Particle Swarm Optimization for solving Single Objective Bound Constrained Numerical Optimization Problems

George Stoica
Faculty of Computer Science
"Alexandru Ioan Cuza" University
Iași, Romania
george.stoica@info.uaic.ro

Amihaesei Sergiu
Faculty of Computer Science
"Alexandru Ioan Cuza" University
Iași, Romania
amihaeseisergiu@gmail.com

***Abstract*—Single objective bound constrained problems represent a well studied field due to numerous real-world applications and are still a challenging task. This paper proposes a solution using Particle Swarm Optimization (PSO) and studies the improvement of this algorithm compared to the standard Genetic Algorithm (GA) on benchmark functions from the CEC 2022 Single Objective Bound Constrained Numerical Optimization problems. Choosing PSO instead of GA brings better on this set of problems, while using a caching mechanism and applying further optimizations helps improve the performance of the algorithm even more.**

***Index Terms*—optimization, nature inspired methods, genetic algorithms, particle swarm optimization, CEC 2022**

I. Introduction

Many optimization problems can be found in real-life situations and some of them can be solved by either minimizing or maximizing one or more objective functions. Deterministic algorithms are usually not able to solve such problems, due to their computational complexity. Population based methods however are able to give good results due to their stochastic approach and by mimicking an optimization process inspired from nature.

Genetic Algorithms (GAs) are one of the often ignored candidates for solving these problems. In a previous paper a method to improve the performance of the standard GA on the CEC 2022 problems was studied and this paper will refer to those results.

Particle Swarm Optimization (PSO) algorithm is a metaheuristic inspired by animals' social behaviour, like the movement of organisms in a bird flock or a fish school. It optimizes a problem in an iterative way, by having a set (swarm) of candidate solutions (particles) that are moved around in the search-space using simple mathematical formulae. The movement of particles is influenced by different factors that help guide the swarm towards the best solutions, namely the particle's and the swarm's best known positions.

In this paper the feasibility of PSO is addressed along with several modifications in order to improve its results. Applying a PSO with the same parameters on a wide range of functions is known to give average results according to the No Free Lunch theorem [14]. Even so, it is possible to optimize an algorithm to solve specific classes of problems, such as the objective functions from the CEC 2022 competition.

The rest of the paper is organized as follows. In Section II the CEC 2022 competition and the benchmark functions are introduced, while in Section III previous submissions are analyzed. Section IV consists of a description of the improvements to the standard PSO algorithm. The experimental results and the setup is described in Section V and finally, the conclusion in Section VI.

II. Competition

The CEC 2022 competition consists of 12 minimization problems that are to be treated as black-box problems. The number of dimensions are either 10 or 20, and the functions to be minimized have their global optimum shifted and rotated. For 10 dimensions, the maximum number of function evaluations (maxFES) is 200'000, while for 20 dimensions maxFES is 1'000'000. The search range is between $[-100, 100]^D$.

The code used in this paper uses a personal implementation of all the functions from the competition, validated against the official implementation through testing.

One point of interest is that in the implementation provided by the organizers of the competition, one function (Shifted and full Rotated Expanded Schaffer's f6 Function) is wrong.

```python
def schaffer_F7_func(x, nx, Os, Mr, s_flag, r_flag):
    f = 0.0
    z = sr_func(x, nx, Os, Mr, 1.0, s_flag, r_flag)
    for i in range(nx - 1):
        z[i] = pow(y[i] * y[i] + y[i + 1] * y[i +
        ↪  1], 0.5)
        tmp = np.sin(50.0 * pow(z[i], 0.2))
        f += pow(z[i], 0.5) + pow(z[i], 0.5) * tmp *
        ↪  tmp
    f = f * f / (nx - 1) / (nx - 1)
    return f
```

As it can be seen, after this function is shifted and rotated, the result of these operations is in z, which is subsequently overwritten by values from y, a global vector. This global vector y is touched and modified in many places in the provided code, making difficult to reason locally about it's state, however in this particular place it happens that y contains the result of the shift (but not rotate) operation for x. As a consequence, this code provides a way to compute a shifted (but not rotated) Expanded Schaffer's f6 Function.

Moreover, the second hybrid function which uses Schaffer's f6 Function also suffers due to the usage of the global vector y. The second hybrid function is a function created by partitioning all dimensions of the vector into several subsets, on which a different function is applied, the results being ultimately summed together. The hybrid method should have applied the Schaffer's f6 function (not shifted and not rotated) on the last 20% dimensions from a given permutation. Because the Schaffer's function uses the global y vector (which in this case does not hold the shifted values due to the function not being shifted), the current behavior is that the first 20% dimensions are used to compute a value. As an example, for 10 dimensions, the Schaffer's f6 function should have been applied on 1st and 4th dimensions, however it is applied on the 1st and 2nd dimensions. The 2nd dimension is also used when calculating Rastrigin's function, which means that this dimension needs to minimize both functions at the same time. The 4th dimension is never used when calculating any function, therefore its values are never used in the minimization process.

III. Previous solutions

Previous submission for this competitions have used variants of nature inspired algorithms such as EA [10], DE [9], [12], [3], [2], Cuckoo search [8] and hybrids like Multi-Population Exploration-only Exploitation-only Hybrid [1].

A previous direction of research used GA as an optimization algorithm and in this paper those results will be compared with the results obtained by using PSO.

IV. Particle Swarm Optimization improvements

As it has been mentioned before, parameters for the PSO can be adjusted in order to get good results for a specific problem. But a good set of parameters that minimize all functions together would not have adequate results for each function in particular. In order to increase the optimization power of the PSO several strategies have been implemented.

A. Caching

Due to the fact that the competition limits the number of function evaluations to be made, using a caching strategy helps both the exploration and the exploitation process. In this paper the caching is done by using a KDTree to save the already evaluated vectors and their value. We update the KDTree at each function evaluation, and we do rebalancing several times across the algorithm in order to decrease the search time.

We have identified three strategies that can be used to retrieve close neighbours for a given vector, which are similar to the three strategies of the Hill Climbing Algorithm: the best neighbour, the worst neighbour and the first neighbour. We use the neighbour retrieved only if the distance between it and our query vector is less than an ϵ which decreases over time. The parameter ϵ starts with a value of 0.01 and constantly decreases with each function call until it reaches $1e-8$.

Our experiments have shown that we get worse results by using the best neighbour, and we have similar results when using the first neighbour or the worst neighbour.

B. Mutation

In PSO, the swarm may be damped to an equilibrium state. For an extreme case, if the particles have the same locations, the same past best values, and are all in zero velocities at a certain stage (for example, initialization stage), then the swarm is in stationary equilibrium with no possibility to improve [15].

According to the research done in [15], in order top prevent the trend of the swarm stagnating when being in equilibrium, a negative entropy can be introduced to provoke chaos within the particles, following Eq. 1.

$$\text{if } rand(0,1) < c \text{ then } v_{i,d} = rand(-|b-a|, |b-a|) \quad (1)$$

Where a and b are the search-space lower and upper bound, and $c \in [0,1]$. In this paper, c was a parameter that could be adjusted in order to better minimize the benchmark functions.

C. Regeneration

Due to the fact that PSO converges very fast, we have applied a population regeneration technique [7] if the best particle was not updated for several epochs. Combined with the caching strategy, this technique allows us to explore more.

D. Multiple swarms

Since there are multiple functions that need to be optimized simultaneously, we employed multiple swarms that each have different parameters which better particularize to a specific function. Moreover, the swarms are attracted to a global best of the swarms, besides their local swarm one. In that way some swarms are used to perform exploration, while smaller swarms will move to the global best and exploit the solution.

E. Topologies

The Star topology might lead to the swarm being trapped in a local optimum. Thus, we have implemented the Ring topology in which each particle communicates only with two neighbors. The neighbors are not based on any distance, instead they are assigned when we initialize the swarm. By using the Ring topology, the swarm is more focused on exploration, complementing the Multiple swarms optimization.

Another optimization that we have brought to topologies is a genetic algorithm that evolves them over time. Thus, every dimension of a particle will have a specific topology associated with it. Since there are only two topologies, we can represent them as a binary array, where 0 represents the Star topology and 1 the Ring topology. The crossover and mutation are performed as in the standard genetic algorithm, using a single cutoff point for crossover and randomly changing some bits for mutation. We have also used elitism, which copies the top performing topology bit arrays according to the particle's evaluations. Selection is also performed based on the particle's evaluations associated to a topology bit array.

The parameters used for the genetic algorithm were the following:

- crossOverProbability = 0.7
- mutationProbability = 0.001
- elitesPercentage = 0.5

F. Particle evolution

Based on the idea of evolving the topologies, we have decided to also evolve the particles. This evolution acts on the particle's position and its velocity. The mutation is performed as described above, by replacing a velocity dimension's value with a random one. The crossover is performed in the same manner as in the standard genetic algorithm, using a single cutoff point.

The parameters used for the genetic algorithm were the following:

- crossOverProbability = 0.1
- mutationProbability = chaosCoefficient
- elitesPercentage = 0.5

V. Experimental results and analysis

The programming language used to implement the PSO and run the experiments is C++, using the C++20 standard and compiled using gcc 11.2.0.

We compare the experimental results of the current version and the previous version. All experiments for PSO are run respecting the conditions for the competition, the algorithm is stopped either when maxFES is reached or when the global optimum ($1.E$-8) is reached. For each function the sample size is 30.

For the previous PSO, the parameters used are 0.3 for inertia, 1.0 for cognition and 3.0 for social. The chaos coefficient is 0.001 and the population size is 500.

The current version of the PSO uses multiple swarms, each with different parameters.

Similar results for some of the easier functions are obtained by both GA and PSO. However for the more complex functions PSO wins by a large margin. It must be mentioned that the comparison is not fair because the results for the GA do not use the same implementation as those from PSO, for which the implementation was adjusted to follow the official one.

PSO doesn't normally have to rely on operations like crossover and mutation in order to explore the search-space. Instead, PSO spreads the swarm around the search-space and the particles actually traverse the function domain in order to find better solutions. Moreover, the swarm acts in a collective way, always getting pulled by the best global value.

The optimization mentioned in Eq. 1 helps PSO avoid premature geometrical convergence, and thus it improves the algorithm's results. The particles are more inclined to exploring the search-space, although in a chaotic way that could impact the social behaviour and implicitly the solution convergence. Another major benefit of the introduction of chaos is that the particles don't get stuck as easily in local optimums, which is especially useful in a case in which there are many multi-modal functions at play.

PSO's solutions converge more rapidly because the particles are actively searching the function domain themselves and can stumble upon better solutions if they are on their path. This effect is more amplified since the particles could even be pulled out of their trajectory by the introduction of chaos.

TABLE I
PSO 10 dimensions

Function	*Mean*	*Median*	*Std*	*Min*	*Max*
zakharov	1.9E-06	1.7E-06	1.3E-06	1.8E-07	6.0E-06
rosenbrock	6.3E+00	4.0E+00	1.2E+01	1.8E-08	7.1E+01
schaffer	4.9E-04	4.5E-04	1.8E-04	2.6E-04	9.9E-04
rastrigin	2.1E+01	2.2E+01	7.5E+00	5.0E+00	4.0E+01
levy	1.3E+00	7.7E-01	1.9E+00	4.8E-08	8.8E+00
hf01	3.0E+02	6.8E+00	1.1E+03	4.8E-01	5.2E+03
hf02	1.2E+01	2.0E+01	9.7E+00	4.9E-05	2.0E+01
hf03	1.7E+01	2.0E+01	7.3E+00	8.0E-02	2.1E+01
cf01	2.3E+02	2.3E+02	6.5E-03	2.3E+02	2.3E+02
cf02	1.0E+02	1.0E+02	5.5E-02	1.0E+02	1.0E+02
cf03	8.7E+01	6.7E-03	1.1E+02	2.5E-03	4.0E+02
cf04	1.6E+02	1.6E+02	1.0E+00	1.6E+02	1.7E+02
$\sum$ mean	9.41E+02				

TABLE II
PSO 20 dimensions

Function	*Mean*	*Median*	*Std*	*Min*	*Max*
zakharov	0	0	0	0	0
rosenbrock	4.0E+01	4.9E+01	1.8E+01	0	5.6E+01
schaffer	0	0	0	0	0
rastrigin	6.1E+01	5.6E+01	2.0E+01	2.4E+01	1.1E+02
levy	5.2E+02	4.2E+02	3.8E+02	6.5E+01	2.1E+03
hf01	3.9E+03	1.8E+02	6.9E+03	4.1E+00	2.3E+04
hf02	4.6E+01	4.1E+01	2.0E+01	2.0E+01	1.0E+02
hf03	3.5E+01	2.1E+01	3.7E+01	2.0E+01	1.6E+02
cf01	1.8E+02	1.8E+02	9.1E-02	1.8E+02	1.8E+02
cf02	9.2E+01	1.0E+02	4.4E+01	6.2E-02	2.4E+02
cf03	3.4E+02	3.0E+02	1.1E+02	0	7.6E+02
cf04	2.7E+02	2.6E+02	2.4E+01	2.4E+02	3.5E+02
$\sum$ mean	5.46E+03				

TABLE III
GPSO 10 dimensions

Function	*Mean*	*Median*	*Std*	*Min*	*Max*
zakharov	7.6E-08	0	2.7E-07	0	1.5E-06
rosenbrock	3.0E-07	0	1.0E-06	0	4.4E-06
schaffer	1.8E-01	1.4E-01	2.3E-01	3.2E-04	1.3E+00
rastrigin	5.0E+00	5.0E+00	1.6E+00	2.0E+00	9.0E+00
levy	2.3E-01	2.0E-02	4.3E-01	0	1.5E+00
hf01	6.0E-01	3.8E-01	4.6E-01	5.7E-02	1.6E+00
hf02	4.0E+00	4.0E+00	2.6E+00	2.5E-03	8.7E+00
hf03	6.2E+00	5.5E+00	3.5E+00	8.0E-01	1.8E+01
cf01	2.3E+02	2.3E+02	0	2.3E+02	2.3E+02
cf02	1.0E+02	1.0E+02	4.8E-02	1.0E+02	1.0E+02
cf03	2.1E-05	6.9E-06	3.3E-05	3.8E-07	1.2E-04
cf04	1.6E+02	1.6E+02	1.0E+00	1.6E+02	1.6E+02
$\sum$ mean	5.09E+02				

We can see in the GPSO tables that the new results after applying the optimizations are much better than the previous

TABLE IV
GPSO 20 DIMENSIONS

Function	***Mean***	***Median***	***Std***	***Min***	***Max***
zakharov	4.3E-07	0	1.5E-06	0	8.0E-06
rosenbrock	1.8E+01	4.0E+00	2.1E+01	0	4.9E+01
schaffer	2.4E+00	1.8E+00	1.3E+00	6.8E-01	5.2E+00
rastrigin	2.3E+01	2.3E+01	3.5E+00	1.6E+01	2.9E+01
levy	2.7E+01	2.4E+01	1.3E+01	7.8E+00	5.7E+01
hf01	3.9E+00	3.7E+00	2.7E+00	3.3E-01	1.2E+01
hf02	2.5E+01	2.6E+01	4.6E+00	1.1E+01	3.5E+01
hf03	2.3E+01	2.2E+01	2.6E+00	1.2E+01	2.7E+01
cf01	1.8E+02	1.8E+02	0	1.8E+02	1.8E+02
cf02	9.8E+01	1.0E+02	1.6E+01	1.1E+01	1.0E+02
cf03	1.7E+02	3.0E+02	1.5E+02	3.5E-07	3.0E+02
cf04	2.4E+02	2.4E+02	3.6E+00	2.4E+02	2.5E+02
$\sum$ mean	8.12E+02				

ones. Even if some of the basic functions scored a bit worse, the more complex ones like hf01, hf02, hf03 and cf03 obtained much better results.

During our experiments we have observed that even the hard functions (cf01, cf02 and cf04) can be minimized. When running our algorithm 10.000 times, we find some runs for which a value close to 0 is obtained for one of the previous mentioned functions. This happens consistently, but we didn't include this in our results due to the fact that while it can be reproduced for a sample of 10.000 runs, for 30 runs it rarely happens. Taking this into consideration, we estimate that our algorithm converges towards a plateau of these functions where the minimum is located in a very narrow zone hard to reach unless a proper combination of velocity and direction appears for a particle which does happen randomly with a very small probability. We have tried to combat this without success by using different and even opposite techniques, such as decreasing the inertia, cognition and social factors in order to make smaller steps towards the end, or using jitter in order to enhance the exploration of the plateau. While improving the results on some of the easier functions, none of these methods gave us significant results on the hard functions. Therefore our supposition is that the PSO Algorithm is not able to locate the global optimum on the hard functions on the current geometry.

VI. Conclusion and Future Work

We have approached the CEC22 problems with several approaches, firstly using the GA and later the PSO. The GA was not able to get good results on the hard functions even while ignoring the constraints of the competition, while PSO achieved better results while respecting the rules. The PSO and GPSO is able to converge and find good local optima for most of the functions in the required maxFES. However it is very difficult to improve the results even without careful parameter fine-tuning and specific optimizations that take advantage of the caching strategy. Using different topologies and more swarms allows us to improve the results even more on several functions, while running the experiments multiple times shows us that it's possible to reach 0 for each problem. One approach to solve these problems is by using a multi-swarm approach and adjusting the parameters for each swarm to minimize only one function. However, while this approach could achieve better results for most of the functions (except for the hard ones which are blocked in a local optima), this is not a solution that scales well since we would need a carefully fine-tuned swarm for each function. Moreover, we are also constrained by the maxFES, which do not allow us to run the algorithm enough for each swarm. What remains to be seen is whether by using a different algorithm such as EA, DE or LShade, we are able to minimize the functions more than we could do with PSO, and search potential optimizations for them.

References

[1] Antonio Bolufé-Röhler and Stephen Chen. A multi-population exploration-only exploitation-only hybrid on cec-2020 single objective bound constrained problems. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[2] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. Single objective real-parameter optimization: Algorithm jso. In *2017 IEEE congress on evolutionary computation (CEC)*, pages 1311–1318. IEEE, 2017.

[3] Petr Bujok and Josef Tvrdík. Enhanced individual-dependent differential evolution with population size adaptation. In *2017 IEEE congress on evolutionary computation (CEC)*, pages 1358–1365. IEEE, 2017.

[4] Eugen Croitoru. Details about the ga and it's implementation, available at https://profs.info.uaic.ro/∼eugennc/teaching/ga/.

[5] Abhishek Kumar, Kenneth V. Price, Ali Wagdy Mohamed, Anas A. Hadi, and P. N. Suganthan. Problem definitions and evaluation criteria for the cec 2022 special session and competition on single objective bound constrained numerical optimization. In *https://github.com/P-N-Suganthan/2022-SO-BO/blob/main/CEC2022%20TR.pdf*, 2022.

[6] Lester James Miranda. Pyswarms: a research toolkit for particle swarm optimization in python. *Journal of Open Source Software*, 3(21):433, 2018.

[7] Albert Ramona and Ojoc Georgiana. Enhanced particle swarm optimization algorithm with population regeneration for single objective bound constrained numerical optimization. T1Conf, March 2022.

[8] Rohit Salgotra, Urvinder Singh, Sriparna Saha, and Amir H Gandomi. Improving cuckoo search: incorporating changes for cec 2017 and cec 2020 benchmark problems. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7. IEEE, 2020.

[9] Karam M Sallam, Saber M Elsayed, Ripon K Chakrabortty, and Michael J Ryan. Improved multi-operator differential evolution algorithm for solving unconstrained problems. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[10] Karam M Sallam, Saber M Elsayed, Ruhul A Sarker, and Daryl L Essam. Multi-method based orthogonal experimental design algorithm for solving cec2017 competition problems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1350–1357. IEEE, 2017.

[11] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE, 1998.

[12] Vladimir Stanovov, Shakhnaz Akhmedova, and Eugene Semenkin. Ranked archive differential evolution with selective pressure for cec 2020 numerical optimization. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7. IEEE, 2020.

[13] Dongshu Wang, Dapei Tan, and Lei Liu. Particle swarm optimization algorithm: an overview. *Soft Computing*, 22(2):387–408, 2018.

[14] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[15] Xiao-Feng Xie, Wen-Jun Zhang, and Zhi-Lian Yang. Dissipative particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1456–1461. IEEE, 2002.