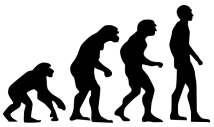
Parallel Execution for a Genetic Algorithm

Implementation, Comparison and Observations

Vlad-Stefan Tudor
coordinator: Prof. Bogdan Dumitrescu

2020

***Abstract*—Nature has been a reliable source of inspiration for mankind even as problems grew more and more abstract, its surprisingly simple and straight-forward methods helped solve otherwise head-aching problems. Evolutionary algorithms rely on the biological mechanisms of *selection*, *mutation* and *crossover* to find solutions for optimization problems by following adaptations of the individuals towards the pints of minimum or maximum.**



I. The basics of a genetic algorithm

Genetic Algorithms have been introduced by John Holland and further developed by David Goldberg, based mainly on the concept of Darwin's theory of evolution, that is, in short *survival of the fittest*.

We expect to see that as individuals in each generation are evaluated against a defined criteria they will become more and more adapted to it throughout future generations. Nature rewards adapted individuals with a higher chance of survival - and later on, reproduction - and penalizes unfit individuals. That, in time, leaves off only the ones presenting the features of most usefulness for the environment in which the population has developed.

A. *From Biology to Computer Science*

Similarly, a function of a number of parameters can be optimised with a genetic algorithm. This can be extremely useful for irregular functions that are too analytically complex for other optimisation methods. The parameters of the function will be the traits of the individuals and the fitness will be the value of the parameters through the function. If the search is for the point if maximum, individuals with higher fitness values will be rewarded.

Rewards can be modelled as weights in the computations of probability of selection for reproduction. Individuals can be ordered on the basis of their fitness score. Every implementation offers indefinite ways to take into account the performance of the individual. Mutation is necessary for escaping local optima. In other words, a certain deviation from the known path allows for the possibility of discovery of new and better routes.

B. *Main steps of the Algorithm*

The paradigm translates the mechanisms of species evolution into an algorithm with the following main steps:

1) **Generate a starting population**: the initial individuals have random parameters - they have not yet undergone any evolution.
2) **Fitness evolution**: the individuals' scores are calculated based on their parameters
3) **Selection**: individuals selected for reproduction is done with regards to the order of the fitness scores but also with a probabilistic component. This way, there is a chance even for the low-performers that may still contain useful behaviors, though not yet developed.
4) **Crossover**: two selected individuals will produce an offspring that will get its parameters as combinations of the parameters of the parents.
5) **Mutation**: each newly created individual will have a certain degree of variation from the strictly deterministic result of the parents' genes combination.
6) **Repeat steps for the next generation**

C. *Selecting an Individual*

The method used is knows as *Roulette Wheel Selection* [3]. The main idea is that chances should exist for all individuals but their odds should be proportional to each ones' fitness score. There should be a random selection (hence the roulette) on a spatial distribution of a pie-chart. Then a normal random selection is done. Naturally, individuals assigned to larger sections of the roulette have better chances.

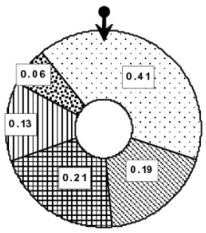


Fig. 1: Roulette Wheel Selection for 5 Individuals with different scores [4]

II. Parallel Execution

From a Computer Science point of view, algorithms running on multiple processors at the same time can mean great improvements in terms of performance. However, for a genetic algorithm, parallel execution means more than that - it actually represents a piece of the puzzle in the analogy with biological evolution.

A. *The Islands Model*

Let's imagine that evolution happens in remote areas, with different environment specifics, meaning certain adaptations that are developed throughout the generations. The final result may differ from one environment to the other: they may share a similar direction, but some may simply outperform others. However, since the initial point is chosen randomly, the local path may lead to a dead-end. Therefore, a different environment may produce a totally unexpected optima.

An even better model supposes that these environments - *islands*, let's say - are not totally remote. Individuals may *migrate* from one place to the other, bringing alongside the parameters that they developed in their home-land and merging them with the local population. This can act as directional hints that will be easily ignored if they don't present an improvement.

B. *Implementing Migration*

Migration has to be done with regards to the fitness score, after all, it's the strongest in the population that are able to break off the habit and withstand the journey. So, a first meta-parameter of the algorithm would be the proportion of population willing to move. Calculated to the size of the population, the number of migrants will be selected in order from the best fit individual to least. The second meta-parameter is the chance that they actually make it to the destination. This added randomness again gives the chance, ever so slight, that even the low performing individuals might migrate.

III. Effects of meta-parameters

Simulations have been run for an arbitrary function.

$$x_1^2 + x_2^2 + x_2 \cdot x_3 \cdot x_4^3 - (sin(x_4 \cdot x_2 + x_2 + x_4))^2$$

The search domain is for every direction (-100,100). The optimization will consist in finding a maximum value, which empirically is of order of magnitude close to 10^6

A. *Mutation*

First, let's discuss the effects of mutation. For each environment a mutation range is defined from the meta-parameter *mutation*. This means that whenever a new individual is created, apart from the linear combination of the parameters, a random number will be added, representing a mutation. This adds variety to a population that would otherwise just grow to have almost identical individuals.

Figure 2 shows the evolution of 5 different environments. No migration means that their behaviour is totally independent.

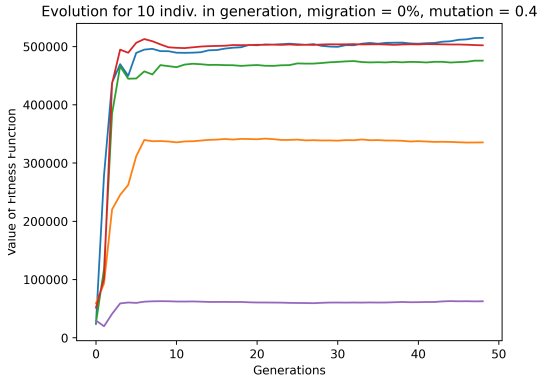


Fig. 2: Reference - evolution of 5 islands

We see that 3 of the islands bounce around the same value ($5 \cdot 10^5$), which can be the global maximum. However the other 2 functions settled at much lower values, which are probably local points of optima.

Note that *mutation* meta parameter refers to the range in which, for every environment, the acting mutation factor is selected. So, even if mutation is set at 10, the mutation factor for a certain island could still be as low as 0.1.

If we increase the mutation, the behaviour becomes less stable, but this means there can be greater chances of escaping the local optima of favour of finding the global maximum - see Figure 3. Now, strangely enough, we see that previous maximum was only local as well, as the true maximum seems now to be around $8 \cdot 10^5$.

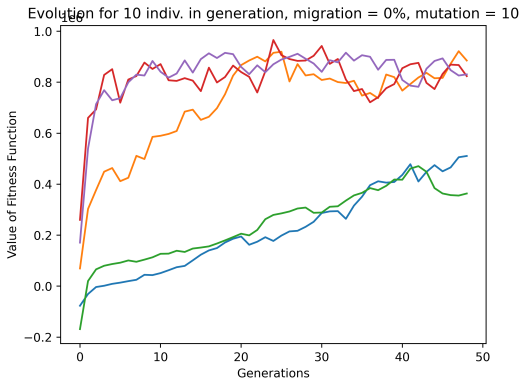


Fig. 3: Increasing mutation

A large mutation should be accommodated with a large enough population size, so that variation is damped between individuals.

B. *Number of individuals in generation*

Choosing the best number of individuals in a generation is a discussion of compromise. Too many and the algorithm will run very slowly; too few and the genetic pool will not be diverse enough.

As discussed in Section 2, more individuals dampens the oscillating evolution caused by a large mutation factor.

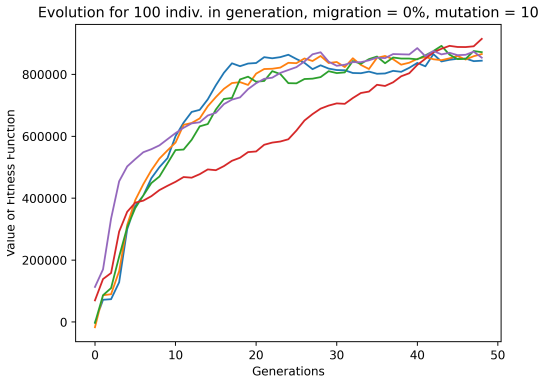


Fig. 4: Mutation effects for larger populations

Just as an idea, in Figure 5 this is a comparison between similar mutation factors but different population sizes - on the left, population size is 10, on the right it's 200. The evolution that stabilises to an almost constant value is an island with a mutation factor chosen very low.

There is no formula for calculating these parameters but

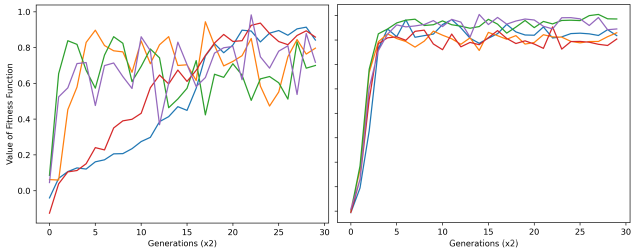


Fig. 5: Comparison for mutation = 50, for population size 10 (left) and population size 200 (right)

intuition is rapidly formed when experimenting with values.

C. *Number of Generations*

This meta-parameter does not alter results after the threshold of stabilisation. If the evolution seems to have settled around a certain value for a number of successive generations, a larger number of generations would not produce different results. However if the evolution looks like it would have kept growing, then the simulation should be run for more iterations.

D. *Migration*

All the previous examples have been simulated without any migration, meaning that the evolution of the islands were totally independent from one another. As we said, migration may help slower-evolving environments by offering them directional hints from their faster-paced counterparts. In turn, there can be useful information even in the environments that are converging slowly. As an example, they may as an inertial mass: it's hard to be moved from the spot, but once it's running, it go along well after the others have hit a halt.

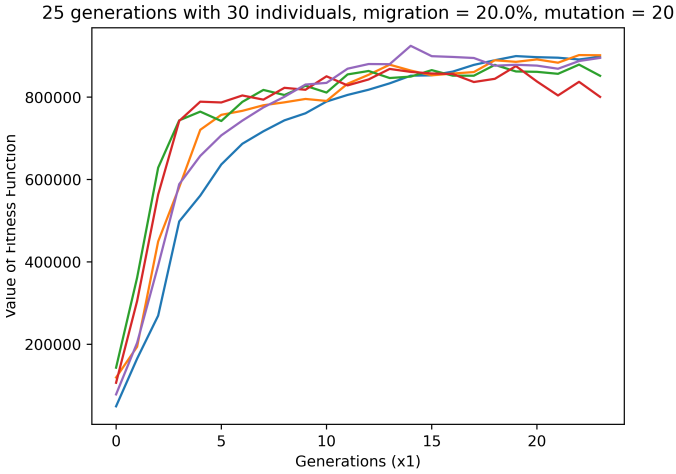


Fig. 6: Example of evolution with island migration

The first thing to be expected is that the algorithm should converge towards a unique solution, meaning that all the environments should approach the same value. That doesn't mean that they will overlap perfectly, as 100% migration means that all individuals successfully migrate to one of the other islands (not all the other islands).

To see the effect of migration we can look at Figure 7. All the other parameters are fixed. We can see that as migration gets stronger, the diversity gets cut down, as the system starts to behave like a single entity.

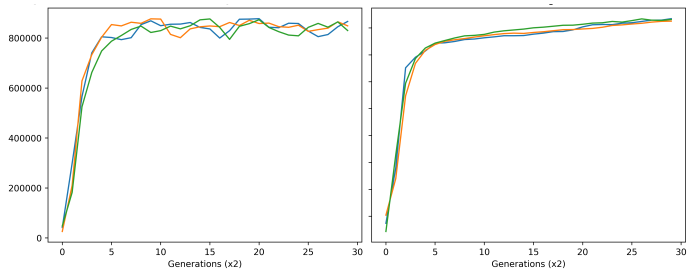


Fig. 7: Comparison between migration of 16% (left) and 100% (right)

Migration should not be set very high. It's not important for the algorithm that all the environments produce good

results - one will usually drag the others along. Therefore, a good rule of thumb would be to have diverse islands with different mutation rates and starting points and to set the migration meta-parameter just enough so that a well-performing individual can make its way to the top from any environment.

Note that migration does not produce larger and larger populations. The function that computes the next generation only does so for a constant number, set at the beginning, by selecting mainly (see the Roulette selection) the best fit-individuals, from a however big population.

IV. Is parallel better?

A. *Starting point argument*

When dealing with a nonlinear function, the starting point of the algorithm can make the difference between a local optima and a global one. The algorithm may get trapped in what it thinks it is the absolute maximum or minimum just because the starting point was in the proximity of such an area.

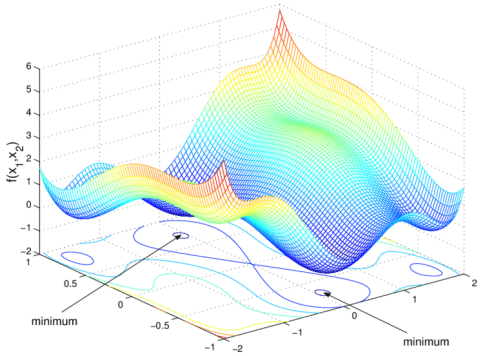


Fig. 8: Global and local optima of a nonlinear function [2]

Consider a fixed mutation factor of 1, which is rather small for the dynamic of this particular function, to illustrate the importance of the starting point. This means that it would be chance that would allow for finding of the optima in a reasonable time.

After 700 generations (note the scale - 50×14), only some of

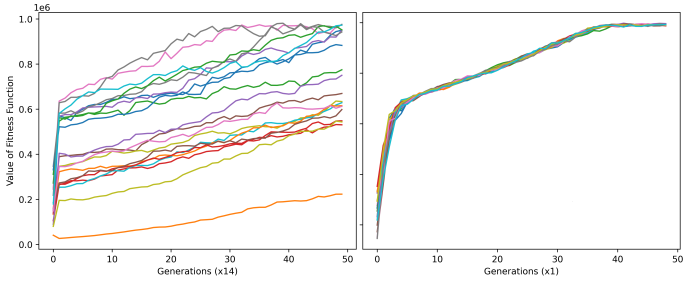


Fig. 9: Importance of diversity in starting point, 700 generations (left) of 20 independent environments and 50 generations for 20 parallel environments

the environments have found the maximum value while, with migration, the parallel-operating algorithm has converged to a solution in only 50 generations. In terms of execution time, the 50 generations of the environments with migration rate of 50% take about 11.2 seconds to run.

Evolving an independent environment takes 7.5 seconds but since the starting point is merely a guess, it would be like randomly betting on a single evolution from the left image on the left. Most of them finish far off from the optima so to have a reliable solution you would still have to perform multiple simulations. Computing 20 independent (non-migrating) environments for 700 generations took 1 minute and 48 seconds, and the solution still does not feel robust enough, looking at how each evolution oscillates.

B. *Mutation Factor*

We have to consider that in a general case, we have almost no information of the function. This raises problems in choosing a mutation factor. If it is too small, it can take a very long time for the algorithm to converge or it will end of getting stuck in a local optima. If it is too big with regards to the natural variations of the function, the search can simply leap over the optima.

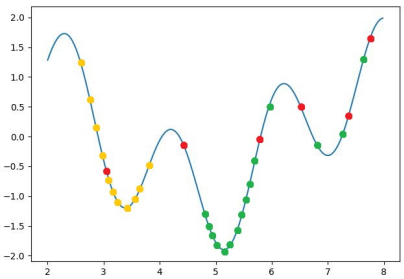


Fig. 10: Mutation problem in optimization: with yellow, individuals with low mutation; with red individuals with high mutation; with green, a correct mutation factor

If we consider the evolution displayed in yellow in Figure 10, we see it starting from the left and slowly converging to an optima. Because its mutation is low, new individuals are unlikely to appear too far from the rest, and in the close proximity the point is indeed a minimum of the function. Starting from the right, with red, wee see an evolution that encounters the opposite problem. It's so dynamic that it fails to stabilise even when it gets closer to the global optima - it simply leaps over it. Green shows a correct choice of mutation factor. In this case, we discuss the minimum value of the function, but the problem is identical to both minimum and maximum optimization.

Similarly, in our optimization problem it's unknown which would be an appropriate value for the mutation. Therefore, since it would ultimately be a guess, there are far better chances to get it right with more than a single bet. But for truly discussing parallelism, we have to see whether those multiple attempts perform better individually or when communicating information.

For this comparison we have chosen a random mutation factor for each island, in the range (0,10), uniform distribution. What Figure 11 shows feels natural: we see on the left that there are unlucky choices of mutation, getting stuck at a local optima, not even close to the true, global maximum value. Other environments converge towards the correct result but not all at satisfactory rates.

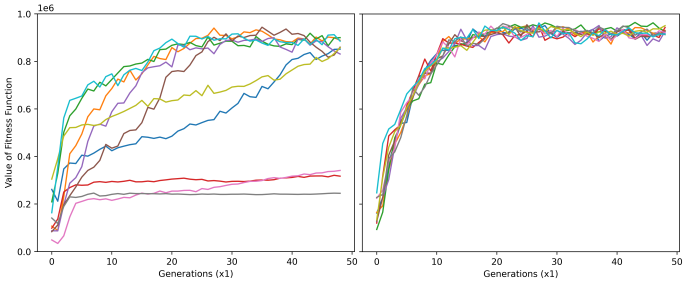


Fig. 11: Evolution of environments with random mutation (0-10), independently (left) and with migration (right)

On the right, due to migration (factor of 25%), all the trajectories are bundled together and converging to the optima after about 20 generations. In terms of execution time, execution with migration takes 4.6 seconds, 1.6 seconds more than the 10 independent processors. If we were to run a single processor, it would take considerably less, but more generations could be necessary and, moreover, there is only a slight probability that an optima would be found from the first try.

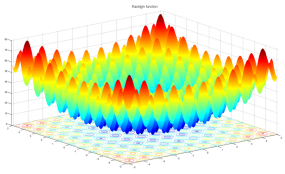
C. *Results for classic optimization functions [5]*

Note that the function values will be negative, since the algorithm finds the minimum by searching for the maximum of the symmetrical function $-f(x)$

- **Rastrigin Function** (2 variables)

$$f(x) = 2 \cdot 10 + \sum_{i=1}^{n} (x_i^2 - 10 \cdot cos(2\pi x_i))$$

$$f(0,0) = 0$$



  - 30 islands without migration: -0.7 (11.2 seconds)
  - 30 islands with migration: -0.08 (13.4 seconds)
  - parameters: 30 generations, 70 individuals in generation, maximum mutation 0.2

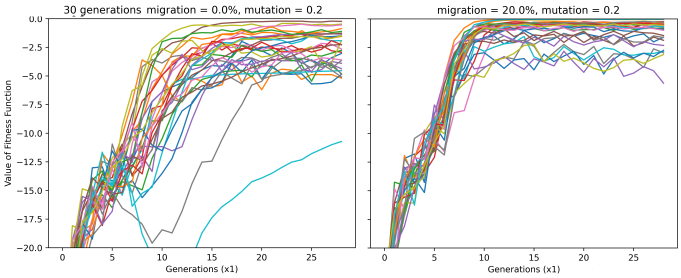
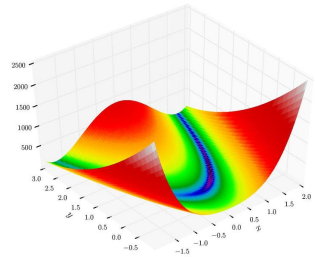


Fig. 12: Optimization of Rastrigin function, 30 islands, without migration (left) and with migration(right)

- **Rosenbrock Function** (3 variables)

$$f(x) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2)$$

$$f(1,1,1) = 0$$



  - 30 islands without migration: -23.9 (5.1 seconds)
  - 30 islands with migration: -1.26 (6.3 seconds)
  - parameters: 20 generations, 50 individuals in generation, maximum mutation 0.1

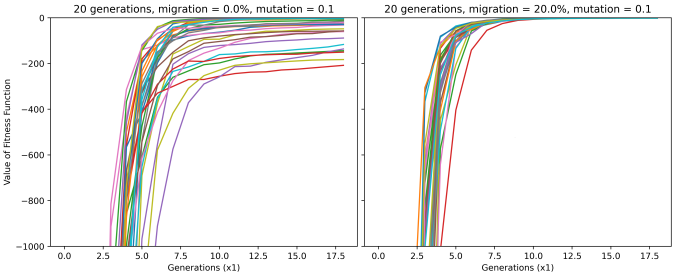


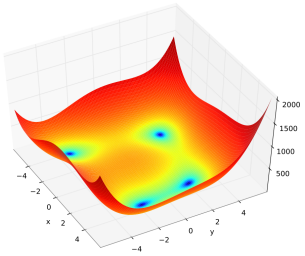
Fig. 13: Optimization of Rosenbrock function, 30 islands, without migration (left) and with migration(right)

- **Himmelblau Function** (2 variables)

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

$$f(3,2) = 0, f(-2.80, 3.13) = 0$$

$$f(-3.77, -3.28) = 0, f(3.58, -1.84) = 0$$



  - 30 islands without migration: -1.0 (8.9 seconds)
  - 30 islands with migration: -0.4 (11.1 seconds)
  - parameters: 50 generations, 50 individuals in generation, maximum mutation 1

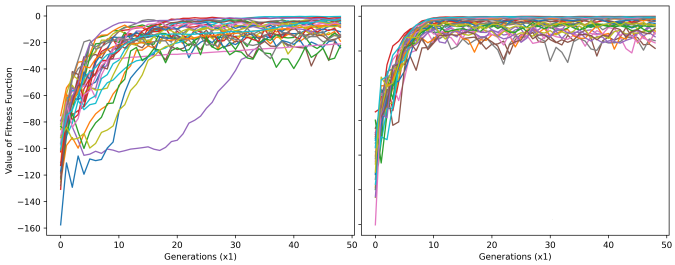


Fig. 14: Optimization of Himmelblau function, 30 islands, without migration (left) and with migration(right)

V. Conclusion

Genetic Algorithms are, by definition a game of chance whose strength is in large numbers. It is not a surprise, nor needs demonstrating, that a stochastic method performs better with multiple running instances. The question has been whether these parallel instances should exchange information, and whether that would help convergence speed and stability of the results. There also is the problem of processing power - it was important to note how the trade-off stands between execution times and improvements in results.

From what the numbers tell, this approach not only helps in terms of the solutions approaching the same value, which is a comforting sign when dealing with probability and chance, but also can solve the problem of deciding on the meta-parameters of the algorithm. Having communicating parallel processes allows for defining a much broader range for the meta-parameters, as the algorithm will slowly select the best-behaving configuration. Moreover, a fast-evolving environment that hits a plateau may be surprisingly put again into action by data emerging from the slower-evolving environments.

Furthermore, a conclusion can be derived also outside the spectrum of numbers. The model very well resembles human migration patterns - with islands evolving by their own rules but also accepting individuals that may bring knowledge from the outside. And given how communication this has been our entire history the main goal of civilisation, we cannot imagine that a solitary genetic algorithm would outperform it's community-open counterparts.

VI. Python Implementation

Full code is available at:

`github.com/tdrvlad/Parallel-Genetic-Algorithm`

Credits for the basis of the code goes to : [1].

References

[1] Genetic Algorithms Explained. https://hackernoon.com/genetic-algorithms-explained-a-python-implementation-sd4w374i. Last accesed: 06/05/2020.

[2] Nonlinear Optimization. https://www.researchgate.net/figure/Example-of-a-nonlinear-optimization-problem-with-the-humpback-function-as-objectiv fig1_263048325. Last accesed: 06/05/2020.

[3] Roulette Wheel Selection. http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php. Last accesed: 06/05/2020.

[4] Roulette Wheel Selection Ilustration. https://www.researchgate.net/figure/Fitness-Proportionate-Selection-a-Roulette-Wheel-Sampling-b-Stochastic-Universal_ fig2_265879767. Last accesed: 06/05/2020.

[5] Test functions for optimization. https://en.wikipedia.org/wiki/Test_functions_for_optimization. Last accesed: 06/05/2020.