Yilin Liu

Exploring the Use of Large-scale Multi-objective Evolutionary Algorithms for Training Neural Network Weights

FAKULTÄT FÜR
INFORMATIK

Intelligent Cooperative Systems
Computational Intelligence

Exploring the Use of Large-scale Multi-objective Evolutionary Algorithms for Training Neural Network Weights

Bachelor Thesis

Yilin Liu

March 18, 2021

Supervisor: Prof. Dr.-Ing. habil. Sanaz Mostaghim

Advisor: Dr.-Ing. Heiner Zille

Yilin Liu: *Exploring the Use of Large-scale Multi-objective Evolutionary Algorithms for Training Neural Network Weights*
Otto-von-Guericke Universität
Intelligent Cooperative Systems
Computational Intelligence
Magdeburg, 2021.

Abstract

In recent years, the study of multi-objective optimization problems with large-scale variables has become one of the hot research directions in the field of evolutionary computation, and at the same time neural networks have been applied in more and more fields. Current training methods for neural networks are mainly based on the back-propagation algorithm. In this work, we treat the weight training of neutral networks as a multi-objective optimization problem and optimize the problem using large-scale multi-objective evolutionary algorithms, where the network's architecture remains unchanged during the optimization process. We use Non-dominated Sorting Genetic Algorithm (NSGA-II), as well as large-scale optimization techniques such as Grouped and Linked Polynomial Mutation and Linear Combination Search Mechanism. In addition, we propose a combination of Grouped and Linked Polynomial Mutation and Linear Combination Search Mechanism, and apply it to this particular problem like the other algorithms.

We compare the results obtained by the evolutionary algorithms and the BP algorithm using the "adam" optimizer and explore the behavior of the evolutionary algorithms in the optimization process as well as the effectiveness of different large-scale optimization techniques for this particular problem. The experiments show that the evolutionary algorithms can achieve acceptable results on feedforward neural networks with a relatively small number of weights, and that the applied large-scale optimization technique, namely Grouped and Linked Polynomial Mutation, has the ability to improve the performance of NSGA-II on this problem. However, there is still a gap between the evolutionary algorithm and the BP algorithm when the number of weights is very large, and in optimizing convolutional neural networks.

Preface

I want to express my sincere gratitude to everyone who has supported and helped me during the writing of this thesis. First of all I would like to thank Sanaz Mostaghim for giving me the opportunity to complete this thesis in the chair of computational intelligence. I am very grateful to Heiner Zille for providing me with the topic and for his patient, thorough and detailed guidance and help from the beginning to the end of the thesis. Finally I want to thank my family for their continuous support.

Contents

List of Figures

List of Tables

1 Introduction

In recent years, in the field of multi-objective optimization algorithms, solving multi-objective optimization problem (MOP) containing large-scale decision variables has become one of the hot research topics. Evolutionary algorithms (EA), as one of the ideal methods for solving MOPs, combined with different search strategies and optimization techniques, also provide an effective research method for large-scale complex MOPs.

Deep learning, another rapidly growing field of machine learning in recent years, has been successfully applied to image speech recognition, automatic machine translation and many other fields. The universal approximation theorem [10] states that neural networks are capable of approximating arbitrarily complex functions with arbitrary accuracy. The power of neural networks often comes with a huge number of parameters, and how to train neural networks effectively and efficiently has been one of the main focuses of research in the field of deep learning.

As training a neural network can be viewed as an optimization problem that contains neural network weights as decision variables, and the proposed large-scale optimization techniques also have the potential to make it practical to use EA to solve this problem. In addition, the global search space of EA, the ability to include multiple optimization objectives simultaneously, and some drawbacks of the back-propagation (BP) algorithm as the dominant method for training neural networks nowadays also make the use of EA more attractive. The thesis examines the feasibility of training neural network weights using large-scale multi-objective evolutionary algorithms.

1.1 Motivation

At present, the mainstream algorithms for training neural networks are based on the BP algorithm. After the scalar of the loss function is obtained through

forward propagation through the network, the information provided by the loss function is used for backward propagation to calculate the gradients. With the obtained gradients, the weight can be updated by different gradient descent algorithms. The emergence of BP algorithm simplifies the process of calculating the weights of deep neural networks.

The BP algorithm has contributed to the great success of neural networks, but the algorithm itself has some shortcomings, such as:

- Slow convergence rate

  The BP algorithm converges very slowly. For deep neural networks, the gradient solution of each layer of the network depends on the gradient of the subsequent layer. Therefore, for any layer, we cannot determine the gradient of the current layer when the gradient of the subsequent layer is unknown [30].

- Local search mehtod

  The BP algorithm is an optimization method of local search. In optimization problems, the aim is to find the global optima, but BP as a greedy algorithm is likely to fall into local optima [30].

- Gradient vanishing and gradient explosion

  Backpropagation passes the updated information of the gradient layer by layer. As the number of layers increases, the updated information of these gradients may attenuate or increase exponentially, making the network weights unable to be updated reasonably.

These shortcomings reflect the need to explore new training methods for neural networks and also the advantages of using the EA to train networks. As a global search method, EAs are not constrained by local optimality. When using EAs, weight updates are not based on gradients and thus the problems caused by gradients do not arise. In addition, the efficiency of EAs can be greatly improved by combining them with distributed computing.

Moreover, although EAs generally could take longer time to find the optima, unlike the BP algorithm that converges too slowly as complained above, the high time cost of EAs is due to their global search space. In addition, because the nature of the EA is very suitable for parallel processing, distributed computing can greatly accelerate its running time, some of the specific implementations of which can be found in [40][20]. Although there have been many

research results in distributed back-propagation over the years, the two main forms are currently synchronous and asynchronous distributed training, each with its own drawbacks [2].

1.2 Goals

The thesis is dedicated to exploring the performance of large-scale multi-objective evolutionary algorithms in training neural networks. This means that the EA will be applied to neural networks of different types and architectures. More specifically, all weights and biases in the neural networks with fixed architectures are encoded into a form that can be processed by the EA and then optimized by the EA. During the optimization process, the architecture of the neural network remains unchanged. In addition to the standard EA, two techniques designed for large-scale optimization problems that have been proposed in recent years are applied to test whether they provide performance improvement for the EA on this optimization problem. Moreover, in this thesis, a combination of the these two techniques is proposed, and again the results will be compared with those of other methods. The process of these methods will be described in the subsequent sections. In order to verify the performance of different methods, multiple data sets of different complexity are used and compared with the BP-based training algorithm.

The thesis also aims to identify the unique advantages and disadvantages of the EA for training neural networks. Furthermore, this work should also make a certain contribution to exploring the limit of the number of parameters that a large-scale EA can handle.

1.3 Structure of Thesis

A general overview of the content of the chapters of this thesis is provided here. In the first chapter we introduced the motivation for using large-scale multi-objective EAs to optimize the weights of neural networks and the goals of this thesis. Chapter 2 provides an introduction to the fundamental concepts involved in this thesis. Next, the related work of this thesis is enumerated and summarised in Chapter 3. Subsequently, Chapter 4 describes the dataset and network architecture used in the thesis, as well as the specific implementation

of the used EAs to optimize the network weights. The experimental results and their corresponding analysis are described in Chapter 5. Finally, Chapter 6 provides the conclusions of the thesis in light of our goals and discusses future work.

2 Fundamentals

The basic concepts covered in the thesis are introduced in this chapter. First are the components of a neural network and the way in which feedforward neural and convolutional neural networks process data and output results. Then comes the process of optimizing the network weights by the BP algorithm. This is followed by the definition of multi-objective optimization. Finally, the optimization process of EA is described and the various operators involved in the process.

2.1 Neural Networks

Artificial neural networks (ANNs) were inspired by the biological structure of the human brain and the way it processes information. Similar to human neural networks, artificial neural networks are composed of neurons which cooperate with each other to learn and process information. As early as the 1950s, artificial neuron models, called "perceptrons" were proposed in [35].

Figure 2.1 shows the structure of a simple perceptron, where $X_i (i = 1, ..., n)$ is the input information, $w_i (i = 1, ..., n)$ represents the weight of each connection, b is called the bias which is often just a constant, and f denotes a certain activation function. The input information is summed over the weights by the following formula:

$$z = \sum_{i=1}^{n} w_i X_i + b \tag{2.1}$$

After that we transform z using a specific activation function to obtain the activation of the perceptron act, i.e:

$$act = f(z) \tag{2.2}$$

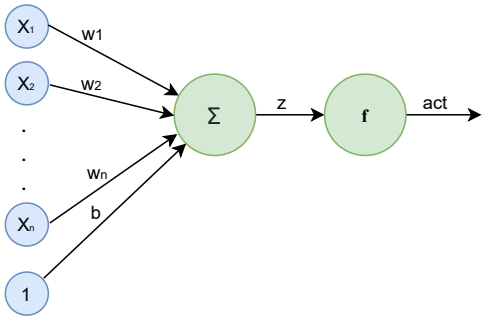


Figure 2.1: Structure of a Perceptron

The purpose of using activation functions is to add nonlinear factors to perceptrons or neural networks to give them the ability to solve nonlinear problems. Some popular activation functions are shown as follows:

- Step Function:

$$f(z) = \begin{cases} 0, \text{ for } z \geq 0 \\ 1, \text{ for } z < 0 \end{cases} \tag{2.3}$$

- Sigmoid or Logistic Activation Function:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

- ReLU:

$$f(z) = max(0, z) \tag{2.5}$$

- Softmax:

$$f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.6}$$

  The softmax function is a more generalized logical activation function for multi-class classification problems. It can also be considered as a normalization that outputs a normalized probability distribution for all classes.

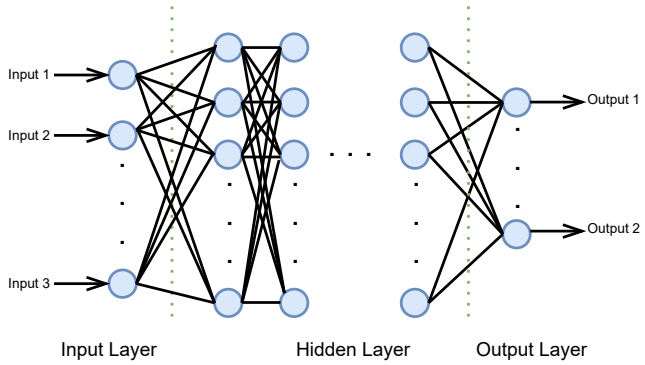


Figure 2.2: Structure of a Feed-Forward Neural Network

2.1.1 Feed-Forward Neural Networks

To solve complex problems in the real world, we need a much more complex network than a simple perceptron.

As shown in Figure 2.2, a typical feedforward network consists of input layer, hidden layer and output layer. The neurons in the input layer are used to receive information about the problem to be solved, and the neurons in the hidden layer are responsible for processing the input information. The output layer serves as the final layer of the neural network to output results. Before the output layer, the output of each layer is directly used as the input of the next layer. The layers are connected to each other by weights, and each neuron works like the perceptron described previously. These weights would be changed continuously by some specific learning methods during training so that the neural network gradually has the ability to solve problems.

2.1.2 Convolutional Neural Networks

Another widely used neural network type is the convolutional neural network (CNN), which is very good at processing data with a grid structure, such as images, videos, etc. A CNN is a neural network that uses convolution instead of matrix multiplication in at least one of its layers. A typical convolutional

neural network consists of convolution layer, pooling layer and fully connected layer.

The convolutional layer consists of a set of filters based on convolutional operations, which are also called "convolutional kernels". Mathematically, the convolution of two functions f, g can be expressed as:

$$(f * g)(n) = \begin{cases} \int_{-\infty}^{\infty} f(\tau)g(n-\tau)\mathrm{d}\tau, \text{ if cotinuous} \\ \sum_{\tau=-\infty}^{\infty} f(\tau)g(n-\tau), \text{ if discrete} \end{cases} \tag{2.7}$$

In general, filters or convolutional kernels use convolution to extract information from the input. The output of each filter combined with the input is the filtered image, which is also called 'convolved feature' or 'convolved feature map'.

Since CNNs are good at processing data with grid structure such as images, a two-dimensional image I is used as an example. The corresponding filter K is a two-dimensional matrix, and their convolution can be expressed as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.8}$$

, where i, j, m, n are pixels of I and K respectively [21].

Convolutional layers use a set of filters and convolution of the input image to generate a new filtered image. Each filter can be thought of as a specific feature extractor that is used to extract specific features from the input image. The larger the convolution value of a particular filter in a region of the input image, the closer that region is to the feature represented by that filter. The use of convolution allows us to use filters that are much smaller than the input image to detect and efficiently extract useful information. Also, within a convolutional layer, different regions of an image share the same set of filters, which means that the number of parameters of the neural network can be greatly reduced.

However, another problem that needs to be considered is that neighboring pixels in an image usually have similar values, so the values of the neighboring pixels in the output image of the convolutional layer are also extremely similar, which means that the output of the convolutional layer could often contain a lot of redundant information. Therefore, CNNs usually introduce pooling layers to adjust the output of the convolutional layer. Intuitively, pooling

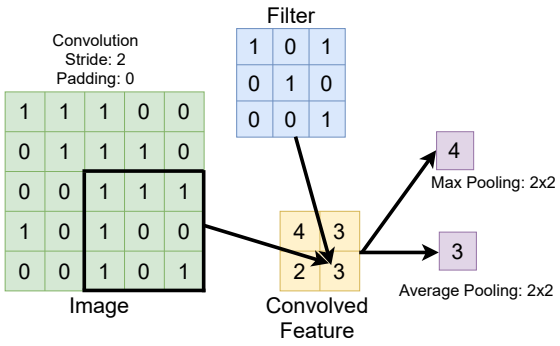


Figure 2.3: Convolution and Pooling of CNN

is just down-sampling, which reduces the number of values contained in the output of a convolutional layer by decreasing its dimensionality with a fixed rule. Continuing with the example of a two-dimensional image, the pooling layer is equivalent to a window that slides over the image with a certain stride, getting one value at a time within the window according to fixed rules. For example, Max-Pooling extracts the maximum value within the window and Average-Pooling extracts the average of all pixels in the window. The size of the window and the stride are flexible. A large window and stride means that more information of the image may be lost during pooling, but the output of the pooling layer also has a smaller dimension. The way the convolutional and pooling layers process the data can be referred to Figure 2.3.

The pooling operation also has a special property, namely "Translation Invariance", which means that for the pooling layer, the resulting output is constant when the pixels in the input image change slightly, which allows the robustness of the network to be enhanced.

The last part of the CNN is fully-connected layer. After dimensionality reduction of the data by convolutional and pooling layers, the final output image, i.e., two-dimensional data, is flattened to serve as input to fully connected layer, and the result of the whole network is outputted through the fully connected layer.

2.2 Backpropagation(BP) Algorithm

The BP algorithm was proposed in 1986 and the core idea is to compute the gradient efficiently by making the information from the loss function flow backwards along the network. The BP algorithm consists of two processes, forward propagation and backward propagation, which are iterative until the termination condition is satisfied. The following is an example of a three-layer neural network with n-dimensional input $\vec{X}(x_1, x_2, ..., x_n)$, m-dimensional actual output $\vec{Y}(y_1, y_2, ..., y_m)$ and the corresponding expectation $\vec{O}(o_1, o_2, ..., o_m)$ to introduce the derivation process of the algorithm. The two weights matrices of the network are $\vec{W}_1$, $\vec{W}_2$. That is, $\vec{W}_1$, $\vec{W}_2$ represent the weights to connect the input and the hidden layer, the hidden layer and the output layer of the network respectively. The two biases are b_1, b_2. For simplicity, we denote the output of the hidden layer by h, and the activation function f is an identity function.

- Forward Propagation

  The input information is passed through the input layer with the hidden layer and the result is outputted by the output layer. The output results and expectations are evaluated by an loss function, i.e.:

  $$h = f(z_1) = f\left(\vec{W}_1\vec{X} + b_1\right) = \vec{W}_1\vec{X} + b_1 \tag{2.9}$$

  $$Y = hz_2 = h\vec{W}_2 + b_2 \tag{2.10}$$

  Assume that the loss function is mean squared error:

  $$E = \frac{1}{2}\left(\vec{Y} - \vec{O}\right)^2 \tag{2.11}$$

- Back Propagation

  Now we take the information provided by the loss function and back propagate it along the network to update the gradient. The degree of weight update is determined by the learning rate η of the BP algorithm

  - Update a certain weight w_{2i} in $\vec{W}_2$, according to the chain rule:

    $$\frac{\mathrm{d}E}{\mathrm{d}w_{2i}} = \frac{\mathrm{d}E}{\mathrm{d}\vec{Y}}\frac{\mathrm{d}\vec{Y}}{\mathrm{d}w_{2i}} \tag{2.12}$$

$$\frac{\mathrm{d}E}{\mathrm{d}w_{2i}} = \left(\vec{Y} - \vec{O}\right) h \tag{2.13}$$

$$\triangle w_{2i} = -\eta \left(\vec{Y} - \vec{O}\right) h \tag{2.14}$$

– Update a certain weight w_{1j} in $\vec{W}_1$:

$$\frac{\mathrm{d}E}{\mathrm{d}w_{1j}} = \frac{\mathrm{d}E}{\mathrm{d}\vec{Y}} \frac{\mathrm{d}\vec{Y}}{\mathrm{d}h} \frac{\mathrm{d}h}{\mathrm{d}w_{1j}} \tag{2.15}$$

$$\frac{\mathrm{d}E}{\mathrm{d}w_{1j}} = \left(\vec{Y} - \vec{O}\right) \vec{W}_2 X \tag{2.16}$$

$$\triangle w_{1j} = -\eta \left(\vec{Y} - \vec{O}\right) \vec{W}_2 X \tag{2.17}$$

There are different training approachs based on the BP algorithm, for instance, Batch Gradient Descent(BGD), Stochastic Gradient Descent(SGD) and Minibatch Gradient Descent(MGD). The difference between them is the number of samples used for each gradient calculation. When using BGD we use the entire dataset to calculate the gradients and update the weights based on the average of the gradients of all samples. However, this approach is too costly when the dataset has many samples. SGD considers only one sample at a time and updates the network with the gradient of this one sample. Frequent updating of weights makes the convergence faster, but SGD oscillates back and forth at the minimum due to the constant change of losses. Another approach is MGD, where MGD divides the entire dataset into several fixed-size batches and updates the weights each time using the average gradient of all samples in a batch. This improves the training efficiency while maintaining some convergence stability [21].

The BP algorithm has low computational cost and good generality. Therefore, the training method based on BP algorithm has become the preferred method for training neural networks nowadays. However, the shortcomings of the BP algorithm itself are obvious.

– Local Minima

When there are multiple local minima, the gradient of the BP-based gradient descent algorithm is zero when it falls into a certain local minima, so the network performance cannot continue to improve by increasing the number of iterations.

– Gradient Explosion and Gradient Vanishing

It is the problem of gradients being multiplied repeatedly in deep neural networks, showing exponential increase or decrease and resulting in weights that cannot be reasonably updated. This is a problem in the design of BP algorithm itself and one of the main troubles in training large neural networks.

According to the above example of the derivation of the BP algorithm, for a certain weight w_{1j} in the first layer of a multilayer neural network without bias, the update amount is:

$$\triangle w_{1j} = -\eta \frac{\mathrm{d}E}{\mathrm{d}h_n} \frac{\mathrm{d}h_n}{\mathrm{d}z_n} \frac{\mathrm{d}z_n}{\mathrm{d}h_{n-1}} ... \frac{\mathrm{d}z_2}{\mathrm{d}h_1} \frac{\mathrm{d}h_1}{\mathrm{d}z_1} \frac{\mathrm{d}z_1}{\mathrm{d}w_{1j}} \tag{2.18}$$

$$\triangle w_{1j} = -\eta \frac{\mathrm{d}E}{\mathrm{d}h_n} f'(z_n) \vec{W}_n f'(z_{n-1}) \vec{W}_{n-1} ... f'(z_2) \vec{W}_2 f'(z_1) \tag{2.19}$$

It can be seen that when $f'(z_i)\vec{W}_i$, $i = (1, 2, 3, .., n)$ is greater than 1, the magnitude of weight updates becomes exponentially larger as the number of network layers increases. Conversely, when it is less than 1, the amount of weight updates decreases exponentially. That is, the so-called "gradient explosion" and "gradient vanishing".

2.3 Multi-Objective Optimization

The optimization problem can be understood as the process of selecting the optimal solution among multiple potential solutions for a problem based on a particular objective, according to [27], the definition of an optimization problem is:

Definition 1 *A Tuple* $(\Omega, f, \prec)$ *describes an optimization problem: with a search space* Ω*, a fitness function* $f : \Omega \longrightarrow \mathbb{R}$ *assigns a quality value to each solution, and a relation operator* $\prec \in \{<, >\}$.

A MOP is an optimization problem that contains more than one objective. Mathematically, a MOP can be represented as:

$$min\ \vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), ..., f_m(\vec{x}))$$
$$\vec{x} \in \Omega \subseteq R^n$$

where f_i represents objectives. In the real world, most optimization problems are multi-objective, and in general, the objectives of a MOP are conflicting. This means that the improvement of one objective may lead to the deterioration of another or more objectives. Therefore, it is often impossible to find a solution that are optimal in every objective simultaneously.

Therefore, algorithms for solving MOPs usually focus on finding approximations to the true pareto optimal set among all possible solutions. There are several concepts that need to be introduced. First, a feasible solution A in a population whose corresponding value in the objective space is not dominated by all other solutions that have been found, i.e., if there are no solutions for which all objective values are better than those of A. Then A can be called a non-dominated solution. Pareto optimal is then the set of non-dominated solutions in the whole search space, and it is also known as pareto front. The feasible solution corresponding to the pareto front in the space of decision variables is called pareto set. The relationship between the concepts above can be also seen in Figure 2.4.

2.4 Evolutionary Algorithms

EA, as one of the mainstream methods for solving optimization problems, is a stochastic global optimization method designed by simulating the evolutionary mechanism of organisms in nature. It incorporates the mechanisms of natural selection and genetics into the optimization process of the problem and derives the solution to the optimization problem through reproduction as well as selection. The process of EA is illustrated in Figure 2.5, and several concepts of EA are introduced in the following.

It is also important to note that in EA "population" refers to a set of individuals, and there can be duplicate individuals in a population. Each population

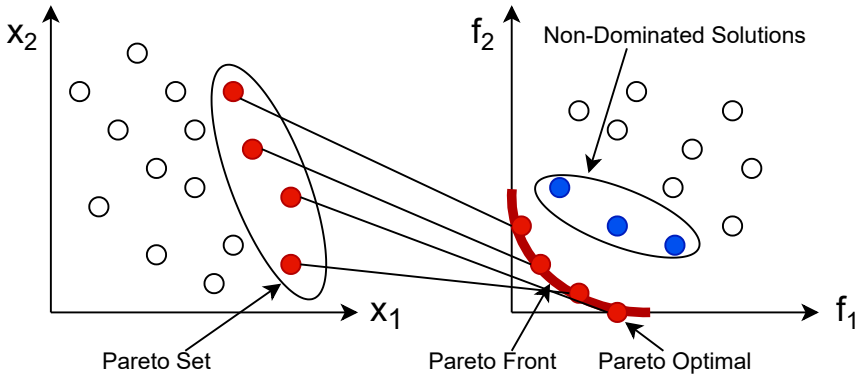


Figure 2.4: Examples of Pareto Set, Pareto Front and Non-Dominated Solutions

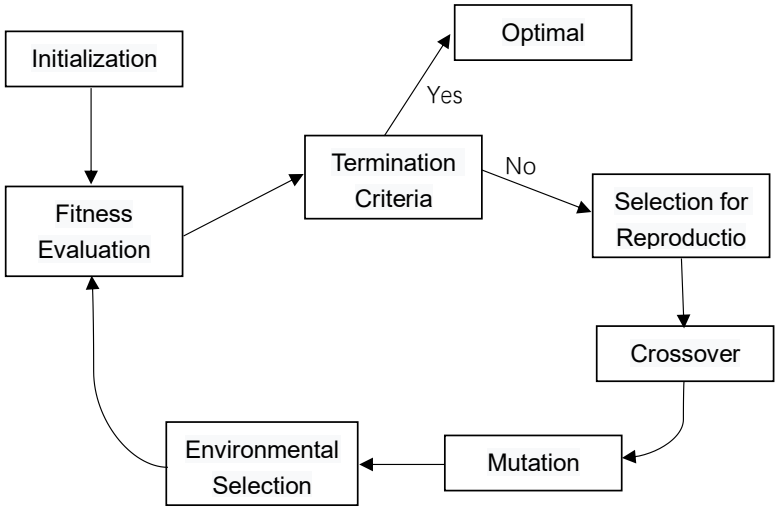


Figure 2.5: General Process of the Evolutionary Algorithm

corresponds a generation in the EA, and generally the size of the population does not change. A population can be also seen as a basic unit of EA, where crossover, mutation and selection operators take place.

2.4.1 Encoding

In order to use the EA to solve a real problem, it is necessary to first encode the objects in the real problem into expressions that the EA can handle. The possible solutions in the original problem are called phenotypes, and after encoding these possible solutions are called genotypes or chromosomes, which are also treated as individuals in the EA. Thus, encoding can also be understood as a mapping from the space of phenotypes to the space of genotypes. After encoding, the EA happens in the genotype space. As in nature, genotypes or chromosomes consist of multiple genes, and each genotype can be decoded into a possible solution [17].

Binary encoding is a common way of encoding. In binary encoding, chromosomes are strings of 1s and 0s. Another common encoding method is real value encoding, where the genes of the chromosomes are real value. Different problems often have different suitable encoding schemes. For example, for an optimization problem consisting of a series of binary decision problems, feasible solutions can be very easily representated by binary encoding. For the travelling salesman problem [28], the permutation of cities can be used as a way to encode. For the neural network weight training in this thesis, real value encoding is applied, and will be described in Chapter 4.

Although there is no fixed rule for the encoding of EA, we often consider that a reasonable encoding should have the following properties [27], [17]:

- All feasible solutions can be represented.
- Similar phenotypes are encoded to have similar genotypes.
- Similar genotypes have similar fitness values.
- The set of genotypes is closed under evolutionary operators.

2.4.2 Fitness Function

The differences between individuals in nature lead to different survival rates of different individuals, and natural selection allows some genes to be preserved

while others are eliminated. The fitness function in EA is the indicator and basis for this process. The fitness of each individual is measured by the fitness function, and the probability of individuals surviving to the next generation is determined by their fitness. Often the objectives and fitness functions of an optimization problem refer to the same thing, or the fitness function is just a simple transformation of the objectives.

For MOPs, the fitness of each individual of the EA is measured in terms of all fitness functions. Different versions of EA have different designs of how to calculate the overall fitness value. For example, the fitness function of the Multi Objective Genetic Algorithm (MOGA) [34] is

$$fitness(X) = 1 + nq(X)$$

, where $nq(X)$ denotes the number of individuals dominated by individual X.

2.4.3 Selection for Reproduction

According to genetics, the genes of each individual are passed on to the offspring, and the role of section for reproduction is to select individuals with good fitness to be the parents of the next generation. Together with other evolutionary operators, it ensures the continuous improvement of the quality of individuals.

Selection is usually based on fitness, i.e., individuals with better fitness also have a higher probability of being selected. To ensure that the algorithm does not fall into a local optimum, this process is not deterministic. An individual with poor fitness also has a chance to be selected, because it may be able to produce a better solution through other operators. Commonly used selection algorithms are roulette wheel selection and tournament selection, etc [27].

Roulette wheel selection is one of the most commonly used selection method, in which the selection probability of each individual is proportional to its fitness value, and the greater the fitness, the greater the selection probability. Tournament selection takes n individuals from the entire population and these individuals compete with each other on the basis of their fitness and the winner will be selected.

2.4.4 Crossover

In order to explore new feasible solutions, EA need to generate new chromosomes from old ones. Such operators are called variation operators, and crossover is one of it. Crossover operator generally generates new chromosomes based on the genes of two chromosomes. The new chromosome contains both genes from father and genes from mother. The logic behind crossover is that we want the two superior chromosomes to produce offspring that inherit the superior genes from both of them and thus have better fitness [17].

Crossover is a random operator. It is indeterminate which genes from which parent are derived by the offspring. There are many different methods of crossover, such as point crossover, uniform crossover, etc [27]. It is important to note that the old chromosomes used to generate offspring are not necessarily two. It is possible to generate new chromosomes using one or more than two chromosomes.

Simulated Binary Crossover (SBX), which is mainly used in EAs with real value encoding. Two selected individuals $X^1(x_1^1, x_2^1, ..., x_n^1)$ and $X^2(x_1^2, x_2^2, ..., x_n^2)$ generate their offspring $C^1(c_1^1, c_2^1, ..., c_n^1)$ and $C^2(c_1^2, c_2^2, ..., c_n^2)$ by the equation 2.20:

$$\begin{cases} c_i^1 = 0.5 \times [(1+\beta) \cdot x_i^1 + (1-\beta) \cdot x_i^2] \\ c_i^2 = 0.5 \times [(1-\beta) \cdot x_i^1 + (1+\beta) \cdot x_i^2] \end{cases} \tag{2.20}$$

,where β is determined by a random number μ between 0 and 1 according to the equation 2.21:

$$\beta = \begin{cases} (2\mu)^{\frac{1}{\eta+1}} \text{ , if } \mu \leq 0.5 \\ (2\,(1-\mu))^{\frac{1}{\eta+1}} \text{ , otherweise} \end{cases} \tag{2.21}$$

η could be any nonnegative real number, the larger the value of η the higher the probability of generating offsprings close to their parents [11].

2.4.5 Mutation

Another variation operator is mutation, which is the generation of new chromosomes by changing a small number of genes in one chromosome. The goal is to generate new genes to enrich the current gene pool. Like crossover operations, mutation is a random operation. Which gene of a chromosome is mutated to what value is usually done by a series of random operations. It is

also worth noting that the way mutations are made often depends on the way they are encoded. For example, for binary encoding, the inversion of a bit in a bit string can be considered a mutation. For some specific problems, mutations may change the length of the chromosome. For example, when optimizing the structure of a neural network, adding neurons or hidden layers.

Polynomial Mutation [12] is one of the most commonly used mutation methods for real value encoding. The specific procedure of which is shown below:

$$x_k = x_k + \sigma(u_k - l_k) \tag{2.22}$$

$$\sigma = \begin{cases} [2u + (1-2u)(1-\sigma_1)^{\eta_m+1}]^{\frac{1}{\eta_m+1}}, \text{ if u} \leq 0.5 \\ 1 - [2(1-u) + 2(u-0.5)(1-\eta_2)^{\eta_m+1}]^{\eta_m+1}, \text{ if u} > 0.5 \end{cases} \tag{2.23}$$

$$\sigma_1 = \frac{x_k - l_k}{u_k - l_k} \tag{2.24}$$

$$\sigma_2 = \frac{u_k - x_k}{u_k - l_k} \tag{2.25}$$

, where x_k is the variable to be mutated, u is a random value between 0 and 1, η_m is the distribution index, and u_k and l_k are the upper and lower boundaries of x_k.

2.4.6 Environmental Selection

The role of environmental selection is to select individuals to be retained into the next generation. Since the population size is usually fixed, it is important to select suitable individuals among the individuals of the new current population and their offspring to be retained in the next generation. Similar to selection for reproduction, environmental selection is carried out using methods such as roulette wheel selection, i.e., it tends to select individuals with higher fitness.

2.4.7 Termination Criterion

The termination criterion defines when the EA stops, and it can be divided into two types. One is when the optimal solution to the problem is known, in which case the discovery of the optimal solution can be the termination criterion of EA. However, although EA are global search algorithms, there is no guarantee that the optimal solution will be found. In addition, for most optimization problems, the optimal solution is not known before, or we do

not care about the true global optimum or pareto front. Sometimes what we really want is to find an acceptable solution within an acceptable time limit. Therefore EA usually use other criteria as their stopping conditions, such as:

- The fitness of an individual exceeds a predetermined threshold.
- The algorithm is executed for a fixed amount of time.
- The fitness function is executed a fixed number of times.
- The improvement in fitness over consecutive generations is below a predetermined threshold.

2.5 Performance Metrics

In this thesis, the problems handled by neural networks are all multi-classification problems. The specific datasets will be presented in Chapter 4. Several evaluation criteria for the multiclassification tasks involved in the experiments will be presented here.

- Hypervolume

  Hypervolume was first proposed by E. Zitzler and L. Thiele [48] as a metric to evaluate the quality of the solution set of a multi-objective optimization algorithm. It represents the volume of the hypercube enclosed by the individuals in the solution set and the reference points in the objective space. Thus different reference points can produce different hypervolume values. It is noted in [48] that when the solution set A is superior to another solution set B, the hypervolume of A will also be larger than the hypervolume of B.

- Categorical Cross Entropy

  Categorical cross entropy is one of the common metrics used by neural networks when dealing with multi-classification problems. After applying the Softmax activation function, the output of the neural network predicts the probability that the input data belongs to each class. For a multi-classification problem, the probability of the input belonging to two classes is assumed to be p and $1 - p$, respectively. Then the cross entropy can be calculated as:

$$L_i = -\sum_j \vec{O}_{i,j} log(\vec{Y}_{i,j}) \tag{2.26}$$

, where O refer to the expected outputs, Y are the network predictions, i denotes the sample and j denotes the class.

Categorical cross entropy is used to evaluate the difference between the probability distribution of the current output and the actual probability distribution, the smaller the value of categorical cross entropy, the smaller the difference between the output and the expectation.

Before introducing the following metrics, the meaning of several abbreviations needs to be explained. "True Positive (TP)" indicates the number of positive samples classified as positive correctly in the experiment. "True Negative (TN)" stands for the number of negative samples correctly classified as negative. "False Positive (FP)" means the number of negative samples were incorrectly classified as positive. "False Negative (FN)" means the number of true samples that are incorrectly classified as negative.

- Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.27}$$

- Precision

$$Precision = \frac{TP}{TP + FP} \tag{2.28}$$

- Recall

$$Recall = \frac{TP}{TP + FN} \tag{2.29}$$

All the above described are the calculation methods of each index when dealing with the binary classification problem. However, when it comes to multi-classification problems, there are different ways to calculate each metric. Take

the method provided in Sklearn as an example, "micro" calculates metrics globally by counting the total true positives, false negatives and false positives. "macro" calculates metrics for each label and then calculate their unweighted mean. Similarly, "wighted" calculates metrics for each label and then calculate their mean using the number of true samples for each label as weights [4].

Different metrics assess the goodness of the results from different aspects. Accuracy is used to measure the percentage of samples that are correctly predicted, precision measures how many of the predicted positive samples are actually positive, and recall measures how many of the true positive samples were successfully predicted. Most of the time, a single metric does not give a complete picture of the model's performance. For example, a model that can only accurately classify positive samples can achieve a high accuracy rate on a dataset containing mostly positive samples. However, applying this model to a problem where determining a positive sample as negative can have serious consequences can be problematic. Precision is a good metric when the cost of FP is high. And Recall is more applicable when the cost of FN is high, as can be seen from 2.29.

3 Related Work

In recent years, breakthroughs in large-scale EA have made it possible to train neural networks using EA. EA, as one of the most widely used EA, have also been used in several ways to improve the performance of neural networks. This section provides a survey of the combination of EA with neural networks and several techniques designed for EA to solve large-scale optimization problems.

3.1 Neural network hyperparameters

The hyperparameters of neural networks have a huge impact on neural networks. The learning rate determines the convergence speed and accuracy, and the choice of hyperparameters is usually highly dependent on intuition and experience. Although some methods of scheduling learning rates have been proposed, for example, time-dependent scheduling [38] and adaptive scheduling [33][39]. However, these methods themselves also contain hyperparameters, which makes them not optimal for different network architectures and different data sets. Thus, the idea of using EA to optimize neural network hyperparameters was born.

In [25], the EA is used to optimize the learning rates of a neural network with good results, where each layer of the neural network has a different learning rate at each training step, and these learning rates are used as chromosomes in the population of the EA for evolutionary adaptation. [24] proposes an approach that combines the BP algorithm with a EA, called "learning-rate-optimizing genetic back-propagation (LOG-BP)". Each population of the EA consists of multiple neural networks, where the learning rate and weights of each neural network are encoded as an individual. At each step, all neural networks in the current population are run in parallel, and these are then evaluated and selected based on fitness function and selection operator, with the

best individuals being retained into the next generation. For each individual, the learning rate is changed by the mutation operator, while the training of the weights remains the responsibility of the BP algorithm. Furthermore, benefiting from the global search capability of EA, the BP algorithm can get rid of local optimality.

The architecture of a neural network can also be considered as one of its hyperparameters, which includes the number of hidden layers, the number of units each hidden layer contains, the type of neural network activation function and, in the case of a CNN, the size of the kernel size. A proper architecture gives the network the ability to capture the information required in the input, to process the information appropriately to achieve a high accuracy, and to ensure the training efficiency of the network.

In [29], Frank H. F. Leung, H. K. Lam, S. H. Ling, and Peter K. S. Tam proposed an improved EA to downsize the architecture of the neural network. They encode the intra-layer connections of feedforward neural networks into chromosomes, where each connection is a binary variable with 1 representing a maintained connection and 0 representing a deleted connection. According to the results they show, a fully connected network can be successfully turned into a partially connected network by learning. [7] points out the complex relationship between different hyperparameters and network structure, and the optimal hyperparameters of different architectures are not the same. [16] proposed "Evolutionary Deep Networks (EDEN)". In this paper, EA are used to optimize the convolutional neural network architecture and its associated hyperparameters. Each EDEN chromosome is composed of two genes: learning rate and network architecture. They preset different types of neural network layers, and architecture genes represent the order in which these layers are composed. EDEN performs well by using different datasets for validation.

NeuroEvolution of Augmenting Topologies (NEAT) [43] is a well-known approach to optimize the architecture of neural networks using evolutionary algorithms, and here we will mainly introduce its encoding and the corresponding evolutionary operators. NEAT encodes the neuron nodes and connections of a neural network in the manner shown in Figure 3.1. For a three-layer neural network, each layer contains 1, 2, and 1 neuron nodes, respectively. The genotyp of this network is $[(0 : True, 1 : True, 2 : True, 3 : True)]$, where $1, 2, 3, 4$ represnet the number of connctions shown in the connection set of the Figure

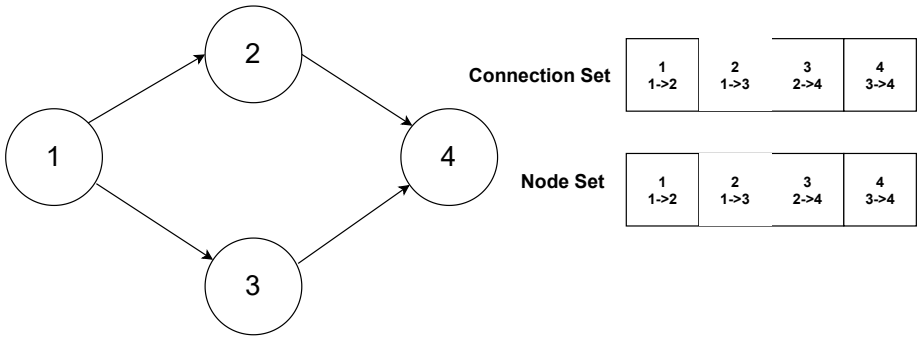


Figure 3.1: Encoding in NEAT

3.1. And $Ture$ means that this connection exists in the current network, e.g. this connection is active.

With respect to the crossover operator, NEAT selects two individuals in the population, and the crossover can be divided into four cases. As shown in Figure 3.2, where the connections in red are deactivated ($False$):

- If the two individuals have the same connection, one of the connection is selected randomly (Node 1, 2 ,3).
- If one of them possesses a connection that the other does not, this connection is kept to the offspring (Node 4, 5, 6, 7).
- If one has extra connections, such as the Node 9, 10, then whether to keep these connections to the offspring depends on the relative fitness of that individual, as in the figure, the last two connections of $Parent$ 2 are extra connections and are kept to the offspring because $Parent$ 2 has a higher fitness than $Parent$ 1.
- If the state of a connection in at least one of the two parent individuals is $False$, then the state of that connection in the offspring is set to $False$ (Node 2, 3).

As illustrated in Figure 3.1, there are two cases of mutation, i.e., connection mutation and node mutation. To perform an the connection mutation, two nodes are selected among all nodes of an individual and a connection is generated between these two nodes. For node mutation, an active connection (a connection with the value of $True$) of the individual is selected and a node

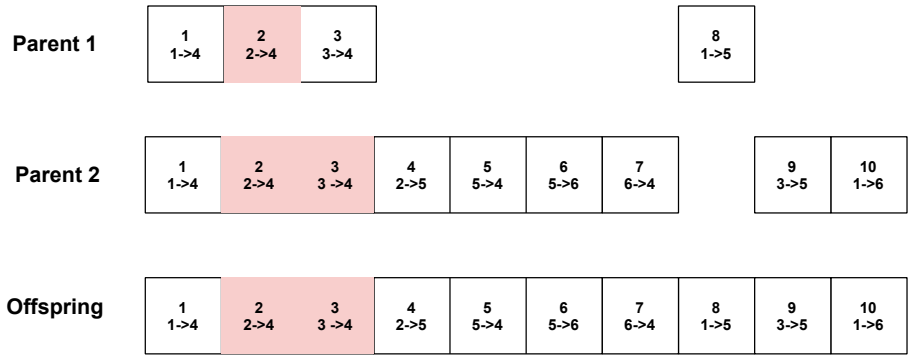


Figure 3.2: Crossover in NEAT

is added to this connection. As in the figure, the connection 3 (3 -> 4) is selectied and a node 6 is added between the nodes 3 and 4.

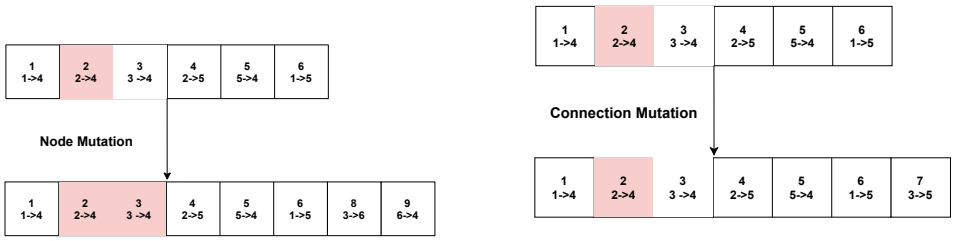


Figure 3.3: Mutation in NEAT

Based on NEAT, HyperNEAT uses a hypercube-based indirect encoding approach for the evolution of large neural networks, more details can be found in [42].

3.2 Neural network weights

There have been several cases of using EA to train network weights:
[22] trained neural networks with a small number of parameters using simple EA and showed that networks trained using EA outperformed the BP algorithm. [18] attempts to train neural networks with a large number of parameters through EA through improved coding schemes. They regard each

chromosome as a two-dimensional array composed of n genes, and each gene is an array representing a layer of the network and contains all the information of that layer. This means that for the convolutional layer, a gene contains all the values of the convolutional filter, and for the fully connected layer, it represents all the ingoing weights of the layer. This approach was applied in a CNN with 4540 parameters for Mnist and a CNN with 291,870 parameters for Cifar10, but both have an unsatisfactory accuracy of around 40%. These two datasets are also used in this thesis, and are described in Chapter 4 in detail.

An additional benefit of using EA to train a neural network is that the training of the network can be treated as a MOP. A neural network performing a particular task can be thought of as a sparse multi-objective optimization problem, meaning that most of the decision variables in the resulting solution to the problem are zero. For example, in the image classification problem, the neural network only needs to be able to recognise a few specific features of the input image in order to successfully perform the classification task.

In [44], they propose a population initialization method and a offspring generation method with masks in such a way that a significant number of decision variables in each individual always have a value of 0. These two proposed approches are combined with NSGA-II to optimize the network weights. The objectives of optimization is the error rate and complexity of the neural network. Taking the initialization method as an example, each individual in the initialized population is determined by a real-valued variable and its corresponding mask, e.g. an individual can be calculated by a real value vector d and its mask like:

$$d \cdot mask = (1, 2, 3, 4, 5) \cdot (1, 0, 0, 0, 1) = (1, 0, 0, 0, 5)$$

Assume that the number of decision variables is D and the population size is N. The first step of initialization is to generate a $D \times D$ random matrix Dec and an $D \times D$ identity matrix as mask. A population Q is obtained by multiplying Dec and the mask row by row. The different levels of non-dominated solution set in Q are derived by non-dominated sorting. This is to determine the importance of each variable in this optimization problem, Q is not the final initialized population.

At the second step, first a random matrix A of size $N \times D$ is generated, and a $N \times D$ matrix mask consisting of zeros. Starting from the first individual,

n random picks are made, each time picking two decision variables in different positions, n is not necessarily the same for different individuals. The importance of these two variables is compared according to the number of the non-dominated solution set obtained from the first step. The final initialized population is calculated by multiplying each row of A and its corresponding mask. Namely, the values in the positions selected during n random picks for each individual are the random values in the corresponding indices of matrix A, and the values of other position are 0.

This method was successfully applied on neural networks consisting of 321 variables to 1241 variables dealing with a binary classification problem. The method performs weaker than the gradient descent-based training method in terms of error rate in the output, but EA can significantly reduce the complexity of the network with a guaranteed accuracy to a certain extent.

The encoding method of the EA also plays a very important role. In [23], the connections between each neuron of the neural network and the weights of each connection are encoded into two different adjacency matrices, that are optimized simultaneously during the optimization process, in order to optimize the network architecture as well as the weights at the same time. The method was successfully applied to a three-layer neural network containing up to 10 neurons in the hidden layer, which means that a appropriate compromise solution can be chosen in the non-dominated solution set derived by the algorithm.

EA can also work with the BP algorithm to train the network weights. [8] use the results of EA as the initial values of the network weights, and then continue to use the BP algorithm to further train the network. In this combination, EA serves as a means of pre-training, providing better initial values so that BP can obtain better results. In addition, the global search space of EA also helps to free BP from local optima. According to the experiments, the BP algorithm that applies the initial values provided by EA achieves higher results. Similarly, [32] first uses the BP algorithm to train the neural network to shorten the convergence time, before using EA to find the global optima. The examined network contains 9 weights and according to the results, EA can significantly further optimize the results obtained by BP.

3.3 Multi-Objective Evolutionary Algorithm

Multi-objective EAs refer to the EAs proposed for solving MOPs. For instance, [34] [19] and [13]. Non-dominated Sorting Genetic Algorithm (NSGA-II) [14] is a well-known multi-objective evolutionary algorithm based on pareto optimal and also one of the algorithms used in Chapter 5. Before introducing the specific process, two special operators involved in NSGA-II are introduced, namely, fast non-dominated sorting and crowding distance assignment.

- Fast Non-dominated Sorting

  NSGA-II performs a non-dominated sorting to determine each level of non-dominated fronts based on the fitness of each individual as evaluated by the multi-objective fitness function, i.e. non-dominated fronts, as shown in the Pseudo-code 1. In 1, N_p is the domination counter, which is incremented each time a solution is encountered that can dominate the current solution. The first level of non-dominated solutions are those with $N_p = 0$, the solutions in the current population that are not dominated by any other individual. In the next step we ignore the set of first-level non-dominated solutions and then find all remaining solutions in the same way until all individuals in the current population are classified as non-dominated fronts.

- Crowding Distance Assignment

  In NSGA-II, the crowding distance of each individual in each non-dominated front is also calculated and used as the basis for survival selection and selection for reproduction. the crowding distance of the individuals within a non-dominated front can be computed according to the process illustrated by Pseudo-code 2. In 2, $obj(F_i[j])$ represents the value of the objective obj for individual j in non-dominated front F_i. $Max(objFun())$ and $Min(objFun())$ indicate the maximum and minimum values of obj respectively.

  For the bi-objective optimization function, the crowding distance of an individual can be interpreted as the maximum rectangle that the individual can generate without touching other individuals within the objective space. The greater the crowding distance of an individual, the more different it is from the rest of the population. The selection of individuals with greater crowding distance in subsequent selection operators will help to ensure the divisity of the gene pool.

The process of NSGA-II is shown in Figure 3.4, based on the above mentioned operators.

With respect to the selection for reproduction operator, NSGA-II uses binary tournament selection, which means that only two individuals are taken from the population at a time. In this case, tournament selection is based on the rank of the non-dominated fronts and the crowding distance of an individual. If the non-dominated front number of individual A is lower than that of individual B, i.e. $F_A < F_B$, then A is selected. If A and B are in the same rank of non-dominated front, then their crowding distance is compared, and the one with the greater crowding distance wins. Based on the selected parents, SBX and polynomial mutations are used to produce offspring.

In NSGA-II elitism selection is also used as the environmental selection oprator, which is illustrated in the following:

As shown in the Figure 3.5, assume the population size is N, and the currently we are at the $t-th$ generation. The process of elitism selection can be expressed as the following procedure:

Step 1 Combine the generated new offspring Q_t with its parent population P_t into R of size $2N$

Step 2 Perform Non-Dominated sorting on R to obtain the non-dominated solution sets $F_i(i = 1, 2, ..., n)$

Step 3 Calculate the crowding distance of each non-dominated solution set $Z_i(i = 1, 2, ..., n)$

Step 4 Add the non-dominated solution set F to the next parent population P_{t+1} in order according to the non-domination rank

Step 5 If the size of P_{t+1} will exceed N when adding $F_m(1 \leq m \leq n)$, then add the individuals in $F_m(1 \leq m \leq n)$ to P_{t+1} in order of largest to smallest crowding distance, so that the size of P_{t+1} reaches N

3.4 Large-scale evolutionary algorithm techniques

EA suffers from the curse of dimensionality, so it has been a long-standing research hotspot in the field of evolutionary computation to use EA to efficiently

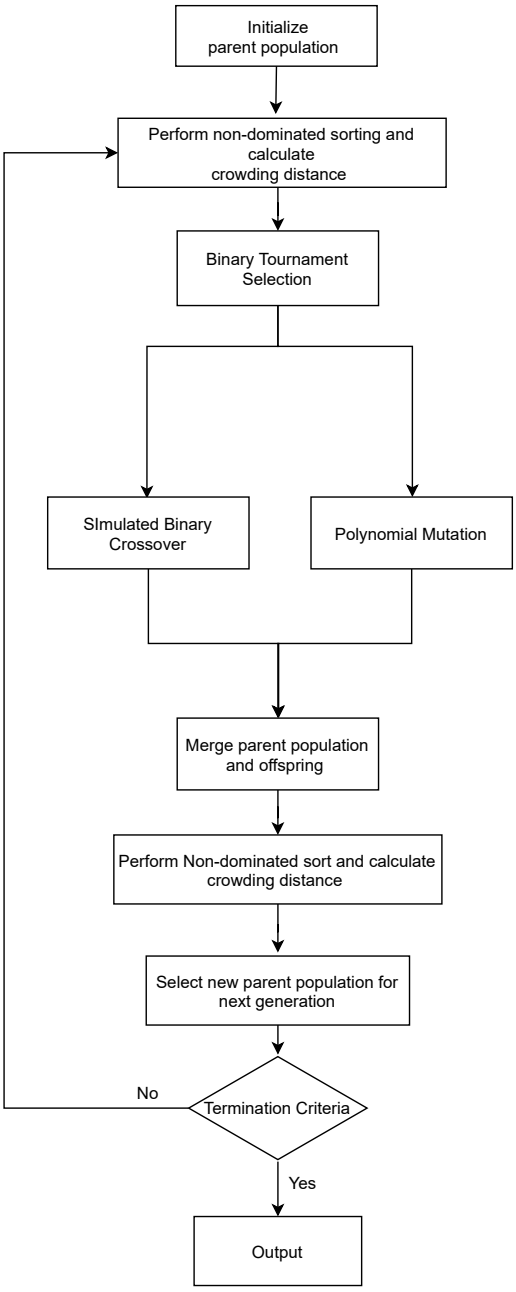


Figure 3.4: The Process of NSGA-II

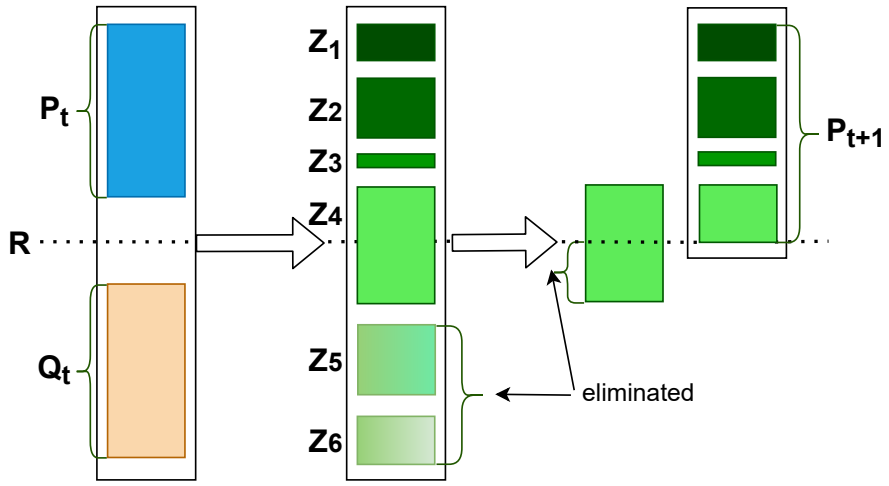


Figure 3.5: Elitism selection

solve large-scale optimization problems containing huge numbers of decision variables. The early research has centered on large-scale single-objective optimization problems, and many algorithms as well as techniques have been proposed. Most of them are based on the idea of Cooperative Coevolution(CC), which is to divide a large-scale problem into a set of smaller problems by dividing the variables in an individual into multiple groups. The divided subproblems are called "species", and each species is supposed to be a part of the solution. The population of each species is often referred to the "subpopulation". CC optimizes one individual species at a time, while keeping the remaining species unchanged, and the individuals in each subpopulation combine with randomly selected individuals from other subpopulations to form the complete solution, which will be evaluated by the fitness function. Some proposed methods and applications based on CC can be found in [45][9][37].

In comparison, large-scale MOP is more difficult to handle because in single-objective problems, the operations on the decision variables in the EA is based only on a single objective, whereas in MOPs, the relationships of decision variables are extended to multiple objectives, different objectives need to be taken into account simultaneously in the optimization process. In [41], a random-

based dynamic grouping (RDG) strategy is proposed as the basis for grouping in CC, where the size of the grouping of decision variables can be randomly selected among a set of predefined values. Thus, the decision variables are randomly grouped into groups of different sizes. [31] also uses the advanced grouping strategy to solve large-scale MOPs using CC.

But CC is not the only way to solve large-scale MOP. In MOPs, each species has its own non-dominated solution set, and to finally piece together the individual non-dominated solution sets out of the true pareto front. This may lead to a more complex and even intractable problem. Therefore, there are many methods that are not designed based on CC, several of them are described below.

In 2016, H. Zille, H. Ishibuchit, S. Mostaghim and Y. Nojima presented three mutation operators designed for problems containing large-scale variables, multiple objectives, i.e. Linked Polynomial Mutation, Grouped Polynomial Mutation, and Grouped and Linked Polynomial Mutation [46]. All three methods improve on the polynomial mutation.

In Linked Polynomial Mutation, all mutated variables have the same amount of change. That is, all variables in the Polynomial Mutation share the same u. The variables optimized in the second proposed approach, Grouped Polynomial Mutation, are first grouped and then a group is uniformly selected in which all variables are mutated according to the standard Polynomial Mutation. The last mutation approach, Grouped and Linked Polynomial Mutation, combines the first two approaches, in which all variables are first grouped and then the same amount of changes is carried out on all variables within a uniformly selected group. Two different grouping mechanisms are used, i.e. Ordered Grouping and Differential Grouping [36], one based on the absolute values of variables, and the other treats the grouping of variables as a single-objective optimization problem by locating those variables that interact with each other and then grouping them into a single group. Intuitively, although this brings additional cost, the variables can be grouped more reasonably, i.e., according to a objective. However, according to the experimental results, Ordered Grouping achieves better results in most problems. Therefore, the paper also identified how to better combine Differential Grouping with Linked Mutation as a future research point.

In general, all three mutation methods aim to maintain the relationship between the interacting variables in different ways, thus preventing the degra-

dation of the solution quality caused by the disruption of this relationship. The proposed approaches perform well on problems with 1000 variables and 2 objectives.

In 2019, H. Zille and S. Mostaghim proposed another technique designed for large-scale MOPs, which is called "Linear Combination Search Mechanism". This technique works together with other EAs. The first step is to let any of the EAs run for a period of time to obtain a non-dominated solution set. After a period of learning, the solutions in the non-dominated solution set should already be at a certain quality by this time. The proposed Linear Combination Search Mechanism is built on the non-dominated solution set obtained from the first step. The paper argues that the quality of solution can be further improved by a proper linear combination of each solution in this set. Therefore, in the second step, i.e., Linear Combination Search Mechanism, the optimization problem is transformed from the original optimization problem containing a large number of variables to finding the linear combination coefficients of the solutions in the non-dominated solution set provided by the previous round of evolutionary algorithms at a certain point in time.

This means that the dimensionality of the optimization problem is then transformed to the size of the non-dominated solution set obtained in the previous round as well. As Figure shows, suppose that the non-dominated solution set X we obtained from the first round contains three solutions, i.e., $X^1 = (x_1^1, x_2^1, ..., x_n^1), X^2 = (x_1^2, x_2^2, ..., x_n^2), X^3 = (x_1^3, x_2^3, ..., x_n^3)$. Then the dimension of the optimization problem to be solved by the Linear Search Mechanism is 3, and the non-dominated solution set Y obtained from the Linear Combination Search Mechanism contains two solutions, i.e. $Y^1 = (y_1^1, y_2^1, y_3^1), Y^2 = (y_1^2, y_2^2, y_3^2)$. Then the solutions generated by X multiplying Y, i.e.:

$$Y^1 \cdot X = y_1^1 X^1 + y_2^1 X^2 + y_3^1 X^3 \qquad Y^2 \cdot X = y_1^2 X^1 + y_2^2 X^2 + y_3^2 X^3$$

, are integrated into the initial population of the next round[47].

The Linear Search Mechanism can be considered as a problem transformation ideology, so this technique is compatible with any evolutionary algorithm, and it can also use any evolutionary algorithm to optimize the combination coefficients. According to the evaluation of the algorithm on problems with 30-514 decision variables and 2-5 objectives, this technique can improve the performance of existing methods.

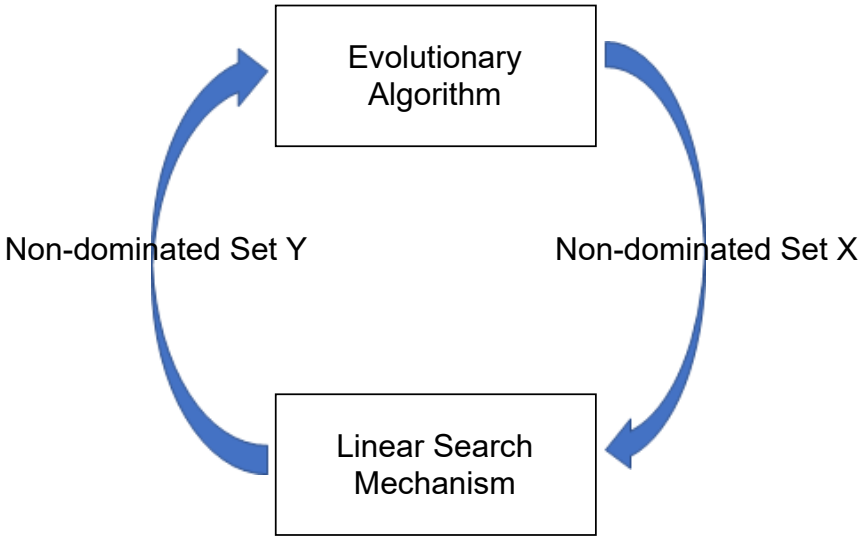


Figure 3.6: Linear Search Mechanism

Algorithm 1 Non-Dominated Sorting

Require: P: Population
Ensure: F: non-dominated fronts

```
F = [ ]
for p in P do
    S_p = [ ]
    n_p = 0
    for q i ∈ P do
        if p < q then                                   ▷ If p dominates q
            S_p.append(q)    ▷ Add q to the set of solutions dominated by p
        else if p > q then                              ▷ If q dominates p
            np+ = 1                ▷ Increment the domination counter of p
        end if
    end for
end for
if n_p == 0 then                                ▷ p belongs to the first front
    p_rank = 1
    F_1.append(p)
end if
F.append(F_1)
i = 1                                                   ▷ i: the front counter
while F[i] do
    Q = [ ]                                               ▷ Q: the next front
    for p in F[i] do
        for q in S_p do
            n_q− = 1
            if n_q == 0 then                ▷ q belongs to the next front
                q_rank = i + 1
                Q.append(q)
            end if
        end for
    end for
    i = i + 1
    F_i = Q
    F.append(F_i)
end while
```

Algorithm 2 Crowding Distance Assignment

Require: F_i: non-dominated fronts
Require: M: objectives
1: $nLen = len(I)$ ▷ Number of solutions in F_i
2: **for** i *in* F_i **do**
3: $i.distance = 0$ ▷ Initialize distance
4: **end for**
5: **for** obj *in* M **do**
6: $F_i = sort(F_i, obj)$ ▷ Sorting in ascending order according to the objective obj
7: $F_i[0] = F_i[nLen - 1] = \infty$ ▷ Set the distance to infinity for the first and last individual
8: **for** j in range(1, nlen-2) **do**:
9: $F_i[j].distance = F_i[j].distance + (obj(F_i[j + 1]) - obj(F_i[j - 1]))/(Max(obj()) - Min(obj()))$
10: **end for**
11: **end for**

4 Implementation

The Chapter 2 described the shortcomings of the BP algorithm, the definition of the multi-objective optimization problem and the specific procedure of EA. The main goal of this thesis is to evaluate the applicability of the large-scale multi-objective evolutionary algorithm for training neural networks. This chapter describes the data set to be used for the experiments in Chapter 5, the neural network architecture, as well as the design of the algorithm used for the experiments.

4.1 Datasets And Networks

The number of neurons in the input layer and output layer of a feedforward neural network is usually determined by the data and the specific optimization problem. In CNNs, although the number of parameters in the convolutional and pooling layers is not determined by the input data, it can affect the size of their output, which in turn affects the number of parameters in the first fully connected layer at the end of the network after the output of the previous layer is flattened.

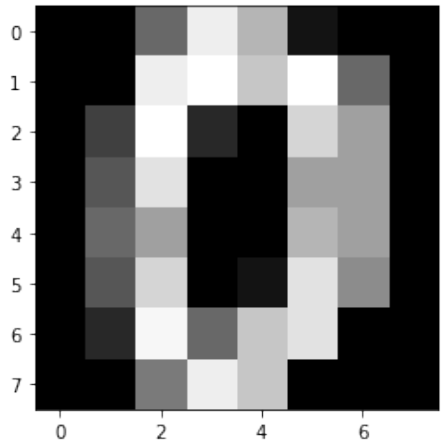
In addition, the archtectures of the network also determines the number of variables to be processed by EA, such as the number of hidden layers and the number of neurons per layer in the feedforward network, the number of convolutional layers and the size of each filter layer in CNNs.

As mentioned before, the performance of EA can be affected by dimensions of the problem to be optimized. We chose datasets with different complexity and neural networkswith different structures to verify its performance. Three datasets are used, inorder of complexity, Load Digits[3], Mnist[5], and Cifar10[26].

Load Digits is a small dataset of handwritten digits, in which each image is a grayscale image, and represents a number between 0 and 9. Each image

consists of 8×8 pixels, where each pixel is an interger in the range between 0 and 16. Similarly, one of the most well-known datasets in the field of deep learning, the Mnist dataset consists of a training set with 60,000 samples and a test set with 10,000 samples. Each sample is a grayscale image consisting of 28×28 pixels, and also represents a handwritten digit between 0 and 9.

Cifar10 is another symbolic dataset. Like Mnist, Cifar10 contains 50,000 training samples and 10,000 validation samples. But the samples in Cifar10 are more complex. The training set is stored in five batches, each containing 10,000 images. Each image contains 3072 pixels, and consisting of three channels.The first 1024 elements are the red channel, the next 1024 elements are the green channel, and the last 1024 elements are the blue channel. Each element in Mnist and Cifar10 ranges from 0 to 255.



$$\begin{bmatrix} 0 & 0 & 5 & 13 & 9 & 1 & 0 & 0 \\ 0 & 0 & 13 & 15 & 10 & 15 & 5 & 0 \\ 0 & 3 & 15 & 2 & 0 & 11 & 8 & 0 \\ 0 & 4 & 12 & 0 & 0 & 8 & 8 & 0 \\ 0 & 5 & 8 & 0 & 0 & 9 & 8 & 0 \\ 0 & 4 & 11 & 0 & 1 & 12 & 7 & 0 \\ 0 & 2 & 14 & 5 & 10 & 12 & 0 & 0 \\ 0 & 0 & 6 & 13 & 10 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.1: '0' in Load Digits

In the three datasets used, each sample has its own corresponding label as the expectation in neural networks, which is the actual value of the handwritten digit for Load Digits and Mnist or the true category for Cifar10. The data in all the three datasets can be classified into 10 classes, which means that the task to be handled by the neural network is a supervised learning multi-classification task.

Four different neural networks including two feedforward networks as well as two CNNs are used. The feedforward neural networks, i.e. Network I 4.1 and Network II 4.2 contain one and two hidden layers consisting of 128 neurons, respectively. The specific architectures of the used CNNs are shown by Tables 4.3, 4.4.

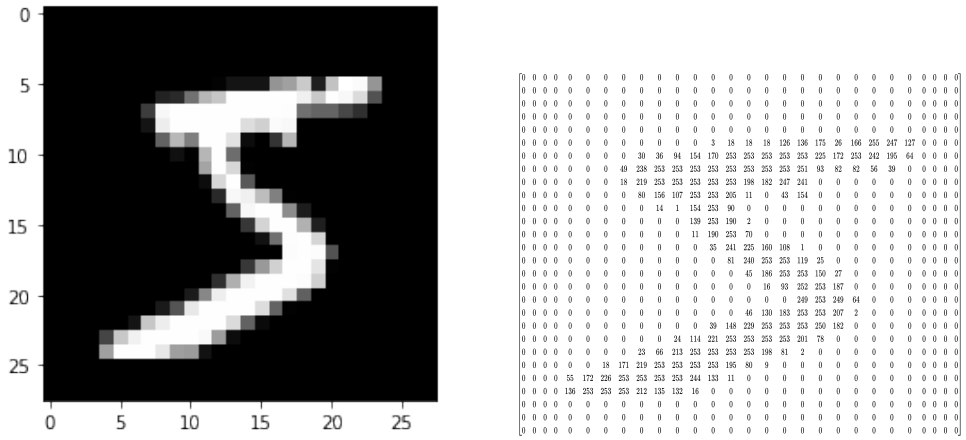


Figure 4.2: ’5’ in Mnist

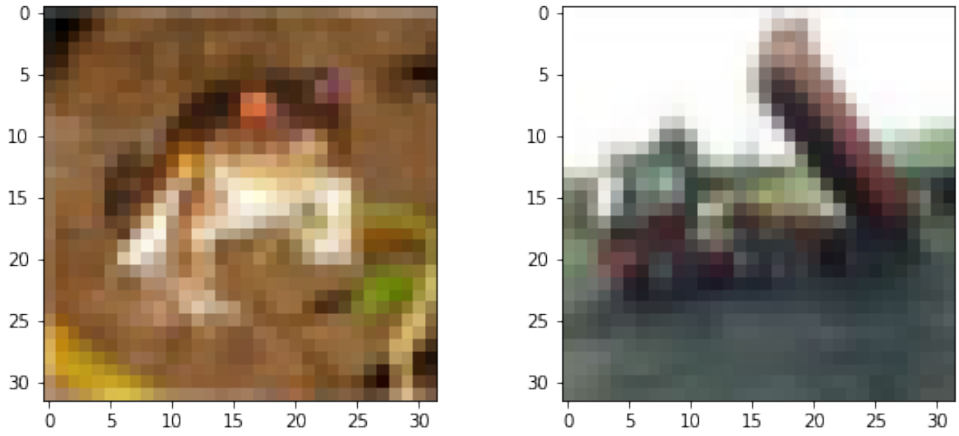


Figure 4.3: Samples in Cifar10

Table 4.1: Network-I

Layer(type)	Output shape Load Digits	Output shape Mnist	Output shape Cifar10
input_1 (InputLayer)	(None, 64)	(None, 784)	(None, 3072)
dense	(None, 128)	(None, 128)	(None, 128)
dense_1	(None, 10)	(None, 10)	(None, 10)
Total Parameters:	9,610	101,770	394,634

NetworkIV are taken from [18]. The other three are custom designs. Since the datasets used are all 10 classification problems, the last layer of each network, i.e. the output layer, consists of 10 neurons. Furthermore, it should be noted that in Table 4.3 the global average pooling layer is used, which means that each feature map is converted to a real number regardless of the size of of it. The downside of this operation is that compressing the input feature maps into a real number results in a loss of some of the information, and the amount of information lost increases as the input size increases. But the consequent benefit is the huge reduction in the number of parameters of the network. Thus the performance of EAs in networks containing a relatively small number of parameters can be observed and tested.

In the tables, the layer 'dense' is referred to the fully connected layer and "None" is the batch size in the experiments, which is the number of samples used to verify each individual fitness at each time, and the specific parameter settings will be introduced in Chapter 5. For networks 4.1 and 4.1, the number in the second position is the size of the output of each layer, i.e., the number of neurons in each layer. For networks 4.3 and 4.4, the last three positions in the parentheses are the size of the output of each layer, where the last position refers to the number of channels of the output, which corresponds to the number of filters in the current convolutional layer. For the second row of Load Digits in Table 4.4, e.g. $(None, 8, 8, 6)$, "None" is the position for batch size, $(8, 8, 6)$ is the size of the output after processing the original samples in the first layer, and $(8, 8)$ is the size of each output image. This means that the first layer contains 6 1×1 filters.

These three datasets are all about multiple classification tasks of images, and they all fit feedforward neural networks and CNNs. With these three datasets, the performance of the combination of evolutionary algorithms and neural networks can be experimented and observed from multiple aspects as the pa-

Table 4.2: Network-II

Layer(type)	Output shape Load Digits	Output shape Mnist	Output shape Cifar10
input_1 (InputLayer)	(None, 64)	(None, 784)	(None, 3072)
dense	(None, 128)	(None, 128)	(None, 128)
dense_1	(None, 128)	(None, 128)	(None, 128)
dense_2	(None, 10)	(None, 10)	(None, 10)
Total Parameters:	26122	142,542	411,146

Table 4.3: Network-III

Layer(type)	Output shape Load Digits	Output shape Mnist	Output shape Cifar10
input_1 (InputLayer)	(None, 8, 8, 1)	(None, 28, 28, 1)	(None, 32, 32, 3)
conv2d (Conv2D)	(None, 6, 6, 8)	(None, 26, 26, 8)	(None, 30, 30, 8)
average_pooling2d (AveragePooling2D)	(None, 3, 3, 8)	(None, 13, 13, 8)	(None, 15, 15, 8)
conv2d_1 (Conv2D)	(None, 1, 1, 16)	(None, 11, 11, 16)	(None, 13, 13, 16)
global_average_pooling2d (GlobalAveragePooling2D)	(None, 16)	(None, 16)	(None, 16)
dense (Dense)	(None, 10)	(None, 10)	(None, 10)
Total Parameters:	1384	1,384	1528

Table 4.4: Network-IV

Layer(type)	Output shape Load Digits	Output shape Mnist	Output shape Cifar10
input_1 (InputLayer)	(None, 8, 8, 1)	(None, 28, 28, 1)	(None, 32, 32, 3)
conv2d (Conv2D)	(None, 8, 8, 6)	(None, 28, 28, 6)	(None, 32, 32, 6)
average_pooling2d (AveragePooling2D)	(None, 4, 4, 6)	(None, 14, 14, 6)	(None, 16, 16, 6)
conv2d_1 (Conv2D)	(None, 4, 4, 16)	(None, 14, 14, 16)	(None, 16, 16, 16)
average_pooling2d (AveragePooling2D)	(None, 2, 2, 16)	(None, 7, 7, 16)	(None, 8, 8, 16)
conv2d_2 (Conv2D)	(None, 2, 2, 120)	(None, 7, 7, 120)	(None, 8, 8, 120)
average_pooling2d_2 (AveragePooling2D)	(None, 1, 1, 120)	(None, 3, 3, 120)	(None, 4, 4, 120)
flatten (Flatten)	(None, 120)	(None, 1080)	(None, 1920)
dense (Dense)	(None, 120)	(None, 120)	(None, 120)
dense_1 (Dense)	(None, 84)	(None, 84)	(None, 84)
dense_2 (Dense)	(None, 10)	(None, 10)	(None, 10)
Total Parameters:	27,342	142,542	243,354

rameters as well as the network structure change. The increasing complexity of the three datasets used also implies an increase in the difficulty of processing them. In order to have the ability to process more complex datasets, neural networks may need to become deeper. For example, the network often needs to contain many parameters to perform the Cifar10 classification task excellently. For example, the network in [15] contains 632 million parameters and achieved an accuracy of 99.5%, another network in this paper with 307 million parameters achieved an accuracy of 99.42%. But for Mnist, a network containing 1400000 parameters achieved an accuracy of 99.83% [6].

As mentioned in Chapter 2, the number of layers and the number of neurons in a network usually represents the ability of the network to process data, but at the same time EA shows performance degradation as the number of decision variables increases.

The two feedforward neural networks do not have the ability to achieve good results on the Cifar10 dataset. CNNs, although very good at processing image data, still need to be very deep when processing more complex images. Therefore, Network III and Netwrok IV also do not have the ability to obtain good results with Cifar10. NetworkV though is capable of obtaining satisfactory results on all datasets, but the huge amount of weights is certainly a challenge for the EA.

4.2 Algorithms Design

The weight training of a neural network can be considered as an optimization problem, where the weights can be considered as decision variables, and the output of the network using the weights represented by the current individual determines the fitness of this individual on a given dataset. Crossover and mutation ensure the diversity of the gene pool, allowing a fixed network architecture to continuously try new combinations of weights, and the selection operator ensures performance improvement from generation to generation.

The main goal of this thesis is to verify the suitability of a multi-objective large-scale optimization evolutionary algorithm for training neural networks, so a multi-objective evolutionary algorithm as well as large-scale optimization techniques need to be chosen. In this thesis, NSGA-II is chosen as well as Grouped and Linked Mutation and Linear Combination Search Mechanism

introduced in Section 3.4, and a combination of these two methodes is proposed and is introduced in Section 4.2.4. In the following, the generic design of the optimization process will be presented first, followed by the different methods used.

4.2.1 Encoding

The weights of the neural network are floating point values, so real value coding is chosen. In this thesis, all weights of the network are encoded directly in a one-dimensional array, which is the individuals in populations. The value of each weight can be seen as a gene in the chromosome, and the dimensionality of the individuals is equal to the number of weights of the network.

By encoding in this way, all feasible solutions have the potential to be discovered by the evolutionary operators and closed under them. Each gene represents only one parameter in the entire parameter set of the neural network, and similar genotypes represent similar phenotypes and do not have very distinct fitness values. That is, the real value encoding also conforms to the properties that a good encoding should have as enumerated in Section 2.4.1.

4.2.2 Fitness Function

Since all the datasets we selected are multi-classification problems, Precision and Recall are used, metrics commonly used in classification problems, as the fitness functions. More specifically, the goal of the evolutionary algorithm in the optimization process is to minimize 1 - precision and 1 - recall. In each generation, individuals in the population are sequentially imported into the neural network as their weights. The fitness of each individual is calculated based on the actual output of the input dataset versus the expectation. More specifically, the Macro-average method is used to calculate precision and recall, i.e., $p_i(i = 1, 2, .., n)$ or $r_i(i = 1, 2, .., n)$ represents each class' own precision or recall, and $\frac{\sum_{i=1}^{n} p_i}{n}$ and $\frac{\sum_{i=1}^{n} r_i}{n}$ represent the overall precision and recall achieved by the current neural network on a certain dataset.

4.2.3 Algorithms

As mentioned before, the four genetic algorithms used are shown below:

1. NSGA-II
2. NSGA-II with Grouped and Linked Polynomial Mutation (GroupLinkNSGA-II)
3. NSGA-II with Grouped and Linked Polynomial Mutation and Linear Combination-based Search Algorithm (LCSAbasedGroupLinkNSGA-II)
4. NSGA-II with Linear Combination-based Search Algorithm (LCSAbasedNSGA-II)
5. BP

NSGA-II is used to verify the applicability of the multi-objective optimization evolutionary algorithm itself to this problem. Algorithm 2 and 4 are based on NSGA-II, respectively, using the two large-scale techniques introduced in Section 3.4, to verify the impact of the large-scale techniques here on the convergence rate, and the quality of the solution, etc. More specifically, in Algorithm 2, Grouped and Linked Polynomial Mutation is used instead of the Polynomial Mutation in the standard NSGA-II. In Algorithm 4, NSGA-II is used as a step before Linear Combination Search Mechanism to provide non-dominated solutions. Linear Combination Search Mechanism itself is also optimized by the standard NSGA-II. Algorithm 3 will be introduced below. Finally, the BP algorithm will also be used on the same dataset as well as on the same networks to compare the differences with the above evolutionary algorithms.

4.2.4 LCSAbasedGroupLinkNSGA-II

The purpose of using Algorithm 2 is to maintain the relationship between the interacting neurons, while using Linear Search Mechanism is to reduce the dimensionality of individuals, thus allowing the EA to further improve the quality of the solution free from the dimensionality problem. Naturally, the combination of these two methods is also presented in this thesis and will be compared with other algorithms.

In the first step of this method, NSGA-II with Grouped and Linked Polynomial Mutation is used to provide non-dominated solutions for the Linear Combination Search Mechanism. The Linear Combination Search mechanism then continue to use NSGA-II with Polynomial Mutation to find linear combination coefficients of the non-dominated solutions from the previous step, and

the solution derived by Linear Combination Search Mechanism are integrated into the population at the step, and the optimization process will continue.

In this combination, we use NSGA-II with Grouped and Linked Mutation in the first step to protect the interconnections between variables during mutation. The linear combination of the non-dominated solutions can be viewed as the value of each weight of one of the solutions mutated by the combination of the weights at the corresponding positions of the other solutions. Since the values of the weights at the same positions of each solution are not necessarily the same, the amount of mutation of each weight is also different. Thus whether the linear combination between different solutions would destroy the relationship between the different variables maintained by the first step, and by extension the quality of the solution, is a point worth noting in the experiment.

5 Experiment/Evaluation

In this chapter, experiments based on the dataset, network structure, and algorithms from Chapter 4 are presented. First, the goals of the experiments are explained. Next, the parameter settings used for the experiments are described. Finally, the results of the experiments and the analysis of them are shown.

5.1 Experiment Goals

In conjunction with the goals of this thesis introduced in Section 1.2, the three algorithms described in the previous chapter will be compared with NSGA-II to explore the performance of different evolutionary algorithm strategies along with large-scale optimization techniques, i.e., Grouped and Linked Mutation and Linear Search Mechanism. We also analyze empirically the convergence rate of the algorithms demonstrated by hypervolume and the quality of the solutions obtained by the algorithms in independent runs.

The objectives of this chapter can be generalized to the following three points:

- Compare the results obtained by the EAs with those obtained by the BP algorithm.

  This is one of the main objectives of this thesis. The suitability of the EA for training neural networks containing a large number of parameters will be demonstrated by the results obtained with the algorithm presented in Chapter 4 of this thesis.

- Compare the differences between different EAs.

  The application of large-scale techniques of the EA is also one of the research points in this thesis. Based on NSGA-II, the impact of large-scale optimization techniques on EAs when training neural networks is reflected by the convergence rate and results of different EAs.

- Study the performance of the EA during optimization.

  The significance of applying different architectures and data sets also includes exploring the variation in performance of the EA under different circumstances. For example the effect of the number of parameters and different network types on the results and on the speed of convergence.

5.2 Parameter Setting

The parameters used for the experiments will be presented in this section. The five algorithms including the BP algorithm will be experimented with three datasets and five networks respectively. We first present the general parameters of the experiments, and then the parameters in the EAs are demonstrated. Finally the parameters used in the BP algorithm are described.

The stopping criterion for all experiments is 100000 evaluations. NSGA-II and GroupLinkNSGA-II both have a population size of 100 and run for 1000 generations. Referring to the original paper, the two LCSA-based algorithms have a population size of 40 and each run alternately for 100 generations until the number of evaluations reaches 100000. i.e., the original EA runs for 13 generations and the Linear Combination Search Mechanism runs for 12 generations. For all algorithms random initialization of the populations ensures the diversity of the initial populations. In addition, the stopping criterion for BP is also 100000 evaluations.

NSGA-II for all methods uses SBX with a crossover probability of 0.9 and Polynomial Mutation with a probability of $\frac{1}{n}$, where n represents the number of decision variables. In Algorithms 2 and 3, the variables are first sorted according to their absolute values and then divided into four equal groups. If the number of variables is not divisible by 4, the remaining variables will be divided equally as well. For example, when the number of variables is 1003, the first 753 variables are sorted into three groups of 251 each. The last 250 variables are divided into one group. And the distribution index for all the mutation operators is set to 20 and for all the crossover operators is set to 15.

As mentioned above, the BP algorithm is just an efficient way to compute gradients. Gradient descent methods use the BP algorithm to train neural networks. Due to the size of the data contained in different datasets we use two different methods. Mnist and Cifar10 contain a large number of samples,

we used MGD, the batch size is set to 10000. With respect to Load Digits, since it contains only 1797 samples, Batch Gradient Descent is used. The BP algorithm uses the 'adam' adaptive learning rate provided by keras[1], and the loss function is set to categorical crossentropy.

The same applies to the EAs, namely on Minist and Cifar10, each individual is evaluated using 10,000 samples at a time. When the dataset is Load Digits, 1797 samples are used to evaluate each individual. It is very important to mention that for the Load Digits dataset, the training process and the results of the algorithms are based on the entire dataset. Whereas for the Mnist and Cifar10 datasets, the final results of all algorithms were evaluated on one batch of the training set. There are various reasons for doing so. First, the main goal of this thesis is to explore the possibility of using EAs to train neural networks containing a relatively large number of weights, and we are more concerned with the performance of EAs in the training process and the results achieved on the training set, as well as their differences from BP algorithms in the training process. Second, for the Mnist and Cifar10 datasets, like the size of their own test sets, a batch consists of 10,000 samples, so one batch has the ability to fully represent the current performance of the model. Third, the purpose of using a test set is usually to verify whether the model has the problem of overfitting, and our initial experimental results show that the EA can not achieve satisfactory results on the training sets of Mnist and Cifar10, so using a test set to verify the performance of the model in this case also loses its main meaning.

5.3 Results

All experimental results will be presented in this section. Based on the selected datasets, network architecturse and algorithms, there are 60 combinations. The accuracy, precision,, recall and cross entropy obtained by the 5 algorithms on different datasets and networks are compared. For all the tables and figures the prefix "LCSAbased" is indicated by "L", i.e. LGroupLinkNSGA-II for LCSAbasedGroupLinkNSGA-II and LNSGA-II for LCSAbasedNSGA-II. And for all the tables the best performance is marked in bold, the green cells mean that the current value is not statistically worse than the best value. All values in the tables are the median of the corresponding metrics finally achieved by the algorithms. The values in parentheses are the interquartile range (IQR) of

the acquired data. Asterisks indicate statistical significance to algorithm that obtained the best result. Statistical significance is tested using a one-sided Mann-Whitney-U Test, namely whether the current result is statistically worse than the best result. Moreover, statistical significance is assumed for a value of $p < 0.05$.

Tables 5.1 5.2 5.3 5.4 show the median of the highest precision, recall, accuracy and sparse cross entropy of different algorithms, respectively. For simplicity, we refer to the sparse cross entropy directly as cross entropy in the following tables and figures. Table 5.5 shows the median of the highest hypervolumes achieved by the four EAs, the performance of the EAs. The reference point for the hypervolumes is set to (precision = 1, recall = 1).

Figure A.0.1 - A.0.12, show the variation over time of the corresponding metrics for the runs of the EAs that delivered the median hypervolumes, and the BP-based algorithm that delivered the median losses. Furthermore, the variation of hypervolumes of EAs are also shown in these figures using the runs that obtained the median hypervolumes. All figures are placed in the Appendix, but for the convenience of reading, some are also placed in this chapter. Namely, the variations of hypervolumes for different EAs can also be found in Figure 5.3 - Figure 5.17.

Table 5.1: Median and IQR Values of the Precision

Datasets		NSGA-II	GroupLinkNSGA-II	LGroupLinkNSGA-II	LNSGA-II	BP
Load Digits	Network-I	0.812909*(0.123152)	0.823192*(0.101071)	0.752023*(0.12307)	0.785645*(0.094069)	1.0(0)
	Network-II	0.806179*(0.060999)	0.913363*(0.021554)	0.697385*(0.123820)	0.705958*(0.072404)	1.0(0)
	Network-III	0.704370*(0.161957)	0.799547*(0.170653)	0.641068*(0.070010)	0.658810*(0.130588)	1.0(0)
	Network-IV	0.010184(0)	0.010184(0)	0.010184(0)	0.010184(0)	0.009683*(0)
Mnist	Network-I	0.855494*(0.062186)	0.891253*(0.056009)	0.665873*(0.047591)	0.751198*(0.068068)	1.0(0)
	Network-II	0.800167*(0.074733)	0.840366*(0.081702)	0.627655*(0.069820)	0.714800*(0.071027)	1.0(0)
	Network-III	0.311560*(0.054902)	0.429949*(0.110601)	0.389353*(0.106447)	0.334900*(0.042572)	0.974035(0.005660)
	Network-IV	0.552361*(0.138872)	0.622950*(0.085122)	0.500845*(0.092759)	0.534918*(0.105963)	0.822531(0.003439)
Cifar10	Network-I	0.554766*(0.077114)	0.567374*(0.068315)	0.471833*(0.107279)	0.530502*(0.095679)	0.681802(0.088059)
	Network-II	0.551142*(0.092026)	0.566742*(0.118291)	0.464489*(0.075062)	0.527004*(0.080876)	0.754934(0.249271)
	Network-III	0.396763*(0.095720)	0.440374*(0.061039)	0.391606*(0.077946)	0.371845*(0.086917)	0.579661(0.023721)
	Network-IV	0.524921*(0.091339)	0.518470*(0.082588)	0.433002*(0.072390)	0.442024*(0.065164)	0.955768(0.042355)

Table 5.2: Median and IQR Values of the Recall

Datasets		NSGA-II	GroupLinkNSGA-II	LGroupLinkNSGA-II	LNSGA-II	BP
Load Digits	Network-I	0.694065*(0.130328)	0.773579*(0.059196)	0.616726*(0.070181)	0.621765*(0.060610)	1.0(0)
	Network-II	0.684036*(0.073602)	0.795826*(0.096343)	0.559762*(0.100018)	0.580659*(0.070825)	1.0(0)
	Network-III	0.595526*(0.097675)	0.663891*(0.103974)	0.512260*(0.073198)	0.529100*(0.080168)	1.0(0)
	Network-IV	0.100000(0)	0.100000(0)	0.100000(0)	0.100000(0)	0.100000(0)
Mnist	Network-I	0.635407*(0.080200)	0.641650*(0.072121)	0.451311*(0.037295)	0.506785*(0.058395)	1.0(0)
	Network-II	0.531139*(0.059299)	0.570037*(0.106353)	0.401700*(0.058013)	0.472707*(0.083809)	1.0(0)
	Network-III	0.344650*(0.047539)	0.373035*(0.064843)	0.334260*(0.039011)	0.295657*(0.054057)	0.973845(0.005692)
	Network-IV	0.392684*(0.052696)	0.431667*(0.045667)	0.338274*(0.035396)	0.367580*(0.043737)	0.819279(0.003639)
Cifar10	Network-I	0.221661*(0.027065)	0.210466*(0.027547)	0.193603*(0.013441)	0.201558*(0.015076)	0.677300(0.081040)
	Network-II	0.222373*(0.022724)	0.212765*(0.016997)	0.183138*(0.017983)	0.200888*(0.014008)	0.754240(0.254880)
	Network-III	0.224004*(0.023204)	0.234308*(0.025414)	0.210758*(0.013539)	0.208850*(0.015643)	0.585020(0.023300)
	Network-IV	0.224072*(0.020842)	0.218299*(0.020767)	0.203201*(0.022857)	0.203784*(0.015111)	0.955280(0.042760)

Table 5.3: Median and IQR Values of the Accruracy

Datasets		NSGA-II	GroupLinkNSGA-II	LGroupLinkNSGA-II	LNSGA-II	BP
Load Digits	Network-I	0.692265*(0.134669)	0.760712*(0.069004)	0.619366*(0.074012)	0.622148*(0.065109)	1.0(0)
	Network-II	0.680579*(0.074012)	0.762382*(0.135782)	0.557596*(0.101836)	0.578742*(0.072343)	1.0(0)
	Network-III	0.595437*(0.100167)	0.622148*(0.098497)	0.510851*(0.069560)	0.527546*(0.081247)	1.0(0)
	Network-IV	0.101836(0)	0.101836(0)	0.101836(0)	0.101836(0)	0.096828*(0)
Mnist	Network-I	0.646600*(0.083900)	0.615700*(0.098600)	0.462100*(0.035700)	0.513900*(0.053100)	1.0(0)
	Network-II	0.540100*(0.067600)	0.530900*(0.135600)	0.415100*(0.053400)	0.484300*(0.084300)	1.0(0)
	Network-III	0.344800*(0.052300)	0.339600*(0.048700)	0.349800*(0.046600)	0.310600*(0.051500)	0.973900(0.004917)
	Network-IV	0.398900*(0.048700)	0.369500*(0.124600)	0.351300*(0.034000)	0.376900*(0.046800)	0.820850(0.003300)
Cifar10	Network-I	0.223400*(0.023400)	0.200400*(0.039400)	0.194100*(0.015200)	0.200800*(0.016500)	0.687400(0.072720)
	Network-II	0.219600*(0.018100)	0.144200*(0.063900)	0.184800*(0.012600)	0.203100*(0.014800)	0.758180(0.229319)
	Network-III	0.225900*(0.022200)	0.216200*(0.043300)	0.212800*(0.013800)	0.210500*(0.020700)	0.584220(0.019800)
	Network-IV	0.224000*(0.018800)	0.196500*(0.068900)	0.205600*(0.025900)	0.206600*(0.016300)	0.963180(0.022740)

Table 5.4: Median and IQR Values of the Sparse Cross Entropy

Datasets		NSGA-II	GroupLinkNSGA-II	LGroupLinkNSGA-II	LNSGA-II	BP
Load Digits	Network-I	4.825046*(2.178484)	3.889904*(1.195232)	9.554366*(1.162265)	9.553262*(1.861953)	7.85-07(8.84e-07)
	Network-II	5.121771*(1.155196)	3.733891*(2.214670)	7.310092*(1.621064)	6.5835690*(1.166025)	2.19e-09(1.72e-09)
	Network-III	6.136486*(2.105881)	4.896635*(2.068229)	7.561236*(2.726746)	7.450176*(1.802853)	8.62e-09(5.71e-09)
	Network-IV	14.476695*(0)	14.476695*(0)	14.476695*(0)	14.476695*(0)	2.302485(2.38e-07)
Mnist	Network-I	5.023498*(1.353680)	5.333379*(1.520316)	8.437825*(1.636688)	7.769745*(1.026578)	5.29e-09(2.92e-10)
	Network-II	7.233501*(1.117515)	7.136312*(2.747428)	9.300287*(1.304872)	8.312119*(1.359001)	2.37e-09(1.24e-09)
	Network-III	2.803399*(0.504067)	2.294361*(0.030706)	2.610348*(1.075024)	2.955021*(2.036813)	0.084723(0.018320)
	Network-IV	9.073910*(1.549884)	3.822587*(2.862825)	2.407509*(6.104386)	9.48391*(1.479647)	0.498663(0.009614)
Cifar10	Network-I	12.130472*(0.586608)	11.402417*(6.004159)	2.437667*(0.151389)	12.633842*(0.532989)	0.860224(0.192517)
	Network-II	12.495234*(0.330149)	12.913002*(1.967161)	4.305560*(2.453097)	12.772014*(0.265141)	0.689151(0.610990)
	Network-III	2.606194*(1.592817)	2.334268*(0.198116)	2.302332*(0.077286)	3.086458*(1.975569)	1.182215(0.049221)
	Network-IV	11.819444*(2.179063)	7.075566*(10.483262)	2.333559*(3.734024)	7.581497*(8.367398)	0.158344(0.065668)

Table 5.5: Median and IQR Values of the Hypervolumes of the Evolutionary Algorithms

Datasets		NSGA-II	GroupLinkNSGA-II	LGroupLinkNSGA-II	LNSGA-II
Load Digits	Network-I	0.577558*(0.182405)	0.677845(0.116770)	0.455891*(0.115107)	0.472723*(0.110415)
	Network-II	0.530660*(0.118815)	0.710358(0.125104)	0.366519*(0.113490)	0.409219*(0.118815)
	Network-III	0.412469*(0.176168)	0.542358(0.200112)	0.310138*(0.052671)	0.345965*(0.133281)
	Network-IV	0.001018(0)	0.001018(0)	0.001018(0)	0.001018(0)
Mnist	Network-I	0.534402(0.085378)	0.5449445(0.094772)	0.293618*(0.030892)	0.374805*(0.059133)
	Network-II	0.414122(0.090935)	0.460198(0.133754)	0.238863*(0.079129)	0.336512*(0.089500)
	Network-III	0.104952*(0.037001)	0.161945(0.072958)	0.123047*(0.043498)	0.092595(0.030233)
	Network-IV	0.208508*(0.073681)	0.247629(0.050216)	0.162910*(0.043104)	0.192804*(0.040330)
Cifar10	Network-I	0.122391(0.029481)	0.119721(0.016411)	0.093120*(0.026937)	0.104229*(0.023421)
	Network-II	0.118879(0.028213)	0.113592(0.029297)	0.078917*(0.014740)	0.103582*(0.018065)
	Network-III	0.084721*(0.025927)	0.101537(0.020264)	0.076033*(0.015814)	0.073602*(0.019493)
	Network-IV	0.114244(0.028890)	0.103253(0.011701)	0.087426*(0.015468)	0.087752*(0.009367)

5.3.1 Comparison of the EA and the BP Algorithm

According to Table 5.1 - 5.4, Figure A.0.4 show that for this combination, all algorithms fail to converge and the IQR values of the results obtained in 21 independent runs are all 0. Moreover, Figure 5.20 shows that, finally, all individuals of the population in the final generation of the NSGA-II have the same precision and recall, and all other EAs behave exactly the same way. Furthermore, different algorithms always reach the same value on this combination and fail to further convergence, or even change. Similarly, the cross entropy of the BP algorithm also remains the same after the super small changes in the first few steps.

Because we use random initialization, all individuals in a population must contain different combinations of weights in the 21 independent runs of optimization, and most of the time, except for the first few steps of the BP algorithm, the network always stubbornly outputs the same result regardless of the weights it consists of. This means that for the combination of Network-IV and Digits, the network cannot extract useful information of the input image, or the network contains too many convolutional and pooling layers, and the input information is processed layer after layer, and finally the 120 values generated before the fully connected layers cannot effectively represent the different features between different digits. Because the comparison of the results of this combination with those of other combinations has no other meaning for the goals of this thesis, it is excluded from the following comparisons.

It is noticeable that for all combinations, the BP algorithm outperforms the EAs. This is reflected in all used metrics. In comparision with the EAs, the BP algorithm converges much faster and converges quickly to the optimal values of accuracy, precision, and recall for the combinations of Load Digits and Network-I, II, III. Mnist and Network-I, II. Furthermore, as we can see from the hypervolumes of the Figure 5.3-5.17, although still not comparable to the BP algorithm, EA can have a faster convergence rate in the early stages of optimization, but when the algorithm reaches a certain level of fitness, the convergence rate shows a very noticeable decrease. Certainly, there are differences between different EAs, and these are further illustrated in Seciton 5.3.2.

Moreover, Figure 5.19 shows the non-dominated front achieved by NSGA-II on Mnist and Network-IV, which also reflects the trade-off between two conflicting objectives in the optimization process of the EA. This is an advantage that the EA has over the BP algorithm.

However, the EAs reflect a large gap with the BP algorithm in optimizing CNN. In Table 5.1, 5.2, 5.3, Load Digits and Network-II and III, for example, it is shown by the BP algorithm that both Network-II and III have the ability to achieve a precision recall and accuracy of 1. And in this case, Network-II contains 26122 parameters, while Network-III contains only 1384 parameters, and the EAs are expected to perform better on Network-III. However, the opposite result is achieved, and the EAs perform considerably worse on the two CNNs than the feedforward neural networks with more parameters. Similarly, in the tables, a similar phenomenon is observed for the Mnist dataset, and even stronger. With similar results obtained by the BP algorithm, the gap between the EAs and the BP algorithm on the feedforward neural network and the CNN is further widened.

This may be due to the nature of parameter sharing of CNNs, where the values of the parameters of each convolutional layer need to be applied to different pixels of the input during the optimization process, and multiple pixels need to be taken into account in optimizing the convolutional layers. The consequent change in the value of one convolutional layer is more likely to cause degradation of the solution quality compared to the feedforward neural network, and the presence of multiple conflicting objectives also potentially exacerbates this problem. As results, it is more difficult for the EAs to find the direction to improve the quality of the solutions in this trade-off.

Figure 5.1: Hypervolumes of Load Digits

Figure 5.2: Legend

Figure 5.3: Network-I

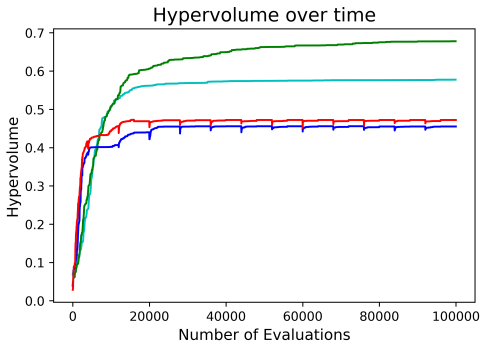


Figure 5.4: Network-II

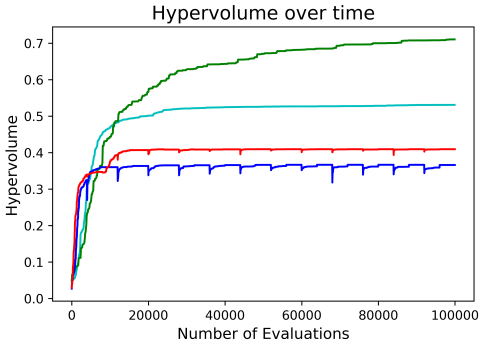


Figure 5.5: Network-III

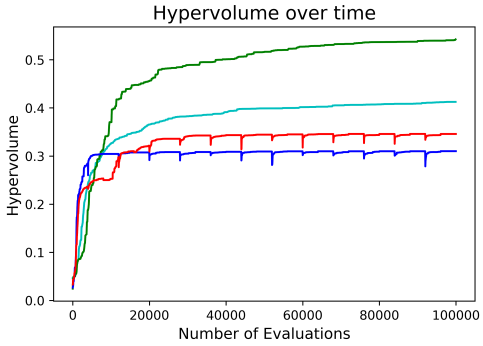


Figure 5.6: Hypervolumes of Mnist

Figure 5.7: Legend

Figure 5.8: Network-I

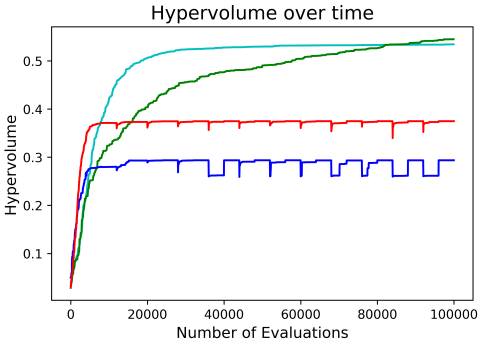


Figure 5.9: Network-II

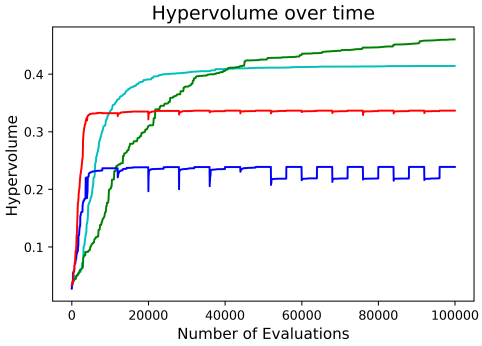


Figure 5.10: Network-III

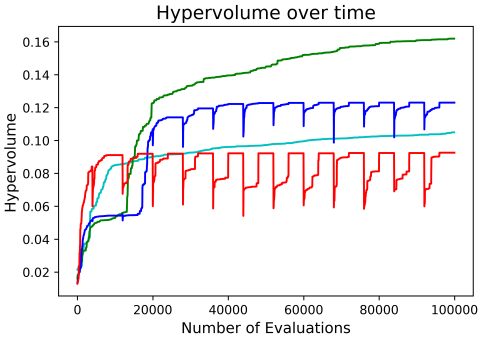


Figure 5.11: Network-IV

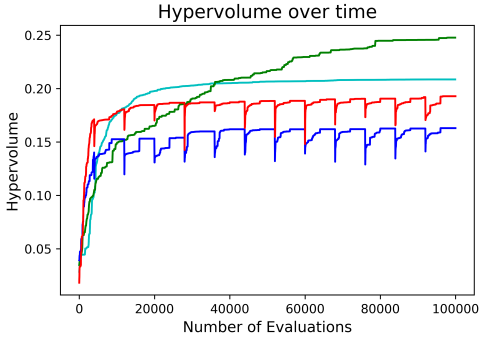


Figure 5.12: Hypervolumes of Cifar10

Figure 5.13: Legend

Figure 5.14: Network-I

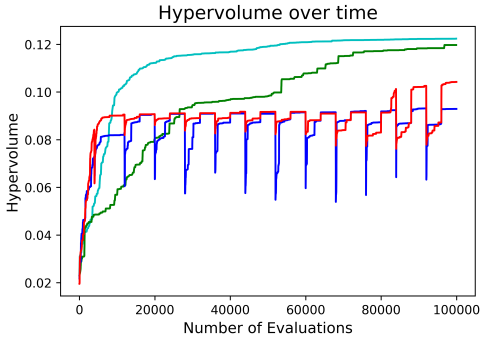


Figure 5.15: Network-II

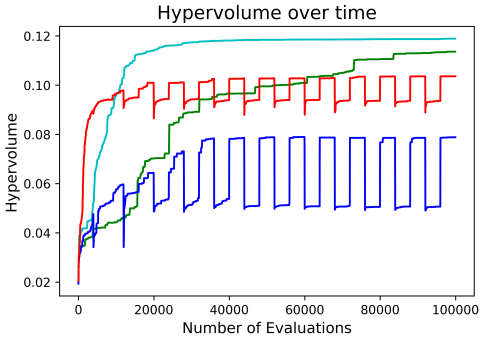


Figure 5.16: Network-III

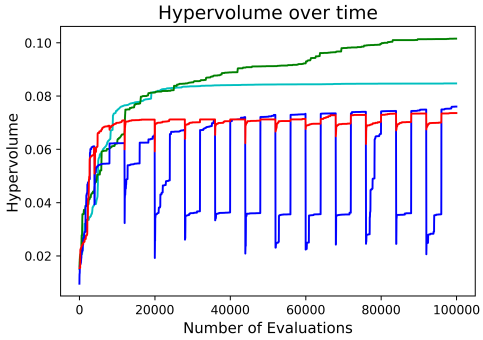


Figure 5.17: Network-IV

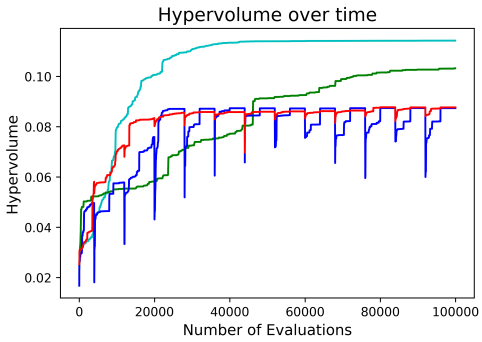


Figure 5.18: Non-dominated Solutions of NSGA-II

Figure 5.19: Network-IV and Mnist

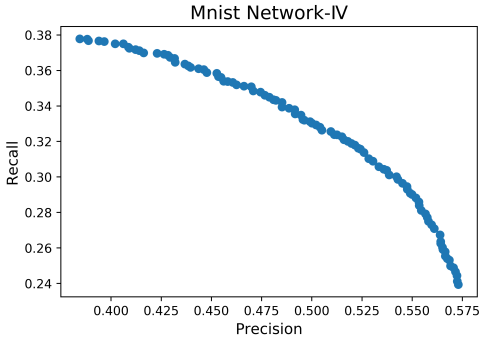
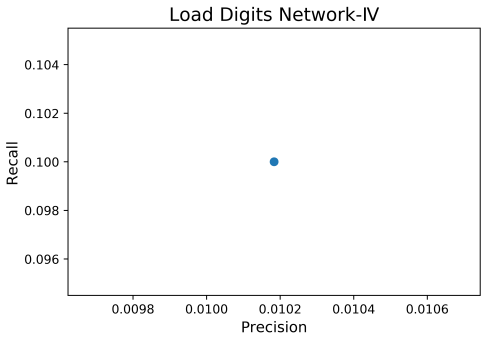


Figure 5.20: Network-IV and Load Digits



NSGA-II and GroupLinkNSGA-II achieved acceptable results on the Load Digits dataset, but the gap between EA and BP increased with the dataset and network complexity. In general, there is still a gap between EAs and BP in optimizing neural networks.

5.3.2 Comparison of the differences between different EAs

The experiments use the different four EAs mentioned in Chapter 4. According to the experimental results there are also some differences between them. According to Table 5.5, in most combinations, Grouped and Linked Mutation can improve the performance of the original NSGA-II by maintaining the relationships between parameters of the networks during the optimization process, and for both Digits and Mnist datasets, the median hypervolume obtained by GroupLinkNSGA-II is statistically superior to that of NSGA-II and the other two using Linear Combination Search Mechanism. Furthermore, as can be seen in Figure 5.3 - 5.17, in most cases, GroupLinkNSGA-II converges more slowly compared to NSGA-II in the early stages of optimization, but it maintains the momentum of convergence when other EAs stop converging.

Regarding LCSA, as presented in Section 5.2, the EA and this mechanism are each run in 100 alternating generations, for a total of 2500 generations. Figure 5.21 and 5.22 show the hypervolume achieved by each of these two on different

combinations of datasets and networks during the optimization process using different colors, where LCSA1 corresponds to the Linear Combination Search Mechanism of GroupLinkNSGA-II and LCSA2 corresponds to the Linear Combination Search Mechanism of NSGA-II. It should be clarified that each time LCSA starts, it randomly generates a number of linear combination coefficients of population size, e.g. 40. Its initial population is then the combination of the non-dominated set provided by the previous evolutionary algorithm and these randomly generated coefficients. As can be seen in the figure, sometimes LCSA1 and LCSA2 are not connected to the EA with which they are combined. The beginning of these breaks in the figures represents the start of the LCSA, and the algorithm starts to search for more linear combinations of factors from the randomly generated initial population. In some figures, the breaks appear at both ends of LCSA1 and LCSA2. This is because when the LSCA ends, if the optimal linear combinations obtained are still worse than the non-dominated solutions provided by its previous EA, then the selection operator of the original EA, i.e., the selection operator of GroupLinkNSGA-II and NSGA-II, selects the non-dominated set of the previous EA as the initial population for the next step, which then forms the end of the breaks.

From these figures, it can be seen that the hypervolumes can be improved by LCSA during the first few rotations, but thereafter LCSA does not provide any further improvement in solution quality by finding linear combinations of the non-dominated solution sets provided by the previous step. Another noteworthy issue is that, as seen in Table 5.5 and Figure 5.21 5.22, LGroupLinkNSGA-II performs less well than LNSGA-II. This validates, to some extent, the conjecture raised at the end of Section 4.2.4 that LCSA has the potential to disrupt the relationship between variables maintained by GroupLinkNSGA-II. In LGroupLinkNSGA-II, the relationships between the different variables originally maintained by the GroupLinkNSGA-II are not considered in the process of LCSA. This means that the relationships between the different variables in the individuals provided by the LCSA received by the GroupLinkNSGA-II may already be destroyed, which prevents GroupLinkNSGA-II from further improving the quality of the solution by maintaining the relationships between the variables on its original basis. Another phenomenon that supports this conjecture is that the difference between the hypervolumes at the beginning of the breaks of LGroupLinkNSGA-II in the figures, that is, the hypervolumes at the beginning of LCSA1, and the hypervolumes at the previous step of GroupLinkNSGA-II are much larger than that at LNSGA-II, i.e. NSGA-II

and LCSA2. That is, the quality of the solution decays more severely due to the destruction of the relationship between the variables.

It is also worth noting that the original EA, i.e., GroupLinkNSGA-II and NSGA-II, cannot converge further when the hypervolume reaches a certain value, although this is the hypervolume which is still much smaller than the value that the original EA can reach, as shown in Table 5.5. The reason for this problem may be that the solution set provided by LCSA to the next step of the original EA, i.e., the solution derived by linear combination, may cause a reduction in the initial population diversity of the next step of the original EA. In addition, the relatively small population size (40) and the huge amount of parameters of the neural network are likely to aggravate this situation. The reduction in population diversity makes it difficult for the algorithm to escape from local optima, which affects the quality of the final solution.

Overall, according to the experiments, GroupLinkNSGA-II shows its improvement over the original EA in most combinations. While LCSA-based algorithms perform less well than NSGA-II and GroupLinkNSGA-II.

5.3.3 Variation of the Performance of EAs

According to the experimental results, in general, the performance of EA is affected by the type of network as well as the number of parameters. The effect of network type on CNN has been mentioned in Section 5.3.1. The effect caused by the number of parameters on EA is focused on in this subsection.

According to Table 5.5, for Network-I and II, the performance of the EAs decrease along with the number of parameters in most cases. GroupLinkNSGA-II performs significantly better for Network-II than Network-I under Digits dataset, while Network-II performs weaker than Network-I under Mnist and Cifar10 datasets. This may be due to the 26122 variables GroupLinkNSGA-II can still handle for the combined Digits and Network-II. While under Mnist and Cifar10, the difference in the number of parameters between Network-I and II is too large, which affects the performance of the algorithm. Interestingly for the two CNNs under the Mnist and Cifar10 datasets, the used EAs outperform Network-IV than Network-III, although Network-IV has a much larger number of parameters than III.

Figure 5.21: Hypervolume of LCSAbased-Algorithms over time

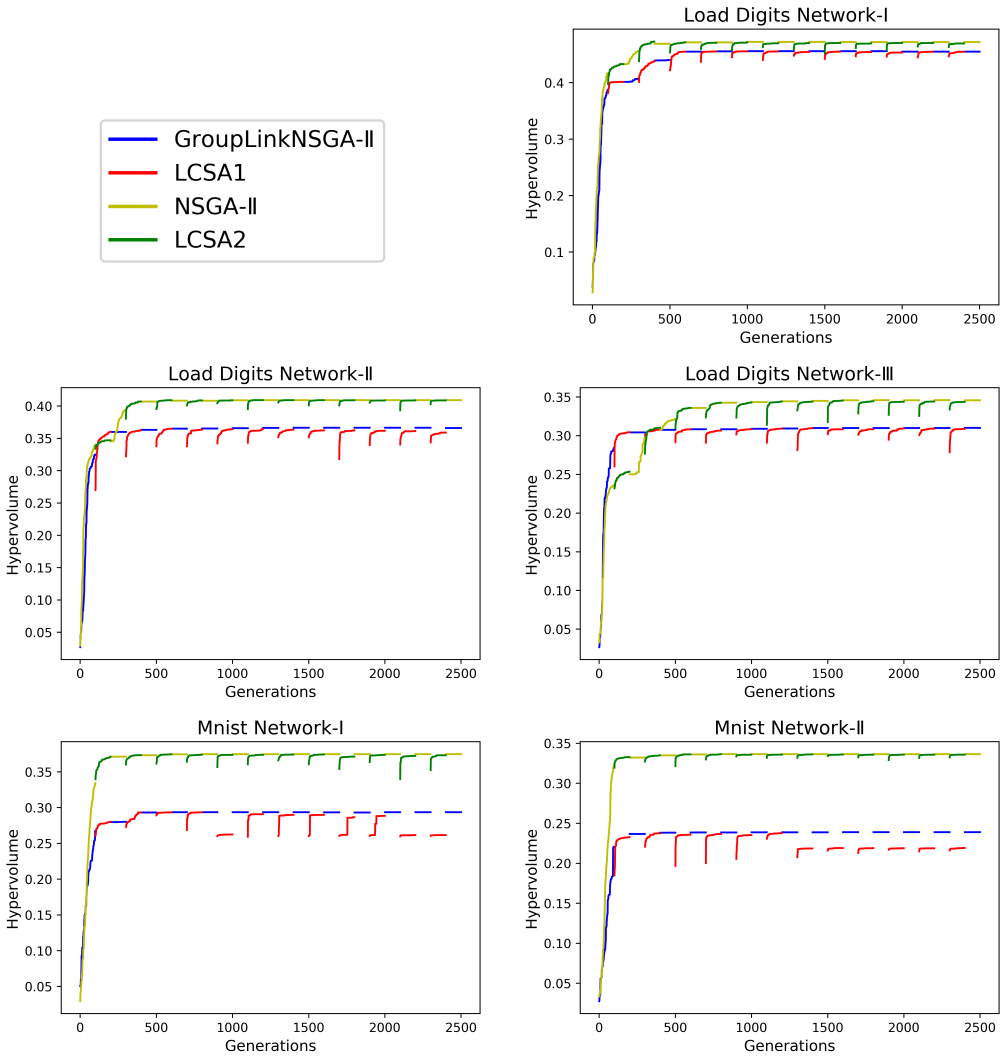
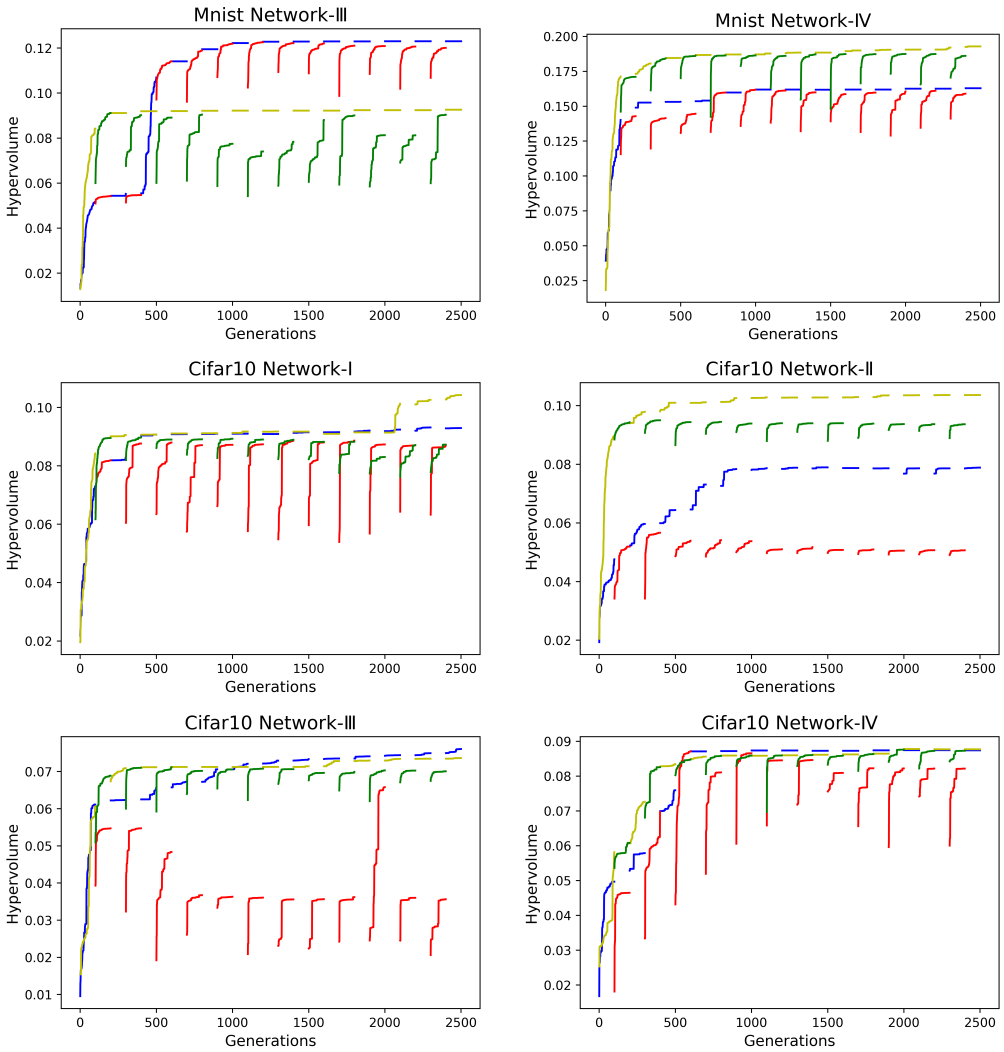


Figure 5.22: Hypervolume of LCSAbased-Algorithms over time



For the two combinations with the most parameters of among all combinations, namely Cifar10 and Network-II and IV, the EAs achieve solutions of much lower quality than the BP algorithm.

And from the comparison of hypervolumes in each EA in Figures 5.3 and 5.4, 5.8 and 5.9, the convergence of the EAs slows down as the parameters increase. Similarly, for all networks, EAs with Cifar10 converges slower than that with Mnist, and EAs with Mnist converges slower than that with Load Digits.

5.3.4 Summary

In general, this chapter compares and analyzes the differences between the EA and the BP algorithms based on the experimental results. The results obtained show the gap between the EA and the BP algorithms in terms of training neural network weights. When using the feedforward neural network for the Load Digits dataset, the used EAs can achieve relatively good results, although its convergence rate is much slower than that of the BP algorithm. On the relatively more complex datasets, the performance of EAs is attenuated by the number of decision variables and is significantly inferior to the BP algorithm. In addition, we analyze the reasons why the used EAs achieved lower than expected results on CNNs despite the fact that they contain fewer variables. Namely, the feature of parameter sharing of the convolutional layer result in each weight affecting multiple input pixels simultaneously, and in optimization, the EAs need not only to make trade-offs between conflicting objectives, but also to consider the impact of each variable on different pixels. We also show the tradeoff of EAs between conflicting objectives through the non-dominated front obtained by NSGA-II, which is one of the advantages of the EA over the BP algorithm.

The differences between the different EAs are also analyzed in this chapter, and the results show that Grouped and Linked Polynomial Mutation can bring improvements to NSGA-II in most combinations, but the consequent shortcoming is its slow convergence in the early stages of optimization. For the LCSA-based algorithms, the combination of LCSA and GroupLinkNSGA-II performs relatively poorly, most likely due to the disruption of the interaction between variables by LCSA.

Finally, the effects of the number of parameters on the EA are also observed. Although the used EAs can still converge on the experimental combination

containing the most parameters, the decay caused by the increase in the number of parameters on the convergence rate of the algorithm is also evident.

6 Conclusion & Future Work

In this thesis, the performance of training neural networks with fixed architectures and a large number of parameters using multi-objective EAs is explored and the results are compared with a BP-based training algorithm. The thesis treats neural network weights training as an MOP, and use NSGA-II and algorithms designed for large-scale optimization problems to solve this MOP. In addition, the thesis proposes a combination based on the cited large-scale optimization techniques, and the proposed approach attempts to further optimize the solution quality by finding linear combinations of the set of non-dominated solutions obtained by the EA at a certain point in time while maintaining the relationships between the variables in the large-scale optimization problem. The different performances of these EAs used in training neural networks are also analyzed. Furthermore, the performance of the different algorithms in the optimization process and the factors contributing to it are also focused. This includes the type of networks, the number of parameters and the interactions between algorithms.

This thesis concludes that EAs can obtain acceptable results on simpler data sets and that trade-offs between conflicting objectives can be made during the training process. Moreover, large-scale optimization techniques can bring some improvements to evolutionary algorithms for neural network optimization. However, there is a gap between the EA and the BP-based algorithm when there are too many parameters and when faced with more complex networks and datasets. That is, there is a significant decay in the performance of EA with the growth of parameters, and the gap with the BP algorithm is further widened again. Although the proposed combination method is worse than the other algorithms used, we have explored and analyzed the reasons for this. From another perspective, the improvement brought by using large-scale optimization techniques to multi-objective EAs also offers hope for the application of EAs to this particular problem.

Future work based on this thesis could be to explore the specific reasons why EAs achieve poorer results on CNNs compared to those on feedforward neural networks, and their specific behavior when training CNNs. Regarding the LCSA-based algorithm, it is worth examining whether LSCA causes a decrease in population diversity, and whether the performance and results of the algorithms are directly related to the population diversity. In addition, the difference between LGroupLinkNSGA-II and LNSGA-II can also be a part of this future work.

Moreover, the datasets selected in this thesis are all balanced datasets, and sparse cross entropy measures the distance between the actual output distribution and the expected output distribution, and the BP-based algorithm uses this metric as the loss function to some extent implying simultaneous optimization for precision, recall and accuracy. However, it is also worth exploring whether the EA can fully exploit its advantages of multi-objective optimization when it comes to certain specific problems with unbalanced datasets.

Appendix

Figure A.0.1: Network-I + Load Digits

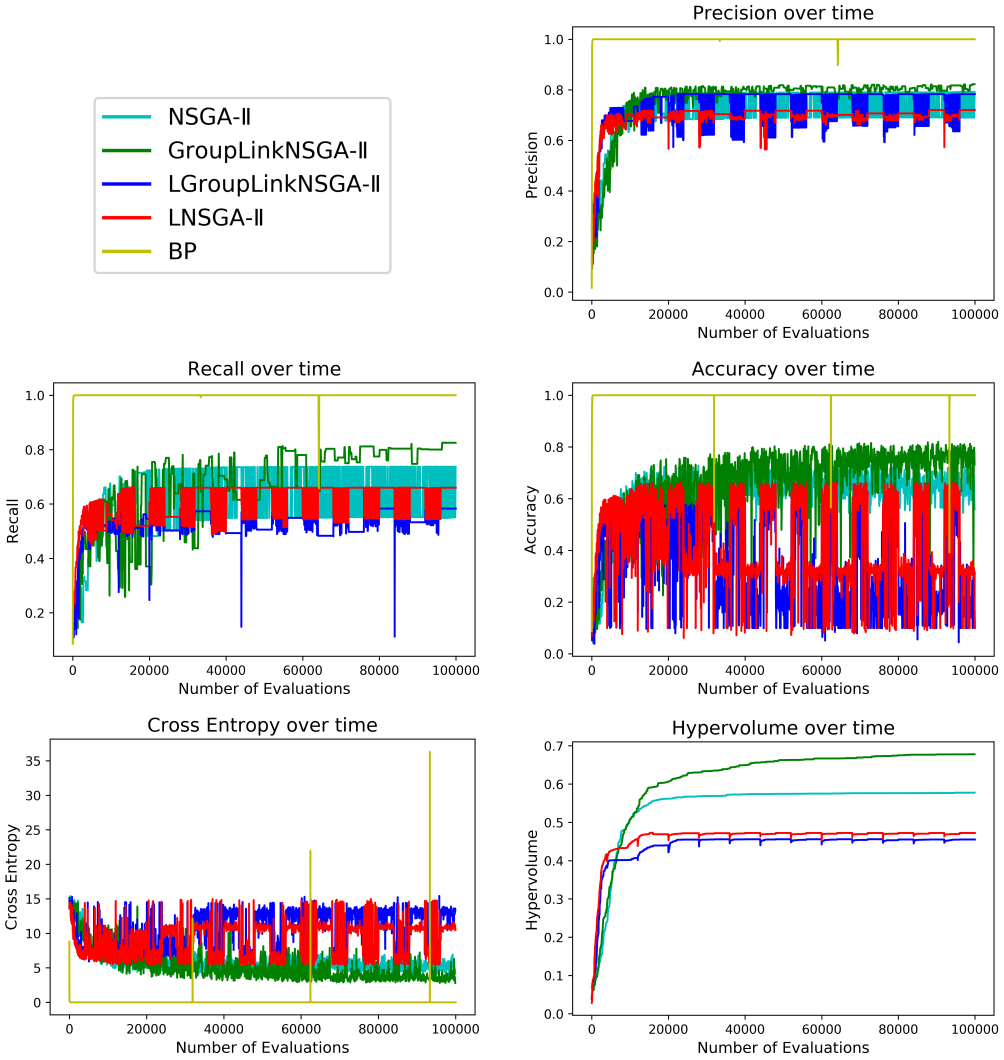


Figure A.0.2: Network-II + Load Digits

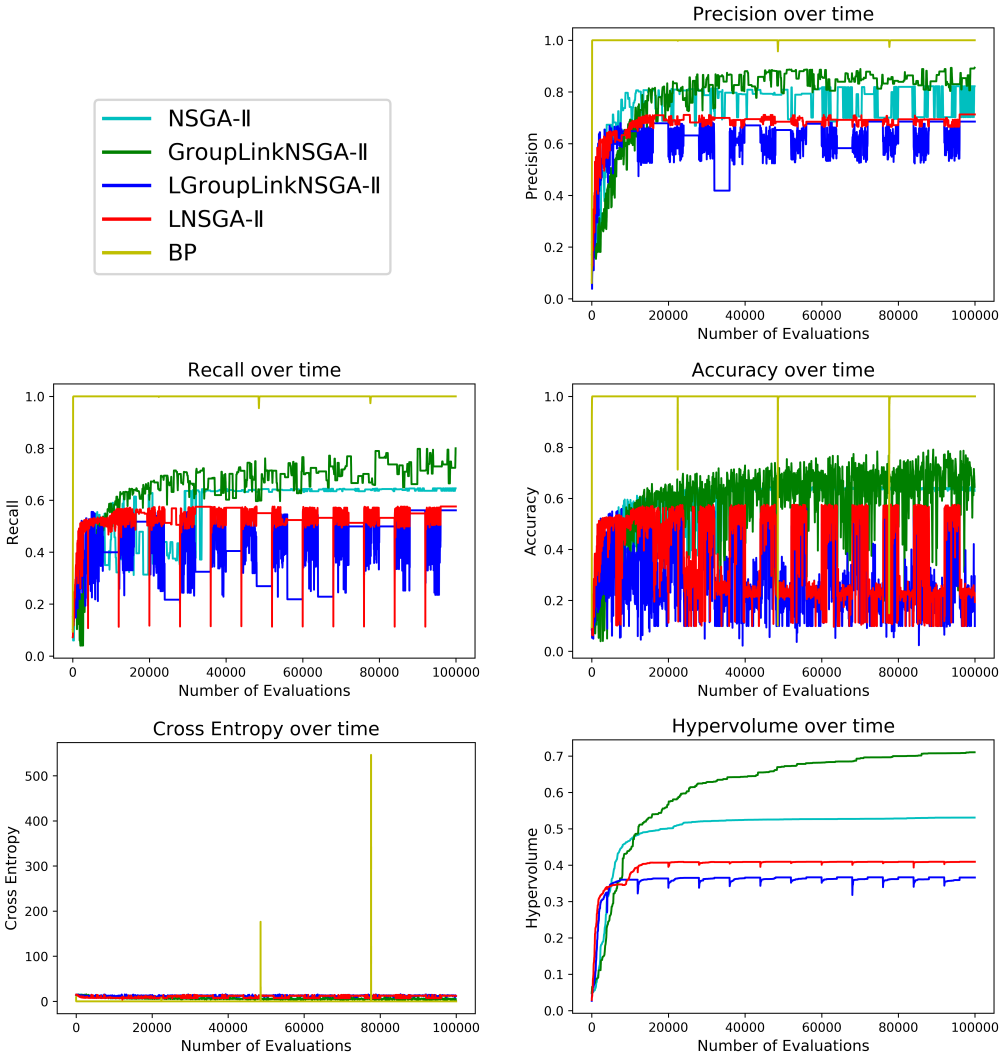


Figure A.0.3: Network-III + Load Digits

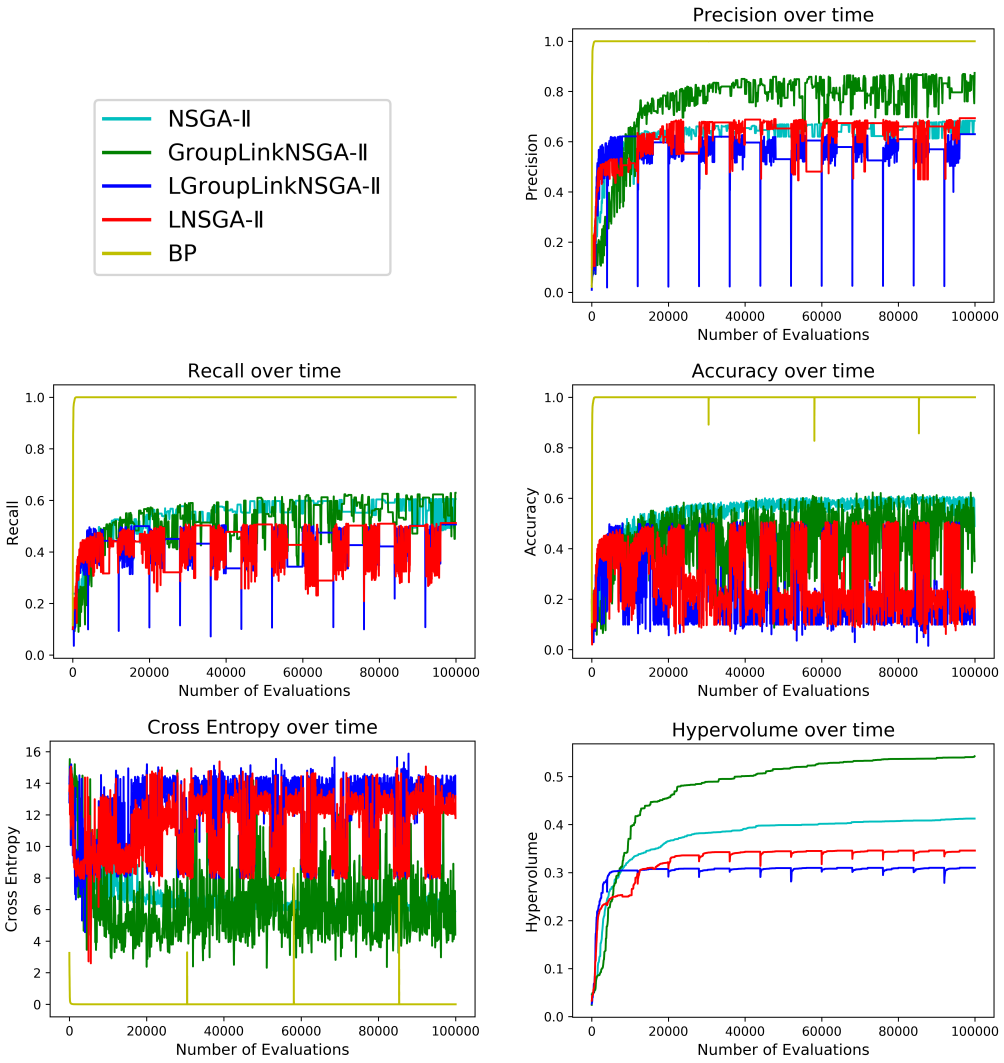


Figure A.0.4: Network-IV + Load Digits

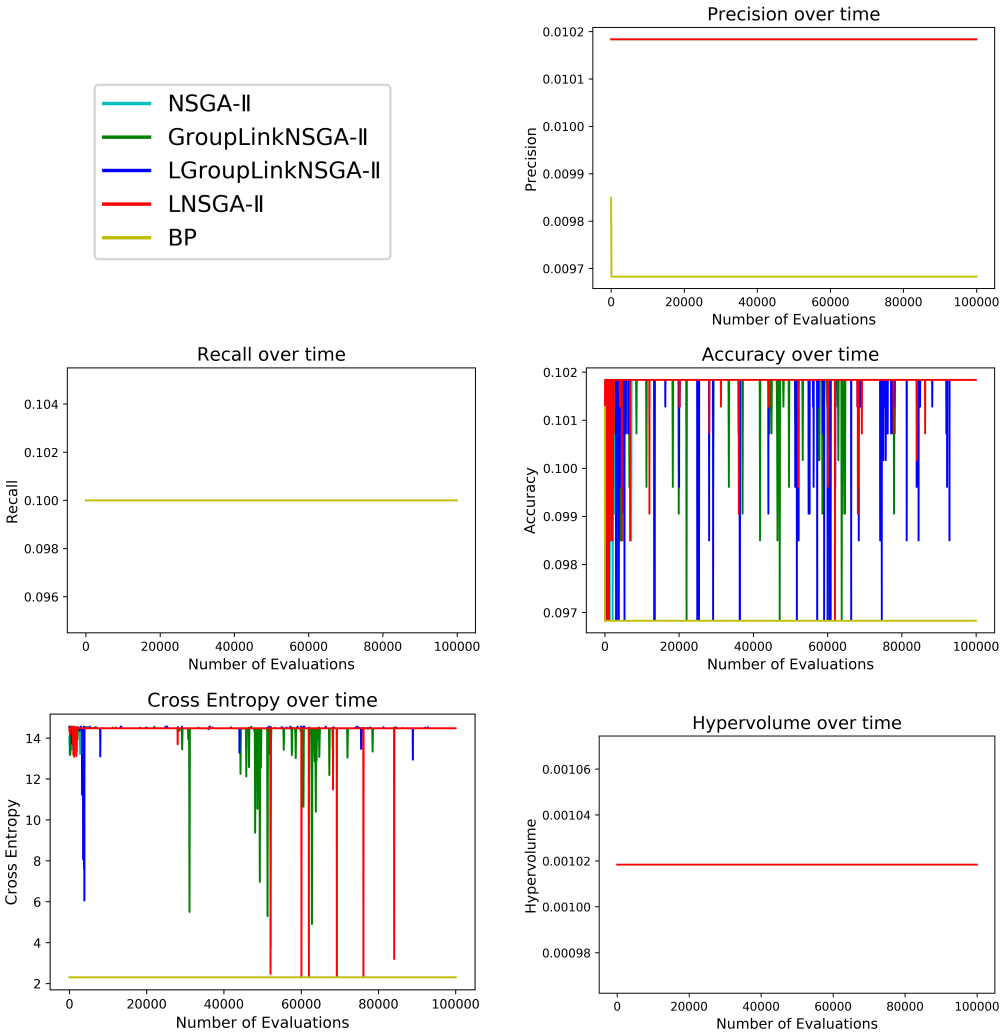


Figure A.0.5: Network-I + Mnist

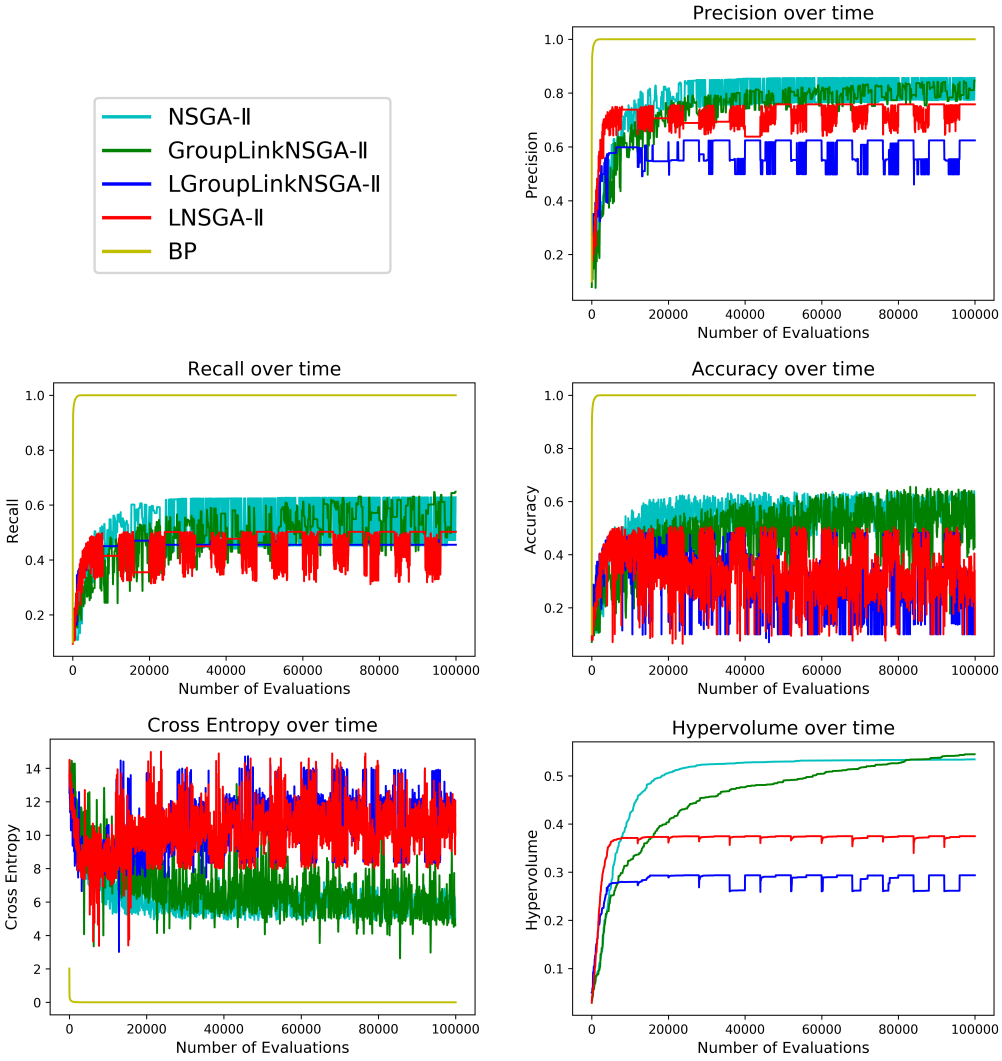


Figure A.0.6: Network-II + Mnist

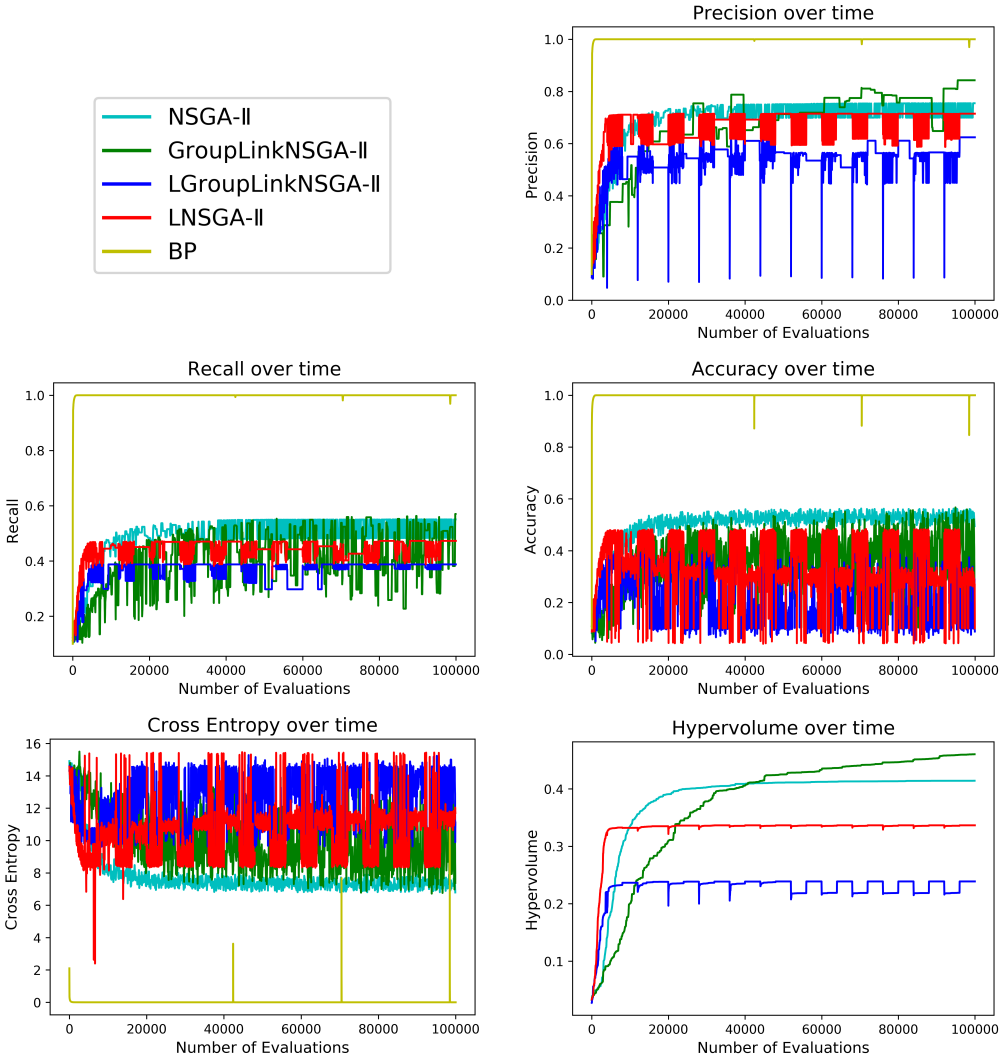


Figure A.0.7: Network-III + Mnist

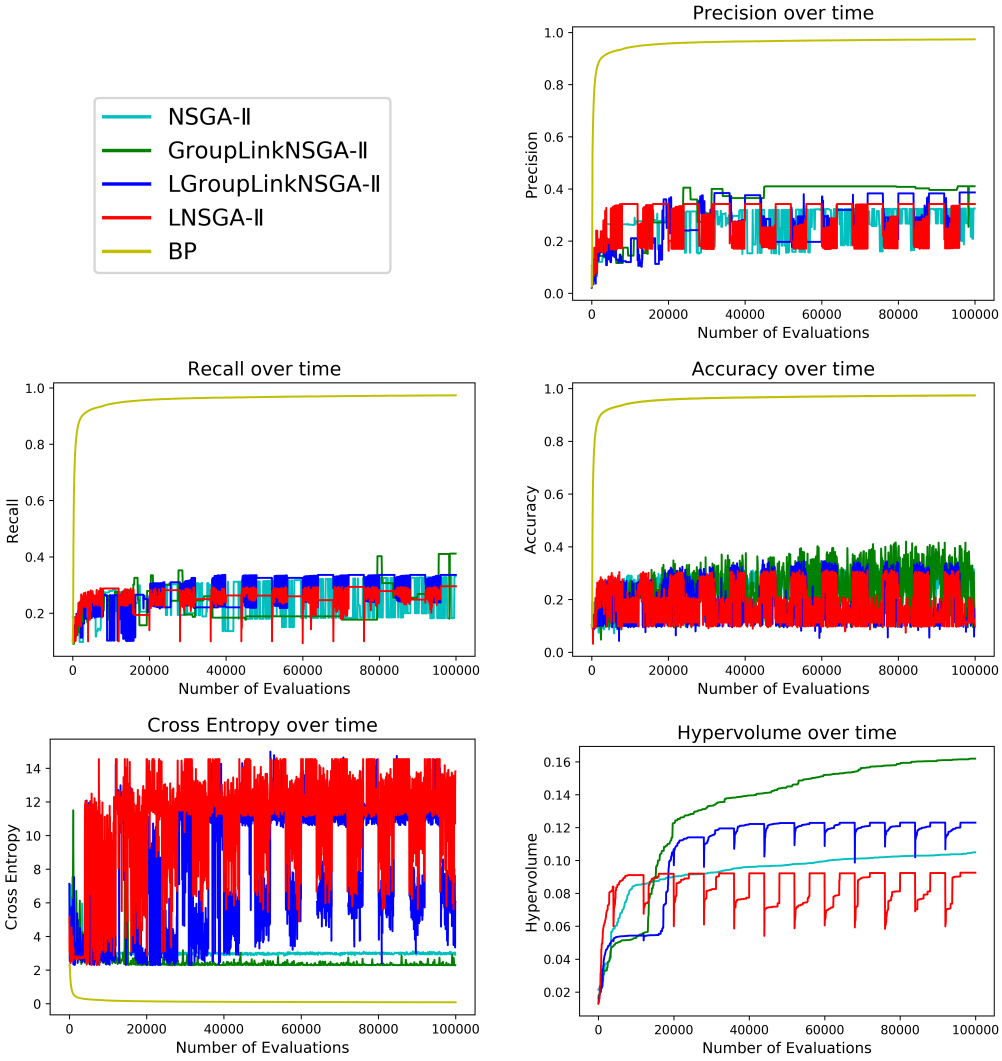


Figure A.0.8: Network-IV + Mnist

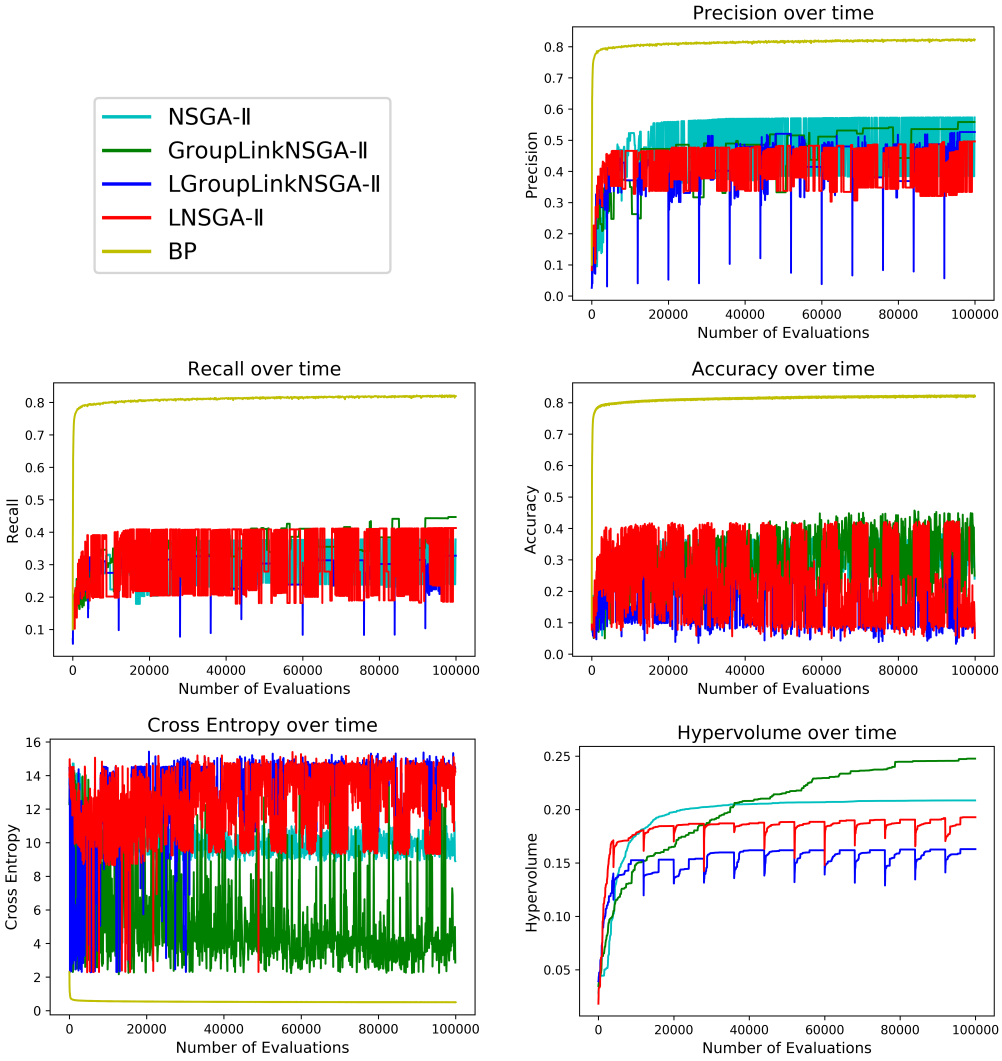


Figure A.0.9: Network-I + Cifar

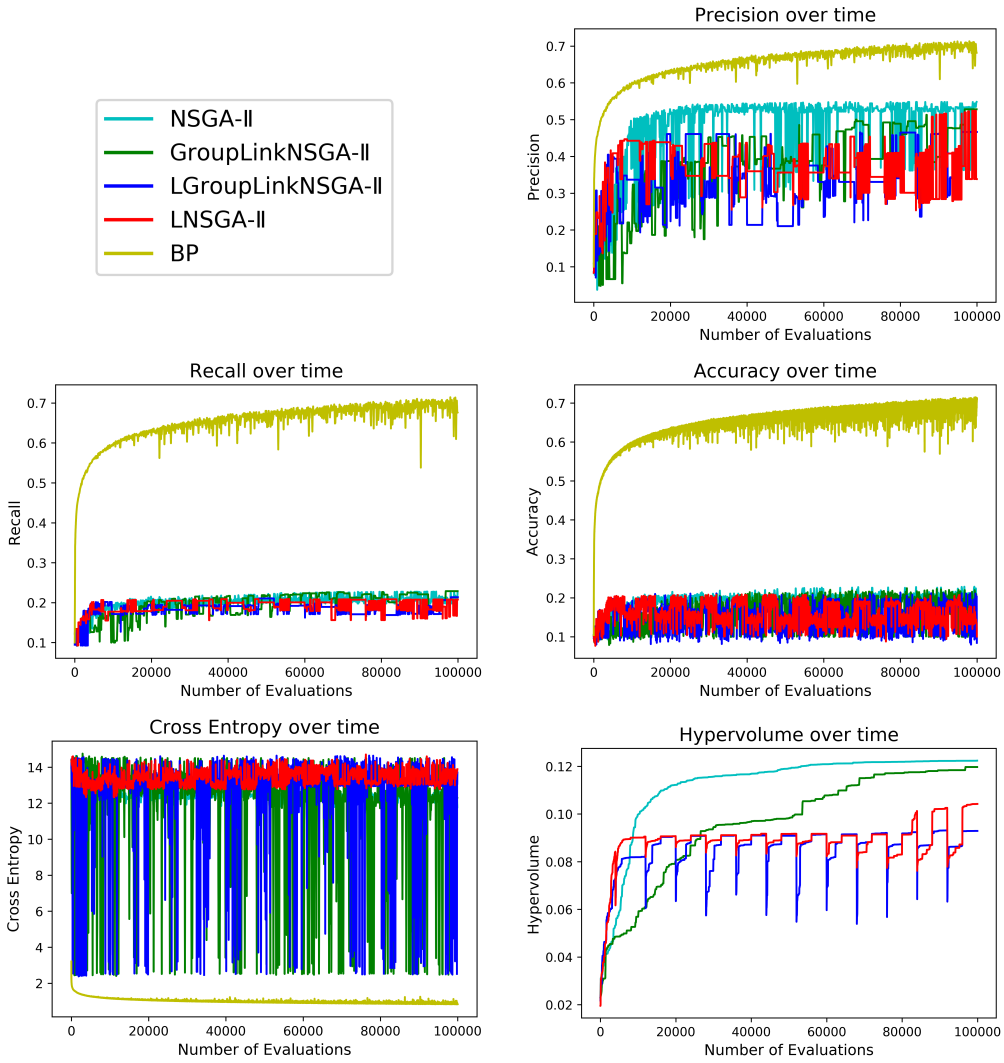


Figure A.0.10: Network-II + Cifar

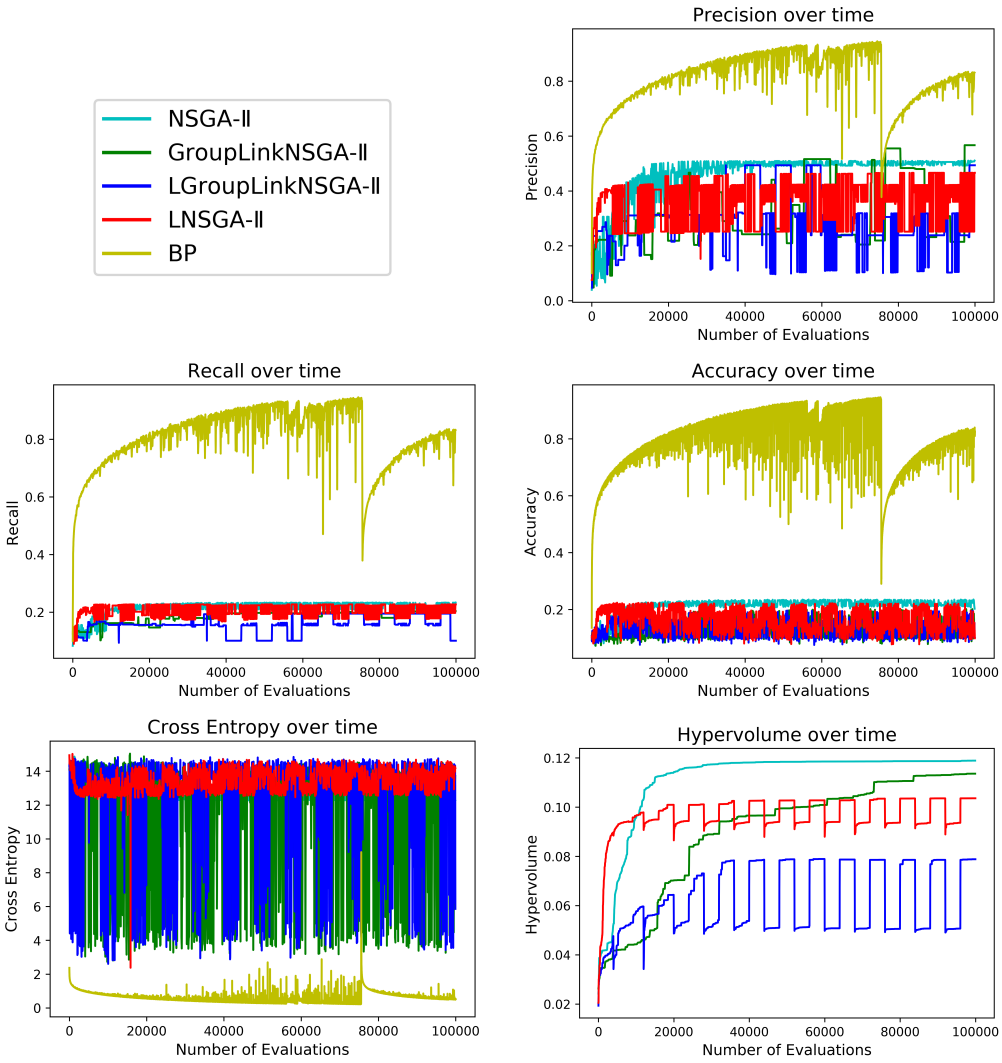


Figure A.0.11: Network-III + Cifar

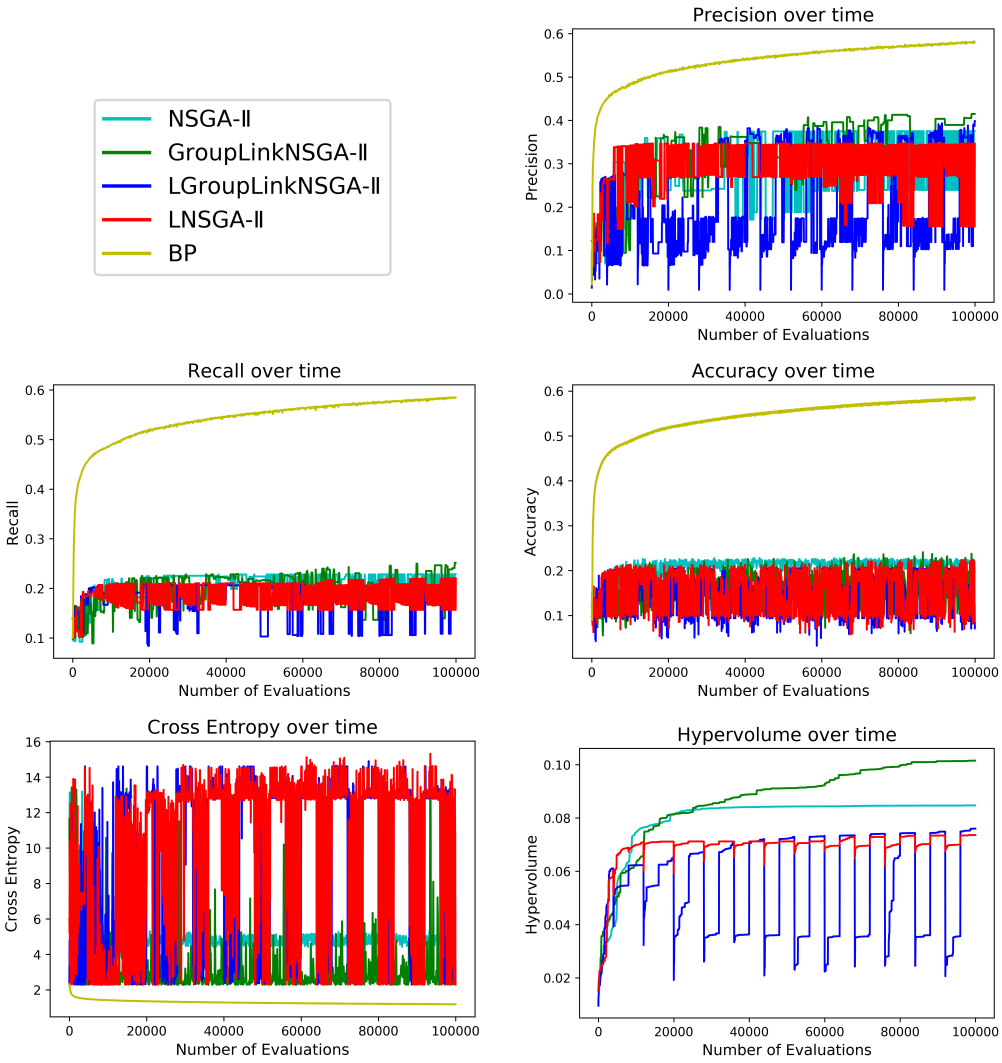
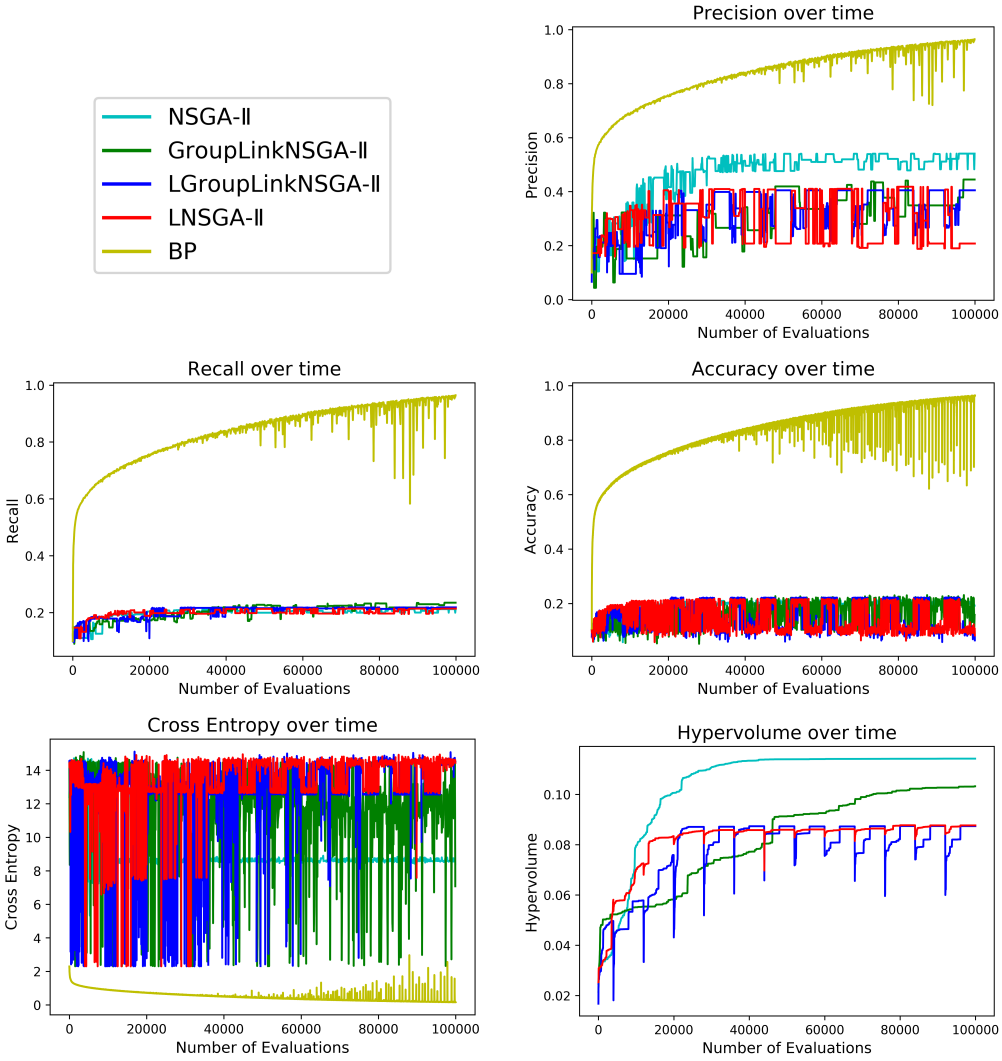


Figure A.0.12: Network-IV + Cifar



Bibliography

[1] Adam, https://keras.io/api/optimizers/adam/, accessed: 2021-02-14.

[2] Distributed tensorflow, https://www.oreilly.com/content/distributed-tensorflow/, accessed: 2021-01-17.

[3] Examples using sklearn.datasets.loaddigits, https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html, accessed: 2021-01-14.

[4] sklearn.metrics.precision_score, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precisionscore, accessed: 2021-01-14.

[5] The mnist database, http://yann.lecun.com/exdb/mnist/, accessed: 2021-01-14.

[6] Yahia Assiri. Stochastic optimization of plain convolutional neural networks with simple methods. *arXiv preprint arXiv:2001.08856*, 2020.

[7] Thomas M Breuel. The effects of hyperparameters on sgd training of neural networks. *arXiv preprint arXiv:1508.02788*, 2015.

[8] Qi Chen, Mutao Huang, and Ronghui Wang. Genetic algorithm–back propagation (ga-bp) neural network for chlorophyll-a concentration inversion using landsat 8 oli data. In *E3S Web of Conferences*, volume 143, page 02002. EDP Sciences, 2020.

[9] Wenxiang Chen, Thomas Weise, Zhenyu Yang, and Ke Tang. Large-scale global optimization using cooperative coevolution with variable interaction learning. In *International Conference on Parallel Problem Solving from Nature*, pages 300–309. Springer, 2010.

[10] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[11] Kalyanmoy Deb and Hans-Georg Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary computation*, 9(2):197–221, 2001.

[12] Kalyanmoy Deb and Mayank Goyal. A combined genetic adaptive search (geneas) for engineering design. *Computer Science and informatics*, 26:30–45, 1996.

[13] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4):577–601, 2013.

[14] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[16] Emmanuel Dufourq and Bruce A Bassett. Eden: Evolutionary deep networks for efficient machine learning. In *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, pages 110–115. IEEE, 2017.

[17] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. Springer, 2015.

[18] Parsa Esfahanian and Mohammad Akhavan. Gacnn: Training deep convolutional neural networks with genetic algorithm. *arXiv preprint arXiv:1909.13354*, 2019.

[19] Ashish Ghosh and Mrinal Kanti Das. Non-dominated rank based sorting genetic algorithms. *Fundamenta Informaticae*, 83(3):231–252, 2008.

[20] Yue-Jiao Gong, Wei-Neng Chen, Zhi-Hui Zhan, Jun Zhang, Yun Li, Qingfu Zhang, and Jing-Jing Li. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, 34:286–300, 2015.

[21] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[22] Jatinder ND Gupta and Randall S Sexton. Comparing backpropagation with a genetic algorithm for neural network training. *Omega*, 27(6):679–684, 1999.

[23] Yaochu Jin and Bernhard Sendhoff. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415, 2008.

[24] Yasusi Kanada. Optimizing neural-network learning rate by using a genetic algorithm with per-epoch mutations. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1472–1479. IEEE, 2016.

[25] Heung Bum Kim, Sung Hoon Jung, Tag Gon Kim, and Kyu Ho Park. Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates. *Neurocomputing*, 11(1):101–106, 1996.

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[27] Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, and Matthias Steinbrecher. *Computational intelligence: a methodological introduction*. Springer, 2016.

[28] Jan Karel Lenstra and AHG Rinnooy Kan. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, 26(4):717–733, 1975.

[29] Frank Hung-Fat Leung, Hak-Keung Lam, Sai-Ho Ling, and Peter Kwong-Shun Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, 14(1):79–88, 2003.

[30] Jing Li, Ji-hang Cheng, Jing-yuan Shi, and Fei Huang. Brief introduction of back propagation (bp) neural network algorithm and its improvement. In *Advances in computer science and information engineering*, pages 553–558. Springer, 2012.

[31] Ruochen Liu, Rui Ren, Jin Liu, and Jing Liu. A clustering and dimensionality reduction based evolutionary algorithm for large-scale multiobjective problems. *Applied Soft Computing*, 89:106120, 2020.

[32] Chun Lu, Bingxue Shi, and Lu Chen. Hybrid bp-ga for multilayer feedforward neural networks. In *ICECS 2000. 7th IEEE International Conference on Electronics, Circuits and Systems (Cat. No. 00EX445)*, volume 2, pages 958–961. IEEE, 2000.

[33] George D. Magoulas, Michael N. Vrahatis, and George S Androulakis. Improving the convergence of the backpropagation algorithm using learning rate mptation methods. *Neural Computation*, 11(7):1769–1796, 1999.

[34] Tadahiko Murata and Hisao Ishibuchi. Moga: Multi-objective genetic algorithms. In *IEEE international conference on evolutionary computation*, volume 1, pages 289–294, 1995.

[35] Richard E Nisbett and Timothy D Wilson. Telling more than we can know: verbal reports on mental processes. *Psychological review*, 84(3):231, 1977.

[36] Mohammad Nabi Omidvar, Xiaodong Li, Yi Mei, and Xin Yao. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on evolutionary computation*, 18(3):378–393, 2013.

[37] Mitchell A Potter. *The design and analysis of a computational model of cooperative coevolution*. PhD thesis, Citeseer, 1997.

[38] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[39] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International Conference on Machine Learning*, pages 343–351, 2013.

[40] V Skorpil, V Oujezsky, P Cika, and M Tuleja. Parallel processing of genetic algorithms in python language. In *2019 PhotonIcs & Electromagnetics Research Symposium-Spring (PIERS-Spring)*, pages 3727–3731. IEEE, 2019.

[41] An Song, Qiang Yang, Wei-Neng Chen, and Jun Zhang. A random-based dynamic grouping strategy for large scale multi-objective optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 468–475. IEEE, 2016.

[42] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[43] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[44] Ye Tian, Xingyi Zhang, Chao Wang, and Yaochu Jin. An evolutionary algorithm for large-scale sparse multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 24(2):380–393, 2019.

[45] Zhenyu Yang, Jingqiao Zhang, Ke Tang, Xin Yao, and Arthur C Sanderson. An adaptive coevolutionary differential evolution algorithm for large-scale optimization. In *2009 IEEE Congress on Evolutionary Computation*, pages 102–109. IEEE, 2009.

[46] Heiner Zille, Hisao Ishibuchi, Sanaz Mostaghim, and Yusuke Nojima. Mutation operators based on variable grouping for multi-objective large-scale optimization. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[47] Heiner Zille and Sanaz Mostaghim. Linear search mechanism for multi-and many-objective optimisation. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 399–410. Springer, 2019.

[48] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*, 7(2):117–132, 2003.