Biologically Inspired Computation: Coursework 2 : Particle Swarm Optimization

Hugo Millet - Timothe Petitjean

PART 1 : Algorithms interesting aspects

1.1. Genetic Algorithm

The first thing to notice in our genetic algorithm code is the way we calculate the fitness of an individual :

```
def calc_fitness(population, bench_funct):
    # Calculating the fitness value of each individual in the population
    fitness = np.zeros([np.shape(population)[0]])
    for i in range(np.shape(population)[0]):
      res = bench_funct(population[i,:]) # output of the benchmark function
      fitness[i] = -bench_funct(population[i,:]) # We want to minimize so we define the fitness is opposite of the value

    return fitness
```

As we want to find a global minimum of a function, we set the fitness as the opposite of the output. This non-scaled method is therefore very effective on functions with a large scale for outputs (which is the case for most of the benchmarks functions), but less on functions with a narrow range of output.

It is also relevant to show the way we apply the mutation to the children :

```
def mutation(children, p_mutation, abs_bound):

  for child in children: # for each child
    for i in range(np.shape(child)[0]): # for each gene
      if random.random() < p_mutation: # with a probability p_mutation
        child[i] += np.random.normal(child[i], 0.3*abs_bound) # the gene takes a random number in a gaussian distribution around the initial value
        if child[i] < -abs_bound: # Keeping the values in the boundaries
          child[i] = -abs_bound
        elif child[i] > abs_bound:
          child[i] = abs_bound

  return children
```

We chose to apply a gaussian distribution centered around the previous value, in order to have mutated values closer to the initial value, made from the parents which are supposed to be good individuals. The drawback of this is that the algorithm explores a less wide range of solutions.

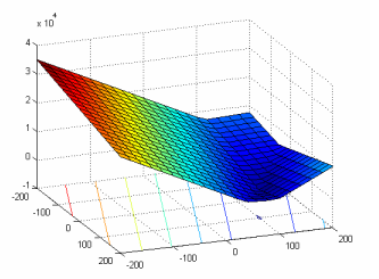
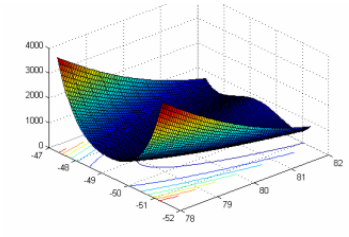
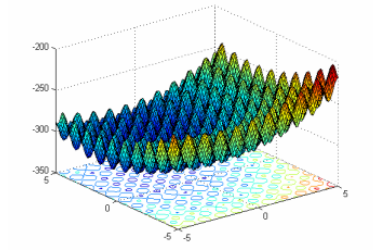
1.2. Particle Swarm Optimization Algorithm

In order to avoid premature convergence, just like mutation, we chose to randomly select the informants for each particle. We then keep the same informants for one optimization run.

```
def find_best_informant(Informants, position, position_individual, count, Swarmsize):
  Informants_Nb = [randint(0,Swarmsize-1) for _ in range(Informants)]
  Best_pos_Inf = position_individual
  for i in Informants_Nb:
    if position[i].objective_values <= position_individual.objective_values:
      Best_pos_Inf = position[i]
  return Best_pos_Inf
```

PART 2 : Benchmark Function Choice

With a goal of variety, we chose the 3 following benchmark functions to test our algorithm on :

F5 : Schwefel's problem 2.6 with global optimum on bounds	F6 : Shifted Rosenbrock's function	F9 : Shifted Rastrigin's function
		

To justify our choice of functions, we can create a table with characteristics for each of them :

Landscape \ Function	F5	F6	F9
Modes	Unimodal	Multimodal	Multimodal
Location of optimum	On bounds	Not on bounds	Not on bounds
Range of inputs	[-100;100]	[-100;100]	[-5;5]
Range of output (max is in order of magnitude)	[-310, 4*10^4]	[390;10^8]	[-330;-200]
Shifted	No	Yes	Yes

As we can see, all the parameters are varying across the different functions and the numerical range for inputs and outputs are widely spreaded.

PART 3 : Performance Comparison

Now, it is time to compare the performance of our two algorithms for these 3 benchmarks functions. The metrics we will use to measure it are :

- The mean difference to objective (MDO), being the mean of the difference between the global minimum and the best result at the end of each of the 10 iterations of the algorithm.
- The mean input relative error (MIRE), being the mean of the relative error for each input for each best result of the 10 iterations.

After research on documentation and tuning to obtain good results, we have set the simulation parameters at :

- 50 individuals per generation
- 10 000 generations

Functions \ Algorithms	Genetic Algorithm	Particle Swarm Optimization
F5	MDO : 0.7378 MIRE : 0,00032 compilation time : 12min	MDO :$1.25e^{-12}$ MIRE -$8e^{-16}$ compilation time :4mn

F6	MDO :346 953 MIRE : 1.035 compilation time : 15min	MDO :35 924 592 MIRE :4.44 compilation time :3 mn
F9	MDO : 16.81 MIRE : 0,3 compilation time : 12min	MDO :84.53 MIRE :-0.0524 compilation time :3mn

Some remarks can be made from the analysis of this result :

- Both algorithms give us good results for two of the three functions, we can say that their algorithms are coherent
- The PSO algorithm seems to be extremely precise on F5, that can be due to the homogeneous landscape or the fact that the optimum is on the bounds
- Both algorithms perform poorly to have a good output on F6, despite having inputs close to the optimum solutions. Moreover, in 2D, this function doesn't seem to have a very abrupt or uneven landscape around the optimum. The explanation for our results can be that this is not the case in 10D.
- On the F9 function, GA gets closer to the objective output but PSO has inputs closer to the optimum's.
- Overall, the PSO is faster to compute.

PART 4 : Hyperparameters variation and optimization

4.1. Genetic Algorithm

We noticed that for our genetic algorithm the fitness evolution evolves a lot in the first few generations, but very few in the last ones. We observed this phenomenon by plotting the standard deviation of the fitness of all the individuals per generation and very fast, the SD tended to 0, with occasional pikes due to mutations. This is why we chose to increase the mutation frequency, in order to explore more solutions specially in the last generations where all the individuals look the same.

Increasing the tournament size increases the chance of the best individual to be selected and therefore we lose diversity in the parents that will reproduce, but the children will have a better fitness. Having more parents increases the diversity in the children, but increases computation time, and the same applies for the number of children and the number of individuals in the population.

4.2. Particle Swarm Optimization

In order to get the Best Hyperparameters for PSO, we did various research on them, to find an optimal value for both unimodal and multimodal problems.

We chose to take 6 Informants for every particle, so that, as Garcia-Nieto and all [1] said : "*a number of 6±2 informants in the operation of PSO may compute new improved particles for longer, even in complex problems with single and multi-funnel landscapes*".

For the velocity ratio α, while trying to improve our algorithm, we noticed that if α was too high or too low, the values could sometimes make more mistakes, especially if the particle's values are close to the bounds.

We applied weight values β-ε based on the statements of Sean Luke [2].

References

[1] J. Garcia-Nieto and E. Alba, “Why six informants is optimal in PSO,” in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO ’12*, 2012, p. 25. doi: 10.1145/2330163.2330168.

[2] Sean Luke, *Essentials of Metaheuristics, Lulu, second edition*. 2013.

[3] K. E. Parsopoulos and M. N. Vrahatis, “Parameter selection and adaptation in Unified Particle Swarm Optimization,” *Math Comput Model*, vol. 46, no. 1–2, pp. 198–213, Jul. 2007, doi: 10.1016/j.mcm.2006.12.019.

[4] D. Bratton and J. Kennedy, “Defining a Standard for Particle Swarm Optimization,” in *2007 IEEE Swarm Intelligence Symposium*, Apr. 2007, pp. 120–127. doi: 10.1109/SIS.2007.368035.