NEURAL ARCHITECTURE SEARCH - EVOLUTIONARY ALGORITHMS

Dinu Catalin Viorel
3697037
viorel.dinu00@gmail.com

August 20, 2024

1 Introduction

Neural network has achieved significant success in applications of various domains, and this success requires novel and proper architecture designs. Neural Architecture Search (NAS) addresses the technique of automatically learning and constructing a well-performing architecture for specific tasks. The NASBench-101 provides a dataset mapping neural architectures to their training and evaluation metrics. It allows us to evaluate various neural architectures quickly and test NAS algorithms within a limited time.
NAS-Bench-101 restricts the search space of NAS in feedforward structures. Practically, it limits the size of the space as follows:

- Using only L = 3 operators:
  1. 3×3 convolution (CONV-3X3);
  2. 1×1 convolution (CONV-3X3);
  3. 3×3 max-pool (MAX-3X3);
- Consisting of directed acyclic graphs on V = 7 nodes.
- Maximal number of edges is 9.

NAS-Bench-101 represents the 7-vertex directed acyclic graph using a 7×7 upper-triangular binary matrix and uses a list of 5 labels indicating the operation for each of the 5 intermediate vertices.

2 Optimization Problem

To resolve the problem of NAS-Bench-101, we want to find the configuration with the best accuracy. To map this task to an optimisation task, we have to define a representation for every NAS-Bench-101 solution and a fitness function we want to maximise.
We will define a possible solution to the task as follows:

$$x = [x_1, x_2, ..., x_{21}, x_{22}, ..., x_{26}]$$

where:

$$x_i \in \{0, 1\}, i \in [1...21]$$
$$x_i \in \{0, 1, 2\}, i \in [22...26]$$

so:

$$x \in \{0, 1\}^{21} \times \{0, 1, 2\}^5$$

The above representation is equivalent to the following NAS structure:

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

$$\text{Operations at the 7vertices} = [\text{INPUT}, \text{type}_1, \text{type}_2, \text{type}_3, \text{type}_4, \text{type}_5, \text{OUTPUT}], \text{where:}$$

$$\text{type}_i = \begin{cases} \text{CONV-3x3, if } x_{21+i} = 0 \\ \text{CONV-1x1, if } x_{21+i} = 1 \\ \text{MAX-3x3, if } x_{21+i} = 2 \end{cases} , i \in [1...5]$$

We will define the fitness function that we want to maximise as:

$$f : \{0,1\}^{21} \times \{0,1,2\}^5 \to [0,1]$$
$$f(x) = \text{the accuracy of the model}$$

3 Algorithms

3.1 Genetic Algorithm

Algorithm 1: Genetic Algorithm

Input : population size $\to \mu$
offspring size $\to \lambda$
Crossover probability $\to p_c$
Crossover type (to decide whether to use n-point or uniform crossover) $\to$ cross-type
Environment type $\to$ es-type
mutation rate $\to p$
budget $\to B$

Termination : The algorithm terminates when the B<0 or f_opt = 1

parents $\leftarrow$ [];
parents_fit $\leftarrow$ [];
`/* Create the initial population of size` μ `of valid parents */`
while *len(parents)* $<\mu$ **do**
    x $\leftarrow$ random_sampling_model();
    if *is_valid(x)* **then**
        parents $\leftarrow$ parents $\oplus$ x;
        parents_fit $\leftarrow$ parents_fit $\oplus$ f(x);
    end
end
$B \leftarrow B$ - μ;
f_opt $\leftarrow$ max(parent_fit);
while *f_opt < 1 and B > 0* **do**
    offsprings $\leftarrow$ [];
    offsprings_fit $\leftarrow$ [];
    `/* Create the offspring population of size` λ `of valid offsprings */`
    while *B > 0 and len(offsprings)* $< \lambda$ **do**
        `/* Select 2 parents by tournament selection; Section 3.1.1 */`
        parent_1, parent_2 $\leftarrow$ tournament_selection(parents, parents_fit);
        `/* Apply crossover on those parents; Section 3.1.2 */`
        offspring_1, offspring_2 $\leftarrow$ crossover(parent_1, parent_2, cross-type);

```
    /* Apply mutation on the offsprings got from crossover; Section 3.1.3          */
    x_1 ← mutation(offspring_1, p);
    x_2 ← mutation(offspring_2, p);
    if is_valid(x_1) then
        offsprings ← offsprings ⊕ x_1;
        offsprings_fit ← offsprings_fit ⊕ f(x_1);
        B ← B - 1;
    end
    if B > 0 and len(offsprings) < λ and is_valid(x_2) then
        offsprings ← offsprings ⊕ x_2;
        offsprings_fit ← offsprings_fit ⊕ f(x_2);
        B ← B - 1;
    end
end
f_opt ← max(f_opt, max(offsprings_fit));
/* Create the new parents population; Section 3.1.4                            */
parents parent_fit ← environment_selection(parents, offsprings, ga-type)
end
```

3.1.1 Selection

For the parent selection process, we have used **tournament selection**. We assign a probability to choose a parent from the parent population proportional to their fitness.

$$
\begin{aligned}
P &= \text{all parents population} \\
|P| &= \mu \\
x_i &\in P, i \in [1...\mu] \\
f_i &= f(x_i) \\
f_{min} &= min(f_i) \\
p_i &= \text{ probability to choose i-th element for crossover} \\
p_i &= \frac{f_i - f_{min}}{\sum_{j=1}^{\mu} f_j - \mu * f_{min}}, \forall i \in [1...\mu]
\end{aligned}
$$

Having the probabilities for all the parents from the current population, we will sample 2 different random parents(parent$_1$, parent$_2$) for crossover.

3.1.2 Crossover

We have two choices for crossover operations:

- **Uniform crossover** - Fig. 1
  For every bit in the initial two parents, we generate a Bernoulli random variable with a probability of 0.5 (we call the set of those Bernoulli trials the crossover mask). The first offspring is generated by taking elements from the first parent if the mask is 0 and from the second parent if the mask is 1. The second offspring is generated by taking elements from the first parent if the mask is 1 and from the second parent if the mask is 0.

$$
\begin{aligned}
\text{parent}_1 &= [x_1^1, ..., x_{26}^1] \\
\text{parent}_2 &= [x_1^2, ..., x_{26}^2] \\
\text{mask} &= \text{ the crossover masl} \\
\text{mask}_i &= Bernoulli(0.5) \\
&\downarrow \textbf{crossover} \\
\text{offspring}_1 &= [y_1^1, ..., y_{26}^1] \\
\text{offspring}_2 &= [y_1^2, ..., y_{26}^2]
\end{aligned}
$$

$$y_i^1 = \begin{cases} x_i^1, & \text{if mask}_i = 0 \\ x_i^2, & \text{otherwise} \end{cases}, \forall i \in [1...26]$$

$$y_i^2 = \begin{cases} x_i^2, & \text{if mask}_i = 0 \\ x_i^1, & \text{otherwise} \end{cases}, \forall i \in [1...26]$$

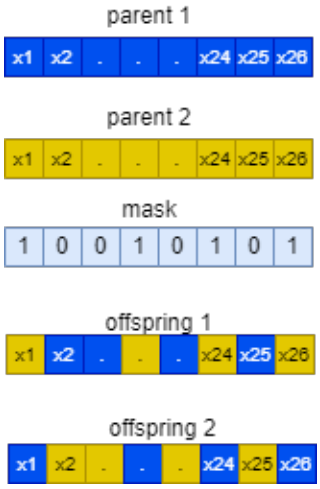

Figure 1: Uniform crossover

- **N-points crossover** - Fig. 2
  The n-points crossover operator sets n number of crossover points on both parents, so the parents are split into n+1 parts. We will use alternating those parts to generate the offsprings. The first offspring comprises the odd i-th parts from the first parent and the even i-th parts from the second parent. The second offspring comprises the even i-th parts from the first parent and the odd i-th parts from the second parent.
  We split the parents into n+1 parts for the n-point crossover

$$\begin{aligned} \text{parent}_1 &= [\text{part}_1^1, ..., \text{part}_{n+1}^1] \\ \text{parent}_2 &= [\text{part}_1^2, ..., \text{part}_{n+1}^2] \\ |\text{part}_i^1| &= |\text{part}_i^2|, \forall i \in [1...(n+1)] \\ &\downarrow \textbf{crossover} \\ \text{offspring}_1 &= [\text{part}_1^1, \text{part}_2^2, ..., \text{part}_{n+1}^{(n+1)\%2}] \\ \text{offspring}_2 &= [\text{part}_1^2, \text{part}_2^1, ..., \text{part}_{n+1}^{((n+1)+1)\%2}] \end{aligned}$$

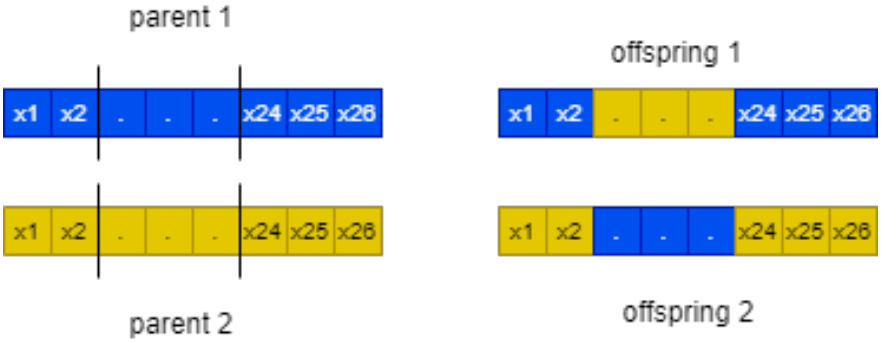

Figure 2: 2 points crossover

For both operations, we have a probability p_c to happen. If the crossover is not happening, offspring$_1$ = parent$_1$ and offspring$_2$ = parent$_2$.

3.1.3 Mutation

We defined our **mutation operator** to work with our representation. The first 21 bits ($x_i \in \{0,1\}, \forall i \in [1...21]$) are flipped from their initial state to the opposite state. The last 5 bits ($x_i \in \{0,1,2\}, \forall i \in [21...26]$) are converted to either $x_i - 1$ or $x_i + 1$, both with equal probability (0.5). We consider our set ({0,1,2}) to be periodic (fig. 3). For every bit of the representation, there is a probability(p) that the mutation will happen.

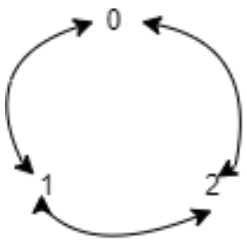

Figure 3: Cycling set {0,1,2}

$$
\begin{aligned}
&x = [x_1, x_2, ..., x_{21}, x_{22}, ..., x_{26}] \\
&\quad \downarrow \textbf{mutation} \\
&x' = [x'_1, x'_2, ..., x'_{21}, x'_{22}, ..., x'_{26}] \\
&s_i = \mathcal{B}ernoulli(p) \leftarrow \text{if mutation happens on the i-th bit} \\
&x'_i = \begin{cases} 1 - x'_i, \text{ if } s_i == 1 \\ x'_i, \text{ otherwise} \end{cases} \quad , \forall i \in [1...21] \\
&x'_i = \begin{cases} \begin{cases} x_i + 1, \text{ if } \mathcal{U}(0,1) < 0.5 \\ x_i - 1, \text{ otherwise} \end{cases} & \text{if } s_i == 1 \\ xs_i, \text{ otherwise} \end{cases} \quad \forall i \in [21...26]
\end{aligned}
$$

3.1.4 Environment Selection

By environment selection, we mean the strategy we used to create a new population from the current population of parents and offsprings. We distinguish 2 different types:

- **Comma** (μ, λ) **environment selection**
  In this context, a new population is selected only from the newly generated offsprings. The new population comprises the top μ best by fitness from the offspring population.

$$
\begin{aligned}
P_t &= \text{ parent population at time } t \\
|P_t| &= \mu \\
O_t &= \text{ offspring population at time } t \\
|O_t| &= \lambda, (\lambda >> \mu) \\
P_{t+1} &= \text{ parent population at time } t+1 \\
P_{t+1} &= top_\mu(O_t)
\end{aligned}
$$

- **Plus** $(\mu + \lambda)$ **environment selection** In this context, a new population is selected from the reunion of parents and newly generated offsprings. The new population comprises the top μ best by fitness from the reunion of offspring and parent population.

$$
P_t = \text{ parent population at time } t
$$

$$
\begin{aligned}
|P_t| &= \mu \\
O_t &= \text{ offspring population at time } t \\
|O_t| &= \lambda \\
P_{t+1} &= \text{ parent population at time } t+1 \\
P_{t+1} &= top_\mu(P_T \cup O_t)
\end{aligned}
$$

3.2 Evolution Strategy

3.2.1 Representation

The nature of the NAS-Bench-101 problem implies optimising a function of discrete variables. To use evolution strategies, we propose the following encoding of the solution in continuous space:

$$
\begin{aligned}
z &= [z_1, ..., z_{21}, z_{22}, ..., z_{36}] \\
z &\in [0,1]^{21} \times \mathbb{R}^{15}
\end{aligned}
$$

Where, for the first 21 bits we define z_i as the probability of x_i to be 1.

$$z_i = \mathbb{P}(x_i = 1) \in [0,1], i \in [1...21]$$

For the last 5 elements of x, we assign a weight to all the 3 possible values (0,1,2). Then to decode it we will compute $\mathbb{P}(x_i = 0), \mathbb{P}(x_i = 1), \mathbb{P}(x_i = 2)$

$$z_i \in \mathbb{R}, i \in [22...36]$$

We decode this representation to the discrete representation on which we can apply f.Firstly, we decode the binary part of x:

$$x_i = \begin{cases} 0, \text{ if } z_i < 0.5 \\ 1, \text{ if } z_i \geq 0.5 \end{cases}, i \in [1...21]$$

Lastly, we decode the $x_{21} \rightarrow x_{26}$, where we choose the value with the highest probability.

$$
\begin{aligned}
&\forall i \in [21...26] : \\
&v_0 = z_{21+(i-21)*3} \\
&v_1 = z_{21+(i-21)*3+1} \\
&v_2 = z_{21+(i-21)*3+2} \\
&\mathbb{P}(x_i = k) = \frac{e^{v_k}}{e^{v_0} + e^{v_1} + e^{v_2}} \\
&x_i = k, \text{ where: } \mathbb{P}(x_i = k) = maximum
\end{aligned}
$$

We distinguish two different representations depending on the type of step from the Evolution Strategies:

- Single step size (one σ):

$$z = [[z_1, ..., z_{36}], \sigma]$$

- Individual step size (individual σ_i):

$$z = [[z_1, ..., z_{36}], [\sigma_1, ..., \sigma_{36}]]$$

3.2.2 Pseudo-code

Algorithm 2: Evolution Strategy

Input : step-type ← type of steps (one σ / individual σ_i)
μ ← population size
λ ← no. of generated offspring
ρ ← no. of parents used for recombination
τ_0 ← learning rate (it is uses if step-type is one σ)
τ' ← global learning rate (it is used if step-type is individual σ_i)
τ ← local learning rate (it is used if step-type is individual σ_i)
es-type ← type of environment selection (+/,)
B ← budget

Termination : B = 0 or f_opt = 1

parents ← [];
parents_σ ← [];
parents_fit ← [];
`/* Create the initial population of size` μ `of valid parents */`
while *len(parents)* <μ **do**
  z,σ ← random_sampling_model(step-type);
  decoded_z ← decode(z);
  if *is_valid(decoded_z)* **then**
    parents ← parents $\oplus$ z;
    parents_σ ← parents $\oplus$ σ;
    parents_fit ← parents_fit $\oplus$ f(decoded_z);
  end
end
$B \leftarrow B$ - μ;
f_opt ← max(parent_fit);
`/* Create the offspring population of size` λ `of valid offsprings */`
while *f_opt != 1 and* B *> 0* **do**
  offsprings ← [];
  offsprings_σ ← [];
  offsprings_fit ← [];
  while *len(offsprings)* < λ *and* B *> 0* **do**
    `/* Apply recombination on a subset of parents parents; Section 3.2.3 */`
    z, σ ← recombination(parents, parents_σ, ρ);
    `/* Apply mutation on the generated ofspring; Section 3.2.4 */`
    z, σ ← mutation(z, σ, step-type, τ_0, τ, τ');
    decoded_z ← decode(z);
    if *is_valid(decoded_z)* **then**
      offsprings ← offsprings $\oplus$ z;
      offsprings_σ ← offsprings_σ $\oplus$ σ;
      offsprings_fit ← offsprings_fit $\oplus$ f(decoded_z);
      $B \leftarrow B - 1$;
    end
  end
  f_opt ← max(f_opt, max(offsprings_fit));
  `/* Create the new parents population; Section 3.2.5 */`
  parents, parents_σ, parent_fit ← environment_selection(parents, offsprings, es-type)
end

3.2.3 Recombination

We used an ρ-intermediary recombination. We uniformly random sample ρ parents from the population of all parents. The newly generated offspring consists of the mean values of the selected parents.
Let:

$$P = \text{all parents population}$$
$$|P| = \mu$$

We uniformly random sample a set of parents to recombine:

$$\begin{aligned} Y &= \text{the set of randomly selected parents from } P \\ |Y| &= \rho \\ o &= \text{the offspring} \\ o &= \frac{1}{\rho}\sum_{z \in Y} z \end{aligned}$$

For $\rho = 2$, we have standard intermediary recombination with two parents. For $\rho = \mu$, we have global intermediary recombination.

3.2.4 Mutation

The mutation operator depends on the type of step used for representing the solution:

- **Single step size** (σ)

$$\begin{aligned} z &= [[z_1, ..., z_{36}], \sigma] \\ &\downarrow \textbf{mutation} \\ z' &= [[z'_1, ..., z'_{36}], \sigma'] \\ \sigma' &= \sigma * e^{\mathcal{N}(0,\tau_0)} \\ z'_i &= z_i + \mathcal{N}(0, \sigma'), \forall i \end{aligned}$$

- **Individual step size** (σ_i)

$$\begin{aligned} z &= [[z_1, ..., z_{36}], [\sigma_1, ..., \sigma_{36}]] \\ &\downarrow \textbf{mutation} \\ z' &= [[z'_1, ..., z'_{36}], [\sigma'_1, ..., \sigma'_{36}]] \\ g &= \mathcal{N}(0, \tau') \text{ for global learning rate we sample only one to use for all } \sigma_i \\ \sigma'_i &= \sigma_i * e^{(g+\mathcal{N}(0,\tau))} \\ z'_i &= z_i + \mathcal{N}(0, \sigma'_i) \end{aligned}$$

3.2.5 Environment Selection

By environment selection, we mean the strategy we used to create a new population from the current population of parents and offsprings. We distinguish 2 different types:

- **Comma** (μ, λ) **environment selection**
  In this context, a new population is selected only from the newly generated offsprings. The new population comprises the top μ best by fitness from the offspring population.

$$\begin{aligned} P_t &= \text{ parent population at time } t \\ |P_t| &= \mu \\ O_t &= \text{ offspring population at time } t \\ |O_t| &= \lambda, (\lambda >> \mu) \\ P_{t+1} &= \text{ parent population at time } t+1 \\ P_{t+1} &= top_\mu(O_t) \end{aligned}$$

- **Plus** $(\mu + \lambda)$ **environment selection** In this context, a new population is selected from the reunion of parents and newly generated offsprings. The new population comprises the top μ best by fitness from the reunion of offspring and parent population.

$$\begin{aligned} P_t &= \text{ parent population at time } t \\ |P_t| &= \mu \end{aligned}$$

$$
\begin{aligned}
O_t &= \text{ offspring population at time } t \\
|O_t| &= \lambda \\
P_{t+1} &= \text{ parent population at time } t+1 \\
P_{t+1} &= top_\mu(P_T \cup O_t)
\end{aligned}
$$

4 Experimental Results

For our experiments we will compute 3 metrics:

- Expecting Running Time (ERT)
- Empirical Cumulative Distribution Function of the running time (ECDF)
- Area under ECDF curve (AUC)

To combat randomness, we test every configuration for 20 different runs. Moreover, to compare how fast the algorithms converged, we have a fixed budget of 5000 function evaluations.

4.1 Genetic Algorithms

4.1.1 Crossover

We want to study which crossover algorithm and which crossover rate are the best in the context of the genetic algorithm. For the experiments presented in Table1, we used the following fixed hyperparameters:

- $\mu = 10$
- $\lambda = 10$
- environment selection type = "+"
- mutation rate = $\frac{1}{26}$

Algorithm name	**Crossover_Type**	**Crossover_Rate**	**Mean** $f(x_{run}^{best})$	**AUC**
ga(10+10)_1point_6_26	1 point	0.6	0.948	0.1537
ga(10+10)_1point_95_26	1 point	0.95	0.948	0.1503
ga(10+10)_2point_6_26	2 point	0.6	0.949	0.1451
ga(10+10)_2point_95_26	2 point	0.95	**0.950**	**0.1555**
ga(10+10)_uniform_6_26	uniform	0.6	0.948	0.1425
ga(10+10)_uniform_95_26	uniform	0.95	0.949	0.1455

Table 1: The AUC value and mean best fitness achieved over the 20 runs for different crossovers and crossover rates.

From Table 1, we can observe that a higher crossover rate increase both the mean best value over the 20 runs and the AUC value. Moreover, the highest values for both metrics is achieved by the 2-point crossover.

4.1.2 Mutation

We have also experimented with different Mutation rates to find which of these rates are the best for the genetic algorithm. For the experiments presented in Table2, we used the following fixed hyperparameters:

- $\mu = 10$
- $\lambda = 10$
- environment selection type = "+"
- crossover type = "uniform"
- crossover rate = 0.95

Algorithm name	Mutation_Rate	Mean $f(x_{run}^{best})$	AUC
ga(10+10)_uniform_95_21	1/21	0.948	**0.1576**
ga(10+10)_uniform_95_26	1/26	0.949	0.1455
ga(10+10)_uniform_95_5	1/5	0.949	0.1546

Table 2: The AUC value and mean best fitness achieved over the 20 runs for different mutation rates.

From Table 2, we can observe that AUC has the highest value for a mutation rate of 1/21, but it does not have the highest mean value.
Higher value for AUC means that we have a higher number of solutions closer to the optimal fitness, which does not necessarily mean that all of these solutions converge to the optimum value. Hence sometimes for the highest value of AUC, we may not have the highest mean value.

4.2 Evolution Strategy

4.2.1 Recombination

In this section we want to see how the number of parents used for recombination influence the performance of the algorithm. For the experiments in this section, we used a fixed value for the following hyperparameters:

- $\mu = 10$
- $\lambda = 10$
- environment selection type = "+"
- One σ : $\tau_0 = \frac{1}{\sqrt{35}}$
- Individual σ_i : $\tau' = \frac{1}{\sqrt{35}}$ and $\tau = \frac{1}{\sqrt{2\sqrt{35}}}$

Algorithm name	Step Type	ρ	Mean $f(x_{run}^{best})$	AUC
es(10+10)_1step_ro2	One σ	2	0.949	0.1577
es(10+10)_1step_ro3	One σ	3	0.949	**0.1613**
es(10+10)_1step_ro-half	One σ	$\frac{\mu}{2}$	0.949	0.1579
es(10+10)_1step_ro-all	One σ	μ	0.949	0.1448
es(10+10)_nstep_ro2	Individual σ_i	2	0.949	0.1497
es(10+10)_nstep_ro3	Individual σ_i	3	0.948	0.1524
es(10+10)_nstep_ro-half	Individual σ_i	$\frac{\mu}{2}$	0.950	0.1518
es(10+10)_nstep_ro-all	Individual σ_i	μ	0.950	**0.1543**

Table 3: The AUC value and mean best fitness achieved over the 20 runs for different numbers of parents used for recombination.

From Table 3, we can observe that although the single step has the highest AUC value when ρ = 3, we cannot see a clear trend for which the value of AUC might increase or decrease. On the other hand, for individual step we can see that the value of AUC increases for higher values of ρ.

4.2.2 Mutation

We also want to evaluate how different learning rates affects the evolution strategies. For both type of evolution strategies (One σ and Individual σ_i) we have the following fixed hyperparameters:

- $\mu = 10$
- $\lambda = 10$
- environment selection type = "+"
- $\rho = \mu$

Algorithm name	Step Type	τ_0	τ'	τ	Mean $f(x_{run}^{best})$	AUC
es(10+10)_1step_lr15	One σ	$\frac{1}{\sqrt{15}}$	#	#	0.948	**0.1577**
es(10+10)_1step_lr21	One σ	$\frac{1}{\sqrt{21}}$	#	#	0.949	0.1500
es(10+10)_1step_lr35	One σ	$\frac{1}{\sqrt{35}}$	#	#	0.949	0.1448
es(10+10)_nstep_lr15	Individual σ_i	#	$\frac{1}{\sqrt{35}}$	$\frac{1}{\sqrt{2*\sqrt{15}}}$	0.949	**0.1623**
es(10+10)_nstep_lr21	Individual σ_i	#	$\frac{1}{\sqrt{35}}$	$\frac{1}{\sqrt{2*\sqrt{21}}}$	0.950	0.1508
es(10+10)_nstep_lr35	Individual σ_i	#	$\frac{1}{\sqrt{35}}$	$\frac{1}{\sqrt{2*\sqrt{35}}}$	0.950	0.1543

Table 4: The AUC value and mean best fitness achieved over the 20 runs for different learning rate

From Table 2, we observe how for higher learning we get higher AUC, while for lower learning rate we get higher mean best fitness over the 20 runs. A bigger learning rate helps the algorithm to get faster in the proximity of the optimum value, but it makes it converge slower.

4.3 Genetic Algorithms vs Evolution Strategies

In the end we want to compare Genetic Algorithms with Evolution Strategies. In this section, we will compare different configuration of those algorithms (λ $(+/,)$ μ). We have the following fixed algorithms:

- Genetic Algorithm
  - Uniform crossover
  - Crossover rate = 0.95
  - Mutation rate = $\frac{1}{21}$
- Evolution Strategie
  - Individual σ_i
  - $\rho = \mu$
  - $\tau' = \frac{1}{\sqrt{35}}$
  - $\tau = \frac{1}{\sqrt{2\sqrt{35}}}$

Algorithm name	μ	λ	Environment type	Mean $f(x_{run}^{best})$	AUC
ga(10+10)_uniform_95_21	10	10	Plus(+)	0.949	0.1577
ga(10,10)_uniform_95_21	10	10	Comma(,)	0.946	0.1426
ea(10+10)_nstep	10	10	Plus(+)	0.950	**0.1543**
ea(10,10)_nstep	10	10	Comma(,)	0.945	0.1066
ga(10+30)_uniform_95_21	10	30	Plus(+)	0.949	0.1604
ga(10,30)_uniform_95_21	10	30	Comma(,)	0.949	0.1601
ea(10+30)_nstep	10	30	Plus(+)	0.950	0.1509
ea(10,30)_nstep	10	30	Comma(,)	0.948	0.1551
ga(20+20)_uniform_95_21	20	20	Plus(+)	0.949	**0.1628**
ga(20,20)_uniform_95_21	20	20	Comma(,)	0.946	0.1509
ea(20+20)_nstep	20	20	Plus(+)	0.950	0.1011
ea(20,20)_nstep	20	20	Comma(,)	0.946	0.1434

Table 5: The AUC value and mean best fitness achieved over the 20 runs for different (λ $(+/,)$ μ)- configurations of GA and ES

From Table 5, we observe that Comma(,) environment selection is worse compared to Plus(+) environment selection. Overall, GA gets better values for AUC, while ES get better values for mean best fitness over the 20 runs. Green rows in Table 5 represent the configuration used in our submitted implementations.

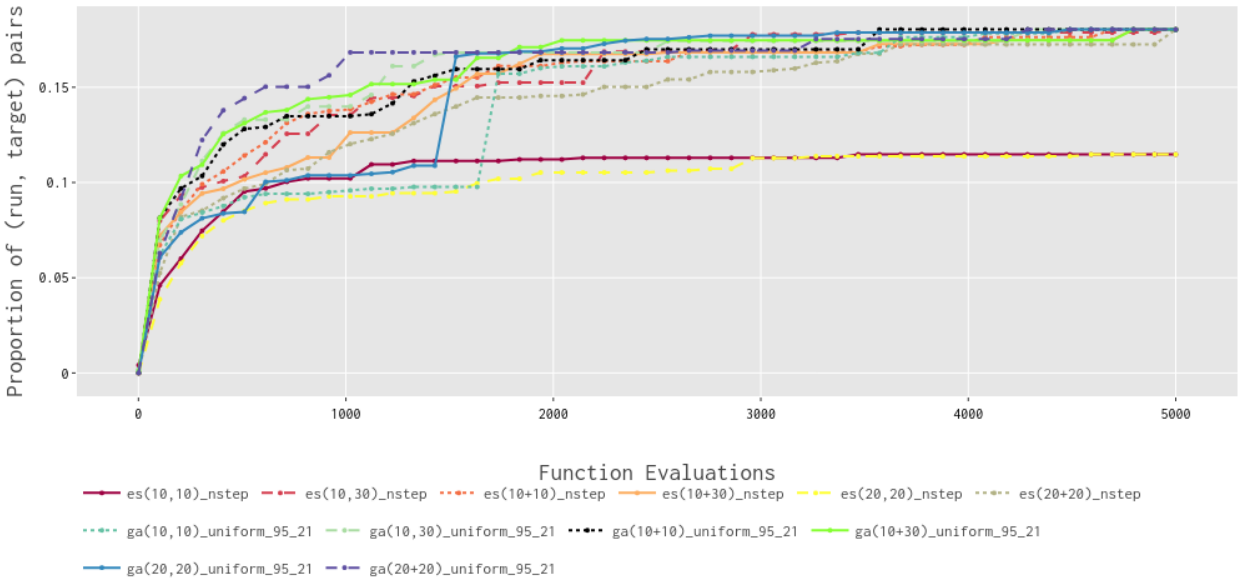

Figure 4: ECDF for different configurations of GA and ES

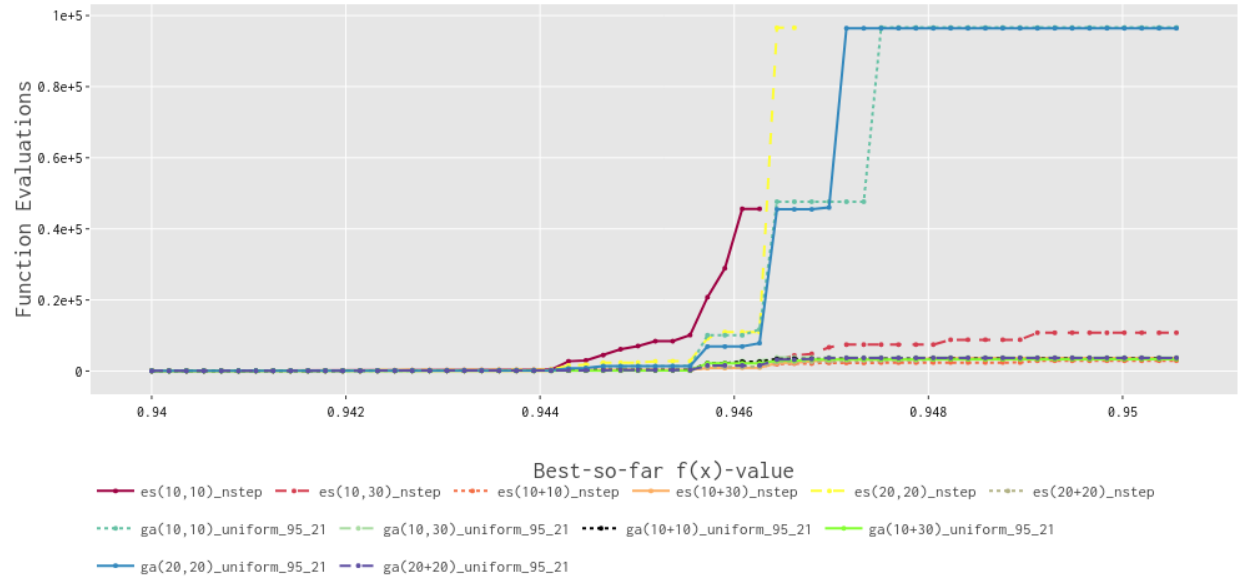

Figure 5: ERT for different configurations of GA and ES

From fig. 5, we observe that comma(,) environment selection converges way slower than the plus(+) environment selection counterpart.

5 Discussion and Conclusion

From our experiments, we can conclude:

1) We suggest using individual σ_i for evolution strategy to resolve NAS problem, having better overall results.
2) We observe that the evolution strategy and genetic algorithms benefits from plus(+) environment selection for solving the NAS problem.
3) Overall, Genetic Algorithms get better results than Evolution Strategies. The nature of the NAS problem implies using discrete variables with which genetic algorithms can work directly.
4) We recommend using uniform crossover and a crossover probability of 0.95 for solving the NAS problem with Genetic Algorithms