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

Chabi Babatounde¹, Bastien Poggi¹, Thierry Antoine-Santoni¹ and Antoine Aiello²

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.
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 \( \{x_1, x_2, ..., x_n\} \) denoted \( S \), a set of constraints \( C \) and a goal function \( f : S \rightarrow \mathbb{R} \). Solving an optimization problem consists in finding the optimal values of \( x_1, x_2, ... \) and \( x_n \) which minimise or maximise the objective function on \( C \). Meta-heuristics are used to find these optimal values. Meta-heuristics (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 meta-heuristic 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.

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 individ-
uals. 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 $P_1$ and $P_2$ 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 $\{22, 25\}$ and $\{12, 25\}$.

It is also possible to make a combination of each variable from each parent according to a random constant $\alpha$ with $\alpha \in [0, 1]$. In this case, parents $P_1$ and $P_2$ give 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.

```plaintext
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 is formed by evaluation and standardization. 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 cloning 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.

  \[
  \text{clones} = \text{antibodies} \times \frac{1}{\text{adaptation}} \tag{1}
  \]

- **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.

<table>
<thead>
<tr>
<th>Genetic</th>
<th>Genetic algorithm</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set genome possible</td>
<td>Set of possible genes</td>
<td>Search space</td>
</tr>
<tr>
<td>Individual</td>
<td>Coded solution</td>
<td>Solution</td>
</tr>
<tr>
<td>Genome</td>
<td>Details of a solution</td>
<td>Values of each parameter</td>
</tr>
<tr>
<td>Adaptation</td>
<td>Fitness</td>
<td>Adaptation</td>
</tr>
<tr>
<td>Natural selection</td>
<td>Random selection</td>
<td>Diversification</td>
</tr>
<tr>
<td>Genetic mutation</td>
<td>Value change</td>
<td>Diversification</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Immune system</th>
<th>Immune Algorithm</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>All possible antibodies</td>
<td>All possible solutions</td>
<td>Search space</td>
</tr>
<tr>
<td>Antibody</td>
<td>Coded solution</td>
<td>Solution</td>
</tr>
<tr>
<td>Cell receptor</td>
<td>Details of a solution</td>
<td>Values of each parameter</td>
</tr>
<tr>
<td>Adaptation</td>
<td>Fitness</td>
<td>Adaptation</td>
</tr>
</tbody>
</table>

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

Algorithm 2: Immune Algorithm.

```python
immuneSystem = randomlyGenerated();
evaluate(immuneSystem);
normalize(immuneSystem);
while Not(Solution found OR stop condition) do
    clones = createClones(immuneSystem);
    clones = mutate(clones);
    evaluate(clones);
    normalize(clones);
    newAntibodies = generateNewClones();
    evaluate(newAntibodies);
    normalize(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
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.

```plaintext
enenergy = 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;
end if
end while
```

---

**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 sec-
tion 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 $\alpha$ 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.

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

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 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.

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

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.

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

![Graphical representation of the model (variables to be optimized in red).](image)

### 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.

<table>
<thead>
<tr>
<th>Time interval (t)</th>
<th>Behaviour</th>
<th>Equation</th>
</tr>
</thead>
</table>
| $t = 0$          | Initialization | $\begin{align*}
\varepsilon_t &= \varepsilon_0 \\
x_t &= x_0 + 10 \\
y_t &= y_0
\end{align*}$ |
| $t \in [0, 30]$  | The subject is at rest: sleeping | $\begin{align*}
\varepsilon_t &= \varepsilon_{t-1} + G_s \\
x_t &= x_{t-1} + X_s \cos(t) \\
y_t &= y_{t-1} + \frac{Y_s}{\varepsilon_{t-1}}
\end{align*}$ |
| $t \in [30, 50]$ | The subject is wandering around: wandering | $\begin{align*}
\varepsilon_t &= \varepsilon_{t-1} - G_w \\
x_t &= (x_{t-1} + \frac{X_w}{\varepsilon_{t-1}})^{\frac{1}{2}} \\
y_t &= y_{t-1} - \frac{Y_w}{\varepsilon_{t-1}} \sin(t) - Y_w
\end{align*}$ |
| $t \in [50, 80]$ | The subject is feeding: Eating | $\begin{align*}
\varepsilon_t &= \varepsilon_{t-1} + G_e \\
x_t &= (x_{t-1} + \frac{X_e}{\varepsilon_{t-1}})^{\frac{1}{2}} \\
y_t &= y_{t-1} + \frac{Y_e}{\varepsilon_{t-1}}
\end{align*}$ |
| $t \in [80, 100]$ | The subject is wandering around: wandering | $\begin{align*}
\varepsilon_t &= \varepsilon_{t-1} - G_w \\
x_t &= (x_{t-1} + \frac{X_w}{\varepsilon_{t-1}})^{\frac{1}{2}} \\
y_t &= y_{t-1} - \frac{Y_w}{\varepsilon_{t-1}} \sin(t) - Y_w
\end{align*}$ |

Table 4: Experimental conditions.

<table>
<thead>
<tr>
<th>Maximum Iteration</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables to be optimized</td>
<td>10</td>
</tr>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Cross-breeding ratio (Genetic algorithm)</td>
<td>60%</td>
</tr>
</tbody>
</table>

Table 5: Computer used.

<table>
<thead>
<tr>
<th>Brand</th>
<th>DELL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Latitude 5480</td>
</tr>
<tr>
<td>RAM</td>
<td>16 Go</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel i57300HQ</td>
</tr>
<tr>
<td>CPU</td>
<td>2.500GHz x 4</td>
</tr>
<tr>
<td>Operating system</td>
<td>Ubuntu 20.04.1</td>
</tr>
</tbody>
</table>

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. Further, 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.

<table>
<thead>
<tr>
<th>Meta-heuristics</th>
<th>solution with adaptation ≤ 200</th>
<th>Execution time average</th>
<th>Adaptation average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genetic algorithm</td>
<td>42 out of 50</td>
<td>921 seconds</td>
<td>63.11389372</td>
</tr>
<tr>
<td>Immune algorithm</td>
<td>37 out of 50</td>
<td>1453 seconds</td>
<td>97.19834646</td>
</tr>
</tbody>
</table>

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


