Evolutionary Computing

TP16 - Immigrants Insertion

Joana Brás[1 [2021179983]] and Pedro Rodrigues[2 [2018283166]]

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia da Universidade de Coimbra
Portugal
joanabras@student.dei.uc.pt pedror@student.dei.uc.pt

Abstract. Premature converge is a well known problem in the context of evolutionary based optimization strategies. Furthermore, researchers even advocate that the key solution for this problem is the preservation of diversity within the population of candidate solutions. Hence, with our work we study the immigrant insertion evolutionary strategy and evaluate its impact in the preservation of the population's diversity, allowing the algorithm to further explore the search space (Random Immigrants) or exploit some candidate solution through local search (Elitist Immigrants).

Keywords: Evolutionary Computing · Evolutionary Algorithm · Evolutionary Strategies · Immigrant Insertion · Optimization

1 Introduction

The usage of evolutionary algorithms is widely considered a solution for a huge variety of real world problems, namely ones that require the search of solutions in wide space, whilst and having a very rugged fitness landscape. The usage of other algorithms for solving these problems is also considered an option for some situations where the problem is well defined and presents some particular property. Although, for example, in NP-Hard problems where an optimal solution cannot be found with ease or in reasonable time, the evolutionary strategy presents itself as a method that provides a nature inspired search process that takes into consideration both the exploration and exploitation of candidate solutions in the search space.

Nonetheless, evolutionary algorithms, being heuristic in their nature, are susceptible to premature convergence. This problem arises from the fact that, during the search the candidate solution, the algorithm may evolve in the direction of optimizing a solution that after a certain time will converge to a local minima/maxima of the objective function. To counter this effect, the diversity of candidate solutions must be kept, existing multiple strategies, one of them being the insertion of the, so called immigrants during the search process.

The immigrant insertion strategy consists in replacing the worst individuals in the set of candidate solutions, at the end of each iteration of the algorithm,

with new ones that contribute to the diversity of the population. Some of the methods for generation immigrants are the random and the elitist ones. In the random immigrants method, new individuals are generated by sampling random solutions in the search space, on the other hand, in the elitist immigrants method, the new individuals are generated by mutating the best solution found so far. Note that the random immigrants are designed to introduce global search and the elitist ones local search.

Our goal with this work is to study the influence of the insertion of immigrants in the performance of the algorithm. Therefore, given a set of benchmark problems and an evolutionary algorithm empirically parameterized, we will test if the insertion of random and elitist immigrants will improve the performance, at the very least, from an anytime perspective.

2 Implementation

In order for the construction of the evolutionary algorithm we started by implementing the required recombination, variation, selection and immigrant insertion operators that constitute the core of this evolution strategy.

Starting by the recombination operators we implemented the "*N-Point Crossover*" and "*Uniform Crossover*" which work for both binary and floating point genotypes. Additionally we also implemented a crossover operator that works exclusively with floating point representations of the genotype, the "*(Whole) Arithmetic Crossover*".

For the mutation operators, in binary representations we implemented a uniform mutation operator which we called "*Uniform (Bit Flip) Mutation*". As for the floating point representation we implemented a "*Uniform Mutation*" operator as well as a "*Gaussian Mutation*" one.

For the selection strategy in terms of parent selection we implemented the "*K-Way Tournament Selection*" and "*Roulette Wheel Selection*" operators. In terms of the survivor selection we used the "*Elitism*" strategy.

That being said, the only thing left was the immigrant insertion and the evolutionary algorithm main loop which we also did, after which we were able to test the algorithm with some parametrizations and gather results.

3 Benchmarks

The choice of benchmark problems to test our algorithm was based the type of the problem (representation) and ruggedness of the fitness landscape best suited for the test of the immigrant insertion strategy. Therefore, we chose the *João Brandão' Numbers* problem for the fact that it has a binary representation, and the fitness function has to take into consideration two conflicting objectives that need to be maximized and minimized (the fitness being a scalarization of these two). Also, we chose all the real valued multi dimensional functions for

multiple different properties of the landscape in terms of local optima, modality, monotony, etc... .The chosen functions were Sphere (De Jong's F1 function), Rosenbrock (De Jong's F2 function), Step (De Jong's F3 function), Quartic (De Jong's F4 function).

Problem	JB	Sphere	Rosenbrock	step	quartic
Size	30	20	20	20	20
Genotype Domain	{0, 1}	(-5.12, 5.12)	(-2.048, 2.048)	(-5.12, 5.12)	(-1.28, 1.28)
Representation	Binary	Float	Float	Float	Float

Table 1. Benchmark Problems Description

The table 1 describes in detail the benchmark problem parametrization used for testing the evolutionary algorithms.

4 Experimental Setup

For testing the evolutionary algorithm with immigrants insertion we began determining a suitable set of parameters that produced good results when the plugged into an evolutionary algorithm. In order for that to happen we tested experimented the values described in the tables below, 2 and 3 and selected the algorithm that showed the most promise, given the results we obtained. Note that results were obtained empirically by performing a grid search over the whole set of parameters, taking care to select different seeds.

Should the reader like to analyse the results we obtained from the grid search the results can be found in the data annexed to this file.

Parameter	**Values**
Problem	Function(Sphere, Rosenbrock, Step, Quartic)
Generations	1000, 1500 and 2000
Population size	30, 50 and100
Population dimensionality	20
Crossover function	Two Point Crossover, Arithmetic Crossover and Uniform Crossover
Crossover probability	0.85, 0.9 and 0.95
Mutation function	Gaussian Mutation and Uniform Mutation
Mutation rate	0.05, 0.1 and 0.15
Parent's selection function	K Tournament Selection and Roulette Wheel Selection
Survivor's selection function	Elitism

Table 2: Parameters analyzed of the benchmark problem of the different functions

Parameter	**Values**
Problem	João Brandão's Numbers
Generations	1000, 1500 and 2000
Individual Size	30
Population size	30, 50 and100
Population dimensionality	20
Crossover function	Two Point Crossover and Uniform Crossover
Crossover probability	0.85, 0.9 and 0.95
Mutation function	Uniform Bitflip Mutation
Mutation rate	0.05, 0.1 and 0.15
Parent's selection function	K Tournament Selection and Roulette Wheel Selection
Survivor's selection function	Elitism

Table 3: Parameters analyzed of the benchmark problem João Brandão's numbers

Every Crossover Function and Mutation Function takes as input the probability of crossover and probability of mutation, respectively. Both Arithmetic Crossover and Uniform Crossover functions have as input the domain of the function mentioned in 1. Aside from these parameters, we identified the additional inputs defined in table 4.

Parameter	**Parameter Type**	**Subparameter**	**Value**
Crossover function	N Point Crossover	points	2
	Arithmetic Crossover	sigma	0.7
Mutation function	Gaussian Mutation	sigma	Random values
Parent selection function	K Tournament Selection	size	100
		k	3
	Roulette Wheel Selection	size	100
Survivor's selection function	Elitism	elite	0.2

Table 4: Additional parameters whose value was established as described.

Afterwards we proceeded with a more structured approach. Having the parameters defined, the next step was to test the algorithm with the immigrant insertion. We considered both strategies for inserting immigrants: random and elitist - and we set the percentage of random immigrants inserted at the end of each run to 20%. The three algorithms (default and the two with immigrant) were ran thirty times and the best fitness values after each one of the runs was stored, in order to apply statistical tests and determine if there is any statistical difference in performance between the different algorithm versions.

5 Results

5.1 Parameters used

Our first experiment was to determine which parameters worked best for our chosen benchmark problems. To that end, we did 30 runs using every parameter mentioned earlier. We checked the best fitness of each set of values for the five benchmark functions we chose. The common ones are the population size (100), the number of generations (2000) and the survivor selection (Elitism: the last 20% of the population with the worse fitness is discarted). The specific function parameter values are the following:

- João Brandão's numbers:
  mutation = UniformBitflipMutation(probability=0.10)
  crossover = NPointCrossover(probability=0.85, n_points=2)
  selection = KTournamentSelection(size=100, k=3)

- Ronsenbrock Function numbers:
  mutation = GaussianMutation(probability=0.05, sigma = random, domain = [-2.048, 2.048])
  crossover = UniformCrossover(probability=0.85, domain = [-2.048, 2.048])
  selection = KTournamentSelection(size=100, k=3)

- Quartic Function:
  mutation = GaussianMutation(probability=0.05, sigma = random, domain = [-1.28, 1.28])
  crossover = ArithmeticCrossover(probability=0.9, domain = [-1.28, 1.28], alpha=0.7)
  selection = KTournamentSelection(size=100, k=3)

- Sphere Function:
  mutation = GaussianMutation(probability=0.05, sigma = random, domain = [-1.28, 1.28])
  crossover = ArithmeticCrossover(probability=0.9, domain = [-1.28, 1.28], alpha=0.7)
  selection = KTournamentSelection(size=100, k=3)

- Step Function:
  mutation = mutation = GaussianMutation(probability=0.05, sigma = random, domain = [-1.28,1.28])
  crossover = NPointCrossover(probability=0.85, n_points=2)
  selection = KTournamentSelection(size=100, k=3)

5.2 João Brandão's Numbers

On the figure 1 we have represented the evolution of the fitness of the benchmark problem "João Brandão's numbers" using the same parameters referred previously. On the graphic we see three different lines, referring to different evolutions. The color blue refers to the evolution of the fitness of the algorithm with no immigrant insertion. The line with the color orange is the evolution algorithm adding random immigrants to the population and with green we represent the evolution of inserting mutations of the best individual.

In the figure we can observe that the Random Immigrants's curve reaches a higher fitness than the other two curves, as well as reaching the maximum value faster. The worst results were obtained using elitist immigrants.

Since this is a maximization problem, the curve starts at a low fitness (since the individuals are generated randomly which translates in a higher number of violations) and stabilizes after less than 250 generations for all the algorithms.

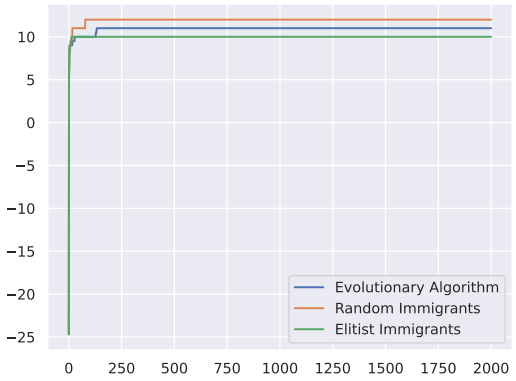

Fig. 1. Evolution of the fitness of the benchmark problem "João Brandão's numbers" using the same parameters refered previously, using the seed 768.

The result of the statistical tests to access if the random or elitist algorithms are better than the default one are presented in the table below. The values in the columns KSTest and Wilcoxon correspond to the pvalue of the test and effect size is also shown in order to visualize how significant is the difference between algorithms. The result column is showing whether we accept that both algorithms are equal or that are differences between them. The results tell us that there is a difference in the elitist strategy.

Algorithm	KSTest	Wilcoxon	Effect Size	Result
Default	1.44e-19	———	———	———
Random Immigrants	1.442e-19	1.0	0.1339	Accept
Elitist Immigrants	5.58e-17	1.0e-5	-0.749	Reject

5.3 Ronsenbrock

The same color scheme is used as mentioned previously (blue represents the progression of the fitness of the best individual in each generation without inserting immigrants, orange and green is the same progression but adding random immigrants and elitist immigrants).

Opposed to the previous figure, on figure 2, the initial fitness is elevated and lowers to a more reasonable value. That occurs due to the fact that this is a minimization problem and such, the goal is to obtain the lowest fitness possible.

Once again, inserting random immigrants results in a better fitness than the other two algorithms, but it takes longer to achieve that same result. The worst results were obtained with the non insertion of immigrants. As we can see the results tell us that there is no difference, contrary to what we expected

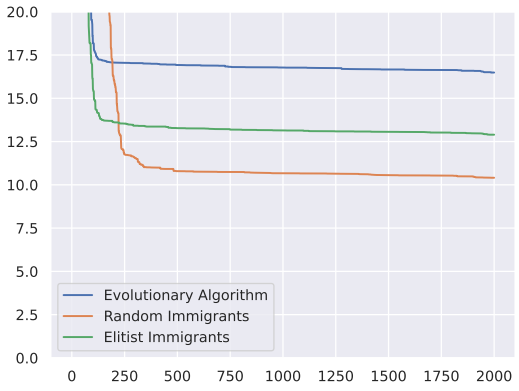

Fig. 2. Evolution of the fitness of the benchmark problem "Ronsenbrock" using the same parameters referred previously, using the seed 417.

The result of the statistical tests to access if the random or elitist algorithms are better than the default one are presented in the table below. As we can see the results tell us that there is no difference, contrary to what we expected.

Algorithm	KSTest	Wilcoxon	Effect Size	Result
Default	0.059	———	———	———
Random Immigrants	1.91e-18	0.21	-0.179	Accept
Elitist Immigrants	2.48e-14	0.7812	0.005	Accept

5.4 Quartic

As mentioned, all functions are a minimization problem, where we want to obtain the lowest fitness possible, as such, the individuals start with a high fitness and improve on every generation.

As we can see in figure 3, the worse fitness is inserting elitist immigrants. Both remaining algorithms (random immigrants and non immigrant insertion) have a similar evolution, being that having outside individuals improves the fitness slightly.

Both random immigrants and non immigrant insertion algorithms stabilize at a later generation while as adding elitist immigrants stabilizes on an early generation.

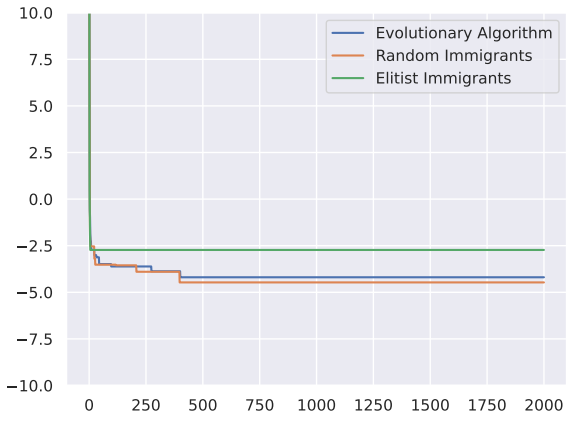

Fig. 3. Evolution of the fitness of the benchmark problem "Quartic" using the same parameters referred previously, using the seed 261.

The result of the statistical tests to access if the random or elitist algorithms are better than the default one are presented in the table below. As we can see the results tell us that there is no difference, contrary to what we expected.

Algorithm	KSTest	Wilcoxon	Effect Size	Result
Default	0.38	———	———	———
Random Immigrants	0.0003	0.503	-0.069	Accept
Elitist Immigrants	0.0001	1.73	-0.858	Accept

Table 5. Benchmark Problems Description

5.5 Sphere

As we can observe once again, both random immigrants and non immigrant insertion algorithms have a similar evolution. Although on early generations the random immigrant algorithm has a slightly better fitness evolution, after around 750 generations, they both become similar.

The worst curve is, once more, the one representing the evolution of inserting mutations to the best individual in the population. Despite this fact, it stabilizes at an earlier generation, when comparing to the other two algorithms.

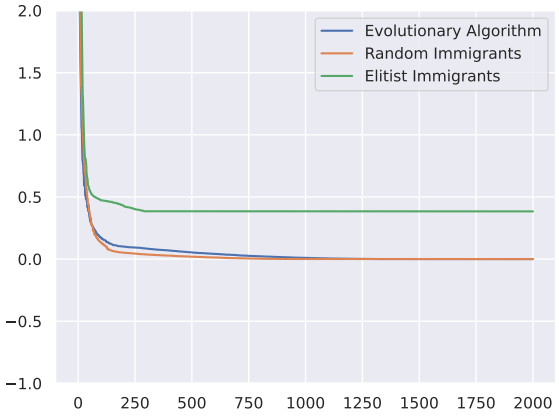

Fig. 4. Evolution of the fitness of the benchmark problem "Sphere" using the same parameters referred previously, using the seed 270.

The result of the statistical tests to access if the random or elitist algorithms are better than the default one are presented in the table below. As we can see the results tell us that there is no difference, contrary to what we expected.

Algorithm	KSTest	Wilcoxon	Effect Size	Result
Default	5.98e-12	———	———	———
Random Immigrants	9.06e-10	0.503	-0.069	Accept
Elitist Immigrants	5.29e-12	1.73	-0.858	Accept

5.6 Step

The evolution of the fitness on each generation using the Step function has a somewhat shaky curve, which, is to be expected due to the fact that it is a function with integers.

Unlike any other benchmark problem we analyzed, all the algorithms seem to have a similar evolution, however, it is possible to notice that the algorithm that provides a worse fitness curve is the "standard" evolutionary algorithm.

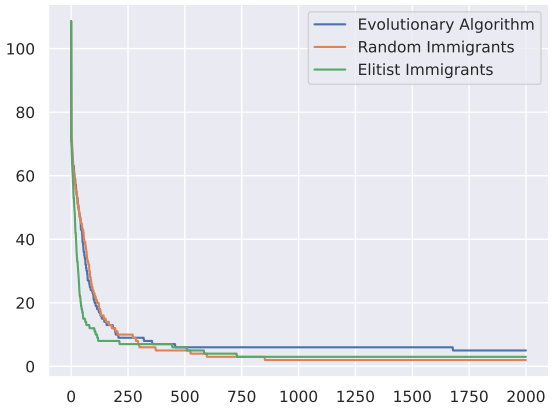

Fig. 5. Evolution of the fitness of the benchmark problem "Sphere" using the same parameters referred previously, using the seed 212.

Algorithm	KSTest	Wilcoxon	Effect Size	Result
Default	8.64e-9	———	———	———
Random Immigrants	5.80e-6	0.0057	-0.46	Reject
Elitist Immigrants	4.332e-5	0.0016	-0.54	Reject

The result of the statistical tests to access if the random or elitist algorithms are better than the default one are presented in the table below. As we can see the results tell us that there is statistical difference between the default algorithm and ones with immigrants

6 Conclusion

Overall, we notice that, for every benchmark problem, the worse algorithm is elitest immigrant insertion. This may be due to the fact that mutating a good individual may result in a worse individual that the initial population. We also noticed a resemblance between adding random individuals and adding none at all. Along with the tests we made, we can conclude, with some uncertainty that the algorithms have no significant statistic difference between them for most functions (except for the Step Function). It is possible, however, that with a bit more parameterization, this difference would be easily identifiable. We also noticed that all curves stabilize at generations lower than 750. This means that increasing the number of generations may result in finding a better individual (as it happens with the Step function for the no immigrant insertion algorithm) but generally speaking and according to the results we obtained, the algorithm tends to stabilize and be "stuck" around a certain individual.