UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Alberto Becerra Tomé, Javier González Béjar

Computational Intelligence

Work 2. Optimizing a difficult function with Evolution Strategies

1. Introduction

This exercise presents an exploration of optimization strategies, focusing on the application of Evolution Strategies (ESs) to the minimization or maximization of multivariate functions. In our case, we will be dealing with a Constrained Non-Linear Pressure Vessel Design Problem, which aims at minimizing the cost of the material by optimizing the design variables of the vessel. However, absolute minimization cannot be achieved as some constraints are needed to be met to uphold the safety, efficacy, and physics of the vessel.

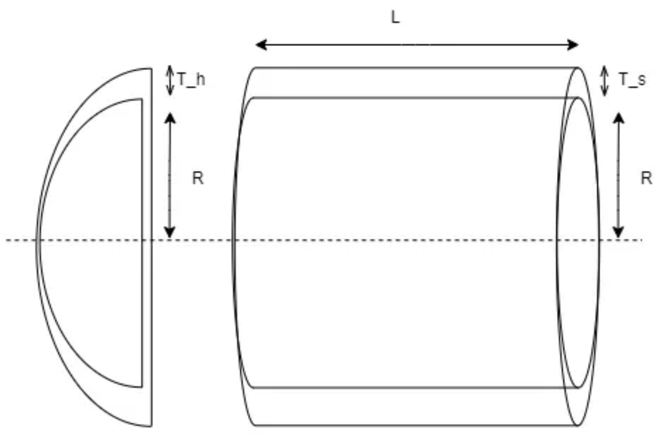
The study involves comparing the performance of ESs considering factors such as execution time, the number of generations, fitness function calls, and the algorithm's success rate.

2. Problem Overview

The task involves optimizing the design of a compressed air storage tank, which operates at a working pressure of 2000 psi and has a maximum capacity of 750 cubic feet. This tank is designed in a cylindrical shape with hemispherical ends, as depicted in Figure 1. The primary goal is to reduce the overall cost, which encompasses the expenses for materials, shaping, and welding. The design involves four key variables: the thickness of the vessel's wall (Ts = x1), the thickness of the ends (Th = x2), the inner radius of the cylinder (R = x3), and the length of the cylindrical section excluding the ends (L = x4). These variables are represented in inches and are collectively denoted as X = (Ts, Th, R, L) = (x1, x2, x3, x4).

The design involves four variables of interest:

- **Inner Radius (R)**: continuous, spanning from 10 to 200 inches.
- **Length of Cylindrical Section (L)**: continuous, spanning from 10 to 200 inches.
- **Thickness of the Body (T_s)**: takes discrete values measured from 0.0625 to 6.1875 inches in steps of 0.0625.
- **Thickness of the Cap (T_h):** takes discrete values measured from 0.0625 to 6.1875 inches in steps of 0.0625.



This problem has four different constraints, one of them being non-linear. Below it can be found the function we want to minimize along with its constraints:

$$\textit{minimize}$$

$$f(\boldsymbol{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

$$\textit{subject to:}$$

$$g_1(\boldsymbol{x}) = -x_1 + 0.0193x_3 \leq 0,$$

$$g_2(\boldsymbol{x}) = -x_2 + 0.00954x_3 \leq 0,$$

$$g_3(\boldsymbol{x}) = -\pi x_3^2x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0,$$

$$g_4(\boldsymbol{x}) = x_4 - 240 \leq 0$$

Figure 2: Problem equation and constraints

The problem domain is the compact [0.0625, 6.1875] x [0.0625, 6.1875] x [10, 200] x [10, 200]. The Weierstrass Extreme Value Theorem states that a continuous function on a compact set attains its maximum and minimum values. Given that f is a continuous (polynomial) function, and the domain is compact (closed and bounded in R^4), it is true that f will attain global optima (both maxima and minima) within this space. However, it's important to note that this theorem doesn't specify the location of these optima or whether they are unique.

The problem arrives when x3 and x4 are **imposed to be multiples of 0.0625**. This is done introducing *round* function in the definition of the cost. If x3 and x4 can only take on discrete values, the domain of the function becomes a set of discrete points rather than a continuous space. This changes the nature of the problem:

1. **Compactness**: If the range of x3 and x4 is finite and they only take discrete values, the domain is still compact since it is closed and bounded. However, it's no longer a continuum but a discrete set of points.

2. **Application of Weierstrass Theorem**: The Weierstrass theorem applies to continuous functions on compact sets. In a discrete setting, the concept of continuity is different. However, the essential idea that the function will attain its maximum and minimum values still holds because a finite set of points can be exhaustively searched.

3. **Optimization Implications**: For optimization, this means that while the function will have global optima, these optima might only exist at discrete points. The optimization technique might need to be adjusted to account for the discrete nature of x3 and x4. In particular, methods that rely on gradient or require infinitesimally small steps might not be directly applicable, and discrete optimization methods might be more suitable.

In this scenario, we are dealing with a complex constrained problem that encompasses a domain of variables incorporating both continuous and discrete elements, along with four constraints, including a nonlinear one (g3). This presents an ideal opportunity to evaluate the effectiveness of our Classic Evolutionary Programming Algorithm.

3. Previous Work

In Garg et al. [1] work, it is highlighted that numerous researchers have tackled a specific structural optimization problem within a defined variable region:

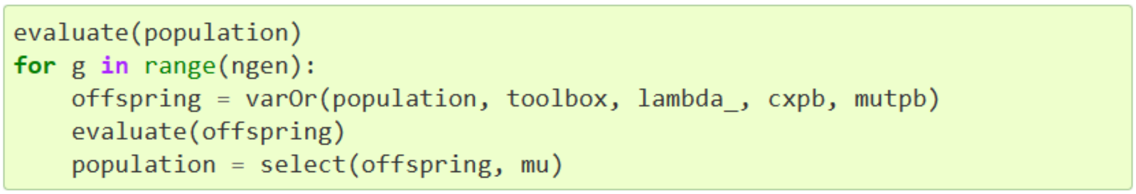
Region I: $1 \times 0.0625 \leq x1, x2 \leq 99 \times 0.0625; 10 \leq x3, x4 \leq 200$

In [1], ABC algorithm is used to perform this task. The Artificial Bee Colony (ABC) algorithm and Gaussian Evolution Strategies (GES) are both used for numerical optimization but differ in their approach. The ABC algorithm is inspired by honey bee foraging and uses three types of bees (employed, onlooker, and scouts) for solution exploration and exploitation. It emphasizes collaborative information sharing and is effective in avoiding local optima. GES, based on evolutionary principles, uses Gaussian mutations to evolve solutions, focusing on balancing exploration and exploitation. Its effectiveness lies in the adaptive fine-tuning of solutions across generations. While both methods are valuable for optimization, their suitability varies based on the problem, with ABC excelling in collaborative exploration and GES in evolutionary adaptation. In this work, results using both methods are compared.

4. Experimental Setup

Algorithm approach

For this problem we will proceed using an approach of the (μ, λ)-Gaussian Evolutionary Strategies algorithm, which consists in taking a population and iteratively evolving it through the application of mutation and recombination operations. However, our evolution method will only use mutation and not crossover to generate offspring, as we are interested in evolving the behaviour of individuals. With the appropriate parameters, this algorithm can find a balance between exploration and exploitation, enhancing the diversity of the population while focusing on promising solutions. Going through a dynamic interplay of mutation, and selection mechanisms, (μ, λ) algorithm aims to navigate the solution space effectively to optimize the given objective.



```
evaluate(population)
for g in range(ngen):
    offspring = varOr(population, toolbox, lambda_, cxpb, mutpb)
    evaluate(offspring)
    population = select(offspring, mu)
```

Figure 3: (μ, λ) Evolutionary algorithm pseudocode

As seen in the pseudocode in Figure 3, initially, we evaluate individuals from the population. Following that, the evolutionary process loop begins, generating a set number of offspring, based on *lambda* parameter, from the existing population using the *varOr* function, which in our case consists in mutation and reproduction. After the offspring are created, their fitness is evaluated, and the subsequent generation's population is selected. Finally, once the specified number of generations *ngen* is completed, the algorithm provides a tuple containing the final population and a *Logbook* detailing the progression and statistics of the evolution.

Problem initialization

The optimization problem setup requires instantiating a Fitness Class with negatives weights, which indicate the direction in which the optimization algorithm should search for solutions. We want to achieve a solution that minimizes the value of the fitness value so we can reduce material costs. In evolutionary algorithms, a solution to the optimization problem is represented as an individual, and in our code, Individual is defined as an array containing the values for each vessel design variable. The Strategy class is introduced to represent the evolutionary strategy, which guides how the individuals evolve over generations.

So, to specify our initial population, we initialize individuals with both their variable values and their strategy ones. This algorithm works by using Gaussian Mutation, log-normal self-adaptation of the strategy parameter, the standard deviation in our instance, and elitism to choose the best individuals from the pooled collection of parents and offspring while using relative fitness through tournaments. Initializing the generation variables involves simple uniform sampling within the domain. However, strategy parameter initialization is subjective; a smaller standard deviation encourages exploitation, while a larger one favours exploration. In this case, we aimed for a standard deviation covering roughly 1-20% of the total domain size.

Offspring generation

As it was stated, we base or offspring generation on reproduction and mutation of the evolution strategies, no crossover. For the mutation, first, the strategy is mutated according to an extended log normal rule described below:

$$\boldsymbol{\sigma_t} = \exp\big(\tau_0 \mathcal{N}_0(0,1)\big)\left[\sigma_{t-1,1}\exp\big(\tau\mathcal{N}_1(0,1)\big), \dots, \sigma_{t-1,n}\exp\big(\tau\mathcal{N}_n(0,1)\big)\right]$$

Where:

$$\tau_0 = \frac{c}{\sqrt{2n}}, \tau = \frac{c}{\sqrt{2\sqrt{n}}}$$

Then, the individual is mutated by a normal distribution of mean 0 and standard deviation of the current strategy $\boldsymbol{\sigma_t}$.

We will be using a learning parameter *c* of 1 and an *indpb* parameter of 0.3, which is the independent probability for each attribute to be mutated. This means that, on average, about 30% of the genes in an individual will undergo mutation in each generation, introducing the necessary variability for effective exploration while maintaining stability in the evolutionary process.

Selection process

The selection process is based on the Tournament Selection method, which consists of randomly selecting a subset of individuals from the population and then choosing the best individual from that subset. The size of the subset is defined by the tournament size parameter.

We are also using elitism to ensure that the best-performing individuals from the current generation are directly preserved in the next generation. This strategy helps maintain the fittest solutions over generations, preventing the loss of valuable genetic material. By retaining top individuals unchanged, elitism contributes to the algorithm's efficiency in converging towards optimal or near-optimal solutions in the search space.

Offspring evaluation

In the final step of our evolutionary algorithm, we assess the offspring values using the *pressure_vessel* function, with the aim of minimizing the resulting fitness. To evaluate the feasibility of individuals, we generate a list of constraint values, determining their adherence to the problem constraints. In the case of infeasible individuals, we apply a penalty to guide the optimization towards feasible solutions. This integrated approach ensures that our algorithm not only strives for optimal fitness but also considers and penalizes deviations from the problem constraints to encourage feasible and practical solutions.

5. Analysis of results

The experiment consisted in 100 independent runs using the following configuration:

- Number of parents (μ): 1000
- Offspring size (λ): 1000
- Probability of Mutation (*mutpb*): 0.8
- Number of generations: 1000
- Number of fitness function calls: 1,000,000
- Elitism ratio: 0.1
- Tournament size: 8

Best Results

In Table 1, the results obtained using the ABC algorithm in [1] are shown. The best fitness obtained in this work can be observed in Table 2, with the design variables which takes to a fitness of 5800.698498 for the cost of the design, improving the rest of the obtained results.

The values of the variables respect the constraints, being in the allowed ranges and satisfying g1 to g4. The values of the restrictions when evaluated on the solution are the following ones:

[-0.01697966460, -1.750173924e-05, -0.06424950, -69.2157340]

And, in addition to that, the condition of x1 and x2 being multiples of 0.0625 are respected too, being x1 equal to 6 times the step value and 13 in the case of x2 (result of rounding x/0.0625 to the units).

X1	X2	X3	X4	Best Fitness	Experiment Time (s)
0.778197	0.384665	40.321054	199.98023	5885.403282	0.575

Table 1: Comparison of the best solution for pressure vessel design problem found in [1]

X1	X2	X3	X4	Best Fitness	Experiment Time (s)
0.838733	0.406210	42.577929	170.784265	5800.698498	76.422271

Table 2: Best results obtained using ESs.

Mean Fitness	Std	Median Fitness	Worst Fitness	Unique Individuals	Infeasible Individuals
5804.725420	51.754588	5800.6984	6870.9704	14	0

Table 3: Characteristics of the population

In Table 3, some extra information about the population with the best solution is included. The median value coincident with the minimum indicates that more than the fifty percent of the values of the fitness evaluation on the individual are coincident. The unique individuals measure has been very useful to check how explorative the algorithm was for each of the generations. In this experiment, there are 14 individuals out of the 1000 that were initially generated.

Statistics over the multiple runs

In Table 4, the results for different numbers of runs for each experiment using the different methods mentioned in [1] are shown. A good measure of the robustness of the algorithms is how scattered the different results for each run are. In the case of the method used in [1], it seems to be notably more robust than ours.

Best	**Mean**	**Median**	**Worst**	**Std Dev**
5885.4032828	5887.55702	5886.149289	5895.12680	2.7452903

Table 4: Statistical results on 30 runs of ABC algorithm from [1]

Best	**Mean**	**Median**	**Worst**	**Std Dev**	**Unique**	**Feasible**
5800.69849	5973.14344	5916.124942	6720.53667	190.5736	14.74	998.54

Table 5: Statistical results on 100 runs for our experiment

Even if the result is better in our best experiment, finding this one could have been due to luck. To make it statistically significant, we pay attention to the average and standard error of the mean. Standard error gives the accuracy of a sample mean by measuring the sample-to-sample variability of the sample means. The SEM describes how precise the mean of the sample is as an estimate of the true mean of the population. A 95% confidence interval can be obtained by multiplying SE by 1.96. The result for our mean is 5973±37 and, in the case of the result in [1], knowing that the number of runs is 30, the obtained confidence interval is 5887±1. As a conclusion, even if the minimum value obtained with our method is lower than in the case of [1], it is less robust, leading to a lower value in most of the cases.

In Figures 5 and 6 the distribution of the different metrics across all the experiments can be observed. Significant dispersion can be observed in all the cases. The location of Q1 and Q3 quartiles around 30 seconds in the case of running time and 5900 to 6100 in the rest of the metrics provides us some sensitivity about the global performance of the algorithm in the task.

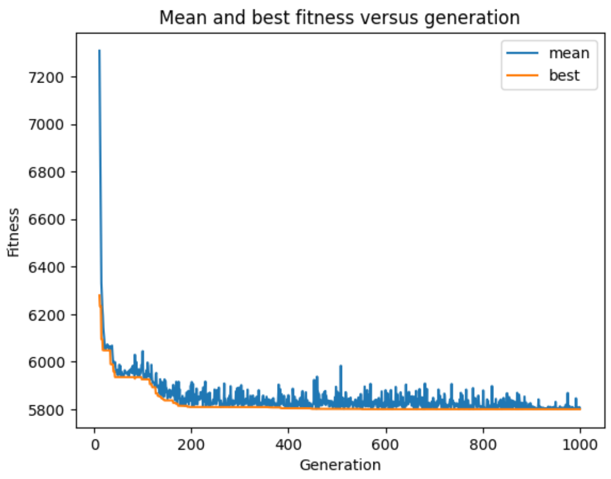
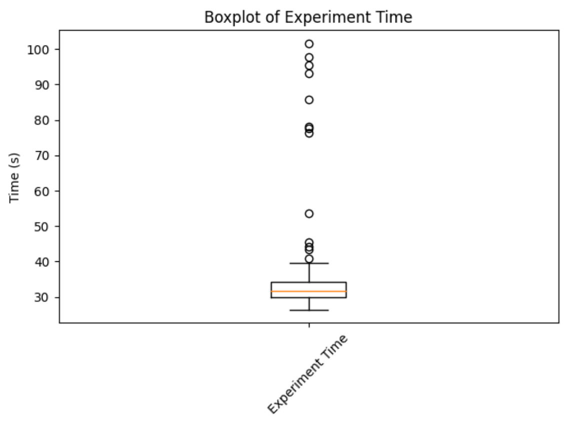
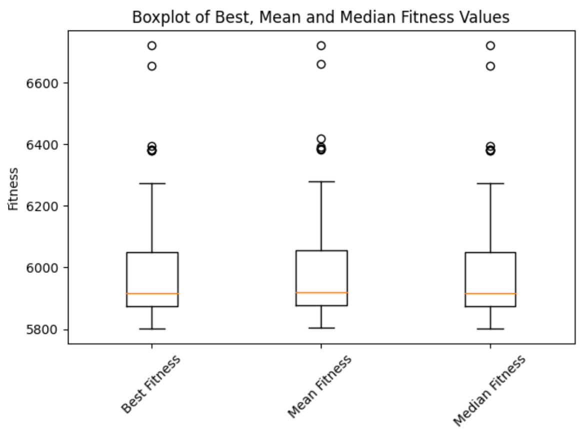


Figure 4: Mean and best fitness values over generations for the best experiment





Figures 5 and 5: Boxplot of Experiment Time (left). Boxplot of Best, Mean and Median Fitness Values (right)

6. Conclusions

We explored the use of Evolutionary Strategies to tackle a complex, real-world engineering optimization problem that's both non-linear, constrained and with mixed integer and real search space. When we compared our approach to an Artificial Bee Colony Algorithm mentioned in a study [1], we discovered that our method consistently yielded notable results in finding nearly optimal solutions over multiple independent runs. However, there was a considerable variation in the quality of the best solution from each trial, underscoring a certain inconsistency in our algorithm. This variation implies that our method isn't quite dependable if you're looking for reliable results in a few runs. Instead, it seems that running our algorithm through a large number of iterations is necessary to consistently approach an optimal solution.

Experimentation with different parameters of the ES algorithm and an implementation from scratch using DEAP python framework for genetic algorithms have been performed. Nearly optimal results have been obtained for the problem, having found a balance between exploration and exploitation.

7. References

[1] Solving structural engineering design optimization problems using an artificial bee colony algorithm, Journal of Industrial & Management Optimization,10,3,777,794,2013–11–1, Harish Garg,1547–5816_2014_3_777, Artificial bee colony, structural design optimization, nonlinear constraint., constraints handling.

[2] Chehouri, Adam & Younes, Rafic & Perron, Jean & Ilinca, Adrian. (2016). A constraint-handling technique for genetic algorithms using a violation factor. 12. 350–362. 10.3844/jcssp.2016.350–362.