Using Meta-heuristics to Optimize the Parameterization of Algorithms in Simulation Models

Chabi Babatounde[1], Bastien Poggi[1], Thierry Antoine-Santoni[1] and Antoine Aiello[2]
[1]*Université de Corse, UMR CNRS 6134, Science Pour l'Environnement, Corte, France*
[2]*Université de Corse, UMS CNRS 3514, STELLA MARE, Biguglia, France*

Keywords: Meta-heuristics, Optimization, Algorithm, Modeling and Simulation.

Abstract: The research topic of the laboratory Science Pour l'Environnement (SPE) and the laboratory STELLA MARE of Université de Corse, focus on solving the environmental problems of our time. Various research teams focus their work on modeling and simulation of complex systems and behavioral modeling of species. Generally, in this modeling process (abstractions from the real world), we observe that the parameterization of the models is usually very tedious, carried out in an empirical or intuitive way based on assumptions specific to each modeler. There are also several modeling techniques which are generally parameterized intuitively and empirically. We have therefore proposed an approach to optimize the parameterization of models based on the algorithms of these models. This approach uses meta-heuristics, a class of optimization algorithms inspired by nature for which we obtain remarkable results. The use of meta-heuristics in this approach is justified by the nature of the problem to be solved. Indeed, the parameterization of models can be considered as a complex problem with a very large solution space that needs to be explored in an intelligent way. The risk of a combinatorial explosion is also very high because of the number of variables to be optimized. The advantage of this approach that we propose is that it allows an evolutive optimization of the model parameterization as the data arrives. For the validation of this approach, we used simulated data from a theoretical model. The validation of this theoretical model opens possibilities of applications on real world models.

1 INTRODUCTION

The evolution of human activities and demographics in recent centuries have brought about several changes on the planet, with the result that a large number of species are in danger of extinction, even though they have a major ecological and economic impact. It is in this context that the SPE Laboratory and the STELLA MARE platform have been working since their creation to develop strategies to ensure the restoration and maintenance of Corsica's fishing resources. With the democratization of the Internet of Things in recent years (Dave, 2011), large volumes of data have been collected on species of interest in order to understand their behavior and model them (Schichl, 2004); (Wong and Ming, 2019), in order to make predictions. There are various modeling approaches in the literature (Yin and McKay2, 2018); (Sharma and Sharma, 2014a); (Zeigler, 2017) that produce acceptable results, but the configuration of these methods is generally very tedious and performed intuitively, based on assumptions specific to each expert. Since there is precisely no exact knowledge about the system to be modeled and the knowledge of the experts is limited, biases may be introduced. Based on this observation, our research focuses on refining the configuration of the models by adding a layer of optimization by meta-heuristics. The present work has consisted in using meta-heuristic optimization to improve the quality of simulations through intelligent model parameterization. The implementation of this approach allows to take in input datasets and the algorithm of the model to be configured to produce after processing, an optimal parameterization. The advantage of such an approach is that it is scalable and allows a dynamic refinement of the model as new data is received.

In the Section 2 of this paper, we present the works related to our work, and in the section 3, we review the general concepts related to optimization and meta-heuristics. We then present the approach we propose for the optimization of model parameterization as well as the different variants of algorithms implemented in Section 5.

Babatounde, C., Poggi, B., Antoine-Santoni, T. and Aiello, A.
Using Meta-heuristics to Optimize the Parameterization of Algorithms in Simulation Models.
DOI: 10.5220/0010508102150223
In *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2021)*, pages 215-223
ISBN: 978-989-758-528-9

2 RELATED WORK

Over the last twenty years, it has become increasingly common to see a particular emphasis on optimization in modeling and simulation work; (Sharma and Sharma, 2014b); (Allegrini et al., 2015); (Hutterer, 2010).Today, it is almost impossible to see any modeling work that does not involve optimization, because the ultimate goal of modeling is to get as close as possible to the real system, which means solving optimization problems at several levels, such as determining the optimal structure and the optimal parameters of the model. When we consider the parameterization of models, optimization methods are much more used in the design of behavioral models than in phenomenological models. This is mainly due to the fact that for phenomenological models, the relationships that rule the system to be modeled are known, contrary to behavioral models where it is necessary to find relationships that link the inputs and outputs of the system. The main methods used for this type of problem are automatic programming and artificial neural networks. Automatic programming started in the 90's with genetic algorithms (genetic programming) (Koza, 1992). With the evolution of computers and the availability of more computing power, meta-heuristics started to be more and more present in the world of modeling and automatic programming in particular. New meta-heuristics were appeared but they focus much more on the generation of models than on their refinement. It is this refinement that we find in machine learning models and more specifically in artificial neural networks under different angles: optimization of weights, network architecture, activation nodes, learning parameters, learning environment, etc. (Gandomi, 2013). The advantage of meta-heuristics in this process over "classical" gradient descent methods is that meta-heuristics can overcome limitations related mainly to blocking in the local extrema (Ojha et al., 2017).

3 OPTIMIZATION PROBLEM AND META-HEURISTICS

An optimization problem is defined by its dimension $n \in \mathbb{N}$, its set of variables $\{x_1, x_2, \dots, x_n\}$ denoted S, a set of constraints C and a goal function $f : S \mapsto \mathbb{R}$. Solving an optimization problem consists in finding the optimal values of $x_1, x_2, \dots$ and x_n which minimise or maximise the objective function on C. Metaheuristics are used to find these optimal values. Metaheuristics (Tarraq, 2021); (Abdel-Basset et al., 2018); (Glover and Sörensen, 2015); (Xin-She, 2014); (Xin-She, 2011); (Lee and El-Sharkawi, 2008), also called "nature-inspired methods" form a class of optimization methods which are based on probabilistic and random reasoning and allow the resolution of problems for which "classical" optimization methods also called "deterministic methods" (Moxnes, 2015) do not allow results to be obtained. Meta- heuristics are based on an iterative process which allows convergence towards the solution. In other words, a metaheuristic starts from one or more solutions which it improves around two central concepts: diversification and intensification. Diversification makes it possible to search in the research space for areas where solutions could be found and intensification makes it possible to improve the solutions found in the identified areas. A multitude of algorithms and implementations of meta-heuristics can be found in the literature. As far as they are concerned, the development of these algorithms goes through five (05) generic phases that are executed in an iterative manner until a solution is found or one of the stopping conditions is satisfied as shown in Figure 1 which presents these different phases.

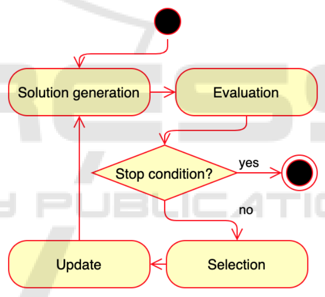

Figure 1: Meta-heuristics execution phases (Poggi, 2014).

There are several meta-heuristic P's in the literature. In order to make a performance comparison between the meta-heuristics we will focus on the two most intuitive and which are inspired by the animal kingdom. These are the genetic algorithm and the immune algorithm.

3.1 Genetic Algorithms

Genetic algorithms (Sopov, 2017); (Oltean, 2005) (McCall, 2005); (Wong and Ming, 2019) are optimization algorithms based on analogies of genetics and natural evolution (selection, cross-breeding, mutations). Table 1 presents the analogy between the concepts of genetics and genetic algorithms. A genetic algorithm searches for the extrema of a defined function in a data space. The behaviour of a genetic algorithm is described by Algorithm 1. It begins by randomly generating a population of individuals. Then, in order to move on to the next generation, three main operations inspired by natural evolution are applied, which are repeated for all individuals in the population.

- **Selection:** Couples of parents $P1$ and $P2$ are selected according to their adaptation to the problem to be solved.
- **Cross-breeding:** Each pair of parents (P_i, P_{i+1}) selected is crossed to produce pairs of children(C_i, C_{i+1}). There are various crossing techniques. It can be for example an exchange of variables between parents (P_1 and P_2 with respective genomes $\{22,25\}$ and $\{12,15\}$ give children C_1 and C_2 with respective genomes $\{22,15\}$ et $\{12,25\}$).
  It is also possible to make a combination of each variable from each parent according to a random constant α with $\alpha \in [0,1]$. In this case, parents P_1 and P_2give children C_1 with the genome $\{22\alpha + 12(1-\alpha), 25\alpha + 15(1-\alpha)\}$ and C_2 with the genome $\{22(1-\alpha) + 12\alpha, 25(1-\alpha) + 15\alpha\}$
- **Mutation:** There are several mutation techniques in the literature and they are generally based on probabilistic approaches. The children (C_i, C_{i+1}) and individuals who have been mutated are then evaluated before being included in the new population.

Algorithm 1: Genetic algorithms.

```
population = generatePopulation( );
evaluate(population);
while Not(Solution found OR stop condition)
 do
    for $i \leftarrow 1$ to populationSize do
        parents = selectParents(population) ;
        children = crossBreeding(parents) ;
        children = mutate(children) ;
        evaluate(population) ;
        population = update(children) ;
return (solution) ;
```

3.2 Immune Algorithm

Immune algorithm (Darmoul, 2010); (Zhang, 2011) is based on the way the body's immune system defends itself against foreign bodies. Indeed, to defend the organism, the immune system is capable of creating a very wide variety of cells and molecules that can specifically recognize and eliminate a large number of foreign invaders. This ability to recognize an "aggressor" is used here to recognize solutions to the optimization problem. There are several types of immune algorithms. For this study, we use the CLONALG type immune algorithm (Sharma and Sharma, 2011). Table 2 presents the analogy made between the immune system of organisms and the immune algorithm of CLONALG type. Like the immune system of organisms, at the beginning of the process, a set of antibodies is generated. Each antibody of the immune system thus formed is evaluated and standardized. Then, an iterative process begins that leads to the solution. With each iteration, a quantity of solutions (antibodies) is produced and evolved. The evolution of the solutions in this algorithm, as with an immune system, is based on a principle of cloning and mutation. A solution better adapted to the problem will be much more cloned than a less adapted solution.

Similarly, a less suitable solution will undergo a greater change than a more suitable solution. As the optimization variables in our case are real, the mutation of a solution will consist in incrementing or decrementing the value of one or more variables. This technique allows convergence towards the desired solution. A phase which particularly influences this algorithm on its performance is the cloning phase. To determine the clonnig rate (quantity of clones produced), several approaches exist:

- **The Probabilistic Approach:** This approach allows the rate to be determined in proportion to the adjustment according to equation 1. When the number obtained is not an integer, it is rounded.

$$clones = antibodies * \frac{1}{adaptation} \tag{1}$$

- **The Empirical Approach:** Using data collected from several experiments with our algorithm, we manage to define a relation between a score attributed to the solution and the number of clones.
- **The Stochastic Approach:** The number of mutations and clones is set randomly within a predefined interval.

The process of an immune algorithm is described in Algorithm 2. Figure 2 illustrates the steps of an immune algorithm in the search for the value 10.

By exploring these few meta-heuristic algorithms, we realize the interest of their use for complex problems, which is in adequacy with our problem of "intelligent" parameterization of simulation models. To this end, the approach we propose is presented in Section 4.

Table 1: Analogy between genetics, genetic algorithm and optimization.

Genetic	Genetic algorithm	Optimization
Set genome possible	Set of possible genes	Search space
Individual	Coded solution	Solution
Genome	Details of a solution	Values of each parameter
Adaptation	Fitness	Adaptation
Natural selection	Random selection	Diversification
Genetic mutation	Value change	Diversification

Table 2: Analogy between Genetics, Immune Algorithm and optimization.

Immune system	Immune Algorithm	Optimization
All possible antibodies	All possible solutions	Search space
Antibody	Coded solution	Solution
Cell receptor	Details of a solution	Values of each parameter
Adaptation	Fitness	Adaptation

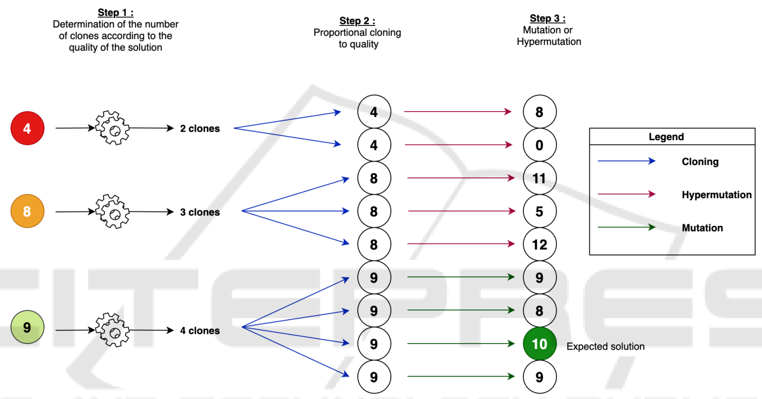
Step 1 :
Determination of the number of clones according to the quality of the solution
Step 2 :
Proportional cloning to quality
Step 3 :
Mutation or Hypermutation
4
8
9
2 clones
3 clones
4 clones
4
4
8
8
8
9
9
9
9
8
0
11
5
12
9
8
10
9
Expected solution
Legend
Cloning
Hypermutation
Mutation

Figure 2: Searching for the value 10 by an immune algorithm.

Algorithm 2: Immune Algorithm.

```
immuneSystem = randomlyGenerated( );
evaluate(immuneSystem);
normalize(immuneSystem) ;
while Not(Solution found OR stop condition)
 do
    clones = createClones(immuneSystem) ;
    clones = mutate(clones) ;
    evaluate(clones) ;
    nomarlize(clones) ;
    newAntibodies = generateNewClones() ;
    evaluate(newAntibodies) ;
    nomarlize(newAntibodies) ;
    immuneSystem = merge(clones,
      newAntibodies, immuneSystem) ;
return (solution) ;
```

4 PROPOSED APPROACH

We propose an architecture allowing to optimise the parameterization of the algorithms implemented in the models. Figure 3 presents the different components of this proposed architecture and the way they communicate.

4.1 Global Architecture

The proposed architecture for algorithm optimization is composed of four main entities:

- **The Optimizer:** It leads the optimization process. It contains the execution logic and behaves like a scheduler by choosing the order of execution of each task. Its role is mainly to choose an optimization algorithm and to parameterize it for the optimization process.
- **Meta-heuristics:** This entity can be considered as a library of meta-heuristics. It implements and

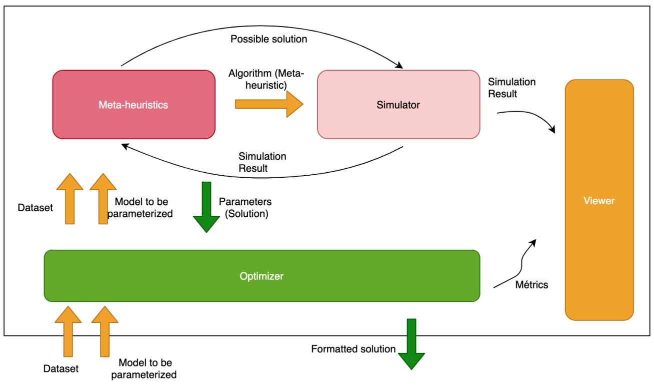

Figure 3: Proposed architecture.

makes available meta-heuristics for the process of optimization.

- **The Simulator:** The simulator, allows to produce for each solution obtained, the corresponding data set. This phase is essential to evaluate the solution obtained.
- **The Viewer:** It allows to visually render the dataset, the algorithm to be optimized as well as the evolution of the process.

All these entities cooperate together to find a solution in the following way: The model to be optimized and the reference dataset are passed to the ***optimizer***, which guides the optimization process. It chooses the ***meta-heuristics*** and parameters for the optimization. During the iterative process, the meta-heuristics will collaborate with the simulator to evaluate each solution obtained and move towards the optimal solution. At the end of the optimization process, the solution and the process metrics are sent back to the optimizer where the solution is rendered in the format specified by the user. It should be noted that throughout this process, the ***viewer*** is used to display the algorithm, the simulation results and the evolution of the solutions obtained.

4.2 Abstraction

We have given our own representation for some entities to facilitate data exchange between the elements. This is mainly the model and the solution. We will illustrate each element with simple examples to facilitate understanding.

The model to be optimized is defined by an algorithm (a logical instruction sequence) characterized by its variables and operations to be executed. In our proposed approach, we then represent a model like a graph. Each node of the graph represents an action or a phase in the execution of the algorithm. The way the nodes are linked defines the way the algorithm is executed. To illustrate this, let's consider the algorithm 3 which is a simple algorithm that calculates the remaining energy in a system up to $t = 100$. The corresponding representation is given in Figure 4.

Algorithm 3: Example of a model algorithm.

```
energy = 100 ;
t = 0 ;
running_cost = 5;
cost_at_rest = 2;
while $t \leq 100$ do
    if $running == True$ then
        energy = energy - running_cost ;
    else
        energy = energy - cost_at_rest ;
    t = t + 1 ;
```

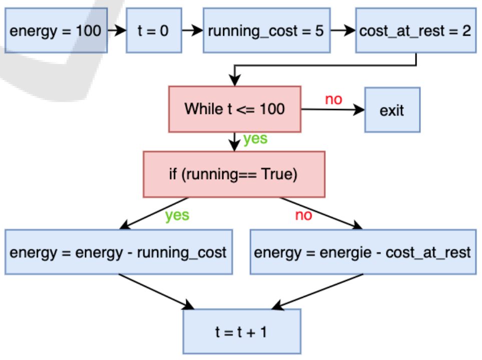

Figure 4: Representation of algorithm 3.

4.3 Meta-heuristics Implementation

For this first implementation, we are focusing on the genetic algorithm and the Immune algorithm of CLONALG type in order to carry out performance comparisons. In accordance with the concepts specific to each of these algorithms (presented in section 3), their implementation in the context of the approach complies with the following rules: For the genetic algorithm, we have:

- **Encoding:** All the parameters of the solution are converted to standardized binary form (IEEE754) and concatenated.
- **Cross-breeding Operator:** We make a crossover by exchanging between the solutions. This exchange consists in randomly defining the number of crossings (Xin-She, 2014) as well as the points where to realize them.
- **Mutation Operator:** A sufficiently large α value (0.95) is set at the beginning of the optimization. At each crossover, a value $\theta \in [0,1]$ is generated. If $\theta > \alpha$ then a mutation occurs. Otherwise nothing is changed

On the side of the CLONALG type Immune algorithm, the specificity was to implement the three approaches presented in Section 3.2 in order to determine the number of clones to be generated. This allows to vary the search method.

4.4 Solutions Evaluation

The adaptation of a solution is measured by the difference between the produced values and the awaited values. Several methods exist in the literature (Lee and El-Sharkawi, 2008); (Shtovba, 2005)to measure this difference. In this work, we use the root mean square error because it is simple to implement and explicit. Equation 2 then becomes the cost function to be minimized during the optimization process.

$$G_{(x,y)} = \frac{1}{2n} \sum_{i=1}^{n} ((x_i - x_i')^2 + (y_i - y_i')^2) \tag{2}$$

With (x_i, y_i) the coordinates of point i in the data set and (x_i', y_i') the coordinates of point in the simulation result. The higher the G value, the less suitable the solution.

5 ARCHITECTURE VALIDATION

5.1 Validation Model

The validation process of the proposed approach consisted in using data obtained by simulating a model to find the parameterization used for the configuration of the model, or an optimal parameterization. We base our tests on a conceptual animal species model that can be updated to be closer to a real-world animal species. The choice of this model type is explained by an exact knowledge of the awaited behaviour. This behaviour is defined over one hundred (100) units of time where each interval corresponds to behavioural functions. Each of the behavioural functions is also defined by a set of specific operations that vary the position of the subject and the level of energy it uses. The variation in position and level of energy is a function of several constants that characterize each behavioural function. It is these constants that must be optimized, and in the case of the model, there are ten (10). The data set produced by the model is a collection of positions and the amount of energy at each instant. Table 3 shows the distribution of behavioural functions by time interval and defines the associated position and energy variation functions. The task will be to find the optimal values for the constants ε_0, G_s, X_s, Y_s, G_w , X_w , Y_w, G_e, X_e and Y_e and Ye that give the closest possible output to the trajectory and the value of the energy in the reference data set.

By representing this algorithm in graph form, we obtain Figure 5.

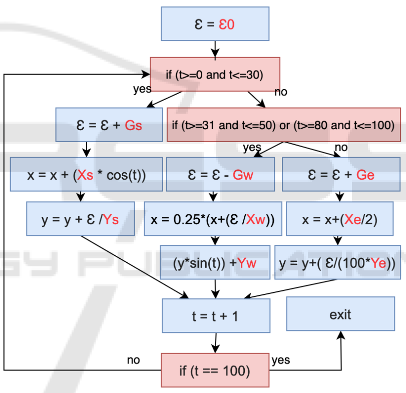

Figure 5: Graphical representation of the model (variables to be optimized in red).

5.2 Experimental Results

A series of fifty (50) tests were carried out with the data obtained by simulating the model presented in Section 5.1. The experimental conditions are recorded in Table 4 and Table 6. The Table 6 show one of the results obtained during the series of tests.

Figures 6, 7 and 8 show the state of the solutions obtained at different phases of the optimization process in relation to the awaited result. On the side of trajectory and energy, a large gap between the awaited solution and the solution obtained at the first iteration can be seen (Figure 6). At the halfway point (Figure 7), the gap between the obtained trajectory and the expected trajectory is considerably is. But on the other hand about the energy, the difference is created even more. This could be explained by the fact that

Table 3: Distribution of behaviours by time slot.

Time interval (t)	Behaviour	Equation
$t = 0$	Initialization	$\begin{cases} \varepsilon &= \varepsilon_0 \\ x &= 10 \\ y &= 0 \end{cases}$
$t \in]0, 30]$	The subject is at rest : sleeping	$\begin{cases} \varepsilon_t &= \varepsilon_{t-1} + G_s \\ x_t &= x_{t-1} + X_s cos(t) \\ y_t &= y_{t-1} + \frac{e}{Y_s} \end{cases}$
$t \in]30, 50]$	The subject is wandering around : wandering	$\begin{cases} \varepsilon_t &= \varepsilon_{t-1} - G_w \\ x_t &= (x_{t-1} + \frac{e}{X_w})\frac{1}{4} \\ y_t &= y_{t-1} sin(t) - Y_w \end{cases}$
$t \in]50, 80]$	The subject is feeding : Eating	$\begin{cases} \varepsilon_t &= \varepsilon_{t-1} + G_e \\ x_t &= x_{t-1} + \frac{1}{2} X_e \\ y_t &= y_{t-1} + \frac{e}{Y_e} \end{cases}$
$t \in]80, 100]$	The subject is wandering around : wandering	$\begin{cases} \varepsilon_t &= \varepsilon_{t-1} - G_w \\ x_t &= (x_{t-1} + \frac{e}{X_w})\frac{1}{4} \\ y_t &= y_{t-1} sin(t) - Y_w \end{cases}$

Table 4: Experimental conditions.

Maximum Iteration	500
Variables to be optimized	10
Population size	100
Cross-breeding ratio (Genetic algorithm)	60%

Table 5: Computer used.

Brand	DELL
Model	Latitude 5480
RAM	16 Go
Processor	Intel i57300HQ
CPU	2.500GHz x 4
Operating system	Ubuntu 20.04.1

Table 6: An example of the results obtained during the experiments.

found before the last iteration	No
Iteration of the solution	421
Adaptation of the solution	99.7862
Adaptation of the worst solution	100.1378

the variables which intervene in the definition of the trajectory are close to an optimal value, which is not the case for the energy side. At the end of the process, the solution obtained gives a result very similar to that of the dataset. In two out of fifty tests, we obtained exactly the awaited path, i.e. a solution with an adaptation equal to zero, which shows that we were managing to converge towards a solution we were expecting. This is also shown in Figures 9. We can see that very quickly we converge towards a solution and that over the iterations, the entire population is also greatly improved. This can be explained by the stabilization of the population around an optimal solution. The performance of each meta-heuristic is recorded in Table 7 and clearly reveals the superiority of the genetic algorithm over the immune algorithm for this type of problem. The performances of the genetic algorithm outshines the immune algorithm, with solutions found faster in less time with lower adaptation.

6 CONCLUSION AND PERSPECTIVES

In this paper, we have done preliminary work on the optimization of the parameterization of simulation models using meta-heuristics. We have observed that meta-heuristics allow us to have good results with a certain number of variables to be optimized within an acceptable period of time. However, interesting perspectives are contemplated and will be the subject of future work. At first, the architecture will be used to optimize the parameterization of other types of models, in particular machine learning models. Futher, the performance of this architecture and the algorithms implemented will be studied in order to improve the choice of parameters and to define the types of problems for which the use of such an architecture is suitable.

Table 7: Performance of each meta heuristic.

Meta-heuristics	solution with adaptation ≤ 200	Execution time average	Adaptation average
Genetic algorithm	42 out of 50	921 seconds	63.11389372
Immune algorithm	37 out of 50	1453 seconds	97.19834646

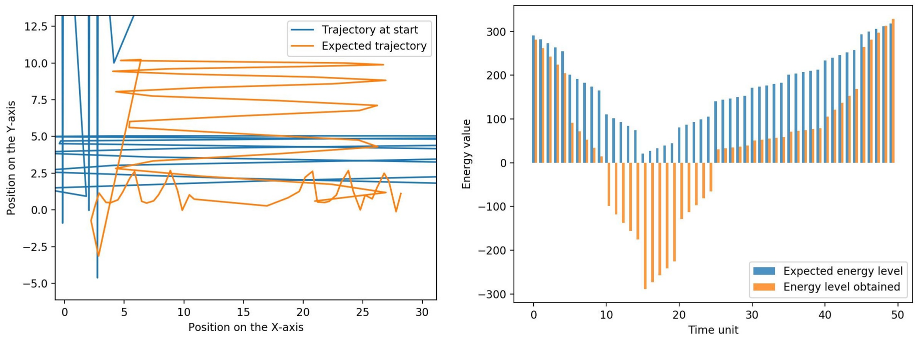

Figure 6: Comparison between the solution obtained at the beginning and the awaited one.

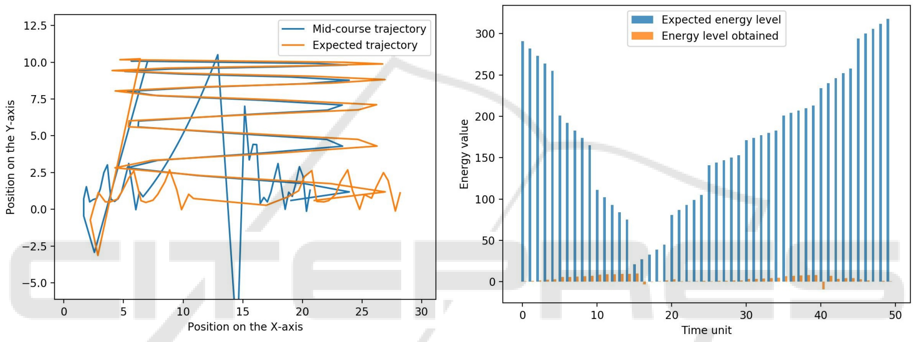

Figure 7: Comparison between the solution obtained at mid-term and the awaited one.

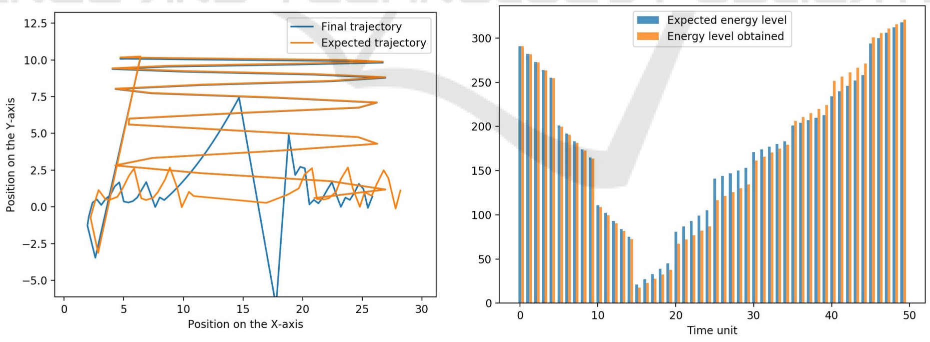

Figure 8: Comparison between the obtained solution and the awaited one.

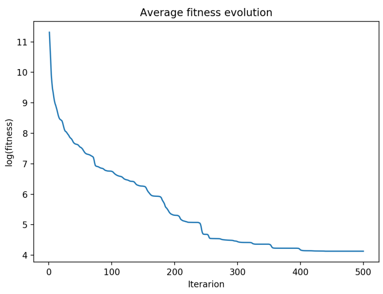

Figure 9: Evolution of the average adaptation of the population.

REFERENCES

Abdel-Basset, M., Abdel-Fatah, L., and Sangaiah, A. K. (2018). Metaheuristic algorithms : A comprehensive review. In *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, Intelligent Data-Centric Systems, pages 185–231.

Allegrini, J., Orehounig, K., Mavromatidis, G., Ruesch, F., Dorer, V., and Evins, R. (2015). A review of modelling approaches and tools for the simulation of district-scale energy systems. *Renewable and Sustainable Energy Reviews*, 52:1391–1404.

Darmoul, S. (2010). Etude de la contribution des systèmes immunitaires artificiels au pilotage de systèmes de production en environnement perturbé. 7.

Dave, E. (2011). *L'Internet des objets : Comment l'évolution actuelle d'Internet transforme-t-elle le monde ?* Livre blanc, Cisco IBSG.

Gandomi, A.; Xin-She, Y. T. S. A. A. (2013). Metaheuristic algorithms in modeling and optimization. *Metaheuristic Applications in Structures and Infrastructures*, 1.

Glover, F. and Sörensen, K. (2015). *Metaheuristics*. Scholarpedia.

Hutterer, S.; Auinger, F. A. M. S. G. (2010). Overview: A simulation based metaheuristic optimization approach to optimal power dispatch related to a smart electric grid. *Life System Modeling and Intelligent Computing*, 6329.

Koza, J. (1992). Genetic programming: On the programming of computers by means of natural selection.

Lee, K. Y. and El-Sharkawi, M. A. (2008). Modern heuristic optimization techniques: Theory and applications to power systems. John Wiley & Sons.

McCall, J. (2005). Genetic algorithms for modelling and optimisation. *Journal of Computational and Applied Mathematics*, 184(1):205–222. Special Issue on Mathematics Applied to Immunology.

Moxnes, E. (2015). *An Introduction to Deterministic and Stochastic Optimization*. Analytical methods for Dynamic Modelers, mit press edition.

Ojha, V. K., Abraham, A., and Snášel, V. (2017). Metaheuristic design of feedforward neural networks: A review of two decades of research. *Engineering Applications of Artificial Intelligence*, 60:97–116.

Oltean, M. (2005). Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13:387–410.

Poggi, B. (2014). *Développement de concepts et outils d'aide à la décision pour l'optimisation via simulation : intégration des métaheuristiques au formalisme DEVS*. Doctorat en Informatique, Université de Corse Pascal - Paoli.

Schichl, H. (2004). Models and the history of modeling. In Kallrath, J., editor, *Modeling Languages in Mathematical Optimization*. Kluwer Academic Publishers.

Sharma, A. and Sharma, A. (2014a). A review of modeling and simulation techniques. *International Journal for Research in Technological Studies*.

Sharma, A. and Sharma, A. (2014b). A review of modeling and simulation techniques. *International Journal for Research in Technological Studies*, 1.

Sharma, A. and Sharma, D. (2011). Clonal selection algorithm for classification. *Artificial Immune Systems, ICARIS*, 2011:361–370.

Shtovba, S. D. (2005). Ant algorithms : Theory and applications. *Programming and Computer Software*, 31:4.

Sopov, E. (2017). Genetic programming hyper-heuristic for the automated synthesis of selection operators in genetic algorithms. *IJCCI*.

Tarraq, A. Elmariami, F. B. A. H. T. (2021). Meta-heuristic optimization methods applied to renewable distributed generation planning: A review. *Conference: The International Conference on Innovation, Modern Applied Science & Environmental Studies (ICIES2020)*, 234.

Wong, W. and Ming, C. I. (2019). A review on metaheuristic algorithms: Recent trends, benchmarking and applications. In *2019 7th International Conference on Smart Computing Communications (ICSCC)*, pages 1–5.

Xin-She, Y. (2011). *Metaheuristic Optimization*. Scholarpedia.

Xin-She, Y. (2014). Nature-inspired optimization algorithms. pages 15–21.

Yin, C. and McKay2, A. (2018). Introduction to modeling and simulation techniques. In *The 8th International Symposium on Computational Intelligence and Industrial Applications (ISCIIA2018)*, pages 2–6, Shandong, China, P., 2018. Tengzhou.

Zeigler, P. (2017). How can modeling and simulation help engineering of system of systems? *Computational Frameworks, Systems, Models and Applications*, pages 1–46.

Zhang, J. (2011). Artificial immune algorithm to function optimization problems. pages 667–670.