LS-PON: a Prediction-based Local Search for Neural Architecture Search

Meyssa Zouambi, Julie Jacques, Clarisse Dhaenens

LS-PON: a Prediction-based Local Search for Neural Architecture Search

Meyssa Zouambi[1][0000−0003−2484−3179], Julie Jacques[1,2][0000−0001−6260−9629], and Clarisse Dhaenens[1][0000−0002−6590−7215]

[1] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
{meyssa.zouambi, clarisse.dhaenens}@univ-lille.fr
[2] Faculté de Gestion, Economie et Sciences, Lille Catholic University, Lille, France
julie.jacques@univ-catholille.fr

Abstract. Neural architecture search (NAS) is a subdomain of AutoML that consists of automating the design of neural networks. NAS has become a hot topic in the last few years. As a result, many methods are being developed in this area. Local search (LS), on the other hand, is a famous heuristic that has been around for many years. It is extensively used for optimization problems due to its simplicity and efficiency. LS has a lot of advantages in the world of NAS; it can naturally exploit methods that accelerate the global search time such as weight inheritance and network morphism. LS is also easy to implement and does not require a complex encoding or any parameter tuning. In the present work, we aim at making LS faster by guiding the exploration of the neighborhood. Our objective is to limit the number of solution evaluations, which are particularly time-consuming in NAS. We propose the method LS-PON (Local Search with a Predicted Order of Neighbors) that uses linear regression models to order the exploration of neighbors during the search. LS-PON, unlike other prediction-based NAS methods, requires neither pre-sampling nor tuning. Our experiments on popular NAS benchmarks show that LS-PON keeps the simplicity and advantages of LS while being as efficient in quality as state-of-the-art methods and can be more than twice as fast as classical LS.

Keywords: Neural architecture search · Local search · Optimization · Machine learning.

1 Introduction

Neural architecture search has attracted a lot of attention in recent years. Researchers aim to develop the most efficient algorithms to automate the time-consuming task of architecture design. Although many complex methods were proposed in the past few years, the necessity of having such complex algorithms is sometimes questioned [28]. Local search is a simple heuristic that proved its efficiency in the field of NAS. Recent works that studied local search for NAS proved that it is competitive with state-of-the-art methods [24, 3]. On top of its easy implementation, LS has a lot of advantages in this area. First, it does

not require a complex encoding or any parameter tuning. It can also easily exploit strategies that accelerate global search time like weight inheritance [20] and network morphism [23].

Our work focuses on using machine learning to speed up local search for NAS. The objective is to keep all of the benefits that LS offers while being more time-efficient. We achieve this speed-up by ordering the exploration of neighbors of a solution using performance predictors. Evaluating architectures in NAS is the most time-consuming step of the process, so to be efficient, it is worth visiting the most promising neighbors first. We rely on machine learning to predict the ranking of a solution based on its accuracy and determine the neighborhood order. Unlike most prediction-based NAS methods, this one does not require any pre-sampling or parameter-tuning. The method starts with a random ordering and progressively gives more accurate rankings. The experimental results show that LS-PON can be more than two times faster than classical LS while achieving similar state-of-the-art performances.

The contributions of our work are summarized as follows:

- We create a simple and parameter-free method for NAS based on a local search and machine learning. This method is easy to implement and requires neither pre-sampling nor parameter-tuning.
- We use three different NAS benchmarks (with small-scale and large-scale search spaces) to validate the performance of our method. We confirm the competitiveness of LS and show that our method gives similar results while being significantly faster in almost all cases.

The remaining of this paper is organized as follows: related works are presented in section 2. Section 3 provides a brief description of neural architecture search and local search. Section 4 details the proposed approach. Section 5 is dedicated to the experiments. It presents the benchmarks, the experimental protocol, and the results obtained. Section 6 gives the conclusion of our findings.

2 Related Works

Very few works investigated the efficiency of local search for NAS. In [24], authors explore the theoretical characterization of the landscape and its effect on the performance of local search. They proved that LS is a very competitive method when the noise in the evaluation pipeline is reduced to a minimum. Within this setting, they demonstrate that hill-climbing -the most known form of LS- can outperform many popular state-of-the-art algorithms on the popular NAS benchmark datasets. These results are confirmed in [3], where LS is employed in a multi-objective context. It shows that local search competes with state-of-the-art evolutionary algorithms, even up to thousands of evaluated architectures in the multi-objective setting. LS is therefore a method that is very easy to implement yet yields competitive performances against more complex algorithms.

Local search was also used in conjunction with network morphism as seen in [6, 12]. Network morphism is a popular method to rapidly search efficient

convolutional neural networks [23]. This technique allows the expansion of the network using function preserving operations and prevents training the resulting architectures from scratch. LS is a natural way of exploiting this method since its resulting architecture generates "neighbors" of the current solution.

Another way to speed up the search is to avoid the computationally expensive network training by using performance predictors [7]. Performance predictors help to identify good architectures using their characteristics only. They vary from simple decision trees to deep neural networks [18, 15, 1]. Recent works, however, opt for using complex models to accurately represent the huge number of possible architectures of the search space [22, 14]. These methods usually require a pre-sampling step to gather enough architecture-performance pairs for building the prediction model. To evaluate these NAS approaches, it is necessary to consider the training time required to sample and train architectures for the predictor, as well as the design and tuning of its hyperparameters. The latter task can be considered as counterproductive in this context, especially if the prediction models used are neural networks.

In [26], an interesting perspective is given on performance predictors. Authors state that using multiple simple prediction models at different stages of the search can be more efficient than using a single complex predictor. They emphasize that the goal of NAS is to sample the best architectures, and most of the solutions in the search space will not be evaluated at all. So it is not necessary for a predictor to accurately estimate the performance of all of them. This work iteratively creates weak predictors to determine which architecture is the best in the current subset of the search space. This aspect of locality is important for prediction. As this work demonstrates, solutions close to each other in a search space are more likely to fit well using a simple predictor.

3 Background

Before proceeding to the proposed method, we provide in this section a brief overview of neural architecture search and local search.

3.1 Neural Architecture Search

To design an efficient neural network, many parameters need to be taken into account, such as the number of layers, the type of operations to use, the hyperparameters linked to each type of operation, etc. This leaves us with a lot of potential models that perform differently based on their architecture.

The goal of neural architecture search is to automatically find an architecture a among a set of architectures $\mathcal{A}$, that achieves the best objective value. Usually, the purpose is to minimize the error on the validation dataset after training the network on the training dataset.

Formally, we can express the NAS problem as follows:

$$\arg\min_{a \in \mathcal{A}} = \mathcal{L}\left(a, \mathcal{D}_{\mathrm{t}}, \mathcal{D}_{\mathrm{v}}\right)$$

$\mathcal{A}$ defines the search space, it contains all potential neural architectures. $\mathcal{L}(\cdot)$ is the cost function that measures the error of the architecture a on the validation dataset $\mathcal{D}_{\text{v}}$ after being trained on the training dataset $\mathcal{D}_{\text{t}}$.

Due to the large search space of possible architectures in NAS, many search strategies were developed to explore it more efficiently. The most popular NAS strategies are gradient-based approaches [16], evolutionary algorithms [17], and reinforcement learning strategies [9]. During the search, architectures are sampled and evaluated. The classical way of evaluating an architecture is to train it using data from a training set and then assess its performance on a validation set. This task is time-consuming and is considered to be the bottleneck of most NAS algorithms. For this reason, many techniques were proposed for estimating the performance of neural networks without having to fully train them. Such methods are weight sharing [27], network morphism [6, 12], weight inheritance [20], and neural predictors [25].

3.2 Local Search

Local search is a popular heuristic used to approximately solve NP-hard optimization problems. It starts from an initial solution s_0, chosen at random or by using another heuristic. It then generates neighbors of this solution by applying a *neighborhood* function N. This function applies small changes to the current solution to create neighboring ones that are close to it. Different ways of exploring the neighborhood lead to different variants of local search. The heuristic evaluates these neighbors using a cost function f that assesses their quality. It substitutes the current solution s with a better one from $N(s)$. After this update, it reiterates the process until convergence. The search stops when no neighbor is better than the current solution, so we can no longer improve it (the heuristic reaches a local optimum).

The most popular form of LS is called *hill-climbing*. In this form, the search updates the current solution with a better one from the neighborhood. Several exploration strategies can be chosen: the *first-improve* updates the current solution with the first improving neighbor found. The *best-improve* strategy updates the solution after evaluating all neighbors and picking the one that improves it the most. The *worst-improve* strategy evaluates all neighbors and chooses the one that improves the current solution the least.

In our work, we will be using hill-climbing, with the first-improve strategy, which allows exploring only a subset of the neighbors of each solution, as our purpose is to evaluate the least number of architectures for a faster search.

4 Proposed Approach

In the following, we define the important components of the proposed method and its process. It is important to note that the solution encoding, neighborhood function, and solution evaluation highly depend on the NAS task. In this paper, we focus on image classification (using NAS benchmarks) but the global idea of the proposed approach can be adapted to other types of tasks as well.

4.1 Solution Encoding

In the context of NAS, a solution s defines an architecture of a neural network. A representation that can be translated to a neural network can be used to encode it. In this work, the used encoding is a list of categorical, discrete, and/or continuous values for representing an architecture. This list can specify the type of operations of each node in the network, the connections between these nodes, the hyperparameters used for each operation, etc. This representation in the form of a list of values is similar to an entry of tabular data. It allows to directly use it as a dataset for building machine learning predictors later in this work.

Figure 1, shows a simplified example of a CNN encoding. On the top, the encoding is represented, which is the list of operations applied to the data and their hyperparameters (note that other hyperparameters not mentioned in the encoding are fixed and not optimized during the search). On the bottom, there is the corresponding CNN with its operations. In our work, we use three different NAS benchmarks, NAS-Bench-201 [4], MacroNasBenchmark [3], and NAS-Bench-301 [21]. Each benchmark defines its own set of operations and their corresponding parameters. This gives a different number of possible solutions for each of them, which defines the size of the search space.

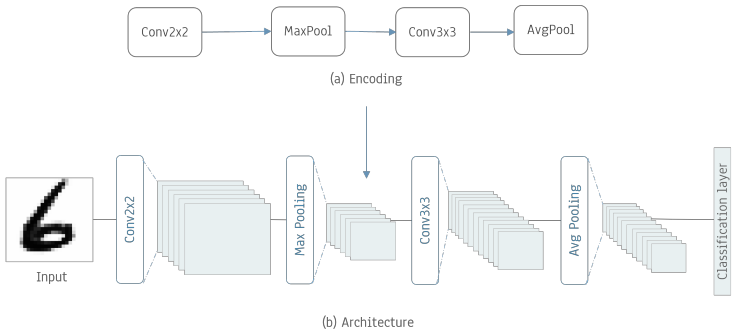


Fig. 1: Encoding of an architecture.

4.2 Neighborhood Function

The neighborhood function N generates a neighborhood $N(s)$ that contains the architectures close to s. These architectures can defer by a single element such as an operation or an edge from the initial solution. In our work, we use the *one exchange neighborhood* as defined in [8].

In the studied NAS benchmarks, variables are all categorical. Hence, a neighbor is obtained by modifying a single variable. The neighborhood of a solution is the set of solutions obtained by selecting one by one each variable and enumerating all the possible values. The number of neighbors for each solution in NAS-Bench-201, MacroNasBenchmark, and NAS-Bench-301 are respectively 24,

28, and 136 neighbors. The size of the neighborhood is relatively small, but let us recall that the evaluation of a solution is very costly.

Figure 2 shows an example of a neighbor generation. On top, there is the current solution. Generating one neighbor consists in choosing one operation and changing it by another, for example here the max pooling operation is replaced with a convolution of 3x3 kernel.

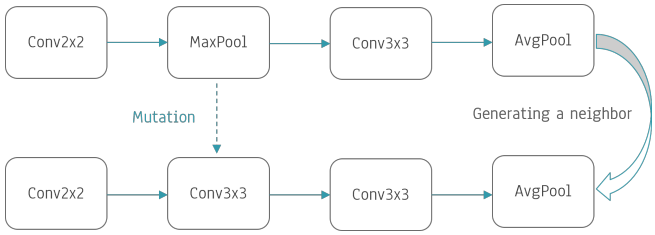


Fig. 2: Neighbor generation with *One exchange neighborhood* function.

4.3 Solution Evaluation

Evaluating an architecture is the most time-consuming step in NAS. It requires training the architecture using a training set and assessing its performance using a validation set.

Depending on the task, architecture, and hardware used, this step can take several hours for a single evaluation. Therefore, in this work, we use NAS benchmarks that provide surrogate performance metrics on both the training set and validation set for all possible architectures. Since these benchmarks deal with image classification, their evaluation is based on the classification accuracy and is calculated as the sum of well-classified images divided by the total number of images of the set.

4.4 Performance Prediction

In a normal setting, the local search engine evaluates neighbors in a random order. The proposed method, however, orders these neighbors to evaluate the most promising ones first. This order relies on performance predictions made with linear regression models. We choose to work with linear regressions, as they are simple, easy to implement, and do not require parameter tuning. This choice is also based on the work presented in [26], which states that using simple predictors is sufficient to estimate the performances of architectures that are close to each other.

As a reminder, linear regression is a method that assumes a linear relationship between a set of variables $X = (x_1, \ldots, x_p)$ and an output variable y as follows:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

Coefficients $\beta = (\beta_0, \ldots, \beta_p)$ are learned by minimizing the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

In this work, the linear models are trained on architecture-performance pairs. The y variable is the performance we want to predict using the x_i variables which are the list of values describing an architecture. Note that in this work, the predicted performance is a score corresponding to the ranking of a solution (the normalized value of its rank). Since the encoding contains categorical data, we use one-hot-encoding to create a binary column for each category and use it as a numerical value for the linear regression.

Our method does not require any pre-sampling to build the dataset for architecture-performance pairs. The first models created can be random, which is equivalent to using a random order for the neighbors. Each architecture evaluated during the search, is added to a history database. They will be used in future iterations for creating more accurate prediction models.

4.5 LS-PON Process

Our LS-PON algorithm is an improved version of hill-climbing specifically designed for NAS. As we mentioned earlier, hill-climbing updates the current solution as soon as it finds a better one in the neighborhood. For this reason, the speed of this method relies on the order in which the neighbors are evaluated. The algorithm progresses more quickly if we assess the performance of the most promising neighbors first. Then, the method is more likely to immediately improve the current solution and move on with the search.

To determine the best order in which to evaluate neighborhoods, we use linear regression models. The models do not need to be sophisticated or accurate to yield good performances, but just good enough to provide an approximate ranking of the best neighbors quickly.

The process of LS-PON is illustrated in Figure 3 and works as follows: after choosing an initial solution, the method generates the neighborhood of this architecture in step one (1). In step two (2), the method creates a linear regression model to predict the performance of architectures based on their parameters. It predicts the ranking of the neighbors and evaluates them in that order in the third step (3). Each evaluated architecture gets added to the database of architecture-performance pairs (step 4). This database will later be used to create a new (more accurate) linear model in the next iteration. If the solution is not better than the current one, it moves to the next one in the neighborhood (step 5a), else, it updates the current solution and reiterates the process (step 5b). If the search can no longer improve the current solution and the max budget of evaluations is not reached, it samples a new random solution and restarts the search. Algorithm 1 gives a detailed description of the proposed method.

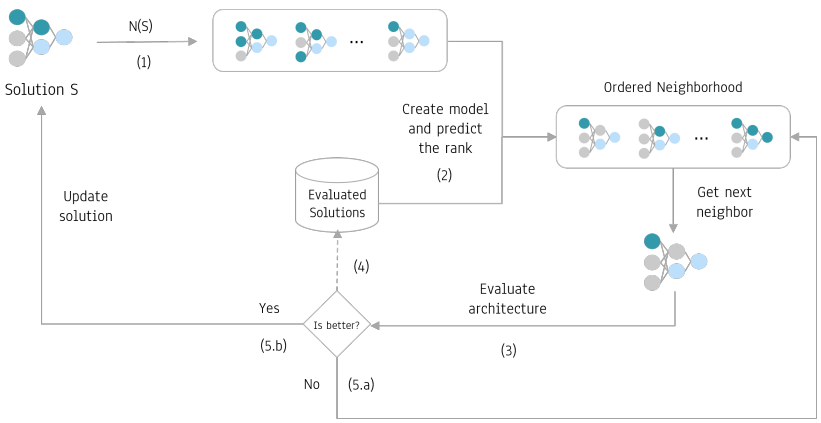


Fig. 3: Main steps of LS-PON.

Algorithm 1 Local search with a predicted order of neighbors - LS-PON

Input: $\mathcal{A}$: search space
N: neighborhood function
$maxBudget$: maximum number of evaluations
Initialization
Randomly pick an architecture $a \in \mathcal{A}$
$acc\_a, best\_acc \leftarrow Evaluate(a)$
$D \leftarrow \cup\{(a, acc\_a)\}$
$nbEval \leftarrow 1$
while $nbEval \leq maxBudget$ **do**
  Create prediction model M and train it using D
  Predict rank of each $u \in N(a)$ using M
  Order $N(a)$ based on the predicted ranks
  for $u \in$ Ordered $N(a)$ **do**
    $acc \leftarrow Evaluate(u)$
    $D \leftarrow D \cup \{(u, acc)\}$
    $nbEval \leftarrow nbEval + 1$
    if $acc \geq best\_acc$ **then**
      Update current solution: $a \leftarrow u$
      Update best score: $best\_acc \leftarrow acc$
      Exit for loop
    end if
    if $nbEval > maxBudget$ **then**
      return best architecture
    end if
  end for
  if best solution not updated, randomly sample a new one and continue searching
end while
Output: Best architecture found

5 Experiments

To test the performance of the proposed algorithm, three different benchmarks are used: NAS-Bench-201 [4], NAS-Bench-301 [21], and MacroNASBenchmark [3]. The choice of the first two benchmarks is to test the algorithm in a cell-based search space, with a small-scale and a large-scale setting. The last benchmark is for assessing the performance in a macro search space. The difference between the two types of search spaces can be found in [5]. In the following, a detailed description of each benchmark is given. This section also presents the experimentation protocol as well as the results and their discussion.

5.1 Benchmark Details

NAS-Bench-201 [4] The NAS-Bench-201 benchmark consists of 15,625 unique architectures, representing the search space. An architecture of the NAS-Bench-201 search space consists of one repeated cell. This cell is a complete directed acyclic graph (DAG) of 4 nodes and six edges. Each edge can take one of the five available operations (1×1 convolution, 3×3 convolution, 3×3 avg-pooling, skip, no connection), which leads to 15,625 possible architectures. For each of these architectures, precomputed training, validation, and test accuracies are provided for CIFAR-10, CIFAR-100 [10], and ImageNet-16-120 [2]. CIFAR and ImageNet [11] are the most popular datasets for image classification. Two solutions from this benchmark are neighbors if they differ by exactly one operation in one of their edges. Thus the number of neighbors is 24.

MacroNASBenchmark [3] The MacroNASBenchmark is a relatively large-scale benchmark with more than 200k unique architectures. It is based on a macro-level search consisting of 14 unrepeated modules. Each module can take one of three options (Identity, MBConv with expansion rate 6 and kernel size 3x3, MBConv with expansion rate 6 and kernel size of 5x5). For each architecture, precomputed validation and test accuracies are provided for CIFAR-10 and CIFAR-100. A neighboring solution in this benchmark is a solution that differs by one module from the current solution.

NAS-Bench-301 [21] The NAS-Bench-301 is a surrogate benchmark based on the DARTS [16] search space for the CIFAR-10 dataset. This popular search space for large-scale NAS experiments contains 10^{18} architectures. An architecture of the DARTS search space consists of two repeated cells, a convolutional cell and a reduction cell, each with six nodes. The first two nodes are input from previous layers, and the last four nodes can take any DAG structure such that each node has degree two. Each edge can take one of the eight possible operations. In this benchmark, the classification accuracies of the architectures on the CIFAR-10 datasets are estimated through a surrogate model, removing the constraint to evaluate the entire search space. A neighboring solution is a solution that differs by an edge connection or an operation from the current solution.

5.2 Experimentation Protocol

The experimentation protocol used to evaluate our algorithm goes as follows:

For NAS-Bench-201 and MacroNasBenchmark, we use the validation accuracy as the search metric. Hence, the reference will be the best validation accuracy (Optimal) provided by the benchmark. We compare the algorithm to a standard local search, and a random search. For NAS-Bench-201, we further compare the results to Regularized Evolution Algorithm (REA) [19], which is the best-achieving algorithm reported on the NAS-Bench-201 paper [4]; results on REA are taken as-is from this paper. We set the maximum number of evaluations for these two benchmarks to 1500 evaluated architectures.

For NAS-Bench-301 [21], we report the validation accuracy given by the surrogate model provided in the benchmark. In the literature, experiments conducted on this search space suggest that the best architectures perform around 95% of validation accuracy [21]. As previously, we compare our algorithm against a local search and a random search. Since it has a larger search space than the previous two benchmarks, we set the maximum number of evaluations to 3000 architectures.

For the prediction models, we use the linear regression model from the Scikit-learn library v0.23 (*sklearn.linear_model.LinearRegression* with default parameters).

For each benchmark, all experiments are averaged over 150 runs. LS and LS-PON start with the same initial solution in every run. Note that the algorithms will restart after converging as long as they have not exhausted the maximum number of evaluations budget. We compare the algorithms from different standpoints: the mean accuracy, the convergence speed, and the dynamic of each method.

5.3 Results

Table 1 reports the validation accuracy for random search (RS), local search (LS), and LS-PON on the three benchmarks. It also reports the REA [19] method for NAS-Bench-201 for comparison. We notice that LS and LS-PON give state-of-the-art results in all benchmarks. Finding either optimal or close to optimal accuracy in all cases. With LS-PON slightly surpassing LS in some cases. Both methods significantly surpass random search. For REA, despite being the best-reported algorithm on the NAS-Bench-201 paper, random search still outperforms it in all available datasets. This could suggest that the hyperparameters of REA are too specific to the NAS problem it was designed for and did not generalize well. This also confirms that random search is a strong baseline in NAS [13].

Since both algorithms, LS and LS-PON, give similar performances on data quality, to further compare them, we need to analyze their speed. Indeed, in a non-benchmark setting, each sampled architecture needs to be trained. Training an architecture takes significantly more time than running a local search. For

Table 1: Performance evaluation on the three benchmarks. Each algorithm uses the validation accuracy as a search signal. *Optimal* indicates the highest validation accuracy provided by the benchmark. We report the mean and std deviation of 150 runs for Random search, LS, LS-PON. † taken from [4].

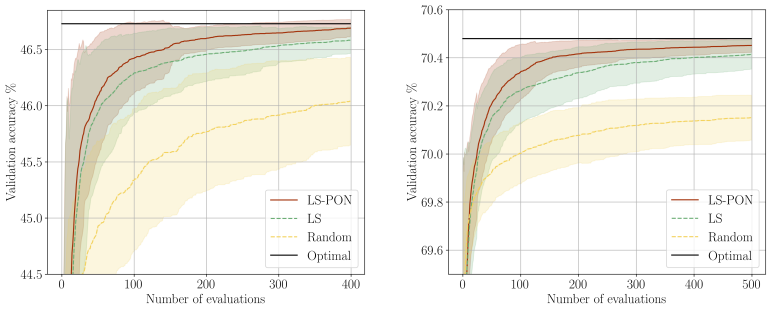
	NAS-Bench-201			**MacroNasBench**		**NAS-Bench-301**
Method	**Cifar10**	**Cifar100**	**ImageNet**	**Cifar10**	**Cifar100**	**Cifar10**
REA†	91.19±0.31	71.81±1.12	45.15±0.89	–	–	–
RS	91.48±0.09	72.93±0.38	46.39±0.23	92.22±0.06	70.22±0.08	94.52±0.08
LS	**91.61±0.00**	**73.49±0.00**	46.72±0.03	92.46±0.04	70.45±0.02	**95.11±0.05**
LS-PON	**91.61±0.00**	**73.49±0.00**	**46.73±0.00**	**92.48±0.02**	**70.47±0.02**	**95.11±0.06**
Optimal	**91.61**	**73.49**	**46.73**	**92.49**	**70.48**	**≈95**

Table 2: Speed evaluation on the three benchmarks. The table indicates the mean and std deviation of the number of evaluated architectures before convergence. Values in bold mean statistically better (Wilcoxon's test).

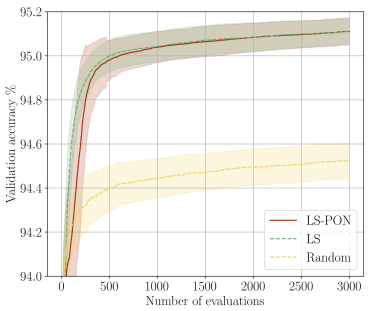
	NAS-Bench-201			**MacroNasBench**		**NAS-Bench-301**
Method	**Cifar10**	**Cifar100**	**ImageNet**	**Cifar10**	**Cifar100**	**Cifar10**
LS	76±54	70±48	617±389	566±414	628±393	**1499±859**
LS-PON	**64±54**	**44±23**	**276±159**	**426±346**	**483±417**	1588±857
Speedup	**18%**	**59%**	**123%**	**32%**	**30%**	**-6%**

this reason, the number of sampled architectures is a strong indicator of the speed of each method.

Table 2 reports the mean and standard deviation of the number of evaluated architectures during the search. Both LS and LS-PON restart after convergence as long as the evaluation budget is not reached. Since both algorithms end with almost similar accuracies, we use a *leveled* convergence for a fair speed comparison. We calculate how many evaluations were required for each algorithm to reach the same final accuracy. Since random search never surpasses these two methods after exhausting all of its evaluation budget (1500 for the first two benchmarks and 3000 for the last one) it is omitted from this table. The acceleration obtained using LS-PON is reported in the last line of the table. It shows that in both NAS-Bench-201 and MacroNasBenchmark, there is a significant speedup ranging from 18% to up to 123%. In NAS-Bench-301, however,



(a) NAS201-IMNT (b) MNBench-C100



(c) NAS301-C10

Fig. 4: Example of the evolution of the validation accuracy across the number of evaluated architectures for a dataset of each benchmark. NAS201-IMNT, MNBench-C100, and NAS301-C10 correspond to NAS-Bench-201 for ImageNet16-120, MacroNasBenchmark for Cifar100, and NAS-Bench-301 for Cifar10. Results are averaged for 150 runs for all algorithms. The shaded region indicates the standard deviation of each search method.

LS-PON shows less efficiency with a slower convergence time. Figure 4 gives examples that illustrate the evolution of the validation accuracy across the number of evaluated architectures for a different dataset in each benchmark. It shows how LS-PON is faster than LS in NAS-Bench-201 and MacroNasBenchmark for ImageNet16-120 (IMNT) and Cifar100 respectively, and how it is slightly less efficient in NAS-Bench-301 for its Cifar10 dataset.

Table 3 further shows the dynamic of these methods. It reports the number of evaluated neighbors before improving on the current solution. In NAS-Bench-201 and MacroNasBenchmark, it requires from 2 times to 3 times the number of evaluations for LS to find a better neighbor compared to LS-PON. For this reason, LS-PON progresses more quickly during the search. In NAS-Bench-301, however, it takes around 11% more neighbors evaluation for LS-PON to find a better neighbor compared to LS, which makes LS-PON slightly slower in this case.

Table 3: Results on all benchmarks. The table indicates the mean and std deviation of the number of required evaluations before improving on the current solution during the search. Values in bold mean statistically better (Wilcoxon's test)

.

	NAS-Bench-201			MacroNasBench		NAS-Bench-301
Method	**Cifar10**	**Cifar100**	**ImageNet**	**Cifar10**	**Cifar100**	**Cifar10**
LS	4.28±4.14	4.29±4.10	4.20±4.10	5.07±5.54	5.20±5.68	**17.33±25.31**
LS-PON	**1.87±2.32**	**1.69±1.99**	**1.61±1.71**	**2.35±3.02**	**2.28±2.77**	19.33±29.60

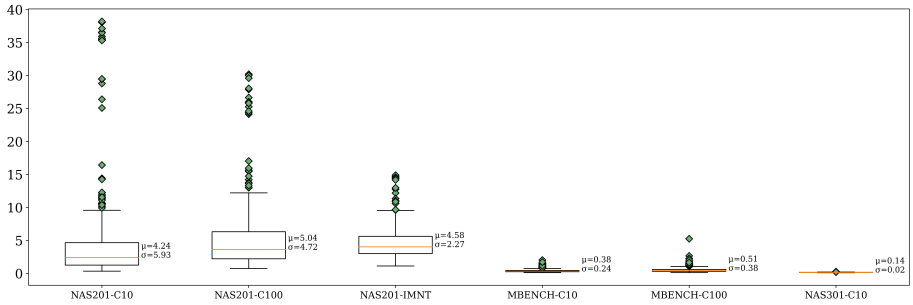


Fig. 5: Box plots represent the distribution of distances between the validation accuracy of a solution and its neighbors, and diamonds represent outliers. NAS201, MNBench, NAS301 stand for NAS-Bench-201, MacroNasBenchmark and NAS-Bench-301. C10, C100 and IMNT are respectively Cifar10, Cifar100 and ImageNet16-120.

To investigate the reason behind this, and understand the effectiveness of this method in the different benchmarks/datasets, an analysis on the distance between the current solution's accuracy and its neighbors accuracy is conducted. This analysis seeks to determine if the difference between the neighbors accuracies affects the method's effectiveness. To do this, the mean absolute error between the current solution accuracy, and its neighbors accuracy is calculated for a sample of solutions. Results of this are represented in Figure 5. We see that for the three datasets in NAS-Bench-201, there is a notable difference between the neighbors performances. Hence, the predictor can more easily classify them and identify the best ones. This difference is less visible on MacroNasBenchmark but the benchmark still has a certain number of outliers (represented by diamonds in the Figure) that can be easily recognized by the predictor.

On the other hand, in NAS-Bench-301's dataset, the difference between the neighbors is negligible (there is a mean of 0.14% difference in their accuracy)

and there are also not many noticeable outliers to recognize during the search. The search space of this benchmark is mostly composed of good architectures with very close performances as presented in their paper [21].

This shows that the method is more efficient if the improvement to make is relatively observable. Which is expected in problems where the search space contains a diverse set of solutions.

6 Conclusion

In this paper, we introduced LS-PON, an improved local search based on performance predictors for neighborhood ordering. This method is fast, does not require any hyper-parameter tuning, is easy to implement, and yields state-of-the-art results on three popular NAS benchmarks.

LS-PON proved that it can be more than twice as fast as the LS without the ordering mechanism. Its effectiveness relies on the diversity of solutions in the search space and works better if there is an observable difference between neighbors.

In the future, we will test other types of predictors and analyze their impact on the results. We also aim at making this method more robust to search spaces that mostly contain solutions with very close performances.

Another interesting addition would be to apply this method in a multi-objective context, and see if it scales well with the growing objectives of architecture design, such as the size of the network, the energy consumption, the inference times, etc.

References

1. B. Baker, O. Gupta, R. Raskar, and N. Naik. Accelerating neural architecture search using performance prediction. *arXiv:1705.10823*, 2017.
2. P. Chrabaszcz, I. Loshchilov, and F. Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv:1707.08819*, 2017.
3. T. Den Ottelander, A. Dushatskiy, M. Virgolin, and P. A. Bosman. Local search is a remarkably strong baseline for neural architecture search. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 465–479. Springer, 2021.
4. X. Dong and Y. Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv:2001.00326*, 2020.
5. T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search. pages 69–86.
6. T. Elsken, J.-H. Metzen, and F. Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv:1711.04528*, 2017.
7. T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, pages 1997–2017, 2019.
8. F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

9. Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur. Reinforcement learning for neural architecture search: A review. *Image and Vision Computing*, pages 57–66, 2019.
10. A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.
11. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.
12. A. Kwasigroch, M. Grochowski, and M. Mikolajczyk. Deep neural network architecture search using network morphism. In *International Conference on Methods and Models in Automation and Robotics*, pages 30–35. IEEE, 2019.
13. L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, pages 367–377. PMLR, 2020.
14. Y. Li, M. Dong, Y. Wang, and C. Xu. Neural architecture search in a proxy validation loss landscape. In *International Conference on Machine Learning*, pages 5853–5862. PMLR, 2020.
15. C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision*, pages 19–34, 2018.
16. H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv:1806.09055*, 2018.
17. Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
18. R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, and T.-Y. Liu. Accuracy prediction with non-neural model for neural architecture search. *arXiv:2007.04785*, 2020.
19. E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, pages 4780–4789, 2019.
20. E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.
21. J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv:2008.09777*, 2020.
22. C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, and J. Liang. Npenas: Neural predictor guided evolution for neural architecture search. *arXiv:2003.12857*, 2020.
23. T. Wei, C. Wang, Y. Rui, and C. W. Chen. Network morphism. In *International Conference on Machine Learning*, pages 564–572. PMLR, 2016.
24. C. White, S. Nolen, and Y. Savani. Exploring the loss landscape in neural architecture search. *arXiv:2005.02960*, 2020.
25. C. White, A. Zela, B. Ru, Y. Liu, and F. Hutter. How powerful are performance predictors in neural architecture search? *arXiv:2104.01177*, 2021.
26. J. Wu, X. Dai, D. Chen, Y. Chen, M. Liu, Y. Yu, Z. Wang, Z. Liu, M. Chen, and L. Yuan. Weak nas predictors are all you need. *arXiv:2102.10490*, 2021.
27. L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, L. Wang, Z. Chen, A. Xiao, J. Chang, X. Zhang, et al. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *ACM Computing Surveys (CSUR)*, pages 1–37, 2021.
28. K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann. Evaluating the search phase of neural architecture search. *arXiv:1902.08142*, 2019.