Annals of Mathematics and Artificial Intelligence
https://doi.org/10.1007/s10472-022-09823-0

A data-driven approach to neural architecture search initialization

Kalifou René Traoré[1,2] · Andrés Camero[2,3] · Xiao Xiang Zhu[1,2]

Accepted: 19 October 2022

Abstract
Algorithmic design in neural architecture search (NAS) has received a lot of attention, aiming to improve performance and reduce computational cost. Despite the great advances made, few authors have proposed to tailor initialization techniques for NAS. However, the literature shows that a good initial set of solutions facilitates finding the optima. Therefore, in this study, we propose a data-driven technique to initialize a population-based NAS algorithm. First, we perform a calibrated clustering analysis of the search space, and second, we extract the centroids and use them to initialize a NAS algorithm. We benchmark our proposed approach against random and Latin hypercube sampling initialization using three population-based algorithms, namely a genetic algorithm, an evolutionary algorithm, and aging evolution, on CIFAR-10. More specifically, we use NAS-Bench-101 to leverage the availability of NAS benchmarks. The results show that compared to random and Latin hypercube sampling, the proposed initialization technique enables achieving significant long-term improvements for two of the search baselines, and sometimes in various search scenarios (various training budget). Besides, we also investigate how an initial population gathered on the tabular benchmark can be used for improving search on another dataset, the So2Sat LCZ-42. Our results show similar improvements on the target dataset, despite a limited training budget. Moreover, we analyse the distributions of solutions obtained and find that that the population provided by the data-driven initialization technique enables retrieving local optima (maxima) of high fitness and similar configurations.

✉ Xiao Xiang Zhu
xiaoxiang.zhu@tum.de

Kalifou René Traoré
kalifou.traore@dlr.de

Andrés Camero
andres.camerounzueta@dlr.de

[1] Data Science in Earth Observation, Technical University of Munich, Arcisstrasse 21, 80333, Munich, Bavaria, Germany

[2] Remote Sensing Institute, German Aerospace Center (DLR), Münchener Strasse 20, 82234, Weßling, Bavaria, Germany

[3] Helmholtz AI, Munich, Germany

Published online: 22 March 2023

Keywords AutoML · Neural architecture search · Evolutionary computation · Search · Initialization

Mathematics Subject Classification (2010) 68Txx · 68T20 · 68T45

1 Introduction

Deep learning has successfully been applied to a wide variety of problems, showing (in many cases) *super human* performance [1, 2]. However, this success has been followed by an increasing complexity of the deep learning models, that in most cases are manually designed [3]. State-of-the-art *deep neural networks* (DNNs) may have several million parameters, thus automating the design of DNNs is the *logical* next step.

Neural architecture search (NAS) is the process of automating architecture engineering [3, 4]. Great advances have been made in this matter, but in practice NAS has not been adopted yet [3, 4]. From an optimization point of view, NAS is a challenging task. It requires dealing with huge search spaces (the deeper the model, the bigger the search space), mixed type solutions (i.e., a combination of integer, discrete, and real values to represent the model), and solutions that are *expensive* to evaluate (i.e., training a DNN on a large data set may take several days).

Researchers are working to alleviate NAS challenges. Several (optimization) approaches have been tailored to design neural networks [5–7], to speed up model evaluation [8, 9], and a lot of effort have been put to design NAS search spaces [5, 6]. Recently, some authors have released performance evaluation databases [10, 11], aiming to democratize NAS, i.e., *everyone* can test a NAS algorithm regardless of having powerful computational resources, and to improve the reproducibility [3].

Despite the great advances made so far, there remain open questions. For example, many NAS approaches can be categorized as population-based algorithms [3, 4]. However, little attention has been drawn to the initialization of the population. Considering all the available resources, including NAS databases, we propose to address the following question: *Can we improve the performance of a population-based NAS algorithm by initializing its population with a data-driven approach?* To this problem, we propose a novel approach to initializing population-based NAS algorithms. First, a tailored clustering analysis of a target search space is performed. Second, after obtaining satisfying clustering results, the centroids are extracted and used to initialize a population based NAS algorithm.

To validate our proposal, we selected three population-based NAS algorithms: an evolutionary algorithm (EA) [12], a genetic algorithm (GA) [13], and aging evolution (AE) [6], and benchmark it on the NAS-bench-101 [10] dataset, against the most popular initialization methods (random initialization and Latin hypercube sampling). We find that centroids extracted using *Bayesian Gaussian Mixture of models* (BGM) for clustering are a promising approach to initialize the population. Particularly, our approach used with GA shows significant long-term improvements (after 2000 iterations, in test) over random initialization and Latin hypercube sampling. When used with EA, a faster convergence (in validation) and a significant long-term improvement over random initialization and Latin hypercube sampling is observed. Besides, we also investigate how an initial population gathered with our proposal on a tabular benchmark can be used for augmenting search on a real world problem, the So2Sat LCZ-42 scene classification dataset. Our results show that this set of solutions help accelerate the convergence EA on the target dataset, despite a short training budget.

Last but not least, we investigate the distributions of the solutions found by the benchmarked algorithms. More precisely, we compare their configurations based on the initialization used for each algorithm. Our results suggest that the proposed initialization method (centroids) enable retrieving local optima (maxima) of high fitness and similar configurations.

To summarize, the contributions of this paper are the following:

- We propose a novel data-driven initialization method for population-based NAS algorithms: After performing a clustering analysis of the search space, we use the obtained centroids to initialize a NAS algorithm. The aim of the proposal is to improve the convergence and long-term performances of a target algorithm, without modifying it.
- We benchmark the proposal on three NAS algorithms and against popular initialization baselines, and report improvements over other initialization techniques.
- We investigate how the proposed method using a tabular benchmark as input can be used for augmenting search on another dataset, the So2Sat LCZ-42. We report similar improvements on the target dataset, despite a limited training budget.

The remainder of this article is organized as follows: The following section introduces NAS and briefly summarizes the state-of-the-art of initialization techniques. Section 3 describes the proposed methodology. Section 4 introduces the experimental setup. Section 5 presents the results. Finally, Section 6 outlines the conclusions and proposes future work.

2 Related work

In this section, we summarize some of the most relevant works related to our proposal. First, we introduce NAS and highlight the state-of-the art, with a special emphasis on metaheuristic approaches. Second, we outline the population initialization problem. Third, we describe the three baseline algorithms that we aim to initialize.

2.1 Neural architecture search

NAS is the process of automating architecture engineering [3]. Currently, it is considered to be a subfield of AutoML [14]. However, its roots can be tracked to the late 1980s, where the use of *evolutionary computation* was explored to design and train neural networks [15–19]. These ideas gather together under the *neuroevolution* concept, and in the 2000s gained popularity thanks to the *NeuroEvolution of Augmenting Topologies* (NEAT) method [20], a *genetic algorithm* (GA) that increasingly evolves complex neural network topologies and weights. Later, due to apparition of deep learning, the neuroevolution research started to attract attention again [3, 4].

From the optimization (algorithm) point of view, many approaches can be found in the neuroevolution literature, ranging from evolutionary algorithm (EA) [21], GA [22], *harmony search* [23] and *mixed integer parallel efficient global optimization* technique [24], to Bayesian optimization [7]. And also, from the point of view of the neural network architecture, e.g., recurrent neural network [25], convolutional neural network [26] and generative adversarial networks [27].

On the other hand, in the past few years, a *new branch* of NAS approaches emerged based on *continuous optimization* (e.g., DARTS [5]). Particularly, these approaches search over a large graph of overlapping configurations (i.e., the *Super-Net*) using a gradient-based approach. These recent improvements result in large speedups in terms of search time [3] but also in some case in a lack of robustness and interpretability [28].

Despite the NAS approach used, the literature stresses the importance of optimizing the architecture of a deep neural network (given a particular problem). The main challenges of NAS are three-fold: First, the number of the parameters increases in proportion to the number of layers, thus the search space is huge. Second, the search space is (usually) a mix of categorical (e.g., the type of operation, the activation functions, ...), real (e.g., the weights), and integer (e.g., the number of hidden layers, the number of neurons per layer, ...) or discrete (e.g., the adjacency matrix) values, resulting in a complicated problem, i.e., each parameter type require a different optimization approach. Third, the evaluation of an architecture is extremely resource and time-consuming. Therefore, NAS problems fall into the family of expensive optimization problems.

To cope with the latter problem, i.e., the evaluation cost, and aiming to improve reproducibility, lots of effort have been made to provide open-source benchmarks for NAS. Several areas of applied *machine learning* have been included in these benchmarks, including *computer vision* (CV) [11, 29–31] and *natural language processing* (NLP) [32], among others [3]. Also, some authors have explored techniques to speed up the performance evaluation, including learning curve-based estimation [8], one-shot (weight sharing) [33, 34], training-free methods (a.k.a., 0-shot) [9, 35, 36], among others. The mixed search space problem has been faced from multiple perspectives, ranging from tailored encoding [37, 38], and specific operations [25], to mixed (hybrid) approaches [39].

Finally, to tackle the problems that arise due to the size of the search space (first challenge), several authors have invested time tailoring the design of the search space [5, 6], providing tools to assess its *quality* [40, 41], and proposing techniques to adapt the search space [7, 42], among others. Despite all the advances made in this regard, the initialization of NAS algorithms (especially the population-based ones) has not received much attention. However, it is important to remark that starting from a set of *good* solutions is key to solve large-scale optimization problems using a population of finite size [43].

2.2 Population initialization techniques

All population-based metaheuristic algorithms share a common step: The population initialization. The goal of this initialization is to provide a first set of solutions, that (normally) will be improved in an iterative way until the termination criteria is met. A *good* (or *bad*) initial population facilitates (or prevents) finding the optima [43–45], and this is especially true for large-scale optimization problems that use a small population size [43], which is the most common case. Therefore, the greater the search space (given a limited population size), the smaller the chance to cover promising regions of the search space [46].

In the past few decades, some authors have started to propose initialization techniques aiming to boost the performance of population-based metaheuristic algorithms (mainly EA) [47, 48]. Great advances have been made, for example, [49] shows that initialization can increase the probability of finding global optima, [50] shows that stability can be improved, and in [51] it is show that the solution quality is related to the initialization, among many others [47, 48].

However, in black-box optimization problems, such as NAS, it is not possible to determine beforehand what is a good and bad solution. Therefore, not all initialization techniques are suitable for NAS. Moreover, few practical rules of thumb are provided in the literature to choose an appropriate initialization technique. Thus, from a practitioner perspective, it is unclear how to choose the right initialization technique [47].

Considering all these limitations, most practical NAS implementations rely on (quasi)random [49] initialization or Latin hypercube sampling (LHS) [7, 52–54]. Therefore, in this study, we propose to tackle the population initialization problem for NAS.

2.3 Baseline algorithms

Next, we introduce three important population-based algorithm, that we consider as baselines for the benchmark of initialization techniques. Particularly, a Genetic Algorithm (GA), an Evolutionary Algorithm (EA), and the *Aging Evolution* (AE) [6], a popular EA-based algorithm specifically designed for NAS on CV problems.

2.3.1 Genetic algorithm

A GA is a population-based meta-heuristic algorithm inspired by natural evolution [13]. At a glance, a *population* of *individuals* (a.k.a. solution) is evolved using selection, crossover, mutation, and replacement operations. Particularly, we use the implementation available in the latest version (1.3.1) of DEAP library [55]. Algorithm 1 presents a high-level view of the implemented GA.

```
input: The size of the population $pop\_size$, the crossover probability $cx\_p$, mutation
probabilities $mut\_p$ and $mut\_i$, and the maximum number of evaluations
$max\_evaluations$. Alongside with $train$, $validation$, and $test$ data sets.
SOLUTION $\leftarrow \emptyset$
POPULATION $\leftarrow$ Initialize($pop\_size$)
Evaluate(POPULATION, $train$, $validation$)
EVALUATIONS $\leftarrow pop\_size$
SOLUTION $\leftarrow$ Best(SOLUTION, POPULATION)
while EVALUATIONS $\leq max\_evaluations$ do
    OFFSPRING $\leftarrow \emptyset$
    for $j \leftarrow 1$ to $pop\_size$ do
        PARENT_1 $\leftarrow$ BinaryTournament(POPULATION)
        PARENT_2 $\leftarrow$ BinaryTournament(POPULATION)
        CHILD $\leftarrow$ SinglePointCrossover(PARENT_1, PARENT_2, $cx\_p$)
        OFFSPRING $\leftarrow$ OFFSPRING + CHILD
    end
    OFFSPRING $\leftarrow$ Mutate(OFFSPRING, $mut\_p$, $mut\_i$)
    Evaluate(OFFSPRING, $train$, $validation$)
    POPULATION $\leftarrow$ OFFSPRING
    SOLUTION $\leftarrow$ Best(SOLUTION, POPULATION)
    EVALUATIONS $\leftarrow$ EVALUATIONS + $pop\_size$
end
PERFORMANCE $\leftarrow$ Evaluate(SOLUTION, $train$, $test$)
return SOLUTION, PERFORMANCE
```

Algorithm 1 Genetic algorithm.

Particularly, an individual encodes a neural network architecture (in the given search space) by a mix of binary entries, that represent the adjacency matrix of the architecture, and categorical values, that correspond to the operations on the edges of the adjacency matrix. Please refer to Section 4.1 for more details.

An initial *population* of size *pop_size* is initialized by the function `Initialize(·)`. Particularly, we define three variations to initialize the population: Random initialization, LHS, and our proposed method (Section 3). An *individual* is evaluated by the function `Evaluate(`$\mathcal{D}_1$`, `$\mathcal{D}_2$`)`. The decoded architecture is trained using SGD on $\mathcal{D}_1$ data set, and evaluated (accuracy) on $\mathcal{D}_2$ data set (a.k.a., the fitness). Then, the best *solution* (i.e., the one with the highest accuracy) of the *population* is selected by the `Best(·)` function.

Then, the evolution takes place. First, an *offspring* of size *pop_size* is created. More specifically, each *offspring* individual is created by a single point crossover operation `SinglePointCrossover(`$\mathcal{P}_1$`, `$\mathcal{P}_2$`, cx_p)` with probability *cx_p*, where $\mathcal{P}_i$ is selected using a binary tournament operation `BinaryTournament(·)`. Note that with probability 1 - *cx_p* one of the parents $\mathcal{P}_i$ is returned unmodified. Later, the *offspring* is mutated with probability *mut_p* by the function `Mutate(·)`. If mutated, each position is mutated using bit-flip (for the binary entries) or round-robin (for the categorical values) with probability *mut_i*. The *offspring* is evaluated using `Evaluate(·)`, and the current *population* is replaced by the *offspring*. Finally, the best *solution* is updated, i.e., if the fitness of the best individual in the *population* is higher than the current best *solution*, then the best individual become the best *solution*. Once the number of *max_evaluations* is reached, the best solution is evaluated using the *test* data set.

2.3.2 $(\mu + \lambda)$ Evolutionary algorithm

The $(\mu + \lambda)$EA [12], a generic population-based metaheuristic algorithm, evolves a population of μ individuals by creating λ offspring. Then, both the original population and the offspring are combined, and the best μ individuals replace the population. Algorithm 2 presents a high-level view of the $(\mu + \lambda)$EA basic implementation provided by the latest version (1.3.1) of DEAP library [55].

The *population* (refer to Section 2.3.1) of size μ is initialized using the `Initialize(·)` function. Then, the *population* is evaluated using the `Evaluate(·)` function (refer to Section 2.3.1). Then, the evolutionary process takes place. First, an *offspring* of size λ is generated by randomly sampling (with uniform probability) individuals from the *population*. Following, the *offspring* is mutated using the `Mutate(·)` function (refer to Section 2.3.1), and the offspring is evaluated. In the last evolutionary step, the *population* and the *offspring* are combined, ranked according to their fitness, and the top μ individuals are selected by the `RankSelection(·)` to replace the current *population*.

Once the number of evaluations is greater than *max_evaluations*, the best individual of the *population* is selected by `Best(·)`, i.e., the *solution*. Finally, the *solution* is evaluated on the *test* data set.

2.3.3 Aging evolution

A few years ago, the *Aging Evolution* (AE) [6], an *EA*-based approach to NAS, became popular because it achieved state-of-the-art performance on classical CV benchmarks.

```
input: The size of the population $\mu$, the size of the offspring $\lambda$, mutation probabilities
mut_p and mut_i, and the maximum number of evaluations max_evaluations.
Alongside with train, validation, and test data sets.
POPULATION $\leftarrow$ Initialize($\mu$)
Evaluate(POPULATION, train, validation)
EVALUATIONS $\leftarrow \mu$
while EVALUATIONS $\leq$ max_evaluations do
    OFFSPRING $\leftarrow$ RandomSample(POPULATION, $\lambda$)
    OFFSPRING $\leftarrow$ Mutate(OFFSPRING, mut_p, mut_i)
    Evaluate(OFFSPRING, train, validation)
    POPULATION $\leftarrow$ RankSelection(POPULATION + OFFSPRING, $\mu$)
    EVALUATIONS $\leftarrow$ EVALUATIONS + $\lambda$
end
SOLUTION $\leftarrow$ Best(POPULATION)
PERFORMANCE $\leftarrow$ Evaluate(SOLUTION, train, test)
return SOLUTION, PERFORMANCE
```

Algorithm 2 $(\mu + \lambda)$ Evolutionary algorithm.

Algorithm 3 outline AE. Notice that the nomenclature does not match exactly the one proposed in [6], instead the algorithm presents a version that is *closer* to Algorithms 1 and 2. Also, we used the implementation available on NASBench-101 repository.

```
input: The size of the population pop_size, the size of the tournament k, and the
maximum number of evaluations max_evaluations. Alongside with train, validation,
and test data sets.
SOLUTION $\leftarrow \emptyset$
POPULATION $\leftarrow$ Initialize(pop_size)
Evaluate(POPULATION, train, validation)
SOLUTION $\leftarrow$ Best(SOLUTION, POPULATION)
EVALUATIONS $\leftarrow$ pop_size
while EVALUATIONS $\leq$ max_evaluations do
    OFFSPRING $\leftarrow$ Tournament(POPULATION, k)
    OFFSPRING $\leftarrow$ Mutate(OFFSPRING)
    Evaluate(OFFSPRING)
    SOLUTION $\leftarrow$ Best(SOLUTION, OFFSPRING)
    Enqueue(POPULATION, OFFSPRING)
    Dequeue(POPULATION, OFFSPRING)
    EVALUATIONS $\leftarrow$ EVALUATIONS + 1
end
PERFORMANCE $\leftarrow$ Evaluate(SOLUTION, train, test)
return SOLUTION, PERFORMANCE
```

Algorithm 3 Aging evolution.

AE is a steady state EA, where the *oldest* individual of the population is replaced by the offspring. Particularly, the population of size *pop_size* is initialized using the function

`Initialize`(·), evaluated using the function `Evaluate(·)` (we are *reusing* the function defined above in this section), and the best *solution* is selected from the population by the function `Best(·)`.

Then, the evolution begins. First, an individual (i.e., the *offspring*) is selected using a *k* tournament selection. Second, the *offspring* is mutated by a two-step process `Mutate(·)`: (i) a *hidden state mutation*, the connections between operations in a graph-represented solution (cell) are modified, and (ii) an *operation mutation*, the operation within the *cell* is modified. Then, the *offspring* is evaluated, and if its performance is higher than the previous best seen *solution*, the solution is replaced by the *offspring* using the function `Best(·)`, In the last step of the evolution, the *oldest* individual of the population (i.e., the earliest evaluated one in the population) is replaced by the *offspring* using the `Enqueue(·)` (add the new one) and `Dequeue(·)` (remove the oldest one) functions. The authors of AE claim that exists a parallel between the introduced age-based removal to a *regularization* of the evolution.

The evolution continues until the number of evaluated candidate solutions exceed the predefined budget *max_evaluations*. Finally, the best *solution* of the population is returned.

3 A data-driven approach to initializing a NAS search strategy

This section introduces our search initialization technique. First, we describe the overall pipeline of the methodology. Second, we detail the *feature engineering* of the proposed approach.

3.1 Pipeline

This study sets out to answer the following question: *Can we improve the convergence of a population-based NAS algorithm by initializing it with a data-driven approach?*

To address this issue, we propose a data-driven initialization technique, depicted in Fig. 1 and formally described in Algorithm 4. Particularly, we propose to perform a cluster analysis on a randomly selected set of solutions. Then, we propose to use the centroids to initialize the population of a population-based NAS algorithm.

Let us consider a machine learning task and a search space Ω (i.e., a set of neural architectures). Then, let α be a clustering algorithm, and μ the size of the population to retrieve. First, (*i*) we sample a set of `N` architectures (`RandomSampling(·)`), were `N` comes from

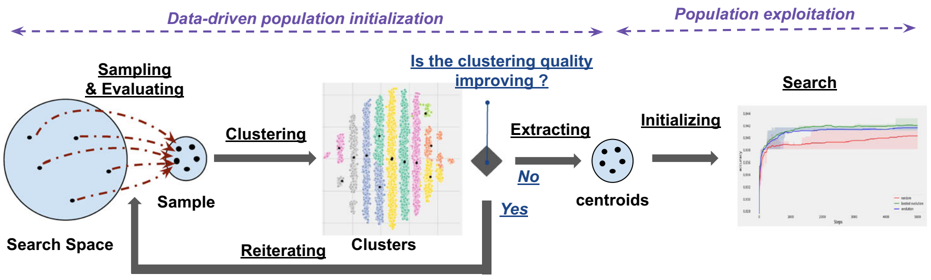


Fig. 1 The pipeline of the proposed data-driven approach to search initialization

a predefined list of sample sizes (*list_sample_size*). Then, (*ii*) we train and evaluate the performance of all sampled architectures (`EvaluateModel(·)`). Later, (*iii*) we encode (`EncodeModel(·)`) each solution using the desired feature representation ε (for more details, refer to Section 3.2). Note that as the feature vector consists of an architecture representation and its performance in test (see Table 1), the resulting clusters should relate to specific behaviors (performances) on the learned task.

Then, (*iv*) we reduce the dimension of the input features (`ReduceDimensions(·)`), as processing high dimensional and sparse data could be extremely challenging. Following, (*v*) we assign a cluster to all samples (`ClusterFeatures(·)`) using the desired clustering technique α and a predefined number of clusters μ (a.k.a., the population size). In the following step, (*vi*) we assess qualitatively and quantitatively the results using the metrics `M` (`Assess(·)`). If the quality of the new clustering is not improving the quality achieved so far, the process is stopped. Otherwise, the `clusters` are stored, a new sample size is retrieved from *list_sample_size*, and the whole process is repeated.

Once the termination criteria is met, i.e., all the sample sizes (from *list_sample_size*) have been evaluated *or* the clustering quality is not improving (refer to `Assess(·)`), the `centroids` of the `clusters` are extracted (`ExtractCentroids(·)`) and decoded (`DecodeModel(·)`).

Finally, the set of decoded centroids, i.e. the set of architectures that correspond to the centroids of the clusters (`init_architectures`), is returned. Then, the idea is to use this set of architectures (`init_architectures`) to initialize a population-based NAS algorithm.

To summarize, our data-driven initialization approach is:

- *Composite*: It is a multistep initialization procedure relying on sampling a search space, clustering it, and initializing an algorithm with the centroids extracted.
- *Generic*: It is not application-specific, in fact the clustering could be done on any type of search space given an encoding including a solution representation and its fitness evaluation.
- *Stochastic*: The stochasticity of the procedure depends on the randomness of the tool selected for clustering.

Also, it is important to note that the clusters may be analyzed to obtain insights regarding the *archetypes* (i.e., the representative architectures), including the most frequent operations and the connection between the operations (i.e., the edges in the graph).

Besides, our method does not require a target search baseline to be adapted. It can be seen as an external and complementary module helping augment a pre-existing algorithm. Indeed, an algorithm to be initialize would receive an initial population of desired sized, and ready to be used for deployment.

Table 1 Description of the feature representation encoding the solutions of the search space

Feature representation	Components
Short	Adjacency matrix + operations + fitness
Long	Expanded adjacency matrix + operations + fitness

Each component is in the form of a list, and the '+' symbol refers to the concatenation operator. The fitness component consists in the list of fitness measured for the various training budget available for a given solution

```
input: The desired population size $\mu$, a list of sample sizes $list\_sample\_sizes$ (decreasing
order), the feature encoding type $\varepsilon$, a clustering algorithm $\alpha$, the list of metrics to evaluate
the clustering quality $M$, and a search space $\Omega$.
CLUSTER_QUALITY ← ∅
for SAMPLE_SIZE ∈ LIST_SAMPLE_SIZES do
    SAMPLE ← RandomSampling(Ω, SAMPLE_SIZE)
    SAMPLE ← EvaluateModel(SAMPLE)
    ENCODED ← EncodeModel(SAMPLE, $\varepsilon$)
    REDUCED ← ReduceDimensions(ENCODED)
    LABELS ← ClusterFeatures(REDUCED, $\alpha$, $\mu$)
    NEW_CLUSTER_QUALITY ← Assess(REDUCED, LABELS, $M$)
    if CLUSTER_QUALITY > NEW_CLUSTER_QUALITY then
        break
    end
    CLUSTER_QUALITY ← NEW_CLUSTER_QUALITY
    CLUSTERS ← (ENCODED, LABELS)
end
CENTROIDS ← ExtractCentroids(CLUSTERS)
INIT_ARCHITECTURES ← DecodeModel(CENTROIDS, $\varepsilon$)
return INIT_ARCHITECTURES
```

Algorithm 4 Data-driven initialization technique.

3.2 Feature representation

To best take advantage of information about the search space when clustering, we first introduce a minimal feature engineering.

As we look to uncover models and structures relevant to NAS algorithms via clustering, we seek a feature representation encoding an architecture as well as its performances. As in [10], we consider neural architectures identified by an elementary component repeated in blocks, a feed-forward cell. This cell is a directed acyclic graph (DAG), with a maximum number of operations (nodes), a maximum number of transformations (edges) and a fixed set of possible operations (e.g., max pool, convolution 3x3) labeling each node. A cell is in practice represented as a list of selected operations and an adjacency matrix of variable size.

In order to use such data in the proposed pipeline, we construct two versions of clustering feature representation, briefly described in Table 1. The first one (**Original**, or *Short Encoding*) consists in concatenating for each model, its adjacency matrix, the list of operations, and the list of performances in test for all available training duration $\{t_0, t_1, t_2, t_3\}$. Note that this is a variable length feature representation due to the nature of the adjacency matrix.

Alternatively, the second representation (**Binary**, or *Long Encoding*) corresponds to using the expanded adjacency matrix, i.e., the matrix that considers all possible operations (according to the constraints of the search space). This is a fixed length encoding. Figure 2 shows an example of a generic adjacency matrix being tranformed into an expanded one, right before the step of flattening. Additionaly, Fig. 13 (and Table 3) provide a visualization of such matrix for solutions retrieved in various algorithm initialization settings.

Moreover, for both encodings, the vector form of the adjacency matrix is obtained by a flattening in row-major fashion, where consecutive elements in a row are put next to each other, while iterating over the different rows.

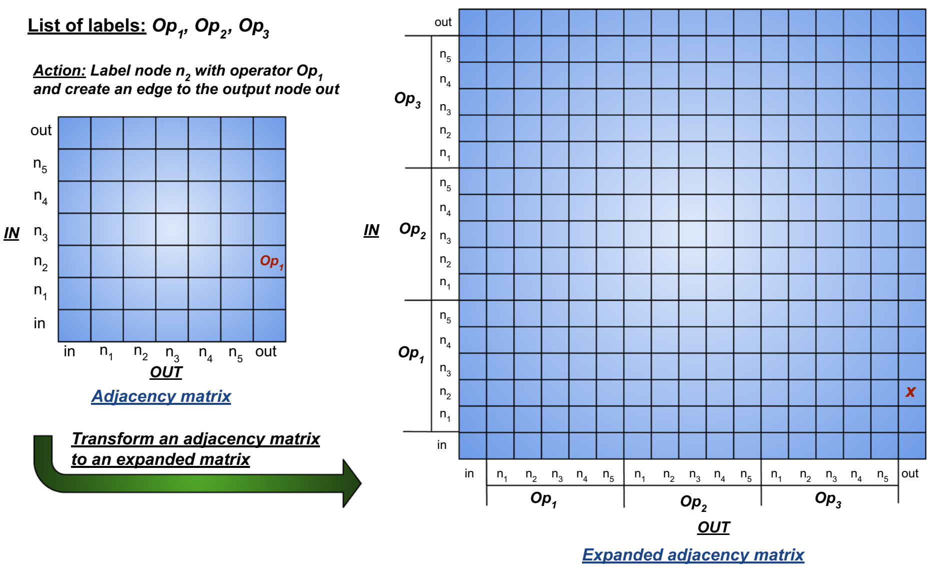

List of labels: Op_1, Op_2, Op_3
Action: Label node n_2 with operator Op_1 and create an edge to the output node out
out
n_5
n_4
IN
n_3
n_2
Op_1
n_1
in
OUT
Adjacency matrix
Transform an adjacency matrix to an expanded matrix
Op_3
Op_2
Op_1
x
Expanded adjacency matrix

Fig. 2 A visual description of the transformation of a standard adjacency matrix into an expanded adjacency matrix for a neural cell in NASBench-101

4 Experimental setup

The experiments performed aim to validate that the initialization of a population-based NAS Algorithm can benefit from models identified via Clustering Analysis of a search space. In this Section, first, we introduce the problem used to validate our proposal. Second, we present the parameters used for performing the experiments on clustering. Third, we detail the performance metrics used to assess the quality of the clustering.

4.1 NASBench-101

NASBench-101 is a database of neural network architectures and their performance evaluated on the data set of CIFAR-10. It contains $N = 450K$ unique architectures [10]. Indeed, to tackle the given machine learning task of CIFAR-10, all contained models use of a classical image classification structure similar to ResNet. Indeed, the backbone of a model contains a head, a body, and a tail. Its body is made by alternating three (3) times a block with a down-sampling module. Each block is obtained by repeating three (3) times a module called ‘cell’. A cell is a computational unit that can be represented by a DAG. It consists of an input node, an output node and intermediate nodes representing operations (convolution 3x3, convolution 1x1, maxpool 3x3), and connections indicating features being transformed. Therefore, each architecture differs by its cell. In practice, the DAG of a model is encoded by an adjacency matrix and a list of operations labelling the associated nodes. The constraints on such DAG are the following: There can be at most $N = 7$ nodes and $E = 9$ edges in a cell.

Moreover, all models were trained for 108 epochs using the same experimental setting (i.e., learning rate, etc.), but performance evaluations in training, validation, and test were also provided after 4, 12 and 36 epochs.

4.2 Hyperparameters for clustering

The clustering experiments were done with a set of randomly sampled models. The size of the sample is identified in Section 5.2. The considered clustering algorithms are *k*-means, DBSCAN, BIRCH, spectral clustering, and a BGM. All were obtained from the latest version (0.24.1) of the *scikit-learn* library [56]. Table 2 shows the hyperparameters selected for each method, including the maximum number of iterations (*Max iter*), the number of samples used at initialization (*N init*), and other algorithm-specific parameters (*Other*). Note that they are either default (NA) or slightly modified to provide satisfying clustering performances.

4.3 Clustering performance evaluation

Moreover, we use various ways of assessing the quality of the results for each step of the approach. Regarding the preparation of the initialization, we propose to measure the clustering performance using the following three (3) standard metrics: Silhouette coefficient [57], Calinski-Harabasz [58] and Davies-Bouldin index [59]. These metrics inform on how well separated and dense are the resulting clusters. They all apply in the context of clustering with missing labels, which is relevant, as we seek to investigate relevant clusters and features for NAS algorithms without prior assumptions, and well-defined clusters should provide us with centroids (i.e., models) that are good representatives of all the architectures in the search space. The Silhouette coefficient is a metric comprised between -1 and +1, with higher values associated to more dense and separated clusters The Calinski-Harabasz index also rates a better defined clusters with higher values. Similarly, the Davies-Bouldin index, measures a *similarity* between clusters, providing smaller values for better clustering. Additionally, we propose to corroborate the quantitative assessment with a qualitative analysis (visual inspection) for validation before the next step.

Regarding the step of exploitation, we assess the quality of the proposed initialization method using the performance of the search algorithm initialized (best obtained accuracy in test). More importantly, we compare the performance against initializing the same algorithm with random and LHS initialization. Also, as a sanity check, we compare the results against a *random search*.

5 Results

In this section, we present results on clustering for accelerating NAS algorithms. First, we show results on selecting the proper tool for dimension reduction and the hyperparameters

Table 2 Hyperparameters of the clustering algorithms

Method	Max iter	N init	Other
KMEANS	500	50	kmeans++ init
DBSCAN	500	200	eps=0.30
BIRCH	500	NA	threshold=0.12
SPECTRAL	500	NA	NA
BGM	500	NA	Dirichlet weight distribution, full co-variance

for the clustering. Then, we show results on identifying the number of clusters providing satisfying clustering performances. We also present results on qualitatively assessing the clusters quality for various algorithms. Then, we provide results on improving NAS performances using a centroid-based initialization for three (3) NAS evolutionary algorithms. Last but not least, we show results of a quantitative assessment for solutions found from the bench-marking, in the form of matrices of cell occurrence.

5.1 Dimension reduction

To begin our experimental study, we seek to calibrate the dimension reduction of the input features. To do so, we arbitrarily fix the number of samples to $N = 10000$, in order to perform relevant experiments.

Figure 3 shows clustering performance as a function of the number of components of input features. The blue and red curves display performance using respectively the Short (Original) and the Long encoding (Binary). The dimension reduction is performed using PCA, and the clustering with k-means using a fixed number of clusters $N = 10$.

Using the Short encoding, the three metrics are in favor of using a small number of components for input features via PCA. Indeed, the smaller the number of components the higher the Silhouette and Calinski-Harabasz scores, and the lower the Davies-Bouldin index, with optimal values for using two (2) components. The same is observed when using the Long encoding.

Using the Long encoding yields slightly better performance than the Short encoding, with a sensible improvement on larger number of components with PCA.

As both encoding are rather sparse (length of up to 58, or up to 298), we study the effect of this sparsity in the dimension reduction tool. Figure 4 shows clustering performances as a function of the number of components of input features, for various reduction tool. The blue and red curves display performances using respectively PCA and Truncated SVD as dimension reduction tools. Plot (a) and (b) display results using respectively the Short and the Long encoding. The clustering is performed with k-means.

Trying an alternative dimensional reduction tool (Truncated SVD) more suitable for highly sparse data does not worsen results on the Short encoding (see Fig. 4a). Moreover, it allows for a slight improvement over PCA when using the Long encoding (see Fig. 4b).

To summarize, the findings show that reducing the dimensions of the input features to 2D provides the best performances on both encoding. Using the Long encoding improves the results. Also, using Truncated SVD shows slight improvements as it is more suitable for sparse data. Given these findings, the following experiments are performed using Truncated SVD for a 2D reduction of input.

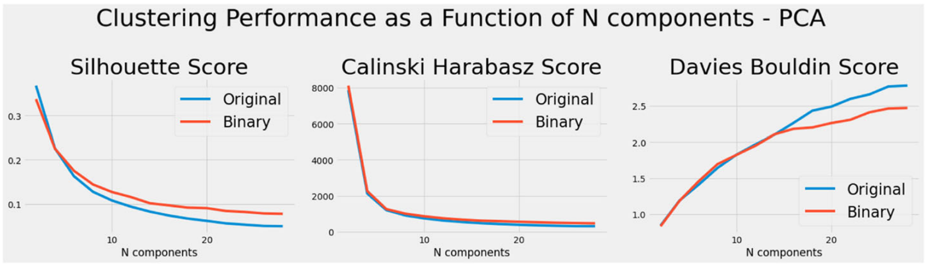
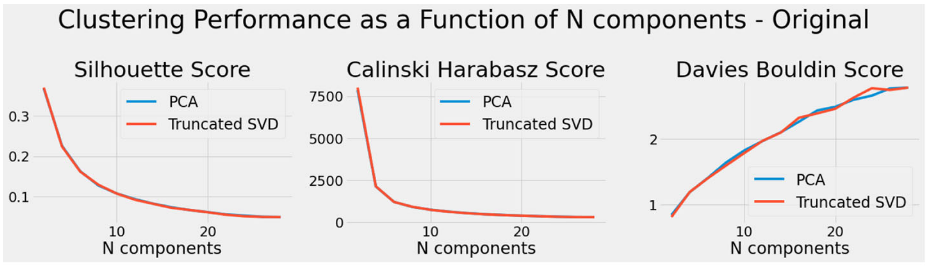
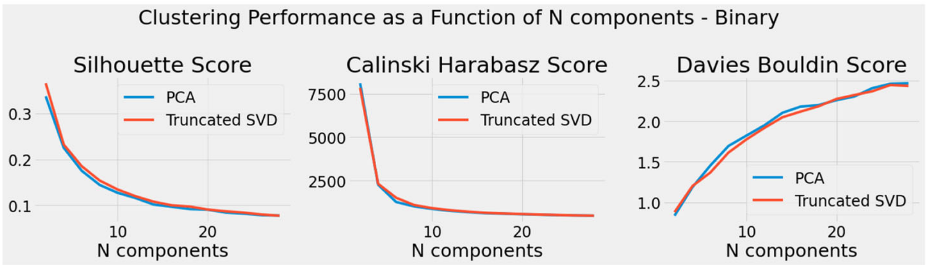


Fig. 3 Input feature reduction for clustering, with an arbitrary number of clusters $N = 10$



(a) Using the Short encoding



(b) Using the Long encoding

Fig. 4 Identifying the proper reduction tool for sparse data, using an arbitrary number of clusters of $N = 10$

5.2 Number of samples

Having identified a suitable dimension reduction tool (Truncated SVD) and value for the number of components to reduce to (N=2), we now seek to find a satisfying number of samples for the clustering.

Figure 5 shows clustering performances as a function of the number of clusters, for various sample sizes. All were obtained when clustering with the Short encoding for feature representation. The blue, orange, green, red, purple and brown curves are for respectively using 100, 500, 1000, 2500, 5000 and 10000 random samples. This range of values enables us to consider small to intermediately large complexity for our proposal.

Overall, all performance metrics points towards the use of large number of clusters. Indeed, the higher the number of clusters, the higher the Silhouette and Calinski-Harabasz

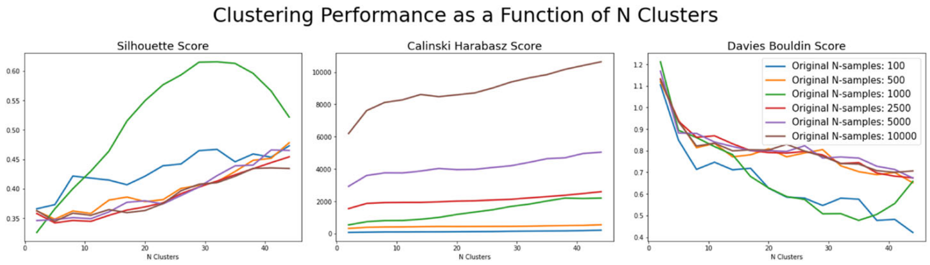


Fig. 5 Identifying the proper sample size, when clustering using Truncated SVD for dimension reduction ($N = 2$ components) and k-means

scores, and the lower the Davies-Bouldin index. These trends are observed for all sample sizes. Moreover, for both the Silhouette score and the Davies-Bouldin index, the plotted functions are of similar values (overlap of clustering score as function of the number of clusters) for most sample sizes. Only the Calinsky-Harabasz index discriminates towards the use of increasingly larger sample sizes.

Given these findings, we identify $N = 10000$ (the largest tested value) as the sample size to use for optimal clustering in future experiments.

5.3 Number of clusters

Next, we look closer into the number of clusters to use, when clustering with various feature representations.

Figure 6 shows clustering performance as a function of the number of clusters. The blue and red curves display the performance results using respectively the Short and the Long encoding. All input features were reduced to two (2) components using Truncated SVD, and the clustering is performed with k-means.

Using both encodings, all performance metrics point towards the use of large number of clusters. The higher the number of clusters, the higher the Silhouette and Calinski-Harabasz scores, and the lower the Davies-Bouldin index. Additionally, the clustering performances seem to reach a plateau for intermediate values of number of clusters between twenty (20) and thirty (30). This is observed when considering the Calinski-Harabasz score and the Davies-Bouldin index, and for both encodings. Then, the performance slightly improves, when the number of clusters exceeds this range. In practice, we found in Section 5.4 that for a queried number of clusters greater or equal to twenty (20), the number of retrieved clusters using the *scikit-learn* toolbox is of $N_{short} = 19$ and $N_{long} = 13$ when using, respectively, the Short and Long Encodings.

Therefore, results suggest using an intermediate- to-large number of clusters for improving the k-means clustering performances, with a preference for the Short encoding. Therefore, we set the number of clusters to $N_{short} = 19$ and $N_{long} = 13$ when using, respectively, the Short and Long Encodings for the following experiments.

5.4 Qualitative cluster analysis

As an additional way to validate the clustering results, we seek to visualize the clusters, and compare them to the natural layout of the reduced data.

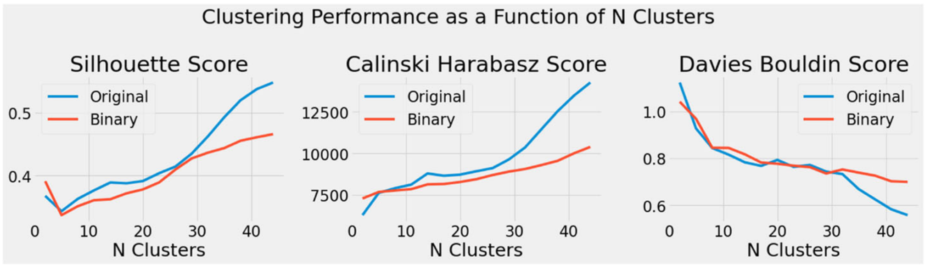


Fig. 6 Identifying the proper number of clusters, using Truncated SVD for dimension reduction ($N = 2$ components) and k-means

Figure 7 depicts visual clustering results for five algorithms: k-means, spectral clustering, DBSCAN, Birch, and BGM. All input features were reduced to two (2) components using Truncated SVD. Plots (a) and (b) display results using the Short and the Long encoding, respectively.

When using the Short encoding (Fig. 7a), the clusters seem to have natural horizontal to diagonal (45 degree) layout. This layout is not well captured by the evaluated algorithms. The BGM seems to provide the most satisfying results, despite little calibration.

When using the Long encoding (Fig. 7b), clusters naturally layout in well separated vertical columns. This is also best captured by BGM.

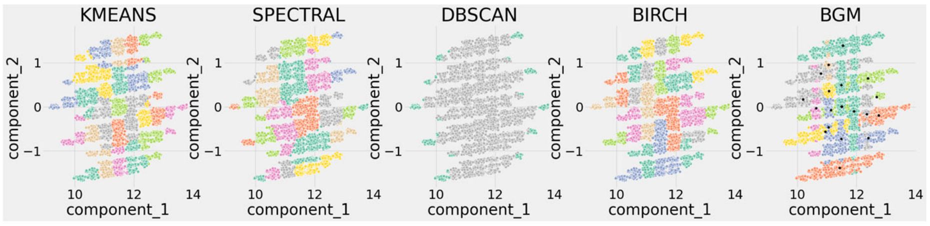
Overall, results suggest using BGM for robust clustering on both feature representations.

5.5 Initialization benchmark

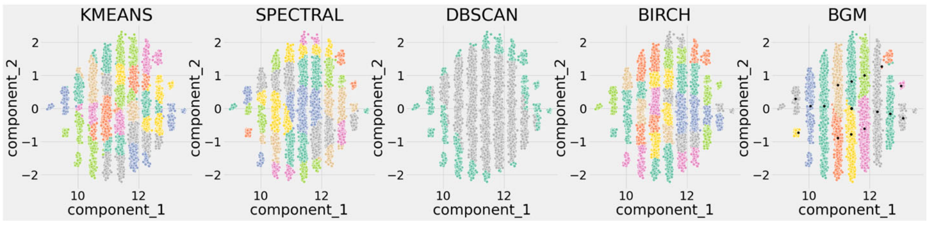
5.5.1 CIFAR-10

In order to assess the quality of the centroids extracted, we use them for initializing the baselines algorithms GA, $(\mu + \lambda)$EA, and *Aging Evolution*.

Figure 8 shows performance in validation for the three search baselines. The color red stands for the random sampling initialization (*rand*), blue for LHS, and green for centroids (i.e., our approach). From left to right, the GA, EA, and Aging Evolution results are plotted. The top row corresponds to 36 epochs of training, and the bottom one to 108 epochs. In all cases, each algorithm is executed 100 independent times. Each plot provides with the mean fitness of the current population (bold), complemented with the range of fitness (min/max). The centroids are initialized considering the Short encoding and BGM previous results. The population size is set to 19 (i.e., the number of centroids) in all cases (refer to Section 5.3).



(a) Using the Short encoding



(b) Using the Long encoding

Fig. 7 Qualitative analysis of clustering for both feature representations. Here we use Truncated SVD for dimension reduction, and $N = 2$ components

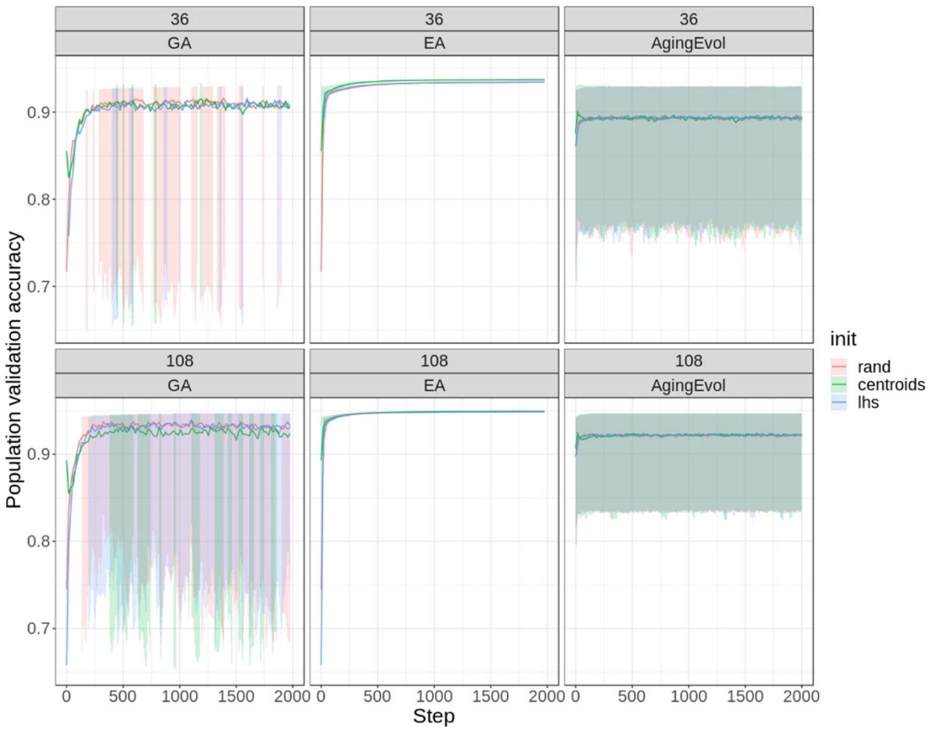


Fig. 8 Performances in validation of various NAS algorithms, when clustering with the Short Encoding

Regarding GA (Algorithm 1), *pop_size*=19, *max_evaluations*=1995 (i.e., population size 19, evolved 104 generations), *mut_p*=0.2, *cx_p*=0.5, and *ind_pb*=0.05. Regarding EA (Algorithm 2), $\mu = \lambda$=19, *mut_p*=0.8, and *ind_pb*=0.1. Regarding Aging Evolution (Algorithm 3), it is run with a tournament size $k = 10$.

For all three baselines, we observe that the centroid-based initialization provides with the highest initial mean population fitness. On the other hand, both LHS and random sampling-based initialization provide with a very low initial mean fitness (up to 20 percentage points of difference). The EA takes the best advantage of this improved initial population: It converges faster and has long-term improvements over an initialization with random sampling or LHS. Both GA and Aging Evolution fail to benefit from such improvements as their mean population fitness plummets after a few iterations, and reaches similar values to those of the alternative initialization techniques (rand or LHS). This is observed when searching either after 36 or 108 epochs of training.

Figure 9 summarizes the benchmark provided in Fig. 8. In particular, it provides box plots of performance in test for the best found solutions (100 runs) after 2000 search evaluations. It also complements the three baseline algorithms, i.e., GA, EA, and Aging Evolution, with random search (RS). The plot on the left corresponds to 36 epochs of training (i.e., the evaluation of the solutions), and the right one to 108 epochs.

Overall, performances in test after deployment (2000 evaluations) are similar to those in validation. Indeed, the ranking is preserved: The EA reaches the highest mean fitness, for all initialization settings. EA and GA have very narrow fitness distributions, while Aging

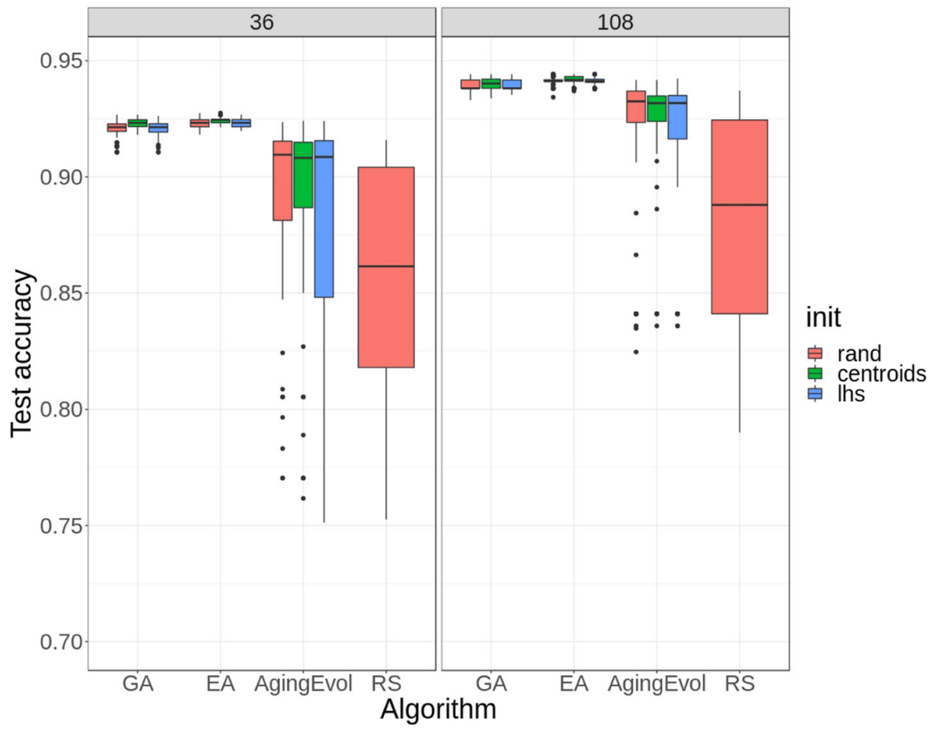


Fig. 9 Benchmark of NAS algorithm performances after 2000 iterations. The search is performed either when training solutions for 36 or 108 epochs. The data-driven initialization techniques involve the Short encoding

Evolution has a more spread one. All the baselines improve over Random Search. For GA and EA, centroids help reach higher mean test fitness over other initialization techniques.

To complement these results, we performed a Wilcoxon rank-sum test. For GA and when selecting after 36 epochs of training, the p-value for the centroid-based initialization versus random sampling is $4.093 \cdot 10^{-7}$. Versus LHS, it is $2.324 \cdot 10^{-7}$. When selecting after 108 epochs, it is 0.69 versus random sampling, and 0.039 versus LHS. For EA and when selecting after 36 epochs of training, the p-value for the centroid-based initialization versus random sampling is $5.611 \cdot 10^{-8}$. Versus LHS, it is $3.767 \cdot 10^{-6}$. When selecting after 108 epochs, it is 0.006 versus random sampling, and 0.006 versus LHS. Thus, the centroid-based initialization significantly improves over LHS and random sampling, for both EA and GA when selecting after 36 epochs of training. For EA, it improves significantly over LHS and random sampling in all training budgets.

Figure 10 depicts the results for Long encoding benchmark. In all cases, we set the pop_size=13 (=$\mu = \lambda$), following the recommendations from Section 5.3. The number of evaluations is set to 1989 (153 generations).

Similarly, centroids obtained considering the Long encoding enable all baseline algorithms to have an improved initial mean population fitness. Also, EA is the best at taking advantage of this initialization (centroids), with an improved convergence, up until 500 to 1000 evaluations.

Figure 11 summarizes the benchmark provided in Fig. 10 with performances in test, in the same fashion as Fig. 9

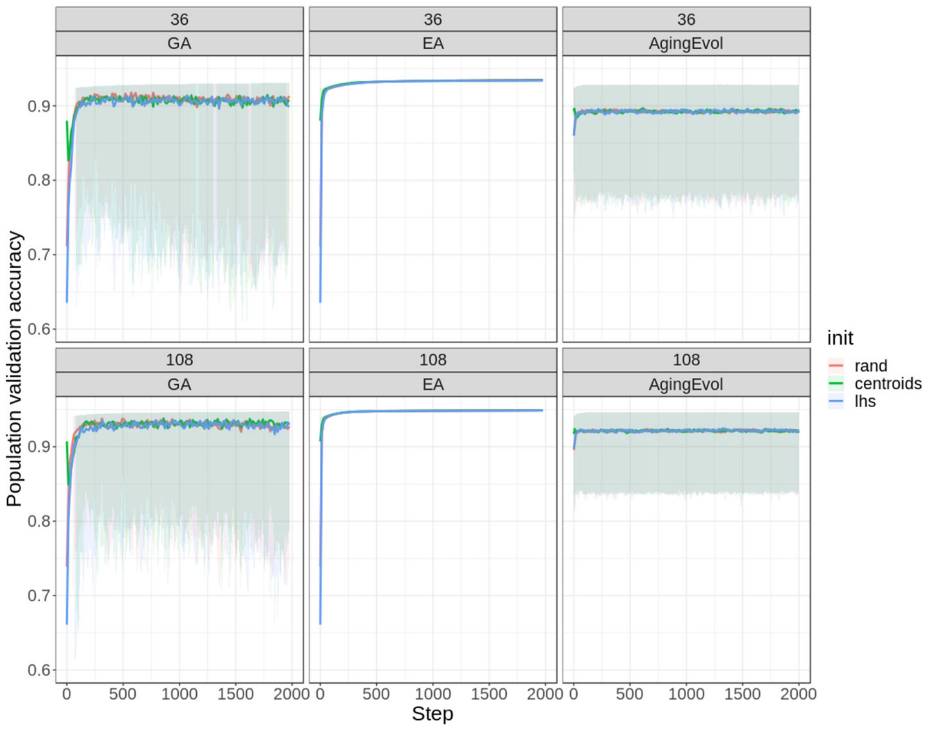


Fig. 10 Performances in validation of various NAS algorithms, when clustering with the Long encoding

Results of performance in test after deployment are similar to those obtained considering the Short encoding for finding the initial population. However, the centroids do not provide with improvements to the final performance of the baseline algorithm. Also, we notice that the centroid-based initialization worsen the distribution of fitness of solutions found by Aging Evolution (i.e., larger variance).

To summarize, the centroids extracted from a fitness-based clustering of the search space seem to be a promising strategy to initialize a population-based search algorithms. We observe improved convergence and long-term performances of EA with a centroid-based initialization, over LHS and rand, when considering the Short encoding. In the case of searching with only 36 epochs of budget, it also helps final test performances for GA (Short encoding).

The limited improvements when clustering with the Long encoding might be explained by the fact that the baselines (EA, GA, and *Aging Evolution*) are deployed on models using the Short encoding. Note that experiments using the Long encoding were discarded because of the increased complexity for the search procedure. Future work might explore this option, as it could help better exploit the extracted population.

5.5.2 So2Sat LCZ-42

Next, we ask ourselves: *can we re-use an initial population provided by the proposed technique, but for other purposes (e.g., datasets)?*

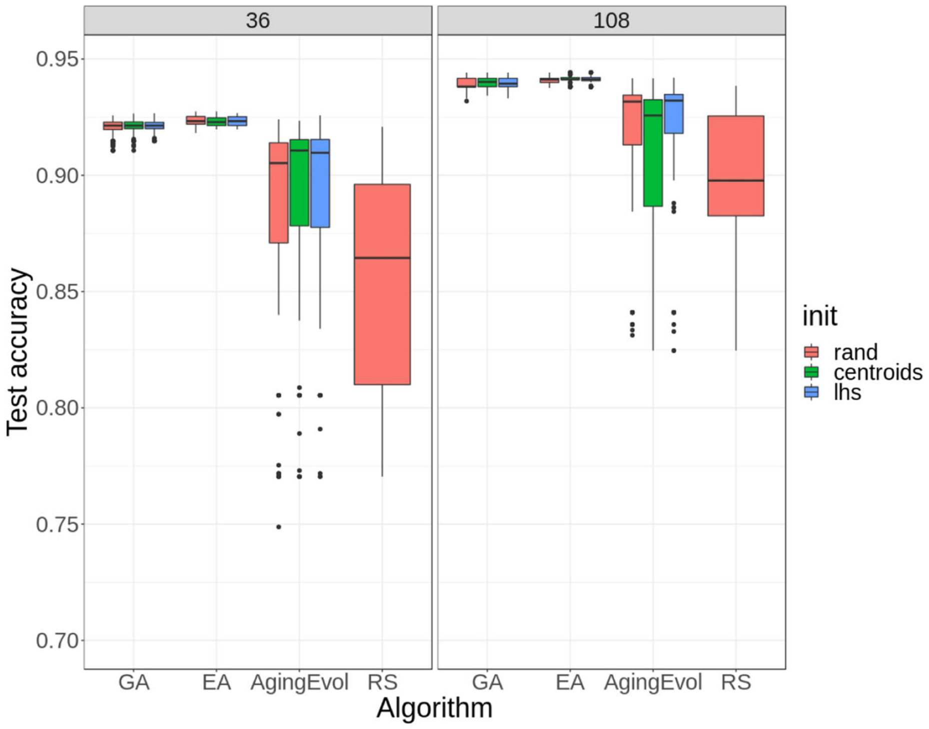


Fig. 11 Benchmark of NAS algorithm performances after 2000 iterations. The search is performed either when training solutions for 36 or 108 epochs. The data-driven initialization techniques involve the Long encoding

To answer this question, we perform experiments using the real-world image classification dataset So2Sat LCZ-42. More precisely, we seek to know if an initial population obtained using as input a sample from NASBench-101, can benefit a search algorithm deployed on the So2Sat LCZ-42 [60]. Given the results obtained in the previous sections and because of practical limitations, we only consider EA for search and random sampling as alternative sampling baseline. Indeed, EA is the algorithm for which the most success is observed using our approach, and random sampling is more stable than LHS. We also limit the number of evaluations to $N_{evaluations} = 100$ and training budget of 4 epochs. Figure 12 shows such results using the Short Encoding, with *pop_size*=19 (=$\mu = \lambda$). Both settings are executed five independent times.

Overall, we observe similar results of convergence on LCZ42 than initially seen on CIFAR-10. Indeed, over all the trials, the mean test accuracy of the initial population provided by the centroids is much larger (about 14 percentage points) than the one obtained by random sampling. Besides, this gap reduces after 1 generation (19 evaluations), but the centroid-based approach remains improving.

To summarize, early results of initializing search on a complementary dataset, indicate that initial population gathered on NASBench-101 also enables to accelerate the convergence of the algorithm on another dataset, even in the case of low training budgets. Since the improvements in fitness increases with more training budget, this suggests that the gap in fitness might even become wider in profit of the proposed method when training longer.

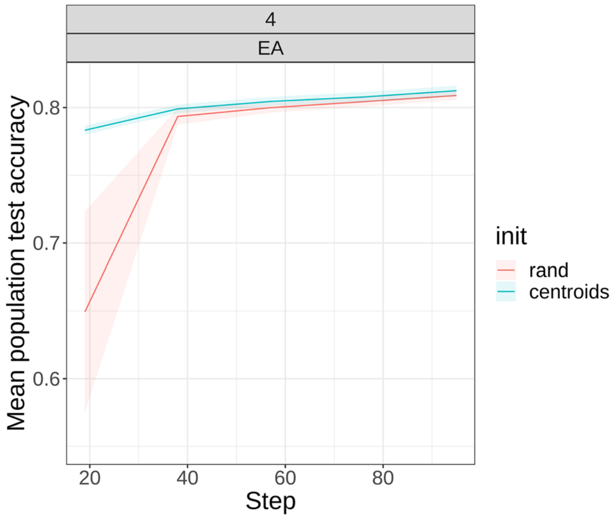


Fig. 12 Performances in test of various initialization techniques used with EA, when clustering with the Short Encoding, on the So2Sat LCZ42. Each curve shows the mean (bold line) and deviation of the performance in test, for its respective setting (five runs)

5.6 Visualization of the solutions found

Last but not least, we look to gain insights into the solutions found by the algorithms deployed in Section 5.5.1.

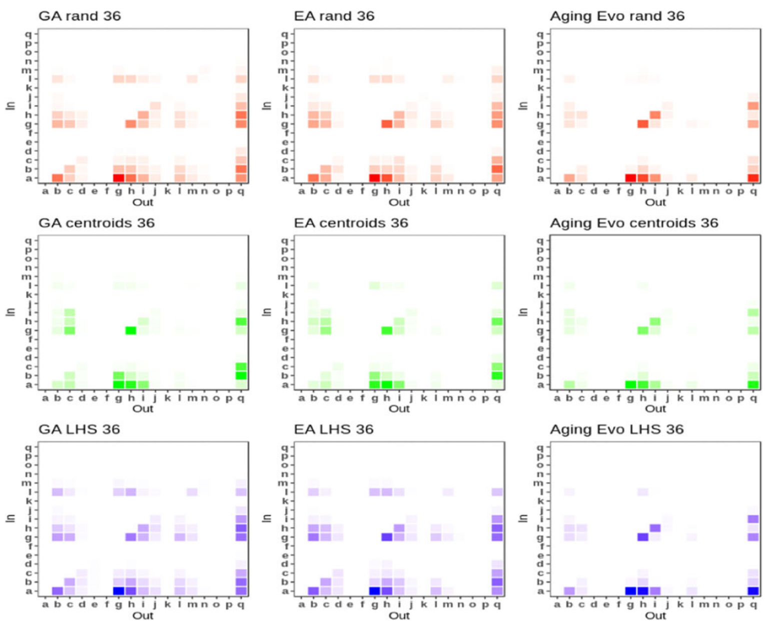
Figure 13 provides a visualization of solutions found (100 independent runs) by the search baselines, for all initialization settings, considering the Short encoding. More precisely, it shows the frequency of connections on the expanded adjacency matrix (100 solutions), for each baseline. The darker, the higher the frequency. Figure 13a and b show results when searching respectively after 36 or 108 epochs of training. From left to right appear results for GA, EA, and Aging Evolution. From top to bottom appear results using as initialization random sampling (rand), centroids, and LHS.

Figure 14 provides the same visualization of solutions found, but considering the Long encoding.

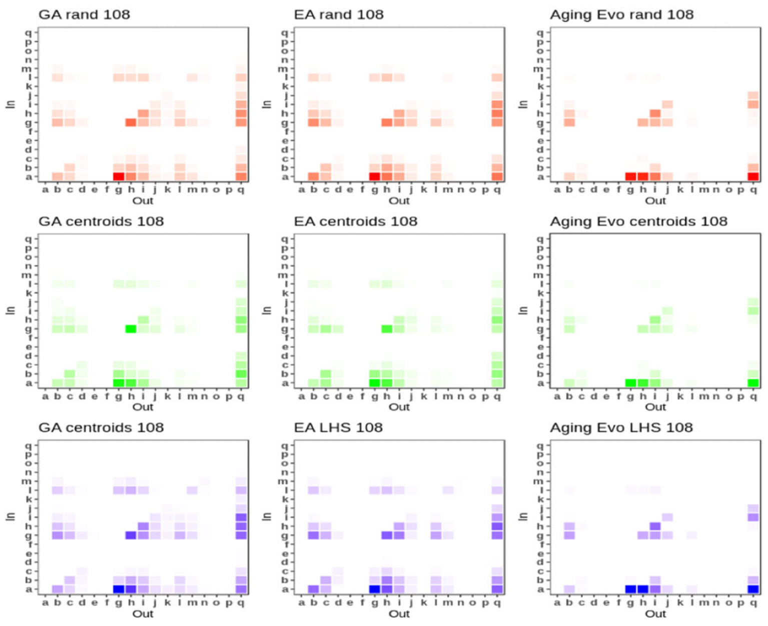
Table 3 provides a legend for the labelling of both Figs. 13 and 14. Note that the matrices shown describe distributions of elementary *feed-forward modules* that are DAGs with *nodes*, *edges* and *node labels*. In each figure, all adjacency matrices are labelled using seventeen characters ranging from a to q, where each reffers to either the input node (resp. a), the output node (resp. q), or one of the five possible intermediated nodes (at most) and its node label. Then, the remaining letters are reffering to the intermediated nodes being tagged with either the first (resp. b to f), the second (resp. g to k) or third (resp. l to p) node label. Section 4.1 and Fig. 2 offer explainations on the construction of such matrices.

Overall, the connections gathered from the solutions found after 36 epochs of training differ from those found after 108 epochs. In the first case, the *activations* on the adjacency matrices have clusters that are more restricted, as opposed to the more widespread and larger clusters obtained when searching after 108 epochs of training.

Besides, we also observe a difference in the output based on the algorithm used to find the solutions. EA and GA provide solutions whose connections are overall similar, in the form

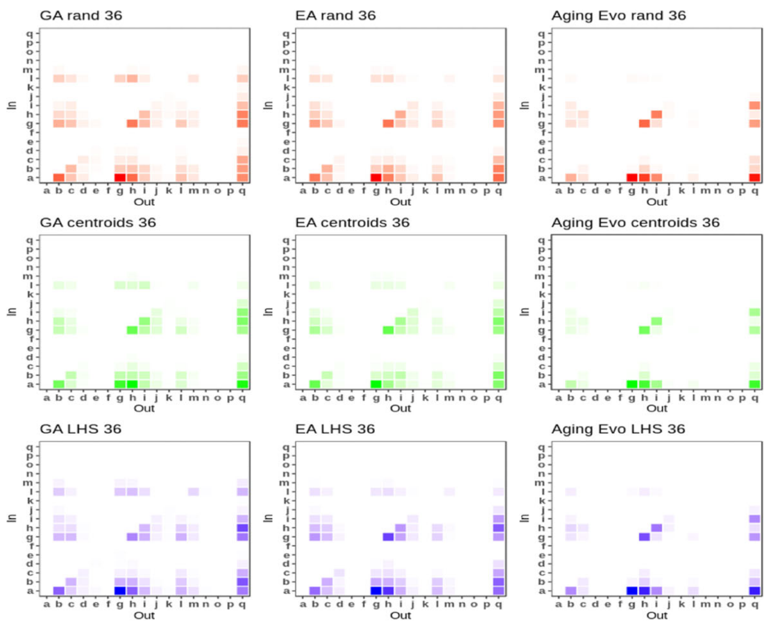


(a) With a budget of 36 epochs

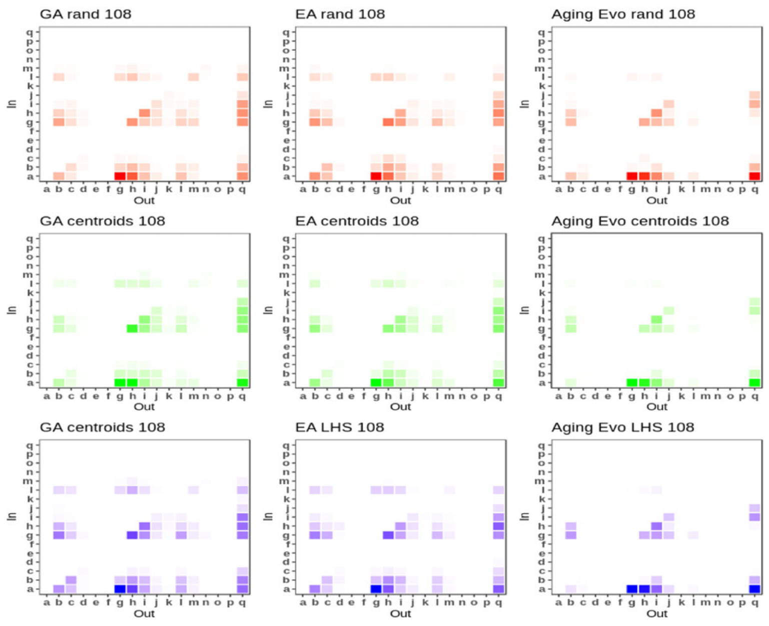


(b) With a budget of 108 epochs

Fig. 13 Visualization of solutions found (N=100) considering the Short Encoding



(a) With a budget of 36 epochs



(b) With a budget of 108 epochs

Fig. 14 Visualization of solutions found (N=100) considering the Long Encoding

Table 3 Legend for Figs. 13 and 14: Each character is associated to a combination of *node label* and *node index* according to the search-space of the NASBench-101 database

Figure label	Node Label	Node index
a	input	0
b	Conv 1x1 + BatchNorm + Relu	1
c	Conv 1x1 + BatchNorm + Relu	2
d	Conv 1x1 + BatchNorm + Relu	3
e	Conv 1x1 + BatchNorm + Relu	4
f	Conv 1x1 + BatchNorm + Relu	5
g	Conv 3x3 + BatchNorm + Relu	1
h	Conv 3x3 + BatchNorm + Relu	2
i	Conv 3x3 + BatchNorm + Relu	3
j	Conv 3x3 + BatchNorm + Relu	4
k	Conv 3x3 + BatchNorm + Relu	5
l	MaxPool 3x3 + BatchNorm + Relu	1
m	MaxPool 3x3 + BatchNorm + Relu	2
n	MaxPool 3x3 + BatchNorm + Relu	3
o	MaxPool 3x3 + BatchNorm + Relu	4
p	MaxPool 3x3 + BatchNorm + Relu	5
q	output	6

of widespread clusters. On the other had, the Aging Evolution has patterns of connections in its cells that are regrouped and in slightly smaller cluster.

Furthermore, we analyzed the solutions based on the initialization technique used when deploying search. Across all settings, it appears to be more diverse solutions (on average more activated cell in adjacency matrices) obtained via LHS and random sampling, than for a centroid-based initialization.

To summarize, the longer the training allowed when selecting models, the more diverse are the solutions retrieved. Also, EA and GA tend to find more diverse solutions than Aging Evolution. When it comes to initialization, the centroids-based approach results in solutions that are more similar to each other, with matrices of adjacency that are less activated.

We find that the patterns highlighted in this section correlate with the findings of the authors in [41]. In the study, the authors show that on the search space of NASBench-101, the longer the training the more narrow the fitness distribution with most solutions having close to the top fitness after 108 epochs of training. They also showed that the fitness landscape becomes flat, with many local optima. Therefore, when searching with a training budget of 108 epochs and a fixed number of iterations, a search algorithm is likely to retrieve more diverse solutions than after 36 epochs, since most of them satisfy the criterion of high fitness.

When it comes to the differences based on the algorithm to be used, this could be explained both the very rugged landscape (many local maxima) and the nature of the algorithms. Indeed, as Aging Evolution provides with non-diverse sets of solutions, which could be explained by it being stuck in local maxima and not diversifying enough, i.e., discarding *old* solutions.

Regarding the centroids, Section 5.5 already shows that they consist of an initial population of particularly high average fitness, with little variance. This could be explained by the centroids being potential local maxima of high fitness and very diverse nature, since coming from distinct clusters.

6 Conclusion

In this study, we seek to gain insights about a search space of image classification models in order to improve the performance of NAS algorithms. More precisely, we want to know if the convergence of a search strategy could be improved using a data-driven initialization technique exploiting the search space.

For this purpose, we propose a novel approach to improve the performances of a NAS search strategy. First, we perform a clustering analysis of the search space, involving a sequence of sub-tasks. It summarizes as follows: we sample models from a search space, reduce their dimension, perform a clustering. After a careful tuning of the clustering pipeline (number of dimensions, clusters, etc.), we select the algorithm providing the best qualitative and quantitative results. Second, we extract and use the centroids as an initial population to a search strategy.

We validate our proposal by initializing three (3) evolutionary algorithms, namely a genetic algorithm (GA), an evolutionary algorithm (EA), and Aging Evolution (AE), and benchmark our data-driven initialization method against conventional initialization baselines, i.e., random initialization and Latin Hypercube Sampling (LHS). To test the algorithms, we query the dataset of NAS-Bench-101, providing with a search space of image classifiers and their fitness evaluation on CIFAR-10. Our results show that centroids extracted using BGM for clustering are a promising approach to initialize a population-based algorithm. In the scenario of selecting models trained only 36 epochs, this approach used with GA shows significant long-term improvements (after 2000 iterations, in test) over random initialization and LHS, when using a Short encoding. When used with EA, it shows faster convergence (in validation) and significant long-term improvements over random initialization and LHS, when using a Short encoding and for all training budgets. Additional investigations on the distributions of the solutions found by the algorithms suggest that centroids enable retrieving local optima (maxima) of high fitness and similar configurations. Besides, we also investigate how an initial population gathered with our proposal on a tabular benchmark can be used for augmenting search on a real world problem, the So2Sat LCZ-42 scene classification dataset. Our early results show that this set of solutions help accelerate the convergence of EA on the target dataset, despite a short training budget.

Moreover, the cost of the approach lies in the size of the sample to collect (10k individuals), to serve as input to the initialization technique. More precisely, the computationally costly steps are the training and evaluation of the sample, while the sampling and clustering only require a few minutes to run. We argue that this drawback (computational cost) can be alleviated by using tabular or surrogate NAS benchmarks for obtaining free fitness evaluations of the sample. We demonstrate that once collected, the sample help initialiaze search for other applications. In order words, this cost can either be avoided (use of available benchmarks) and or limited (transfer to other applications).

As future work, we propose to investigate performances of this approach when selecting models on the Long Encoding. We also propose to study in depth the obtained clusters to gain more insights on obtained performances , and to explore different sampling strategies to select the models for the clustering. One might also explore the benefits of such data-driven initialization method on other families of algorithms (Bayesian optimization, local search, etc.).

Acknowledgements Authors acknowledge support by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. [ERC-2016-StG-714087], Acronym: *So2Sat*), by the Helmholtz Association through the Framework of Helmholtz AI [grant number: ZT-I-PF-5-01] - Local Unit "Munich Unit @Aeronautics, Space and Transport (MASTr)" and Helmholtz Excellent Professorship "Data Science in Earth Observation - Big Data Fusion for Urban Research"(W2-W3-100), by the German Federal Ministry of Education and Research (BMBF) in the framework of the international future AI lab "AI4EO – Artificial Intelligence for Earth Observation: Reasoning, Uncertainties, Ethics and Beyond" (Grant number: 01DD20001), the grant DeToL, and a DAAD Research fellowship.

Funding Open Access funding enabled and organized by Projekt DEAL. The funding is stated in Acknowledgment section.

Data Availability https://github.com/kalifou/data-driven-initialization-to-search.

Code Availability https://github.com/kalifou/data-driven-initialization-to-search.

Declarations

Consent for Publication All authors have checked the manuscript and have agreed to the submission.

Conflict of Interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Haykin, S.: Neural Networks and Learning Machines, vol. 3. Pearson Upper Saddle River, Hoboken (2009)
2. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436 (2015)
3. Elsken, T., Metzen, J.H., Hutter, F., et al.: Neural architecture search: A survey. J. Mach. Learn. Res. **20**(55), 1–21 (2019)
4. Ojha, V.K., Abraham, A., Snášel, V.: Metaheuristic design of feedforward neural networks: a review of two decades of research. Eng. Appl. Artif. Intel. **60**, 97–116 (2017)
5. Hanxiao, L., Karen, S., Yiming, Y.: Darts: Differentiable architecture search. International Conference on Learning Representations (2019)
6. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 4780–4789 (2019). https://doi.org/10.1609/aaai.v33i01.33014780
7. Camero, A., Wang, H., Alba, E., Bäck, T.: Bayesian neural architecture search using a training-free performance metric. Applied Soft Computing 107356 (2021)
8. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Proceedings of the 24th International Conference on Artificial Intelligence. IJCAI'15, pp. 3460–3468. AAAI Press (2015)
9. Camero, A., Toutouh, J., Alba, E.: Low-cost recurrent neural network expected performance evaluation. arXiv:1805.07159 (2018)
10. Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: NAS-bench-101: Towards reproducible neural architecture search. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 7105–7114. PMLR, Long Beach (2019). http://proceedings.mlr.press/v97/ying19a.html
11. Dong, X., Yang, Y.: Nas-Bench-201: Extending the scope of reproducible neural architecture search. In: International Conference on Learning Representations (ICLR) (2020)

12. Back, T.: Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, Oxford (1996)
13. Holland, J.H.: Outline for a logical theory of adaptive systems. J. ACM (JACM) **9**(3), 297–314 (1962)
14. Hutter, F., Kotthoff, L., Vanschoren, J.: Automated Machine Learning - Methods, Systems, Challenges. Springer, Berlin (2019)
15. Engel, J.: Teaching Feed-Forward neural networks by simulated annealing. Complex Systems **2**, 641–648 (1988)
16. Montana, D.J., Davis, L.: Training feedforward neural networks using genetic algorithms. Proceedings of the 11th International Joint Conference on Artificial intelligence **1**(89), 762–767 (1989)
17. Alba, E., Aldana, J., Troya, J.: Genetic algorithms as heuristics for optimizing ANN design. In: Artificial Neural Nets and Genetic Algorithms, pp. 683–690. Springer, Berlin (1993)
18. Alba, E., Aldana, J., Troya, J.M.: Full automatic ann design: A genetic approach. In: International Workshop on Artificial Neural Networks, pp. 399–404. Springer (1993)
19. Yao, X.: A review of evolutionary artificial neural networks. Int. J. Intell. Syst. **8**(4), 539–567 (1993)
20. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
21. Camero, A., Toutouh, J., Alba, E.: Random error sampling-based recurrent neural network architecture optimization. Eng. Appl. Artif. Intel. **103946**, 96 (2020)
22. Zhining, Y., Yunming, P.: The genetic convolutional neural network model based on random sample. Int. J. u-and e-Service Sci. Technol. **8**(11), 317–326 (2015)
23. Rosa, G., Papa, J., Marana, A., Scheirer, W., Cox, D.: Fine-Tuning Convolutional Neural Networks Using Harmony Search. In: Pardo, A., Kittler, J. (eds.) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, pp. 683–690. Springer, Cham (2015)
24. Van Stein, B., Wang, H., Bäck, T.: Automatic configuration of deep neural networks with parallel efficient global optimization. In: 2019 International Joint Conference on Neural Networks (IJCNN), pp. 1–7. IEEE (2019)
25. Ororbia, A., ElSaid, A., Desell, T.: Investigating recurrent neural network memory structures using neuro-evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 446–455. ACM (2019)
26. 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, Netherlands (2019)
27. Wang, C., Xu, C., Yao, X., Tao, D.: Evolutionary generative adversarial networks. IEEE Trans. Evol. Comput. **23**(6), 921–934 (2019)
28. Yang, A., Esperança, P.M., Carlucci, F.M.: Nas evaluation is frustratingly hard. In: International Conference on Learning Representations (2020). https://openreview.net/forum?id=HygrdpVKvr. Accessed 01 April 2022
29. Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: Nas-bench-101: Towards reproducible neural architecture search. In: International Conference on Machine Learning, pp. 7105–7114. PMLR (2019)
30. Zela, A., Siems, J., Frank, H.: Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. International Conference on Learning Representations (2020)
31. Siems, J., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., Hutter, F.: Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. arXiv:2008.09777 (2020)
32. Klyuchnikov, N., Trofimov, I., Artemova, E., Salnikov, M., Fedorov, M., Burnaev, E.: NAS-Bench-NLP: Neural architecture search benchmark for natural language processing. arXiv:2006.07116 (2020)
33. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International Conference on Machine Learning, pp. 4095–4104. PMLR (2018)
34. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Smash: one-shot model architecture search through hypernetworks. arXiv:1708.05344 (2017)
35. Camero, A., Toutouh, J., Alba, E.: Comparing deep recurrent networks based on the mae random sampling, a first approach. In: Conf of the Spanish Association for Artificial Intelligence, pp. 24–33. Springer (2018)
36. Lin, M., Wang, P., Sun, Z., Chen, H., Sun, X., Qian, Q., Li, H., Jin, R.: Zen-nas: a zero-shot nas for high-performance image recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 347–356 (2021)
37. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with Analog Genetic Encoding. In: Parallel Problem Solving from Nature-PPSN Ix, Pp, pp. 671–680. Springer, Berlin (2006)

38. Ning, X., Zheng, Y., Zhao, T., Wang, Y., Yang, H.: A generic graph-based neural architecture encoding scheme for predictor-based nas. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16, pp. 189–204. Springer (2020)
39. Chu, X., Zhang, B., Ma, H., Xu, R., Li, Q.: Fast, accurate and lightweight super-resolution with neural architecture search. In: 2020 25th International Conference on Pattern Recognition (ICPR), pp. 59–64. IEEE (2021)
40. Nunes, M., Fraga, P.M., Pappa, G.L.: Fitness landscape analysis of graph neural network architecture search spaces. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '21, pp. 876–884. Association for Computing Machinery (2021). https://doi.org/10.1145/3449639.3459318
41. Traoré, K.R., Camero, A., Zhu, X.X.: Fitness Landscape Footprint: A Framework to Compare Neural Architecture Search Problems (2021)
42. Zhang, T., Lei, C., Zhang, Z., Meng, X.-B., Chen, C.P.: As-nas: Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning. IEEE Transactions on Evolutionary Computation (2021)
43. Maaranen, H., Miettinen, K., Mäkelä, M.M.: Quasi-random initial population for genetic algorithms. Comput. Math.. Appl. **47**(12), 1885–1895 (2004)
44. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.: Quasi-oppositional differential evolution. In: 2007 IEEE Congress on Evolutionary Computation, pp. 2229–2236. IEEE (2007)
45. Clerc, M.: Initialisations for particle swarm optimisation Online at http://clerc.maurice.free.fr/pso. Accessed 01 April 2022 (2008)
46. Helwig, S., Wanka, R.: Theoretical analysis of initial particle swarm behavior. In: International Conference on Parallel Problem Solving from Nature, pp. 889–898. Springer (2008)
47. Kazimipour, B., Li, X., Qin, K.: A review of population initialization techniques for evolutionary algorithms. In: Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014 (2014). https://doi.org/10.1109/CEC.2014.6900618
48. Kazimipour, B., Li, X., Qin, A.K.: Initialization methods for large scale global optimization. In: 2013 IEEE Congress on Evolutionary Computation, pp. 2750–2757. IEEE (2013)
49. Kimura, S., Matsumura, K.: Genetic algorithms using low-discrepancy sequences. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, pp. 1341–1346 (2005)
50. Morrison, R.W.: Dispersion-based population initialization. In: Genetic and Evolutionary Computation Conference, pp. 1210–1221. Springer (2003)
51. Ma, Z., Vandenbosch, G.A.: Impact of random number generators on the performance of particle swarm optimization in antenna design. In: 2012 6th European Conference on Antennas and Propagation (EUCAP), pp. 925–929. IEEE (2012)
52. Poles, S., Fu, Y., Rigoni, E.: The effect of initial population sampling on the convergence of multi-objective genetic algorithms. In: Multiobjective Programming and Goal Programming, pp. 123–133. Springer, Berlin (2009)
53. Mousavirad, S.J., Bidgoli, A.A., Rahnamayan, S.: Tackling deceptive optimization problems using opposition-based de with center-based latin hypercube initialization. In: 2019 14th International Conference on Computer Science & Education (ICCSE), pp. 394–400. IEEE (2019)
54. Medeiros, H.R., Izidio, D.M., Ferreira, A.P.D.A., Da, S., Barros, E.N.: Latin hypercube initialization strategy for design space exploration of deep neural network architectures. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 295–296 (2019)
55. Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., Gagné, C.: DEAP: Evolutionary Algorithms made easy. J. Mach. Learn. Res. **13**, 2171–2175 (2012)
56. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
57. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. **20**, 53–65 (1987). https://doi.org/10.1016/0377-0427(87)90125-7
58. Caliński, T., Harabasz, J.: A dendrite method for cluster analysis. Commun. Stat. **3**(1), 1–27 (1974). https://doi.org/10.1080/03610927408827101
59. Davies, D.L., Bouldin, D.W.: A cluster separation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-1(2) 224–227 (1979)
60. Zhu, X.X., Hu, J., Qiu, C., Shi, Y., Kang, J., Mou, L., Bagheri, H., Haberle, M., Hua, Y., Huang, R., et al.: So2sat lcz42: a benchmark data set for the classification of global local climate zones [software and data sets]. IEEE Geosci. Remote Sens. **8**(3), 76–89 (2020)