Do More Elaborate Search Strategies Lead to Better Neural Architecture Search Performance?

Tom den Ottelander

M.Sc. Thesis

TU Delft

CWI

Do More Elaborate Search Strategies Lead to Better Neural Architecture Search Performance?

by

Tom den Ottelander

to obtain the degree of Master of Science
in Computer Science, within the field of Data Science & Technology,
specialized in Algorithmics, at the Delft University of Technology,
to be defended publicly on Wednesday October 28th, 2020 at 10:00 AM.

Student number:	4274954		
Project duration:	December 1, 2019 – October 28, 2020		
Thesis committee:	Prof. Dr. P.A.N. Bosman,	TU Delft, CWI,	Chair & Supervisor
	Dr. M. De Weerdt,	TU Delft,	Committee member
	Dr. J. Van Gemert,	TU Delft,	Committee member
	M.Sc. A. Dushatskiy,	TU Delft, CWI,	Daily supervisor
	Dr. M. Virgolin,	TU Delft, CWI,	Daily supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Delft University of Technology
Faculty of Electrical Engineering, Mathematics & Computer Science
Van Mourik Broekmanweg 6
Delft, The Netherlands

Centrum Wiskunde & Informatica
Life Sciences & Health Group
Science Park 123
Amsterdam, The Netherlands

Abstract

Computer vision tasks, like supervised image classification, are effectively tackled by convolutional neural networks, provided that the architecture, which defines the structure of the network, is set correctly. Neural Architecture Search (NAS) is a relatively young and increasingly popular field that is concerned with automatically optimizing the architecture of neural networks. Previously known work shows that even though a recent trend has been to develop increasingly complex search strategies for NAS, several search strategies do not significantly outperform simple approaches like randomly sampling from the search space on single-objective NAS tasks. Additionally, proper ablation studies are often missing. Therefore, it is currently uncertain at best which mechanisms are key for an algorithm to have to achieve excellent NAS performance. In the first part of this thesis, Local Search (LS) and a differently biased form of random search, are proposed for multi-objective (MO) NAS. The multi-objective version of NAS is studied less and understanding the trade-off between between multiple objectives for architectures is arguably more interesting. We find that very simple algorithms can achieve search performance close to that of state-of-the-art evolutionary algorithms (EAs), while outperforming plain random search. Additionally, we find that the quality of the set of architectures found by LS is similar to the those found by the EAs, if compared with respect to test accuracy. Nevertheless, from the compared search strategies the Multi-Objective Gene-pool Optimal Mixing Evolutionary Algorithm (MO-GOMEA), a state-of-the-art model-based EA, achieves the best performance. In the second part of this thesis, it is explored which mechanisms are essential for MO-GOMEA to achieve an excellent search performance for NAS spaces. We find that the automatic population-sizing scheme of MO-GOMEA offers a welcome anytime-performance, but objective space clustering has only a small beneficial impact. The number of clusters can be set arbitrarily. Special (extreme) clusters that optimize for one objective only can be enabled to the practitioner's preference, resulting in different search behaviors. The improvement in performance gained by automatically detecting and exploiting dependencies within architectures is limited: this model-based aspect of MO-GOMEA seems only helpful for finding highly accurate networks.

Preface

This thesis concludes my master's in Computer Science at the Technical University of Delft and simultaneously concludes my time as a student at this university. With pride and fulfillment I look back on the years on this campus and the challenges I was presented with. I started my educational career with a bachelor's in Industrial Design Engineering, also at the Technical University of Delft. This study enabled me to use the creativity that I love to express, but then the minor Software Design & Application opened my eyes to computer science, a field involving more exact sciences. After following a bridging program in order to be accepted for the master Computer Science, I was certain (and still am) that this is the field in which I want to develop myself further in the future. I have completed computer science courses with great joy, in particular the software, algorithmic, data science and machine learning courses. This led me to investigating Neural Architecture Search, a young and interesting field that combines some of these topics by employing optimization algorithms to search for optimal machine learning models. While it has been an exciting and new experience for me to learn about this by actually performing science, exploring literature, setting up experiments and discussing their results in an academically sound way, I am proud, yet relieved that this journey is coming to an end.

Tom den Ottelander
Delft, The Netherlands
October 9, 2020

Acknowledgements

I am grateful for the excellent supervision I received from Peter Bosman, Marco Virgolin and Arkadiy Dushatskiy. Besides their expertise and helpful steering during the academic process, they were understanding and compassionate of the less pleasant events that happened during the journey on my side. Their personal touch on the supervision gave me the strength to temporarily put a brake on the project and get back in the saddle when the time was right and for that, I would like to thank them very much.

After becoming acquainted with Peter Bosman by talking about our shared interest in Star Wars, it quickly became apparent to me that his expert knowledge on the science fiction genre extends to the field of evolutionary algorithms and that he was the right person to have as supervisor of my thesis. His critical questions and passionate enthusiasm for the subject confirmed this and have been an important factor for the successful completion of this master's thesis.

I am glad to have had the opportunity to work closely together with Marco Virgolin and Arkadiy Dushatskiy. Their steering of my project at a close quarter and always being available for questions and uncertainties have been truly helpful. They have shown me of what it is like to be an academic and I am proud to have them as co-authors on the article we wrote together.

I am glad to have been allowed to work on this project as a trainee at the Centrum Wiskunde & Informatica (CWI) in Amsterdam as a part of the Life Sciences & Health group and I want to thank my other colleagues there for their friendly welcome and the fun and interesting conversations that we had. The place felt home from the beginning and has been a knowledgeable and motivating working place during the project.

Last year, partly because of a pandemic outbreak, it has become ever more clear how important my friends are to me and how they have helped me not just through the past year, but also throughout my entire time as a student. Thank you for offering me an environment where I can be myself, and where I can mentally recharge to excel at moments when it is demanded of me.

A final thanks goes out to my parents, which have been there along my side not only to cheer for me at every high point, but also to support me at every low point. Without them, this journey would not have succeeded the way I intended it to.

Contents

1 Introduction

Machine Learning (ML), the process of teaching a machine to do a task by giving it many examples of desired behavior without explicitly programming it to accomplish the task, is something that has led to inspiring and astonishing results in the recent decades. Nowadays, modern cars contain increasingly advanced driver-assistance systems, intelligent virtual assistants are accessible to anyone, medical diagnoses are assisted by model-based predictions [40] and fraudulent online transactions are detected faster than ever before [63]. More and more tasks are being automated, because the tasks are tedious for humans, time-consuming, or the automation simply offers a better performance.

The field of Deep Learning (DL), a sub-field of machine learning, focuses on the innovations and applications of Neural Networks. These networks, inspired by the structure and functioning of our own brains, are graphs of connected layers (which describe the *architecture*) that can be trained to learn a function from data. Once trained, these networks can then be applied to unseen input, thereby automating the task it has been trained to learn. The advantage of neural networks, compared to traditional machine learning approaches, is that less manual labor is required before it can be applied on data. However, to learn or approximate a good function for the task at hand is only possible if the architecture is good enough.

The concepts of deep learning have been around for a long time, but the technological (hardware) advancements, like better GPUs and the development of ML-driven TPUs[1] [27], and the availability of large amounts of training data [32] have enabled a true explosion of the field during the past decade. The versatile applicability of neural networks have led to impressive successes in a variety of fields, including image processing [19, 40], natural language processing [2, 85], games [67, 68] and genomics [15]. Another crucial factor for recent breakthroughs in DL, are novel architectural feats (e.g. the invention of residual connections [22]). Some tasks require specific architectures, but it is not always straightforward how to choose a suitable architecture. Proper architecture design requires experience, intuition and often involves expensive trial-and-error. It is therefore not surprising that humans want to automate this task as well. As a result, the field of Neural Architecture Search (NAS) has emerged. Automating the automation, if you will.

Research on NAS has been gaining popularity quickly. For example, more than 250 articles on NAS research have been published in 2019 alone[2]. In performing NAS, a search algorithm is used to find suitable architectures from a manually defined search space. More and more new algorithms are proposed, with increasing complexity. Frequently, the primary focus of publications is to discover architectures with new state-of-the-art performances. In doing so, proper comparisons of the proposed new search method against simple search strategies like random search are often missing. As shown in [83] and [86], several NAS algorithms are no better than random search when validated properly. Manually (re-)designing the architecture search space such that even randomly sampling can yield good results, makes the use of advanced NAS algorithms debatable.

These results [83, 86] only show that random search is a competitive search strategy for single-objective NAS tasks (i.e., only optimizing for highly accurate architectures). This thesis, however,

[1] https://blog.google/topics/google-cloud/google-cloud-offer-tpus-machine-learning/
[2] https://www.automl.org/automl/literature-on-neural-architecture-search/

focuses primarily on multi-objective NAS (i.e., optimizing architectures for both accuracy and for example model complexity or efficiency), because understanding the trade-off between multiple objectives is arguably more interesting. One objective of this thesis is to explore how simple algorithms fare on multi-objective NAS, by proposing simple, yet well-performing baselines, to ensure that the added value of new (complex) NAS algorithms can be properly validated. Additionally, a NAS benchmark dataset is proposed that is different from existing ones in the way the encoding of a network determines the structure of the network and is made publicly available to facilitate the development of new NAS methods. We ease the computational burden by providing pre-computed performance metrics for trained and evaluated networks in the search space.

Additionally, ablation studies on complex new search algorithms for NAS are often lacking in literature. It is therefore hard to determine what mechanisms of an approach cause the improvement in the NAS process. Various types of search strategies have been investigated in literature. A popular class among them is Evolutionary Algorithms (EAs). These population-based heuristic optimization algorithms, relying on selection and making variations of encoded candidate solutions, are particularly suited to deal with rugged objective spaces. Furthermore, EAs have proven themselves very capable for multi-objective optimization. In this thesis, NAS is considered mainly in the context of EAs. A particular class of EAs are model-based EAs, which are aimed at detecting and making use of structure within a problem. Specifically, linkage models are learned from the solutions found so far, in order to capture and exploit dependencies between problem variables. Layers of a network are connected to one another, which could potentially cause dependencies within a network's encoding (which is often layer-wise), making model-based EAs interesting optimization algorithms to apply to NAS problems. A state-of-the-art model-based EA is the Multi-Objective Gene-pool Optimal Mixing Evolutionary Algorithm (MO-GOMEA), which relies on recombination of solutions within clusters, as well as capturing and exploiting possible linkage that exists between variables in the encoding. This thesis will explore which mechanisms are necessary to get excellent performances in searching NAS spaces, by ablating MO-GOMEA on a multi-objective NAS setting. This should provide insight in how to approach and successfully perform NAS using model-based EAs.

1.1. Research Objectives

First, it is investigated how well multi-objective NAS can be performed with simple algorithms, to get insight in how difficult optimizing an architecture actually is. Thus, the first objective of this thesis is:

Research Question 1. *What simple, yet well-performing algorithms can provide a solid baseline in multi-objective neural architecture search?*

Second, since the lack of ablation studies conceal which mechanisms are crucial to get excellent performances in searching NAS spaces, investigating this is the second objective of this thesis:

Research Question 1. *What mechanisms make a state-of-the-art multi-objective evolutionary algorithm perform well on a multi-objective neural architecture search problem?*

1.2. Outline

This thesis is organized in the following way. Chapter 2 covers the background material that forms the basis of the research. Specifically, it will explain evolutionary algorithms, convolutional neural networks and neural architecture search in more detail. Chapter 3 describes the search spaces and experimental setup used in the experiments in Chapters 4 and 5. Then, Chapter 4 will include experiments and results to answer Research Question 1, by giving insight in two proposed baselines for NAS to compare new algorithms against, also by referring to a preprint written as part of this thesis. Chapter 5 provides experiments and results to answer Research Question 2, by providing an ablation study of MO-GOMEA on NAS. The discussion containing all insights gained from this thesis is found in Chapter 6. Additionally, some pointers for future work are given. Lastly, Chapter 7 concludes the findings in the thesis and provides a prospect on the field of NAS for the nearby future.

1.3. Abbreviations

This section lists all abbreviations and acronyms used throughout the thesis

Abbreviation	Meaning
(C)NN	(Convolutional) Neural Network
DL	Deep Learning
EA	Evolutionary Algorithm
LS	Local Search
MB-EA	Model-Based Evolutionary Algorithm
ML	Machine Learning
MO-EA	Multi-Objective Evolutionary Algorithm
(MO-)GOMEA	(Multi-Objective) Gene-pool Optimal Mixing Evolutionary Algorithm
NAS	Neural Architecture Search
NSGA-II	Non-dominated Sorting Genetic Algorithm II
RS	Random Search
SOTA	State-Of-The-Art
USRS	Uniform Size Random Search

2

Background

In this chapter, the underlying theory and notions, as well as the reformulation of the research objectives in this thesis will be provided. Section 2.1 will explain about evolutionary algorithms, how they work and what their strengths are. Section 2.2 explains the specifics of neural networks, with a particular focus on convolutional neural networks. Subsequently, Section 2.3 will give an overview of the current state of research on neural architecture search, as well as an explanation of the process and the factors involved. Finally, the research questions that will be addressed by this thesis are reformulated in Section 2.4.

2.1. Evolutionary Algorithms

Evolutionary algorithms (EAs) [23] are stochastic local search algorithms for optimization, containing mechanisms that are inspired by the process of natural evolution. Populations of encoded solutions are improved throughout generations by utilizing operators that enable exploitation and exploration of the search space, such as cross-over, mutation and selection. Every individual has a *fitness*, an indicator which tells how good (or "fit") a solution is, obtained by evaluating the fitness function on the encoding of a solution. New generations are being evolved until a termination criterion is met, which could be, e.g., a solution of the desired quality is found, the optimization has exceeded a number of evaluations or time limit, or all diversity is lost in the population.

Many forms of EAs exist, among others Genetic Algorithms [18], Genetic Programming [33], Evolution Strategies [6] and Evolutionary Programming [16]. EAs exist for single-objective as well as multi-objective problems, for discrete as well as continuous problems. This thesis focuses mainly on Genetic Algorithms (GAs), which are applied on problems where solutions are encoded as fixed-length discrete vectors (similar to most NAS settings).

2.1.1. Encoding

Candidate solutions, also called *individuals*, are encoded into *genotypes*: strings of ℓ variables ($[x_1, x_2, \ldots, x_\ell]$), either continuous or discrete, depending on the optimization problem. These genotypes contain the *genetic information* about the solution. Encodings can be direct or indirect: with direct encodings, the values for the variables directly represent the properties of a solution, whereas with indirect encodings, the values represent a sort of blueprint, in order to "generate" the solution. The encoding $\boldsymbol{x}$ is directly the input for the fitness function $f(\boldsymbol{x})$, which is related (and in some cases similar) to the objective function and is specific to the optimization problem. For example, a very simple discrete (in this case, binary) optimization problem is One-Max, in which solutions that contain more ones are considered better and the global optimum is the all-one solution. The fitness function is formulated as $f(\boldsymbol{x}) = \sum x_i$, and is illustrated for three direct encodings in Figure 2.1.

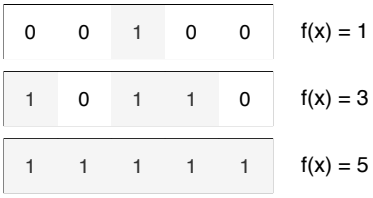


Figure 2.1: Three encodings and their fitness on the One-Max problem

2.1.2. Mechanisms of EAs

After choosing a population size n, the first *generation* of n individuals is initialized, usually by randomly sampling each variable x_i in the encoding from its domain $\mathbb{A}_i$. Every individual is evaluated and assigned its fitness. Note that this causes a warm-up phase without any directed search in the beginning, effectively acting like random search during the first n evaluations. Then, variation and selection operators are applied to the initial population, to perform exploration and exploitation of the search space. The process of EAs is visualized in Figure 2.2, although the order and specifics of operators can vary among different EAs. The upcoming paragraphs describe commonly used operators for EAs, with a particular focus on mechanisms operating on fixed-length discrete vectors.

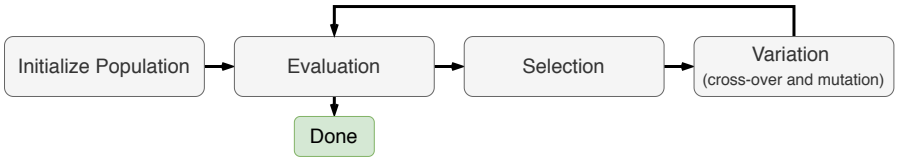


Figure 2.2: General process of EAs.

Selection

Selection operators resemble the concept "survival of the fittest" and are used to select the most promising (fittest) individuals for reproduction, thereby improving the mean fitness of the population. A popular selection operator is tournament selection, in which a group of k individuals is randomly selected from the population; the fittest individual of this group is the winner and is allowed to reproduce. The selection pressure can be increased by increasing the tournament size k, resulting in faster convergence. This does not automatically mean that a higher selection pressure is better, because it can result in *premature convergence*, i.e., losing all diversity in the population without achieving the global optimum.

Alternatively, other selection mechanisms exist. Truncation selection selects the fittest k individuals for reproduction, therefore not giving weak individuals any chance, even though they could still carry potentially good genetic material. In roulette-wheel/proportionate selection, k individuals are randomly selected from the population with a probability proportionate to their fitness. The downside is that the population is quickly dominated by individuals that have a far-above average fitness and it does not ensure elitism. Other, more specialized and problem-catered selection mechanisms exist. For example, NSGA-II, a multi-objective GA, selects solutions based on their rank and crowding distance (inverse density around a solution), to ensure a balanced selection of individuals (according to all objectives).

Variation

Variation is classically realized via *cross-over* and *mutation*. Cross-over recombines two parent individuals to create offspring by exchanging parts of the solution. How genetic material is exchanged varies for different cross-over schemes. Univariate cross-over is the most common, in which every variable has an independent probability of $p = 0.5$ to exchange values between parents. A cross-over operator that preserves more structure of the reproducing parents is 1-, 2-, or k-point cross-over, where k indices are chosen within $[0, \ell]$ that denote the cutting points. The genetic material between the cuts is exchanged per block (see Figure 2.3, top-left). The result of cross-over are two *offspring* individuals, both containing genetic material from either parent, hopefully combined in a way that the good genetic material from both parents are gathered in the same offspring individual. Figure 2.3 (left) visualizes univariate and 2-point cross-over between two parent individuals.

To increase the likelihood of a successful cross-over, model-based EAs attempt to learn, during an optimization run, an understanding of effective ways of recombining individuals, such that fewer

evaluations of the objective function are necessary to evolve good solutions. For instance, a pre-defined structure is learned from the statistics of the population in order to adapt the cross-over mask to the optimization problem at hand. Section 5.1 will discuss model-based EAs in more detail.

Another way to perform variation is by *mutation*. Typically, for discrete domains, for each variable x_i, the value is changed to a randomly sampled value from the domain $\mathbb{A}_i$ with probability p_m. Mutation is an especially important variation operator for exploration in continuous domains, to ensure that the values for a variable are not limited to the initialized values of the n individuals in the first generation. Mutation can also be valuable for discrete domains, because it prevents premature convergence of the GA when all diversity is lost. Figure 2.3 (right) shows the result of a mutation of an individual.

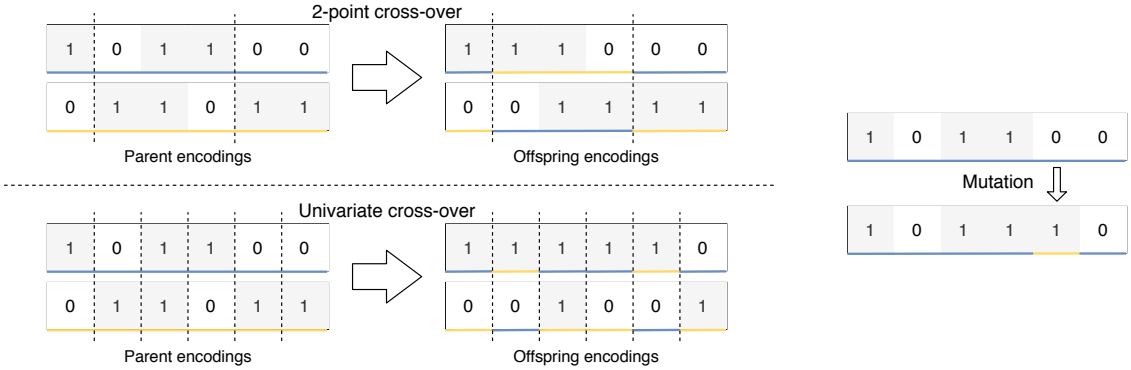


Figure 2.3: 2-Point and univariate cross-over (left) and mutation (right).

2.1.3. Multi-Objective optimization

There exist EAs for multi-objective (MO) optimization problems, and are commonly accepted as being particularly effective in tackling these. MO-optimization problems have multiple objective functions that need to be optimized, e.g., for a solution $\boldsymbol{x}$, the objective function is $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ..., f_m(\boldsymbol{x}))$ for m objectives. Since these objectives usually conflict, there exists no single solution (a *utopian* solution) that optimizes all objectives at once. Therefore, there exists a front of multiple optimal solutions, all with a different trade of the objectives.

Therefore, it is not straightforward how to define which solution is better, which for single-objective optimization can be done by just comparing scalar values. Therefore, the definition of (strict) Pareto domination is used: solution $\boldsymbol{x}$ dominates solution $\boldsymbol{y}$ (written as $\boldsymbol{x} \succ \boldsymbol{y}$) if and only if $\forall i \in \{1, 2, ..., m\} : f_i(\boldsymbol{x}) \geq f_i(\boldsymbol{y})$ and $\boldsymbol{f}(\boldsymbol{x}) \neq \boldsymbol{f}(\boldsymbol{y})$. The optimal front of solutions obtainable in a multi-objective problem is defined by the Pareto Set, i.e., $\mathcal{P}_S = \{\boldsymbol{x} \mid \nexists \boldsymbol{y} : \boldsymbol{y} \succ \boldsymbol{x}\}$, that consists of all non-dominated solutions in the search space.

Multiple metrics exist to assess the performance of MO-optimization algorithms, because the spread, proximity and density of the obtained front of solutions are all important. Section 3.4 explains why we choose the hypervolume to evaluate and compare different algorithms. EAs perform well on MO-optimization problems partly because the individuals in the population can explore multiple parts of the objective space simultaneously.

2.1.4. Properties of EAs

The fact that EAs optimize on encodings, which in turn are evaluated with a black-box objective function of which the EA requires no further knowledge, the complexity of the problem is in terms of application decoupled from the complexity of the optimization algorithm. It is therefore possible to tackle a complex problem with a rather simple EA. This makes EAs particularly attractive for practitioners for problems that are difficult to model, as EAs do for instance not require gradients to function properly, but merely only require a mapping between the solution and its encoding, as well as an objective function. However, when gradients are available, algorithms that leverage those are usually more efficient, because EAs tend to evaluate many more solutions during the optimization in order to detect and leverage the same information. Furthermore, the fact that EAs are population-based causes them to explore different parts of the search space (and objective space in MO-optimization) simultaneously and therefore makes them robust on rough fitness landscapes with (many) local optima [54]. Additionally, the population-based

property of EAs enable parallelization potential in terms of evaluating multiple individuals concurrently.

These properties make EAs powerful and versatile, and enable them to get good results on a wide range of real-world problems [47, 48] (an overview of recent applications is found in [70]). Also, considering that NAS is mainly performed in discrete search spaces, the class of GAs should be a suitable candidate.

Nevertheless, the configuration of an EA influences its performance, therefore some tasks remain that are not always straightforward to tackle: 1) choosing the right population size and 2) specifying the variation operators that align well with the problem at hand. The first problem is addressed by employing population-sizing schedules [21], which will be explained in Section 5.1.2. Model-based EAs try to solve the latter problem, by attempting to capture and exploit inter-dependencies between variables, therefore adapting the configuration of operators to the inherent structure of the optimization task at hand. Two examples of state-of-the-art model-based EAs are GOMEA [76] and its multi-objective variant MO-GOMEA [45], which will be covered more thoroughly in Section 5.1.

2.2. Convolutional Neural Networks

At the center of the field of DL are neural networks: computation models consisting of connected *layers* that can learn to perform a specific task, by training the model on many examples of correct behavior on the task. A variety of different types of layers (e.g. fully connected, convolutional, recurrent) exist to give a network specific properties. Connecting many layers gives the network a certain depth, hence the name *deep* neural networks and *deep* learning. Many types and shapes of neural networks exist[1], each one designed with a specific purpose. A rather popular class of neural networks are Convolutional Neural Networks (CNNs). CNNs are particularly effective for computer vision tasks, like image segmentation [62], i.e., locating an object in an image (e.g., marking a cancerous cell in a CT-scan), and image classification [38, 65] (e.g., classifying the tissue in a CT-scan as healthy or unhealthy). In this thesis, only feed-forward convolutional neural networks (CNNs) [35] for supervised (i.e., labeled data is given) image classification tasks are considered. Like for fully connected neural networks, in order to train a CNN to perform a task well, many (annotated) training examples are required.

2.2.1. Structure

The inputs of a neural network are the gray-scale or RGB-values of the pixels of the image to be classified. For each layer, the output of the previous layer is processed and passed through to the next layer. This is repeated consecutively for every layer, until there is a final predicted classification at the output layer. Depending on the type of layer, different operations are performed. CNNs usually contain three different types of layers: First, *convolutional layers* and *pooling layers* are stacked in an alternating pattern, and the final (few) layers are often *fully connected* (or simply called *dense*) *layers*. Figure 2.4 shows a typical CNN architecture. Each of the layers will be explained in the paragraphs below.

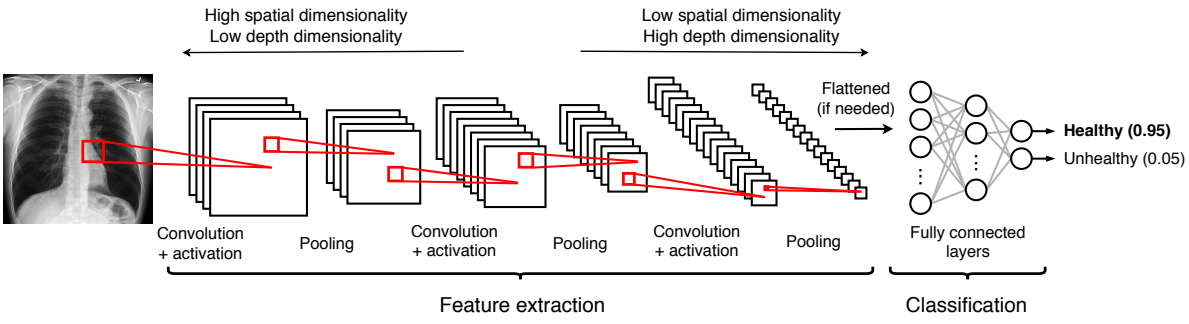


Figure 2.4: Typical architectural scheme for a Convolutional Neural Net, with two possible classes (healthy and unhealthy tissue) for the input image in this example. X-ray input image source: `https://radiopaedia.org/cases/normal-chest-x-ray`

[1] `https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464`

Convolutional Layers

Convolutional layers give the CNN its ability to recognize patterns and objects. The input of convolutional layer is a $D \times H \times W$ image, with D being the depth and W, H being the spatial dimensions width and height respectively. For example, the input for a CNN could be an RGB-image, in which the depth corresponds to the three color-channels. Convolutional layers contain learnable (convolutional) filters of a specified kernel size k, which are applied to every layer of the input in a sliding window manner, in order to produce a *feature map* as output. That means, the receptive field (of size $k \times k$) is moved over the values of the input, applying the filter on the values within the receptive field, resulting in one output value for each position in the feature map. To illustrate this, the filter in Figure 2.5 is applied on every colored square in the input image, with the resulting value in the output feature map shown in the respective color.

In practice, a convolutional layer often applies more than one filter on the input in order to increase the depth D of the resulting feature map. Since the filters are shared for different areas of the input, convolutional layers contain far less trainable parameters than full-connected layers.

Mathematically, the output for position i, j in the feature map with input image $\boldsymbol{X}$ and filter $\boldsymbol{W}$ with kernel size k is defined as $f_{i,j}(\boldsymbol{X}, \boldsymbol{W}) = \sum_{n=0..k-1} \sum_{m=0..k-1} \boldsymbol{X}_{i+n,j+m} \cdot \boldsymbol{W}_{n,m}$. A convolution with a filter of kernel size 3×3 over an image of size 2×2 is shown in Figure 2.5.

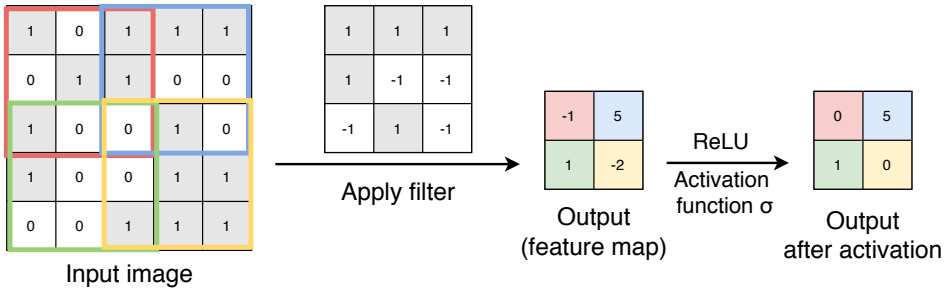


Figure 2.5: Example of a convolution with a filter of size 3×3 on an input image of 5×5, applied with stride 2. Four convolutional operations result in a feature map of 2×2, on which the activation function is applied (element-wise). The filter has learned (by training) to activate on the pattern in the blue area of the input, resulting in a high value at the position marked with blue in the feature map. Discrete values are used for simplification, but in practice every value is continuous.

Convolutional layers can be parametrized with *stride*: this allows the sliding receptive field to take larger steps on the input, thereby lowering the spatial dimensionality of the resulting feature map. For example, stride 2 is applied to the convolution in Figure 2.5. Notice that a convolution (also without stride) may change the output's spatial dimensions. A common way to prevent this from happening is to use padding, such that the output has the same spatial dimensions as the un-padded input. For example, the input image can be padded with 0's around the borders, or the image can be mirrored at the border as padding. Since padding prevents the spatial dimensions from reducing (too quickly), it allows for the construction of very deep networks. Additionally, padding allows to preserve the information at the borders, which is otherwise not possible. Furthermore, padding can be used to make sure the convolution is calculable.

A non-linear activation function σ is applied to the output of a convolutional layer. Convolutions are linear operations, so without non-linearity in-between two convolutional layers, these convolutions can effectively be replaced by one convolution. By extension, the network would only be able to learn linear relations between inputs and outputs, therefore greatly reducing the complexity of patterns it can discern. The CNN's learning ability is clearly limited without these non-linear functions. The most prominently used non-linear function is the *Rectified Linear Unit* (ReLU), but others (*sigmoid*, *tanh*, *Leaky ReLU*, etc.) are also used in practice.

Stacking convolutional layers gives the network the ability to recognize things. The first few layers will recognize low-level features like lines and very basic shapes. The deeper in the network, the more high-level features can be recognized, like objects, constructed from several low-level features.

Pooling layers

Pooling layers down-sample a feature map's resolution by performing a function on a window of values (of a specified size). Different pooling functions exist, with average pooling and maximum (max) pooling

being the most frequently used, which take the average and the maximum of all values in the window respectively. Both are exemplified in Figure 2.6. These pooling layers reduce noise and compress the output, therefore enabling a CNN to have less parameters in total, since subsequent layers are dealing with inputs of smaller spatial dimensions. This compression ability also makes a CNN more robust to small variances in exact position and composition of features, which can occur in images by shifting and rotation transformations of the objects of interest. Similarly to convolutional layers, a stride can be applied to compress the output even more. An important difference between pooling and convolutional layers is that pooling layers are specified, while (the filters in) convolutional layers are learned.

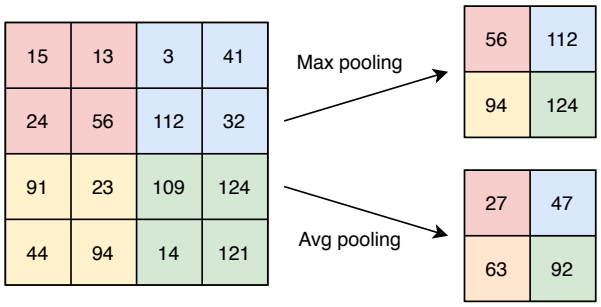


Figure 2.6: Example of 2×2 maximum and average pooling on a 4×4 image with a stride 2.

Fully connected layers

Fully connected layers are frequently used as the last (few) layers in a CNN. There is a weighted connection from every input node to every output node and therefore these layers can exhibit enormous amounts of trainable parameters. As with the output of convolutional layers, an activation function σ is applied to each calculation to ensure non-linearity. The value for the output of a node can be described as function f_i that involves a matrix multiplication of input image (or feature map) $\boldsymbol{X}$ with the weights $\boldsymbol{w}_i$ of the connections from each input node to the output node i, a bias term b_i and an activation function σ:

$$f_i(\boldsymbol{X}) = \sigma(b_i + \boldsymbol{w}_i^T \boldsymbol{X})$$

The fact that both this operation and convolutions can be described as matrix multiplications, is one of the reasons for the successes of neural networks, because these calculations can be efficiently performed by GPUs.

The combination of convolutional and pooling layers cause the input of large spatial dimensions and small depth to transform into feature maps with small spatial dimensions (recall that both convolutions and pooling layers have a compression ability) and large depth. Actually, the stage before the fully connected layers acts as a feature extractor in CNNs. Now, it is the job of the so-called dense layers to perform a classification on it, by transforming the extracted feature maps into the desired output dimension. For image classification with n classes, it is often desirable to output a prediction probability vector of n values. This can be considered as the prediction confidence score for a class and can be realized by applying a Softmax function on the values of the output nodes. Outputting a prediction probability vector instead of a one-hot vector can support with training the network, as explained in Section 2.2.2. In practice, either the class with highest confidence can be the output, or the choice can be passed to a practitioner when the network is uncertain of its classification result.

2.2.2. Training

Convolutional neural networks, like many machine learning techniques, require many training examples to perform well. Research on training methods have yielded efficient ways to train neural networks. For example, *Stochastic Gradient Descent (SGD)* is a particularly effective method, which, combined with the *backpropagation algorithm* [64], has become the standard approach for training (convolutional) neural networks. As a first and one-time step, all weights in the model to be trained are initialized randomly. Then, a training example is picked randomly from the training set and evaluated using a *forward pass* (i.e., we just classify the image with the untrained network). The classification output of the network, together with the desired classification is used to calculate a *loss*. Different loss functions can be used. A common one is cross entropy loss, which penalizes confident but wrong classifications

more severely than slightly wrong classifications (recall, the output of a network is a probability vector on the classes in the dataset). With the loss calculated, the *backpropagation algorithm* can be applied. The gradient of the loss with respect to the weights is calculated layer by layer in the *backward pass*, making efficient use of the chain rule. This gradient is then used to update the weights in the correct direction, according to some learning rate. This procedure is repeated multiple times for every example in the training set (one cycle over all training samples is called an *epoch*), until a satisfactory classification accuracy by the network is obtained.

Although *SGD* is already a powerful tool to train networks, researchers have invented extra mechanisms that can improve the training procedure. Examples are learning rate decay and schedules, RMSProp, Adam [31], batch normalization [26] and data augmentation tricks. While improvements of efficiency in training procedures have been an enabling factor in the massive adoption of deep learning techniques and their applications, training a network can still be considered an expensive procedure. For example, training a neural network can take a few minutes to hours or days on end, depending on the task, training procedure and implementation (GPU vs CPU).

2.2.3. Generalization

Similarly to many ML techniques, CNNs are susceptible to *overfitting*. Models that are too complex might overfit on the training data by learning too specific patterns, and therefore affect the ability to generalize beyond the training data. A commonly used technique to get a sense of overfitting of a trained network is by using a validation set. This is a subset of the available training data, which is held-out from the training procedure. Measuring the classification accuracy on this dataset will give an insight on the network's performance on unseen data. The consequence is that less data is used for training, so to improve upon this, (k-fold) cross-validation can be used. The training data is segmented into k parts. The desired model is trained from scratch on all but one part and validated on the left-out part. This is done for all k parts and the final validation accuracy is the average of the classification accuracies of all k trained models on their respective validation data.

However, while these methods give an indication of the generalization ability of models, it does not solve overfitting. According to the principle of Occam's razor, when one has to choose between two different models (hypotheses) that perform equally well, the one based on the fewest assumptions should be favored, hence the least complex model should be chosen. This also applies to neural networks: when making the network (the model to learn) too complex, the network is likely to overfit on the training data, therefore not generalizing well beyond it. So-called *regularization* techniques exist to combat overfitting, examples are ensembling (combining the outcome of multiple trained models), dropout (disabling parts of the network during training) and early stopping (prematurely stopping the training to prevent learning too specific patterns).

On the other end of the spectrum is underfitting, which happens when the model is too simple to perform well on the training set. The model does not allow to learn complex decision boundaries, therefore the prediction accuracy is poor on both the training and the test set. Overfitting and underfitting are both shown in Figure 2.7. The ideal model complexity range that is shown is not always straightforward to know beforehand. Since the model complexity is mainly determined by the architecture, proper architecture design is crucial for a good performance.

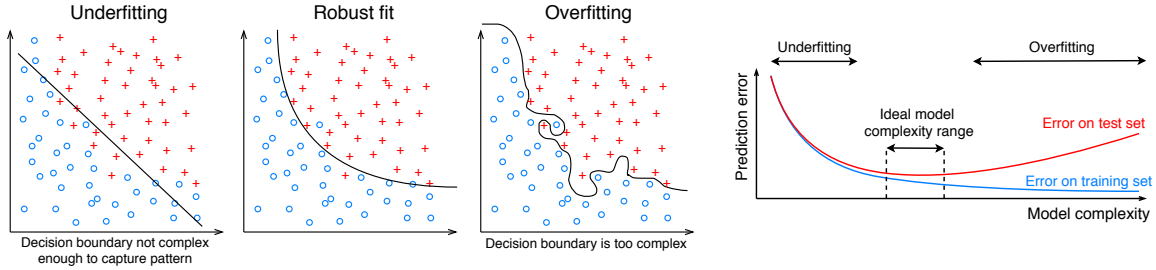


Figure 2.7: Left: Decision boundaries for underfitting, robust fitting and overfitting models on a binary classification task. Right: Prediction error for the training set and testing set for fully trained models of differing complexity.

2.2.4. History of CNNs

The first appearance of a CNN is in [35], in which it was used to automatically read handwritten digits of zip codes. Iterations upon this idea resulted in LeNet [36], a CNN designed to classify handwritten characters, which worked better because of the appliance of convolutional, pooling and non-linear layers in sequence. Then, after the interest in neural networks cooled down for a while, AlexNet [34] showed the potential of CNNs by winning ILSVRC (an annual image recognition contest) by a large margin in 2012, using a much wider and deeper version of LeNet. This sparked a true Deep Learning revolution, as innovations were following up fast. GoogLeNet [74], proposed in 2014, greatly reduced the required amount of parameters by introducing an innovative architectural block of layers called the Inception Module. In the same year, VGGNet [69] was introduced, which showed the importance of depth in networks. In the subsequent year, ResNet[22] won the ILSVRC with the novel architectural feat of residual connections, which skip layers to combat the vanishing gradient problem to enable the training of very deep networks. In 2016, DenseNet [25] was proposed, which connects every layer within the CNN.

In the recent years, a movement has arisen to make (C)NNs more light-weight and accessible for embedded devices, e.g., to deploy them on mobile devices. In this direction, multiple versions of MobileNet [24] are proposed, which aim to significantly decrease the number of parameters without compromising too much in accuracy, by decomposing the standard convolution into a depth-wise and a point-wise convolution. The MacroNAS search space that is proposed in Chapter 3 makes use of efficient inverted bottleneck layers from MobileNetV2 [66]. Other efforts are spent on decreasing the computational requirements, which resulted in for example ShuffleNet [49].

For a more thorough analysis of CNNs over the past recent years, the reader is referred to [1] and [29].

2.2.5. Architecture

Before training a CNN, the architecture[2] must be decided. The number of convolutional layers and pooling layers, their hyperparameters (kernel size, stride, etc.), extra (residual) connections, the activation functions are all variables that influence the performance of a CNN. Proper architecture design is hard, it requires experience and often involves (expensive) trial and error. Some breakthroughs in DL were the result of the invention of novel architectural feats as mentioned in Section 2.2.4. For both of these reasons, it is not surprising that people try to automate the process of choosing a suitable architecture. The field of Neural Architecture Search (NAS) focuses its attention on improving this automation and will be explained and discussed in the next section.

2.3. Neural Architecture Search

Neural Architecture Search, a sub-field of AutoML, is a relatively young field that focuses its research on the automation of neural network architecture design. The drive to automate the difficult, time-consuming and error-prone task of choosing a suitable architecture is fueled by the desire to find new state-of-the-art performing neural network architectures and structures. It is straightforward to see that advances in NAS have the potential to revolutionize the field of Deep Learning, if we take into consideration that some breakthroughs in DL were the result of the invention of new types of architectural feats (e.g. think of residual connections in ResNet [22]).

Early works [82] were concerned with the optimization of the wiring of separate nodes, as well as optimizing their weights, mainly by employing evolutionary algorithms [71], which led to the name *neuroevolution*. However, as neural networks got bigger over the years, optimization of the weights themselves could be done much more efficiently by applying backpropagation to utilize gradients. The works [90] (2016) and [58] (2017) first showed the true potential of optimizing the architecture of modern neural networks with reinforcement learning and evolutionary algorithms respectively, although requiring thousands of GPU hours. Nevertheless, this caused a true explosion of interest in this field and it has been quickly gaining popularity since. This resulted (among other things) in Amoeba-Net, the first ever NAS-optimized architecture to beat state-of-the-art handcrafted architectures in 2018 [59].

[2] In the remainder of the thesis, "architecture" and "neural network" are referred to interchangeably.

2.3.1. Decomposition of NAS

According to [14], three core concepts can be distinguished in NAS. First, for performing NAS, a *search space* needs to defined, to delimit the types of networks to consider and to keep the search tractable. The *search strategy* will perform an optimization on this search space, by making use of the score for a network obtained by the *performance estimation*. The overall process of NAS is visualized in Figure 2.8. Although this thesis will focus mainly on the search strategy part of NAS, this section will address each concept in detail.

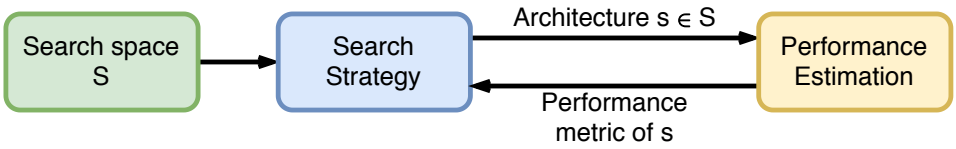


Figure 2.8: Three components of the NAS process and their relation.

Search Space

Search spaces can be roughly divided in macro and micro-level architecture spaces [14] (also referred to as global and cell-based search spaces respectively [60, 80]). In the former, the composition of different types of cells is optimized. Cells are computational graphs themselves, containing one or multiple operations and/or layers, but the contents of the available types of cells are fixed throughout the search. Figure 2.9 (left) shows an example of a macro-level architecture space. In micro-level architecture search (Figure 2.9, right), it is the cells that undergo optimization. These optimized cells appear repeatedly in the global architecture, which in turn is manually fixed. Micro-level search therefore allows no networks with different structures in the beginning and deeper in the network. Additionally, micro-level search spaces involve more prior knowledge by manually giving shape to the overall architecture, therefore inhibiting the discovery of truly novel architectures.

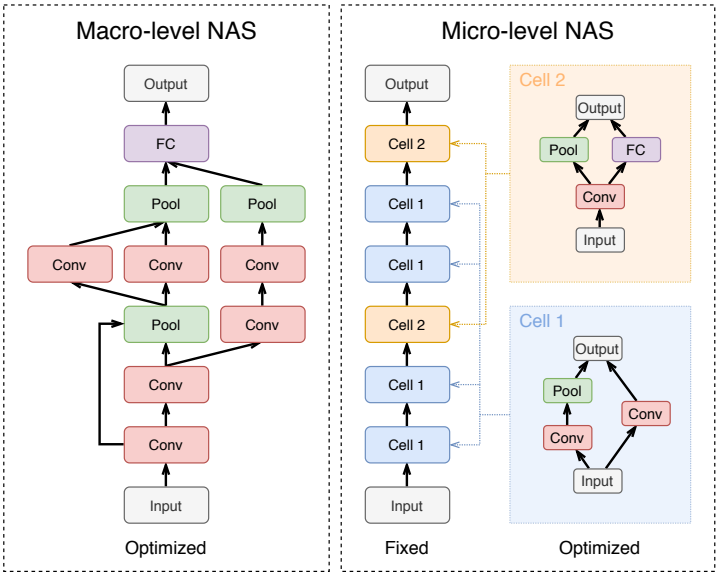


Figure 2.9: Examples for macro-level (left) and micro-level (right) architecture search spaces

Early works on NAS [58, 72, 81, 90] considered mainly macro-level search spaces, but inspired by the repeating structures in GoogLeNet and ResNet, some works [89, 91] have discovered that optimizing blocks which are repeatedly stacked in an overall architecture greatly reduced the size of the search space, while still yielding greatly performing architectures. In fact, the NAS-optimized architecture to outperform hand-designed architectures for the first time was the result of searching a micro-level search space [58]. It also enables to find a good-performing cells on a subset or easier task of the designated task and then transfer these cells into a larger overall architecture to tackle the larger/more difficult task [91]. These advantages shifted the tendency to use micro-level search spaces instead of macro-level

search spaces. However, since another trend in NAS emerged, optimizing architectures multi-objectively to discover efficient and compact networks, more flexibility of the architecture is required, for which macro-level search spaces are more suited. Also, keeping in mind the goal of discovering truly novel architectures, macro-level search spaces allow more freedom, although specifying the search space to allow many options while keeping it tractable remains a challenge. It is easy to incorporate architectural structures that are known to perform well, but hard to allow and encode the freedom required to discover new well-performing structures, which also result in computable networks; the network should always be a directed acyclic graph that maps the input dimension to the output dimension.

Alternative, hybrid types of search spaces exist, in which both the cells and their overall composition are searched [52]. Additionally, some search spaces also allow the hyperparameters of the training process to be optimized [87].

The types of search spaces mentioned so far are all discrete search spaces. There exists another type of search space, one that is formulated in a continuous and differentiable manner and therefore allows for efficient gradient-based approaches to search for well-performing architectures. The results in [43] show that searching the continuous relaxation of the architecture space enables more efficient optimization of the architecture, although the overall architecture is already fixed to some degree at the start of the procedure.

This thesis will focus primarily on discrete macro-architecture search, in order to pursue NAS that allows most freedom in its architecture space. Therefore, the works that are mentioned in the remainder of the thesis are mainly approaches that tackle macro-level search spaces.

Search Strategy

The search strategy is the algorithmic part of a NAS approach that is responsible for finding good architectures in the search space, by making use of performance estimations of considered networks. Since the performance estimation is the most expensive part of NAS, it is affordable to use a "costly" search strategy. That however does not mean that we should, but the extra searching power it provides could help to reduce the number of networks that need to be evaluated to discover an architecture of the desired quality. Search strategies employed in NAS approaches vary, with evolutionary algorithms (EAs) and reinforcement learning (RL) as the two most popular choices.

With RL-based approaches, the creation of the architecture is modeled using states, actions and rewards that are used to improve the policy, i.e., the sequential decision making process. In [90] a second network (a recurrent neural network, RNN) is trained to sequentially sample a string (encoding), that is transformed into a candidate architecture, which is trained and evaluated. Its performance is used as a feedback signal to train the RNN to improve its sampling process. In another approach [9], NAS is formulated as an uninterrupted decision making process, in which a partially trained network acts as the state. The actions by the agent mutate the network, after which the network is trained further. Its estimated performance is used as reward for improving the policy. Many more RL-based approaches exist, among other works [3, 89].

Using EAs, the architectures in the search space are encoded and optimization on a population of encodings should lead to the desired exploration and exploitation of the search space. Operators within EAs (the type of cross-over, mutation and selection operators) vary wildly, so it is candid that many different configurations of EAs are tried as search strategy. EAs are very capable of dealing with objective functions that contain many local optima and thrive on multi-objective problems. Additionally, EAs offer a parallelization potential for training multiple architectures in a generation at once. But, they also suffer from overhead at the initialization of the first generation, i.e., the evaluation of architectures that are randomly picked without any directed search (Section 5.1.2 discusses a method to address this issue, however). Nonetheless, in [59] it is shown that when comparing RL, Evolution and RS on NAS, EAs have better anytime performance.

The works in [73, 81] use a Genetic Algorithm to optimize macro-level search spaces and achieve state-of-the-art prediction accuracies (at the time of publication), while [72] uses a Cartesian Genetic Programming approach to enlarge the flexibility of architectures in the search space. The work in [58] uses an EA that is reminiscent of NEAT [71], but is scaled-up to handle cells suitable for image recognition. Also on micro-level search spaces EAs remain a popular search strategy, e.g., among other works [59, 79]. More single-objective NAS approaches are summarized in Appendix E and for a complete overview that also includes multi-objective (EA-based) NAS approaches, the reader is referred to [17].

Various other search strategies exist, like Bayesian optimization [28], Monte Carlo Tree Search [78] and gradient-based approaches [43].

This thesis focuses primarily on evolutionary algorithms as search strategy for NAS and therefore in the remainder of the thesis, mainly EA-based approaches are mentioned. The most relevant EA-based approaches for multi-objective NAS are discussed in Section 2.3.2.

Performance Estimation

The performance estimation is yet another component that increases the number of possibilities to approach NAS. To get an estimation of how good an architecture is, it needs to be trained and evaluated, or at least, this is the most common approach. Therefore, this component specifies the most expensive part of NAS. The better the training of a network, the more reliable its performance is estimated, but a trade-off needs to be made to keep the NAS approach feasible.

Initial NAS techniques train every network fully [81, 90], such that no final refinement is necessary and the search can be stopped at any moment, yielding a fully-trained network. However, the result is that thousands of GPU hours are required to produce an architecture of the desired quality [90]. Some approaches [4, 41] make use of a lower-fidelity estimate of networks, e.g., by training it only for a few epochs, as this can already give an indication of its potential. The finally selected model is trained fully. For example, in [41], a surrogate model is trained to predict the performance of converged networks based on partially trained networks. This technique biases towards fast learners however [52].

To ease the burden of training every network from scratch, the field has been looking into weight-sharing techniques [8, 56, 58]. Particularly for approaches that "grow" networks [72], inheriting weights whenever possible can cut training costs [58], although the final ranking of networks could deviate from the true ranking when every network is trained from scratch [86]. Such approaches should obviously be carefully adjusted to the search strategy, to make sensible use of the efficiency it can offer.

Another weight-sharing technique is the one-shot approach [5, 59]: one *supernet* is trained, which contains all architectures in the search space as sub-networks. The large network needs to be trained only once and when performing the search, the sub-networks, which inherit their weights from the *supernet*, are (possibly fine-tuned briefly and) evaluated. Since the evaluation of a network is much cheaper than training it, this technique cuts the training costs significantly. A graphical illustration of a one-shot approach is shown in Figure 2.10.

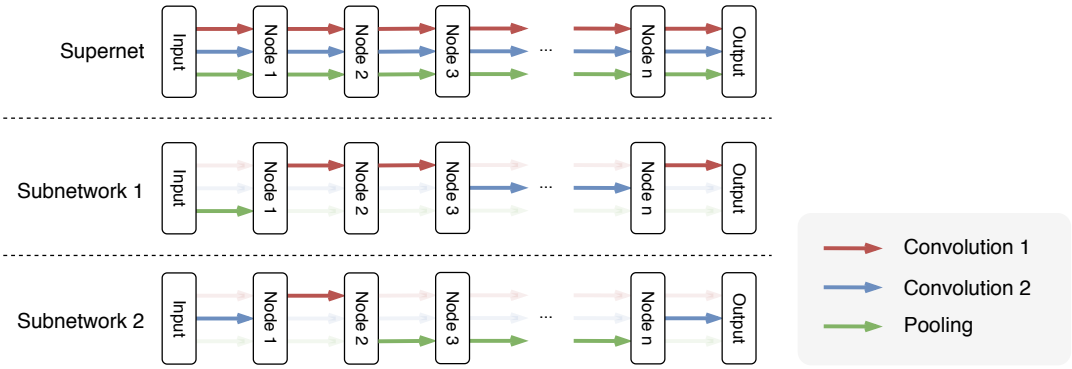


Figure 2.10: One-shot NAS, including pooling and two differently configured convolutions as options for each transition between nodes.

The evaluation of networks is influenced by training hyperparameters like the learning rate (schedule). There are few approaches that incorporate the optimization of these training hyperparameters also into the search [52]. The performance estimation of networks obviously only has access to training and validation data, so the performance of architectures on test data remains unknown during the search.

Overall NAS routine

In practice, the boundary between components is not as clear-cut as just selecting one for each component. Certain components have impact on other components, e.g., a GA expects a specific structure of the encoding of the search space to perform the search on, and as lower-fidelity performance estimations can possibly favor smaller models, this affects the search strategy. Additionally, the eventual

simultaneous optimization of training parameters also influences the search. Therefore, it is important to keep an eye on the overall NAS process by carefully combining the components.

Nevertheless, due to the many different possibilities for each of the components of NAS, there exist an infinite number of different setups for NAS. Additionally, NAS is expensive in nature, although recent publications have been working towards reducing costs of performing NAS [51, 75]. These two aspects of NAS combined make it hard to compare different approaches. The lack of validation and comparison between NAS techniques, which can lead to undeserved crediting of certain components of an approach when achieving new state-of-the-art results, is discussed in Section 2.3.4.

To evaluate NAS approaches, the most commonly used performance indicator is the validation accuracy of the best network found so far against wall-clock time, while also accounting for any parallel search routines, if utilized. In fact, this is also recommended according to the "best practices for research on NAS" paper [39]. This thesis, however, chooses another metric to evaluate different NAS approaches against, namely to compare against the number of network evaluations, in order to take a more algorithmic perspective on the search strategy. This choice is further motivated in Section 3.4.1.

Although it is necessary to keep an eye on every aspect of NAS during the development of new methods, this thesis focuses mainly on the search strategy of NAS. This is done in order to get a better understanding of how difficult the problem of optimizing an architecture actually is and what properties a search algorithm should have to get a good searching performance.

2.3.2. Multi-Objective NAS

The foremost objective of NAS is to search for well-performing networks, i.e., networks with high prediction and/or classification accuracy. However, as NAS became a more widely recognized field, an interest emerged for optimizing the architecture in other aspects besides its accuracy. For some applications (e.g., the deployment of neural networks on embedded systems, like mobile phones) it is necessary for the network to have a minimized size or latency. Therefore, people have been performing NAS by treating it as a multi-objective optimization problem, to optimize on a secondary objective next to prediction accuracy. Usually the objectives to optimize for are conflicting, therefore there is not one globally best solution. Instead, the global solution is a set of architectures, also called a Pareto Front, which consists of all architectures in the search space with optimal trade-offs (explained in more detail in Section 2.1). Only EA-based approaches are mentioned, although other approaches (e.g. RL) also exist for multi-objective NAS.

In [10], a novel combination of NSGA-II [11] (a multi-objective EA) and RL is used as approach to obtain a front of networks that trade off accuracy and compactness. Also [30] builds upon NSGA-II to find accurate networks that additionally have a rapid inference for deployment on embedded devices. In [13], Lamarckian mechanisms are included in the multi-objective EA as search strategy, in which child networks inherit the weights of parent networks, therefore decreasing the computational requirements for this approach. Although the proposed method is favorably compared with random search, it is unclear whether this is because time is saved on training due to its Lamarckian mechanism, or the EA searches the space more efficiently. In NSGA-Net [44], NSGA-II is used to discover accurate and compact architectures on the same search space as in [81], with an addition of (optimizable) skip connections that bypass entire computational blocks. Although superiority of NSGA-Net compared to random search is shown, the presented ablation study does not make it entirely clear if the improvement is caused by the extra Bayesian Optimization sampling step at the end, or because a more powerful search algorithm is used.

2.3.3. Pre-computed NAS benchmark datasets

In order to promote reproducibility and allow for more thorough analysis of approaches, an emerging trend is the creation of NAS benchmark datasets (also promoted to be used in [39]). By evaluating all architectures in a specified search space and storing the performance indicators, the most expensive part of doing NAS is cached and readily available to optimize on. NAS-Bench-101 [84] is the first proposed NAS benchmark dataset, which contains performance indicators of all architectures in a micro-level search space. Other known benchmark sets are NAS-Bench-201 [12] and NAS-Bench-1Shot1 [88]. As part of this thesis, a paper is written in collaboration with the supervisors, in which MacroNAS-C10 and MacroNAS-C100 are proposed, the first ever NAS benchmark datasets on a macro-level architecture search space. These benchmark datasets are used to conduct many experiments in this thesis. Chapter 3 explains in more detail about this NAS benchmark dataset.

2.3.4. Validation of NAS

There is a global focus on reaching new SOTA neural network performances, while there is fewer attention for which of the actual components are vital to achieve it. As shown by [86] and [83], most optimization algorithms applied in single-objective NAS (i.e., only focusing on prediction accuracy) do not perform significantly better than random search (RS). This is a result of search spaces which are tuned manually in such a way that almost every randomly sampled network is well-performing. This finding is not surprising, since many papers do not use baseline algorithms to compare against. If any is used, it is generally random search. To know if an algorithm is better than the simplest thing one can do (i.e. random sampling), comparisons to baselines are crucial [39].

Since it is debatable whether powerful optimization algorithms are necessary to obtain a decent performance on NAS, it is interesting to see whether simple algorithms are able to get a good performance. This thesis will therefore look into alternative baselines next to RS, that are simple to use, perform reasonably good and can be used to compare future NAS optimization algorithms against, in order to address **Research Question 1**.

Additionally, this thesis will address **Research Question 2**, by focusing more specifically on which parts of a state-of-the-art EA (MO-GOMEA) are vital for excellent optimization of architectures in a multi-objective, macro-level NAS problem. To do so, many experiments are conducted, which make efficiently used of a cached NAS benchmark dataset.

2.4. Reformulation of Research Objectives

The above background material taken as a basis, this thesis will zoom in on two particular topics, in order to answer the research questions that come with these topics. The first topic involves the usage of baselines in NAS (Section 2.4.1), while the second topic is concerned with the performance and behaviour of EAs when applied on NAS (Section 2.4.2).

2.4.1. Baselines for NAS

As shown in recent works [83, 86], many NAS approaches do not perform better than random search. Additionally, comparisons against such a simple baseline is often not even included in new works. In order to truly make progress in research on NAS, baselines to validate approaches are required [39]. While only RS is sometimes considered as baseline, and sophisticated algorithms like EAs are usually the least complex algorithms applied to NAS, there is a large range of heuristics algorithms (e.g. local search) in between, which are, to the best of my knowledge, never considered for NAS. This thesis will therefore explore whether there exist better performing, but still simple algorithms that can provide a baseline for future NAS research.

Research Question 1. *What simple, yet well-performing algorithms can provide a solid baseline in multi-objective neural architecture search?*

Chapter 4 is devoted to answering this research question.

2.4.2. Ablation of EAs on NAS

EAs are a class of algorithms often considered for NAS and while many different EA-based approaches exist, few publications go into detail on which parts of the EA work well for the specific NAS problem at hand [17]. Instead, the focus is often on just finding architectures with a new state-of-the-art prediction accuracy. It is therefore interesting to see what the contribution of each mechanism of an EA actually is and how they should be configured, such that we can design better and more efficient EAs for future NAS problems. This thesis will focus mainly on MO-GOMEA and investigates which of its components is responsible for its behavior and performance. Next to stating the second research question focusing on this, also several sub-research questions are formulated that focus on the specific features of MO-GOMEA, which are motivated in Section 5.1.

Research Question 2. *What mechanisms make a state-of-the-art multi-objective evolutionary algorithm perform well on a multi-objective macro-level neural architecture search problem?*

Research Question 2A. *How does an automatic population-sizing schedule affect the performance of MO-GOMEA for NAS, compared to fixed population sizes?*

Research Question 2B. *How does using clustering in objective space affect the performance of MO-GOMEA for NAS?*

Research Question 2C. *How does an adaptive clustering schedule affect the performance of MO-GOMEA for NAS, compared to fixed amounts of clusters?*

Research Question 2D. *How do extreme clusters affect the performance of MO-GOMEA for NAS?*

Research Question 2E. *How does automatic learning of dependency structures in the problem encoding affect the performance of MO-GOMEA for NAS?*

Chapter 5 is devoted to answering these research questions.

3

Experimental Setup

This chapter will describe the NAS search spaces used in this thesis and the methods that are shared among all experiments. Since NAS is an expensive process that requires GPUs to run for hours on end, the focus is first to find a suitable search space that has pre-computed performance indicators available for each architecture in the search space. In that way, many experiments can be performed without bearing the costs of training networks in each run. First, in Section 3.1, a brief overview of the current state of NAS benchmark datasets (with pre-computed values) is given. Also, a motivation is provided for the creation of a new benchmark, since the currently existing benchmark datasets do not suffice the research goals of this thesis. In Section 3.2, MacroNAS-C10 and MacroNAS-C100 are presented, which will serve as the main search spaces for the experiments in this thesis. Also a mapped encoding for the commonly known micro-architecture NAS benchmark dataset NAS-Bench-101 [84] is presented in Section 3.3, which is used in Chapter 4. At last, in Section 3.4, the common methods and experimental setups for the experiments performed in this thesis are highlighted.

3.1. Cached NAS benchmark datasets in literature

With a core motivation of improving reproducibility and comparability for NAS approaches, the first ever NAS benchmark dataset called NAS-Bench-101 [84] is created, which contains pre-computed values for each architecture in its search space. This micro-architecture search space allows the optimization of the composition of at most 7 nodes (excluding the input and output nodes, 5 nodes to be optimized) and their types in cells that are repeated throughout the final architecture. So although the cells themselves allow multiple parallel paths, the final architecture has a sequentially chained structure. The options for the cells are limited to convolutional layers (3×3 and 1×1) and max-pooling layers (3×3). This search space is designed for CIFAR-10, is mainly used for single-objective NAS and contains 423K unique architectures. The fact that the cells are encoded by an adjacency matrix results in infeasible networks and choosing how to handle these can considerably change how a search algorithm behaves. Another lab came up with an extension, named NAS-Bench-201 [12], in which the edges between the nodes are the layers/operations in a cell (which in turn is stacked 5 times in the macro-architecture skeleton). It allows for the optimization of edges between 4 nodes with 5 different types of operations. Next to convolutional layers (3×3 and 1×1) and average pooling layers (3×3), also zeroize (identity) layers and skip-connections are supported. All architectures in the search space are evaluated on CIFAR-10, CIFAR-100 and ImageNet, so transferability of well-performing cells of one task to the other can be verified with this benchmark dataset. Although more diagnostic information is provided, it supports only 15,625 different cells. NAS-Bench-1Shot1 [88] transforms NAS-Bench-101, without any additional training of networks, into three benchmark datasets that are specialized for one-shot NAS approaches.

Unfortunately these benchmarks do not suffice the research goals of this thesis. The focus of this thesis is on multi-objective NAS, for which at least NAS-Bench-101 is not suitable, which is empirically shown in Chapter 4. Additionally, all benchmarks with pre-computed values feature micro-architecture search spaces. While this kind of search spaces reduce the size of the search space and allow transferability of well-performing cells between classification tasks, it also limits the flexibility of the final architectures and still requires a practitioner to choose the macro-architecture skeleton. Instead, a

macro-architecture search space is desired, which is also a competitive NAS approach, although no NAS benchmark datasets with this structure exist yet. Additionally, it is desirable to have a large and diverse enough search space (preferably without infeasible networks as in NAS-Bench-101), such that thorough comparisons between search algorithms can be made in the early and in the late phase of the search. The search space should also not be hand-crafted too much such that the majority of networks is good, resulting in random search being a very competitive search strategy on it. For these reasons, two new benchmark datasets are proposed, MacroNAS-C10 and MacroNAS-C100, for well-known image classification tasks CIFAR-10 and CIFAR-100 respectively.

3.2. MacroNAS-C10 and MacroNAS-C100[1]

MacroNAS-C10 and MacroNAS-C100 feature the same macro-level search space and contain sequentially connected architectures of different sizes. The variables of the encoding of length $\ell = 14$ refer to the cells in the network, which can have either option from two differently configured MBConv blocks [66] and identity cells. The MBConv blocks are chosen to pursue networks that are also efficient, to allow a sensible multi-objective approach. The identity cell propagates the input without altering it, and can therefore be seen as a placeholder cell, enabling architectures of shorter length. Stacks of cells are divided by (unsearchable) reduction cells at fixed positions, as shown in Figure 3.1. Note that the encoding is redundant because of the identity cells: The search space of $3^{14} = 4,782,969$ encodings (of which none are infeasible) maps to $208,537$ unique architectures. The networks are trained using a one-shot approach; one supernet that contains all options for every cell is trained by training sampled paths for each batch of training samples. After training the full supernet, each path is evaluated using the same train/test/validation split. More details on the creation of these benchmarks can be found in Appendix A. The benchmarks are not designed to contain state-of-the-art accuracies, as shown in Table 3.1, but are designed to show differences in searching behaviors of NAS optimization algorithms. More statistics and visual analyses of MacroNAS-C10 and MacroNAS-C100 can be found in Appendix B.

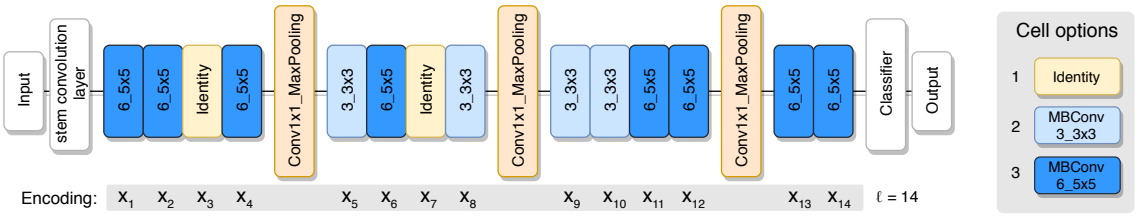


Figure 3.1: Example network in the MacroNAS search space.

	MacroNAS-C10	MacroNAS-C100
Size of search space	4,782,969	
Size of objective space (number of different networks)	208,537	
Redundancy	95.64%	
Best acc_{val}	0.9249	0.7049
Best acc_{test}	0.9190	0.7051
Correlation between acc_{val} and acc_{test}	0.979	0.986

Table 3.1: Statistics of the MacroNAS benchmark datasets

3.3. NAS-Bench-101

Besides the MacroNAS benchmarks, also NAS-Bench-101 is used (in Chapter 4) to investigate how algorithms fare on a micro-level NAS task. The encoding in the original paper combines a binary triangular matrix as adjacency matrix to indicate how the nodes in the cells are connected, with a string of length 5 that defines the type of each node. By unfolding the adjacency matrix and concatenating the options, the encoding is mapped to a string of length $\ell = 26$ with mixed symbols (the first 21 variables

[1] Although I was involved in the iterative design process of creating the new macro-level NAS benchmark, the final realization of the MacroNAS benchmarks is not done me, so I cannot take credit for it. This section however briefly explains its characteristics and how it is created

are binary and the last 5 are ternary), such that it can be used by GAs (and MO-GOMEA in particular). This mapping is visually displayed in Figure 3.2. Although there is not one straightforward ordering of the variables, it does not matter for MO-GOMEA that variables indicating connections between similar nodes may be spread out in the encoding: MO-GOMEA does not use point-wise (1-point, 2-point, etc.) cross-over, but flexible masks that can contain any group of variables in the encoding. Using this string-like encoding, it is possible to define infeasible networks, e.g., when there is no path from the starting to the ending node of the cell, therefore the objective values of such encodings are set to 0. The evaluation of infeasible networks is not counted towards the total number of evaluations of a search strategy.

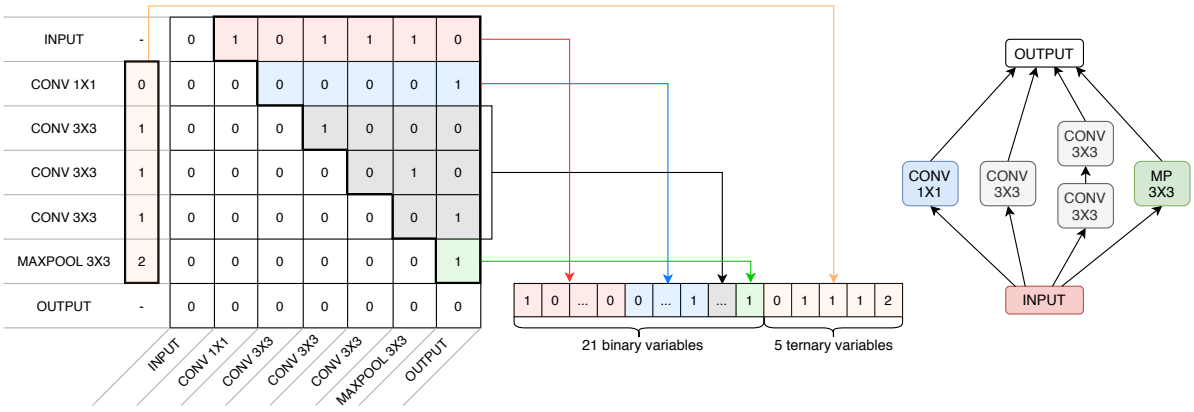


Figure 3.2: Translation of NAS-Bench-101 encoding to a linear encoding used by the algorithms in the experiments. On the right the resulting (Inception-like) cell is shown for the given encoding.

3.4. Methods

First, Section 3.4.1 explains which objectives are optimized and how their different ranges are accounted for. Section 3.4.2 explains how the redundant mapping from search to objective space is handled. Finally, Section 3.4.3 provides specifics on the number of runs and termination criteria.

3.4.1. Objectives

Two objectives are chosen to optimize for; maximization of the prediction accuracy and minimization of the model complexity. In all experiments the validation accuracy (acc_{val}) is chosen as metric for the prediction accuracy. For the model complexity, the number of parameters is chosen as a proxy when searching on NAS-Bench-101. When searching on MacroNAS, the number of Mega Multiply–ACcumulate operations (MMAC) of a network is chosen as complexity metric, which represents the number of operations a GPU performs when a forward pass is performed and is therefore related to the number of parameters. The reason for different metrics is based on the availability in the datasets, since MMAC information is not included in NAS-Bench-101, although it is better connected to the complexity and efficiency of a network and therefore chosen for MacroNAS.

To assess the performance of search algorithms, the hypervolume of the enclosed obtained front is calculated (using the origin as reference point) against the number of unique evaluations. Although many other metrics for multi-objective optimization exist, hypervolume is used for multiple reasons: 1) it is the most commonly used metric in MO-optimization [61], 2) it is Pareto-compliant, i.e., when new non-dominated solutions are found, the hypervolume always increases, 3) it combines spread and proximity in one scalar value and 4) it does not require to know the full set of optimal solutions. Reasons 2) and 4) are advantages over metrics like (inverse) generational distance, spread and front occupation, although the Pareto Front can be easily obtained for the benchmarks. To attribute both objectives an equal contribution in the hypervolume, the number of parameters and MMACs are normalized to the range of the validation accuracy, which is $[0, 1]$, using the minimum and maximum occurring values in the dataset. These are trivial to retrieve from these datasets, as they belong to the most simple and most complex architectures and can be calculated without evaluating networks, which makes it also possible to obtain them in a real-world scenario. Besides the hypervolume, also the achieved fronts over time are observed, to inspect behavioral characteristics of different optimization algorithms.

The all-identity network, i.e., the architecture with only identity layers and which has the least complexity and the worst accuracy, significantly impacts the hypervolume, but it is arguably the least interesting solution to actively search for. Therefore, it is included in the search from the beginning. That means, for population-based algorithms it is inserted into the population as the first individual and for other algorithms it is the first evaluated individual.

The choice to compare algorithms against the number of evaluations, instead of measuring against (wall-clock) time, is motivated by multiple reasons: 1) We make use of the pre-computed values in MacroNAS, that are obtained by a one-shot approach. It is not straightforward how to extract training times for separate nets or how to deal with this one-time training cost. 2) The aim is not to find a new SOTA NAS approach, but merely to analyze the search behavior of different NAS strategies from an algorithmic perspective. 3) If it were possible to obtain training times, the cost of approaches would also be highly dependent on the epoch/time budget for training networks. Analysis of the robustness of algorithms on the trade-off between reliable estimations (fully-trained networks) and fast evaluations (partially-trained networks) are outside the scope of this thesis. Nevertheless, Chapter 4 contains a brief analysis of the complexity distribution of all evaluated networks by different algorithms, because relatively simple networks take less time to train for a fixed epoch budget.

3.4.2. Caching of performance indicators

NAS is an overall expensive process, so in order to prevent unnecessary computations, the obtained performance indicators of a trained and evaluated architecture can be cached and re-used when it is evaluated by the optimization algorithm more than once. However, it could be argued that re-evaluating networks could provide robustness against fluctuating validation accuracies that are the result of stochastic training and network initialization. Nevertheless, it is still decided to evaluate each network only once, for two particular reasons: 1) Training a network is costly, therefore preferably not done more often than necessary. 2) Counting repeated evaluations would penalize algorithms that by nature train the networks repeatedly more often, e.g., population-based algorithms like EAs.

Subsequently, since both MacroNAS and NAS-Bench-101 exhibit redundancy in their encoding, there is a preliminary check before training and evaluation whether the resulting architecture has already been trained before as result of querying another encoding (which maps to the same architecture). Therefore, throughout all experiments, evaluating multiple encodings that all describe the same architecture are treated as one evaluation.

3.4.3. Specifics of runs

Unless indicated otherwise, every algorithm or configuration is run 30 times for an experiment. Convergence graphs show the medians and 25/75th percentile of the achieved hypervolume at different numbers of evaluations, in order to account for (extreme) outliers that would otherwise (severely) affect the mean value of the hypervolume. Although algorithms are terminated after $50,000$ evaluations (unless stated otherwise), both the early phase and the late phase will be inspected. Obtained fronts are stored and the hypervolume thereof is calculated at fixed intervals in the following scheme: $[1, 2, 3, ..., 10, 15, 20, ..., 100, 150, 200, ...1000, 1500, 2000, ..., 10000, 15000, 20000, ..., 50000]$. This enables showing meaningful results in graphs with both a linear scale and a logarithmic scale.

4

Baselines for NAS

Baselines and comparisons to them are essential in the proposition of new algorithms. They give meaning to new algorithms and their performance. If a problem can be solved by a very simple algorithm, such as random search, designing complex algorithms is pointless.

As already discussed in Chapter 2, there is a serious lack of comparisons with baselines in NAS research [37, 39]. Multiple reasons exist: 1) NAS algorithms vary wildly and different parts of NAS (search space, search strategy, performance estimation) make it difficult to compare approaches; 2) Research on NAS has an expensive nature, therefore running more experiments is sometimes not feasible; 3) The general focus is on getting new state-of-the-art results, and once this is presented, people generally accept the approach. In order to understand if complex algorithmic design endeavors are worthwhile, research needs to be done on adequate baselines to compare against. Since this thesis focuses mainly on the search phase of NAS, effort is spent on finding new (better) baseline algorithms to improve the quality of research on NAS. Specifically, this chapter aims to answer the first research question:

Research Question 1. *What simple, yet well-performing algorithms can provide a solid baseline in multi-objective neural architecture search?*

First of all, it is important to define what makes an algorithm a baseline. Although there is no official definition of a baseline algorithm, multiple interpretations exist. In this thesis, we move away from the idea of using SOTA algorithms as baseline, since SOTA algorithms on NAS are not always validated properly against very simple algorithms themselves. The baselines that we look for should have the following properties:

- A baseline is (algorithmically) **simple** and therefore easy to understand. It should be straightforward to know exactly how it performs the search on the problem at hand.
- A baseline is **easy to use** and implement, such that researches can effortlessly compare the performance of their newly proposed algorithm with the baseline algorithm's performance.
- A baseline is **doing something right**. Although obvious, the goal is still to get a good performance on the task at hand, so it needs to show converging behavior.

To give some examples, a baseline for a regression problem might consist of returning the mean value of the variable to regress. A baseline for a classification problem might consist of returning the most occurring class in the dataset. In NAS research, if baselines are used, it is commonly random search. Architectures are sampled uniformly from the search space and once the resource budget has been exhausted, the best-found network(s) is/are returned (and fully trained if not done already).

The community however repeatedly finds new, typically complex algorithms to apply on NAS. Between fully-fledged EAs and random search, lie a plethora of heuristic-based algorithms that are used only scarcely for NAS. Simpler, greedy and/or locally searching algorithms might perform equally well as complex algorithms, while still offering an easy-to-understand exploration of the search space.

As part of this thesis, an article has been written together with the supervisors (Arkadiy Dushatskiy, Marco Virgolin and Peter A. N. Bosman), which is publicly available as arXiv preprint and proposes

the use of a local-search (LS) algorithm as NAS baseline to the community [55]. The LS algorithm and the most important experimental results are provided in Section 4.1, but the interested reader is encouraged to read the full article in Appendix A. Additionally, another baseline called Uniform Size Random Search (USRS) is proposed and applied on NAS benchmark datasets in Section 4.2. This complementary baseline lies between RS and LS in terms of complexity and performance, and is not discussed in the preprint.

4.1. Local Search

In this section, a simple LS algorithm is proposed as search strategy for NAS. Its performance is demonstrated on NAS-Bench-101, MacroNAS-C10, MacroNAS-C100 and an enlarged version of MacroNAS. This enlarged search space allows more options for the normal cells and makes also the reduction cells optimizable, resulting in a space containing 673B encodings, mapped to 104B architectures. The reader is referred to the preprint article in Appendix A for more details on this search space. The increased size of the search space makes it infeasible to train and evaluate all networks beforehand, so no precomputed values are available. Therefore, the experiments are performed in real-time, which limits the number of runs to 6 and evaluations to 2,500 for all algorithms.

In Section 4.1.2, the performance of the proposed LS algorithm is evaluated alongside the default baseline RS, and two state-of-the-art multi-objective EAs; NSGA-II and MO-GOMEA. NSGA-II [11] is used, because even though it has been around for almost 20 years, it is still by far the most popular MO-EA. The other EA is MO-GOMEA[45], a powerful model-based EA, capable of automatically learning and exploiting the underlying structure of a problem. Additionally, it employs an automatic population-sizing scheme and clustering in objective space. In the same section also the behavior and the generalization of the different optimization algorithms are analyzed.

4.1.1. Local Search Pseudocode

Algorithm 1: Local Search (LS)
Input: Ω: the sample space of cell types; ℓ: the number of cells to search.
Output: P_f: the optimized front.

```
P_f ← InitializeElitistArchive()
while resource/time budget is not exhausted do
    α ← rand(0,1)
    s ← randomlyInitialize(ℓ, Ω)
     // cells are uniformly random sampled from Ω
    f_s ← evaluate(s)
    P_f ← UpdateElitistArchive(P_f, s, f_s)
    for i ∈ randomlyPermute([1, ..., ℓ]) do
        s' ← copy(s)
        for c ∈ (Ω \ {s_i}) do
            s'_i ← c
            f_s' ← evaluate(s')
            P_f ← UpdateElitistArchive(P_f, s, f_s)
            if scalarize(f_s', α) > scalarize(f_s, α) then
                s_i ← c
                f_s ← f_s'
return P_f
```

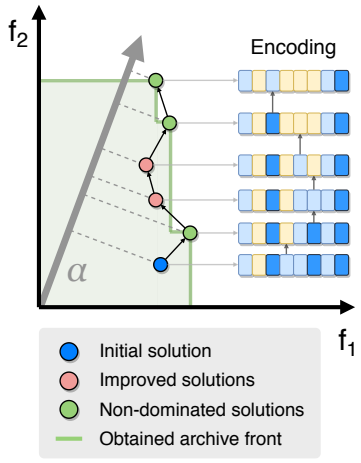


Figure 4.1: Graphical description of one iteration in the proposed LS algorithm. Evaluations that do not enlarge the front are not shown.

The LS algorithm uses the structure of a linear encoding as input, together with the options for each variable in the encoding. The pseudo-code is given in Algorithm 1 and a graphical description of an iteration of the algorithm is given in Figure 4.1. LS first randomly chooses a scalarization factor α, which is used to direct the search towards the part of the front with that specific trade-off in objective values. An encoding is initialized at random and the resulting architecture is evaluated. Then, every variable in the encoding is traversed in a random order. LS greedily searches the local neighborhood of the architecture in the encoding space: For each of the variables, every option is tried, the network is evaluated, and the best option (according to the scalarized fitness) is kept. Every variable is greedily

optimized only once, because after the variables have all been traversed, a new network is randomly initialized and a new scalarization factor is sampled. LS is designed for multi-objective NAS tasks, but could also be used in the single-objective case by removing the scalarization (or simply setting $\alpha = 1$) and the elitist archive operations, thereby only keeping track of the best-performing network.

4.1.2. Experiments and Results[1]

Results of initial experiments on NAS-Bench-101, MacroNAS-C10 and MacroNAS-C100 are shown in Figure 4.2 by the convergence of the obtained archives to the true Pareto Front by all four algorithms in terms of hypervolume against network evaluations. For NAS-Bench-101, differences between the performance of the algorithms are very small. On the MacroNAS benchmarks, RS is clearly inferior to the other algorithms, therefore leaving room for improvement over the default baseline.

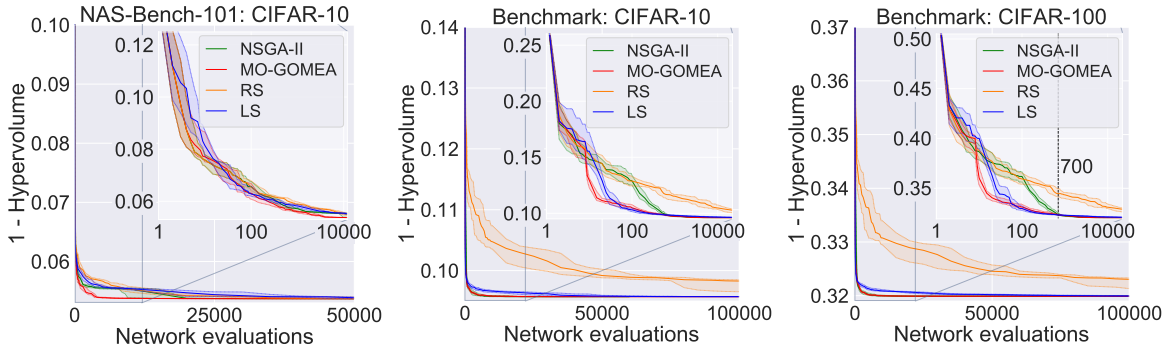


Figure 4.2: Convergence graphs on NAS-Bench-101 (left), MacroNAS-C10 (middle) and MacroNAS-C100 (right). Medians (solid lines) and 25/75th percentiles (bands) of 30 runs are shown. Note that only the horizontal axes are the same, and are in logarithmic scale in the zoomed-in views (insets).

In the subsequent experiment the performance of the algorithms on MacroNAS-C100 in terms of hypervolume against the number of network evaluations (see Figure 4.4, MacroNAS-C10 provided similar results), together with their achieved archives (see Figure 4.3), are studied in more detail. MO-GOMEA performs overall best, but it is also shown that LS finds a diverse front quickly, by finding architectures in sparsely populated areas in the objective space. The advantage of the LS algorithm is that intermediate steps of greedily improving the encoding according to its scalarization also provides architectures for the elitist archive, therefore already finding multiple solutions in tuning only one encoding. NSGA-II and RS struggle to find efficient networks in the beginning, but NSGA-II is later able to find similar fronts as MO-GOMEA and LS.

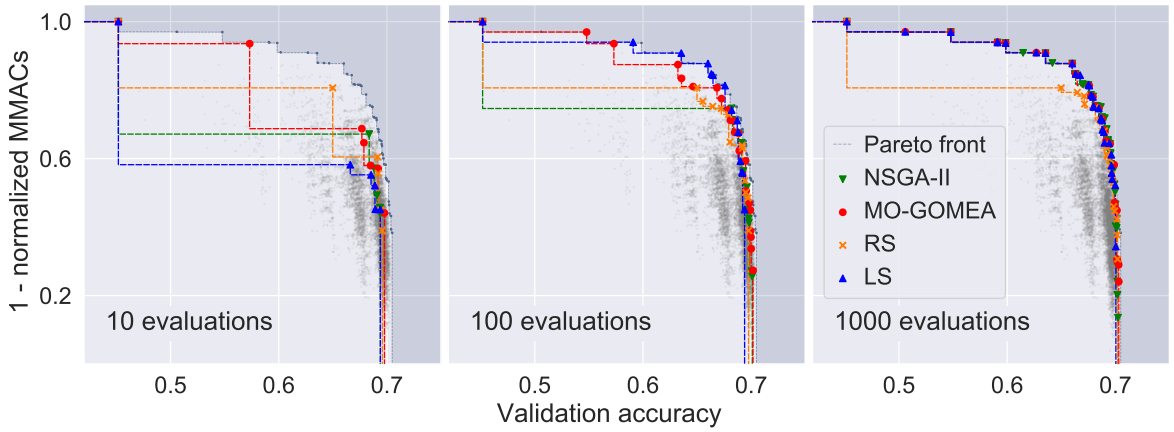


Figure 4.3: Evolving archives for one example run on the MacroNAS-C100 benchmark dataset. Gray points represent all possible architectures.

[1]This section closely follows the results presented in the preprint in Appendix A.

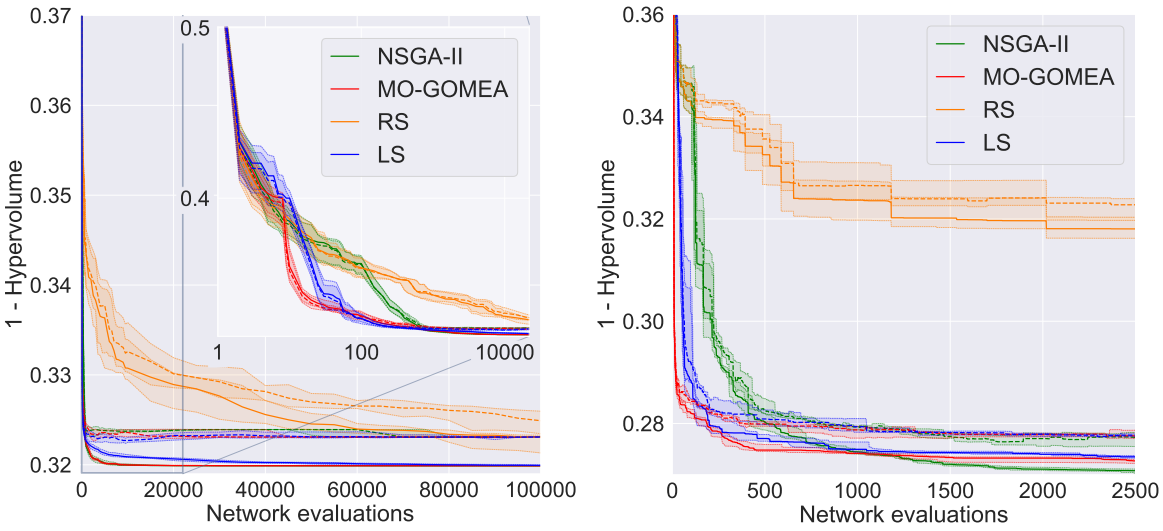


Figure 4.4: Convergence graphs regarding acc_{val} (solid lines) and acc_{test} (dashed lines) on MacroNAS-C100 (left) and on CIFAR-100 on the enlarged search space (right). Medians (lines) and 25/75th percentiles (bands) of respectively 30 and 6 runs are shown for all algorithms. Note the different ranges for the vertical axes and the logarithmic scale for the horizontal axis in the zoomed-in view.

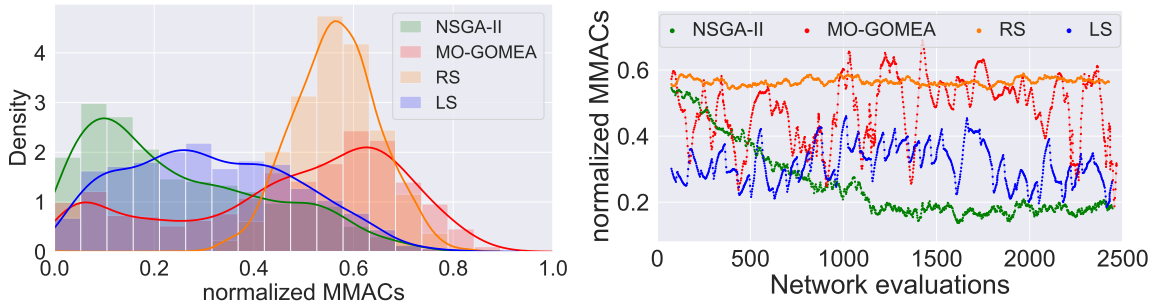


Figure 4.5: Distribution of MMACs of evaluated nets over 2500 evaluations on the large-scale experiment (left) and throughout one example run (right), smoothed for readability (moving avg. filter of size 75).

The last experiment in the preprint involves real-time NAS runs on the enlarged version of the MacroNAS search space. Again, as shown in Figure 4.4 (right), LS proves itself to be considerably outperforming RS, and closely performing as good as NSGA-II and MO-GOMEA. The searching behavior is analyzed in terms of complexity of the networks evaluated by the algorithms, as shown in Figure 4.5. The right graph, which displays MMACs against the number of evaluations, visibly shows different iterations of LS (even after applying the moving average filter), since networks evaluated one after in one LS iteration are quite similar. RS naively samples many complex networks, a result of relatively many complex networks in the search space. NSGA-II steers its search towards simpler networks throughout the NAS process, because of its crowding distance operator, therefore discovering networks at the top of the front later than MO-GOMEA and LS. MO-GOMEA evaluates networks all over the place, due to its clustering mechanism that searches at multiple parts of the front interchangeably. Although the comparison between algorithms is not based on time, this analysis favors NSGA-II and LS, as they sample more simple networks that take less time to train when given an equal number of epochs for training.

A final look on generalization (shown in Figure 4.4 for both MacroNAS-C100 and the large-scale experiment) gives us another important insight. As the architectures are only evaluated based on validation accuracy, there will always be a gap towards the accuracy on the test set. The substantial difference in performance between RS and the other algorithms stays, but even though the more powerfully searching EAs achieve a better hypervolume w.r.t. validation accuracy than LS, the advantage more or less disappears when the hypervolume w.r.t. test accuracy is compared. The less powerful search ability of LS causes it to be less prone to overfitting on these NAS problems and therefore questions

the value of developing even more complex search strategies for NAS.

The conclusion is therefore that the performed experiments provide evidence that LS can be considered a strong alternative to harder-to-implement algorithms like NSGA-II and MO-GOMEA, without considerably sacrificing in performance. Additionally, the currently most-used baseline RS is outperformed by LS, as RS struggles to overcome the distributional properties of the search spaces, therefore not reliably covering the full front.

4.2. Improved RS: Uniform Size Random Search

Random Search can perform poorly on a multi-objective NAS problem if the density of different parts at the front are skewed, because it samples from the search space uniformly at random. For example, the choice for the encoding of the architectures in the MacroNAS-C10 and MacroNAS-C100 benchmark datasets, as well as in the large-scale version of it, has as consequence that very few small networks exist, as shown in Figure 4.6 (left). Since the RS strategy samples a different cell for each variable, the chance of sampling a relatively small (with ≤ 5 non-identity layers) network is only 1.8%. Figure 4.6 (right) shows that finding these networks is necessary to cover the upper part of the Pareto Front.

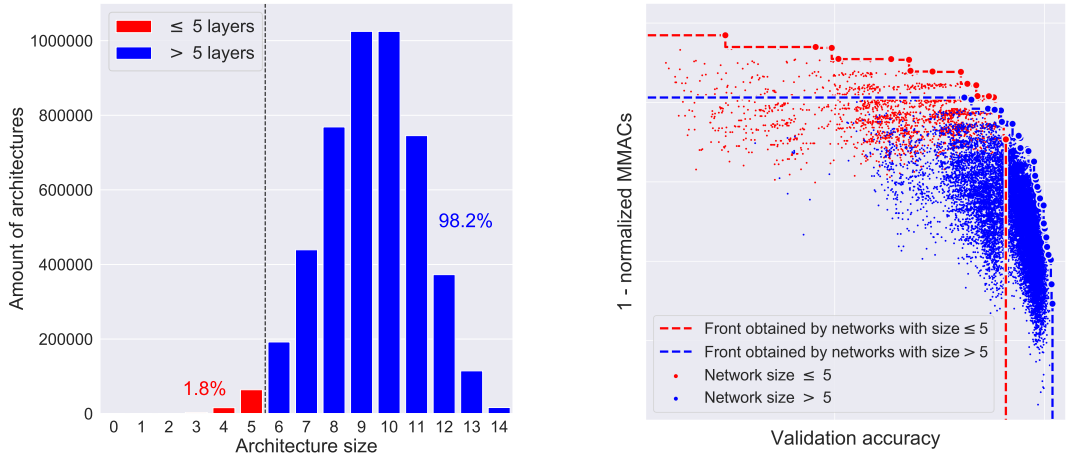


Figure 4.6: Frequency (left) and scattered distribution (right) of two differently-sized groups of networks in the MacroNAS-C100 benchmark dataset.

In order to minimally upgrade RS to deal with this skewed distribution of architectures, Uniform Size Random Search (USRS) is proposed. USRS is compared to LS, the naive baseline RS and the two EAs NSGA-II and MO-GOMEA on the search spaces NAS-Bench-101, MacroNAS-C10 and MacroNAS-C100.

4.2.1. Uniform Size Random Search Pseudocode

USRS makes sampling networks of any size equally likely, by first uniformly sampling the size of the architecture and then randomly sampling any network of that size. The pseudo-code is shown below in Algorithm 2.

Algorithm 2: Uniform Size Random Search (USRS)
Input: ℓ_{min}, ℓ_{max}: the minimum and maximum size of architectures in the search space.
Output: the optimized front P_f.

while *resource/time budget not exhausted* **do**
  $\ell \leftarrow$ `SampleSize(`ℓ_{min}, ℓ_{max}`);`
  $s \leftarrow$ `SampleNetwork(`ℓ`);`
  $f_s \leftarrow$ `Evaluate(`s`);`
  $P_f \leftarrow$ `UpdateElitistArchive(`P_f, s, f_s`);`
end
return P_f

4.2.2. Experiments and Results

MacroNAS-C10 and MacroNAS-C100

As shown in Figure 4.7, on both MacroNAS-C10 and MacroNAS-C100, USRS immediately starts expanding its front of found solutions, quicker than any other algorithm it is compared to. It is however quickly caught up by MO-GOMEA (around 10-20 evaluations) and LS (around 50-70 evaluations). Nevertheless, as seen in all graphs of Figure 4.7, USRS outperforms plain RS markedly. This can also be seen in Figure 4.8, as USRS quickly finds a nicely spreaded front in only 100 evaluations, whereas RS struggles to find networks on the top part of the Pareto Front even at 1000 evaluations. The ability of LS to improve upon found solutions during the subsequent steps in its algorithm obviously results in finding architectures on the Pareto Front more quickly than USRS and RS (see e.g. the achieved fronts at 100 evaluations). USRS lacks this ability, but still manages to find a good front regardless.

These results show that RS with a slight modification can be made more powerful than the naive version of RS. However, the fact that USRS performs well on these search spaces is also a result of the properties of the search space and the chosen performance metric (i.e., the hypervolume). First sampling a size for the network to be evaluated makes it so that USRS searches more diversely throughout the objective space, whereas the naive RS samples parts of the objective space proportional to the density, which is not always favorable.

While the performance of USRS is not investigated for the enlarged search space, it is expected that the results translate. USRS tackles the skewed distribution of the solutions in the objective space, which is even more present in the enlarged search space: for each normal cell, there is one identity option against four effective options (compared to one identity option against two effective options in MacroNAS). It is considered future work to confirm this assumption.

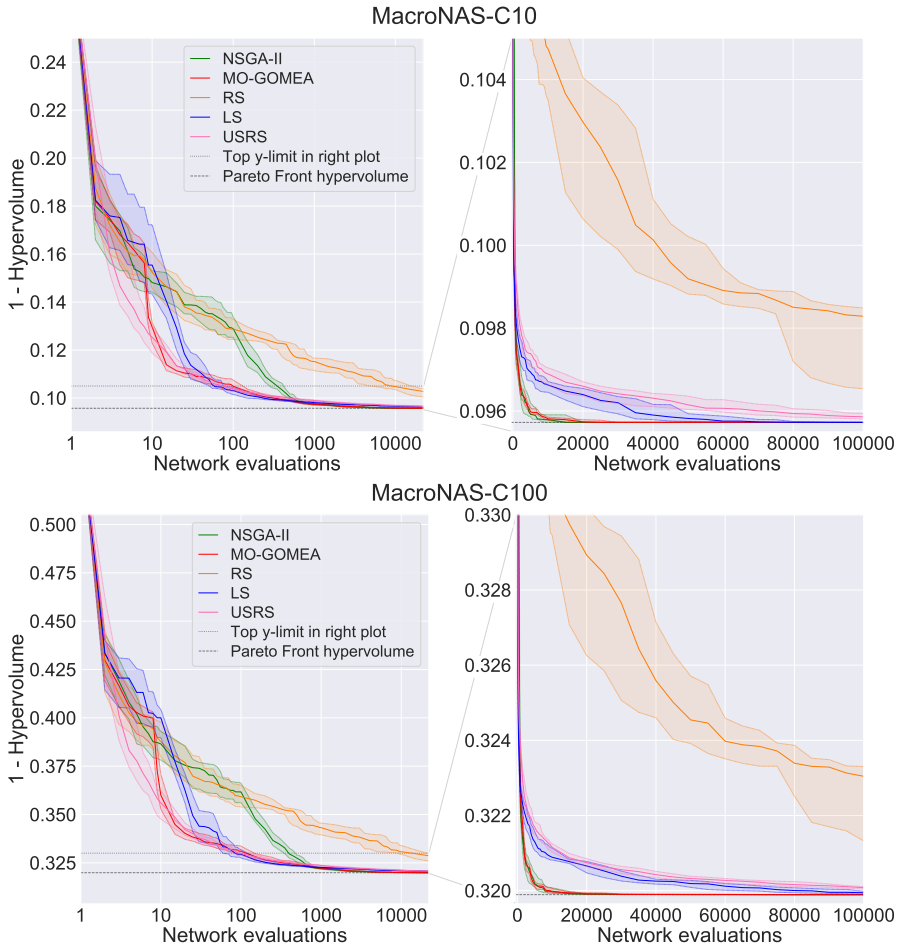


Figure 4.7: Medians and 25/75th percentiles of convergence for multiple algorithms on MacroNAS-C10 (top) and MacroNAS-C100 (bottom).

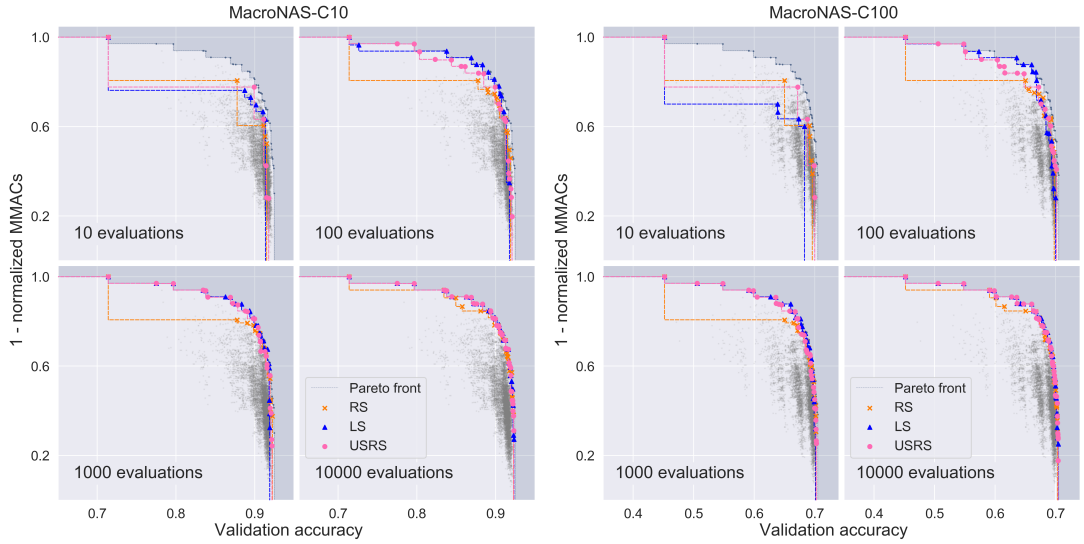


Figure 4.8: Achieved Fronts w.r.t. validation accuracy after different numbers of network evaluations by three baseline algorithms RS, LS and USRS on MacroNAS-C10 (left) and MacroNAS-C100 (right)

NAS-Bench-101

First of all, as also discussed in the preprint article, NAS-Bench-101 can be considered to be not ideal for multi-objective NAS. Figure 4.9 (right) shows a front that leaves few room for a practitioner to make a deliberate trade-off between the two objectives. Compromising for top accuracy for just a little bit will result in almost equally performing networks, but with far less parameters. While these architectures can be desired solutions in practice, the networks are found quickly (e.g., RS and USRS find solutions in this area after already 10 evaluations), making it hard to thoroughly analyze the search phase of multi-objective NAS on this benchmark dataset.

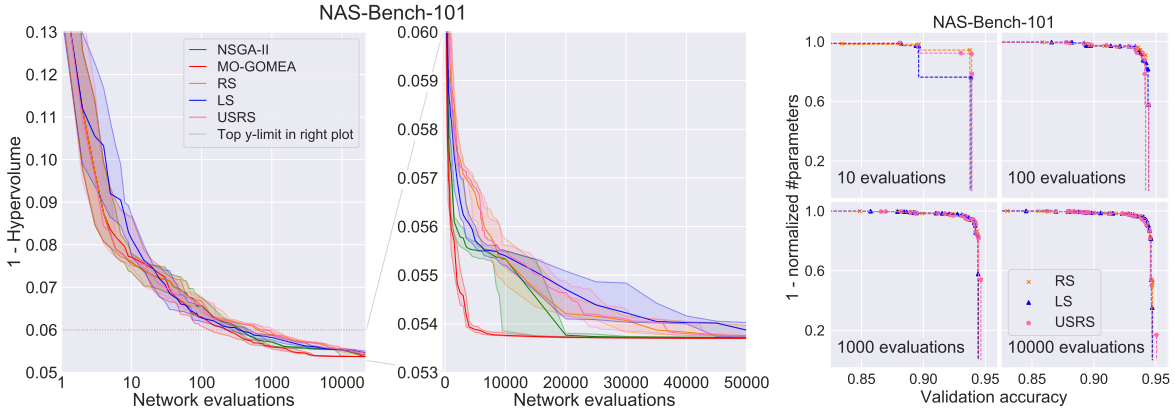


Figure 4.9: Medians and 25/75th percentile of convergence for multiple algorithms on NAS-Bench-101 (left and middle). Achieved Fronts w.r.t. validation accuracy at different numbers of network evaluations by three baseline algorithms RS, LS and USRS on NAS-Bench-101 (right).

Additionally, as also discussed in the article, differences between algorithms are small on NAS-Bench-101. In fact, USRS does not even perform better than RS in terms of convergence of the hypervolume (as shown in Figure 4.9, left and middle). Moreover, fronts discovered by all three baseline algorithms look very alike for 100, 1,000 and 10,000 network evaluations (Figure 4.9, right). Solutions of the final front are already found after 100 evaluations and in the remainder of the search this front is filled with more solutions, but does not move a lot anymore. Next to the above statements about the benchmark dataset itself, there are two reasons for the performance shown by USRS:

1. Although the macro-architecture is fixed for NAS-Bench-101, the micro-level cell structure, which is optimized in the search phase, is graph-based. The current implementation of USRS on NAS-

Bench-101 effectively removes some number of nodes from the final network (by setting variables that control connections to and from them to 0), based on the sampled size. However, that does not fix how the remaining nodes are connected, which can be connected sequentially, in parallel or in any other way possible. Therefore, the sampled size does not directly control the depth of the network, which is often an important factor for its prediction ability. Additionally, the macro-architecture is fixed, therefore the sampled size in USRS does not modify this.

2. There is a more balanced objective space than with the MacroNAS benchmarks. Therefore, it is not necessary to correct RS for sampling from every size, as this is done already naturally.

This experiment shows that the size-sampling upgrade on RS is not an improvement on every search space.

4.3. Conclusion

In this chapter two new baselines for NAS are proposed. LS performs very well on the macro-architecture search spaces of MacroNAS, even on the enlarged version where RS has increased difficulty to find a decent front. It has also become apparent how different algorithms explore the search space in distinct ways: MO-GOMEA and LS perform a balanced search and even though MO-GOMEA excels at finding the networks with the best validation accuracy, LS finds nets that generalize better to the test set. NSGA-II gradually changes the area in which it searches, and finds a good coverage of the front. RS finds well-predicting networks, but is not well equipped to find simple networks.

Also USRS outperforms RS on these search spaces, as the skewed density in these search spaces are efficiently combated. Positive results do not transfer to NAS-Bench-101, but the underlying idea is still interesting: by tweaking and smartly aligning an remarkably simple algorithm like RS with the search space at hand, the performance can be greatly improved.

In developing new (complex) NAS algorithms and approaches, comparing against baselines is a necessity to make sure the added complexity and searching power results in improved performance. Comparing against naive RS could be a first step. Then, to ensure that results are still worth noting, comparisons could be made against the performance of slightly smarter baselines, like USRS or LS, which can already offer adequate performances without adding too much complexity. Also, if a practitioner's goal is to just get a range of decent architectures to choose from, algorithms like USRS and LS can be a first go-to without having to spend much effort on implementing complex search algorithms for NAS. Admittedly, if the search space is encoded differently, the algorithms need to be tweaked to align with it. But like RS, USRS and LS are both sufficiently simple to be understood, so this should be a straightforward modification.

5

Ablation of MO-GOMEA on Macro-Level NAS

Many articles in the field propose new search algorithms to apply on NAS problems, often with the purpose of finding networks with new state-of-the-art performances. Nevertheless, ablation studies showing which components of their newly proposed algorithm are necessary to get the demonstrated performance, are often missing [37]. Especially for EAs, many combinations and configurations of operators exist, with each mechanism delivering a specific property to the overall algorithm. For example, mutation operators are crucial for EAs deployed on continuous problems, whereas deceptive problems (like the artificial Trap5 problem, see Section 5.1.2) require specific cross-over masks that mix entire blocks of variables. The operators and their configuration that are required depend on the problem at hand, and is not always straightforward to deduce beforehand. This chapter will research what effect different components of an EA have on the performance when applied to NAS problems. Specifically, this chapter aims to answer Research Question 2:

Research Question 2. *What mechanisms make a state-of-the-art multi-objective evolutionary algorithm perform well on a multi-objective macro-level neural architecture search problem?*

In order to answer this question, this chapter will more closely examine the performance of MO-GOMEA, an MO-EA that performs well on NAS-Bench-101, MacroNAS-C10 and -C100 in the experiments in Chapter 4. MO-GOMEA is a model-based EA, which relies on optimal mixing of learned inter-dependencies that may exist between problem variables, an automatic population-sizing and clustering schedule and clustering in objective space. The experiments in this chapter are aimed to perform an ablation study on MO-GOMEA, in order to understand which components are vital to achieve a good performance on NAS.

First, MO-GOMEA and its components are explained in more detail in Section 5.1. After this, more detailed sub-research questions are formulated that work towards answering Research Question 2. Section 5.2 explains the methods and problem setup that is used for the experiments, whereas Section 5.3 presents results of the experiments. Finally, a practical perspective and concluding remarks on the results are given in Section 5.4.

5.1. Background

Generally speaking, two classes of EAs exist. One class are problem-specific EAs, which are tailored to the problem at hand and perform very well on it. This class however also requires problem-specific information, which is not always trivial to obtain. Another class of EAs that does not require problem-specific knowledge are model-based EAs, that adapt to the problem at hand, by learning and exploiting problem-specific properties during the optimization. This makes model-based EAs not necessarily the best-performing algorithms, but they are versatile for a wide range of problems and applicable without much tuning. One of such EAs is the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) [76]. In this section, GOMEA and its multi-objective extension MO-GOMEA [45] will be

explained.

5.1.1. GOMEA

GOMEA [76] is a state-of-the-art model-based EA that performs cross-over using a structure that is learned during the optimization. The template structure, which is often a *linkage tree* in the case of GOMEA, is configured based on the population of individuals, such that potential linkage that exists between variables in the encoding is discovered and exploited during the search. This linkage, which can also be called inter-dependency, may be non-trivial to know before optimization, therefore having the algorithm detect it by itself is useful. Besides learning the hidden structures in a problem, GOMEA makes use of Optimal Mixing (Section 5.1.1), Forced Improvements (Section 5.1.1) and the Interleaved Multi-start Scheme (Section 5.1.2).

Optimal mixing

GOMEA is different than ordinary EAs in the sense that when evolving a new generation, multiple cross-over operations are performed on an individual (the *receiver*) in the optimal mixing phase, which is performed for every individual in the population. For each cross-over operation, a *donor* is randomly picked from the population to supply genetic material to the receiver. The change is only accepted if the receiver has an increased fitness with the updated genotype, otherwise the change is reverted (see Figure 5.1). To specify which variables are exchanged during cross-over, some sort of structure is required, which can be defined by a *Family Of Subsets (FOS)*. Ideally, the structure of the FOS should allow dependencies between variables to be modeled.

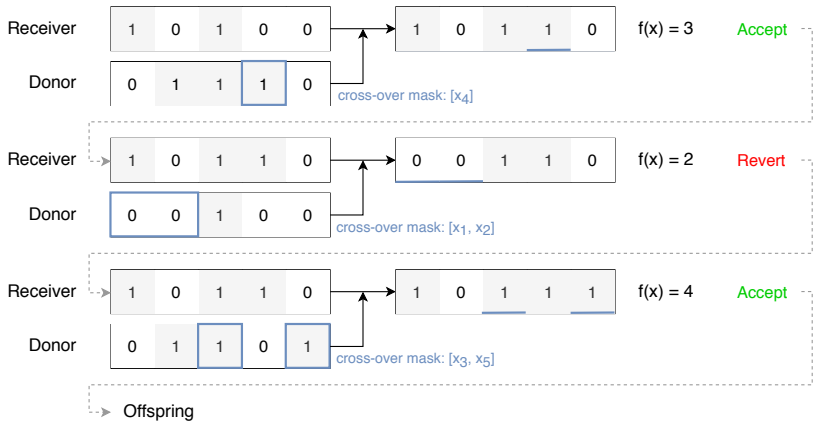


Figure 5.1: Three cross-over operations in the Optimal Mixing phase of GOMEA, each with a randomly picked donor that supplies genetic material in the mask specified by the blue borders.

Formally, a FOS is defined as a set F of subsets F_i of a main set S. In this case, S is the set containing all ℓ variables in the encoding, i.e., $S = \{x_1, x_2, ..., x_\ell\}$. An example is the univariate FOS, which is the set that contains all variables as singletons, i.e., $F_i = \{x_i\}$. The univariate FOS cannot not model dependencies between (groups of) variables, however.

The Marginal Product (MP) FOS allows some dependency between variables to be modeled. In the MP FOS, every variable is present in exactly one of the subsets, such that for every subsets $F_i, F_j (i \neq j)$ in F, $F_i \cap F_j = \emptyset$ and $\cup_{i=1..\ell} F_i = S$. The MP FOS can be configured by a greedy algorithm which iteratively merges subsets that minimize the Minimium Description Length (MDL) of the individuals in the population the most [20]. The MP FOS constrains a variable to be part of one group of dependent variables only, though.

The Linkage Tree (LT) FOS, which is also used in GOMEA, allows modeling of dependencies between variables on different levels. This hierarchical FOS contains every variable as separate subset, just like in the univariate FOS. Additionally, merges of subsets exist in F, such that the final FOS is a hierarchical tree of subsequently merged subsets, with the root containing all variables, i.e., it is equal to S. That means, a set that is a subset of another set (as result of a merge) cannot be part of another merge. An example of an LT is shown in Figure 5.2. Formally defined, for every set F_i with $|F_i| > 1$ in the LT, there are exactly two sets F_j and F_k that are mutually exclusive, i.e., $F_j \cap F_k = \emptyset$, but their union results

in F_i, i.e., $F_j \cup F_k = F_i$. This LT FOS allows variables to occur in different subsets of different sizes, therefore a richer dependency model can be captured and learned.

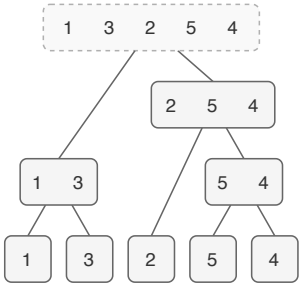


Figure 5.2: Example of a Linkage Tree FOS. The root is not considered in the final LT FOS.

Similarly to learning the MP FOS structure, the LT FOS is also learned through a greedy algorithm. First, the mutual information is calculated between every pair of single variables with $I(X_i, X_j) = H(X_i) + H(X_j) - H(X_i, X_j)$, in which $H(X_i)$ is the marginal entropy of variable X_i and $H(X_i, X_j)$ is the joint entropy of variables X_i and X_j. The entropy values are calculated from the frequency of the occurring values for the variables (single and combined) in the population. This procedure yields a Mutual Information Matrix (MIM) that contains the mutual information for every pair of variables. A high mutual information for a pair of variables means that certain values for the variables appear together more often in the population, possibly as result of their improvement upon an individual's fitness, which in turn increased the likeliness of these individuals getting selected over other individuals. The combination is therefore valuable and might suggest that there exists dependency between these variables.

To circumvent the computational complexity of counting frequencies for large substructures in the LT, a faster clustering technique called the unweighted pair group method with arithmetic mean (UPGMA) is used, which is more efficient, because it only uses the mutual information between pairs of single variables. The distance of two subsets can be efficiently calculated using the pre-computed values in the MIM, by:

$$\frac{1}{|X_{F_i}||X_{F_j}|}\sum_{X \in X_{F_i}}\sum_{Y \in X_{F_j}} H(X,Y) - I(X,Y)$$

Building the LT is done from the leaves up, so initially all elements from the univariate FOS are added to the LT. Then, the two subsets that result in the lowest UPGMA distance are merged and added to the LT. The merged subsets are replaced with the new subset in the pool of subsets available for merging. This procedure is repeated until there are two subsets left; the root that contains all variables (i.e., equal to S) is not added to the FOS, because applying cross-over using this mask results in receiving all donor material, therefore the receiver becoming identical to the already evaluated donor.

The resulting Linkage Tree is traversed during the cross-over phase, each time taking a subset F_i as cross-over mask. Since the LT contains exactly $2\ell - 2$ subsets, the cross-over phase for one individual requires at most $2\ell - 2$ evaluations (if the genotype is unchanged after receiving donor genetic material, no evaluation is performed).

In the context of NAS, it could be argued that since a neural network has multiple connected layers that propagate information, a layer depends on the processed information by the layer before it. Whether such dependencies exist and whether a model-based EA is able to discover and exploit it is interesting to find out for the development of new NAS approaches. So, whether the usage of an automatic linkage learning mechanism is beneficial to the performance and efficiency of search strategies on NAS, is experimentally investigated in order to answer **Research Question 2E** (see Section 5.1.3).

Forced Improvements

The most recent version of GOMEA makes use of so-called *Forced Improvements (FI)* [7]. When a solution is not improved by any cross-over in the optimal mixing phase, another round of optimal mixing (using the same LT) is performed with the best solution found so far (the elitist solution) as donor. The optimal mixing is terminated when a cross-over results in the first improvement of the fitness. If no improvement of the fitness is observed throughout FI, the individual is replaced with the elitist solution. Initially, tournament selection was performed in GOMEA after optimal mixing,

but this results in the diversity decreasing too quickly. FI replaces the tournament selection without increasing the selection pressure continuously, but only does this when no improvement can be found in the ordinary optimal mixing procedure.

5.1.2. MO-GOMEA

In order to approach multi-objective optimization problems with GOMEA, the multi-objective variant MO-GOMEA [45] was invented. Similarly to GOMEA, MO-GOMEA uses the LT FOS for optimal mixing, but also makes use of clustering in objective space and an elitist archive to keep track of solutions.

Elitist Archive

To keep track of the best solutions found so far, an elitist archive is used. This archive follows the definition of a non-dominated set $\mathcal{S}$: $\nexists \boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}$, s.t. $\boldsymbol{x} \succ \boldsymbol{y}$, so the archive can only contain non-dominated solutions. After every evaluation in MO-GOMEA that led to an improvement, the elitist archive is updated with the new solution, by adding it to the archive if it is not dominated by any solution in the archive and potentially deleting the dominated solutions from the archive. At the end of the optimization, the elitist archive is the returned set of solutions. In this thesis, only an elitist archive of unlimited size is considered.

Clustering

Another mechanism used by MO-GOMEA is clustering. The individuals of the population are clustered in the objective space by an adapted *k-means-clustering* algorithm, which allows for overlapping clusters to improve the probability of obtaining an evenly-spread set of solutions. Since solutions can have different characteristics at different regions of the front, different dependency models may be required to effectively recombine individuals. Therefore, an LT is learned for every cluster separately. The optimal mixing is also restricted to individuals within a cluster, so the donor is always randomly picked from the cluster of the receiver. A cross-over is accepted when 1) the new solution dominates the old one, 2) the fitness for all objectives stays the same, or 3) the new solution is not dominated by any solution in the elitist archive. Forced improvements are also applied in MO-GOMEA, and the donor is randomly chosen from the elitist archive. If FI does not yield any accepted solution, the individual is replaced by a random solution from the elitist archive. Furthermore, when optimizing for m objectives, m clusters that emerge on the extreme ends of the objective space are designated to be extreme clusters, one for each of the m objectives. Cross-over in extreme clusters is the same as in GOMEA, i.e., individuals are only compared on the basis of their fitness in one objective. This can help improving the proximity on the extreme ends of the front more efficiently. The solution with the best fitness in the corresponding objective is used as donor in the FI-phase in the extreme clusters. Figure 5.3 illustrates the clustering-based approach of MO-GOMEA.

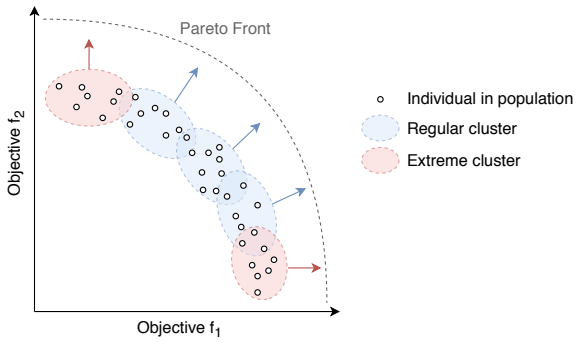


Figure 5.3: Clustering mechanism in MO-GOMEA when 5 clusters are used. Note the distinction between regular and extreme clusters. The arrows display the direction of the optimization for each cluster.

The importance of clustering in objective space and the usage of extreme clusters for NAS is investigated by experiments to answer **Research Questions 2B** and **2D** (see Section 5.1.3).

Parameter-less

Two crucial parameters still need be set before we can get a satisfactory performance, but those are very problem dependent: the population size and the number of clusters. The population size needs to be large enough to be able to find solutions of the desired quality, but should not be too large to prevent overly diversifying the search, which results in the evaluations budget being exhausted before solutions of the desired quality are found. The number of clusters is also important, since there need to be enough individuals in a cluster to learn a sensible dependency model, but too many individuals in a cluster prevents it from thoroughly exploring specific parts of the front. Predicting the required population size and cluster number can be very hard. On artificial problems a preliminary analysis can be done to find the optimal population size and number of clusters, but this is often infeasible in a real-world scenario, when it is very expensive to run the EA multiple times. This especially holds for NAS, for which the optimization routine should ideally be executed only once for a desirable result.

To overcome this issue and to make MO-GOMEA easier to use, a mechanism is employed that removes the requirement of setting the population size and the number of clusters: the *Interleaved Multi-start Scheme (IMS)*, first proposed in [21]. Populations with different population sizes n are scheduled to evolve generations in MO-GOMEA in an interleaved fashion. Small populations are evolved for a base number of generations (default generation base = 2) before the next population (usually of doubled size) is allowed to evolve a generation. Figure 5.4 visualizes such a schedule, in which every round some populations get to evolve a generation, smaller populations more often than larger populations. Additionally, to prevent inefficient populations from performing evaluations, a population P_i is terminated if all points on its front are either covered or dominated by the points of the front of a larger population P_j.

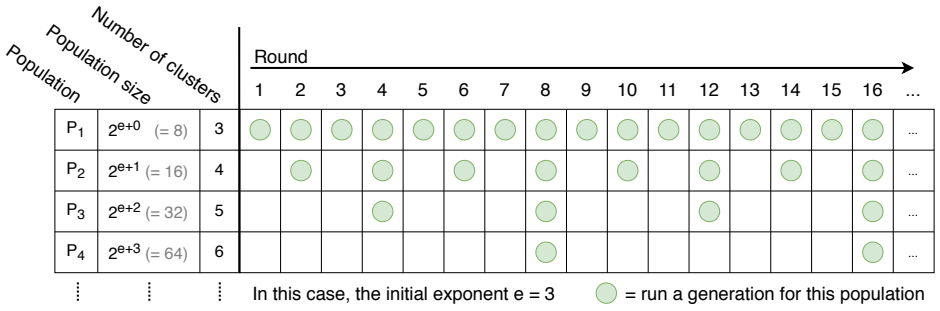


Figure 5.4: Interleaved Multi-start Scheme (*IMS*) with population base 2, starting exponent $e = 3$ and generation base 2.

The number of clusters is also a hyperparameter that should be set differently depending on the problem, but it is also clearly connected to the population size. Therefore, it can be set together with the population size in the *IMS*. This is done by a scheme that sets the number of clusters to $(m + 1)$ (for m objectives) in the initial population, and increases it with 1 for every subsequent population size.

To combine the results of different populations, the elitist archive is shared among the populations. Therefore also obtained knowledge is shared between populations, since the donor in FI is one of the solutions in the elitist archive, which can be beneficial to "revive" populations that are stuck.

The effect of the population sizing scheme compared to fixed population sizes, as well as the effect of the configuration of the number of clusters on the searching performance on NAS, are both examined by experiments that aim to answer **Research Questions 2A** and **2C** (see Section 5.1.3).

Performance

GOMEA and MO-GOMEA perform and scale well on artificial problems that contain linkage, like Trap5 [76] and Trap5-Inverse Trap5 [45] respectively. The binary representation of solutions for Trap5 contains deceptive groups of 5 linked bits: the bits of the optimal group all have value 1, while all suboptimal groups increase in fitness when more bits in the group have the value 0. For the multi-objective optimization problem Trap5-Inverse Trap5, the fitness is inverted for the second objective (i.e., the all-0 group is optimal). The fact that incremental, non-informed steps in improving groups often leads to a local optimum (i.e., the all-0 solution) that is encoding-wise very different from the global optimum (i.e., the all-1 solution), makes these optimization problems deceptive. For example,

ordinary GAs that fall for these deceptive traps require larger population sizes and more evaluations to solve these problems to optimality, compared to GOMEA and MO-GOMEA. These model-based EAs tackle the deceptive Trap problems more efficiently, by learning the underlying dependency model and subsequently mix entire groups of variables as building blocks.

However, it is also observed that MO-GOMEA performs poorly on Leading-Ones-Trailing-Zeros (LOTZ) without mutation operators [45]. In LOTZ, consecutive leading 1's in the encoding contribute to one objective, while consecutive trailing 0's contribute to the other objective. As a result, a 1 as last bit and 0 as first bit will not contribute to the fitness for each objective if all remaining bits are not 1 and 0, respectively. Solutions with a trailing 1 or a leading 0 are therefore likely to be discarded from the population in the beginning of the search, but these bits are crucial for finding the extreme solutions (i.e., the all-0 and all-1 solutions) on the Pareto Front. The LT in MO-GOMEA is unable to capture this kind of conditional dependency. Mutating bits from Pareto-Optimal solutions (i.e., no non-contributing bits in-between leading 1's and trailing 0's) is therefore a more effective strategy. For example, LOTZ is better tackled by NSGA-II and MOEA/D, which both contain mutation operators [45].

These results point out that problems with a different problem structure require different mechanisms to efficiently solve them and that the learning of a dependency model is ineffective when the problem structure exhibits a different type of linkage, or none at all, but this is often hard to know beforehand for real-world problems. Nevertheless, since the most recent version of MO-GOMEA also features two types of mutation that can be switched on and off, it is a versatile MO-EA that has been proven effective for real-world problems as well [46, 50].

5.1.3. Formulation of Sub-Research Questions

In total, 5 sub-research questions are formulated, focusing on the population sizing scheme, objective space clustering and the learning of dependency structures. These questions will be addressed in order to answer **Research Question 2**.

Research Question 2A. *How does an automatic population-sizing schedule affect the performance of MO-GOMEA for NAS, compared to fixed population sizes?*

Research Question 2B. *How does using clustering in objective space affect the performance of MO-GOMEA for NAS?*

Research Question 2C. *How does an adaptive clustering schedule affect the performance of MO-GOMEA for NAS, compared to fixed amounts of clusters?*

Research Question 2D. *How do extreme clusters affect the performance of MO-GOMEA for NAS?*

Research Question 2E. *How does automatic learning of dependency structures in the problem encoding affect the performance of MO-GOMEA for NAS?*

5.2. Methods

The macro-architecture search spaces MacroNAS-C10 and MacroNAS-C100 are used for the experiments in this chapter. The experimental setup is similar to the setup for the experiments in Chapter 4 and is largely explained in Chapter 3. Every experiment will have forced improvements enabled and mutation disabled in the configurations of MO-GOMEA. Other aspects are varied in each experiment. When populations of fixed size are almost (prematurely) converged, MO-GOMEA tends to perform a tremendous number of evaluations of architectures it has already evaluated before, while evaluating only very few newly discovered architectures. So especially for Experiment 1 in Section 5.3.1, an extra termination criterion of 10 minutes is implemented, because the limit of 50,000 unique network evaluations is often never reached. Note that this is only the time the search algorithm needs, since for evaluation of networks the pre-computed values of the benchmark dataset are used (in comparison, MO-GOMEA with IMS takes ~5 seconds to reach 50,000 unique network evaluations).

For testing whether results are significantly different, a nonparametric Mann Whitney U test is performed, from which p-values[1] are reported in tables and shown in figures. The hypervolumes of all runs are compared between different configurations of MO-GOMEA, at multiple points in time (different

[1]It is left to the reader to draw conclusions of significance based on the reported p-values.

number of evaluations), by testing for the hypotheses:

H_0: The distribution of hypervolume convergence of configuration A is equal to the distribution of configuration B at X evaluations

H_1: The distribution of hypervolume convergence of configuration A is different/better/worse than the distribution of configuration B at X evaluations

During the experiments, the attention is drawn away from the discussion on whether the generalization of found networks to the test set influences the final performance of MO-GOMEA, although this is addressed in Section 5.4. Instead, the emphasis is put on the actual search phase, i.e., searching for architectures with the best trade-off on *validation* accuracy and model complexity.

5.3. Experiments

In total 5 experiments are performed, each with the purpose of answering a specific sub-research question, as shown in Table 5.1. Each of the experiments will be covered in a separate section, continued by a section that will explain what the results mean from a practical perspective.

Experiment	RQ	Designed to show ...
1	2A	The effect of using the *IMS* compared to fixed population sizes
2	2B	The effect of clustering in general
3	2C	The effect of the number of clusters used
4	2D	The effect of extreme clusters
5	2E	The effect of automatic model learning

Table 5.1: Experiments and their goals

5.3.1. Experiment 1: Effect of *IMS* compared to fixed population sizes

The population size can be a determining factor for the performance of an EA [21], as also explained in Section 5.1.2. It is therefore interesting to investigate what effect the *IMS* has when solving a NAS problem. In this experiment, MO-GOMEA with *IMS* enabled is compared to MO-GOMEA with various fixed population sizes. The number of clusters, however, is linked to the *IMS* and increases together with the population size. From here on out, this clustering schedule linked to the *IMS* is referred to as *adaptive clustering*. In order to make the comparison as fair as possible, for each fixed population size the number of clusters is set similar to the value in the adaptive clustering scheme, as shown in Table 5.2.

Population size	**8**	16	**32**	64	**128**	256	**512**	1024
Number of clusters	**3**	4	**5**	6	**7**	8	**9**	10

Table 5.2: Number of clusters for each population in the *IMS*. Bold values indicate the fixed population sizes and number of clusters used in the experiment.

Results Figure 5.5 shows a clear and logical trade-off: smaller fixed-size populations dominate the optimization early, while large populations excel at a later stage in the optimization, also achieving a better hypervolume ultimately. MO-GOMEA with *IMS* enabled seems to perform similarly to small fixed populations in the beginning, which makes sense, because it is only optimizing small populations then. In the long term, MO-GOMEA with *IMS* is slightly lagging behind compared to fixed-size populations in terms of evaluations needed to reach a certain value for the hypervolume, which is caused by overhead. Nevertheless, it is able to reach the same hypervolume as fixed (large) populations, without requiring larger populations: After $17,500$ evaluations on average, MO-GOMEA performs the first optimal mixing phase of evolving the population of size 512, and the population of size 1024 is initialized only after $28,000$ evaluations on average. At this point, negligible differences between MO-GOMEA with *IMS* and population size 512 are observed, as also shown in Figure 5.6 (at $10,000$ evaluations already) and in the MacroNAS-C100 table in Figure 5.7. Nevertheless, if a larger, more complicated search space requires higher population sizes, MO-GOMEA with *IMS* would always be able to improve upon its results eventually.

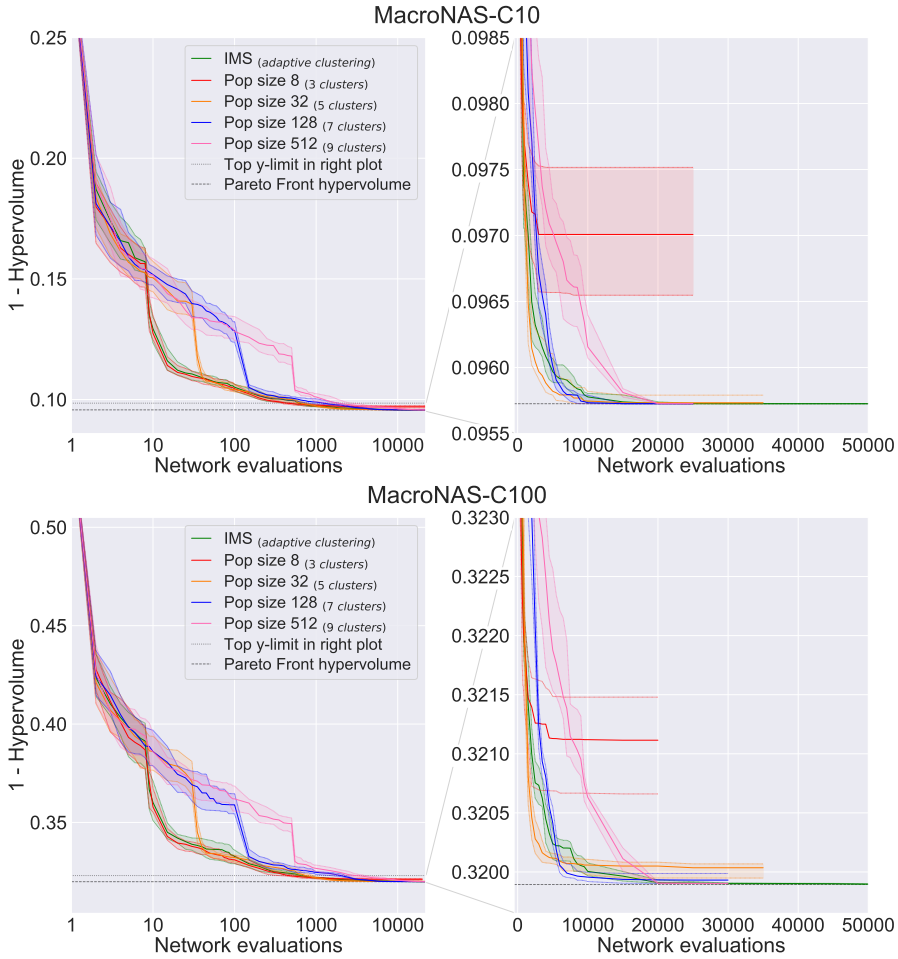


Figure 5.5: Experiment 1: Medians and 25/75th percentiles for 30 runs of MO-GOMEA with varying population sizes on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). The number of clusters is chosen according to Table 5.2. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

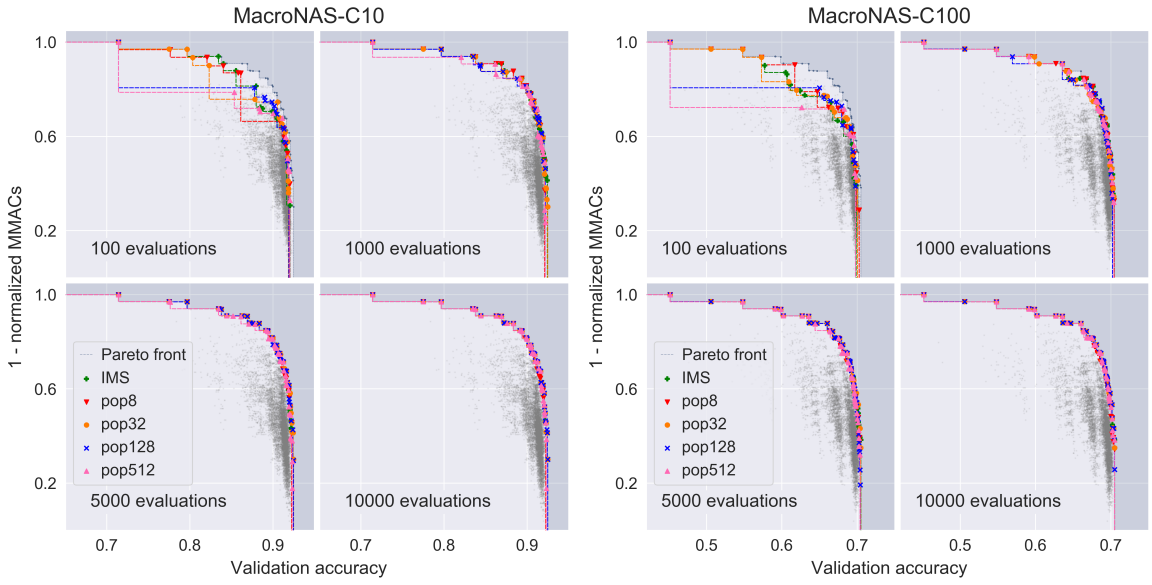


Figure 5.6: Experiment 1: Achieved fronts for one example run of MO-GOMEA for varying population sizes on MacroNAS-C10 (left) and MacroNAS-C100 (right).

Figure 5.6 shows that MO-GOMEA with fixed large populations (128 and 512) at both MacroNAS-C10 and MacroNAS-C100 struggle to find simple networks in the early search phase (100 evaluations). This is certainly caused by the random initialization phase, in which sampling small networks is unlikely, as shown by RS in Chapter 4. At $1,000$ evaluations, the large populations still have not discovered some solutions on the Pareto Front, contrary to smaller populations. At $5,000$ and $10,000$ evaluations, differences are not really visible anymore in Figure 5.6, but from Figure 5.5, we know that MO-GOMEA with *IMS* and with fixed large populations will be able to find the last remaining undiscovered solutions, thereby covering the full Pareto Front.

The p-values from significance tests (Mann Whitney U) shown in the heatmap tables in Figure 5.7 indicate that MO-GOMEA with *IMS* performs overall best on MacroNAS-C100 throughout the course of the search, as almost all cells are either red (*IMS* is better than the fixed population size) or white (there is no significant difference). On MacroNAS-C10, MO-GOMEA with *IMS* performs generally equally as MO-GOMEA with population size 32, except during the initialization phase. Furthermore, MO-GOMEA with fixed population sizes 128 and 512 seem to do better than *IMS* just before their search is concluded (because of convergence, or exceeding the time limit of 10 minutes). But bear in mind, that if the objective function still allows for improvement, MO-GOMEA with *IMS* will always continue to improve, since new, larger populations will be used. However, as it is hard to decide beforehand which population size is most suited for the resources available without doing extensive preliminary analysis, using the MO-GOMEA with *IMS* would be the overall best choice.

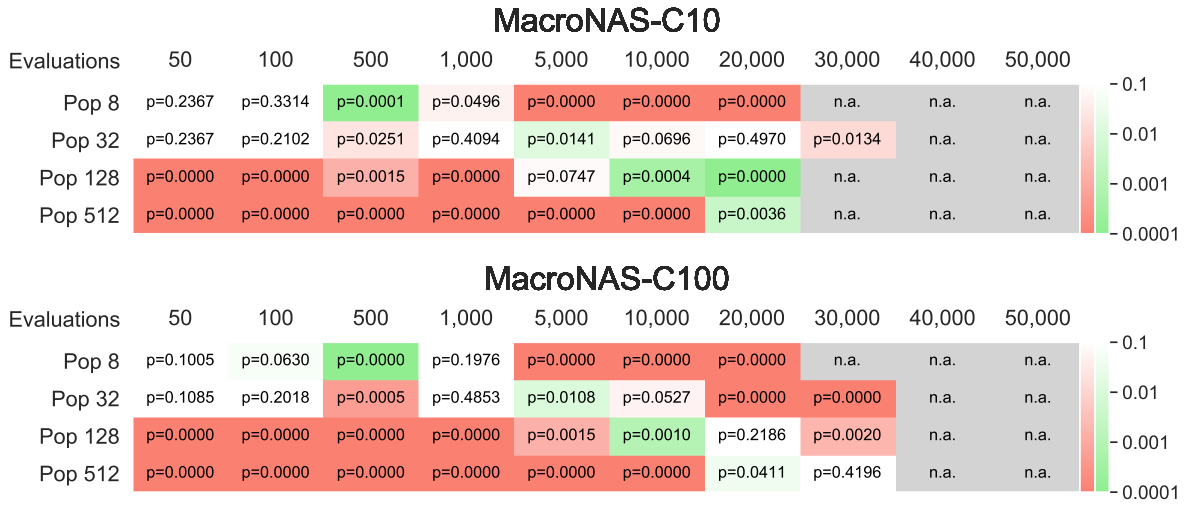


Figure 5.7: P-values of Mann Whitney U tests are shown for comparisons between optimizations of MO-GOMEA using fixed populations and MO-GOMEA using *IMS*, on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). Red indicates that the results of the fixed population are worse than *IMS* results, green indicates that the results are better than *IMS* results.

Conclusion The automatic population-sizing scheme is therefore attractive for when "anytime" results are desired. That means, if the optimization budget is unknown beforehand, MO-GOMEA with *IMS* will always be able to deliver results close to when the optimal (fixed) population size had been used. This is an important insight for a routine with a particularly expensive nature like NAS, in which it is undesirable to run the optimization more than once.

In further experiments, the *IMS* is always enabled and is not considered a changing factor anymore.

5.3.2. Experiment 2: Effect of (adaptive) clustering in objective space

The first clustering-related experiment is designed to show whether the same performance can be achieved without any form of clustering. MO-GOMEA with adaptive clustering and without any clustering (i.e., the number of clusters is set to 1) is applied on MacroNAS-C10 and MacroNAS-C100, using the *IMS*, Forced Improvements and learned LT cross-over.

Results Figure 5.8 shows the results. There is a slightly better performance by MO-GOMEA using clustering, which is (visually) apparent throughout almost the entirety of the search for both MacroNAS-

C100 and MacroNAS-C10, except for a the window between 300 and 600 evaluations on MacroNAS-C10. For both search spaces, some of the runs of MO-GOMEA without clustering are eventually able to get an equally good hypervolume as with clustering, although this happens only after 25,000 evaluations. However, the bands of MO-GOMEA without clustering in Figure 5.8 show that not every run is guaranteed to fully cover the Pareto Front.

The confirmation that the results of MO-GOMEA without clustering are inferior to those with clustering until at least 50,000 evaluations is shown by the p-values of Mann Whitney U tests between the two configurations. This is displayed by the green segments underneath the graphs in Figure 5.5 when the clustering algorithm has a (significantly) better hypervolume at that number of network evaluations, whereas gray segments mean no significant differences. For MacroNAS-C10 there is a slightly (significantly) better hypervolume between 150 and 300 evaluations, and consistently from 650 evaluations onward. For MacroNAS-C100, the significantly better hypervolume is already achieved at 15 evaluations and apart from a few small windows (70 to 90 and 350 to 600 evaluations), it stays better with a low p-value (high significance).

The corresponding achieved fronts of both versions of MO-GOMEA, as shown in Figure 5.9, display what areas of the objective space are explored throughout the search. When clustering is used, the front is pushed at all positions, such that there is always a diverse front. Despite the fact that this has a positive impact on the hypervolume, networks at the Pareto Front with a roughly equal trade-off between accuracy and complexity are discovered later than when not using clustering. Conversely, when no clustering is used, MO-GOMEA fails to discover some efficient networks (i.e., networks on the top of the front) early, but those are still discovered later on.

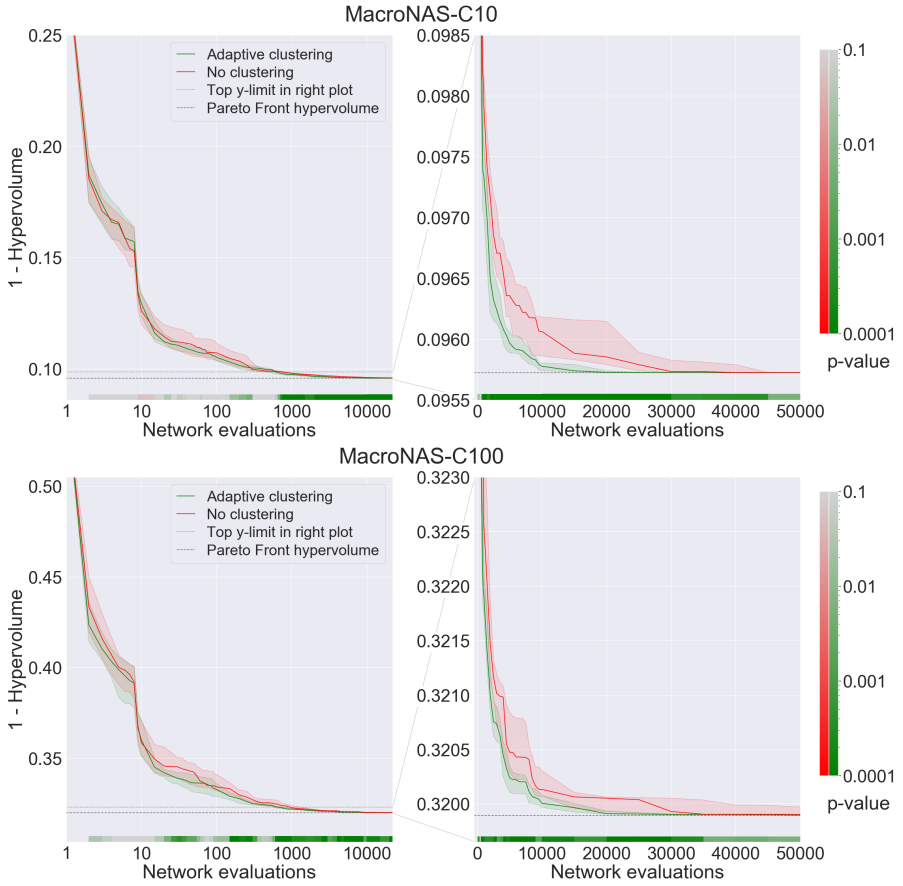


Figure 5.8: Experiment 2: Medians and 25/75th percentile of 30 runs of MO-GOMEA with (adaptive) clustering compared to no clustering on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). Both configurations use *IMS*, forced improvements and learned LT cross-over. The bars underneath the graph show the p-values of significance tests between the two configurations by means of the color scale on the right. Grey means no significant difference, green/red means MO-GOMEA with clustering is better/worse than MO-GOMEA without clustering, with certain p-values. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

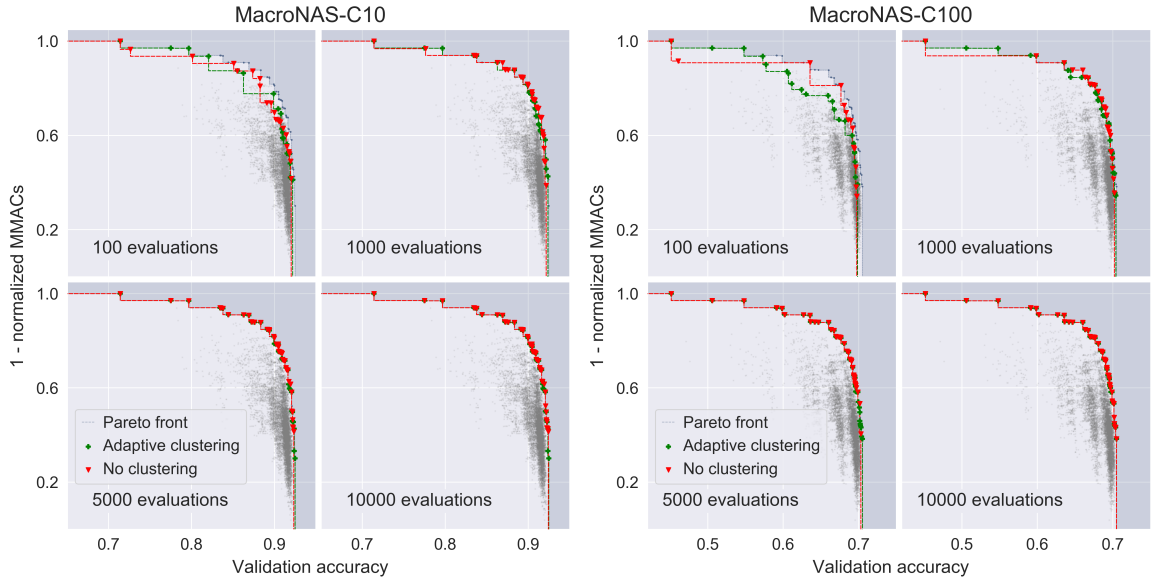


Figure 5.9: Experiment 2: Achieved fronts for one example run of MO-GOMEA for adaptive and no clustering on MacroNAS-C10 (left) and MacroNAS-C100 (right).

Conclusion This trade-off therefore makes it hard so say whether clustering should be used on these search spaces. In terms of hypervolume, it is advised to use it, since there is always either a significant improvement, or no significant difference when using clustering compared to no clustering. But as the achieved fronts display different objective space exploration behaviors, there is no conclusive answer, at least not on these NAS search spaces.

However, this experiment compared only the adaptive clustering variant to not using clustering. The number of clusters used, as well as the usage of extreme clusters might also influence the performance of MO-GOMEA, something that is investigated in upcoming Sections 5.3.3 and 5.3.4 respectively.

5.3.3. Experiment 3: Effect of the number of clusters

The third experiment shows the result of using a fixed number of clusters, compared to the (adaptive) clustering scheme tied to the *IMS*. Therefore, the number of clusters is varied and fixed, while other factors, like *IMS* (enabled), forced improvement (enabled) and the cross-over operator (learned LT) are fixed. There is however one particular detail of the clustering mechanism that needs attention and that is the usage of extreme clusters. Using only two clusters will result in two extreme clusters in this NAS problem (as there are two objectives), therefore there is no cluster that optimizes the front for the objectives in a combined way. Since this interferes with results focused on the number of clusters, the extreme clusters are disabled in this experiment (this experiment but with extreme clusters enabled is shown in Appendix C.1).

Results Figure 5.10 shows the convergence of the different configurations of MO-GOMEA on both MacroNAS-C10 and MacroNAS-C100. Apparently, any number of clusters works well, as all lines and their bands overlap each other. One observation can be made though, which is that MO-GOMEA with only 2 clusters appears to have the worst hypervolume for its 25th percentile for a large part of the search after 5,000 evaluations (for both MacroNAS-C10 and -C100), giving the impression that preferably more than 2 clusters should be used for a more stable search. This presumption is weakly confirmed by significance tests; the only few p-values < 0.01 resulting from Mann Whitney U tests on all pairs of configurations suggest that the configuration with 2 clusters is inferior[2]. Nevertheless, these significance results also tell that there are generally not any differences between the configurations. Pareto fronts are shown in Appendix C.1, but any visually interesting searching behavior can basically

[2]MacroNAS-C100: Adaptive clustering compared to 2 clusters, at 40,000 (p-value=0.0077) and 50,000 (p-value=0.0086) evaluations. MacroNAS-C10: Adaptive clustering compared to 2 clusters, at 40,000 (p-value=0.0066) and 50,000 (p-value=0.0055) evaluations, and 8 clusters compared to 2 clusters, at 5,000 evaluations (p-value=0.0045). In all cases, 2 clusters is inferior. Significance tested at [50, 100, 500, 1000, 5000, 10000, 20000, 30000, 40000, 50000] evaluations

be attributed to noise, as the fronts show the achieved fronts for one run of each configuration only, and taking different runs results in seemingly different orders of discovery of solutions.

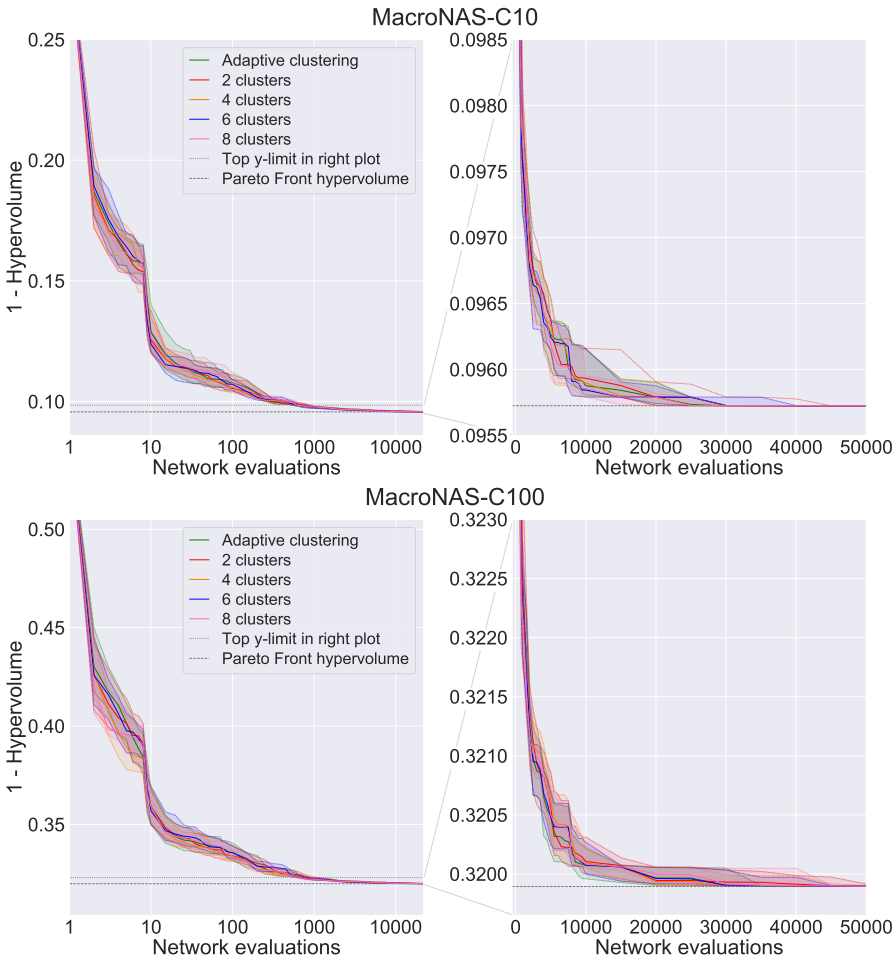


Figure 5.10: Experiment 3: Medians and 25/75th percentile of 30 runs of MO-GOMEA with different numbers of clusters on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). Both configurations use *IMS*, forced improvements and learned LT cross-over. Extreme clusters are disabled. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

Conclusion It does not really matter how many clusters are used, as long as clustering is enabled, although it seems that using more than 2 clusters is advised to get a more predictable hypervolume as result. Therefore, adaptive clustering might as well be used, because it does not perform worse than fixed numbers of clusters and gives a more stable cluster sub-population size for different population sizes.

This experiment does not take into account extreme clusters. The next experiment will therefore examine whether extreme clusters improve the performance of MO-GOMEA on the MacroNAS benchmark datasets.

5.3.4. Experiment 4: Effect of extreme clusters

This experiment zooms in on the effect of using extreme clusters within the clustering mechanism of MO-GOMEA. These clusters optimize only single-objectively in the corresponding objective, and so there is one per objective. Two versions of MO-GOMEA are compared, one with the extreme clusters enabled and one with the extreme clusters disabled. Both configurations of MO-GOMEA make use of the *IMS*, adaptive clustering, forced improvements and a learned LT cross-over.

Results Figure 5.11 shows the convergence of both configurations on MacroNAS-C10 and MacroNAS-C100. At first glance, it seems to pay off to make use of extreme clusters, as the green line (extreme

clusters enabled) is under the red line (extreme clusters disabled) at all times, except for when both configurations have discovered all solutions on the Pareto Front. This is confirmed by the p-values of statistical testing (Mann Whitney U), which are shown in the graphs by the colored bar underneath. On MacroNAS-C10, most of the bar before convergence has a green color, indicating that hypervolumes obtained by MO-GOMEA runs using extreme clusters are better with low p-values (high significance) at that number of evaluations. Regarding MacroNAS-C100, there are several windows in which there is no significant difference between the two configurations, although if there is any, it is MO-GOMEA with extreme clusters enabled that produces a better hypervolume. Just briefly after initialization of the first population (of size 8), MO-GOMEA without extreme clusters does better on MacroNAS-C100. There is no fixed cluster that is always optimized first, so apparently optimizing a non-extreme cluster first (which clearly always happens when extreme clusters are not used) results in improving upon the hypervolume most. Nevertheless, this pattern is not visible on MacroNAS-C10 and the small advantage very quickly vanishes (within a few evaluations), therefore this observation can be disregarded.

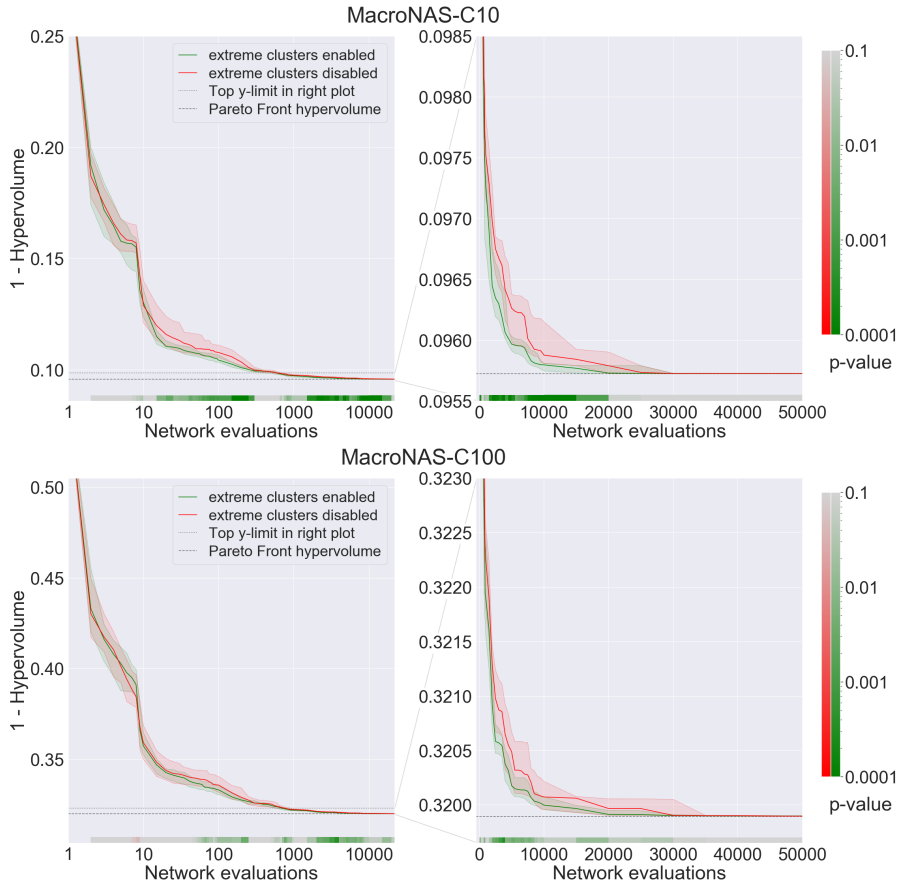


Figure 5.11: Experiment 4: Medians and 25/75th percentile of 30 runs of MO-GOMEA with extreme clusters enabled and disabled, on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). Both configurations use *IMS* (together with adaptive clustering), forced improvements and learned LT cross-over. The bars underneath the graph show the p-values of significance tests (Mann Whitney U) between the two configurations by means of the color scale on the right. Grey means no significant difference, green/red means MO-GOMEA with extreme clusters is better/worse than MO-GOMEA without extreme clusters, with certain p-values. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

When looking at the achieved fronts by both configurations in Figure 5.12, expected but interesting behaviour can be observed. Logically, MO-GOMEA with extreme clusters enabled finds the solutions on both ends of the Pareto Front earlier than when disabled. However, at 50 and 100 evaluations this seems to be at the cost of finding good balanced solutions, which are discovered earlier when no extreme fronts are used. Later on, at 1000 and 5000 evaluations, the difference between both configurations is not visible anymore, as both configurations are able to discover the full Pareto Front. This is also visible in 5.11, where both configurations achieve the optimal hypervolume, although the version with extreme clusters does so faster.

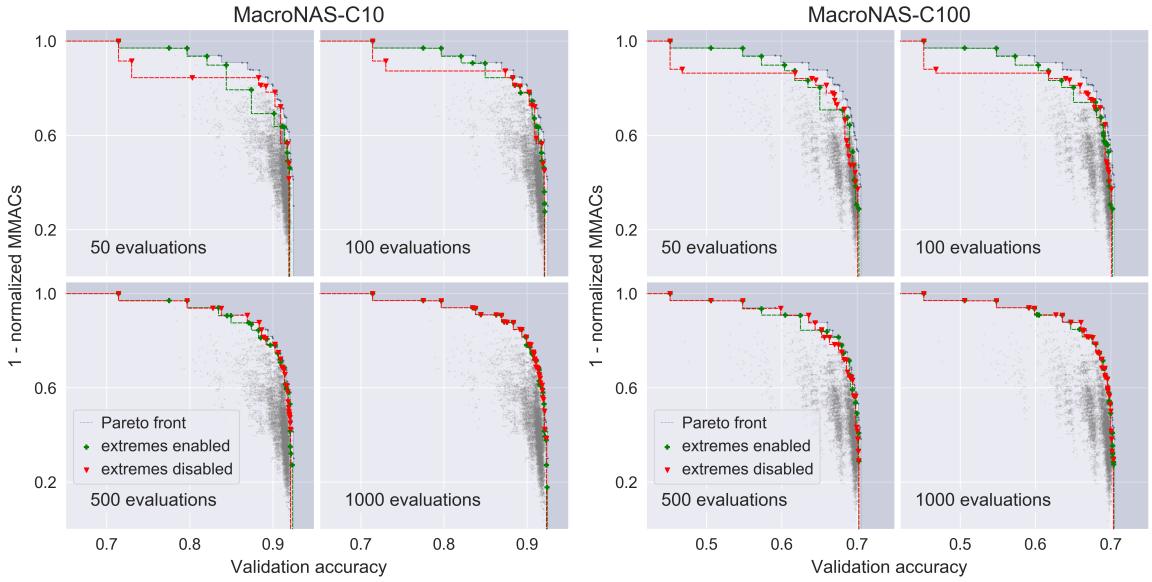


Figure 5.12: Experiment 4: Achieved fronts for one example run of MO-GOMEA for extreme clusters enabled and disabled on MacroNAS-C10 (left) and MacroNAS-C100 (right).

Conclusion Enabling or disabling the usage of extreme clusters in MO-GOMEA leads to different searching behavior, so deciding whether or not to use extreme clusters can be based on the importance of the extreme ends of the front. Disabling extreme clusters leads to a faster coverage of balanced solutions in the front, which might be preferred by a practitioner, since multi-objective NAS is aimed at finding solutions with a good trade-off. Nevertheless, using extreme clusters leads to a more diverse front in the first part of the search, therefore giving a practitioner a more diverse range of architectures to choose from. Therefore no universal advice can be given, as both enabling and disabling extreme fronts lead to different front discovery behaviors by MO-GOMEA.

5.3.5. Experiment 5: Effect of linkage learning

The last experiment is concerned with the model-based characteristic of MO-GOMEA. Dependencies that exist between variables should be captured and exploited by MO-GOMEA, but if the encodings in the search space exhibit very little dependency, the automatic model learning in MO-GOMEA is pointless. In this experiment, the default MO-GOMEA with the learned Linkage Tree (LLT) is compared to MO-GOMEA with a randomly initialized Linkage Tree (RLT) and with a univariate structure (in randomized order) for optimal mixing. Forced Improvements, IMS, adaptive clustering and extreme clusters are enabled for all configurations.

Results Figure 5.13 shows the convergence of the different configurations, with the univariate FOS lagging behind on both LLT and RLT between 10 and 100 evaluations. The reason for this is likely that it mixes too slow, because it will be optimizing at most 7 individuals (recall, 14 cross-over operations for each individual) with only small single-variable modifications, which are not always accepted. The LLT and the RLT seem to perform alike, both on the short and the long term, on both MacroNAS-C10 and -C100. The bars underneath the graphs shows p-values for the differences of MO-GOMEA with the LLT compared to the RLT FOS (top bar) and to the Univariate FOS (bottom bar) by means of the colors. Although the LLT FOS and Univariate FOS trade off better performances during the search, the RLT FOS is almost never (significantly) outperformed by the LLT FOS. Nevertheless, the many gray areas in the bar indicate that there are mostly no significant differences throughout the search. Perhaps differences are visible in the behavior of both configurations.

Figure 5.14 shows the achieved fronts of one run of each configuration. The univariate FOS causes the slower convergence in the beginning of the optimization by making MO-GOMEA struggle to find balanced solutions, although the efficient and reasonably accurate networks are found fast. The LLT FOS seems exploit learned information by approaching the front of balanced solutions earlier than the RLT FOS.

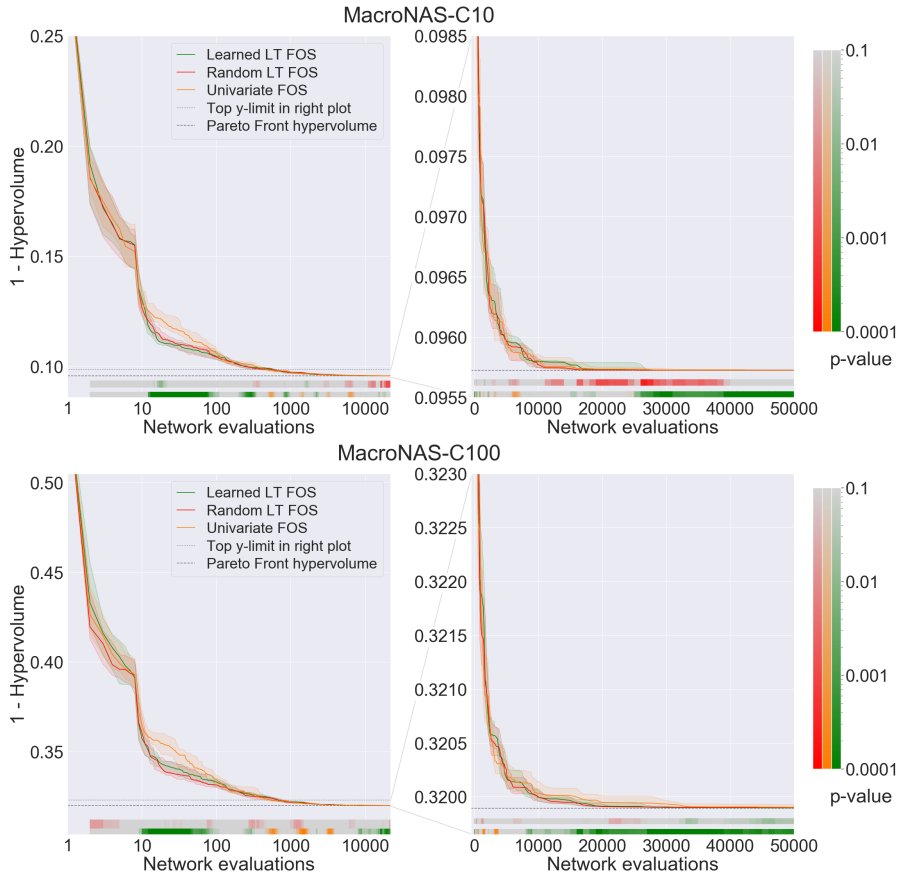


Figure 5.13: Experiment 5: Medians and 25/75th percentile of 30 runs of MO-GOMEA with different FOS schemes in the Optimal Mixing phase, on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). All configurations use *IMS* (together with adaptive clustering) and forced improvements. The colored bars indicate p-values for Mann Whitney U tests for differences of LLT FOS compared to RLT FOS (top bar) and Univariate FOS (bottom bar). Grey means no significant difference, green/red/yellow means MO-GOMEA using a LLT FOS is better/worse than MO-GOMEA using a RLT FOS and better/worse than MO-GOMEA using a Univariate FOS, with certain p-values. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

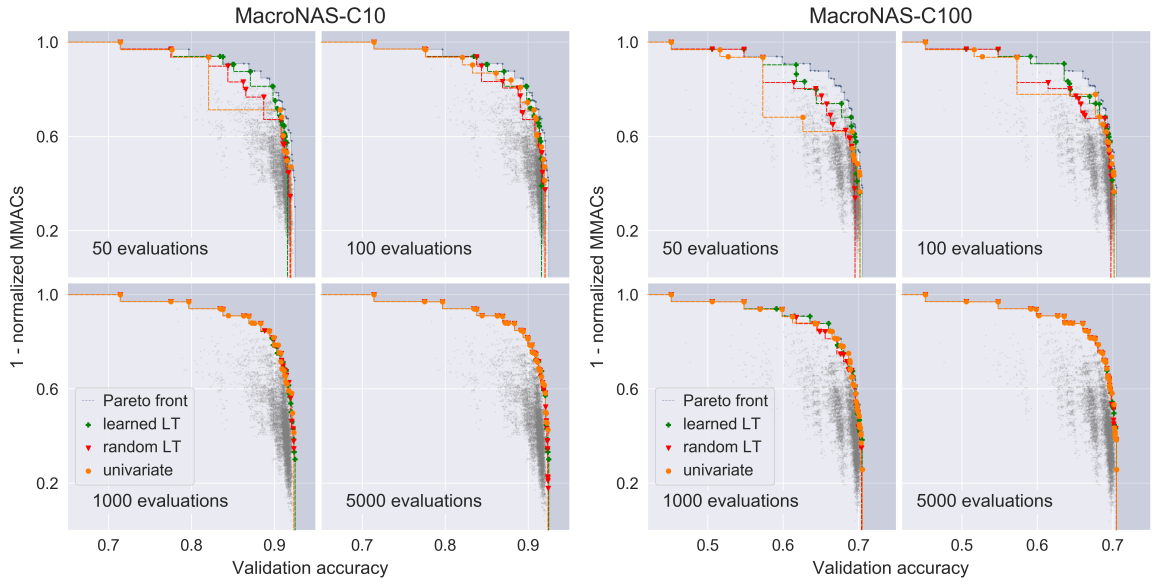


Figure 5.14: Experiment 5: Achieved fronts for one example run of MO-GOMEA for different FOS structures in the Optimal Mixing phase on MacroNAS-C10 (left) and MacroNAS-C100 (right).

To more thoroughly understand what is happening, the ratio of successful mixing attempts for differently sized FOS elements are shown for the LLT FOS and the RLT FOS on MacroNAS-C100 in Figure 5.15, separately for the two extreme clusters and the mixed clusters[3]. The differences between the LLT FOS and the RLT FOS are small for the extreme MMACs cluster, likely because the few efficient networks are found quickly and do not possess interesting dependency structures. For the mixed clusters, the LLT seems to have found groups of variables of size 10 (so almost entire solutions) that mix well, probably causing the quicker approach of the front.

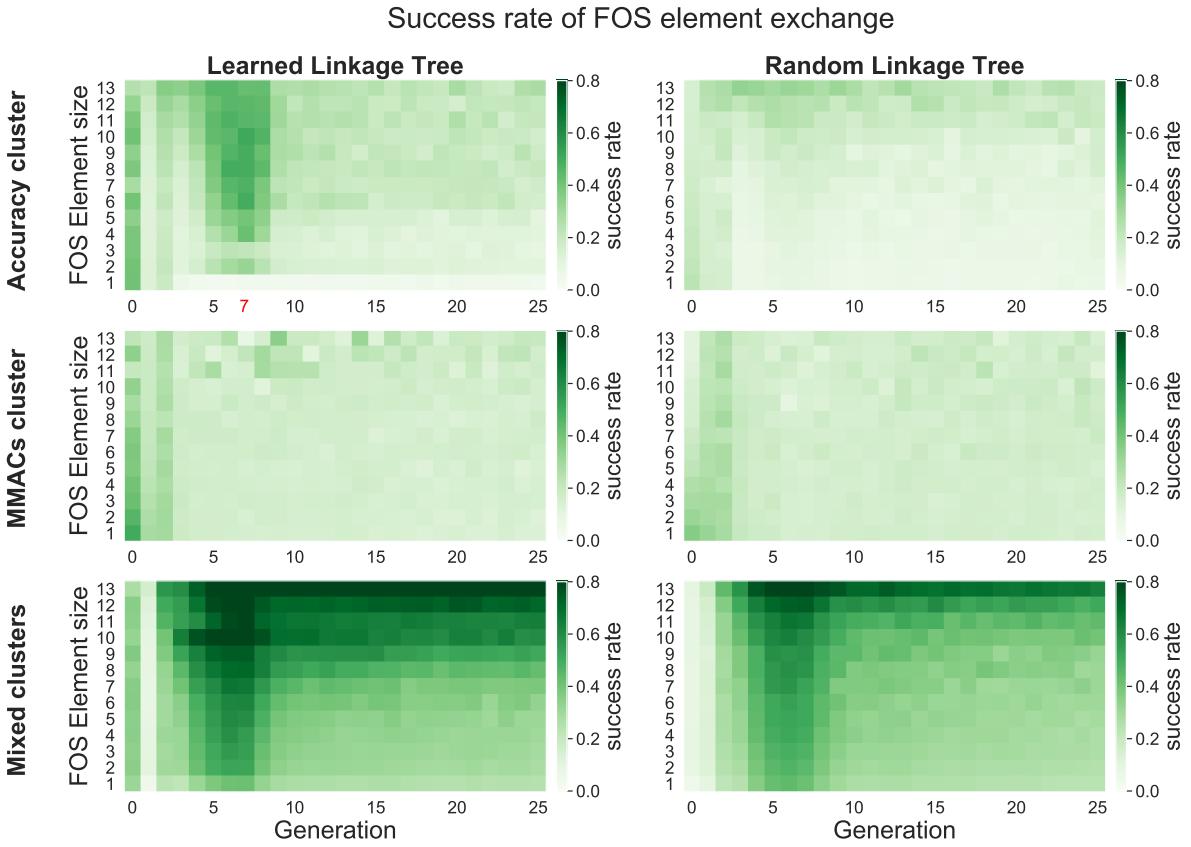


Figure 5.15: Experiment 5: Heatmap of success rates of FOS element exchange for the LLT (left) and the RLT (right) on MacroNAS-C100, against the generations in MO-GOMEA. The results are split into the two extreme clusters and mixed clusters. Results are combined from 100 runs. Results from mixing in the forced improvements phase are not included. Pattern of generations 15-25 continues after 25 generations, as shown in Appendix C.2.

The largest difference between the LLT and RLT FOS can be observed in the extreme cluster that single-objectively optimizes for accuracy. Apparently, the LLT contains groups of variables that result in an improved accuracy when mixed. The most dependencies seem to be captured at generation 7, for which the groups of elements that resulted in the most successful cross-over are shown with green bars in Figure 5.16. Additionally, the ratio of a specific FOS element being present in the final LLT, out of all LTs that are learned, is shown by the orange bars. FOS-elements that contain only one variable do not contain interesting patterns, but for higher sizes the results become more interesting. The last variables of the encoding $[13, 14]$ do not mix well independently, but together they mix well and are detected frequently. Variables 13 and 14 also appear in almost all other successful FOS elements of size higher than 2. Also the group of variables $[5, 7, 8]$ appear to share dependency, because they have a high mixing success ratio in both FOS elements of size 3 and 5 (combined with $[13, 14]$ reaches even a 100% mixing success rate). Furthermore, the graph displaying FOS elements of size 8 shows a very high mixing success rate (but unfortunately a low detection rate) for the FOS element $[1, 2, 3, 5, 7, 8, 13, 14]$, by combining $[5, 7, 8]$ and $[13, 14]$ with $[1, 2, 3]$, an element that, although not very successful when mixing on itself, is

[3]Since the clusters change over time in terms of position and the individuals they cover, it is not possible to inspect every mixed cluster separately

still relatively frequently detected for size 3 FOS elements. Note all groups $[1, 2, 3, (4)]$, $[5, (6), 7, 8]$ and $[13, 14]$ are groups of variables denoting cells that appear in blocks separated by reduction layers from other cells (see Figure 3.1). The element $[1, 2, 3, 5, 7, 8, 13, 14]$, with the addition of variables 4 and 6 for size 10, and subsequently variables 10 and 11 (also a reduction-layer-separated tuple) for respectively size 11 and 12, leads to FOS elements that are all detected more frequently and still have around 50% success ratio. In Appendix C.2, graphs are shown with the FOS elements sorted on detection rate, in which a trend becomes visible: many tuples and triplets of variables that appear between the same reduction layers are detected more frequently.

Knowing this, it makes sense to focus on mixing full blocks between reduction layers, although it restricts MO-GOMEA from delicately mixing single variables that might be necessary to find the final solutions on the front. In that case, a combination of MO-GOMEA mixing only full blocks and a subsequent iteration of local search (as presented in Section 4.1.1) to fine-tune the solutions might be an interesting combination to investigate. If optimizing single variables turns out to be unnecessary, removing the univariate structure from the LT FOS, such that only non-singleton are mixed can be an interesting option as well, as it cuts the evaluations needed for optimal mixing in half, thereby accelerating the search. A third option is to start with only tuples as leaves of the LT, and using the same procedure to build the rest of the LT. The performance of the latter option, although not learned, but in different orders assembled *tuple trees*, is shown in additional experiments in Appendix C.2.

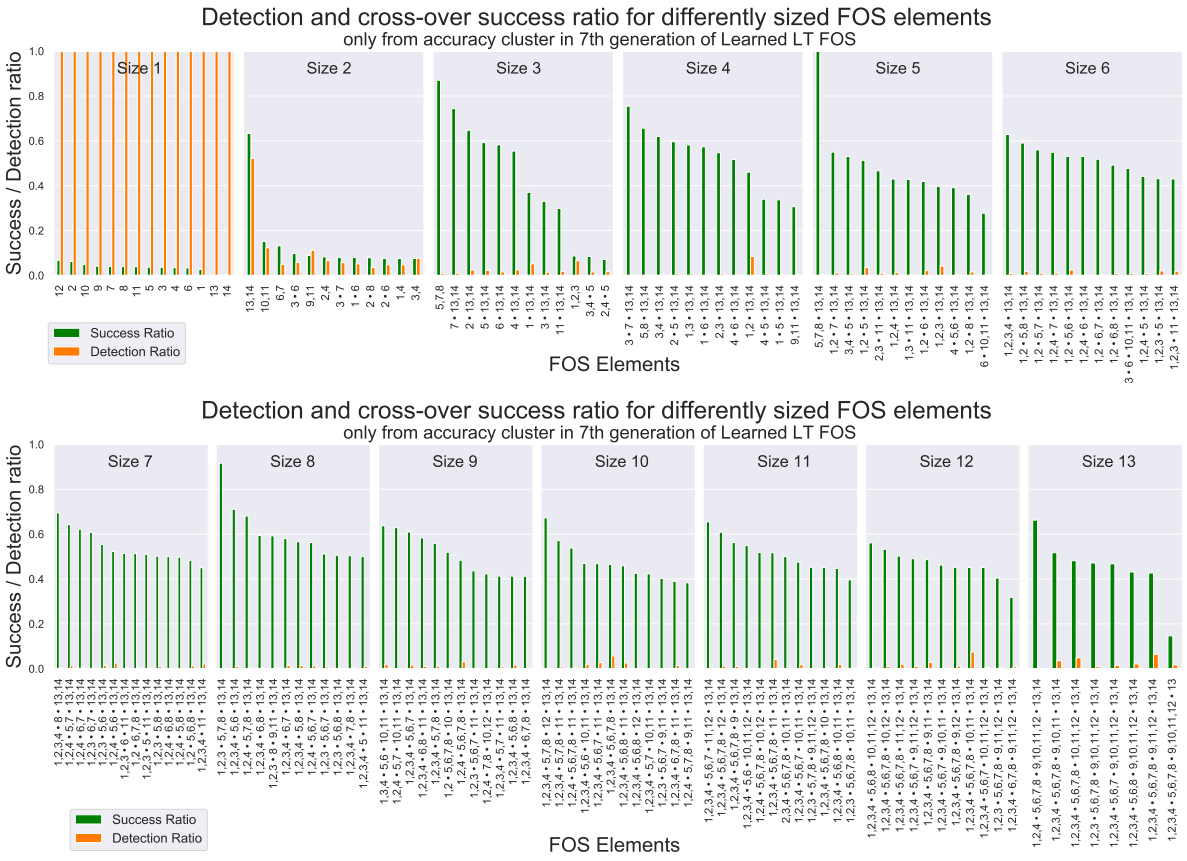


Figure 5.16: Experiment 5: Success and detection rate of differently sized FOS elements, of generation 7 of MO-GOMEA with LLT FOS on MacroNAS-C100. Success ratio of a FOS element: # of mixes result in accepted change / # of total mixes tried. Detection ratio of a FOS element: # appearances in LLT / # of LT's learned. Results are combined from 100 runs. Only the 12 most successful FOS elements that appear in ≥ 50 cross-over operations are shown. Elements within a FOS element are separated by a "•" when the cells belonging to them are separated by reduction layers (see Figure 3.1). Results from mixing in the forced improvements phase are not included. Results sorted on detection ratio are shown in Appendix C.2

Conclusion Some structure is learned by MO-GOMEA, although the existing dependencies between variables are mostly present in the region of the front containing highly accurate networks. These networks are harder to find than efficient networks, for which linkage learning seems ineffective. The

latter can be partly the result of much redundancy in the encodings of solutions in this region of the front, which causes noise in the linkage learning routine (recall that the identity layers necessary for finding efficient networks cause multiple encodings to map to the same Pareto-Optimal architecture). Additionally, and more importantly, these architectures are simply easier to find and do not require learned dependency models to be discovered.

Overall, linkage learning does not result in great improvements, but it also does not hurt to use it, also because the resources consumed for learning the structure by the search algorithm are almost always negligible compared to the resources consumed by training and evaluating the networks. Possibly better, more efficiently mixing cross-over masks can be designed, although these would then only hold for this search space. Other NAS search spaces might encode the architectures in an entirely different way, therefore also requiring differently configured cross-over masks. The advantage of model-based architectures is that if there is any underlying dependency present, it is captured and exploited automatically (provided a correctly configurable structure is used), so no manual modeling of the cross-over mask is required. Therefore, it is required to include it in the search algorithm to find new state-of-the-art networks, although undemanding practitioners can safely ignore it if the effort required to implement it is too high.

Using the knowledge obtained from this automatic linkage learning analysis to construct a better, more MacroNAS-tailored FOS structure is considered future work, although involving blocks in-between reduction layers might provide a first step in the right direction.

5.4. Concluding remarks on results

There is one thing that the results of all experiments have in common: The differences and their significance hold in theory, but one could question what the results would mean in practice. In other words, how substantial are the effects of the different configurations of MO-GOMEA in the experiments. The convergence of the hypervolume of 30 runs of MO-GOMEA with *IMS*, adaptive clustering, forced improvements and learned LT cross-over is once again measured against the number of evaluations. To get the hypervolume with respect to the test accuracy, the elitist archive for a certain number of evaluations is taken and the test accuracy for every solution in it is looked up in the NAS benchmark dataset. Since the validation accuracy is not necessarily perfectly correlated with the test accuracy, the non-dominated solutions are filtered out. Then, the hypervolume of the resulting front is calculated with test accuracy and the number of MMACs of every solution as input. The consequence is that if the Pareto Front w.r.t. test accuracy contains different solutions than the Front w.r.t. validation accuracy, algorithms that converge and cover the Pareto Front during the search, will never possess the maximally achievable hypervolume for test accuracy. Figure 5.17 shows that on both MacroNAS-C10 and MacroNAS-C100 the optimal hypervolume for test accuracy is never reached, even though the optimal hypervolume for validation accuracy is achieved.

Before further discussion of the results, one clarification is needed. In Figure 5.17, the median hypervolume of MO-GOMEA w.r.t. the test accuracy is better until roughly 100 evaluations on MacroNAS-C100. A possible explanation for this is that since the networks in the benchmark dataset have been trained with a one-shot approach, which makes use of just one specific train/validation/test-split of the data, the resulting test dataset turned out to be easier to classify than the validation dataset. The validation set possibly contains more images that deviate from the average images (i.e., outliers) in CIFAR-100, therefore harder to predict as accurate. Figure 5.18 (right) shows that this circumstance is mostly present for efficient networks, whereas well-performing (but complex) architectures seem to suffer less from this manifestation. Most likely these networks are complex enough to be better equipped of dealing with outliers.

This observation does not affect the point that will be made in this section. As shown in Figure 5.17, there is a clearly visible gap between the hypervolume convergence w.r.t. validation accuracy and test accuracy. Note that the zoomed-in graphs on the right side of Figure 5.17 have a larger hypervolume range (vertical axis) than other convergence graphs in this chapter. Therefore, this gap is much larger than any gap between different configurations in the experiments in this chapter. Additionally, in both MacroNAS-C10 and MacroNAS-C100, there comes a number of evaluations where the test hypervolume gets slightly worse, even though the validation hypervolume is still increasing. At this point, overfitting on the search space happens. The few networks with better validation accuracy take much effort to find, but turn out to have worse test accuracy compared to earlier found architectures. This is a result

of the stochasticity in the initialization, training and evaluation of the networks and the unpredictable discrepancies between the validation and the test data.

This raises the question, when to stop searching? At some point in time, the gain of finding even better architectures are not worth the computational resources needed for this search. This depends of course on the domain and the requirements, but the actual performance (test accuracy) of the produced set of solutions should absolutely not deteriorate by searching further. Although this is an important issue, it is a different objective to improve upon this issue, compared to improving the search phase of NAS by analyzing and tweaking the optimization algorithm. This will further be discussed in the discussion in Chapter 6.

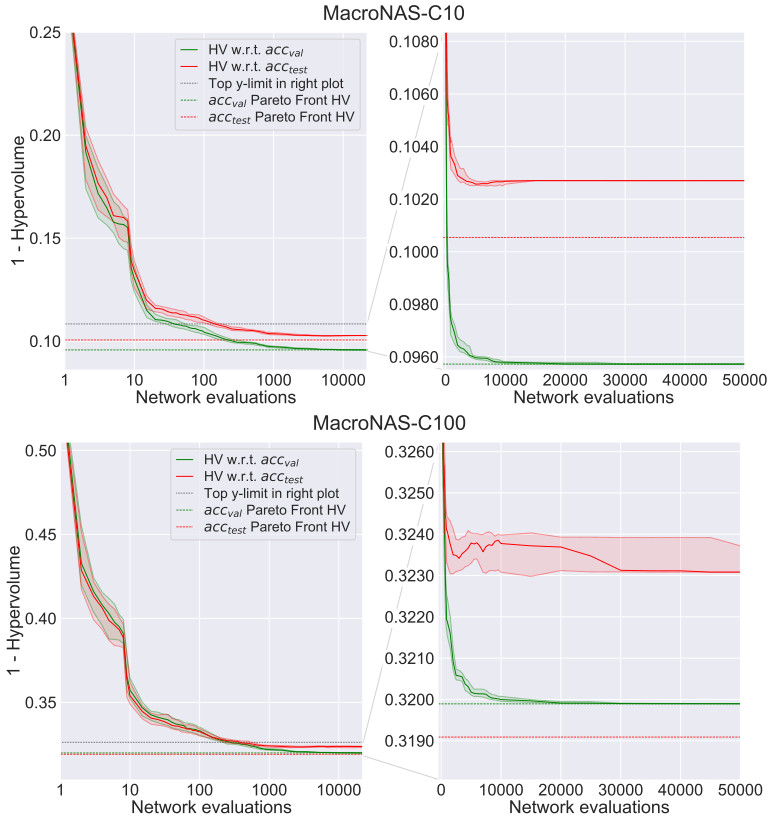


Figure 5.17: Medians and 25/75th percentile of 30 runs of MO-GOMEA on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). Convergence of hypervolume terms of validation accuracy and test accuracy of the same runs is shown by the different curves. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

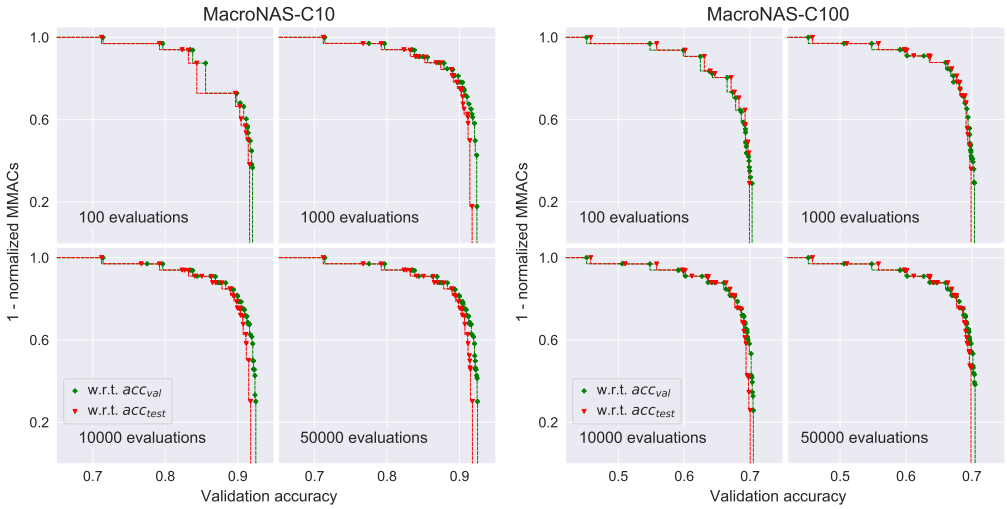


Figure 5.18: Achieved fronts for one example run of MO-GOMEA w.r.t. acc_{val} and acc_{test} on MacroNAS-C10 (left) and MacroNAS-C100 (right).

6

Discussion and Future Work

The meaning of most results have already been discussed in their respective section. This chapter will however take a slight step back to discuss a higher-level view on the results and the experiments performed in this thesis.

6.1. Substantiality of results

As already mentioned in Section 5.4, some of the differences between configurations are significant in theory, but the substantiality is small compared to the gap between the validation accuracy and test accuracy, therefore questioning the practical usefulness of further optimizing the search phase of NAS (within the scope of the type of benchmarks considered in this thesis). In Chapter 4 it is already shown that even with simple random and locally searching algorithms a good front can be obtained. In fact, using LS, which is a less powerful optimization algorithm than the EAs NSGA-II and MO-GOMEA, a front is found with better generalizing networks. Overfitting is therefore also for a meta-learning problem like NAS an aspect that needs to be taken into account when developing new NAS approaches. The chosen optimization algorithm should only be sufficiently complex and/or powerful, in contrast to the increasingly complex search strategies that are proposed for NAS nowadays.

On the one hand overfitting can be seen as a separate issue that is inherent to every machine learning technique when test data is absent (which is usually the case in real-world scenarios): the best an approach can do is optimize for models that perform a task well on the training and validation data. Although the discrepancy between train/validation data and test data is an important aspect to keep in mind when developing new NAS approaches, it is one that falls outside the scope of this thesis.

On the other hand, the search strategy is just one part of NAS, which raises the question with what margin the results can be improved when all NAS components are optimized simultaneously. The performance estimation component is also crucial for a good NAS performance: the better the networks are trained, the more reliable the performance estimation of an architecture will be. The result is a more smooth and reliable fitness landscape (for as far as that is possible in NAS), at the cost of expensive evaluations. At the other end are very quick and rough architecture performance estimations that can be obtained by shortly training a network. This allows the search strategy to do more evaluations within the same time, but a noisier, less accurate fitness landscape needs to be searched. The simultaneous optimization of the performance estimation component together with the search strategy also falls outside the scope of this thesis.

6.2. Comparisons based on evaluations versus time

This thesis only makes comparisons based on the number of evaluations an algorithm requires to achieve its front of networks. Although it is interesting to inspect the search strategies from an algorithmic point of view, it does not take into account training times of networks. Differently sized networks can have higher or lower training times depending on their size (while still using an equal epoch budget for training each network) and should therefore not be treated equally. An algorithm that trains relatively many small networks (but uses more evaluations) could be superior to algorithms that train relatively many large networks, if the comparison is based on (wall-clock) time. To see how the conclusions

translate to time-based comparisons is considered future work.

6.3. Mainly MacroNAS

The experiments in this thesis focus mainly on the proposed macro-level search space MacroNAS. Although the networks in the search space can be of variable size, all networks are sequentially connected and only allow multi-branched structures within the cell options. For networks with state-of-the-art accuracy performance, skip-connections are required between different parts of the network, which are not supported in MacroNAS. Although it would be interesting future work to investigate how the differently configured search algorithms behave on search spaces that allow more freedom in the structure of its networks, it also requires intricate design of the encoding of such a space to make it searchable by algorithms. Macro-level NAS benchmarks with pre-computed values that include these options do not exist yet, so performing a search in real-time would be an expensive endeavor. Without applying efficient, but ranking-affecting NAS techniques like weight-sharing and one-shot that ease the computational burden, this is not yet feasible for academics without access to immense computational resources.

Apart from more intricate search spaces, there is also no guarantee that the results from comparing different configurations of MO-GOMEA on MacroNAS translate one-to-one to other, similar but differently-encoded search spaces, as also reported in [77]. However, changing the encoding is in some sense similar to changing the operators of the EA; the cross-over mask that works best for a search space is the best-aligned one. For example, the artificial problem Trap5 with tightly encoded trap functions requires a different cross-over mask than when variables for a trap function are spread out in the encoding. Although it is considered future work to confirm transferable performances of different configurations of MO-GOMEA, it is expected that MO-GOMEA, due to its general applicability by the IMS, adaptive clustering and automatic dependency model learning cause it to behave well also on other search spaces.

6.4. Robustness

Although most results are combined from 30 differently-seeded runs, the evaluations of networks are not stochastic, because of the usage of NAS benchmarks with pre-computed values. The result therefore is that for each run, the Pareto Front contains the same solutions. Also peculiarities in the benchmarks, e.g., the interesting train/validation/test-split in MacroNAS-C100 that cause the test accuracy to be higher than validation accuracy for some networks (see Section 5.4), are affecting all runs of every algorithm (so in that sense the comparison is still valid), but it does not show how different search strategies deal with other peculiarities in different data splits. Though it is possible to assess the robustness of different algorithms w.r.t. differently seeded runs (so, different solutions that an algorithm considers first), it is hard to assess their robustness w.r.t. varying noise in the performance estimation part of NAS.

Additionally, even though the range of MMACs of networks is normalized to the range of $[0, 1]$, there is a discrepancy compared to the range of validation accuracies; for MacroNAS-C10 it is (roughly) $[0.70, 0.93]$ and for MacroNAS-C100 (roughly) $[0.45, 0.71]$. The result is that the MMAC range is roughly 4 times larger than the accuracy range, consequently affecting the hypervolume. A solution that is 5 percent better in terms of the effective accuracy range has less impact on the hypervolume than a solution that is 5 percent better in terms of the effective MMACs range, as illustrated in Figure 6.1. This is however compensated by adding the all-identity network to the initial population, such that the influence of this skewed objective space is reduced. Still, normalizing the accuracy as well, although it can be hard to predict the effective range of the accuracy beforehand, can influence how the algorithms compare w.r.t. hypervolume convergence, and might even have beneficial impact on the search performance of algorithms. Alternatively, changing the reference point for calculating the hypervolume might also be effective to some extent.

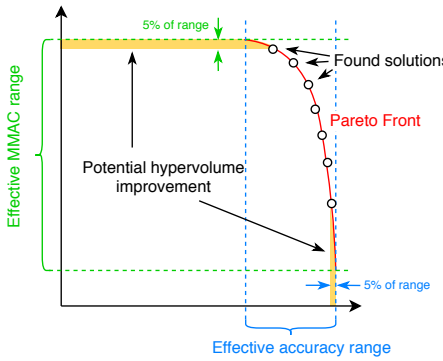


Figure 6.1: Skewed Pareto front: potential hypervolume gains are the largest for improvement in the MMAC objective, compared to the accuracy objective.

7 Conclusion

7.1. Baselines for NAS

In Chapter 4, two new baseline algorithms, Local Search (LS) and Uniform Size Random Search (USRS) are proposed, in order to answer Research Question 1:

Research Question 1. *What simple, yet well-performing algorithms can provide a solid baseline in multi-objective neural architecture search?*

These algorithms are meant to facilitate research on NAS, by providing good performance on multi-objective NAS search spaces while still being simple and easy to implement. LS performs close to state-of-the-art EAs MO-GOMEA and NSGA-II, even on an enlarged search space containing 673 Billion encodings (mapped to 104 Billion architectures), by performing one round of greedy optimization for each variable in an initially randomly sampled encoding. USRS on the other hand is an improvement upon naive RS by first sampling the size of the network to evaluate, as RS falls victim to the skewed distribution of the objective space. Both LS and USRS are easy to understand and implement and therefore provide accessible alternatives to RS for comparing new NAS approaches against. Also practitioners that quickly want to assemble and perform a NAS approach can benefit from the proposed algorithms, by trading off only a slight decrease in performance for a huge increase in usability.

In addition, the first ever macro-architecture NAS benchmark datasets called MacroNAS-C10 and MacroNAS-C100 are proposed, which allow for multi-objective optimization of variable-length architectures with regard to validation accuracy and model complexity and provides pre-computed values for all architectures in the search space. Hopefully this benchmark will support the developments in the research area of NAS, by allowing academics to compare optimization algorithms for NAS in a computational power friendly way.

7.2. Effectiveness of EA mechanisms on NAS

In Chapter 5 an ablation study of MO-GOMEA on the MacroNAS benchmark datasets is performed, in order to answer Research Question 2:

Research Question 2. *What mechanisms make a state-of-the-art multi-objective evolutionary algorithm perform well on a multi-objective macro-level neural architecture search problem?*

The automatic population-sizing scheme offers a welcome anytime-performance configuration, minimizing the random search in the initialization phase and sacrificing only negligible performance which would be gained by selecting the optimal population size; something that is very hard to know beforehand without doing expensive analysis. While the performance gained by employing clustering in objective space is not huge, the behavior of MO-GOMEA is affected in that a more diverse front is achieved over not using clustering, although both configurations discover an equally good front when allowing the search to continue for an extensive number of evaluations. The precise number of clusters used does not matter on MacroNAS-C10 and -C100, so using the adaptive clustering scheme tied

to the *IMS* is advised. The utilization of extreme clusters depends on the focus of the practitioner; extreme clusters explore the extreme ends of the front more thoroughly, but also endures a slightly slower discovery of balanced solutions compared to not single-objectively optimizing a cluster for each objective. At last, although there is not much improvement by using a learned LT over a random LT, it can help to align cross-over masks with the search space at hand, which is advantageous if other search spaces exhibit linkage that is unpredictable beforehand. The LLT picks up on the presence of dependencies between variables that denote cells in-between reduction layers, but this linkage is mainly present in sub-populations in the highly-accurate networks range. For efficient networks, architectures with low MMACs in this case, automatic model-learning is not necessary. For approaching the front containing networks that balance the two objectives, automatic dependency-learning mechanisms can accelerate the search, although it is uncertain how exactly. Still, model-based EAs can provide an effective approach to NAS problems that does not require much problem-specific tuning.

7.3. Prospects on NAS

Although many impressive results have been published in the field of NAS in the recent years, it is still in its early days, therefore it is uncertain where exactly the field of research is headed. After working on this thesis and plunging myself into the field of NAS, the possibilities and limitations of NAS have become more clear. In my opinion, NAS will ultimately be used for three purposes:

1. Finding new state-of-the-art performing neural networks, by tuning the search space to include structures that are known to perform well on the task at hand and applying powerful search algorithms, possibly combined with hyper-parameter tuning. For some domains, e.g., logistics or medical, pushing for every percentage of accuracy can be worth it, and NAS can help to achieve it.

2. Discovering very innovative networks and novel architectural feats, by widening the search space rigorously to allow lots of freedom in architectures. Finding the next "residual connection" or "inception module" that will result in the next DL breakthrough, does not come without challenges. Defining the search space will be the most challenging task, because it should allow for structures that we, as biased researchers, cannot think of yet. Moreover, performing a sensible search on such a vast search space is also not straightforward and the computational requirements might be a bottleneck, so smart tricks and work-arounds in this direction need to be (further) developed. Still, this purpose of NAS is one that has the largest potential to have an exceptional and long-lasting impact on the field of DL.

3. Refining existing well-performing architectures, by finding alternative structures that have a (slightly) different shape, but are at least or almost as powerful as the original architecture. One example of this is the improvement of LSTM-modules [57]. In this direction lie also the goal of redesigning networks for so-called *edge-devices*, like mobile phones and Internet-Of-Things devices, which typically have limited resources (memory, CPU, GPU) and might not have reliable internet connections to access resources in the cloud. These limitations cause the devices to be unable to train and/or utilize large and powerful networks. NAS can therefore be used to find architectures that perform the same task but require less resources, thus enhancing the potential of these devices.

Besides these purposes, there are a few topics within NAS that are gaining more interest very recently, of which the entirety of the fields of NAS and DL can profit.

Optimizing performance estimation Since the time and resources needed for training and evaluating networks are the largest bottleneck of doing NAS, effort will be spent on trying to reduce or circumvent it. Next to optimization and developing an understanding of the search phase, as is the focus in this thesis, scientists have already put in work to improve upon the evaluation part of NAS as well. For example, as mentioned in Section 2.3, weight sharing between networks already greatly decreases the time spent on training networks, although this is suspected to influence the (outcome of the) search [86]. Recent works have tried to assess the performance of a network by its initialized state, without even training them [51]. Although the scoring of the CNNs is not perfectly accurate, this method can still be used to weed out poor architectures in order to prevent unnecessary evaluations. Another work [42] trains

candidate architectures on unsupervised tasks on the training data (e.g., predicting the reconstruction of a patch-shuffled image). Both the works [51] and [42] therefore suggest that the potential of an architecture might be inferred from its relation to the dataset. This is something that, once better researched and understood, could take NAS to the next level, thereby compressing a procedure that takes hours into something that takes seconds. This makes NAS increasingly accessible for a many more interested parties (e.g., practitioners), that do not have access to dozens of GPUs. Additionally, this would make exploring gigantic search spaces tractable, and this would improve the chance of finding truly novel architectures and architectural feats. The design and encoding of such an open search space will remain a challenge.

New types of layers The invention of convolutions, one of the most crucial components in computer vision, was a breakthrough in Deep Learning, though it was a handcrafted idea. That raises the question, can we discover such structures automatically? Recent work [53] attempts to learn convolutions from scratch. This will definitely spark other scientists to focus on discovering novel layers and/or building blocks, not only for computer vision, but perhaps also discovering the "convolutional layer" for other deep learning areas, say natural language processing tasks for example.

Other DL domains The majority of existing research on NAS has focused on finding good performing architectures for image processing tasks, which are mostly convolutional neural networks. However, the interest grows for applying NAS on other DL tasks, e.g., to optimize the architecture of recurrent neural networks (RNNs), LSTMs, and generative adversarial networks (GANs). Since the first approaches for these other DL tasks have surfaced already, it is not a question of whether they will be competing with state-of-the-art, but when.

Bibliography

[1] Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S. Awwal, and Vijayan K. Asari. The history began from alexnet: A comprehensive survey on deep learning approaches, 2018.

[2] Hany Hassan Awadalla, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, Will Lewis, Mu Li, et al. Achieving human parity on automatic chinese to english news translation. *arXiv preprint arXiv:1803.05567*, 2018.

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[4] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[5] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 550–559, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/bender18a.html`.

[6] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[7] Peter AN Bosman and Dirk Thierens. Linkage neighbors, optimal mixing and forced improvements in genetic algorithms. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 585–592, 2012.

[8] Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018.

[9] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[10] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Hailong Ma. Multi-objective reinforced evolution in mobile neural architecture search. *arXiv preprint arXiv:1901.01074*, 2019.

[11] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[12] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[13] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019.

[14] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[15] Gökcen Eraslan, Žiga Avsec, Julien Gagneur, and Fabian J Theis. Deep learning: new computational modelling techniques for genomics. *Nature Reviews Genetics*, 20(7):389–403, 2019.

[16] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. *Artificial intelligence through simulated evolution*. Wiley, 1966.

[17] Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *arXiv preprint arXiv:2006.05415*, 2020.

[18] David E Goldberg. *Genetic algorithms*. Pearson Education India, 2006.

[19] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[20] Georges Harik et al. Linkage learning via probabilistic modeling in the ECGA. *IlliGAL report*, 99010, 1999.

[21] Georges R Harik and Fernando G Lobo. A parameter-less genetic algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pages 258–265, 1999.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[24] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[25] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[27] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[28] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2016–2025. Curran Associates, Inc., 2018.

[29] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, pages 1–62, 2020.

[30] Ye-Hoon Kim, Bhargava Reddy, Sojung Yun, and Chanwon Seo. NEMO: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In *ICML 2017 AutoML Workshop*, 2017.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[32] Rob Kitchin. *The data revolution: Big data, open data, data infrastructures and their consequences*. Sage, 2014.

[33] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[35] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[36] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[37] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR, 2020.

[38] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen. Medical image classification with convolutional neural network. In *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 844–848, 2014.

[39] Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. *arXiv preprint arXiv:1909.02453*, 2019.

[40] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60 – 88, 2017. ISSN 1361-8415.

[41] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[42] Chenxi Liu, Piotr Dollár, Kaiming He, Ross Girshick, Alan Yuille, and Saining Xie. Are labels necessary for neural architecture search? *arXiv preprint arXiv:2003.12056*, 2020.

[43] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

[44] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. NSGA-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 419–427, 2019.

[45] Ngoc Hoang Luong, Han La Poutré, and Peter A. N. Bosman. Multi-objective gene-pool optimal mixing evolutionary algorithms. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 357–364, New York, NY, USA, 2014.

[46] Ngoc Hoang Luong, Marinus OW Grond, Han La Poutré, and Peter AN Bosman. Scalable and practical multi-objective distribution network expansion planning. In *2015 IEEE Power & Energy Society General Meeting*, pages 1–5. IEEE, 2015.

[47] Ngoc Hoang Luong, Tanja Alderliesten, Arjan Bel, Yury Niatsetski, and Peter A. N. Bosman. Application and benchmarking of multi-objective evolutionary algorithms on high-dose-rate brachytherapy planning for prostate cancer treatment. *Swarm and Evolutionary Computation*, 40:37–52, 2018.

[48] Mingyun Lv, Jun Li, Huafei Du, Weiyu Zhu, and Junhui Meng. Solar array layout optimization for stratospheric airships using numerical method. *Energy Conversion and Management*, 135:160–169, 2017.

[49] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[50] Stefanus C. Maree, Ngoc Hoang Luong, Ernst S. Kooreman, Niek van Wieringen, Arjan Bel, Karel A. Hinnen, Henrike Westerveld, Bradley R. Pieters, Peter A.N. Bosman, and Tanja Alderliesten. Evaluation of bi-objective treatment planning for high-dose-rate prostate brachytherapy—a retrospective observer study. *Brachytherapy*, 18(3):396 – 403, 2019.

[51] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. *arXiv preprint arXiv:2006.04647*, 2020.

[52] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[53] Behnam Neyshabur. Towards learning convolutions from scratch, 2020.

[54] V. Nissen and J. Propach. On the robustness of population-based versus point-based optimization in the presence of noise. *IEEE Transactions on Evolutionary Computation*, 2(3):107–119, 1998.

[55] Tom den Ottelander, Arkadiy Dushatskiy, Marco Virgolin, and Peter A. N. Bosman. Local search is a remarkably strong baseline for neural architecture search, 2020.

[56] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of Machine Learning Research*, volume 80, pages 4095–4104. PMLR, 2018.

[57] Aditya Rawal and Risto Miikkulainen. From nodes to networks: Evolving recurrent neural networks. *arXiv preprint arXiv:1803.04439*, 2018.

[58] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2902–2911, International Convention Centre, Sydney, Australia, 2017. PMLR.

[59] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[60] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020.

[61] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*, pages 1–11. IEEE, 2015.

[62] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241. Springer International Publishing, 2015.

[63] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling. Deep learning detecting fraud in credit card transactions. In *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pages 129–134, 2018.

[64] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[65] B. Sahiner, Heang-Ping Chan, N. Petrick, Datong Wei, M. A. Helvie, D. D. Adler, and M. M. Goodsitt. Classification of mass and normal breast tissue: a convolution neural network classifier with spatial domain and texture images. *IEEE Transactions on Medical Imaging*, 15(5):598–610, 1996.

[66] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[67] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[68] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419): 1140–1144, 2018.

[69] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[70] Adam Slowik and Halina Kwasnicka. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, pages 1–17, 2020.

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

[72] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504, 2017.

[73] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2):394–407, 2019.

[74] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[75] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[76] Dirk Thierens and Peter A. N. Bosman. Optimal mixing evolutionary algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, page 617–624, New York, NY, USA, 2011.

[77] Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search, 2020.

[78] Martin Wistuba. Finding competitive network architectures within a day using UCT. *arXiv preprint arXiv:1712.07420*, 2017.

[79] Martin Wistuba. Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations. In Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 243–258. Springer International Publishing, 2019.

[80] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

[81] Lingxi Xie and Alan Yuille. Genetic CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017.

[82] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[83] Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2019.

[84] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 7105–7114, Long Beach, California, USA, 2019. PMLR.

[85] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[86] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020.

[87] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018.

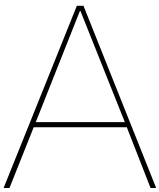
[88] Arber Zela, Julien Siems, and Frank Hutter. NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*, 2019.

[89] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018.

[90] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[91] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Appendices



“Local Search is a Remarkably Strong Baseline for Neural Architecture Search”

Local Search is a Remarkably Strong Baseline for Neural Architecture Search

T. Den Ottelander[1], A. Dushatskiy[1], M. Virgolin[1], and P.A.N. Bosman[1,2]

[1] Centrum Wiskunde & Informatica, Amsterdam, the Netherlands
{tdo,arkadiy.dushatskiy,marco.virgolin,peter.bosman}@cwi.nl
[2] Delft University of Technology, Delft, the Netherlands

Abstract. Neural Architecture Search (NAS), i.e., the automation of neural network design, has gained much popularity in recent years with increasingly complex search algorithms being proposed. Yet, solid comparisons with simple baselines are often missing. At the same time, recent retrospective studies have found many new algorithms to be no better than random search (RS). In this work we consider, for the first time, a simple Local Search (LS) algorithm for NAS. We particularly consider a multi-objective NAS formulation, with network accuracy and network complexity as two objectives, as understanding the trade-off between these two objectives is arguably the most interesting aspect of NAS. The proposed LS algorithm is compared with RS and two evolutionary algorithms (EAs), as these are often heralded as being ideal for multi-objective optimization. To promote reproducibility, we create and release two benchmark datasets, named MacroNAS-C10 and MacroNAS-C100, containing 200K saved network evaluations for two established image classification tasks, CIFAR-10 and CIFAR-100. Our benchmarks are designed to be complementary to existing benchmarks, especially in that they are better suited for multi-objective search. We additionally consider a version of the problem with a much larger architecture space. While we find and show that the considered algorithms explore the search space in fundamentally different ways, we also find that LS substantially outperforms RS and even performs nearly as good as state-of-the-art EAs. We believe that this provides strong evidence that LS is truly a competitive baseline for NAS against which new NAS algorithms should be benchmarked.

Keywords: neural architecture search · local search · evolutionary algorithm · random search · multi-objective NAS · NAS baseline

1 Introduction

Deep learning has achieved excellent results for machine learning tasks in heterogeneous fields, including image recognition [23], natural language processing [21], games [27, 31], and genomics [26]. While the availability of large amounts of data [16] and hardware advancements (e.g., tensor parallel processing units [14]) have been key enabling factors for deep learning, the success of the field is also

due to the ingenious design of competent neural network architectures (consider, e.g., the invention of residual connections [12]).

Proper and innovative architecture design requires experience, intuition, and expensive trial-and-error. This has sparked research on techniques to automate this task, i.e., the field of Neural Architecture Search (NAS) has emerged [10, 33]. NAS research is quickly gaining popularity: 2019 alone counts almost 250 publications[3]. Most NAS proposals present new, typically increasingly complex, NAS algorithms. However, recent work questions whether the need for many of these algorithms is actually justified: [25] and [35] showed that several NAS algorithms are no better than Random Search (RS) when validated properly.

While RS can be considered perhaps the simplest form of a baseline, it is also by far the worst form of heuristic search in many cases, with the worst scalability. For this reason, it is often not even considered in many modern heuristic design studies. Yet, from RS to more complex search heuristics such as EAs, is still quite a leap. In the gap for instance lie almost equally simple and classical, yet far less random in nature, Local Search (LS) techniques. Still LS has, to the best of our knowledge, not been applied to NAS before, except for in [32], a work that happened independently and concurrently to ours (see Sec. 2).

In this paper, we consider LS for NAS and in doing so, we make three contributions: (1) We propose a simple and parameter-less LS algorithm for multi--objective (accuracy vs. architecture complexity) NAS. (2) To enable quick and easy benchmarking, we release[4] two datasets containing cached performances for over 200,000 Convolutional Neural Network (CNN) architectures for CIFAR-10 and CIFAR-100. Our datasets, named MacroNAS-C10 and MacroNAS-C100, are designed to be complementary to similar existing proposals [8, 36]. We remark that we designed these datasets to be able to compare NAS approaches, and not necessarily to obtain State-of-the-Art (SotA) neural networks. (3) We provide evidence on the validity of LS being a superior baseline for NAS by conducting several experiments. In particular, we consider NAS in different search spaces; We compare LS with RS as well as with a classic and a SotA Evolutionary Algorithm (EA); We include an experiment on CIFAR-100 where the possible architectures are less constrained, leading to a large search space containing over 104 Billion possible neural architectures.

In this work, we consider NAS in a multi-objective setting. Single-objective NAS, i.e., searching only for good accuracy-networks, has already been amply investigated in [25,35], where it was found that very different search algorithms, including RS, actually perform very similar. Nevertheless, additional experiments that show the performance of LS for single-objective NAS are included in the Appendix and provide similar insights to those found in [32]. Multi-objective NAS on the other hand has been studied less (see Sec. 2), and is arguably a more interesting setting, where sophisticated search algorithms may potentially

[3] `https://www.automl.org/automl/literature-on-neural-architecture-search/`

[4] Benchmark datasets: `https://github.com/ArkadiyD/MacroNASBenchmark`
Source code for reproducibility: `https://github.com/tdenottelander/MacroNAS`

be needed. Alongside RS, we consider EAs, since they are a natural fit for multi-objective optimization, and are often used for NAS.

2 Related work

NAS can be broadly categorized by the level at which the search is performed, i.e., macro-level and micro-level [10] (different nomenclatures exist [33]). Macro-level search aims at composing (wiring) atomic *cells* of different type. A cell is a computational graph (a sub-net) composed of different operations/layers (convolutions, activation functions, etc.). In micro-level search, the position of cell types within the architecture is pre-decided (often a same cell type is repeated in multiple places), while it is the cells' internal wiring that is optimized. Both levels can also be searched at the same time [17,19], potentially along with hyper-parameter settings [19,34]. In the remainder of the paper, we will refer to "architecture" and "network" (for brevity, "net") interchangeably.

Search algorithms considered for NAS vary wildly. Reinforcement Learning (RL), Bayesian optimization approaches, and EAs are among the most popular [10]; the latter being typically employed in multi-objective scenarios. In this paper, we mostly focus on performing multi-objective NAS at macro-level. Therefore, in the following we report on recent works in this direction, and focus on whether comparisons against a baseline (i.e. RS) were made.

The work in both [6, 15] builds upon NSGA-II [7] to obtain a front of nets that trade off accuracy and compactness. Although nets with SotA performance are ultimately obtained, the search algorithm is not compared with any baseline. In [9], a multi-objective EA is proposed that includes Lamarckian mechanisms: instead of training nets from scratch, child nets inherit the weights of their parents and are only fine-tuned. Although the method is shown to outperform RS, it is unclear whether this is due to a difference with the way the EA searches, or solely because of weight inheritance. In [17], NSGA-Net is presented, which evolves fixed-length architectures at both macro- and micro-level. The authors show, through experimental analysis, that NSGA-Net is superior to RS. Summarizing, most of the literature work considers only RS as a baseline (sometimes in not entirely equal conditions), or no baseline at all.

Importantly, the only work we are aware of that investigated the use of an LS algorithm is [32], which appeared on arXiv shortly after the first version of our paper appeared on the same platform. The work shares strong similarities with ours, as it investigates the use of a simple LS algorithm for NAS, and reaches the same conclusion, i.e., that LS is a strong baseline for NAS. Despite these similarities, our works can be considered complementary to each other: the authors of [32] focus on single-objective NAS benchmarks and provide interesting theoretical underpinnings; we instead focus on multi-objective NAS, and, among other contributions, propose new NAS benchmark datasets.

Since we release benchmark datasets of saved net evaluations for macro-level NAS, we also refer to important related works of this nature. NAS-Bench-101 [36] contains cached performance indicators for 423,000 nets on CIFAR-10,

while NAS-Bench-201 [8] does the same also for CIFAR-100 and ImageNet, but for far less architectures (15,625). Both datasets are built for micro-level NAS. We did not consider NAS-Bench-201 because the limited number of possible architectures can limit the analysis of the search behavior for long run-times. We built new benchmark datasets for three reasons. First, we intended to provide the community with a macro-level search benchmark dataset for better reproducible NAS: we are not aware of other works proposing this. Second, NAS-Bench-101 includes infeasible solutions (the MacroNAS datasets do not), and choosing how to handle them substantially changes the search behavior of an algorithm. Lastly, for NAS-Bench-101 it has already been shown that RS is a competitive baseline in a single-objective setting [36]. We will show however that, in fact, in the search space of NAS-Bench-101 it is hard to do any better than RS, even when considering a multi-objective setting (see Sec. 5.1).

3 Objectives, Encodings, Search algorithms

We now present the objectives to optimize, the encoding schemes we use, and the algorithms we consider in the experimental analysis. For a detailed description of the MacroNAS datasets we introduce, we refer the reader to the Appendix.

3.1 Objectives

The first objective f_1 is to maximize accuracy (fraction of correct net predictions). Particularly, we set f_1 to the validation accuracy (acc_{val}), i.e., the accuracy of the net for a set of data that is not used for training the net, to assess whether the net generalizes. Because the validation set is still used in the NAS optimization loop, we consider the ability of nets with good acc_{val} to generalize to a second set of completely held-out data, i.e., the test set, in Sec. 5.4.

The second objective f_2 is to maximize a net's efficiency, i.e., to minimize the Mega Multiply–ACcumulate operations (MMACs, 1 MMAC $\simeq$ 2 FLOPs), which is the number of GPU computations a net requires to make a prediction. MMACs is a popular complexity metric [28] and is deterministic (in contrast to actual time taken). Because large differences in scale of the objectives can influence search mechanisms (e.g., the crowding distance [7]), we set MMACs to be in the same range as acc_{val}, i.e., we normalize MMACs to lie in $[0, 1]$. The min/max MMAC values for the normalization are trivial to determine as it suffices to consider the smallest/largest achievable nets. For the sake of a consistent optimization direction in both objectives, the secondary objective f_2 becomes maximizing $1 - \mathit{normalize}(\text{MMACs})$. Since no MMACs were reported for NAS-Bench-101, for the experiment in Sec. 5.1 we use $1 - \mathit{normalize}(\#\mathit{parameters})$.

3.2 Encodings

We use a direct encoding that represents feed-forward CNNs. The encoding is an array of discrete variables $[x_1, \ldots, x_\ell]$, where each x_i takes values (cell types) in Ω_i, a position-dependent alphabet.

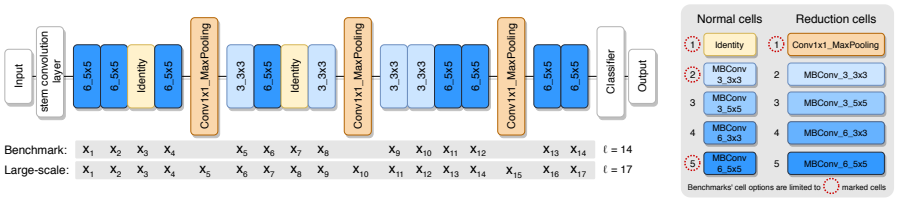


Fig. 1: Best acc_{val} net on CIFAR-10 (left) and cell types (right). For MacroNAS, 3 types are possible for normal cells (types 1, 2 and 5) and reduction cells (positions 5, 10, and 15) are fixed to type 1. For the large-scale experiment, 5 types are available for all cells. For details on cell types see the Appendix.

In the MacroNAS datasets, the number of searchable cells (length of the encoding) is $\ell = 14$. In particular, only *normal* cells, i.e., cells that maintain the height and width of the input they receive [37], can be optimized. Conversely, cells that reduce spatial dimensions (*reduction* cells) are fixed. The alphabet Ω_i is identical for each position i and contains 3 options. One option is the *identity*, i.e., a placeholder cell that forwards information as is. The other two options are commonly used convolutional blocks from literature (described in the Appendix). Fig. 1 shows an example of an encoding. Even though the encoding has a fixed length, it can represent variable-length architectures by means of identity cells. Note that the mapping from encoding to (computationally unique) architecture is redundant. The size of the search space (total number of encodings) is $3^{14} = 4{,}782{,}969$, which maps to 208,537 architectures. The latter number is sufficiently small to allow us to evaluate all architectures and save their performances to create the benchmark datasets.

For the large-scale experiment, we increase the number of variables ℓ to 17 by including the optimization of reduction cells, and also increasing the alphabet size $|\Omega_i|$ to 5, $\forall i$. Here, the position i influences what Ω_i is used, as options for normal cells are different from options for reduction cells (see Fig. 1). The total search space of $5^{17} = 762{,}939{,}453{,}125$ maps to 104,086,030,125 architectures.

The encoding for NAS-Bench-101 [36] (micro-level) is different. Here, both the type of operations in a (repeated) cell, and their connections, can be optimized. The encoding is a binary string of length 21 that represents connections, concatenated with a ternary string of length 5 that represents operations.

3.3 Search algorithms

The first algorithm we consider is NSGA-II [7]. It is by far the most popular multi-objective EA in general. We use most commonly adopted parameter settings of NSGA-II: 2-point crossover, single-variable mutation with probability $p_m = 1/\ell$, tournament size 2 and population size $n_{\text{pop}} = 100$.

Fundamentally better algorithmic design is foundational to EA research. This means building better problem-specific EAs or ones capable of doing better at large classes of interesting problems. In the latter direction lie linkage learning EAs that attempt to automatically detect and exploit building blocks of non-trivial sizes [20, 30]. To the best of our knowledge, such algorithms have not yet been tried for NAS. Studying if they have merit here as well is interesting.

Thus, we consider the Multi-Objective Gene-pool Optimal Mixing Evolutionary Algorithm (MO-GOMEA). MO-GOMEA mostly differs from a classic genetic algorithm in that every generation it computes a model of *linkage*, i.e. the estimated strength of interdependency between variables, and uses this model to propagate potential building blocks during variation [30]. If the NAS search space exhibits any linkage, this algorithm can likely exploit it. Also, MO-GOMEA uses clusters to partition the population in objective space, so it is generally able to improve upon specific parts of the front by recombining within niches. We use one of the latest implementations for discrete optimization [18], which includes the Interleaved Multi-start Scheme (IMS). The IMS allows to avoid the manual setting of population size parameter by evolving populations of increasing sizes in interleaved fashion. For the initial population size, we use $n_{\text{pop}} = 8$ (default).

We further consider RS as a standard baseline. RS generates new nets repeatedly by sampling each cell uniformly at random from the alphabets Ω_i.

Finally, we consider a simple random restart LS algorithm:

1. Initialize a random architecture (as in RS);
2. Sample a scalarization coefficient $\alpha \sim \mathcal{U}(0, 1)$;
3. Consider all variables in random order, as follows:
   3.1. For each variable x_i, evaluate the net obtained by setting x_i to each option in Ω_i. Keep the best according to $\alpha \times f_1 + (1 - \alpha) \times f_2$;
4. Repeat step 1 until the computation budget (evaluations / time) is exhausted.

In other words, we perform one round of first-improvement LS at the level of variables, but best-improvement LS at the level of options for one variable. Fig. 2 illustrates one iteration of our LS algorithm (from now on, we simply refer to it as LS).

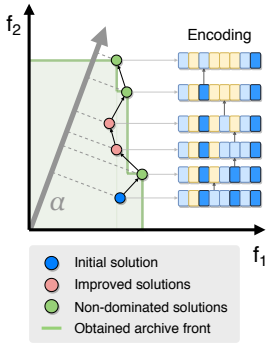


Fig. 2: Example iteration on one random scalarization coefficient α in LS. Only solutions that improve the scalarized objective are shown.

4 Experimental setup

All search algorithms store an archive $\mathcal{A}$ of best-found nets according to (strict) Pareto domination. Formally, it is: $\boldsymbol{x} \succ \boldsymbol{y}$ ("$\boldsymbol{x}$ dominates $\boldsymbol{y}$") iff $f_i(\boldsymbol{x}) \geq f_i(\boldsymbol{y}) \wedge \boldsymbol{f}(\boldsymbol{x}) \neq \boldsymbol{f}(\boldsymbol{y})$, for each objective f_i. The archive thus satisfies $\nexists \boldsymbol{x}, \boldsymbol{y} \in \mathcal{A} : \boldsymbol{x} \succ \boldsymbol{y}$. We use no size limitation for the archive, and update the archive each time a new net is discovered (the time cost is insignificant compared to evaluating a net's performance, see below).

To compare algorithms, we consider the improvement of the hypervolume enclosed by $\mathcal{A}$, using the origin as reference point. For the experiments of Secs. 5.1 and 5.2 (where the Pareto front is known), we also investigated the inverse generational distance [4], and obtained very similar results (omitted for brevity). We only count the evaluations of unique architectures as evaluating a net is

typically costly (minutes to hours). Caching is used to not re-evaluate a same architecture twice.

We handle the net with *identity* cells for all variables differently from the others. This trivial net can be considered fundamentally uninteresting to search for, but it still influences multi-objective metrics. We therefore set NSGA-II and MO-GOMEA to include one instance of it in the population at initialization. Similarly, we include it in the archive of RS and LS from the beginning.

For experiments concerning the benchmark datasets, i.e., where performances for different architectures are pre-stored, we perform 30 runs for each algorithm. For the large-scale search space, where pre-evaluating all nets is simply unfeasible and a single evaluation takes on average 6.5 minutes on Nvidia RTX 2080 Ti GPU, we perform 6 runs for each algorithm. To assess whether differences between algorithms are significant, we adopt pairwise non-parametric Mann-Whitney-U tests with Bonferroni correction.

5 Results

First, we compare the algorithms on NAS-Bench-101 and on MacroNAS-C10/C100. Next, we analyze the performance on MacroNAS-C100, and on the large-scale version for this task, in more detail. Lastly, we discuss what the observed differences mean when considering the nets' ability to generalize.

5.1 Preliminary experiments

We begin by analyzing the results for NAS-Bench-101 (Fig. 3 left). For this benchmark, differences among the algorithms are negligible. This can also be seen in Fig. 4, where fronts obtained by the algorithms are similar at different points in time. These results are in line with the ones reported for the single-objective experiments conducted in [36]: differences among fundamentally different algorithms are relatively small.

Regarding the macro-level search spaces, i.e., on MacroNAS-C10 and MacroNAS-C100 (Fig. 3 middle and right, respectively), a notable difference is found

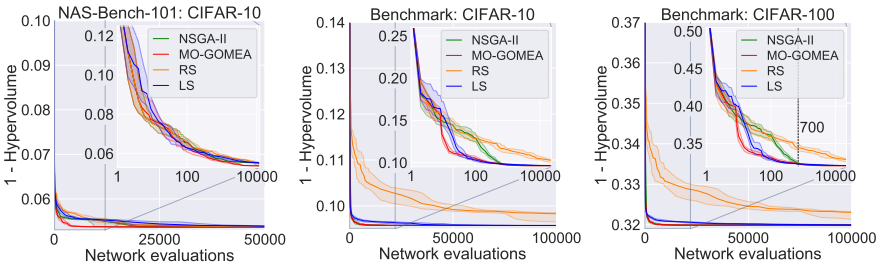


Fig. 3: Convergence graphs on NAS-Bench-101 (left), MacroNAS-C10 (middle) and MacroNAS-C100 (right). Medians (solid lines) and 25/75th percentiles (bands) of 30 runs are shown. Note that only the horizontal axes are the same, and are in logarithmic scale in the zoomed-in views (insets).

between RS and the other algorithms. There are however also small differences between the EAs and LS. In the zoomed-in views it can be seen that, for the first 100 evaluations, NSGA-II performs as good as RS: this is because the initial population ($n_{pop} = 100$) is evaluated. Meanwhile, MO-GOMEA already performed optimization on the small initial populations instantiated by its IMS.

The hypervolume obtained for CIFAR-100 is not as good as the one obtained for CIFAR-10. This is not surprising because CIFAR-100 is considered a harder classification task. On the MacroNAS benchmarks, the best acc_{val} for CIFAR-10 is 0.925, the one for CIFAR-100 is 0.705. Furthermore, note that the fact that the hypervolumes for NAS-Bench-101 reach better values, is mainly because f_2 is different here (based on number of net parameters instead of MMACs). Still, NAS-Bench-101 contains nets with slightly larger acc_{val} (yet easily found by all algorithms), up to 0.951. Since CIFAR-100 is the harder task, we focus on this task in the next sections.

Fig. 4: Evolving archives for one example run on NAS-Bench-101.

5.2 MacroNAS-C100

Fig. 3 (right) shows that, particularly for the EAs, the hypervolume increases rather quickly in the beginning, and not notably afterwards. Covering the entire Pareto front is however hard (e.g., only 21 out of 30 runs of MO-GOMEA cover it within 100,000 evals). Overall, MO-GOMEA performs best, followed by LS until $\sim$ 700 evaluations. At this point, differences between MO-GOMEA, NSGA-II and LS are not significant at the 99% confidence level.

We remark that what these differences truly mean is discussed in terms of generalization in Sec. 5.4. RS is clearly the inferior approach.

To obtain a better understanding of the search the algorithms perform, we present the fronts discovered for CIFAR-100 at different time steps in Fig. 6 and Fig. 11 (left) (we found very similar results for MacroNAS-C10). The plots for 10 and 100 evaluations confirm the fact that MO-GOMEA and LS are the quickest at approaching the front. Note that the archive at 10 evaluations (Fig. 6 left) of LS is already obtained during the very first iteration of LS.

Compared to MO-GOMEA, LS is especially quick at finding the less-densely populated areas in the objective space (low MMACs). This is likely because making networks with less MMACs is straightforward, as more layers can be set to identity, which is effectively searched by the LS (with the proper objective weighting) that tries every option in each cell. Moreover, the EAs need

to rely on the encodings present in the initial population, where an identity cell is sampled in position i only with chance $1/|\Omega_i|$. As shown in Fig. 5, evaluating small architectures is necessary to cover the part of the Pareto front containing efficient nets, as the number of nets there is smaller.

It is clear that RS is inferior to the other algorithms, as it struggles to find efficient nets that occur only sparsely in the search space. This issue is due to the fact that RS does not build up from efficient nets and thus solely relies on the way it samples: sampling a net with many identity cell is rare. For example, the probability of sampling the full-identity net is $1/|\Omega_i|^\ell$ (recall that this net is pre-inserted in the archive because uninteresting).

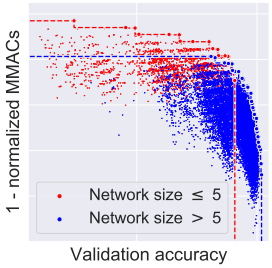


Fig. 5: Distribution of two classes of nets and their Pareto fronts.

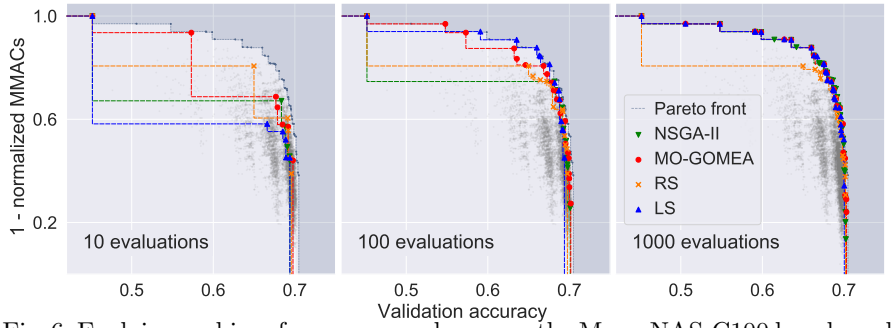


Fig. 6: Evolving archives for one example run on the MacroNAS-C100 benchmark dataset. Gray points represent all possible architectures.

5.3 Large-scale NAS: CIFAR-100

Fig. 10 (right; consider only acc_{val}, i.e., solid lines) shows how the hypervolume improves over time for the large-scale experiment on CIFAR-100. MO-GOMEA and LS obtain a good hypervolume similarly quickly, as observed for MacroNAS-C100. NSGA-II converges more slowly than MO-GOMEA and LS, but it is capable of outperforming them in terms of hypervolume towards the end. Nonetheless, both LS and MO-GOMEA do appear to improve again near the end of our total evaluation budget.

Fig. 8 shows the way the algorithms discover solutions over time, and Fig. 9 shows the density of architectures obtained at 2500 evaluations. Notably, MO-GOMEA discovers nets with the largest acc_{val} (bottom-right points, consistent for all 6 runs). This may be because of two reasons. First, MO-GOMEA exploits linkage, which may be useful to recombine complex building blocks and obtain better performing nets. Second, MO-GOMEA restricts recombination to solutions that are similar.

Regarding RS, from Figs. 7, 9 it can be seen that RS mostly samples complex nets. Yet, as RS lacks any mechanism to refine and exploit solutions, it cannot

match MO-GOMEA's capability of discovering the most accurate nets. The main issue of RS is, as aforementioned, that it samples efficient nets too rarely.

NSGA-II exhibits a very interesting behavior: it searches progressively from more complex to simpler architectures (Figs. 8, 7). This happens because in the beginning it is more likely to generate complex nets (as RS does), while later on NSGA-II's crowding distance operator steers the search towards the areas where points are most scattered, i.e., where efficient nets are located. Moreover, it is likely that its non-linkage informed operators are less likely to generate the best performing nets.

LS is found to be slightly inferior to the EAs in terms of final hypervolume (Fig. 10 right, solid lines), and it does not ultimately obtain the best front (Figs. 8, and 11 middle-right). Having only 6 repetitions, statistical testing results in no significant differences for the hypervolume at the end of the runs (i.e., at 2500 evaluations). What we believe is most important is that LS is remarkably quick at obtaining fairly good nets (Fig. 8), with rather uniform spread in terms of trade-offs (Fig. 7). Moreover, LS has no issues similar to RS, in fact, it outperforms it markedly.

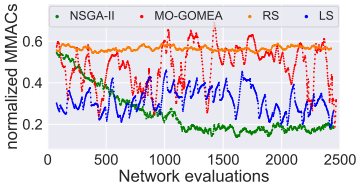


Fig. 7: MMACs of evaluated nets throughout one example run, smoothed for readability (moving avg. filter of size 75).

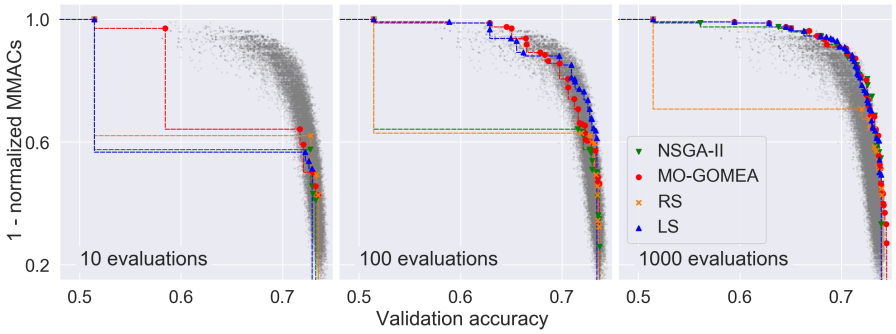


Fig. 8: Evolving archives for one example run of the large-scale NAS for CIFAR-100. Gray points represent all architectures ever discovered.

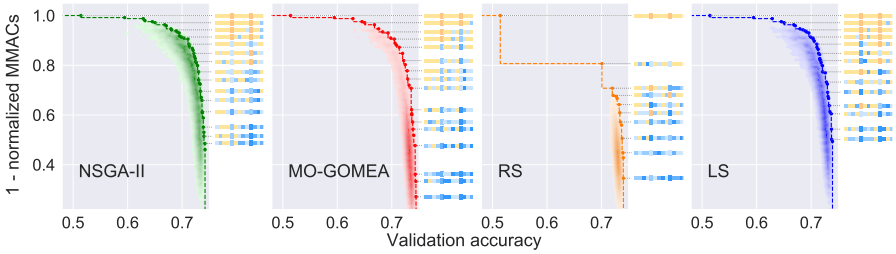


Fig. 9: Fronts obtained at 2500 evaluations by the search algorithms in one example run on the large-scale experiment on CIFAR-100, with a visual description of the nets. The colored cells correspond to the ones displayed in Figure 1. Heatmaps display the distribution of nets evaluated by the algorithms.

5.4 Generalization

Because the validation is used during the search, a risk exists to implicitly start overfitting to it [2]. Hence, a second held-out set is used, i.e., given (a front of) best-acc_{val} nets, their acc_{test} is measured.

As can be seen from Figs. 10 (left) and 11 (left-most two), on MacroNAS-C100 RS clearly remains the worst performing algorithm when considering generalization to the test set. However, MO-GOMEA and NSGA-II lose the lead to LS, between 100 and 18,000 evaluations. This is a consequence of the aforementioned phenomenon: despite the fact that nets are not trained on the set of data on which acc_{val} is measured, acc_{val} is used to select what nets are best, hence the EAs' better search translates to more overfitting to this set. LS is slightly less powerful at searching (worse hypervolume in terms of acc_{val}), yet the nets generalize well to the test set. Crucially, there are no significant differences (at 99% confidence) among LS and the EAs at 100,000 evaluations.

The findings for the enlarged search space reflect the ones found for MacroNAS-C100: the difference in performance between LS and the EAs becomes smaller when considering acc_{test} (see Fig. 10 right). Fig. 11 (right-most two) displays this especially well, as the most accurate nets discovered by MO-GOMEA turn out to be less accurate when tested again. Again, no statistical differences are observed at the end of the runs between LS and the EAs.

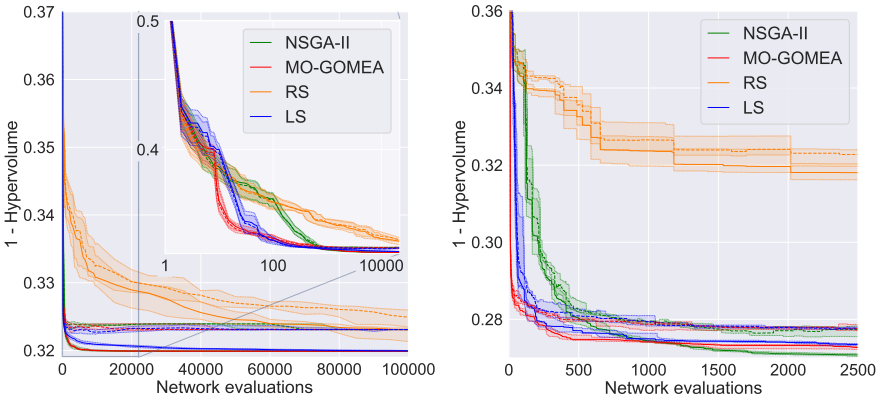


Fig. 10: Convergence graphs regarding acc_{val} (solid lines) and acc_{test} (dashed lines) on MacroNAS-C100 (left) and on CIFAR-100 on the enlarged search space (right). Medians (lines) and 25/75th percentiles (bands) of respectively 30 and 6 runs are shown for all algorithms. Note the different ranges for the vertical axes and the logarithmic scale for the horizontal axis in the zoomed-in view.

6 Discussion

While several works compare algorithms in terms of time, we preferred a comparison in terms of number of evaluations. Using a time budget is reasonable

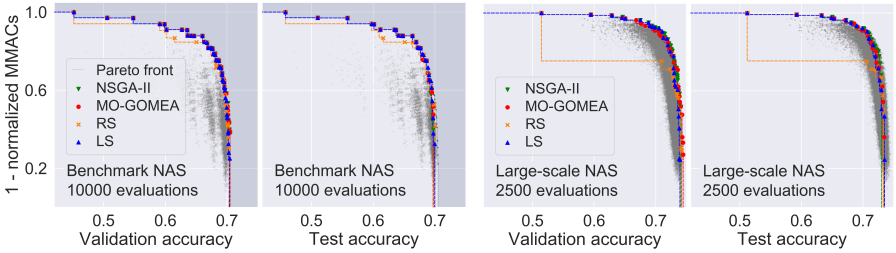


Fig. 11: Validation and test archives for 10,000 evaluations on MacroNAS-C100 (resp. first and second from the left) and 2500 evaluations on large-scale NAS (resp. second and first from the right). Archives are from one example run.

for practical applications, but has its limitations: particularly, time-based experiments are influenced by hyper-parameter choices (e.g., network training budgets), and can put some algorithms at disadvantage (e.g., see [22] for RS).

In our macro-level comparisons, we systematically found that RS performed much worse than the other algorithms. This boiled down to the fact that RS has a very low probability of sampling efficient nets. This simple result is important when considering RS as a baseline to benchmark against: if the sampling process of RS is not well-aligned with the desired objectives, RS can hardly beat an algorithm that includes optimization, even if as simple as LS. Note that on NAS-Bench-101, where the encoding and sampling process is such that small nets are not extremely rare, RS performed similar to the other algorithms. Carefully setting RS to sample solutions in a way that is well-aligned with the objectives may be non-trivial, and in fact harder than using LS.

We showed that the fact that different EAs include different search mechanisms, such as linkage learning and cluster-restricted mating for MO-GOMEA, and non-dominated sorting and crowding distance for NSGA-II, does lead to notable differences in what nets are found at what point in time. This is important to consider in practical applications where the evaluation budget is limited. LS was found to be competitive in this respect, as it quickly obtained a rather uniform scattering of nets with different trade-offs. Moreover, LS proved capable of ultimately competing very well when generalization is accounted for: the capability of SotA EAs to optimize at a very refined level with respect to acc_{val} does not always translate to better acc_{test} due to implicit overfitting.

7 Conclusion

With several experiments, we have shown that a simple, parameter-less local search algorithm can be a competitive baseline for NAS. We found that the proposed local search algorithm competes with state-of-the-art EAs, even up to thousands of evaluated architectures, in a multi-objective setting. By a greedy single-variable search mechanism and a random scalarization of the objectives, LS discovers nets with rather uniform trade-offs between accuracy and complexity in few evaluations. Performance gaps with EAs evident at search time are often lost when generalization is assessed. Importantly, we found that lo-

cal search outperforms random search consistently and markedly. We therefore recommend to adopt local search as a baseline for NAS.

Acknowledgments

This work is part of the research programme Commit2Data with (project #628.011.012), which is financed by the Dutch Research Council (NWO), and part of a project (#15198) that is included in the research program Technology for Oncology, which is financed by the Netherlands Organization for Scientific Research (NWO), the Dutch Cancer Society (KWF), the TKI Life Sciences & Health, Asolutions, Brocacef, and Cancer Health Coach.

References

1. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 550–559. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. springer (2006)
3. Bosman, P.A.N., Luong, N.H., Thierens, D.: Expanding from discrete Cartesian to permutation gene-pool optimal mixing evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 637–644 (2016)
4. Bosman, P.A.: The anticipated mean shift and cluster registration in mixture-based edas for multi-objective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 351–358. GECCO '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1830483.1830549, https://doi.org/10.1145/1830483.1830549
5. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. arXiv preprint arXiv:1812.00332 (2018)
6. Chu, X., Zhang, B., Xu, R., Ma, H.: Multi-objective reinforced evolution in mobile neural architecture search. arXiv preprint arXiv:1901.01074 (2019)
7. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (April 2002)
8. Dong, X., Yang, Y.: NAS-Bench-201: Extending the scope of reproducible neural architecture search. arXiv preprint arXiv:2001.00326 (2020)
9. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via lamarckian evolution. arXiv preprint arXiv:1804.09081 (2018)
10. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. Journal of Machine Learning Research **20**(55), 1–21 (2019)
11. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. arXiv preprint arXiv:1904.00420 (2019)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)

13. Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., Wilson, A.G.: Averaging weights leads to wider optima and better generalization. arXiv preprint arXiv:1803.05407 (2018)
14. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture. pp. 1–12 (2017)
15. Kim, Y.H., Reddy, B., Yun, S., Seo, C.: Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In: JMLR: Workshop and Conference Proceedings. vol. 1, pp. 1–8 (2017)
16. Kitchin, R.: The data revolution: Big data, open data, data infrastructures and their consequences. Sage (2014)
17. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: NSGA-Net: Neural architecture search using multi-objective genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 419–427. GECCO 2019, Association for Computing Machinery, New York, NY, USA (2019)
18. Luong, N.H., Poutré, H.L., Bosman, P.A.N.: Multi-objective gene-pool optimal mixing evolutionary algorithm with the interleaved multi-start scheme. Swarm and Evolutionary Computation **40**, 238 – 254 (2018)
19. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al.: Evolving deep neural networks. In: Artificial Intelligence in the Age of Neural Networks and Brain Computing, pp. 293–312. Elsevier (2019)
20. Pelikan, M., Sastry, K., Cantú-Paz, E.: Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications (Studies in Computational Intelligence). Springer-Verlag, Berlin, Heidelberg (2006)
21. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. OpenAI Blog **1**(8), 9 (2019)
22. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. p. 2902–2911. ICML 2017, JMLR.org (2017)
23. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision **115**(3), 211–252 (2015)
24. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018)
25. Sciuto, C., Yu, K., Jaggi, M., Musat, C., Salzmann, M.: Evaluating the search phase of neural architecture search. arXiv preprint arXiv:1902.08142 (2019)
26. Senior, A.W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A.W., Bridgland, A., et al.: Improved protein structure prediction using potentials from deep learning. Nature pp. 1–5 (2020)
27. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of Go without human knowledge. Nature **550**(7676), 354–359 (2017)
28. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE **105**(12), 2295–2329 (2017)

29. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: MnasNet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)
30. Thierens, D., Bosman, P.A.N.: Optimal mixing evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 617–624. GECCO 2011, Association for Computing Machinery (2011)
31. Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., Silver, D.: AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/` (2019)
32. White, C., Nolen, S., Savani, Y.: Local search is state of the art for NAS benchmarks. arXiv preprint arXiv:2005.02960 (2020)
33. Wistuba, M., Rawat, A., Pedapati, T.: A survey on neural architecture search. arXiv preprint arXiv:1905.01392 (2019)
34. Wong, C., Houlsby, N., Lu, Y., Gesmundo, A.: Transfer learning with Neural AutoML. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 8356–8365. Curran Associates, Inc. (2018)
35. Yang, A., Esperança, P.M., Carlucci, F.M.: NAS evaluation is frustratingly hard. arXiv preprint arXiv:1912.12522 (2019)
36. Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., Hutter, F.: NAS-Bench-101: Towards reproducible neural architecture search. arXiv preprint arXiv:1902.09635 (2019)
37. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)

Appendix

In this section we describe in detail the procedure we used to create the benchmark datasets (for CIFAR-10 and CIFAR-100), and the changes introduced for the large-scale NAS experiment (of Sec. 5.3).

Design of the MacroNAS benchmark dataset

We consider feed-forward CNNs, consisting of 17 sequentially connected cells. Similarly to [36], each net includes two auxiliary components: the stem convolution prior to the first cell, and a classifier after the last cell, which maps the output of the last cell to a prediction score for each class. The stem convolution is implemented by a 3×3 convolutional layer which converts the input image to 32 feature maps. The classifier is implemented by a 1×1 convolutional layer, followed by global average pooling and the final linear layer. Fig. 1 shows an example of our NAS setting.

Similarly to [36], we pre-set certain positions to ensure a feasible memory consumption for all nets. Specifically, cells at positions 5, 10 and 15 (*reduction* cells) reduce the spatial input dimensionality and increase the number of channels: for an input of size $D \times H \times W$ (D is number of feature maps, H and W are the spatial dimensions) they return an output of size $2D \times H/2 \times W/2$. This is implemented by means of a 1×1 convolutional layer, followed by max-pooling. The remaining 14 positions are subjected to search with cells that preserve spatial dimensionality (i.e., *normal* cells).

As options for the searchable cells, we use differently parameterized inverted bottleneck convolutional blocks (MBConv) [24], which are widely used in NAS and known for their efficiency [5,11,29]. We allow a total of 3 cell types: MBConv with expansion factor 3 and kernel size 3; MBConv with expansion factor 6 and kernel size 5; and the *identity* cell. The latter option performs no operation and effectively acts as a placeholder to represent smaller nets, i.e., nets with fewer cells. If two nets are identical except for the position of *identity* cells in-between reduction cells (i.e. from positions 1 to 4, 6 to 9, and 11 to 14), they are computationally equivalent (an example is shown in Fig. 12).

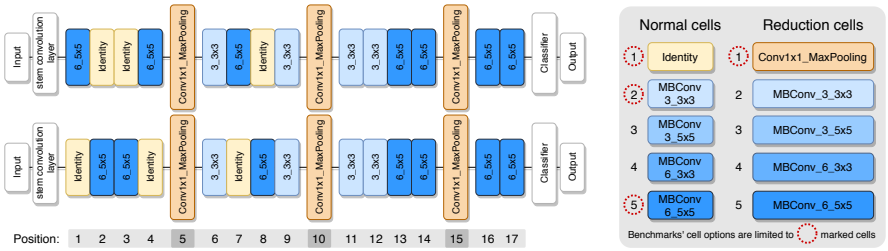


Fig. 12: Two computationally equivalent networks (left) and legend of cell types (right)

To evaluate all nets in a feasible time, we relied on an approach that approximates training from scratch: the one-shot NAS [1, 11]. The approach trains a

supernet, i.e., a massive net that contains all possible nets of the search space as sub-nets. We follow the training procedure described in [11]: for each batch of training samples, one path, i.e., a sequence of cells that represents one of the possible architectures (see Fig. 13), is sampled from the supernet, and weights in the cells that belong to that path are trained. We use standard training settings: cross-entropy loss and stochastic gradient descent. We use stochastic weight averaging (SWA) [13] for robustness. For both CIFAR-10/100, we use a stratified data split as done in [36]: 40K samples for training, 10K for validation, 10K for testing. Further details on the training process are summarized in Table 1.

To evaluate a single architecture, we follow the approach also described in [11]: 1) Take the weights from the trained supernet corresponding to the architecture evaluated; 2) Recalculate parameters of all batch normalization layers using a subset of the training data (we use a fixed random stratified subset of 10K samples); 3) Calculate validation and test accuracies.

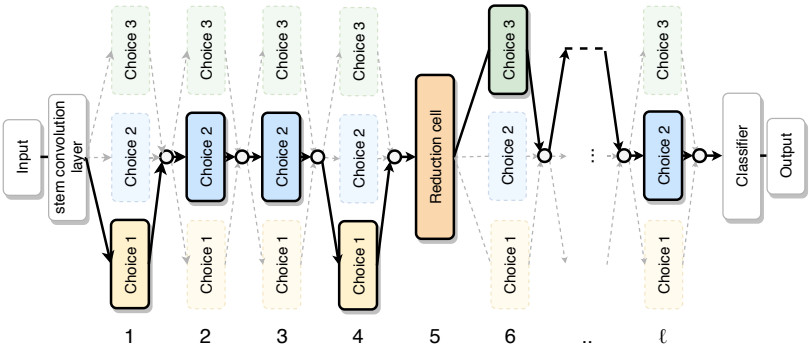


Fig. 13: Graphical description of the *supernet* with a randomly sampled path highlighted. In our case, the choices correspond to the cells of Fig. 12.

Large-scale experiment

In the large-scale experiment, we make both *normal* and *reduction* cells searchable. The options for *normal* cells are now 5, and consist of MBConv blocks with all four possible combinations of the aforementioned expansion factors and kernels sizes ($\{3, 6\} \times \{3, 5\}$), and the identity cell. We now further allow for the search of *reduction* cells, again with 5 options. These options are the same MBConv blocks but with stride 2 and number of channels multiplication, and the same simple *reduction cell* as used in our benchmarks. Recall that the positions of reduction cells are fixed (at indices 5, 10 and 15).

For this experiment we adopt the one-shot NAS approach again to speed-up the search, but include a refinement step, to better approximate the accuracy that nets would achieve if trained from scratch [1]. In particular, to evaluate an architecture, we finetune the weights obtained from the supernet for 20 epochs before calculating validation and test accuracies. Further hyperparameter settings are summarized in Table 1.

Setup	**Benchmarks creation**	**Large-scale experiment**	
Hyperparameter	**Supernet training**	**Supernet training**	**Single net finetuning**
Optimizer	SGD with momentum 0.9, weight decay 10^{-4}	SGD with momentum 0.9, weight decay 10^{-4}	SGD with momentum 0.9, weight decay 10^{-4}
Initial learning rate	0.1	0.1	0.01
Learning rate schedule	Cyclic cosine annealing	Cyclic cosine annealing	Cosine annealing
Using SWA	✓	✓	✗
Number of epochs	370	500	20
Batch size	128	128	256

Table 1: Training hyperparameters

Single-objective NAS experiments

To provide further evidence that our proposed LS algorithm can be considered a competitive baseline in a single-objective setting as well, its performance is evaluated again on NAS-Bench-101 (CIFAR-10) and on the proposed MacroNAS-C10 and MacroNAS-C100 benchmark datasets, by optimizing for validation accuracy only.

Experimental setup. For these experiments LS is even simpler than before (Sec. 3.3), as we are exclusively interested in maximizing validation accuracy and thus no longer need to sample the scalarization coefficient α (i.e., we fix $\alpha = 1$). We compare again to RS, the latest single-objective version of GOMEA [3], and a basic GA that uses uniform cross-over, tournament selection of size 2 upon joint parent–offspring pool, single-variable mutation with probability $p_m = 1/\ell$ and population size 100.

Similarly to Sec. 4, unique architectures are evaluated only once and their accuracy is cached and re-used during the search. Besides showing the convergence of each algorithm across the NAS search spaces, we also show their performance on achieving near-optimality. We do this because finding nets with optimal validation accuracy can potentially be a considerable but pointless effort: obtaining the global optimum can be much harder than finding adequate local optima, and performance can anyhow drop when a net generalization is assessed on test data (as confirmed in Sec. 5.4). Therefore, we particularly investigate what is the effort needed for the algorithms to reach:

$$acc^*_{val} - \epsilon \cdot (acc^*_{val} - \overline{acc_{val}}), \tag{1}$$

where acc^*_{val} is the optimal validation accuracy, ϵ is a small number (we consider $\epsilon \in [0, 0.1]$), and $\overline{acc_{val}}$ represents a 'bottom-line' accuracy, as it is the average accuracy obtained by randomly sampling 1,000 architectures. We particularly compute $(acc^*_{val} - \overline{acc_{val}})$ to re-scale the contribution of ϵ based on the acc_{val} range of the search space in exam. Chosen an ϵ, the more $\overline{acc_{val}}$ is large, i.e., the more is relatively easy to have good nets by random sampling, the more $\epsilon \cdot (acc^*_{val} - \overline{acc_{val}})$ will be small, i.e., Eq. 1 induces more proximity to the optimum.

Results. Fig. 14 shows convergence plots, where it can be seen that GOMEA performs best on all three search spaces, and our proposed LS algorithm is second-best. Both are consistently able to find the optimum. The GA performs similarly to RS on MacroNAS-C10 and MacroNAS-C100, and outperforms RS on NAS-Bench-101. We found mutation to be essential, as without it the GA can get stuck by converging to all-identical architectures, subsequently being unable to discover a never-seen architecture. We also found that increasing the population size by a factor of 5 still results in similar performance to RS. Ultimately the GA is not capable of competing with GOMEA and LS.

In Fig. 15, the gap in performance between GOMEA and the other algorithms becomes smaller when ϵ is larger. On these NAS search spaces, using a sophisticated search algorithm like GOMEA will result in finding the optimum (in terms of acc_{val}) faster and more reliably. However, finding a 'good' architecture instead of the optimal one is typically sufficient in NAS: the gains in acc_{val} can lack substantiality (see the vertical axis of Fig. 14), and generalization gaps (acc_{val} - acc_{test}) can be of relatively large magnitude (Sec. 5.4). All in all, our single-objective results confirm that simpler-to-implement algorithms like LS and RS remain strong competitors.

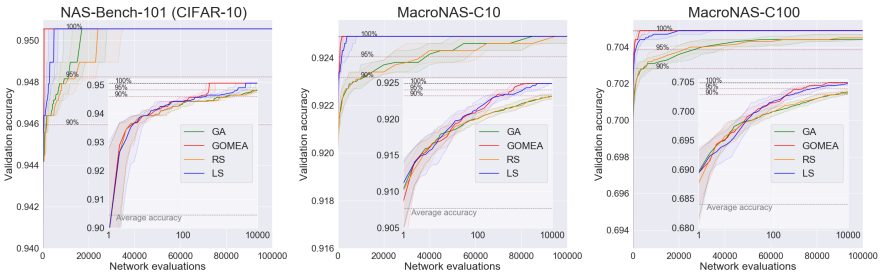


Fig. 14: Convergence on NAS-Bench-101 (left), MacroNAS-C10 (middle) and MacroNAS-C100 (right). Medians (solid lines) and 25/75th percentiles (bands) of 100 runs are shown. Horizontal dashed lines indicate $acc^*_{val} - \epsilon \cdot (acc^*_{val} - \overline{acc_{val}})$ for $\epsilon \in \{0\%,\ 5\%,\ 10\%\}$.

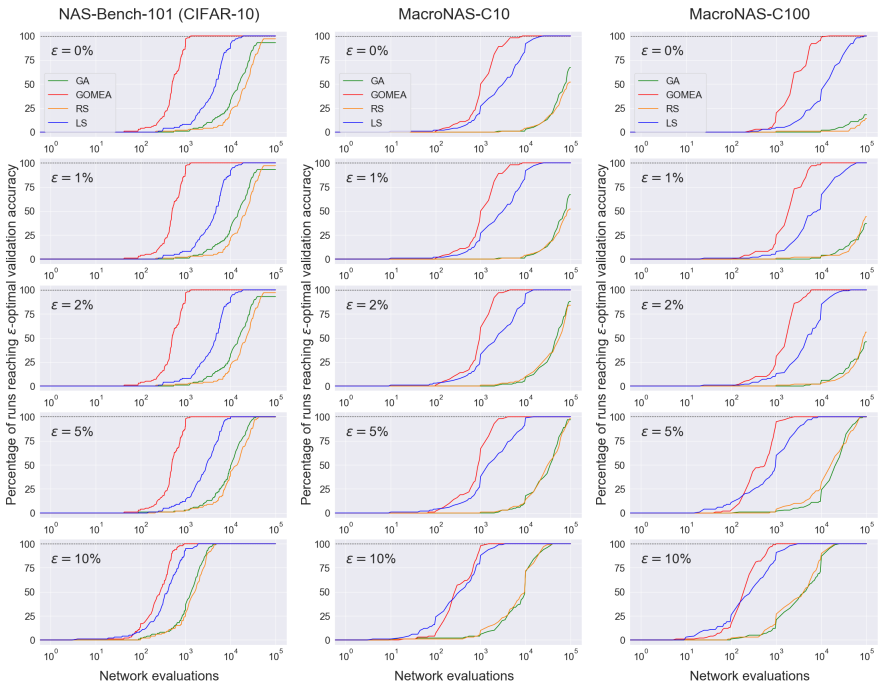


Fig. 15: Number of runs (out of 100) reaching $acc^*_{val} - \epsilon \cdot (acc^*_{val} - \overline{acc_{val}})$ against number of network evaluations on NAS-Bench-101 (left), MacroNAS-C10 (middle) and MacroNAS-C100 (right).

B

Analysis of the MacroNAS benchmark dataset

The creation of the benchmark is described in detail in the appendix of the paper in Appendix A. This section provides a brief (visual) analysis of MacroNAS-C10 and MacroNAS-C100.

Figure B.1 shows the distribution of architectures in the objective space. Figure B.2 displays the correlation of acc_{val} and acc_{test} of all networks. The distribution in terms of acc_{val} and MMACs of differently sized networks is shown in Figure B.3. Table B.1 provides general information about both benchmark datasets. Tables B.2 and B.3 provide a statistical view on the objective values and encoding properties present in the benchmark datasets.

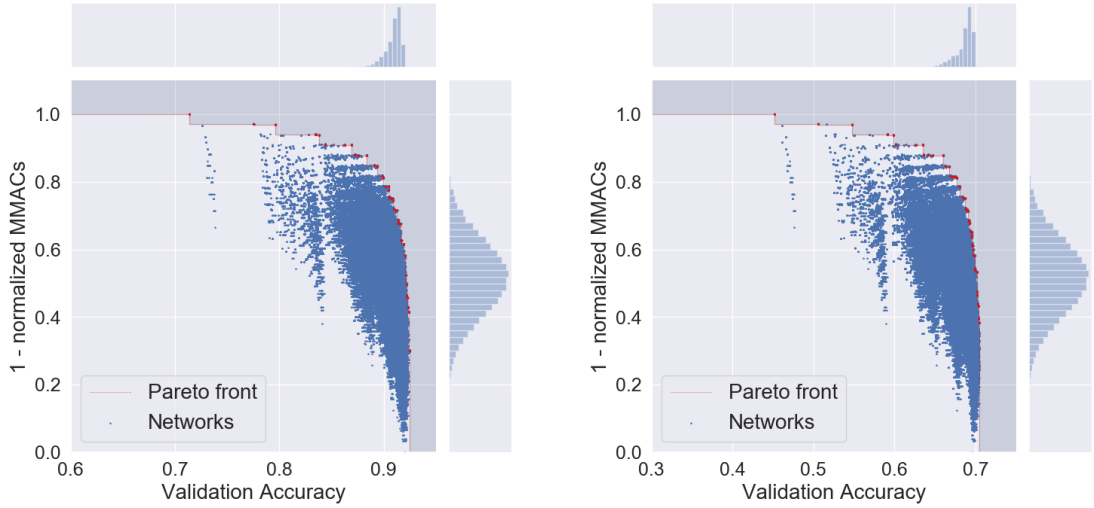


Figure B.1: Objective space and corresponding Pareto Front of MacroNAS-C10 (left) and MacroNAS-C100 (right). Note the different ranges on the horizontal axis.

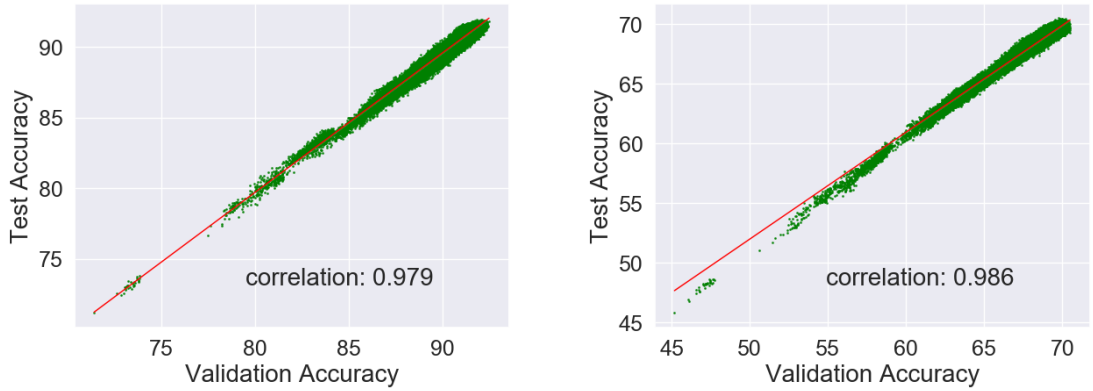


Figure B.2: Correlation between acc_{val} and acc_{test} in MacroNAS-C10 (left) and MacroNAS-C100 (right).

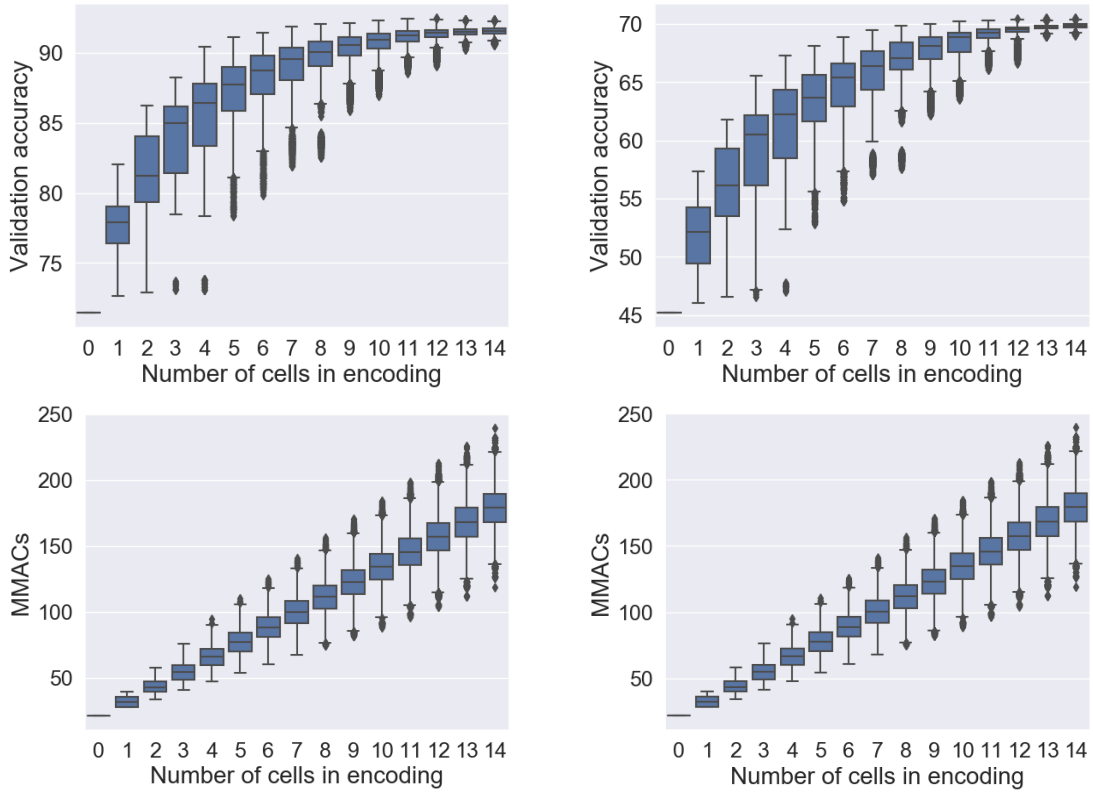


Figure B.3: Boxplots of acc_{val} (top) and MMACs (bottom) separated by number of cells in the encoding for MacroNAS-C10 (left) and MacroNAS-C100 (right).

	MacroNAS-C10	MacroNAS-C100
Size of search space	4,782,969	
Size of objective space (amount of different networks)	208,537	
Redundancy	95.64%	
Best acc_{val}	0.9249	0.7049
Best acc_{test}	0.9190	0.7051
Correlation[1] between acc_{val} and acc_{test}	0.979	0.986

Table B.1: Statistics of the MacroNAS benchmark datasets

	acc_{val}	acc_{test}	MMACs	norm(MMACs)	#parameters	cells in architecture
mean	90.796%	90.326%	126.576	0.483	1,951,519	9.33
std	1.038%	1.044%	24.018	0.110	515,000	1.76
min	71.41%	71.20%	21.31	0.0	547,146	0
25%	90.47%	89.98%	110.14	0.408	1,578,570	8
50%	91.09%	90.62%	126.37	0.482	1,951,082	9
75%	91.44%	90.99%	142.90	0.558	2,318,570	11
max	92.49%	91.9%	239.28	1.0	3,400,010	14

Table B.2: Statistics of objective values and encoding characteristics in MacroNAS-C10

	acc_{val}	acc_{test}	MMACs	norm(MMACs)	#parameters	cells in architecture
mean	68.423%	68.554%	126.807	0.483	2,066,809	9.33
std	1.491%	1.358%	24.018	0.110	515,000	1.76
min	45.18%	45.80%	21.54	0.0	662,436	0
25%	67.97%	68.22%	110.37	0.4075	1,693,860	8
50%	68.92%	69.01%	126.60	0.4820	2,066,372	9
75%	69.36%	69.38%	143.13	0.5578	2,433,860	11
max	70.49%	70.51%	239.51	1.0	3,515,300	14

Table B.3: Statistics of objective values and encoding characteristics in MacroNAS-C100

C

Additional experiments

This appendix contains additional results for (alternative) experiments in Sections 5.3.3 and 5.3.5.

C.1. Experiment 3: Effect of the number of clusters

Obtained fronts Figure C.1 shows the obtained fronts for MO-GOMEA when different numbers of clusters are used. There is hardly a pattern to be recognized, because the results change when runs with other seeds are displayed.

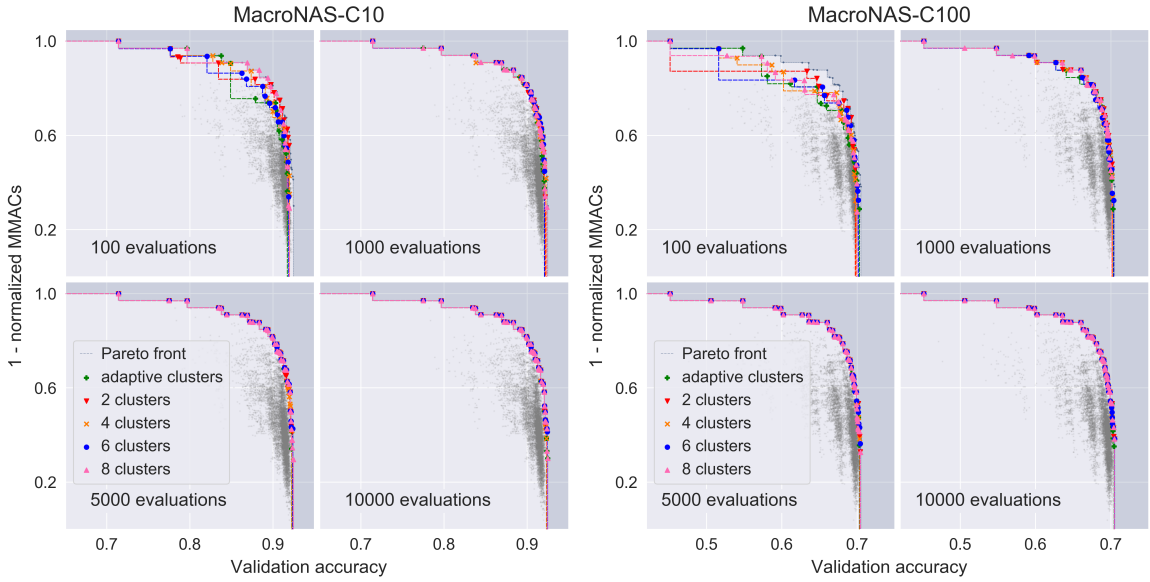


Figure C.1: Experiment 3: Achieved fronts for one example run of MO-GOMEA with different numbers of clusters on MacroNAS-C10 (left) and MacroNAS-C100 (right). **Extreme clusters are disabled**

Extreme clusters enabled As shown in Figure C.2, there is a visually significant difference between MO-GOMEA with 2 clusters and the other configurations of MO-GOMEA. This is the result of the 2 clusters being assigned as extreme clusters, therefore there is no other cluster that performs a multi-objective search on this problem. Obviously, this results in poorer performance than when actual multi-objective search is performed alongside single-objective search. Obtained fronts of single runs are shown in Figure C.3.

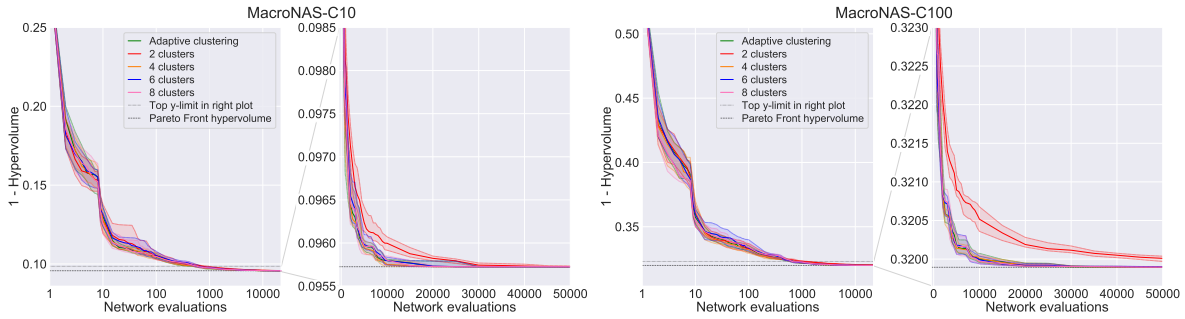


Figure C.2: Experiment 2: Medians and 25/75th percentile of 30 runs of MO-GOMEA with different numbers of clusters on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). All configurations use *IMS*, forced improvements and learned LT crossover. **Extreme clusters are enabled**. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

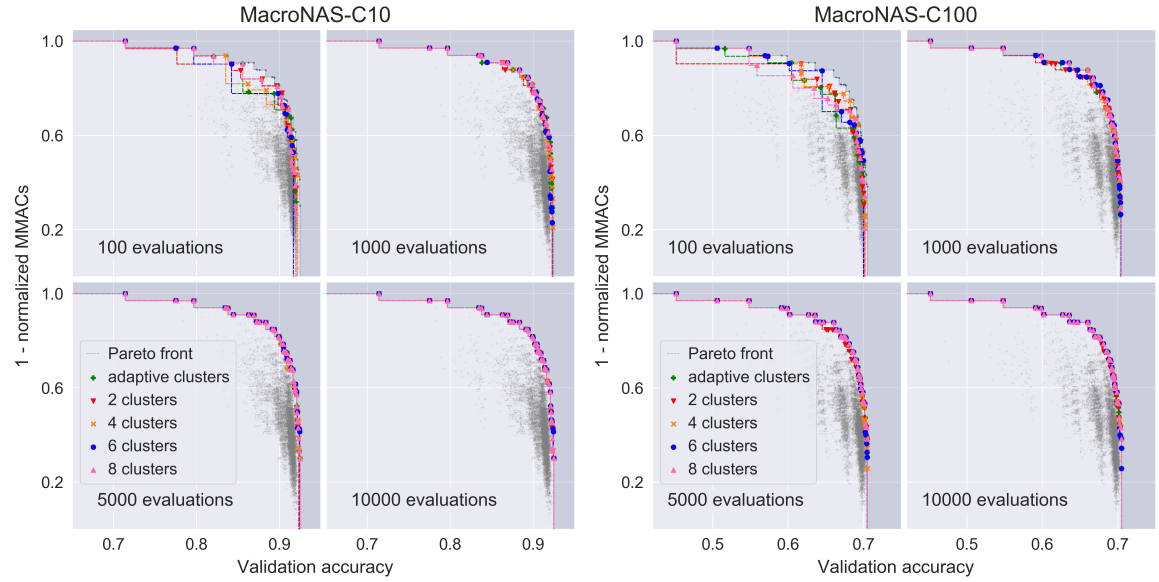


Figure C.3: Experiment 3: Achieved fronts for one example run of MO-GOMEA with different numbers of clusters on MacroNAS-C10 (left) and MacroNAS-C100 (right). **Extreme clusters are enabled**

C.2. Experiment 5: Effect of linkage learning

Figures C.4 and C.5 show hypervolume convergence for some extra FOS structures. In particular, Figure C.4 includes the Incremental LT FOS, the reversed version and a Triplet FOS. Figure C.5 includes ordered variants of the Tuple Tree FOS. The specification for each of the shown FOS elements can be found in Appendix D.

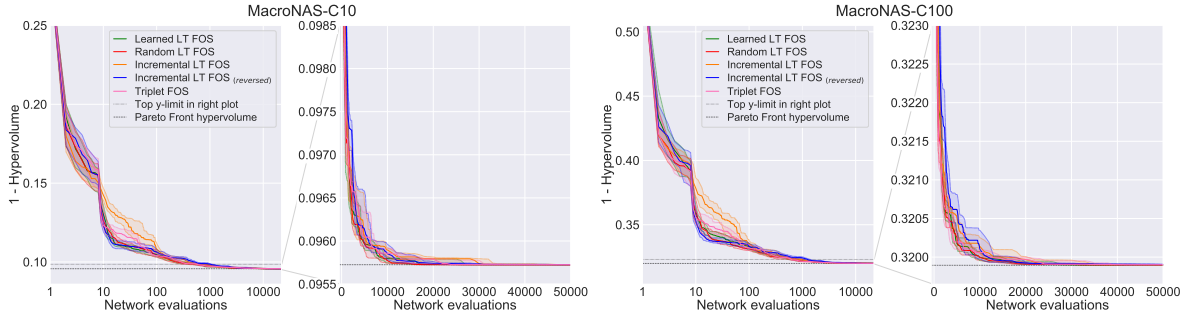


Figure C.4: Experiment 5: Medians and 25/75th percentile of 30 runs of MO-GOMEA with different FOS schemes in the Optimal Mixing phase, on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). All configurations use *IMS* (together with adaptive clustering) and forced improvements. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

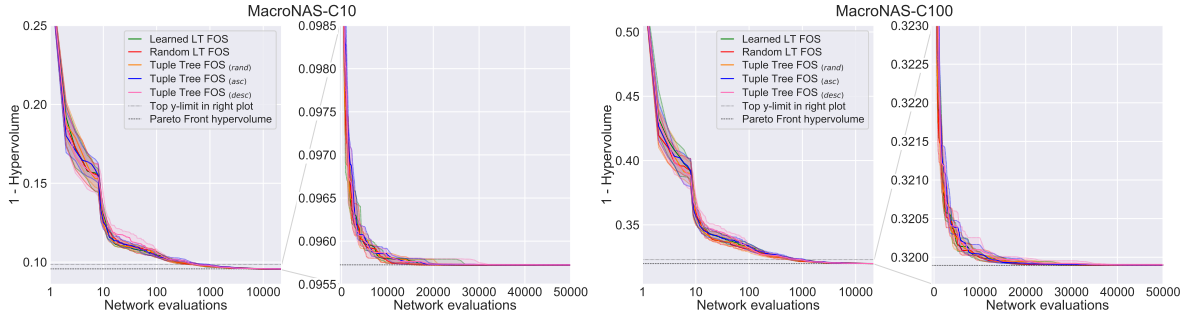


Figure C.5: Experiment 5: Medians and 25/75th percentile of 30 runs of MO-GOMEA with different FOS schemes in the Optimal Mixing phase, on MacroNAS-C10 (top) and MacroNAS-C100 (bottom). All configurations use *IMS* (together with adaptive clustering) and forced improvements. Both graphs display the same data, left has a logarithmic scale, right zooms in with a linear scale.

Figure C.6 shows heatmaps for success ratios for differently sized FOS elements of the LLT for 50 generations.

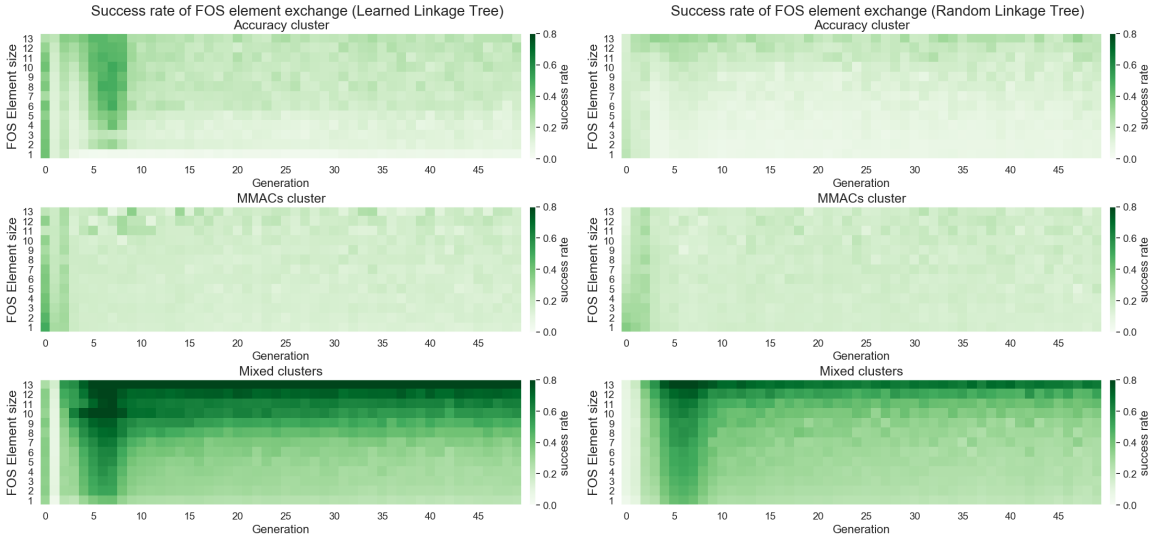


Figure C.6: Experiment 5: Heatmap of success rates of FOS element exchange for the Learned Linkage Tree (left) and the Random Linkage Tree (right) on MacroNAS-C100, against the generations in MO-GOMEA. The results are split into the two extreme clusters and mixed clusters. Results are combined from 100 runs. Results from mixing in the forced improvements phase are not included.

Figure C.7 shows detection and success ratios of specific FOS elements in the LLT, sorted on detection rate.

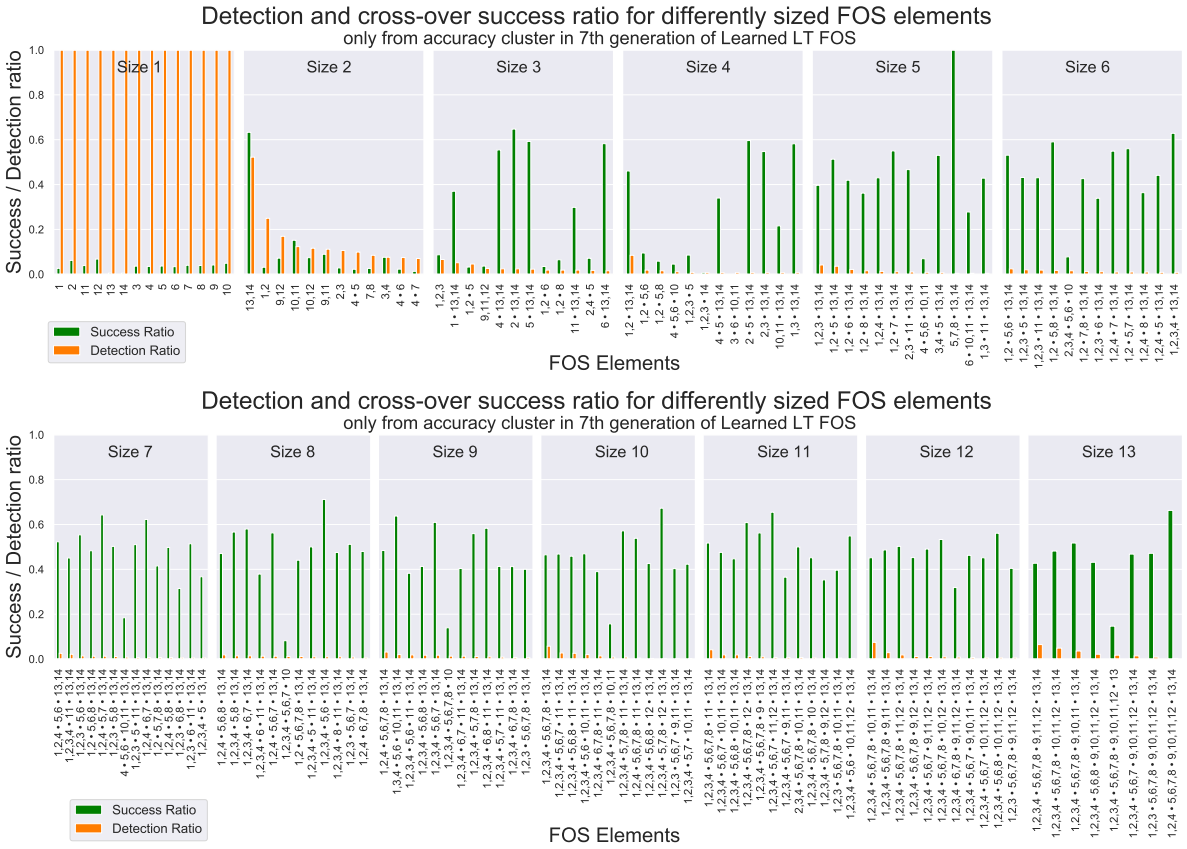


Figure C.7: Experiment 5: Success and detection rate of differently sized FOS elements, of generation 7 of MO-GOMEA with learned LT FOS on MacroNAS-C100. Success ratio: # of mixes with FOS element result in accepted change / # of total mixes using this FOS element mask. Detection ratio: # FOS element appearances in Learned LT / # of LT's learned. Results are combined from a 100 runs. Only the 12 most detected FOS elements that appear in **≥ 50** cross-over operations are shown.

D

Implemented framework for NAS experiments

During the thesis, a significant effort is spent on developing a framework for running optimizations and NAS experiments quickly. This section describes which objective functions, optimization algorithms and other useful features are included in the framework. The code of the framework (mainly C++ and some Python) can be found at `https://github.com/tdenottelander/GA`.

D.1. Objective functions

In order to make sure that the search algorithms were implemented correctly, I implemented some toy problems on which I could compare the scalability of algorithms to known results [76]. It also allowed me to see the performance of algorithms (and different FOS structures) on problems like leading-ones-trailing-zeros, which served as a starting point to approach NAS. Next to these artificial problems, of course also actual NAS tasks are implemented. Besides MacroNAS-C10 (ARK8) and MacroNAS-C100 (ARK7), the process of their creation involved creating many prototypes of NAS search spaces on CIFAR-10 and CIFAR-100 (ARK1-6). NAS-Bench-101 is also included, although it requires messaging between C++ and Python. The large-scale version of MacroNAS is available too, which involves a connection with Python as well.

- Toy problems (number of variables can be varied)
  - Single-objective: One-max, Leading-ones, Leading-ones-trailing-zeros, Trap-N**, NK-landscapes**
  - Multi-objective: Trap-inverse-trap**, MAXCUT, Zero-max-one-max
- NAS
  - ARK1*, ARK2*, ARK3*, ARK4*, ARK5*, ARK6, ARK7 (MacroNAS-C100), ARK8 (MacroNAS-C10), ARK_Online (NAS-Bench-101 and large-scale MacroNAS)

*) only available for single-objective search.
**) can be configured with more parameters besides the number of variables.

D.2. Algorithms

For comparison and to find well-performing configurations of (evolutionary) algorithms, multiple are implemented. For EAs, different selection and variation operators are implemented as modules, making it easy to modify and combine different operators within an EA. For selection operators, (configurable) tournament selection and proportionate selection are implemented. For cross-over operators, univariate, one-point, two-point, three-point and FOS-based cross-over are implemented. Besides the Learned LT FOS, also some pre-specified FOS schemes (see Table D.1) are implemented to use for (MO-)GOMEA.

To develop a better understanding of algorithms, most are implemented by myself. The implemented optimization algorithms are:

- Single-objective: Random Search, (Stochastic) Local Search, Genetic Algorithm (SimpleGA), GOMEA[1]
- Multi-objective: MO Local Search, MO Random Search, NSGA-II, MO-GOMEA[2]

Name	Configurable	FOS
Incremental LT	N/A	$\{\{x_1\},\{x_1,x_2\},...,\{x_1,...,x_\ell\}\}$
Incremental LT (Reversed)	N/A	$\{\{x_\ell\},\{x_\ell,x_{\ell-1}\},...,\{x_\ell,...,x_1\}\}$
Univariate FOS	Order (asc, desc, rand)	$\{\{x_1\},\{x_2\},...,\{x_\ell\}\}$
Triplet FOS	Order (asc, desc, rand)	$\{\{x_1,x_2,x_3\},...,\{x_{\ell-2},x_{\ell-1},x_\ell\}\}$
Triplet Tree FOS	Order (asc, desc, rand)	$\{\{x_1,x_2,x_3\},...,\{x_{\ell-2},x_{\ell-1},x_\ell\},\{x_1,x_2,x_3,x_4,x_5,x_6\},...,\{x_{\ell-5},x_{\ell-4},x_{\ell-3},x_{\ell-2},x_{\ell-1},x_\ell\}\}$
Tuple Tree FOS	Order(asc, desc, rand)	$\{\{x_1,x_2\},\{x_3,x_4\},...,\{x_1,x_2,x_3,x_4\},...\{x_{\ell-3},x_{\ell-2},x_{\ell-1},x_\ell\}\}$
ARK6 FOS	Order (asc, desc, rand)	$\{\{x_1,x_2,x_3,x_4\},...,\{x_9,x_{10},x_{11},x_{12}\},\{x_{13},x_{14}\}\}$
Random Tree FOS	N/A	$\{\{x_i\},\{x_j\},...,\{x_\ell\},\{x_i,x_j\},...,\{x_{\ell-1},x_\ell\},...,\{x_i,...,x_\ell\}\}$ (consider indices i, j, etc. shuffled)
Combinations	Order(asc, desc, rand)	e.g. $FOS_{IncrementalLT}\|\|FOS_{Univariate}$

Table D.1: Pre-specified FOS schemes

D.3. Miscellaneous

Some notable aspects of the framework include:

- Different convergence criteria are implemented: Based on time, (unique) evaluations, optimum (based on fitness or on genotype), covering entire or a percentage of pareto, (epsilon) Pareto distance, or none at all.
- A configurable RoundSchedule class is created for applying the *IMS* for the population-based algorithms.
- An elitist archive is implemented for all multi-objective optimization algorithms, which contains all non-dominated solutions found so far and is updated with every evaluation.
- The framework is designed solely for discrete domains, but allows for non-binary problem encodings. On NAS-Bench-101, an adaption is made such that the encoding is partly binary and partly tertiary.
- A solution library is implemented that contains a mapping to transform encodings into unique architectures, in order to keep track of unique evaluations (note that e.g. in MacroNAS, multiple encodings describe the same architecture)
- A pipeline is created for writing optimization results and statistics as JSON files.
- In order to parallellize real-time training and evaluations of architectures on the different servers available at CWI, some shell scripts are written to distribute the required evaluations via SSH.
- The program can be called from the command line. Almost 30 different command line arguments can be passed when executing the program, allowing for many different configurations.

[1]The linkage learning part is from `https://github.com/marcovirgolin/GP-GOMEA/blob/master/GOMEA/GOMEAFOS.cpp`.
[2]Fully taken from `https://homepages.cwi.nl/~bosman/source_code.php`, but adapted to the framework.

E

Literature overview of single-objective NAS approaches

Title	Authors	Year	cit	cit/y	Name of net/algo	NAS Type: Architecture	NAS Type: Weights	NAS Type: Hyperparameters	Approach	Operations	Flexible macro-Architecture	Flexible micro-architecture	Level	(primitive) Options	Encoding	Dataset (Prediction error)	Training time
Designing Convolutional Neural Network Architectures	M. Suganuma et al.	2018	195	65	CGP-CNN	yes	no	no	EA (CGP)	Selection, mutation	yes	n/a	Layers overall	ResBlock ([32, 64, 128], [3x3, 5x5]), Max pool, avg pool, summation, concatenation	encoding: fixed 2D grid for node	CIFAR-10 (6.75% / 5.98%) default scenario, CIFAR-10 (8.05%) small-data scenario	Default: 14 days, 2 GPUs, Small-data: 5 days, 2 GPUs
Evolving Deep Neural Networks	R. Miikkulainen et al.	2019	292	146	CoDeepNEAT	yes	no	yes	EA	Mutation, but unclear	yes	yes	Blueprints and overall structure	Conv (filters [32, 256], dropout rate [0, 0.7], [init weight scaling [0, 2.0], kernel size [1x1, 3x3], max pool [true, false])	Graph-based? Not mentioned	CIFAR-10 (7.3%), Language Modeling Benchmark, MSCOCO Image Captioning	?
Evolving Neural Networks through Augmenting Topologies	K. O. Stanley, R. Miikkulainen	2002	2642	139.1	NEAT	yes	yes	no	EA (NEAT)	Mutation, crossover, speciation	yes	n/a	Nodes	-	Flexible-length genotype string (w/ historical markings)	Pole balancing	?
Neural Architecture Search with Reinforcement Learning	B. Zoph, Q. V. Le	2017	1204	301	NAS v3 (?)	yes	no	no	RL	n/a	yes	n/a	Layers	unclear		CIFAR-10 (3.65%), Penn Treebank (62.4)	C-10: 800 GPUs, PTB: 96 GPUs
Regularized Evolution for Image Classifier Architecture Search	E. Real et al.	2019	398	199	AmoebaNet	yes	no	no	EA	Selection, mutation	no	yes	Layers within 2 different but repeated blocks	Follows nasnet [11]	NASNet search space	CIFAR-10 (3.34%), ImageNet (16.1%/3.3%)	C-10: 7 days, 450 GPUs
NAS-Bench-101: Towards Reproducible Neural Architecture Search	C. Ying et al.	2019	28	14	NAS-Bench-101	yes	no	no	Multiple	n/a	no	yes	Layers within repeating blocks	3x3 conv, 1x1 conv, 3x3 maxpool	Connection matrix + list of operations per layer	-	-
Learning Transferable Architectures for Scalable Image Recognition	B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le	2018	1055	351.7	NASNet	yes	no	no	RNN	n/a	no	yes	Layers within cells	Identity, 1x3 + 3x1 conv, 1x7 + 7x1 conv, 3x3 dilated conv, 3x3 avg pool, 3x3 max pool, 5x5 max pool, 7x7 max pool, 1x1conv, 3x3conv, 3x3 dw-sep conv, 5x5 dw-sep conv, 7x7 dw-sep conv	NASNet search space	CIFAR-10 (2.40%), ImageNet (19.2%/4.7%)	C-10: 500 GPUs, 4 days
Practical Block-Wise Neural Network Architecture Generation	Z. Zhong, J. Yan, W. Wu et al.	2018	94	31.33	Block-QNN	yes	no	no	RL (Q-learning)	n/a	no	yes	Layers within repeating blocks	Conv1x1, conv3x3, conv5x5, maxpool1x1, maxpool3x3, avgpool1x1, avgpool3x3, identity, Elemental Add, Concat, Terminal	Graph-based?	CIFAR-10 (3.54%), CIFAR-100 (18.06%)	3 days w/ 32 GPUs
Understanding and simplifying one-shot architecture search	G. Bender, P. Kindermans, B. Zoph et al.	2018	98	32.67	One-Shot Top	yes	no	no	Random	n/a	no	yes	Layers within choice blocks within identical cells	1. Identity 2. pair of depthwise separable 3x3 convs 3. pair of depthwise separable 5x5 convs 4. pair of depthw. Sep. 7x7 convs 5. 1x7conv + 7x1conv 6. max pool 7. avg pool	Not specified	CIFAR-10 (3.9%), ImageNet (24.8%)	C-10: 16 GPUs, IN: 4 Cloud TPUs (16 chips)
Progressive Neural Architecture Search	C. Liu, B. Zoph et al.	2018	413	137.7	Progressive NAS / PNAS/ PNASNet-5	yes	no	no	SMBO	Selection (based on surrogate model prediction)	no	yes	Layers within cells	3x3 dw-sep conv, 5x5 dw-sep conv, 7x7 dw-sep conv, 1x7+7x1 conv, Identity, 3x3 avg pool, 3x3 max pool, 3x3 dilated conv	Not specified	CIFAR-10 (3.41%), ImageNet_Mobile (25.8%/8.1%), ImageNet_Large (17.1%/3.8%)	? (But its very efficient)
Evolving Deep Convolutional Neural Networks for Image Classification	Y. Sun, B. Xue, M. Zhang, G. Yen	2019	38	19	EvoCNN	yes	Only init values	no	EA	Slack binary tournament selection, crossover, mutation	yes (within bounds)	n/a	Layers	Conv layer (+ properties), pooling layer (+ properties), full connection layer (+ properties)	String with extra encoded info per layer	Fashion (5.47%), also other small-scale datasets	Fashion: 4 days
Large-scale evolution of image classifiers	E. Real, S. Moore, A. Selle et al.	2017	466	116.5	-	yes	weight sharing	Learning rate only	EA	Mutation, tournament selection	yes	n/a	Layers	Convs only, but with different filter sizes and strides	?	CIFAR-10 (4.4%), CIFAR-100 (23%)	few days, 250 parallel workers
Genetic CNN	L. Xie, A. Yuille	2017	225	56.25	GeNet	yes	no	no	EA (GA)	Roulette wheel selection, mutation on gene-level, crossover on stage-level	manual	yes (within bounds)	Connections between layers inside multiple stages	Convs only (but any connection is possible)	Binary string of upper triangular connection matrix	MNIST (0.34%), CIFAR-10 (7.1%), CIFAR-100 (29.03%), ILSVRC2012 (27.87%/9.74%)	C-10: 17 GPU-days
Hierarchical representations for efficient architecture search	H. Liu, K. Simonyan, O. Vinyals et al.	2018	238	79.33	Hierarchical NAS (?)	yes	no	no	EA (GA)	Mutation, tournament selection	no	yes	Layers within repeating blocks	1x1 conv C channels, 3x3 dw conv, 3x3 sep conv C channels, 3x3 maxpool, 3x3 avgpool, Identity	Upper triangular connection matrix with operations	CIFAR-10 (3.75%), ImageNet (20.3%/5.2%)	1.5 days w/ 200 GPU workers
Efficient Neural Architecture Search via parameter Sharing	H. Pham, M. Guan, B. Zoph, et al.	2018	428	142.7	ENAS	yes	parameter sharing	no	RL (Controller LSTM)	n/a	yes* (can be applied on both)	yes* (can be applied on both)	Both on layers and on layers within repeated blocks	Any	Any	CIFAR-10 (macro: 3.87%, micro: 2.89%), Penn Treebank (55.8)	C-10: , PTB: 10 hours 1 GPU