Information Sciences 661 (2024) 120186

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Efficient multi-objective neural architecture search framework via policy gradient algorithm

Bo Lyu [a], Yin Yang [b], Yuting Cao [b], Pengcheng Wang [a], Jian Zhu [a,d], Jingfei Chang [a], Shiping Wen [c,*]

[a] Zhejiang Lab, Hangzhou, Zhejiang, China
[b] College of Science and Technology, Hamad Bin Khalifa University, Doha 5855, Qatar
[c] Australian AI Institute, Faculty of Engineering and Information Technology, University of Technology Sydney, NSW 2007, Australia
[d] University of Science and Technology of China, Hefei, Anhui, China

ARTICLE INFO

Keywords:
Neural architecture search
Reinforcement learning
Non-differentiable
Supernetwork

ABSTRACT

Differentiable architecture search plays a prominent role in Neural Architecture Search (NAS) and exhibits preferable efficiency than traditional heuristic NAS methods, including those based on evolutionary algorithms (EA) and reinforcement learning (RL). However, differentiable NAS methods encounter challenges when dealing with non-differentiable objectives like energy efficiency, resource constraints, and other non-differentiable metrics, especially under multi-objective search scenarios. While the multi-objective NAS research addresses these challenges, the individual training required for each candidate architecture demands significant computational resources. To bridge this gap, this work combines the efficiency of the differentiable NAS with metrics compatibility in multi-objective NAS. The architectures are discretely sampled by the architecture parameter α within the differentiable NAS framework, and α are directly optimised by the policy gradient algorithm. This approach eliminates the need for a sampling controller to be learned and enables the encompassment of non-differentiable metrics. We provide an efficient NAS framework that can be readily customized to address real-world multi-objective NAS (MNAS) scenarios, encompassing factors such as resource limitations and platform specialization. Notably, compared with other multi-objective NAS methods, our NAS framework effectively decreases the computational burden (accounting for just `1/6` of the NSGA-Net). This search framework is also compatible with the other efficiency and performance improvement strategies under the differentiable NAS framework.

1. Introduction

Neural Architecture Search (NAS) aims to mitigate the laborious process of manually tuning neural network architectures, thus contributing to the advancement of AutoML [1–3]. Neural architecture search has demonstrated significant efficacy not only in the architectures of networks such as CNNs, RNNs, and Transformers but has also exhibited favourable performance in networks like

* Corresponding author.
E-mail addresses: bo.lyu@zhejianglab.com (B. Lyu), yyang@hbku.edu.qa (Y. Yang), ycao@hbku.edu.qa (Y. Cao), wangpengcheng@zhejianglab.com (P. Wang), qijian.zhu@zhejianglab.com (J. Zhu), cjf_chang@zhejianglab.com (J. Chang), shiping.wen@uts.edu.au (S. Wen).

https://doi.org/10.1016/j.ins.2024.120186
Received 10 July 2023; Received in revised form 5 January 2024; Accepted 18 January 2024

Available online 26 January 2024

GNNs [4,5] and SNNs [6]. The Reinforcement Learning (RL) based NAS method [7] utilizes the controller to sequentially sample the candidate architectures and validate the corresponding performance, which serves as the reward of the RL framework. These approaches incur substantial computational overhead in terms of resource and time costs, making it challenging for ordinary research institutes and common commercial organizations (e.g., 20,000 GPU-days in [7] and 2,000 in [8]). To address this efficiency issue, subsequent studies aim to enhance the search procedure, such as the ENAS framework [9]. With the advent of differentiable NAS, as introduced by Liu et al. [10], which boasts enhanced efficiency, the focus of research has gradually shifted away from RL-based NAS approaches. Differentiable NAS research makes significant progress by continuously relaxing the search space, resulting in a differentiable loss w.r.t architecture parameters. This enables direct search through gradient-based optimization. The weight-sharing approach, facilitated by the unified differentiable supernetwork [11], eliminates the need for training candidate architectures individually from scratch, leading to substantial time and computational savings. Despite its high search efficiency, differentiable NAS research often overlooks non-differentiable objectives – those without differentiable proxy loss functions for direct optimization – such as energy, latency, and memory consumption. These objectives are crucial in resource-aware NAS and Multi-objective NAS (MNAS) scenarios and should be jointly considered.

Alongside the remarkable advancements in the differentiable NAS sub-field, there has been a parallel growth in multi-objective NAS methods (such as MnasNet [12], DPP-Net [13], MONAS [14], Pareto-NASH [15], [16], [17]). These methods focus on exploring neural architectures within discrete search spaces while considering multiple metrics, both differentiable and non-differentiable. These MNAS methods heavily rely on heuristic search strategies to enable the flexible customization of reward functions, resulting in significant computational overhead.

Our method builds upon the differentiable NAS framework, in which the candidate operations are progressively eliminated after each search stage, based on the architecture parameters. Simultaneously, we address the "depth gap" [18] by gradually increasing the model's depth. A key distinction is the detachment of the architecture parameter α from the differentiable framework. Instead, we sample architectures by the probability distribution of α, thereby eliminating the necessity of a sampling controller. Consequently, we can directly optimize α using a policy gradient algorithm. In sum, our framework zeroes in on a holistic exploration of the architectural space, seamlessly melding both non-differentiable and differentiable metrics. This integrative strategy harmonizes the strengths of differentiable neural architecture search (NAS) with the versatility of multi-objective NAS approaches, yielding a synergistic confluence of methodologies.

Our contributions may be summarized as follows:

(1) This approach facilitates the incorporation of non-differentiable objectives into a differentiable search framework, synergizing the efficiency of differentiable NAS with compatibility for the objectives of multi-objective NAS.
(2) Our search framework comprehensively tackles challenges associated with the "optimization gap" [19] [1], the "depth gap", and GPU memory consumption.
(3) This framework can be seamlessly integrated with scalability into the recently proposed search and evaluation strategies within the differentiable NAS framework.

Considering the objectives of "Parameters" and "Accuracy", our framework successfully generates high-performance and compact architectures that also exhibit remarkable transferability. We showcase the effectiveness of our approach by achieving promising and compact architectures on CIFAR-10 (1.09M/3.3%, 2.4M/2.95%, 9.57M/2.54%) and CIFAR-100 (2.46M/18.3%, 5.46M/16.73%, 12.88M/15.20%), considering two metrics: Parameters and Test Error. Furthermore, we showcase the transferability of the searched architectures on ImageNet, achieving a performance of 4.21M/24.8% and 5.23M/24.5%. In comparison to other multi-objective NAS methods, our approach remarkably diminishes the search cost, achieving completion within a mere 1.3 GPU-days, which is only 1/6 of the resources required by NSGA-Net.

2. Related work

Individual heuristic NAS. Traditional methods for architecture search typically utilize Evolutionary Algorithms (EA) as the search strategy [20,17,21–24]. These approaches involve mutating high-performing network architectures while discarding less promising ones. In recent years, significant success has been reported in RL-based NAS methods on datasets like CIFAR-10 and PTB [7,8]. These methods demand extensive computational resources, posing an efficiency challenge. To address this issue, ENAS [9] improves the search efficiency by introducing a weight-sharing strategy, building upon the previous works [7,8]. Some research endeavours, grounded in specific search spaces and tasks, have delved into architecture encoding approach and landscape analysis, showcasing that straightforward Local Search [25] methods and Bayesian Optimization [26] methods can achieve favourable outcomes in NAS tasks. Moreover, a recent study [27] has shown that utilizing GPT-4 as an optimizer, having it provide configuration recommendations for candidate architectures, and encoding the validation accuracy of these architectures as prompts to be fed back into GPT-4 enables an iterative interactive process for network architecture search. Overall, the heuristic NAS methods explore architectures within a discrete domain and generally rely on the individual architecture evaluation, which raises concerns about efficiency.

[1] Directly sampling a sub-architecture, where weights are inherited from the overarching super-network, may not yield a precise proxy performance evaluation on the validation set. This disparity stems from the weight-sharing training approach, which primarily focuses on optimizing the super-network as a whole, rather than honing in on the nuances and performance of individual sub-networks.

Differentiable NAS. By leveraging the concept of continuously relaxing the search space, the DARTS method [10] establishes the supernetwork that encompasses all candidate operations. This approach formulates the architecture search task as learning architecture parameters, resulting in significantly improved search efficiency. DARTS and its subsequent works face the challenge of high GPU-memory overhead [28] and the "depth gap" issue [18]. Additionally, differentiable NAS frameworks do not consider the non-differentiable metrics such as `FLOPs` (floating-point operations) and `Latency`, as these metrics differ from `Accuracy`, which is optimized using the differentiable cross-entropy function [19]. ProxylessNAS [29] takes advantage of the regularity of pre-defined chain-style structures. It approximately calculates the expectation value of the inference latency of each chain-style layer, aggregates the expectation values and formulates it as a regularization term, which allows the `Latency` metric to be differentiable. However, ProxylessNAS is not applicable for the non-chain-style backbones, as the `Latency` expectation cannot be calculated using linear transformation functions.

Multi-objective NAS and Platform-aware NAS. Multi-objective NAS methods, such as MnasNet [12] and MONAS [14], focus on searching for architectures while considering multiple evaluation metrics. These studies incorporate various model resource consumption factors, including `Parameters`, `FLOPs`, `Latency`, and `Energy`, to formulate reward-penalty coefficients for `Accuracy` [30,16]. Resorting to single-policy multi-objective reinforcement learning algorithms, these methods facilitate the identification of candidate architectures that strike a balance among these conflicting metrics. DPP-Net [13] introduces progressive search methods that incorporate device-aware characteristics, such as Quality of Service (QoS) and hardware resource requirements (e.g., memory size), which are crucial metrics for deploying deep neural networks. Moreover, it considers different target platforms, such as workstations, mobile devices, and embedded systems. Multi-objective evolutionary algorithms based on decomposition and Pareto non-dominated sorting exhibit outstanding performance in multi-objective optimization problems [31]. In LEMONADE [32], the evolutionary algorithm is introduced to tackle multi-objective NAS, resulting in promising Pareto-optimal performance on CIFAR-10. These methods for multi-objective NAS heavily depend on reinforcement learning or evolutionary algorithms, which suffer from drawbacks in terms of search efficiency. For instance, LEMONADE [32] demands a significant computational cost of `80 GPU-days`.

One-shot NAS. One-shot NAS research, exemplified by SMASH [11] and SPOS [33], involves constructing a unified supernetwork, from which the sub-architectures are heuristically sampled, and evaluated with the shared weights inherited from the supernetwork. Prior to sampling, the supernetwork is uniformly trained without bias towards specific sub-architectures. As discussed in [19], this approach gives rise to the "optimization gap", indicating that a well-optimized supernetwork may not necessarily yield well-performing sub-architectures.

3. Methodology

3.1. Preliminary

In the context of differentiable NAS, the mixed-edge operations $\bar{o}$ at a specific location (i, j) in the directed acyclic graph (DAG) cell is constructed by all primitive operations (o) in candidate operation set $\mathcal{O}$, following Eq. (1):

$$\bar{o}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o'\in\mathcal{O}}\exp\left(\alpha_{o'}^{(i,j)}\right)} o(x) \tag{1}$$

where the α denotes the architecture parameters and x denotes the input feature map. Through this construction methodology, the discrete architectural parameter space is effectively rendered continuous. The architecture search process is formulated to address the bi-level optimization problem, as Eq. (2):

$$\begin{aligned} &\min_{\alpha} \quad \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right) \\ &s.t. \quad w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha) \end{aligned} \tag{2}$$

where lower-level optimization targets the w variable, based on the training dataset (*train*), while the upper-level optimization targets the α variable, based on the validation dataset (*val*), respectively. To address this bi-level optimization problem, DARTS employs alternate optimization to approximately reach the solution.

3.2. Search strategy

Our framework is built upon the foundation of differentiable NAS, as depicted in Fig. 1. Distinctively, our method treats architectural parameters as a sampling policy rather than optimizing them differentially. These parameters are optimized using a policy gradient algorithm in discrete space, following a non-differentiable path, as highlighted in blue. On the other hand, the weight parameters w are optimized following the differentiable route (indicated in red). Our evaluation acceleration strategy is built upon the weight-sharing differentiable NAS framework, where the sampled sub-architectures inherit the weights of the supernetwork.

3.2.1. Training of supernetwork weights

The training of weight parameters w is accomplished by gradient descent, following Eq. (3):

$$w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha) \tag{3}$$

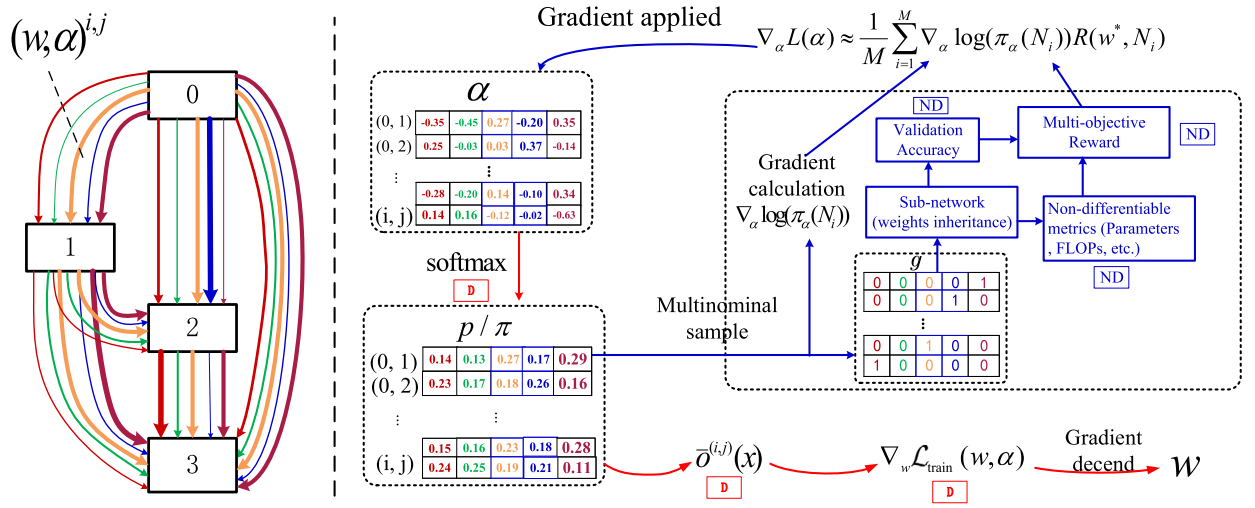

Fig. 1. The overall diagram of our multi-objective NAS framework. The directed acyclic graph (DAG) represents the `Normal/Reduction Cell`, which is characterized by the parameter variables (w,α). To handle non-differentiable optimization objectives, the architecture evaluation is decoupled from the differentiable framework. The red propagation route, denoted by "D", indicates that the propagation is differentiable w.r.t (w,α). Conversely, the "blue" propagation route, marked by "ND", signifies that the propagation is non-differentiable w.r.t α.

where w^* denotes the stage-wise optimization result of the current super-network's weight parameters, constrained by the fixed architecture parameter α.

3.2.2. Sampling sub-networks by architecture parameters

Regarding architecture sampling, for every mixed-edge index (i,j), the probability $p^{(i,j)}$ is computed according to the corresponding $\alpha^{(i,j)}$, as Eq. (4):

$$p^{(i,j)} = softmax(\alpha^{(i,j)}) \tag{4}$$

Subsequently, sampling by a multinomial distribution of probability vector $p^{(i,j)}$ to generate the vector $g^{(i,j)}$, as Eq. (5):

$$g^{(i,j)} \sim Multi(p^{(i,j)}, 1) \tag{5}$$

where the $Multi$ denotes the sampling function based on the multinomial distribution. The sub-networks $(\mathcal{N})$ are organized by sampled g, as Eq. (6):

$$\mathcal{N} = \mathcal{A}(g) \tag{6}$$

where $\mathcal{A}$ represents the differentiable supernetwork organized by stacking `Normal` and `Reduction` cells in the form of DAG (as depicted in Fig. 1), which are composed using mixed-edges as described by Eq. (1).

3.2.3. Optimize architecture parameters by policy gradient algorithm

The architecture search task is to optimize the architecture parameters α to achieve the optimal reward R, following Eq. (7):

$$\max_{\alpha} \mathcal{L}_{val}(w^*,\alpha) = \max_{\alpha} \mathbb{E}_{g\sim\alpha}[R_{val}(w^*,\mathcal{N}_g)] \tag{7}$$

where $\mathcal{L}$ denotes the expected reward from the sampled candidate architectures, which depends on the current architecture parameters α. Since the forward propagation path w.r.t α is non-differentiable, we adopt the policy gradient algorithm as an intuitive approach to estimate the gradient value.

In terms of the policy gradient algorithm, considering the policy function $\pi_\phi(a \mid s)$ is differentiable w.r.t the policy parameter ϕ. The objective function $\mathcal{L}$ is a function dependent on parameter ϕ, denoted as $\mathcal{L}(\phi)$. Let τ represent a trajectory sampled by the policy parameterized by ϕ, denoted as $p_\phi(\tau)$. The derivative of $\mathcal{L}(\phi)$ w.r.t the policy parameter ϕ can be achieved as Eq. (8):

$$\begin{aligned}\frac{\mathcal{L}(\phi)}{\partial\phi} &= \frac{\partial}{\partial\phi}\int p_\phi(\tau)G(\tau)\mathrm{d}\tau \\ &= \int\left(\frac{\partial}{\partial\phi}p_\phi(\tau)\right)G(\tau)\mathrm{d}\tau \\ &= \int p_\phi(\tau)\left(\frac{1}{p_\phi(\tau)}\frac{\partial}{\partial\phi}p_\phi(\tau)\right)G(\tau)\mathrm{d}\tau\end{aligned} \tag{8}$$

Table 1
Notation in RL and the corresponding notation in this work.

RL Notations	Notations in our NAS framework
State (s)	Supernetwork (with weights w) and the target task
Reward (r)	The evaluation metrics of the sampled architecture
Policy (π)	The distribution of architecture parameters α
Action (a)	Sampling the indices that represent the candidate architecture

$$= \int p_\phi(\tau) \left(\frac{\partial}{\partial \phi} \log p_\phi(\tau) \right) G(\tau) \mathrm{d}\tau$$
$$= \mathbb{E}_{\tau \sim p_\phi(\tau)} \left[\frac{\partial}{\partial \phi} \log p_\phi(\tau) G(\tau) \right]$$

where $\frac{\partial}{\partial \phi} \log p_\phi(\tau)$ represents the partial derivative of the function $\log p_\phi(\tau)$ w.r.t ϕ, and $G(\tau)$ denotes the cumulative discounted reward of trajectory τ. The optimization direction is to increase the sampling probability $p_\phi(\tau)$ for trajectories τ with higher cumulative discounted reward $G(\tau)$. From the perspective of reinforcement learning, the probability function $p_\phi(\tau)$ for the sampled trajectory τ w.r.t the policy parameter ϕ is defined as $p_\phi(\tau) = p\left(s_0\right) \prod_{t=0}^{T-1} \pi\left(a_t \mid s_t\right) p\left(s_{t+1} \mid s_t, a_t\right)$. In the context of NAS, this probability function is defined as the policy function, as Eq. (9):

$$p_\phi(\tau) := \pi_\alpha(\mathcal{N}) \tag{9}$$

The cumulative discounted reward is defined as:

$$G(\tau) := R_{val}(w^*(\mathcal{N})) \tag{10}$$

Hence, the gradient calculation for the optimization function w.r.t α can be expressed as Eq. (11):

$$\nabla_\alpha \mathcal{L}_{val}(\alpha) = \mathbb{E}_{\mathcal{N} \sim \pi_\alpha(\mathcal{N})}[R_{val}(w^*(\mathcal{N}), \mathcal{N}) \nabla_\alpha \log(\pi_\alpha(\mathcal{N}))] \tag{11}$$

We employ the classical REINFORCE [34] algorithm, which approximates the expectation using Monte-Carlo sampling, as Eq. (12):

$$\nabla_\alpha \mathcal{L}_{val}(\alpha) \approx \frac{1}{M} \sum_{m=1}^{M} [R_{val}(w^*(\mathcal{N}_m), \mathcal{N}_m)) \nabla_\alpha \log(\pi_\alpha(\mathcal{N}_m)] \tag{12}$$

where M is the sampling number for a single policy gradient iteration. We empirically adopt the REINFORCE algorithm with baseline [34], in which the baseline b is calculated as the moving average value of the sampled sub-architectures, by which means to reduce the variance and provide an unbiased estimation of the gradient, as Eq. (13):

$$\nabla_\alpha \mathcal{L}_{val}(\alpha) \approx \frac{1}{M} \sum_{m=1}^{M} [R_{val}(w^*(\mathcal{N}_m), \mathcal{N}_m) - b) \nabla_\alpha \log(\pi_\alpha(\mathcal{N}_m)] \tag{13}$$

The baseline function is utilized to reduce the variance to reach the unbiased estimation of the gradient, in which b is the moving average of the previous architecture rewards. As aforementioned, we utilize multinomial sampling to strike a balance of exploitation-and-exploration, preventing the early dominance of a single operation branch.

Comparatively, we establish the relationship between the notation employed in our NAS framework and the concepts in reinforcement learning, as presented in Table 1.

3.2.4. Architecture reward design

Undoubtedly, when evaluating performance, the widely utilized metric, `Accuracy`, is frequently employed as a direct reward. Nevertheless, in the realm of multi-objective scenarios, it is crucial to meticulously devise the reward function according to real-world demands. For example, in situations with constrained resources, there exists a trade-off between the device-agnostic and device-related metrics. Inspired by Mnasnet [12], to shape the reward, we consider `Accuracy` and `Parameters` as the evaluation metrics. We have the reward linear w.r.t `Accuracy` while maintaining a non-linear relationship w.r.t `Parameters`. This non-linear relationship is achieved by incorporating a reward-penalty factor into a scalarization function, as described in Eq. (14):

$$R = Acc \cdot (\frac{Params}{P})^\beta \tag{14}$$

where P is the reward-penalty reference and β is the reward-penalty coefficient, both of which rely on empirically experimental tuning. Fig. 2 illustrates the reward function surface, showcasing the cross-sectional curves for specific metrics and their corresponding projection curves. The integration of non-differentiable metrics into the search process can be seamlessly achieved through the scalarization function f, which converts the reward vector into a scalar value, namely single-policy Multi-Objective Reinforcement Learning (MORL) [35]. In a multi-objective optimization scenario, the scalarization function can be customized based on specific requirements. For instance, considering objectives such as `Accuracy`, `Parameters`, and `Latency`, Eq. (15) demonstrates one possible formulation of the scalarization function:

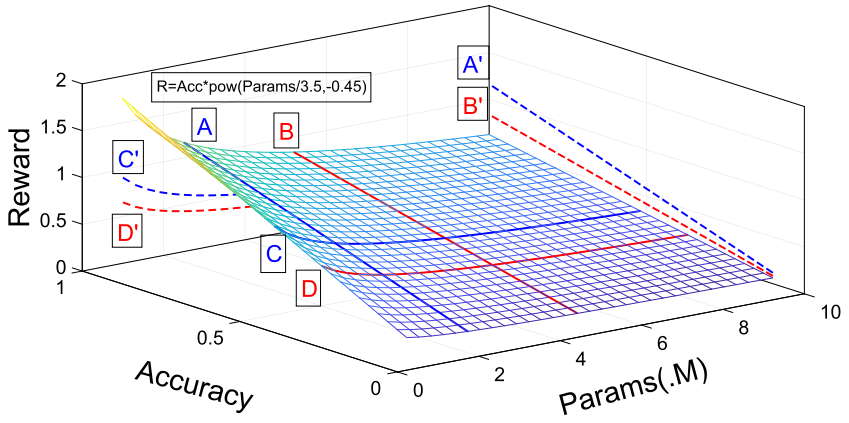

Fig. 2. The reward function surface described by Eq. (14). The spatial curves and their corresponding projection lines under specific `Parameters` and `Accuracy` (maintaining consistency) are displayed. The dashed lines ($A'/B'/C'/D'$) indicate the projection of the spatial curves, which following the equations "A": $0.52 * (Params/3.5)^{-0.45}$, "$B$": $0.37 * (Params/3.5)^{-0.45}$, "$C$": $Acc * (2.5/3.5)^{-0.45}$, "D": $Acc * (Params/3.5)^{-0.45}$, respectively.

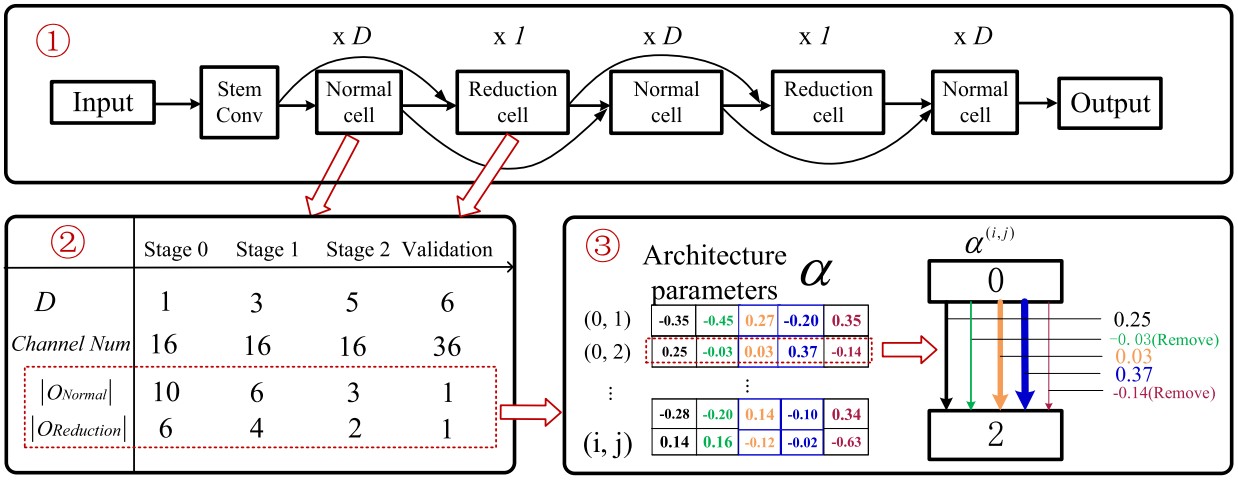

Fig. 3. The schematic diagram of the progressive search framework. ①: The schematic diagram of network macro-architecture, which is constructed by stacked `Normal` cell and `Reduction` cell, The stacked `Normal` cell and `Reduction` cell are parameterized by D (stacked number), Channel number, $\mathcal{O}_{Normal}$ (operation set number of `Normal` cell), $\mathcal{O}_{Reduction}$ (operation set number of `Reduction` cell), as listed in diagram ②; ②: The configuration parameters evolve during the different search stages and the scale of the operation set. The evolving rule conforms to diagram ③; ③: After each search stage, candidate operations corresponding to the edges with lagging architecture parameters at position (i, j) will be removed from the candidate operation set $\mathcal{O}_{i,j}$ that for constructing the super network in the subsequent stage. In the details shown of α vector, the "Remove" indicates that the edges are dominated by others, and are to be removed.

$$R = Acc \cdot [a \cdot (\frac{Params}{P})^{\beta} + c \cdot (\frac{Latency}{L})^{\gamma}] \tag{15}$$

where the reference for `Parameters` and `Latency` is denoted by P and L respectively. The reward-penalty coefficients, denoted as β and γ, are accompanied by the weight coefficients a and c, satisfying the constraint $a + c = 1$.

3.2.5. *Progressively deriving the architecture*

In conventional differentiable NAS procedures, as the search progresses, the persistently outdated mixed-edge connections within each cell (`Normal` cell/`Reduction` cell) not only incur ongoing memory overhead but also introduce the "optimization gap" issue. Following P-DARTS, we adopt the strategy of progressively reducing the searched architecture through the combination of "search space approximation and regularization", as shown in Fig. 3. The reduction of the candidate operation set is achieved by Eq. (16):

$$\mathcal{O}_{k+1}^{(i,j)} = \mathcal{O}_{k}^{(i,j)} \ominus \mathcal{O}_{k}^{(i,j)}(argmin(p^{(i,j)}, n_k))) \tag{16}$$

where n_k represents the reduced number of candidate operation sets, k denotes the stage with $k \geq 0$, and the operator $\ominus$ represents the scale reduction. Conversely, the total number of network layers is increased using Eq. (17):

$$L_{k+1} = L_k + l_{k+1} \tag{17}$$

Our search framework is outlined in Algorithm 1. The algorithm begins with initializing the operations $\bar{o}_k^{(i,j)}$ for each edge (i, j)

Table 2
Designs of our search frameworks and the corresponding motivations.

Our framework		Design	Motivation
Evaluation strategy		Weight-sharing differentiable supernetwork	Weight inheritance for evaluation acceleration, without individual training from scratch
Search strategy	Policy	Architecture parameters α	Directly optimize policy (without controller, relatively low dimensional)
	Sampling	**Multinominal sampling by probabilistic distribution α**	Balance exploitation and exploration
	Optimization	**Optimizing α by policy-gradient algorithm**	Customize reward function flexibly, compatible with non-differentiable objectives and multi-objectives
Search space		Progressively shrinking the operation set	Progressively shrinking the search space, GPU-memory friendly
		Progressively increasing the depth	Targets "`depth gap`"

Algorithm 1: Our search framework.

Input: Candidate operation set $\mathcal{O}$, Cell (DAG) stage number, search stage number S, Layer number of the supernetwork of each stage $D[0], D[1], \cdots, D[S-1]$, Epoch number of each search stage $E[0], E[1], \cdots, E[S-1]$, Training interval of policy gradient $INTERVAL$, Pre-training epoch number e_k;
Output: The optimal architecture;

```
for k = 0 → stage_num − 1 do
    Init õ_k^(i,j) ∈ O_k^(i,j) parameterized by α_k^(i,j) for edge (i, j);
    Init the supernetwork with õ_k^(i,j) and L_k;
    for e = 0 → E[k] − 1 do
        for s = 0 → BatchNum do
            if e > e_k and step % INTERVAL == 0 then
                Update architecture parameters α by Eq. (11)
            end
            Update weights parameters w by Eq. (3)
        end
    end
    Shrinking each O_k^(i,j) by Eq. (16);
    Increase layer number L_k by Eq. (17);
end
Deriving the final architecture.
```

based on the candidate operation set $\mathcal{O}$. These operations are parameterized by $\alpha_k^{(i,j)}$. Simultaneously, a supernetwork is created using these candidate operations, with its layer count set to L_k. The supernetwork undergoes pre-training over a predefined number of epochs $E[k]$, with only the weight parameters w updated based on a defined loss function, as Eq. (3). After the pre-training, the policy gradient optimization (Eq. (11)) is additionally employed to strategically update architecture parameters α. After each search stage, the algorithm applies Eq. (16) to reduce the candidate operation set $\mathcal{O}_k^{i,j}$, while the layer count L_k is augmented as Eq. (17). After the final search stage, the algorithm derives the optimal architecture based on the convergent architecture parameters α.

For clarity, we provide a concise summary of the design choices and their corresponding motivations in Table 2, focusing on three key aspects: evaluation strategy, search strategy, and search space. Specifically, regarding the search strategy, we further analyze it from three distinct perspectives: sampling policy, sampling operation, and optimization policy.

4. Comparison with related works

The underlying assumption for individual heuristic NAS methods, whether based on reinforcement learning (RL) or evolutionary algorithms (EA), is the "performance ranking hypothesis" [38]. This hypothesis suggests that the relative performance of candidate architectures can be effectively determined on proxy-tasks for a reduced number of training epochs. However, for the individual heuristic NAS methods, the drawback lies in the individual training architectures from scratch that cause a significant computational burden. While DARTS successfully addresses the time-consuming search problem, it introduces new challenges regarding GPU-memory cost and the issue of the "depth gap". To mitigate these issues, P-DARTS adopts the progressive search strategy, nevertheless, several concerns still need to be addressed:

(1) It lacks end-to-end optimization, exemplified by the following designs:

Table 3
Relationship with other search strategies and evaluation methods in community.

Method	Evaluation Strategy	Search Strategy
Individual search [7,12]	Individual training of sampled architecture (proxy task)	Heuristic search: (1) RL-based: sampling with controller, of which the parameters as the policy to be optimized (2) Population-based, e.g., EA
One-shot NAS [9,11]	Weight-sharing supernetwork	Heuristic search: (1) RL-based: sampling with controller, of which the parameters as the policy to be optimized (2) Population-based, e.g., EA.
Differentiable NAS [10,36,37]	Weight-sharing differentiable supernetwork	Differentiable optimization of α by gradient descent
Ours	**Weight-sharing differentiable supernetwork**	**Sampling architectures by probability distribution of α, which is the policy to be optimized**

Table 4
Characteristics comparison with the state-of-the-art MNAS methods.

Method	Differentiable metrics	Non-differentiable metrics	Low GPU-memory	Low time cost
NASNet	✔	✗	✔	✗
MnasNet	✔	✔	✗	✗
DARTS	✔	✗	✗	✔
FBNet	✔	✗	✔	✗
ProxylessNAS	✔	✗	✗	✔
P-DARTS	✔	✗	✔	✔
Ours	✔	✔	✔	✔

(a) The searched architecture undergoes processing with one initial channel setting, but its evaluation is conducted with a different setting.
(b) The restriction on "`skip-connect`" operation is applied in the final search stage.
(c) When deriving the final architecture, the indegree of a node is strictly limited to 2. It inevitably results in the "architecture inconsistency gap".

(2) The P-DARTS framework is incapable of non-differentiable evaluation metrics such as `Parameters` and `FLOPs`.

As stated, the method in ProxylessNAS [29] is restricted by a predefined backbone structure, and is not applicable to non-chain-style search spaces like DARTS-like methods, NAS-RL, and NASNet [19]. The Once-for-all (OFA) method [39] introduces the concept of "progressive shrinking training" for the MobileNetV3-based supernetwork. This training approach enables both large-scale and small-scale sub-networks sampled from the supernetwork to achieve satisfactory performance while meeting specific resource constraints. As a result, the training of the supernetwork (without relaxation) and the search process are conducted independently in different stages. Notably, the "progressive shrinking training" in OFA is computationally intensive, requiring substantial time and resource cost, such as `1,200 GPU hours across 32 V100 GPUs`. Our approach directly treats the architecture parameter α as the heuristic optimization variable under the relaxed formulation of the search space. Thus, compared with EA-based NAS method, this eliminates the need for the complex design of architecture encoding schemes (e.g., connection encoding, operation encoding) and crossover/mutation operators. Additionally, our design allows for the integration of strategies proposed in recent differentiable NAS works, such as "search space approximation" [18] to address the "depth gap" issue or the "partially-connected channel" [28] strategy for efficient GPU-memory utilization during the search process.

The comparison and interplay between this study, individual search, one-shot search, and differentiable NAS have been systematically summarized in Table 3. This summary emphasizes the evaluation metrics and search strategies employed, deliberately excluding the dimension of search space for focused analysis. The evaluation strategy of this work is rooted in differentiable NAS, but it incorporates a unique design in its search strategy. Further, we compare our method with other classic NAS techniques from the perspectives of search objectives and strategy properties. A detailed comparison can be found in Table 4. This comparative analysis provides valuable insights into the similarities and distinctions among these methods, shedding light on their respective strengths and limitations.

Table 5
Hyper-parameters setting of architecture search.

Initial channels	16	Layers	5,12,20 †
Pretrain epochs	10	Batch	128
Stage epochs	25	Reference of Parameters	4.4,3.2,1.8 †
LR decay	cosine	Penalty coefficient (`C-10`)	-0.60,-0.60,-0.60 †
Reduced operation num	4,3,2†	Penalty coefficient(`C-100`)	-0.75,-0.85,-0.85 †
Optimizer (α)	Adam	Optimizer (w)	SGD

† The configurations of the 3 stages of the progressive shrinking search.

5. Experiments

5.1. Datasets

We perform experiments on two widely used image classification datasets, namely CIFAR-10 and CIFAR-100. Each dataset comprises `50,000` images for training and `10,000` for testing, all of which are standardized to a spatial resolution of 32×32. Throughout the search, we partition the training set into two equally-sized subsets: one for the lower-level optimization and the other for the upper-level optimization. Upon completing the search process, apart from evaluating the discovered architectures, we additionally assess their performance on ImageNet (ILSVRC2012). Following previous studies [10,18], we adhere to the mobile setting of ImageNet, where the input images have dimensions of 224×224.

5.2. Search space

Regarding the predetermined backbone and search space, we adhere to the DARTS/P-DARTS methodology. When it comes to the operations set, to better demonstrate the scalability of the search strategy and the capacity for model compression, we introduce subtle modifications, outlined as follows:

(1) `Normal Cell`:

a) `none` b) `skip_connect` c) `sep_conv_3x3`
d) `sep_conv_5x5` e) `sep_conv_7x7` f) `dil_conv_3x3`
g) `dil_conv_5x5` h) `conv_1x1` i) `conv_3x3`
j) `conv_3x1_1x3`

(2) `Reduction Cell`:

a) `none` b) `skip_connect` c) `max_pool_3x3`
d) `avg_pool_3x3` e) `max_pool_5x5` f) `max_pool_7x7`

In our search space setting, the initial operation set scale of `Normal` cell ($|\mathcal{O}_N|$) is 10, and the initial operation set scale of `Reduction` cell ($|\mathcal{O}_R|$) is 6. The search space scale is $14^{10} \times 14^6 = 2 \times 10^{18}$, where 14 represents the number of edges in the DAG. Specifically, with 4 intermediate nodes, the edge number is calculated as $2 + 3 + 4 + 5 = 14$.

5.3. Architecture search

5.3.1. Experimental setting

The search experiment is conducted with the `PyTorch 1.4` framework on two `NVIDIA 1080Ti` GPUs. For clarity, we only focus on the metrics of `Accuracy` and `Parameters`. The reward function is calculated as Eq. (14). The search hyperparameters are outlined in Table 5.

5.3.2. Architecture search results

Within a specific search configuration, we conduct five iterations of experiments using different random seeds. Subsequently, we select the optimal architecture for final evaluation based on the validation performance achieved through training from scratch over `100` epochs. The visualization of our searched model (`CIFAR-10-S`, employing a model compression configuration) is presented in Fig. 4, with the `Normal` cell depicted in Fig. 4a, and the `Reduction` cell shown in Fig. 4b. The nodes in the visualization correspond to the feature maps (FMs), while the edges represent the connections established through the searched operation.

Benefit from the end-to-end search, the searched `Normal/Reduction` cell architecture manifests in a more diverse manner. There is no restriction on the indegree of a node, and the inclusion of "`skip_connect`" operations is not enforced. Edges are only removed when the "`none`" operation is selected. This approach provides several benefits. From Fig. 4, we make several other

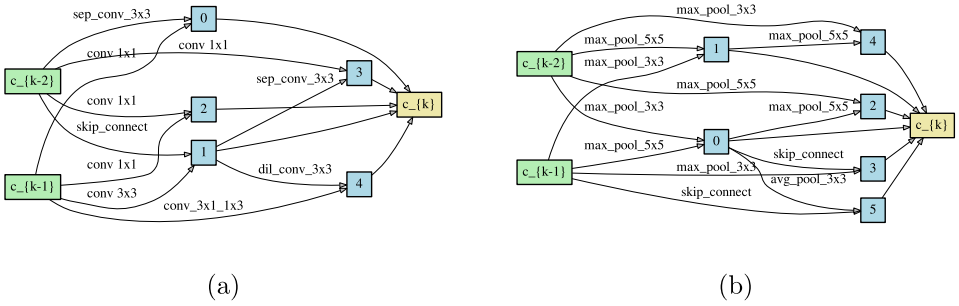

Fig. 4. The schematic illustration of the searched architecture (`CIFAR-10-S`). The `Parameters` is `1.09M` with `16` initial channels. (a) `Normal` cell; (b) `Reduction` cell.

Table 6
Hyper-parameters setting of architecture evaluation on CIFAR-10 and CIFAR-100.

Layers	20	Initial channels	24
Epochs	600	Batch size	128
Optimizer	SGD	LR	0.025
LR scheduler	cosine	Weight decay	3e-4

Table 7
Hyper-parameters setting of architecture evaluation on ImageNet.

Layers	14	Initial channels	36
Weight decay	3e-5	Batch	1024
Optimizer	SGD	Epochs	350
LR	0.1	LR scheduler	linear

observations: First, the deep connections are preserved both in `Normal` cell and `Reduction` cell, e.g., from "`c_{k-1}`" and "`c_{k-2}`" FMs to `3/4/5` FMs. Second, The "`conv_1x1`" operation is frequently selected, particularly during the early propagation stage of the cell, e.g., applied on "`c_{k-1}`" and "`c_{k-2}`". From our view, this primarily stems from the inherent compactness of `conv_1x1` within the `Parameters`, coupled with its exceptional ability to fuse the knowledge among the channels in the Feature Maps (FMs). Third, the "`max_pool_5x5`" is frequently selected in the `Reduction` cell, for it is with a proper receptive field and is also parameter-free.

5.4. Architecture evaluation

To determine the architecture for the final evaluation on CIFAR-10 dataset, we conduct the search framework across 5 runs. Subsequently, we train the resulting five convergent architectures from scratch for a brief period of 100 epochs on the training dataset to select the best one based on performance on the validation dataset. The best architecture is then trained from scratch five times, each for 600 epochs, and we report the test error, expressed as both the mean and standard deviation, on the test dataset.

5.4.1. Architecture evaluation on CIFAR-10 and CIFAR-100

For architecture evaluation, we adhere to the experimental training setting of P-DARTS, as shown in Table 6. Table 8 compares our approach with several state-of-the-art methods, encompassing both manually designed architectures and remarkable NAS architectures. For the sake of fairness, we conduct the search experiment on the identical search space, aligned with P-DARTS, which demonstrates that our models achieve comparable performance to P-DARTS. As an additional comparison, we also examined the scalability of our method in the search spaces with an enhanced candidate operations set (including `conv_3x3` and `conv_3x1_1x3`). The search result includes an exceptionally compact architecture (`S`) with a test error of $3.3 \pm 0.02\%$ and `1.09M Parameters`, a moderate scale architecture (`M`) with a test error of $2.7 \pm 0.05\%$ and `3.2M Parameters`, and a large scale architecture (`L`) with a test error of $2.54 \pm 0.03\%$ and `9.57M Parameters`. Further, our search experiment conducted on CIFAR-100 yields promising results as well. We achieve a test error of $18.3 \pm 0.03\%$ with `2.46M Parameters` for architecture `S`, $16.73 \pm 0.06\%$ test error with `5.46M Parameters` for architecture `M`, and $15.2 \pm 0.07\%$ test error with `12.88M Parameters` for architecture `L`. Our search framework achieves comparable performance to P-DARTS while offering a broader spectrum of model scales. This highlights the superior versatility of our approach, particularly in the context of model compression functionality.

5.4.2. Architecture evaluation on ImageNet

To assess the transferability of the searched architecture, we directly transfer the searched `Normal` cell and `Reduction` cell to the evaluation on ImageNet. Consistent with P-DARTS, the channel number is `36` and the layer number is `14`. The architecture is

Table 8
Comparison of the evaluation results on CIFAR-10 and CIFAR-100.

Architecture	Test Err. (%)		Params	Search Cost	Search
	C-10	C-100	(.M)	(GPU-days)	Method
NASNet-A [8]	2.65	-	3.3	1800	RL
AmoebaNet-A [20]	3.34	-	3.2	3150	Evolution
PNAS [40]	3.41	-	3.2	225	SMBO
RelativeNAS [41]	2.34	15.86	3.93	0.4	Evolution
DARTS (first order) [10]	3	17.76	3.3	1.5	Gradient
SNAS + mild constraint [42]	2.98	-	2.9	1.5	Gradient
ProxylessNAS [29]	2.08	-	5.7	4	Gradient
P-DARTS† (C-10) [18]	2.5	16.55	3.4	0.3	Gradient
P-DARTS† (C-100) [18]	2.62	15.92	3.6	0.3	Gradient
P-DARTS† (C-10-Large) [18]	2.25	15.27	10.5	0.3	Gradient
P-DARTS† (C-100-Large) [18]	2.43	14.64	11	0.3	Gradient
Ours† (`C-10`)	2.47 ± 0.03	-	2.04	1.3⋆	RL
Ours† (`C-100`)	2.58 ± 0.05	-	3.43	1.3⋆	RL
Ours† (`C-10-Large`)	-	15.3 ± 0.04	9.57	1.3⋆	RL
Ours† (`C-100-Large`)	-	14.6 ± 0.03	10.5	1.3⋆	RL
Ours‡ (`CIFAR-10-S`)	3.3 ± 0.02	-	1.09	1.3⋆	RL
Ours‡ (`CIFAR-10-M`)	2.70 ± 0.05	-	3.2	1.3⋆	RL
Ours‡ (`CIFAR-10-L`)	2.54 ± 0.03	-	9.57	1.3⋆	RL
Ours‡ (`CIFAR-100-S`)	-	18.3 ± 0.03	2.46	1.3⋆	RL
Ours‡ (`CIFAR-100-M`)	-	16.73 ± 0.06	5.46	1.3⋆	RL
Ours‡ (`CIFAR-100-L`)	-	15.20 ± 0.07	12.88	1.3⋆	RL

⋆: The experiments were conducted on two NVIDIA 1080Ti GPUs, each with 11 GB memory, over a period of 0.65 days.
†: The search is conducted on the original DARTS search space.
‡: The search is conducted on modified DARTS search space for the evaluation of multi-objective search.

Table 9
Comparison of the evaluation results on ImageNet.

Architecture	Test Err.(%)		Params	Search Cost	Search
	Top-1	Top-5	(.M)	(GPU-days)	Method
NASNet-A [8]	26.0	8.4	5.3	1800	RL
AmoebaNet-A [20]	25.5	8.0	5.1	3150	Evolution
PNAS [40]	25.8	8.1	5.1	225	SMBO
MnasNet-A2 [12]	24.6	7.3	4.8	/†	RL
DARTS (second order) [10]	26.7	8.7	4.7	4.0	Gradient
SNAS (mild constraint) [42]	27.3	9.2	4.3	1.5	Gradient
ProxylessNAS (GPU) [29]	24.9	7.5	7.1	8.3	Gradient
RelativeNAS [41]	24.88	7.7	5.05	0.4	Evolution
P-DARTS (CIFAR-10) [18]	24.4	7.4	4.9	0.3	Gradient
P-DARTS (CIFAR-100) [18]	24.7	7.5	5.1	0.3	Gradient
`CIFAR-10-M`	24.8 ± 0.07	7.5 ± 0.08	4.21	1.3	RL
`CIFAR-100-M`	24.5 ± 0.12	7.6 ± 0.18	5.23	1.3	RL

† MnasNet costs `4.5 days on 64 TPUv2`.

trained from scratch five times, each for 600 epochs, and follows the training hyperparameters settings presented in Table 7. We report the mean and standard deviation of test error on the test dataset, as shown in Table 9. Specifically, the models achieve a Top-1 test error of $24.8 \pm 0.07\%$ with `4.21M Parameters` (`CIFAR-10-M`) and Top-1 test error of $24.5 \pm 0.12\%$ with `5.23M Parameters` (`CIFAR-100-M`).

5.5. Search cost comparison

In terms of search time cost, measured in GPU days, our method exhibits slightly higher time consumption compared to P-DARTS (`1.3 GPU days` vs. `0.3 GPU days`). This disparity arises from the fact that we separate the optimization process of the architecture parameters α from the weight parameters w, unlike P-DARTS, where both are unified within the gradient-based differentiable framework. The additional time cost is incurred due to the sampling and independent evaluation of sub-architectures on the validation set (which is the most time-consuming aspect while relying on the weight-sharing strategy), and the gradient calculation process based on the policy gradient algorithm. This additional overhead is inevitable and justifiable, as it enables the search framework to accommodate non-differentiable objectives and achieve multi-objective NAS. From another point, benefits from the evaluation acceleration of differentiable supernetwork, our framework is significantly more efficient than multi-objective NAS methods based

Table 10
The search cost comparison with other multi-objective NAS methods.

Methods	Search strategy	Search dataset	Search cost (GPU days)	Test Err. (%,Top-1) C10	Test Err. (%,Top-1) ImageNet	Params(.M)
MnasNet [12]	RL	ImageNet	288 (TPUv2)	/	24.8	3.9
LEMONADE [32]	EA	C10	80	2.58/3.05	/	13.1/4.7
DPP-Net [13]	SMBO	C10	8 (1080Ti)	4.36	/	11.4
PARETO-NASH [15]	EA	C10	56	3.5	/	4
NSGA-Net [17]	EA	C10	8 (1080Ti)	3.85	/	3.3
RNSGA-Net [43]	EA	C10	3.11	3.89	/	3.00
ADF-L [44]	RL	C10	7.25 (P100)	2.76	/	3.94
MOEA-PS [45]	EA	C10	2.6	2.77	/	4.34
Ours	RL	C10	**1.3 (1080Ti)**	**2.70**†	24.8‡	**3.2**†/**4.21**‡

†: The `CIFAR-10-M` architecture, layer number is `20`, channel number is `24`.
‡: Transferred architecture from CIFAR-10 to ImageNet, layer number is `14`, and channel number is `36`.

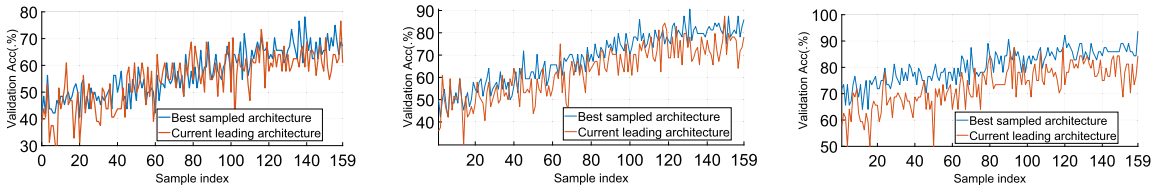

Fig. 5. The optimization effects on validation performance (Acc) throughout the search process. Notably, both the historical best accuracy, denoted as $max(Acc(\mathcal{A}(g)))$, and the current converged architecture's performance denoted as $Acc(\mathcal{A}(argmax(\alpha)))$, exhibit simultaneous improvement. Three stages of the search process: (a) `Stage 1`; (b) `Stage 2`; (c) `Stage 3`.

on individual heuristic search (RL-based, EA-based). For clarity, we present a summary of the comparisons between our search cost and the SOTA multi-objective NAS methods/framework in Table 10. Our method necessitates only `0.65 days` of computation on `2 NVIDIA 1080Ti` GPUs, each equipped with only 11GB memory. When compared to previous promising multi-objective NAS methods, our approach demonstrates a substantial improvement in search resource cost, reducing it to `1.3 GPU-days`, which is just `1/6` of the cost incurred by NSGA-Net [17]. It is crucial to emphasize that the search cost, encompassing both search time and memory cost, is influenced not only by the search strategy but also by factors such as the target dataset, proxy-task setting, and low-fidelity evaluation. Hence, we also include the search settings and metrics in Table 10, including the search dataset, performance, and model scale.

6. Ablation and diagnostic experiments

6.1. Effectiveness of search strategy

The effectiveness of the search algorithm is first validated in terms of a differentiable metric. We utilize validation `Accuracy` as the sole reward for reinforcement learning. Throughout the search process, we monitor two indicators at each stage. The first indicator, denoted as $max(Acc(\mathcal{A}(g)))$, captures the maximum accuracy achieved by $\mathcal{A}(g)$ across the current search process, where $g^{(i,j)} \sim Multi(\pi^{(i,j)}, 1)$. The second indicator represented as $Acc(\mathcal{A}(argmax(\alpha)))$, measures the accuracy of $\mathcal{A}(argmax(\alpha))$, which represents the current converged architecture's performance according to α. Fig. 5 exhibits a steady improvement in both the two indicators. It demonstrates the algorithm's effectiveness in successfully discovering architectures with outstanding precision performance. We additionally delve into the search strategy's capacity for recognizing non-differentiable metrics, specifically the `Parameters`, by uniquely utilizing it as the exclusive reward criterion in the search process. Fig. 6 and Fig. 7 demonstrate that the search algorithm tends to favour the "`conv_3x3`" operation to reach the higher `Accuracy`, with the final index converging towards the "`conv_3x3`" operation.

We conduct further analysis on the influence of integrating performance and computational cost metrics into the search process. Specifically, we compare the results of two penalty coefficients (`0` and `-0.25`) while monitoring the `Parameters` and `Accuracy`. As Fig. 8 shows, when using a penalty coefficient of `0`, we observe a simultaneous increase in both `Parameters` and `Accuracy`, suggesting that the enhancement in `Accuracy` is achieved at the expense of higher `Parameters`. When applying a penalty coefficient of `-0.25`, the `Parameters` remain relatively stable while the `Accuracy` continues to improve.

6.2. Comparative analysis with P-DARTS method

We first present a comparative analysis between our method and P-DARTS for the original search space. These comparative experiments are repeated 10 times, with the resulting curve delineating the mean values and the shaded region denoting the Interquartile

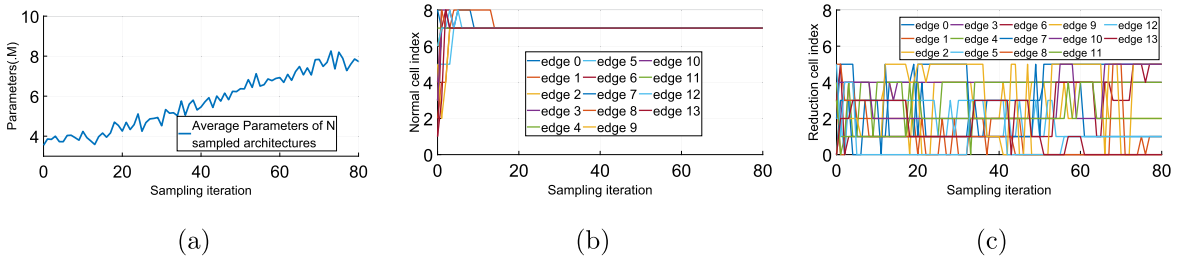

Fig. 6. The architecture search procedure of the maximization of the `Parameters`. The average `Parameters` keeps increasing. The `Normal Cell`, which determines the `Parameters`, converging to the "`conv_3x3`" operation (indexed as `7`). (a) The variation of the average `Parameters` across `N` sampled architectures; (b) The records of the iteration indices of the `Normal Cell` operations; (c) The records of the iteration indices of the `Reduction Cell` operations.

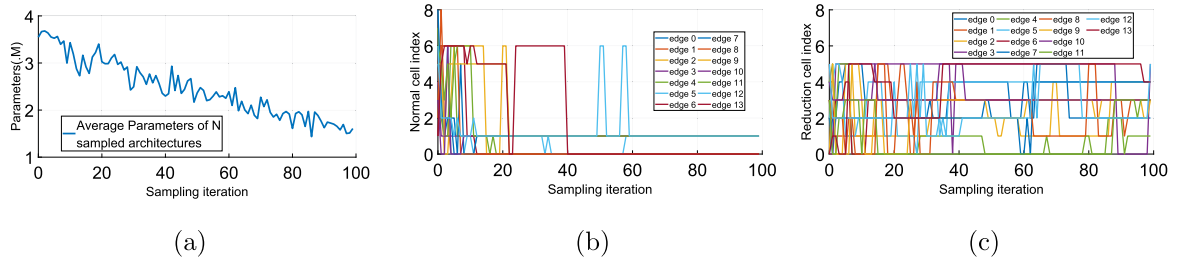

Fig. 7. The architecture search process aims to minimize the `Parameters`. The average `Parameters` progressively decreases, as the `Normal Cell` architecture, responsible for determining the `Parameters`, converges towards the "`skip_connect`" operation (indexed as `1`) and the "`None`" operation (indexed as `0`). (a) The variation of the average `Parameters` across `N` sampled architectures; (b) The iteration indices of the `Normal Cell` operations; (c) The iteration indices of the `Reduction Cell` operations.

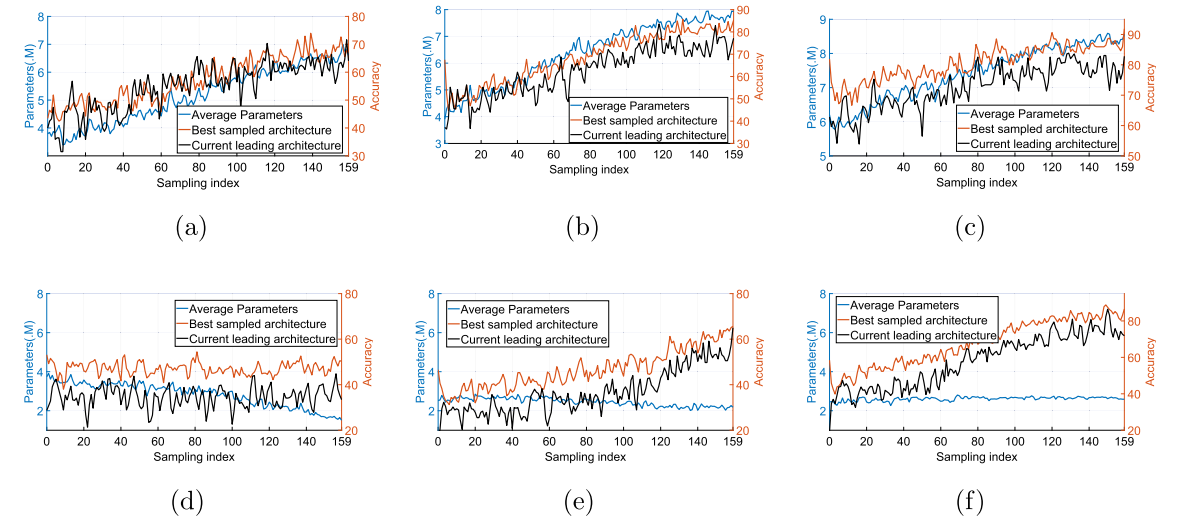

Fig. 8. The reward is formulated with `Parameters` and `Accuracy` metrics by Eq. (14), showing the comparative effects under different reward-penalty coefficients. The left y-axis of each sub-figure represents the `Parameters`, while the right y-axis represents the Top-1 validation accuracy. Three stages of the search process (a), (b), (c): with a penalty coefficient of 0, the improvement of `Accuracy` is achieved by trading off `Parameters`; Three stages of the search process (d), (e), (f): with a penalty coefficient of `-0.25`, the `Parameters` remains constant while the `Accuracy` continues to improve.

Range (IQR), providing a clear visual representation of variability, as Fig. 9 shows. Our experimental framework is conducted with a penalty coefficient of −0.3 for `Parameters`. In contrast, the P-DARTS experiment is conducted without any restrictions on the "`skip_connect`", allowing for a comprehensive assessment of its model scale. Observations reveal a marginal increment in the "`Parameters`" metric of the P-DARTS method during the first and second stages, followed by a slight decrement in the third stage. This phenomenon is attributed to the frequent retention of the "`skip_connect`" operation, which is parameter-free, within the progressive search paradigm. In contrast, our methodology exhibits notable model compression capabilities in both the first and second stages. Further, we conduct the comparative experiment on the modified DARTS search space, which includes the "`conv_3x1_1x3`" and "`conv_3x3`" operations. As Fig. 10 shows, in terms of P-DARTS, there exists a pronounced and rapid increase in the `Parameters` of the leading models. Consequently, when the channel number is set to `36`, the resulting architecture achieves a `Parameters`

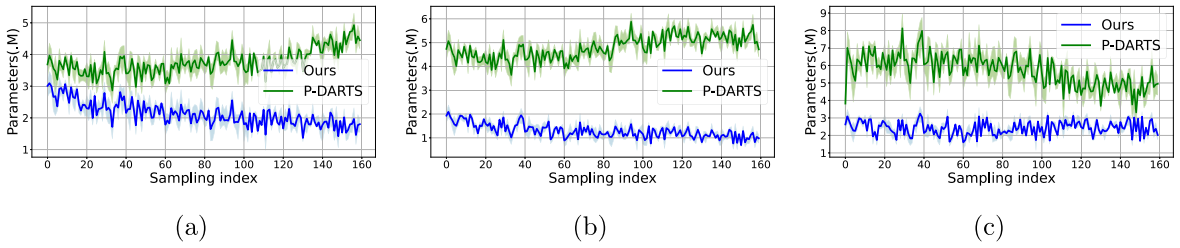

Fig. 9. The comparative experiments of our method versus P-DARTS in the original DARTS search space. The comparative experiments are conducted 10 times, with the curve representing the mean values and the shaded area indicating the Interquartile Range (IQR). Our framework is conducted with a `Parameters` penalty coefficient of -0.3, and the P-DARTS experiment is with no restriction of `skip_connect`. Three stages of the search process: (a) `Stage 0`; (b) `Stage 1`; (c) `Stage 2`.

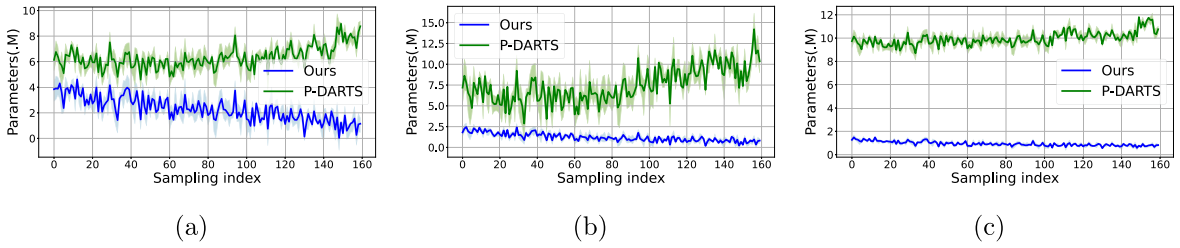

Fig. 10. The comparative experiments of our method versus P-DARTS in the modified search space that with the "`conv_3x1_1x3`" and "`conv_3x3`" primitive operations. The curve represents the variation of the model scale of the leading sub-architecture, with the interquartile range (IQR) denoted by the shaded area. Our framework is conducted with a `Parameters` penalty coefficient of -0.3, and the P-DARTS experiment is with no restriction of "`skip_connect`" operation. Three stages of the search process: (a) `Stage 0`; (b) `Stage 1`; (c) `Stage 2`.

value of approximately `10M`. From our analysis, it becomes evident that P-DARTS attains model compactness primarily through the utilization of compact operations within the original DARTS search space. Thus, it presents significant challenges in achieving a multi-objective architectural search for P-DARTS, e.g., with limitations in terms of model compression capabilities.

7. Conclusion

This study introduces an efficient multi-objective NAS framework, which accommodates the non-differentiable metrics into the differentiable NAS framework. This innovation harnesses the combined strengths of multi-objective NAS and differentiable NAS, rendering it particularly suitable for practical NAS contexts characterized by resource constraints and platform-specific requirements. Our method demonstrates performance on par with state-of-the-art NAS methods, while being notably more efficient in terms of both time and resource utilization compared to prior multi-objective NAS methodologies. Additionally, our framework exhibits scalability and compatibility with the recently introduced search methodology and evaluation strategy within the differentiable NAS framework, as well as certain surrogate-model-based or predictor-based evaluation approaches [46–48]. Our ablation and diagnostic experiments provide insights into the search procedure, monitor the parametric variations, and validate the effectiveness of our search strategy from a process perspective. The diagnostic experiments could be extended to the NAS community, where architecture performance should not be the sole metric for evaluating search methods. It is essential to acknowledge the limitations of our method. First, our framework requires experimental tuning of the penalty-reward reference and coefficients. Second, although we address the non-differentiable objectives, for the sampling in the discrete space, our approach incurs higher computational costs compared to purely differentiable NAS methods. Last but not least, this work adopts a combination of the classic Weighted Sum Method (WSM) and Weighted Product Method (WPM) from the field of MNAS [12,49]. These methods transform the objective vectors into scalar functions to approximate the Pareto optimal solution set, however, there exist notable limitations. First, it is challenging to find suitable weight vectors, denoted as w. Further, the linear aggregation is only applicable to problems with a convex Pareto front, meaning that solutions on non-convex segments are unreachable. Also, selecting a single weight vector can only achieve one Pareto optimal solution and cannot uniformly obtain a Pareto solution set, necessitating multiple executions with different weight configurations to approximate the Pareto optimal frontier. In light of these limitations and building upon this work, we plan to further explore these issues in future research, including self-adaptive tuning of search hyperparameters, e.g., the reference and penalty coefficients.

CRediT authorship contribution statement

Bo Lyu: Writing – original draft, Software, Methodology, Conceptualization. **Yin Yang:** Writing – review & editing. **Yuting Cao:** Validation, Investigation. **Pengcheng Wang:** Validation. **Jian Zhu:** Writing – review & editing. **Jingfei Chang:** Validation. **Shiping Wen:** Supervision.

Declaration of competing interest

There is no conflict of interest in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

This publication was supported by the National Natural Science Foundation of China (62306291), in part by NPRP grants: NPRP 8-274-2-107 from Qatar National Research Fund, in part by Zhejiang Provincial Natural Science Foundation of China under Grant No. LQ24F020029.

References

[1] X. He, K. Zhao, X. Chu, Automl: a survey of the state-of-the-art, Knowl.-Based Syst. 212 (2021) 106622.
[2] Y. Guo, Y. Luo, Z. He, J. Huang, J. Chen, Hierarchical neural architecture search for single image super-resolution, IEEE Signal Process. Lett. 27 (2020) 1255–1259.
[3] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, D. Marculescu, Single-path mobile automl: efficient convnet design and nas hyperparameter optimization, IEEE J. Sel. Top. Signal Process. 14 (4) (2020) 609–622.
[4] K. Zhou, X. Huang, Q. Song, R. Chen, X. Hu, Auto-gnn: neural architecture search of graph neural networks, Front. Big Data 5 (2022) 1–9.
[5] G. Feng, H. Wang, C. Wang, Search for deep graph neural networks, Inf. Sci. 649 (2023) 119617.
[6] Y. Kim, Y. Li, H. Park, Y. Venkatesha, P. Panda, Neural architecture search for spiking neural networks, in: S. Avidan, G.J. Brostow, M. Cissé, G.M. Farinella, T. Hassner (Eds.), Computer Vision - ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXIV, in: Lecture Notes in Computer Science, vol. 13684, 2022, pp. 36–56.
[7] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017, pp. 1–16.
[8] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710.
[9] H. Pham, M.Y. Guan, B. Zoph, Q.V. Le, J. Dean, Efficient neural architecture search via parameter sharing, in: J.G. Dy, A. Krause (Eds.), International Conference on Machine Learning, Stockholm, Sweden, vol. 80, 2018, pp. 4092–4101.
[10] H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, 2019, pp. 1–13.
[11] A. Brock, T. Lim, J.M. Ritchie, N. Weston, SMASH: one-shot model architecture search through hypernetworks, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018, pp. 1–16.
[12] M. Tan, B. Chen, R. Pang, V.K. Vasudevan, M. Sandler, A. Howard, Q.V. Le, Mnasnet: platform-aware neural architecture search for mobile, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2820–2828.
[13] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, M. Sun, Dpp-net: device-aware progressive search for Pareto-optimal neural architectures, in: Proceedings of the European Conference on Computer Vision, 2018, pp. 540–555.
[14] C.-H. Hsu, S.-C. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-H. Chang, T. Pan, Y.-T. Chen, W. Wei, D.-C. Juan, Monas: multi-objective neural architecture search using reinforcement learning, preprint, arXiv:1806.10332, 2018.
[15] F.H. Thomas Elsken, Jan Hendrik Metzen, Multi-objective architecture search for cnns, preprint, arXiv:1804.09081, 2018.
[16] B. Lyu, S. Wen, K. Shi, T. Huang, Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing, IEEE Trans. Cybern. 53 (2) (2023) 1158–1169.
[17] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, Nsga-net: neural architecture search using multi-objective genetic algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2019, pp. 419–427.
[18] X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: bridging the depth gap between search and evaluation, in: 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), IEEE, 2019, pp. 1294–1303.
[19] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, L. Wang, Z. Chen, A. Xiao, J. Chang, X. Zhang, et al., Weight-sharing neural architecture search: a battle to shrink the optimization gap, ACM Comput. Surv. 54 (9) (2021) 1–37.
[20] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in: The Thirty-Third AAAI Conference on Artificial Intelligence, Honolulu, Hawaii, USA, 2019, pp. 4780–4789.
[21] W. Deng, X. Zhang, Y. Zhou, Y. Liu, X. Zhou, H. Chen, H. Zhao, An enhanced fast non-dominated solution sorting genetic algorithm for multi-objective problems, Inf. Sci. 585 (2022) 441–453.
[22] Z. Lu, K. Deb, E.D. Goodman, W. Banzhaf, V.N. Boddeti, Nsganetv2: evolutionary multi-objective surrogate-assisted neural architecture search, in: A. Vedaldi, H. Bischof, T. Brox, J. Frahm (Eds.), Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I, in: Lecture Notes in Computer Science, vol. 12346, 2020, pp. 35–51.
[23] C. Huang, X. Zhou, X. Ran, Y. Liu, W. Deng, W. Deng, Co-evolutionary competitive swarm optimizer with three-phase for large-scale complex optimization problem, Inf. Sci. 619 (2023) 2–18.
[24] H. Lee, S. An, M. Kim, S.J. Hwang, Meta-prediction model for distillation-aware NAS on unseen datasets, in: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023, 2023, pp. 1–11.
[25] C. White, W. Neiswanger, S. Nolen, Y. Savani, A study on encodings for neural architecture search, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual, 2020, pp. 1–13.
[26] C. White, W. Neiswanger, Y. Savani, BANANAS: Bayesian optimization with neural architectures for neural architecture search, in: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, 2021, pp. 10293–10301.
[27] M. Zheng, X. Su, S. You, F. Wang, C. Qian, C. Xu, S. Albanie, Can GPT-4 perform neural architecture search?, CoRR, arXiv:2304.10970 [abs], 2023.
[28] Y. Xu, L. Xie, W. Dai, X. Zhang, X. Chen, G.-J. Qi, H. Xiong, Q. Tian, Partially-connected neural architecture search for reduced computational redundancy, IEEE Trans. Pattern Anal. Mach. Intell. 43 (9) (2021) 2953–2970.

[29] H. Cai, L. Zhu, S. Han, Proxylessnas: direct neural architecture search on target task and hardware, in: International Conference on Learning Representations, New Orleans, LA, USA, 2019, pp. 1–13.
[30] A. Cheng, J. Dong, C. Hsu, S. Chang, M. Sun, S. Chang, J. Pan, Y. Chen, W. Wei, D. Juan, Searching toward pareto-optimal device-aware neural architectures, in: Proceedings of the International Conference on Computer-Aided Design, San Diego, CA, USA, 2018, pp. 136–156.
[31] X. Zhou, X. Cai, H. Zhang, Z. Zhang, T. Jin, H. Chen, W. Deng, Multi-strategy competitive-cooperative co-evolutionary algorithm and its application, Inf. Sci. 635 (2023) 328–344.
[32] T. Elsken, J.H. Metzen, F. Hutter, Efficient multi-objective neural architecture search via lamarckian evolution, preprint, arXiv:1804.09081, 2018.
[33] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, J. Sun, Single path one-shot neural architecture search with uniform sampling, in: Proceedings of the European Conference on Computer Vision, 2019, pp. 544–560.
[34] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (3) (1992) 229–256.
[35] S. Mannor, N. Shimkin, A geometric approach to multi-criterion reinforcement learning, J. Mach. Learn. Res. (2004) 325–360.
[36] L. Huang, S. Sun, J. Zeng, W. Wang, W. Pang, K. Wang, U-darts: uniform-space differentiable architecture search, Inf. Sci. 628 (2023) 339–349.
[37] J. Ji, X. Wang, Fast progressive differentiable architecture search based on adaptive task granularity reorganization, Inf. Sci. 645 (2023) 119326.
[38] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, Q. Tian, Multinomial distribution learning for effective neural architecture search, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1304–1313.
[39] H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han, Once-for-all: train one network and specialize it for efficient deployment, in: International Conference on Learning Representations, Addis Ababa, Ethiopia, 2020, pp. 1–15.
[40] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, K. Murphy, Progressive neural architecture search, in: European Conference on Computer Vision, Munich, Germany, 2018, pp. 19–34.
[41] H. Tan, R. Cheng, S. Huang, C. He, C. Qiu, F. Yang, P. Luo, Relativenas: relative neural architecture search via slow-fast learning, IEEE Trans. Neural Netw. Learn. Syst. 1 (2021) 1–15.
[42] S. Xie, H. Zheng, C. Liu, L. Lin, SNAS: stochastic neural architecture search, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, 2019, pp. 1–17.
[43] L. Tong, B. Du, Neural architecture search via reference point based multi-objective evolutionary algorithm, Pattern Recognit. 132 (2022) 108962.
[44] Z. Chen, F. Zhou, G. Trimponias, Z. Li, Multi-objective neural architecture search via non-stationary policy gradient, CoRR, arXiv:2001.08437 [abs], 2020.
[45] Y. Xue, C. Chen, A. Slowik, Neural architecture search based on a multi-objective evolutionary algorithm with probability stack, IEEE Trans. Evol. Comput. 27 (4) (2023) 778–786.
[46] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, J. Liang, NPENAS: neural predictor guided evolution for neural architecture search, IEEE Trans. Neural Netw. Learn. Syst. 34 (11) (2023) 8441–8455.
[47] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, F. Hutter, Nas-bench-301 and the case for surrogate benchmarks for neural architecture search, preprint, arXiv:2008.09777, 2020.
[48] V. Lopes, B. Degardin, L.A. Alexandre, Are neural architecture search benchmarks well designed? A deeper look into operation importance, Inf. Sci. 650 (2023) 119695.
[49] B. Lyu, H. Yuan, L. Lu, Y. Zhang, Resource-constrained neural architecture search on edge devices, IEEE Trans. Netw. Sci. Eng. 9 (1) (2021) 134–142.