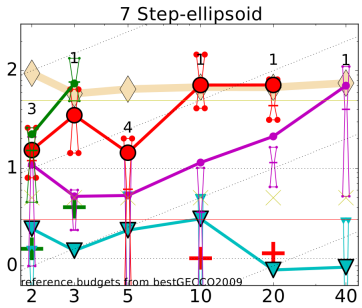

Figure 2.12: F7 Exp. 2

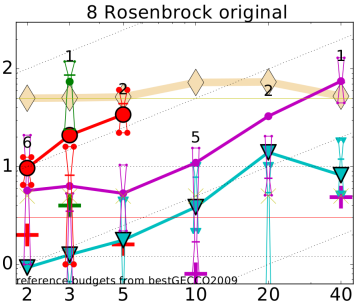

Figure 2.13: F8 Reference

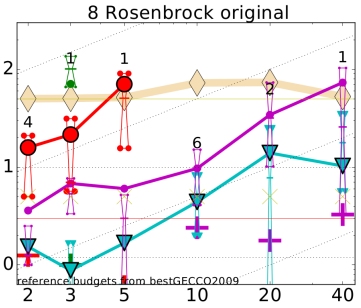

Figure 2.14: F8 Exp. 2

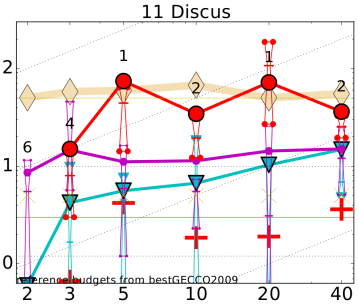

Figure 2.15: F11 Reference

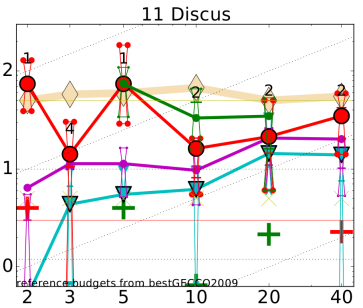

Figure 2.16: F11 Exp. 2

Expectations : random

Results : improvement for F2 and F7, but degradation for F11 (especially red).

2.3.3 Experiment 3

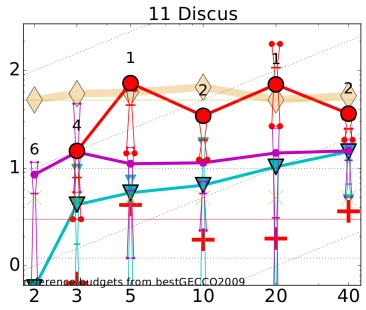

Figure 2.17: F11 Reference

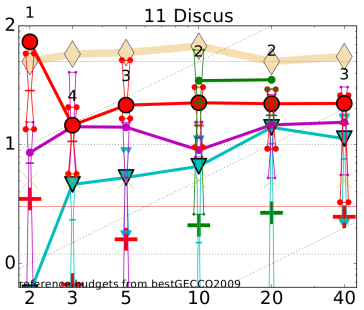

Figure 2.18: F11 Exp. 3

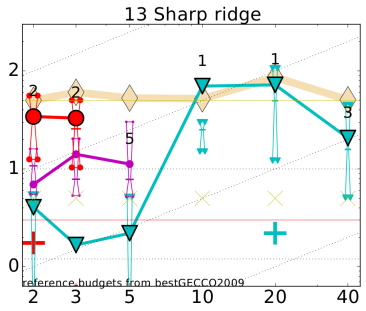

Figure 2.19: F13 Reference

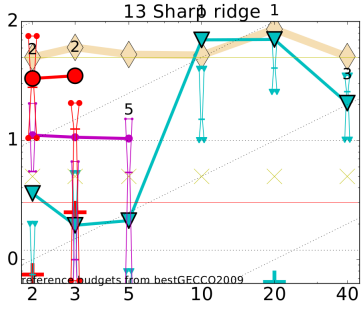

Figure 2.20: F13 Exp. 3

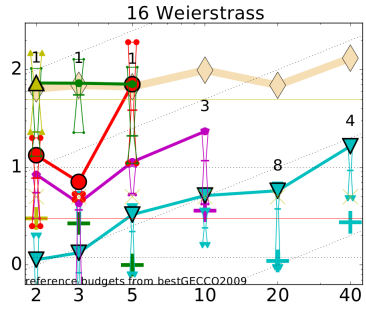

Figure 2.21: F16 Reference

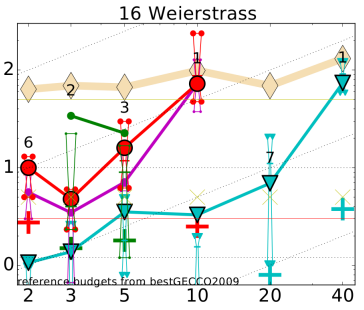

Figure 2.22: F16 Exp. 3

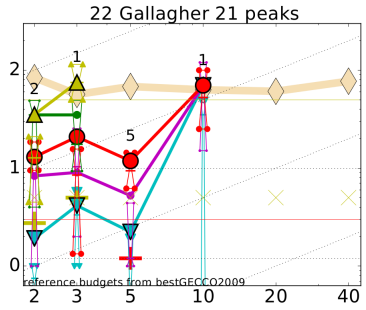

Figure 2.23: F22 Reference

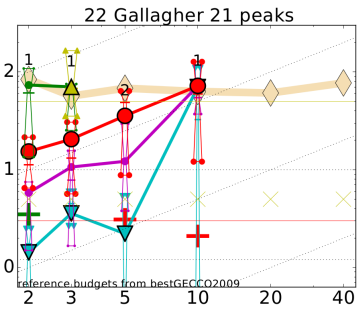

Figure 2.24: F22 Exp. 3

Expectations : random

Results : degradation for F11 (red), but improvement for F22 (red, purple) and especially F16 (red, blue, purple).

2.3.4 Experiment 4

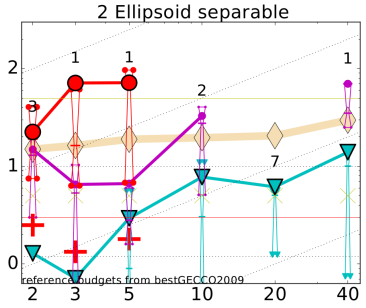

Figure 2.25: F2 Reference

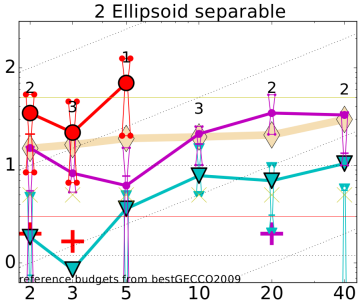

Figure 2.26: F2 Exp. 4

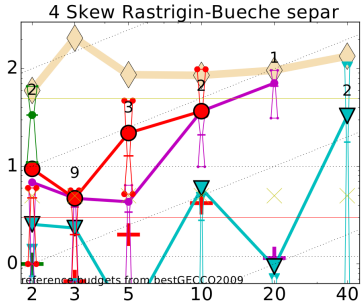

Figure 2.27: F4 Reference

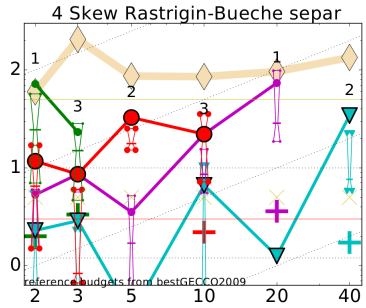

Figure 2.28: F4 Exp. 4

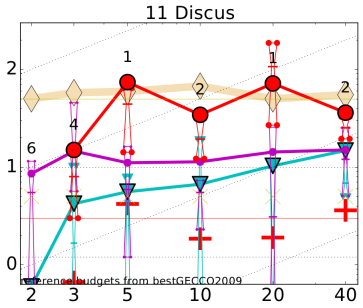

Figure 2.29: F11 Reference

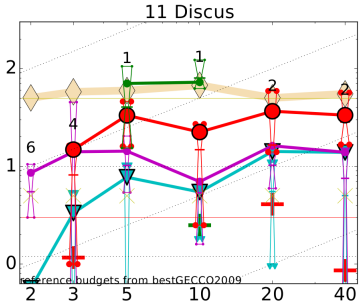

Figure 2.30: F11 Exp. 4

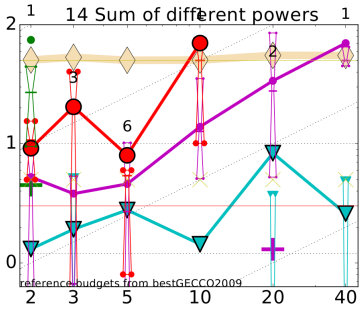

Figure 2.31: F14 Reference

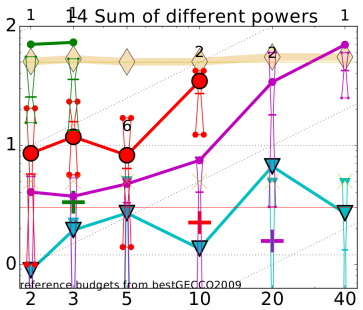

Figure 2.32: F14 Exp. 4

Expectations : degradation

Results : unexpected result. Overall, selecting a random individual did not have noticeable impact, except for F2 where we notice an unexpected improvement.

2.4 Conclusion

We conclude that choosing the worst two chromosomes as parents is a significant improvement. It would be interesting to compare it with best two selection method. We did not notice much difference between crossover and mutation methods, but maybe we did not explore many methods. Finally, it does not seem to matter using either random or worst chromosome removal method.

Appendices

Appendix A

Code

Our work is available at : https://github.com/jd57hw/f21bc

Content :

- Java code implemented in COCO's Java environment (see ExampleExperiment.java)
- results of COCO
- COCO's post-processing charts
- this report

Appendix B

Reference

MAGALHÃES-MENDES, J., 2013. A Comparative Study of Crossover Operators for Genetic Algorithms to Solve the Job Shop Scheduling Problem. *WSEAS Transactions on Computers*, 12(4), pp.164–173.
Available at: http://www.wseas.org/multimedia/journals/computers/2013/5705-156.pdf.

Pongcharoen, P. et al., 2007. Exploration of genetic parameters and operators through travelling salesman problem. *DEPARTMENT OF INDUSTRIAL ENGINEERING, FACULTY OF ENGINEERING, NARESUAN UNIVERSITY, PHITSANULOK, THAILAND. HE*, pp.215–222.
Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.538.5136.

Leung, F.H.F. et al., 2003. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural Networks*, 14(1), pp.79–88.
Available at: http://ieeexplore.ieee.org/document/1176129/.