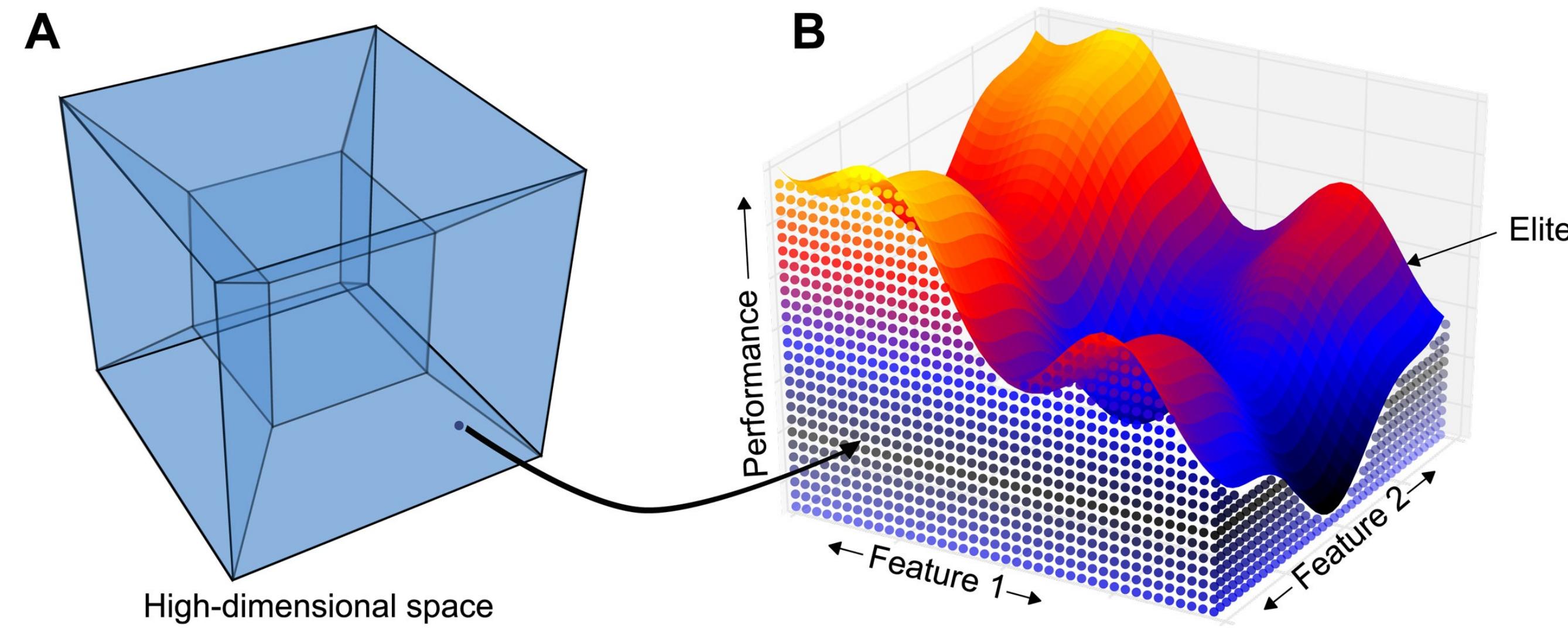
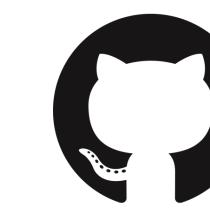


# Quality diversity

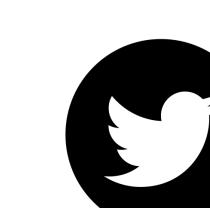
From Novelty Search to the MAP-Elites algorithm



**Jean-Baptiste Mouret – Inria**



@jbmouret

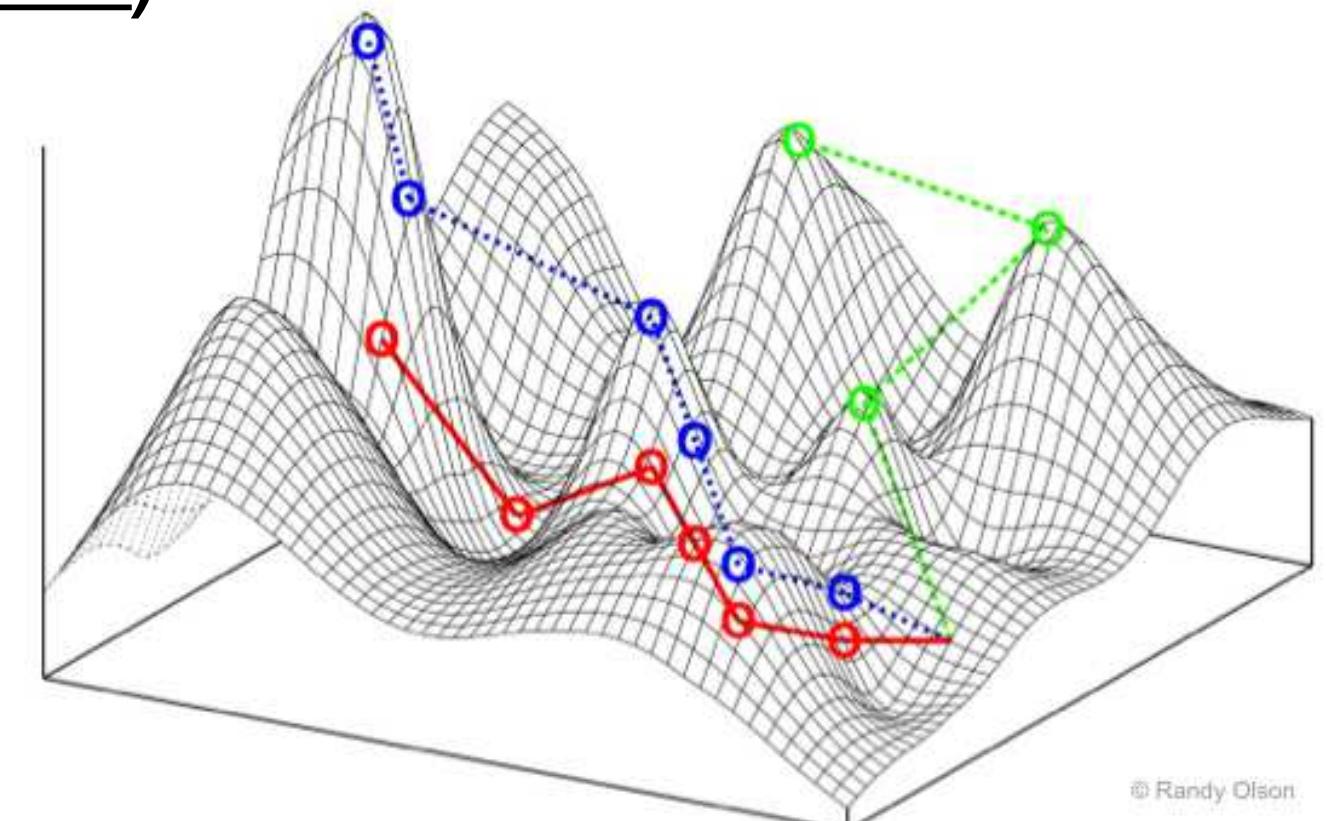


@jb\_mouret

# The problem with artificial evolution

Where is the creativity?

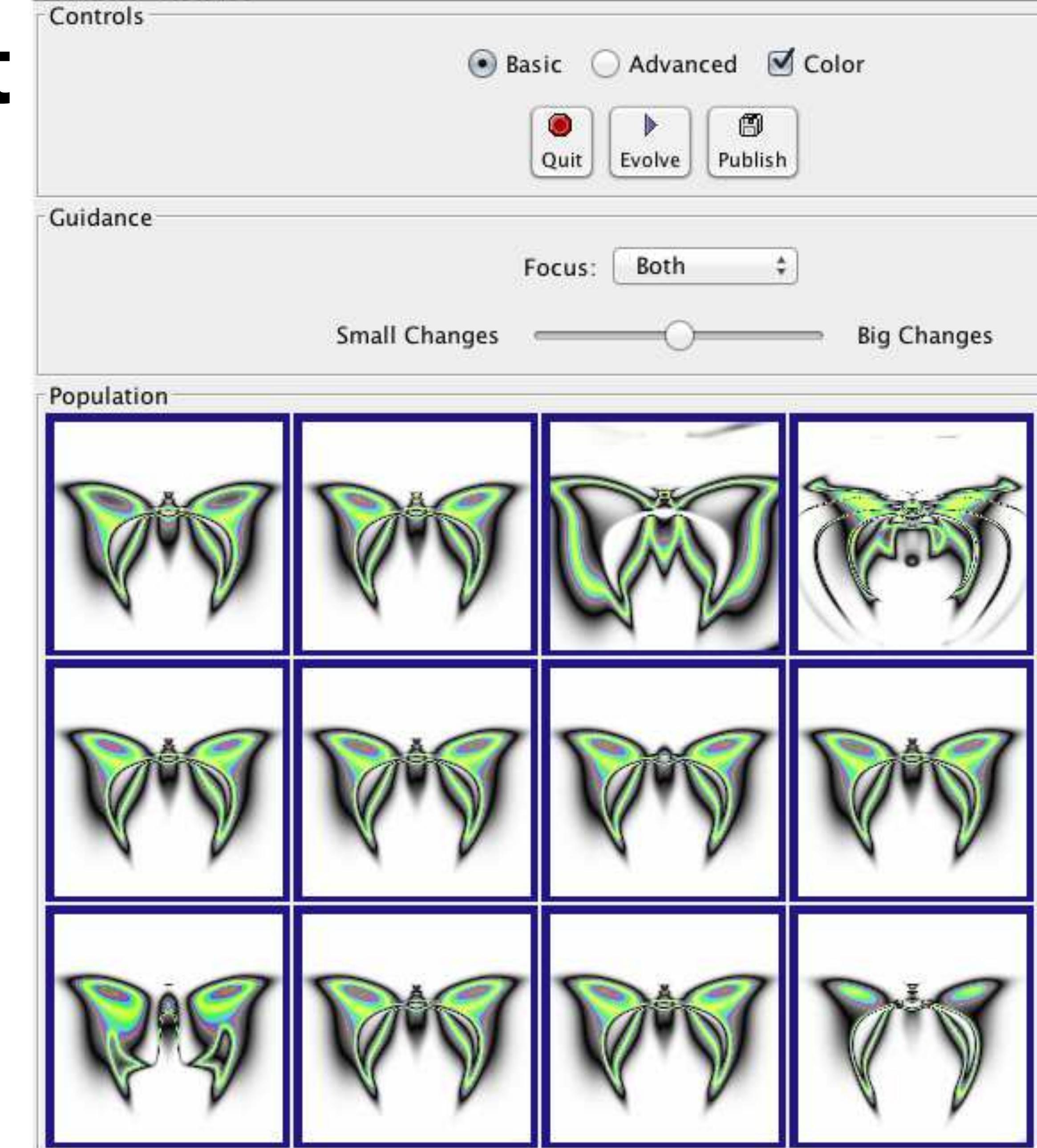
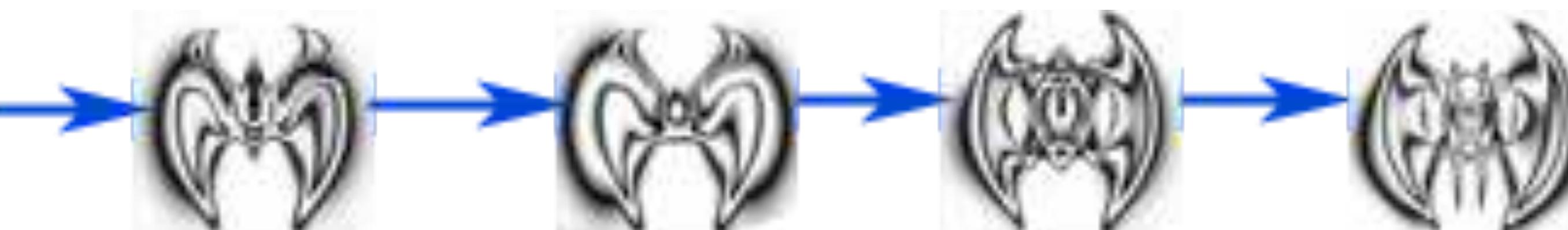
- ES (e.g., CMA-ES) are good (global) black-box optimizers, inc. for RL problems
  - We can evolve the weights of deep neural networks for RL (OpenAI-ES, etc.)
  - MOEA (e.g., NSGA-II) are good multi-objective (black-box) optimizers, inc. for RL problems
  - We can evolve the structure of neural networks (e.g., NEAT & HyperNEAT)
- **but...**
- a lot of “tuning” and “fitness shaping” for the “success stories”
  - (yes, this is not highlighted in the videos / papers)
- **Where is open-ended, creative evolution?**
- **What is missing? how to do better?**



# The Picbreeder Experiment

## Interactive evolution

- Collaborative evolution of images (online)
- inspired by Dawkins' Biomorph
- No goal
- Encoding of images using CPPNs (see neuroevolution part)



K. O. Stanley, J. Lehman (2015) - Why Greatness Cannot Be Planned, - Springer

Secretan J, Beato N, D'Ambrosio DB, Rodriguez A, Campbell A, Folsom-Kovarik JT, Stanley KO. Picbreeder: A case study in collaborative evolutionary exploration of design space. Evolutionary computation. 2011 Sep;19(3):373-403.

# Results from Picbreeder (interactive evolution)



# Interactive evolution (no goal):



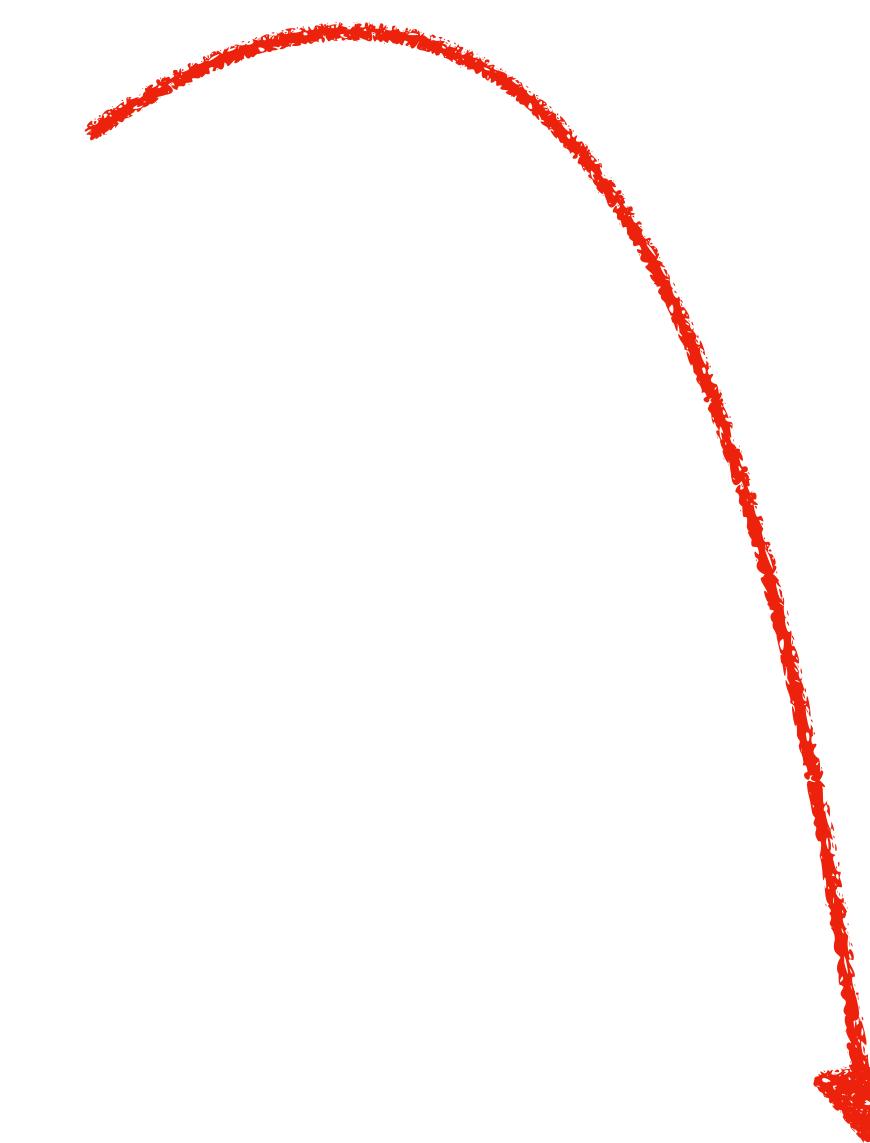
gen 12

gen 20

gen 36

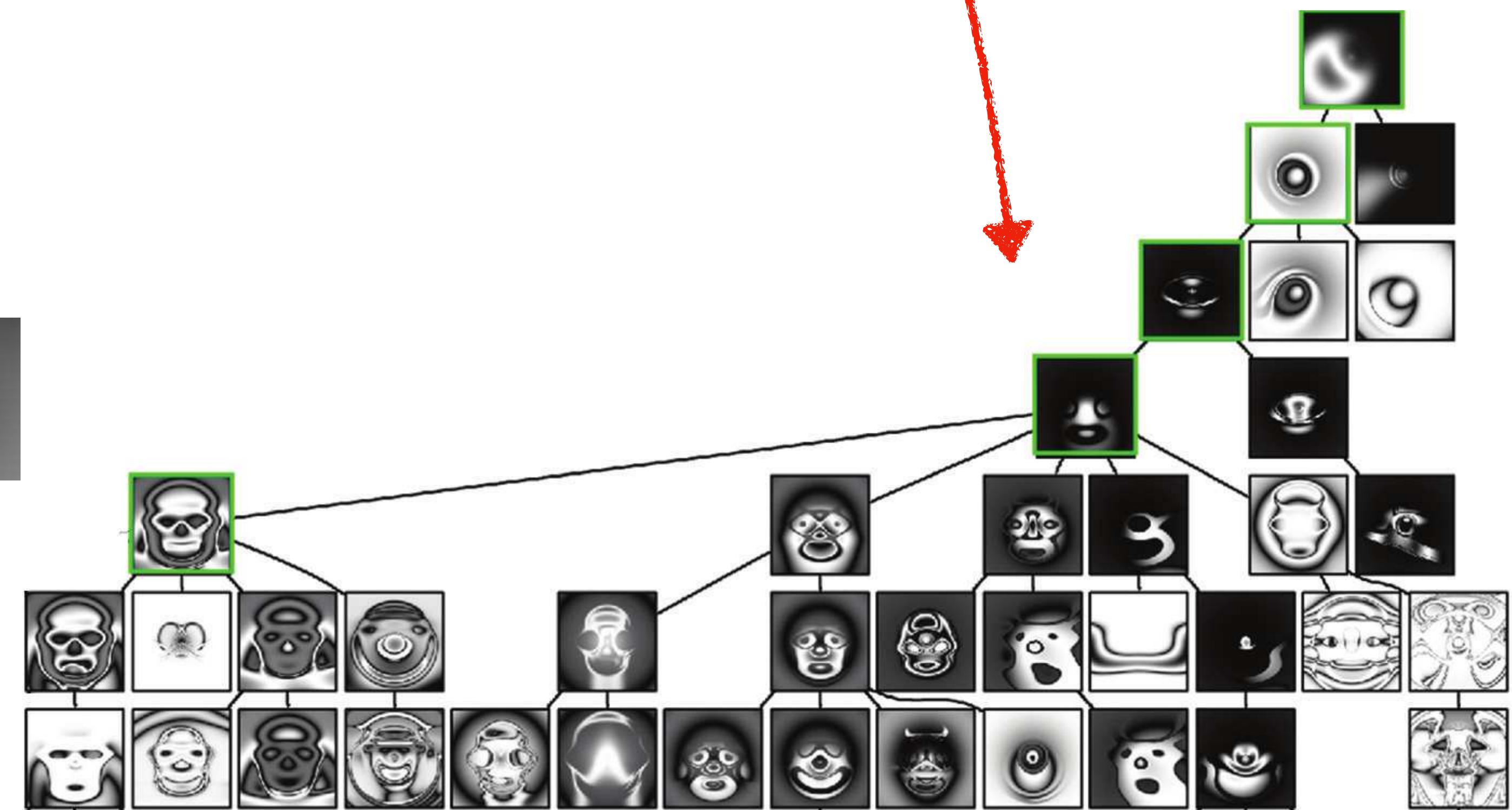
gen 49

gen 74



# Objective-based evolution

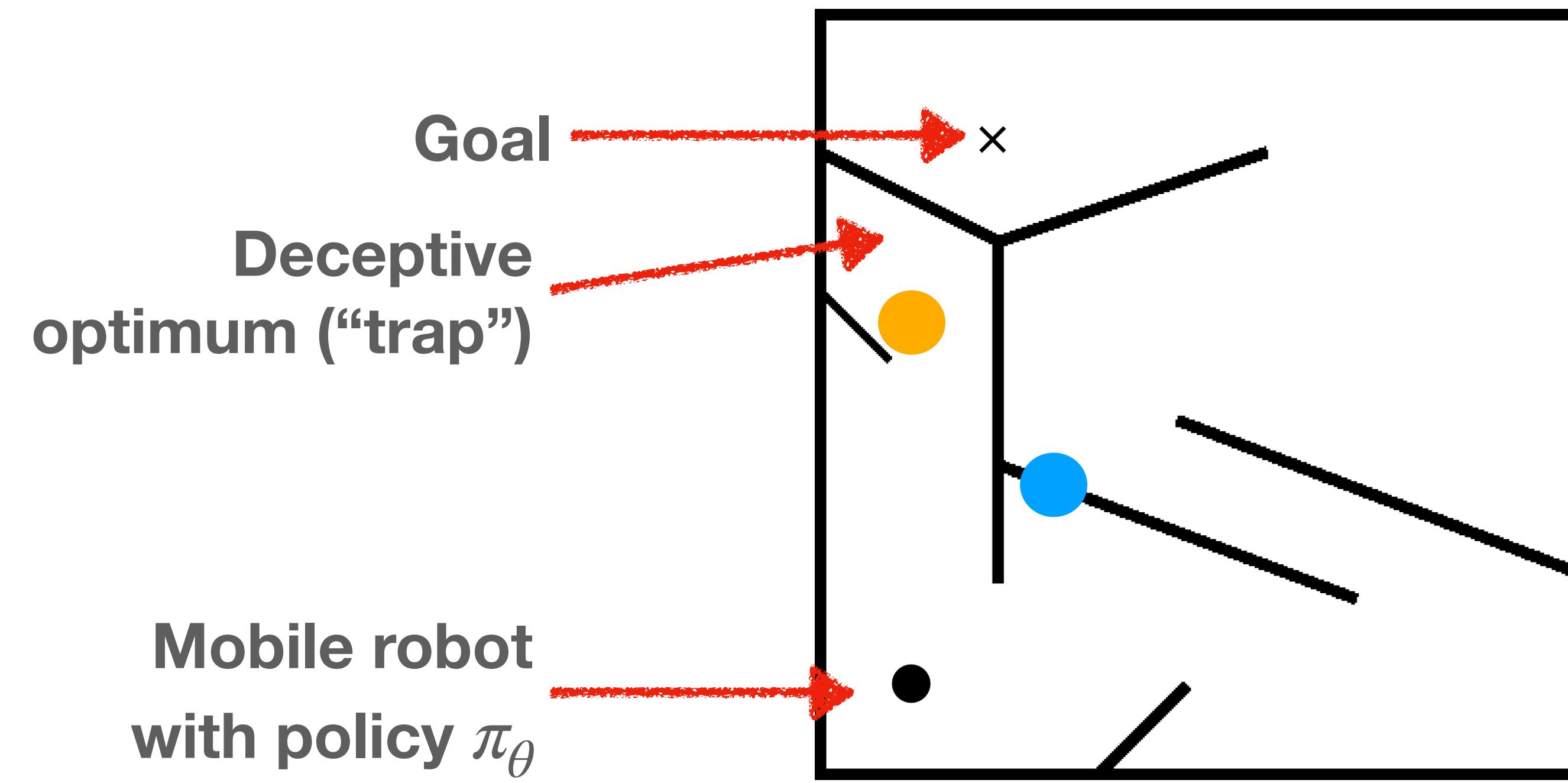
Skull	Run 1	Run 3	Run 5	Run 7	Run 9	Run 11
23f, 57c 74 gen	20f, 24c failed	20f, 29c failed	19f, 24c failed	22f, 28c failed	21f, 28c failed	16f, 22c failed



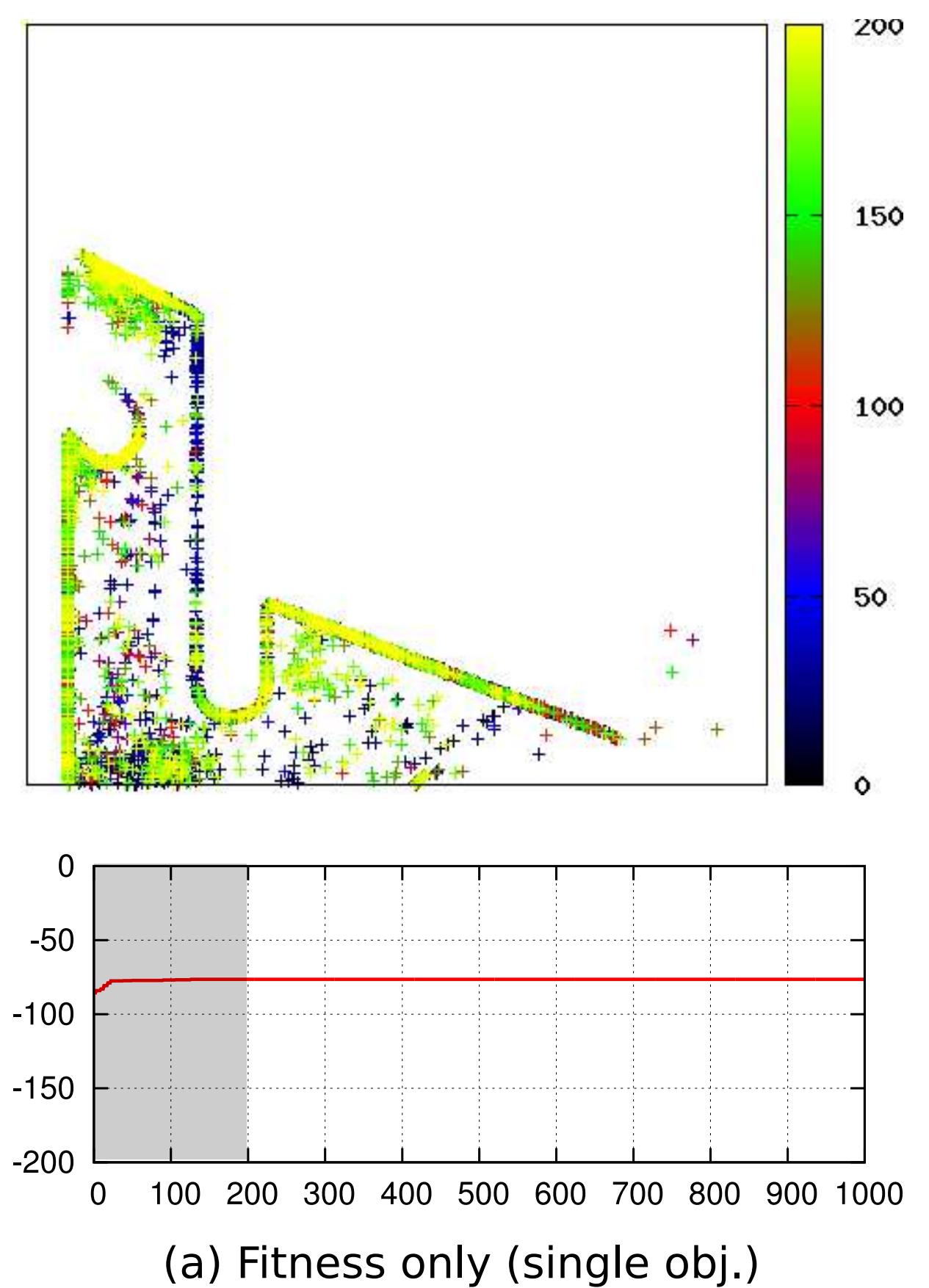
Woolley, B. G., & Stanley, K. O. (2011). On the deleterious effects of a priori objectives on evolution and representation. In Proceedings of the 13th annual conference on Genetic and evolutionary computation (pp. 957-964). ACM.

# Deceptive search spaces

- The “stepping stones” to get to the solution are NOT like the final solution  
... **but this is the “basic” heuristic of most search algorithms:** solutions that are closer (better reward) to the goal should be favored!
- Interpretation: the search space “deceptive” (attractive local minima)  
→ We need more exploration

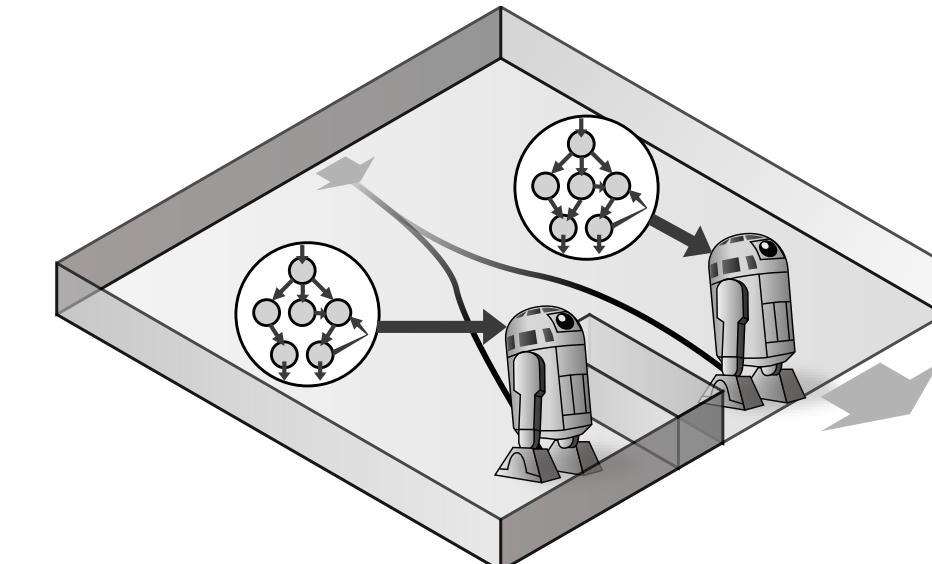


Fitness = distance to the goal at the end of the episode

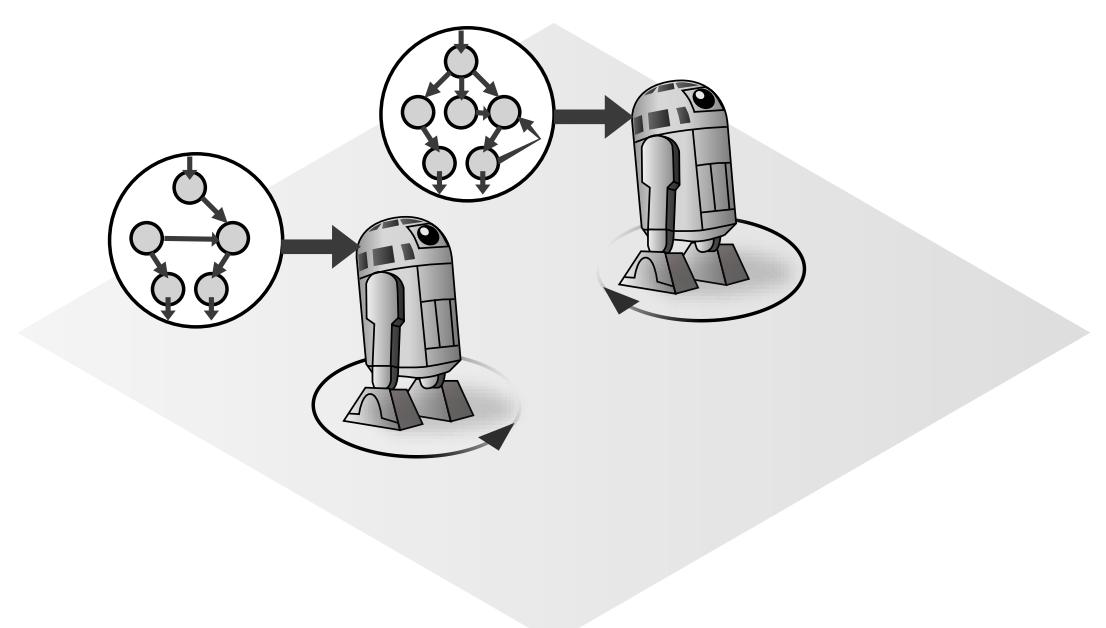


# Novelty Search concept

- Radical idea: what if we ignore the fitness function?  
**... and search for novel “things”**
- In RL/Evo, “things” = behavior  
→ search for novel behaviors



Close genotype / different behavior



Different genotype / same behavior

## Novelty search:

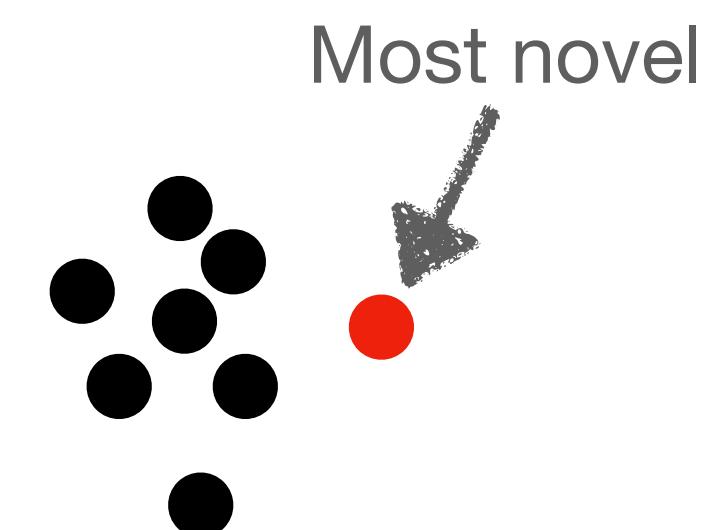
- characterize behavior (e.g., final position, not list) with a vector
- replace fitness by novelty
- ... computed by the behavioral distance to the archive & population

$$\rho(x) = \frac{1}{k} \sum_{j=0}^k d_b(x, \mu_j)$$

Novelty of  $x$

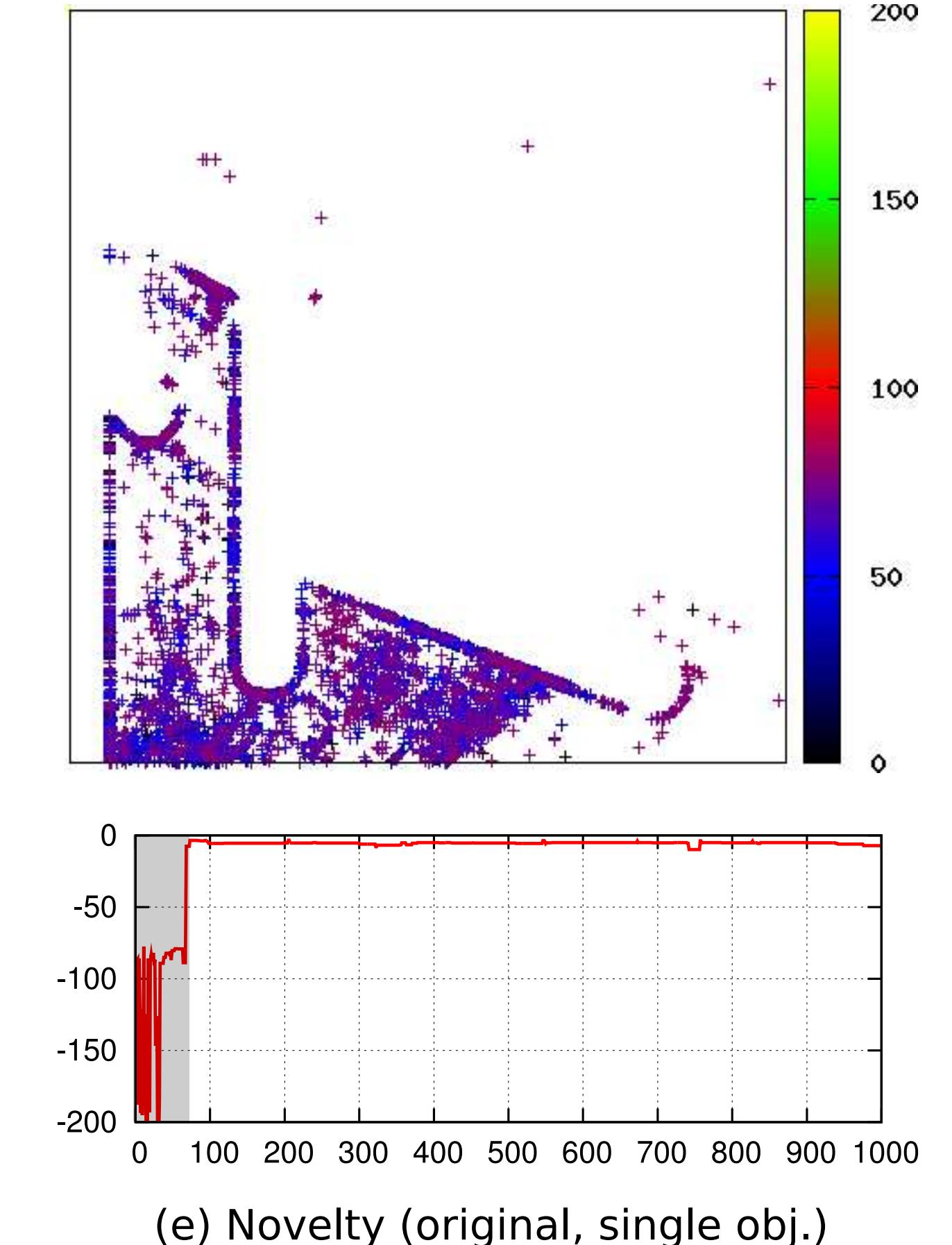
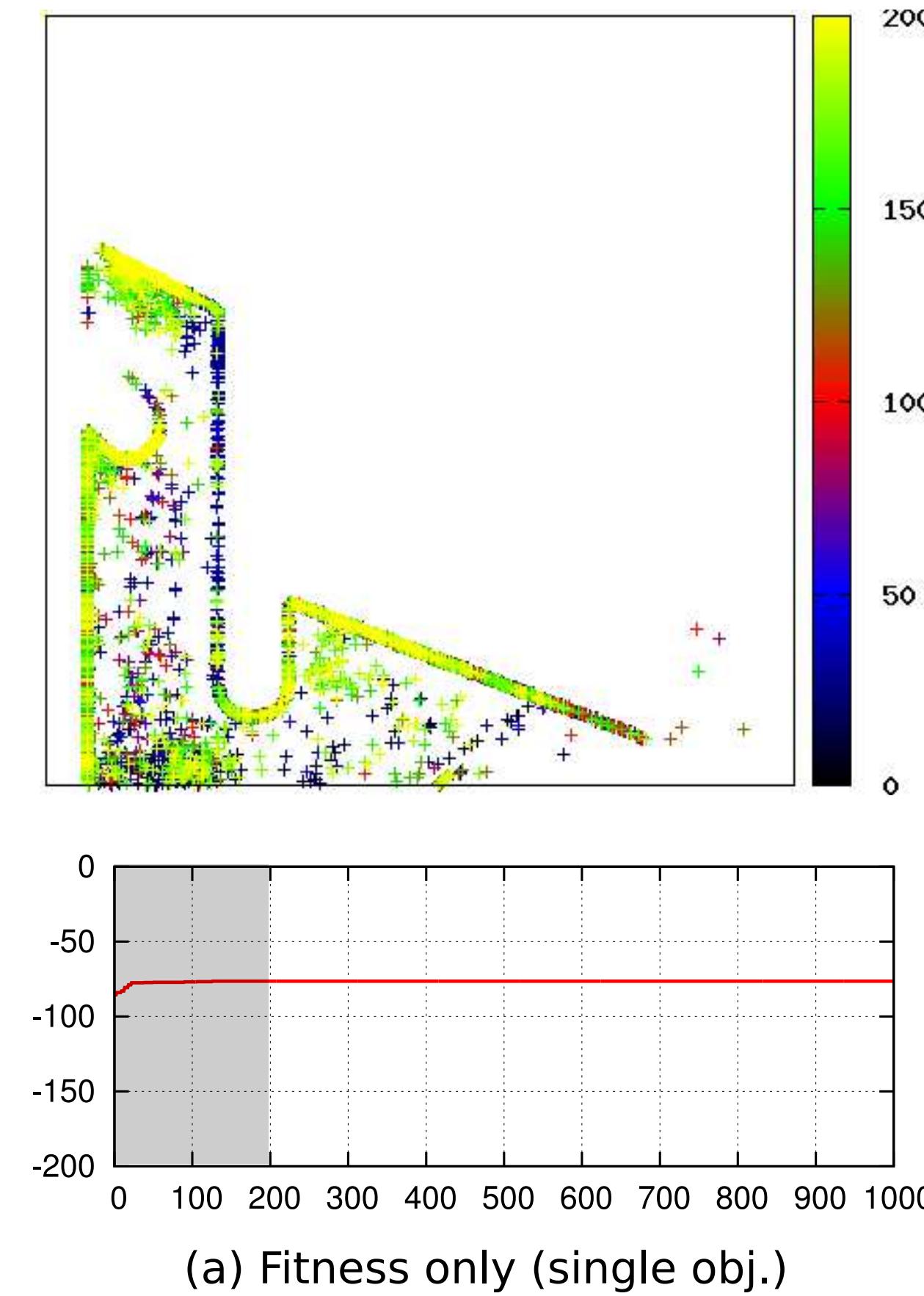
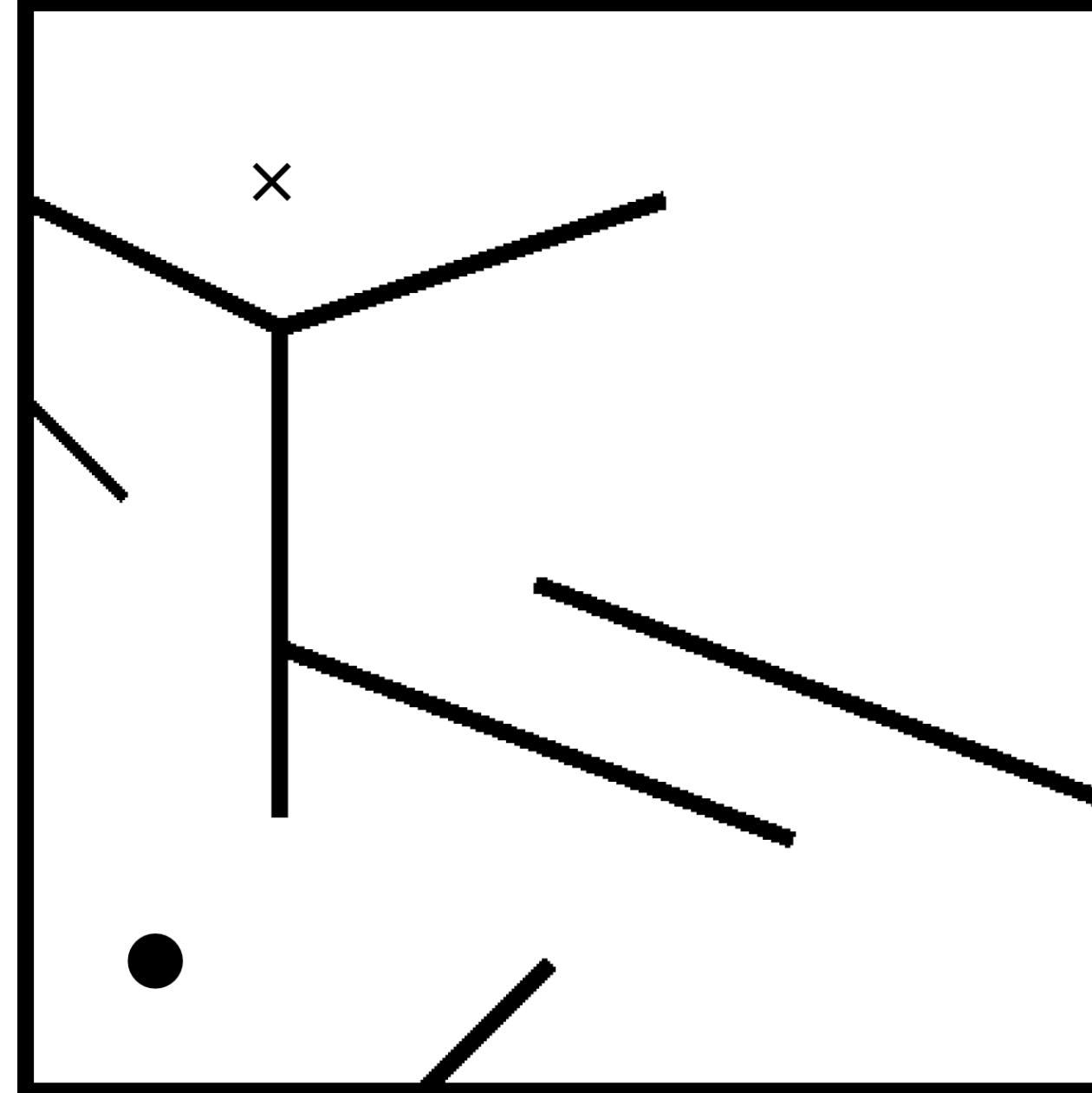
Sum over the  $k$  nearest neighbors from archive and population

behavioral distance between  $x$  and  $\mu_j$



# Novelty search

## deceptive maze

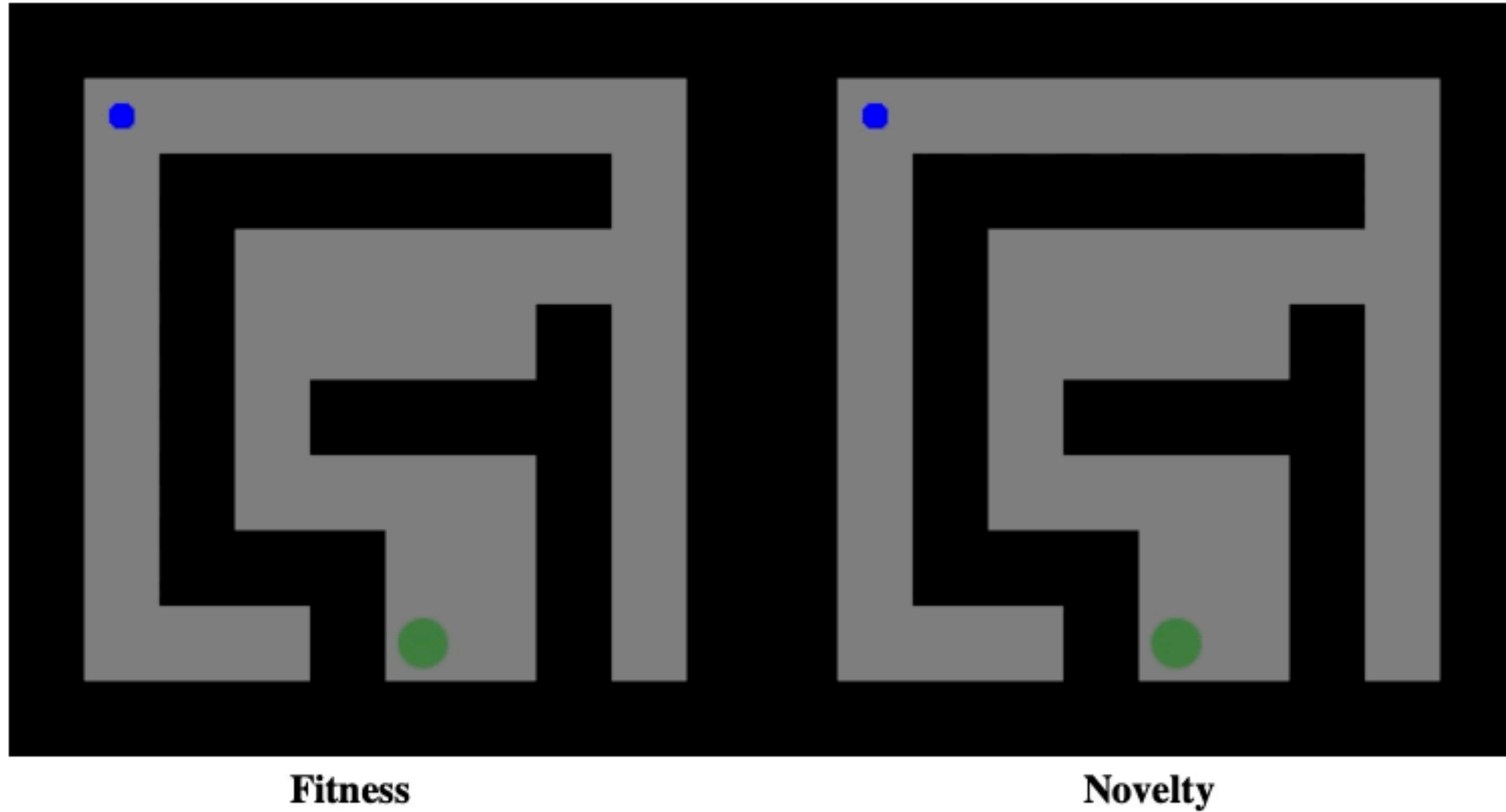


**Behavior space** = (x,y) position at the end of the evaluation period

**Genotype** = neural network (direct encoding, e.g. NEAT)

# Novelty Search: demo

Another maze



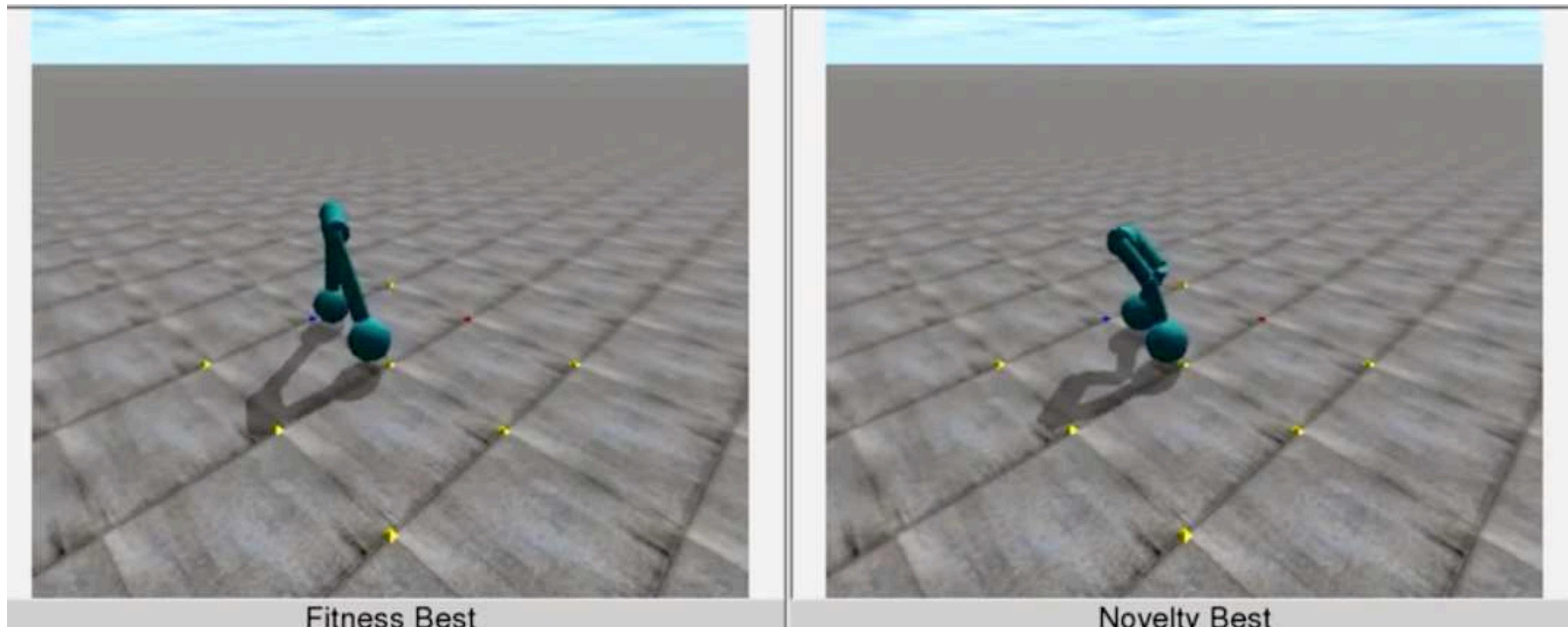
**Fitness**

**Novelty**

# Novelty search

## more complex example

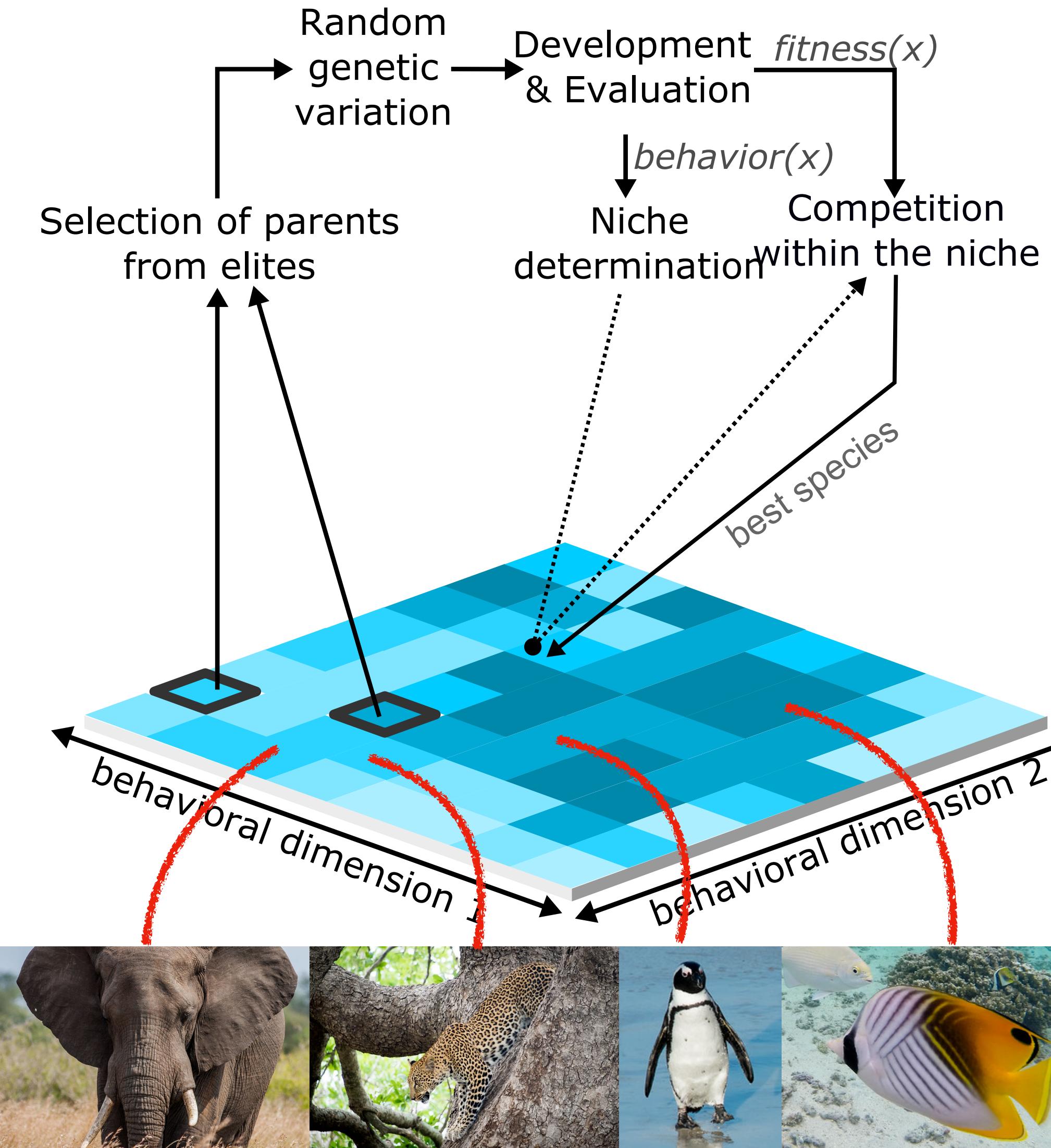
- Once all the “easy behaviors” exist in the archive (e.g. falling)
- ... the agents have to be creative! (e.g., walking)



- behavior = position of the center of mass at the end of the episode

# MAP-Elites

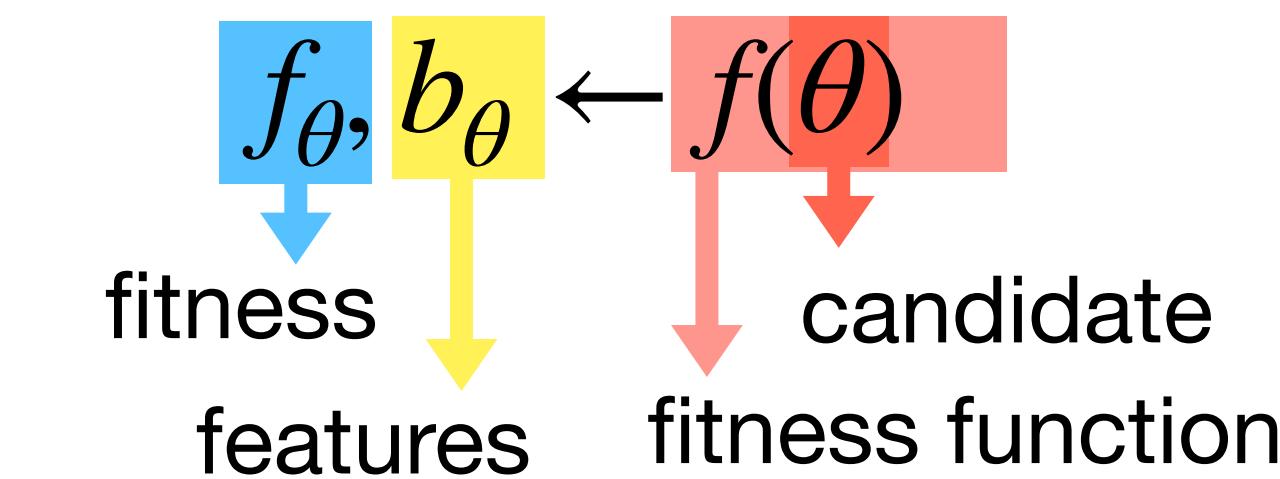
## Multi-dimensional Archive of Phenotypic Elites



**Objective:** Find many good ways of solving a problem

**Assumption:** the fitness/reward function returns:

- a fitness/reward
- a behavioral vector (how is it solved)



**Underlying ideas:**

- closer to natural evolution, emphasize diversity
- more creative process (not pure RL/optimization)
- less exhaustive than Novelty Search

# Example: planar arm

see notebook

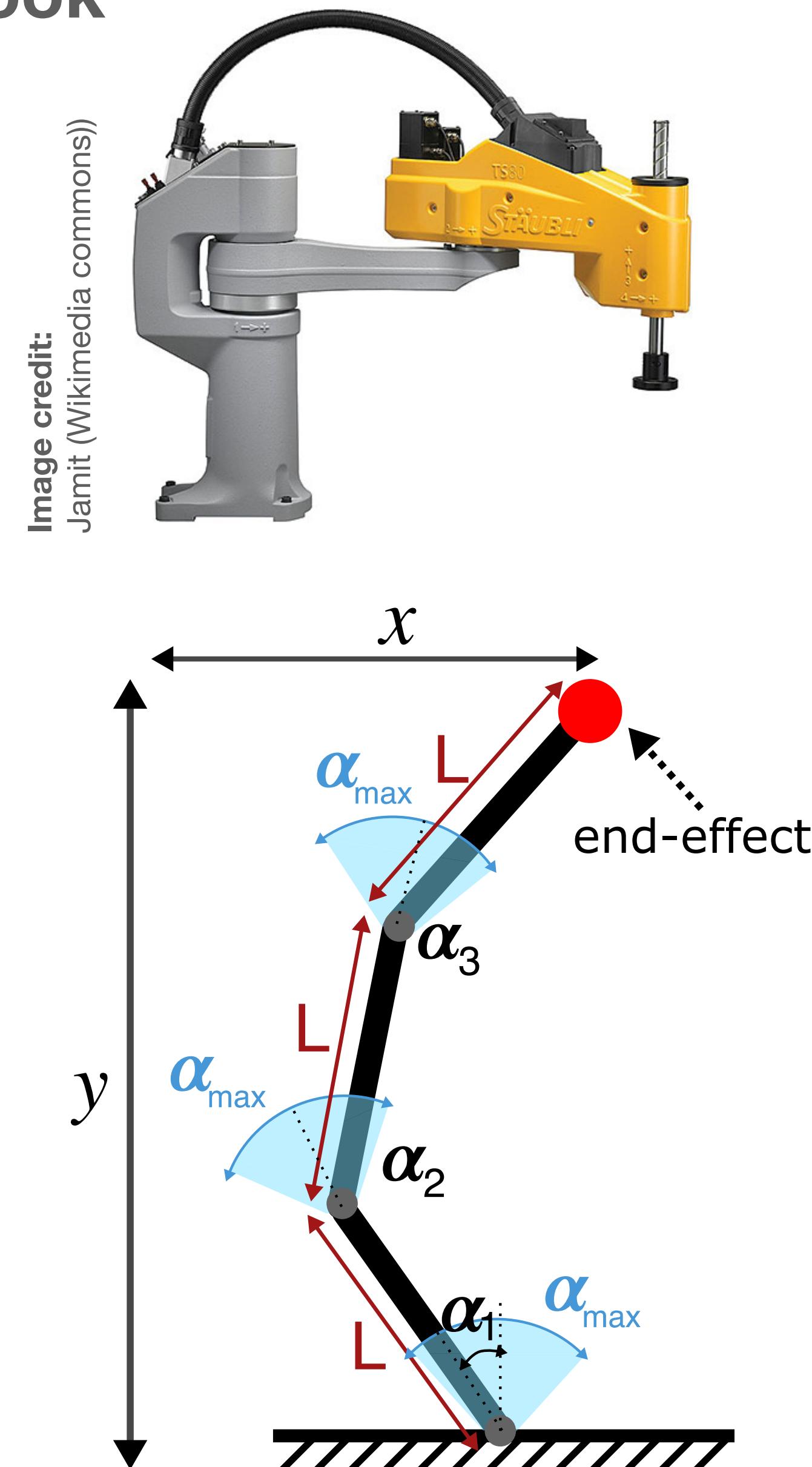
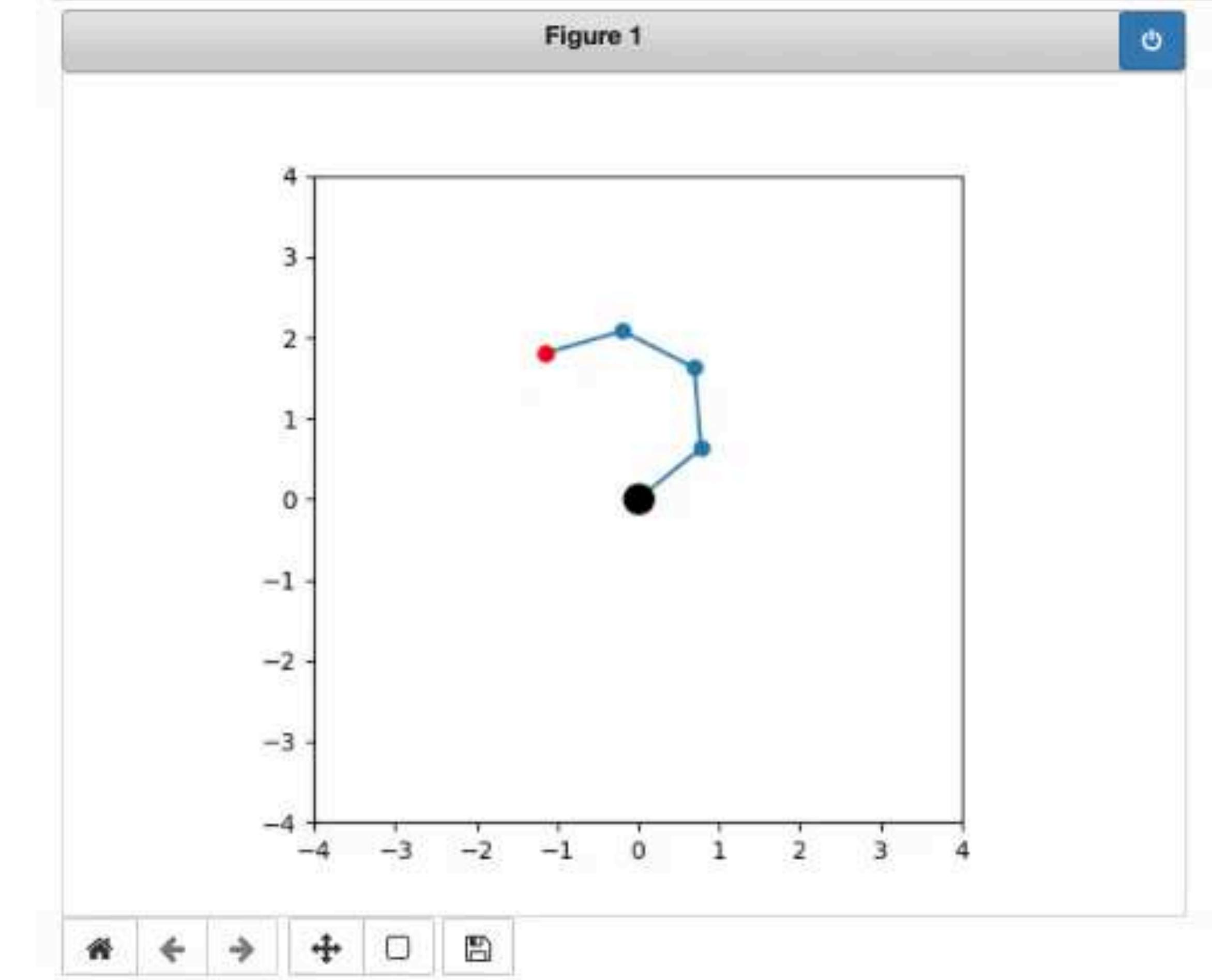


Image credit:  
Jamit (Wikimedia commons)



joint 0	38.84
joint 1	55.96
joint 2	58.18
joint 3	43.12

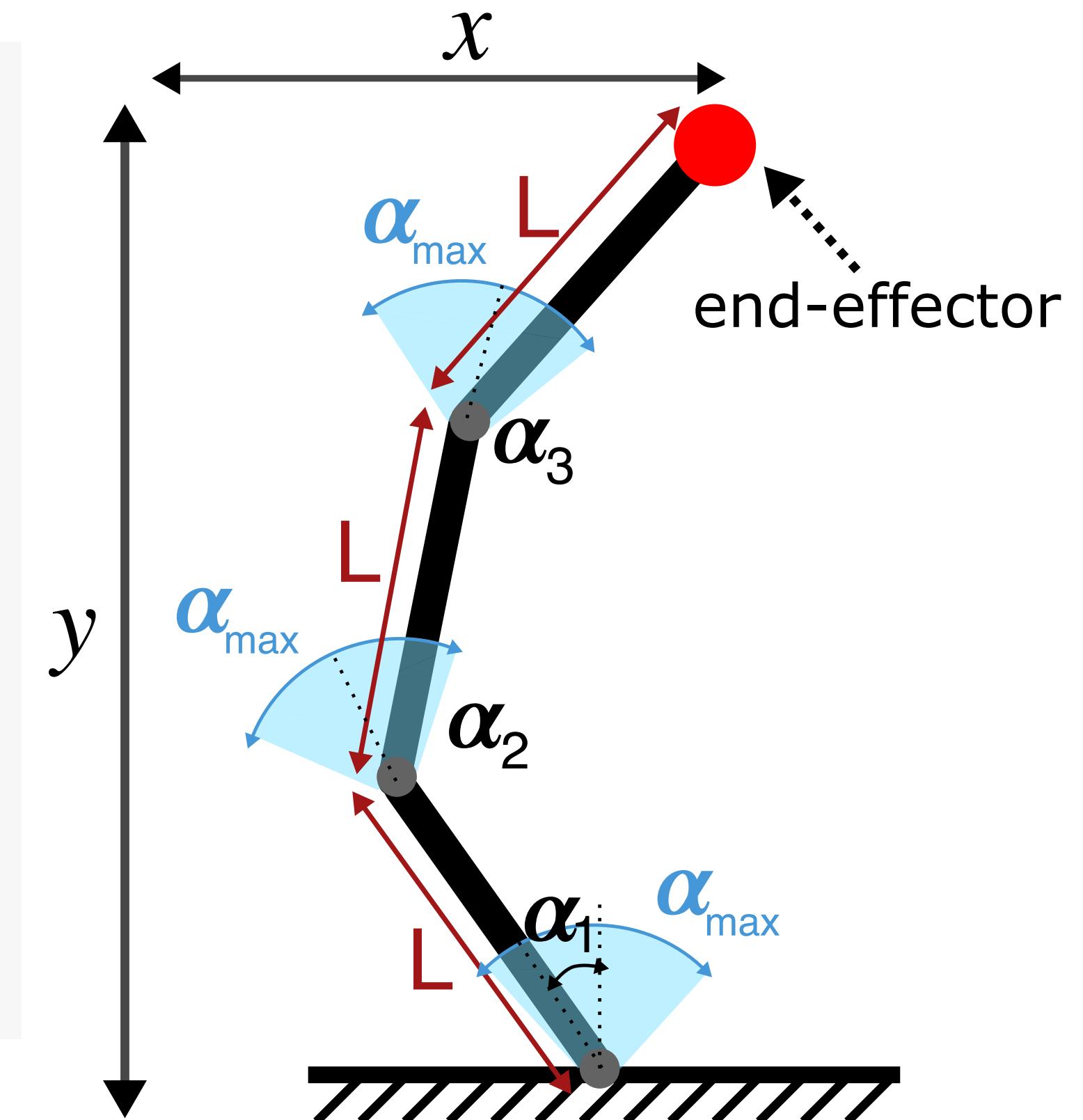
$x=-1.1564570501892633 \quad y=1.8007513064939158 \quad \text{std\_dev}=0.14350350424050143$

- **Search space:**  $[\alpha_1, \dots, \alpha_n]$  ( $n$ -dimensional)
- **Behavior space:**  $(x,y)$  (2-dimensional)

# Example: planar arm

## Fitness function (notebook)

```
def fitness(genotype):  
    # fitness is the standard deviation of joint angles (Smoothness)  
    # (we want to minimize it)  
    fit = 1 - np.std(genotype)  
  
    # now compute the behavior  
    # scale to [0,2pi]  
    g = np.interp(genotype, (0, 1), (0, 2 * math.pi))  
    j = forward_kinematics(g, [1]*len(g))  
    # normalize behavior in [0,1]  
    b = (j[-1,:]) / (2 * len(g)) + 0.5  
    return fit, b# the fitness and the position of the last joint
```



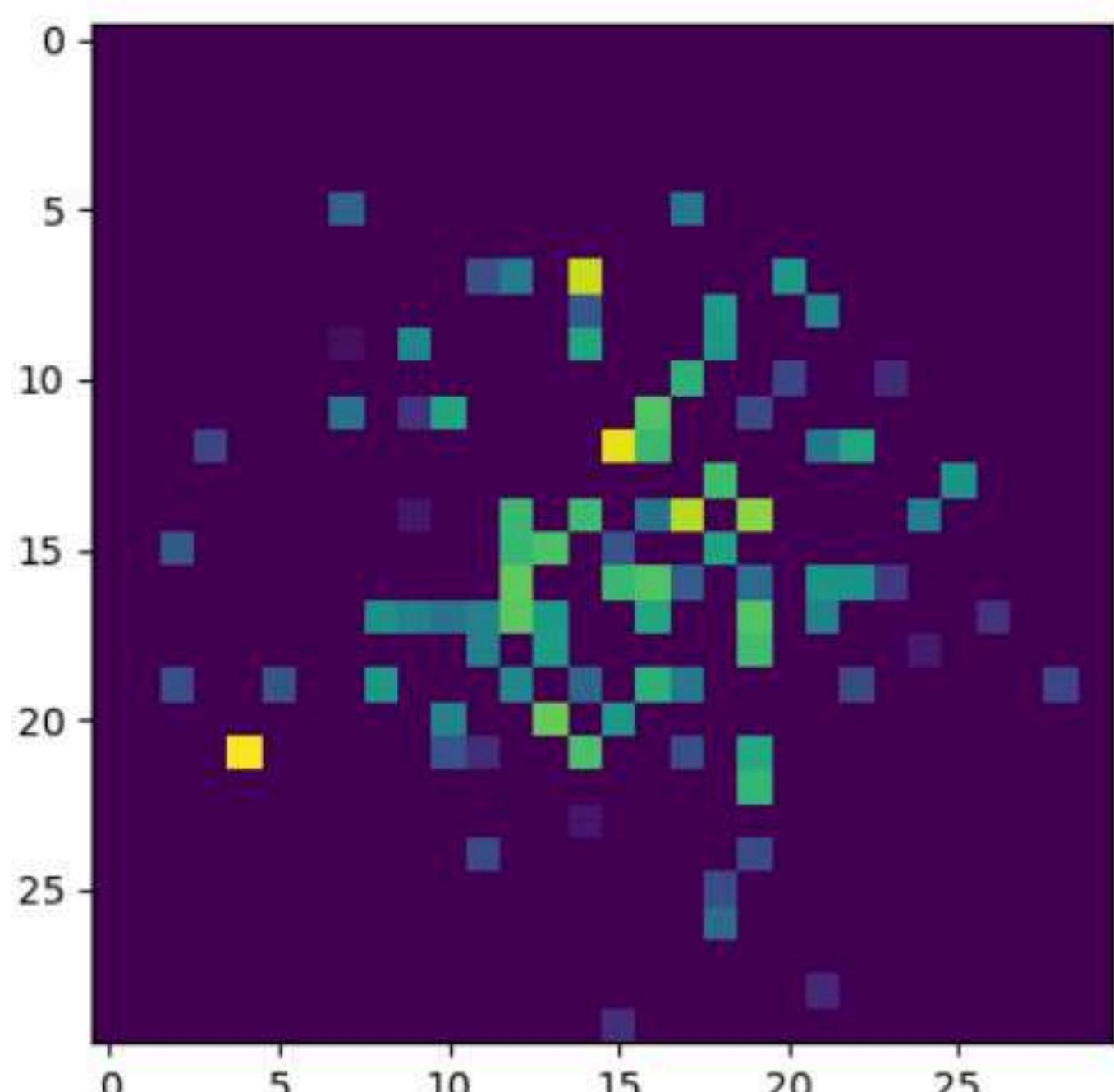
# Example: planar arm

## Archive management

```
# for simplicity, this is 2-Dimensional MAP-Elites
cols = 30
rows = 30
num_random = 100
num_dofs = 4

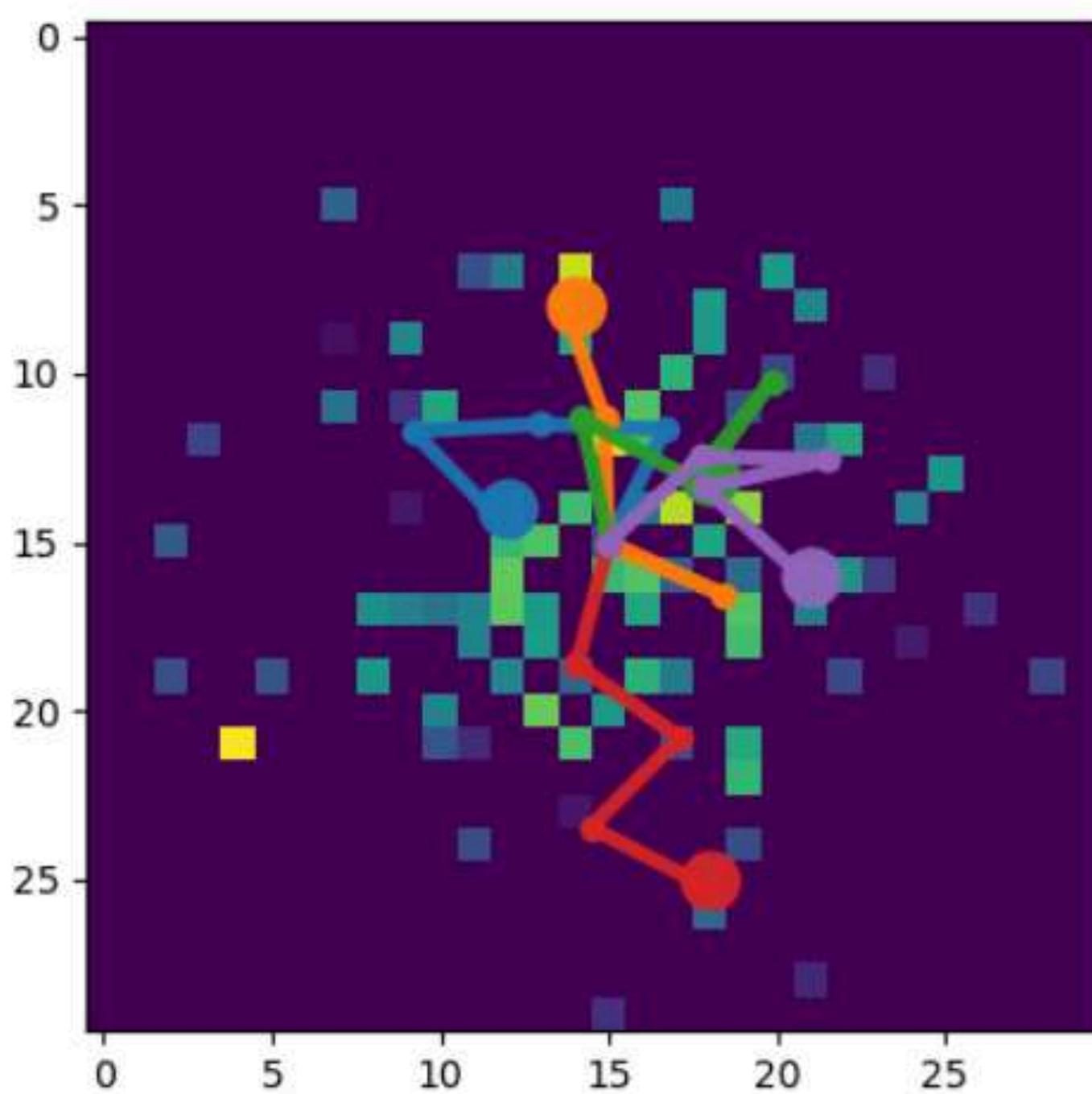
# we should use a dataclass, but this 3.9+
class Species:
    def __init__(self, genotype, behavior, fitness, niche=[]):
        self.genotype = genotype
        self.behavior = behavior
        self.fitness = fitness
        self.niche = niche

def add_to_archive(archive, species):
    n = species.behavior * np.array([rows, cols])
    x,y = min(round(n[0]), rows-1), min(round(n[1]), cols-1)
    if (not (x,y) in archive) or (archive[(x,y)].fitness < species.fitness):
        archive[(x,y)] = species
        species.niche = (x,y)
```



## Archive initialization

```
def init_archive(num_random): # fill the archive with some random solutions
    # create an archive: a dictionnary indexed by coordinates
    archive = {}
    for i in range(0, num_random):
        g = np.random.rand(num_dofs)
        f, b = fitness(g)
        add_to_archive(archive, Species(g, b, f))
    return archive
```

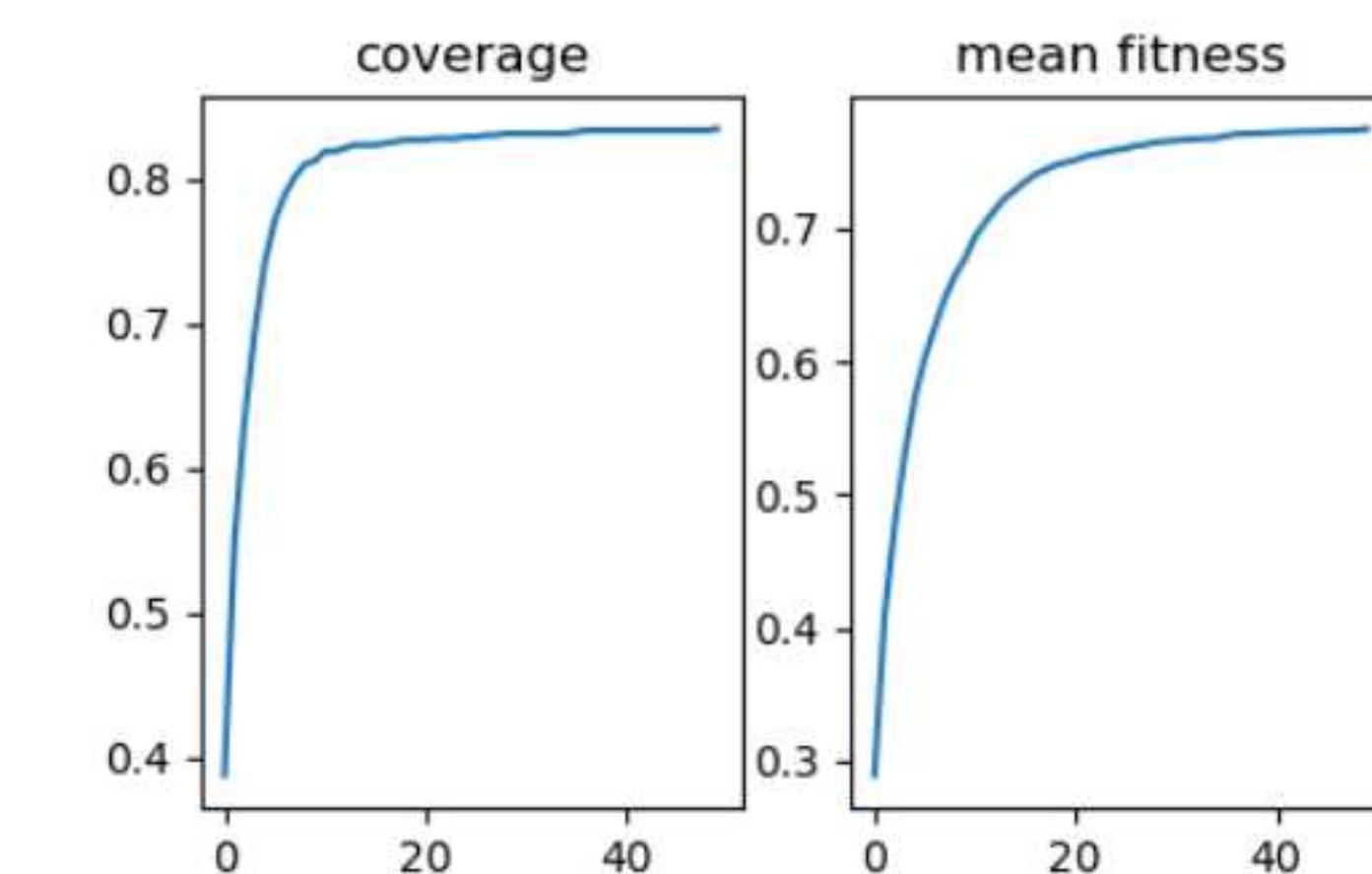
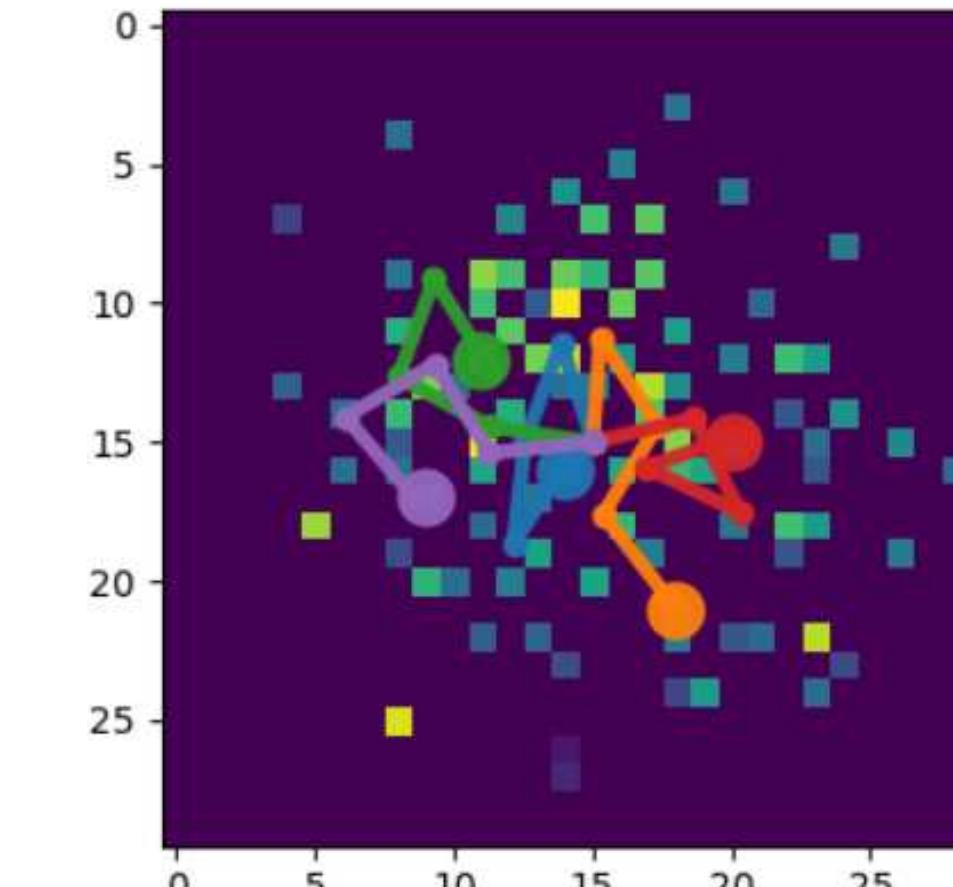
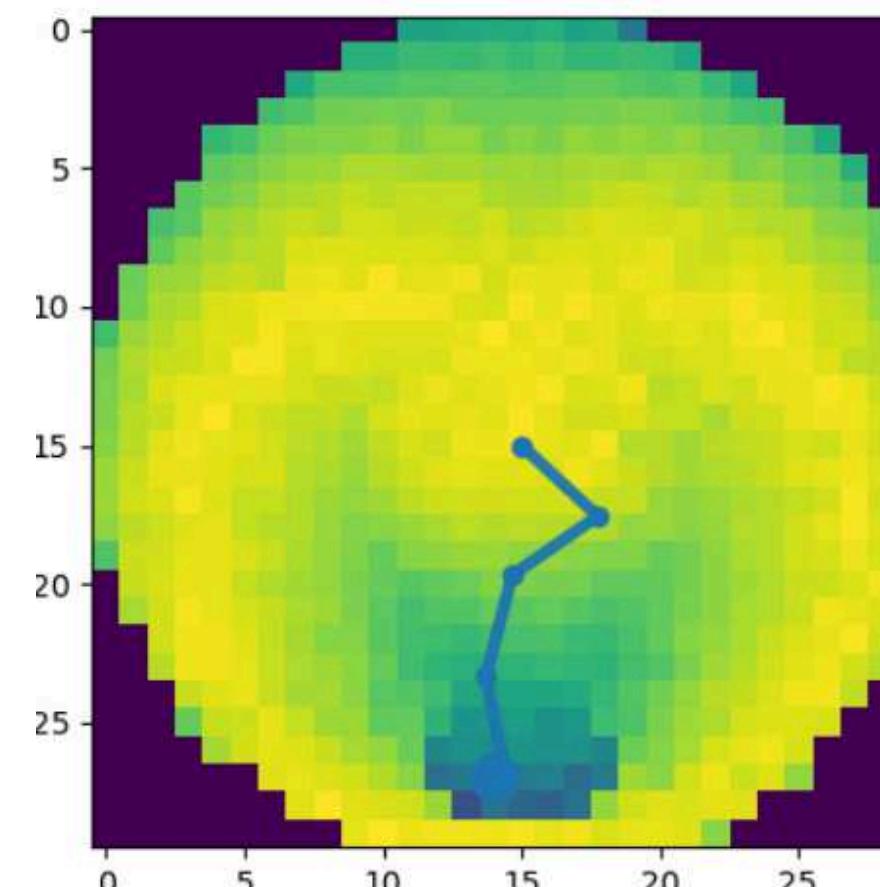
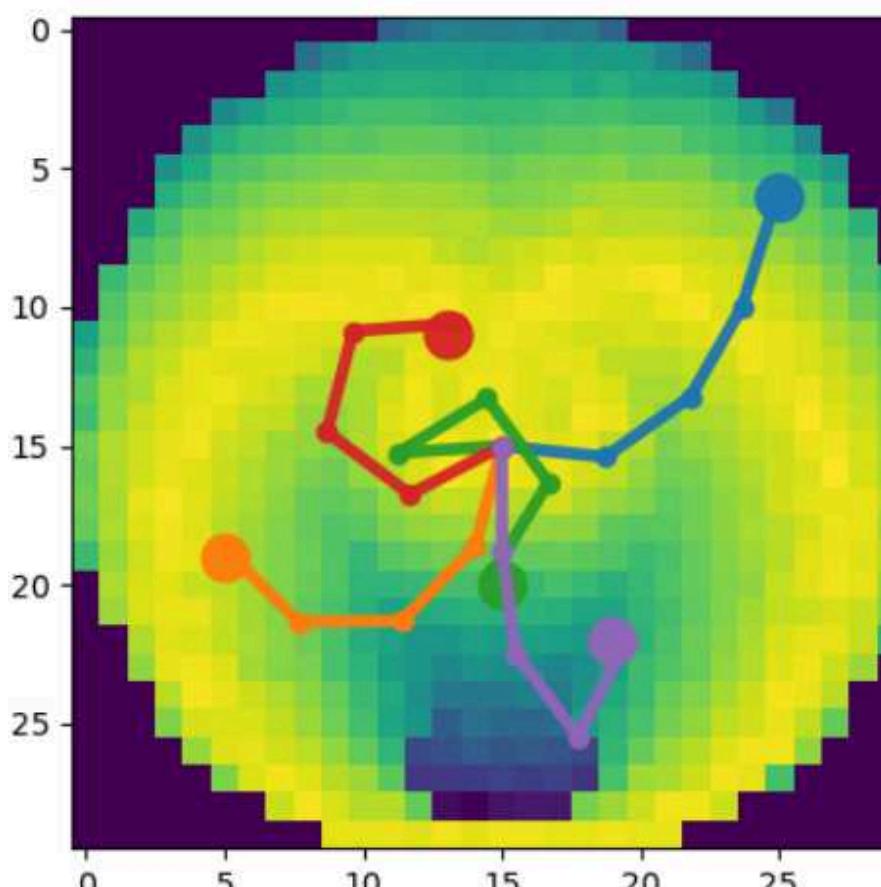
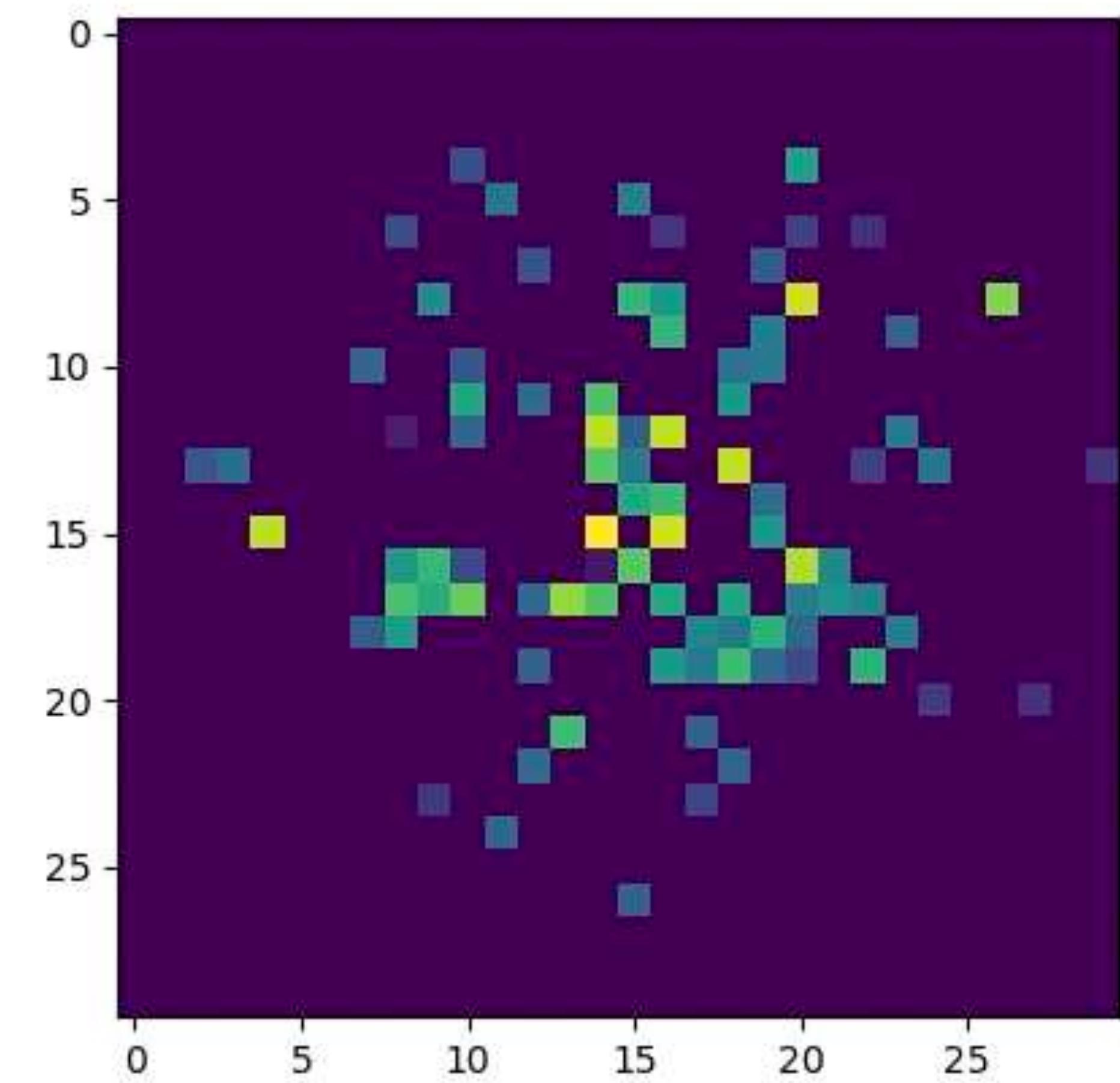


# Example: planar arm

## Main loop

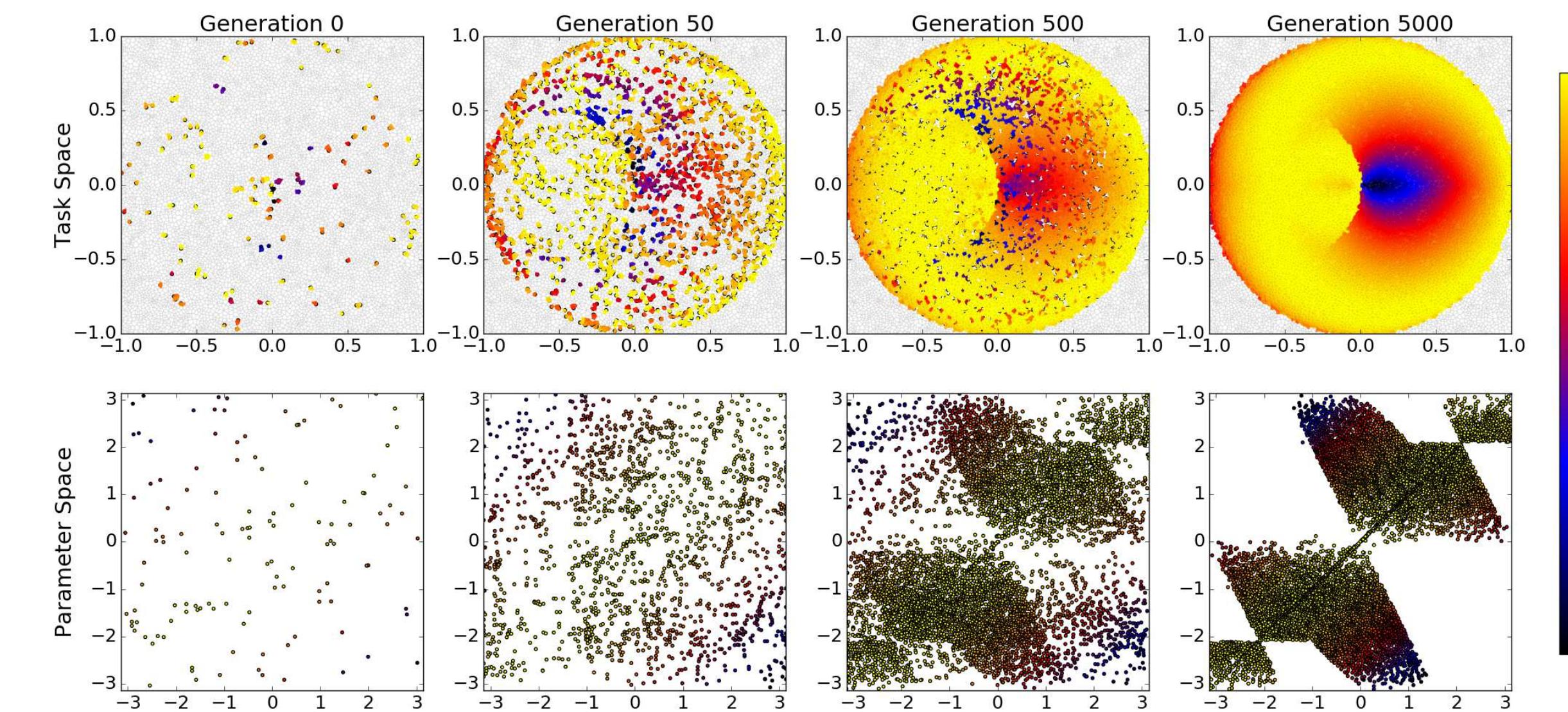
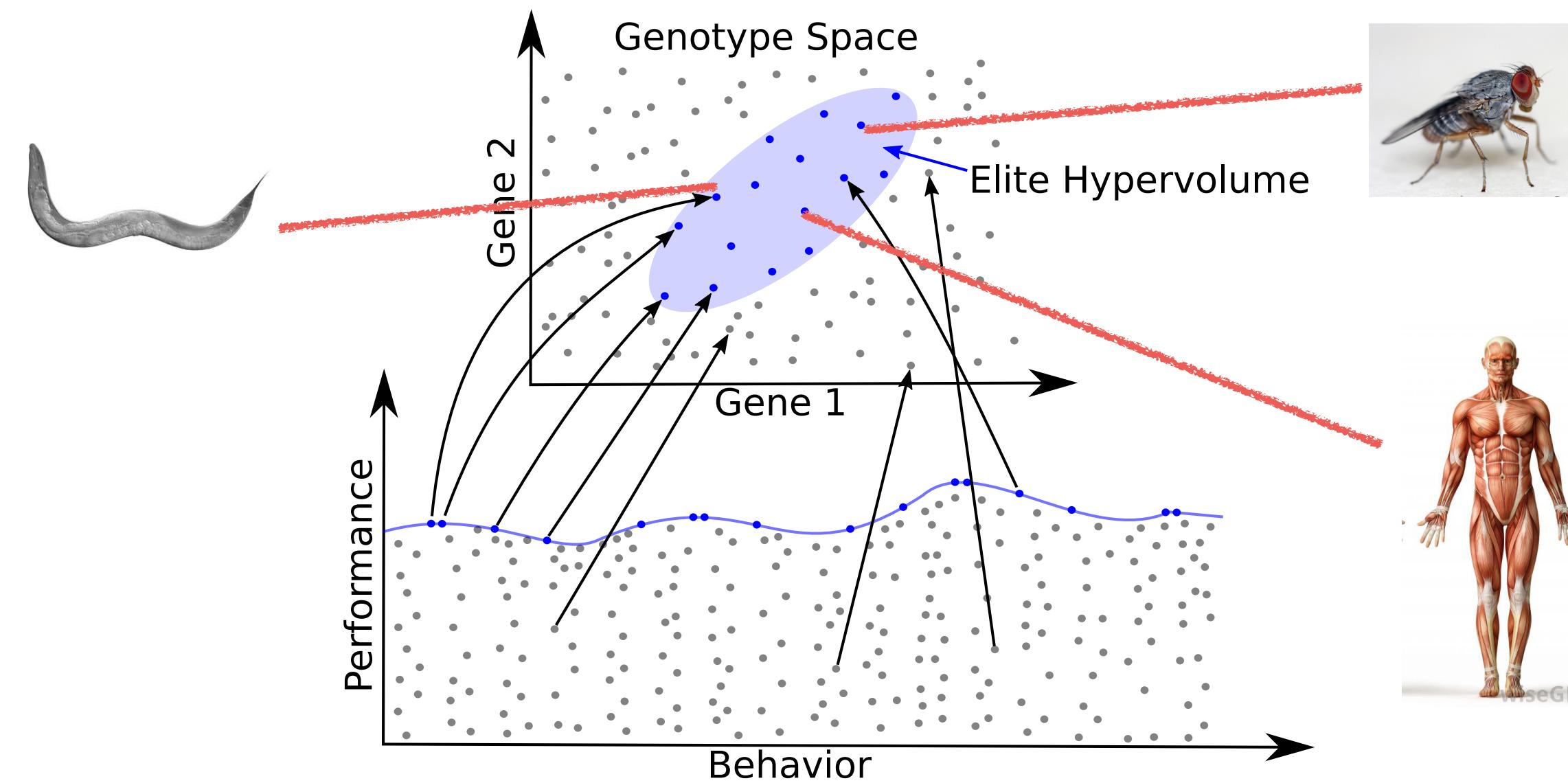
```
num_iterations = 50
batch_size = 500

for j in tqdm(range(0, num_iterations)):
    display_archive(archive)
    for i in range(0, batch_size):
        # pick an existing random point in the archive
        x = random.choice(list(archive.values()))
        # mutate it (we can use more advanced variation techniques: NEAT, etc.)
        g = x.genotype + np.random.normal(0, 0.1, x.genotype.shape[0])
        # compute the fitness
        f, b = fitness(g)
        # add to archive
        add_to_archive(archive, Species(g, b, f))
```

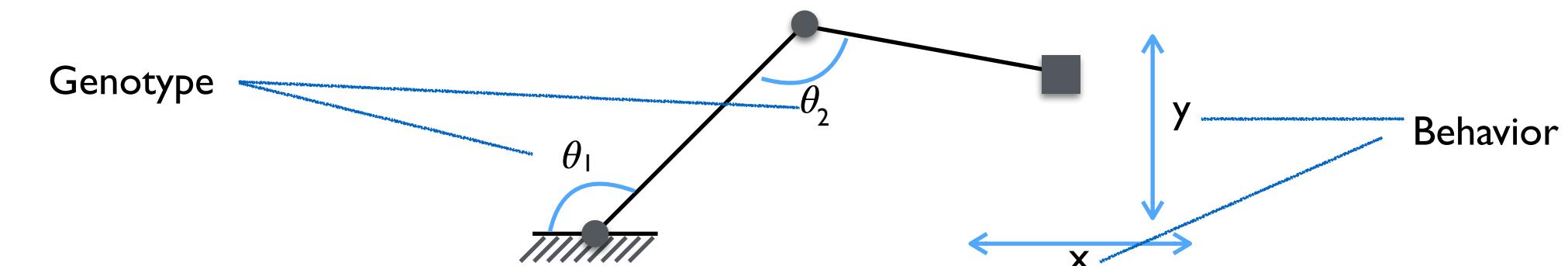


# MAP-Elites: elite hypervolume

Good solutions share common “recipes”



**Human and fruit flies:** 60% of common genes!

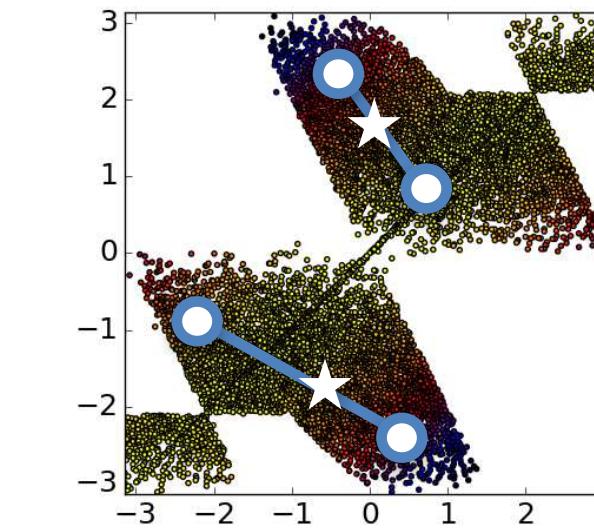


# MAP-Elites: elite hypervolume

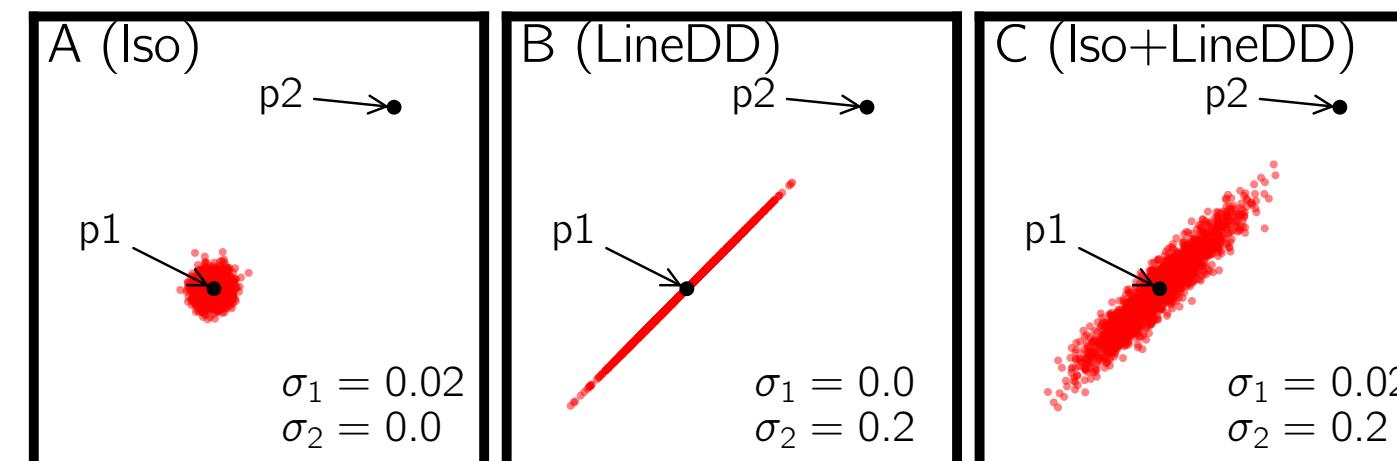
Good solutions share common “recipes”

- What is a good variation operator?  
→ highly likely to generate an individual in the elite hypervolume

★



- if we take two points from a convex volume, any point on the segment is in the volume too



$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \sigma_1 \mathcal{N}(0, \mathbf{I}) + \sigma_2 (\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)}) \mathcal{N}(0, 1)$$

random  
perturbation  
for each dimension

weight a single perturbation  
by the difference between  
the parents

- Directional variation (~ cross-over)
- Adapts the step size

# MAP-Elites: elite hypervolume

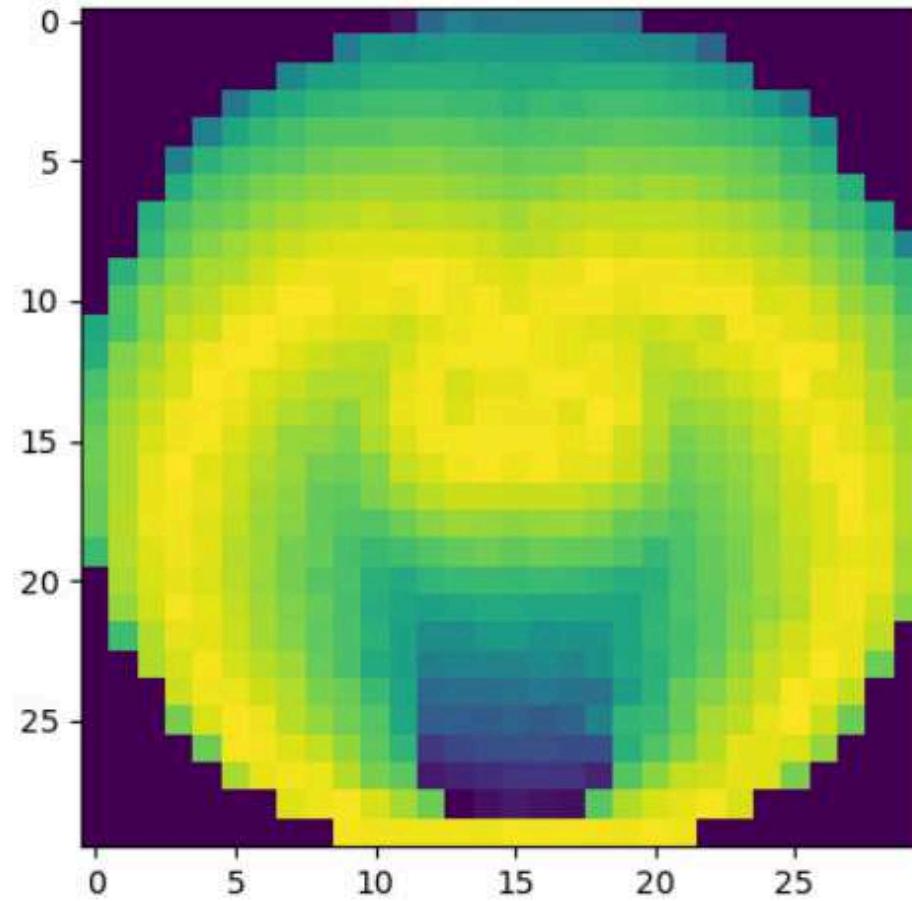
## Notebook

```
# random initialisation
archive = init_archive(num_random)

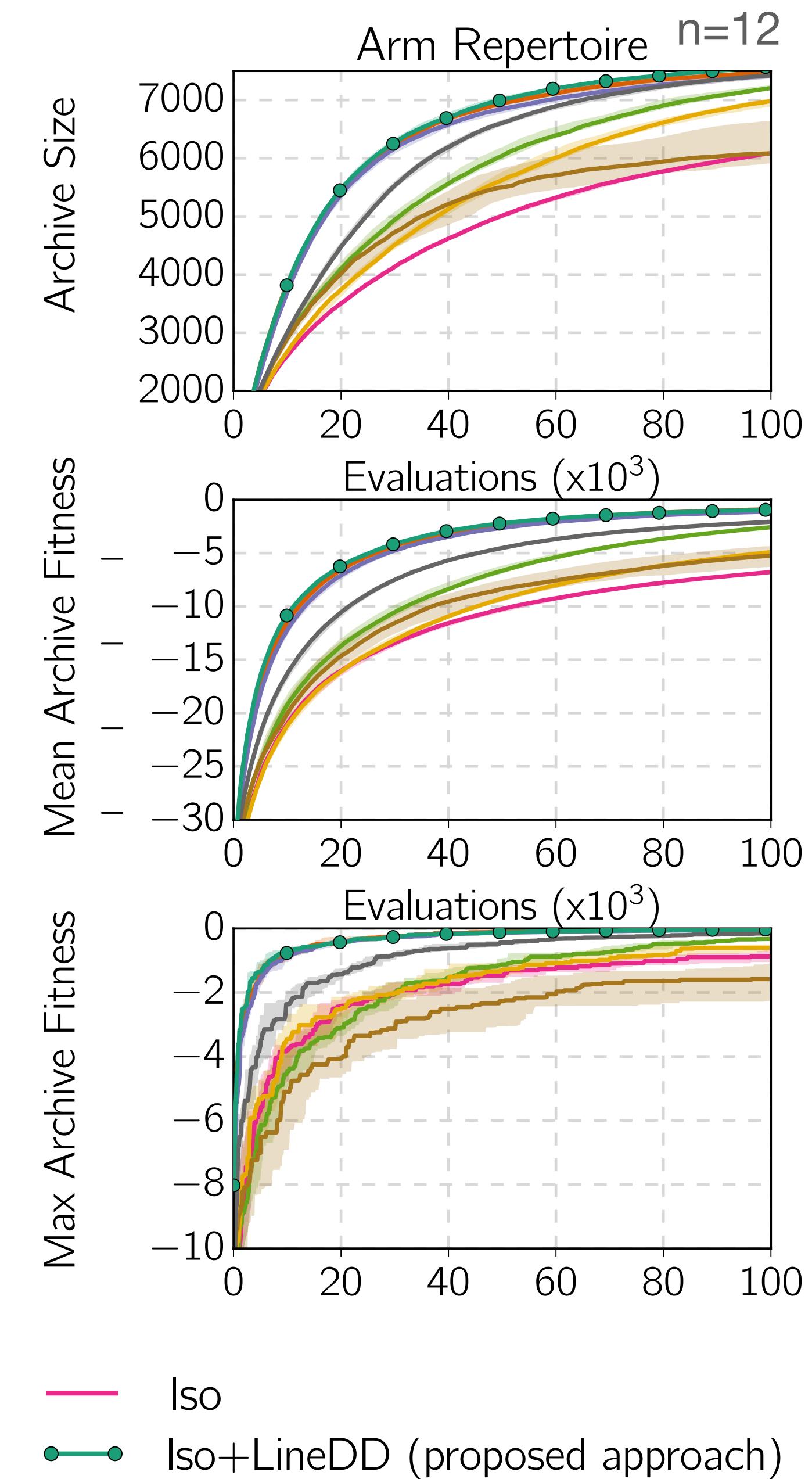
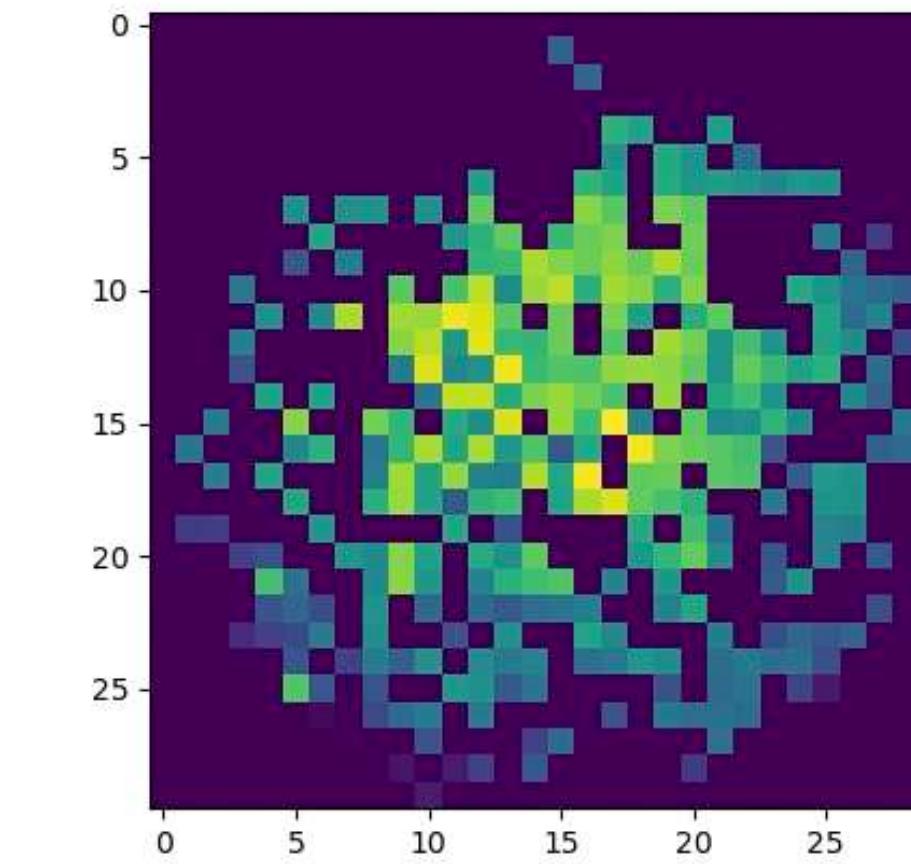
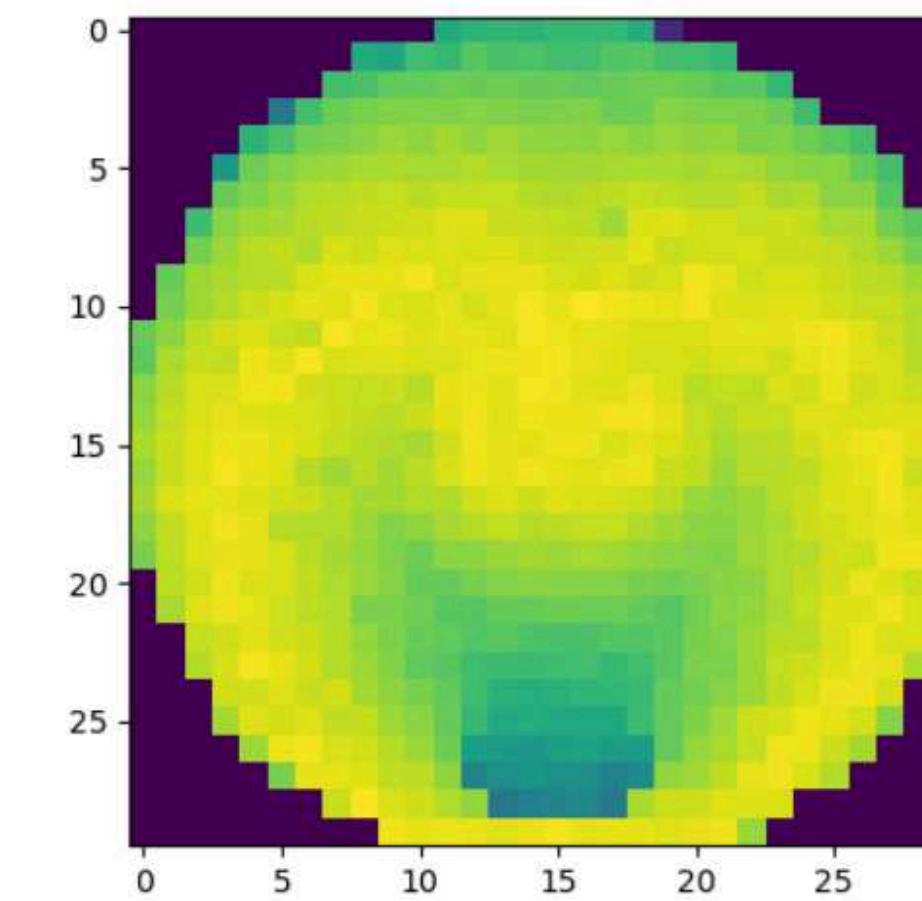
for j in tqdm(range(0, num_iterations)):
    display_archive(archive)
    for i in range(0, batch_size):
        
$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \sigma_1 \mathcal{N}(0, \mathbf{I}) + \sigma_2 (\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)}) \mathcal{N}(0, 1)$$

        # pick TWO points the archive
        x = random.choice(list(archive.values())).genotype
        y = random.choice(list(archive.values())).genotype
        # mutate it (we can use more advanced variation techniques: NEAT, etc.)
        # be careful: the first random.normal() is a 1-d normal
        g = x + (x - y) * np.random.normal(0, 0.1) + np.random.normal(0, 0.025, x.shape[0])
        # compute the fitness
        f, b = fitness(g)
        # add to archive
        add_to_archive(archive, Species(g, b, f))
```

with directional mutation



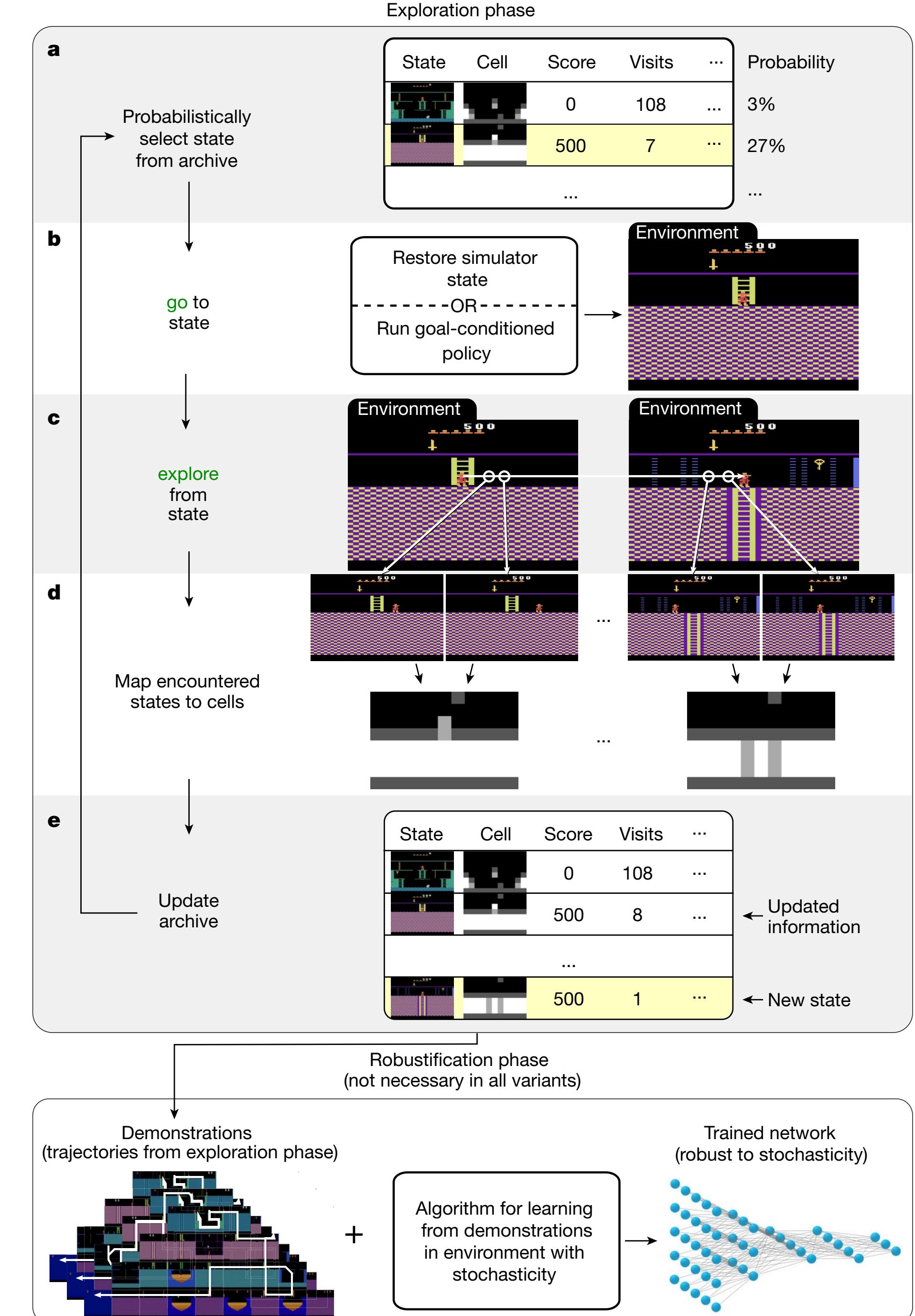
without directional mutation



# Link with Go-Explore

- Exploration concept inspired by MAP-Elites
    - behavior = state traversed
    - keep fastest to reach the state in a archive/map
  - Additions:
    - learn policies from the state sequence (robustification)
    - select cells from the map with weights + other heuristics
  - Best results in the hard games (Montezuma revenge)
$$W = \frac{1}{\sqrt{C_{\text{seen}} + 1}},$$

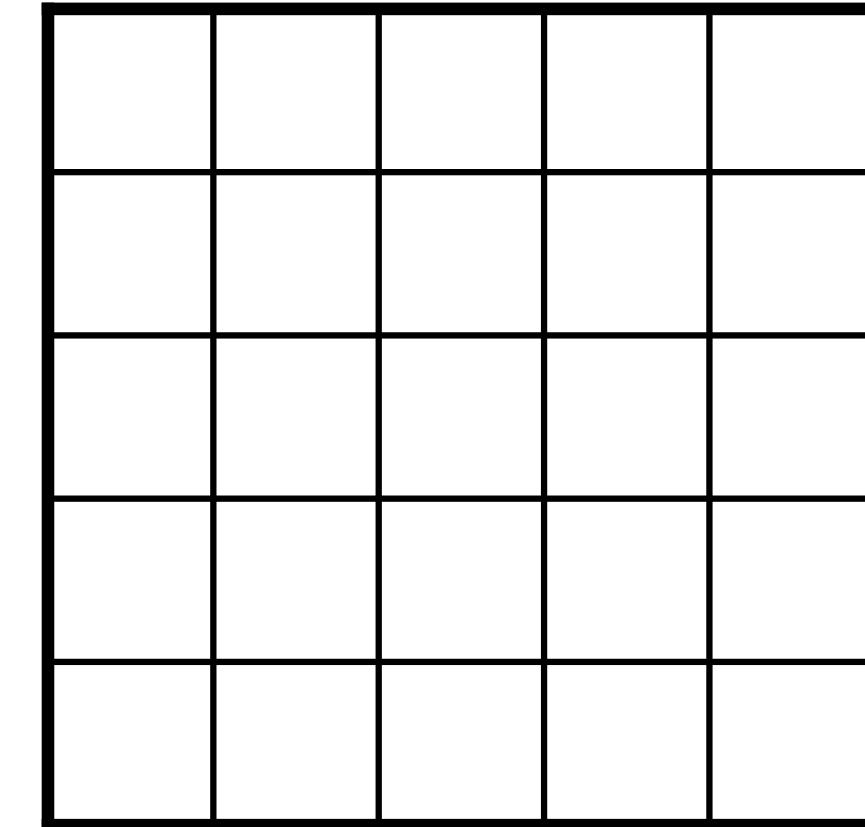
$$W = \frac{1}{\sqrt{C_{\text{seen}} + 1}},$$



**Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., & Clune, J. (2021). First return, then explore. *Nature*, 590(7847), 580-586.**

## Scaling up to high-dimensional behavioral spaces:

MAP-Elites



- **centroidal Voronoi tessellation to split the volume in cells**

→ Vassiliades, V., Chatzilygeroudis, K., & Mouret, J. B. (2017). Using Centroidal Voronoi Tessellations to Scale Up the Multi-dimensional Archive of Phenotypic Elites Algorithm. *IEEE Transactions on Evolutionary Computation*, 9.

- **distance-based archive**

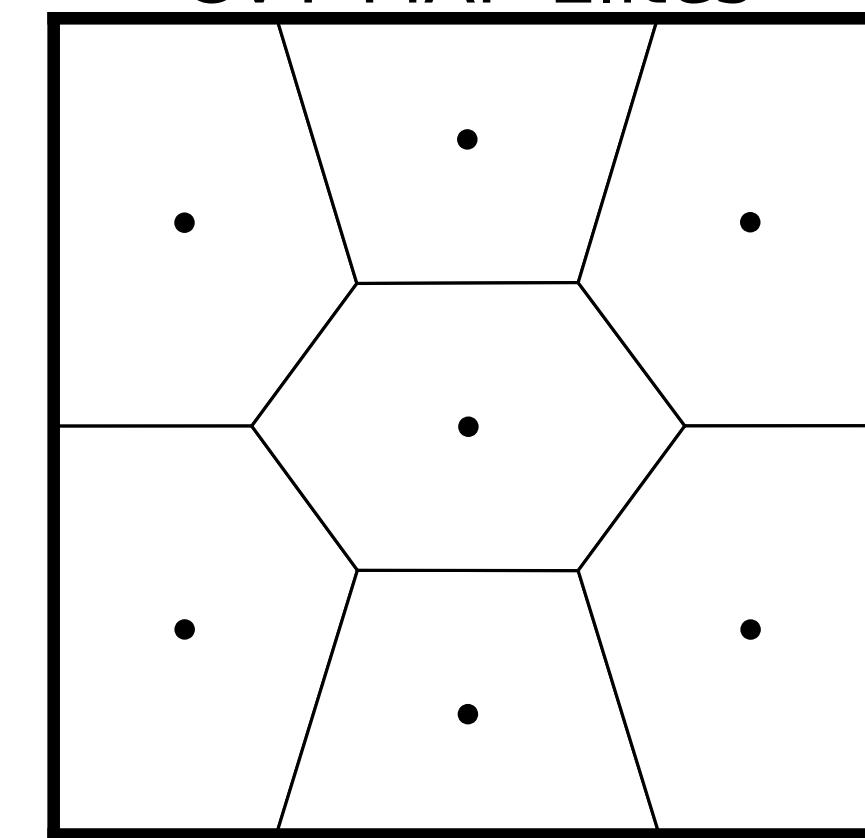
→ Cully, A., & Demiris, Y. (2017). Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2), 245-259.

- **VAE-based dimensionality reduction**

→ Cully, A. (2019). Autonomous skill discovery with quality-diversity and unsupervised descriptors. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 81-89).

## Scaling up to high-dimensional genotypes/search spaces

CVT-MAP-Elites



- **learn the hypervolume with a VAE**

→ Gaier, A., Asteroth, A., & Mouret, J. B. (2020). Discovering representations for black-box optimization. In Proceedings of the 2020 Genetic and Evolutionary Computation Conference (pp. 103-111).

- **take inspiration from OpenAI-ES**

→ Colas, C., Madhavan, V., Huizinga, J., & Clune, J. (2020). Scaling map-elites to deep neuroevolution. In Proceedings of the 2020 Genetic and Evolutionary Computation Conference (pp. 67-75).

## Improve data-efficiency

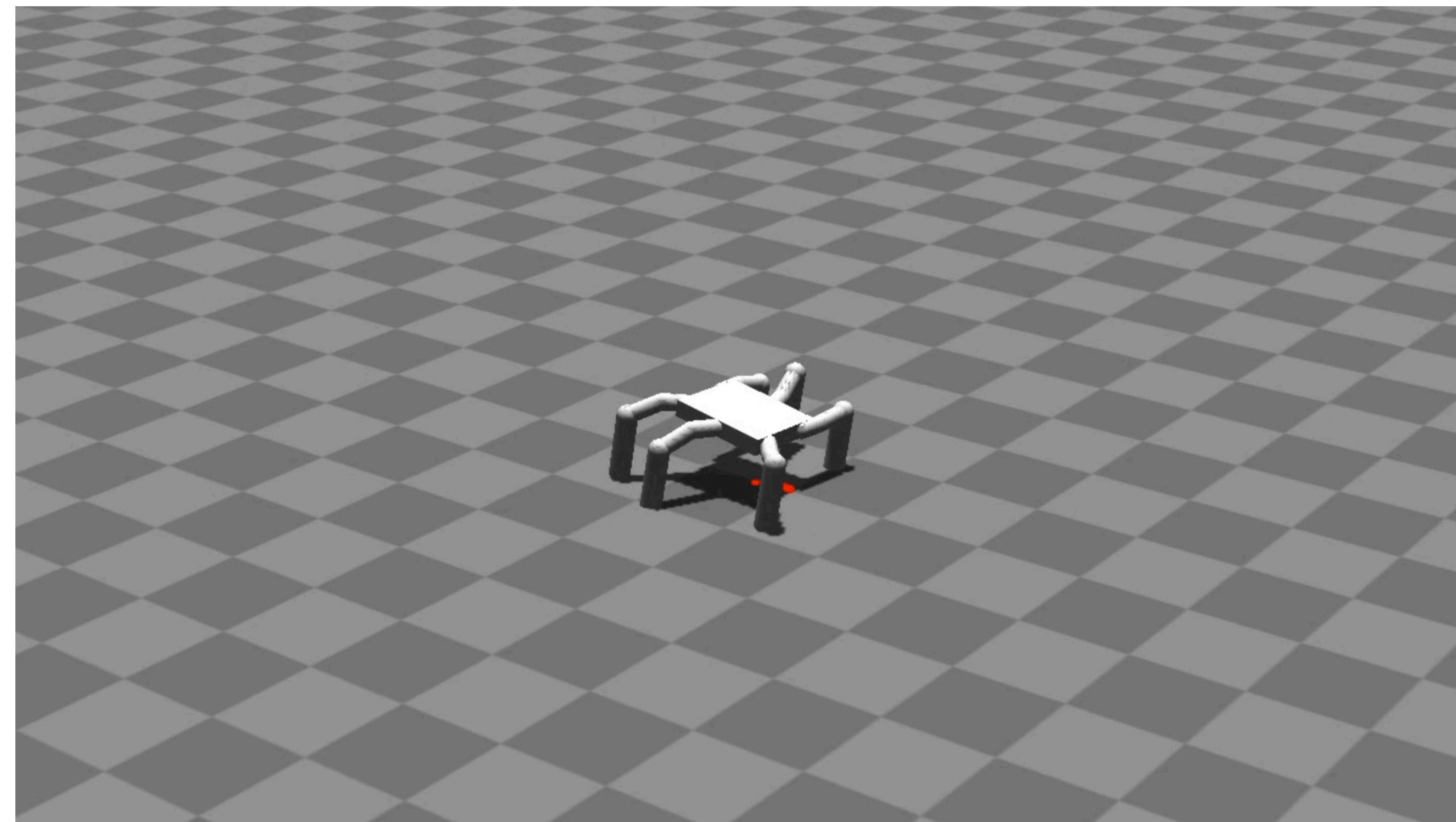
- **with surrogate models**

→ Gaier, A., Asteroth, A., & Mouret, J. B. (2018). Data-efficient design exploration through surrogate-assisted illumination. *Evolutionary computation*, 26(3), 381-410.

- **taking inspiration from CMA-ES**

→ Fontaine, M. C., Togelius, J., Nikolaidis, S., & Hoover, A. K. (2020). Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the 2020 genetic and evolutionary computation conference* (pp. 94-102).

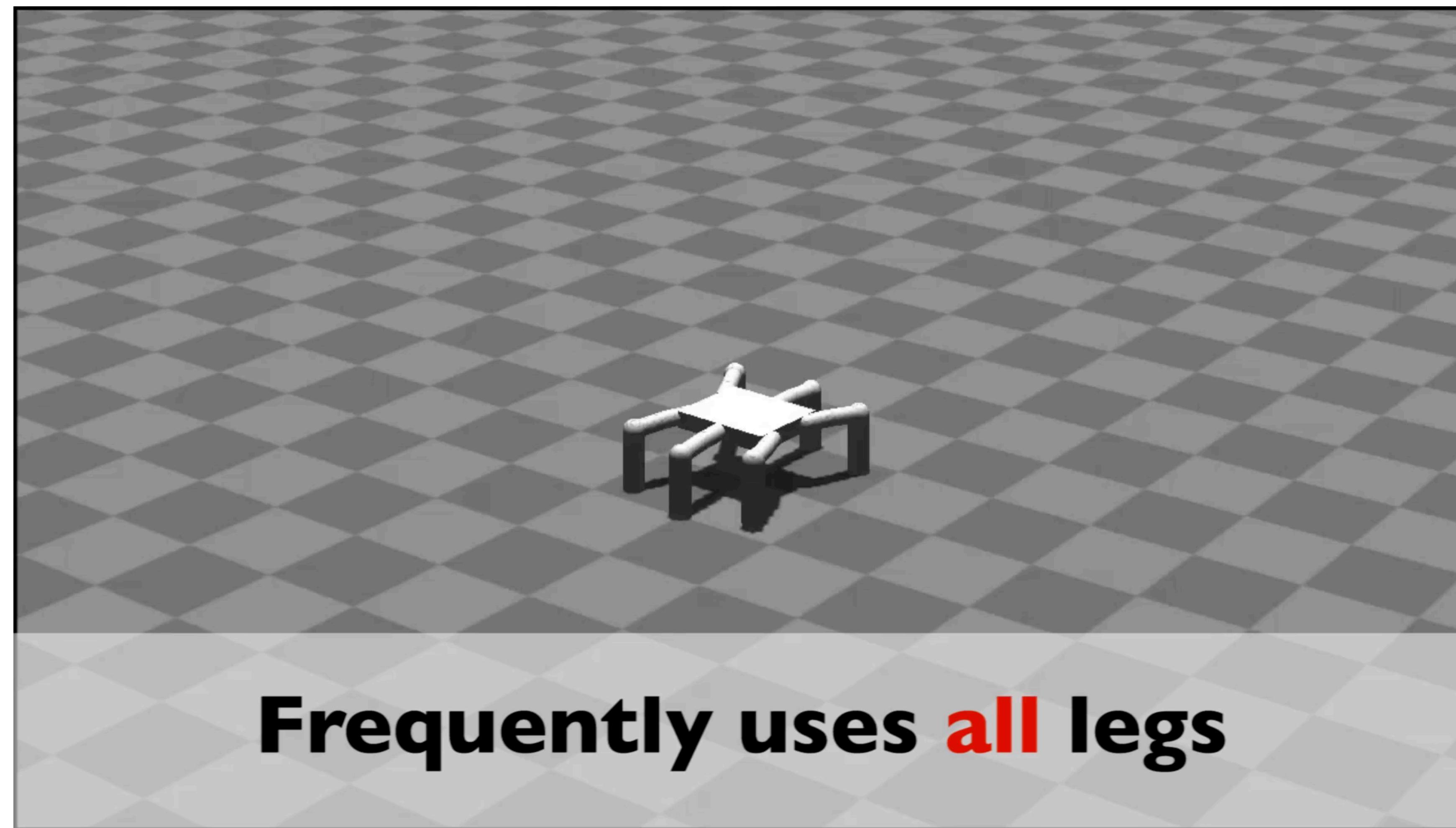
# Example: many ways of walking



**Search space:** 36  
parameters  
(open-loop controller)

**Behavior space:** 6-D  
(% of contact for each  
foot)

# Example: many ways of walking



**Frequently uses **all** legs**

**Search space:** 36 parameters  
(open-loop controller)

**Behavior space:** 6-D  
(% of contact for each foot)

# MAP-Elites vs PPO

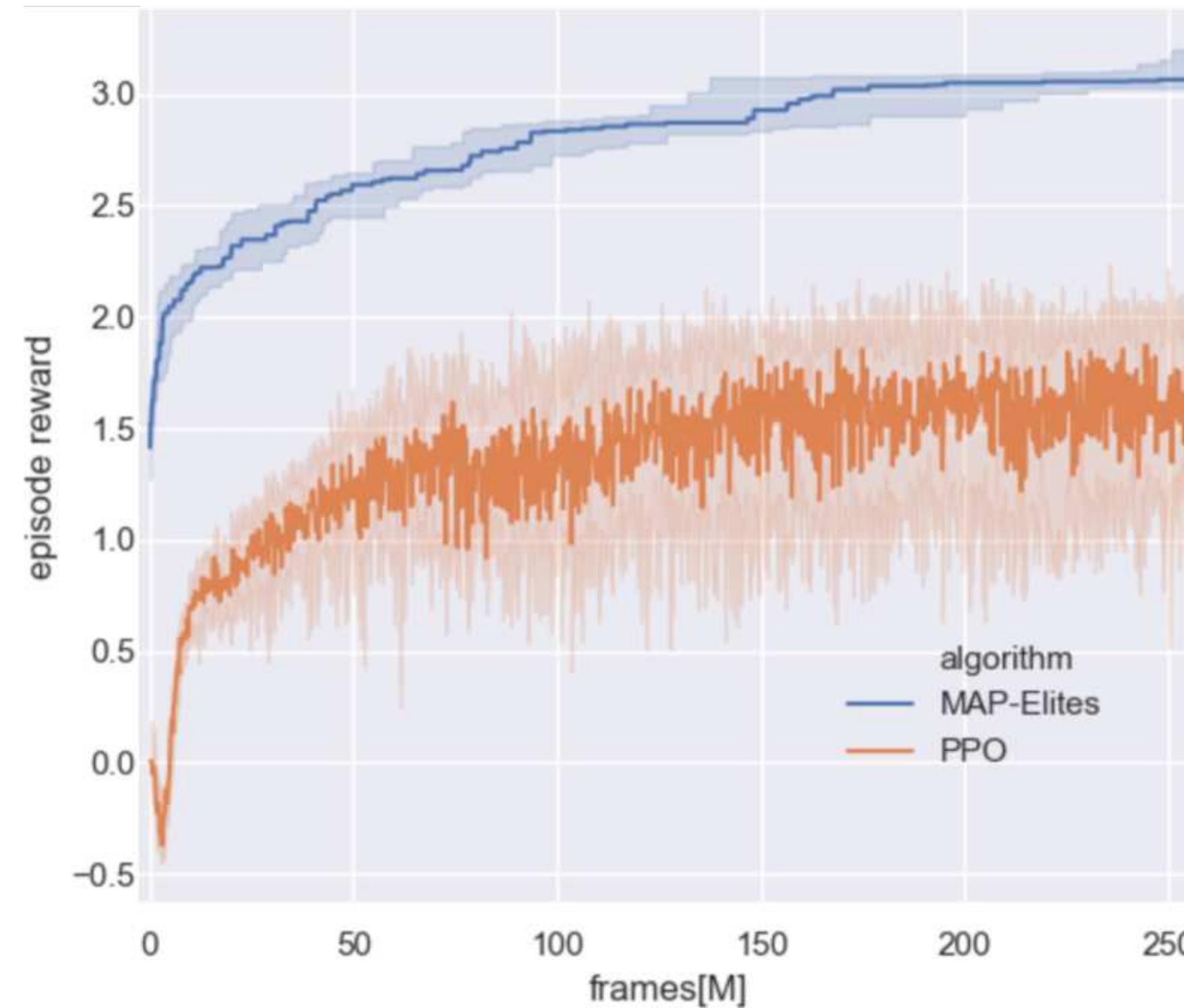
Hexapod robot – neural network

- 2 hidden layers of 6 neurons
- 18 outputs (joint positions)

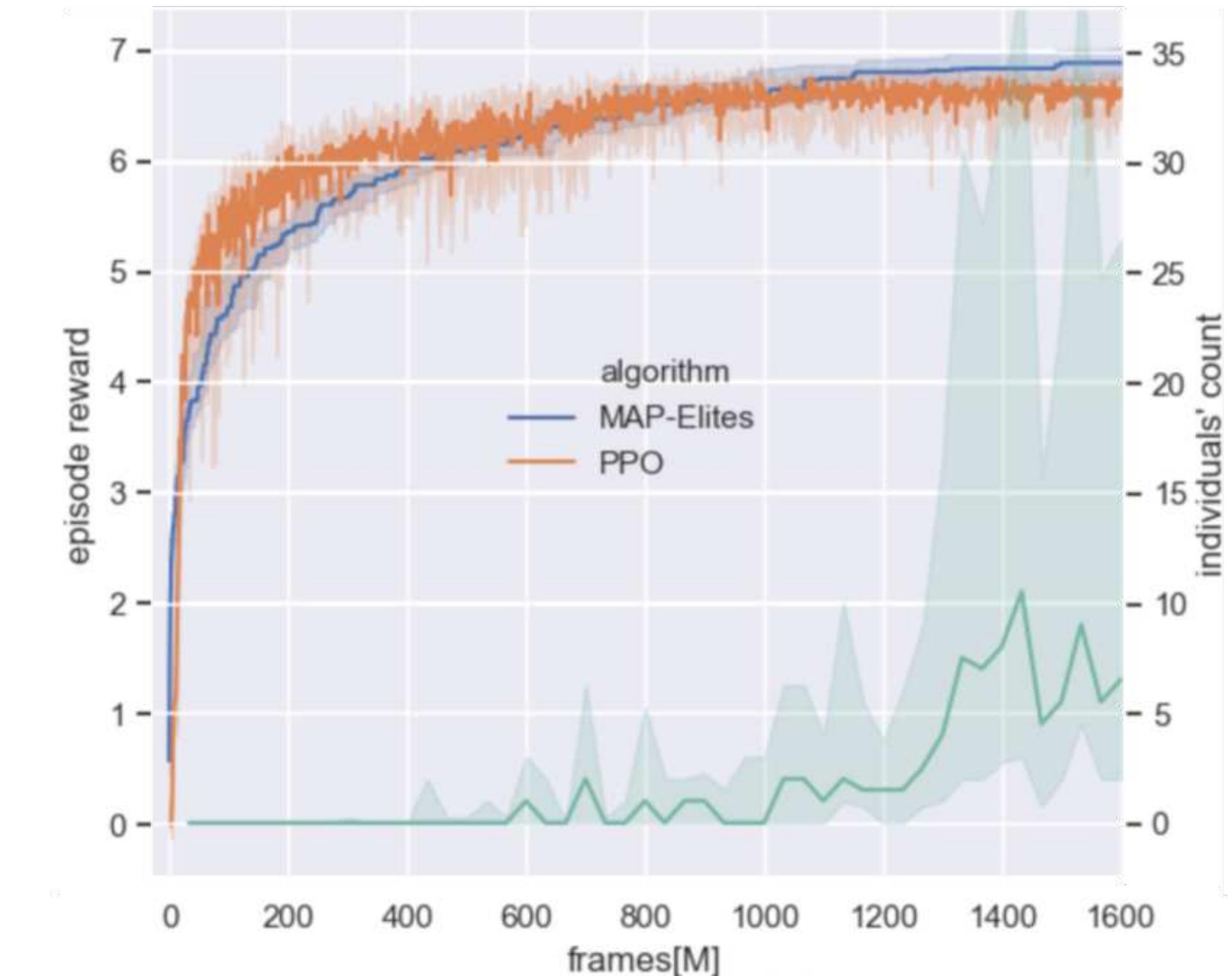


- Not a bad optimizer (small space)!
- Keep in mind: different problem (diversity)

**Open loop: input = time-modulo period**

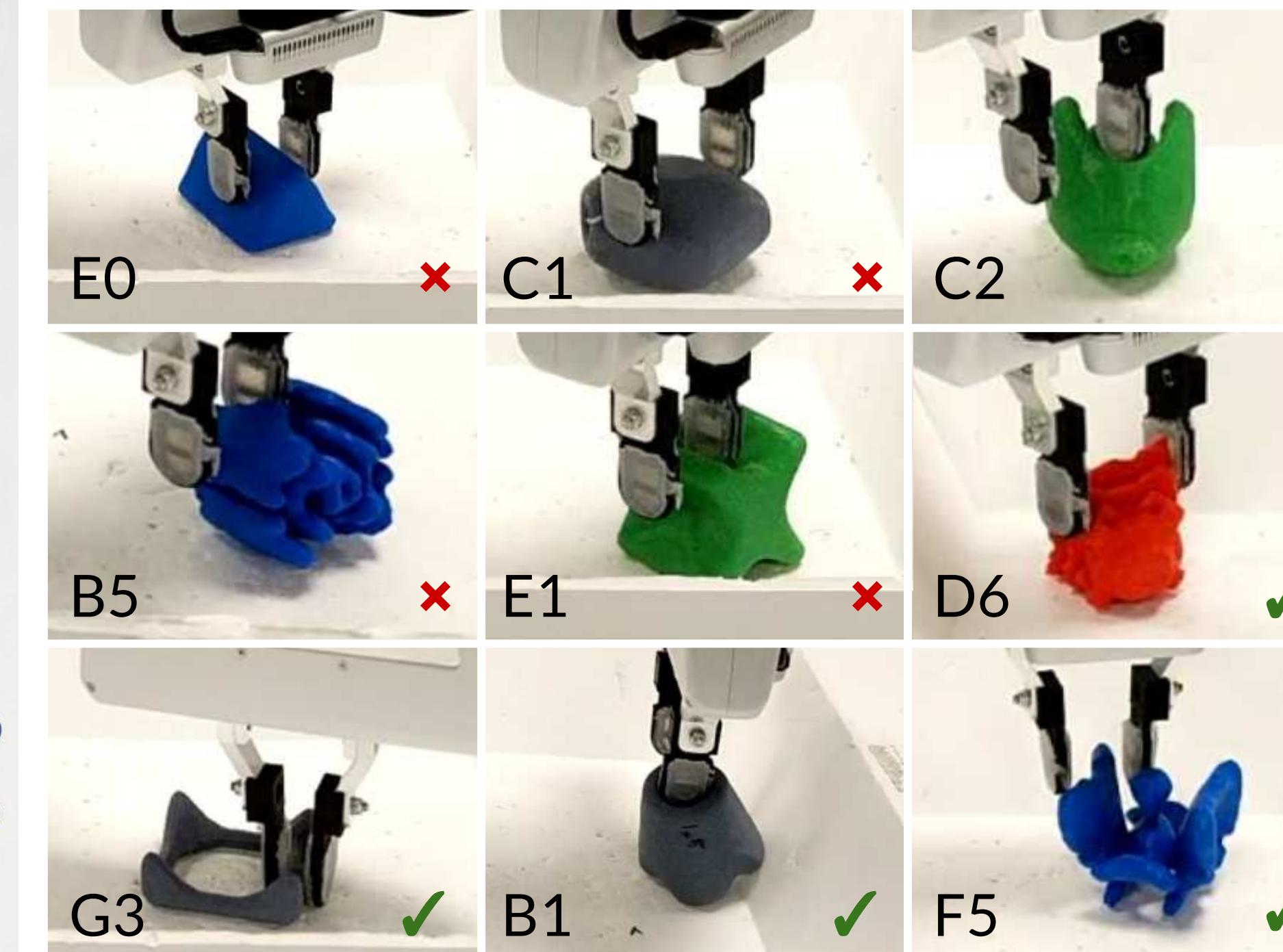
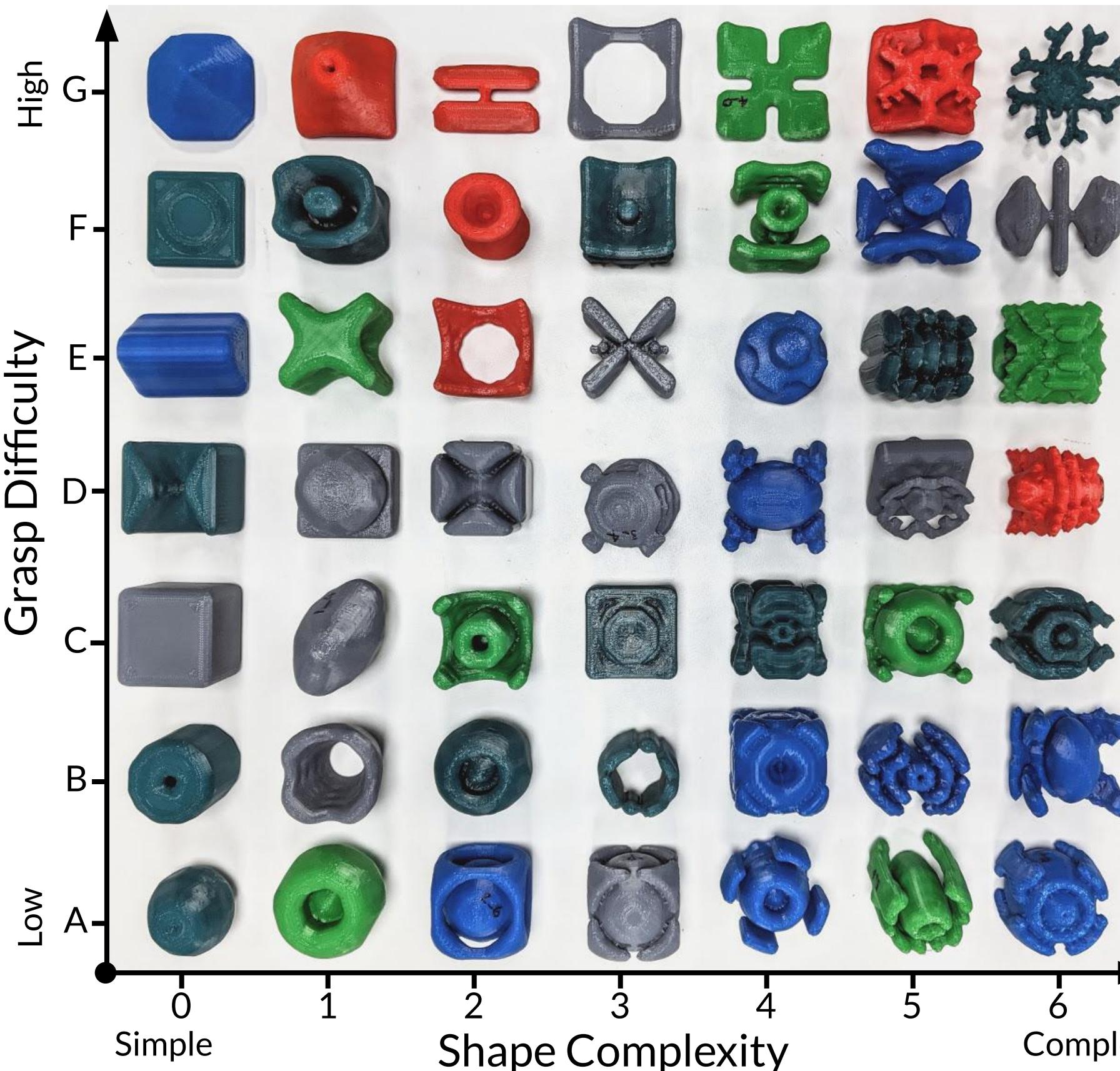


**Closed-loop: input = robot state (98 to 282 weights)**

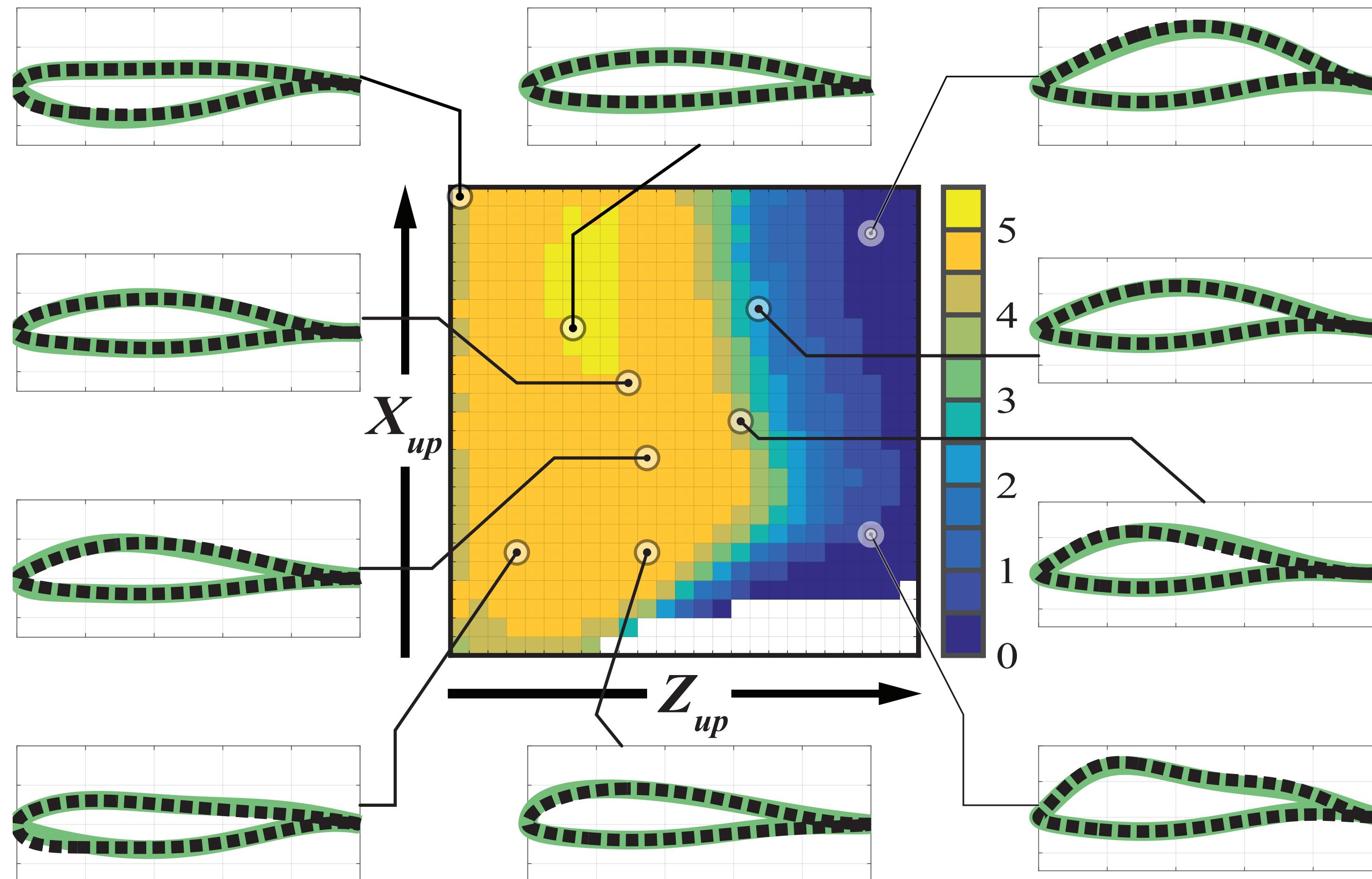


# Example: designing a dataset for grasping

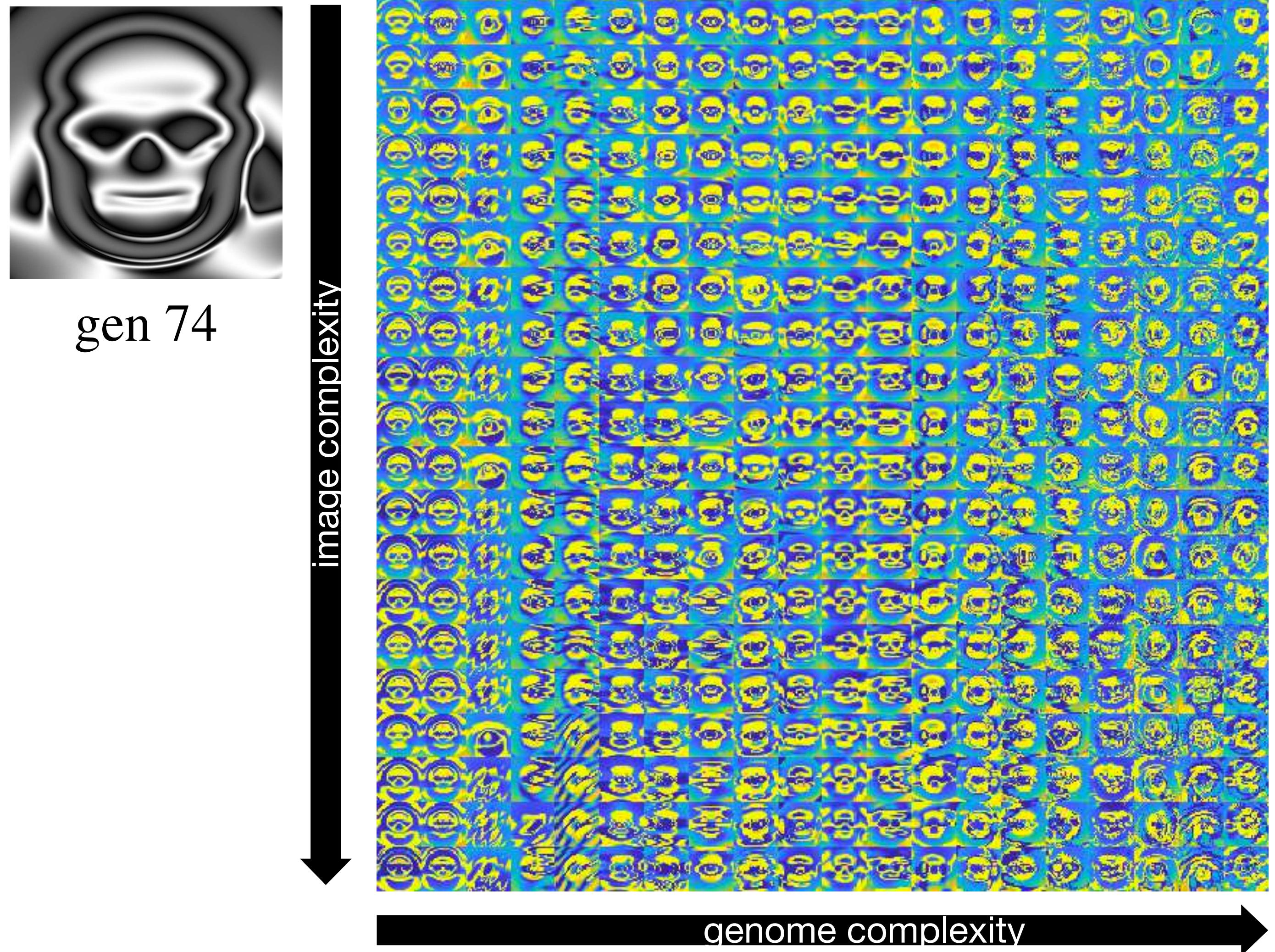
EGAD!



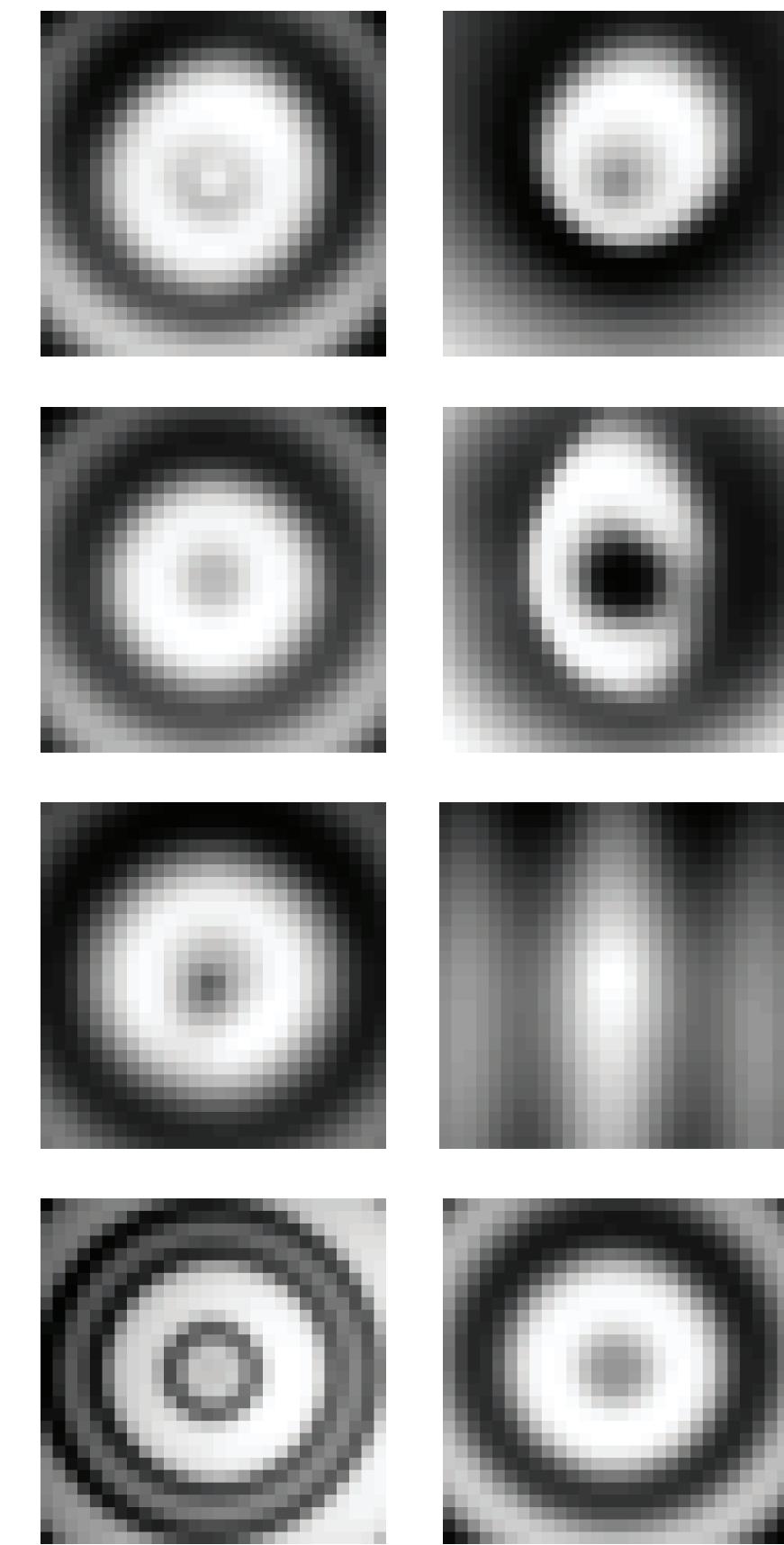
# Example: designing airfoils



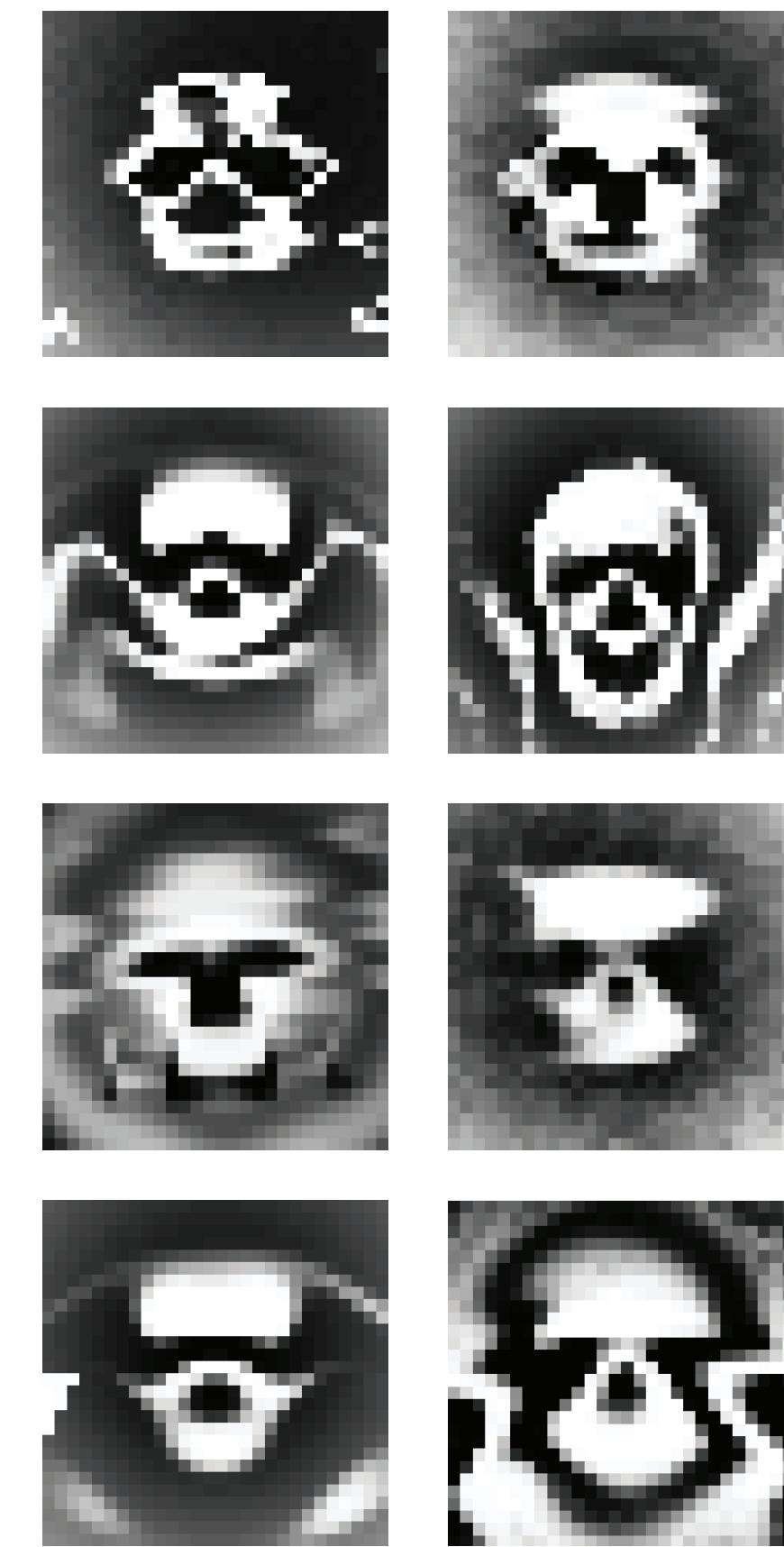
# Trying to find the skull with MAP-Elites



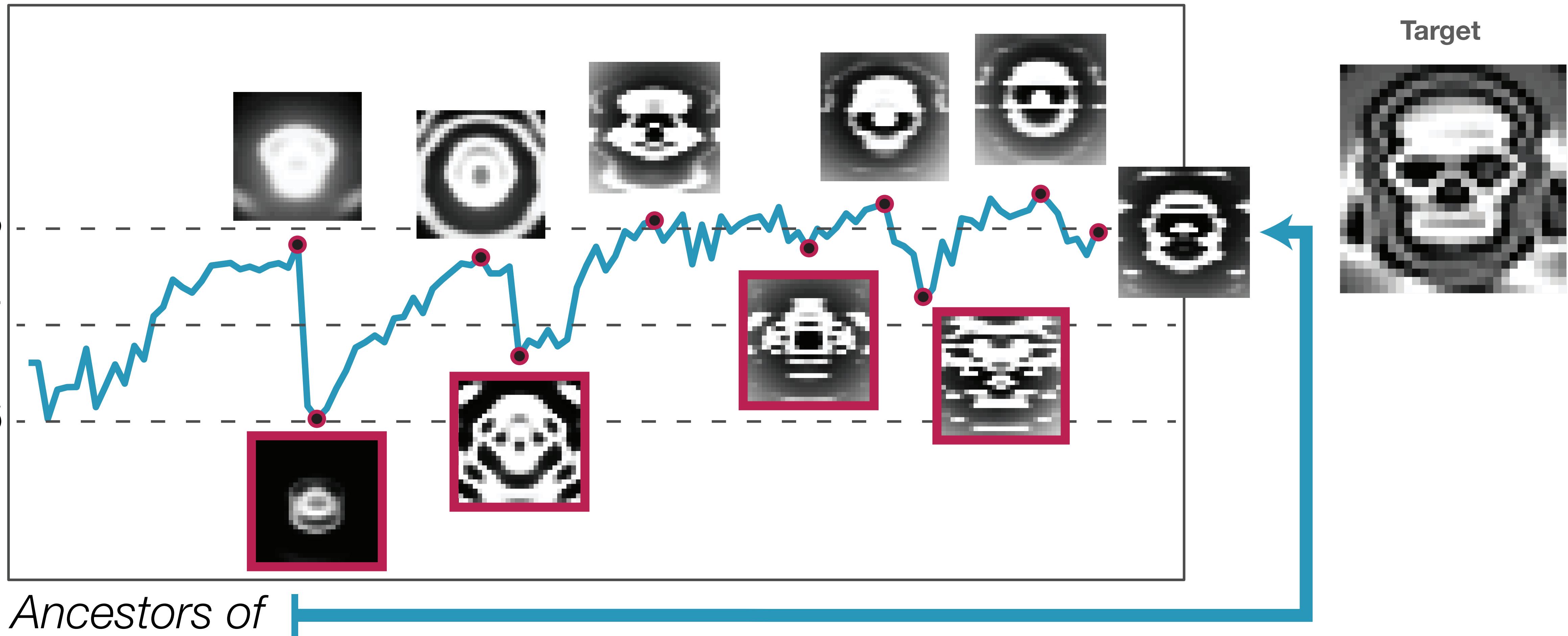
NEAT



MAP-Elites



# Trying to find the skull with MAP-Elites



# Further readings about quality diversity

**iScience**

IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 22, NO. 2, APRIL 2018

245

**Review**  
Evolving the Behavior of Machines: From Micro to Macroevolution

Jean-Baptiste Mouret<sup>1,\*</sup>

**SUMMARY**  
Evolution gave rise to creatures that are arguably more sophisticated than the greatest human-designed systems. This feat has inspired computer scientists since the advent of computing and led to optimization tools that can evolve complex neural networks for machines—an approach known as “neuroevolution.” After a few successes in designing evolvable representations for high-dimensional artifacts, the field has been recently revitalized by going beyond optimization: to many, the wonder of evolution is less in the perfect optimization of each species than in the creativity of such a simple iterative process, that is, in the diversity of species. This modern view of artificial evolution is moving the field away from microevolution, following a fitness gradient in a niche, to macroevolution, filling many niches with highly different species. It already opened promising applications, like evolving gait repertoires, video game levels for different tastes, and diverse designs for aerodynamic bikes.

**INTRODUCTION**  
Evolution by natural selection is the master algorithm of life: an infinite variation/selection loop that gave rise to the astonishing diversity of life-forms that inhabit our planet. That such an apparently simple iterative process is at the origin of so much sophistication has fascinated computer scientists since the advent of computers. Starting from the 1960s, several groups took inspiration from evolutionary biology to develop “artificial evolution” algorithms. They converged to modern “evolutionary algorithms” [De Jong, 2016]. Given a representation for possible variants (a list of numbers [De Jong, 2016], a graph [Sims, 1994], a neural network [Stanley and Miikkulainen, 2002], a grammar [Koza, 1992], and so forth) and a fitness function that measures their performance at the task, all variants loop over the same three steps.

- (1) evaluate the fitness of each individual of the population (evaluation);
- (2) rank then select the individual using their fitness value (selection);
- (3) apply variation operators on the best individuals to create a new population (variation).

The process is bootstrapped by generating an initial population randomly. Depending on the variant, a new population is created at each iteration (non-elitist algorithms) or offspring compete with their parents to stay in the population (elitist algorithm). Two variation operators are used: mutation and crossover. Mutation consists in adding random variations to a single genome; for instance, if the genome is a list of real numbers, mutation can be implemented by adding Gaussian noise to these numbers (in current algorithms, self-adjusting perturbations are used [Hansen et al., 2003]). Crossover consists in mixing two genomes of the population, in the hope of combining their features; in the case of a list of numbers, this can be implemented by adding a linear combination of the elements of the “parents” [Deb and Beyer, 2001] (depending on the representation, crossover is not always used).

From the perspective of computer science, artificial evolution is currently considered as a mathematical optimization algorithm, that is, as an algorithm that finds the maximum of a function. Such algorithms have typical applications in engineering, machine learning, bioinformatics, logistics, etc. [Kochenderfer and Wheeler, 2019] because many problems can be formalized as the maximization (or minimization) of a numerical objective. In the vast landscape of optimization algorithms, evolutionary algorithms are a good

<sup>1</sup>Inria, CNRS, Université de Lorraine, LORIA, Nancy 54000, France  
\*Correspondence:  
jean-baptiste.mouret@inria.fr  
<https://doi.org/10.1016/j.isci.2020.101731>

Science 23, 101731, November 20, 2020 © 2020 The Author(s). 1

This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

CellPress OPEN ACCESS

## Quality and Diversity Optimization: A Unifying Modular Framework

Antoine Cully and Yiannis Demiris, Senior Member, IEEE

**Abstract**—The optimization of functions to find the best solution according to one or several objectives has a central role in many engineering and research fields. Recently, a new family of optimization algorithms, named quality-diversity (QD) optimization, has been introduced, and contrasts with classic algorithms. Instead of searching for a single solution, QD algorithms are searching for a large collection of both diverse and high-performing solutions. The role of this collection is to cover the range of possible solution types as much as possible, and to contain the best solution for each type. One of the contributions of this paper is threefold. First, we present a unifying framework of QD optimization algorithms that covers the main algorithms of this family (multidimensional archive of phenotypic elites and the novelty search with local competition), and that highlights the large variety of variants that can be investigated within this family. Second, we propose algorithms with a new selection mechanism for QD algorithms that outperforms all the algorithms tested in this paper. Lastly, we present a new collection management that overcomes the erosion issues observed when using unstructured collections. These three contributions are supported by extensive experimental comparisons of QD algorithms on three different experimental scenarios.

**Index Terms**—Behavioral diversity, collection of solutions, novelty search, optimization methods, quality-diversity (QD).

### I. INTRODUCTION

SEARCHING for high-quality solutions within a typically high-dimensional search space is an important part of engineering and research. Intensive work has been done in recent decades to produce automated procedures to generate these solutions, which are commonly called “optimization algorithms.” The applications of such algorithms are numerous and range from modeling purposes to product design [1]. More recently, optimization algorithms have become the core of most machine learning techniques. For example, they are used to adjust the weights of neural networks in order to minimize the classification error [2], [3], or to allow robots to learn new behaviors that maximize their velocity or accuracy [4], [5].

Manuscript received September 5, 2016; revised December 26, 2016, March 8, 2017, and May 11, 2017; accepted May 11, 2017. Date of publication June 26, 2017; date of current version March 28, 2018. This work was supported by the EU Horizon 2020 Project PAL under Grant 643783-RIA. (Corresponding author: Antoine Cully.)

Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2BT, U.K. (e-mail: a.cully@imperial.ac.uk; y.demiris@imperial.ac.uk).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>. Digital Object Identifier 10.1109/TEVC.2017.2704781

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see <http://creativecommons.org/licenses/by/3.0/>

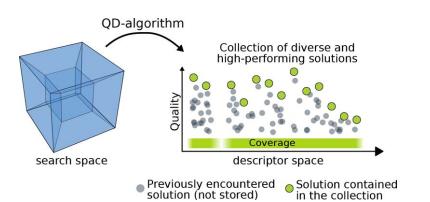
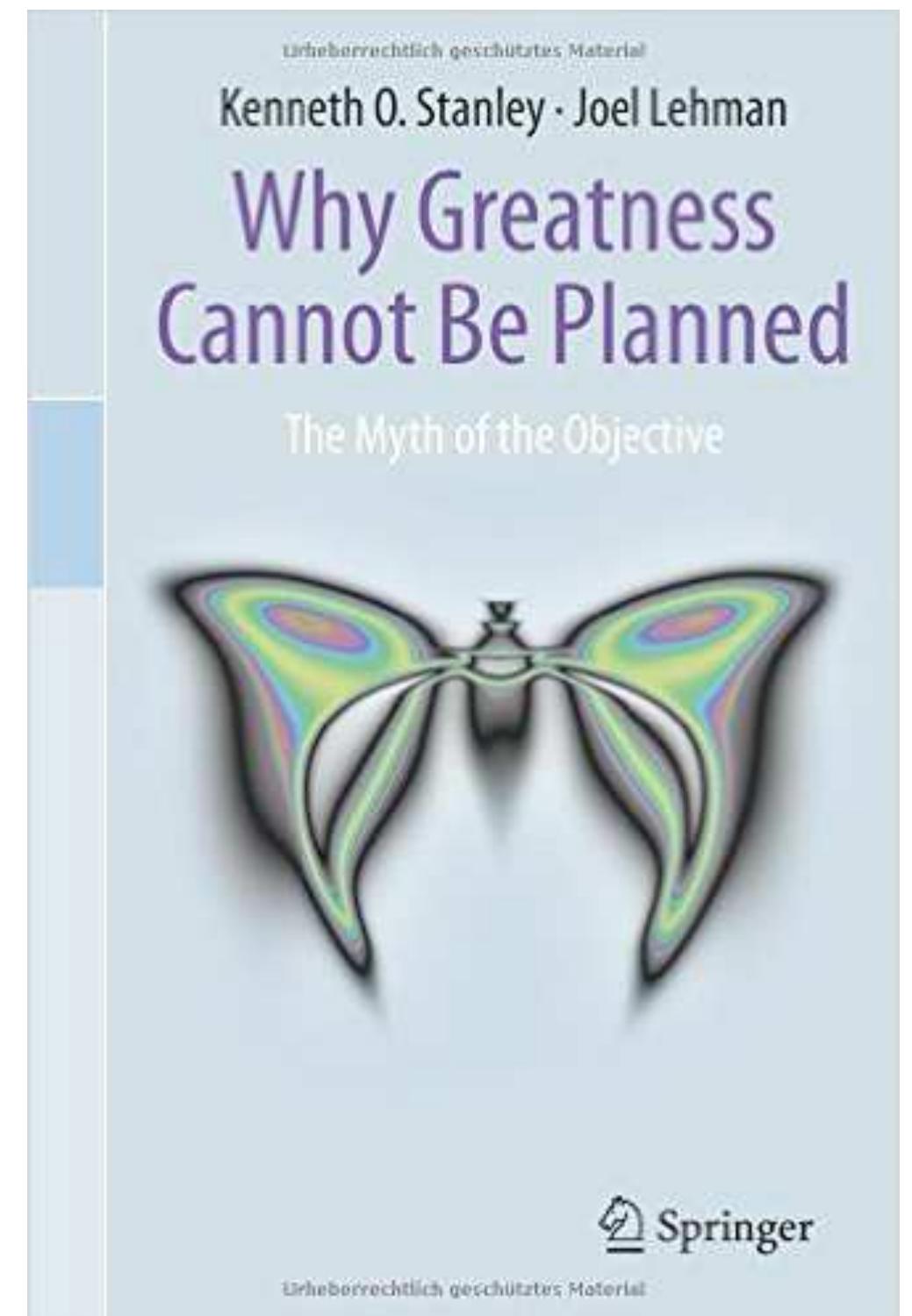


Fig. 1. Objective of a QD algorithm is to generate a collection of both diverse and high-performing solutions. This collection represents a (model free) projection of the high-dimensional search space into a lower dimensional space defined by a set of solution descriptors. The quality of a collection is defined by its coverage of the descriptor space and by the global quality of the solutions that are kept in the collection.

Inspired by the ability of natural evolution to generate species that are well adapted to their environment, evolutionary computation has a long history in the domain of optimization, particularly in stochastic optimization [6]. For example, evolutionary methods have been used to optimize the morphologies and the neural networks of physical robots [7], and to infer the equations behind collected data [8]. These optimization abilities are also the core of evolutionary robotics in which evolutionary algorithms are used to generate neural networks, robot behaviors, or objects [9], [10].

However, from a more general perspective and in contrast with artificial evolution, natural evolution does not produce one effective solution but rather an impressively large set of different organisms, all well adapted to their respective environment. Surprisingly, this divergent search aspect of natural evolution is rarely considered in engineering and research fields, even though the ability to provide a large and diverse set of high-performing solutions appears to be promising for multiple reasons.

For example, in a set of effective solutions, each provides an alternative in the case that one solution turns out to be less effective than expected. This can happen when the optimization process takes place in simulation, and the obtained result does not transfer well to reality (a phenomenon called the reality gap [11]). In this case, a large collection of solutions can quickly provide a working solution [4]. Maintaining multiple solutions and using them concurrently to generate actions or predict actions when done by other agents has also been shown to be very successful in bioinspired motor control and cognitive robotics experiments [12].



**Mouret, J. B. (2020).** Evolving the behavior of machines: from micro to macroevolution. *Isience*, 101731.

<https://quality-diversity.github.io>

**Cully, A., & Demiris, Y. (2017).** Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2), 245–259.

**KO Stanley, J Lehman. Why Greatness Cannot Be Planned (2015) - Springer**