7th ICML Workshop on Automated Machine Learning (2020)

Local Search is State of the Art for NAS Benchmarks

Colin White COLIN@ABACUS.AI
abacus.ai (prev. RealityEngines.AI), San Francisco, CA

Sam Nolen SAM@ABACUS.AI
abacus.ai (prev. RealityEngines.AI), San Francisco, CA

Yash Savani YASH@ABACUS.AI
abacus.ai (prev. RealityEngines.AI), San Francisco, CA

Abstract

Local search is one of the simplest families of algorithms in combinatorial optimization, yet it yields strong approximation guarantees for canonical NP-Complete problems such as the traveling salesman problem and vertex cover. While it is a ubiquitous algorithm in theoretical computer science, local search is often neglected in hyperparameter optimization and neural architecture search.

We show that the simplest local search instantiations achieve state-of-the-art results on multiple NAS benchmarks (NASBench-101 and NASBench-201), outperforming the most popular recent NAS algorithms. However, local search fails to perform well on the much larger DARTS search space. Motivated by these observations, we present a theoretical study which characterizes the performance of local search on graph optimization problems, backed by simulation results. This may be of independent interest beyond NAS. [1]

1. Introduction

Neural architecture search (NAS) is a widely popular area of machine learning, with the goal of automating the development of the best neural network for a given dataset. Hundreds of NAS algorithms have been proposed (Zoph and Le, 2017; Elsken et al., 2018), and with the release of two NAS benchmark datasets (Ying et al., 2019; Dong and Yang, 2020), the extreme computational cost for NAS is no longer a barrier, and it is easier to fairly compare different NAS algorithms. Most of the recently proposed state-of-the-art algorithms are becoming increasingly more complex, many of which use neural networks as subroutines (Wen et al., 2019; White et al., 2019). This trend is problematic because as the complexity of NAS algorithms increases, the amount of necessary "hyper-hyperparameter tuning", or tuning the NAS algorithm itself, increases. Not only is this a vicious cycle (will we start using AutoML algorithms to tune AutoML algorithms?), but the runtime for any hyper-hyperparameter tuning on a new dataset must be added to the total runtime of the NAS algorithm (Lindauer and Hutter, 2019; Yang et al., 2020). Since this information is not always recorded, some NAS algorithms may have under-reported runtimes.

In contrast to prior work, we study a NAS algorithm which can be implemented in five lines of code. Local search is a simple and canonical family of greedy algorithms in combinatorial optimization and has led to famous results in the study of approximation

1. This work was extended to a full-length paper here: `https://arxiv.org/abs/2005.02960`. Our code is available at `https://github.com/realityengines/local_search`.

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

Despite the simplicity of local search, we show that it achieves state-of-the-art results on all four NAS benchmark datasets from NASBench-101 and NASBench-201. However, these benchmark datasets contain at most 4×10^5 architectures. On the DARTS (Liu et al., 2018) search space which contains 10^{18} architectures, local search performed worse than random search. This suggests more generally that strong performance on NAS benchmark datasets does not necessarily imply strong performance on real-world NAS applications.

Motivated by these observations, we present a theoretical study which explains the performance of local search for NAS. We formally define a NAS problem instance by the graph topology, and a probability density function on the architecture accuracies, 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 derive 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 suggests that our theoretical results predict the performance of real datasets reasonably well. Altogether, our results show that the performance of local search depends on the level of *locality* of the search space, as well as the diameter and average neighborhood size in the search space.

Our contributions. We summarize our main contributions below.

- We implement the simple local search algorithm as a baseline for NAS, showing that it achieves state-of-the-art performance on two NASBench datasets (which both have size $< 10^6$) as well as subpar performance on a large search space (of size 10^{18}). We suggest that existing NAS benchmarks may be too small to adequately evaluate NAS algorithms.
- We give a full theoretical characterization of the properties of a dataset necessary for local search to give strong performance. We experimentally validate these results on real datasets. Our results improve the theoretical understanding of local search and lay the groundwork for future studies.

2. Related Work

NAS has gained significant attention in recent years (Zoph and Le, 2017), although the first few techniques have been around since at least the 1990s (Shah et al., 2018; Maziarz et al., 2018). Popular techniques include Bayesian optimization (Kandasamy et al., 2018), reinforcement learning (Pham et al., 2018), gradient descent (Liu et al., 2018), neural prediction (White et al., 2019), and evolution (Real et al., 2019). Prior work has performed local search for NAS using network morphisms, guided by cosine annealing (Elsken et al., 2017). This is a more complex variation of local search for NAS. Very recently, concurrent work has also shown that simple local search is a strong baseline for NAS (Ottelander et al., 2020). Their work considers multi-objective NAS (the objective is a function of accuracy and network complexity), focuses on macro search rather than cell-based search, and gives no theoretical results. The existence of their work strengthens our conclusions, as they were independently and simultaneously verified using different techniques. Recent papers have highlighted the need for fair and reproducible NAS comparisons (Li and Talwalkar, 2019; Lindauer and Hutter, 2019; White et al., 2020), spurring the release of two NAS benchmark datasets (Ying et al., 2019; Dong and Yang, 2020). See the recent survey on NAS (Elsken et al., 2018) for more information. Local search has been studied since at least the 1950s in the context of the traveling salesman problem (Bock, 1958; Croes, 1958) and machine scheduling (Page, 1961). Local search has consistently seen significant attention in theory (Aarts and Lenstra, 1997; Johnson et al., 1988; Balcan et al., 2020) and practice (Bentley, 1992; Johnson and McGeoch, 1997).

3. Preliminaries

In this section, we formally define the local search algorithm, and we define notation that will be used for the rest of the paper. Given a search space A of architectures $v \in A$, denote the objective function as $\ell : A \to [0,1]$. The goal is to find $v^* = \text{argmin}_{v\in A}\ell(v)$, the neural architecture with the minimum validation loss, or else find 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 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 $\forall u \in N(v),\ \ell(v) \leq \ell(u)$. We give pseudocode as Algorithm 1 in the appendix. In Section 5, we also consider two simple variants. In the query_until_lower variant, instead of evaluating every architecture in the neighborhood $N(v)$, we move to the next iteration as soon as we find an architecture $u \in N(v)$ such that $\ell(u) < \ell(v)$. In the continue_at_min variant, we do not stop at a local minimum, instead moving to the second-best architecture found so far.

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 $u \in N(v)$. 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. 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 $\texttt{LS}^{-*}(v) = \{u \mid \exists k \geq 0 \text{ s.t. } \texttt{LS}^{-k}(u) = v\}$. We refer to $\texttt{LS}^{-*}$ as the full preimage of v.

4. A theory of local search

In this section, we give a theoretical analysis of local search for NAS, including a complete characterization of its performance. We present both a general result on local search, as well as a closed-form solution for search spaces satisfying certain constraints. We give an experimental validation of our theoretical results in the next section.

In a NAS application, the topology of the search space is fixed and discrete, while the distribution of validation losses for architectures 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. 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 $\text{pdf}_e(\ell(v), x)$. In Appendix A, we discuss this further by formally defining measurable spaces for all random variables.

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 $\epsilon \geq 0$. We give the full proofs for all of our results in Appendix A. 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. We slightly abuse notation and define $\texttt{LS}^{-*}(v) = \texttt{LS}^{-*}(x)$ when $\ell(v) = x$. In the following theorems and lemmas, 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. 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.

Theorem 1 *Given $|A| = n$, ℓ, s, ϵ, pdf_n, and pdf_e, we have*

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

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

Intuitively, to compute the probability that local search outputs a near-optimal solution, we first compute the probability a point v is a local minimum by integrating $\text{pdf}_e(\ell(v), y)$ from $\ell(v)$ to ∞ over y and raising it to the power $|N(v)| = s$. We multiply this probability by the expected size of v's full preimage to give us the size of the set of nodes that will converge to v. Then we integrate this probability over all values of $\ell(v)$ drawn from pdf_n.

In Appendix A, we prove that $\mathbb{E}[|\texttt{LS}^{-1}(v)|] = s \int_{\ell(v)}^{\infty} \text{pdf}_e(\ell(v), y) \left(\int_{\ell(v)}^{\infty} \text{pdf}_e(y, z) dz \right)^{s-1} dy$, and we give a recursive expression for $\mathbb{E}[|\texttt{LS}^{-k}(v)|]$. 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 achieve 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 A, we use this to show that $\mathbb{E}[|\texttt{LS}^{-*}(v)|]$ can be approximated nearly exactly by $1 + s \cdot G(\ell(v))^s \cdot e^{G(\ell(v))^s}$, where $G(x) = \int_x^{\infty} g(y) dy$. This allows us to give a closed-form expression for Theorem 1 when the local and global distributions are uniform.

Lemma 2 *If* $pdf_n(x) = pdf_e(x,y) = U([0,1])\ \forall x \in A$, *then* $\mathbb{E}[|\{v \mid v = LS^*(v)\}|] = \frac{n}{s+1}$ *and*

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

5. Experiments

In this section, we discuss our experimental setup and results. We use two benchmark NAS datasets and one popular large-scale search space for experiments. The NASBench-101 dataset (Ying et al., 2019) consists of over 423,000 neural architectures pretrained on CIFAR-10, while the NASBench-201 dataset (Dong and Yang, 2020) contains over 15,000 neural architectures pretrained on CIFAR-10, CIFAR-100, and ImageNet-16-120. The DARTS search space (Liu et al., 2018) consists of over 10^{18} architectures and is popular for experiments on CIFAR-10. To promote reproducible research, we discuss how our experiments follow the best practices checklist (Lindauer and Hutter, 2019) in Appendix B.

Local search performance. We evaluate the effectiveness of local search for NAS. First we compare local search to other NAS algorithms on the four benchmark dataset/search space pairs. On the three NASBench-201 datasets, we compare local search to random search, DNGO (Snoek et al., 2015), Regularized Evolution (Real et al., 2019), Bayesian Optimization, BANANAS (White et al., 2019), and NASBOT (Kandasamy et al., 2018). On NASBench-101, we test local search with the aforementioned algorithms, as well as REINFORCE (Williams, 1992) and AlphaX (Wang et al., 2018). For every algorithm, we used the code directly from the corresponding open source repositories and kept the hyperparameters largely unchanged. For more details, see Section B. We ran 200 trials of each algorithm and averaged the results. On NASBench-101 and ImageNet-16-120, we used the query_until_lower variant of local search, and on NASBench-201 CIFAR-10 and CIFAR-100, we used the continue_at_min variant. (In Appendix B, we evaluate all four variants.) See Figure 1. Local search consistently performs the strongest on all four datasets.

Next, we evaluate local search with the query_until_lower variant on the DARTS search space. We ran one trial training all queried architecture to 25 epochs, and another trial training to 50 epochs. We trained the final returned architectures for 600 epochs, using the same training pipeline as prior work (Li and Talwalkar, 2019; Liu et al., 2018). The final test error percent was 3.93 (25 epochs) and 3.49 (50 epochs), compared to 3.29 (Random Search) and 2.68 (DARTS). For a table of results, see Appendix B. One reason for the subpar performance is because the degree of the neighborhood graph is 136, much larger than NASBench-201's 24. For instance, in the 50 epoch trial, 100 queries was not sufficient to get through a single iteration of local search even with the query_until_lower variant.

Simulation Results. We run a local search simulation using the equations in Theorem 1 and Lemma 2, to show that our theoretical results can approximate the datasets in NASBench-201. We focus on modeling the dataset with simpler functions that may be easier to approximate. In particular, we approximate both pdf_n and pdf_e using a normal distribution with mean $u - v$ and standard deviation σ. For each dataset, we approximate the best standard deviation value in the global distribution by aligning the normal distribution with the global histogram. We approximate the best standard deviation value in the local

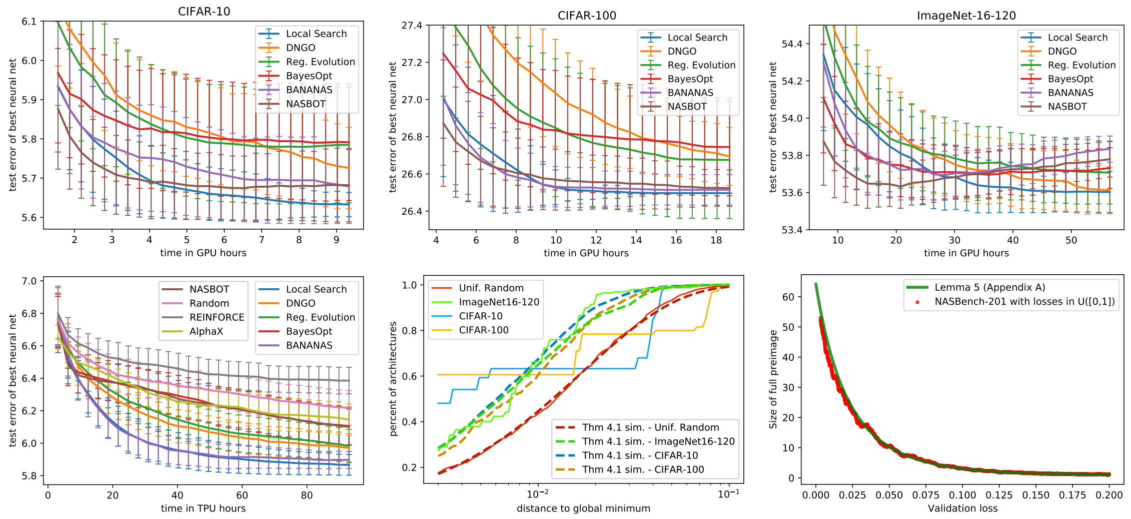

Figure 1: Results on NASBench-201 (top) and NASBench-101 (bottom left). Probability that local search will converge to within ϵ of the global optimum, compared to Theorem 1 (bottom middle). Validation loss vs. size of preimages, compared to theory (bottom right).

distribution by computing the random walk autocorrelation (RWA) on each dataset. RWA is defined as the autocorrelation of the accuracies of points visited during a walk of random single changes through the search space (Weinberger, 1990; Ying et al., 2019). For the full details, see Appendix B. Then we use Theorem 1 to compute the probability that a randomly drawn architecture will converge to within ϵ of the global minimum, when running local search. 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 2. Finally, we compare the preimage sizes of the architectures in NASBench-201 with randomly drawn validation losses to the sizes predicted in Appendix A. See Figure 1. 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 simplest local search algorithm achieves state-of-the-art results on the most popular existing NAS benchmarks (NASBench-101 and -201). We also show that it has subpar performance on the DARTS search space, suggesting that the NAS benchmarks may be too simple and/or small to adequately evaluate NAS methods. We give a theoretical analysis, characterizing the conditions under which local search performs well for NAS, which may be of independent interest. Since local search can be implemented in five lines of code, we encourage local search to be used as a NAS benchmark in future work.

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.

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

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.

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

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.

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.

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.

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

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

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.

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.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.

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

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, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv preprint arXiv:1910.11858*, 2019.

Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. *arXiv preprint arXiv:2007.04965*, 2020.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, pages 229–256, 1992.

Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

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.

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

Appendix A. Details from Section 4

In this section, we give details from Section 4. For pseudocode of local search, see Algorithm 1. For convenience, we restate all theorems and lemmas here.

Algorithm 1 Local search

```
Input: Search space $A$, objective function $\ell$, 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 $v_i$
```

We start by giving a rigorous grounding of our theoretical framework. Recall that the topology of the search space is fixed and discrete, while the distribution of validation losses for architectures is randomized and continuous. This is because training a neural network is not deterministic; in fact, both NASBench-101 and NASBench-201 include validation and test accuracies for three different random seeds for each architecture, to better simulate real NAS experiments. 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.

Let $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ denote a measurable space for the global validation losses induced by the dataset on the architectures, where $\mathcal{B}(\mathbb{R})$ is the Borel σ-algebra on $\mathbb{R}$. The distribution for the validation loss of any architecture in the search space is given by $\text{pdf}_n(x) \forall x \in \mathbb{R}$.

Let $(\mathbb{R}^2, \mathcal{B}(\mathbb{R}^2))$ denote a measurable space for validation losses in a neighborhood of an architecture. Let $E : \mathbb{R}^2 \to \mathbb{R}$ denote a random variable mapping the validation losses of two neighboring architectures to the loss of the second architecture, $E(x, y) \mapsto y$. E has a distribution that is characterized by probability density function $\text{pdf}_e(x, y) \forall x, y \in \mathbb{R}$. This gives us a probability over the validation loss for a neighboring architecture.

Every architecture $v \in A$ has a loss $\ell(v) \in \mathbb{R}$ that is sampled from pdf_n. For any two neighbors $(v, u) \in E_N$, the PDF for the validation loss x of architecture u is given by $\text{pdf}_e(\ell(v), x)$. Note that choices for the distribution pdf_e are constrained by the fixed topology of the search space, as well as the selected distribution pdf_n.

Let $(A, 2^A)$ denote a measurable space over the nodes of the graph. The following theorems and lemmas describe probabilities over sets of the nodes based on the associated global validation loss and neighborhood validation loss probability spaces defined above.

Theorem 1 *Given $|A| = n$, ℓ, s, ϵ, pdf_n, and pdf_e, we have*

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

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

Proof To prove the first statement, we introduce an indicator random variable on the architecture space to test if the architecture is a local minimum $I : A \to \mathbb{R}$, where

$$\begin{aligned} I(v) &= \mathbb{I}\{\texttt{LS}^*(v) = v\} \\ &= \mathbb{I}\{\ell(v) < \ell(u)\ \forall u \text{ s.t. } (u,v) \in E_N\}. \end{aligned}$$

The expected number of local minima in $|A|$ is equal to $|A|$ times the fraction of nodes in A which are local minima. Therefore, we have

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

In line one we use the notation $\mathcal{P}(\{I = 1\}) \equiv \mathcal{P}(\{v \in A \mid I(v) = 1\})$.

To prove the second statement, 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, $I_\epsilon : A \to \mathbb{R}$, where

$$\begin{aligned} I_\epsilon(v) &= \mathbb{I}\{\texttt{LS}^*(v) = u \wedge l(u) - l(v^*) \le \epsilon\} \\ &= \mathbb{I}\{\exists S \in \{\texttt{LS}^{-*}(u) : \texttt{LS}^*(u) = u \wedge l(u) - l(v^*) \le \epsilon\}, v \in S\} \end{aligned}$$

We use this random variable to prove the second statement of the theorem.

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

where the last equality follows from the first half of this theorem. This concludes the proof. ■

Lemma 3 *Given A, ℓ, s, pdf_n, and pdf_e, then for all $v \in A$, we have the following equations.*

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

$$\mathbb{E}[|LS^{-m}(v)|] = \mathbb{E}[|LS^{-1}(v)|] \left(\frac{\int_{\ell(v)}^{\infty} pdf_e(\ell(v), y) \mathbb{E}[|LS^{-(m-1)}(y)|] dy}{\int_{\ell(v)}^{\infty} pdf_e(\ell(v), y) dy} \right). \tag{2}$$

Proof The function $\texttt{LS}^{-1}(v) \in 2^A$ returns a set of nodes which form the preimage of node $v \in A$, namely, the set of all neighbors $u \in N(v)$ with higher validation loss than v, and whose neighbors $w \in N(u)$ excluding v have higher validation loss than $\ell(v)$. Formally,

$$\begin{aligned}\texttt{LS}^{-1}(v) &= \{u \in A \mid \texttt{LS}(u) = v\} \\ &= \{u \in A \mid (v,u) \in E_N, \ell(v) < \ell(u), \{v' \in A \backslash \{v\} \mid (v',u) \in E_N, \ell(v') < \ell(v)\} = \emptyset\}.\end{aligned}$$

Let $\texttt{LS}_v^{-1} : A \to \mathbb{R}$ denote a random variable where $\texttt{LS}_v^{-1}(u) = \mathbb{I}\{u \in \texttt{LS}^{-1}(v)\}$. The probability distribution for $\texttt{LS}_v^{-1}$ gives the probability that a neighbor of v is in the preimage of v. We can multiply this probability by $|N(v)| = s$ to express the expected number of nodes in the preimage of v.

$$\begin{aligned}\mathbb{E}[|\texttt{LS}^{-1}(v)|] &= s \cdot \mathcal{P}(\{\texttt{LS}_v^{-1} = 1\}) \\ &= s \int_{\ell(v)}^{\infty} \text{pdf}_e(\ell(v), y) \left(\int_{l(v)}^{\infty} \text{pdf}_e(y,z) dz \right)^{s-1} dy.\end{aligned}$$

Note that the inner integral is raised to the power of $s-1$, not s, so as not to double count node v. We can use this result to find the preimage of node v after m steps. Let $\texttt{LS}_v^{-m} : A \to \mathbb{R}$ denote a random variable where

$$\begin{aligned}\texttt{LS}_v^{-m}(u) &= \mathbb{I}\{u \in \texttt{LS}^{-m}(v)\} \\ &= \mathbb{I}\{\forall w \in \texttt{LS}^{-1}(v), u \in \texttt{LS}^{-(m-1)}(w)\}.\end{aligned}$$

Following a similar argument as above, we compute the expected size of the m'th preimage set.

$$\begin{aligned}\mathbb{E}[|\texttt{LS}^{-m}(v)|] &= \mathbb{E}[|\texttt{LS}^{-1}(v)|] \cdot \mathbb{E}[|\{\forall w \in A \mid \forall u \in \texttt{LS}^{-1}(v), \texttt{LS}_u^{-(m-1)}(w) = 1\}|] \\ \mathbb{E}[|\texttt{LS}^{-m}(v)|] &= \mathbb{E}[|\texttt{LS}^{-1}(v)|] \left(\frac{\int_{\ell(v)}^{\infty} \text{pdf}_e(\ell(v), y) \mathbb{E}[|\texttt{LS}^{-(m-1)}(y)|] dy}{\int_{\ell(v)}^{\infty} \text{pdf}_e(\ell(v), y) dy} \right)\end{aligned}$$

■

Closed-form solution for single-variate PDFs. Now we give the details for Lemma 2. We start with a lemma that will help us prove Lemma 2. This lemma uses induction to derive a closed-form solution to Lemma 3 in the case where $\text{pdf}_e(x,y)$ is independent of x.

Lemma 4 *Assume there exists a function g such that $\text{pdf}_e(x,y) = g(y)$ for all x. Given $v \in A$, for $m \geq 1$,*

$$\mathbb{E}[|LS^{-m}(v)|] = s^m \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{sm} \cdot \prod_{i=0}^{m-1} \frac{1}{is+1}.$$

Proof Given $v \in A$,

$$\begin{aligned}
\mathbb{E}[|\texttt{LS}^{-1}(v)|] &= s \int_{\ell(v)}^{\infty} \text{pdf}_e(\ell(v), y) \left(\int_{\ell(v)}^{\infty} \text{pdf}_e(y,z) dz \right)^{s-1} dy \\
&= s \int_{\ell(v)}^{\infty} g(y) \left(\int_{\ell(v)}^{\infty} g(z) dz \right)^{s-1} dy \\
&= s \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{s},
\end{aligned}$$

where the first equality follows from Lemma 3. Now we give a proof by induction for the closed-form equation. The base case, $m = 1$, is proven above. Given an integer $n \geq 1$, assume that

$$\mathbb{E}[|\texttt{LS}^{-n}(v)|] = s^n \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{sn} \cdot \prod_{i=0}^{n-1} \frac{1}{is+1}.$$

Then

$$\begin{aligned}
\mathbb{E}[|\texttt{LS}^{-(n+1)}(v)|] &= \mathbb{E}[|\texttt{LS}^{-1}(v)|] \cdot \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{-1} \int_{\ell(v)}^{\infty} g(y) \mathbb{E}[|\texttt{LS}^{-n}(y)|] dy \\
&= s \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{s} \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{-1} \int_{\ell(v)}^{\infty} g(y) \cdot \mathbb{E}[|\texttt{LS}^{-n}(y)|] dy \\
&= s \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{s-1} \int_{\ell(v)}^{\infty} g(y) \cdot s^n \left(\int_{y}^{\infty} g(z) dz \right)^{sn} \cdot \prod_{i=0}^{n-1} \frac{1}{is+1} \cdot dy \\
&= s^{n+1} \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{s-1} \cdot \prod_{i=0}^{n-1} \frac{1}{is+1} \int_{\ell(v)}^{\infty} g(y) \left(\int_{y}^{\infty} g(z) dz \right)^{sn} dy \\
&= s^{n+1} \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{s-1} \cdot \prod_{i=0}^{n-1} \frac{1}{is+1} \left(\int_{\ell(v)}^{\infty} g(z) dz \right)^{sn+1} \frac{1}{sn+1} \\
&= s^{n+1} \left(\int_{\ell(v)}^{\infty} g(y) dy \right)^{s(n+1)} \cdot \prod_{i=0}^{n} \frac{1}{is+1}.
\end{aligned}$$

In the first equality, we used Lemma 3, and in the fourth equality, we used the fact that

$$\frac{\partial}{\partial y} \left(\int_{y}^{\infty} g(z) dz \right)^{sn+1} = g(y) \left(\int_{y}^{\infty} g(z) dz \right)^{sn} (sn+1).$$

This concludes the proof. ■

Next, we prove a lemma which gives strong approximation guarantees on the size of the full preimage of an architecture, again assuming that $\text{pdf}_e(x, y)$ is independent if x.

Lemma 5 *Assume there exists g such that $pdf_e(x,y) = g(y)$ for all x. Denote $G(x) = \int_x^\infty g(y)dy$. For all v, we have*

$$1 + s \cdot G(\ell(v))^s e^{\frac{s}{s+1}G(\ell(v))^s} \leq \mathbb{E}[|LS^{-*}(v)|] \leq 1 + s \cdot G(\ell(v))^s \cdot e^{G(\ell(v))^s}.$$

Proof From Lemma 4, we have

$$\mathbb{E}[|\mathtt{LS}^{-*}(v)|] = \sum_{m=1}^{\infty} \mathbb{E}[|\mathtt{LS}^{-m}(v)|] = \sum_{m=1}^{\infty} \left(s^m G(\ell(v))^{sm} \cdot \prod_{i=0}^{m-1} \frac{1}{is+1} \right). \tag{3}$$

We start with the lower bound. For all $j \geq 1$, $\frac{s}{j(s+1)} \leq \frac{s}{js+1}$. Therefore, for all i,

$$\frac{s^{i-1}}{(i-1)!(s+1)^{i-1}} = \prod_{j=1}^{i-1} \frac{s}{j(s+1)} \leq \prod_{j=1}^{i-1} \frac{s}{js+1}.$$

It follows that

$$\begin{aligned}
\mathbb{E}[|\mathtt{LS}^{-*}(v)|] &= \sum_{i=1}^{\infty} s^i G(\ell(v))^{si} \prod_{j=0}^{i-1} \frac{1}{js+1} \\
&= s \sum_{i=1}^{\infty} G(\ell(v))^{si} \prod_{j=1}^{i-1} \frac{s}{js+1} \\
&\geq sG(\ell(v))^s \sum_{i=1}^{\infty} \left(\frac{1}{(i-1)!} \cdot \left(\frac{sG(\ell(v))^s}{s+1} \right)^{i-1} \right) \\
&= sG(\ell(v))^s \cdot e^{\frac{s}{s+1}G(\ell(v))^s}.
\end{aligned}$$

The final equality comes from the well-known Taylor series $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ (e.g. Abramowitz and Stegun (1948)) evaluated at $x = \frac{s}{s+1} \cdot G(\ell(v))^s$.

Now we prove the upper bound. For all $j \geq 1$, $\frac{s}{js+1} \leq \frac{s}{sj} = \frac{1}{j}$. Therefore for all i,

$$\prod_{j=1}^{i-1} \frac{s}{js+1} \leq \prod_{j=1}^{i-1} \frac{1}{j} = \frac{1}{(i-1)!}.$$

It follows that

$$\begin{aligned}
\mathbb{E}[|\mathtt{LS}^{-m}(v)|] &= \sum_{i=1}^{\infty} s^i G(\ell(v))^{si} \prod_{j=0}^{i-1} \frac{1}{js+1} \\
&= s \sum_{i=1}^{\infty} G(\ell(v))^s \prod_{j=1}^{i-1} \frac{s}{js+1} \\
&\leq sG(\ell(v))^s \sum_{i=1}^{\infty} \left(\frac{1}{(i-1)!} \cdot G(\ell(v))^s \right) \\
&= sG(\ell(v))^s e^{G(\ell(v))^s}.
\end{aligned}$$

The final equality again comes from evaluating the Taylor series for e^x. ■

Now we use Lemma 5 and Theorem 1 to prove Lemma 2.

Lemma 2 *If $pdf_n(x) = pdf_e(x, y) = U([0, 1])\ \forall x \in A$, then $\mathbb{E}[|\{v \mid v = LS^*(v)\}|] = \frac{n}{s+1}$ and*

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

Proof

The probability density function of $U([0, 1])$ is equal to 1 on $[0, 1]$ and 0 otherwise. Let $\ell(v) = x$. Then $\int_x^\infty \mathrm{pdf}_e(x, y)dy = \int_x^1 dy = (1 - x)$. Using Theorem 1, we have

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

Now we plug in Lemma 4 to the second part of Theorem 1.

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

■

Appendix B. Details from Section 5

In this section, we give details and supplementary results for Section 5. We start by giving full details for all search spaces used in our experiments.

NASBench-101 (Ying et al., 2019). The NASBench-101 benchmark dataset consists of over 423,000 unique neural architectures from a cell-based search space, and each architecture comes with precomputed validation, and test accuracies for 108 epochs on CIFAR-10.

The search space consists of a cell with 7 nodes. The first node is the input, and the last node is the output. The remaining five nodes can be either 1×1 convolution, 3×3 convolution, or 3×3 max pooling. The cell can take on any DAG structure from the input to the output with at most 9 edges. The hyper-architecture consists of nine cells stacked sequentially, with each set of three cells separated by downsampling layers. The first layer before the first cell is a convolutional layer, and the hyper-architecture ends with a global average pooling layer and a fully connected layer.

The neighbors of a cell consist of the set of all cells that differ by one operation or edge. Since each cell can have between 1 and 9 edges, the number of neighbors of a cell can range from 1 to 29.

NASBench-201 (Dong and Yang, 2020). The NASBench-201 dataset consists of over $15,000$ unique neural architectures, with precomputed training, 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* takes an operation, and there are five possible operations: 1×1 convolution, 3×3 convolution, 3×3 avg. pooling, skip connect, or none. The hyper-architecture consists of 15 cells stacked sequentially, with each set of five cells separated by residual blocks. The first layer before the first cell is a convolution layer, and the hyper-architecture ends with a global average pooling layer and a fully connected layer.

Since every cell has exactly 6 edges, the total number of possible cells is $5^6 = 15625$. We say that two cells are neighbors if they differ by exactly one operation. Then the diameter of the neighborhood graph is 6, because any cell can reach any other cell by swapping out all 6 of its operations. Each cell has exactly 24 neighbors, because there are 6 edges, and each edge has 4 other choices for an operation. The neighborhood graph is $(K_5)^6$, that is, the Cartesian product of six cliques of size five.

DARTS search space (Liu et al., 2018). The DARTS search space is a popular search space for large-scale NAS experiments. It is a convolutional cell-based search space used for CIFAR-10. The search space consists of two cells, a convolutional cell and a reduction cell, each with six nodes. The hyper-architecture stacks k convolutional cells together with one reduction cell. For each cell, the first two nodes are the outputs from the previous two cells in the hyper-architecture. The next four nodes contain two edges as input, creating a DAG. Each edge can take on one of seven operations.

B.1 Details and additional local search experiments

In this section, we give details and additional experiments for local search on the datasets described above.

For every benchmark NAS algorithm, we used the code directly from its corresponding open-source repository. For regularized evolution, we changed the population size from 50 to 30 to account for fewer queries. For NASBOT, which was designed for macro (non cell-based) NAS, we computed the distance between two cells by taking the earth-mover's distance between the set of row sums, column sums, and node operations, similar to (White et al., 2019). We did not change any hyperparameters for the other baseline algorithms. For vanilla Bayesian optimization, we used the ProBO implementation (Neiswanger et al., 2019). Our experimental setup is the same as prior work (e.g., (Ying et al., 2019)). At each timestep t, we report the test error of the architecture with the best validation error found so far, and we run 200 trials of each algorithm and average the result.

Now we evaluate all four variants of local search on all three NASBench-201 datasets. See Figure 2. See Section 4 for an explanation of the query_until_lower and continue_at_min variants to local search.

Now we show the full table of results for the experiments on the DARTS search space from Section 5. See Table 1.

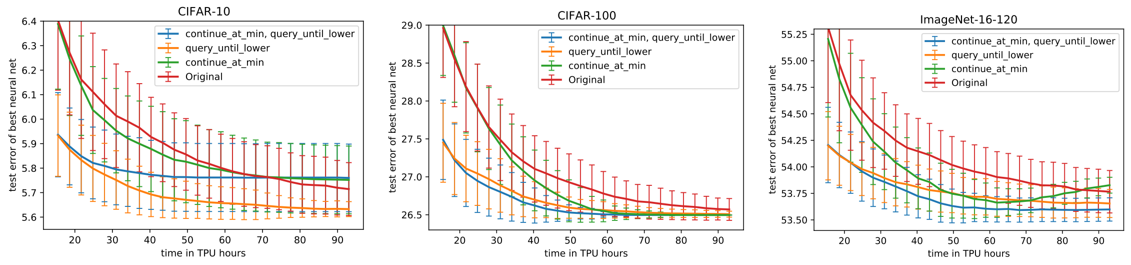

Figure 2: Results for local search variants on CIFAR-10 (left), CIFAR-100 (middle), and ImageNet-16-120 (right) on NASBench-201.

Table 1: Test error percent of the best architectures returned by several NAS algorithms. The runtime is in total GPU-days on a Tesla V100.

NAS Algorithm	Source	Test error	Queries	Runtime
Random search	(Liu et al., 2018)	3.29		4
DARTS	(Liu et al., 2018)	2.68		5
ASHA	(Li and Talwalkar, 2019)	3.08	700	9
BANANAS	(White et al., 2019)	2.64	100	11.8
Local Search 50 epochs	Ours	3.49	100	11.8
Local Search 25 epochs	Ours	3.93	200	11.8

B.2 Details from simulation experiments

In this section, we give more details for our simulation experiment described in Section 5.

We start by visualizing the probability density functions of the three datasets in NASBench-201. See Figure 3. We see more density along the diagonal, meaning that architectures with similar accuracy are more likely to be neighbors. We also see that the PDF is multimodal.

The function we use to approximate the datasets on NASBench-201 is shown below.

$$\text{pdf}(u) = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{u-v}{\sigma}\right)^2} \cdot \left(\int_0^1 \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{w-v}{\sigma}\right)^2} dw\right)^{-1} & \text{if } u \in [0,1], \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

This is a normal distribution with mean $u - v$ and standard deviation of σ, truncated so that it is a valid PDF in $[0, 1]$. In order to choose an appropriate probability density function for modelling the datasets in NASBench-201, we approximate the σ values for both the local and global PDFs.

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. The best values are $\sigma = 0.18,\ 0.1,$ and 0.22 for CIFAR-10, CIFAR-100, and ImageNet16-120, respectively.

Now we plot the random-walk autocorrelation (RWA) described in Section 5. Recall that RWA is defined as the autocorrelation of the accuracies of points visited during a walk of random single changes through the search space (Weinberger, 1990; Stadler, 1996), and was

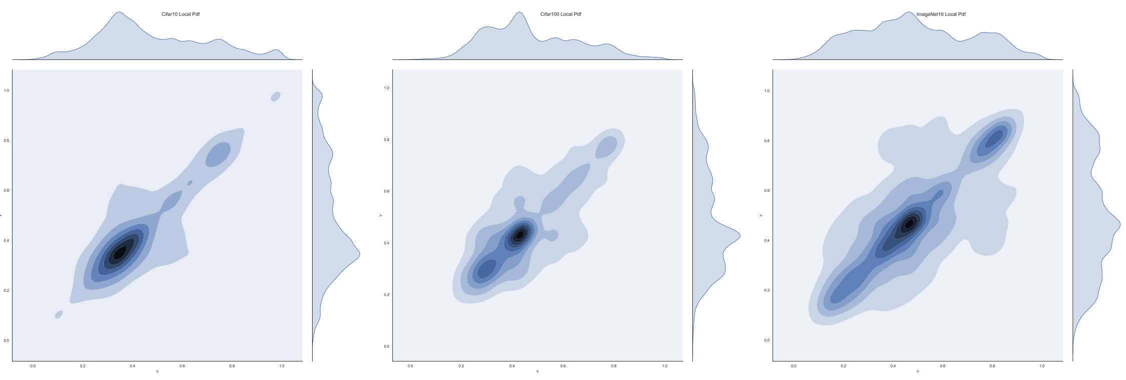

Figure 3: Probability density function for CIFAR-10 (left), CIFAR-100 (middle), and ImageNet16-120 (right) on NASBench-201. For each coordinate (u, v), a darker color indicates that architectures with accuracy u and v are more likely to be neighbors.

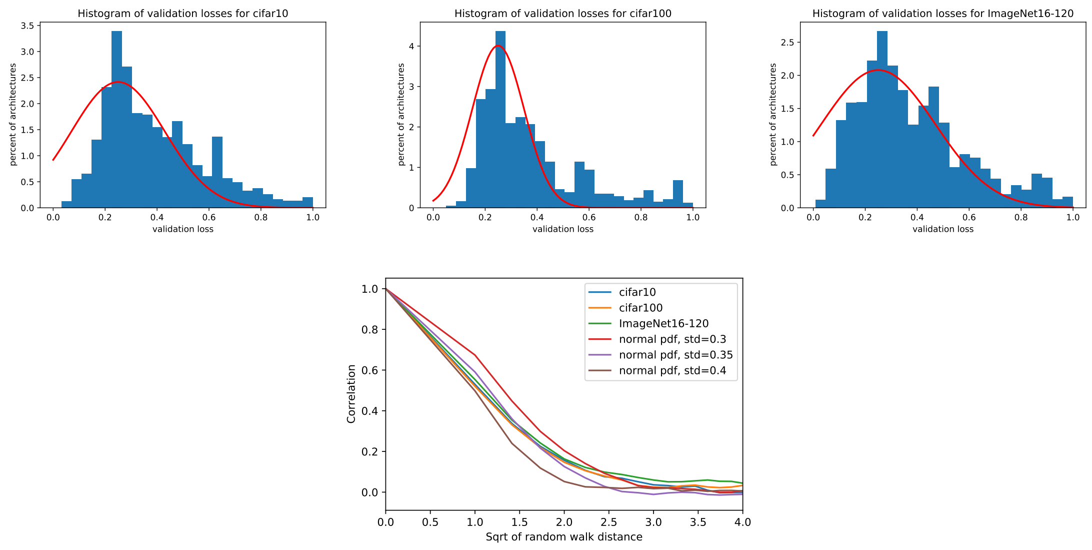

Figure 4: Histogram of validation losses for the three datasets in NASBench-201, fitted with the best values of σ and v in Equation 4 (top). RWA vs. distance for three datasets in NASBench-201, as well as three values of σ in Equation 4 (bottom).

used to measure locality in NASBench-101 in prior work (Ying et al., 2019). We compute the RWA for all three datasets in NASBench-201, by performing a random walk of length 100,000. See Figure 4. We also compute the RWA for Equation 4, and match each dataset with the closest value of σ. We see that a value of $\sigma = 0.35$ is the closest match for all three datasets.

Now for each of the three NASBench-201 datasets, we have estimated the pdf_e and pdf_n distributions. We plug each $(\text{pdf}_e, \text{pdf}_n)$ pair into Theorem 1, which gives a plot of ϵ vs. percent of architectures that converge to within ϵ of the global optimum after running local search. We compare these to the true plot in Figure 1. For the random simulation, we are modeling the case where $\text{pdf}_e = \text{pdf}_n = U([0, 1])$, so we can use Lemma 2 directly.

B.3 Best practices for NAS research

The area of NAS research has had issues with reproducibility and fairness in empirical comparisons (Li and Talwalkar, 2019; Ying et al., 2019), and there is now a checklist for best practices (Lindauer and Hutter, 2019). In order to promote best practices, we discuss each point on the list, and encourage all NAS research papers to do the same.

- **Releasing code.** We have released our code anonymously at `https://github.com/realityengines/local_search`. The training pipelines and search spaces are from popular existing NAS work: NASBench-101, NASBench-201, and DARTS. One thing that is missing is the set of random seeds we used for the DARTS experiments.
- **Comparing NAS methods.** We made fair comparisons due to our use of NASBench-101 and NASBench-201. For baseline comparisons, we used open-source code, a few times adjusting hyperparameters to be more appropriate for the search space. We ran ablation studies, compared to random search, and compared performance over time. We performed 200 trials on tabular benchmarks.
- **Reporting important details.** Local search only has two boolean hyperparameters, so we did not need to tune hyperparameters. We reported the times for the full NAS method and all details for our experimental setup.