

# Thuật toán Định hướng Tiến hóa

*Giới thiệu thuật toán Định hướng Tiến hoá với kỹ thuật Ước lượng hiệu suất hiệu quả trong tìm kiếm kiến trúc mạng*

Nhóm 8 - Neural Architecture Search

Ngày 1 tháng 12 năm 2023

# Mục lục

Thuật toán Tiến hóa Hiệu chỉnh

Thuật toán Định hướng Tiến hóa

Chi tiết triển khai

Ước lượng proxy không tổn kém

## Thuật toán Tiến hóa Hiệu chỉnh

Tiến hóa lão hóa

Tóm tắt thuật toán

## Thuật toán Định hướng Tiến hóa

Hạn chế của các phương pháp hiện tại

Thuật toán đề xuất

## Chi tiết triển khai

Phát biểu bài toán

Tóm tắt thuật toán

## Ước lượng proxy không tổn kém

Ma trận hiệp phương sai Jacobi

Công thức tính toán

## Tìm kiếm tiền hóa

Tiến hóa là một phương pháp đơn giản nhưng hiệu quả để khám phá ra các kiến trúc chất lượng cao, nhưng tồn tại các khuyết điểm như:

- **Tập trung quá mức vào các giải pháp tối ưu địa phương:** Các thuật toán này có xu hướng mắc kẹt ở các giải pháp tốt nhất địa phương mà không khám phá đủ rộng các phần khác của không gian tìm kiếm.
- **Đòi hỏi nhiều tài nguyên tính toán:** Do sự khám phá không gian lớn, các thuật toán này thường cần nhiều thời gian và tài nguyên máy tính để tìm ra giải pháp tối ưu.
- **Khó khăn trong việc điều chỉnh các tham số:** Cần phải cân nhắc kỹ lưỡng việc thiết lập các tham số như tỷ lệ đột biến, kích thước quần thể, v.v., điều này đòi hỏi kiến thức sâu và kinh nghiệm.

## Điểm mới của Thuật toán Tiến hóa lão hóa

Thuật toán Tiến hóa lão hóa<sup>1</sup> cải tiến so với thuật toán tiến hóa thông thường ở các điểm sau:

- **Ưu tiên các giải pháp mới:** Bằng cách loại bỏ các giải pháp cũ hơn trong quần thể, thuật toán khuyến khích sự đa dạng và ngăn chặn sự tập trung quá mức vào các giải pháp tối ưu địa phương.
- **Khám phá hiệu quả hơn:** Sự lão hóa giúp đảm bảo rằng không gian tìm kiếm được khám phá một cách cân bằng hơn, không chỉ tập trung vào những gì đã biết.
- **Đáp ứng nhanh với môi trường đổi thay:** Do các thành viên mới liên tục được thêm vào, thuật toán có khả năng thích ứng tốt hơn với những thay đổi trong môi trường hoặc các yêu cầu mới.

---

<sup>1</sup>Regularized Evolution for Image Classifier Architecture Search

## Mã giả thuật toán

---

**Algorithm 1** Aging Evolution

---

```
population  $\leftarrow$  empty queue ▷ The population.  
history  $\leftarrow \emptyset$  ▷ Will contain all models.  
while  $|population| < P$  do ▷ Initialize population.  
    model.arch  $\leftarrow$  RANDOMARCHITECTURE()  
    model.accuracy  $\leftarrow$  TRAINANDEVAL(model.arch)  
    add model to right of population  
    add model to history  
end while  
while  $|history| < C$  do ▷ Evolve for  $C$  cycles.  
    sample  $\leftarrow \emptyset$  ▷ Parent candidates.  
    while  $|sample| < S$  do  
        candidate  $\leftarrow$  random element from population  
        ▷ The element stays in the population.  
        add candidate to sample  
    end while  
    parent  $\leftarrow$  highest-accuracy model in sample  
    child.arch  $\leftarrow$  MUTATE(parent.arch)  
    child.accuracy  $\leftarrow$  TRAINANDEVAL(child.arch)  
    add child to right of population  
    add child to history  
    remove dead from left of population ▷ Oldest.  
    discard dead  
end while  
return highest-accuracy model in history
```

---

Hình 1: Quy trình thuật toán Tiến hóa lão hóa

## Mô tả thuật toán

1. **Khởi tạo Quần thể:** Quần thể ban đầu được khởi tạo với số lượng  $P$  mô hình có kiến trúc ngẫu nhiên. Mỗi mô hình trong quần thể được đánh giá thông qua quá trình huấn luyện và đánh giá.
2. Lặp cho đến  $C$  chu kỳ: Thuật toán tiến hành trong  $C$  chu kỳ. Trong mỗi chu kỳ, quá trình sau được thực hiện:
  - **Lựa chọn cha mẹ và sinh sản:** Chọn một mô hình từ quần thể (dựa trên tiêu chí nào đó như độ chính xác), sau đó tạo ra mô hình con mới thông qua quá trình đột biến.
  - **Đánh giá mô hình con:** Mô hình con mới được huấn luyện và đánh giá để xác định độ chính xác của nó.
  - **Cập nhật quần thể:** Thêm mô hình con mới vào quần thể và lưu trữ lịch sử của tất cả các mô hình.
  - **Lão hóa:** Loại bỏ mô hình cũ nhất từ quần thể. Điều này tạo điều kiện cho các mô hình mới được thêm vào và giúp đảm bảo đa dạng hóa trong quá trình tìm kiếm.

## Thuật toán Tiến hóa Hiệu chỉnh

Tiến hóa lão hóa

Tóm tắt thuật toán

## Thuật toán Định hướng Tiến hóa

Hạn chế của các phương pháp hiện tại

Thuật toán đề xuất

## Chi tiết triển khai

Phát biểu bài toán

Tóm tắt thuật toán

## Ước lượng proxy không tổn kém

Ma trận hiệp phương sai Jacobi

Công thức tính toán



## Khát quát về các phương pháp tìm kiếm

Trong Tìm kiếm kiến trúc mạng nơ-ron (NAS), các phương pháp phổ biến là: Học tăng cường (RL), Chiến lược tiến hóa (ES), và Giải thuật dựa trên gradient.(DARTS). Mặc dù đạt được kết quả xuất sắc nhưng chi phí đánh giá của hầu hết đều cao, có thể lên đến hàng tháng GPU.

- Để giảm bớt gánh nặng này, xuất hiện các phương pháp tập trung vào thiết kế dựa trên khối (cell): thiết kế các cell nhỏ được nhân bản thông qua một bộ khung ngoài, do đó giảm bớt sự phức tạp của không gian tìm kiếm.
- Hơn nữa, một số chiến lược ước lượng hiệu suất đã được đề xuất để giảm thời gian của các phương pháp NAS, chủ yếu bằng cách thực hiện các ước lượng độ tin cậy thấp<sup>2</sup> (low-fidelity estimates), dự đoán đường cong học tập<sup>3</sup> (learning curve extrapolations), các phương pháp thông kê<sup>4</sup> (quá trình Gauss) hoặc bằng các phương pháp một lần (one-shot methods), nơi trọng số của các mô hình được tạo ra được thừa kế từ một siêu mạng

---

<sup>2</sup>Constrained multifidelity optimization using model calibration

<sup>3</sup>Efficient Bayesian Learning Curve Extrapolation using Prior-Data Fitted Networks

<sup>4</sup>GP-NAS: Gaussian Process Based Neural Architecture Search

## Hạn chế phải đổi mặt

Tuy nhiên, việc tìm kiếm trong không gian có số chiều cao là rất phức tạp, ngay cả khi đã có kiến thức trước về không gian đó. Hầu hết các phương pháp NAS nổi bật không thành công khi tổng quát hóa với các tập dữ liệu mới do hội tụ nhanh chóng đến cực tiểu địa phương. Do đó gây nguy hiểm cho việc tìm kiếm và khả năng áp dụng của phương pháp.

- Cách tiếp cận đáng tin cậy nhất để có thông tin về không gian tìm kiếm trong khi tìm kiếm là đào tạo đầy đủ các kiến trúc được tạo ra và tối ưu hóa việc tìm kiếm dựa trên những cái hiệu suất nhất. Tuy nhiên, điều này rất tốn kém, và kết quả phụ thuộc rất nhiều vào các lược đồ đào tạo và cài đặt khởi tạo.
- Do đó, các ước lượng proxy không tốn kém (zero-cost proxies) trình bày một giải pháp hấp dẫn, nơi thông kê được rút ra từ các kiến trúc được tạo ra để ghi điểm cho chúng ở giai đoạn khởi tạo, do đó không cần huấn luyện. Các phương pháp này hiệu quả về thời gian và có khả năng thực hiện các mối tương quan tốt giữa điểm số và độ chính xác tương ứng khi các kiến trúc được huấn luyện.


## Khát quát Định hướng Tiến hóa (GEA)

Nhóm tác giả đề xuất một phương pháp NAS tận dụng ước lượng proxy không tốn kém để hướng dẫn tìm kiếm một cách hiệu quả<sup>5</sup>.

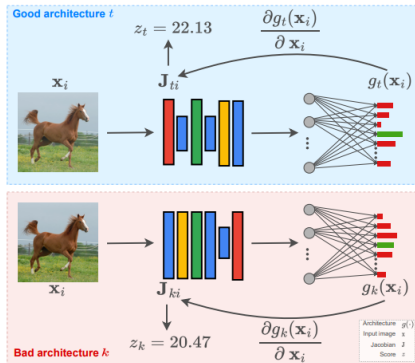
- Bằng cách sử dụng một chiến lược tiến hóa nơi các toán tử có thể bị đột biến và các kiến trúc mới hơn được ưu tiên, GEA buộc phải khai thác các kiến trúc hiệu suất nhất và khám phá không gian tìm kiếm bằng cách thực hiện các đột biến.
- Hơn nữa, chúng ta giải quyết vấn đề thực hiện đánh giá đầy đủ các kiến trúc được tạo ra để có kiến thức về không gian tìm kiếm bằng cách tạo ra nhiều kiến trúc trong mỗi thế hệ, nơi tất cả đều được đánh giá ở giai đoạn khởi tạo bằng một ước lượng proxy không tốn kém và chỉ có kiến trúc có điểm số cao nhất được huấn luyện và giữ lại cho thế hệ tiếp theo.

Làm như vậy, GEA có khả năng liên tục rút ra kiến thức về không gian tìm kiếm mà không làm ảnh hưởng đến việc tìm kiếm, dẫn đến kết quả hàng đầu trong không gian tìm kiếm NAS-Bench-101, NAS-Bench-201 và TransNAS-Bench101.

---

<sup>5</sup>Guided Evolutionary Neural Architecture Search With Ecient Performance Estimation 

## Minh họa đánh giá cá thể<sup>2</sup>



**Hình 2:** Đánh giá 2 kiến trúc sử dụng cùng đầu vào. Các kiến trúc được tạo ra ở mỗi thế hệ được xếp hạng dựa trên một điểm số tương quan với hiệu suất cuối cùng của chúng, điều này quyết định kiến trúc nào được chọn để trở thành một phần của quần thể.

## Thuật toán Tiến hóa Hiệu chỉnh

Tiến hóa lão hóa

Tóm tắt thuật toán

## Thuật toán Định hướng Tiến hóa

Hạn chế của các phương pháp hiện tại

Thuật toán đề xuất

## Chi tiết triển khai

Phát biểu bài toán

Tóm tắt thuật toán

## Ước lượng proxy không tổn kém

Ma trận hiệp phương sai Jacobi

Công thức tính toán

## Mục tiêu bài toán

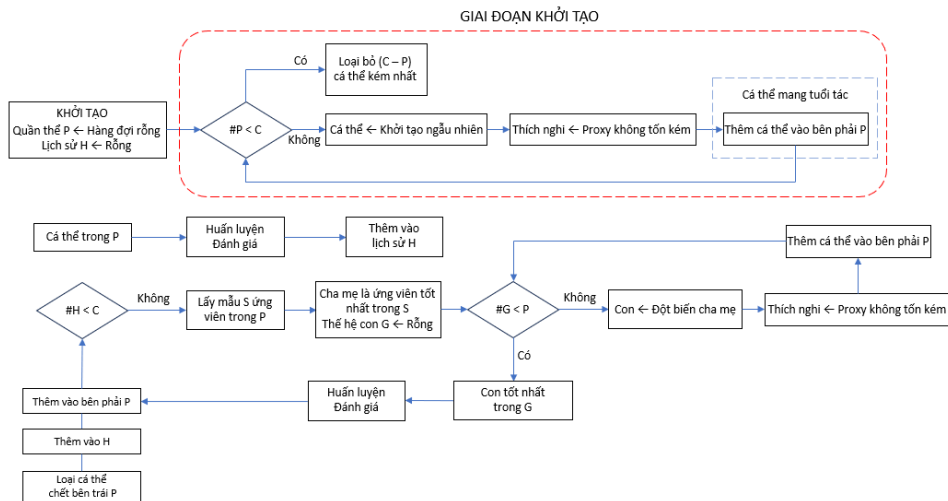
Mục tiêu của các giải thuật NAS là tìm kiếm kiến trúc  $a^*$  trong không gian các kiến trúc  $\mathcal{A}$  mà cực tiểu một hàm mục tiêu  $\mathcal{O}$ .

Thuật toán GEA phát biểu NAS dưới dạng một bài toán tối ưu sử dụng một chiến lược tiền hóa để tiền hóa các kiến trúc  $a \in \mathcal{A}$  thông qua đột biến toán tử và tiền hóa có định hướng. Chúng ta có thể định nghĩa vấn đề dưới dạng một bài toán tối ưu lồng nhau với mục tiêu là tìm kiến trúc mạng cuối cùng  $\mathbb{L}$ , tạo ra thông qua quá trình huấn luyện kiến trúc tối ưu,  $a^*$  bằng tập huấn luyện  $d^{(\text{train})}$  để mà  $a^*$  cực đại hàm mục tiêu  $\mathcal{O}$  (chẳng hạn: độ chính xác) trên tập kiểm chứng  $d^{(\text{validation})}$  như sau:

$$a^* = \arg \max_{a \in \mathcal{A}} \mathcal{O} \left( \mathbb{L} \left( a, d^{(\text{train})} \right), d^{(\text{validation})} \right)$$

Trong phần tiếp theo, ta làm rõ cách GEA sử dụng ước lượng proxy không tồn kém cho việc thiết kế cơ chế hướng dẫn tiền hóa.

## Phương pháp tìm kiếm



Hình 3: Lưu đồ Thuật toán Định hướng Tiến hóa

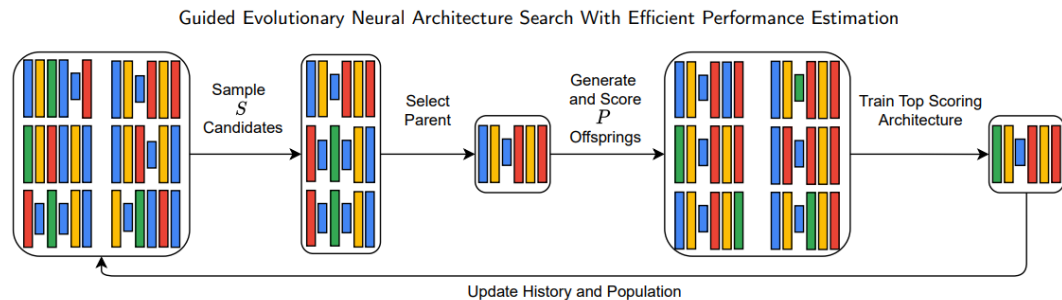
## Tóm tắt thuật toán: Giai đoạn khởi tạo

1. GEA bắt đầu bằng khởi tạo ngẫu nhiên  $C$  kiến trúc từ không gian tìm kiếm  $\mathcal{A}$  bằng cách lấy ngẫu nhiên đồng đều.
2. Đánh giá các kiến trúc bằng chỉ số proxy không tổn kém, mà không thông qua bất kì huấn luyện nào.
3. Sau đó, từ  $C$  kiến trúc được đánh giá, chỉ có  $P$  kiến trúc đạt điểm cao nhất được thêm vào quần thể và được huấn luyện để thu được độ thích nghi  $f$ , là độ chính xác trên tập kiểm chứng sau một số ít vòng lặp.

Bằng cách đánh giá  $C$  kiến trúc ở giai đoạn khởi tạo, GEA thu được kiến thức về không gian tìm kiếm, sau đó được khai thác bằng cách chọn kiến trúc hiệu suất cao nhất, do đó hướng dẫn tìm kiếm sắp tới bằng cách loại bỏ các kiến trúc xấu. Một khi quần thể ban đầu được xác định, sự tiến hóa diễn ra trong một số chu kỳ.



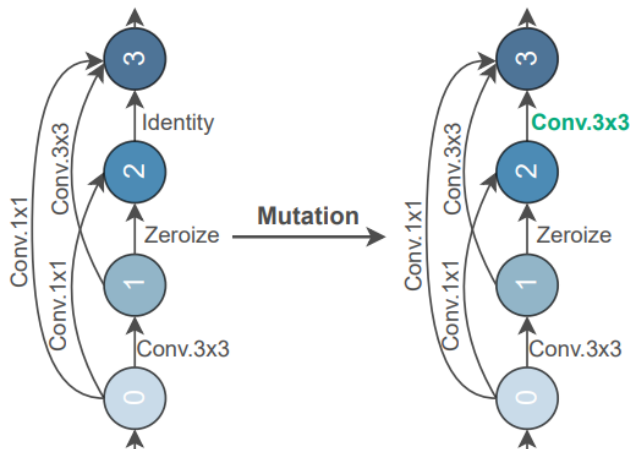
## Tóm tắt thuật toán: Giai đoạn tiến hóa



Hình 4: Quy trình tiến hóa

1. Tại mỗi lần lặp, bước đầu tiên là thực hiện chọn lọc cha mẹ theo thể thức giải đấu (tournament selection):
  - i  $S$  kiến trúc được lấy mẫu ngẫu nhiên và đồng đều từ quần thể.
  - ii Sau đó, kiến trúc có độ thích nghi  $f$  cao nhất, từ nhóm  $S$  kiến trúc được chọn để làm cha mẹ của thế hệ tiếp theo.
  - iii Để tạo ra các kiến trúc mới, GEA thực hiện một đột biến trên kiến trúc cha mẹ, bằng cách thay thế toán tử của kiến trúc bởi một toán tử khác trong tập các toán tử ứng viên.
2.  $P$  kiến trúc mới được tạo ra ở mỗi thế hệ bằng cách thực hiện các đột biến toán tử trên cha mẹ được chọn, sau đó được đánh giá bằng ước lượng proxy không tổn kém. Kiến trúc có điểm số cao nhất được giữ lại và thêm vào quần thể sau khi đánh giá độ thích nghi của nó.

Việc tạo ra và đánh giá  $P$  kiến trúc tăng cường phương pháp tìm kiếm để tìm hướng tốt nhất cho sự tiến hóa của cha mẹ qua không gian tìm kiếm. Điều này cho phép phương pháp được hướng dẫn qua một không gian phức tạp mà không làm ảnh hưởng đến thời gian cần thiết để thực hiện sự tiến hóa hoặc độ phức tạp của phương pháp tìm kiếm.



Hình 5: Minh họa đột biến một toán tử trong một khối sử dụng NAS-Bench-201

3. Khi kiến trúc mới được thêm vào quần thể, một cơ chế điều chỉnh (lựa chọn người sống sót) diễn ra, nơi kiến trúc cũ nhất được loại bỏ, do đó buộc phải khám phá không gian tìm kiếm bằng cách ưu tiên các kiến trúc trẻ hơn đại diện cho các cài đặt mới tiến hóa bởi kiến thức đã có trước đó.

Về bản chất, các giá trị  $P$  cao hơn đại diện cho một mức độ khám phá cao hơn của không gian tìm kiếm, trong khi các giá trị  $S$  cao hơn đại diện cho khai thác cao hơn bằng cách tăng xác suất các kiến trúc tốt nhất trong quần thể được chọn làm cha mẹ cho thế hệ tiếp theo

## Thuật toán Tiến hóa Hiệu chỉnh

Tiến hóa lão hóa

Tóm tắt thuật toán

## Thuật toán Định hướng Tiến hóa

Hạn chế của các phương pháp hiện tại

Thuật toán đề xuất

## Chi tiết triển khai

Phát biểu bài toán

Tóm tắt thuật toán

## Ước lượng proxy không tổn kém

Ma trận hiệp phương sai Jacobi

Công thức tính toán

## Ma trận hiệp phương sai Jacobi

Mục tiêu của việc đánh giá các kiến trúc ngay từ giai đoạn khởi tạo là cung cấp thông tin về không gian tìm kiếm mà không phải chịu chi phí cao của việc đào tạo thực tế, do đó cho phép hướng dẫn tìm kiếm đến cài đặt tối ưu.

- Để làm điều này, chúng ta sử dụng ước lượng proxy không tổn kém dựa trên Ma trận hiệp phương sai Jacobi.
- Điều này cho phép nhanh chóng đánh giá xem một kiến trúc có tốt không mà không cần đào tạo, do đó cho phép lựa chọn một kiến trúc được tạo ra để thêm vào quần thể với sự tự tin cao hơn rằng việc tìm kiếm đang được hướng dẫn một cách chính xác.

## Công thức tính toán

Giả sử có một ánh xạ tuyến tính,  $w_i = g(x_i)$ , mà đầu vào  $x_i \in \mathbb{R}^D$ , thông qua mạng  $g(x_i)$ , với  $x_i$  đại diện cho một hình ảnh thuộc một lô  $X$ , và  $D$  là kích thước đầu vào. Sau đó, Jacobi của ánh xạ tuyến tính có thể được tính toán bằng công thức:

$$J_i = \frac{\partial g(x_i)}{\partial x_i}$$

Điều này cho phép chúng ta đánh giá hành vi kiến trúc cho các hình ảnh khác nhau bằng cách tính toán  $J_i$  cho các điểm dữ liệu khác nhau  $g(x_i)$ , của một lô  $X$  duy nhất với  $i = 1, 2, \dots, N$ :

$$J = \begin{bmatrix} \frac{\partial g(x_1)}{\partial x_1} & \frac{\partial g(x_2)}{\partial x_2} & \dots & \frac{\partial g(x_N)}{\partial x_N} \end{bmatrix}^\top$$

$J$  chứa thông tin về đầu ra của kiến trúc đối với đầu vào cho nhiều hình ảnh. Chúng ta có thể chia đầu vào thành các lớp và đánh giá cách một kiến trúc mô hình hóa các chức năng phức tạp tại giai đoạn khởi tạo và tác động của nó đối với hình ảnh thuộc cùng một lớp.

Để làm điều này, chúng ta chia  $J$  thành nhiều tập hợp, nơi mỗi tập hợp  $M_k$ , chứa tất cả Jacobi thuộc về cùng một lớp  $k$ . Sau đó, chúng ta có thể tính toán ma trận tương quan cho mỗi lớp, kí hiệu  $\Sigma_{M_k}$ , sử dụng các tập hợp đã thu được  $M_k$  với  $k = 1, 2, \dots, K$ .



Các ma trận tương quan cá nhân cung cấp thông tin về cách một kiến trúc đơn lẻ xử lý hình ảnh cho mỗi lớp. Tuy nhiên, các ma trận tương quan khác nhau có thể cho kết quả với kích thước khác nhau, do số lượng hình ảnh mỗi lớp khác nhau. Để có thể so sánh các ma trận tương quan khác nhau, chúng được đánh giá riêng lẻ:

$$E_k = \begin{cases} \sum_{i=1}^N \sum_{j=1}^N \ln (|(\Sigma M_k)_{i,j}| + t), & \text{nếu } K \leq \tau \\ \frac{1}{\sqrt{\#\Sigma_{M_k}}} \sum_{i=1}^N \sum_{j=1}^N \ln (|(\Sigma M_k)_{i,j}| + t), & \text{ngược lại} \end{cases}$$

Trong đó  $t = 1 \times 10^{-5}$ ,  $K$  là số lớp trong lô  $X$  và  $\#(\cdot)$  là số phần tử.

Cuối cùng, một kiến trúc được tính điểm dựa trên các đánh giá cá nhân của các ma trận tương quan bằng cách sử dụng công thức sau:

$$z = \begin{cases} \sum_{w=1}^K |E_w|, & \text{nếu } K \leq \tau \\ \frac{1}{\#E} \sum_{i=1}^K \sum_{j=i+1}^K |E_i - E_j|, & \text{ngược lại} \end{cases}$$

Trong đó  $E$  chứa tất cả các điểm số của các ma trận tương quan. Điểm số cuối cùng phụ thuộc vào số lượng lớp có trong  $X$ , vì các bộ dữ liệu với số lượng lớp cao hơn thường có nhiều nhiều hơn, điều này được giảm bớt bằng cách thực hiện một sự khác biệt chuẩn hóa theo cặp (normalized pair-wise difference). Định nghĩa theo thực nghiệm  $\tau = 100$ , dựa trên không gian tìm kiếm và các bộ dữ liệu được sử dụng.

Sau đó, chúng ta có thể sử dụng  $z$  để xếp hạng các kiến trúc được tạo ra, cung cấp một cơ chế hiệu quả để phân biệt giữa kiến trúc xấu và tốt, từ đó cho phép hướng dẫn tìm kiếm về phía các cài đặt tốt hơn mà không làm tăng chi phí tìm kiếm.