Optimization of Spacecraft Landing: Evolutionary Algorithm

40489803

ABSTRACT

This paper discusses an evolutionary algorithm controlling the landing of a spacecraft. For the optimization of the algorithm different operators are proposed. These, alongside an array of parameters are experimented with to find the best possible combination, based on the average fitness over ten runs. The experiments show, that finding a balance of exploration and exploitation operators is the key to optimizing the evolutionary algorithm. The most impactful operators were crossover and mutation. For further improvements, other selection and mutation operators could be explored, as well as additional tuning of parameters, such as mutation rate.

ACM Reference format:
40489803. 2018. Optimization of Spacecraft Landing: Evolutionary Algorithm. In *Proceedings of Coursework, Edinburgh Napier University, April 2018 (SET10107)*, 6 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Some problems get too complex for computers to solve within a reasonable time frame. These problems usually have the common feature of showing an exponential increase in time complexity to solve the problem, as the problem size is increased. The solution is using evolutionary algorithms. Evolutionary algorithms can be used to explore the expansive search space of possible solutions and finding the optimal one. These algorithms, as the name suggests are inspired by nature's problem solving skills. As the different species of nature adapt to their environment over time, through trial and error they find the best solution, to the niche problem of surviving based on the given conditions. Evolution in nature and in the computational sense involves a population of the same species. In our case these are potential solutions to the problem at hand. How these will progress towards the best possible solution is determined by their fitness. Natural selection in the real world also involves, the survival of the fittest, which means, the specimen with the best qualities out of the population will be the one to have offspring, which as a result will have a better chance to survive. [5] This principle of competition can be applied to computing as well, in the sense that individuals in the population of possible solutions "survive" for the next generation based on how they compare to their peers. Another important feature of evolution is introducing a variation of solution, such as different species in nature. This is used to widen the exploration of the search space, as it circumvents the issue of arriving in a local optima. This are defined as solutions which have relatively high fitness compared in their population, however they are not the best possible solution (global optima) to the problem. [5]

The objective of the paper is to evolve an artificial neural network, which is used to control spacecraft. The spacecraft must be able to land on a platform, without crashing. The fitness in this population of solutions, is based on a number of characteristics: the distance from the center of the landing pad; the velocity after the simulation, which if the rocket lands correctly should be 0; and the amount of fuel the rocket has, this must also be reduced to 0 to achieve the best possible fitness. This means, the problem is a minimization problem, the lower the fitness, the better the individual.

2 METHODOLOGY

To optimize an evolutionary algorithm there are several operators that can be implemented, and parameters that must be tuned. Operator are representations of real world mechanisms that dictate the course of evolution. These fulfill explorative or exploitative functions, in the process of evolution. Exploration in the search space, refers to finding new possible solutions, in evolutionary terms, this means introducing diversity into the population with different mechanisms. [6] Exploration is done to counter the problem of premature convergence. This occurs, when after a certain number of generations all solutions converge around a small range, usually a local optimum. However a higher ratio of exploration can result in, solutions ranging in the search space too much, and the global optimum is not found, the search will be inefficient.

Exploitation operators rather than focusing on unexplored parts of the search space, they center in on an already found good solution and use it and its surroundings in the search space to find the best possible solution. In evolutionary algorithms this can result in the previously mentioned premature convergence, but exploitation is very important in finding the best solution rather than exploring options which have lesser fitness. [6] [4]

A balance must be struck for the evolutionary algorithm to find the solution desired. The below operators explore a variety options for operators, to try and find that balance. This will be done through experimentation of trial and error. The choice of operators greatly influences the outcome of each experiment, thus the design decisions will be thoroughly explained below. Parameters will also be explored as they dictate the behaviours of operators and the algorithm as a whole. Parameter tuning is explained in the Results section 3.

2.1 Population

Population size plays a very important role in evolutionary algorithm. This is the representation of a biological population, which wants to survive. The population size controls the number of individuals currently in the population, however in evolution, it is not the individuals who evolve, but rather the population. This is done by the selection operation, which will help determine the parents who can reproduce children. These children replace some members

SET10107, Edinburgh Napier University

DOI: 10.1145/nnnnnnn.nnnnnnn

of the population and hopefully increasing the over-all population fitness for the next generation.[5]

The size of the population has an impact on diversity of possible solutions, since the initialization is done so that the individuals are randomized within a range, and their fitness varies by design. Increasing population size however will result in more fitness calculations and may increase, increasing processing time. [3] [12]

2.2 Selection

In natural evolution only a certain number of individuals will be able to reproduce, this in computing is done through the selection operator, which will choose 2 or more parents for reproduction. The selection operator allows for higher fitness individuals to create offspring in the hopes of moving the population towards better results. Meaning it is an exploitation operator, used to reduce the search space. The parent selection operators that are explored, favour the better individuals of the population to varying degrees. Two operators other then the original random selection were implemented for this paper. [20] [5] This is also reflected in selection pressure, which is defined as the pressure on the population to reproduce with emphasis on individuals with higher fitness. [1] Higher selection pressure results in a faster evolution, meaning rapid changes, due to choosing the best individuals sooner, but this obviously can lead to premature convergence. Where as low selection pressure will, in turn slow down progress, as it has a higher probability of using lower fitness parents for crossover.

The first operator tested is *tournament selection*. In this method a k number of individuals are selected at random from the population without taking their fitness into account. Then a tournament is held with the potential parents, where the two best ones are chosen for crossover. This way the selection method compares the relative fitness of potential parents, rather than choosing the best possible parents from the whole population. This reduces the risk of premature convergence, since the random choice of potential parents, results in a wider array of exploration. [2] [5] Increasing the tournament size will increase the probability of choosing individuals with higher fitness. This increase is interpreted in the ratio of the tournament size to the population size, meaning if the population size is increased without adjusting the tournament size, the pressure of selection will decrease, and lower fitness individuals will have an increased chance of reproducing. [1] *Tournament selection* with tournament size of $k = 1$ is the same as using random selection. [2]

The other selection mechanism implemented, was *roulette wheel selection*, otherwise known as fitness proportionate selection. It is modeled after a roulette wheel, where each individual occupies a segment of the theoretical wheel proportionate to their fitness. The wheel is then "spun" and whoever it lands on wins the selection. [20] Thus, the operator is designed to account for the fitness of the potential parents to make the selection. This provides higher selection pressure. The probability of selecting an individual increases with their fitness relative to the fitness of the whole population.

Roulette wheel selection is calculated as:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_i}$$

Roulette wheel selection also has the downsides of reaching premature convergence, since individuals with exceptionally high fitness can dominate the wheels proportions, leading to less diversity in the selection. Oppositely if differences of the fitness values are smaller the method starts reflecting random selection. [20] As opposed to *tournament selection* - where selection pressure can be controlled by the tournament size k - this method, does not present any parameters to do so. [5]

2.3 Activation function

The Neural network controlling the simulation is made up of five input layer nodes, three hidden layer nodes - which will be investigated in a following section - and three three output layers by default. And as any neural network, it employs the very important step of an activation function to tune the output of the nodes between layers.[16]

The base algorithm provided uses *tanh* or *Hyperbolic Tangent* activation function. Calculated as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1$$

[16]

The results of this function lie in the range of $[-1; +1]$.

A similar activation function, the *sigmoid* was implemented in the experiments. This activation function was one of the most common in the early history of neural networks. This function has an input range of $(-\infty; +\infty)$, and outputs in the range of $[0; 1]$. [17]

It is calculated as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

[16]

The linear activation function was the subject of the next test. The derivative of this function is a constant, which has no relation to the input value of x. This means, the neural network will not find complex patterns in the data like other non-linear functions would do. [18]

It can be calculated as:

$$f(x) = ax$$

where a is a constant chosen by the implementer. [16]

The last function is *ReLU* [7] - which stands for Rectified Linear Unit - provides a much simpler alternative to the *tanh* and *sigmoid* functions. It transforms all negative values into 0 and returns all positive values. thus the output values are in the range of $[0; +\infty]$.

Calculated as:

$$Relu(x) = max(0, x)$$

[16]

2.4 Crossover

In the natural setting, the individual in the population will reproduce yielding a new generation. In our algorithm, once the selection for parents has been made a new operator must be introduced to simulate reproduction. This is also referred to as recombination, which is a mechanism of introducing variety into the population. As discussed above in Section 2.2 the selection operation found two parents, who will be used to create an offspring, which will embody the characteristics of the parents. Since the two parents were chosen, based on their desirable traits (fitness), the resulting offspring is the combination of the two, meaning it has a chance of having an even higher fitness than the parents. [5]

The base algorithm provided has no crossover implementation and it just returns the parents. Two crossover methods were used for experiments: the first is the simplest approach using *single-point* or one-point crossover. This is done, by randomly selecting a cut-point in the chromosome length of the parents, and exchanging the different sections, creating the offspring. This way two children can be created: the first child has the first section of its chromosome from the first parent, and the second half from the second parent. The second child is the exact inverse of this. [8]

To introduce more diversity in the children, uniform crossover is also implemented. Here each gene in the chromosome is treated individually, there is an equal chance, the child receives a gene from a parent or not. The first child's chromosome is made up of the copied genes from each parent, the second one's is the exact opposite: where the first is assigned a gene from parent 1, the second receives it from parent 2. [8] [5]

2.5 Replacement

In evolutionary algorithms the population size is kept as a constant, even though the individuals produce offspring, these must replace existing individuals from the population. This mechanism simulates the competitive nature of evolution. [5] The replacement operator in the base algorithm was designed to replace the worst individuals from the population with the children, this is a fitness based replacement operation. This kind of replacement has a high probability of resulting in an a higher population wide fitness score. This operation is very exploitative and can lead to rapid improvements, but may also result in premature convergence.

Another option to implement replacement is random replacement, when the child simply replaces a randomly selected individual, however this has a chance of resulting in the individual with the best fitness being replaced.[5]

The last implementation is done by replacing the parents, with the children, this is done to simulate generations work in natural evolution, thus this method is called generational replacement. But, this has the same possible problem as random replacement, where the replaced individual may be the one with the best fitness in the population. [13]

2.6 Mutation

Before the children are introduced into the population and the next cycle of evolution begins, another exploration operation is used, to increase diversity. This can be used to reduce the chances of reaching a local optima. Mutation is applied to the individual genes of the child, they all have a chance of being changed from the gene determined by crossover. This chance is called the mutation rate, and the possible range in, which the change can be made in the proposed algorithm is determined by *delta change*. Mutation is extremely important in evolution, as without it, evolution would stagnate [11]

The relationship of mutation rate, and the range in which the mutation changes the gene can have great effects on how the algorithm evolves. Thus in the experiments, the two parameters are tuned in combination.

2.7 Hidden nodes

The number of hidden nodes effect the generalization power of the neural net, but if the number of neurons is increased the algorithm becomes more complex, and processing time becomes longer. General rules set by [9] include: the number of neurons are set between the size of the input and the output neurons. A second option is for the number to be 2/3 the size of the input and output layer combined (this was the default setting of 5 in out algorithm). The third option tested is the $2n+1$ rule, where n is the number of input nodes.[19] [14]

3 RESULTS

All experiments are done, by running the algorithm ten times, and all parameters except one are kept constant. This is done, to keep the experiments fair and the results easier to understand.

Firstly the provided partially implemented evolutionary algorithm is tested, thus these results can be used as a baseline for further experiments. This uses operators for selection, crossover, mutation and replacement, which are mostly random or not implemented. These are replaced below in a number of experiments aimed to improve the average fitness of the runs. Each experiment uses the parameters and operators with the best results from the preceding experiments. Thus, only the changed parameter is listed in the charts, not all parameters.

3.1 Population size

Figure 1: Population size

The population size is tested for 4 different values shown in Figure1, but evidently on its own this parameter did not show a big dispersion between results. This is explainable with selection

pressure, the ratio of the population size with the later tested tournament selection will indeed have an effect on the effectiveness on the algorithm since increased tournament size to population ratio increases the selection pressure. But population changed on its own will not result in very different values, when the selection is random as it is in the base model.

3.2 Selection operation

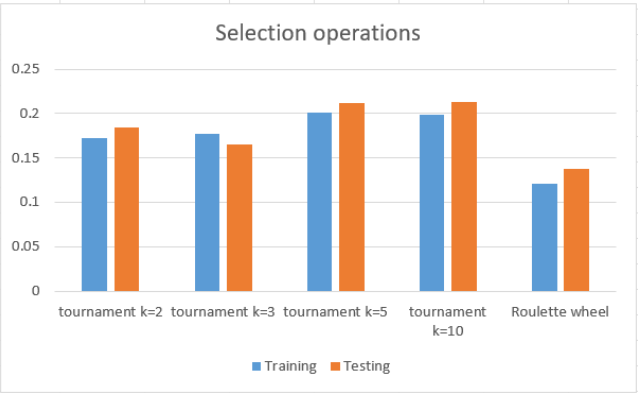

Figure 2: Selection

Firstly tournament selection with increasing number of tournament size was implemented, and as the results show in Figure 2, the ratio of tournament size to population above 0.015 (which is $k = 3$ with $popSize = 200$) will make the selection pressure too high, causing premature convergence.

The second operation for selection was roulette selection, and since this method takes the fitness values of individuals into account it increases enought that the results increase.

3.3 Activation operation

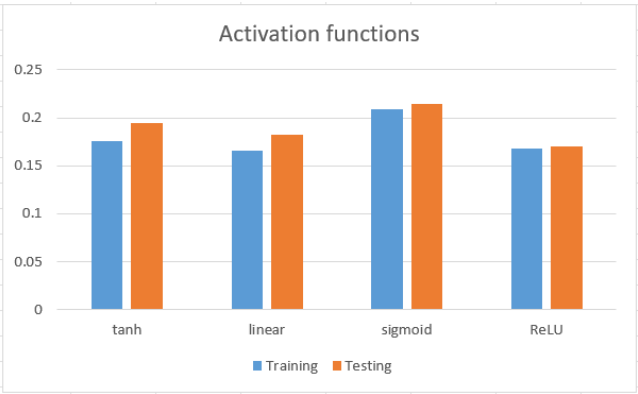

Figure 3: Activation operators

The results of activation operations were fairly close to one another shown in Figure 3 However, linear and ReLU activation functions benefit from much better processing times, because both tanh and sigmoid are more complex as shown in Section 2.3. The results of linear and ReLU were fairly close, however RelU will be chosen, since neural networks use non-linear activation functions for learning.

3.4 Crossover operation

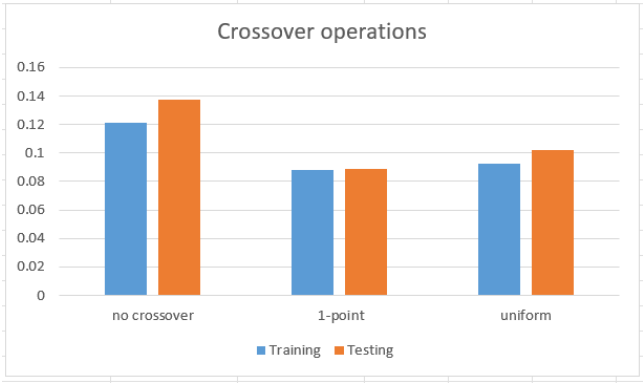

Figure 4: Crossover operators

From the three options 1-point crossover is the clear winner seen in Figure4. It wins out over uniform crossover, since that method introduces too much variety in combination with the applied mutation rate and other variety operators, thus the results spread out too much in the search space. The same is true for the base version, no crossover, results in the algorithm not prioritizing the best solutions.

3.5 Replacement operation

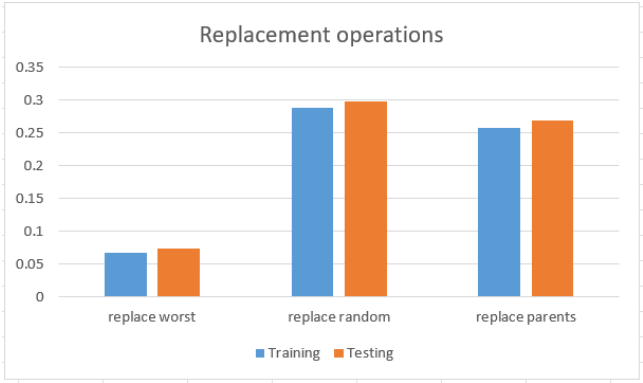

Figure 5: Replacement operators

Replacing the worst individual is the clear winner of this experiment. Figure 5 This can be explained by the problems mentioned in Section 2.5. Random replacement creates too diverse of a population for the results to improve. The problem with replacing the parents has unfortunate consequences as well since the roulette wheel selection operation promotes high selection pressure. Thus if the parents are replaced, there is a high probability the best possible

individuals were replaced, thus the overall fitness will decline as seen in the results.

3.6 Mutation

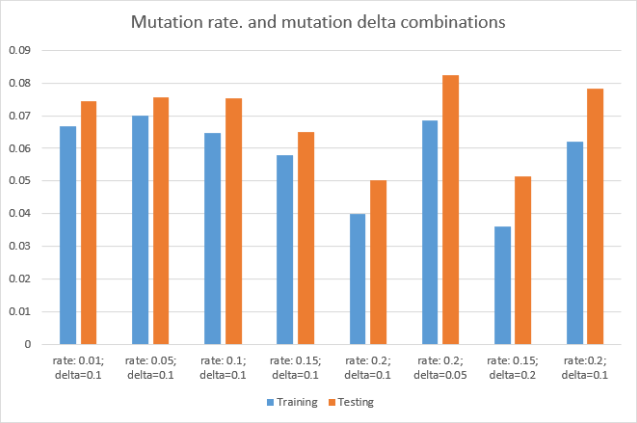

Figure 6: Mutation rate, and amount combinations

The two mutation parameters are tracked simultaneously. We can see in Figure 6, that two combinations reign supreme amongst the results. One combination being a greatly increased mutation rate with the default mutation amount specifically: $mutation\ rate = 0.2$ and $mutationdelta = 0.1$; the other a slightly increased mutation rate with a slightly increased mutation amount, specifically: $mutation\ rate = 1.5$ and $mutationdelta = 0.2$. Both of these greatly increased the diversity in the population, thus the search space was increased and premature convergence averted. These two parameter changes showed the biggest difference compared to the original settings.

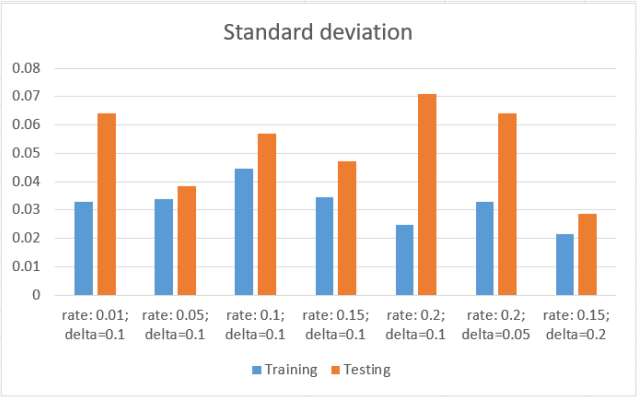

Figure 7: Standard deviation of mutation results

Figure 7 shows the standard deviation of the training and testing results. This shows the biggest drawback of the approach proposed for this paper. The mutation rate, being set this high results in better fitness scores, but this is inconsistent and the tests have a higher distribution.

3.7 Hidden nodes

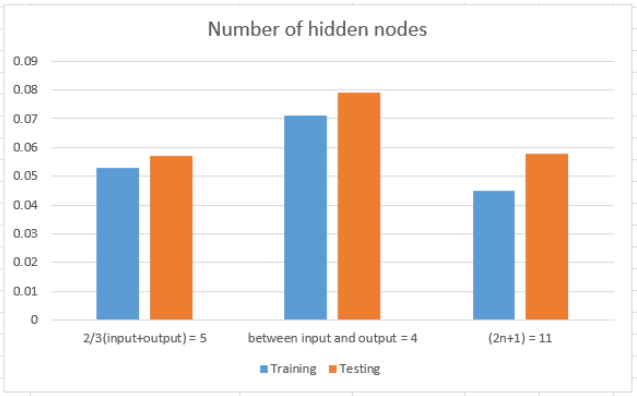

Figure 8: Number of hidden nodes

The number of hidden nodes was tested with the three formulas and the results are fairly close, showing that all three are generally good guidelines, but the $2n + 1$-rule is the best one.

4 CONCLUSION

Through the experiments we can conclude that optimizing an evolutionary algorithm is a fairly complicated task. Although the experiments were not exhaustive, some observations can be made. The parameters with the highest impact on the algorithm were the mutation rate and the mutation amount. Up until those experiments all of the tests would reach a local optima, and the improvements from generations would slow down, or stop. This is expected from an evolutionary algorithm, but the results converged too soon. The increase in mutation rate significantly improved this issue, with the introduction of much needed diversity, the search space was widened, and the results improved. The combination of parameters and operators proposed is by no means the best solution, but it shows that the combination of exploration and exploitation leads to good results, as long as neither is favoured over the other. Whenever one parameter was set to an extreme, the average fitness would decrease. [6]

Another noteworthy improvement was the implementation of the other variety operator in the algorithm, the crossover. The increase in average fitness was remarkable when this operator took effect, especially since the base algorithm had no crossover, and copies of the parents would just be returned to the population, similarly resulting in premature convergence.

However, the proposed algorithm's strengths are not just based in diversity as the above two examples may suggest. The key to optimizing the evolutionary algorithm - as the operator designs show - is balance. Between the use of high selection pressure, when it comes to parent selection, and replacement. But introduction of diversity through mutation and crossover. This combination results in the best results surviving to the next generation, and the best qualities are inherited by the offspring, from the strictly selected parents. But at the same time measures are put in place to prevent premature stagnation of the population.

Another interesting discovery was, that no combination showed particularly consistent results. Throughout the ten runs for each experiment, there would be occasions when the average fitness would plummet or a run would be outstanding compared to the other ones. The inconsistencies in fitness can be explained by the random nature of the chosen operators, and parameters. The initialization of the population, the selection, the recombination, the mutation all introduce random elements into the algorithm, thus the end results are not predictable from the start. The combination of parameters rely heavily on the mutation function, with a fairly high mutation chance of either

The best combination of parameters found in the experiments was the following:

Best model:			
	Parameters:	Training	Testing
	Activation: ReLU	0.084709472	0.101667601
	evaluations: 20 000	0.036012573	0.035912673
	population: 200	0.032050818	0.033370805
	hidden layers: 11	0.063528593	0.063527955
	minimum Genes: -3	0.025178936	0.030989026
	maximumGenes: +3	0.032048278	0.034758263
	mutation rate: 0.15	0.028701395	0.029571595
	mutate change: 0.1	0.031034377	0.031034377
	Operators:	0.01921486	0.01826215
	Roulette Selection	0.016227182	0.016227182
	replace worst		
	1-point crossover		
	Average Fitness	0.036689321	0.038385134

Table 1: Best parameters and operators

5 FUTURE WORK

For improvements and future experiments to the proposed design there are a few operators that could be considered. Roulette wheel selection may be replaced wit *stochastic universal sampling*, which is theoretically similar to the roulette wheel selection. The population is randomly mapped on a line, or a circle, which separated into sections with sizes proportional to the individuals fitness. Then multiple pointers - equal to the number of individuals to be selected - are used to make the selection. This method will not favour the highest fitness individuals as much as roulette wheel, where a couple of outstanding individuals will dominate the selection. Thus, it tries to counteract the issue of premature convergence. [15] [5]

Since the tuning of the mutation operation had such a high impact on the results, it could be explored further. Another mutation operation may also be implemented, or the current method be improved. An optional mutation operator is the proposed by [10]. This method uses a random value from a range to apply to the genes. The range for the random value is determined by Gaussian distribution, this is where its name Gaussian mutation comes from. This Method would give slightly more diverse results than the original implementation, as it is does not use a fixed value for the amount the mutation will change the gene.

REFERENCES

[1] T. Back. 1994. Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. 57–62 vol.1. https://doi.org/10.1109/ICEC.1994.350042

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

[3] Tianshi Chen, Ke Tang, Guoliang Chen, and Xin Yao. 2012. A large population size can be unhelpful in evolutionary algorithms. *Theoretical Computer Science* 436 (2012), 54–70.

[4] Agoston E Eiben and Cornelis A Schippers. 1998. On evolutionary exploration and exploitation. *Fundamenta Informaticae* 35, 1-4 (1998), 35–50.

[5] Agoston Endre Eiben and Selmar K Smit. 2012. Evolutionary algorithm parameters and methods to tune them. In *Autonomous search*. Springer, 15–36.

[6] Anil K Gupta, Ken G Smith, and Christina E Shalley. 2006. The interplay between exploration and exploitation. *Academy of management journal* 49, 4 (2006), 693–706.

[7] Richard LT Hahnloser. 1998. On the piecewise analysis of networks of linear threshold neurons. *Neural Networks* 11, 4 (1998), 691–697.

[8] Oğuzhan Hasançebi and Fuat Erbatur. 2000. Evaluation of crossover techniques in genetic algorithm based optimum structural design. *Computers & Structures* 78, 1-3 (2000), 435–448.

[9] Jeff Heaton. 2008. *Introduction to neural networks with Java*. Heaton Research, Inc.

[10] Robert Hinterding. 1995. Gaussian mutation and self-adaption for numeric genetic algorithms. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, Vol. 1. IEEE, 384.

[11] Robert Hinterding, Harry Gielewski, and Thomas C Peachey. 1995. The Nature of Mutation in Genetic Algorithms.. In *ICGA*. Citeseer, 65–72.

[12] Thomas Jansen, Kenneth A De Jong, and Ingo Wegener. 2005. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation* 13, 4 (2005), 413–440.

[13] Zbigniew Michalewicz and Marc Schoenauer. 2002. Evolutionary Algorithms. *Encyclopedia of Information Systems* 2 (2002), 259–267.

[14] Gagan Mirchandani and Wei Cao. 1989. On hidden nodes for neural nets. *IEEE Transactions on Circuits and systems* 36, 5 (1989), 661–664.

[15] Tania Pencheva, Krassimir Atanassov, and Anthony Shannon. 2009. Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets. In *Proceedings of the tenth international workshop on generalized nets, Sofia*. 1–7.

[16] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. 2020. A Review of Activation Function for Artificial Neural Network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. 281–286. https://doi.org/10.1109/SAMI48414.2020.9108717

[17] Matías Roodschild, Jorge Gotay Sardiñas, and Adrián Will. 2020. A new approach for the vanishing gradient problem on sigmoid activation. *Progress in Artificial Intelligence* 9, 4 (2020), 351–360.

[18] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. 2017. Activation functions in neural networks. *Towards Data Sci* 6, 12 (2017), 310–316.

[19] Dimitris Stathakis. 2009. How many hidden layers and nodes? *International Journal of Remote Sensing* 30, 8 (2009), 2133–2147.

[20] Saneh Lata Yadav and Asha Sohal. 2017. Comparative study of different selection techniques in genetic algorithm. *International Journal of Engineering, Science and Mathematics* 6, 3 (2017), 174–180.