Evolutionary Computing

Arjan Bontsema Rink Stiekema Zelda Zeegers

October 20, 2017

1 Introduction

Evolutionary computation is a topic in computer science where abstract algorithms meet natures logics. When discussing mathematical or computer science related problems and solutions to these problems, it is easy to lose track of what we are trying to achieve. Algorithms can have many phases, functions and parameters that are needed to optimally solve a problem. This can make the programmers feel like they do not have enough RAM to keep all these aspects in line. Linking algorithms to concrete events that we see in nature can help us keep track of the problem and the solution at the same time. On top of that, it can help us find inspiration for new algorithms. Lastly, from time to time we find this algorithm that work stupidly well for a large amount of problems.

It would be ideal, if there was just one algorithm that can be applied to many problems but unfortunately there does not seem to be an one-size-fits-all solution. There may be algorithms that perform extremely well on one problem, but these almost always seem to perform only decent on other problems. In this project, we studied three different functions and programmed several algorithms that optimize the outcome of these functions. Our focus lies on analysing the impact of parameters to these algorithms. Next to this, we tried to categorize these algorithms as being specialist, all-round or unfit for these three functions.

In this study, three different functions will be optimized, namely the Katsuura function, the Bent Cigar function and Schaffers F7 function. Each function over the domain $[-5, 5]^{10}$ has a maximum value of 10. The goal is to find the location of the maximum within a fixed number of evaluations. Several algorithms will be considered to find this optimum. For each function we will try to answer two questions: what is the best evolutionary algorithm to evolve this function and what are the parameters that are these algorithms require.

2 Evolutionary Algorithms

2.1 Particle Swarm Optimization

In particle swarm optimization (PSO), each population member represents a point in space with a position and a velocity. Each individual can be represented by the triple $\langle \bar{x}_i, \bar{v}_i, \bar{p}_i \rangle$. During one generation (or epoch), each individual is replaced by $\langle \bar{x}'_i, \bar{v}'_i, \bar{p}'_i \rangle$, where

$$\begin{aligned}
\bar{x}'_i &= \bar{x}_i + \bar{v}_i \\
\bar{v}'_i &= w \cdot \bar{v}_i + \phi_1 U_1 \cdot (\bar{b}_i - \bar{x}_i) + \phi_2 U_2 \cdot (\bar{c} - \bar{x}_i) \\
\bar{b}'_i &= \begin{cases} \bar{x}'_i & \text{if } f(\bar{x}'_i) < f(\bar{b}_i) \\ \bar{b}_i & \text{otherwise} \end{cases}
\end{aligned}$$

where $\bar{c}$ denotes the populations global best. $U_1, U_2 \sim \mathrm{U}[0,1]$. The value of w can control exploration and exploitation. From most experiences, $\phi_1 = \phi_2$ gives best results [1].

Additionally, a maximum for velocity, v_{max} was preset as parameter. Each iteration, $v = \min(v, v_{max})$. To avoid particles to leave the search space $[-5,5]^{10}$, either mirroring or looping is considered. [2]

2.2 Differential Evolution

In this algorithm, each population member represents a position in the search space. Then during each iteration, for each member $\bar{x}_i$, a new competing candidate $\bar{v}_i$ is computed in the following way:

- Randomly select one allele J.
- If $i \neq J$, with probability CR, $v_i = a_i + F \times (b_i - c_i)$, where a_i, b_i, c_i are randomly drawn from the population, different from x_i. Else, $v_i = x_i$.
- $v_J = x_J$

$\bar{x}_i$ is replaced by $\bar{v}_i$ if a higher fitness is obtained (elitist selection). The algorithm is generalized by adding additional perturbation vectors $F \cdot (\bar{y} - \bar{z})$. It follows that *number of parents* $= 1 + 2 \times$ *number of pertubations*. The variants of Differential Evolution algorithms can be written as DE/a/b/c, where a is the base vector (rand or best), b is the number of perturbation vectors and c is the crossover scheme ("bin" = uniform crossover).

The new agent is based on a so called differential rate F, this influences the rate of averageness between all agents. If the differential rate is high, the values of the new genes will be somewhere in between all used agents and thus have a higher chance of being highly different of what is currently available in the population. When the differential rate is low, the values will be closer to one of the parents, and thus a value closer to the current population. In this sense, it influences how much the agents are changed every generation.

2.3 Custom Algorithm

Besides considering algorithms that have proven to be successful in other situations, we considered a custom model, consisting of several well-known components from evolutionary computation.

For parent selection, tournament selection was used. For a fixed number of rounds, a random subset of the population is drawn. On this subset, elitist selection was applied (best individuals become parent).

For recombination, arithmetic crossover was used, so that each pair of parents produce two children.

$$\begin{aligned} \text{child}_{j1} &= \alpha \cdot \text{parent}_{1j} + (1-\alpha) \cdot \text{parent}_{2j} \\ \text{child}_{j2} &= (1-\alpha) \cdot \text{parent}_{1j} + \alpha \cdot \text{parent}_{2j} \end{aligned}$$

where α is either random $\mathrm{U}[0,1]$ or deterministic. Furthermore, random mutation was added to the algorithm, so that one coordinate of each child was repositioned randomly.

At last, (λ, μ)-selection or the "comma strategy" was used to select survivors. This means that the new population is only selected from the children. Parents will never survive a generation. Note that λ = number of parents = tournament winners $\times$ tournament rounds and μ = number of parents = tournament rounds $\times$ tournament winners.

[1] Bansal JC, Singh PK, Saraswat M, Verma A, Jadon SS, Abraham A (2011) Inertia weight strategies in particle swarm optimization. In: Proceedings of third world congress on nature and biologically inspired computing (NaBIC-2011)

[2] Helwig, S., & Wanka, R. (2007, April). Particle swarm optimization in high-dimensional bounded search spaces. In Swarm Intelligence Symposium, 2007. SIS 2007. IEEE (pp. 198-205). IEEE.

3 Parameter Tuning

The parameters of an algorithm can have a major impact on the performance of the algorithm. Determining these optimal values is called parameter tuning. This can be a very exhaustive process, certainly when the number of parameters is high and they have a broad range. Still, the time spend on tuning can be very viable and make or break the algorithm. We used a method called Random Search for this purpose.

3.1 Random Search

In principle, random search is a very simple process; simply repeatedly assigning random values to the parameters and recording the score is all there is to it. Yet it is proven to be a very strong method in evaluating parameters and usually works better than the seemingly more methodical grid search.[3] In grid search, a list of possible parameters is arranged and then all combinations of these parameters are tested. However, with enough iterations, random search provides a better cover of the parameter search space. After the exhaustive search, the scores are analysed and the impact of the different parameters is examined. Hopefully an optimal combination of parameters can be found based on this data.

3.2 Adaptive parameters

Next to parameter tuning, there are additional parameter control techniques. Since parameters may not be fit to use throughout the entire run, it can be favourable to lower the differential rate from differential evolution as we get closer to an optimum. We hard coded this into our differential evolution algorithm. Once the score is higher than 9.9, we decrease the differential rate in order to take smaller steps towards an optimum. The concept is nothing new and it has been done before, even better than in our implementations. A perfect example of this is a paper by Greiner et al. on self-adapting parameters for differential evolution. This is a way more flexible technique, but the time did not allow it for us to implement this. [4]

4 Other techniques

4.1 Distributed Island Model

The Distributed Island Model is an implicit approach to preserve diversity and avoid premature convergence. The idea is to run multiple populations in parallel. After a fixed number of generations (usually 25-150, or adaptively), individuals are exchanged between populations. Usually 2-5 individuals are exchanged, but the optimal value depends on population size. Since the populations develop themselves individually, they search a broad part of the solution space, but the Distributed Island model prevents them from getting stuck in a local optima, by giving populations a new impulse.

4.2 Initial Dispersion

A common choice for the initial population, would be to choose $(x_1, ..., x_\mu)$ uniformly over the search space $[-5, 5]^{10}$. However, during experiments, it might be that the algorithm is not consistent, namely, not for all seeds it performs optimally. Especially for small populations, a part of the deviation may be caused by an unlucky initialization. Although the algorithm might be designed such that it will discover

[3] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb), 281-305

[4] Brest, J., Greiner, S., Boskovic, B., Mernik, M., & Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE transactions on evolutionary computation, 10(6), 646-657.

the whole search space, algorithms such as DE or PSO may have some dependency on the initial state. A somewhat hacky technique to improve the results by choosing a dispersed initial population, is by computing the dispersion: the sum of all distances within population

$$D((x_1, ..., x_\mu)) = \sum_{i=1}^{\mu} \sum_{j \neq i} \|x_i - x_j\|_2$$

To obtain a dispersed initial population, re-initiate the population until $D((x_1, ..., x_\mu)) > \beta$. The threshold β is set manually. In this way, concentrated populations are retained.

5 Results

Figure 1: Parameter tuning results

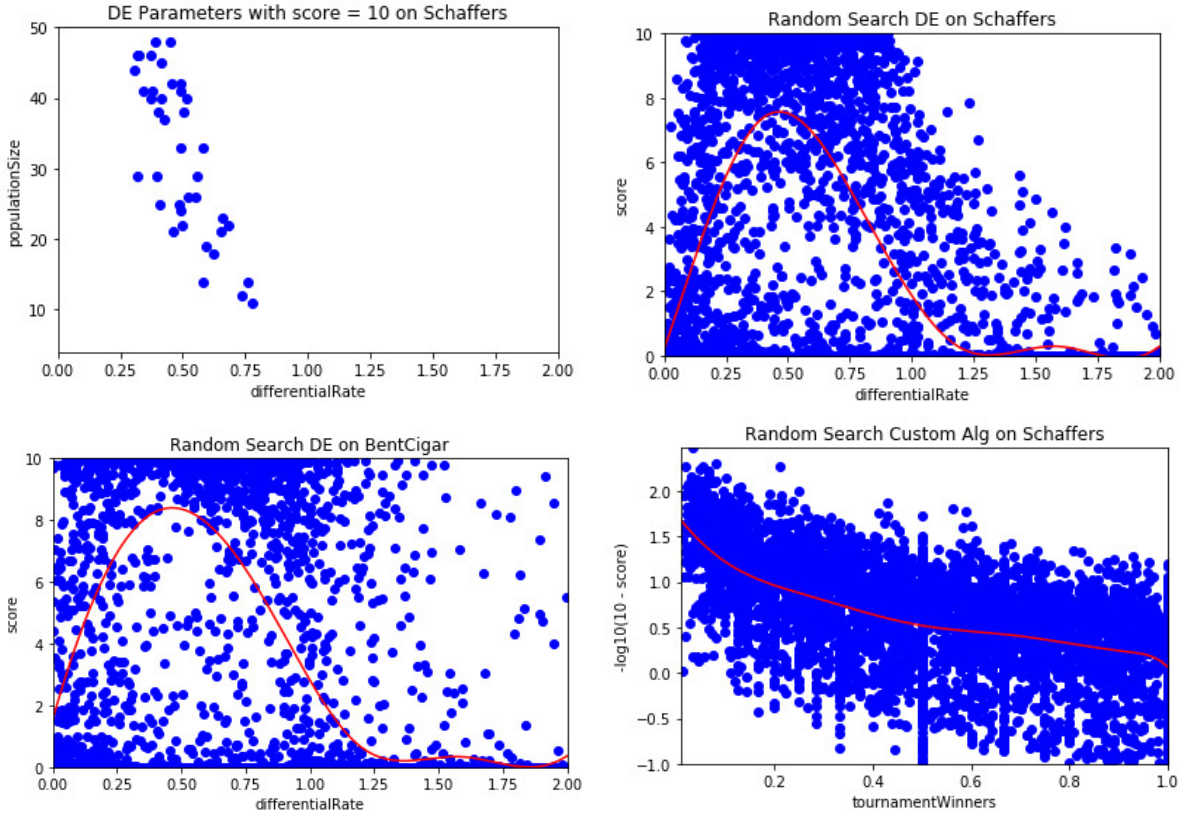

Figure 1 shows the results of the random search parameter tuning. The red lines show the mean scores, of which we chose the highest point for our parameters. From these graphs, it becomes clear that there is a rather small section of the differential rate that performs well. This holds both for Schaffers as for Bent Cigar. The optimal parameter settings for DE, as one can see in table 1, are approximately the same for Schaffers and Bent Cigar.

As we can see in Table 1, the best score is achieved by Differential Evolution on Schaffer's function. This is just the basic DE algorithm, without any additions to it. Interestingly enough, Schaffers function does not benefit from any additional techniques. However, for the Bent Cigar function, a more complex combination of techniques was needed to get the best score. To show the influence of the techniques, we use the logarithmic distance the score has to 10; order= $-\log_{10}(10 - f(x))$. The outcome of this formula gives the number of decimal 9s the score has. For example, 9.99 would give an order of 2. For

	Bent Cigar	Schaffers	Katsuura
Best Average Score	9.99999782072	10.0	1.28046009899
Submission Score	9.99995568369	10.0	0.06211056920
Best Algorithm	DE/rand/2/bin $CR = 0.73$, $F = 0.45$ $\mu = 20$, Island Model	DE/rand/2/bin $CR = 0.7$, $F = 0.5$ $\mu = 30$	DE/rand/2/bin $CR = 0.7$, $F = 0.5$ $\mu = 30$

Table 1: Scores of algorithms

Bent Cigar, the order of DE was 4.3. Using adaptive parameters, this improved to 4.6 and using islands eventually grew the order to 5.3. For the Island Model implementation, 2 islands were created. After each 70 generations, 1 individual was exchanged between islands.

To conclude, we categorize DE as being an all-round algorithm, as it performs well on two out of the three functions. Particle Swarm Optimization is a specialist, as it only performs well on the Bent Cigar function (a score of 9.87714205702 against 2.91068082743 and 1.04152897869 for Schaffers and Katsuura respectively). Lastly, the custom algorithm can be seen as all-round since it performs rather well on both Bent Cigar as Schaffers. Its score for Katsuura is close to DE (1.2337363868).

6 Discussion

The reason that we find a high variability of highest score values using the same optimization algorithms is probably that the analysed functions produce highly different landscapes. Therefore we were not able to get decent results for Katsuura, with a landscape containing a large amount of very narrow spikes that mislead the algorithms. Optimizing Katsuura function is a multimodal problem, and for this type of problems, very specific algorithms should work better. PSO and DE are designed for unimodal problems; therefore it is no surprise that these algorithms do not yield the desired result for Katsuura. Also it can be reasoned that the improvement by additional techniques and specific parameters has to do with the shape of the function. Where Schaffers seems to have a rather big area of parameters that deliver a global optimum, Bent Cigar needs a more precise approach. This also means that it benefits from more diversity (islands) and a more sophisticated parameter usage (adaptive parameters).

It is expected that the custom model might do much better when trying more (combinations of) components. During this study, only a single mutation method was implemented for example. It is expected that other mutation methods, such as Gaussian mutation, might also work quite well on these problems. Also other parent selection methods might work better. Tournament selection is a quite aggressive method to find good candidates and it requires a lot of evaluations, which was restricted for this project. Fitness proportionate selection might be a good alternative.

During the project, an implementation of Simulated Annealing was written. Although this implementation did not work very well, we determined that it could be used as kick-starter for another algorithm to globally search for a good starting point, then using PSO or DE to focus on the local optima. [5] Combining different evolutionary algorithms can largely improve the performance. Unfortunately, there was not enough time for us to apply this and possibly get a better score for the Katsuura function. The focus of our study was mainly on Bent Cigar and Schaffers. Another method to solve multi-modal problems is by combining DE with local search algorithms. [6]

[5] Van Laarhoven, P. J., & Aarts, E. H. (1987). Simulated annealing. In Simulated annealing: Theory and applications (pp. 7-15). Springer Netherlands.

[6] Rnkknen, J. (2009). Continuous Multi-modal Global Optimization with Differential Evolution-Based Methods. Lappeenranta University of Technology.