Evaluating Anytime Performance on NAS-Bench-101

Carlos Vieira
IMD, Universidade Federal do Rio Grande performancedo Norte
Natal, RN, Brazil
carlosv@ufrn.edu.br

Leslie Pérez Cáceres
Escuela de Ingeniería Informática Pontificia Universidad Católica de Valparaíso
Valparaíso, Chile
leslie.perez@pucv.cl

Leonardo C. T. Bezerra
IMD, Universidade Federal do Rio Grande do Norte
Natal, RN, Brazil
leobezerra@imd.ufrn.br

2021 IEEE Congress on Evolutionary Computation (CEC) | 978-1-7281-8393-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/CEC45853.2021.9504902

***Abstract*—Neural architecture search (NAS) is a field where computational effort poses a significant challenge, requiring large computing clusters and specialized hardware. Furthermore, the lack of common experimental guidelines often compromises NAS comparison or induces premature conclusions. In a recent work, NAS-Bench-101 was proposed to help mitigate those factors, providing both a common benchmark and experimental guidelines for its use. In this work, we discuss the design choices in NAS-Bench-101 and propose improvements that increase the potential of the benchmark. First, we bridge NAS and the research on anytime performance, showing how a bi-objective formulation of NAS can improve the insights provided by NAS-Bench-101. Then, we discuss choices made in the design of the benchmark, namely (i) the fixed-size encoding, (ii) the effects of the limited variability available, (iii) the assessment of algorithms only from a TPU time perspective, and; (iv) the number of repetitions proposed. We demonstrate our contributions assessing the best-performing algorithms originally benchmarked on NAS-Bench-101 and also irace, one of the best-performing algorithm configurators from the literature. Results indicate that (i) the anytime performance methodology enriches the insights obtained from the assessment on the original NAS-Bench-101; (ii) algorithm comparison is strongly affected by the design choices discussed, and; (iii) the performance of SMAC in this benchmark is significantly improved by our alternative setups.**

I. Introduction

Neural architecture search (NAS) is likely the most promising field in automated machine learning (AutoML) when the data to be modeled is unstructured. Recent NAS breakthroughs include challenging application domains, such as computer vision and natural language processing [1]–[3]. These promising applications have evidenced the need for intelligent approaches to mitigate the significant computational effort required for the experimental campaigns involved in the development of NAS techniques. The development of representative benchmarks that enable fast and standardized experimentation is thus a need in the NAS field. NAS-Bench-101 [4] is a repository of benchmarked convolutional neural networks for the CIFAR-10 [5] dataset that can be used by NAS researchers when evaluating or proposing algorithms. NAS-Bench-101 enables the evaluation of neural architectures in negligible time, providing NAS developers with a fast test suite, and avoiding the duplication of computational effort when evaluating architectures. Besides reusability, NAS-Bench-101 is also an effort to standardize NAS performance assessment since comparability between different works in the field is in general limited.

In this work, we analyze the design choices in NAS-Bench-101 and how these relate to the performance insights provided by the benchmark. We propose to enrich the insights provided by NAS-Bench-101 in three significant ways. First, we formulate NAS as a bi-objective problem comprising solution quality and computational resources consumed. This anytime formulation [6] extends NAS-Bench-101 final quality approach to an approach that compares the performance dynamics of algorithms in a Pareto-compliant way. Second, we discuss other experimental design choices that underlie NAS-Bench-101 such as (i) fixing the number of vertices, which we believe could restrict the potential of NAS algorithms; (ii) limiting the variability included in the benchmark, which may lead to a weak representation of a NAS scenario, and; (iii) assessing NAS algorithms solely based on TPU time, overlooking the overhead of the search process of the algorithms. Finally, we revisit the experimental guidelines suggested by NAS-Bench-101; specifically, the large number of repetitions suggested to benchmark NAS algorithms.

An experimental assessment of high-performing algorithms complements our theoretical discussion. In more detail, we consider two of the best-performing algorithms identified in the original NAS-Bench-101 assessment, namely regularized evolution (RE [1]) and SMAC [7]. We also include irace [8] in this assessment, an algorithm configurator known to have state-of-the-art performance on the optimization of algorithm design for solution quality [9]. Performing a more reasonable number of repetitions than in the extensive campaign adopted in the original NAS-Bench-101 assessment, we are able to reproduce the same conclusions obtained from the original final-quality performance analysis of RE and SMAC. Furthermore, results show that the algorithm, number of vertices, variability degree, and stopping criterion are interacting factors, confirming our design choice discussion's relevance.

Even if the algorithms assessed have been configured for final-quality performance, our anytime assessment using a bi-objective optimization methodology enriches the analysis of NAS-Bench-101. Specifically, RE is outperformed by the remaining algorithms in the final-quality analysis, but it outperforms irace regarding anytime performance. We further inspect differences between algorithms with empirical attainment function (EAF [10]) difference plots and observe that RE and

SMAC are incomparable under the original NAS-Bench-101 setup. Regarding the latter, SMAC consistently outperforms the remaining algorithms both concerning final-quality and anytime performance. We believe these results further support our argument that NAS-Bench-101 assessment should account for NAS computation time, given that the time required by a SMAC execution is considerably larger compared to the time required by a run of RE or irace.

We conclude our work with an analysis of the high-performing architectures selected by the NAS algorithms. Interestingly, some of these do not include pooling layers, even if convolutional layers do not constrain feature map dimensions. We believe this result highlights that a bi-objective performance formulation should be discussed not only at NAS level, but also at model training. Though NAS-Bench-101 provides temporal snapshots of the training process, only the final accuracy of the models is used when assessing an architecture. Thus, costly architectures may be deemed equivalent to much faster ones, clearly an undesirable behavior.

The remainder of this paper is structured as follows. Section II briefly reviews background concepts related to this work, namely neural architecture search, the most relevant NAS algorithm classes and their performance on NAS-Bench-101, and anytime optimization. Next, Section III presents our theoretical discussion, in particular the bi-objective formulation for NAS-Bench-101 and potential impact of benchmark design choices on both final quality and anytime performance. Section IV details the final quality results observed for the selected algorithms with regards to the previously discussed benchmark characteristics. The anytime performance of the selected algorithms is discussed in Section V. We then conduct an analysis on high-performing architectures in Section VI. Last, we conclude and discuss future work in Section VII.

II. Background

In this work, we bridge the research on NAS and anytime optimization. For context, we define NAS, its application to computer vision, and the inner works of NAS-Bench-101. Next, we detail the most relevant NAS algorithm classes, and discuss their performance on NAS-Bench-101. Finally, we review how a bi-objective formulation of optimization problems can model anytime performance.

A. Neural Architecture Search (NAS)

From a high-level perspective, NAS comprises the design and configuration of neural networks [11]. In the context of deep learning, NAS is being instrumental to challenging fields such as computer vision [1]–[3]. Given the computational overhead it poses, different mitigation approaches have been considered, such as designing architecture cells that are replicated to become larger networks [2], [3]. This is the approach followed by NAS-Bench-101, a benchmark of convolutional networks for CIFAR-10 [5]. Below, we detail the most important characteristics of this benchmark:

Design space. A cell is modeled as a directed acyclic graph (DAG), where each node represents a neural network layer (convolutional or pooling). The cell contains up to five layers (not counting input and output layers). Solutions are encoded as (i) an adjacency matrix, which defines the topology of the cell, and; (ii) a node label list, which determines the types of layers that will be employed.

Architecture evaluation. Performance metrics are provided for different stopping criteria run on tensor processing unit (TPU) clusters. Besides TPU time, queries to the API provided training, validation, and test accuracy. Results are sampled from three different seeds for variability.

B. NAS Algorithms and Their Performance on NAS-Bench-101

Three major classes of NAS algorithms can be identified in the literature [11], namely evolutionary algorithms (EAs), algorithm configurators (AC), and neural networks. In [4], a set of relevant algorithms from each class were applied and compared on NAS-Bench-101. The algorithms were compared using as stopping criterion the total TPU time available for the search process ($10^7 s$). In that comparison, multi-fidelity algorithms such as HyperBand [12] could not benefit from the different stopping criteria provided. In turn, regularized evolution (RE, [1]) and SMAC [7] stood out. In the following, we give the details of both RE and SMAC. Next, we describe irace [8], another relevant AC that we assess in this work.

Regularized evolution is a (μ,1)-EA proposed for NAS that uses accuracy and aging for mating and environmental selection, respectively. In more detail, at each iteration RE maintains a population of candidate architectures. Best-performing architectures regarding accuracy are more likely to be selected to produce a novel candidate architecture. This is achieved by bit-flip mutation to an arbitrary edge, or by replacing an arbitrary node's label with a different label. The novel candidate architecture replaces the oldest candidate in the current population. In particular, authors argue that this aging-based environmental selection promotes regularization.

SMAC is a sequential, model-based algorithm configuration procedure. The configuration process in SMAC starts by performing the evaluation of an initial configuration (e.g. a parameter setting known for its good performance). Then, it alternates between (i) building a random forest model to predict configuration performance; (ii) searching the configuration space for promising configurations using the model as surrogate for performance assessment, and; (iii) evaluating the selected configurations on the target problem.

The good performance obtained by SMAC indicates that out-of-the-box ACs have the potential to be good options for real NAS tasks. Yet, real NAS tasks pose an additional challenge regarding the computation time that is required for the configuration process. This aspect is not fully addressed in NAS-Bench-101, as the configuration task is assessed for final performance. In a real NAS task, the sequential nature of SMAC can be a drawback. Compared to SMAC, HyperBand (HB) has the advantage of being fully parallel, but HB applies limited learning on its search process, relying entirely on multi-fidelity. Another potentially parallelizable algorithm is RE, given the maturity of the literature on parallel EAs; yet, its current version is sequential. In this work, we add to the evaluated methods an AC called irace. Like RE, irace can be classified as an EA and thus inherently parallelizable.

In addition, irace adopts a learning approach to detect high performing areas of the configuration space and focus on them, as described in the following.

irace is an estimation of distribution algorithm that implements iterated racing for configuration evaluation. The AC alternates between (i) applying a racing procedure in which a set of configurations is evaluated on subsets of training instances several times, and; (ii) updating a set of probability distributions used to sample a set of new configurations for a novel iteration (race). At each race, candidate configurations are initially evaluated on a fixed number of instances. Once these evaluations are completed, a statistical test (Friedman's or Student's t-test) is applied to eliminate poor performing configurations from the race. This process continues until the budget assigned for the race is depleted (i.e. number of evaluations) or other convergence criteria are met. A new race is then started using configurations sampled from the updated probability distributions. The most successful applications of irace refer to solution quality optimization, and more recently, it has also been applied to anytime optimization [6].

Note that for ACs a basic configuration scenario comprises (i) the problem samples provided; (ii) the parameter/design space; (iii) a performance metric, and; (iv) the configuration budget. It is important to note that ACs were originally proposed to configure heuristic optimizers, and that some adjustments have to be done when applying them to AutoML. Hyperparameter tuning, for instance, uses folds to represent problem instance distributions. But this is not an option in the deep learning context, as models are commonly trained repetitively on a single dataset. Hence, the application of ACs to deep learning model design is not trivial, as most ACs are not designed for this single-instance scenario and its lack of variability. Regarding parameter space, the way design choices are encoded is critical for a benchmark like NAS-Bench-101, since the parameters should allow the configurator to detect good architecture components and their interactions.

Concerning performance, the measure adopted in the configuration process is an estimation of the real performance across training samples and stochasticity. In more detail, the performance estimation required to assess the quality of a candidate configuration is commonly calculated by aggregating the results obtained from multiple evaluations of the given configuration. The larger the variability in the results, the more evaluations will be required for a precise estimation. For this reason, both irace and SMAC evaluate the same configuration multiple times. By contrast, RE follows the traditional approach in EAs of evaluating each candidate configuration only once. A configuration is re-evaluated only if it is produced in different generations (iterations).

C. Anytime Optimization

Assessing and designing algorithms from the perspective of anytime optimization [6] means that algorithms should be high-performing regardless of the stopping criteria adopted. Though this is always desirable, optimization algorithms are sensitive to the stopping criterion adopted, as a fast-converging search tends to lead to poor final-quality outcomes. In NAS, this is even more important given the cost of specialized computational resources. NAS-Bench-101, for instance, reports results benchmarked for increasing stopping criteria. Yet, multi-fidelity approaches are unable to benefit from this to the extent expected, as mentioned.

An alternative approach to anytime performance is to formulate the underlying optimization problem as bi-objective, where resources consumed and solution quality are objectives to be minimized. Using this approach, the performance assessment theory devised for bi-objective optimization can be employed to draw Pareto-compliant conclusions, as follows:

Set comparison relations. Two sets of solutions that represent different compromise solutions between conflicting objectives can be compared using Pareto set comparison relations. Among the most relevant to our assessment, two solution sets A and B can be considered *incomparable*. In the context of anytime performance where the conflicting objectives are resources consumed and solution quality, an example is an algorithm A finding better solutions faster than another algorithm B, but being outperformed by B in the long run.

Unary performance measures [13]. In many practical situations, two solution sets will be deemed incomparable, but it is still possible to prefer one over another. This is captured by different unary performance measures, such as the hypervolume indicator. The hypervolume is also proven Pareto-compliant, which means that a set cannot be better than another if the hypervolume indicates the opposite. Unary indicators are also scalable as to the number of sets assessed, a desirable aspect in the assessment of optimization algorithms.

Empirical attainment functions (EAFs) [10]. A fine-grained comparison between two sets can help visualize what parts of the objective space are better achieved by each algorithm. EAFs are probability distribution density plots that indicate the frequency with which an algorithm finds solutions in a given region of the objective space. EAF difference plots compare a pair of algorithms by computing the difference in their EAFs, indicating which algorithm performs better in which region of the objective space and with what probability.

III. Enriching the Insights from NAS-Bench-101

As discussed above, NAS-Bench-101 is a very relevant effort towards comparability in NAS research, and a promising testbed for novel algorithms. In this section, we discuss how to further benefit from it. Concretely, we first consider how anytime performance assessment complements final-quality. Next, we discuss the design choices that define NAS-Bench-101, specifically the fixed number of nodes, the limited variability provided, and the stopping criterion adopted. Finally, we discuss the suggested experimental guidelines, in particular the number of repetitions for algorithm evaluation.

A. Evaluating Anytime Performance

The original evaluation setup for NAS-Bench-101 assesses algorithms based on the selected cell architecture performance. Specifically, authors compute the empirical cumulative distribution function (ECDF) in this final-quality approach (though they do not aggregate ECDFs for conclusions). However common in the optimization literature, this approach greatly reduces the benefits of having a pre-computed benchmark,

especially given the extremely large TPU computation time available to NAS algorithms. By contrast, a very large budget is an asset to anytime assessment, as it covers a wide range of different scenarios practitioners may encounter.

One limitation with a bi-objective formulation for anytime performance is that the performance of algorithms must comprise a monotonic curve (the Pareto front). Yet, in machine learning this is only expected for validation error, and it is very likely that an algorithm that performs exceedingly well on validation will decrease its performance on testing due to overfitting. Two alternative solutions can be considered in this context. The first is to render performance curves monotonic. The practical interpretation of this choice is that all best-so-far cell architectures identified by the algorithm would have to be tested when arbitrary stopping criteria were required.[1] A second alternative is to compute the area under the curve depicted by the Pareto front, though without an assurance that conclusions will be Pareto-compliant. In the assessment we conduct in this paper, we opt for the first alternative, given the importance of Pareto-compliance.

B. Experimental Design Choices

Solution encoding in NAS-Bench-101 comprises a fixed-size adjacency matrix to represent a variable-size graph. In more detail, a 7x7-matrix is used to represent a DAG, and nodes that are not connected to the input are ruled out when computing metrics (along with their labels). An alternative is to have the number of nodes as part of the encoding and the sizes of the adjacency matrix and label list dependent on this variable. Though traditional algorithms are often unequiped to deal with such conditional parameters, ACs may explore this formulation to improve their search.

Furthermore, to simulate the variability that experimental data generally has in practice, authors provided results for runs with three different seeds. Though relevant, we believe that the variance provided by this approach would not compensate for the added training time for given ACs. Specifically, algorithms that search the cell design space caching architecture performance will likely consider three times more architectures than algorithms that always query NAS-Bench-101.

Finally, algorithm comparison in NAS-Bench-101 is based on TPU time alone,[2] under the assumption that the CPU time spent by algorithms would be negligible in comparison. Though this is common practice in expensive function evaluation optimization, we argue that this assumption does not hold in a NAS scenario. However expensive, architecture evaluation is performed on TPU clusters, which are highly parallel. In contrast, algorithm processing is traditionally performed in CPUs, and if algorithms do not use an efficient parallel approach, a bottleneck at this point of the process is not compensated by additional TPU power. To preserve comparability with the results from NAS-Bench-101, we maintain the TPU-time assessment, but remark that some of the conclusions should be investigated in a wallclock-time-based future work.

[1]We assume that TPU availability for testing is not an issue, as it is expected to be negligible in comparison to training time.

[2]Though results are reported in that work concerning wallclock time (which should include CPU time), the code provided by the authors to replicate experiments only reports TPU time.

C. Suggested Experimental Guidelines

Among the suggestions from the NAS-Bench-101 proposers to improve comparability between NAS algorithms is to use a large number of repetitions of the algorithms assessed. Indeed, authors employed 500 repetitions from each algorithm considered for a maximum TPU time of 10^7 seconds per run. We argue against this practice, believing it is not realistic and can become counterproductive as follows. Probing algorithms for an excessively large number of repetitions with such a large cutoff time will produce statistics that have little practical meaning. A practitioner aiming at low-probability performance would be inclined to run a significant number of repetitions of the selected algorithm. Yet, NAS algorithms performance improves as a function of the computational budget provided. As such, a reduced number of repetitions using a larger cutoff time would likely produce better results than what the guidelines suggest. Though we cannot simulate the scenario with a budget larger than 10^7 TPU seconds, we show how the relative performance of the algorithms is little affected by using a more reasonable number of repetitions.

The discussion provided in this section evidences the number of ways in which we believe the insights obtained from NAS-Bench-101 could be enriched. In the next section, we conduct a performance assessment of algorithms that were identified as high-performing in NAS-Bench-101, as well as irace, which represents ACs that are able to combine parallelization with learning.

IV. Preliminary Final-Quality Assessment

In this section, we perform experiments to study, with regards to final quality, the main points on NAS-Bench-101 discussed in the previous section. Initially, we demonstrate that with a reduced number of repetitions we are still able to satisfactorily assess relative performance of the algorithms considered here. Next, we compare the algorithms we consider from a final-quality perspective. Additionally, we examine the consequences of limiting evaluation variability. Lastly, we discuss the effects of extending the parameter space to explicitly handle a variable number of nodes.

The following experiments evaluate and compare the best-performing techniques reported in NAS-Bench-101, namely regularized evolution (RE) and SMAC. In this study, we also include irace, an AC that was not evaluated in the NAS-Bench-101 proposal. All NAS algorithms are run on four 24-core Intel Xeon Gold 6252 CPUs running @ 2.10GHz, with 128GB of RAM. We compare all algorithms based solely on TPU time, following the original work. RE and SMAC are run using the code provided by NAS-Bench-101. Additionally, both RE and SMAC are evaluated using parameter settings that showed to lead to good performance in the original NAS-Bench-101 work [4]. For fairness, we preliminarily assessed the performance of irace in NAS-Bench-101 and selected suitable hyperparameters for its application to a final-quality NAS setup. Details of the configuration process and the hyperparameters used are given in supplementary material.[3]

[3]https://github.com/carlosemv/anytime-nasbench-cec2021

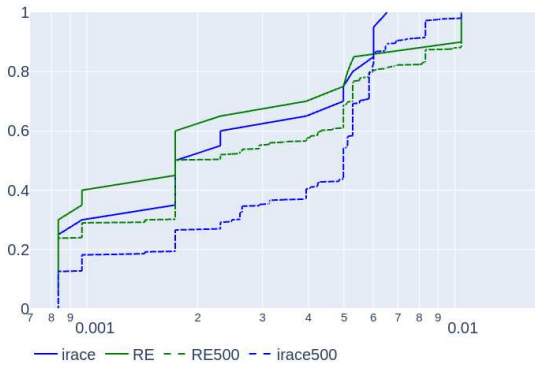

Fig. 1. Empirical cumulative distribution function (x-axis) of the mean final regret (y-axis) assessing the effect of reducing the number of repetitions from 500 (dashed) to 20 (solid) for selected algorithms. Green: RE; blue: irace.

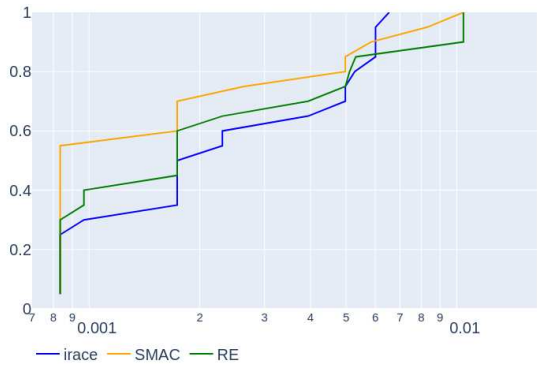

Fig. 2. Empirical cumulative distribution function (x-axis) of the final regret (y-axis) from 20 runs of each algorithm.

A. *Effect of the Number of Repetitions*

We start our assessment discussing the effect of the number of repetitions adopted for algorithm evaluation in NAS-Bench-101. We aim at reproducing as best as possible the experiments presented in the original NAS-Bench-101 paper. Nevertheless, we are constrained by the computational overhead incurred by the sequential nature of SMAC and thus, we are unable to execute the 500 runs of SMAC in our computational setup. We include in the supplementary material runtime statistics for all runs performed here. In the following, we assess if the NAS techniques can be evaluated using fewer runs and if these results are comparable to the ones in the original NAS-Bench-101 work.

Figure 1 shows the ECDFs for final quality, measured as mean test regret, of 20 (solid) and 500 (dashed) runs of RE (green) and irace (blue). Regarding RE, we remark that its performance after 500 executions (dashed green curve) matches the report in the original NAS-Bench-101 assessment. More importantly, the comparison between ECDFs produced with 20 or 500 repetitions for the algorithms highlights two important insights. First, the ECDFs for both algorithms are lowered when a larger number of repetitions is adopted. Indeed, this effect is observed progressively if we increase the number of repetitions gradually, as reported in the supplementary material. Second, the relative performance between irace and RE is not greatly affected by the number of repetitions, with the differences in ECDFs following similar patterns.

B. *Comparing Algorithms for Final Quality*

Figure 2 depicts the ECDFs for final quality, measured as mean test regret, of 20 runs of irace, RE, and SMAC. Once again, all algorithms are run under the setup described in the NAS-Bench-101 proposal with the exception of the number of runs, reduced for these experiments from 500 to 20. Concerning SMAC, results differ w.r.t. the original NAS-Bench-101 assessment given a change we adopt in the evaluation for comparison fairness. Specifically, the original assessment of SMAC considered a single evaluation of each configuration arguing that this led to faster convergence than multiple evaluations. We believe that this faster convergence is an effect of the limited variability provided by NAS-Bench-101 and thus using such strategy may benefit all algorithms. As we will discuss in the following sections, that approach not only speeds convergence for SMAC, but leads to an improvement in its anytime performance.

Concerning results given in Figure 2, the comparison between irace and the remaining algorithms shows that RE and SMAC are able to find better-performing architectures more often than irace. However, irace less often returns poor-performing architectures compared to RE and SMAC. Interestingly, SMAC outperforms not only irace, but also RE, which had not been reported in the NAS-Bench-101 evaluation. We believe this is due to the single versus multiple evaluation per candidate previously discussed, as follows. In [4], authors report that single evaluation speeds up convergence for SMAC. In search optimization, it is rather common that improving convergence speed without accounting for anytime performance leads to a decrease in final-quality performance. In addition, it is important to remark that the stopping criterion adopted does not include CPU time, hence the overhead incurred by learning in SMAC is not accounted for.

For overall conclusions, we compare algorithms based on the area between their ECDF and the y-axis. The smaller the value, the better the performance of the algorithm. Table I groups results by experimental setup (top) and algorithm (bottom). The original setup discussed in this section is labeled **O** (*original*). For this setup, SMAC is the best-ranked algorithm, followed by irace and RE. We performed a Friedman non-parametric test with 98% confidence and Nemenyi's posthoc test. When statistical significance is observed, best-ranked levels in Table I are highlighted in boldface, along with algorithms that are not statistically different to them. Under the setup considered in this section, no statistical difference between the algorithms can be observed.

C. *Evaluation Variability*

As previously discussed, the performance estimation required to assess the quality of an architecture requires multiple evaluations to be precise. In NAS-Bench-101, the variability of the evaluations of the sampled architectures is represented only by three evaluation seeds. Given this characteristic of NAS-Bench-101, we can consider storing evaluation results (caching) to use the saved TPU time to evaluate more candidate architectures. This is essentially equivalent to the evaluation strategy defined in the experimental setup of SMAC

TABLE I
ECDF ANALYSIS GROUPED BY EXPERIMENTAL SETUP (TOP) AND ALGORITHM (BOTTOM). O: ORIGINAL; VS: VARIABLE-SIZED; C: CACHING; CVS: CACHING & VARIABLE-SIZED. BEST-RANKED LEVELS ARE HIGHLIGHTED WHEN STATISTICALLY DIFFERENT THAN THE OTHERS. VALUES IN PARENTHESES ARE MULTIPLIED BY 10^4.

O	SMAC (23)	irace (28)	RE (31)
VS	**SMAC (18)**	irace (46)	RE (47)
C	SMAC (34)	irace (50)	RE (55)
CVS	**SMAC (20)**	irace (42)	RE (50)

RE	O (31)	VS (47)	CVS (50)	C (55)
irace	O (28)	CVS (42)	VS (46)	C (50)
SMAC	VS (18)	CVS (20)	O(23)	C (34)

in the original evaluation of NAS-Bench-101. In this section, we assess algorithms setting to one the maximum number of evaluations per architecture. Though only SMAC has a hyperparameter to limit the number of function evaluations per configuration, we implement caching within the benchmark querying API for RE and irace.

Table I shows results for the setup discussed in this section labeled as **C** (short for *caching*). Results grouped by experimental setup (top) show that the relative performance of the algorithms is not altered by this factor alone. Yet, results grouped by algorithm (bottom) demonstrate that the performance of all algorithms is much worsened when the caching strategy is adopted, though no statistical difference is observed w.r.t. other setups. We remark that irace and RE were originally configured for the setup without caching, which could affect their performance in the caching setup. Interestingly, SMAC performs better by not using caching even having been configured for the caching setup in the original NAS-Bench-101 assessment. Furthermore, it is also important to remark that irace bases its search on statistical tests, which require variability to work. In a sense, though caching allows irace to better explore the NAS search space, it also impairs the exploitation capability of the algorithm.

D. Including the Number of Nodes in the Design Space

The design space adopted in the original NAS-Bench-101 considers a fixed-size encoding of network architecture. It is important to note that, though the encoding is fixed-size, the final architecture they encode is variable in size. Yet, search algorithms are blind to this decoupling between architecture and its representation. We then include the number of nodes as a parameter in the design space that determines the dimensions of the adjacency matrix and node label list. Concerning final-quality ECDF analysis grouped by experimental setup given in Table I (top) for this setup (labeled *variable-sized*, **VS**), the relative performance of the algorithms is not affected. Yet, SMAC is now able to significantly outperform the remaining algorithms. In more detail, results grouped by algorithm (bottom) show that SMAC greatly benefits from the variable-sized approach, whereas RE and irace worsen their performance.

Regarding RE and SMAC, these results are consistent with our previous discussion on the benefits of ACs. In more detail, the variable-size approach poses higher difficulty due to the larger parameter space and/or the conditional dependencies

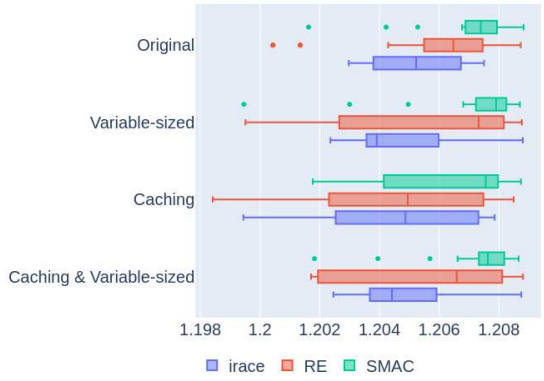

Fig. 3. Boxplots of the hypervolume (x-axis) obtained in 20 runs from each of the algorithms selected, depicted in varying colors. Boxplots are grouped along the y-axis by the experimental setup adopted.

incurred by the new parameter. Under this setup, RE mutation might not be as adequate for the new design space and the algorithm is not able to define as good search trajectories as before. Conversely, ACs have been devised for scenarios that include this kind of parameter, and the performance of SMAC reflects this. Concerning irace, we conjecture that the different probability modeling approaches adopted by SMAC (global) and irace (local) account for the differences in performance between these algorithms for this setup. In addition, we remark that algorithms have been configured for the original setup, and hence an assessment of this encoding using reconfigured algorithms could likely alter our conclusions.

We conclude our preliminary final-quality assessment highlighting that the combination of caching and variable-sized encoding (a setup labeled **CVS**) reveals interactions between these factors. In more detail, Table I shows that combining caching with variable-sized encoding improves over using each of these factors individually for irace. Conversely, for RE and SMAC, caching reduces the benefits of the variable-sized encoding, even if to a much smaller extent than when we compare the original setup with using caching alone.

V. ASSESSING ANYTIME PERFORMANCE

As discussed in Section III, an assessment based solely on final quality excludes from the analysis the time required to achieve a given performance level. In this section, we perform an anytime comparison of all algorithms to investigate how their search dynamics differ. We also discuss the anytime effects of caching and of the variable-sized encoding.

A. Comparing Algorithms for Anytime Performance

Figure 3 gives boxplots of the hypervolume achieved by the algorithms in which runs are grouped by the experimental setup adopted. The smaller the value achieved, the better the anytime performance of the algorithm. The hypervolume indicator requires a reference point, and as traditionally done in the literature we use point $(2.1, 2.1)$. To do so, we initially normalize results to the $[1, 2]$ range respectively using $[0, 10^7]$ and $[0, 1]$ as bounds for TPU time and mean test regret.

For the original setup adopted in NAS-Bench-101, we observe intersections between the boxplots, though clearly SMAC outperforms irace. This is confirmed by the rank sums

TABLE II
RANK SUM (RS) ANALYSIS OF HYPERVOLUMES GROUPED BY EXPERIMENTAL SETUP (TOP) AND ALGORITHM (BOTTOM). O: ORIGINAL; VS: VARIABLE-SIZED; C: CACHING; CVS: CACHING & VARIABLE-SIZED. BEST-RANKED LEVELS ARE HIGHLIGHTED WHEN STATISTICALLY DIFFERENT DIFFERENT THAN OTHERS.

O	**SMAC (30)**	**RE (37)**	irace (53)
VS	**SMAC (30)**	**RE (40)**	irace (50)
C	SMAC (32)	RE (43)	irace (45)
CVS	**SMAC (30)**	**RE (43)**	irace (47)

RE	VS (47)	O (48)	CVS (49)	C (56)
irace	O (45)	CVS (48)	C (52)	CVS (55)
SMAC	VS (43)	CVS (46)	O (54)	C (57)

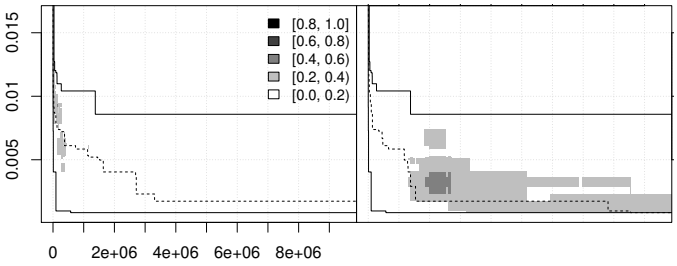

Fig. 4. Empirical attainment function (EAF) difference plots comparing RE (left) and SMAC (right) run 20 times each. x-axis: TPU time; y-axis: mean test regret.

given in Table II (top), where SMAC is followed by RE and irace ranks last. Interestingly, the variance in results increases in this order as well. The relative rankings observed for anytime performance contrast with the rankings observed for final quality in Table I (top), where irace ranked better than RE. The poor anytime performance of irace is related to the budget set for each iteration, which is directly affected by the hyperparameters we configured for final-quality assessment.

We then compare the two best-performing algorithms, i.e., RE (left) and SMAC (right), with the help of empirical attainment function (EAF) difference plots, given in Figure 4. As previously discussed, the x-axis depicts TPU time only, whereas the y-axis gives mean test regret. Dashed lines depict the 50% attainment function from each algorithm, and differences depicted as shaded areas on either side of the plot indicate that the given algorithm presented a better performance at that point of the runs. RE is designed for this type of configuration scenario, favoring a quick intensification of the search. On the other hand, SMAC requires more TPU time to obtain a similar level of performance. This is the consequence of the evaluation strategy implemented by SMAC which, despite the low variability inherent to NAS-Bench-101, assumes the estimation of architecture performance to require several executions to be accurate. Hence, RE has a higher probability of obtaining best performance than SMAC with a lower TPU time budget, while for higher TPU times SMAC provides a better probability. If CPU-time were accounted for, however, the conclusions drawn from this comparison would likely be affected by the sequential nature of SMAC.

B. Caching and Variable-Sized Encoding Effects

The final-quality assessment discussed in the previous section showed that algorithm, variability degree, and solution encoding were interacting factors. Specifically, the only pattern observable in Table I (bottom) referred to the caching strategy, which led to poor results for all algorithms when used with the fixed-size solution encoding. By contrast, Figure 3 shows that these alternative approaches affect most anytime performance as to the variance in the results. RE is the algorithm most affected, whereas SMAC is the least. Distribution shifts are also observed for all algorithms, though they vary as a function of the remaining factors. For SMAC and RE, the variable-sized encoding right-shifts the distributions, though a bit less in the presence of caching. Conversely, irace presents its best anytime performance in the original NAS-Bench-101 setup.

The rank sum analysis given in Table II confirms these findings. On the top grouping, experimental setups do not alter the relative performance of the algorithms, though caching affects statistical significance due to the high variance in SMAC results discussed above. For the bottom grouping, rank sums for the different setups are very similar within each row, again due to the increased variance in results. As previously discussed, these conclusions should consider that all algorithms have been configured for final-quality optimization under the original setup. Yet, they further evidence the need for benchmarking NAS from an anytime performance perspective.

VI. HIGH-PERFORMING CONFIGURATION INSIGHTS

In this section, we analyze the final configurations and thus, the architectures selected by the different NAS techniques. For this purpose, we study the configurations obtained in each run of the algorithms. Figure 5 shows parallel categories plots for SMAC. For clarity, only NAS hyperpameters related to the node label list are given (op_1-op_5). The possible values for these parameters are *conv3* (3×3 convolution), *conv1* (1×1 convolution), *mp* (3×3 max-pooling), or empty in the case of the variable-sized approach. Since topology is defined by the adjacency matrix, no layer order or architecture size should be assumed. Color scaling reflects the mean test regret, which we additionally depict as a discretized variable in the left-most column of the plot. Though conclusions in this section are drawn from all plots analyzed, the remaining ones are given as supplementary material for brevity.

In line with the layer type performance effects reported in NAS-Bench-101, *conv3* is the most frequently selected node value. This was especially true for irace, particularly in the fixed-size approach. In fact, several of the high-performing configurations seem to use only *conv3* nodes, with no *conv1* nor pooling layers. This is surprising, as manual design would certainly include them. We believe that these results reflect the design for accuracy adopted in NAS-Bench-101. In particular, one of the most significant benefits from pooling is reducing training time. Yet, architectures are generally evaluated only as to their accuracy. Though it would be possible to assess some level of trade-off between architecture accuracy and total training time, we have followed the original setup from NAS-Bench-101 where this is not considered. We remark,

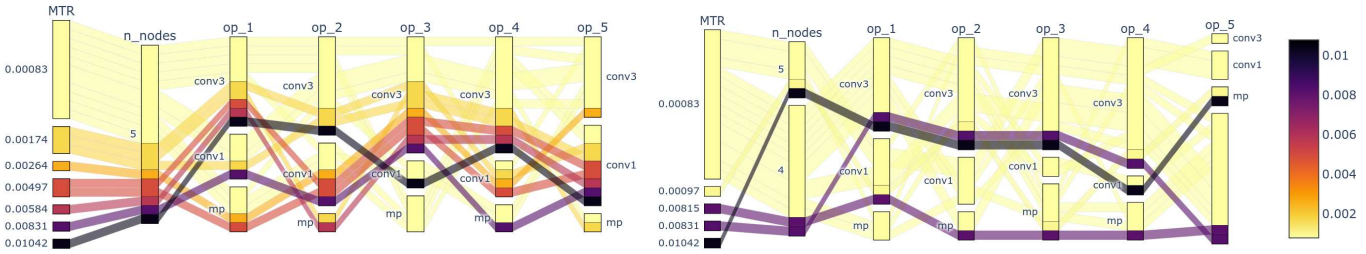

Fig. 5. Parallel categories plots of the 20 architectures selected by SMAC with fixed (left) and variable (right) number of nodes.

though, that this can lead to the selection of extremely costly architectures, such as the *conv3*-based mentioned previously.

Regarding caching, all algorithms have a hard time finding good configurations using this approach. This is a further indication of the usefulness of variability in this benchmark from a final-quality perspective. On the other hand, adopting a variable number of nodes can be beneficial, as is the case for SMAC. Yet, we remark that the architectures given in Fig. 5 may present less than n_{nodes} nodes due to their topology, which is not shown here. Finally, we note that RE finds a top-performing configuration containing a single 1×1 convolution layer. We remark that this is possible due to the scalable architecture approach discussed in Section II.

VII. Conclusion

Deep learning breakthroughs in challenging fields such as computer vision have redefined the research in neural architecture search (NAS [1]–[3]). Besides the traditional contributions from neuroevolution, other effective algorithm classes have shown interesting results, such as algorithm configuration (AC [11]). However, the characteristics of deep learning scenarios differ considerably from optimization scenarios for which most effective ACs have been devised. In addition, the computational cost incurred for evaluating architectures currently prohibits extensive ACs runs.

NAS-Bench-101 is a first effort towards comparability of NAS algorithms. Besides reusable data, authors also provide relevant guidelines for the evaluation and proposal of algorithms. In this work, we have proposed different ways in which the insights produced from this benchmark can be enriched, the most significant being the bi-objective formulation that enables an anytime performance assessment. Not surprisingly, the evaluation of irace [8], RE [1], and SMAC [7] show that the relative performance of the techniques is not always the same from a final-quality or an anytime performance perspective.

A second contribution from our work is to study the effects of design choices embedded in the original NAS-Bench-101 assessment. Specifically, we discuss the effects of a variable-sized encoding and caching, and demonstrate that algorithms are affected in different ways by these alternative setups. Finally, we argue against the use of an excessive number of algorithm repetitions, providing evidence that the budget of a configurator could be better spent on more instances or evaluating more architectures.

Besides the future work possibilities discussed along the paper, we highlight two other important pathways. The first is how to account for architecture training time besides accuracy, likely in a bi-objective formulation of architecture evaluation. The second is extending NAS-Bench-101 to account for more datasets, so effective algorithms devised for multi-instance configuration may identify architectures that are high-performing across different datasets. Altogether, the benefits of these investigations may contribute to reduce dataset-dependent, computationally prohibitive campaigns.

Acknowledgment

Leslie Pérez Cáceres acknowledges the support of Fondecyt project #11190135.

References

[1] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, vol. 33, no. 01, 2019, pp. 4780–4789.

[2] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[3] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *IEEE CVPR*, 2018, pp. 8697–8710.

[4] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *ICML*. PMLR, 2019, pp. 7105–7114.

[5] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[6] M. López-Ibánez and T. Stützle, "Automatically improving the anytime behaviour of optimisation algorithms," *Europ. J. of Oper. Res.*, vol. 235, no. 3, pp. 569–582, 2014.

[7] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *LION*. Springer, 2011, pp. 507–523.

[8] M. López-Ibánez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Oper. Res. Perspect.*, vol. 3, pp. 43–58, 2016.

[9] L. C. Bezerra, M. López-Ibáñez, and T. Stützle, "Automatic configuration of multi-objective optimizers and multi-objective configuration," in *High-Performance Simulation-Based Optimization*. Springer, 2020, pp. 69–92.

[10] M. López-Ibánez, L. Paquete, and T. Stützle, "Exploratory analysis of stochastic local search algorithms in biobjective optimization," in *Experimental methods for the analysis of optimization algorithms*. Springer, 2010, pp. 209–222.

[11] T. Elsken, J. H. Metzen, F. Hutter *et al.*, "Neural architecture search: A survey." *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[12] L. Li, K. Jamieson, G. De Salvo, R. A. Talwalkar, and A. Hyperband, "A novel bandit-based approach to hyperparameter optimization," *Computer Vision and Pattern Recognition, arXiv: 1603.0656*, 2016.

[13] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Trans. on Evol. Computat.*, vol. 7, no. 2, pp. 117–132, 2003.