UNIVERSITY OF AMSTERDAM

Evolutionary Computing Project report

Luca Simonetto - 11413522
Fabrizio Ambrogi - 11403640
Nedko Savov - 11404345

Evolutionary Computing

October 20, 2017

1 Problem statement

The objective of this project is to optimize a function maximization problem, namely to find the best coordinate vector x that maximizes $f(x)$, where a 10 dimensional function f is given. Every dimension is constrained to the $[-5, +5]$ domain, reducing the total landscape to be searched. Therefore, the problem is as a constraint optimization problem (COP). The input is given as the total number of allowed evaluations and three properties of the (unknown) function: *multimodality*, *separability* and *structure*. Having such few details about the fitness landscape, the objective is to develop an algorithm (or a number of them) as robust as possible to the different variations it can encounter.

The implementation relies on an external evaluation software, that scores every input in a scale from 0 to 10, where 0 is the minimum and 10 is the maximum. Outside of the allowed range of coordinates, the evaluation is always 0. By doing that, the search space constraints are indirectly handled, effectively converting the problem into a free optimization problem (FOP). The problem representation as genome used in all the following algorithms is simple - a 10 dimensional real-valued vector x.

This report shows the different approaches that have been taken, the reasoning behind every algorithm and the results obtained with each one. Noticeably, as every algorithm needs a certain number of parameters to be set, attention has been taken on searching from the literature the optimal values for every parameter, or having them being a function of the given input.

While an algorithm could be chosen for each function type, the objective of this work is to search for a method that is general and efficient enough to provide good results on all the proposed functions, in order to reduce the overhead of choosing the best alternative in a real-world scenario.

2 Hypothesis

The scope of the report is to test whether the difference in efficiency between evolutionary algorithms might be due to parameters choice, or if there exists a particular evolutionary algorithm that performs better in a specific set of problems. In the experiment the most common evolutionary methods are metaoptimized for every single parameter in a empirical way (grid search and manual tweaking), with the goal of testing the functions at their top performance to see if there's a clear best option.

3 Evolutionary Algorithms

3.1 Generational and Steady State GA

The first algorithms that have been implemented for this project are the two basic GA flavors: steady state and generational. Each algorithm requires multiple components, namely *selection*, *crossover*, *mutation* and *survival*, and for each component a choice has to be made in order to select the most appropriate version. The main difference between SSGA and GGA is how the population is treated: SSGA is a $(\mu+\lambda)$ algorithm, indicating that after producing λ offspring from the μ parents the new population is chosen from $\mu + \lambda$ candidates. On the other hand, GGA is a (μ, λ) algorithm, as only the λ children pass to the new generation. [Whi94] Motivation for using these algorithms is that many different versions of their highly independent from each other components exist and the right combination can lead to good results. Having defined the basic properties of the two genetic algorithms, the component's variants that have been implemented can be presented:

- **Selection**: Linear Ranking, Tournament and Fitness Proportional Selection, [GD91] along with the addition of windowing for the latter method. Furthermore a form of Tribalistic parent selection was tried, that penalize the creation of couples between elements which are too similar or too different from each other.
- **Crossover**: Average, Weighted Average, Coin Flip, Weighted Coin Flip and N Points Crossover. Every method returns only one child from the two (or more) parents, apart from N Points Crossover, that returns the same number of complementary children as parents given.
- **Mutation**: Gaussian, Triangle and Uniform Mutation.
- **Survival**: Best Fitness.

As the Katsuura and Schaffers functions present a problematic landscape, it has been opted to use elitism in the GGA, allowing the algorithm to pass the best individuals so far into the next population, without mutating or recombining them. This made the algorithm more robust, by retaining the information regarding the best individual. Other than that, stagnation has been added in the SSGA implementation, in order to continuously check if fitness improvements have been made: after every epoch, the best fitness of the population is checked against the best so far, and if no improvements have been made, a stagnation counter is increased. When the counter reaches a user-defined *stagnation threshold*, the worst n% of the individuals in the population is removed, allowing an equal number of random individuals to be added. This allows to better sample the function landscape, and basically restarting the algorithm when stuck in a local optima. If this method doesn't bring enough improvements, a wipeout procedure, initiated again after a user-defined *wipeout threshold*, simply removes all but a few number of individuals in the population. This procedure is the last resort of the procedure, as it effectively restarts the entire genetic algorithm. Although going against the general flow of the SSGA algorithm, this method brought good improvements in the average fitness of the final results.

3.2 Differential Evolution

Against its own name, this algorithm was studied to approach non-differentiable, multimodal and non-linear functions, to be analyzed with a stochastic direct search method

[SP97]. At each epoch part of the population is randomly selected and divided in groups of parents, each group contains a reference, a base and an even number of other parents. The child (or mutant) genome is selected with a coinflip crossover between the reference one and a linear combination of the others. The resulting mutant replaces the reference parent in the population if its score is better, otherwise it dies.
This algorithm uses the following parameters:

- **Coin bias**: The probability, for each gene of the child, whether it comes from the reference or from the combination of the other parents.
- **Type of base**: The base parent can be randomly picked, like the others, or could always be the same for each child, specifically it would be the element with the best fitness in all the population.
- **Number of pairs**: The remaining parents are used in couples, taking the difference in their genes as part of the linear combination. A bigger number of couples means more variation to the pool, but too much variation brings the risk of non-convergence.
- **F**: Value by which all the pair differences are multiplied, before being summed to the base.

This algorithm has proven to be exceptional for Bent Cigar and Schaffer functions with very quick convergence on the global optimum. However, despite its promising initial results and the use of the optimal parameters [Ped10] , it didn't manage to tackle the deep irregularities of Katsuura's function.

3.3 Particle Swarm Optimization

This method, in contrast with the typical evolutionary algorithms inspired by nature, drives the evolution flow in a way that is closer to a physical model. During its execution, the particles move around a hyperplane with a velocity that is influenced by the gravity from past and other particles' results. Seen from a biological point of view, effectively each member of the population has a child influenced by the rest of the elements, with their scores. This algorithm is capable of keeping a genetic memory that contains the best result of the main parent and of the whole population, which influences with different magnitude the genome of the children, seen as a form of attractors.[PKB07]
Apart from the size of the swarm, the parameters of this method are:

- **Inertia**: It is possible to imagine this evolution as a form of movement through an hyperplane, each parent and child are different timestep realizations of the particle. With this value each child will retain and pass on some of the evolutionary drift that brought its genome where it is.
- **Self-best attraction**: Each particle creates a child and dies, but its genome contains the memory of the best element of the bloodline. This parameter stands for how much this familiar genome will weight in the child one.
- **Global-best attraction**: The historic best element of the population has its genome in all the swarm. This value defines how strongly it influences the one of the child.

Since this method doesn't require any derivative to evolve and is thought for complex space exploration, it has been seen as a good candidate for this project. However, multimodal and, especially, very irregular functions as Katsuura appeared to not be the right playground for this algorithm and its success was limited just to the simpler function.

3.4 CMA-ES

The Covariance Matrix Adaptation Evolution Strategy is a stochastic, derivative free method prevalently used with non-convex, non-separable, ill-conditioned, multi-modal or noisy objective functions (like Katsuura, which is also nowhere differentiable). CMA-ES is highly competitive for both local [HO01] and global [HK04] optimization. It is considered to be among the state-of-the-art methods for optimization. CMA-ES makes use of a large number of internal parameters. However, their choice is not left to the user, as optimal values for all but two of them are already assigned as part of the strategy design. This removes the need of tedious parameter tuning. Among the already mentioned motivations is that using evolutionary strategies comes natural when there is a real value vector representation, as it is in this case.

This algorithm works by maintaining a population of λ individuals, scattered on the function landscape according to a single Gaussian distribution with mean m and covariance $\sigma^2\Sigma$. [HAA15] The step size parameter σ is used and evolved to determine the size of the new Gaussian. At each iteration, the likelihood of the μ best individuals in the population is maximized, done by adapting the mean and covariance. The mean is weighted between the best μ individuals. The weighting was chosen to be uniform. The covariance is increased or decreased, respectively when at, or near, a local optimum. For the σ and covariance updates two evolution paths are used. Evolution path is an exponential fading record of mutation steps so far. [HAA15] It contain significant information about the correlation between consecutive iterations.

Usually small populations (λ) work well for CMA-ES. We can benefit from that by using less fitness evaluations per epoch. The number of selected individuals (μ) based on literature should be somewhere between a fourth to a half of the population size. Too big population size wastes evaluations and too small, meaning also a smaller selection size, leads to staying closer to local optima and generally lower accuracy per epoch.

Algorithm restarts were also implemented. A restart is done when stagnation in the fitness has been detected and it has been continuously happening for a certain amount of epochs (a fitness increase with less than 1e-15 after 10 epochs). Using restarts helps when the algorithm moves into a local maximum area away from the global optimum due to randomness.

As usual with self adapting mutations, the step size (σ) has to be clamped at the bottom to prevent extremely small updates. It is also clamped at the top - the value chosen was 2, as it is not too large to go beyond the constraints on the space. It was noticed that most of the time the sigma stays equal to the lower clamping value, increasing only when the algorithm finds itself in a local optimum. This made the choice of lower bound value critical, because its optimal value is different for different functions. If the step size is too big, the algorithm explores too much and is not well directed to an optimum, if instead it is too small - the strategy will advance too slowly towards a maximum. A grid search is done on the lower bound live when the strategy is running within $[0.1, 0.3, 0.4, 0.5, 0.6]$ to find a balanced setting. The numbers were selected based on experimentation. The value is changed after each restart in a round robin style. m is always initialized with the zero vector, Σ with the identity matrix.

For this implementation, a choice had to be made between the $(\mu + \lambda)$ and (μ, λ) flavors of evolution strategy (already discussed in 3.1). The former would keep the Gaussian anchored to the best solution, the latter allows for more exploration of the landscape. After experimentation, it was noticed that (μ, λ) performs significantly better on noisy functions (Katsuura) and slightly worse on the other functions. Therefore, (μ, λ) was selected, with the small modification to add the mean from the previous epoch as an individual to the new population. The reason behind this is to keep a small influence from the past, allowing the population not to drift away if the new individuals have worse fitness.

A disadvantage of CMA-ES is that there are gradient approximation methods that are much faster to converge on not noisy and not very complex differentiable functions. However, these methods can still not handle well functions outside of these categories.

In summary, the algorithm uses truncation selection, multi-parent whole arithmetic recombination (uniformly), Gaussian perturbation adaptable mutation and (μ, λ) survivor selection.

4 Parameters tuning

Having presented the algorithms used in this project, this section covers the techniques and results obtained from the parameter tuning phase, aiming at optimizing the final performance of the methods.
To avoid an indefinite protraction in time for the search, a time limit of 10 seconds was set for each run of an algorithm.

4.1 Genetic Algorithms

The first choice to be made, when using this family of algorithms, is which specific version of each operator to use: based on literature and experimental results, the choice were:

- **Selection**: Linear Ranking has been preferred, as it removes the downsides of Fitness Proportionate selection (premature convergence, loss of selection pressure and function transposition) while still preferring fitter individuals over unfit ones. The Linear Ranking's selection pressure has been set to two different values based on the used algorithm: in SSGA it has a value of 1.75 to be a balanced choice between random selection (1.5) and heavy selection (2), while in GGA it has a value of 2, meaning that the search for a maximum puts lots of evolutionary pressure on the individuals.
- **Mutation**: Gaussian mutation has been preferred, as it has the advantage of having a smooth shape and being able to approximate the other two mutation types with a correctly set variance σ. The optimal value for σ has been searched empirically, and different functions have been discovered needing different values for sigma. The final σ has then been set to an intermediate value of 0.05.
- **Crossover**: as this operator depends heavily on the previous one, it has been chosen after determining the best mutation type, along with the best sigma. In the case of N Points Crossover, the value N has been chosen equal to 2, based on empirical results. Each type has been tested 6 times on every function, in order to determine the average score on each one and compare it with the others.

  For all tests two parents were used for the cross-over: bigger values were tried in early development, but the lack of noticeable gain resulted in using just two of them.

Having chosen the operator types and values, parameters exploration has been performed for the number of elite elements that should survive each generation (GGA) and how many children needs to be created at each epoch (SSGA).

4.2 DE

Starting from the results presented Pedersen's study [Ped10], fine-tuned parameters have been searched with a grid search methodology. Values tested are: for population 20, 50 and 100, F is taken between 0.1 and 0.7, Coin bias between 0.1 and 0.9, Base between random and best, finally couple of parents can be 1 or 2.

4.3 PSO

As in every time-step the whole population moves and has to be re-evaluated, the size of the swarm is a very important parameter of the algorithm, to not exceed in evaluations. For this reason it was tested as 50, 80 and 100. To be coherent with the model, inertia should always be kept under 1, while it was decided to use a half as the lowest value.

Both attractors were independently grid-searched between 0.1 and 0.9

4.4 CMA-ES

For this experiment different values were tested for λ, while μ was always kept as relative to it. As suggested in literature [HAA15, 3.8] [AH05], larger populations lead to better results for noisier and highly multimodal functions, but this also means more evaluations per epoch. Keeping this in mind, the number of samples λ for each epoch was kept between 10 and 40, while μ was varied between half and a fourth of λ.

5 Results

5.1 Parameter Tuning

Since the dimensionality of the grid searches is excessive to represent in its whole extension, only the mean and standard deviation of the 5 best performing combination of parameters will be included for each algorithm. Important to notice is that the statistics are over both trials and functions.

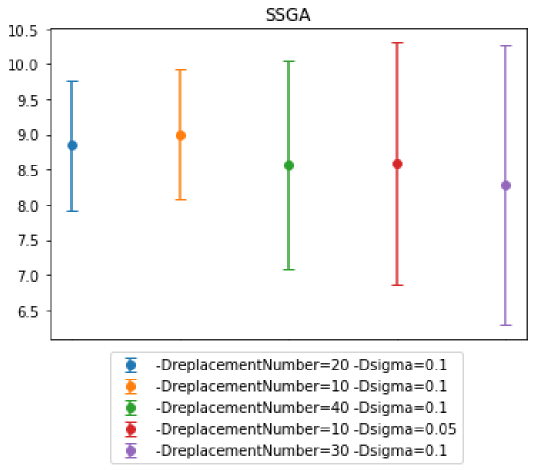

Figure 1: Performance of the best parameters combinations for the Steady-State Genetic Algorithm

SSGA The first observation about Figure 1 is a pleasant surprise for the efficacy of this method, which manages to reach an average of a full 9 over the 3 functions with the following parameters: *sigma = 0.1* and *replacement = 10%* of the population.
From a frequentist point of view there is not a sufficient difference between the means, keeping in mind the standard deviation, to declare one of the combinations as better (t-Test, $p > 0.05$), but a Bayesian approach to the test will elect the one with higher mean as the optimal choice.

GGA The generational version, seen in Figure 2, appears way less impressive, as its best 5 results don't even touch a fourth of the perfect score. Once again there is no statistical confidence in declaring a winner, but the choice for optimal parameters falls on: *elitism = 1*, *sigma = 0.05*.

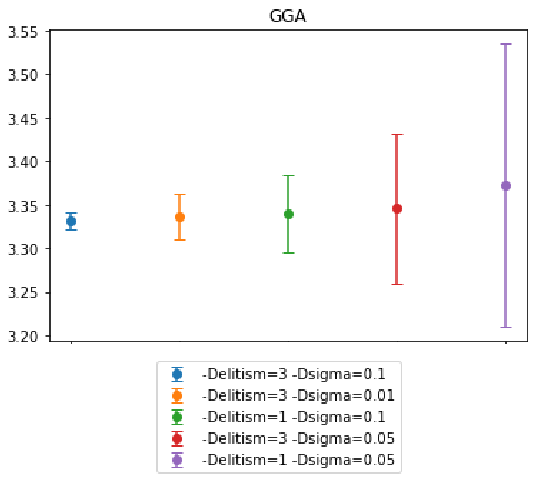

Figure 2: Performance of the best parameters combinations for the Generational Genetic Algorithm

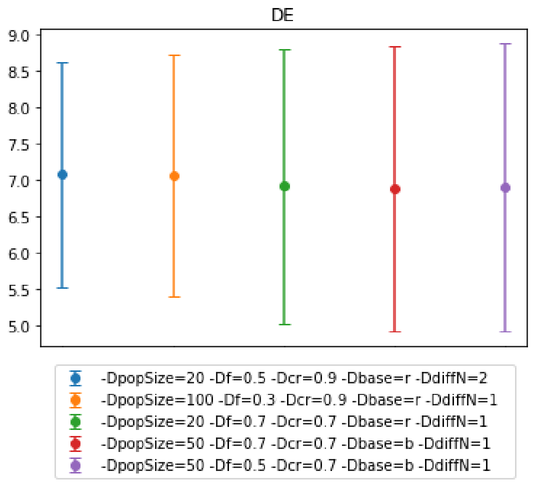

Figure 3: Performance of the best parameters combinations for the Differential Evolution algorithm

DE Figure 3 shows a huge standard deviation in each parameter combination, due to the disparity in performance between functions. This algorithm shines in the first two, simpler, functions and miserably fails in the Katsuura. Such variance makes the choice for an optimal parameter a coin-flip, so it has been decided to manually tweak the parameters to work optimally on the first 2 functions and discard the third. The output of this manual meta-optimization is: *population size = 50*, *F = 0.5*, *coin bias = 0.4*, *base = best* and *couples number = 1*.

PSO The Particle Swarm Optimization performs similarly to the differential Evolution, regarding average and standard deviation, see Figure 4. Once again the big variance is mainly due to the lack in performance on Katsuura. Since manual tuning didn't bring exceptional results for Schaffer's function either (having highly inconsistent scores), it

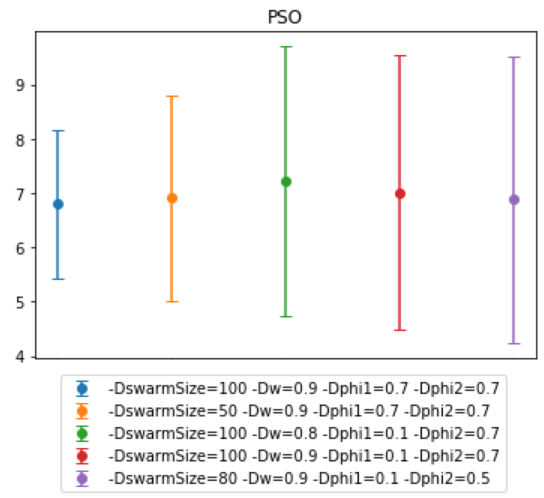

Figure 4: Performance of the best parameters combinations for the Particle Swarm Optimization

was opted to select the optimal value from the grid search, once again chosen as the one with highest average: *swarm size* = *100*, *inertia* = *0.8*, $\phi_1 = 0.1$ and $\phi_2 = 0.7$.

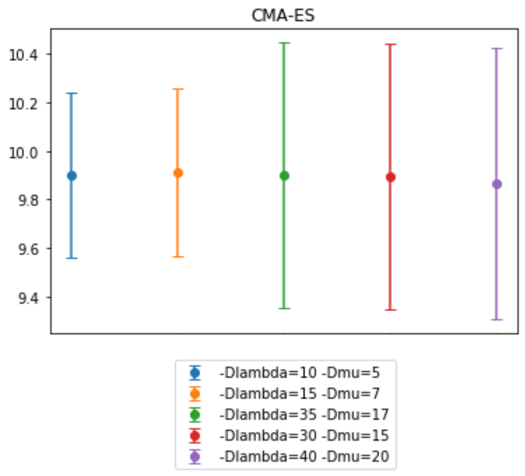

Figure 5: Performance of the best parameters combinations for the Covariance-Matrix Adaptation Evolutionary Strategy

CMA-ES From Figure 5 CMA-ES appears the strongest algorithm, having very low variance and all but perfect results. Furthermore, the parameters combination with higher average is also the one with smaller standard deviation and is therefore selected as the optimal choice: $\lambda = 15$ and $\mu = \lambda/2 = 7$. As in all previous tests there is not enough statistical confidence to actually declare one algorithms better than the others (Student's t-Test, $p - value > 0.05$) and the Bayesian approach is the only option.

5.2 Functions comparison

With the results from the previous study in mind a comparison between all optimal algorithms has been done, with score separated by function to optimize. For this experiment all time constraints were released, as the interest is in efficacy.

Scores

Figure 6 to 8 show the distribution of 10 runs for each pair algorithm-function. Each box and whisker element indicate the measurements for a specific algorithm. The box extends from the lower to upper quartile values of the data, with a line at the median. The whiskers extend from the box to show the range of the data. Flier points are those past the end of the whiskers.

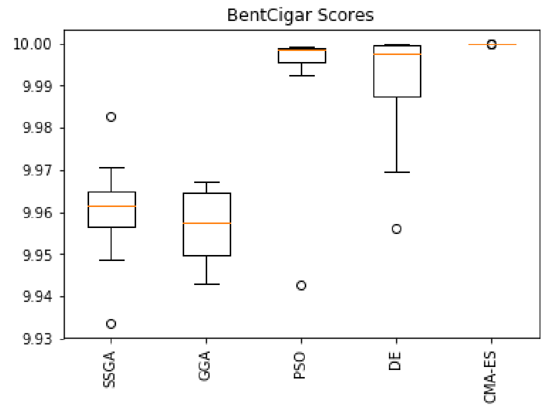

Figure 6: Distribution of scores for each algorithm on the Bent Cigar function

As the Bent Cigar is an easy function to tackle with evolutionary methods, all algorithms score over 99% of the maximum. Notably the median of 3 algorithms is better than 99.9% and CMA-ES seems to alway get a perfect outcome (Figure 6).
Overall, the algorithms with fine tuned parameters could be considered comparable in efficacy when tackling this problem. Especially PSO, DE and CMA-ES.

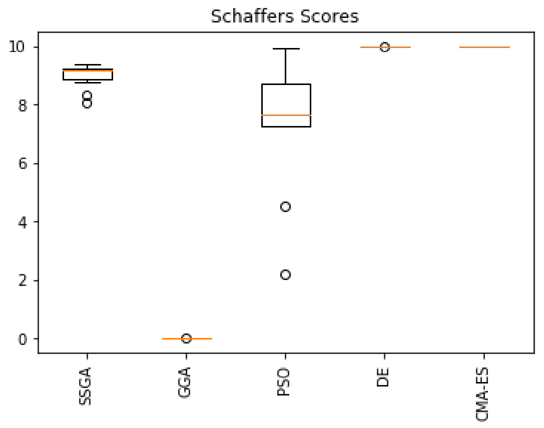

Figure 7: Distribution of scores for each algorithm on the Schaffer's function

In Figure 7 things already start to change, with three algorithms touching the perfect score and only DE and CMA-ES

in a consistent way. One of the Genetic Algorithm crashes to an abysmal average score of almost 0, while the other remains consistent around 90% of the optimum.
Apart from the aforementioned two best methods it's not possible to declare the algorithms comparable in optimizing this function.

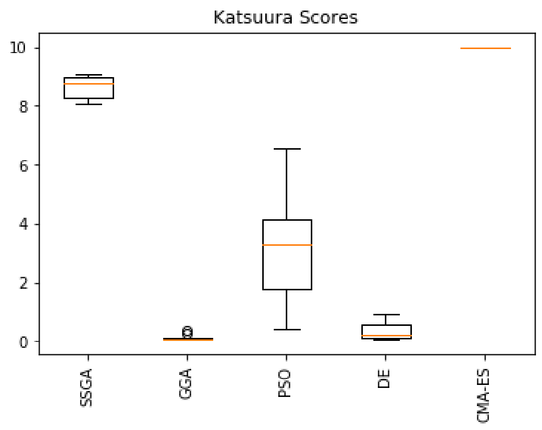

Figure 8: Distribution of scores for each algorithm on the Katsuura's function

For the most complex function the results are clearly incomparable between algorithms (Figure 8). CMA-ES is still reaching the perfect score with total consistency, SSGA maintains an average score close to 9 and all other performances drop. This was expected for the Differential Evolution implementation, as its parameters were impossible to tune for the first functions and Katsuura's together and manual tweaking was not enough to find a valid candidate.

Computational Times

Additionally a study of the time to run the algorithms was performed, to still gain an idea of each algorithm efficiency. The following data resulted: Figure 9 to 11, where the y axis represents the elapsed time in ms.

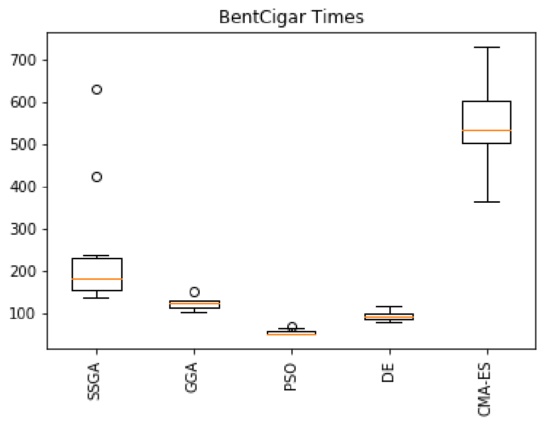

Figure 9: Distribution of times for each algorithm on the Bent Cigar function

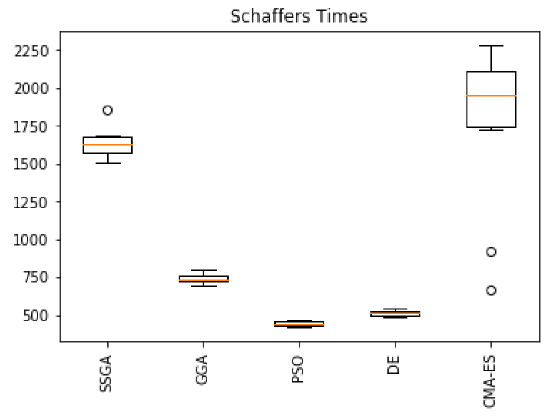

Figure 10: Distribution of times for each algorithm on the Schaffer's function

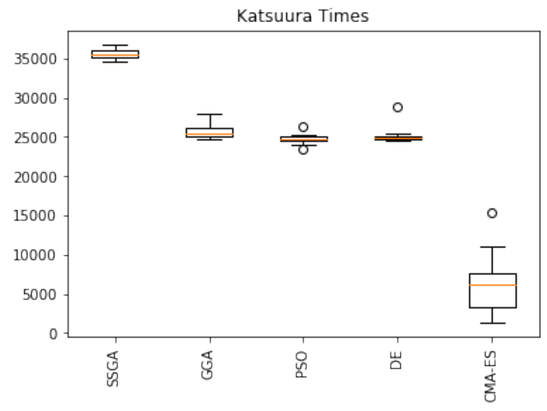

Figure 11: Distribution of times for each algorithm on the Katsuura's function

Only a brief analysis is taken over the computation time, as it is considered a secondary trait of the algorithms. Interesting is how CMA-ES takes the most time for simple functions and the least for the most complex, this consistency in computation time is probably the reason of its success and perfect scores, as its algorithm does not seem to be affected by search space complexity as the ones of the other strategies.

6 Conclusions

Even with an extensive grid search for parameter tuning the performances of the algorithms through functions are not comparable. To statistically test this results a t-Test was applied to the runs of each algorithm in each function. Since there was no homoscedasticity between the sample runs the Welch's version of the t-Test was used and in 1 the p-values are collected.

It is important to note how in all cases the statistic was positive, meaning that CMA-ES is always better than all others, even if not in a statistically significant manner. This is especially clear for the score confrontation between algorithms on the Katsuura's function optimization, where no

Algorithm	BentCigar	Schaffers	Katsuura
SSGA	4.95e-06	5.15e-05	1.97e-06
GGA	7.13e-08	1.62e-57	5.59e-19
PSO	**0.183**	0.0044	9.79e-07
DE	**0.056**	**0.152**	8.84e-15

Table 1: p-value of the Welch's t-Test of the result obtained by CMA-ES and other algorithms. In bold the statistically non-significative differences.

other strategy, despite the parameters fine-tuning, comes even statistically close to CMA-ES results.

In conclusion, all data from the experiments refutes then the initial hypothesis, as an algorithm clearly outperforms the others even when their parameters have been selected to excel in the trial functions. More importantly, it has to be noted that the number of problems tackled by the algorithms is too narrow to declare CMA-ES as the best algorithm at the moment, and more diverse testing functions should be tried. As a last consideration, the fact that CMA-ES required just two parameters to be tuned suggests that it could be a valid initial choice when dealing with an unknown problem, reducing the overall burden on the initial choice of algorithm.

References

[AH05] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In B. McKay et al., editors, *The 2005 IEEE International Congress on Evolutionary Computation (CEC'05)*, volume 2, pages 1769–1776, 2005.

[GD91] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1:69–93, 1991.

[HAA15] Nikolaus Hansen, Dirk V Arnold, and Anne Auger. Evolution strategies. In *Springer handbook of computational intelligence*, pages 871–898. Springer, 2015.

[HK04] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In X. Yao et al., editors, *Parallel Problem Solving from Nature PPSN VIII*, volume 3242 of *LNCS*, pages 282–291. Springer, 2004.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[Ped10] Magnus Erik Hvass Pedersen. Good parameters for differential evolution. *Magnus Erik Hvass Pedersen*, 49, 2010.

[PKB07] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[SP97] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[Whi94] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.