Shark Smell Optimization

Jochem Ram (S2040328), Jerry Schonenberg (S2041022), Wouter van Tol (S2041340), Thomas Wink (S1705148)

Leiden Institute of Advanced Computer Science, The Netherlands

Abstract. In this paper an optimization algorithm is presented which is based on the way sharks find their pray. This algorithm, which was first proposed by Oveis Abedinia, Nima Amjady and Ali Ghasemi [1], uses gradient-based general searches and a set of local searches. We present the pseudocode along with a list of all used functions and deviations from the original paper. We show an ECDF and ERT plot in which we compare SSO with BIPOP [3] in 20 dimensions. We see that our algorithm performs worse than BIPOP on almost every function, and also performs worse than the algorithm in the original paper. This is because the original paper did not specify a lot of details causing some difficulties with the implementation.

1 Introduction

For as long as humans exist, we have looked at nature for inspiration. Most inventions humans have made come from investigating natural phenomena and trying to reproduce what we see. This is still the case nowadays, where we think we know almost everything. It has proven useful to examine the way processes happen in nature and to try to implement this in problems we have ourselves. One example is to look at the way a shark, one of the best hunters in the sea, finds a pray in an efficient way. This can be used in an optimization problem to quickly find the best solution to a complex problem in a high dimensional solution space.
In this paper we present the implementation and results of the shark-smell-optimization (SSO) [1] algorithm. This will be done using IOHprofiler [2] by comparing the performance of the algorithm for different objective functions.

2 Algorithm Description

This algorithm uses an individual (the shark) to find the optimum of the objective function (the wounded fish) by following the gradient (the blood particles) and performing local searches (rotational searches of the shark) around the individual's position at each stage. This is repeated for each individual until stage k_{max} is reached. The number of individuals and stages can be decided by the user.
A single iteration of the algorithm will first compute the calculation of the movement of the individuals in each dimension of the search space. Then a new position is calculated for every position. From this position a random local search will be done, where the best point from this search will be compared with the position from the the first search. The best one of these will be the new starting position for this individual for the next iteration.

2.1 Comparison with standard algorithms

The SSO algorithm can be best compared to simulated annealing. This algorithm is based on the concept of heat treatment in metals, in which a hot metal is slowly heated to let the atoms slowly

settle to minimal energy states. This is used in optimization by perturbing each point randomly and choosing whether to move to that point or not. If the objective function in that point is higher, it will always move. If the objective function is lower, the chance of movement is calculated with the difference in the objective function values.
This algorithm is most similar to SSO since, unlike almost all natural optimization algorithms, these two algorithms are not based on mutation or swarm behavior. They are based on the searches of individual candidate solutions without any communication between the individuals and introduce randomness in their search pattern. SSO does this by choosing random values around the solution candidate and evaluating all of them at once, while simulated annealing does this by choosing a random value around the solution candidate and evaluating immediately. For simulated annealing this leads to a higher chance of reaching a local optimum, but this is mitigated by offering the chance of a restart where an individual is placed back to a previously found significantly better solution candidate.
However, where the SSO uses the gradient to find the movement direction of the search point, simulated annealing uses difference in objective function values in two points. Also, SSO can not move to a position with a lower objective function value while this is one of the strengths of simulated annealing.

3 Pseudo-code

We followed the following notation convention. All user assigned variables are mentioned in Appendix A.

3.1 Variable explanation

- N: The number of individuals in the set/array. This number should be at least 1.
- M: The dimensionality of the search space.
- $k_{\max}$: Maximal amount of stages. This number should be at least 1.
- O: Number of rotational candidate solutions.
- n, m, k and o: Administrative array counters for N, M, k_{max} and O respectively.
- X_n^k: The solution candidate $\mathbb{R}^M$ of an individual at a stage.
- V_n^k: The calculated movement in $\mathbb{R}^M$ for an individual at a stage.
- $v_{n,m}^k$: The calculated movement in a direction for an individual at a stage.
- Y_n^k: The provisional solution candidate in $\mathbb{R}^M$ after movement of an individual at a stage.
- Z_n^k: Set of O provisional solution candidates in $\mathbb{R}^M$ of an individual at a stage after local search.
- x_m: A variable in the search space.
- $x_m^{\min}$, $x_m^{\max}$: The minimal and maximal values in the search space for a variable.
- $f(\mathbf{x})$: Objective function value of $\mathbf{x}$ ($f : \mathbb{R}^M \to \mathbb{R}$).

- $\leftarrow$: Assignment operator.
- $\boldsymbol{\alpha}$: User assigned vector in $\mathbb{R}^{k_{max}}$. Represents the inertia coefficient for each stage. These numbers should be in the range [0, 1]. Together with $\boldsymbol{\eta}$ it should be 1.
- $\boldsymbol{\beta}$: User assigned vector in $\mathbb{R}^{k_{max}}$. Represents the velocity limiter ratio for each stage. This number should be higher than 0.
- $\boldsymbol{\eta}$: User assigned vector in $\mathbb{R}^{k_{max}}$. Limits the gradient of the objective function for each stage. These numbers should be in the range [0, 1]. Together with $\boldsymbol{\alpha}$ it should be 1.
- c: User assigned constant used in calculating the initial velocity. This number should at least be 1.
- $r1$ and $r2$: Random uniform value between 0 and 1 to give the algorithm more stochastic search value.
- $R3$: Vector in $\mathbb{R}^M$ with random uniform values between -1 and 1 to give the algorithm more stochastic search value.
- $\arg\max()$: Takes the variable with the highest objective function value.
- $\arg\max_{a=1}^{A}()$: Iterates over a and takes the variable with the highest objective function value.
- Further notation convention is based on mathematics.

3.2 Used functions

1) $v_{n,m}^k \leftarrow \eta_k \cdot r1 \cdot \frac{\partial f(x)}{\partial x_m}\Big|_{X_n^k} + \alpha_k \cdot r2 \cdot v_{n,m}^{k-1}$

This function is the calculation of movement in dimension m for individual n in stage k. In the first term of the calculation we take the partial derivative of the objective function to x_m, and then multiply it with a velocity limiter constant and a random value for extra stochastic search value. In the second term we take the movement from the last stage in this dimension for this individual and multiply it with a random value and an inertia constant. This function is used in line 17.

2) $\left|v_{n,m}^k\right| > \left|\beta_k \cdot v_{n,m}^{k-1}\right|$

This line limits the acceleration of the movement of an individual in dimension m. This is done by comparing the new velocity to the velocity in the previous stage multiplied with a velocity limiter constant. This function is used in line 18.

3) $Y_n^{k+1} \leftarrow X_n^k + V_n^k$

Assignment of provisional new location based on previous location and movement. This function is used in line 24.

4) $Z_n^{k+1,o} \leftarrow Y_n^{k+1} + R3 \cdot V_n^k$

This function finds O new solution candidates in a local search in the vicinity of Y_n^{k+1}. This is done with the provisional new position Y, and the velocity of this stage multiplied with a random vector R3. This function is used in line 28.

5) $X_n^{k+1} \leftarrow \arg\max(\arg\max_{o=1}^{O}(f(Z_n^{k+1,o})), f(Y_n^{k+1}))$

This function finds the best new location by comparing the provisional new location with the solution candidates from the local search, and testing their objective function value. This function is used in line 32.

3.3 Pseudocode

Algorithm 1 Shark Smell Optimization

$N \leftarrow$ User assigned ▷ Initialize
$k_{\max} \leftarrow$ User assigned
$k \leftarrow 1$
$O \leftarrow$ User assigned
$\boldsymbol{\alpha} \leftarrow$ User assigned
$\boldsymbol{\beta} \leftarrow$ User assigned
$\boldsymbol{\eta} \leftarrow$ User assigned
for $n = 1 \rightarrow n$ **do**
  for $m = 1 \rightarrow m$ **do**
    $x_{n,m}^1 \leftarrow \boldsymbol{\mathcal{U}}(x_m^{\min}, x_m^{\max})$
    $V_{n,m}^0 \leftarrow \frac{x_m^{\max} - x_m^{\min}}{c}$
  end for
end for
while $k \leq k_{max}$ **do**
  for $n = 1 \rightarrow N$ **do** ▷ Calculation of movement per individual
    for $m = 1 \rightarrow M$ **do**
      $v_{n,m}^k \leftarrow \eta_k \cdot r1 \cdot \frac{\partial f(x)}{\partial x_m}\Big|_{X_n^k} + \alpha_k \cdot r2 \cdot v_{n,m}^{k-1}$
      if $\left|v_{n,m}^k\right| > \left|\beta_k \cdot v_{n,m}^{k-1}\right|$ **then**
        $v_{n,m}^k \leftarrow \beta_k \cdot v_{n,m}^{k-1}$
      end if
    end for
  end for
  for $n = 1 \rightarrow N$ **do** ▷ Provisional assignment of new position per individual
    $Y_n^{k+1} \leftarrow X_n^k + V_n^k$
  end for
  for $n = 1 \rightarrow N$ **do** ▷ Random local search from provisional position per individual
    for $o = 1 \rightarrow O$ **do**
      $Z_n^{k+1,o} \leftarrow Y_n^{k+1} + R3 \cdot V_n^k$ ▷ Deviation
    end for
  end for
  for $n = 1 \rightarrow N$ **do** ▷ Definitive assignment of new position per individual
    $X_n^{k+1} \leftarrow \max(\max_{o=1}^{O}(f(Z_n^{k+1,o})), f(Y_n^{k+1}))$
  end for
  $k \leftarrow k + 1$ ▷ Administration
end while

4 Assumptions

For this algorithm we assumed that the objective function should be maximized.
On line 28 of the pseudocode we deviated from the original paper. In the original paper this line was:
$Z_n^{k+1,o} \leftarrow Y_n^{k+1} + R3 \cdot Y_n^{k+1}$
We changed the second Y to a V:
$Z_n^{k+1,o} \leftarrow Y_n^{k+1} + R3 \cdot V_n^{k+1}$
We did this as we believe that the searchspace for the rotational movement should be related to the velocity, and not to the current location. Otherwise we would be placing the random local search points on a line through the origin, which is arbitrary.
On this line we also changed $R3$ to be a vector instead of a number, since this means the rotational search is not just in the direction of the movement but also in other directions.
Since the velocity is based on the previous velocity, and can be at most 4 times as large, we need an initial velocity not equal to 0 or else every new velocity will also be 0. The paper has not mentioned which value they used. We assume that it should be a small number compared to the searchspace, so we chose $V_{n,m}^0 = (x_m^{\max} - x_m^{\min})/c$.
Furthermore, many details were not explained in the original paper such as the chosen values for N, $k_{\max}$, O, or the implementation of the partial derivative. We empirically chose the highest numbers for the constants for which the algorithm still computed quickly enough.

4.1 Problems with derivative implementation

In the paper, the velocity is linearly equated to the derivative. This is a problem since this would mean that for a function with range 10 and function values in the millions, such as the second function with which we test the algorithm, the derivative will be in the millions and thus the velocity as well. We should, however, be changing our position with a value between 0 and 5. Conversely, if the range is in the millions and the function ranges between 1.0 and 2.0, the derivative is too low for the individual to reach the optimum within a reasonable amount of stages. Therefore the derivative should somehow be scaled to the range of the search space.
Also, for a first order approximation of a partial derivative we take the value at a point and the value at that point plus a small distance d. How small this distance is, is very important for the speed and accuracy of the algorithm but was not explained by the paper. This value should also gradually become lower as the individual approaches the optimum, such that the evaluation point does not overshoot the optimum.

5 Experiments

To test the algorithm we use IOHexperimenter for benchmarking and IOHanalyzer to process the data. These are both contained in the tool IOHprofiler [2].
The benchmarking with IOHexperimenter is done across 24 functions with each 5 instances and 3 runs per instance. The tests are done in 5 dimensions and 20 dimensions. The constants we used are shown in table 1.

Constant	Value
N	50
k_{max}	100
O	12
α	0.1
β	4
η	0.9
c	100

Table 1: User assigned constants used in the algorithm

As for the implementation of the derivative, we did not scale the calculated velocity as to compensate for possible extreme values for the derivative. Instead we clamped the position of each individual within the search space and the overshoot of the derivative would very slowly be negated by the inertia term in line 17 of the pseudocode. Since this is not an optimal strategy the algorithm will most likely not perform as well as the original paper suggested [1].
We have tested the algorithm for both $d = (x_m^{\max} - x_m^{\min})/1000$ and $d = (x_m^{\max} - x_m^{\min})/1000000$. The difference between these methods was negligible so we decided for the second option.

To see how well the algorithm performed, we will compare it to BIPOP-CMA-ES [3] and create an Emperical Cumulative Distribution Function(ECDF)-plot for both these algorithms across all functions using the IOHanalyzer. This plot will show the performance of the algorithm compared to other algorithms in one glance.
To see the specific points at which the algorithm performs good or bad we will also make an Estimated Run Time-plot for each function.

5.1 Original experiments

The writers of the original paper did five experiments. In three of these experiments they compared SSO against the 32 most used algorithms in engineering applications. In these comparisons, the algorithm was the best, or equal to the best, in nearly every case. Only on one point did another algorithm perform better. Moreover, SSO was the fastest in each of these experiments.
In the other experiment they compared SSO to three of the best performing algorithms. They did this by testing SSO on a testbed in 10- and 30-dimensional environments with 28 test functions. 5 of these functions are unimodal functions, 15 are multimodal functions and 8 are composition functions. In these two experiments, SSO performed significantly better.
In the final experiment they performed, they put SSO to the test on the real-world problem of PID-controller optimization. Here it outperformed all algorithms on two of the main objective functions in this field. SSO also had the least over- and undershoot in finding the PID-controller target value.

6 Results

After generating the data with the IOHexperimenter it was analysed by the IOHanalyzer.

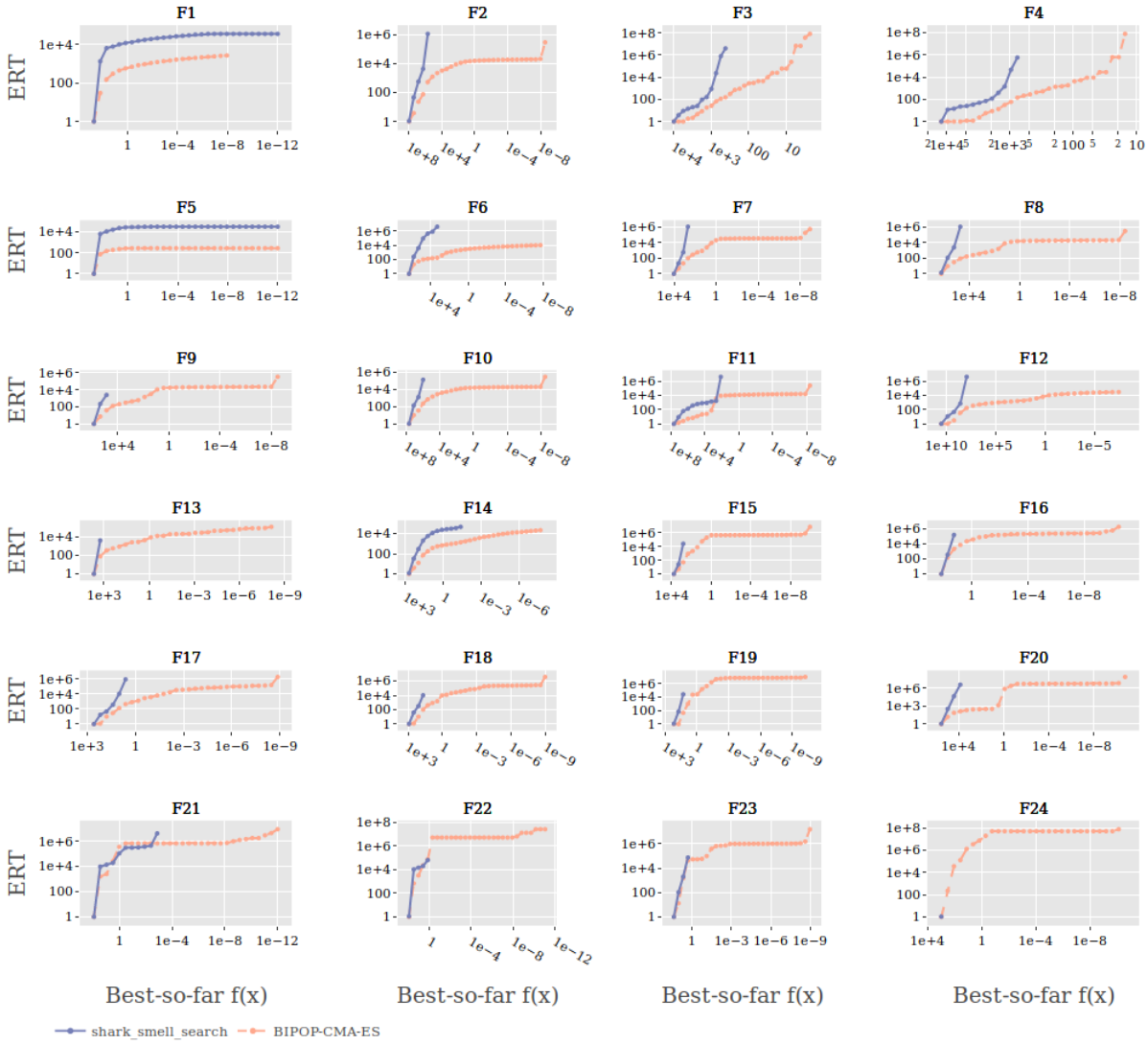

Fig. 1: Estimated Run Time for 20 dimensions

In figure 1 we see the performance of the SSO in blue and the BIPOP algorithm in orange. On the x-axes the deviation from the best objective function value reached is shown, and the y-axes show the estimated runtime needed to reach this value. This means that the lower the line stays, the quicker the algorithm reaches an optimum. Also, the further right the line reaches, the closer the algorithm comes to the optimum.
It is clear that the SSO algorithm performs worse than the BIPOP algorithm on almost every function. However, SSO does quite well on f1, where it finds a better optimum, and f21, where it

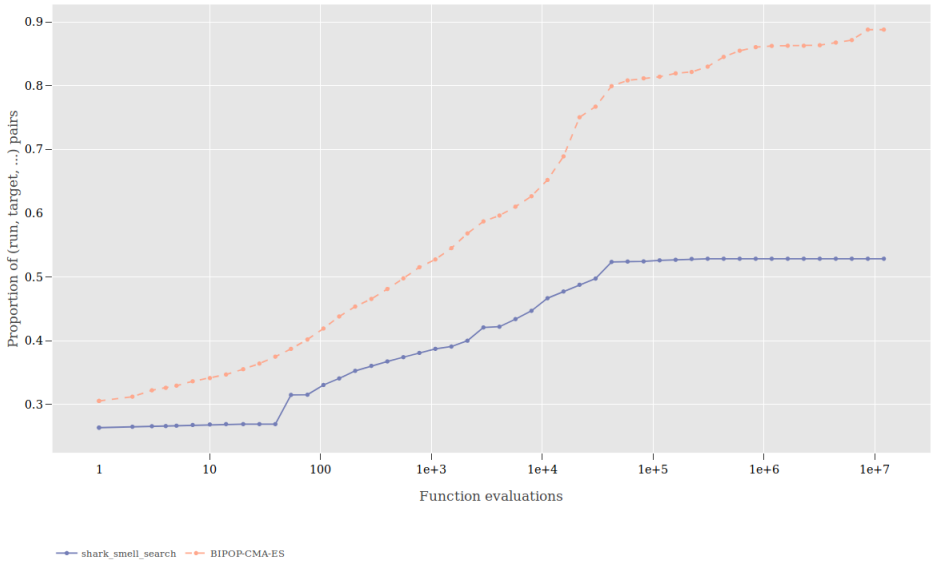

Fig. 2: Empirical Cumulative Distribution Function for 20 dimensions

finds the local area of the optimum quicker. In f24 SSO does not even find a value other than the initialisation value, which is why SSO only has 1 point in this graph.

To see how well the algorithm performs in general, we look at the ECDF-plot in figure 2. In this figure the x-axis shows the amount of evaluations and y-axis shows what proportion of the runs reached the target value for this amount of evaluations. Therefore a steeper slope of the curve means that the algorithm performs better. Since the SSO line is below the BIPOP line on every point, SSO performs worse than BIPOP in general.
Both figures are shown for 20 dimensions, since the figures created for 5 dimensions are very similar and do not add information.

7 Conclusion

The original paper showed that the SSO algorithm performed better than all other algorithms it was compared to and for all functions that were tested. Since our algorithm performs much worse than BIPOP, even though the original paper did not compare SSO with BIPOP, we most likely implemented the algorithm in a different way which performs worse. The most obvious fault is the fact that we need to clamp the positions in the range to make sure the algorithm does not walk out of the searchspace. This most likely means that either the implementation of the algorithm has a fault or the scaling of the derivative to the velocity should have been implemented.
Another explanation for the difference in performance between the original paper and our implementation is that the original paper has better results than they should have. This could be because of the assumption we made for the calculation of the provisional position 4. If the original

paper did not contain a mistake but did in fact implement this and the function with which they tested their algorithm had their optimum in the origin, each rotational movement would find points between the individuals position and the origin, and therefore get closer to the origin for every rotational run. Since the origin is arbitrary, this would mean that the objective functions are chosen poorly and the algorithm accidentally finds the optimum because of this error.
We found that most parts of the algorithm were quite easy to implement, but it was difficult to implement the partial derivative and work with the size of the 3 and 4 dimensional arrays. We also found that it was difficult to find the right variables to use as there were some inconsistencies in the original paper with variable names. The algorithm itself is quite original as it is one of the only algorithms that we know of that does not use mutation or uses swarm mechanics. However, we do not think it is revolutionary because the concept is quite simple and it does not really negate problems with local minimums.

References

1. Abedinia, Oveis, A.N., Ghasemi, A.: A new metaheuristic algorithm based on shark smell optimization. In: Wiley Periodicals. vol. 21, No. 5, pp. 97–116 (2014)
2. Doerr, C., Wang, H., Ye, F., van Rijn, S., Bäck, T.: Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. CoRR **abs/1810.05281** (2018), http://arxiv.org/abs/1810.05281
3. Hansen, N., Auger, A., Brockhoff, D., Tusar, D., Tusar, T.: COCO: performance assessment. CoRR **abs/1605.03560** (2016), http://arxiv.org/abs/1605.03560