Neural-Architecture-Search-Based Multiobjective Cognitive Automation System

Eric Ke Wang, Ship Peng Xu, Chien-Ming Chen, and Neeraj Kumar

***Abstract*—Currently, deep-learning-based cognitive automation for decision-making in industrial informatics is a new hot topic in the field of cognitive computing, among which multiobjective architecture optimization is of great difficulty in the research area. When the existing algorithms face multiobjective cognitive model problems, it often takes a lot of time to continuously set different search preference parameters to generate a new search process. This article mainly aims to solve the problem in a multiobjective neural architecture search process, and the key issue is how to adapt user preferences during architectural search. We propose a new algorithm: linear-prefer coevolutionary algorithm. Compared to the original user-constrained method and the Pareto-dominant NSGA-II algorithm, we have faster adaptation time and better quality of adaptation. At the same time, it can respond to user's needs at a relatively faster pace during the reasoning phase. Based on a large number of comparative test results, our algorithm is superior to the traditional cognitive automation algorithms for the multiobjective problem in search quality.**

***Index Terms*—Cognitive automation, evolutional algorithm, multiobjective, neural architecture search (NAS), Pareto dominant.**

I. Introduction

COGNITIVE automation originates from the artificial intelligence of the computer system that simulates human brain. Through the interaction and continuous learning between human and the natural environment, it helps decision makers reveal extraordinary insights from different types of massive data, so as to realize different degrees of perception, memory, learning, and other cognitive activities [1]. In the era of big data, the scale, type, speed, and complexity of data far exceed the cognitive capacity of human brain. How to effectively realize the cognition of big data also brings great challenges to traditional cognitive computing.

Manuscript received January 28, 2020; revised June 4, 2020; accepted June 7, 2020. Date of publication June 29, 2020; date of current version June 16, 2021. This work was supported in part by the National Key R&D Program of China under Grant 2018YFB1003800 and Grant 2018YFB1003805, in part by the Guangdong Province Natural Science Foundation under Grant 2016A030313660 and Grant 2017A030313365, and in part by the Shenzhen Municipal Science and Technology Innovation Project under Grant JCYJ20160608161351559, Grant KQJSCX70726103044992, Grant JCYJ20170811155158682, and Grant JCYJ20160428092427867. *(Corresponding author: Neeraj Kumar.)*

Eric Ke Wang and Ship Peng Xu are with the Department of Computer Science, Harbin Institute of Technology (Shenzhen), Harbin 518055, China (e-mail: wk_hit@hit.edu.cn; 19s051059@stu.hit.edu.cn).

Chien-Ming Chen is with the Shandong University of Science and Technology, Shandong 266510, China (e-mail: chienmingchen@ieee.org).

Neeraj Kumar is with the Department of Computer Science and Information Engineering, Asia University, Taichung 41354, Taiwan, and also with the Thapar Institute of Engineering and Technology, Patiala 147004, India (e-mail: neeraj.kumar@thapar.edu).

Digital Object Identifier 10.1109/JSYST.2020.3002428

Deep learning (DL), as a new machine learning method, has become an excellent solution to cognitive computing in the era of big data. By building a multilayer machine learning model based on presentation, DL trains massive data and learns useful features to improve the accuracy of recognition, classification, or prediction [2]. An excellent DL cognitive model will get excellent cognitive effect. The most critical problem here is the selection of deep neural network model.

With rapid development of the industrial applications in various areas, massive data have been accumulated. Processing and analyzing the data content to obtain valuable information has become a key task. More and more DL and neural network models have been applied to industrial applications. However, the design and verification of a suitable and excellent DL-based cognitive automation model for the application of industrial applications often requires a lot of human labor, such as selection algorithm, super parameter adjustment, iterative modeling, and model evaluation. Therefore, if there is a way that can automatically find the right solution to the current problem, it can effectively save the artificial labor of scientists and liberate the creativity of researchers. Neural architecture search (NAS) is a new research direction and one of the hot topics in automated machine learning [3], [4]. By designing a cost-effective search method, the neural network structure with strong generalization ability and hardware friendliness can be obtained automatically, which saves a lot of human cost. Its deployment and use are more suitable for industrial promotion and application.

NAS is designed to solve the problem that designing DL networks heavily relies on profound knowledge of experts, aiming to achieve competitive performance while automating the neural network architecture designing process. For some task such as image recognition, NAS has yielded very promising results.

A very important branching problem of NAS is how to effectively perform the multiobjective NAS process. For historical reasons, there are little related works about multiobjective NAS. However, the architecture of many neural networks often requires not only high accuracy, but also low resource consumption. Therefore, multiobjective NAS is an area that needs to be studied. In contrast, many current algorithms often require users to set the tradeoff between accuracy and resource consumption in advance. Thus, different search preference parameters will be set to generate a new search process to fit the user's new request. Owing to the user's preference drift, previous algorithms about multiobjective NAS often waste a lot of time on adapting users' preferences.

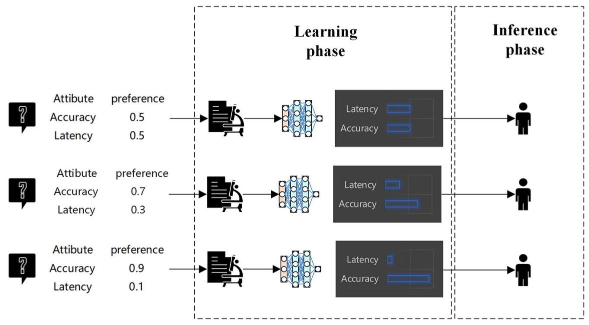

Fig. 1. Given preference scenario.

As shown in Fig. 1, given the preference, the platform first obtains the user's preference and then learns in the user's preferred condition. After obtaining the network structure that meets the user's needs, the network is returned to the user. In this scenario, each inference of the user's preferences means a time-consuming learning process. In the process of single inference, each user needs to spend more resources. Moreover, faced with a user scenario that requires rapid deployment, the server still needs a long search time, which often cannot meet the requirements of rapid deployment of the user.

A. Contribution

In order to solve the key problem in the multiobjective neural network architecture search process, i.e., how to adaptively set user preferences when searching for architecture, we propose a user preference delayed method.

1) We split the traditional NAS process into two relatively independent phases: learning phase and inference phase.
2) In the learning phase, we use a coevolutionary algorithm with preference and optimal individual to explore the solution set that satisfies user preferences.
3) In the inference phase, we can respond to the user's new preferences using this solution set within a short time.
4) The main purpose is to satisfy the problem that users can update search preferences, and the system can adapt the preference quickly. In learning phase

In this article, we use the average adaptive quality and the average adaptive speed as indicators. The algorithm of this article is superior to other algorithms in both indicators.

B. Organization

The rest of this article is organized as follows. Section II presents related work. Section III describes the working of the proposed scheme in terms of multiobjective optimization, delayed preference scenario, and linear-prefer coevolutionary (LCNAS) algorithm. Section IV highlights the experiments results. Finally, Section V concludes this article.

II. Related Work

Our model is related to both areas: multiobjective optimization and NAS. In the following related work, we mainly introduce them from two aspects.

A. Neural Architecture Search

The NAS can be divided into the following subproblems: the design of the search space, the selection of the search strategy, and the acceleration of the search process.

Search space can be divided into chain search space, multibranch search space [5], [6], search space based on basic unit construction [7], and hierarchical representations of search space [8]. The main methods of NAS can be divided into three categories, using reinforcement learning and evolutionary learning, as well as other algorithmic methods. In terms of reinforcement learning, Zoph *et al.* [7] used a recurrent neural network (RNN) as a controller to transform the sequence generated by the RNN into a model and learn neural architectures through reinforcement learning. In terms of evolutionary algorithms, Liu *et al.* [9] and Real *et al.* [10], [11] designed a search space for evolutionary algorithms and used genetic algorithms to search neural architectures in this search space. In other algorithms, Hutter *et al.* [12] used sequential model-based optimization as a search algorithm, and Liu *et al.* [13] proposed to use differentiable NAS to discover hardware-aware efficient convolutional networks. When it comes to the acceleration of the NAS process, there are some approaches that were proved to be effective, such as parameter sharing [14], network morphism [15], and one-shot architecture search [16].

B. Multiobjective NAS

At present, there are two types of algorithm commonly used in multiobjective NAS: one is the search algorithm under the constraint condition [17], [18], and the other is the multitarget search algorithm under the Pareto boundary [19]–[24].

In user-constrained algorithms, users often need to set a series of constraints or preferences in advance and then construct the neural structure search process as a one-time search process under specific constraints. It leads to two main problems: 1) the search process can only find a neural network architecture that satisfies a single constraint or user preference. When the user's constraints or preferences change, the search process needs to be researched; and 2) the choice of constraints or preferences is usually problem dependent, so it is difficult to the search process without information about the problem characteristics.

On the one hand, the Pareto-dominant strategy algorithm (such as NSGA-II [27]) is suitable for low-dimensional multiobjective optimization problems. However, for high-dimensional multiobjective optimization problems, the use of Pareto dominance often leads to weak selection pressure. Besides, in the high-dimensional space, the computational complexity of using the Pareto algorithm is also relatively high.

To get relatively more search results that satisfy user preferences in the search time while reducing the search computational complexity, we propose the LCNAS algorithm.

C. Delayed Linear Preference Scenario

The delayed linear preference scenario is introduced in Yang's work [30] for building a multiobjective reinforcement learning algorithm. We simplified this scenario as two phases, used it to rebuild the NAS process and introduced the original thoughts of this scenario as follows.

A preference equation is given as follows: $t : R^m \to R$ indicates that m, which mapped the target vectors to a single scalar. Given a feasible solution $\mathbf{x}$ and its target vector $\hat{y} = \mathbf{F}(\mathbf{x}) = (y_1, \ldots, y_m)$, the utility of the strategy can be defined as $t(\hat{y})$. The practical utility of this feasible solution is $t_\omega(\mathbf{x}) = \boldsymbol{\omega}^\top \mathbf{x}$ for a feasible solution $\mathbf{x}$, where the preference weight is $\omega \in \Omega$, $\omega = (\omega_1, \ldots, \omega_n)$, $\sum_{n=1}^{n} \omega = 1$. In this particular case, the preference solution can only come from the convex coverage set (CCS) of the Pareto boundary.

In a multiobjective optimization problem, given its Pareto boundary $\mathcal{P}^*$, this CCS can be defined as

$$\begin{aligned} \text{CCS} := \{ \hat{\mathbf{x}} \in \mathcal{P}^* | \exists \omega \in \Omega \text{ s.t. } \omega^{\mathrm{T}} \mathbf{F}(\hat{\mathbf{x}}) \\ \geq \boldsymbol{\omega}^{\mathrm{T}} \mathbf{F}(\hat{\mathbf{x}}'), \forall \hat{\mathbf{x}}' \in \mathcal{P}^* \}. \end{aligned} \tag{1}$$

Another key feature of delayed linear preference is delayed. In the original article, Yang divided the delayed linear preference scenario as three consecutive phases: learning phase, analysis phase, and execution phase. Being delayed means that linear preferences are initially unspecified, and no specific linear preferences will be given until the later stages of the scenario.

D. Decomposition-Based Multiobjective Evolutionary Algorithms (MOEAs) Using Adaptive Weights

For effectively studying the CCS in the learning phase, we use preference coevolution strategy. In fact, the idea of user preference for coevolution stems from the decomposition-based MOEAs using adaptive weights. The preference weight can be regarded as the weights used to inspire the algorithm to attain a better result, which means that it can guide the evolutionary algorithm to optimize along the given directions.

There are a variety of methods for weight adaptation. Zhang *et al.* [25] proposed to incorporate a dynamic weight design method into MOEA/D (denoted as DMOEA/D) in order to enable MOEA/D to perform well on problems having different geometries. Wang *et al.* [26] proposed an algorithm called preference-inspired coevolutionary algorithm using weights, in which weights are coevolved with candidate solutions during the search process.

However, most decomposition-based MOEAs using adaptive weights algorithms try to use the coevolution algorithm to get a Pareto boundary, neglecting the fact that the weight-based approach naturally has an advantage in solving convex problems. Therefore, we apply the weight-based algorithm to approximate in the NAS task.

III. Methodologies

In this section, we illustrate how to form the NAS as delayed linear preference scenario problem and introduce the difference between the scenario based on the given preference and the scenario based on the delayed preference. After formulating the problem, we propose a novel algorithm called linear preference coevolution algorithm for effectively searching the CCS in the learning phase.

A. Multiobjective Optimization

First, we will give a general description of the multiobjective optimization problem. For a multiobjective optimization problem with n decision variables and m target variables, we can express it as follows:

$$\begin{cases} \max & \mathbf{y} = \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})) \\ \text{s.t.} & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \ldots, q \\ h_j(\mathbf{x}) = 0, & j = 1, 2, \ldots, p \end{cases} \tag{2}$$

$\mathrm{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}) \in \mathrm{X} \subset \mathrm{R}^\mathrm{n}$ is the n-dimensional decision vector and X is the n-dimensional decision space. $y = (y_1, \ldots, y_m) \in Y \subset R^m$ is an m-dimensional target vector. YY is the m-dimensional target space. The objective function $\mathbf{F}(x)$ defines m mapping functions from the decision space to the target space; $g_i(x) \leq 0 (i = 1, 2, \ldots, q)$ defines q inequality constraints; and $h(x) = 0 (j = 1, 2, \ldots, p)$ defines p equality constraints.

B. Delayed Preference Scenario

Typically, users upload to the platform based on their own user needs, while the platform processes the user's needs and then process a one-time search to find the appropriate architecture and return it to the user. The needs of different users are often different, and the user's needs will change over time.

It describes $\mathbf{M}_\mathcal{L}$ as follows, using a mathematical language:

$$\mathbf{m} \in \mathbf{M}_\mathcal{L} \Rightarrow \exists \omega \in \Omega \text{ s.t. } \forall \mathbf{m}' \in \mathbf{M}, \omega^\top \mathbf{F}(\mathbf{m}) \geq \omega^\top \mathbf{F}(\mathbf{m}'). \tag{3}$$

$\mathbf{M}$ is contained in $\mathbf{M}_\mathcal{L}$ only if there is at least one linear preference f_ω under this preference equation; no other individual $\mathbf{m}$ can produce higher practical utility.

In the learning phase, the computing resources of the search are relatively sufficient, but the linear preference is unknown at this moment. For multiobjective evolutionary algorithms, we focus on whether we can approximate the CCS with $\mathbf{M}_\mathcal{L}$.

At this stage, the goal of multiobjective evolutionary learning is to obtain an $\mathbf{M}_\mathcal{L}$ to approximate CCS. For any related preference vector ω, there is always one in $\mathbf{M}_\mathcal{L}$ that is the best result for ω. In general, the learning phase is prepared to handle various preferences in the future.

In the inference phase, a linear preference equation $f_\omega(\cdot) = \omega$ will be given. Less time will be used to give an optimal individual $\mathbf{m}_\omega$ as a response from the already learned policy set $\mathbf{M}_\mathcal{L}$. We refer to this process of selecting the optimal strategy for a particular preference as preference adaptation. Because the CCS that needs to be learned can be very large, it is not very easy to effectively select the best individual under the current preference through the previously learned $\mathbf{M}_\mathcal{L}$.

In the case of the delayed preference based on Fig. 2, the platform only needs to record the optimal individual set $\mathbf{M}_\mathcal{L}$ in one learning process. In the inference phase, it is possible to adapt each user with a very limited resource consumption. In other words, in this scenario, the platform can efficiently and quickly infer the optimal solution under current user's preference. Therefore, based on the delayed preference scenario, although it may bring more resource consumption in the learning

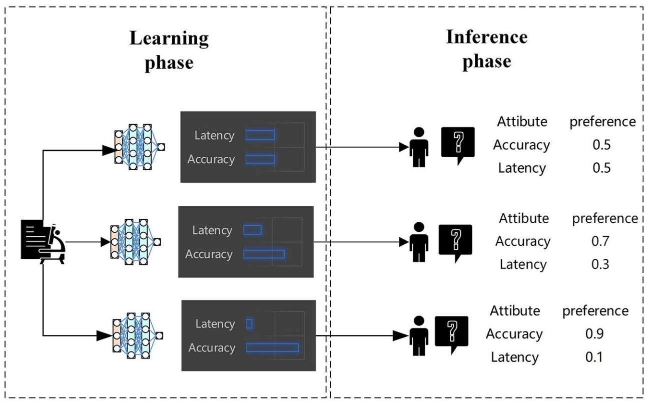

Fig. 2. Delayed preference scenario.

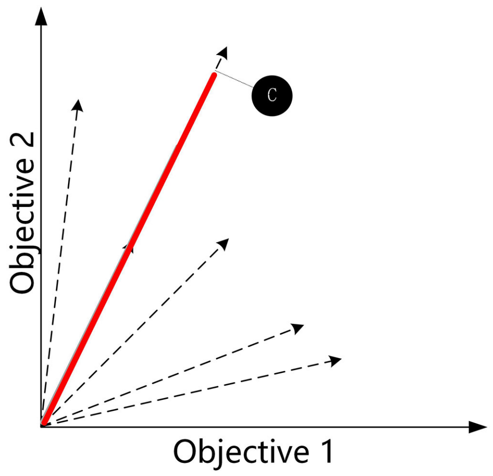

Fig. 3. Description of the maximum utility individual function.

phase, if the large number of users and the changing needs of these users are taken into consideration, the scenario of delayed preference has a great significance for neural network search. It can save more computing resources, respond more rapidly to the needs of users for deployment, and can better handle changing user preferences.

C. LCNAS Algorithm

In the learning phase, an important question is how to effectively use an evolutionary algorithm to approximate CCS on the given a set of preferences. To solve this problem, we propose a linear-prefer hierarchical sorting algorithm. Through this algorithm, we can effectively select individuals that are dominant under linear preference. Another important problem is that multiobjective evolutionary algorithms tend to optimize only in the direction of a given preference, which results in a solution that does not effectively cover the entire CCS. To solve this problem, we have adopted a strategy of preference coevolution. In this strategy, we generate a new set of preferences during each iteration. Then, we will select the preferences that will enable the selected individuals to achieve maximum utility, as shown in Fig. 3.

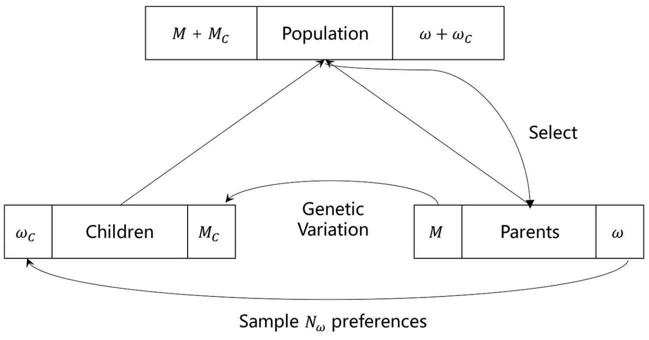

Fig. 4. Implementation framework of LCNAS.

Algorithm 1: LCNAS Algorithm.

```
Input: a preference sampling distribution $\mathcal{D}_\omega$, minibatch
    sizes preference $N_\Omega$, the number of individuals in
    each generation $N$, the initial population $\mathcal{M}_0$
    $\mathcal{PMF}^* = \emptyset$
    $M = \mathcal{M}_0$
    $F_M$ = objectiveFuction $(M)$
    $\mathcal{PMF}^*$ = updatePMF$(\mathcal{PMF}^*, F_M)$
    Sample $N_\Omega$ preferences in which $\omega \sim \mathcal{D}_\omega$ as $\Omega$
    for each $t = [1, 2, \ldots, T]$ do
      $M_C$ = generate-new-pop$(\mathcal{M})$
      $F_{M_C}$ = objectiveFuction $(M_C)$
      $M = M \cup M_C$
      $F_M = F_M \cup F_{M_C}$
      Sample $N_\Omega$ preferences in which $\omega \sim \mathcal{D}_\omega$ as $\Omega_C$
      $\Omega = \Omega \cup \Omega_C$
      $(F_M, M, \Omega)$ = coEvolve $(F_M, M, \Omega)$
      $\mathcal{PMF}^*$ = updatePMF $(\mathcal{PMF}^*, M, F_M, \Omega)$
    end for
  Output: $\mathcal{PMF}^*$
```

Specifically, we implemented the LCNAS algorithm in the framework of the diagram shown in Fig. 4.

M and Ω are fixed-size N and N_Ω vectors, representing population of candidate individuals and preferences. At each generation t, genetic variation operators are applied to parents M, producing N offspring M_C. Meanwhile, N_Ω new weight vectors ω_C are generated. M and M_C, Ω and Ω_C, will be joined, respectively. The population will then be sorted using a linear-prefer operator, and the best N individuals will be selected as the new candidate population M. N_Ω preferences will be selected as the new preferences vector Ω. Additionally, we will use an offline dictionary $\mathcal{PMF}^*$ to store the candidate solutions.

The pseudocode of LCNAS is presented in Algorithm 1. The core part of LCNAS lies in the function **coEvolve**, which will be elaborated next.

1) Coevolve Algorithm: Function **coEvolve** evaluates the performance of candidate individuals and preferences and then selects new parent population M and Ω, respectively. The parent population M will be sorted, using linear-prefer front sort. A preference will be selected when ranking selected candidate solution as the best.

The pseudocode of function coEvolve is as follows.

Algorithm 2: coEvolve Algorithm.

```
Input: preference vector $\Omega$, candidate solutions $M$, and its
    objective values $F_M$
  $\mathcal{F} = $ linear-dominate-front-sort$(\mathrm{M}, \Omega)$
  $\mathrm{M_{c+1}} = \emptyset$ and $\mathrm{j} = 1$
  while $|M_{t+1}| + |\mathcal{F}_d| \leq \mathbb{N}$ do
    $M_{t+1} = M_{t+1} \cup \mathcal{F}_4$
    $\mathrm{i} = \mathrm{i} + 1$
  end while
  $M_{t+1} = M_{t+1} \cup \mathcal{F}_l[1 : (\mathrm{N} - [M_{t+1}|)]$
  $M = M_{t+1}$
  for $m \in M$ do
    $\omega = $ selectP$(m, \widetilde{\omega})$
  end for
Output: $\Omega$, $M$, $F_M$
```

1) Line 1 applies function **linear-dominate-front-sort** to sort the individuals using linear-dominate operator. The pseudocode of linear-prefer-front-sort is represented in Algorithm 3. We define the **linear-dominate** operator as **linear-dominate**(m, w, Ω), where the individual $m,\ w \in M, \Omega$ is the preference vector

$$\textbf{linear-dominate}(m, w, \Omega) = \begin{cases} \text{True}, & \omega^{\mathrm{T}}\mathbf{F}(\mathrm{m}) \geq \omega^{\mathrm{T}}\mathbf{F}(\mathrm{w}), \forall \omega \in \Omega \\ \text{False}, & \text{otherwise} \end{cases}.$$

2) Line 2 applies function **selectP** to select the most suitable preference vector from W for each of the survived candidate solutions. We define the maximum individual utility function as **selectP**$(m,\ \widetilde{\omega}\ )$, where the individual $m \in M$, ω is a preference on the preference distribution $\mathcal{D}_\omega$:

$$\text{select}P(M, \omega) = \max_{\omega} \omega^T F(m)\ \omega \in \widetilde{\omega}. \tag{4}$$

Algorithm 3: Linear-Dominate Front Sort Algorithm.

```
Input: $M$, and its objective values $F_M$, preference vector $\Omega$
  for $m \in M$ do
    $S_m = \varnothing$
    $n_m = 0$
    for $w \in M$ do
      if $lineardominate(m, w)$ then
        $S_m = S_m \cup \{w\}$
      else if $lineardominate((w, m)$ then
        $n_m = n_m + 1$
      end if
      if $n_m = 0$ then
        $m_{\text{rank}} = 1$
        $\mathcal{F}_1 = \mathcal{F}_1 \cup \{\mathrm{m}\}$
      end if
      $\mathrm{j} = 1$
      while $\mathcal{F}_i \neq \emptyset$ do
        $\mathrm{W} = \emptyset$
        for $m \in M$ do
          for $w \in M$ do
            $n_m = n_m\ C1$
            if $n_m = 0$ then
              $w_{\text{ tank }} = i + 1$
              $W = W\ \cup \{w\}$
            end if
          end for
        end for
      end while
    end for
  end for
Output: $\Omega$, $M$, $F_M$
```

IV. Experiment and Results

A. Experimental Settings

This section of the experiment is set up to explore the performance of LCNAS under the given data set CIFAR-10.

In our work, we use the NASBench to evaluate the performance data. NASBench is a tabular dataset, which maps convolutional neural network architectures to their trained and evaluated performance on CIFAR-10.

For both algorithms, we set the genetic process of initial population number $N = 100$ and $T = 10$ times, where $p_M = 0.8, q_M = 0.1, p_C = 0.2$, and $q_C = 0.3$. For the environment of preference adaptation, we set the preference number to $N_\omega = 100$. Our test of the effectiveness of the two algorithms is to test the $(acc(m), trainingTime(m)$, and $param(m)$ of the model obtained by a fixed training round, a fixed number of populations, and the number of genetic processes. $(acc(m), trainingTime(m)$, and $param(m)$, respectively, correspond to three indexes of a neural network model: accuracy, the convergence time for training the model, and the quantity of parameters [31]. For the ordinary genetic algorithm, we take the method of taking the optimal group three times.

B. Evaluation Metrics

In our work, we introduce two indicator to evaluate the quantitative performance of LCNAS under the multitarget NAS. The first indicator is the quality of adaptation, which is used to evaluate the average utility of different algorithms under the same user preference. The second indicator is the speed of adaptation, which is the average time it takes for the learning phase and the inference phase, corresponding to a set of user preferences.

1) Adaptive Utility (AU): For a preference set of $\Omega_u = (\omega_1, \ldots, \omega_n)$ from n user preferences, we use a multiobjective evolutionary algorithm A to learn the set of individuals, which approximates the CCS under Ω_u. In the inference stage, we can get a set of response individual $M_u = m_1, \ldots, m_n$ Then, we can define the AU as

$$\mathrm{AU}(A) = \sum_{i=1}^{n} \omega_i^\top \boldsymbol{F}\,(m_i). \tag{5}$$

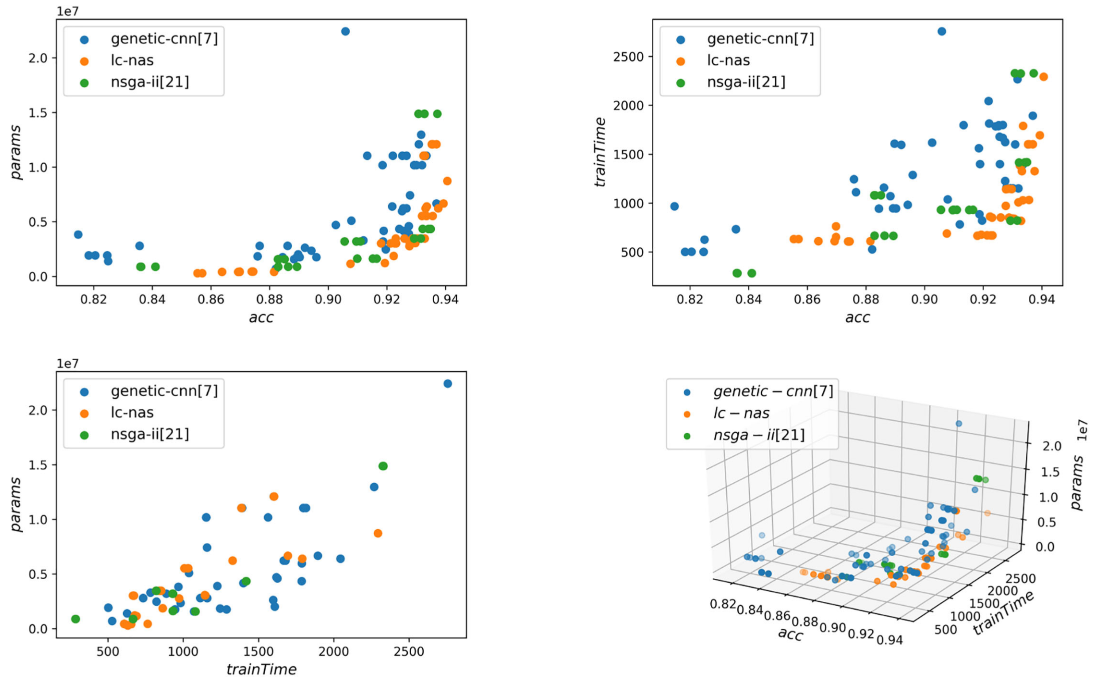

Fig. 5. Solution set visualization.

Among them, $\boldsymbol{F}$ is the objective function, which indicates the accuracy and resource consumption index of the structure in the NAS.

2) Adaptive Time: For a preference set of $\Omega_u = (\omega_1, \ldots, \omega_n)$ from n user preferences, we use a multiobjective evolutionary algorithm A to learn this set of preferences and use t_L to represent the time used in learning phase. In the inference phase, we can use the time cost for responding each preference as $t_1 \ldots t_n$. Then, we can define the adaptive time as

$$\mathrm{AT}(A) = t_L + \sum_{i=1}^{n} t_i. \tag{6}$$

3) Coverage: Coverage is the ability to measure multiobjective algorithms to find all potential optimal solutions. For a multiobjective evolutionary algorithm A, we use M to represent the optimal population set obtained by the algorithm and M_{target} to represent the desired target population set. We define that coverage can be defined in the form of F1 scores:

$$\mathrm{CR}_A(M^*, M_{\text{target}}) = 2 \cdot \frac{\text{precision} \ \cdot \ \text{recall}}{\text{precision} \ + \text{recall}} \tag{7}$$

where precision $= |M^* \cap M_{\text{target}}|/|M^*|$, which represents the ratio of the optimal solution in the solution set, and recall $= |M^* \cap M_{\text{target}}|/|M_{\text{target}}|$, which represents the ratio of the optimal solution obtained to the entire optimal solution.

C. Experimental Results and Experimental Analysis

First, in Fig. 5, we visualize the solution set that was finally obtained using the genetic-CNN [4], LCNAS, and NSGA-II [27] algorithms after the iteration of 100 generations. It is the comparison of pairs of indexes, including the pair of accuracy and parameter, the pair of accuracy, and training time.

TABLE I
AU METRIC

	AU	
N_u	genetic-cnn[4]	**lc-nas**
1	0.190	**0.191**
2	0.274	**0.275**
4	0.353	**0.355**
8	0.290	**0.291**
16	0.313	**0.314**
32	0.341	**0.342**
64	0.307	**0.308**

From an intuitive point of view, we can speculate that the solution set obtained using LCNAS is of higher quality, meaning that the solution set obtained with LCNAS is more accurate and has less parameter and training time. In order to quantitatively evaluate the solution sets, we will introduce the concept of AU to evaluate the quality of the solution set obtained by algorithms. However, if we want to compare the two NAS algorithms using AU, we must perform normalization operation. The reason is that in the previous section, the three parameters of the NAS mentioned, i.e., the accuracy rate, the training time, and the parameter quantity, often have different quantity sizes. Here, we have selected the data of the last generation of the two algorithms to be normalized, and the normalized results are shown in Fig. 6.

After using the normalized operation, we compare the AU values of the two algorithms, as shown in Table I. From the table, we can see that as the number of user preferences increases, the average utility value of the two algorithms shows a trend of increasing first, then decreasing, and then stabilizing. Then, under each user preference number, the average utility of LCNAS is higher than the genetic-CNN algorithm. Therefore, given a set of preferences, LCNAS can achieve greater average utility.

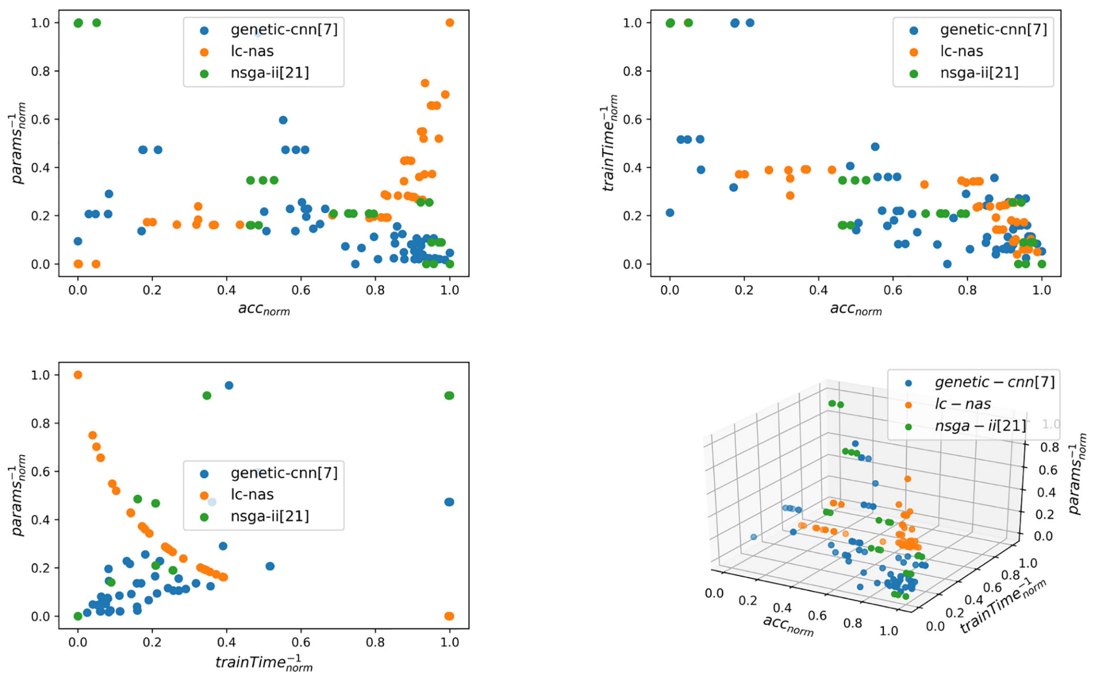

Fig. 6. Normalized results.

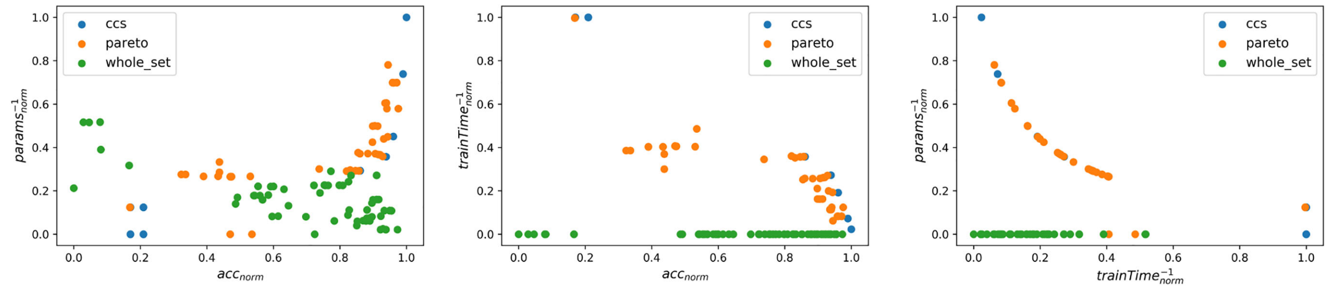

Fig. 7. Pareto and CCS visualization.

TABLE II
METRICS COMPARISON WITH NONLINEAR-DOMINATED ALGORITHMS

	Precision		recall		Coverage Ratio	
	CCS	Pareto Front	CCS	Pareto Front	CCS	Pareto Front
genetic-cnn[4]	0.270	0.230	0.338	0.404	0.300	0.293
lc-nas	**0.520**	**0.510**	**0.863**	**0.930**	**0.649**	**0.659**
nsga-ii[27]	0.530	0.470	0.663	0.825	0.589	0.599

In addition, in order to compare the search capabilities of the two algorithms in the same dimension, we introduce the concept of coverage. First, we combine and normalize the last-generation solution sets obtained by the two algorithms to obtain a set of solutions. Then, we find the corresponding Pareto solution set and CCS solution set in this solution set. We visualizes the pareto solution set and the CCS solution set of the merged set, as shown in Fig. 7.

We use the coverage index to evaluate the solution set obtained by the two algorithms, and the results are shown in Table II. Among them, the first row shows the approximation effect of the two algorithms on the CCS. We can see that LCNAS performs better in terms of recall and coverage. The second shows the approximation effect of the two algorithms on Pareto front. The LCNAS algorithm better approaches the Pareto boundary to some extent and wins higher recall and accuracy than the CNN algorithm.

D. Results and Comparison With Other Methods

To further assess the performance of our proposed algorithm, we performed another experiment on CIFAR-10. Performance comparison, where the amount of parameters and the accuracy are mainly focused on, was carried out between the following methods and manual models:

1) The architecture model LC-NET-V1, obtained by the LC-NAS method in the NASBENCH101 search space;
2) MobileNet V1,V2, [28], [29], which are manually designed architecture aiming for less parameters while retaining high predictive performance;
3) NASNet [7], which is the NAS by reinforcement learning and previously achieved state-of-the-art performance on CIFAR-10;

TABLE III
COMPARISON OF MODELS

MODEL	PARAMS	ACC (%)	PARAMS/ACC
MOBILENET	834K	95.0	8.78
MOBILENET V2	850K	95.4	8.91
NASNET	926K	95.3	9.72
RANDOM SEARCH	1200K	94.7	12.67
DPP-NET	1.0M	95.3	10.5
LC-NAS	**873K**	**94.1**	**9.27**

4) DPP-net [19], which is a multiobjective NAS method proposed by Dong *et al.*;
5) a random search baseline of NAS.

Table III shows a summary of the performance comparison experiments between the above methods and manual models.

As shown in Table III, the LC-NET-V1 achieves a considerable accuracy with relatively few parameters. In addition, LC-NET-V1 obtains the smallest amount of parameters, compared with other NAS method and is close to MobileNet V1 and MobileNet V2, which are designed manually for mobile performance.

This relatively good model was searched by the LC-NAS method, under the relatively limited search space of NAS-BENCH101. It is on the same level with other search methods that consume huge GPU models in terms of accuracy and is superior to other methods in terms of parameters.

V. Conclusion

In this article, we proposed a preference coevolutionary algorithm and a new scenario for the original multiobjective cognitive automation for industrial informatics, which divided the cognitive automation process into two steps: learning phase and inference phase. In the learning phase, the preference coevolutionary algorithm was employed to explore the solution set that satisfies user preferences. In the inference phase, the algorithm could respond to the user's new preferences in a short time using this solution set. Experimental results showed that our algorithm is superior to other algorithms. Compared to the original user-constrained method and the Pareto-dominant NSGA-II algorithm, it had faster adaptation time and better quality of adaptation.

References

[1] A. Fasth-Berglund and J. Stahre, "Cognitive automation strategy for reconfigurable and sustainable assembly systems," *Assembly Autom.*, vol. 33, no. 3, pp. 294–303, 2013.

[2] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[3] Y. He *et al.*, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis*, 2018, pp. 815–832.

[4] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, pp. 1–21, 2019.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[6] B. Li, K. Xu, X. Cui, Y. Wang, X. Ai, and Y. Wang, "Multi-scale DenseNet-based electricity theft detection," in *Intelligent Computing Theories and Application*. Cham, Switzerland: Springer-Verlag, 2018.

[7] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.

[8] C. Liu *et al.*, "Auto-deepLab: Hierarchical neural architecture search for semantic image segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 82–92.

[9] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 13–25.

[10] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. 33rd AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.

[11] Esteban Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[12] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. Int. Conf. Learn. Intell. Optim.*, 2011, pp. 507–523.

[13] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 106–118.

[14] H. Pham, M. Y. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.

[15] C. Liu *et al.*, "Progressive neural architecture search," *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 19–35.

[16] M. Cho, M. Soltani, and C. Hegde, "One-shot neural architecture search via compressive sensing," 2019, *arXiv:1906.02869*.

[17] M. Tan *et al.*, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2815–2823.

[18] L. Tan, B. Brotherton, and T. Sherwood, "Bit-split string-matching engines for intrusion detection and prevention," *ACM Trans. Archit. Code Optim.*, vol. 3, pp. 3–34, 2006.

[19] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "DPP-Net: Device-aware progressive search for Pareto-optimal neural architectures," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 517–531.

[20] N. Kumar, N. Chilamkurti, and S. C. Misra, "Bayesian coalition game for the Internet of Things: An ambient intelligence-based evaluation," *IEEE Commun. Mag.*, vol. 53, no. 1, pp. 48–55, Jan. 2015.

[21] N. Kumar, A. V. Vasilakos, and J. J. P. C. Rodrigues, "A multi-tenant cloud-based DC nano grid for self-sustained smart buildings in smart cities," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 14–21, Mar. 2017.

[22] J. Lian *et al.*, "Deep-learning-based small surface defect detection via an exaggerated local variation-based generative adversarial network," *IEEE Trans. Ind. Inform.*, vol. 16, no. 2, pp. 1343–1351, Feb. 2020.

[23] A. Makkar and N. Kumar, "User behavior analysis-based smart energy management for webpage ranking: Learning automata-based solution," *Sustain. Comput.: Inform. Syst.*, vol. 20, pp. 174–191, 2018.

[24] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1442–1464.

[25] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

[26] R. Wang, R. C. Purshouse, and P. J. Fleming, "Preference-inspired co-evolutionary algorithms using weight vectors," *Eur. J. Oper. Res.*, vol. 243, no. 2, pp. 423–441, 2015.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[28] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

[30] R. Yang *et al.*, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in *Proc. Conf. Adv. Neural Info. Process. Syst.*, 2019, pp. 14636–14647.

[31] B. Wu *et al.*, "Shift: A zero FLOP, zero parameter alternative to spatial convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9127–9135.