Parameter Tuning of Evolutionary Algorithms for Landing Spacecraft

40218080

ABSTRACT

Evolutionary algorithms are popular for their efficacy in solving optimisation problems, however, finding the best combination of parameters settings is not an easy task. This project employs a steady state genetic algorithm and an application of simulated annealing to evolve the weights of a multi-layer perceptron (MLP) controller. Through extensive parameter research and tuning, it was possible to discern which operators and parameters are more effective for evolving the MLP weights. The successful identification and combination of these finally allowed the MLP to safely perform automated moon landing.

ACM Reference format:
40218080. 2019. Parameter Tuning of Evolutionary Algorithms for Landing Spacecraft. In *Proceedings of Coursework, Edinburgh Napier University, April 2019 (SET10107),* 6 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The aim of this coursework was to develop an evolutionary algorithm that would optimise the weights of a multi-neuron percepton for employment in the automated moon landing problem. This report documents the implementation and parameter tuning of two solutions: a steady state genetic algorithm and an application of the simulated annealing technique.

2 APPROACH

This section describes the approach taken to solve the problem of minimising fitness, outlining the types of evolutionary algorithms (EAs) considered (2.1 and 2.2), the evolution operators, the parameter optimisation techniques (2.4) and activation functions (2.3) researched.

2.1 Steady State Genetic Algorithm (GA)

The steady state GA [12] implemented carries out a sequence of operations during each generation. It:

(1) Selects 2 parents from the population.
(2) Applies crossover to obtain two offspring.
(3) Mutates the offspring.
(4) Replaces other members of the population with the offspring.
(5) Injects an immigrant into the population.

The different types of functions used for these operations are described in the following sections.

2.1.1 Initialisation. Randomly generated solutions rarely fall within the optimal fitness range. As a consequence, the following initialisation operators were introduced.

Opposition-based. The first initialisation operator was derived from the opposition-based learning framework proposed by Tizhoosh [19]. He introduces the "anti-chromosome", which is a chromosome whose all gene values are replaced with their additive inverse (i.e. the sign of all alleles is flipped). During the initialisation phase a chromosome P is randomly generated and its opposite anti-chromosome OP is computed. Defining as f a fitness function used to measure the chromosome optimality, if $f(OP) \geq f(P)$, then solution P can be replaced by OP; otherwise the solution P is kept (i.e. only the fittest solution between two opposite chromosomes joins the initial population). This is repeated for N population members. Although this approach is targeted at problems where no a priori information about the solution exists, it was chosen for its proven increase in convergence speed and its ease of implementation.

Fittest N. This initialisation technique consists in generating a large population P (e.g. 1100 chromosomes) and only selecting the fittest N solutions to enter the evolution cycle (e.g. 100 chromosomes). This technique is more computationally and memory expensive than random and opposite-learning initialisations, especially for larger values of P. It was chosen because the probability of generating a fitter solution increases as the population P grows. Since the generation of solutions is random, the chromosomes that enter the evolution cycle are still diverse (i.e. they do not reside on the same local optimum hill).

2.1.2 Selection. Random selection does not alone lead to great solutions. During design the aim was to leverage heuristics to pick parent chromosomes which can lead to fitter offspring, therefore, the following selection operators were introduced.

Tournament selection. Considering a population P, this selection method sees N solutions picked at random take part in a tournament where only the fittest solution is allowed to progress and join the next generation [2]. A "selection pressure" parameter, which corresponds to the tournament size, defines the degree to which the better individuals are favoured.

This technique was chosen because it is simple to code, has a low time complexity $O(N)$, and is robust in the presence of noise in the fitness function [10].

Fitness proportionate. This selection method, also know as Roulette wheel selection, follows the logic: "[t]he better fitted an individual, the larger the probability of its survival and mating" [7, p. 1], that is it has a probability to select an individual proportionally to its fitness. Considering a population of N individuals, each having a fitness ω_i where $i = 1 \ldots N$, the selection probability of an individual $p_i = \frac{\omega_i}{\sum_{i=1}^{N} \omega_i}$.

SET10107, Edinburgh Napier University

DOI: 10.1145/nnnnnnn.nnnnnnn

This selection method has a complexity $O(N)$, and was used due to its simplicity of implementation and interpretation.

Ranked fitness proportionate. The fitness proportionate selection can sometimes lead to non-ideal outcomes. For example, it can stagnate the search if there is a lack of selective pressure (e.g. when the individuals fitnesses differs considerably), which may lead to premature convergence [21].

The rank fitness proportionate (or rank selection) reduces the disproportion between individuals, which means that all the chromosomes have a chance to be selected. It ranks the population according to their fitness, which results in a list where the worst chromosome has fitness 1 and the best has fitness N (i.e. population size) [16]. Although this method can lead to slower convergence, it may produce better results since scaling issues are effectively overcome. It has a complexity $O(N)$.

Best selection. This simple selection algorithm encourages an elitist approach via selecting only the chromosomes with the best fitness. In the case of this project, the top 2 chromosomes in the population are selected as parents before proceeding with the crossover operation. It has a complexity $O(1)$.

2.1.3 Crossover. Crossover is used to recombine the information from two parents into one or two new chromosomes [20]. The repeated application of crossover is useful to decorrelate the population's genes. All the crossover operators covered during the course of the module were implemented; these are listed below:

- Arithmetic crossover. It averages two chromosomes genes and results in only one child.
- 1 and 2-point crossover. They select n cut-points along the chromosome length and swaps the parts from either sides of the split, resulting in two children.
- Uniform crossover considers each gene independently and makes a random choice about whether to swap a gene g_i between two parents for the whole length of the chromosome; it also results in two children.

2.1.4 Mutation. The role of mutation is to produce possibly fitter variants of existing solutions, which may also result in an increased diversity of the population.

The literature suggests that mutation can be more useful than crossover when the population size is small, while the opposite is true when the population size is large [18]. Therefore, since the population size used during testing is quite large, the presence of a mutation operator may not lead to much different results (see Section 3 for more). Nonetheless, apart from the standard mutation operator provided, the following were implemented.

Constrained mutation. This was devised to exploit the mutation operation to only accept a solution that is more fit than the original one. With the aim to be as granular and precise as possible, this operator checks the chromosome fitness every time a gene value is modified. If the gene mutation brings improvements to the fitness then the chromosome is modified, otherwise the mutation is cancelled. This happens iteratively for each gene in the chromosome.

Annealing mutation. This type of operator is inspired by the simulated annealing [9] and the (1+1) evolution [5] strategies. An explanation of its working model can be found in Section 2.2.

2.1.5 Replacement. This operator swaps the chromosomes considered in the current evolution cycle with others in the population [15].

Replace worst. It replaces the least fit chromosome in the entire population, regardless of the fitness of the current solution.

Tournament replacement. Similar to tournament selection, this operator runs a tournament among N chromosomes where the least fit individual is replaced. For this project, the tournament size N was set equal to the one used in tournament selection.

2.1.6 Immigration. The aim of immigration is to introduce new solutions (i.e. immigrants) and replace a small portion of low performing individuals in the current population [11]. **Random immigrants** maintain population diversity and were used to replace the worst fit population members.

2.2 Simulated Annealing

Simulated annealing (SA), also known as shotgun hill climbing, was implemented as a second type of evolutionary algorithm. Similar to the hill climbing family of local search, SA is an iterative algorithm that starts from one chromosome and incrementally mutates it to improves its fitness [9]. This means that when a chromosome is mutated the old one discarded, thus removing redundancy and history of past solutions. SA employs a cooling rate that has control over convergence: slower cooling and more iterations lead to better solutions.

Practically, this algorithm starts with a certain temperature T which decreases over time at a some cooling rate r. The algorithm generates a random chromosome p which is mutated during each evolution cycle through the switch of two of its genes picked at random. If the mutated chromosome has a higher fitness than the original, it is replaced; if it has a lower fitness, it still has a probability $1/(1 + \exp \frac{\Delta f}{T})$ to replace the original chromosome, where Δf is the difference in fitness between two chromosomes.

2.3 Activation Functions

An activation function determines the output of the multi-layer perceptron used in the project, mapping the resulting values into the range $[-1, 1]$. The activation functions tested are listed below:

- The older tanh (provided) and heaviside/unit step functions.
- Part of the rectifier function family, composed of the today most commonly used ReLU and its variants Leaky ReLU [8], ELU [3] and SELU [6].
- The more recent Swish [14] and HardELiSH [1] functions.

Explaining every activation function falls out of the scope of this report. However, it is evidenced that the rectifier family is proven to speed up computation and, in some types of networks, to lead to higher classification accuracies [13], while the Swish and HardELiSH function were recently found to outperform them when applied to GAs for classification [1].

2.4 Parameter Optimisation

Setting the values of the parameters of an evolutionary algorithm is crucial to optimally solve a problem. This can be carried out via parameter tuning and parameter control [4]. Parameter tuning

requires that the parameters are set before the run of the algorithm and remain fixed during the run. These can be set according to empirical evidence or through "automated parameter tuning" [17]. On the other hand, parameter control sees the algorithm starting with initial parameter values that are changed during the run.

Although automated parameter tuning and parameter control were thought to lead to better results, these were finally disregarded due to time limitations. As a consequence, results were gathered through an empirical analysis of quantitative data: the fitness achieved on the training and test dataset was recorded with respect to the chosen parameters. This is further explained in the next section.

3 RESULTS

An empirical research was carried out to find what evolution operators lead to the best fitness, and to understand the effectiveness of the parameter values chosen. All evolution operators were tested, as well as small ranges of parameter values (backed by empirical evidence).

All tests were run for 20,000 generations. Unless otherwise stated, the results are averaged over 10 runs and were achieved using the following parameter settings:

- Initialisation: opposition-based.
- Selection: tournament with size 20.
- Crossover: 2-point.
- Mutation: standard with rate 0.45 and probability 0.90.
- Replacement: tournament with size 20.
- Immigration: random.
- Activation function: ELU.
- Hidden nodes: 10.
- Population size: 60.
- Min and max gene: -1 and 1.

The next sections outline the results achieved.

3.1 Initialisation

The random, opposition-based, and fittest N initialisation operators were tested, achieving the following performance on the train and test sets:

Operator	Training set fitness	Test set fitness
RANDOM	0.0761	0.1768
FITTEST N	0.0812	0.1651
OP-BASED	0.0764	0.1591

All operators consider a population of 60 individuals, however, the fittest N operator initialises 1100 individuals before filtering the ones that will form the final population. As opposed to what initially theorised, the higher number of N members in the population did not bring a major improvement over random initialisation. Instead, the less computationally expensive opposition-based operator achieved the best fitness on the test set among the three.

3.2 Selection

The selection operators random, tournament (size 20), fitness proportionate (ROULETTE), ranked fitness proportionate (RANK), and best were tested. The results were averaged over 20 runs, achieving the following performance on the train and test sets:

Operator	Training set fitness	Test set fitness
RANDOM	0.0629	0.1394
TOURNAMENT	0.0359	0.0925
ROULETTE	0.0552	0.0916
RANK	0.0432	0.1047
BEST	0.0424	0.0985

Fitness proportionate selection achieves the best result on the test dataset, which is followed by tournament selection with size 20. It is observed that the results are not too distant from each other. For example, random selection achieves a fitness only slightly greater than the more complex rank selection (such difference is 0.0347). Morover, it is noted that the BEST selection method achieves a result reasonably close to the top performing methods, with the advantage of working in a $O(1)$ space complexity.
Since the tournament selection effectiveness depends on the number of competing individuals, tests were run to find the optimal parameter and averaged over 20 runs. Not all size values were tested due to computational costs.

Tournament size	Training set fitness	Test set fitness
5	0.0307	0.0751
10	0.0310	0.0679
20	0.0359	0.0925
30	0.0621	0.1407
50	0.0488	0.1005
70	0.0722	0.1527
90	0.0595	0.1491

The table above shows that the a tournament of size 10 achieves the best fitness, outperforming the fitness proportionate results previously outlined. A higher selection pressure was thought to perform better than a lower one, however, the smallest sizes unexpectedly perform better. This could be due to the increased diversity maintained by the lowest tournament sizes.

3.3 Crossover

The crossover operators uniform, 1-point, 2-points, and arithmetic were tested. During an evolution cycle, crossover is always applied. The results are shown below.

Operator	Training set fitness	Test set fitness
UNIFORM	0.0789	0.1723
1-POINT	0.0701	0.1551
2-POINT	0.0516	0.1086
ARITHMETIC	0.0781	0.1767

The 2-point crossover operator is found to perform substantially better than the other operators. It is observed that the gap between 2-point crossover and the other techniques is quite large. It

is thought that 2-point crossover also is effective in maintaining population diversity, which allows for faster exploration the fitness landscape.

3.4 Mutation

The standard, constrained and annealing mutation operators were tested. During an evolution cycle, standard and constrained mutation were applied depending on the parameters mutation rate and mutation change, set respectively to 0.05 and 0.90. The results are shown below.

Operator	Training set fitness	Test set fitness
STANDARD	0.0299	0.0628
CONSTRAINED	0.0252	0.0681
ANNEALING	0.1403	0.2446

The standard mutation revealed most effective. Although annealing works well when used alone (see Section 3.8), it does not achieve a good result when applied to the steady state GA. Next, the mutation rate and the mutation change values were tested with the aim to maximise the method performance.

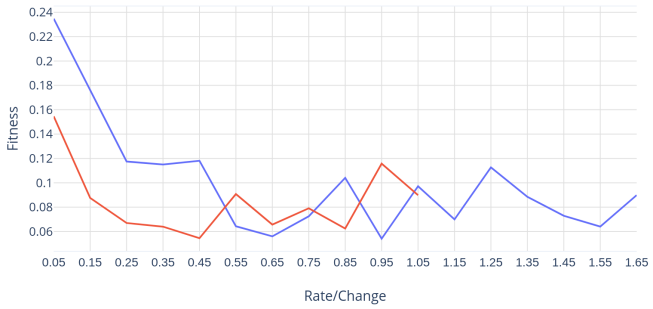

Figure 1: Above is shown the fitness on the training set with respect to the mutation change (blue) and mutation rate (red).

It can be observed from Figure 1 that the best values for mutation change and mutation rate are respectively 0.95 and 0.45. It is noted that results were limited by the computational power available: to find the best combination between the two parameters, these should be cross tested.

3.5 Replacement

The replace worst and tournament (size 20) replacement operators were tested. The results are shown below.

Operator	Training set fitness	Test set fitness
WORST	0.0766	0.1660
TOURNAMENT	0.0648	0.1607

The tournament operator performs slightly better than the worst replacement operator in the tests performed. It is remarked, however, that these results are limited and may be affected by the natural evolution cycle.

3.6 Activation Type

The activation functions tanh, heaviside/unit step, ReLU, Leaky ReLU [8], ELU [3], SELU [6], Swish [14] and HardELiSH [1] were tested. The results are shown below.

Activation function	Training set fitness	Test set fitness
TANH	0.0255	0.0646
STEP	0.0214	0.0925
RELU	0.0232	0.0518
LEAKY-R	0.0180	0.0602
ELU	0.0297	0.1009
SELU	0.0160	0.0296
SWISH	0.0204	0.0437
HARD-ELISH	0.1044	0.2129

The SELU [6] activation function performs best and has a final fitness of 0.0296; it is followed by the Swish [14] function with fitness 0.0437. Although Basirat and Roth [1] show that SELU is outperformed by Swish and HardELiSH when applied to some evolutionary algorithms, this project shows the opposite. Further tests are required to assess the real efficacy of the considered activation functions. It is added that the ReLU function proved to be the fastest due to its lower computational requirements.

3.7 Population Size, Hidden Nodes, Min and Max Genes, and Immigration

Further tests were carried out to find the parameters that would provide the best fitness. For the sake of brevity, these are collected under the same section.

The fitness tendency with respect to the population size is shown in Table 2. It is found that a population of 50 individuals performs best.

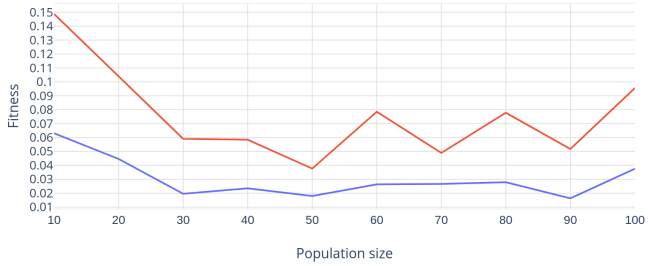

Figure 2: Fitness on the training set (blue) and the test set (red).

The number of hidden nodes was tested and the best value was found to be 13, although only slightly outperforming the 5 and 8 nodes setting. This might be because having more hidden nodes allows more genes per chromosome, which means they can more granularly be improved to map a good solution. Figure 3 shows the fitness trend with respect to the number of nodes.

The network minimum and maximum weight values were tested. The results are show in the table below, however, it was found that the default -1 and 1 performed best.

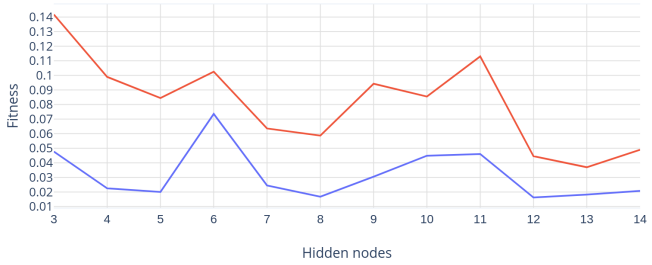

Figure 3: Fitness on the training set (blue) and the test set (red).

Min gene	Max gene	Training set fitness	Test set fitness
-0.7	0.7	0.0370	0.0847
-0.80	0.80	0.0195	0.0625
-0.90	0.90	0.0276	0.0797
-1.0	1.0	0.0264	0.0510
-2.0	2.0	0.0317	0.0662
-3.0	3.0	0.0377	0.0993
-4.0	4.0	0.0320	0.0620
-5.0	5.0	0.0379	0.0916
-6.0	6.0	0.0509	0.1195

Although the immigration method was thought to increase diversity and help avoid local optima, this was found to worsen convergence. On the test set, using the same settings and averaging between 20 runs, random immigration achieved a fitness of 0.08024, while removing the immigration method achieved 0.04862.

3.8 Simulated Annealing

The second algorithm chosen, simulated annealing (SA), proved an effective and simple solution to overcome local optima and improve the fitness of the the single chromosome considered. Figure 4 is shown the fitness with respect to the cooling rate. The best result was achieved with a value of 0.0011.

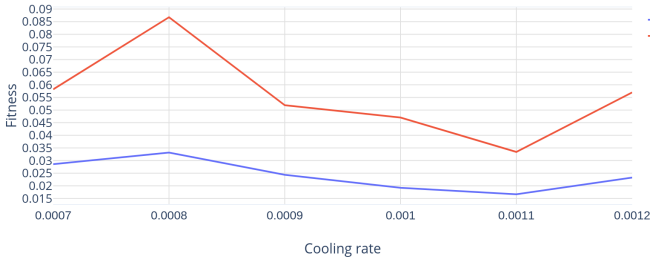

Figure 4: Fitness on the training set (blue) and the test set (red).

Moreover, similarly to the steady state GA, it was found that simulated annealing achieves the best fitness with the SELU activation function [6].

Although SA does not obtain lower results than the steady state GA, it is important to note that SA does not require any parameter tuning except for the cooling rate, and it is easier to understand and to code since it only consists of a few operations.

4 CONCLUSION AND FUTURE WORK

The previous sections allowed to find the best evolutionary operators and parameters among the ones considered. For the final testing phase, these settings were combined and achieved the following results averaged over 10 runs for the two evolutionary algorithms implemented:

Algorithm	Activation function	Training set fitness	Test set fitness
Steady stated GA	SELU	0.0130	0.0263
Simulated annealing	SELU	0.0270	0.0611

The parameters were set as follows: parameter settings:

- Initialisation: opposition-based.
- Selection: tournament with size 10.
- Crossover: 2-point.
- Mutation: standard with rate 0.45 and probability 0.95.
- Replacement: tournament with size 10.
- Immigration: no immigration.
- Activation function: SELU.
- Hidden nodes: 13.
- Population size: 50.
- Min and max gene: -1 and 1.
- Cooling rate: 0.0011.

The steady state GA performs best between the two algorithms implemented, achieving an accuracy on the test set of 0.0263. Simulated annealing performs reasonably well and required little time to be tuned. This makes it an appealing choice in the case time is limited.

These results are positive and mark the successful completion of the project: a multi-layer perceptron was evolved and is now able land spacecraft.

Overall, the parameter search was useful to understand which components most affect the final fitness. For example, the crossover operator, immigration and the activation function are found to be more relevant than the initialisation or replacement methods. Furthermore, the search was useful to discern the best approach with respect to the reviewed literature: for example, although the literature advised in favour of certain activation functions, it was found that SELU was the most effective for the algorithms implemented.

In hindsight, time should have been allocated to the implementation of automated parameter tuning [17] to avoid the tedious and time consuming testing of the many possible parameter combinations.

Future work could look into implementing such automated parameter tuning technique or parameter control [4] to exploit the large parameter space. Additionally, since crossover was found to have a great influence on results, other crossover operators (e.g. higher N-points) could be researched.

REFERENCES

[1] Mina Basirat and Peter M. Roth. 2018. The Quest for the Golden Activation Function. *CoRR* abs/1808.00783 (2018). arXiv:1808.00783 http://arxiv.org/abs/1808.00783

[2] Tobias Blickle. 2000. Tournament selection. *Evolutionary computation* 1 (2000), 181–186.

[3] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[4] Agoston E Eiben, Zbigniew Michalewicz, Marc Schoenauer, and James E Smith. 2007. Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*. Springer, 19–46.

[5] Tobias Glasmachers. 2017. Global Convergence of the (1+1) Evolution Strategy. *CoRR* abs/1706.02887 (2017). arXiv:1706.02887 http://arxiv.org/abs/1706.02887

[6] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. In *Advances in neural information processing systems*. 971–980.

[7] Adam Lipowski and Dorota Lipowska. 2012. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications* 391, 6 (2012), 2193–2196.

[8] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30. 3.

[9] Samir W Mahfoud and David E Goldberg. 1995. Parallel recombinative simulated annealing: a genetic algorithm. *Parallel computing* 21, 1 (1995), 1–28.

[10] Brad L Miller, David E Goldberg, et al. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* 9, 3 (1995), 193–212.

[11] TTH Ng and GSB Leng. 2005. Engineering Applications of Artificial Intelligence. *Department of Mechanical Engineering, National University of Singapore. Singapore* (2005), 439–445.

[12] David Noever and Subbiah Baskaran. 1992. Steady state vs. generational genetic algorithms: A comparison of time complexity and convergence properties. *Preprint series* (1992), 92–07.

[13] Dabal Pedamonti. 2018. Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *arXiv preprint arXiv:1804.02763* (2018).

[14] Prajit Ramachandran, Barret Zoph, and Quoc V Le. 2017. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941* 7 (2017).

[15] Carlos Segura, Carlos A Coello Coello, Eduardo Segredo, and Arturo Hernández Aguirre. 2016. A novel diversity-based replacement strategy for evolutionary algorithms. *IEEE transactions on cybernetics* 46, 12 (2016), 3233–3246.

[16] Smile: Statistical Machine Intelligence and Learning Engine. 2019. GeneticAlgorithm: Selection. (2019). Retrieved April 10th, 2019, from https://haifengl.github.io/smile/api/java/smile/gap/GeneticAlgorithm.Selection.html#RANK.

[17] Selmar K Smit and AE Eiben. 2010. Parameter tuning of evolutionary algorithms: Generalist vs. specialist. In *European Conference on the Applications of Evolutionary Computation*. Springer, 542–551.

[18] William M Spears. 1995. Adapting crossover in evolutionary algorithms.. In *Evolutionary programming*. 367–384.

[19] Hamid R Tizhoosh. 2005. Opposition-based learning: a new scheme for machine intelligence. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Vol. 1. IEEE, 695–701.

[20] Sun-Chong Wang. 2003. *Genetic Algorithm*. Springer US, Boston, MA, 101–116. https://doi.org/10.1007/978-1-4615-0377-4_6

[21] L Darrell Whitley et al. 1989. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best.. In *Icga*, Vol. 89. Fairfax, VA, 116–123.