 doi:10.20944/preprints202101.0048.v1

Letter

AutoMH: Automatically Create Evolutionary Metaheuristic Algorithms Using Reinforced Learning

Boris Almonacid [1,2,†]

[1] Global Change Science; boris.almonacid@globalchange.science
[2] CoronaWhy.
† An extended version is currently in peer review.

Received: 31 December 2020

Abstract: Machine learning research has been able to solve problems in multiple aspects. An open area of research is machine learning for solving optimisation problems. An optimisation problem can be solved using a metaheuristic algorithm, which is able to find a solution in a reasonable amount of time. However, there is a problem, the time required to find an appropriate metaheuristic algorithm, that would have the convenient configurations to solve a set of optimisation problems properly. A solution approach is shown here, using a proposal that automatically creates metaheuristic algorithms aided by a reinforced learning approach. Based on the experiments performed, the approach succeeded in creating a metaheuristic algorithm that managed to solve a large number of different continuous domain optimisation problems. This work's implications are immediate because they describe a basis for the generation of metaheuristic algorithms in real-time.

Keywords: Artificial intelligence; Machine Learning; Reinforced Learning; Optimisation; Metaheuristic; Metaheuristic Generation

1. Introduction

The use of metaheuristic algorithms has become an approach that is widely used to solve a variety of optimisation problems [1], such as optimisation problems in the field of health, logistics, agriculture, mining, space, robotics, to name a few. The diversity of metaheuristic algorithms in the last decade has grown widely [1], with a great diversity of components, routines, selectors, internals, and especially a great variety of parameters. This diversity leads to different difficulties, such as, for example, being able to find a specific configuration of parameters for a specific type of optimisation problem. A situation that induces and generates a difficulty in being able to choose a metaheuristic algorithm adequately.

Various strategies have been adopted to minimise the effort of manual configurations. One area is machine learning, specifically in reinforced learning [2], where various advances have been made. For example, a general method to reformulate reinforced learning problems as optimisation tasks and apply the particle swarm metaheuristic algorithm to find optimal solutions [3]. A solution to solve the vehicle routing problem [4], Feature Selection [5], a design of a plane frame [6], or Resource Allocation problems [7]. Other approaches include Learnheuristics [8], Q-Learning [9], Meta-learning [10], and Hyper-heuristic [11,12], which provide diverse perspectives on tackling optimisation problems. In [13] the use of Multi-agent reinforcement learning is proposed, which allow for an upgrade in the reinforcement learning area, which generally uses a single agent.

In algorithm generation, there is an approach that uses the construction of a centralised hybrid metaheuristic cooperative strategy to solve optimisation problems [14]. Another approach, [15] uses a set of instructions to create a set of machine learning algorithms in real-time. A basis for understanding the scope of these approaches can be found in [16] which provides the taxonomy of combinations with

 doi:10.20944/preprints202101.0048.v1

metaheuristics, with mathematical programming, with constraint programming and machine learning. Open problems and a current status of the area can be found in [17], and [18].

For this research, an extension of the basic model of reinforced learning is proposed. The Agent will be called the learning agent, which contains two processes called the analysis process and the action process. The environment is the one that integrates a more significant number of changes because in addition to incorporating a set of optimisation problems to solve, it combines a swarm of non-intelligent agents, which have the purpose of executing the optimisation problems in a metaheuristic algorithm that improves its internal structure during each evolutionary process. The final objective is to be able to find through an evolutionary generation process the best metaheuristic algorithm(s) that solve the set of problems entered by the user.

In the scope of this research, he focuses on contributing within the area of High-level data-driven metaheuristics, specifically on the topic of *metaheuristic generation* by *reinforcement learning*. The main benefits expected from this work are as follows:

- Find new metaheuristic algorithms in real-time that solve one or some optimisation problems given by the user.
- Allow the extension of new components to generate new metaheuristic algorithms, as new operators or as new indivisible exploration intensification functions.
- Contribute to the area of machine learning, specifically in the integration of reinforced learning to solve optimisation problems.

The rest of this paper is structured as follow: In section 2, the proposed design and the formalisation of its parts are detailed. In section 3, the tests performed and their results are detailed. Finally, in section 4 concludes and provides guidelines for future work.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

2. Materials and Methods

2.1. Proposed reinforcement learning framework for the automatic creation of metaheuristics

The proposed reinforcement learning framework (AutoMH) is to be able to find through a learning agent one or a set of metaheuristic algorithms that are capable of finding the best solution for a portfolio of optimisation problems P.

Definition 1. *A continuous optimisation problem P is defined by minimise $p(x)$ subject to $l \leq x \leq u$, where $x = \{x_1, x_2, \ldots, x_d\}$, d is the dimensions of the optimisation problem, $l = \{l_1, l_2, \ldots, l_d\}$, and $u = \{u_1, u_2, \ldots, u_d\}$ are the lower bounds and the upper bounds of the corresponding variables in x, which define the feasible domain of the problem.*

The figure 1 details the main parts of the framework architecture. It consists of two basic parts of a reinforcement learning system: the *Learning Agent* and the *Environment*.

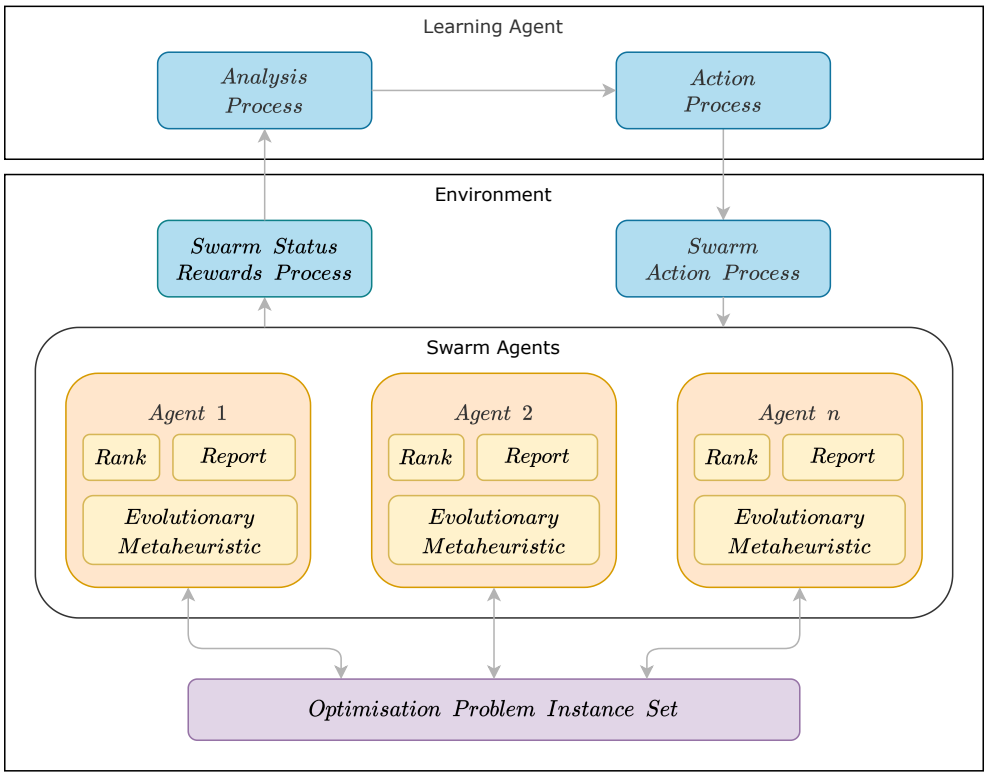

Figure 1. Proposed reinforcement learning framework for the automatic creation of metaheuristics

Learning Agent: The learning agent has the function of analysing the data generated by the environment through the *analysis process* and taking actions that will affect through a set of actions the internal behaviour of each agent in the swarm of agents through the *action process*.

Environment: The environment is composed of three elements:

- A Optimisation Problem Instance Set: Corresponds to a portfolio of optimisation problems $P = \{p_1, p_2, \ldots, p_n\}$ that must be resolved by the agents. An example of optimisation problems is described in the Appendix A, and in the table 5.
- A Swarm Agents: It is a set of non-intelligent agents $A = \{A_1, A_2, \ldots, A_n\}$.
- Two Swarm Process: The *swarm status rewards process* to extract information, and the *swarm action process* to add new information to the agents.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

Swarm Agents: Each agent separately is in charge of executing the tests of the set of optimisation problems P. Formally, an agent is determined by Definition 2.

Definition 2. *A Agent A_i is defined by the 3-tuple $A_i = \langle M, Q, R \rangle$, where:*

- *A metaheuristic algorithm M, which is an empty structure named template. This structure is modified at runtime through the swarm action process.*
- *A qualification Q corresponds to a variable that indicates the value of the rank assigned to the agent.*
- *A report R corresponds to a set of data structures in which the results of the optimisation tests are stored. The stored data correspond to a matrix of summaries with best, worst, mean, std, fitness, and solution for each optimisation problem, and a matrix of details of each iteration for each execution of each optimisation problem.*

2.2. Instruction

An instruction is an ordered grouping of elements with the objective of producing a change in the value of a variable. An instruction is made up of four elements: a variable, an assignment operator, an operator, and a function.

The composition of an instruction is detailed in the equation 1. Where, from right to left: f_{x_t} is the function that is applied using the current value of the variable x_t in order to generate a new value, Δ is the operator that will be applied with the value of the variable x_t and with the value obtained by applying the function f_{x_t}, and the symbol $\leftarrow$ is the assignment operator for a new value that will be assigned in x_{t+1}.

$$\begin{array}{cccccc} \textit{variable} & \textit{assignment} & \textit{operator} & \textit{variable} & \textit{operator} & \textit{function} \\ x_{t+1} & \leftarrow & & x_t & \Delta & f(x_t) \end{array} \tag{1}$$

Formally, an instruction is determined by Definition 3. Additionally, instructions can derive into instruction types such as an *exploration instruction* ε, which is defined by function 2, or an *intensification instruction* γ that is defined by function 3.

Definition 3. *An instruction I is composed by a variable x, one generic instruction operator $O = \{\Delta_k(x) \mid k \in K\}$, and one exploration function $G = \{g_j(x) \mid j \in J\}$ or one intensification function $H = \{h_i(x) \mid i \in I\}$, where $K = \{1 \dots l\}$, $J = \{1 \dots m\}$, and $I = \{1 \dots n\}$. The value of n, m, and l are determined by the initial information of the system.*

$$\varepsilon \Leftrightarrow f_1(x, \Delta_k, g_j) = x\ \Delta_k\ g_j(x) \tag{2}$$

$$\gamma \Leftrightarrow f_2(x, \Delta_k, h_i) = x\ \Delta_k\ h_i(x) \tag{3}$$

For this research, a portfolio of functions with exploration and intensification instructions described in tables 1, 2 and 3 will be used.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

Table 1. List of basic functions.

Identifier	Name	Function	Code
I01	Sine	$f(x) = sin(x)$	x = SIN(X)
I02	Cosine	$f(x) = cos(x)$	x = COS(X)
I03	Tangent	$f(x) = tan(x)$	x = TAN(X)
I04	Inverse Sine	$f(x) = arcsin(x)$	x = ARCSIN(X)
I05	Inverse Cosine	$f(x) = arccos(x)$	x = ARCCOS(X)
I06	Inverse Tangent	$f(x) = arctan(x)$	x = ARCTAN(X)
I07	Absolute	$f(x) = \|x\|$	x = ABS(X)
I08	Square root	$f(x) = \sqrt{x}$	x = SQRT(X)
I09	Exponential function	$f(x) = e^x$	x = EXP(X)
I10	Exponential function minus 1	$f(x) = e^x - 1$	x = EXP1(X)
I11	Natural logarithm	$f(x) = \ln(x)$	x = LN(X)
I12	Base-2 logarithm of x	$f(x) = \log_2(x)$	x = LOG2(X)
I13	Base-10 logarithm of x	$f(x) = \log_{10}(x)$	x = LOG10(X)
I14	natural logarithm of one plus	$f(x) = \ln(1 + x)$	x = LN(1 + X)

Table 2. List of random number functions.

Identifier	Name	Function	Code	Description
I100	Uniform F1	$U_{f1} \sim (l, u)$	x = UNIFORM1(L, U)	
I101	Uniform F2	$U_{f2} \sim (l, u)$	x = UNIFORM2(L, U)	u = lb + (ub - lb)/2
I102	Uniform F3	$U_{f3} \sim (l, u)$	x = UNIFORM3(L, U)	l = lb + (ub - lb)/2
I103	Uniform F4	$U_{f4} \sim (l, u)$	x = UNIFORM4(L, U)	l = lb + (ub - lb)/3 u = lb + (ub - lb)/3*2
I104	Uniform F5	$U_{f5} \sim (l, u)$	x = UNIFORM5(L, U)	u = lb + (ub - lb)/4
I105	Uniform F6	$U_{f6} \sim (l, u)$	x = UNIFORM6(L, U)	l = lb + (ub - lb)/4 u = lb + (ub - lb)/2
I106	Uniform F7	$U_{f7} \sim (l, u)$	x = UNIFORM7(L, U)	l = lb + (ub - lb)/2 u = lb + (ub - lb)/4*3
I107	Uniform F8	$U_{f8} \sim (l, u)$	x = UNIFORM8(L, U)	l = lb + (ub - lb)/4*3
I108	Uniform F9	$U_{f9} \sim (-1, 1)$	x = UNIFORM9(-1, 1)	
I109	Uniform F10	$U_{f10} \sim (0, 1)$	x = UNIFORM10(0, 1)	
I110	Uniform F11	$U_{f11} \sim (-1, 0)$	x = UNIFORM11(-1, 0)	
I111	Uniform F12	$U_{f12} \sim (0.5, 0.5)$	x = UNIFORM12(0.5, 0.5)	
I112	Beta F1	$B_{f1} \sim (0.5, 0.5, 1)$	x = BETA1(0.5, 0.5, 1)	
I113	Beta F2	$B_{f2} \sim (5, 1, 1)$	x = BETA2(5, 1, 1)	
I114	Beta F3	$B_{f3} \sim (1, 3, 1)$	x = BETA3(1, 3, 1)	
I115	Beta F4	$B_{f4} \sim (2, 2, 1)$	x = BETA4(2, 2, 1)	
I116	Beta F5	$B_{f5} \sim (2, 5, 1)$	x = BETA5(2, 5, 1)	
I117	Triangular F1	$T_{f1} \sim (lb, m, ub)$	x = TRIANGULAR1(LB, M, UB)	m = lb + (ub - lb)/2
I118	Triangular F2	$T_{f2} \sim (lb, m, ub)$	x = TRIANGULAR2(LB, M, UB)	m = lb + (ub - lb)/4
I119	Triangular F3	$T_{f3} \sim (lb, m, ub)$	x = TRIANGULAR3(LB, M, UB)	m = lb + (ub - lb)/3
I120	Triangular F4	$T_{f4} \sim (lb, m, ub)$	x = TRIANGULAR4(LB, M, UB)	m = lb + ((ub - lb)/4)*3
I121	Triangular F5	$T_{f5} \sim (lb, m, ub)$	x = TRIANGULAR5(LB, M, UB)	m = lb + ((ub - lb)/3)*2

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

Table 3. List of constants.

ID	Name	Symbol	Value	Code
I200	Meissel–Mertens	M_1	0.26149 72128 47642 78375 54268 38608 69585	A077761( )
I201	Bernstein's	β	0.28016 94990 23869 13303	A073001( )
I202	Gauss–Kuzmin–Wirsing	λ	0.30366 30028 98732 65859 74481 21901 55623	A038517( )
I203	Hafner–Sarnak–McCurley	σ	0.35323 63718 54995 98454 35165 50432 68201	A085849( )
I204	Omega	Ω	0.56714 32904 09783 87299 99686 62210 35554	A030178( )
I205	Euler–Mascheroni	γ	0.57721 56649 01532 86060 65120 90082 40243	A001620( )
I206	Twin prime	C_2	0.66016 18158 46869 57392 78121 10014 55577	A005597( )
I207	Conway's	λ_c	1.30357 72690 34296 39125 70991 12152 55189	A014715( )
I208	Ramanujan–Soldner	μ	1.45136 92348 83381 05028 39684 85892 02744	A070769( )
I209	Golden ratio	φ	1.61803 39887 49894 84820 45868 34365 63811	A001622( )
I210	Euler's number	e	2.71828 18284 59045 23536 02874 71352 66249	A001113( )
I211	Pi	π	3.14159 26535 89793 23846 26433 83279 50288	A000796( )
I212	Reciprocal Fibonacci	ψ	3.35988 56662 43177 55317 20113 02918 92717	A079586( )

Definition 4. *An operator Δ is a mathematical symbol that indicates that a specific operation must be performed on a variable and a function.*

Table 4. List of operators.

Identifier	Name	Math	Code
O00	None	$x \leftarrow x$	x = x
O01	Plus	$x \leftarrow x + f(x)$	x = x + f(x)
O02	Subtract	$x \leftarrow x - f(x)$	x = x - f(x)
O03	Multiply	$x \leftarrow x * f(x)$	x = x * f(x)
O04	Divide	$x \leftarrow \frac{x}{f(x)}$	x = x / f(x)

2.3. Evolutionary metaheuristic algorithm

A metaheuristic algorithm M is a template that changes in each cycle depending on the decisions made by the learning agent through the action process. Changes to its structure are made at runtime through the swarm action process. The structure of the metaheuristic algorithm template is constituted by the SETUP, STEP, and END functions.

Definition 5. *An evolutionary metaheuristic algorithm is defined by the 3-tuple $M = \langle \tau, E, \Gamma \rangle$, where: τ is a metaheuristic template that is composed by* SETUP, STEP, END, *and* RUN *functions, E is a dynamic array of exploration functions $E = [\epsilon_1, \epsilon_2, \ldots, \epsilon_n]$, and Γ is a set of intensification functions $\Gamma = [\gamma_1, \gamma_2, \ldots, \gamma_n]$.*

The description of the functions are described below:

- The STARTED function is in charge of initialising the variables of the optimisation problem. Initialisation is carried out using one or more scan instructions. Subsequently, the current fitness is calculated and the solution is stored.
- The STEP function is the main core of the template. In this function the main modifications are made in the evolutionary metaheuristic algorithm. Actions are carried out such as adding, modifying or deleting instructions both of the type of exploration instructions, as well as intensification instructions. Subsequently, the new aptitude of the solution is calculated, and the new aptitude and solution is stored in the event that it is better than the previous one.
- The END function is executed when the end condition of the metaheuristic algorithm ends. Its function is to extract the solution found and its associated aptitude.

Figure 2 describes an example of a template that has already been modified by the learning agent. The STARTED function has a single instruction that is composed of the operator NONE with the code

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

O00, and by the UNIFORM10(0, 1) function with the code I109. The STEP function is composed of a exploration instruction <O02, I06> and two intensification instructions <O02, I06> and <O03, I14>.

```
# Example metaheuristic template with instructions.

def Started(P=0.5):
    # Exploration instruction.
    x = uniform10()            # <O00, I109>
    # Calculate fitness.
    current_fitness = f(x)
    solution = x

def Step():

    if random_number < P:
        # Exploration instructions.
        x = x + triangular2()  # <O01, I118>
    else:
        # Intensification instructions.
        x = x - arctan(x)      # <O02, I06>
        x = x * ln(1+x)        # <O03, I14>

    # Calculate fitness for minimization problems.
    new_fitness = f(x)
    if new_fitness < current_fitness:
        # Update fitness, and solution.
        current_fitness = new_fitness
        solution = x
        return True
    else:
        return False

def End():
    return x, solution
```

Figure 2. Metaheuristic Template

The RUN function has the function of executing the STARTED, STEP and END functions. From executing the Started function to initialise the algorithm, executing the while loop with the calls to the STEP function, and ending the algorithm when the stopping criterion is met. When this condition is fulfilled, the End function will be called to extract the final solution of the optimisation problem P. The pseudocode of the RUN function is described in figure 3.

```
# Parent template
def Run():
  # Create a empty solution
  solution = None
  # Started function.
  Started()
  # Execute while the term criterion is not met.
  while term_condition_is_met() is False:
    # Step function.
    result = Step()
    # If there is a new result.
    if result is True
      # Update solution.
      solution = get_solution()
    term_condition_update()
  # End function.
  End()
```

Figure 3. Parent Template

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

2.4. Swarm Status Rewards Process

The *Swarm Status Rewards Process* consists of a process to collect the information generated by the swarm of agents when executing the metaheuristic algorithm. Store the information for each problem P, for each execution of the problem P, the results regarding the best solution, the fitness, and the fitness values for each iteration. The values are stored in a structure called Report. $Report = \{best\_solutions, best\_fitness, fitness\_iterations\}$.

2.5. Analysis Process

The objective is to rank each agent in the swarm. The measure used to order the agents is the average value obtained for each optimisation problem. The average value is represented by the matrix $Q \in \mathbb{R}^{m \times n}$ (See 4), where: $m \in \{1, p\}$ $n \in \{1, a\}$, p is the number of optimisation problems and a is the number of agents in the swarm.

$$Q_{m,n} = \begin{bmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,n} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m,1} & q_{m,2} & \cdots & q_{m,n} \end{bmatrix} \tag{4}$$

Subsequently, a series of operations are carried out:

1. The assignment of ranges is carried out using the data provided by the Q matrix. The method used is the minimum (competition) method, which consists in that to perform the ranking to each value, the minimum of the ranges that would have been assigned is assigned to all tied values. The ranking result is stored in matrix $R \in \mathbb{N}^{m \times n}$ (See 5), where: $m \in \{1, p\}$ $n \in \{1, a\}$, p is the number of optimisation problems and a is the number of agents in the swarm. The minimum method is performed for each row of the matrix Q, which will consider that each problem will have its own ranking among all the agents. The ranking result for each row will be stored in matrix R. (See 6), where: $n \in \{1, a\}$, and a is the number of agents in the swarm.

$$R_{m,n} = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,n} \end{bmatrix} \tag{5}$$

2. A sum is made by the value of each column in matrix R. Each sum will correspond to the final ranking value for each agent in the swarm. The values of each sum are stored in a vector $S \in \mathbb{N}^n$

$$S_n = \begin{bmatrix} \sum_{i=1}^{m} r_{i,1} & \sum_{i=1}^{m} r_{i,2} & \cdots & \sum_{i=1}^{m} r_{i,n} \end{bmatrix} \tag{6}$$

2.6. Action Process

The *action process* takes the information generated by the *analysis process* and performs a swarm modification procedure. The steps performed are:

1. Sort the swarm agents from the best ranking to the worst ranking.
2. Count in a variable c the number of agents with the best ranking.
3. Eliminate an amount c of the worst agents of the swarm.
4. Make a copy of the best agents in an amount c and add them to the end of the swarm.
5. Mark the best agents in an amount c with the status NONE.
6. Mark the rest of the agents with the status MODIFY.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

The status NONE means that the agent will not have any modifications made to its metaheuristic algorithm. The status MODIFY means that the agent is enabled to carry out modifications in the structure of its metaheuristic algorithm.

2.7. Swarm Action Process

The Action Process has the function of making a modification in the agents of the swarm that have the MODIFY state. To carry out the modifications, each agent obtains a random integer by means of a uniform discrete distribution, the value obtained will correspond to a type of action that will modify the instruction structure of the metaheuristic algorithm. The allowed instructions are: {ADD, REPLACE, REMOVE}, for instructions like: {INTENSIFICATION INSTRUCTION, EXPLORATION INSTRUCTION}. Figure 4 shows the three types of modifications that are made in the metaheuristic algorithm.

From these modifications, the agent is able to repeat the optimisation tests again, in order to observe if the changes in its structure generate better or worse results.

Figure 4. Example of modifications in the structure of the metaheuristic algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

3. Experiments

This section describes the characteristics of the tests and the results obtained by their execution.

Methodology: The methodology consists of collecting a portfolio of continuous function optimisation problems. Perform configuration learning system with a fixed number of agents, evolutionary iterations to be performed, and the type of mutation. For each non-intelligent agent, including the number of iterations and the number of executions each metaheuristic algorithm will perform. Finally, incorporate a list of intensification functions, exploration functions, and operators that will be used to create the instructions. Table 5 indicates the parameters used in the experiment.

Environment: The experiment was developed in Python language version 3.8.5, running on a MSI P65 Creator 9SE laptop with Intel Core I7 9750H CPUs @ 2.60Ghz, 16 GB RAM, and Windows 10 Pro Os build 19041.685.

Table 5. Parameters used in the tests.

Name	Value	Description
Number of agents	30	The number of agents in the swarm.
Evolutionary iterations	200	The number of times the agents in the swarm have to repeat the optimisation tests once the structure of their algorithm is modified by the *learning agent*.
Mutation Selection	Random	The mutation is carried out by randomly choosing an action of the type {ADD, REPLACE, REMOVE}.
Metaheuristic iterations	100	The maximum number of iterations that the metaheuristic executes.
Metaheuristic executions	5	The number of times the metaheuristic is executed.
Operators	O01, O02, O03, O04	The operators allowed to modify the metaheuristic template.
Started	I100, I103, I108, I109, I110, I111	The exploration functions allowed for modifying the metaheuristic template in the STARTED function.
Intensification	I01, I02, I03, I04, I05, I06, I07, I08, I09, I10, I11, I12, I13, I14	The intensification functions allowed for the modification of the metaheuristic template in the STEP function.
Exploration	I100, I101, I102, I103, I104, I105, I106, I107, I108, I109, I110, I111, I112, I113, I114, I115, I116, I117, I118, I119, I120, I121, I200, I201, I202, I203, I204, I205, I206, I207, I208, I209, I210, I211, I212	The exploration instructions allowed for the modification of the metaheuristic template in the STEP function.
Dimension	20	The dimension of the optimisation problems.

Description of the dataset: Table 6 describes 13 types of problems collected in the literature. Problems P01 to P07 correspond to unimodal functions, and problems P08 to P13 correspond to multimodal functions. Detailed descriptions of these functions are given in Appendix A.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

Table 6. Continuous optimisation problem dataset

Identifier	Function Name	Domain	$f_{min}(x^*)$	$x^* = [x_1, x_2, \ldots, x_n]$	Details
P01	Sphere	[-100, 100]	0	f(0, 0, ..., 0)	A1
P02	Schwefel Function 2.22	[-10, 10]	0	f(0, 0, ..., 0)	A2
P03	Schwefel Function 1.2	[-100, 100]	0	f(0, 0, ..., 0)	A3
P04	Schwefel Function 2.21	[-100, 100]	0	f(0, 0, ..., 0)	A4
P05	Rosenbrock's	[-30, 30]	0	f(1, 1, ..., 1)	A5
P06	Step	[-100, 100]	0	f($x_1, x_2, \ldots, x_d$), $x_i \in [-0.5, 0.5)$, $i = \{1, 2, \ldots, d\}$	A6
P07	Quartic	[-1.28, 1.28]	0	f(0, 0, ..., 0)	A7
P08	Schwefel Function 2.26	[-500, 500]	0	f(4.209 687 462 275 036e2, ..., 4.209 687 462 275 036e2)	A8
P09	Rastrigin	[-5.12, 5.12]	0	f(0, 0, ..., 0)	A9
P10	Ackley	[-32, 32]	0	f(0, 0, ..., 0)	A10
P11	Griewank	[-600, 600]	0	f(0, 0, ..., 0)	A11
P12	Generalized Penalized Function 1	[-50, 50]	0	f(1, 1, ..., 1)	A12
P13	Generalized Penalized Function 2	[-50, 50]	0	f(1, 1, ..., 1)	A13

3.1. Results

Table 7 describes the results obtained at the end of the test with the 200 evolutionary iterations. This result corresponds to the best agent in the swarm. The results are the best, and worst fitness found for 13 continuous optimisation problems. In addition, the average and standard deviation of the fitness have been incorporated.

Table 7. The description of the table has the following attributes: Column 1 (Identifier) indicates the assigned identifier for each test instance. Column 2 (Best) indicate the best optimum value found. Column 3 (Mean), and column 4 (Std.) describe the mean, and standard deviation respectively Column 5 (Worst) indicate the worst optimum value found.

Identifier	Best	Mean	Std.	Worst
P1	0.0000	0.0000	0.0000	0.0000
P2	0.0000	0.0000	0.0000	0.0000
P3	0.0000	0.0000	0.0000	0.0000
P4	0.0000	0.0000	0.0000	0.0000
P5	18.8171	18.8171	0.0000	18.8171
P6	0.0000	0.0000	0.0000	0.0000
P7	0.0000	0.0000	0.0000	0.0000
P8	4767.8746	4767.8746	0.0000	4767.8746
P9	0.0000	0.0000	0.0000	0.0000
P10	0.0000	0.0000	0.0000	0.0000
P11	0.0000	0.0000	0.0000	0.0000
P12	0.1167	0.2373	0.0749	0.3375
P13	1.0750	1.0750	0.0000	1.0750

The instructions with the best algorithm found during the 200 evolutionary iterations are:

Started function: Instruction for generating random solution.

1. <O00, I111>

Step function: Intensification instructions.

1. <O04, I01>
2. <O03, I12>

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

3. <O01, I10>

Step function: Exploration instructions.

1. <O04, I202>
2. <O04, I200>
3. <O03, I117>
4. <O02, I109>
5. <O02, I200>

Figures 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, and 17 show the convergence of the evolutionary algorithm during all iterations for each execution performed. Additionally, it has built an enlarged view of convergence when the metaheuristic find the best fitness.

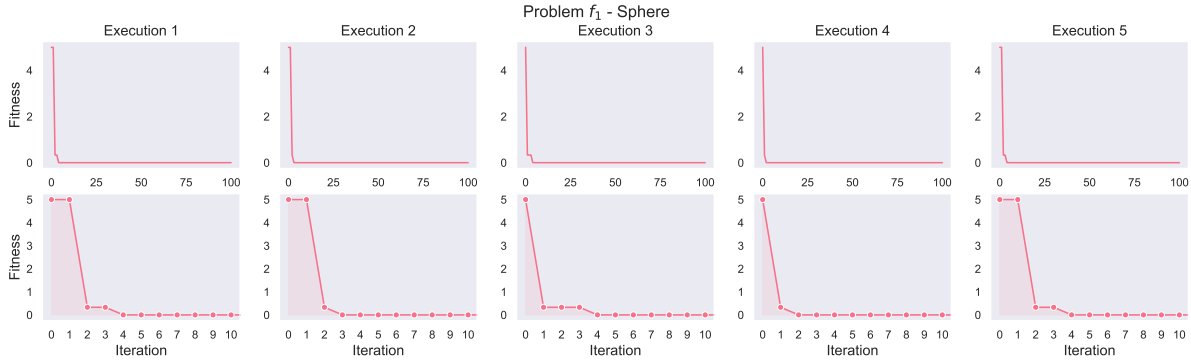

Figure 5. Convergence graph for five executions of problem F1 - Sphere. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

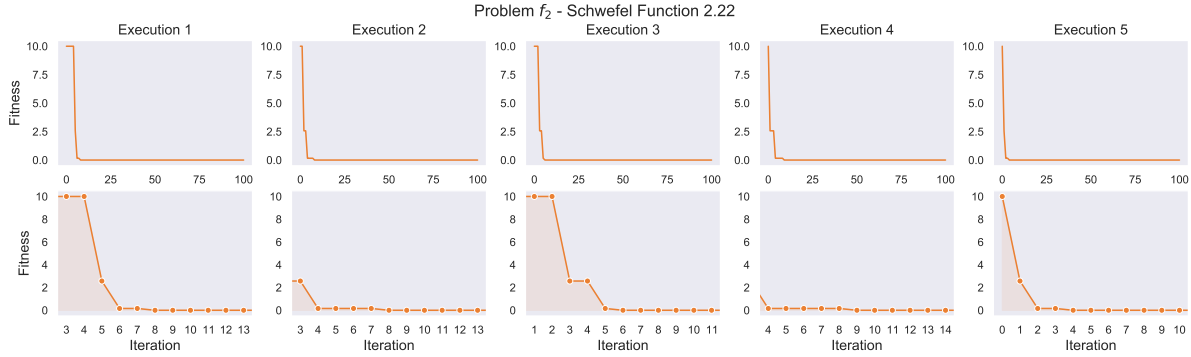

Figure 6. Convergence graph for five executions of problem F2 - Schwefel Function 2.22. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

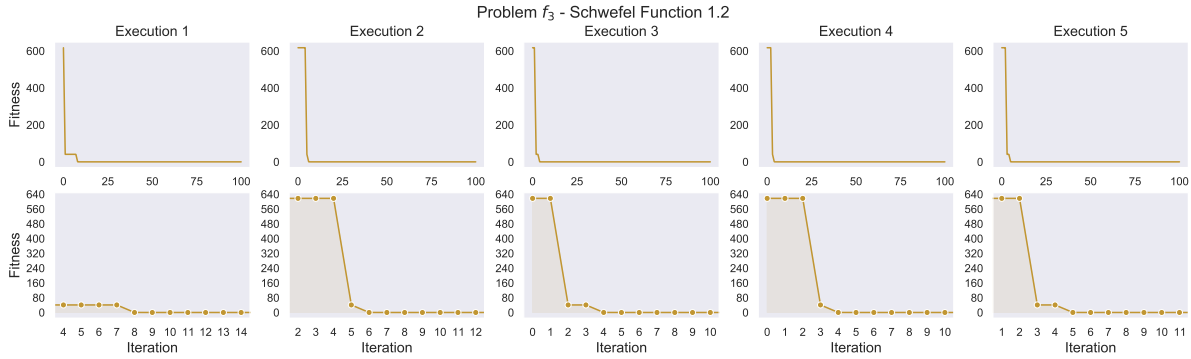

Figure 7. Convergence graph for five executions of problem F3 - Schwefel Function 1.2. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

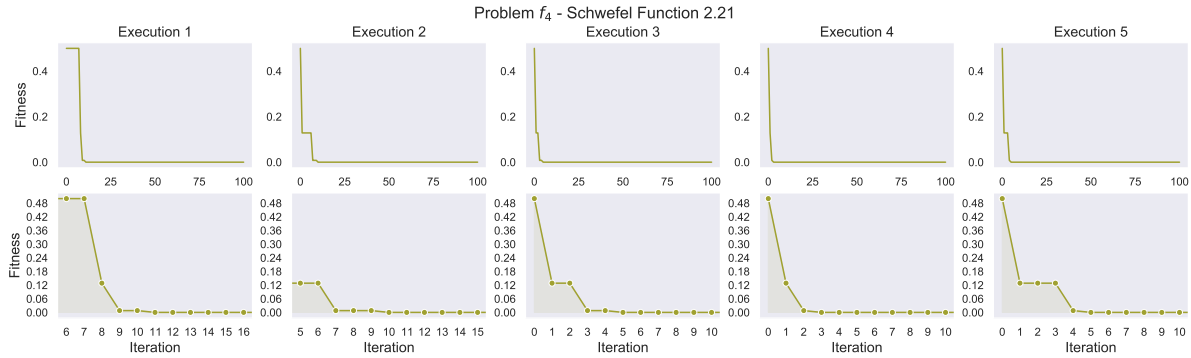

Figure 8. Convergence graph for five executions of problem F4 - Schwefel Function 2.21. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

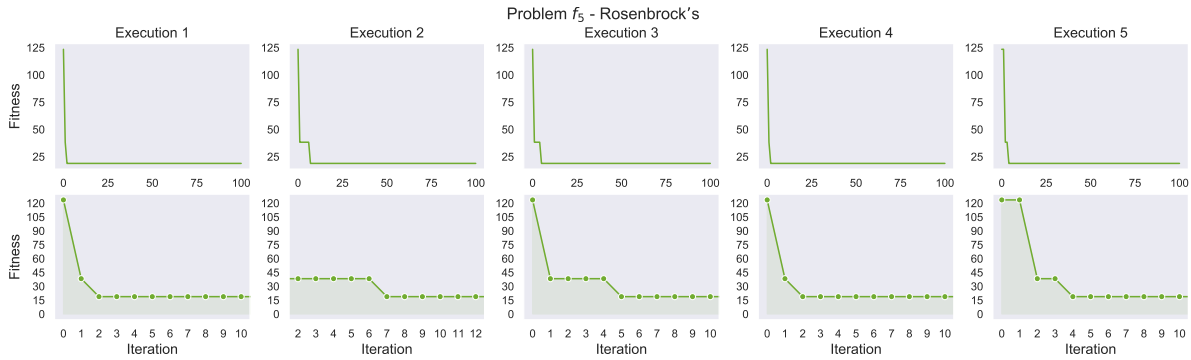

Figure 9. Convergence graph for five executions of problem F5 - Rosenbrock's. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

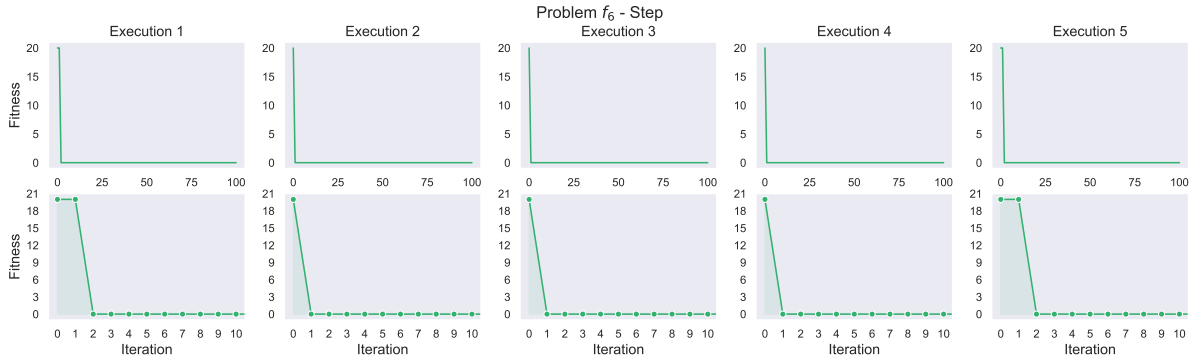

Figure 10. Convergence graph for five executions of problem F6 - Step. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

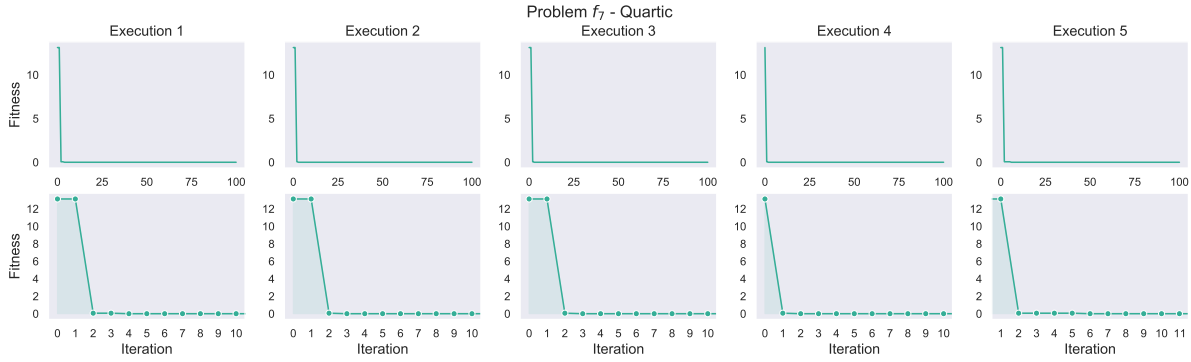

Figure 11. Convergence graph for five executions of problem F7 - Quartic. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

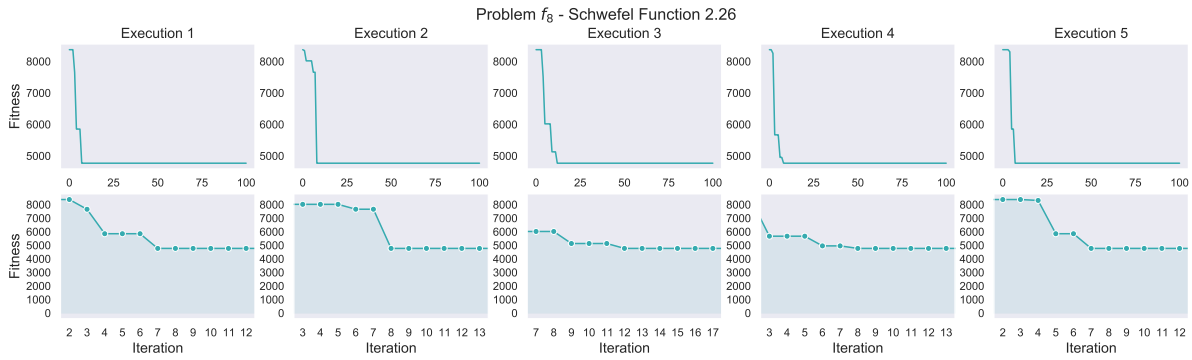

Figure 12. Convergence graph for five executions of problem F8 - Schwefel Function 2.26. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

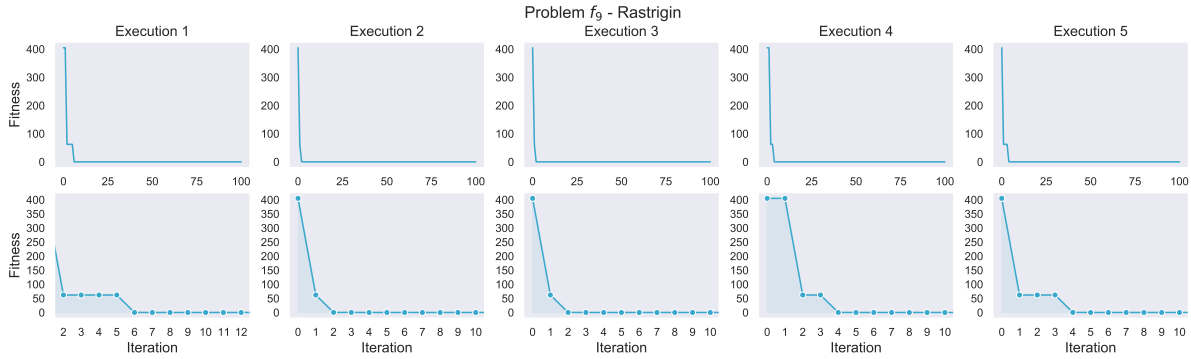

Figure 13. Convergence graph for five executions of problem F9 - Rastrigin. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

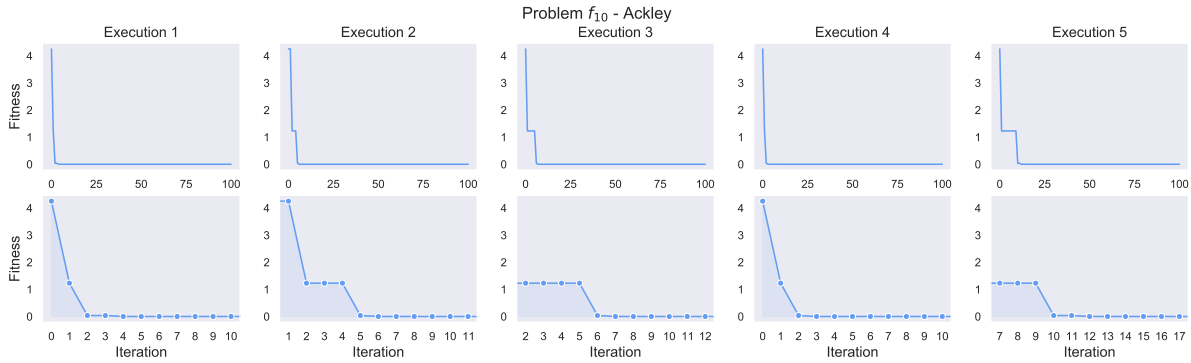

Figure 14. Convergence graph for five executions of problem F10 - Ackley. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

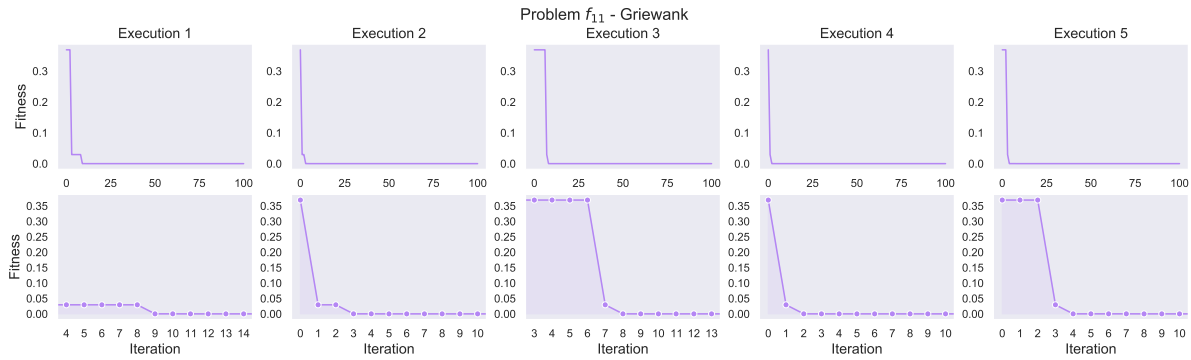

Figure 15. Convergence graph for five executions of problem F11 - Griewank. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

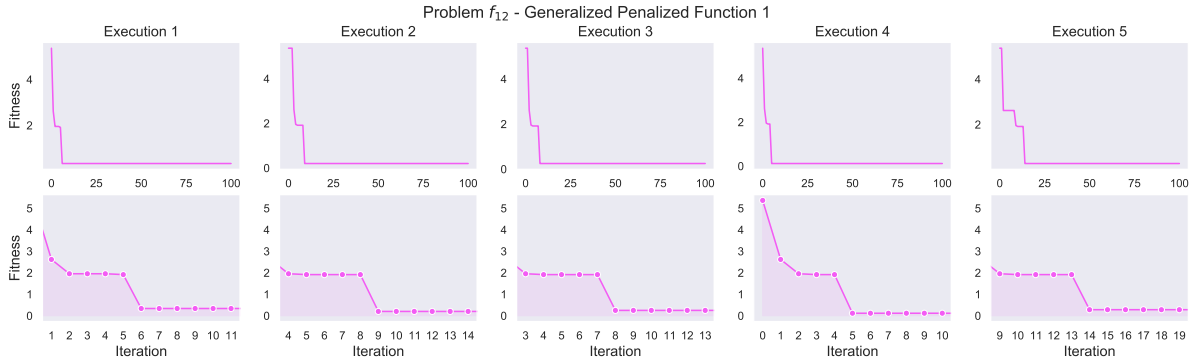

Figure 16. Convergence graph for five executions of problem F12 - Generalized Penalized Function 1. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

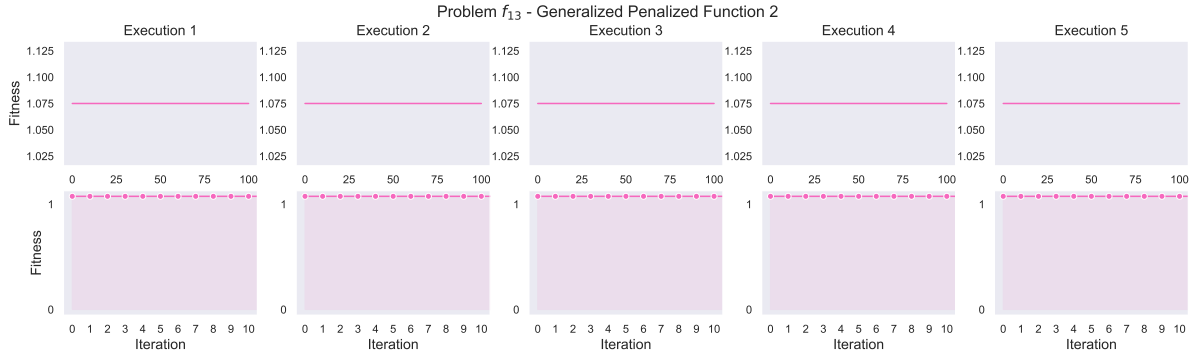

Figure 17. Convergence graph for five executions of problem F13 - Generalized Penalized Function 2. The top graph describes the convergence from iteration 0 to iteration 100. The lower graph describes the moment in which the best aptitude is obtained during the execution of the algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

4. Discussion

According to the results of table 7, the best algorithm that has been found through the evolutionary process was able to find the value of the global optimum of 9 out of 13 optimisation problems. The problems are P1, P2, P3, P4, P6, P7, P9, P10, and P11, where the best fitness found with the value 0.0000, with a standard deviation and average fitness value of 0.0000. Considering the convergence graphs 5, 6, 7, 8, 10, 11, 13, and 14, it can be seen that the optimal global value was reached during the first 10 of the 100 iterations in which the algorithm was executed. These data provide visual evidence that the algorithm managed to find the optimal global value with a limited number of iterations.

For problem P12, the best fitness was obtained for a value of 0.1167, with a standard deviation of 0.0749 and an average value of 0.2373. Problems P5, P8, and P13 did not achieve good fitness values; their standard deviation is 0.0000. When performing a visual inspection on convergence graphs 9, 6, and 7, it can be observed that from a point in their execution, the algorithms cannot obtain a new fitness value, maintaining a single value throughout the iteration. This may be because the instructions that make up the proposed algorithm are not suitable for those specific problems. This point is important because we can obtain important information that this group of problems can be solved with another technique or generate tests with only those optimisation problems.

Considering the results of section 3 of the instructions obtained from the best algorithm found. These instructions can be translated into pseudocode, so that a final display algorithm found can be seen in Figure 18.

```
# Best algorithm found in the exploratory experiment.

def Started(P=0.5):
    # Exploration instruction.
    x = uniform12(0.5, 0.5)      # <O00, I111>
    # Calculate fitness.
    current_fitness = f(x)
    solution = x

def Step():

    if random_number < P:
        # Exploration instructions.
        x = x / A038517()        # <O04, I202>
        x = x / A077761()        # <O04, I200>
        x = x * triangular1()    # <O03, I117>
        x = x - uniform10(0, 1)  # <O02, I109>
        x = x - A077761()        # <O02, I200>
    else:
        # Intensification instructions.
        x = x / sin(x)           # <O04, I01>
        x = x * log2(x)          # <O03, I12>
        x = x / exp1(x)          # <O01, I10>

    # Calculate fitness for minimization problems.
    new_fitness = f(x)
    if new_fitness < current_fitness:
        # Update fitness, and solution.
        current_fitness = new_fitness
        solution = x
        return True
    else:
        return False

def End():
    # Return fitness and solution.
    return x, solution
```

Figure 18. Best algorithm found during the 200 evolutionary iterations.

Finally, this work introduces a reinforced learning framework to generate metaheuristic algorithms (AutoMH). According to the exploratory tests carried out, the AutoMH has managed to find a metaheuristic algorithm that finds the optimal global value of 9 of the 13 continuous optimisation

 doi:10.20944/preprints202101.0048.v1

problems. New operators, or new indivisible intensification and exploration functions, can be integrated as future work, in such a way that the variety of new metaheuristic algorithms that can be found is enriched. Another point is to use a greater number of non-intelligent agents (the experiment was carried out with 30 agents), to increase the options of having a greater number of proposed algorithms that solve the problems entered.

Author Contributions: Conceptualisation, B.A.; methodology, B.A.; software, B.A.; validation, B.A.; formal analysis, B.A.; investigation, B.A.; resources, B.A.; data curation, B.A.; writing–original draft preparation, B.A.; writing–review and editing, B.A.; visualisation, B.A.; supervision, B.A.; project administration, B.A.; funding acquisition, B.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was privately funded by PhD (h.c.) Sonia Álvarez. This research does not receive external funding from governments, universities or companies.

Conflicts of Interest: The authors declare no conflict of interest.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

Appendix A

Appendix A.1

The appendix contains details of the continuous optimisation dataset for P01 to P13 functions.

Definition A1. *P01 - Sphere. The Sphere function is defined by the objective function A1. The function is defined and evaluated in the input domain $x_i \in [-100, 100]$ for all $i = \{1, 2, \ldots, d\}$. The function has one global minimum at $f_{min}(x^*) = 0$ with $x^* = [0, 0, \ldots, 0]$.*

$$f(x) = \sum_{i=1}^{d} x_i^2 \tag{A1}$$

Definition A2. *P02 - Schwefel 2.22. The Schwefel 2.22 function is defined by the objective function A2. The function is defined and evaluated in the input domain $x_i \in [-10, 10]$ for all $i = \{1, 2, \ldots, d\}$. The function has one global minimum at $f_{min}(x^*) = 0$ with $x^* = [0, 0, \ldots, 0]$.*

$$f(x) = \sum_{i=1}^{d} |x_i| + \prod_{i=1}^{d} |x_i| \tag{A2}$$

Definition A3. *P03 - Schwefel 1.2. The Schwefel 1.2 function is defined by the objective function A3. The function is defined and evaluated in the input domain $x_i \in [-100, 100]$ for all $i = \{1, 2, \ldots, d\}$. The function has one global minimum at $f_{min}(x^*) = 0$ with $x^* = [0, 0, \ldots, 0]$.*

$$f(x) = \sum_{i=1}^{d} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{A3}$$

Definition A4. *P04 - Schwefel 2.21. The Schwefel 2.21 function is defined by the objective function A4. The function is defined and evaluated in the input domain $x_i \in [-100, 100]$ for all $i = \{1, 2, \ldots, d\}$. The function has one global minimum at $f_{min}(x^*) = 0$ with $x^* = [0, 0, \ldots, 0]$.*

$$f(x) = \max_{i=1,\ldots,d} |x_i| \tag{A4}$$

Definition A5. *P05 - Rosenbrock's. The Rosenbrock's function is defined by the objective function A5. The function is defined and evaluated in the input domain $x_i \in [-30, 30]$ for all $i = \{1, 2, \ldots, d\}$. The function has one global minimum at $f_{min}(x^*) = 0$ with $x^* = [1, 1, \ldots, 1]$.*

$$f(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{A5}$$

Definition A6. *P06 - Step. The Step function is defined by the objective function A6. The function is defined and evaluated in the input domain $x_i \in [-100, 100]$ for all $i = \{1, 2, \ldots, d\}$. The function has one global minimum at $f_{min}(x^*) = 0$ with $x^* = [x_1, x_2, \ldots, x_d]$, $x_i \in [-0.5, 0.5)$.*

$$f(x) = \sum_{i=1}^{d} (\lfloor x_i + 0.5 \rfloor)^2 \tag{A6}$$

Definition A7. *P07 - Noisy Quartic. The Noisy Quartic function is defined by the objective function A7. The function is defined and evaluated in the input domain $x_i \in [-1.28, 1.28]$ for all $i = \{1, 2, \ldots, d\}$. The function*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

has one global minimum at $f_{min}(x^*) = 0 + \sum_{i=1}^{d} \eta_i$ *with* $x^* = [0, 0, \ldots, 0]$. *Where,* η *is a random number bounded between* $[0, 1)$.

$$f(x) = \sum_{i=1}^{d} (ix_i^4 + \eta_i) \tag{A7}$$

Definition A8. *P08 - Schwefel Function 2.26. The Schwefel function 2.26 is defined by the objective function A8. The function is defined and evaluated in the input domain* $x_i \in [-500, 500]$ *for all* $i = \{1, 2, \ldots, d\}$. *The function has one global minimum at* $f_{min}(x^*) = 0$ *with* $x^* = [4.209687462275036e + 002, 4.209687462275036e + 002, \ldots, 4.209687462275036e + 002]$.

$$f(x) = 4.189828872724338e + 002 \times d - \sum_{i=1}^{d} x_i sin(\sqrt{|x_i|}) \tag{A8}$$

Definition A9. *P09 - Rastrigin Function. The Rastrigin function 2.26 is defined by the objective function A9. The function is defined and evaluated in the input domain* $x_i \in [-5.12, 5.12]$ *for all* $i = \{1, 2, \ldots, d\}$. *The function has one global minimum at* $f_{min}(x^*) = 0$ *with* $x^* = [0, 0, \ldots, 0]$.

$$f(x) = \sum_{i=1}^{d} [x_i^2 - 10cos(2\pi x_i) + 10] \tag{A9}$$

Definition A10. *P10 - Ackley Function. The Ackley function is defined by the objective function A10. The function is defined and evaluated in the input domain* $x_i \in [-32, 32]$ *for all* $i = \{1, 2, \ldots, d\}$. *The function has one global minimum at* $f_{min}(x^*) = 0$ *with* $x^* = [0, 0, \ldots, 0]$.

$$f(\boldsymbol{x}) = -a \ exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(cx_i)\right) + a + exp(1) \tag{A10}$$

$$where: a = 20$$
$$b = 0.2$$

Definition A11. *P11 - Griewank function. The Griewank function is defined by the objective function A11. The function is defined and evaluated in the input domain* $x_i \in [-600, 600]$ *for all* $i = \{1, 2, \ldots, d\}$. *The function has one global minimum at* $f_{min}(x^*) = 0$ *with* $x^* = [0, 0, \ldots, 0]$.

$$f(x) = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{A11}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 January 2021 doi:10.20944/preprints202101.0048.v1

Definition A12. *P12 - Generalized Penalized Function 1. The Generalized Penalized function 1 is defined by the objective function A12. The function is defined and evaluated in the input domain $x_i \in [-50, 50]$ for all $i = \{1, 2, \ldots, d\}$. The function has one global minimum at $f_{min}(x^*) = 0$ with $x^* = [-1, -1, \ldots, -1]$.*

$$f(x) = \frac{\pi}{d} \times \left\{ 10 \sin^2(\pi y_1) + \sum_{i=1}^{d-1} [(y_i - 1)^2 (1 + 10 \sin^2(\pi y_{i+1}))] + (y_d - 1)^2 \right\} + \sum_{i=1}^{d} u(x_i, a, k, m)$$

$$y_i = 1 + \frac{1}{4}(x_i + 1) \qquad u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & if\ x_i > a \\ 0 & if\ -a \leq x_i \leq a \\ k(-x_i - a)^m & if\ x_i < -a \end{cases}$$

$$a = 10$$
$$k = 100$$
$$m = 4$$

(A12)

Definition A13. *P13 - Generalized Penalized Function 2. The Generalized Penalized function 2 is defined by the objective function A13. The function is defined and evaluated in the input domain $x_i \in [-50, 50]$ for all $i = \{1, 2, \ldots, d\}$. The function has one global minimum at $f_{min}(x^*) = 0$ with $x^* = [1, 1, \ldots, 1]$.*

$$f(x) = 0.1 \times \left\{ \sin^2(3\pi x_1) + \sum_{i=1}^{d-1} [(x_i - 1)^2 (1 + \sin^2(3\pi x_{i+1}))] + [(x_n - 1)^2 (1 + \sin^2(2\pi x_n))] \right\} +$$
$$\sum_{i=1}^{d} u(x_i, a, k, m)$$

$$y_i = 1 + \frac{1}{4}(x_i + 1) \qquad u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & if\ x_i > a \\ 0 & if\ -a \leq x_i \leq a \\ k(-x_i - a)^m & if\ x_i < -a \end{cases}$$

$$a = 5$$
$$k = 100$$
$$m = 4$$

(A13)

References

1. Tovey, C.A. Nature-Inspired Heuristics: Overview and Critique. In *Recent Advances in Optimization and Modeling of Contemporary Problems*; INFORMS, 2018; pp. 158–192.
2. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement learning: A survey. *Journal of artificial intelligence research* **1996**, *4*, 237–285.
3. Hein, D.; Hentschel, A.; Runkler, T.A.; Udluft, S. Particle swarm optimization for model predictive control in reinforcement learning environments. In *Critical Developments and Applications of Swarm Intelligence*; IGI Global, 2018; pp. 401–427.
4. Nazari, M.; Oroojlooy, A.; Snyder, L.; Takác, M. Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems* **2018**, *31*, 9839–9849.
5. Sadeg, S.; Hamdad, L.; Remache, A.R.; Karech, M.N.; Benatchba, K.; Habbas, Z. QBSO-FS: A Reinforcement Learning Based Bee Swarm Optimization Metaheuristic for Feature Selection. International Work-Conference on Artificial Neural Networks. Springer, 2019, pp. 785–796.
6. Hayashi, K.; Ohsaki, M. Reinforcement learning for optimum design of a plane frame under static loads. *Engineering with Computers* **2020**, pp. 1–13.
7. Solozabal, R.; Ceberio, J.; Takáč, M. Constrained combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:2006.11984* **2020**.
8. Calvet, L.; de Armas, J.; Masip, D.; Juan, A.A. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics* **2017**, *15*, 261–280.
9. Barrett, T.; Clements, W.; Foerster, J.; Lvovsky, A. Exploratory combinatorial optimization with reinforcement learning. Proceedings of the AAAI Conference on Artificial Intelligence, 2020, Vol. 34, pp. 3243–3250.
10. Kanda, J.; de Carvalho, A.; Hruschka, E.; Soares, C.; Brazdil, P. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features. *Neurocomputing* **2016**, *205*, 393–406.
11. Yu, S.; Aleti, A.; Barca, J.C.; Song, A. Hyper-heuristic online learning for self-assembling swarm robots. International Conference on Computational Science. Springer, 2018, pp. 167–180.
12. de Santiago Júnior, V.A.; Özcan, E.; de Carvalho, V.R. Hyper-Heuristics based on Reinforcement Learning, Balanced Heuristic Selection and Group Decision Acceptance. *Applied Soft Computing* **2020**, *97*, 106760.
13. Wai, H.T.; Yang, Z.; Wang, Z.; Hong, M. Multi-agent reinforcement learning via double averaging primal-dual optimization. *Advances in Neural Information Processing Systems* **2018**, *31*, 9649–9660.
14. Cadenas, J.M.; Garrido, M.C.; Muñoz, E. Using machine learning in a cooperative hybrid parallel strategy of metaheuristics. *Information Sciences* **2009**, *179*, 3255–3267.
15. Real, E.; Liang, C.; So, D.; Le, Q. AutoML-zero: evolving machine learning algorithms from scratch. International Conference on Machine Learning. PMLR, 2020, pp. 8007–8019.
16. Talbi, E.G. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Annals of Operations Research* **2016**, *240*, 171–215.
17. Talbi, E.G. Machine learning into metaheuristics: A survey and taxonomy of data-driven metaheuristics **2020**.
18. Bengio, Y.; Lodi, A.; Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research* **2020**.