Exploring the Loss Landscape in Neural Architecture Search

Colin White[1] **Sam Nolen**[1] **Yash Savani**[1,2]

[1]Abacus.AI. {colin, sam}@abacus.ai
[2]Carnegie Mellon University. ysavani@cs.cmu.edu

Abstract

Neural architecture search (NAS) has seen a steep rise in interest over the last few years. Many algorithms for NAS consist of searching through a space of architectures by iteratively choosing an architecture, evaluating its performance by training it, and using all prior evaluations to come up with the next choice. The evaluation step is noisy - the final accuracy varies based on the random initialization of the weights. Prior work has focused on devising new search algorithms to handle this noise, rather than quantifying or understanding the level of noise in architecture evaluations. In this work, we show that *(1)* the simplest hill-climbing algorithm is a powerful baseline for NAS, and *(2)*, when the noise in popular NAS benchmark datasets is reduced to a minimum, hill-climbing to outperforms many popular state-of-the-art algorithms. We further back up this observation by showing that the number of local minima is substantially reduced as the noise decreases, and by giving a theoretical characterization of the performance of local search in NAS. Based on our findings, for NAS research we suggest *(1)* using local search as a baseline, and *(2)* denoising the training pipeline when possible.

1 INTRODUCTION

Neural architecture search (NAS) is a widely popular area of machine learning which seeks to automate the development of the best neural network for a given dataset. Many methods for NAS have been proposed, including reinforcement learning, gradient descent, and Bayesian optimization [Elsken et al., 2018, Zoph and Le, 2017, Liu et al., 2018b]. Many popular NAS algorithms can be instantiated by the optimization problem $\min_{a \in A} f(a)$, where A denotes a set of architectures (the *search space*) and $f(a)$ denotes the objective function for a, often set to the validation accuracy of a after training using a fixed set of hyperparameters. With the release of three NAS benchmark datasets [Ying et al., 2019, Dong and Yang, 2020, Siems et al., 2020], the extreme computational cost for NAS is no longer a barrier, and it is easier to fairly compare different algorithms. However, recent work has noticed that the training step used in these benchmarks is quite stochastic [Ying et al., 2019, Siems et al., 2020]. In NAS, where the goal is to search over complex neural architectures, a noisy reward function makes the problem even more challenging. In fact, recent innovations are designed specifically to handle the noisy objective function [Real et al., 2019, Zaidi et al., 2020]. A natural question is therefore, how much of the complexity of NAS can be attributed to the noise in the training pipeline?

In this work, [1] we answer this question by showing that the difficulty of NAS is highly correlated with the noise in the training pipeline. We show that *(1)* the simplest local search algorithm (hill-climbing) is already a strong baseline in NAS, and *(2)*, when the noise in the training pipeline is reduced to a minimum, local search is sufficient to outperform many state-of-the-art techniques.

Local search is a simple and canonical greedy algorithm in combinatorial optimization and has led to famous results in the study of approximation algorithms [Michiels et al., 2007, Cohen-Addad et al., 2016, Friggstad et al., 2019]. However, local search has been neglected in the field of NAS; a recent paper even suggests that it performs poorly due to the number of local minima throughout the search space [Wang et al., 2018]. The most basic form of local search, often called the hill-climbing algorithm, consists of starting with a random architecture and then iteratively training all architectures in its neighborhood, choosing the best one for the next iteration. The neighborhood is typically defined as all architectures which differ by one operation

[1]See the full-length paper here: https://arxiv.org/abs/2005.02960. Our code is available at https://github.com/naszilla/naszilla.

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

or edge. Local search finishes when it reaches a (local or global) optimum, or when it exhausts its runtime budget.

We show that on NAS-Bench-101, 201, and 301/ DARTS [Ying et al., 2019, Dong and Yang, 2020, Siems et al., 2020, Liu et al., 2018b], if the noise is reduced to a minimum, then local search is competitive with all popular state-of-the-art NAS algorithms. This result is especially surprising because local search, which can be implemented in five lines of code (Algorithm 1), is in stark contrast to most state-of-the-art NAS algorithms, which have many moving parts and even use neural networks as subroutines [Wen et al., 2019, Shi et al., 2019]. This suggests that the complexity in prior methods may have been developed to deal with the noisy reward function. We also experimentally show that as the noise in the training pipeline increases, the number of local minima increases, and the basin of attraction of the global minimum decreases. These results further suggest that NAS becomes easier when the noise is reduced.

Motivated by these findings, we also present a theoretical study to better understand the performance of local search under different levels of noise. The underlying optimization problem in NAS is a hybrid between discrete optimization, on a graph topology, and continuous optimization, on the distribution of architecture accuracies. We formally define a NAS problem instance by the graph topology, a global probability density function (PDF) on the architecture accuracies, and a local PDF on the accuracies between neighboring architectures, and we derive a set of equations which calculate the probability that a randomly drawn architecture will converge to within ϵ of the global optimum, for all $\epsilon > 0$. As a corollary, we give equations for the expected number of local minima, and the expected size of the preimage of a local minimum. These results completely characterize the performance of local search. To the best of our knowledge, this is the first result which theoretically predicts the performance of a NAS algorithm, and may be of independent interest within discrete optimization. We run simulations which suggest that our theoretical results predict the performance on real datasets reasonably well.

Our findings raise a few points for the field of NAS. Since much of the difficulty is in the stochasticity of the training pipeline, denoising the training pipeline as much as possible is worthwhile for future work. Second, our work suggests that local methods for NAS may be promising. That is, methods which explore the search space by iteratively making small edits to the best architectures found so far. Furthermore, we suggest using local search as a baseline for future work. We release our code, and we discuss our adherence to the NAS research checklist [Lindauer and Hutter, 2019].

Our contributions. We summarize our contributions.

- We show that local search is a strong baseline in its own right, and outperforms many state-of-the-art NAS algorithms across three popular benchmark datasets when the noise in the training pipeline is reduced to a minimum. We also show that the number of local minima increases as the noise in the training pipeline increases. Our results suggest that making the training pipeline in NAS more consistent, is just as worthwhile as coming up with novel search algorithms.
- We give a theoretical characterization of the properties of a dataset necessary for local search to give strong performance. We experimentally validate these results on real neural architecture search datasets. Our results improve the theoretical understanding of local search and lay the groundwork for future studies.

2 RELATED WORK

Local search has been studied since at least the 1950s in the context of the traveling salesman problem [Bock, 1958, Croes, 1958], machine scheduling [Page, 1961], and graph partitioning [Kernighan and Lin, 1970]. Local search has consistently seen significant attention in theory [Aarts and Lenstra, 1997, Balcan et al., 2020, Johnson et al., 1988] and practice [Bentley, 1992, Johnson and McGeoch, 1997]. There is also a large variety of work in local optimization of noisy functions, handling the noise by averaging the objective function over multiple evaluations [Rakshit et al., 2017, Akimoto et al., 2015], using surrogate models [Booker et al., 1998, Caballero and Grossmann, 2008], or using regularization [Real et al., 2019].

NAS has gained popularity in recent years [Kitano, 1990, Stanley and Miikkulainen, 2002, Zoph and Le, 2017], although the first few techniques have been around since at least the 1990s [Yao, 1999, Shah et al., 2018]. Common techniques include Bayesian optimization [Kandasamy et al., 2018, Jin et al., 2018], reinforcement learning [Zoph and Le, 2017, Pham et al., 2018, Liu et al., 2018a], gradient descent [Liu et al., 2018b, Dong and Yang, 2019], prediction-based [White et al., 2021a, Shi et al., 2019, White et al., 2021b], evolution [Maziarz et al., 2018, Real et al., 2019], and using novel encodings to improve the search [White et al., 2020, Yan et al., 2020, 2021].

Recent papers have highlighted the need for fair and reproducible NAS comparisons [Li and Talwalkar, 2019, Lindauer and Hutter, 2019], spurring the release of three NAS benchmark datasets [Ying et al., 2019, Dong and Yang, 2020, Siems et al., 2020], each of which utilize tens of thousands of pretrained neural networks. See the recent survey [Elsken et al., 2018] for a more comprehensive overview on NAS.

There has been some prior work using local search for NAS. Elsken et al. [2017] use local search with network morphisms guided by cosine annealing, which is a more complex variant. Wang et al. [2018] use local search as a baseline, but kill the run after encountering a local minimum

rather than using the remaining runtime budget to start a new run. Concurrent work has also shown that simple local search is a strong baseline on NASBench-101 [Ottelander et al., 2020] for multi-objective NAS (where the objective is a function of accuracy and network complexity). This work focuses on macro search rather than cell-based search, and does not investigate the effect of noise on the performance. The existence of this work strengthens one of our conclusions (that local search is a strong NAS algorithm) because it is now independently verified.

3 PRELIMINARIES

In this section, we formally define the local search algorithm and notation that will be used for the rest of the paper. Given a set A, denote an objective function $\ell : A \to [0, \infty)$. We refer to A as a search space of neural architectures, and $\ell(v)$ as the expected validation loss of $v \in A$ over a fixed dataset and training pipeline. When running a NAS algorithm, we have access to a noisy version of ℓ, i.e., when we train an architecture a, we receive loss$= \ell(v) + x$ for noise x drawn from a distribution $\mathcal{D}_v$ (we explore different families of distributions in Sections 4 and 5). The goal is to find $v^* = \text{argmin}_{v \in A} \ell(v)$, the neural architecture with the minimum validation loss, or an architecture whose validation loss is within ϵ of the minimum, for some small $\epsilon > 0$. We define a neighborhood function $N : A \to 2^A$. For instance, $N(v)$ might represent the set of all neural architectures which differ from v by one operation or edge.

Local search in its simplest form (also called the hill-climbing algorithm) is defined as follows. Start with a random architecture v and evaluate $\ell(v)$ by training v. Iteratively train all architectures in $N(v)$, and then replace v with the architecture u such that $u = \text{argmin}_{w \in N(v)} \ell(w)$. Continue until we reach an architecture v such that $\forall u \in N(v),\ \ell(v) \leq \ell(u)$, i.e., we reach a local minimum. See Algorithm 1. We often place a runtime bound on the algorithm, in which case the algorithm returns the architecture v with the lowest value of $\ell(v)$ when it exhausts the runtime budget. In Section 4, we also consider two simple variants. In the query_until_lower variant, instead of evaluating every architecture in the neighborhood $N(v)$ and picking the best one, we draw architectures $u \in N(v)$ at random without replacement and move to the next iteration as soon as $\ell(u) < \ell(v)$. In the continue_at_min variant, we do not stop at a local minimum, instead moving to the second-best architecture found so far and continuing until we exhaust the runtime budget. One final variant, which we explore in Appendix A, is choosing k initial architectures at random, and setting v_1 to be the architecture with the lowest objective value.

Notation. Now we define the notation used in Sections 4 and 5. Given a search space A and a neighborhood function N, we define the neighborhood graph $G_N = (A, E_N)$ such that for $u, v \in A$, the edge (u, v) is in E_N if and only if $v \in N(u)$. We assume that $v \in N(u)$ implies $u \in N(v)$, therefore, the neighborhood graph is undirected. We only consider symmetric neighborhood functions, that is, $v \in N(u)$ implies $u \in N(v)$. Therefore, we may assume that the neighborhood graph is undirected. Given G, N, and a loss function ℓ, define $\texttt{LS} : A \to A$ such that $\forall v \in A$, $\texttt{LS}(v) = \text{argmin}_{u \in N(v)} \ell(u)$ if $\min_{u \in N(v)} \ell(u) < \ell(v)$, and $\texttt{LS}(v) = \emptyset$ otherwise. In other words, $\texttt{LS}(v)$ denotes the architecture after performing one iteration of local search starting from v. See Figure 4.1 for an example. For integers $k \geq 1$, recursively define $\texttt{LS}^k(v) = \texttt{LS}(\texttt{LS}^{k-1}(v))$. We set $\texttt{LS}^0(v) = v$ and denote $\texttt{LS}^*(v) = \min_{k | \texttt{LS}^k(v) \neq \emptyset} \texttt{LS}^k(v)$, that is, the output when running local search to convergence, starting at v. Similarly, define the preimage $\texttt{LS}^{-k}(v) = \{u \mid \texttt{LS}^k(u) = v\}$ for integers $k \geq 1$ and $\texttt{LS}^{-*}(v) = \{u \mid \exists k \geq 0 \text{ s.t. } \texttt{LS}^{-k}(u) = v\}$. That is, $\texttt{LS}^{-*}(v)$ is a multifunction which defines the set of all points u which reach v at some point during local search. We refer to $\texttt{LS}^{-*}$ as the full preimage of v.

Algorithm 1 Local search

Input: Search space A, objective function ℓ, neighborhood function N

1. Pick an architecture $v_1 \in A$ uniformly at random
2. Evaluate $\ell(v_1)$; denote a dummy variable $\ell(v_0) = \infty$; set $i = 1$
3. While $\ell(v_i) < \ell(v_{i-1})$:
   i. Evaluate $\ell(u)$ for all $u \in N(v_i)$
   ii. Set $v_{i+1} = \text{argmin}_{u \in N(v_i)} \ell(u)$; set $i = i + 1$

Output: Architecture v_i

4 EXPERIMENTS

In this section, we discuss our experimental setup and results. To promote reproduciblity, we discuss how our experiments follow the best practices checklist [Lindauer and Hutter, 2019] in Appendix A, and we release our code at `https://github.com/naszilla/naszilla`. We start by describing the benchmark datasets used in our experiments.

NAS benchmarks. To conduct our experiments, we use the three most popular NAS benchmarks: NASBench-101, 201, and 301/DARTS. NASBench-101 consists of over 423,000 unique neural architectures with precomputed validation and test accuracies for 108 epochs on CIFAR-10. The cell-based search space consists of five nodes which can take on any DAG structure, and each node can be one of three operations. Each architecture was trained a total of three times using different random seeds. The NASBench-201 dataset consists of $5^6 = 15,625$ unique neural architectures, with precomputed validation and test accuracies for 200

epochs on CIFAR-10, CIFAR-100, and ImageNet-16-120. The search space consists of a cell which is a complete directed acyclic graph over 4 nodes. Therefore, there are $\binom{4}{2} = 6$ edges. Each *edge* can be one of five operations. As in NASBench-101, on each dataset, each architecture was trained three times using different random seeds.

The DARTS [Liu et al., 2018b] search space is a popular search space for large-scale cell-based NAS experiments on CIFAR-10. The search space contains roughly 10^{18} architectures, consisting of two 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 on any DAG structure such that each node has degree two. Each edge can take one of eight operations. Recently, a surrogate benchmark, dubbed NASBench-301 [Siems et al., 2020], has been created for the DARTS search space. The surrogate benchmark is created using an ensemble of XGBoost models [Chen and Guestrin, 2016], each initialized with different random weights.

Local search performance. We evaluate the relative performance of local search for NAS in settings with and without noise. In a real-world NAS experiment, noise reduction can be achieved by modifying the training hyperparameters, which we discuss later in this section. However, modifying the hyperparameters of the NASBench architectures is impractical due to the extreme computational cost needed to create the benchmarks [Ying et al., 2019, Dong and Yang, 2020]. Instead, we artificially remove much of the noise in these benchmarks using two different techniques. On NASBench-101 and 201, where each architecture was independently trained three times, the standard way to use the benchmark is to draw validation accuracies at random. However, for each architecture, we can average all three validation accuracies to obtain a less noisy estimate. On NASBench-301, where the standard way to evaluate architectures is by using one estimate from the ensemble uniformly at random, we can take the mean of all of the ensemble estimates. This is shown to be less noisy even than the data used to train the ensemble itself [Siems et al., 2020]. In general, we can easily control the noise of NASBench-301 by returning the mean of the ensemble estimates plus a random normal variable with mean 0 and standard deviation equal to the standard deviation of the ensemble estimates, multiplied by a constant x. $x = 0$ corresponds to the denoised setting, while $x = 1$ corresponds to the standard setting.

We compare local search to random search, regularized evolution [Real et al., 2019], Bayesian optimization, and BANANAS [White et al., 2021a]. For every algorithm, we used the code directly from the corresponding open source repositories. For more details on the implementations, see Appendix A. We gave each algorithm a budget of 300 evaluations. For each algorithm, we recorded the test loss of the architecture with the best validation loss that has been queried so far. We ran 200 trials of each algorithm and averaged the results. For local search, we set $N(v)$ to denote all architectures which differ by one operation or edge. If local search converged before its budget, it started a new run. On NASBench-101 and 301, we used the query_until_lower variant of local search, and on NASBench-201, we used the continue_at_min variant. See Figure 4.2. On both NASBench-101 and 301, local search outperforms all other algorithms when the noise is minimal, amd performs similarly to Bayesian optimization in the standard setting. We include the results for all three datasets of NASBench-201 in Appendix A.

Table 1: Avg. num. of iterations until convergence, num. of local minima, and percent of initial architectures to reach the global minimum, for CIFAR-10 on NASBench-201.

Version	# iters	# local min.	% reached global min.
Denoised	5.36	21	47.4
Standard	4.97	55	6.71
Random	2.56	616	0.717

Local minima statistics. Now we further show that denoised NAS is a simpler optimization problem by computing statistics about the loss landscape of the noisy and denoised search spaces. We start by running experiments on NASBench-201. Since this benchmark is only size 15625, the global minimum is known, which allows us to compute the percent of architectures that converge to the global minimum when running local search. We also compute the number of local minima and average number of iterations of local search before convergence. We run experiments using the standard and denoised versions of NASBench-201 (defined earlier in this section), and we also use a fully randomized version by replacing the validation error for each architecture with a number drawn uniformly from $[0, 1]$. For each experiment, we started local search from *all* 15625 initial seeds for local search, and averaged the results. See Table 1. On the denoised search space, almost half of the 15625 architectures converge to the global minimum, but under 7% reach the global minimum on the standard search space. In Figure 4.1, we give a visualization of the topologies of the denoised and fully random search spaces.

Finally, we run experiments on NASBench-301. Due to the extreme size (10^{18}), the global minimum is not known. However, as described above, the surrogate nature of NASBench-301 allows for a more fine-grained control of the noise. In Figure 4.3, we plot the performance of NAS algorithms with respect to the level of noise in the search space. We also show that the average number of local search iterations needed for convergence decreases with noise.

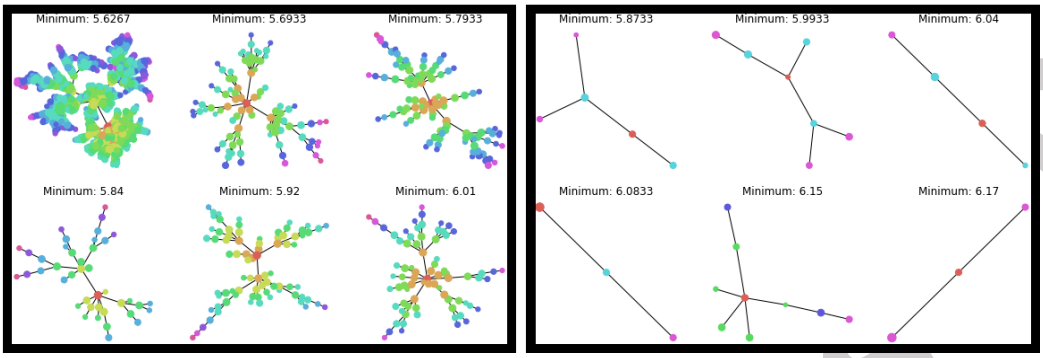

Figure 4.1: The local search tree for the architectures with the six lowest test losses (colored red) on CIFAR-10 on NASBench-201, denoised (left) or fully random (right). Each edge represents an iteration of local search, from the colder-colored node to the warmer-colored node. While 47.4% of architectures reach the global minimum in the denoised version, only 0.71% of architectures reach the global minimum in the random version.

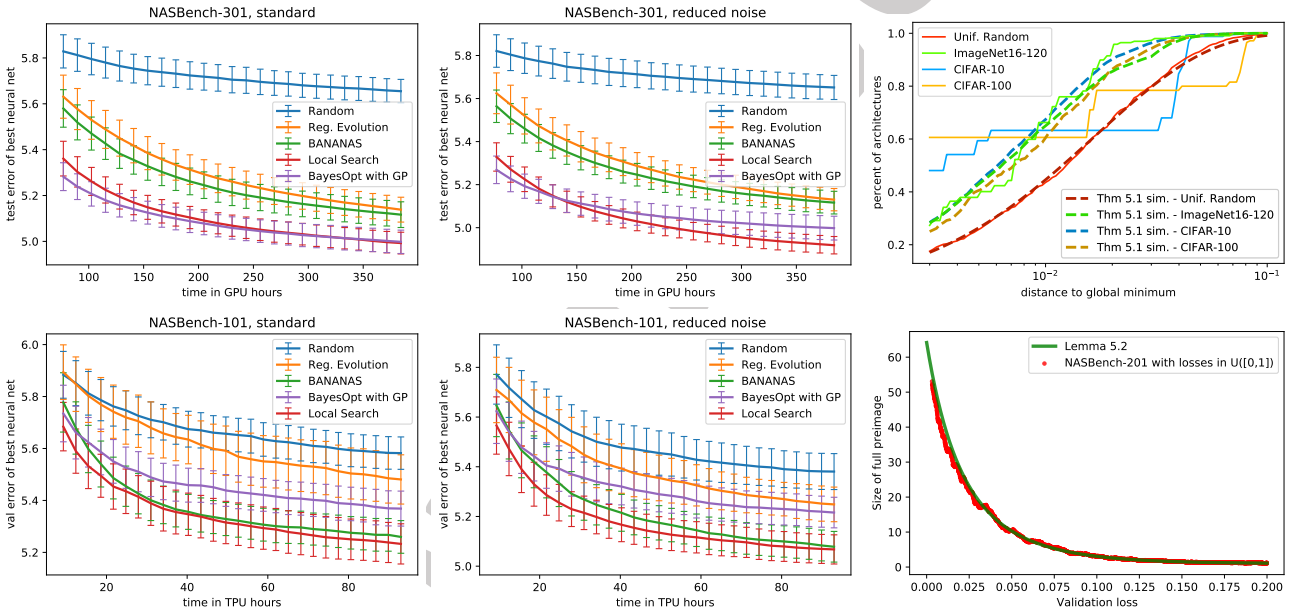

Figure 4.2: Performance of NAS algorithms on standard and denoised versions of NASBench-101 (top left/middle) and NASBench-301 (bottom left/middle). Probability that local search will converge to within ϵ of the global optimum, compared to Theorem 5.1 (top right). Validation loss vs. size of preimages, compared to Lemma 5.2 (bottom right).

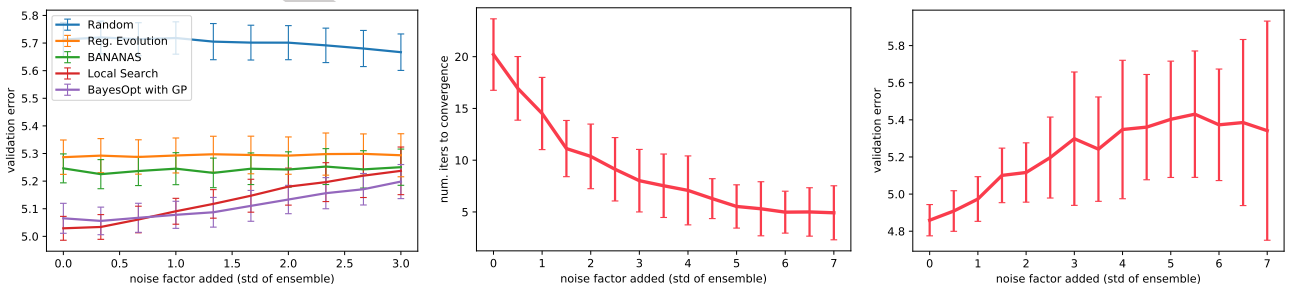

Figure 4.3: Amount of noise present in the architecture evaluation step of NASBench-301 vs. performance of NAS algorithms (left), iterations (middle), and validation error (right).

Discussion. The simple local search algorithm achieves competitive performance on all NAS benchmarks, beating out many popular algorithms. Furthermore, we see a distinct trend across different benchmarks showing that local search performs best (relative to other algorithms) when the noise in the training pipeline is reduced to a minimum. Further experimentation shows that with less noise, there are fewer local minima and local search takes more iterations to converge. These results suggest that NAS becomes substantially easier when the noise is reduced - enough for a very simple algorithm to achieve strong performance. Since local search can be implemented in five lines of code, we encourage local search to be used as a benchmark in future work. We also suggest denoising the noise in the training pipeline. This can be achieved by techniques such as cosine annealing the learning rate [Loshchilov and Hutter, 2016], batch normalization [Ioffe and Szegedy, 2015], and regularization techniques such as dropout [Baldi and Sadowski, 2013] and early-stopping [Prechelt, 1998].

5 THEORETICAL CHARACTERIZATION

In this section, we give a theoretical analysis of local search for NAS, including a complete characterization of its performance. We present a general result which can be applied to any NAS search space. We also give an experimental validation of our results at the end of the section, which suggests that our theoretical results predict the performance of real datasets reasonably well.

In a NAS application, the topology of the search space is fixed and discrete, while the distribution of validation losses is randomized and continuous, due to the non-deterministic nature of training a neural network. Therefore, we assume that the validation loss for a trained architecture is sampled from a global probability distribution, and for each architecture, the validation losses of its neighbors are sampled from a local probability distribution. Recall the definitions of G_N and $\texttt{LS}$ from the end of Section 3. Given a graph $G_N = (A, E_N)$, each node $v \in A$ has a loss $\ell(v) \in \mathbb{R}$ sampled from a PDF which we denote by pdf_n. For any two neighbors $(v, u) \in E_N$, the PDF for the validation loss x of architecture u is given by $\text{pdf}_e(\ell(v), x)$. Choices for the distribution pdf_e are constrained by the fixed topology of the search space, as well as the distribution pdf_n. In Appendix B, we discuss this further by formally defining measurable spaces for all random variables in our framework.

Our main result is a formula for the fraction of nodes in the search space which are local minima, as well as a formula for the fraction of nodes v such that the loss of $\texttt{LS}^*(v)$ is within ϵ of the loss of the global optimum, for all $\epsilon \geq 0$. In other words, we give a formula for the probability that the local search algorithm outputs a solution that is close to optimal. Note that such a formula characterizes the performance of local search. We give the full proofs for all of our results in Appendix B. For the rest of this section, we assume for all $v \in A$, $|N(v)| = s$, and we assume G_N is vertex transitive (given $u, v \in A$, there exists an automorphism of G_N which maps u to v). Let v^* denote the architecture with the global minimum loss, therefore the support of the distribution of validation losses is a subset of $[\ell(v^*), \infty)$. That is, $\int_{\ell(v)}^{\infty} \text{pdf}_n(v)dv = 1$. Technically, the integrals in this section are Lebesgue integrals. However, we use the more standard Riemann-Stieltjes notation for clarity. We also slightly abuse notation and define $\texttt{LS}^{-*}(v) = \texttt{LS}^{-*}(x)$ when $\ell(v) = x$. In the following statements, we assume there is a fixed graph G_N, and the validation accuracies are randomly assigned from a distribution defined by pdf_n and pdf_e. Therefore, the expectations are over the random draws from pdf_n and pdf_e.[2]

Theorem 5.1. Given $|A| = n$, ℓ, s, ϵ, pdf_n, and pdf_e,

$$\mathbb{E}[|\{v \in A \mid \texttt{LS}^*(v) = v\}|] = n \int_{\ell(v^*)}^{\infty} \text{pdf}_n(x) \left(\int_x^{\infty} \text{pdf}_e(x, y)dy \right)^s dx, \text{ and}$$

$$\mathbb{E}[|\{v \in A \mid \ell(\texttt{LS}^*(v)) - \ell(v^*) \leq \epsilon\}|] = n \int_{\ell(v^*)}^{\ell(v^*)+\epsilon} \text{pdf}_n(x) \left(\int_x^{\infty} \text{pdf}_e(x, y)dy \right)^s \cdot \mathbb{E}[|\texttt{LS}^{-*}(x)|]dx.$$

Proof sketch.. To prove the first statement, we introduce an indicator random variable to test if the architecture is a local minimum: $I(v) = \mathbb{I}\{\texttt{LS}^*(v) = v\}$. Then

$$\begin{aligned}\mathbb{E}[|\{v \in A \mid \texttt{LS}^*(v) = v\}|] &= n \cdot P(\{v \in A \mid I(v) = 1\}) \\ &= n \int_{\ell(v^*)}^{\infty} \text{pdf}_n(x) \left(\int_x^{\infty} \text{pdf}_e(x, y)dy \right)^s dx.\end{aligned}$$

Intuitively, in the proof of the second statement, we follow similar reasoning but multiply the probability in the outer integral by the expected size of v's full preimage to weight the integral by the probability a random point will converge to v. Formally, we introduce an indicator random variable on the architecture space that tests if a node will terminate on a local minimum that is within ϵ of the global minimum:

$$I_\epsilon(v) = \mathbb{I}\{\texttt{LS}^*(v) = u \wedge l(u) - l(v^*) \leq \epsilon\}$$

[2] In particular, given a node v with validation loss $\ell(v)$ the probability distribution for the validation loss of a neighbor depends only on $\ell(v)$ and pdf_e, which makes the local search procedure similar to a Markov process. Our experiments in Figure 4.2 suggest this is a reasonable assumption in practice.

We use this random variable along with the first statement of the theorem, to prove the second statement.

$$\begin{aligned}&\mathbb{E}[|\{v \in A \mid \ell(\texttt{LS}^*(v)) - \ell(v^*) \leq \epsilon\}|] \\ &= n \cdot P(\{I_\epsilon = 1\}) \\ &= n \int_{\ell(v^*)}^{\ell(v^*)+\epsilon} \text{pdf}_n(x) \left(\int_{\ell(v)}^{\infty} \text{pdf}_e(x,y)dy \right)^s \\ &\quad \cdot \mathbb{E}[|\texttt{LS}^{-*}(x)|]dx\end{aligned}$$

□

In Appendix B, we use Theorem 5.1 along with Chebyshev's Inequality [Chebyshev, 1867] to show that, in the case where the validation accuracy of each architecture has Gaussian noise, the expected number of local minima can be bounded in terms of the standard deviation of the noise. In the next lemma, we derive a recursive equation for $|\texttt{LS}^{-*}(v)|$. We define the *branching fraction* of graph G_N as $b_k = |N_k(v)| / \left(|N_{k-1}(v)| \cdot |N(v)|\right)$, where $N_k(v)$ denotes the set of nodes which are distance k to v in G_N. For example, the branching fraction of a tree with degree d is 1 for all k, and the branching fraction of a clique is $b_1 = 1$ and $b_k = 0$ for all $k > 1$. One more example is as follows. In Appendix A, we show that the neighborhood graph of the NASBench-201 search space is $(K_5)^6$ and therefore its branching factor is $b_k = \frac{6-k+1}{6k}$.

Lemma 5.2. *Given A, ℓ, s, pdf_n, and pdf_e, then for all $v \in A$, we have the following equations.*

$$\mathbb{E}[|\texttt{LS}^{-1}(v)|] = s \int_{\ell(v)}^{\infty} \text{pdf}_e(\ell(v), y) \cdot \left(\int_{\ell(v)}^{\infty} \text{pdf}_e(y,z)dz \right)^{s-1} dy, \text{ and} \tag{5.1}$$

$$\mathbb{E}[|\texttt{LS}^{-k}(v)|] = b_{k-1} \cdot \mathbb{E}[|\texttt{LS}^{-1}(v)|] \cdot \left(\frac{\int_{\ell(v)}^{\infty} \text{pdf}_e(\ell(v), y)\mathbb{E}[|\texttt{LS}^{-(k-1)}(y)|]dy}{\int_{\ell(v)}^{\infty} \text{pdf}_e(\ell(v), y)dy} \right). \tag{5.2}$$

For some PDFs, it is not possible to find a closed-form solution for $\mathbb{E}[|\texttt{LS}^{-k}(v)|]$ because arbitrary functions may not have closed-form antiderivatives. By assuming there exists a function g such that $\text{pdf}_e(x, y) = g(y)$ for all x, we can use induction to find a closed-form expression for $\mathbb{E}[|\texttt{LS}^{-k}(v)|]$. This includes the uniform distribution ($g(y) = 1$ for $y \in [0, 1]$), as well as distributions that are polynomials in x. In Appendix B, we use this to show that $\mathbb{E}[|\texttt{LS}^{-*}(v)|]$ can be approximated by $1 + s \cdot G(\ell(v))^s \cdot e^{G(\ell(v))^s}$, where $G(x) = \int_x^{\infty} g(y)dy$. Now we use a similar technique to give a closed-form expression for Theorem 5.1 when the local and global distributions are uniform. We stress that this lemma is simply an application of Lemma 5.2, and our main results (Theorem 5.1 and Lemma 5.2) hold without any assumptions on the local and global distributions.

Lemma 5.3. *If $\text{pdf}_n(x) = \text{pdf}_e(x, y) = U([0, 1])\ \forall x \in A$, then $\mathbb{E}[|\{v \mid v = \texttt{LS}^*(v)\}|] = \frac{n}{s+1}$ and*

$$\mathbb{E}[|\{v \mid \ell(\texttt{LS}^*(v)) - \ell(v^*) \leq \epsilon\}|] = n \sum_{i=0}^{\infty} \left(\frac{s^i \left(1 - (1-\epsilon)^{(i+1)s+1}\right)}{(i+1)s+1} \cdot \prod_{j=0}^{i-1} \frac{b_j}{js+1} \right).$$

Proof sketch.. The probability density function of $U([0, 1])$ is equal to 1 on $[0, 1]$ and 0 otherwise. Then $\int_x^{\infty} \text{pdf}_e(x, y)dy = \int_x^1 dy = (1 - x)$. We use this in combination with Theorem 5.1 to prove the first statement:

$$\mathbb{E}[|\{v \mid v = \texttt{LS}^*(v)\}|] = n \int_{\ell(v^*)}^{\infty} 1 \cdot (1-x)^s \, dx = \frac{n}{s+1}.$$

To prove the second statement, first we use induction on the expression in Lemma 5.2 to show that for all $v \in A$,

$$\begin{aligned}\mathbb{E}[|\texttt{LS}^{-*}(v)|] &= \sum_{k=0}^{\infty} \mathbb{E}[|\texttt{LS}^{-k}(v)|] \\ &= \sum_{k=0}^{\infty} \left(s^k (1 - \ell(v))^{sk} \cdot \prod_{i=0}^{k-1} \frac{b_i}{is+1} \right).\end{aligned}$$

We plug this into the second part of Theorem 5.1:

$$\begin{aligned}&\mathbb{E}[|\{v \mid \ell(\texttt{LS}^*(v)) - \ell(v^*) \leq \epsilon\}|] \\ &= n \int_{\ell(v^*)}^{\ell(v^*)+\epsilon} 1 \cdot (1-x)^s \sum_{k=0}^{\infty} \mathbb{E}[|\texttt{LS}^{-k}(x)|]dx \\ &= n \int_{\ell(v^*)}^{\ell(v^*)+\epsilon} (1-x)^s \sum_{k=0}^{\infty} \left(s^k (1-x)^{sk} \prod_{i=0}^{k-1} \frac{b_j}{is+1} \right) dx \\ &= n \sum_{i=0}^{\infty} \left(\frac{s^i \left(1 - (1-\epsilon)^{(i+1)s+1}\right)}{(i+1)s+1} \cdot \prod_{j=0}^{i-1} \frac{b_j}{js+1} \right).\end{aligned}$$

□

In the next section, we show that our theoretical results can be used to predict the performance of local search.

Simulation Results. We run a local search simulation using the equations in the previous section as a means of experimentally validating our theoretical results with real data (we use NASBench-201). In order to use these equations, first we must approximate the local and global probability density functions of the three datasets in NASBench-201. We note that approximating these distributions are not feasible for large search spaces; the purpose of our theoretical results are meant only to provide a deeper understanding of local search and lay the groundwork for future studies. We start by visualizing the probability density functions of the three datasets. See Figure 5.2. We see the most density

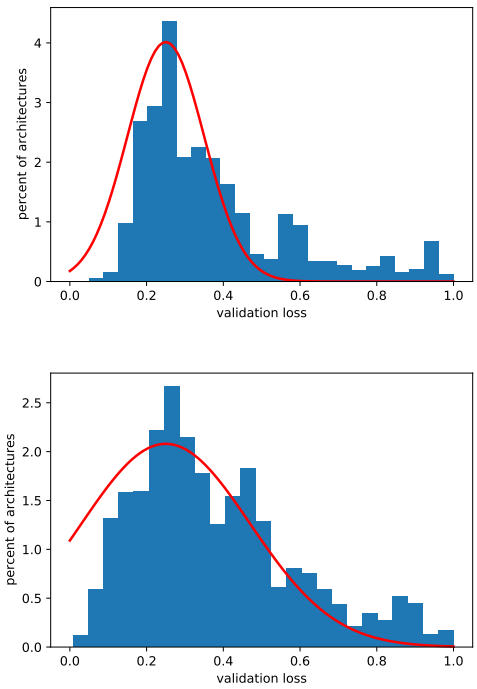

Figure 5.1: Histogram of validation losses for CIFAR-100 (top) and ImageNet16-120 (bottom) in NASBench-201, fitted with the best values of σ and v in Equation 5.3.

along the diagonal, meaning that architectures with similar accuracy are more likely to be neighbors. Therefore, we can approximate the PDFs by using the following equation:

$$\text{pdf}(u) = \frac{\frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{u-v}{\sigma}\right)^2}}{\int_0^1 \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{w-v}{\sigma}\right)^2} dw} \tag{5.3}$$

This is a normal distribution with mean $u - v$ and standard deviation σ, truncated so that it is a valid PDF in $[0, 1]$. We note that prior work has also modeled architecture accuracies in NAS with a normal distribution [Real et al., 2019]. To model the global PDF for each dataset, we plot a histogram of the validation losses and match them to the closest-fitting values of σ and v. See Figure 5.1. The best values of σ are 0.18, 0.1, and 0.22 for CIFAR-10, CIFAR-100, and ImageNet16-120, respectively, and the best values for v are all 0.25. To model the local PDF for each dataset, we compute the random walk autocorrelation (RWA) on each dataset. RWA is defined as the autocorrelation of the accuracies of points visited during a random walk on the neighborhood graph [Weinberger, 1990, Stadler, 1996], and was used to measure locality in NASBench-101 in prior work [Ying et al., 2019]. For the full details of the steps taken to model the datasets in NASBench-201, see Appendix A.

Now we use Theorem 5.1 to compute the probability that a randomly drawn architecture will converge to within ϵ of the global minimum when running local search. Since there is no closed-form solution for the expression in Lemma 5.2, we compute Theorem 5.1 up to the 5th preimage. We compare this to the experimental results on NASBench-201. We also compare the performance of the NASBench-201 search space with validation losses drawn uniformly at random, to the performance predicted by Lemma 5.3. Finally, we compare the preimage sizes of the architectures in NASBench-201 with randomly drawn validation losses to the sizes predicted in Lemma 5.2. See Figure 4.2. Our theory exactly predicts the performance and the preimage sizes of the uniform random NASBench-201 dataset. On the three image datasets, our theory predicts the performance fairly accurately, but is not perfect due to our assumption that the distribution of accuracies is unimodal.

6 CONCLUSION

We show that the difficulty of NAS scales dramatically with the level of noise in the architecture evaluation pipeline, on popular NAS benchmarks (NASBench-101, 201, and 301). In particular, the simplest local search algorithm is sufficient to outperform popular state-of-the-art NAS algorithms when the noise in the evaluation pipeline is reduced to a minimum. We further show that as the noise increases, the number of local minima increases, and the basin of attraction to the global minimum shrinks. This suggests that when the noise in popular NAS benchmarks is reduced to a minimum, the number of local minima decreases, making the loss landscape easy to traverse. Since local search is a simple technique that often gives competitive performance, we encourage local search to be used as a benchmark for NAS in the future. We also suggest denoising the training pipeline whenever possible in future NAS applications.

Motivated by our findings, we give a theoretical study which explains the performance of local search for NAS on different search spaces. We define a probabilistic graph optimization framework to study NAS problems, and we give a characterization of the performance of local search for NAS in our framework. Our results improve the theoretical understanding of local search and lay the groundwork for future studies. We validate this theory with experimental results. Investigating more sophisticated variants of local search for NAS such as Tabu search, simulated annealing, or multi-fidelity local search, are interesting next steps.

ACKNOWLEDGEMENTS

This work done while all authors were employed at Abacus.AI. We thank Willie Neiswanger for his help with this project.

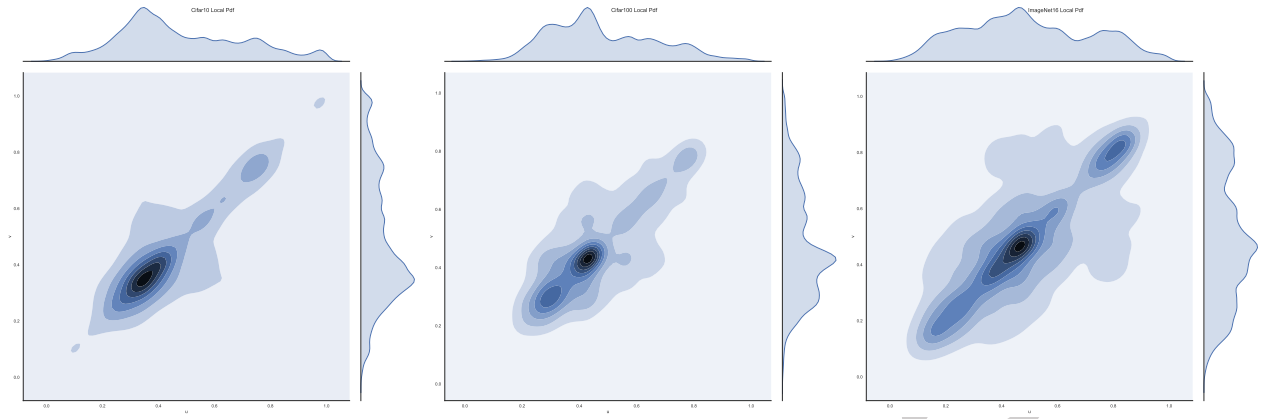

Figure 5.2: Probability density function for CIFAR-10, CIFAR-100, and ImageNet16-120 on NASBench-201. For each coordinate (u, v), a darker color indicates that architectures with accuracy u and v are more likely to be neighbors.

References

E Aarts and JK Lenstra. Local search in combinatorial optimization. *John Wiley & Sons, Inc.*, 1997.

Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. 1948.

Youhei Akimoto, Sandra Astete-Morales, and Olivier Teytaud. Analysis of runtime of optimization algorithms for noisy functions over discrete codomains. *Theoretical Computer Science*, 605:42–50, 2015.

Maria-Florina Balcan, Nika Haghtalab, and Colin White. K-center clustering under perturbation resilience. *ACM Trans. Algorithms*, 16(2), 2020. ISSN 1549-6325.

Pierre Baldi and Peter J Sadowski. Understanding dropout. *Advances in neural information processing systems*, 26: 2814–2822, 2013.

Jon Jouis Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing*, 4(4): 387–411, 1992.

Frederick Bock. An algorithm for solving travelling-salesman and related network optimization problems. In *Operations Research*, volume 6, pages 897–897, 1958.

Andrew J Booker, JE Dennis, Paul D Frank, David B Serafini, and Virginia Torczon. Optimization using surrogate objectives on a helicopter test example. In *Computational Methods for Optimal Design and Control*, pages 49–58. Springer, 1998.

José A Caballero and Ignacio E Grossmann. An algorithm for the use of surrogate models in modular flowsheet optimization. *AIChE journal*, 54(10):2633–2650, 2008.

Pafnutii Lvovich Chebyshev. Des valeurs moyennes. *J. Math. Pures Appl*, 12(2):177–184, 1867.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Vincent Cohen-Addad, Philip N Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 353–364, 2016.

Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 1761–1770, 2019.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

Zachary Friggstad, Mohsen Rezapour, and Mohammad R Salavatipour. Local search yields a ptas for k-means in doubling metrics. *SIAM Journal on Computing*, 48(2): 452–480, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 2018.

David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1(1):215–310, 1997.

David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of computer and system sciences*, 37(1):79–100, 1988.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018.

Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex systems*, 4(4):461–476, 1990.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

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

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Krzysztof Maziarz, Andrey Khorlin, Quentin de Laroussilhe, and Andrea Gesmundo. Evolutionary-neural hybrid agents for architecture search. *arXiv preprint arXiv:1811.09828*, 2018.

Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical aspects of local search*. Springer Science & Business Media, 2007.

Willie Neiswanger, Kirthevasan Kandasamy, Barnabas Poczos, Jeff Schneider, and Eric Xing. Probo: a framework for using probabilistic programming in bayesian optimization. *arXiv preprint arXiv:1901.11515*, 2019.

T Den Ottelander, A Dushatskiy, M Virgolin, and Peter AN Bosman. Local search is a remarkably strong baseline for neural architecture search. *arXiv preprint arXiv:2004.08996*, 2020.

ES Page. An approach to the scheduling of jobs on machines. *Journal of the Royal Statistical Society: Series B (Methodological)*, 23(2):484–492, 1961.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

Pratyusha Rakshit, Amit Konar, and Swagatam Das. Noisy evolutionary optimization algorithms–a comprehensive survey. *Swarm and Evolutionary Computation*, 33:18–45, 2017.

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

Syed Asif Raza Shah, Wenji Wu, Qiming Lu, Liang Zhang, Sajith Sasidharan, Phil DeMar, Chin Guok, John Macauley, Eric Pouyoul, Jin Kim, et al. Amoebanet: An sdn-enabled network service for big data science. *Journal of Network and Computer Applications*, 119:70–82, 2018.

Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T Kwok, and Tong Zhang. Multi-objective neural architecture search via predictive network performance optimization. *arXiv preprint arXiv:1911.09336*, 2019.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

Peter F Stadler. Landscapes and their correlation functions. *Journal of Mathematical chemistry*, 20(1), 1996.

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

Linnan Wang, Yiyang Zhao, Yuu Jinnai, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1805.07440*, 2018.

Edward Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological cybernetics*, 63(5), 1990.

Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. *arXiv preprint arXiv:1912.00848*, 2019.

Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. In *NeurIPS*, 2020.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *AAAI*, 2021a.

Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *arXiv preprint arXiv:2104.01177*, 2021b.

Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? In *NeurIPS*, 2020.

Shen Yan, Kaiqiang Song, Fei Liu, and Mi Zhang. Cate: Computation-aware neural architecture encoding with transformers. In *ICML*, 2021.

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

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.

Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris Holmes, Frank Hutter, and Yee Whye Teh. Neural ensemble search for performant and calibrated predictions. *arXiv preprint arXiv:2006.08573*, 2020.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.