Graph is All You Need? Lightweight Data-agnostic Neural Architecture Search without Training

Zhenhan Huang[1] **Tejaswini Pedapati**[2] **Pin-Yu Chen**[2] **Chunheng Jiang**[1] **Jianxi Gao**[1]

[1]Rensselaer Polytechnic Institute
[2]IBM Thomas J. Watson Research Center

Abstract Neural network search (NAS) automates the design of neural architectures. However, training the candidates generated by the search algorithm for performance evaluation incurs considerable computational overhead. We propose a method dubbed NASGraph that remarkably reduces computational costs by converting neural architectures to graphs and searching in the graph space. We empirically find a graph measure average degree, despite its simplicity, powerful enough as the NAS proxy in lieu of the evaluation metric. Our proposed method is training-free, data-agnostic, and lightweight. Besides, our method is able to achieve competitive performance on various NAS benchmarks including NASBench-101, NASBench-201, and NDS. We also demonstrate that NASGraph generalizes to more challenging tasks on Micro TransNAS-Bench-101.

1 Introduction

The objective of NAS is to find an optimal neural architecture $a^* = \arg\min_{a \in \mathcal{A}} f(a)$, where $f(a)$ denotes the performance (e.g., a task-specific loss function) of the neural architecture a trained for a fixed number of epochs using a dataset, and $\mathcal{A}$ is the search space. The recent emergence of training-free NAS Mellor et al. (2021); Abdelfattah et al. (2021) pushes the boundary of efficient NAS techniques further and greatly eases the computational burden. Training-Free NAS computes a proxy metric in place of accuracy to rank the candidate architectures. The proxy metric is obtained by a single forward/backward propagation using a training dataset.

In this paper, we take a novel perspective on mapping neural networks to graphs: we treat inputs to neural components as graph nodes and use relationship between inputs to neighboring graph components to determine the connectivity between pairs of graph nodes. In this way, we are able to convert a neural architecture to a DAG $G(V, E)$ where node set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_{ij}\}\ \forall i, j$ s.t. there exists an edge from v_i to v_j. After establishing the mapping, we extract the associated graph measures as NAS metrics to rank neural architectures. We note that the entire process is training-free and data-agnostic (i.e. do not require any training data).

We summarize our **main contributions** as follows:

- We propose *NASGraph*, a training-free and data-agnostic method for NAS. *NASGraph* maps the neural architecture space to the graph space.
- Using the extracted graph measures for NAS, *NASGraph* achieves competitive performance on NAS-Bench-101, NAS-Bench-201, Micro TransNAS-Bench-101 and NDS benchmarks.
- In comparison to existing training-free NAS techniques, we show that the computation of *NASGraph* is lightweight (only requires CPU).

2 Related Work

One-shot NAS. One-shot NAS constructs a supernet subsuming all candidate architectures in the search space. In other words, subnetworks of the supernet are candidate architectures. The

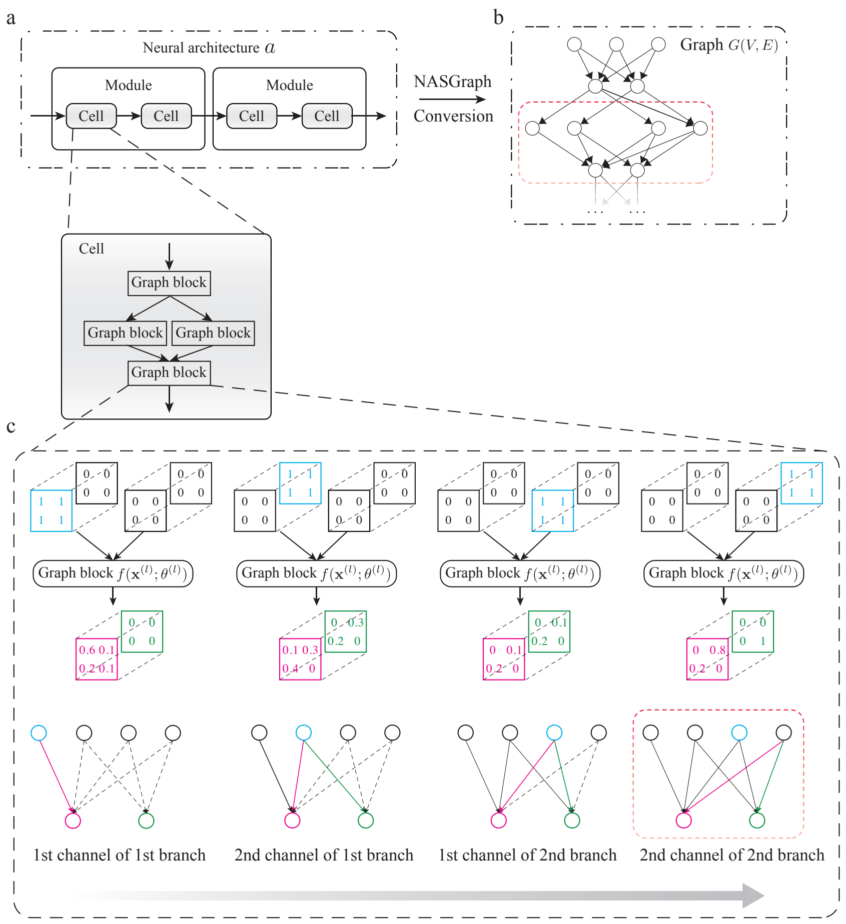
a
Neural architecture a
Module
Cell
Module
Cell
NASGraph
Conversion
b
Graph $G(V,E)$
Cell
Graph block
c
Graph block $f(\mathbf{x}^{(l)};\theta^{(l)})$
1st channel of 1st branch
2nd channel of 1st branch
1st channel of 2nd branch
2nd channel of 2nd branch

Figure 1: An overview of the *NASGraph* framework: the connectivity of graph nodes is determined by the forward propagation of the corresponding graph blocks. In the toy example shown in the bottom of the figure, if the output from the forward propagation is all-zeros matrix $\mathbb{O}$, there is no connection. Otherwise, the connection is built between a pair of graph nodes. The orange rectangles in (b) and (c) mark how a subgraph generated by a single forward propagation constitutes a part of the whole graph.

supernet method is faster than the conventional NAS methods because it enables weight sharing among all the candidate architectures in the search space.

Training-Free NAS. Training-Free NAS uses models with randomly initialized weights to obtain the saliency metrics that rank these models. Since there is no need for training models, this routine is considerably faster even compared to one-shot NAS. NASWOT Mellor et al. (2021) applies the theory on the linear regions in deep networks Hanin and Rolnick (2019) to achieve NAS without training. The saliency metrics for the pruning-at-initialization work in network pruning are also found to be effective in zero-cost NAS Abdelfattah et al. (2021). In addition, TENAS Chen et al. (2021) uses metrics from the condition number of neural tangent kernel and the number of linear regions. A comparison of saliency metrics Krishnakumar et al. (2022) finds that these metrics might contain complementary information and hence a combination of metrics can be helpful in NAS.

3 NASGraph: A Graph-Based Framework for Data-Agnostic and Training-Free NAS

Figure 1 shows an overview of our proposed *NASGraph* framework. A neural architecture is uniquely mapped to a DAG, i.e. $a \mapsto G(V,E)$. After the conversion, graph measures are computed to rank the performance of the corresponding neural architectures. We also note that our notion of graph refers to the specific graph representation of a neural architecture a via our proposed graph conversion techniques, instead of the original computation graph of a.

Graph block. The basic element in the *NASGraph* framework is the graph block. We use the notation $f^{(l)}(\mathbf{x}^{(l)};\theta^{(l)})$ to represent the l-th graph block, where $\mathbf{x}^{(l)}$ is the input to the graph block and $\theta^{(l)}$ is the model parameter. We combine several neural components (e.g. Conv and ReLU) to a graph block such that the output of the graph block $y^{(l)} = f^{(l)}(\mathbf{x}^{(l)};\theta^{(l)})$ is non-negative.

Conversion method. Inspired by the iterative re-evaluation method in the network pruning Verdenius et al. (2020); Tanaka et al. (2020), we apply conversion to each graph block independently. Similarly, we also use all-ones matrix as the input to the graph block $\mathbf{x}^{(l)} = \mathbb{1}^{C^{(l-1)}\times H^{(l-1)}\times W^{(l-1)}}$ in the forward propagation process, where $C^{(l-1)}$ is the number of channels, $H^{(l-1)}$ is the image height, and $W^{(l-1)}$ is the image width for the input to l-th graph block. This helps us get an unbiased estimate of the contribution of the input to the output. Further, to determine the contribution of the c-th channel of the input on the channels of the output for the l-th graph block, we apply a mask $\mathcal{M}_c^{(l)}$ to the input so that only the c-th channel $(\mathbf{x}^{(l)}_{d_1d_2d_3})_{d_1=c}$ is an all-ones matrix $\mathbb{1}^{H^{(l-1)}\times W^{(l-1)}}$ and other channels are zero matrices $\mathbb{0}^{H^{(l-1)}\times W^{(l-1)}}$. A toy example is shown in the bottom of Figure 1. We evaluate the contribution of the c-th channel $(\mathbf{x}^{(l)}_{d_1d_2d_3})_{d_1=c}$ to the output $\mathbf{y}^{(l)}$ by performing a forward propagation as described by:

$$\mathbf{y}_c^{(l)} = f^{(l)}(\mathcal{M}_c^{(l)} \odot \mathbf{x}^{(l)};\theta^{(l)}) \tag{1}$$

where $f^{(l)}(\cdot)$ is the l-th graph block, $\odot$ is the Hadamard product, and $\theta^{(l)}$ is the parameters of the l-th graph block. The score $\omega_{i^{(l-1)}j^{(l)}}$ for the edge e_{ij} between node $i^{(l-1)}$ and $j^{(l)}$ is determined by:

$$\omega_{i^{(l-1)}j^{(l)}} = \sum_{d_2=1}^{H^{(l)}}\sum_{d_3=1}^{W^{(l)}}((y_i^{(l)})_{d_1d_2d_3})_{d_1=j} \tag{2}$$

If $\omega_{i^{(l-1)}j^{(l)}}$ is larger than 0, we build an edge between node $i^{(l-1)}$ and node $j^{(l)}$ that indicates the relationship between i-th channel of the input $\mathbf{x}^{(l)}$ and j-th channel of the output $\mathbf{y}^{(l)}$. We use a virtual input graph block of identity operation to take the input to the neural architecture into consideration. After looping though all graph blocks, we can uniquely construct a graph.

Improving the search efficiency of NASGraph. To reduce computational overhead, NAS typically uses a training-reduced proxy to obtain the performance of neural architectures. A systematic study is reported in EcoNAS Zhou et al. (2020) where four reducing factors are analyzed: (1) number of epochs, (2) resolution of input images, (3) number of training samples, (4) number of channels for Convolution Neural Networks (CNNs). Following their convention, to accelerate *NASGraph*, we also consider the surrogate models, i.e. models with computationally reduced settings. We dub the surrogate model ***NASGraph*(h, c, m)**, where h is the number of channels, c is the number of search cells in a module, and m is the number of modules.

4 Performance Evaluation

4.1 Experiment Setup

We use AMD EPYC 7232P CPU in the computation of *NASGraph*. To compute the performance of baselines requiring GPUs, a single NVIDIA A40 GPU is used. We reduce the number of channels to be 16 and the number of cells to be 1, i.e. *NASGraph*(16, 1, 3) is used as the surrogate model.

4.2 Scoring neural architectures using average degree

The distribution of the average degree of graphs vs the performance of neural architectures is plotted in Figure 2 (a)-(l). Colorbar indicates the number of architectures in logarithmic base

10. We compute both Spearman's ranking correlation ρ and Kendall's Tau ranking correlation τ between them. In Table 1 we compare the performance of average degree against all the baselines on NAS-Bench-201 benchmark. Our method can rank architectures effectively and outperforms the baselines on most of the datasets.

We also evaluate our method on the NDS benchmark. The result is summarized in Table 4. Similar to NASWOT Mellor et al. (2021), we use 1000 randomly sampled architectures. Same subset of architectures is used for all methods.

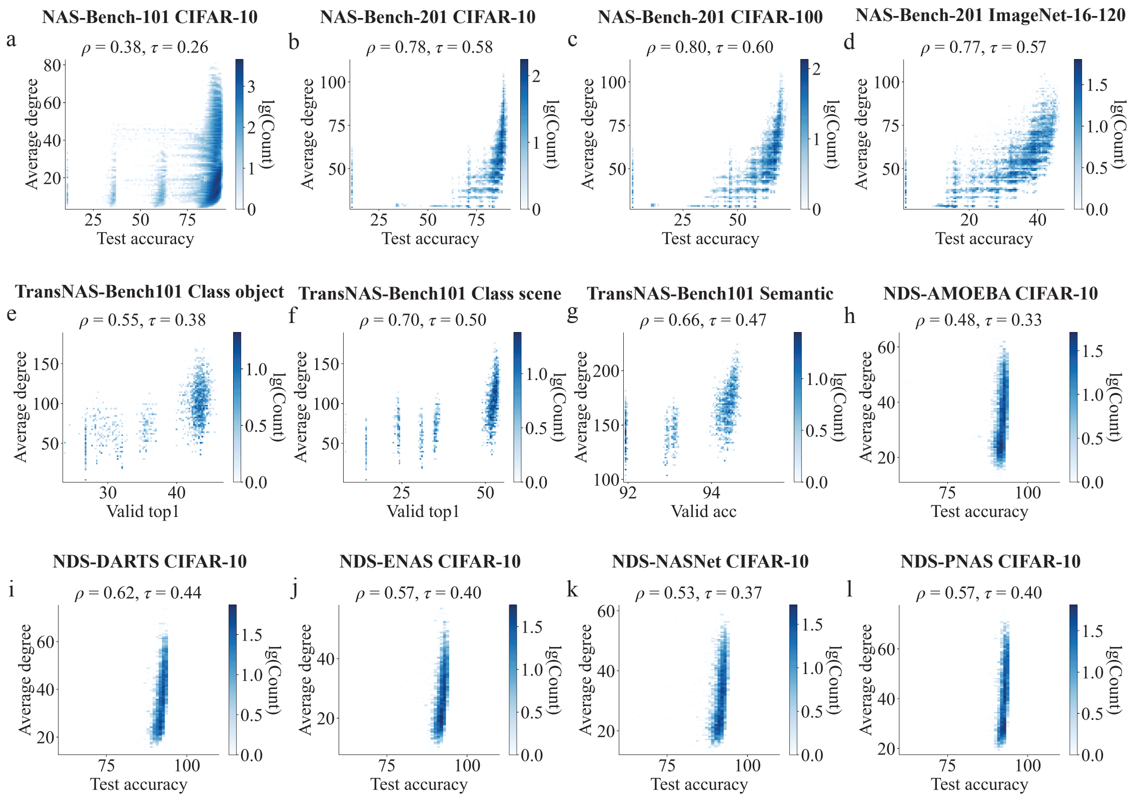

Figure 2: The ranking correlation between the performance of neural architecture a and the graph measures of the corresponding graph $G(V, E)$. In the *NASGraph* framework, each neural architecture is uniquely mapped to a graph, i.e. $a \mapsto G(V, E)$.

To explore the generality of the proposed *NASGraph* framework, we examine the performance of the graph measure on micro TransNAS-Bench-101. The comparison is shown in Table 5. As reported in Krishnakumar et al. (2022), there is a pronounced variation in the ranks of training-free proxies when changing NAS benchmarks. For the *class_object* downstream task, the average degree gives the best performance. For other downstream tasks, our method also exhibits a competitive performance.

4.3 Training-Free architecture search using NASGraph

We evaluate the performance of our metric when used as an alternative to validation accuracy during random search. $N = 100$ and $N = 200$ architectures are randomly sampled from NAS-Bench-201 and the training-free metrics are used to perform the search. We repeat the search process 100 times, and the mean and the standard deviation are reported in Table 6. Ground truth (GT) indicates the highest validation accuracy and the highest test accuracy for the validation and the test columns respectively. It is essential to highlight the fact that all the metrics in baselines are computed based on single A40 GPU (in GPU second) while *NASGraph* score is calculated on single CPU (in CPU second). *NASGraph* using the surrogate model NASGraph(16, 1, 3) can outperform the other training-free metrics on CIFAR-10 and CIFAR-100 datasets with a higher mean value and a lower standard deviation. At a small cost of performance, the surrogate model NASGraph(1, 1, 3) can have a significant improvement in the computation efficiency.

5 Broader Impact Statement

We propose a way to bridge the graph space and neural network space. Various benchmarks prove the efficacy of the proposed method. The establishment of the bridge enables one to shed light on neural architecture design through the graph theory. Besides, the computation of the graph measures is light-weight compared to the forward and backward process of neural architectures. In terms of efficiency, graph measures are good candidates for the NAS metric.

6 Limitation

In this work, we focus on simple graph measures such as average degree. The average degree is a global property for a graph. Dedicated graph measures such as graph measures related to communities in a graph enable a deeper understanding of neural architectures.

7 Discussion

In this paper, we proposed *NASGraph*, a novel graph-based method for NAS featuring lightweight (CPU-only) computation and is data-agnostic and training-free. Extensive experimental results verified the high correlation between the graph measure of *NASGraph* and performance of neural architectures in several benchmarks. Compared to existing NAS methods, *NASGraph* provides a more stable and accurate prediction of the architecture performance and can be used for efficient architecture search. We also show that our graph measures can be combined with existing data-dependent metrics to further improve NAS. We believe our findings provide a new perspective and useful tools for studying NAS through the lens of graph theory and analysis.

Our method addresses some limitations in current NAS methods (e.g. data dependency, GPU requirement, and operation preference) and attains new state-of-the-art NAS performance. The authors do not find any immediate ethical concerns or negative societal impacts from this study.

References

Abdelfattah, M. S., Mehrotra, A., Dudziak, Ł., and Lane, N. D. (2021). Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*.

Chen, W., Gong, X., and Wang, Z. (2021). Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *arXiv preprint arXiv:2102.11535*.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*.

Dong, X. and Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. (2018). Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*.

Duan, Y., Chen, X., Xu, H., Chen, Z., Liang, X., Zhang, T., and Li, Z. (2021). Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5251–5260.

Hanin, B. and Rolnick, D. (2019). Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, pages 2596–2604. PMLR.

Krishnakumar, A., White, C., Zela, A., Tu, R., Safari, M., and Hutter, F. (2022). Nas-bench-suite-zero: Accelerating research on zero cost proxies. *arXiv preprint arXiv:2210.03230*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. Technical report, Univeristy of Toronto.

Lee, N., Ajanthan, T., and Torr, P. H. (2018). Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*.

Li, G., Yang, Y., Bhardwaj, K., and Marculescu, R. (2023). Zico: Zero-shot nas via inverse coefficient of variation on gradients. *arXiv preprint arXiv:2301.11300*.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. (2018a). Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34.

Liu, H., Simonyan, K., and Yang, Y. (2018b). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Mellor, J., Turner, J., Storkey, A., and Crowley, E. J. (2021). Neural architecture search without training. In *International Conference on Machine Learning*, pages 7588–7598. PMLR.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR.

Radosavovic, I., Johnson, J., Xie, S., Lo, W.-Y., and Dollár, P. (2019). On network design spaces for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1882–1890.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems*, 33:6377–6389.

Theis, L., Korshunova, I., Tejani, A., and Huszár, F. (2018). Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*.

Verdenius, S., Stol, M., and Forré, P. (2020). Pruning via iterative ranking of sensitivity statistics. *arXiv preprint arXiv:2006.00896*.

Wang, C., Zhang, G., and Grosse, R. (2020). Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR.

Zamir, A. R., Sax, A., Shen, W., Guibas, L. J., Malik, J., and Savarese, S. (2018). Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722.

Zhou, D., Zhou, X., Zhang, W., Loy, C. C., Yi, S., Zhang, X., and Ouyang, W. (2020). Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11396–11404.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Submission Checklist

All submissions must include a section containing the AutoML submission checklist, shown below. The submission checklist does not count towards the page limit and must be included at submission and camera-ready time. The submission checklist draws upon related submission checklists: the NeurIPS '22 checklist and the NAS checklist.

For each question, change the default `\answerTODO{}` (typeset [TODO]) to `\answerYes{[justification]}` (typeset [Yes]), `\answerNo{[justification]}` (typeset [No]), or `\answerNA{[justification]}` (typeset [N/A]). **You must include a brief justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license of the code and datasets? [Yes] See Section 1.
- Did you include all the code for running experiments? [No] We include the code we wrote for conducting the experiments, but complete replication depends on proprietary libraries for executing on a private compute cluster. The code therefore is not runnable without modification. To compensate, we provide a runnable but non-parallelized version of the code that could replicate the results at the expense of a greater wall-clock time.
- Did you include the license of the datasets? [N/A] Our experiments were conducted on publicly available datasets and we have not introduced new datasets.

Please note that if you answer a question with `\answerNo{}`, we expect that you provide an explanation and/or compensation for the omission. For example, if you cannot provide complete evaluation code for some reason, you might instead provide code for a minimal reproduction of the main insights of your paper.

Please do not modify the questions and only use the provided macros for your answers. Note that the submission checklist does not count towards the page limit. If you choose to modify `instructions.tex`, please delete these instructions and only keep the Submission Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The abstract and introduction accurately reflect the contribution and scope.
   (b) Did you describe the limitations of your work? [Yes] It is in the section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our work does not have negative societal impacts.
   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` [Yes] We read the guidelines and confirm that our paper conforms to them.
2. If you ran experiments...
   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [Yes] We use the same evaluation protocol for all methods.
   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [Yes] We include the experiment setup in subsection 4.1 and search spaces in B.

(c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] We repeat experiment for 100 times for the search efficiency comparison and 8 times for the correlation computation.

(d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [Yes] We report the standard deviation.

(e) Did you report the statistical significance of your results? [No] We report the mean and standard deviation. There is no statistical hypothesis testing, so we do not report statistical significance.

(f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [No] We use the standard benchmarks.

(g) Did you compare performance over time and describe how you selected the maximum duration? [Yes] We compare the running efficiency and report in the table 6.

(h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We report the computational resources.

(i) Did you run ablation studies to assess the impact of different components of your approach? [N/A] Our method focuses on the NAS metric. We test different graph measures.

3. With respect to the code used to obtain your results…

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] We include our code in the section A and the instruction on how to run the code in the code repository.

(b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] The example is included in the instruction.

(c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [Yes] Code repository is well documented.

(d) Did you include the raw results of running your experiments with the given code, data, and instructions? [No] We do not include the raw results. There is an instruction on how to run our code.

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [Yes] There is an instruction on how to reproduce our results shown in tables.

4. If you used existing assets (e.g., code, data, models)…

(a) Did you cite the creators of used assets? [Yes] We cite all benchmarks we use.

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A] We only use publicly available benchmarks.

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The benchmark we use does not contain personally identifiable information or offensive content.

5. If you created/released new assets (e.g., code, data, models)…

(a) Did you mention the license of the new assets (e.g., as part of your code submission)? [Yes] We do mention the license.

(b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [Yes] We include the code in GitHub URL.

6. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] Our work does not involve crowdsourcing or research with human subjects.

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Our work does not involve crowdsourcing or research with human subjects.

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] Our work does not involve crowdsourcing or research with human subjects.

7. If you included theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [Yes] We include full set of assumptions for graph measures.

   (b) Did you include complete proofs of all theoretical results? [N/A] We do not have theory proof.

A Code Availability

Our code is publicly available at `https://github.com/Zhenhan-Huang/NASGraph-Public`.

B NAS Benchmarks and Baseline Models

NAS benchmarks. To examine the effectiveness of our proposed *NASGraph*, we use neural architectures on NAS-Bench-101 Ying et al. (2019), NAS-Bench-201 Dong and Yang (2020), TransNAS-Bench-101 Duan et al. (2021) and Network Design Space (NDS) Radosavovic et al. (2019). NAS-Bench-101 is the first NAS benchmark consisting of 423,624 neural architectures trained on the CIFAR-10 dataset Krizhevsky et al. (2009). The training statistics are reported at 108th epoch. NAS-Bench-201 is built for prototyping NAS algorithms. It contains 15,625 neural architectures trained on CIFAR-10, CIFAR-100 Krizhevsky et al. (2009) and ImageNet-16-120 Chrabaszcz et al. (2017) datasets. The training statistics are reported at 200th epoch for these three datasets. In addition to standard NAS benchmarks, we also examine the performance of *NASGraph* on TransNAS-Bench-101, specifically the micro search space. The micro (cell-level) TransNAS-Bench-101 has 4,096 architectures trained on different tasks using the Taskonomy dataset Zamir et al. (2018). The NDS benchmark includes AmoebaNet Real et al. (2019), DARTS Liu et al. (2018b), ENAS Pham et al. (2018), NASNet Zoph and Le (2016) and PNAS search spaces Liu et al. (2018a).

Baselines. We use metrics in training-free NAS as our baselines. `zico` Li et al. (2023) is based on the theory of Gram Matrix Du et al. (2018). `relu_logdet` (also dubbed `naswot`) Mellor et al. (2021) applies the theory on the number of linear regions to represent the model expressivity. `jacob_cov` Mellor et al. (2021) is based on the correlation of Jacobians with inputs. `grad_norm` Abdelfattah et al. (2021) sums the Euclidean norm of the gradients. `snip` Lee et al. (2018) is related to the connection sensitivity of neural network model. `grasp` Wang et al. (2020) is based on the assumption that gradient flow is preserved in the efficient training. `fisher` Theis et al. (2018) estimates fisher information of model parameters, `synflow` Tanaka et al. (2020) preserves the total flow of synaptic strength.

C Performance Comparison on Various NAS Benchmarks

Table 1 shows the performance comparison on the NAS-Bench-201 benchmark. The correlation is computed on all architectures in the search space. Our method overall achieves the best performance.

Table 1: Comparison of the ranking correlation between *NASGraph* and training-free NAS methods using single metric on NAS-Bench-201. Correlations are calculated between the metrics and test accuracies.

Method	Metric	CIFAR-10		CIFAR-100		ImageNet-16-120	
		ρ	τ	ρ	τ	ρ	τ
NASWOT	`naswot`	0.76	0.57	0.79	**0.61**	0.71	0.55
ZiCo[‡]	`zico`	0.74	0.55	0.78	0.58	0.76	0.56
TENAS	NTK	-	-	-	-0.42	-	-
	NLR	-	-	-	-0.50	-	-
Zero-Cost NAS	`grad_norm`	0.58	0.42	0.64	0.47	0.58	0.43
	`snip`	0.58	0.43	0.63	0.47	0.58	0.43
	`grasp`	0.48	0.33	0.54	0.38	0.56	0.40
	`fisher`	0.36	0.26	0.39	0.28	0.33	0.25
	`synflow`	0.74	0.54	0.76	0.57	0.75	0.56
	`jacob_cov`	0.73	0.55	0.71	0.55	0.71	0.54
Ours	`avg_deg`	**0.78**	**0.58**	**0.80**	0.60	**0.77**	**0.57**

‡ Original implementation of ZiCo uses cutout data augmentation. To make a fair comparison, we recalculate the correlation without cutout data augmentation.

Table 2: Comparison of the ranking correlations using multiple metrics with training-free NAS methods on NAS-Bench-201. Correlations between the combined metric and NAS Benchmark performance are reported. TENAS Chen et al. (2021) combines rankings by NTK and NLR. Zero-Cost NAS Abdelfattah et al. (2021) takes a majority vote among the three metrics: `synflow`, `jacob_cov` and `snip`. Our method combines the rankings of `avg_deg` and `jacob_cov`.

Method		CIFAR-10		CIFAR-100		ImageNet-16-120	
		ρ	τ	ρ	τ	ρ	τ
TENAS	Rank combine	-	-	-	0.64	-	-
Zero-Cost NAS	Voting	0.82	-	0.83	-	0.82	-
Ours	Rank combine	**0.85**	0.66	**0.85**	**0.67**	**0.82**	0.64

Table 2 shows the performance of combining multiple NAS metrics. The decision is made by either voting or a combination of ranks. We find that a combination of graph measure `avg_deg` with `jacob_cov` is able to achieve the best performance.

Table 3 compares the bias existing in NAS metrics. Bias exhibits the preference of NAS metrics on various operations in the NAS search space. Our method overall has the lowest bias compared to baseline methods.

Table 3: Comparison of the accumulated frequency difference between training-free NAS methods and GT on top 10% architectures of the NAS-Bench-201. GT ranks architectures ranked by test accuracies. Lower value means less bias (i.e. closer to GT).

Metric	CIFAR-10	CIFAR-100	ImageNet-16-120	Average bias
`relu_logdet`	0.3	0.27	0.19	0.25
`grad_norm`	0.32	0.3	0.24	0.29
`snip`	0.31	0.29	0.24	0.28
`grasp`	0.31	0.28	0.24	0.28
`fisher`	0.52	0.52	0.53	0.52
`synflow`	0.22	0.18	0.27	0.22
`jacob_cov`	0.39	0.42	0.22	0.34
		Our method		
`avg_deg`	0.22	**0.14**	0.27	0.21
`comb_rank`	**0.17**	0.17	**0.12**	**0.15**

Table 4 shows the performance on the NDS benchmark that consists of multiple search spaces such as DARTS search space.

Table 4: Comparison of Kendall's Tau ranking correlations τ between test accuracies and training-free NAS metrics on the NDS benchmark.

Metric	AMOEBA	DARTS	ENAS	NASNet	PNAS
`grad_norm`	-0.12	0.22	0.06	-0.05	0.15
`snip`	-0.09	0.26	0.10	-0.02	0.18
`grasp`	0.02	-0.04	0.03	0.18	-0.01
`fisher`	-0.12	0.19	0.04	-0.07	0.14
`jacov_cov`	0.22	0.19	0.11	0.05	0.10
`synflow`	-0.06	0.30	0.14	0.04	0.21
`naswot`	0.22	**0.47**	0.37	0.30	0.38
`avg_deg` (Ours)	**0.32**	0.45	**0.41**	**0.37**	**0.40**

Table 5 shows the performance on the micro TransNAS-Bench-101 benchmark. Our proposed method is comparable with the baseline methods. We notice that NAS methods hardly have a consistent performance. For example, although `jacob_cov` has an inferior performance on the

NAS-Bench-201 benchmark, it has a relatively good performance on the TransNAS-Bench-101 benchmark.

Table 5: Comparison of Spearman's ranking correlations ρ between validation accuracies and training-free NAS metrics on micro TransNAS-Bench-101. The baseline performance is extracted from Krishnakumar et al. (2022). Note that `synflow` and `avg_deg` are data-agnostic. CO is `class_object`, CS is `class_scene`, RL is `room_layout`, SS is `segment_semantic`.

Metric	Micro TransNAS-Bench-101			
	CO	CS	RL	SS
`plain`	0.34	0.24	0.36	-0.02
`grasp`	-0.22	-0.27	-0.29	0.00
`fisher`	0.44	0.66	0.30	0.12
`epe_nas`	0.39	0.51	**0.40**	0.00
`grad_norm`	0.39	0.65	0.25	0.60
`snip`	0.45	0.70	0.32	0.68
`synflow`	0.48	0.72	0.30	0.00
`l2_norm`	0.32	0.53	0.18	0.48
`params`	0.45	0.64	0.30	0.68
`zen`	0.54	0.72	0.38	0.67
`jacob_cov`	0.51	**0.75**	**0.40**	**0.80**
`flops`	0.46	0.65	0.30	0.69
`naswot`	0.39	0.60	0.25	0.53
`zico`	0.55	0.68	0.26	0.61
`avg_deg` (Ours)	**0.55**	0.70	0.37	0.66

Table 6 shows the efficiency comparison. Although our method can achieve a fast running time using only CPU. By using the proxy model, our method is the fastest.

Table 6: Comparison of training-free NAS metrics using random search method. The same subset of architectures is randomly chosen from NAS-Bench-201 for all metrics. GT reports the performance of the best architecture in that subset.

Metric	Running time	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
N = 100							
`relu_logdet`	52.72 GPU sec.	89.51 ± 0.96	89.22 ± 1.03	69.48 ± 1.44	69.58 ± 1.50	42.92 ± 2.41	43.27 ± 2.62
`grad_norm`	364.68 GPU sec.	88.28 ± 1.42	87.94 ± 1.48	65.96 ± 3.11	66.13 ± 3.10	34.97 ± 6.82	34.96 ± 7.06
`snip`	363.71 GPU sec.	88.29 ± 1.42	87.95 ± 1.48	66.14 ± 2.96	66.32 ± 2.97	35.44 ± 6.49	35.44 ± 6.72
`grasp`	377.29 GPU sec.	88.06 ± 1.55	87.74 ± 1.58	66.27 ± 3.50	66.38 ± 3.55	35.20 ± 6.76	35.19 ± 6.93
`fisher`	315.57 GPU sec.	84.08 ± 6.68	83.70 ± 6.67	61.77 ± 7.26	61.89 ± 7.44	30.80 ± 8.02	30.49 ± 8.33
`synflow`	360.15 GPU sec.	89.91 ± 0.87	89.67 ± 0.88	70.03 ± 1.79	70.17 ± 1.79	41.89 ± 4.13	42.23 ± 4.24
`jacob_cov`	360.48 GPU sec.	88.68 ± 1.56	88.32 ± 1.59	67.45 ± 2.91	67.57 ± 3.03	40.64 ± 3.54	40.76 ± 3.77
`avg_deg` (NASGraph(1, 1, 3))	7.78 CPU sec.	89.74 ± 0.77	89.53 ± 0.75	69.90 ± 1.38	70.01 ± 1.43	42.00 ± 2.80	40.73 ± 4.14
`avg_deg` (NASGraph(16, 1, 3))	106.18 CPU sec.	89.95 ± 0.49	89.73 ± 0.52	70.17 ± 1.06	70.29 ± 1.10	42.72 ± 2.33	43.15 ± 2.29
GT	-	90.98 ± 0.36	90.77 ± 0.31	71.48 ± 0.86	71.69 ± 0.81	45.45 ± 0.67	45.74 ± 0.65
N = 200							
`relu_logdet`	90.39 GPU sec.	89.64 ± 0.81	89.33 ± 0.84	69.65 ± 1.36	69.87 ± 1.33	43.25 ± 2.22	43.62 ± 2.37
`grad_norm`	644.23 GPU sec.	88.23 ± 1.51	87.87 ± 1.53	65.46 ± 3.34	65.67 ± 3.42	35.08 ± 7.05	35.00 ± 7.26
`snip`	712.58 GPU sec.	88.23 ± 1.51	87.87 ± 1.54	65.68 ± 3.16	65.89 ± 3.21	35.08 ± 7.05	35.00 ± 7.26
`grasp`	692.74 GPU sec.	88.31 ± 1.35	87.96 ± 1.37	65.97 ± 3.21	66.16 ± 3.28	34.83 ± 6.63	34.74 ± 6.81
`fisher`	622.92 GPU sec.	85.55 ± 4.91	85.24 ± 4.92	61.69 ± 5.62	61.86 ± 5.77	29.39 ± 6.38	29.04 ± 6.65
`synflow`	742.74 GPU sec.	89.87 ± 0.85	89.61 ± 0.85	69.93 ± 1.84	70.05 ± 1.89	41.54 ± 3.76	41.93 ± 3.77
`jacob_cov`	688.77 GPU sec.	88.34 ± 1.67	88.00 ± 1.71	67.39 ± 2.93	67.55 ± 3.05	40.95 ± 3.24	41.04 ± 3.41
`avg_deg` (NASGraph(1, 1, 3))	15.98 CPU sec.	89.92 ± 0.61	89.69 ± 0.62	70.25 ± 1.20	70.42 ± 1.21	41.96 ± 2.44	42.48 ± 2.39
`avg_deg` (NASGraph(16, 1, 3))	217.21 CPU sec.	89.96 ± 0.38	89.73 ± 0.43	70.22 ± 0.99	70.45 ± 0.98	42.27 ± 2.36	42.76 ± 2.36
GT	-	91.14 ± 0.25	90.91 ± 0.24	71.84 ± 0.76	72.04 ± 0.72	45.72 ± 0.54	46.01 ± 0.50