**DiBB: Distributing Black-Box Optimization**

Giuseppe Cuccu   
Luca Rolshoven∗  
Fabien Vorpe∗  
Philippe Cudré-Mauroux   
name.surname@unifr.ch   
Exascale Infolab   
University of Fribourg   
Switzerland

**ABSTRACT**

DiBB (for Distributing Black-Box) is a meta-algorithm and frame-work that addresses the decades-old scalability issue of Black-Box Optimization (BBO), including Evolutionary Computation. Algo-rithmically, it does so by creating out-of-the-box a Partially Sep-arable (PS) version of any existing black-box algorithm. This is done by leveraging expert knowledge about the task at hand to define *blocks* of parameters expected to have significant correla-tion, such as weights entering a same neuron/layer in a neuroevo-lution application. DiBB distributes the computation to a set of machines without further customization, while still retaining the advanced features of the underlying BBO algorithm, such as scale invariance and step-size adaptation, which are typically lost in recent distributed ES implementations. This is achieved by instan-tiating a separate instance of the underlying base algorithm for each block, running on a dedicated machine, with DiBB handling communication and constructing complete individuals for evalu-ation on the original task. DiBB’s performance scales *constantly* with the number of parameter-blocks defined, which should allow for unprecedented applications on large clusters. Our reference implementation (Python, on GitHub and PyPI) demonstrates a 5x speed-up on COCO/BBOB using our new PS-CMA-ES. We also showcase a neuroevolution application (11 590 weights) on the PyBullet Walker2D with our new PS-LM-MA-ES.

**CCS CONCEPTS**

• **Computing methodologies** → **Randomized search**.

**KEYWORDS**

Black-Box Optimization, Distributed Algorithms, Parallelization, Evolution Strategies, Neuroevolution

∗This paper includes contributions from Rolshoven’s and Vorpe’s Master Thesis work.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

*GECCO ’22, July 9–13, 2022, Boston, MA, USA*  
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-9237-2/22/07...$15.00 <https://doi.org/10.1145/3512290.3528764>

341

GECCO ’22, July 9–13, 2022, Boston, MA, USA Cuccu, et al.

(e.g. sep-CMA-ES [27] and SNES [29]), trading off significant con-vergence speed for wall-clock speed by relinquishing covariance information altogether [18]. Previous work by the authors also cov-ers a generalization of this trade-off by establishing a *block-diagonal covariance* matrix [6], leveraging the fact that the correlation among variables is not uniform for most complex problems, and that this information is often available to the user based on the target task. This provides initial inspiration for this work, as discussed below.

**1.1**  **Intuition and design**

BBO algorithms are designed to work in any unknowable (black-box) environment. By design, BBO algorithms cannot integrate task knowledge, in order to ensure that no illegitimate assumptions are made in their design. In most real-world applications however, some degree of expert knowledge is often available about the end task, making it rather a *gray-box* setting. Expert knowledge about the correlation between variables is often simple to deduce, as it is task-specific.

For example in neuroevolution, where evolutionary algorithms learn the parameters of a neural network, weights of connections entering the same neuron are by necessity highly correlated, as the network’s equation aggregates them in a linear combination prior to activation. While weights entering different neurons are not entirely uncorrelated, the expectation on their covariance is (relatively) significantly lower. A similar reasoning is easily made about neurons entering a same layer versus neurons in different layers, and remains true as the network expands towards deep networks1. The assumption of *partial correlation*, which allows partitioning the variables into highly intra-correlated **blocks**, leads to the blocks being separable between one another, i.e. low inter-correlation. The corresponding covariance matrix becomes *block-diagonal*, as explored in our previous work [6].

This paper extends the concept by leveraging the underlying assumption of separability between blocks to optimize each block of variables using a different, independent instance of the same BBO algorithm. As a consequence, block-wise computation can now be distributed across multiple machines: not only the individual fitnesses can be evaluated in an embarrassingly parallel fashion, but different blocks can be searched asynchronously, and each BBO instance can be distributed to different nodes in a cluster (see Figure 1).

Based on the above insights, this paper proposes a new meta-algorithm and framework for Distributed Black Box optimization, named **DiBB**. DiBB is particularly suited for large-scale problems evidently structured, as not infrequent (arguably common) in real-world applications. We provide rigorous theoretical arguments for the sample efficiency of this approach in Section 2, and further explore the example application of neuroevolution in Section 4.3.

**1.2**  **Challenges**

A major challenge comes with solution evaluation: a BBO instance running on one machine will produce a population of samples that are *incomplete individuals*, as the variables constituting a complete

1The fundamental assumption of partial separability across network layers however does not seem to be very well studied in the literature, despite a considerable body of work on neural network loss landscapes [10, 31]. This can be partially tracked to the limited availability of algorithms making use of partial correlation information.

342

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DiBB: Distributing Black-Box Optimization | | |  | GECCO ’22, July 9–13, 2022, Boston, MA, USA | | |
| ***Head*** *node* | *BLOCK WORKER* | | *Worker node* ***A*** | *BLOCK WORKER* | | *Worker node* ***B*** |
| *DIBB* | *BBO instance* | Local copy of |  | *BBO instance* | Local copy of  **global** | |
|  |
| *MAIN ROUTINE* | **global** |
|  | **Local** | reference solution: | **Local** | reference solution: | |
| Object Store and | reference solution: | A B **C D** E F | |  |  | | --- | --- | |  |  | |  |  |   objective function   |  |  | | --- | --- | | *FITNESS*  *EVALUATOR*  Complex / expensive |  | |  |  | | reference solution: | A B C D **E F** | |  |  | | --- | --- | |  |  | |  |  |   objective function   |  |  | | --- | --- | | *FITNESS*  *EVALUATOR*  Complex / expensive |  | |  |  | |
| Communication | Complete solutions | **E F** | Complete solutions |
| **C D** |
| array |
| Candidate samples: | ready for | Candidate samples: | ready for |
| **Global** | evaluation: | evaluation: |
| **X X** | A B **X X** E F | **P T** | A B C D **P T** |
| reference solution: |
| **Y Z** | **S P** |
| **A B C D E F** | A B **Y Z** E F | A B C D **S P** |
| **Y Y** | **R T** |
| A B **Y Y** E F | A B C D **R T** |

**Figure 1: Example architecture of DiBB as a framework. This is a sample instantiation on a cluster with 3 nodes: one acts as the head node, running the main routine, while two worker nodes encapsulate the underlying BBO and host the pools of Fitness Evaluators. This schema highlights how DiBB leverages Partial Separability in the construction of full samples ready for evaluation, and how the (potentially expensive) objective function is evaluated locally on the worker nodes.**

**2**  **A PRIMER ON EVOLUTION STRATEGIES**

DiBB can be applied to any BBO algorithm to great and immediate advantage: even with methods that already treat the parameters as separable, DiBB provides a parallel and distributed implementation at the only cost of minimal overhead (for nontrivial applications). In this section however, we highlight modern Evolution Strategies as the state-of-the-art family of continuous black-box optimiza-tion methods, which we expect to gain the most from this new approach. The rationale here is that these methods have proven and popularized the importance of maintaining covariance infor-mation for search performance, and are thus best positioned to gain from DiBB’s scalability into distributed computation. This choice however should not be interpreted as limited to neuroevolution applications, as DiBB users can adapt the partitioning of variables into blocks depending on each problem at hand.

ES are direct search methods, which optimize a black-box objec-adaptive Gaussian distribution. We briefly review the types of ES tive function *𝑓* : R*𝑑*→ R by sampling candidate points from an most relevant for our discussion. The classic variant is the (1+1)-ES [26]. Its central algorithmic mechanism is step-size adaptation, i.e. its ability to actively adapt the standard deviation *𝜎 >* 0 of its Gaussian sampling distribution N (*𝑚, 𝜎*2*𝐼*) to the current needs. For the last 20 years, CMA-ES [18] has been the gold standard in ES research. Many variants exist, such as Natural Evolution Strategies (NES; Wierstra et al. 34). Its most important mechanism going be-yond “simple” step-size adaptive ES is covariance matrix adaptation (CMA), which means that not only the global step size *𝜎*, but also the full covariance matrix *𝐶* of the Gaussian N (*𝑚, 𝜎*2*𝐶*) is adapted to the problem at hand.

CMA-ES is a powerful optimizer; however, it was not designed for high-dimensional applications with hundreds of thousands of variables or more. Its internal parameters are not tuned with such a ~~2~~ regime in mind, and learning a full covariance matrix with*𝑑* ~~(~~*𝑑*~~+~~1~~)~~ parameters is inherently slow. Such problems are commonly ad-dressed by placing specific restrictions on *𝐶*, such as being repre-sented by a diagonal matrix [27, 29], or a diagonal plus a low-rank matrix [1, 22, 23], or a block-diagonal matrix [6]. The number of parameters of the covariance matrix can hence be chosen flexibly

343

GECCO ’22, July 9–13, 2022, Boston, MA, USA Cuccu, et al.

The OpenAI-ES [28], while being highly distributable, features neither step-size adaptation nor covariance adaptation. Based on the NES framework of Wierstra et al. [34], it leverages the ability of ES to estimate the natural gradient of *𝑓* from samples, and then applies the ADAM optimizer on top. In effect, this is roughly comparable to using a diagonal *𝐶*.

CMA has a price in terms of *algorithm internal* complexity, and in addition the adaptation process is slow even in terms of sample complexity—due to large hidden constants in its performance es-timation. The above convergence rates measure time in terms of the number of objective function evaluations (sample complexity). When scaling up CMA to high dimensions however, we need to take the following concepts into consideration. Algorithm internal complexity refers to the required (amortized) number of operations needed for creating a sample and for updating the internal state—the covariance matrix in particular. This concept will be re-explored later when introducing distributed computing, as DiBB employs multiple machines to evaluate proportionally more samples without impacting wall-clock time. For now, regarding sample complexity, we distinguish between the number of samples needed to learn the covariance matrix, and the number of samples needed to solve the problem.

For example, a (small) neural network with *𝑑* = 104weights results Learning *𝐶* with up to Θ(*𝑑*2) parameters is sample-inefficient.

in*𝑑* ~~(~~*𝑑*~~+~~1~~)~~ these takes hundreds of millions of samples, each of which can be ~~2~~ ≈ 5 · 107parameters of the covariance matrix. Learning

significantly expensive: imagine for example a continuous control task involving a robot interacting in a physics simulation. This quickly becomes far too slow for practical use, even without taking into consideration the (typically expensive) covariance update step in the underlying algorithm. For larger *𝑑*, even the storage of the full covariance matrix *𝐶* quickly becomes prohibitive. Furthermore, performing computations with *𝐶* scales at least linear with the number of its parameters, which amounts to an internal complexity of Ω(*𝑑*2) for full CMA. Since network evaluation scales linearly with the number of weights*𝑑* (in a direct encoding scheme), CMA quickly becomes the computational bottleneck. Therefore, a different trade-off between fast convergence and internal complexity is needed, which can be realized for example with block-diagonal and low-rank structures, and combinations thereof. Or, in the case of the proposed work, by leveraging the partial correlation assumption to construct a block-diagonal covariance matrix.

**2.3**  **Implications for Neural Network Training**

Diagonal, block-diagonal, and low-rank schemes successfully lower both internal and sample costs of CMA significantly (at the expense of higher sample complexity for solving the overall problem). There-fore, they are key to the application of modern ES based on CMA to real-world problems, especially involving expensive physics sim-ulations as common in reinforcement learning applications.

In neural network training, weight spaces are often extremely high-dimensional. However, they also come with a canonical struc-ture, induced by the network topology. Hence, a block-diagonal covariance structure with one block per layer (or per neuron) is a natural choice. It should be noted that there exist approaches for identifying a problem decomposition automatically [24], if needed,

344

DiBB: Distributing Black-Box Optimization

The BW encapsulates the actual BBO algorithm, and runs fully asynchronously from the others, though contemporary.

The BWs exchange information by uploading to the head node the state of their search after each update (i.e. a *gen-eration* in ES); updates from the other nodes are also pulled before generating a new population.

(4) In our implementation, each BW can spawn a pool of *Fitness Evaluators* (FE) on the same machine, used to automatically manage limited computational resources (such as available CPUs).

(5) Alternatively, the user can request to evaluate trivial opti-mization functions on the BW directly (either sequentially or using multithreading), which is useful when the over-head of maintaining a discrete pool of evaluators would be significant with respect to the cost of the actual evaluation task.

The Block Workers communicate with the head node in each generation, while generation cycles are defined asynchronously and autonomously by each BW. Consider a problem with *𝑑* variables *𝑥*1*, . . . ,𝑥𝑑*, and a BW optimizing *𝑏* variables *𝑥𝑎, . . . ,𝑥𝑎*+*𝑏*−1, denoted as the vector *𝑥𝐵* for short. The head node maintains a *reference solution* ¯*𝑥* ∈ R*𝑑*, which fulfills a two-fold purpose: it serves as an anytime-estimate of the state of the search (current optimum), and it provides a unifying context to the BWs and the FEs.

Intuitively, each BW is only aware of the variables in one block, and can only generate samples for the corresponding variables. These incomplete samples however need to be constructed as part of a complete solution in order to be scored on the task. In our recurring example of neuroevolution, the sample could correspond to the weights for a neuron, or layer, while obviously only full networks can be evaluated on the task. We address this issue by leveraging once again our assumption of partial separability. After our hypothesis of the correlation across blocks being negligible, we can evaluate each block in isolation by inserting it in the context of the (current) reference solution, as obtained from the head node. Hypothesize for a moment that there is no correlation inter-block—we will address the validity of this statement just below. In this case, each layer can be scored fairly by constructing a full reference network, then swapping the corresponding weights in the target layer for the block sample, and finally evaluating the resulting complete network on the task. Different independent sam-ples from a same block (individuals) would receive fair evaluation in this fashion as long as they are evaluated on the same refer-ence network. This is in fact constructed by assembling the partial sample into the global reference solutions. Partial samples are con-structed by aggregating the reference or center sample from each of the BBO instances running on each block, which as mentioned is maintained in the head node by each block instance at the end of each generation.

More formally: at the start of each generation, the BW receives the current reference solution ¯*𝑥*1*, . . . ,* ¯*𝑥𝑑* from the head node.2 The block-level ES samples a population of candidate solutions *𝑦*1*, . . .𝑦𝜆*∈ R*𝑏*. If *𝑏* is small, then sampling from a Gaussian with full covariance matrix N (*𝑚, 𝜎*2*𝐶*) is feasible. The *𝑏*-dimensional

2It would suffice to send ¯*𝑥*1*, . . . ,* ¯*𝑥𝑎*−1*,* ¯*𝑥𝑎*+*𝑏, . . . ,* ¯*𝑥𝑑* to the BW, but the difference has been negligible in our experiments so far up to 30 blocks.

345

GECCO ’22, July 9–13, 2022, Boston, MA, USA Cuccu, et al.

The problem is smartly sidestepped by Salimans et al. [28] by synchronizing the *entropy* (random number generation) across the machines, which makes them generate the same samples, however at the price of a fully-synchronous implementation (and greatly sim-plified algorithm). The same method can be implemented into DiBB to allow a Block Worker to spawn Fitness Evaluators across multiple machines, which would immediately improve performance partic-ularly in the case of large population sizes. With DiBB however, the actual updates would remain asynchronous, which implies that the FEs spawned in the cluster could actually be shared between BWs. Since this communication overhead occurs only once per (block-wise) generation in DiBB, and not once per fitness evalua-tion, our implementation has considerable less pronounced network overhead than in the distribution of fitness evaluation in a plain ES.

**4**  **EXPERIMENTS**

This section describes the setup used to empirically assess the performance of DiBB. With our experiments, we aim to address the following research questions:   
 Q1: How well does the block-diagonal approach work, compared to diagonal- and full-covariance CMA?

Q2: How does DiBB’s performance scale to a large number of machines and cores?

Q3: Is DiBB well-suited for neuroevolution applications?

We assess the first two questions on the standard COmparing Con-tinuous Optimizers (COCO) Black Box Optimization Benchmark (BBOB), using both the standard [17] and the *large-scale* [9] bench-mark suites. For answering the third question, we showcase a neu-roevolution application in the challenging OpenAI Gym 2D Walker environment.

**4.1**  **Setup and reference implementation**

We tested DiBB on a variety of hardware solutions. The COCO-BBOB experiments were run on a cluster of 24 low-performance machines, all based on an Intel(R)Core(TM)i7-2600 CPU @ 3.40GHz (4 cores/8 threads each), and 32 GB of RAM. These machines are far from state-of-the-art performance, which leaves a significant margin of improvement for the timings presented in our results. This decision was taken to encourage interested labs to reproduce our results on whatever hardware they can put together, without expectation of dedicating any significant budget.

The cluster setup is simplified by the included managing scripts. Running on a single machine with a single block and no Fitness Evaluators roughly corresponds to running the underlying BBO algorithm alone (plus overhead, and with parallel fitness evaluation), and can be achieved without setup with a syntax alike to CMA-ES. Spawning multiple BWs and FEs automatically scales to the available resources, as declared in the managing script using a simple list of network IPs.

The experiments below are based on our reference implementa-tion of DiBB written in Python, which leverages the Ray distributed computation library3. The code is released open source on GitHub

3<https://www.ray.io/>— a Python framework for distributed computing

346

|  |  |  |  |
| --- | --- | --- | --- |
| DiBB: Distributing Black-Box Optimization |  | GECCO ’22, July 9–13, 2022, Boston, MA, USA | |
|  |  |  |

**Figure 2: ECDF plots of the performance of PS-CMA-ES, sep-CMA-ES and standard CMA-ES in dimension 320. From left to right: f1-f5 with Exp. 1/Exp. 3, Exp. 2/Exp. 4, then f10-f14 with Exp. 1/Exp. 3, Exp. 2/Exp. 4. The curves show the fraction of precision targets reached over time, measured as number of function evaluations divided by problem dimension, on a logarithmic scale. For additional information on ECDF plots, please refer to [16].**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of** | **Number** | **Block** | **Duration** |
| **dimensions** | **of blocks** | **size** |
| 40  80  160  320  640 | 1  2  4  8  16 | 40  40  40  40  40 | 25h 13m 47s 43h 26m 57s 40h 21m 01s 44h 53m 29s 61h 20m 11s |
| **Table 2: Timing results for Experiment 4: BBOB large-scale suite with fixed block size (**5× **budget). Larger problem di-mensions are addressed here with increasing the number of blocks. Since each block is optimized in parallel, the increase in run time of the experiments is only limited to communi-cation overhead, thus small as problem dimensions grow.** | | | |

process by providing publicly available performance data for many state-of-the-art algorithms.7   
 We test the scaling of DiBB’s performance as we vary in turn the problem dimensions and the block size, from the related but radi-cally different perspectives of sample efficiency (i.e. convergence speed, how many samples it takes for the algorithm to reach con-vergence) and effective run time (i.e. wall-clock speed, how long it actually takes for the algorithm to converge in real time). In the case of DiBB, run time is significantly (positively) impacted by its inher-ent distributed implementation, as each BW evaluates the fitness of its individuals locally in its dedicated machine, asynchronously from all other BWs.

To this end, we ran the following experiments:   
 (1) Constant number of blocks, increasing *𝑑* and block size, on the BBOB suite (bbob\_fnb)   
 (2) Constant block size, increasing *𝑑* and number of blocks on the BBOB suite (bbob\_fbs)   
 (3) Constant number of blocks, increasing *𝑑* and block size on the large-scale suite (bbob\_ls\_fnb)   
 (4) Constant block size, increasing *𝑑* and number of blocks on the large-scale suite (bbob\_ls\_fbs)   
 (5) Comparison of the wall-clock speed between plain CMA-ES and PS-CMA-ES using different block sizes   
Complete details of the experimental setup for reproducibility pur-pose are found in Appendix A.1.

7See [https://numbbo.github.io/data-archive/bbob/.](https://numbbo.github.io/data-archive/bbob/)

347

GECCO ’22, July 9–13, 2022, Boston, MA, USA Cuccu, et al.

|  |  |  |  |
| --- | --- | --- | --- |
| **Blocks** | **Duration** | **Relative Time** | **Speed Gain** |
| 1  2  4  8  16 | 17h 43m 41s 17h 33m 30s 06h 32m 19s 03h 56m 56s 03h 06m 05s | 100.00%  99.04%  36.88%  22.27%  17.49% | 0.00%  0.97%  171.13% 348.94% 471.62% |

**Table 3: Timing results for Experiment 5: BBOB large-scale suite in 160 dimensions using different block sizes. The first row (one block) corresponds to running CMA-ES without DiBB. When using only two blocks, we observe almost the same wall-clock time. For larger number of blocks however, the time is significantly reduced.**

40d BBOB large-scale suite with one block (another single-machine run).

Tables 1 and 2 compare the run times of DiBB with different numbers of blocks, block sizes, and dimensions on the BBOB large-scale suite. Several effects come together: in larger dimensions, function evaluation time and communication overhead grow lin-early. Due to the fixed budget multiplier used in COCO, the overall function evaluation budget also grows linearly. On the other hand, CMA’s computational effort grows quadratically in the block size. We clearly observe that the runtime grows far more benign when using a fixed block size, which indicates that CMA overhead indeed quickly becomes the dominating term. Hence, the block size should be kept tightly under control. This answers question Q2. Addition-ally, Table 3 compares the wall-clock speed of PS-CMA-ES using different block sizes on the 160d BBOB large-scale problems suite. For one vs. two blocks, there is almost no difference in the duration of the experiment. However, for four or more blocks, the wall-clock time diminished drastically, leading to a speed gain of 471%. Note that for BBOB and most other benchmark problems, evaluations are unrealistically cheap, so DiBB’s parallelization overhead becomes relatively significant. For a more realistic function evaluation cost (i.e. as little as 10ms) the overhead becomes negligible, highlighting the benefit of parallel evaluations.

**4.3**  **PyBullet Walker 2D**

Although DiBB can be applied to any existing BBO and problem, its value in neuroevolution applications has undeniably been one of the original inspirations, particularly in relation to neural net-works of large size. Therefore we present our results on a complex reinforcement learning control task: the Walker 2D environment from PyBullet [5], instantiated through the OpenAI Gym [3].

In this task, an agent controls a basic 2D robotic walker using the PyBullet physics simulator. The observation is a 22-dimensional reading of the environment (e.g. aperture and angular velocity of each joint, etc.), while the action is a 6-dimensional torque-control signal. The goal of the task is for the robot to walk the farthest distance possible in the allotted time, without toppling, which in turn requires learning a usable gait.

The policy network is feed-forward and fully connected, com-posed by two hidden layers of sizes [128*,* 64], using ReLU activa-tions. The output layer is composed of six neurons with rescaled tanh activation normalized to the range of motor commands.

348

DiBB: Distributing Black-Box Optimization

**REFERENCES**   
[1] Youhei Akimoto, Anne Auger, and Nikolaus Hansen. 2014. Comparison-based natural gradient optimization in high dimension. In *Genetic and Evolutionary*  *Computation Conference*. ACM, 373–380.

[2] Charles Audet and Warren Hare. 2017. *Derivative-free and blackbox optimization*. Springer.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John OpenAI Gym. Schulman, Jie Tang, and Wojciech Zaremba. 2016.

arXiv[:arXiv:1606.01540](https://arxiv.org/abs/arXiv:1606.01540)   
[4] Patry[k Chrabaszcz, Ily](https://arxiv.org/abs/arXiv:1606.01540)a Loshchilov, and Frank Hutter. 2018. *Back to basics:*  *Benchmarking Canonical Evolution Strategies for Playing ATARI*. Technical Report 1802.08842. arXiv.org.

[5] Erwin Coumans and Yunfei Bai. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. [http://pybullet.org.](http://pybullet.org)

[6] Giuseppe Cuccu and Faustino Gomez. 2012. Block di[agonal natural evo](http://pybullet.org)lution strategies. In *International Conference on Parallel Problem Solving from Nature*. Springer, 488–497.

[7] Giuseppe Cuccu, Julian Togelius, and Philippe Cudré-Mauroux. 2019. Playing Atari with six neurons. In *Proceedings of the 18th International Conference on*  *Autonomous Agents and MultiAgent Systems*. International Foundation for Au- tonomous Agents and Multiagent Systems, 998–1006. [https://exascale.info/](https://exascale.info/assets/pdf/cuccu2019aamas.pdf)  [assets/pdf/cuccu2019aamas.pdf](https://exascale.info/assets/pdf/cuccu2019aamas.pdf)   
[8] [Jeffrey Dean and Sanjay Ghema](https://exascale.info/assets/pdf/cuccu2019aamas.pdf)wat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113. [https://doi.org/10.1145/](https://doi.org/10.1145/1327452.1327492)  [1327452.1327492](https://doi.org/10.1145/1327452.1327492)   
[9] [Ouassim Ait Elh](https://doi.org/10.1145/1327452.1327492)ara, Konstantinos Varelas, Duc Manh Nguyen, Tea Tušar, Dimo Brockhoff, Nikolaus Hansen, and Anne Auger. 2019. COCO: The Large Scale Black-Box Optimization Benchmarking (bbob-largescale) Test Suite. *arXiv*  *preprint arXiv:1903.06396* (2019).

[10] Stanislav Fort and Stanislaw Jastrzebski. 2019. Large scale structure of neural network loss landscapes. *Advances in Neural Information Processing Systems* 32 (2019), 6709–6717.

[11] Hervé Fournier and Olivier Teytaud. 2011. Lower bounds for comparison based evolution strategies using vc-dimension and sign patterns. *Algorithmica* 59, 3 (2011), 387–408.

[12] David Ha and Jürgen Schmidhuber. 2018. *Recurrent world models facilitate policy*  *evolution*. Technical Report 1809.01999. arXiv.org.

[13] Nikolaus Hansen. 1996. Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation. In *IEEE International Conference on Evolutionary Computation, 1996*. 312–317.

[14] Nikolaus Hansen. 2019. A global surrogate assisted CMA-ES. In *Proceedings of the*  *Genetic and Evolutionary Computation Conference*. ACM, Prague Czech Republic, 664–672. <https://doi.org/10.1145/3321707.3321842>  
[15] Nikolaus H[ansen, Dirk V Arnold, and Anne Auger.](https://doi.org/10.1145/3321707.3321842) 2015. Evolution strategies. In *Springer handbook of computational intelligence*. Springer, 871–898.

[16] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík.

2010. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*. 1689–1696.

[17] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. 2009. *Real-*

*Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Defini-tions*. Technical Report RR-6869. INRIA.

[18] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195.

[19] Verena Heidrich-Meisner and Christian Igel. 2009. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms* 64, 4 (2009), 152–168. [20] Christian Igel. 2003. Neuroevolution for Reinforcement Learning using Evolution Strategies. In *The 2003 Congress on Evolutionary Computation (CEC’03)*, Vol. 4. IEEE, 2588–2595.

[21] Jens Jägersküpper. 2006. How the (1+1)-ES using isotropic mutations minimizes positive definite quadratic forms. *Theoretical Computer Science* 361, 1 (2006), 38–56.

[22] Ilya Loshchilov. 2014. A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization. In *Genetic and Evolutionary Computation Conference*. ACM, 397–404.

[23] Ilya Loshchilov, Tobias Glasmachers, and Hans-Georg Beyer. 2018. Large Scale Black-box Optimization by Limited-Memory Matrix Adaptation. *IEEE Transac-tions on Evolutionary Computation* 99 (2018).

[24] Mohammad Nabi Omidvar, Xiaodong Li, Yi Mei, and Xin Yao. 2013. Coopera-tive co-evolution with differential grouping for large scale optimization. *IEEE Transactions on Evolutionary Computation* 18, 3 (2013), 378–393.

[25] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. 2017.

*Parameter space noise for exploration*. Technical Report 1706.01905. arXiv.org. [26] Ingo Rechenberg. 1973. *Evolutionsstrategie: Optimierung technischer Systeme nach*  *Prinzipien der biologischen Evolution*. Frommann-Holzboog.

349