# A Max-Flow Based Approach for Neural Architecture Search

Chao Xue[1,2(✉)], Xiaoxing Wang[3], Junchi Yan[3], and Chun-Guang Li[1]

[1] School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing, People's Republic of China
{xch,lichunguang}@bupt.edu.cn
[2] JD Explore Academy, Beijing, People's Republic of China
[3] Department of Computer Science and Engineering and MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, People's Republic of China
{figure1_wxx,yanjunchi}@sjtu.edu.cn

**Abstract.** Neural Architecture Search (NAS) aims to automatically produce network architectures suitable to specific tasks on given datasets. Unlike previous NAS strategies based on reinforcement learning, genetic algorithm, Bayesian optimization, and differential programming, we formulate the NAS task as a Max-Flow problem on search space consisting of Directed Acyclic Graph (DAG) and thus propose a novel NAS approach, called MF-NAS, which defines the search space and designs the search strategy in a fully graphic manner. In MF-NAS, parallel edges with capacities are induced by combining different operations, including skip connection, convolutions and pooling, and the weights and capacities of the parallel edges are updated iteratively during the search process. Moreover, we interpret MF-NAS from the perspective of non-parametric density estimation and show the relationship between the flow of a graph and the corresponding classification accuracy of a neural network architecture. We evaluate the competitive efficacy of our proposed MF-NAS across different datasets with different search spaces that are used in DARTS/ENAS and NAS-Bench-201.

## 1 Introduction

Recent advances in deep neural networks result in growing interests in automated machine learning (AutoML), whose goal is to optimize hyper-parameters and to identify network architectures suitable to specific datasets without much human intervention. The target of AutoML can be generally formalized as follows:

$$x^* \in \arg\min_{x \in \mathcal{X}} f(x), \tag{1}$$

where $f : \mathcal{X} \to \mathbb{R}$ is a function defined over a search space $\mathcal{X}$.


S. Avidan et al. (Eds.): ECCV 2022, LNCS 13680, pp. 685–701, 2022.
https://doi.org/10.1007/978-3-031-20044-1_39

In the past few years, white-box-based and black-box-based approaches have been dedicated to developing algorithms for AutoML. In a white-box formulation [27,47], the form of function $f$ is explicitly known, so that $x \in \mathcal{X}$ can be optimized in a differential programming manner. In comparison, in a black-box assumption [8,37], the function $f$ can only be evaluated at $x \in \mathcal{X}$, yielding noisy observations.

One important factor for AutoML is the definition and construction of search space $\mathcal{X}$. Different search spaces $\mathcal{X}$ give rise to different settings for AutoML, and thus promote a variety of search strategies. Given a convex set $\mathcal{X} \subset \mathbb{R}^d$, then Eq. (1) can be viewed as a hyper-parameter optimization (HPO) problem. And Bayesian Optimization (BO) [6,37,39] can provide an elegant compromise in terms of capturing a surrogate model to indicate the likelihood of function $f$ and maximizing an acquisition function to trade off exploration and exploitation. Given a tree-based search space $\mathcal{X} \subset \mathcal{T}$, which is often used for either building the hyper-parameters' dependency or searching for a macro neural architecture where the layers are stacked sequentially, some works [38,40] manage to capture the dependency among layers and to decide a proper exploration-exploitation balance by using Monte Carlo Tree Search (MCTS) strategy. Moreover, if the searching space is extended to a Directed Acyclic Graph (DAG), i.e. $\mathcal{X} \subset \mathcal{G}$, where $\mathcal{G}$ is a DAG search space, then the multi-branch and the skip relationship of hyper-parameters can be established. This elastic expression leads to the research direction called Neural Architecture Search (NAS). In [12,15,27,43,49] the architecture is represented as a super-net, and a white-box approach (i.e., differential programming based approach) is used to update architecture's importance and network's weight. However, due to the approximation in bi-level optimization, the differential based methods (both one-shot and single-path approach) suffer instability, i.e.during the search process, the architecture collapsing occurs and thus the skip-layer tends to dominate [10,41]. On the contrary, some black-box BO methods are also introduced to design NAS strategy. Nevertheless, because of the inconsistency between the vector space $\mathbb{R}^d$ and the DAG space $\mathcal{G}$, BO solutions need extra efforts for encoding the neural architecture [19,30,35,45].

Another important factor for AutoML is search strategy, which has been explored by most of the existing methods for NAS based on reinforcement learning, genetic algorithm, Bayesian optimization, and differential programming. These methods decouple the search space and search strategy, leading to lower efficiency due to ignorance on graph property. To bridge the gap, we focus on designing an efficient and stable search strategy on a DAG search space in a fully graphic manner, which is akin to using MCTS in a tree-based search space or adopting Gaussian process (GP) in an Euclidean space. A few prior works [21,46] have pioneered to perform NAS using graph theory. However, [21] is limited to linear search space and lacks evaluation results on real datasets; whereas [46] lacks feedback reward, making it an open-loop strategy—rather than a NAS method it is virtually a method to define search space. Recently, GFlowNet [4,5] views a Markov Decision Processes (MDP) as a flow network, and connects the flow-matching (conservation) conditions to the generated policy with the target reward function. Though GFlowNet targets at sampling a diverse set of candi-

dates, which may be not the case of NAS, GFlowNet builds a solid foundation for achieving the black-box optimization with the flow network.

In this paper, by introducing the maximum-flow calculation in a flow network, we formulate the NAS task as a multi-graph maximum-flow problem directly defined on a DAG search space with edge capacity indicating the contribution of the corresponding operation to the model performance. To be specific, our contributions are three-fold:

1. We propose a Max-Flow based NAS approach, called MF-NAS, which defines the search space and the search strategy both in a fully graphic manner.
2. To our best knowledge, we make the first attempt to address the NAS task from a multi-graph maximum-flow perspective.
3. We conduct extensive experiments to evaluate the proposed search strategy across multiple datasets on multiple search spaces and show competitive performance.

## 2 Preliminaries

### 2.1 Multi-graph Flow in Graph Theory

Consider a directed graph $G := (V, E)$, where $V$ is the vertex set and $E$ is the edge set. A multi-graph is defined as a graph with multiple edges (i.e., parallel edges) between two vertices. A flow graph is a directed graph where each edge has a capacity and receives a flow that is limited by the edge capacity. The formal definition of the problem for finding a feasible and maximal flow on a multi-graph is given as follows.

**Definition 1 (Maximum-Flow on Multi-Graph).** *Given a directed graph $G = (V, E)$ with a source node $s \in V$, a sink node $t \in V$, edge set $E = \{e^k_{u,v} | u, v \in V, k \in \mathcal{K}\}$, edge capacity function $c : V \times V \times \mathcal{K} \to \mathbb{R}$, where $\mathcal{K} := \{0, 1, \ldots, K-1\}$ and $K$ is number of parallel edges, then the max-flow on the multi-graph $G$ is defined as a feasible $s$-$t$-flow function $f : V \times V \times \mathcal{K} \to \mathbb{R}$ that maximizes the flow value $|f| := \sum_{u \in V, k \in \mathcal{K}} f(e^k_{u,t})$ on $G$, where $f(e^k_{u,t})$ denotes the flow on the $k$-th edge from node $u$ to node $t$.*

Flow graph [2,44] has been a useful tool for modeling network traffic, circulation, and etc. In this paper, we demonstrate that NAS can be modeled as a maximum-flow selection problem on a multi-graph.

### 2.2 Hyperband and ENAS

Both Hyperband [24] and ENAS [32] are black-box methods. Hyperband [24] speeds up random search by using an early-stopping strategy to allocate resources adaptively. NAS and ENAS [32,52] both involve training a RNN controller in a loop, where the controller samples a child model (i.e., candidate architecture) for training and its achieved performance is fed back to the controller as a reward to train the LSTM controller. Specifically, in NAS, the child

models are trained from scratch to convergence, whereas in ENAS, the child models share their weights to reduces the cost of architecture search. Our proposed MF-NAS applies to the pipeline of NAS: the controller is responsible for generating candidate architectures (i.e., child models), then the parameters of the controller and the weights of child models are updated alternately. Unlike NAS and ENAS, our MF-NAS adopts a max-flow-based controller.

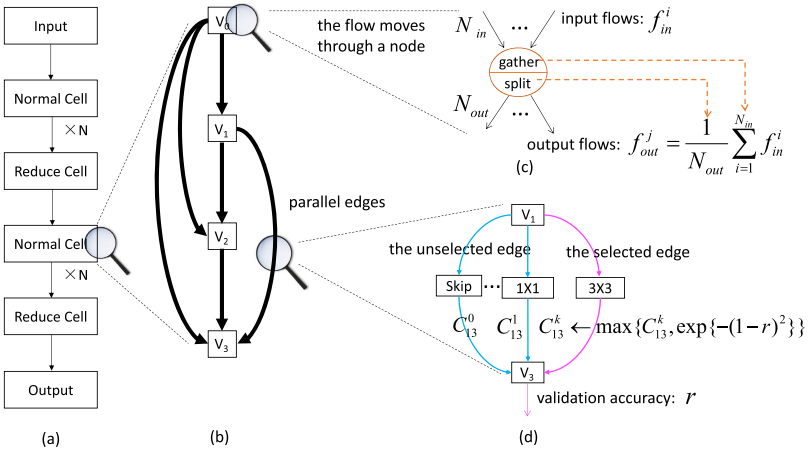


**Fig. 1.** Illustration of NAS as a max-flow problem. a) A cell-based search space used in DARTS and NAS-Bench-201. b) Connections within a cell in NAS-Bench-201, where bold lines indicate a group of parallel edges (i.e., candidate operations). c) The flow moves through a node following the conservation law, where ihe input flows are first collected by a gather function (e.g., summation function) and then the converged flow is split into output flows. d) Parallel edges with limited capacities between the node pair. In each search step, only one path is selected to construct a child model, and the capacity of the selected edge may be updated with respect to validation accuracy.

## 3 Methodology

In this section, we formulate the NAS task as a max-flow problem on a multi-graph at first, and then show that MF-NAS can be interpreted from a non-parametric density estimation perspective. Finally, we integrate MF-NAS into the classical AutoML pipelines, demonstrating that MF-NAS enjoys wide applicability.

We regard the architecture as a directed acyclic graph (DAG) referring to NAS [52], ENAS [32], DARTS [27] and its subsequent works [7,16,41]. For clarity, we give a sketch of the search space in DARTS and NAS-Bench-201 in a multi-graph perspective in Fig. 1. Specifically, operations are regarded as parallel edges between a pair nodes with learned capacities indicating the importance of the architecture, and the moving flow across nodes follows the conservation law [4].

### 3.1 Neural Architecture Search with Maximum-Flow

Given a training dataset $\mathcal{D}$ for a certain task, NAS aims to find an architecture $m \in \mathcal{A}$ that maximizes a posterior probability, i.e.,

$$m^* = \arg\min_{m \in \mathcal{A}} p(m|\mathcal{D}). \tag{2}$$

To formulate the task of NAS, we use a directed multi-graph to represent the cell-based search space, where parallel edges are the operations, including convolutions, skip connection, pooling, and no connection.

Instead of finding the optimal solution with maximum posterior probability as in Eq. (2), in this paper, we consider the flow value on the operation-induced multi-graph to be the candidate architecture's fitness, which will be further interpreted in Sect. 3.2. Consequently, we convert the task of searching for the best architecture to a task of finding a feasible $s$-$t$-flow function $f$ that achieves the maximum-flow value, that is:

$$f^* := \arg\min_{f} \sum_{u \in V, k \in \mathcal{K}} f(e^k_{u,t}). \tag{3}$$

Then, the optimal architecture $m^*$ is selected as a set of edges whose flows are nonzero according to the optimal $s$-$t$-flow function $f$:

$$m^* := \{e^k_{u,v} | u, v \in V \text{ and } k \in \mathcal{K}, \text{ where } f^*(e^k_{u,v}) > 0\}. \tag{4}$$

In this way, the optimization problem in (2) can be solved by addressing the optimal $s$-$t$-flow problem in (3) and (4).

Next, we give a proposition to show that the maximum-flow of the flow network in Fig. 1 (a) can be obtained by calculating the maximum-flow of the *normal cell* and the *reduce cell*, separately.

**Proposition 1.** *Suppose that the architecture is formed by connecting the normal cells $m_{normal}$ and the reduce cells $m_{reduce}$ in a chain manner. Then, the set of edges $m^*$ that maximize the flow value $|f|$ as defined in Definition 1 can be achieved by calculating the union set of that maximizes the flow value of the normal cell $m^*_{normal}$ and that maximizes the flow value of the reduce cell $m^*_{reduce}$, i.e., $m^* = m^*_{normal} \cup m^*_{reduce}$*[1].

*Proof.* This can be proved by mathematical induction. **Base case:** if the layer of architecture is only one (formed by one cell), obviously, the statement holds. **Inductive step:** assume the statement holds for $n$-layers architecture (i.e., $m^*_n = m^*_{normal} \cup m^*_{reduce}$)—this is the induction hypothesis (IH), then the statement will be proved to hold for $n+1$-layers architecture as shown in Fig. 2(c). Without loss of generality, we consider the $n+1$-th cell as a normal cell. Denote $|f_{normal}|$, $|f_n|$ and $|f_{n+1}|$ as the flow value of the normal cell in Fig. 2(a), the

[1] If there are only the normal cells in the architecture, as NAS-Bench-201, then $m^* = m^*_{normal}$.

flow of the $n$-layers and $n+1$-layer network in Fig. 2(c), respectively. According to Max-Flow Min-Cut Theorem [44], the flow of the $n+1$-layer network in the chain can be represented as max $|f_{n+1}| = min\{\text{max } |f_n|, \text{ max } |f_{normal}|\}$. If max $|f_n| >=$ max $|f_{normal}|$, then maximizing the flow of $n+1$-layer network is equivalent to maximize that of the normal cell, and thus $m^*_{n+1} = m^*_n \cup m^*_{normal} \overset{IH}{=} m^*_{normal} \cup m^*_{reduce}$. If max $|f_n| <$ max $|f_{normal}|$, then the maximum flow problem of $n+1$-layers network is equivalent to maximize the flow of $n$-layers network. Any $m_{normal}$ that satisfies max $|f_{normal}| >$ max $|f_n|$ will not affect the result, hence $m^*_{normal}$ can be chosen as the optimal solution $m^*_{n+1} = m^*_n \cup m^*_{normal} \overset{IH}{=} m^*_{normal} \cup m^*_{reduce}$. This completes the proof.

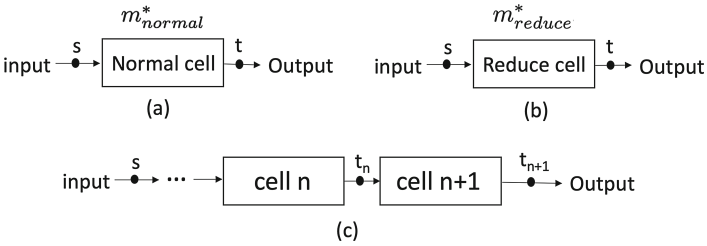


**Fig. 2.** Illustration of mathematical induction for flow network analysis (see Proposition 1): a) base case for normal cell; b) base case for reduce cell; c) inductive step.

Hence, Eq. (3) can be solved in a greedy manner by separately optimizing inside each cell:

$$\begin{aligned} \underset{f}{\text{argmax}} \quad & \sum_{u \in V_{cell}, k \in \mathcal{K}} f(e^k_{u,t}) \\ \text{s.t.} \quad & f(e^k_{u,v}) \leq c(e^k_{u,v}), \quad \forall u, v \in V_{cell}, \ \forall k \in \mathcal{K} \\ & \sum_{u \in V_{cell}} \sum_{k \neq K-1} \mathbb{I}(e^k_{u,v}) = M, \quad \forall v \in V_{cell} \\ & \mathbb{I}(e^{K-1}_{u,v}) = \prod_{k \neq K-1} (1 - \mathbb{I}(e^k_{u,v})), \quad \forall u, v \in V_{cell}, \end{aligned} \tag{5}$$

where $V_{cell}$ is the node set of a normal cell or a reduce cell, $t$ is the sink node of the cell, the index of the sink node is related to the number of nodes $N$ inside the cell, $M$ is the input degree of node, $K$ and $f(e^k_{u,v})$ are defined in Definition 1, and the operation whose index is $K-1$ denotes *no connection*[2], and $\mathbb{I}(e^k_{u,v})$ is an indicator which is 0 if $f(e^k_{u,v}) = 0$ and otherwise 1. The capacities $c(\cdot)$ of

[2] For the search space in ENAS and DARTS, we set $N = 4$, $M = 2$, $K = 8$; for the search space in NAS-Bench-201, we set $N = 3$ and $K = 5$ without constraining $M$.

the parallel edges in the multi-graph flow network are updated by the candidate model's reward (i.e., validation accuracy) as follows:

$$c(e_{u,v}^k) = \begin{cases} e^{-(1-r)^2} & \text{if } e_{u,v}^k \in m^* \& e^{-(1-r)^2} > c(e_{u,v}^k) \\ c(e_{u,v}^k) & \text{otherwise,} \end{cases} \tag{6}$$

where $m^*$ is the edge set defined in Eq. (4), $r$ is the accuracy on validation set for image classification, thus the capacity $c(\cdot)$ will be scaled in $[0, 1]$.

MF-NAS benefits from the update rule in two aspects. On one hand, removing less important operations has little influence on final prediction accuracy, while deleting some important operations can lead to a significant drop [3]. We use Eq. (6) to update the capacities, making important operations less prone to be removed. On the other hand, the update formula of Eq. (6) can be viewed as a contraction mapping of the accuracy, which promotes the exploration capability.

We can use dynamic programming (DP) to solve Eq. (5). Take the DARTS search space as an example. In the DARTS search space, we assume that there are two previous nodes of each cell whose initial flow value can be set to one (i.e., the upper-bound of the accuracy), and the feature maps of different nodes in one cell are concatenated as the output. As shown in Fig. 3, there are two states in each node within the cell of the search space: 1) there exists a link between the $(n-1)$-th node and the $n$-th node 2) there does not exist a link between the $(n-1)$-th node and the $n$-th node. Denote $V_{cell}^n$ as the sub-cell which is terminated by the $n$-th node (i.e., the nodes in the sub-cell are $1, 2, ..., n$), whose max-flow value can be defined as:

$$F_n^* := \max_f \sum_{u \in V_{cell}^{n-1}, k \in \mathcal{K}} f(e_{u,n}^k). \tag{7}$$

We initialize $F_{-1}^* = 1$ and $F_0^* = 1$, and then recursively compute the flow value as follows:

$$F_n^* = \max\{F_n^{(1)},\ F_n^{(2)}\}, \tag{8}$$

where

$$F_n^{(1)} = 2F_{n-1}^* - F_{n-2}^* + \max_{u \in V_{cell}^{n-2}, k \in \mathcal{K}} c(e_{u,n}^k), \tag{9}$$

$$F_n^{(2)} = F_{n-1}^* + \max_{u \in V_{cell}^{n-2}, k \in \mathcal{K}} c(e_{u,n}^k) + \max_{u \in V_{cell}^{n-2} \setminus \{u^*\}, k \in \mathcal{K} \setminus \{k^*\}} c(e_{u,n}^k), \tag{10}$$

in which $(u^*, k^*) := \arg\max_{u \in V_{cell}^{n-2}, k \in \mathcal{K}} c(e_{u,n}^k)$. We can see that updating the two states corresponds to computing $F_n^{(1)}$ and $F_n^{(2)}$, respectively. By using the DP algorithm as in Eq. (8) and the selection method in Eq. (4), we get the candidates architecture $m^*$ for the next running. There is an equivalent representation of Eq. (8), where the flow value is normalized before moving to the next nodes, making the factors slightly different. If the hypothesis space in Eq. (5) is small enough, random search can be a practical alternative to solve Eq. (5).

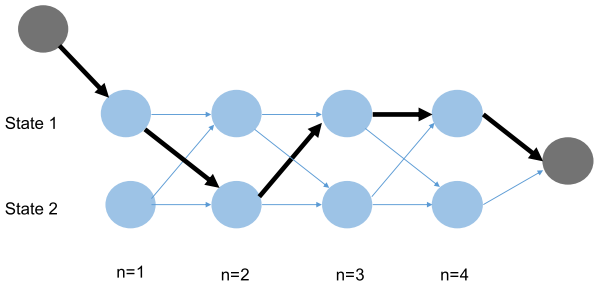


**Fig. 3.** Illustration for the dynamic programming (DP) to solve Eq. (5).

### 3.2 MF-NAS: A Probabilistic Perspective

We now describe the motivation for the maximum-flow formulation as well as the relationship between the flow of a graph and the performance (i.e., classification accuracy) of a network architecture.

Let $\Omega_1$ be the set of available operations, $\Omega_2$ be the set of feasible edges in a search space, $\Omega = \Omega_1 \times \Omega_2$, and $\mathcal{A}$ be a class (i.e., set of sets). Then each element $A \in \mathcal{A}$ represents a feasible architecture in the search space, and it is a subset of $\Omega$, i.e., $A \subset \omega$. Let $\phi : \mathcal{A} \to \mathbb{R}$ be a set function, indicating the reward of an architecture. We define a probability measure of an architecture[3] that can achieve the best reward, i.e.,

$$p(A) := p(\phi(A) = r^*), \tag{11}$$

where $r^* = \max_{A' \in \mathcal{A}} \phi(A')$. Furthermore, the conditional probability over an architecture given a specific edge in it can be derived by $p(A|a) := p(A|a \in A) = p(\phi(A) = r^*|a \in A)$. Here, $a$ is the same denotation as $e^k_{u,v}$ in Definition 1.

To make less assumptions about the conditional distribution, we use a non-parametric approach to estimate the density. Choosing a Gaussian kernel function gives rise to the following kernel density estimation model (KDE):

$$p(\phi(A) = r^*|a \in A) = \frac{1}{N} \sum_{\substack{i=1 \\ a \in A_i}}^{N} \frac{\exp\{-\frac{||r^* - \phi(A_i)||^2}{2\sigma^2}\}}{\sqrt{2\pi}\sigma}, \tag{12}$$

where $N$ architectures containing edge $a$ are sampled. For a small enough $\sigma$, in the exponential term the one for which $\|r^* - \phi(A_i)\|^2$ is the smallest term will approach zero most slowly, and hence the sample architectures with the best reward—$A^*_s = \arg\max_{A_i} \phi(A_i)$ will dominate the density. That is, Eq. (12) can be approximated by:

$$p(\phi(A) = r^*|a \in A) \propto \exp\{-(r^* - \phi(A^*_s))^2\}. \tag{13}$$

[3] Many architectures can get the same best reward.

For image classification tasks, the reward is measured by the accuracy on validation set, and the upper bound of the reward can be set as $r^* = 1$. Obviously, Eq. (13) holds the same form as the equation of capacity update in Eq. (6). As a consequence, the information flow on edge can be interpreted by an approximated conditional probability as shown in Eq. (13), which reveals the preference for different architectures that can achieve the best performance given the contained edge. We further show that the propagation of the conditional probability can be viewed as a flow moving from the input edges to the output edges. In Fig. 4, three basic types of the network topology are illustrated: a) one-input/one-output, where an architecture $A$ containing the output edge has only one input edge, and thus the conditional probabilities conditioned on the output and input are identical; b) many-input/one-output, where an architecture containing any input edge will flow through the output edge, and thus the probability conditioned on the output edge is the summation of the probabilities conditioned on the input edges; c) one-input/many-output, where an architecture containing the input edge spreads to multiple outputs which are chosen uniformly as a priori, and thus the probability is divided equally by the number of output edges. Note that all the network topology including residual or multi-branches graphs can be derived from the three basic types of topology mentioned above. In a nutshell, the probability defined in Eq. (11) can be propagated as a flow, therefore finding an architecture with the highest validation accuracy is equivalent to finding an architecture with the maximum conditional probability on the output edge, which is also equivalent to finding a maximum flow on the DAG graph with the capacity defined in Eq. (6).

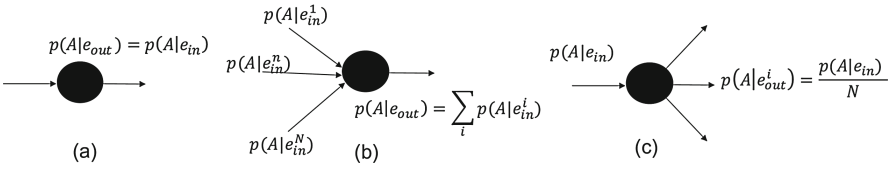


**Fig. 4.** Three types of network topology. a) one-to-one; b) many-to-one; c) one-to-many.

### 3.3 MF-NAS Pipeline

The proposed MF-NAS provides a candidate architecture generation method in a single-path NAS solution. To demonstrate the efficiency and the stability, we apply MF-NAS to a multi-fidelity pipeline. For example, Hyperband [24] selects top-$k$ from its candidate pool; whereas MF-NAS dynamically generates new promising candidates as the search progresses by maximizing the network flow. The reason is that the maximum-flow controller involves the edge capacity constraint; and the flow value on one edge can be different for the same net flow value of a cell. In other words, multiple architectures may result in the

**Algorithm 1** Max-Flow based Neural Architecture Search (MF-NAS)

**Input:**

Search space $S$, Capacity matrix with zeros initialization, and the fixed hyper-parameters $H$, e.g., the learning rate and its decay policy, scale of the chain cell structure, input degree of node in the cell, $N$ search rounds, $n_i$ candidates, each candidate trains $r_i$ steps in round $i$;

Initialization: $\epsilon$-greedy factor $\epsilon$;

**Output:**

The optimal architecture $m^*$;

**for** $i \in \{0, 1, ..., N-1\}$ **do**

  **if** i $= 0$ **then**

    Generate $n_0$ architectures $m_0^0, ..., m_0^{n_0-1}$ randomly;

  **end if**

  Train $m_i^0, \ldots, m_i^{n_i-1}$ with $r_i$ steps;

  Evaluate validation accuracy, update network capacity $c$ by Eq. 6;

  **for** $j \in \{0, 1, ..., n_{i+1}-1\}$ **do**

    **if** $random() < \epsilon$ **then**

      $m_{i+1}^j \leftarrow$ an architecture generated randomly;

    **else**

      $m_{i+1}^j \leftarrow$ an architecture with the maximum-flow calculated by Eq. 5 and Eq. 4;

    **end if**

  **end for**

**end for**

Model selection: return the best architecture $m^*$ with the maximum validation accuracy seen so far.

same net flow value. To enhance the exploration capability of MF-NAS, we choose the candidate architecture using an $\epsilon$-greedy strategy [28]. With this strategy, a random architecture is taken with probability $\epsilon$, and the max-flow architecture is chosen with probability $1-\epsilon$. We anneal $\epsilon$ from 1 to 0 such that the controller begins from an exploration phase and slowly starts to moving towards the exploitation phase. We also keep a small portion of top architectures during different rounds to stabilize the searching process. We refer to Algorithm 1 for giving an implementation for MF-NAS, which aims at finding an architecture whose classification accuracy on validation set is maximized on the operation-induced DAG search space.

## 4 Experiments

To evaluate the effectiveness of our proposed MF-NAS approach, we conduct experiments with different DAG search space on several benchmark datasets, including CIFAR-10 [20], CIFAR-100 [20], STL-10 [13], FRUITS [29], FLOWER-102 [31], Caltech-256 [17] and ImageNet [34].

**Settings in Experiments**. Specifically, we evaluate the efficiency and stability of MF-NAS in three settings: a) the micro search space used in ENAS [32] and

DARTS [27], b) search space of NAS-Bench-201 [16], and c) performance on ImageNet classification. Furthermore, we set the initial learning rate as 0.025 with a cosine scheduler, use SGD with a momentum 0.9. The maximal search round is $N = 4$, and there are n = [30, 20, 10, 5] candidates that are trained r = [30, 60, 70, 80] epochs in different rounds. $\epsilon$ = [1.0, 0.6, 0.5, 0.25] indicates it decays for every search round. Experiments are conducted on RTX 3090 and NVIDIA V100 GPUs.

**Table 1.** Comparison with classification architectures on CIFAR-10. Similar to other NAS algorithms, the search cost of MF-NAS does not include the final evaluation cost.

| Methods | Test error % | Params (M) | FLOPs (M) | Search cost GPU-days | Search Method |
|---|---|---|---|---|---|
| AmoebaNet-B [33] | 2.55 ± 0.05 | 2.8 | 506 | 3150 | Evolution |
| Hierarchical Evo [26] | 3.75 ± 0.12 | 15.7 | – | 300 | Evolution |
| PNAS [25] | 3.41 ± 0.09 | 3.2 | 730 | 225 | SMBO |
| DARTS (1st order) [27] | 3.00 ± 0.14 | 3.3 | 519 | 0.4 | Gradient |
| DARTS (2nd order) [27] | 2.76 ± 0.09 | 3.4 | 547 | 1.0 | Gradient |
| SNAS (moderate) [47] | 2.85 ± 0.02 | 2.8 | 441 | 1.5 | Gradient |
| P-DARTS [11] | 2.50 | 3.4 | 551 | 0.3 | Gradient |
| PC-DARTS (1st order) [48] | 2.57 ± 0.07 | 3.6 | 576 | 0.1 | Gradient |
| BayseNAS [51] | 2.81 ± 0.04 | 3.4 | – | 0.2 | Gradient |
| SGAS [22] | 2.66 ± 0.24 | 3.7 | – | 0.25 | Gradient |
| DARTS+PT [41] | 2.61 ± 0.08 | 3.0 | – | 0.8 | Gradient |
| NASNet-A [53] | 2.65 | 3.3 | 624 | 1800 | RL |
| ENAS [32] | 2.89 | 4.6 | 626 | 0.5 | RL |
| ENAS+$\epsilon$-greedy | 2.82 | 3.7 | 578 | 0.5 | RL |
| Hyperband [24] | 2.96 ± 0.19 | 2.9 | 476 | 2.2 | Random |
| BANANAS [45] | 2.64 | – | – | 11.8 | BO |
| MF-NAS | 2.63 ± 0.16 | 3.3 | 529 | 1.0 | Max-Flow |
| MF-NAS (best) | 2.40 | 4.0 | 653 | 1.0 | Max-Flow |

### 4.1 Results on Micro Cells Based Search Space in ENAS/DARTS

We evaluate MF-NAS on micro cells-based DAG search space used in ENAS [32] and DARTS [27][4]. The main difference between MF-NAS and other algorithms lies in the selection method for candidate networks. To be specific, MF-NAS applies optimization algorithms with graph frameworks to choose candidates; whereas others generate candidates by means of Bayesian optimization, reinforcement learning, evolution or gradient-based methods. For demonstrating the efficiency of MF-NAS, we implement MF-NAS following the controller-child pipeline [32], but do not use the weight-sharing. As shown in the Table 1, max-flow based method gets a 2.63% error rate with 3.3M parameters by taking about one search days, which is more efficient than RL/Evolution/Random/BO based

[4] Precisely, MF-NAS uses the search space of DARTS.

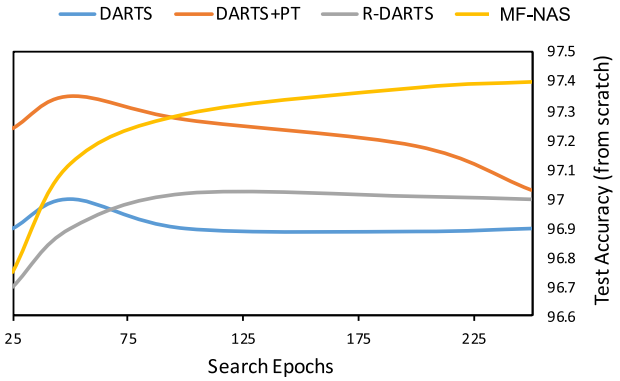


**Fig. 5.** Stability of DARTS (w/ its amendment versions) and MF-NAS on CIFAR-10.

methods on DAG search space. This comes from the fact that the proposed MF-NAS are coupled with DAG search space, and thus they can take advantage of the graph structure, i.e. the relation of the edges and nodes. Also, unlike the gradient-based methods, MF-NAS does not suffer the skip domination issue or the discretization problem [41]. To validate the stability of MF-NAS, we evaluate the performance at different search steps in Fig. 5.

Figure 5 compares the stability of DARTS [27] and its amendment versions [41,50] with MF-NAS. Note that the search epoch is not related to real search time. For DARTS+PT [41], we get our results on the super-nets of DARTS pre-trained from 25 to 250 training epochs. Both vanilla DARTS and DARTS+PT suffer from the skip layer dominated problem [7,50], i.e. there are about 6 skip connections in the normal cell at 250 epochs. The ultimate performance of differential methods usually decreases over search epochs because they use some approximations to solve the bi-level optimization. In contrast, thanks to the maximum-flow solution, MF-NAS can steadily achieve improved performance.

**Table 2.** Top-1 test set accuracy with one GPU day budget.

| NAS method | STL-10 ($96 \times 96$) | FRUITS ($100 \times 100$) | FLOWER-102 ($256 \times 256$) | Caltech-256 ($256 \times 256$) |
|---|---|---|---|---|
| Hyperband | $0.79 \pm 0.02$ | $0.99 \pm 0.0006$ | $0.95 \pm 0.002$ | $0.50 \pm 0.06$ |
| ENAS | $0.79 \pm 0.02$ | $0.97 \pm 0.0012$ | $0.93 \pm 0.007$ | $0.49 \pm 0.05$ |
| DARTS | $0.80 \pm 0.01$ | $0.99 \pm 0.0018$ | $0.94 \pm 0.006$ | $0.60 \pm 0.03$ |
| MF-NAS | $0.80 \pm 0.02$ | $0.99 \pm 0.0009$ | $0.95 \pm 0.002$ | $0.65 \pm 0.02$ |

Based on this search space, we evaluate MF-NAS's generalization ability. We use other four datasets with different image resolutions, including STL-10 [13], FRUITS [29], FLOWER-102 [31], and Caltech-256 [17]. ENAS [32], DARTS [27], and Hyperband [24] are evaluated as the baselines whose search protocols cover reinforcement learning, differential method and closed-loop random search.

Similar to ENAS, we extend Hyperband to support neural network search in the same search space of DARTS. The experiments are under the budget constraints of one GPU day. As shown in Table 2, MF-NAS outperforms other methods on the last dataset. We analyze that Caltech-256 is a challenging dataset, on which the early performance of an architecture does not align with its ultimate performance precisely, and thus Hyperband fails. Additionally, the limited GPU memory restricts the batch size for DARTS on large-scale images, which can cause a performance drop [36].

### 4.2 Results on Search Space of NAS-Bench-201

In NAS-Bench-201, the architectures are stacked by 17 cells with five operations, and three datasets are evaluated, including CIFAR-10 [20], CIFAR-100 [20], and ImageNet-16-120 [16]. We follow the training settings and split protocol as the paper [16]. We search for the architecture based on MF-NAS three times with different random seeds, and the results are reported in Table 3, showing that our method achieves competitive results with the state-of-art methods. We observe that ENAS can also benefit from the $\epsilon$-greedy strategy, indicating that ENAS may suffer from the weight-sharing problem: it is hard to figure out whether the improvement under the weight-sharing strategy is attributed to a better-chosen architecture or comes from well-trained weights. Thus, there will still be a big room to improve the weight-sharing strategy in the NAS area.

**Table 3.** Top-1 accuracy (%) on NAS-Bench-201.

| Methods | CIFAR-10 | | CIFAR-100 | | ImageNet-16-120 | |
|---|---|---|---|---|---|---|
| | Valid | Test | Valid | Test | Valid | Test |
| DARTS-V1 [27] | 39.77 ± 0.00 | 54.30 ± 0.00 | 15.03 ± 0.00 | 15.61 ± 0.00 | 16.43 ± 0.00 | 16.32 ± 0.00 |
| DARTS-V2 [27] | 39.77 ± 0.00 | 54.30 ± 0.00 | 15.03 ± 0.00 | 15.61 ± 0.00 | 16.43 ± 0.00 | 16.32 ± 0.00 |
| GDAS [15] | 89.89 ± 0.08 | 93.61 ± 0.09 | 71.34 ± 0.04 | 70.70 ± 0.30 | 41.59 ± 1.33 | 41.71 ± 0.98 |
| SETN [14] | 84.04 ± 0.28 | 87.64 ± 0.00 | 58.86 ± 0.06 | 59.05 ± 0.24 | 33.06 ± 0.02 | 32.52 ± 0.21 |
| RSPS [23] | 82.25 ± 4.90 | 86.09 ± 4.90 | 56.56 ± 8.91 | 56.39 ± 8.70 | 31.17 ± 6.28 | 30.27 ± 5.80 |
| DARTS-PT [41] | – | 88.11 | – | – | – | – |
| DARTS-PT (fix $\alpha$) [41] | – | 93.80 | – | – | – | – |
| ENAS [32] | 37.55 ± 3.14 | 53.80 ± 0.71 | 13.47 ± 2.21 | 14.00 ± 2.27 | 14.88 ± 2.19 | 14.87 ± 2.05 |
| ENAS+$\epsilon$-greedy | 77.37 ± 3.08 | 84.10 ± 3.16 | 56.85 ± 4.47 | 57.31 ± 4.95 | 32.16 ± 2.95 | 32.93 ± 3.14 |
| MF-NAS | 27437 | 93.72 ± 0.55 | 71.23 ± 1.07 | 71.86 ± 0.64 | 44.16 ± 0.10 | 44.33 ± 0.18 |
| ResNet [18] | 90.83 | 93.97 | 70.42 | 70.86 | 44.53 | 43.63 |
| Optimal | 91.61 | 94.37 | 73.49 | 73.51 | 46.77 | 47.31 |

### 4.3 Results on ImageNet Classification

For ImageNet [34], the one-shot NAS approaches cost too much memory to build a super-net. Works [9,27,53] directly transfer the architecture searched on CIFAR-10 to ImageNet with little modification. Specifically, the search phase

runs on CIFAR-10, while the evaluation phase uses the architecture transferred from the searched cells with deeper and wider scales as suggested in DARTS [27]. Learning rate decay policy is different between the last two blocks of Table 4. Table 4 evaluates the performance on ImageNet. In general, MF-NAS can get competitive results in transfer cases.

**Table 4.** Study on full ImageNet. The first block represents the direct search case while the last two blocks show the transfer cases. † and ‡ represent lr decay strategy as DARTS and P-DARTS/PC-DARTS, respectively.

| Methods | Params | FLOPs | GPU (days) | Top-1 Acc |
|---|---|---|---|---|
| NASNet-A | 5.3M | 620M | 1800 | 0.740 |
| AmoebaNet-A | 5.1M | 541M | 3150 | 0.745 |
| DARTS† | 4.9M | 545M | 1 | 0.731 |
| MF-NAS† | 4.9M | 549M | 0.4 | 0.744 |
| P-DARTS‡ | 4.9M | 557M | 0.3 | 0.756 |
| PC-DARTS‡ | 5.3M | 586M | 0.1 | 0.749 |
| MF-NAS‡ | 4.9M | 549M | 0.4 | 0.753 |

### 4.4 Ablation Study on $\epsilon$-greedy

The proposed MF-NAS follows the reinforcement learning based NAS method [1] to involve the $\epsilon$-greedy strategy into the algorithm pipeline. To eliminate the influence of $\epsilon$-greedy, we show the ablation study in Table. 5.

**Table 5.** Test accuracy (%) with different $\epsilon$-greedy strategies on CIFAR-10. $\epsilon \equiv 0$ means the $\epsilon$-greedy strategy has no effect on the corresponding NAS method; Note that $\epsilon \equiv 1$ degrades to Random Search.

| Methods | $\epsilon \equiv 0$ | $\epsilon \equiv 1$ | $\epsilon$-greedy used in our paper |
|---|---|---|---|
| Hyperband | 97.04 | 96.75 | 97.06 |
| ENAS | 97.11 | 96.75 | 97.18 |
| MF-NAS | 97.32 | 96.75 | 97.40 |

The results show that MF-NAS and ENAS [32] can benefit from the $\epsilon$-greedy strategy. In fact, $\epsilon$-greedy strategy can enhance the exploration capacity, which may be more critical in a maximum-flow/RL strategy than a random search solution. As a consequence, max-flow-based and reinforcement learning-based NAS methods turn out to be more efficient than random search schemes considering the pure random search solution only gets $96.75 \pm 0.2$ % accuracy.

## 5 Conclusion and Future Work

We have proposed a multi-graph maximum-flow approach for NAS, called MF-NAS, which casts the problem of optimal architecture search as finding a path on DAG. In the search phase, the weights and the capacities of the parallel edges in the multi-graph are updated alternately. Extensive experiments on the image classification task demonstrated the effectiveness of our proposal. As the future work, we will explore for using larger search space, with non-linear gather functions in a general maximum-flow network, and test for other vision tasks, e.g., segmentation and detection [42].

**Acknowledgments.** J. Yan is supported by the National Key Research and Development Program of China under grant 2020AAA0107600. C.-G. Li is supported by the National Natural Science Foundation of China under grant 61876022.

## References

1. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. In: ICLR (2017)
2. Bein, W.W., Brucker, P., Tamir, A.: Minimum cost flow algorithms for series-parallel networks. Discrete Appl. Math. **10**, 117–124 (1985)
3. Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., Le, Q.V.: Understanding and simplifying one-shot architecture search. In: ICML (2018)
4. Bengio, E., Jain, M., Korablyov, M., Precup, D., Bengio, Y.: Flow network based generative models for non-iterative diverse candidate generation. In: NeurIPS (2021)
5. Bengio, Y., Deleu, T., Hu, E.J., Lahlou, S., Tiwari, M., Bengio, E.: GFlowNet foundations. arXiv preprint arXiv:2111.09266 (2021)
6. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: NeurIPS (2011)
7. Bi, K., Hu, C., Xie, L., Chen, X., Wei, L., Tian, Q.: Stabilizing DARTS with amended gradient estimation on architectural parameters. arXiv:1910.11831 (2019)
8. Bonilla, E.V., Chai, K.M.A., Williams, C.K.I.: Multi-task gaussian process prediction. In: NeurIPS (2007)
9. Chao, X., Mengting, H., Xueqi, H., Chun-Guang, L.: Automated search space and search strategy selection for AutoML. Pattern Recognit. **124**, 108474 (2022)
10. Chen, X., Hsieh, C.J.: Stabilizing differentiable architecture search via perturbation-based regularization. In: ICLR (2020)
11. Chen, X., Xie, L., Wu, J., Tian, Q.: Progressive differentiable architecture search: bridging the depth gap between search and evaluation. In: ICCV (2019)
12. Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X., Yan, J.: Darts-: robustly stepping out of performance collapse without indicators. In: ICLR (2021)
13. Coates, A., Ng, A.Y., Lee, H.: An analysis of single-layer networks in unsupervised feature learning. In: AISTATS (2011)
14. Dong, X., Yang, Y.: One-shot neural architecture search via self-evaluated template network. In: ICCV (2019)
15. Dong, X., Yang, Y.: Searching for a robust neural architecture in four GPU hours. In: CVPR (2019)

16. Dong, X., Yang, Y.: An algorithm-agnostic NAS benchmark. In: ICLR (2020)
17. Griffin, G., Holub, A., Perona, P.: Caltech-256 object category dataset (2007)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
19. Kandasamy, K., Neiswanger, W., Schneider, J., Póczos, B., Xing, E.P.: Neural architecture search with Bayesian optimisation and optimal transport. In: NeurIPS (2018)
20. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Technical report, Citeseer (2009)
21. de Laroussilhe, Q., Jastrzkebski, S., Houlsby, N., Gesmundo, A.: Neural architecture search over a graph search space. CoRR (2018)
22. Li, G., Qian, G., Delgadillo, I.C., Muller, M., Thabet, A., Ghanem, B.: SGAS: sequential greedy architecture search. In: CVPR (2020)
23. Li, L., Talwalkar, A.: Random search and reproducibility for neural architecture search. In: UAI (2019)
24. Li, L., Jamieson, K.G., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: a novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res., 185:1–185:52 (2017)
25. Liu, C., et al.: Progressive neural architecture search. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11205, pp. 19–35. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01246-5_2
26. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. In: ICLR (2018)
27. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: ICLR (2019)
28. Mnih, V., et al.: Nature (2015)
29. Muresan, H., Oltean, M.: Fruit recognition from images using deep learning. Acta Universitatis Sapientiae Informatica (2018)
30. Nguyen, V., Le, T., Yamada, M., Osborne, M.A.: Optimal transport kernels for sequential and parallel neural architecture search. In: ICML (2021)
31. Nilsback, M., Zisserman, A.: Automated flower classification over a large number of classes. In: ICVGIP (2008)
32. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: ICML (2018)
33. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI (2019)
34. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. In: IJCV (2015)
35. Shi, H., Pi, R., Xu, H., Li, Z., Kwok, J., Zhang, T.: Bridging the gap between sample-based and one-shot neural architecture search with BONAS. In: NeurIPS (2020)
36. Smith, S.L., Kindermans, P., Ying, C., Le, Q.V.: Don't decay the learning rate, increase the batch size. In: ICLR (2018)
37. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: NeurIPS (2012)
38. Su, X., et al.: Prioritized architecture sampling with Monto-Carlo tree search. In: CVPR (2021)
39. Swersky, K., Snoek, J., Adams, R.P.: Multi-task Bayesian optimization. In: NeurIPS (2013)
40. Wang, L., Fonseca, R., Tian, Y.: Learning search space partition for black-box optimization using Monte Carlo tree search. In: NeurIPS (2020)

41. Wang, R., Cheng, M., Chen, X., Tang, X., Hsieh, C.J.: Rethinking architecture selection in differentiable NAS. In: ICLR (2021)
42. Wang, X., Lin, J., Zhao, J., Yang, X., Yan, J.: EAutoDet: efficient architecture search for object detection. In: Farinella, T. (ed.) ECCV 2022. LNCS, vol. 13680, pp. 668–684 (2022)
43. Wang, X., Xue, C., Yan, J., Yang, X., Hu, Y., Sun, K.: MergeNAS: merge operations into one for differentiable architecture search. In: IJCAI (2020)
44. West, D.B., et al.: Introduction to Graph Theory, vol. 2. Prentice Hall, Upper Saddle River (1996)
45. White, C., Neiswanger, W., Savani, Y.: Bananas: Bayesian optimization with neural architectures for neural architecture search. In: AAAI (2021)
46. Xie, S., Kirillov, A., Girshick, R.B., He, K.: Exploring randomly wired neural networks for image recognition. In: ICCV (2019)
47. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: stochastic neural architecture search. In: ICLR (2019)
48. Xu, Y., et al.: PC-DARTS: partial channel connections for memory-efficient architecture search. In: ICLR (2019)
49. Xue, C., Wang, X., Yan, J., Hu, Y., Yang, X., Sun, K.: Rethinking Bi-level optimization in neural architecture search: a gibbs sampling perspective. In: AAAI (2021)
50. Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., Hutter, F.: Understanding and robustifying differentiable architecture search. In: ICLR (2020)
51. Zhou, H., Yang, M., Wang, J., Pan, W.: BayesNAS: a Bayesian approach for neural architecture search. In: ICML (2019)
52. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: ICLR (2017)
53. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR (2018)