DIFFERENTIAL EVOLUTION FOR NEURAL ARCHITECTURE SEARCH

Noor Awad[1]*, **Neeratyoy Mallik**[1]*, & **Frank Hutter**[1,2]
[1]Department of Computer Science, University of Freiburg
[2]Bosch Center for Artificial Intelligence
Freiburg, Germany
{awad,mallik,fh}@cs.uni-freiburg.de

ABSTRACT

Neural architecture search (NAS) methods rely on a *search strategy* for deciding which architectures to evaluate next and a *performance estimation strategy* for assessing their performance (e.g., using full evaluations, multi-fidelity evaluations, or the one-shot model). In this paper, we focus on the search strategy and demonstrate that the simple yet powerful evolutionary algorithm of *differential evolution* (DE) yields state-of-the-art performance for NAS, comparing favourably to regularized evolution and Bayesian optimization. It yields improved and more robust results for 13 tabular NAS benchmarks based on NAS-Bench-101, NAS-Bench-1Shot1, NAS-Bench-201 and NAS-HPO bench.

1 INTRODUCTION

Evolutionary algorithms have a long history for neural architecture search (NAS), for combined search of architectures and weights (Stanley et al., 2019; 2009), (Mason et al., 2017a; Baioletti et al., 2020), search of just the architecture (Real et al., 2017; Elsken et al., 2018; Liu et al., 2017), and multi-objective optimization of performance and resource consumption (Elsken et al., 2019b). Recently, regularized evolution (Real et al., 2018) has been shown to yield very robust performance on NAS benchmarks (Ying et al., 2019) and found novel neural architectures for object recognition in CIFAR-10 (Real et al., 2018) and to an improved version of the transformer (So et al., 2019).

Here, we study the use of the popular evolutionary algorithm of *differential evolution* (DE, Storn & Kenneth (1997)) for NAS. DE has previously been used to search basic neural network architectures. For example, Mineu et al. (2010) used DE to search for layers, neurons, weights and connections for architectures using a special local and global neighbourhood strategy for the mutation operation, Bhuiyan (2009) introduced a simple DE algorithm without the use of a crossover operation, and Zhang et al. (2019) used DE to jointly evolve architectures and weights, followed by the Levenberg-Marquardt algorithm to finetune the generated weights. Other works that use DE to evolve basic neural architectures can be found in (Dhahri et al., 2012; Mason et al., 2017b). In this paper and different from the above, rather than developing a customized DE version for a specific task, we standardize and benchmark the use of a simple, yet effective DE, for a wide range of NAS benchmarks .

DE has been used as one of many algorithms for a recent benchmark of joint hyperparameter optimization and NAS Klein et al. (2018), and did not yield state-of-the-art performance there. However, that study used a simple SciPy Virtanen et al. (2020) implementation, and we demonstrate that with a better implementation and a fixed, robust hyperparameter setting, DE does indeed achieve state-of-the-art performance on a wide range of recent NAS benchmarks compared to other blackbox optimizers.

Most recent progress in NAS focuses on exploiting the one-shot model introduced by Pham et al. (2018), prominently based on extensions of differentiable architecture search (DARTS (Liu et al., 2018)). However, the one-shot model in general (Sciuto et al., 2019) and DARTS in particular (Zela et al., 2020b) feature several failure modes. For this reason, using the terminology of Elsken et al.

*Equal contribution

(2019a), we do not employ the one-shot model as a performance estimation strategy to evaluate different search strategies, but rather stick to the simpler performance estimation strategy of full evaluations. While a large-scale evaluation would normally be completely infeasible in this setting due to the high computational cost of full evaluations, this analysis is made possible by the recent availability of tabular NAS benchmarks (Ying et al., 2019).

We first describe a canonical version of differential evolution (DE; Section 2), then describe how to apply DE to NAS (Section 3), and then Section 4 demonstrates that the resulting algorithm outperforms the previous best search strategies on a wide range of 13 benchmarks based on NAS-Bench-101 (Ying et al., 2019) NAS-Bench-1Shot1 (Zela et al., 2020c), NAS-Bench-201 (Dong & Yang, 2020), and NAS-HPO-Bench (Klein & Hutter, 2019).

2 Canonical Differential Evolution

Differential Evolution (DE, Storn & Kenneth (1997)) is an evolutionary algorithm that is based on four steps (initialization, mutation, crossover and selection). We describe these below, deferring details to Appendix A. In its canonical form, DE is described for continuous optimization.

Initialization. DE is a population-based meta-heuristic algorithm which consists of a population of NP individuals. Each individual is considered a solution and expressed as a vector of D-dimensional decision variables, which are initialized uniformly at random in the search range.

Mutation. A new child/offspring is produced using the mutation operation for each individual in the population by a so called mutation strategy. The classical DE uses $rand/1$ mutation, in which three random individuals/parents $X_{r_1}, X_{r_2}, X_{r_3}$ are chosen to generate a new vector $V_{i,g}$ as follows:

$$V_{i,g} = X_{r_1,g} + F \cdot (X_{r_2,g} - X_{r_3,g}) \tag{1}$$

where $V_{i,g}$ is the mutant vector generated for each individual $X_{i,g}$ in the population, F is the scaling factor (which usually takes values within the range $[0, 1]$), and r_1, r_2, r_3 are the indices of different randomly selected individuals. The subscript g indicates the generation index, or iteration number.

Crossover. After the mutation, a crossover operation is applied to each target vector $X_{i,g}$ and its corresponding mutant vector $V_{i,g}$ to generate a trial vector $U_{i,g}$. We use a simple binomial crossover, which chooses the value for each dimension i from $V_{i,g}$ with probability Cr and from $X_{i,g}$ otherwise.

Selection. After generating the trial vector $U_{i,g}$, DE computes its function value $f(U_{i,g})$, keeping $U_{i,g}$ if it performs at least as well as $X_{i,g}$ and reverting back to $X_{i,g}$ otherwise.

3 Differential Evolution for NAS

Recent NAS approaches and benchmarks parameterize cell structures of deep neural networks as directed graphs (Zoph et al., 2018; Ying et al., 2019; Zela et al., 2020a; Dong & Yang, 2020). The realisation of a candidate cell structure can be seen as an assignment of operations from a set of choices or a range of values, such as the choice of operator on an edge or the choice of predecessors of a node in the directed graph.

We found the best way of applying DE when parameters are discrete or categorical is to keep the population in a continuous space, perform canonical DE as usual as described in Section 2, and only discretize copies of individuals to evaluate them. If we instead dealt with a discrete population space, then the diversity of population would drop dramatically, leading to many individuals having the same parameter values; the resulting population would then have many duplicates, lowering the diversity of the difference distribution and making it hard for DE to explore effectively.

The modified canonical DE we used for NAS is presented in Algorithm 1 of Appendix B. Figure 1 shows the general framework of our DE implementation. We scale all NAS parameters to [0, 1] to let DE work on individuals from a uniform, continuous space. The continuous value for $U_{i,g}$ needs to be mapped back to the original space of the NAS parameters before the function evaluation. In Algorithm 1, we use a method *discretized_architecture* to do this; this method retrieves the following values X^i depending on the parameter's type:

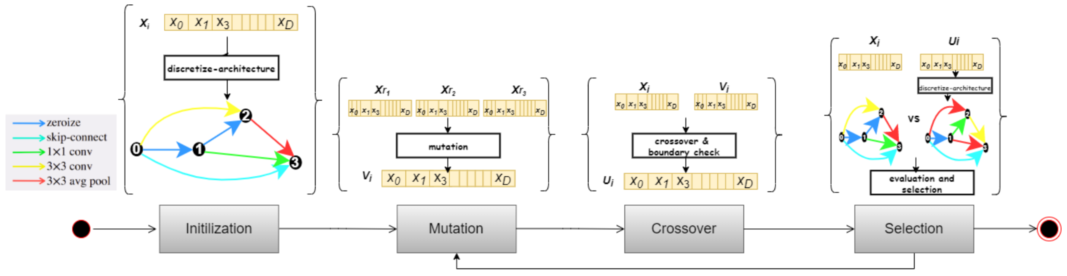


Figure 1: DE for NAS Framework

- *Integer* and *float* parameters: $X^i \in [a_i, b_i]$ are retrieved as: $a_i + (b_i - a_i) \cdot U_{i,g}$, where the integer parameters are additionally rounded.
- *Ordinal* and *categorical* parameters $X^i \in \{x_1, ..., x_n\}$: the range [0, 1] is divided uniformly into n bins.

We illustrate the discretization in Figure 2. For the categorical parameter $X^2 \in \{$'*1x1 conv*', '*skip*', '*3x3 conv*'$\}$, the corresponding continuous DE space maps to $[0, 1/3)$ for '*1x1 conv*', $[1/3, 2/3)$ for '*skip*', and $[2/3, 1]$ for '*3x3 conv*'. As seen in Figure 2, the *difference vector* and the randomly sampled candidate individuals determine how the search space is spanned to find a *mutant vector* that participates in the selection process. The resultant mutant can lie on any of the 9 grids formed in Figure 2 for the 2-dimensional case.

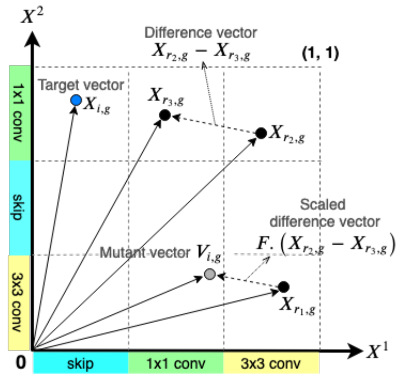
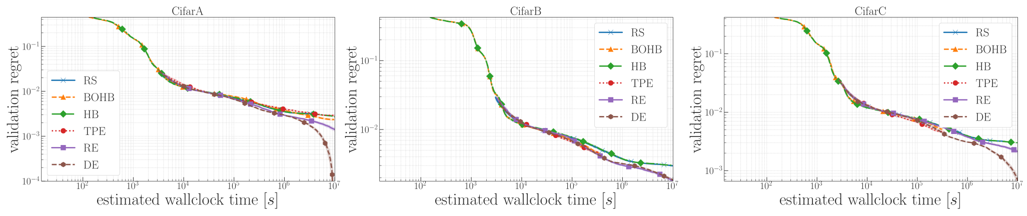


Figure 2: Illustration of DE mutation on categorical parameterization of NAS cell space

One drawback for such an approach might arise in the case of a conditional parameter space. However, just like in NAS-Bench-101 (Ying et al., 2019), the function value for an *invalid architecture* can simply be a maximal error of 1 (at no computational cost, even in a non-tabular benchmark). Such individuals will be guaranteed to lose in the selection process, thereby implicitly avoiding invalid architectures over time.

4 Experiments

We evaluate DE's performance on four recent NAS benchmarks: NAS-Bench-101 (Ying et al., 2019), NAS-HPO (Klein & Hutter, 2019), NAS-Bench-1shot1 (Zela et al., 2020a) and NAS-Bench-201 (Dong & Yang, 2020). We compare against several baseline algorithms, namely Random Search (RS) (Bergstra & Bengio, 2012), BOHB (Falkner et al., 2018), Tree Parzen Estimator (TPE) (Bergstra et al., 2011), Hyperband (HB) (Li et al., 2018) and regularized evolution (RE) (Real et al., 2018). Appendix C has more details about the used algorithms and their hyperparameter settings. For DE, we set scaling factor F and crossover rate Cr to 0.5 over all the generations. For the population size NP, we tested several values (provided in Appendix in F) and chose 20 for our experiments. We consider RE as the *primary* baseline algorithm (run until $10Ms$) since it belongs to the same family of algorithms as DE and has been shown to perform robustly many times before. We provide a comparison of the robustness between RE and DE in Appendix E. For each algorithm, we performed 500 independent runs and report the mean performance of the immediate validation regret (Ying et al., 2019). Throughout, we evaluate algorithms in the anytime setting, showing performance of the best found configuration over time as suggested by Ying et al. (2019) and Lindauer & Hutter (2019). In all our plots, the x-axis shows *estimated wall-clock time*, as the cumulative time taken for training each of the architectures found as returned by the NAS benchmarks. Due to the space limitation, we show the test regret plots for NAS-101 and NAS-1shot1,



(a) CifarA (b) CifarB (c) CifarC

Figure 3: A comparison of the mean validation regret performance of 500 independent runs as a function of estimated training time for NAS-Bench-101 on CifarA, CifarB and CifarC.

and also discuss the experiments for NAS-Bench-201 and NAS-HPO in Appendix D.1, D.2, D.3, D.4, respectively. We compare our implementation with the popular SciPy-DE code Virtanen et al. (2020) in Appendix G. Our code for DE and for reproducing our experiments is publicly available at `https://github.com/automl/DE-NAS`.

4.1 NAS-Bench-101

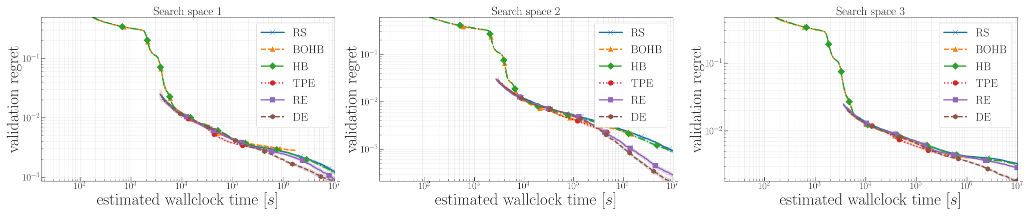
In this experiment we investigated DE's performance on the cell search space of 423k unique cell architectures of a convolutional neural network for CIFAR-10 defined by NAS-Bench-101 (Ying et al., 2019). We study three different search spaces: *CifarA* contains the main search space discussed by Ying et al. (2019), and *CifarB* and *CifarC* are variants of the same space with alternative encodings (treating the edge parameters as categorical parameters with 21 choices and continuous $\in [0, 1]$, respectively). Figure 3 presents a comparison of the performance of compared algorithms showing the mean validation regret of 500 independent runs as a function of the estimated training time. We show our results for test regret in Appendix D.1. HB and BOHB are multi-fidelity optimization algorithms which evaluate at fewer epochs while other algorithms evaluate only at E_{max}. However, NAS-Bench-101 features a low rank correlation between the performance obtained with different budgets (Ying et al., 2019), and thus these algorithms do not perform better than the other algorithms that only use the maximum number of epoch. The other algorithms (RS, TPE, and RE) follow the same behaviour at the beginning of the search for all 3 encodings of the search space, and in the end the evolutionary algorithms RE and DE clearly yield the best performance. DE shows much better final performance for *CifarA* and *CifarC* and competitive performance with RE for *CifarB*. It appears that DE is able to exploit high-dimensional spaces well and handle mixed-types better. This may be attributed to NAS-Bench-101's locality property (Ying et al., 2019) along with DE's search method, since a DE population with individuals from a good region will be able to exploit further and get near the global optimum.

4.2 NAS-Bench-1Shot1

NAS-Bench-1Shot1 (Zela et al., 2020c) was created from the search space of NAS-Bench-101 by keeping the network-level topology intact and modifying the cell-level topology to allow the application of modern weight sharing algorithms for three search spaces with $6,240$ (search space 1), $29,160$ (search space 2), and $363,648$ (search space 3) architectures. Figure 4 shows our results for the mean performance on validation regret while we present a comparison on test regret in Appendix D.2. For search space 1, all the algorithms achieve nearly the same error at the beginning of the search, then DE converges faster until other algorithms catch up. For search space 2, RE and DE converge fastest. For search space 3, the most complex (high-dimensional) and largest (10x more architectures than space 2, and 100x more than space 1), DE clearly outperforms all other algorithms and converges fastest.

5 Conclusion

We demonstrated that Differential Evolution can be utilised as an alternative search strategy for the growing field of NAS. We also demonstrated DE's ability to handle mixed data types and high-dimensional spaces. DE may thus be a good candidate for NAS in very large spaces that may help



(a) Search Space 1 (b) Search Space 2 (c) Search Space 3

Figure 4: A comparison of the mean validation regret performance of 500 independent runs as a function of estimated training time for NAS-1Shot1 on the three different search spaces.

discover new, yet unknown, architectural design patterns. Since DE naturally lends itself well to parallelization, future work includes providing a parallel implementation. We are also interested in combining DE with different performance estimation strategies, such as multi-fidelity methods and the one-shot model.

ACKNOWLEDGMENTS

The authors acknowledge funding by the Robert Bosch GmbH for financial support.

References

M. Baioletti, G. Di Bari, A. Milani, and Valentina Poggioni. Differential evolution for neural networks optimization. *Mathematics*, 8(1), 2020.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *JMLR*, 13:281–305, 2012.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proc. of NeurIPS'11*, pp. 2546–2554, 2011.

Md. Zakirul Alam Bhuiyan. An algorithm for determining neural network architecture using differential evolution. In *International Conference on Business Intelligence and Financial Engineering*, July 2009.

U. Chakraborty. *Advances in Differential Evolution*. Springer, 2008.

H. Dhahri, A. Alimi, and A. Abraham. Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network. *Neurocomputing*, 97:131–140, November 2012.

X. Dong and Y. Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv:2001.00326 [cs.CV]*, 2020.

T. Elsken, J. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv:1804.09081 [stat.ML]*, 2018.

T. Elsken, J. Metzen, and F. Hutter. Neural architecture search: A survey. *JMLR*, 20:1–21, 2019a.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019b. URL `https://openreview.net/forum?id=ByME42AqK7`.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proc. of ICML'18*, pp. 1437–1446, 2018.

P. Groenen and M. van Velden. Multidimensional scaling by majorization: A review. *Journal of Statistical Software, Articles*, 73(8):1–26, 2016.

A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv:1905.04970 [cs.LG]*, 2019.

A. Klein, E. Christiansen K. Murphy, and F. Hutter. Towards reproducible neural architecture and hyperparameter search. In *ICML Reproducibility in Machine Learning Workshop*, 2018.

J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, Mar 1964.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18(185):1–52, 2018.

Marius Lindauer and Frank Hutter. Best Practices for Scientific Research on Neural Architecture Search. *arXiv e-prints*, art. arXiv:1909.02453, Sep 2019.

H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv:1711.00436 [cs.LG]*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. URL `http://arxiv.org/abs/1806.09055`.

K. Mason, J. Duggan, and E. Howley. Neural network topology and weight optimization through neuro differential evolution. In *Genetic and Evolutionary Computation Conference*, July 2017a.

K. Mason, J. Duggan, and E. Howley. Evolving multi-objective neural networks using differential evolution for dynamic economic emission dispatch. In *Genetic and Evolutionary Computation Conference*, pp. 1287–1294, July 2017b.

Nicole L. Mineu, Teresa B. Ludermir, and Leandro M. Almeida. Topology optimization for artificial neural networks using differential evolution. In *International Joint Conference on Neural Networks*, October 2010.

S. Das S. Mullick and P. Suganthan. Recent advances in differential evolution–an updated survey. *Elsevier SWEVO. Swarm and Evolutionary Computation*, 27:1–30, 2016.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. URL `http://arxiv.org/abs/1802.03268`.

E. Real, S. Moore, A. Selle, S. Saxena, Y. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *arXiv:1703.01041 [cs.NE]*, 2017.

E. Real, A. Aggarwal, Y. Huang, and Q. Le. Regularized evolution for image classifier architecture search. *arXiv:1802.01548 [cs.NE]*, 2018.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *CoRR*, abs/1902.08142, 2019. URL `http://arxiv.org/abs/1902.08142`.

David R. So, Chen Liang, and Quoc V. Le. The evolved transformer. *CoRR*, abs/1901.11117, 2019. URL `http://arxiv.org/abs/1901.11117`.

K. Stanley, D. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

K. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(8):24–35, 2019.

P. Kenneth R. Storn and J. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer, 2005.

R. Storn and P. Kenneth. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *J GLOBAL OPTIM*, 11:341–359, 1997.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. URL `https://doi.org/10.1038/s41592-019-0686-2`.

C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7105–7114, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/ying19a.html`.

A. Zela, J. Siems, and F. Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. *arXiv:1902.09635 [cs.LG]*, 2020a.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020b. URL `https://openreview.net/forum?id=H1gDNyrKDS`.

Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search, 2020c.

L. Zhang, H. Li, and X. Kong. Evolving feedforward artificial neural networks using a two-stage approach. *Neurocomputing*, 360:25–36, September 2019.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

A Canonical Differential Evolution

Differential Evolution (DE, Storn & Kenneth (1997)) is a simple yet effective evolutionary algorithm which proves the quality of its performance to solve a variety of optimization problems Storn & Lampinen (2005), Mullick & Suganthan (2016). This algorithm was originally introduced in 1995 by Storn and Price Storn & Kenneth (1997), and later attracted the attention of many researchers to propose new improved state-of-the-art DE algorithms to solve contemporary optimization problems Chakraborty (2008). DE algorithm is based on four steps (initialization, mutation, crossover and selection) we describe in the following.

Initialization. DE is a population-based meta-heuristic algorithm which consists of a population of NP individuals. Each individual is considered a solution and expressed as a vector of D-dimensional decision variables, which are initialized uniformly at random in the search range as follows:

$$pop_g = (X_{i,g}^1, X_{i,g}^2, ..., X_{i,g}^D), i = 1, 2, ..., NP \tag{2}$$

where g is the generation number, D is the dimension of the problem being solved and NP is the population size. We then evaluate the function $f(X)$ being optimized for each individual.

Mutation. A new child/offspring is produced using the mutation operation for each individual in the population by a so-called mutation strategy. In the classical DE, $rand/1$ is used in which three random individuals/parents denoted as $X_{r_1}, X_{r_2}, X_{r_3}$ are chosen to generate new vector V_i as follows:

$$V_{i,g} = X_{r_1,g} + F \cdot (X_{r_2,g} - X_{r_3,g}) \tag{3}$$

where $V_{i,g}$ is the mutant vector which we generate for each individual $X_{i,g}$ in the population space. F is the scaling factor that usually takes values within the range [0, 1] and r_1, r_2, r_3 are the indices of different random selected indices i.e., $r_1 \neq r_2 \neq r_3 \in [1, NP]$. As Eq.3 allows some parameters to be outside the search range, we check each parameter in $V_{i,g}$ and reset if it happens to be outside the boundaries.

Crossover. We apply uniform (binomial) crossover operation to each target vector $X_{i,g}$ and its corresponding mutant vector $V_{i,g}$ to generate a final trial vector $U_{i,g}$ as follows:

$$u_{i,g}^j = \begin{cases} v_{i,g}^j & \text{if } (rand < Cr) \text{ or } (j = j_{rand}) \\ x_{i,g}^j & \text{otherwise} \end{cases} \tag{4}$$

The crossover rate Cr is real-valued and is usually specified in the range [0, 1]. This variable controls the portion of parameter values that are copied from the mutant vector. j_{rand} is a random integer in the range [1, D]. The jth parameter value is copied from the mutant vector $V_{i,g}$ to the corresponding position in the trial vector $U_{i,g}$ if a random number is less than or equal to Cr. If the condition is not satisfied, then the jth position is copied from the target vector $X_{i,g}$.

Selection. After we generate the final offspring, the selection operation takes place to determine whether the target (the parent, $X_{i,g}$) or the trial (the offspring, $U_{i,g}$) vector survives to the next generation by comparing their function values. The offspring replaces the parent if it has a better function value as shown below. Otherwise, the new offspring is discarded, and the target vector remains in the population for the next generation.

$$X_{i,g} = \begin{cases} U_{i,g} & \text{if } (f(U_{i,g}) \leq f(X_{i,g})) \\ X_{i,g} & \text{otherwise} \end{cases} \tag{5}$$

We iterate the last three steps until a pre-defined number of function evaluation is reached.

B DE for NAS

B.1 Implementation details

The pseudo-code of DEHB algorithm is presented in Algorithm 1.

Algorithm 1: DE-NAS Algorithm

```
Input:
$f$ - NAS problem
$F$ - scaling factor (default $F = 0.5$)
$Cr$ - crossover rate (default $CR = 0.5$)
$NP$ - population size
Output: Return best found architecture in pop
g = 0
while ($|pop| < NP$)                                              ▷ Initialize population
 do
    $pop_i \leftarrow$ random_configuration();
    $pop'_i \leftarrow$ discretized_architecture($pop_i$);         ▷ Convert to discrete space
    $fitness_i \leftarrow$ evaluate_architecture($pop'_i$);
end
while ($g < g_{max}$) do
    $V_g \leftarrow$ mutate($pop_g$);                               ▷ Mutation operation
    $U_g \leftarrow$ crossover($v_g$, $pop_g$);                     ▷ Crossover operation
    $U'_g \leftarrow$ discretized_population($U_g$);
    $fitness_g \leftarrow$ evaluate_population($U'_g$);
    $pop_{g+1}$,$fitness_{g+1} \leftarrow$ select($pop_g$, $U_g$);   ▷ Selection operation
    $g$ = $g$+1;
end
```

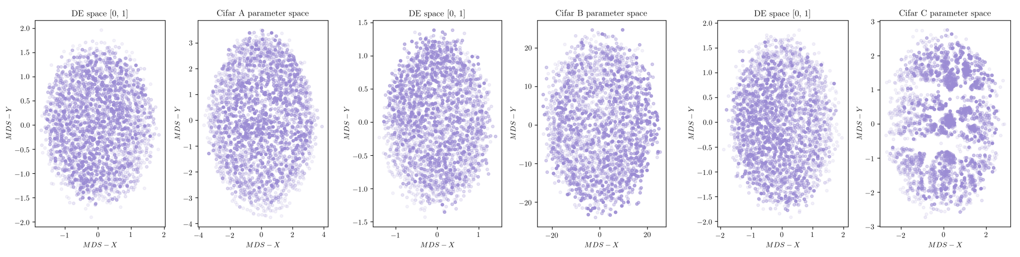
B.2 Continuous to NAS-space mapping

As discussed in Section 3, DE performs best under a uniform, continuous parameter space of [0, 1], where it can leverage the spanning of the search space using the scaled difference vector mutation. This however requires an adequate mapping back to the NAS parameter space to realize the architecture and obtain the evaluation on it. Given that every architecture parameter has a defined range or set of values, we have devised a simple mapping that takes into account the domain of every parameter and accordingly scales the DE individual back to the NAS parameter space. In the 4 benchmark analysis we performed, we encountered 4 different types of parameters, which in a general setting also suffices to broadly capture different parameter types: Integer, Float, Categorical and Ordinal. DE's search should be robust to the binning of Integer, Categorical and Ordinal parameters in the [0, 1] range, and the scaling of Floats to [0, 1]. Convergence of DE would ideally lead to a population of individuals from around the global optimal. DE has no prior information on the parameter space to begin with. It explores and exploits the unit hypercube formed by scaling all parameters to [0, 1] by responding to the objective function quality. In the Figures 5, 6, 7, we illustrate these notions by comparing the trajectory of optimisation in the [0, 1] space versus the original parameter space.

The optimisation trajectories were generated by running DE with a population of 20 for 100 generations. We use multi-dimensional scaling (MDS) Kruskal (1964) based on the $SMACOF$ algorithm Groenen & van Velden (2016) to project the two spaces in 2-dimensions for ease of visualisation. The *transparency* of the points correspond to the age of the point in the optimisation trajectory, with the most *opaque* point being the last found configuration. Figures 5, 6, 7 aim to show the correspondence of the search trajectories between the DE space of [0, 1] and the original NAS parameter space. Possible reasons for the empty space artefacts may arise due to the non-linear mapping of MDS that tries to preserve the pairwise distances in the 2-dimensional projections. Also, the variance in the number of categories in the domain of different categorical parameters in the benchmarks may be affecting the scale of the Euclidean distances in the original space, unlike the compressed scale of [0, 1] for DE. Nonetheless, the convergence of the search in both the spaces is illustrated adequately.

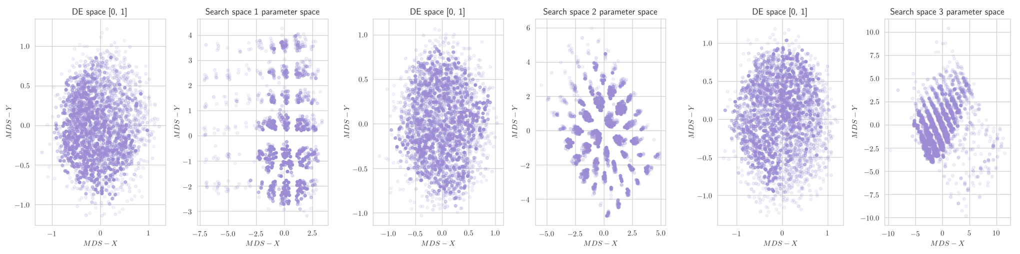
Boundary constraints

During the evolution process, it may happen that certain mutants/offsprings are at the edge of the unit hypercube that DE operates in. That is, certain X^i may not be within [0, 1]. Two ways to handle



(a) CifarA (b) CifarB (c) CifarC

Figure 5: MDS plots for NAS-Bench-101 comparing the optimisation trajectory in the DE space and the corresponding NAS parameter space



(a) Search space 1 (b) Search space 2 (c) Search space 3

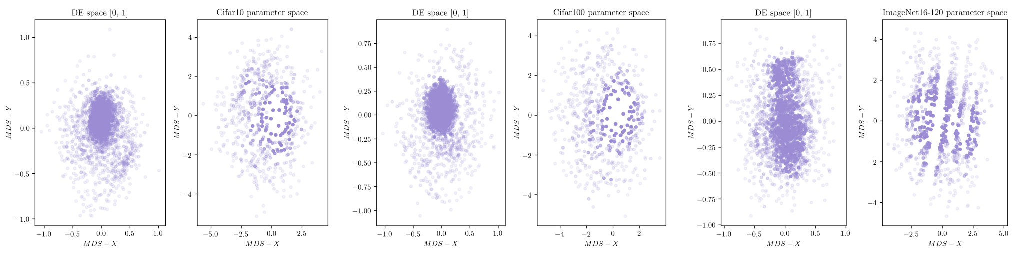
Figure 6: MDS plots for NAS-Bench-1shot1 comparing the optimisation trajectory in the DE space and the corresponding NAS parameter space

that would be to either clip values to 0 or 1, or randomly assign a value to X^i from [0, 1]. Thus ensuring that the subsequent continuous-to-discrete mapping will yield a legitimate configuration. In all experiments in this work, we used *random assignment* to handle boundary checks.

C Baseline Algorithms

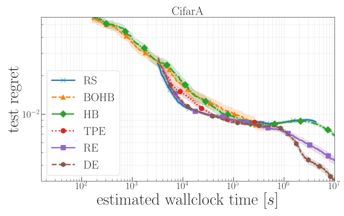
In our experiments, we used a well-performing setting for each of the used algorithms, except for the parameter free random search algorithm, as originally proposed by NAS-Bench-101 and the respective papers. As introduced by the different benchmarks being used here, the same encoding structure is used for all algorithms.

Random Search (RS) We sample random architectures in the configuration space from a uniform distribution in each generation.

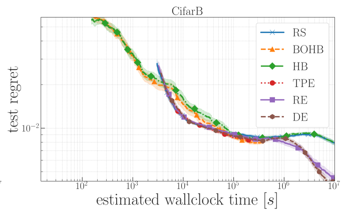


(a) Cifar10 (b) Cifar100 (c) ImageNet16-120

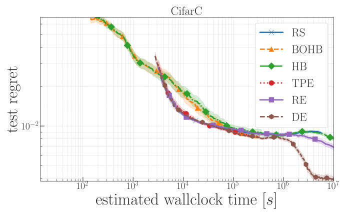
Figure 7: MDS plots for NAS-Bench-201 comparing the optimisation trajectory in the DE space and the corresponding NAS parameter space



(a) CifarA

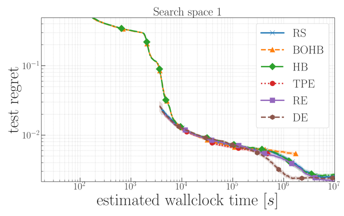


(b) CifarB

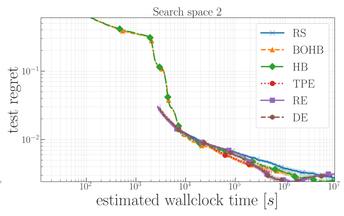


(c) CifarC

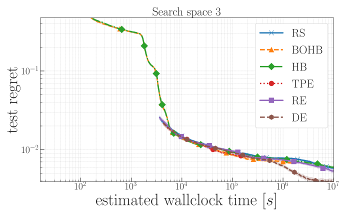
Figure 8: A comparison of the mean test regret performance of 500 independent runs as a function of estimated training time for NAS-Bench-101 on CifarA, CifarB and CifarC.



(a) Search Space 1



(b) Search Space 2



(c) Search Space 3

Figure 9: A comparison of the mean test regret performance of 500 independent runs as a function of estimated training time for NAS-1Shot1 on the three different search spaces.

BOHB We used the implementation from `https://github.com/automl/HpBandSter`. In Ying et al. (2019), they identified the settings of key hyperparameters as: the number of samples to optimize the acquisition function is set to 4, the minimum bandwidth for the kernel density estimator is set to 0.3 and bandwidth factor is set to 3. In our experiments, we deploy the same settings.

Hyperband (HB) We used the implementation from `https://github.com/automl/HpBandSter`. We set $\eta = 3$ and this parameter is not free to change since there is no other different budgets included in the NAS benchmarks we used for another value setting.

Tree-structured Parzen estimator (TPE) We used the open-source implementation from `https://github.com/hyperopt/hyperopt`. We kept the settings of hyperparameters to their default.

Regularized Evolution (RE) We used the implementation from Real et al. (2018). We initially sample an edge or operator uniformly at random, then we perform the mutation. After reaching the population size, RE kills the oldest member at each iteration. As recommended by Ying et al. (2019), the population size (PS) and sample size (TS) are set to 100 and 10 respectively.

D Experiments

D.1 NAS-Bench-101

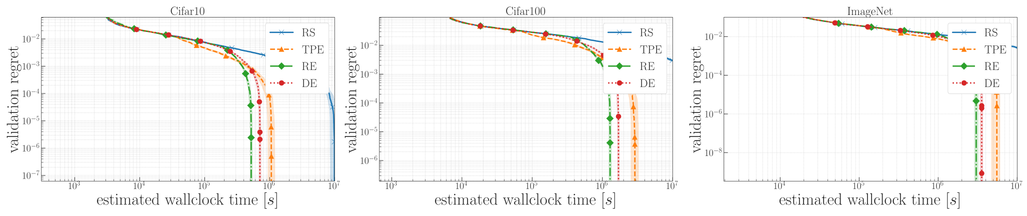
Figure 8 presents a comparison of the performance of compared algorithms showing the mean test regret of 500 independent runs as a function of the estimated training time for NAS-101.

D.2 NAS-Bench-1Shot1

Figure 9 presents a comparison of the performance of compared algorithms showing the mean test regret of 500 independent runs as a function of the estimated training time for NAS-1shot1.

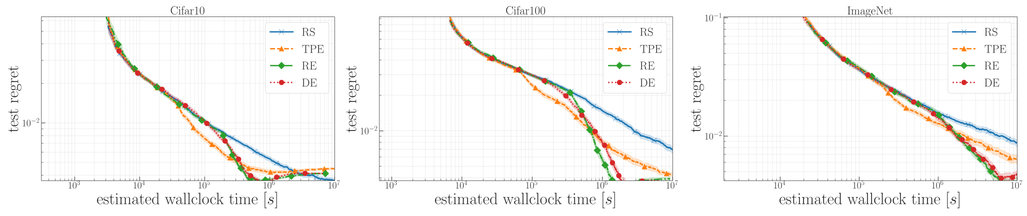
D.3 NAS-Bench-201

This benchmark follows the same direction as NAS-Bench-1Shot1 where they define cells as a DAG with 4 nodes and no restrictions on the maximal number of edges, while defining operations on the



(a) Cifar10 (b) Cifar100 (c) ImageNet16-120

Figure 10: A comparison of the mean validation regret performance of 500 independent runs as a function of estimated training time for NAS-201 on Cifar10, Cifar100 and ImageNet.



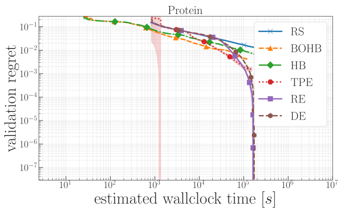
(a) Cifar10 (b) Cifar100 (c) ImageNet16-120

Figure 11: A comparison of the mean test regret performance of 500 independent runs as a function of estimated training time for NAS-201 on Cifar10, Cifar100 and ImageNet.

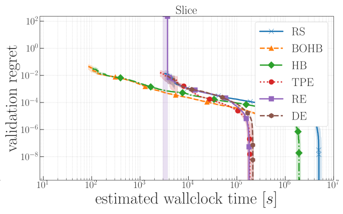
edges. With 5 defined operations, the space contains $15,625$ architectures. Other than Cifar-10, NAS-Bench-201 provides evaluations on 2 additional datasets, CIFAR-100 and ImageNet16-120. Figure 10 summarizes the mean performance of validation regret as a function of the estimated training time for 500 independent runs. We present the results for test regret in Figure 11. The budget-aware algorithms, HB and BOHB, are not included in our comparison since the benchmark does not yet allow (at the time of this work) to query the dataset at different epoch lengths. Referring to Cifar10, we conclude that even though TPE starts outperforming all other algorithms, it converges much slower towards the global optimum at the later end. DE achieves nearly the same test regret as RE, and surprisingly RS recovers from the misleading early evaluations and is able to converge faster than all other compared algorithms. For Cifar100, we observe a strong competition between RE and DE in which they achieve the same regret behaviour at the start of the search and then DE is slightly better than RE. Later RE converges faster and DE is able to perform comparably or slightly better at the end. For ImageNet, TPE achieves the same behaviours as in Cifar10 where it converges faster than all other algorithms but then it shows slower convergence. DE and RE outperform TPE at the later end while both of them achieve nearly the same test regret.

D.4 NAS-HPO

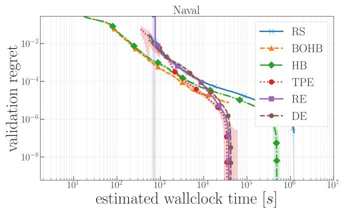
This benchmark was constructed as a joint architecture and hyperparameter optimization search problem for a 2-layer feedforward neural network and a linear output layer with parameterized architecture details and training parameters. The search space consists of a large grid of configurations on four popular UCI datasets for regression: protein structure, slice localization, naval propulsion and parkinsons telemonitoring. Figure 12 summarizes the mean performance of validation regret as a function of the estimated training time for 500 independent runs. We show the results for test regret in Figure 13. Although the multi-fidelity optimization algorithms HB and BOHB achieve good performance at the start of the search, they converge slower compared to DE and RE for all four datasets. We also observe that DE achieves the same test regret as RE for Protein and Slice datasets, and converges faster for Naval and Parkinsons. HB achieves a reasonable performance relatively quickly but it is not able to outperform RE and DE at the end of the search.



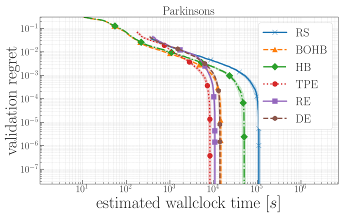
(a) Protein Structure



(b) Slice Localization

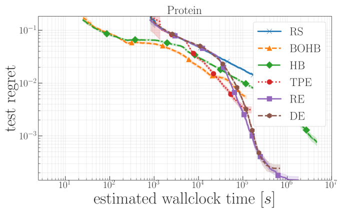


(c) Naval Propulsion

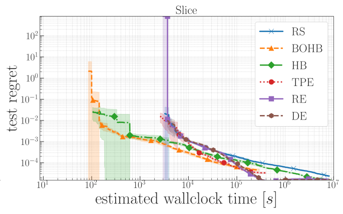


(d) Parkinsons Telemonitoring

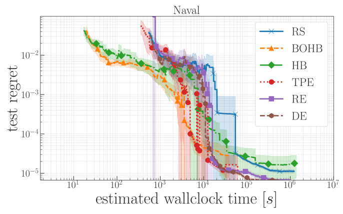
Figure 12: A comparison of the mean validation regret performance of 500 independent runs as a function of estimated training time for NAS-HPO on four UCI datasets: protein , slice, naval and parkinsons.



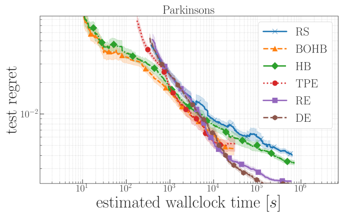
(a) Protein Structure



(b) Slice Localization

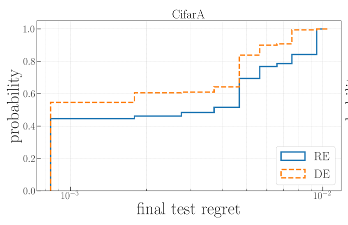


(c) Naval Propulsion

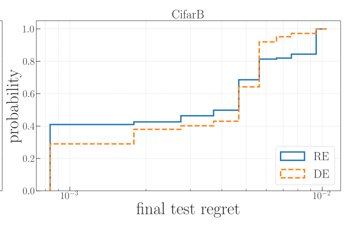


(d) Parkinsons Telemonitoring

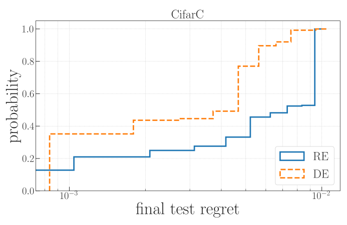
Figure 13: A comparison of the mean test regret performance of 500 independent runs as a function of estimated training time for NAS-HPO on four UCI datasets: protein , slice, naval and parkinsons.



(a) CifarA

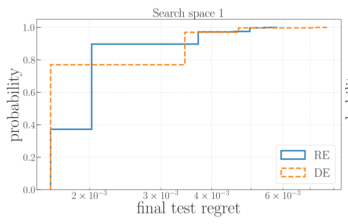


(b) CifarB

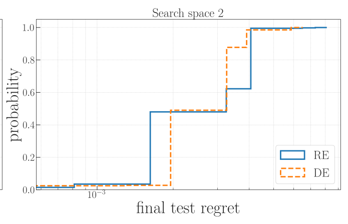


(c) CifarC

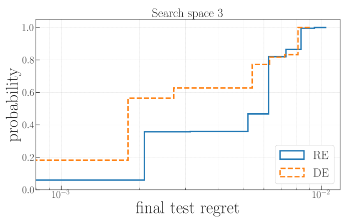
Figure 14: Empirical cumulative distribution of the final performance over all runs of DE versus RE over 500 runs after 10M seconds for NAS-Bench-101 on CifarA, CifarB and CifarC.



(a) Search Space 1



(b) Search Space 2



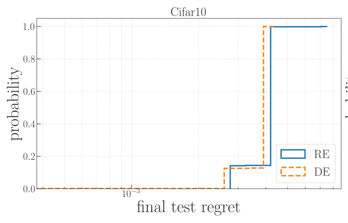
(c) Search Space 3

Figure 15: Empirical cumulative distribution of the final performance over all runs of DE versus RE over 500 runs after 10M seconds for NAS-Bench-1Shot1 on the three different search spaces.

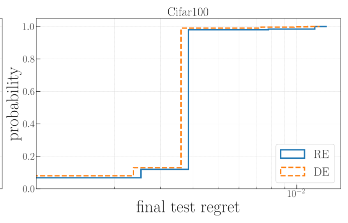
E ROBUSTNESS

In addition to obtaining a good performance, we test the robustness based on how an algorithm is sensitive to the randomness in both training process and search method. Since RE seems to be the competitor to DE, we show the empirical cumulative distribution of the *final test regret* after $10M$ seconds across 500 runs of RE and DE. Based on Figures 14, 15, 16 and 17 which present a comparison of the robustness between DE and RE for the used benchmarks, we can conclude the following:

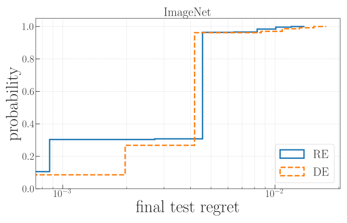
- For NAS-Bench-101, DE is robust in solving CifarA and CifarC while RE is better in solving CifarB.
- For NAS-Bench-1Shot1, DE is more robust to solve the three search spaces while we can say that RE is competitive in search space 2.
- For NAS-201, RE is more robust than DE in ImageNet while DE is competitively robust to RE in Cifar10 and Cifar100.
- For NAS-HPO, DE shows more robust performance in Slice and Parkinsons datasets. For Protein and Naval datasets, DE is competitively robust to RE.



(a) Cifar10

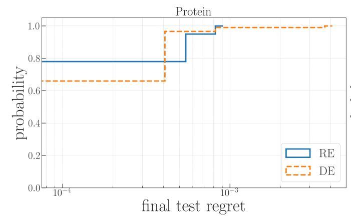


(b) Cifar100

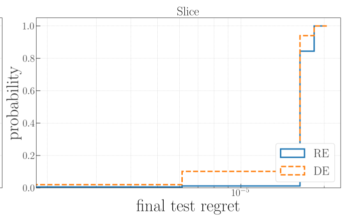


(c) ImageNet

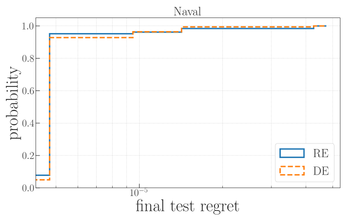
Figure 16: Empirical cumulative distribution of the final performance over all runs of DE versus RE over 500 runs after 10M seconds for NAS-Bench-201 on Cifar10, Cifar100 and ImageNet.



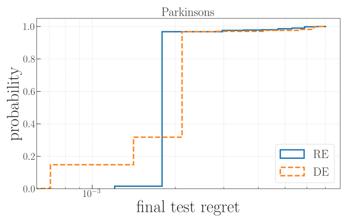
(a) Protein Structure



(b) Slice Localization



(c) Naval Propulsion



(d) Parkinsons Telemonitoring

Figure 17: Empirical cumulative distribution of the final performance over all runs of DE versus RE over 500 runs after 10M seconds for NAS-HPO on four UCI datasets: protein , slice, naval and parkinsons.

Table 1: A comparison of mean and standard deviation of the final test regret over 500 runs for the different optimization algorithms for NAS-Bench-101

	NAS-Bench-101		
	CifarA	**CifarB**	**CifarC**
RS	0.0655 ± 0.00267	0.0647 ± 0.00308	0.0657 ± 0.00254
BOHB	0.0649 ± 0.00703	0.0648 ± 0.00203	0.065 ± 0.0023
HB	0.06367 ± 0.00307	0.0647 ± 0.00292	0.0648 ± 0.00292
TPE	0.0654 ± 0.00235	0.0651 ± 0.00223	0.0652 ± 0.00236
RE	0.0612 ± 0.00342	0.0613 ± 0.00321	0.0637 ± 0.00378
DE	**0.0598** ± **0.00262**	**0.0611** ± **0.00225**	**0.0606** ± **0.00248**

We also report a comparison of the mean and standard deviation of the final found text regret over 500 runs for the different optimization algorithms in Tables 1, 2, 3 and 4 for NAS-Bench-101, NAS-Bench-1Shot1, NAS-Bench-201 and NAS-HPO respectively.

F Evaluating Population Size in DE

In this section, we provide an ablation study on the choice of the population size in DE. Figures 18, 19, 20 and 21 show a comparison of different population sizes over 500 independent runs on the different benchmark used: NAS-Bench-101, NAS-1Shot1, NAS201 and NAS-HPO respectively.

Table 2: A comparison of mean and standard deviation of the final test regret over 500 runs for the different optimization algorithms for NAS-Bench-1Shot1

	NAS-Bench-1Shot1		
	Search space 1	**Search space 2**	**Search space 3**
RS	0.0571 ± 0.00133	0.0603 ± 0.00183	0.0592 ± 0.00221
BOHB	0.0599 ± 0.00271	0.0606 ± 0.00215	0.0602 ± 0.00213
HB	0.0572 ± 0.00134	**0.06** ± **0.00178**	0.0594 ± 0.00221
TPE	0.0599 ± 0.00282	0.0609 ± 0.00232	0.0612 ± 0.00174
RE	**0.0566** ± **0.00076**	0.0607 ± 0.00122	0.0588 ± 0.00261
DE	**0.0569** ± **0.00097**	0.0605 ± 0.00113	**0.0573** ± **0.00303**

Table 3: A comparison of mean and standard deviation of the final text regret over 500 runs for the different optimization algorithms for NAS-Bench-201

	NAS-Bench-201		
	Cifar10	**Cifar100**	**ImageNet16-120**
RS	**0.0884 ± 0.00168**	0.2719 ± 0.00509	0.5403 ± 0.00619
BOHB	-	-	-
HB	-	-	-
TPE	0.0892 ± 0.00093	0.2693 ± 0.00191	0.5379 ± 0.00587
RE	0.0889 ± 0.00057	0.2689 ± 0.00157	**0.5358 ± 0.00347**
DE	0.0889 ± 0.00054	**0.2687 ± 0.00129**	0.5362 ± 0.00348

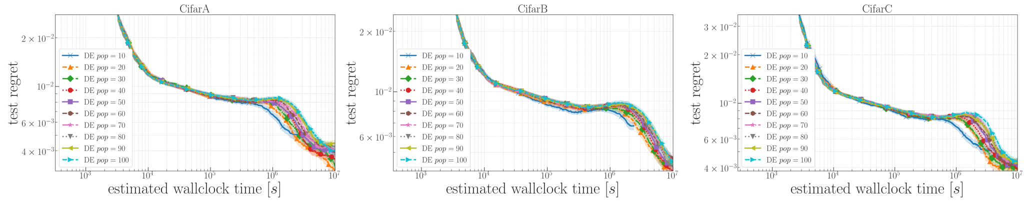
Table 4: A comparison of mean and standard deviation of the final text regret over 500 runs for the different optimization algorithms for NAS-HPO

	NAS-HPO			
	Protein	**Slice**	**Naval**	**Parkinsons**
RS	0.2176 ± 0.00246	0.00017 ± 1.85e-05	4.03e-05 ± 1.2e-05	0.0083 ± 0.00242
BOHB	0.2208 ± 0.00446	0.00019 ± 6.82e-05	5.73e-05 ± 2.3e-04	0.0089 ± 0.00685
HB	0.2161 ± 0.00124	**0.00016 ± 9.75e-06**	4.72e-05 ± 1.2e-04	0.0077 ± 0.00181
TPE	0.2185 ± 0.00486	0.00018 ± 3.17e-05	3.96e-05 ± 1.15e-05	0.0094 ± 0.009
RE	**0.2155 ± 0.00028**	**0.00016 ± 2.06e-06**	3.59e-05 ± 5.62e-06	0.0065 ± 0.00056
DE	0.2156 ± 0.00048	**0.00016 ± 3.54e-06**	**3.58e-05 ± 3.81e-06**	**0.0064 ± 0.00078**

Referring to these figures, we can conclude that DE is insensitive to the choice of population size at the beginning of the search till it reaches around 10M seconds, but later DE starts to converge at different speeds. Apparently the use of large population size slow the DE search towards the global optimum. We attribute this to many evaluations that DE performs for each generation. Setting the population size to 10 shows faster convergence but in most cases it gets trapped in a local optimum and is not able to recover from the bad regions. Hence, for our experiments we set the population size to 20 as it shows the best convergence behaviour across the generations for different benchmarks.

G Comparison with SciPy's DE

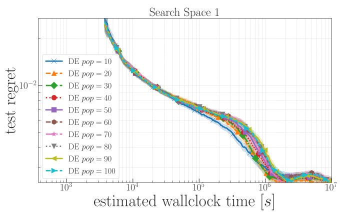
SciPy Virtanen et al. (2020) is one of the most popular open-source Python packages that offers a multitude of tools as abstraction for numerical methods. SciPy also has an implementation of DE that is feature-laden and is capable of running the classical DE with different mutation and crossover strategies [1]. However, we argue that such a flexible DE requires good choices of hyperparameters settings. The primary objective of this work is to standardize and benchmark an implementation of DE that achieves strong and robust performance for NAS. The design was intentionally kept light to make DE-NAS easy to study, customize if required, and also re-implement in other frameworks that

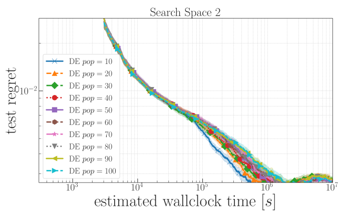


(a) CifarA (b) CifarB (c) CifarC

Figure 18: A comparison of different population sizes of DE over 500 independent runs for NAS-Bench-101 on CifarA, CifarB and CifarC.

[1]which is publicly available at `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html`


(a) Search Space 1


(b) Search Space 2

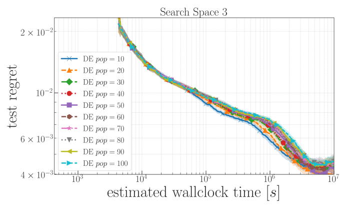
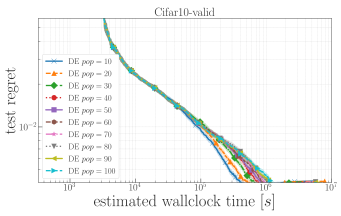
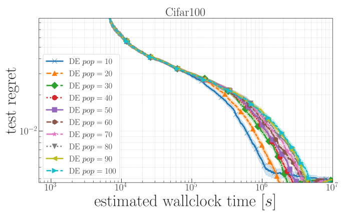

(c) Search Space 3

Figure 19: A comparison of different population sizes of DE over 500 independent runs for NAS-Bench-1Shot1 on the three different search spaces.


(a) Cifar10


(b) Cifar100

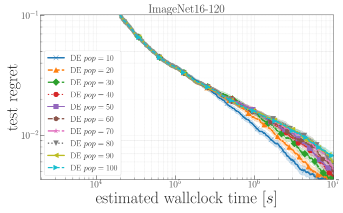
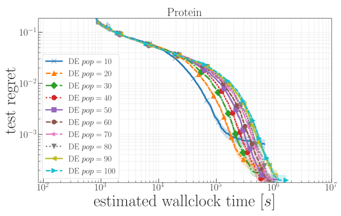
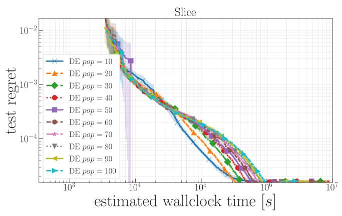
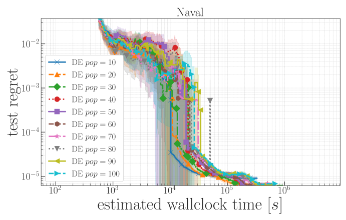

(c) ImageNet

Figure 20: A comparison of different population sizes of DE over 500 independent runs for NAS-Bench-201 on Cifar10, Cifar100 and ImageNet.


(a) Protein Structure


(b) Slice Localization


(c) Naval Propulsion

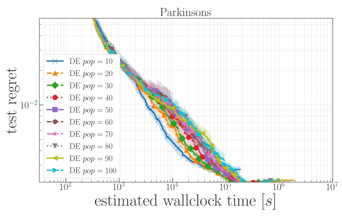
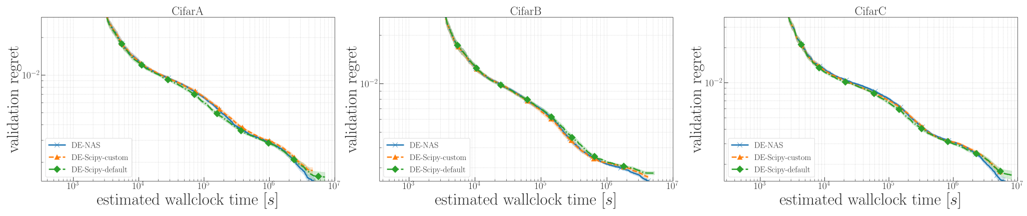
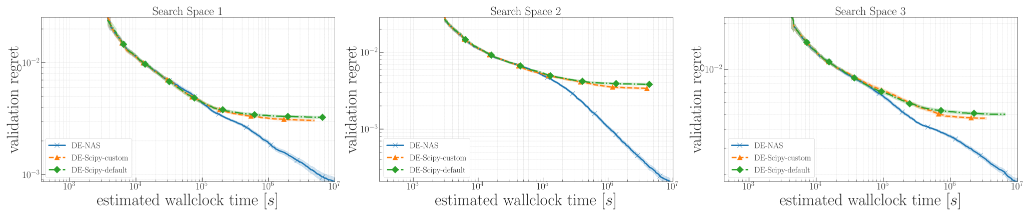

(d) Parkinsons Telemonitoring

Figure 21: A comparison of different population sizes of DE over 500 independent runs for NAS-HPO on four UCI datasets: protein , slice, naval and parkinsons.



(a) CifarA (b) CifarB (c) CifarC

Figure 22: Comparison of DE-NAS with Scipy-DE over 500 independent runs for NAS-Bench-101 on CifarA, CifarB and CifarC.

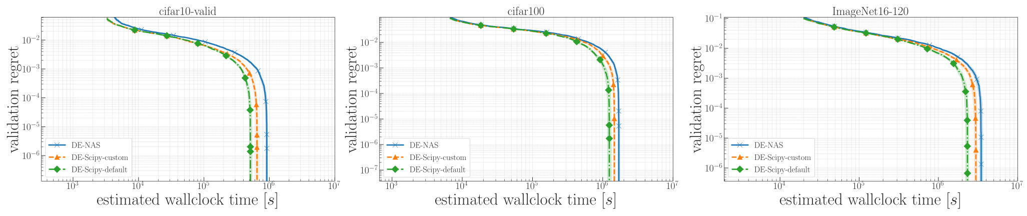


(a) Search Space 1 (b) Search Space 2 (c) Search Space 3

Figure 23: Comparison of DE-NAS with Scipy-DE over 500 independent runs for NAS-1shot1.

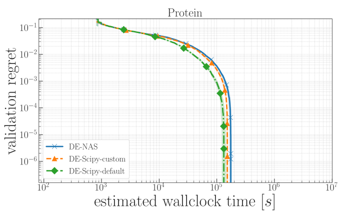
need to run NAS or other optimization benchmarks with different types of search spaces. However, for a fair assessment, we ran Scipy's DE on all the NAS benchmarks used in this work and compared it to our implementation. Since Scipy's DE just deals with continuous space, we consider the same use of our discretization method to be able to run a NAS problem.

Figures 22, 23, 24 and 25 show the comparison of SciPy DE and DE-NAS. All the algorithms were run with the same population size, the total generations evolved, and therefore the total number of function evaluations. The plots show the mean validation regret for 500 independent runs. *DE-Scipy-default* is the version of DE that runs with the default hyperparameter settings in SciPy which uses ($F \in [0, 2]$, $Cr \in [0, 1]$). *DE-Scipy-custom* is the version of DE-Scipy with the same hyperparameter settings we used in our implementation ($F = Cr = 0.5$). It must be noted that *DE-Scipy-default* implicitly uses a large population size for its default settings ($NP = NP_{init} \times D$) but for a fair comparison, we set it to 20 as we use in our implementation. Despite being simpler and purely stochastic, DE-NAS is much better than *DE-Scipy-default* for NAS-101 and NAS-1shot1. This may highlight that specialized algorithms may have more modes of failure that affect robustness. Also as we expected, DE-NAS and *DE-Scipy-custom* perform comparably across 10 different benchmarks (NAS-101, NAS-HPO, NAS-201). This illustrates that our DE-NAS is well suited and evidently a better choice for NAS with DE for search. The divergence in performance between *DE-Scipy-custom* and our DE-NAS for NAS-1shot1 benchmark is however curious and its discussion remains open for future work.

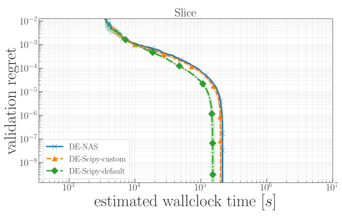


(a) Cifar10 (b) Cifar100 (c) ImageNet

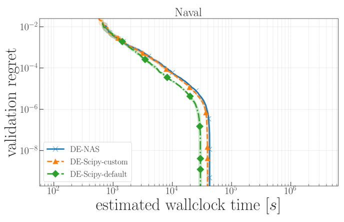
Figure 24: Comparison of DE-NAS with Scipy-DE over 500 independent runs for NAS-201.



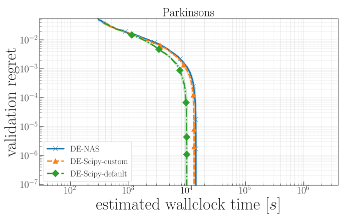
(a) Protein Structure



(b) Slice Localization



(c) Naval Propulsion



(d) Parkinsons Telemonitoring

Figure 25: Comparison of DE-NAS with Scipy-DE over 500 independent runs for NAS-HPO.