SOLVING A NAS PROBLEM WITH EVOLUTIONARY ALGORITHMS

Ruilin Ma
S3500926
r.ma.2@umail.leidenuniv.nl

December 7, 2022

1 Introduction

NAS, or Neural Architecture Search, is a method for automating the design of artificial neural networks. It has been used to build networks that are as good as or better than manually constructed topologies. The NAS problem in this assignment can be reduced to a 26-dimensional discrete optimization problem, with the first 21 variables $x_i \in \{0, 1\}, i \in [1...21]$ representing the upper-triangular binary matrix of NAS and the last 5 variables $x_i \in \{0, 1, 2\}, i \in [22...26]$ representing the operations of the 5 intermediate NAS vertices, as illustrated in Figure1. The NAS-Bench-101 is utilized as a benchmark in this assignment as it offers a dataset that maps neural networks to their training and evaluation metrics. All resources come from the package provided in this course.

$$\text{Adjacency matrix} = \begin{bmatrix} 0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0 & 0 & x_7 & x_8 & x_9 & x_{10} & x_{11} \\ 0 & 0 & 0 & x_{12} & x_{13} & x_{14} & x_{15} \\ 0 & 0 & 0 & 0 & x_{16} & x_{17} & x_{18} \\ 0 & 0 & 0 & 0 & 0 & x_{19} & x_{20} \\ 0 & 0 & 0 & 0 & 0 & 0 & x_{21} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\text{Operations at the 7vertices} = [\text{INPUT}, x_{22}, x_{23}, x_{24}, x_{25}, x_{26}, \text{OUTPUT}],$$

Figure 1: The representation of the NAS problem investigated [1].

To perform optimizations, evolutionary algorithms employ mechanisms inspired by biological evolution. The family includes several types, two of which are used in this assignment: Genetic Algorithm (GA) and Evolution Strategy (ES). The Genetic Algorithm employs natural-inspired operations such as mutation, crossover, and selection. The Evolution Strategy employs methods that are similar but have different approaches: mutation, recombination, and selection. Both techniques mimic natural evolution, with GA generally applying to bitstring problems and ES capable of dealing with real values.
To solve the previously mentioned NAS problem, a Genetic Algorithm and an Evolution strategy are developed in this assignment. Both algorithms are evaluated using the NAS-Bench-101 and the results are illustrated and analyzed with the IOHanalyzer.

2 Algorithms

2.1 Genetic Algorithm

The framework of the GA approach to solving the NAS problem is described in the following Algorithm 1. The listed parameters are in accordance with the submitted algorithm. The comparison between the different algorithm configurations tested is explained in the section3 Experimental Results below.
Initialization and validity check. The algorithm is initialized by generating solutions at random. However, because the problem is associated with a functional neural architecture, not all of the generated random solutions are valid for representing the architecture. To conserve the evaluation budget, a validity check is performed to ensure that all

generated solutions used to populate the population are valid. Although later modifications to solutions may turn a valid configuration into an invalid one, we still check the validity at the start to eliminate those that are far from a valid architecture.

Population size. Following initialization, the algorithm select μ solutions (parents) from the population size of p to perform modifications. We set $\mu = p$ so that all solutions within the population are used to generate offspring. The offspring size is set to be identical to the parent size. For the submitted algorithm, the population size p and the parent/offspring size μ are all set to 10.

Crossover and Mutation. The algorithm then performs crossover and mutation on the parents. Crossover is performed first on all parent solutions. We use $uniform\ crossover$ as the operator and randomly select two parents to produce two offspring with a possibility of p_c. The offspring then go through the mutation phase with a possibility of p_m. Following the mutation, an additional validity check is performed to ensure that the offspring we produced is suitable for further evaluation. If the mutation happens to create an invalid solution, the mutation will be repeated until the mutated offspring pass the validity check. The p_c is set to 0.6 and the p_m is set to 0.25.

Evaluation and Selection. The algorithm then evaluates the solutions after they have been modified. It evaluates the generated offspring and records the corresponding fitness. After evaluating the fitness of each solution, the algorithm performs a selection to determine which solutions will be used to form the next population. We looked at the $Roulette\ wheel\ selection$, the $Tournament\ selection$, and the $Rank\ selection$ as selection operators, and eventually adopted the $Tournament\ selection$ with a tournament size $= 5$.

The Genetic Algorithm seeks solutions until it reaches the 5000 budget limit. To conserve the budget, we skip the first selection phase which comes right after the random initialization. This results in a slightly lower startup performance, but as evolution continues, the side effect is quickly eliminated.

Algorithm 1: A feasible Genetic Algorithm approach to the NAS problem
Parameter settings listed in accordance with the configuration used in the submitted algorithm

Input : Population size p; Parent size and offspring size μ; $p = \mu = 10$
Crossover probability $p_c = 0.6$
Mutation probability $p_m = 0.25$
tournament size = 5

Termination : The algorithm terminates when budget B exceeds the limit of 5000

```
Initialize the population with randomly generated solutions;
Check the validity of the initialized solutions within the population;
Discard the invalid solutions and regenerate until the population size μ is fulfilled;
for i = 1 to B do
    Perform uniform Crossover of the randomly selected two parents at a possibility of p_c = 0.6;
    Mutation of the offspring bitstring at a possibility of p_m = 0.25 and perform validity check;
    Evaluate the generated offspring and record their fitness;
    Perform a tournament selection on solutions to decide the individuals of the next population;
end
```

2.2 Evolution Strategy

The framework of the ES approach to solving the NAS problem is described in the following Algorithm 2. The ES approach resembles the GA approach in various ways and the overall implementation procedure are mostly similar. The main difference includes the following.

1. **Decode and encode.** The ES approach performs crossover and mutation on a real value string but evaluates fitness on a bit string, requiring decode (transform the real value string to bitstring) and encode (transform the bitstring to real value string) functions to transform the solutions accordingly. Each solution is represented by a string of real values. We divide the first 21 bits of the solution string into seven segments, each with three bits of 0 or 1. This can be represented by an integer between 0 and 7. The last five bits of 0,1 and 2 are regarded as a whole, which can be represented by an integer ranging from 0 to 242. As a result, we obtained a real value string made up of eight elements that can be used for mutation and recombination. For example, a solution of $[1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 2, 0, 0, 0, 0]$ can be represented as $[4, 6, 4, 4, 4, 5, 2, 162]$. All actions are carried out with real value strings, which are temporarily transformed into the 26-bit bitstrings for validity check and fitness evaluation.
2. **Mutation and recombination.** The ES approach places a greater emphasis on mutation and employs a different method of carrying it out. Unlike GA, which uses a coin-flipping method, ES uses normally distributed

variations that are applied to all individuals. Moreover, unlike in GA, where the only parameter affecting the process is the fixed mutation rate, the step size will change during the mutation process. The mutation and recombination happen definitely for every generation, which is to say that in the ES approach, the mutation rate and recombination rate are all set to 1. We adopt the $one-\sigma$ mutation and $intermidiate\ recombination$ as the operator for the ES approach. It is also worth noting that the mutation may introduce decimal numbers into the real-value string, which was originally composed of integers. This will cause issues with the decoding process. To address this issue, a rounding function is introduced that converts all decimals to integers before they are decoded into the bitstring.

3. **Selection.** The selection is deterministic, with two criteria: the (μ, λ) selection (comma selection) and the $(\mu + \lambda)$ selection (plus selection). Both indicate that the algorithm generates λ offspring from μ parents, and the comma selection only evaluates the fitness of the offspring and chooses the next population among them, implying that the fitness of the next population may deteriorate. The plus selection, on the other hand, evaluates and selects among both offspring and parents to form the next population, which accepts only improvements. The comma selection is used in this implementation.

Algorithm 2: A feasible Evolution Strategy approach to the NAS problem
Parameter settings listed in accordance with the configuration used in the submitted algorithm

Input : Population size p; Parent size μ; $p = \mu = 4$
Offspring size $\lambda = 20$
Learning rate $\tau = 0.196$
Termination : The algorithm terminates when budget B exceeds the limit of 5000

```
Initialize the population with randomly generated real-value solutions and their corresponding σs;
Decode and check the validity of the initialized solutions within the population;
Discard the invalid solutions and regenerate until the population size μ is fulfilled;
for i = 1 to B do
    Perform an intermediate recombination to the parents to generate offspring;
    Perform a 1 − σ Mutation of the offspring and decode the solution for validity check;
    Evaluate the generated offspring and record their fitness;
    Perform a comma selection to determine the next population;
end
```

We test various algorithm configurations to find the best set of parameters for implementation. The above configuration is based on the best configuration found. Section 3 below illustrates and explains some of the comparisons of different settings.

3 Experimental Results

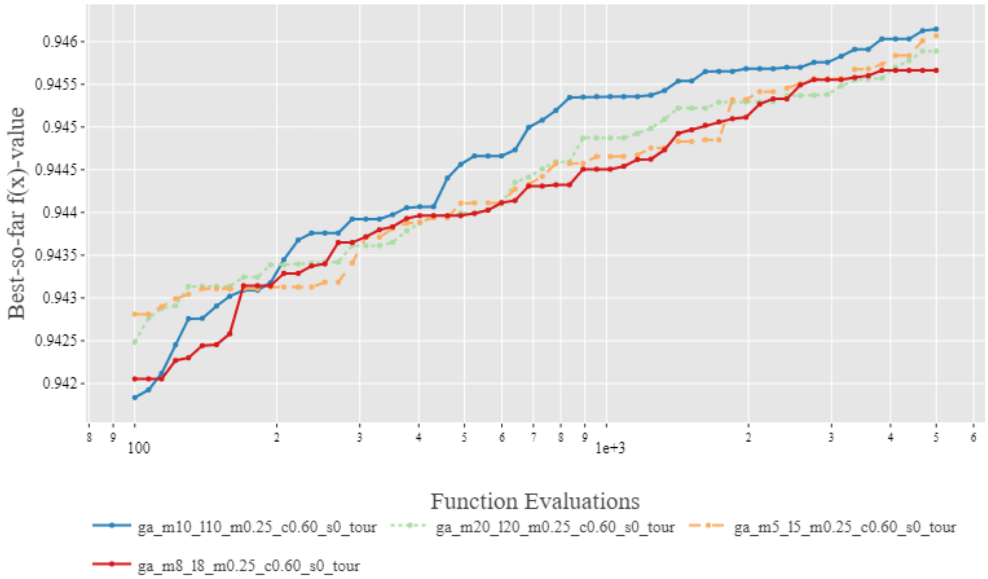
The NAS-bench-101 tool was used to obtain all results, which were then visualized using the IOHAnalyzer. In all of the listed results, **the random seed is set to seed 0 to fight randomness.** Plots for the average best-found fitness are segmented from budget 100 to 5000 to make it easier to look at.

3.1 Genetic Algorithm

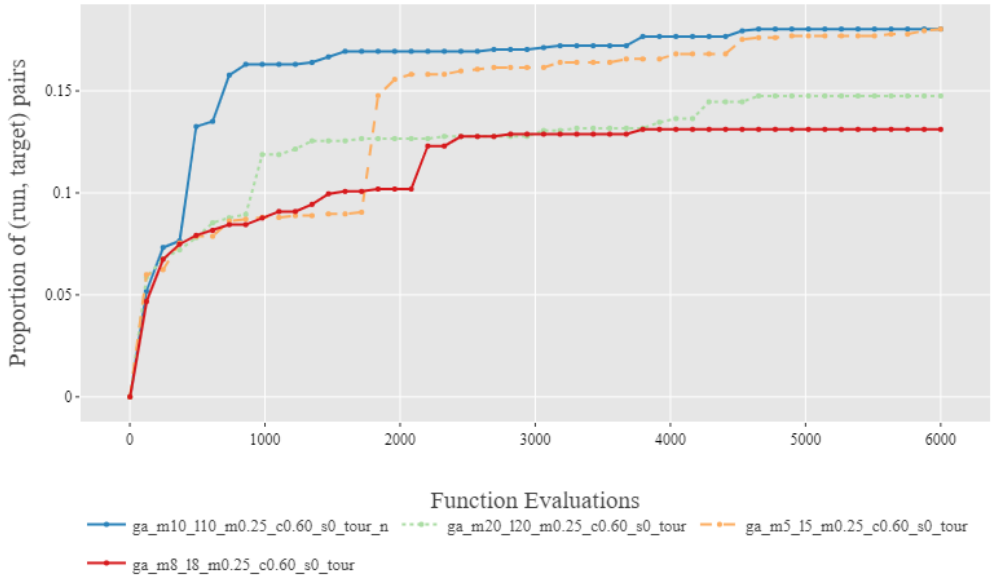
In the process of developing our genetic algorithm, we experimented with various configurations to pinpoint the best parameters set. We tested algorithms with different population sizes (μ), different mutation rates (p_m), various crossover rates (p_c) and distinct selection methods.

3.1.1 Population size

The population size (μ) of the Genetic Algorithm greatly affects the performance. The population size or parent size is set to be equal to the offspring size. Four different configurations are tested, with a population size of 5,8,10,20 respectively ($ga\_m5\_l5\_..., ga\_m8\_l8\_..., ga\_m10\_l10\_..., ga\_m20\_l20\_...$). Other parameters are identically configured.
As shown in Figure 2, the population size of 10 outperforms the other configurations in terms of the AUC of the ECDF curve and the expected target value. Changing the size to too large(20) or too small(5) will likely reduce performance. It is also worth noting that in the developed algorithm, population sizes that can be divided by 5 perform better than those that cannot. The setting with a population size of 10 records a clearly better output as shown in Figure 2a and 2b.



(a) The average best-found fitness value curve of algorithms with varying population sizes



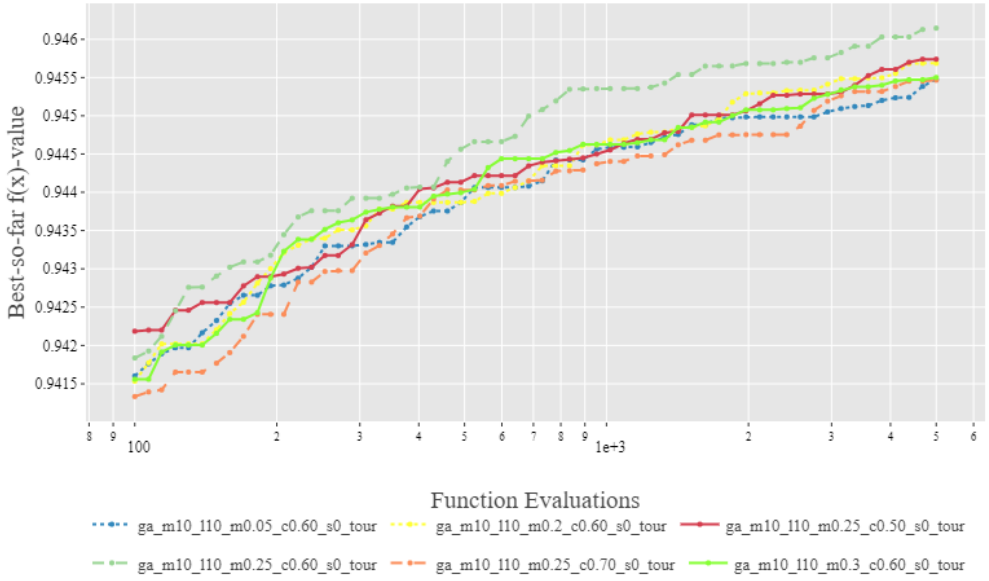
(b) The ECDF curve of algorithms with varying population size

Figure 2: The influence of the population size on GA approach

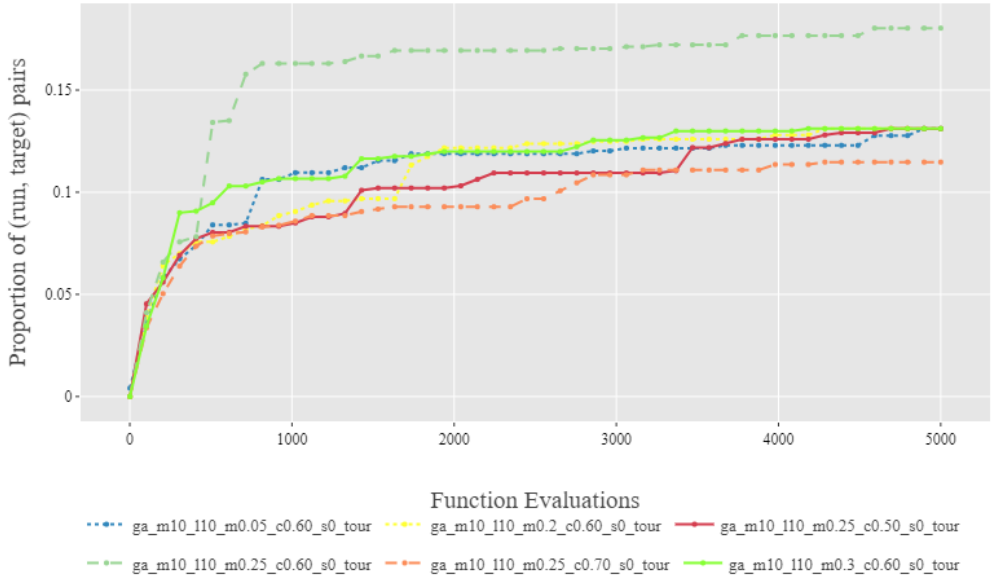
3.1.2 Mutation rate and crossover rate

The mutation rate and crossover rate control how likely will the parents reproduce their offspring in these manners. If the mutation rate is too small, for example, smaller than $\frac{1}{n}$, the mutation phase will not make any sense. And if that becomes too large, the process will be nothing more than a random search. So controlling these parameters is of great significance.
The tested configurations involve the following settings: $p_c = 0.6, p_m = 0.2; p_c = 0.5, p_m = 0.25; p_c = 0.6, p_m = 0.25; p_c = 0.7, p_m = 0.25; p_c = 0.6, p_m = 0.3$. As can be noticed in Figure 3a and 3b, the configuration with a mutation rate of 0.25 and a crossover rate of 0.6 outperforms the rest configurations no matter in what evaluation approach. It also achieves the highest area under the curve (AUC) value in all tested algorithms.

The sweet spot for mutation rate in the developed Genetic Algorithm is between 0.2 and 0.3, and the sweet spot for crossover rate is between 0.5 and 0.7, as shown in Figure 3. This is reasonable given the small size of the dataset, which may be influenced more by exploitation (mutation) than exploration (crossover).



(a) The average best-found fitness value curve of algorithms with varying mutation and crossover rates



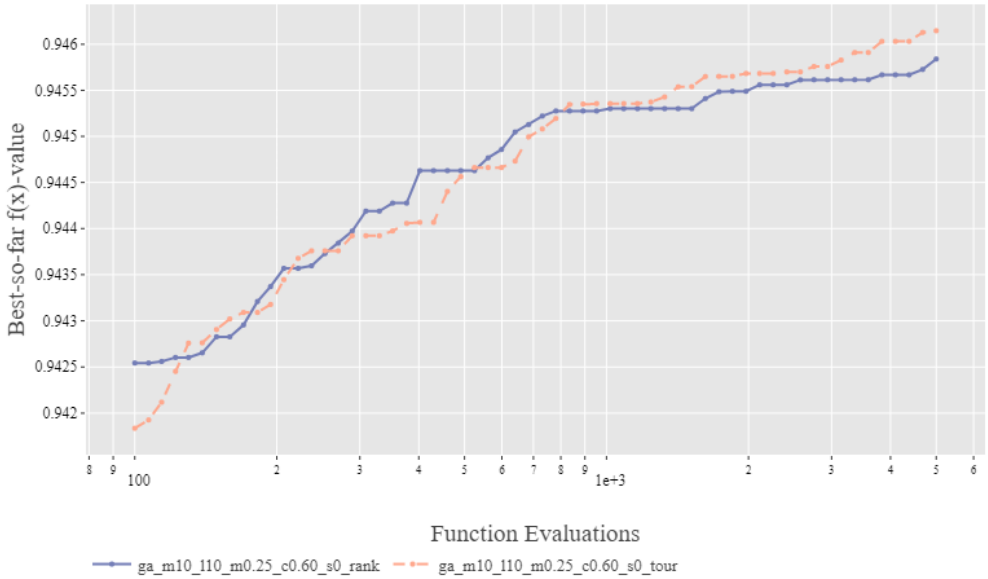
(b) The ECDF curve of algorithms with varying mutation and crossover rates

Figure 3: The influence of the mutation and crossover rate on GA approach

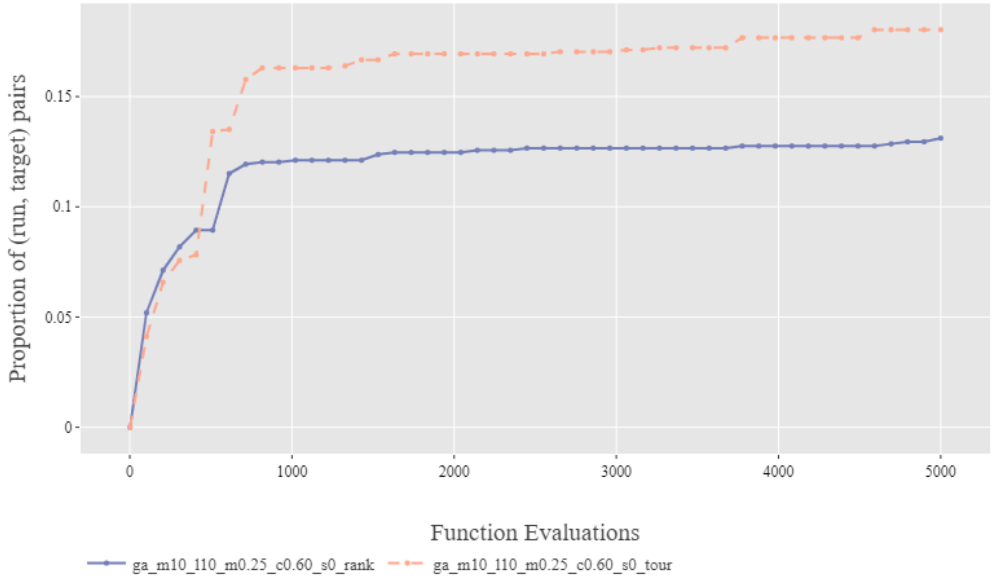
3.1.3 Selection methods

We experimented with two different selection methods during fine-tuning, which are the tournament selection and the rank selection. the reason for not using the Roulette wheel selection is that it may not perform well when the fitness values are close, which is exactly the case in this assignment. The following Figure 4 describes the comparison between these approaches.

It is clear from Figures 4a and 4b that the tournament selection outperforms the rank selection, with a large margin between the two methods on both scales. This is possible because tournament selection introduces more randomness into the selection process, which broadens the area being explored. As a result, the tournament selection method will be used in the submitted version of the algorithm. It is also worth noting that the algorithm's tournament size is set to 5 because it is the best-observed configuration in the experiments; however, the results are missing due to a computer error, and thus they are not listed in this figure.



(a) The average best-found fitness value curve of algorithms with varying selection methods



(b) The ECDF curve of algorithms with varying selection methods

Figure 4: The influence of the selection method on GA approach

3.2 Evolution Strategy

In the process of developing our evolution strategy, we experimented with various configurations to pinpoint the best parameters set. We tested algorithms with different population sizes (μ), different offspring sizes(λ), various learning rates (τ) and distinct selection methods.

3.2.1 Population and offspring size

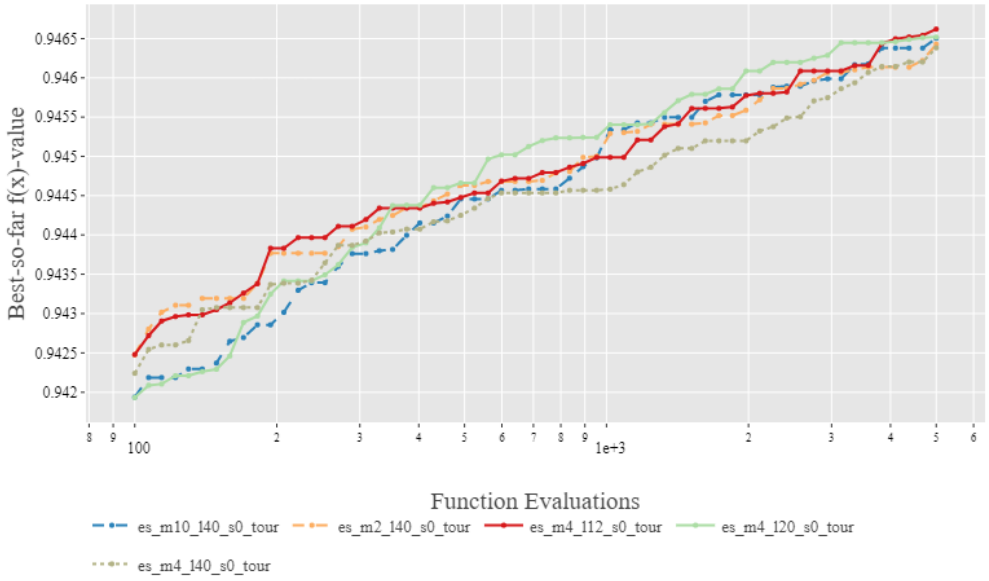
The population size (μ) and offspring size (λ) play a vital role in the performance of an Evolution strategy, as they determine how many parents are to be modified and produce how many offspring, which are the main players of evolutionary algorithms. The following Figure 5 shows the influence of the population and offspring size on the Evolution Strategy.
In terms of population size, the offspring size of three of the listed configurations is set to 40, while the population sizes are set to 4,5, and 10 respectively ($es\_m4\_l40\_s0\_tour, es\_m5\_l40\_s0\_tour, es\_m10\_l40\_s0\_tour$). Figure 5a shows that a population size of 5 outperforms a smaller or larger population size in terms of best-so-far performance given a fixed budget, which is also true for the ECDF curve and AUC values, as shown in Figure 5b. Although we initially believed that a larger population would contribute to better performance, the results we obtained show that a smaller population size of approximately 5 individuals has better average fitness. We deduce that it could be due to the limited number of evaluation results in the given dataset.
In terms of offspring size, the population size of three of the listed configurations is all set to 4, based on the idea that smaller population sizes may yield better results, and the offspring sizes are set to 12, 20, and 40, respectively. ($es\_m4\_l12\_s0\_tour, es\_m4\_l20\_s0\_tour, es\_m4\_l40\_s0\_tour$). All other parameters are identically set. It can be observed that an offspring size of 20 is better than a smaller or larger population size in terms of best-so-far performance given a fixed budget, and the ECDF curve together with AUC values, which can be observed in Figure 5a and Figure5b. It stands to reason that having an intermediate offspring size results in better performance, because having too few children limits the exploration of the search space in one generation, while having too many children exhausts the budget too quickly, leaving little time for the population to evolve.

3.2.2 Learning rate

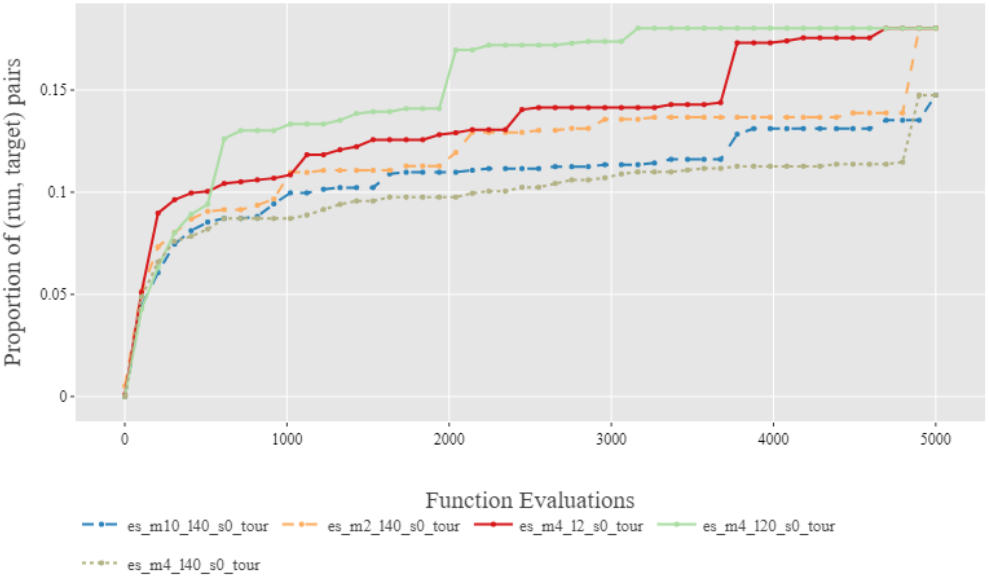
The Evolution Strategy's learning rate also has a significant impact on the algorithm's performance, because every step of mutation, which is heavily emphasized by the ES approach, is heavily dependent on the learning rate, as it is the parameter that controls how far a solution can mutate. A learning rate that is too high may cause overshooting, while one that is too low may cause undershooting. As a result, selecting the appropriate learning rate may result in better results.
As is shown in Figure 6 below, distinct learning rates lead to distinct algorithm performances. Three algorithms are evaluated ($es\_m4\_l20\_s0\_tour_{20}, es\_m4\_l20\_s0\_tour, es\_m4\_l20\_s0\_tour_{30}$), respectively with a learning rate of 0.223, 0.196 and 0.183. Other parameters are identically configured with the best-observed set obtained from the previous experiments. It could be observed that a larger learning rate of 0.223 achieves a better average best-found fitness value in a given budget evaluation, as shown in Figure 6a, while a medium learning rate of 0.196 obtains a better performance in terms of the AUC value as shown in Figure 6b.

3.3 Performance of submitted algorithms

After comparing the experimented configurations discussed above, we decided to implement the algorithms with parameters set to the values listed in Algorithms1 and 2. Figure 7 compares the overall performance of the two approaches to a random search baseline (RS). It can be seen that the Genetic Algorithm has the highest AUC value of the three, followed by the Evolution strategy and random search. The Evolution Strategy, on the other hand, achieves the highest expected target value, followed by the Genetic Algorithm and random search. The AUC values of these algorithms are listed in Table 1.
The area under the ECDF curve of the Genetic Algorithm is the largest among all approaches, as also proven by the AUC value. This suggests that the Genetic Algorithm is more likely to find a better solution within a smaller budget, implying a greater ability to evolve to good solutions quickly. The Evolution Strategy's average best-found fitness value is the highest, implying that ES can find a solution with a higher fitness value within the given 5000 budget. In other words, the GA can evolve quickly, while the ES can evolve into strong solutions.



(a) The average best-found fitness value curve of algorithms with varying population size and offspring size

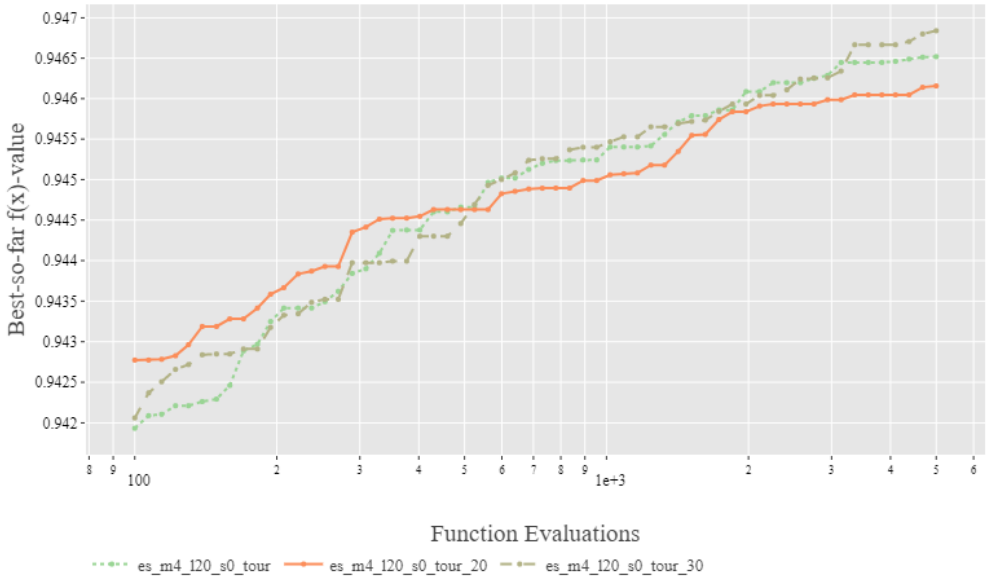


(b) The ECDF curve of algorithms with varying population size and offspring size

Figure 5: The influence of the population size on ES approach

Table 1: Performance comparison of tested algorithms

Algorithm	**AUC(linear axis)**	**Average best-found fitness**
GA	0.15922	0.946
ES	0.15285	0.947
RS	0.12712	0.946



(a) The average best-found fitness value curve of algorithms with different learning rates

(b) The ECDF curve of algorithms with varying learning rates

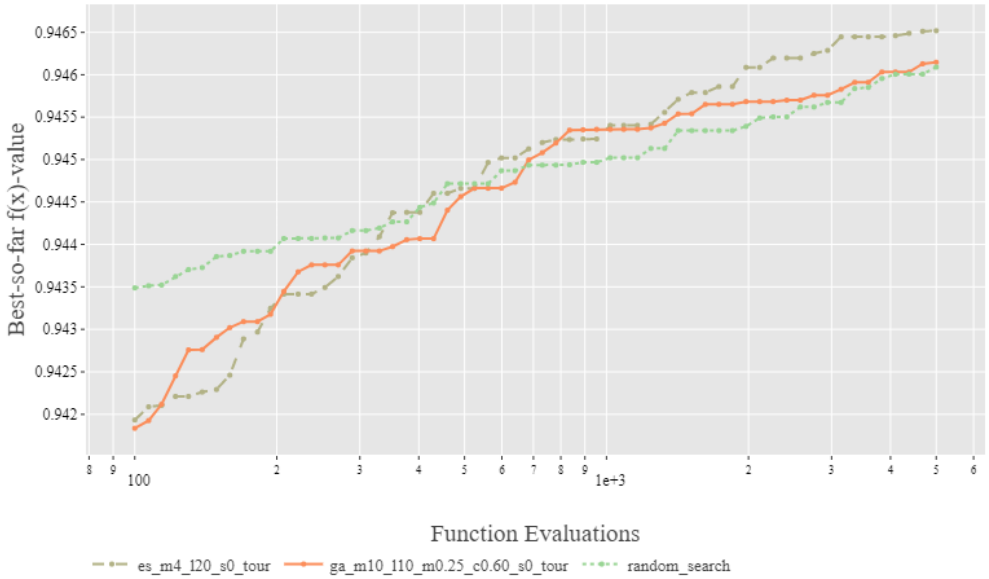
Figure 6: The influence of the learning rate on ES approach

4 Discussion and Conclusion

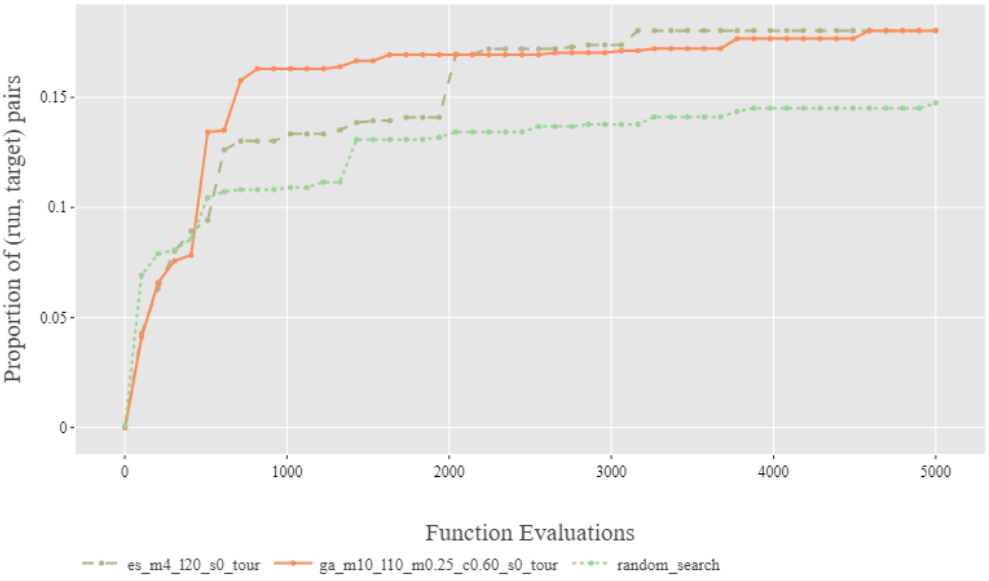
We created a Genetic Algorithm and an Evolution Strategy to solve a NAS problem in this assignment. Both evolutionary algorithms outperform the random search, demonstrating evolution's power. We'd like to highlight a few points.

1) We suggest using population size $\mu = 10, p_c = 0.6, p_m = 0.25$ for the Genetic Algorithm to solve the problem. The tournament selection method is also recommended.

2) We suggest using population size $\mu = 4$, offspring size $\lambda = 20$ and a learning rate of 0.196 for the Evolution Strategy to solve the problem.



(a) The average best-found fitness value curve of the submitted Genetic Algorithm and Evolution Strategy



(b) The ECDF curve of the submitted Genetic Algorithm and Evolution Strategy

Figure 7: Comparison between submitted GA and ES approach and a random search baseline

3) We observe that the genetic algorithm benefits from a small crossover rate and a medium mutation rate in solving the NAS problem. A population size that can be divided by 5 may also improve the general performance. The evolution strategy on the other hand benefits from a small population size and an offspring size.

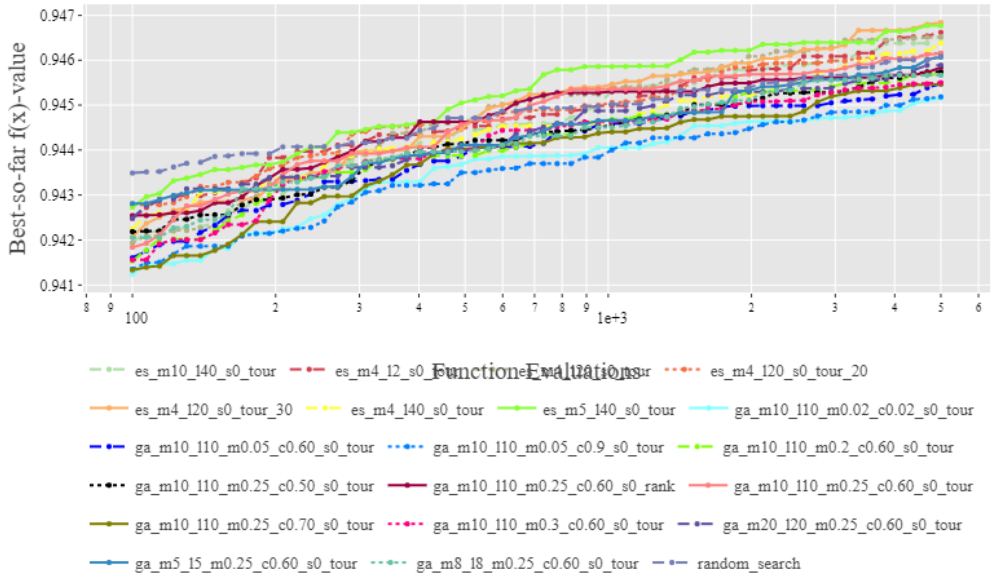
4) The validity check function written by us in these algorithms, which uses parts of the nas_ioh.py file, is not functioning as expected. This is evident because some of the solutions chosen actually exceed the edge limit of 9. They did, however, pass the validity test and returned a positive fitness value after being evaluated.

5) We believe the dataset is not large in terms of search space and evaluation results. Many solutions that are different in many bits actually return the exact fitness value after being evaluated, reducing the need for exploration. It also

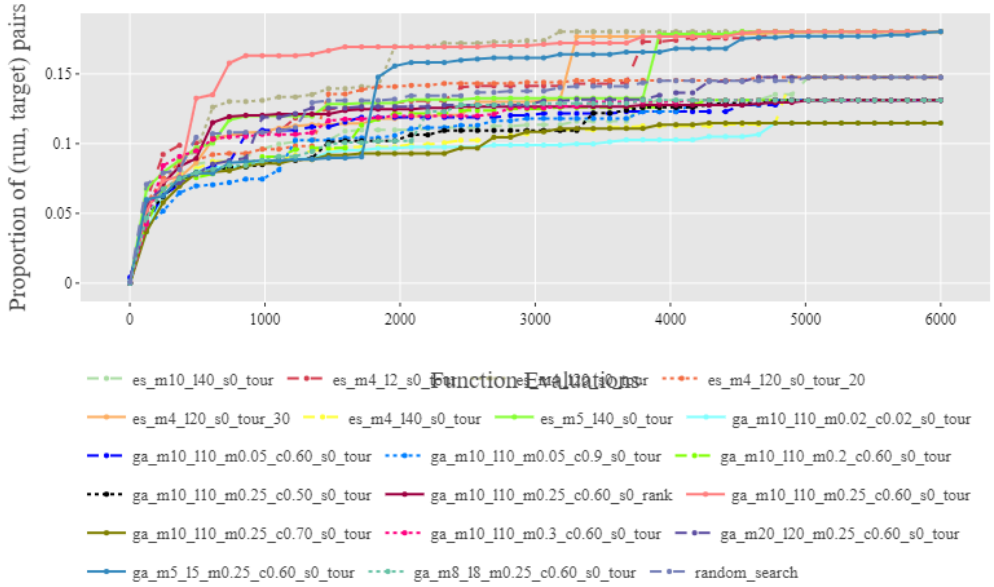
makes random search quite competent in this problem because the random search is very likely to find the optimal point even with a budget of 5000.

5 Appendix

The following Figure 8 depicts a subset of the various algorithm settings tested during the experiments. Because not all of them are extracted and explained in section 3, they are documented here.



(a) The average best-found fitness value curve of algorithms tested



(b) The ECDF curve of algorithms tested

Figure 8: Documentation of various algorithm configurations tested during the experiment

References

[1] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114, Long Beach, California, USA, 09–15 Jun 2019. PMLR.