Introduction to Evolutionary Robotics

Robotics Class, 2017, Prof. Laschi

Teaching Assistant:

Francesco Corucci
f.corucci@santannapisa.it
http://sssa.bioroboticsinstitute.it/user/1507
@f_corucci

Who am I

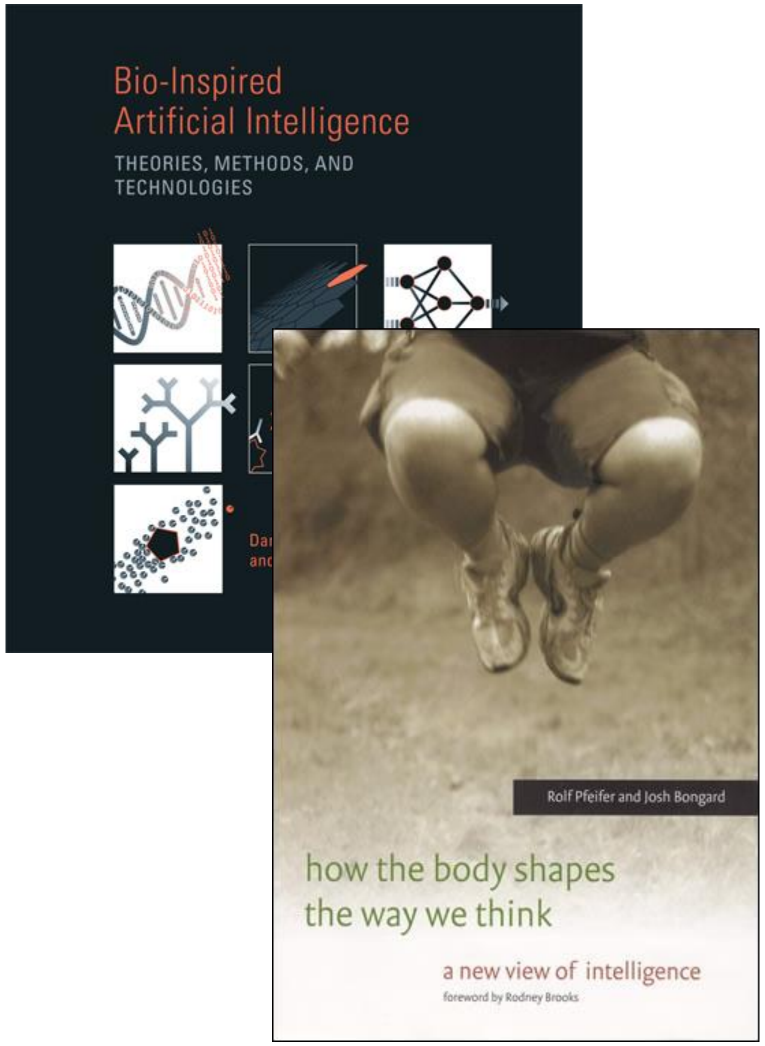
- 4th year PhD Student in BioRobotics at The BioRobotics Institute, Scuola Superiore Sant’Anna, Pisa (IT)
(Advisor: Prof. Laschi)
- Visiting researcher, Morphology, Evolution and Cognition Lab, Vermont Complex Systems Center, University of Vermont (USA)
(Advisor: Prof. Josh Bongard)

→ TA for this course, 4h on *Evolutionary Robotics* + projects on this topic

More info: http://sssa.bioroboticsinstitute.it/user/1507

Part of this material is inspired by or adapted from:

- Floreano, Matiussi, "*Bio-Inspired Artificial Intelligence - Theories, Methods, and Technologies*", MIT press (2008).
- Pfeifer, Bongard, *"How the body shapes the way we think: a new view of intelligence"*. MIT press (2006).
- *"Evolutionary Robotics"* class (CS206) at University of Vermont (USA), Prof. J. Bongard
- *LudoBots*, a Reddit-based MOOC on Evolutionary Robotics



WWW.REDDIT.COM/R/LUDOBOTS
MASSIVELY
OPEN
ONLINE
COURSE
EDUCATION. REDDEFINED.

https://www.reddit.com/r/ludobots/wiki/index
http://www.reddit.com/r/ludobots
http://www.meclab.org/

Outline

- **Introduction and motivation**
- **Evolutionary Algorithms**
- **Human-competitive design and *“perverse instantiation”***
- **Developmental encodings and co-evolution of artificial brains and bodies**
- **The “reality gap” problem**

Appendix:

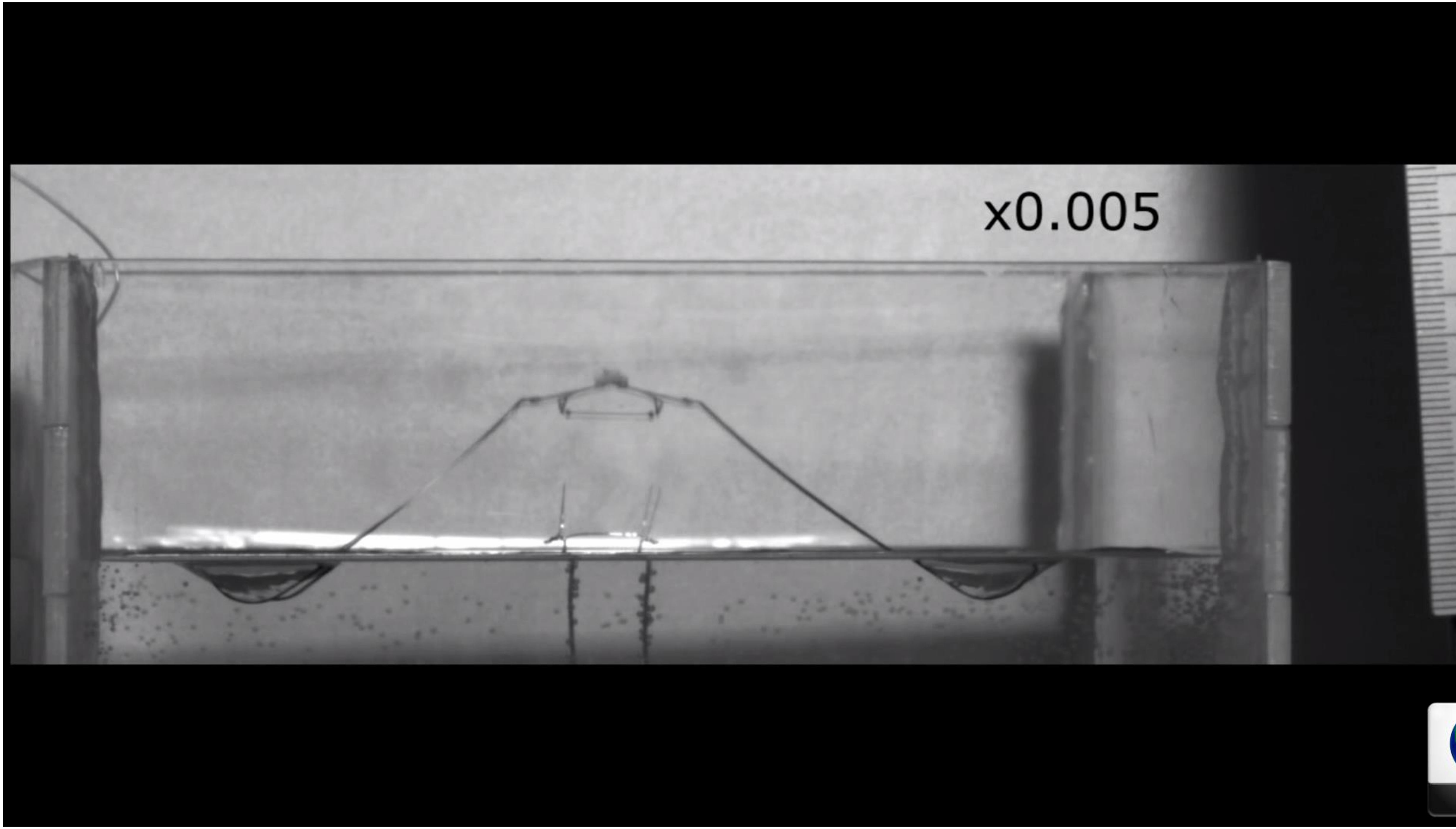
- **VoxCad**, multi-material physics engine for soft-robot analysis, design and evolution

Introduction and motivation

Biorobotics: sometimes successful…

Passive dynamic walker, Mc Geer 1990

Biorobotics: sometimes successful…



Koh et al. 2015, Science

…sometimes not so much

Robots failing at the Darpa Robotic Challenge 2015 (IEEE Spectrum)

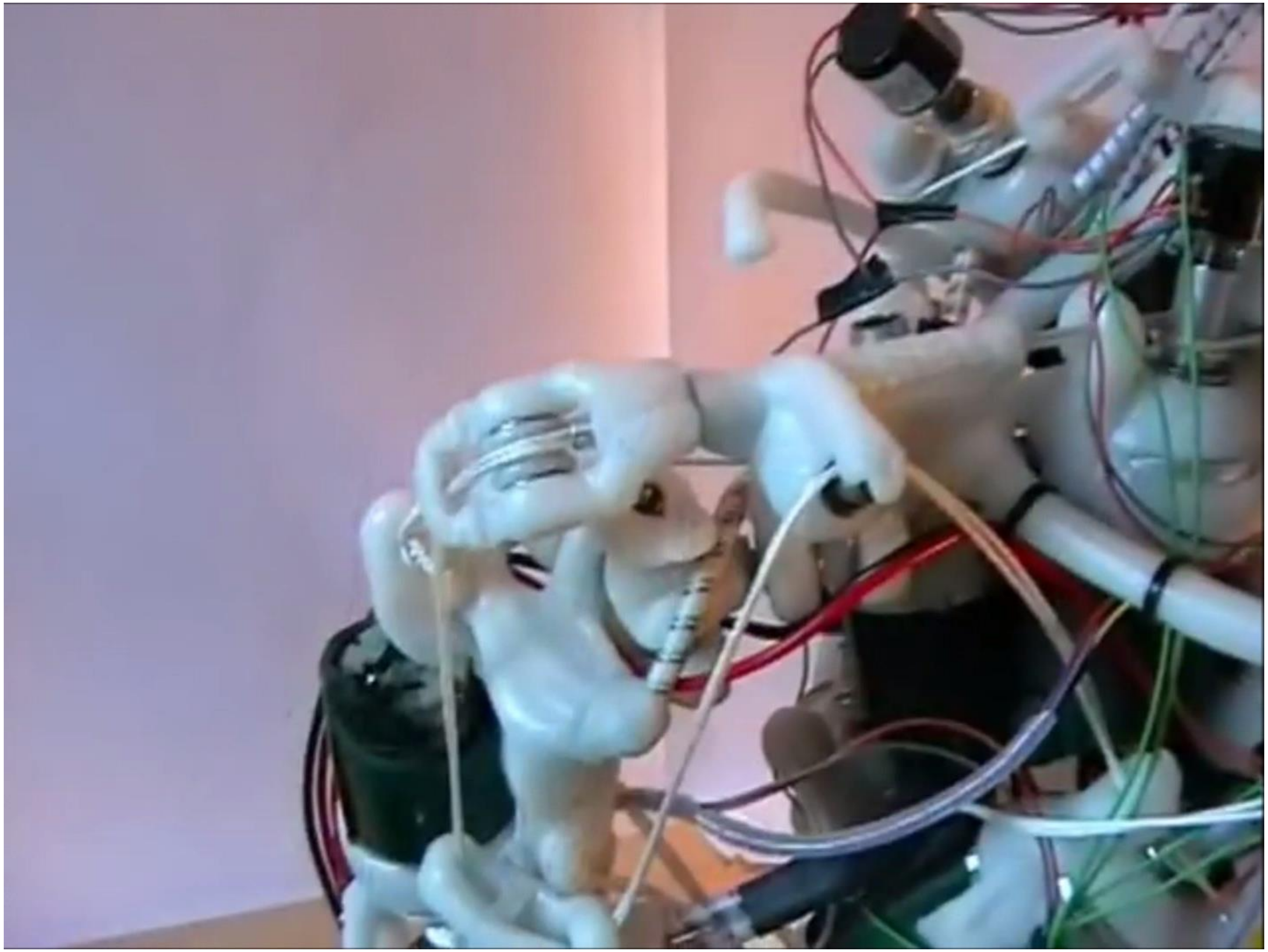
Biorobotics: challenges and risks

Biorobotics focuses on something usually very <u>complex</u> (e.g. a complete creature, a specific behavior, etc.) and tries to <u>extract underlying principles</u>, that can <u>guide the design</u> of robotic artifacts

Biorobotics: challenges and risks

- Requires a lot of human knowledge
- Difficult to extract insights, easier to just replicate/add complexity, that:
  - may or may not be useful
  - often we don't know how to handle
  - can hinder underlying principles
- Top down complexity + attempt to simplify

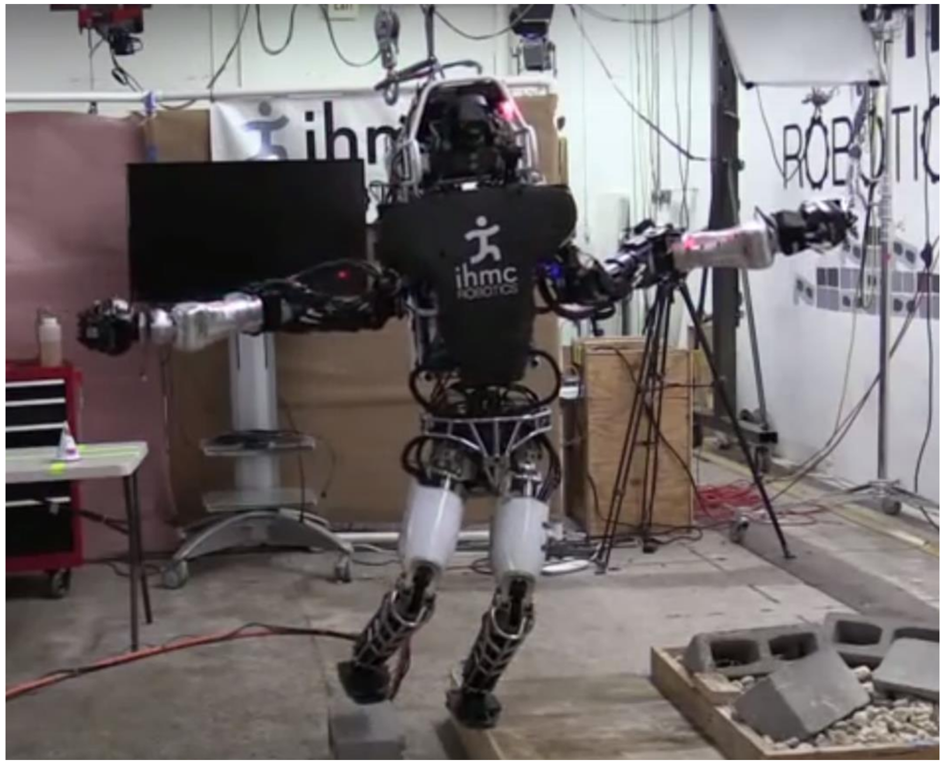
Biorobotics: challenges and risks

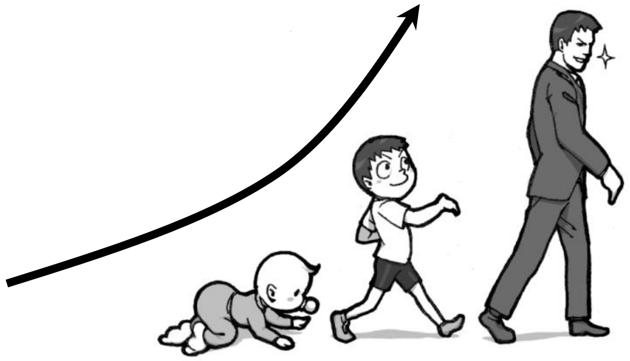


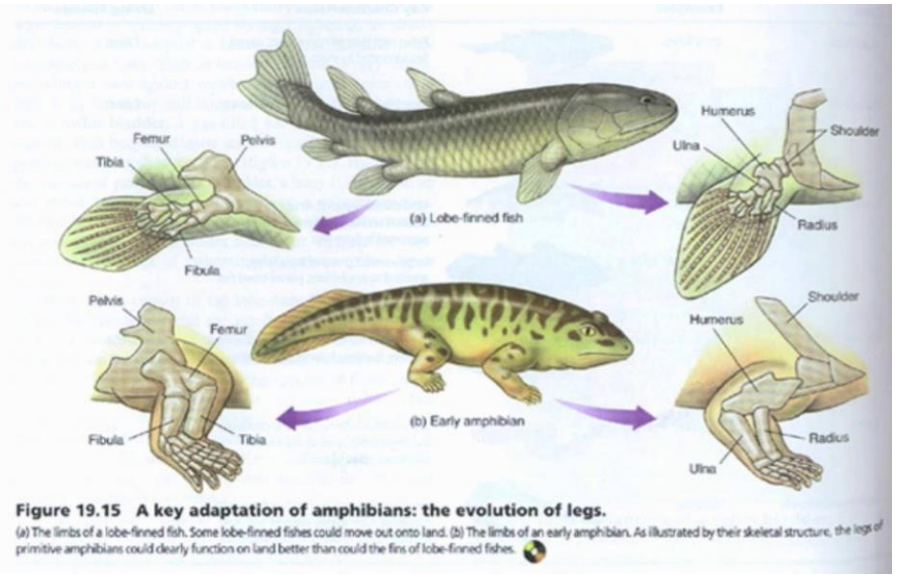
ECCE robot, Embodied Cognition in a Compliantly Engineered Robot

URL

Locomotion

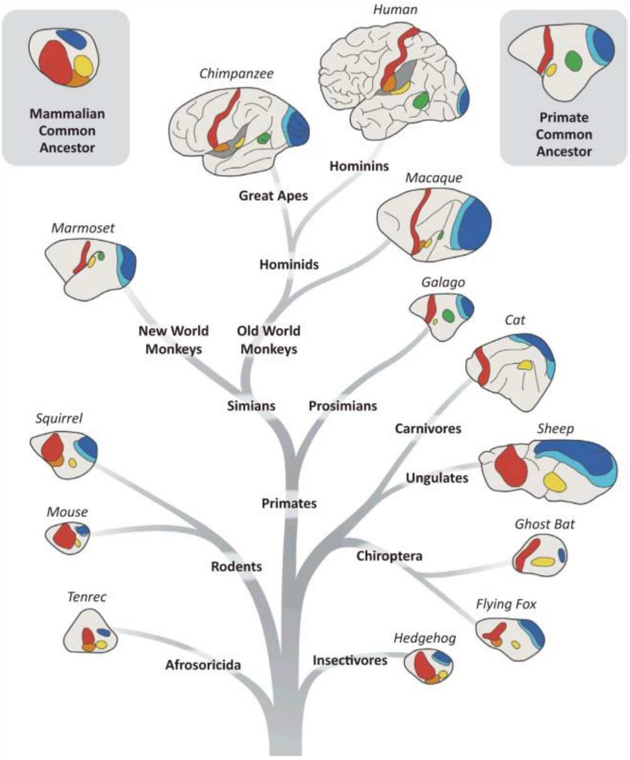






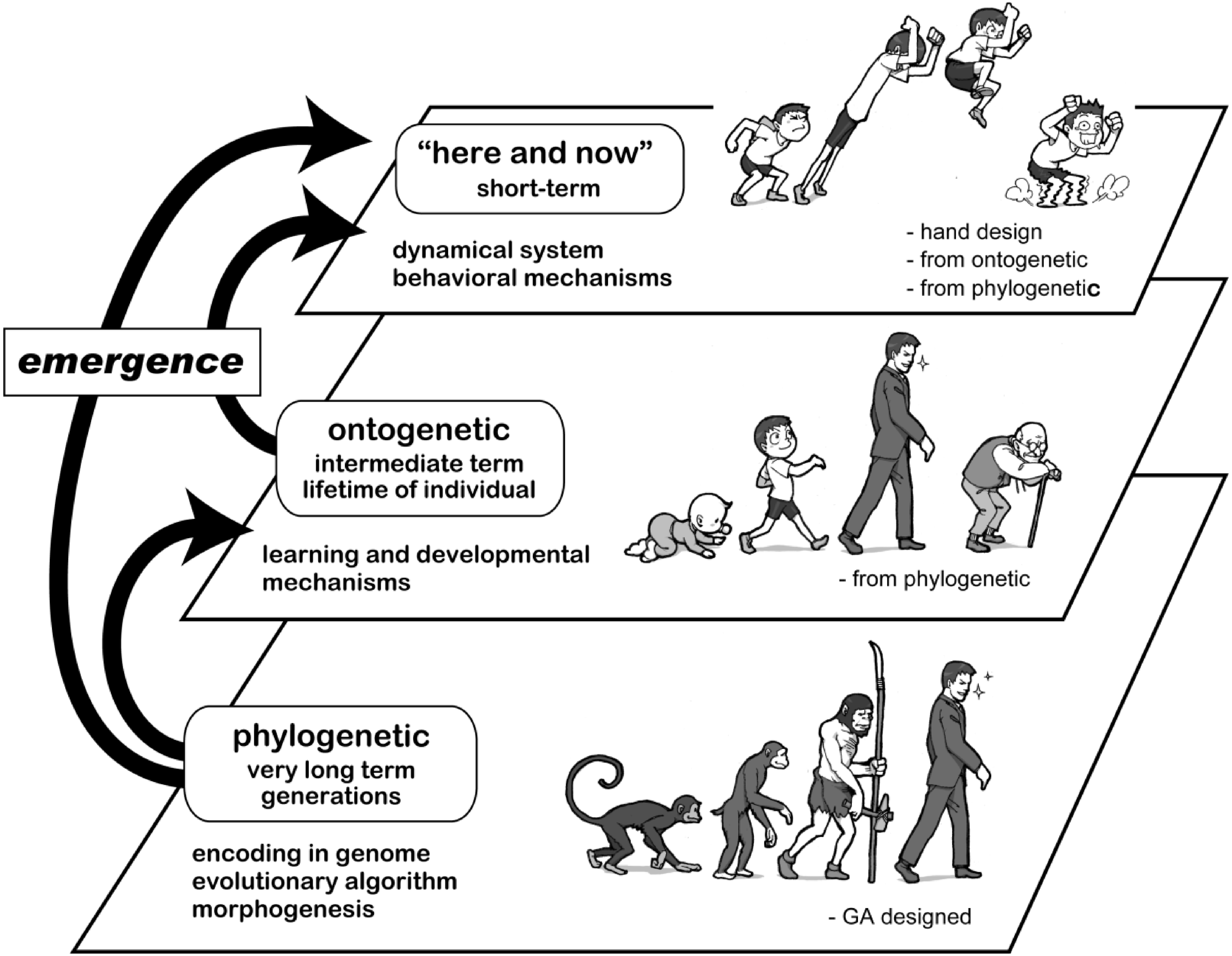
Different ways to approach the same problems

Cognition

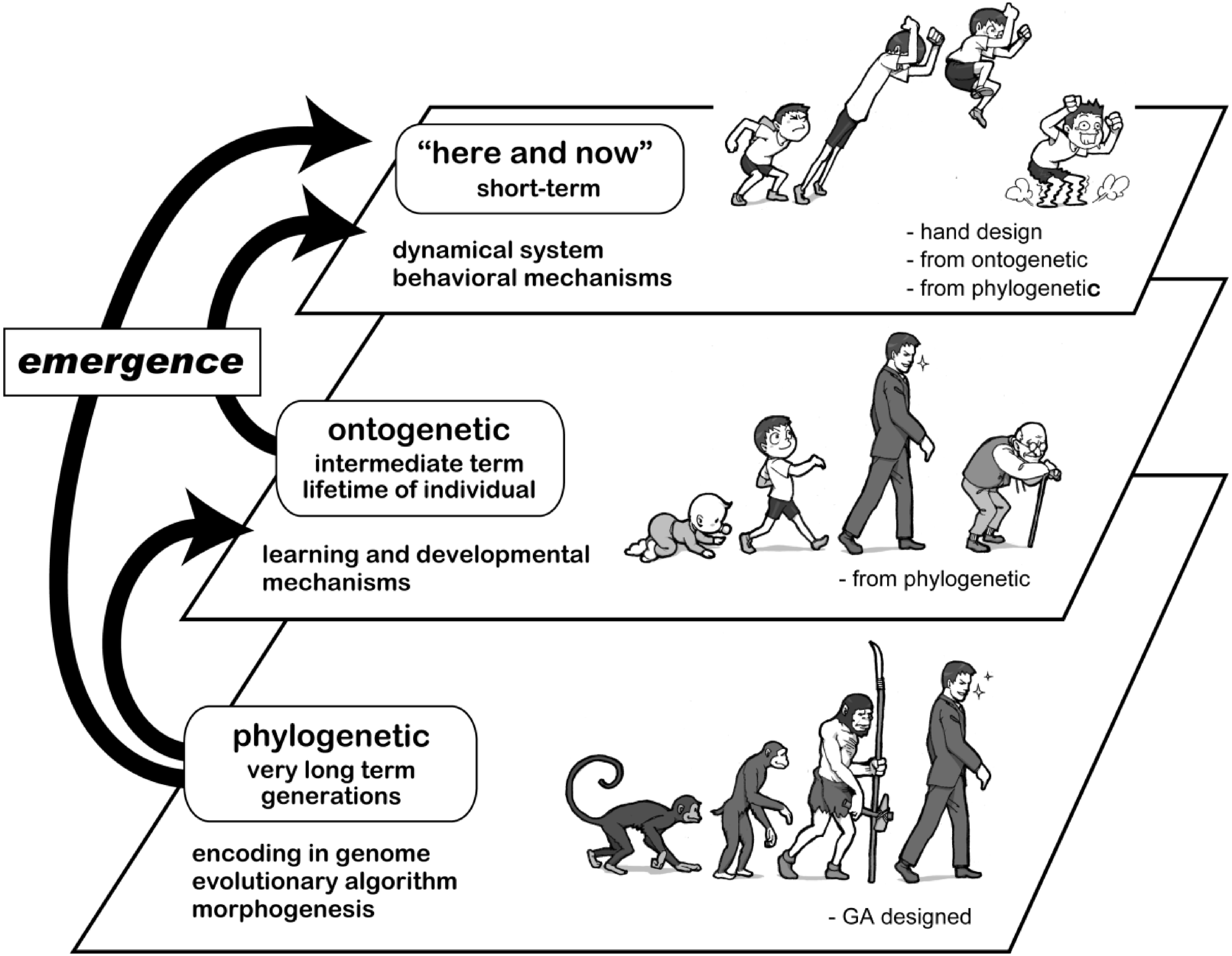


Sources:
Boston Dynamics
http://sti.epfl.ch/page-56108-en.html
https://www.youtube.com/watch?v=wAGMRQIVsf4
http://scienceblogs.com/pharyngula/2013/12/27/frugal-to-the-point-of-vacuity

A more comprehensive, bottom-up approach



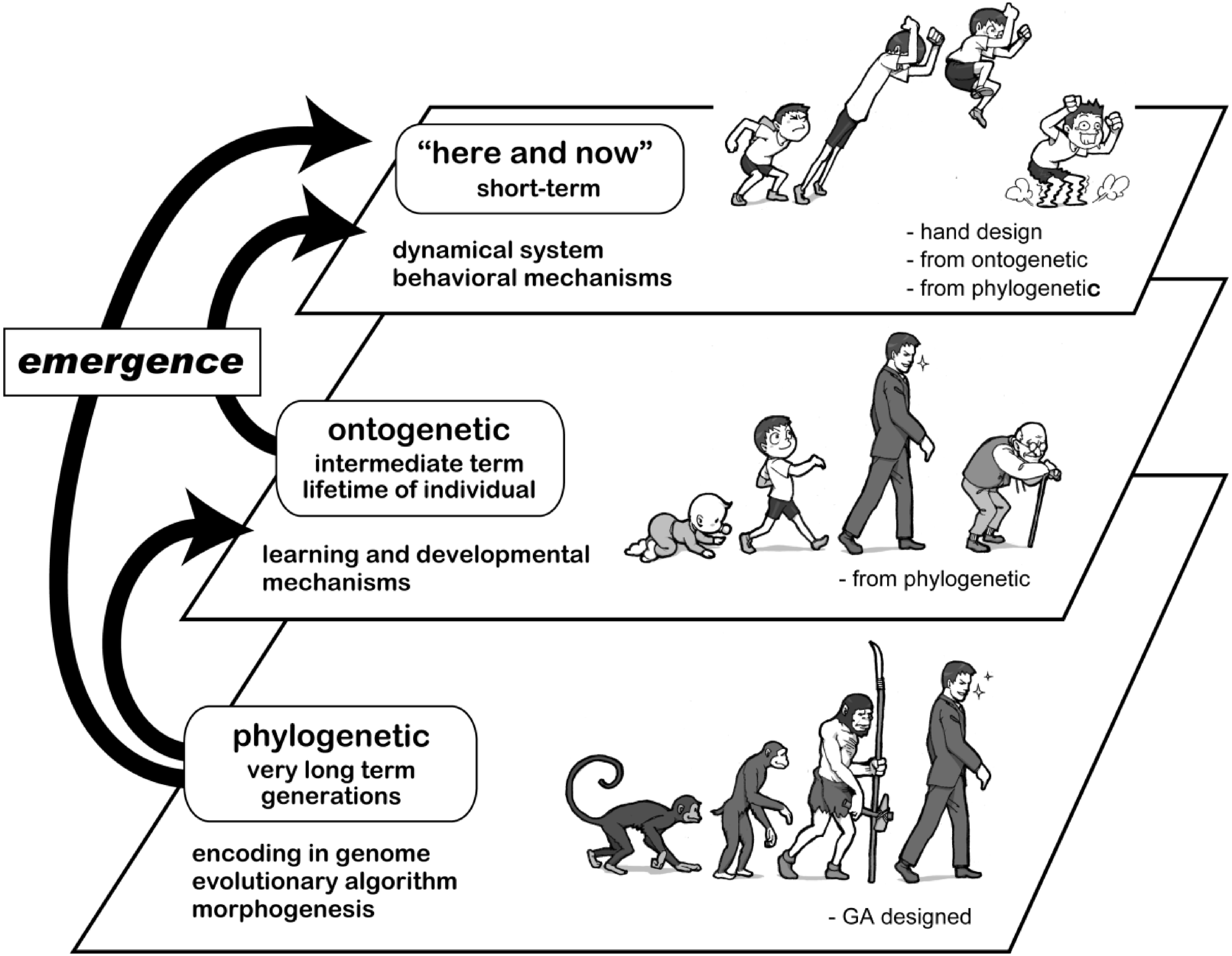
From: Pfeifer, Bongard, *How the body shapes the way we think, MIT press*



A more comprehensive, bottom-up approach

Note that mechanisms such as development and learning are themselves a product of evolution
→ Stepping stone

From: Pfeifer, Bongard, *How the body shapes the way we think, MIT press*



A more comprehensive, bottom-up approach

Note that mechanisms such as development and learning are themselves a product of evolution
→ Stepping stone

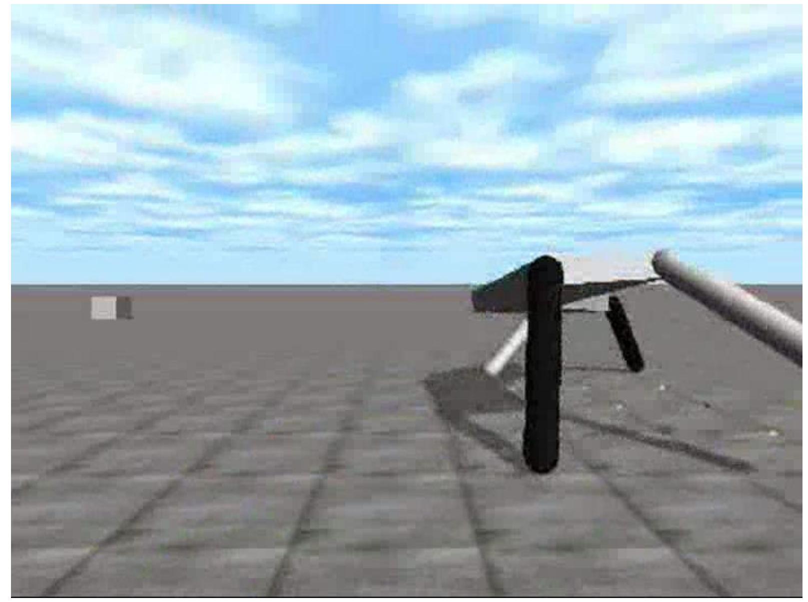
Actually, the mechanisms at the base of evolution are, somehow, themselves a product of evolution!

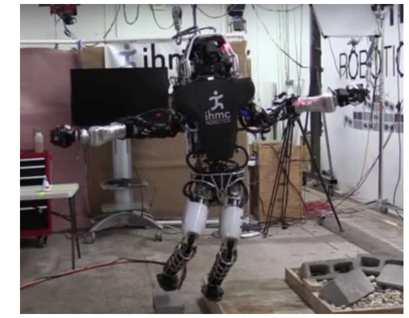
From: Pfeifer, Bongard, *How the body shapes the way we think, MIT press*

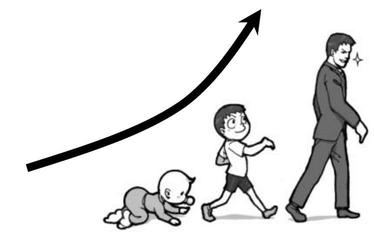
A bottom-up perspective can simplify hard problems

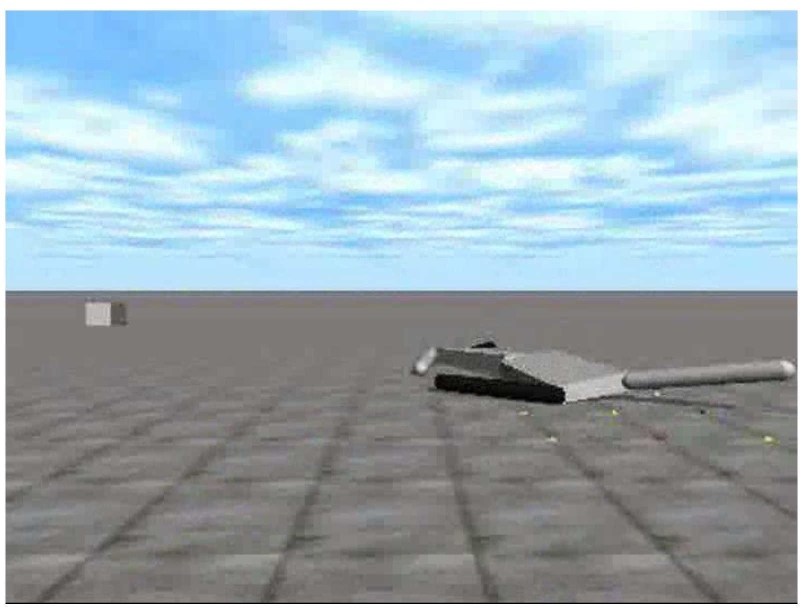
- Physically realistic virtual environment
- <u>Robot</u>: rigid segments
- <u>Control</u>: Continuous Time Recurrent Neural Network (CTRNN)

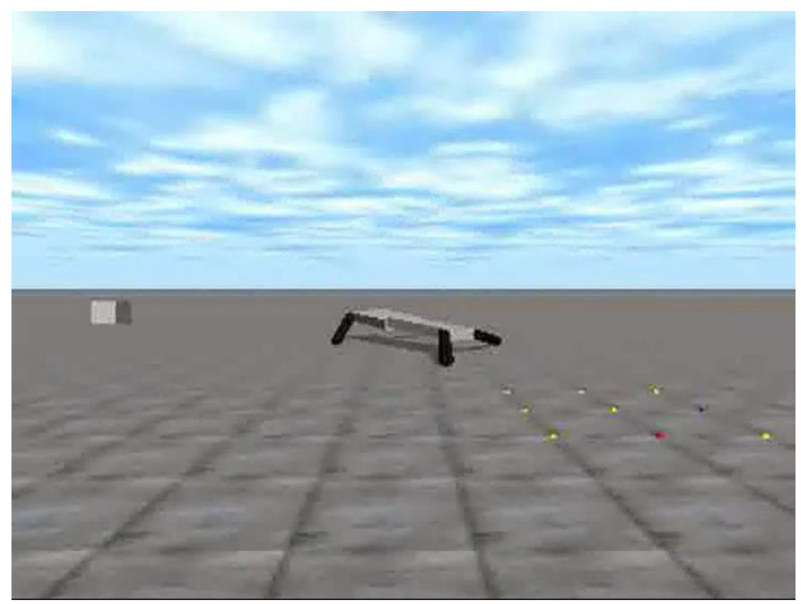
→ ***Morphological scaffolding*** (or, *training wheels* for your robot)

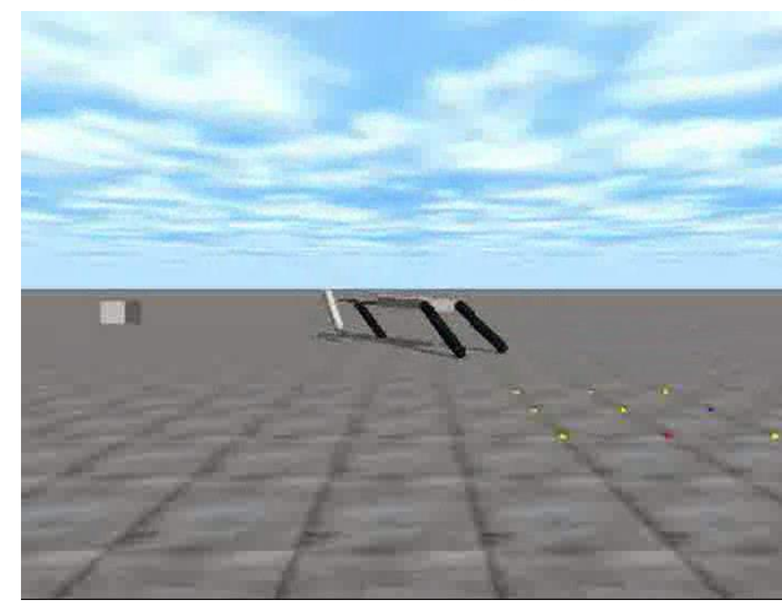






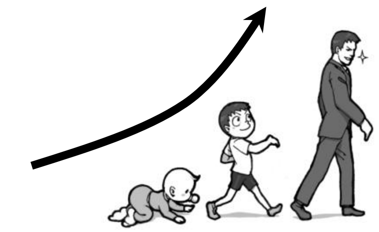






Bongard, Josh. "Morphological change in machines accelerates the evolution of robust behavior." *Proceedings of the National Academy of Sciences* 108.4 (2011): 1234-1239.

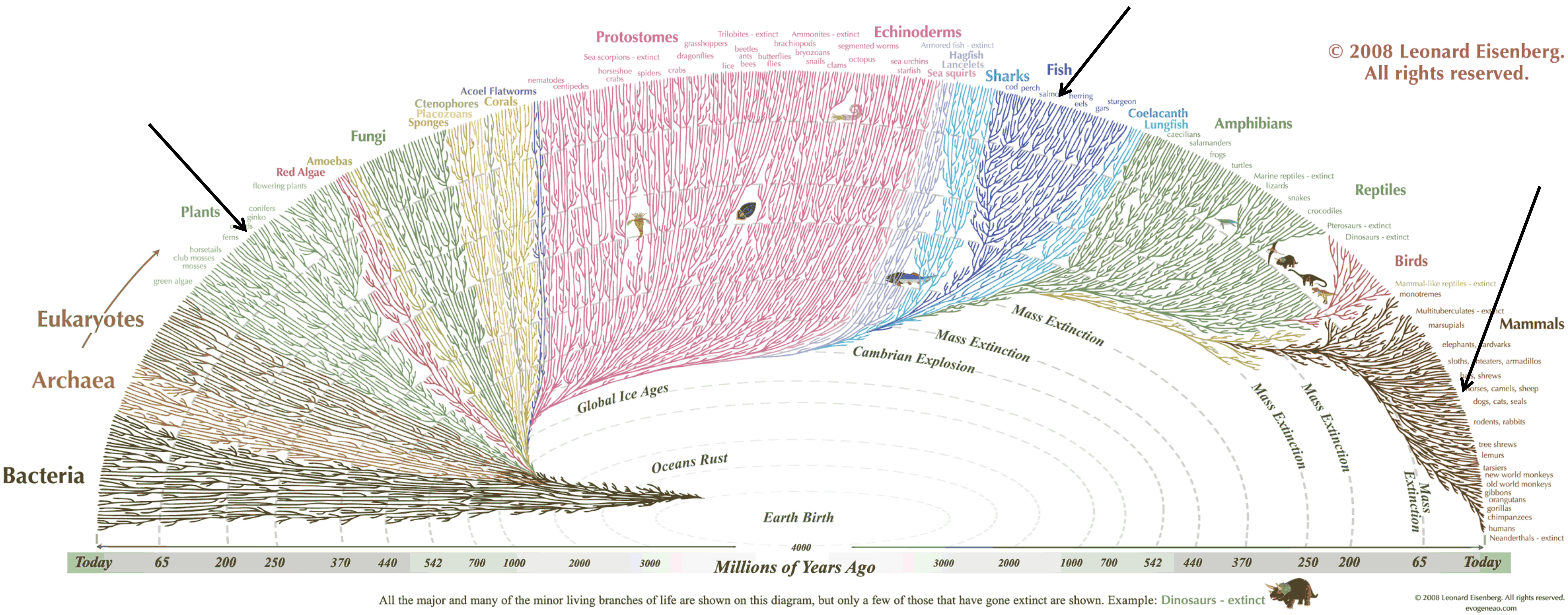
A bottom-up perspective can simplify hard problems



→ ***Morphological scaffolding* allows to find faster and more robust gaits**

Bongard, Josh. "Morphological change in machines accelerates the evolution of robust behavior." *Proceedings of the National Academy of Sciences* 108.4 (2011): 1234-1239.

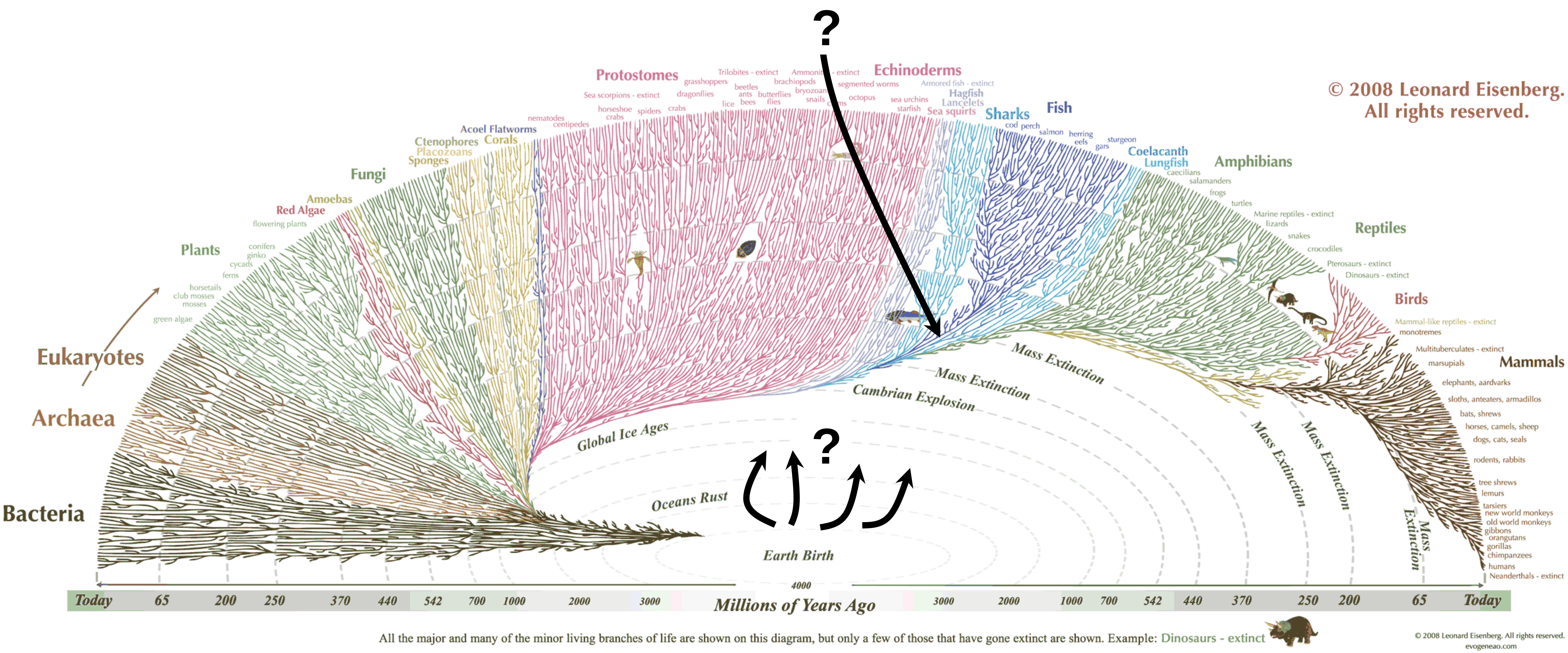
Biorobotics and biology look at a single instance of natural processes



But:

- The tips of this tree are <u>not necessarily optimal</u> (evolutionary <u>vestiges</u> and <u>compromises</u>)
- We <u>can only observe and manipulate certain aspects of living systems</u>
- *«Fossils tell no tales», John Long,* most <u>features of extinct creatures can only be guessed</u> from indirect observations

Biorobotics and biology look at a single instance of natural processes



What if we could:

- Tame and steer natural processes at our leisure
- Look at many possible instances of such processes
- Have complete access to evolving individuals, and complete control over the environment in which they evolve

A more fundamental, bottom up approach to bioinspiration

Why not focusing on natural processes (evolution, development) instead of on their products?

Why not building up complexity (morphological, neurological) in a bottom-up fashion, only when and where needed?

Overall, instead of imitating nature's designs…

…why not imitating nature's approach to design?

“*If you wish to make an apple pie from scratch you must first invent the universe*”

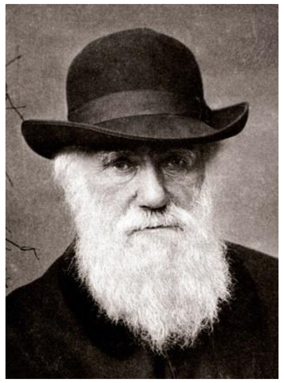
Carl Sagan, astronomer, cosmologist, astrophysicist, astrobiologist

“*Nothing in Biology Makes Sense Except in the Light of Evolution*”

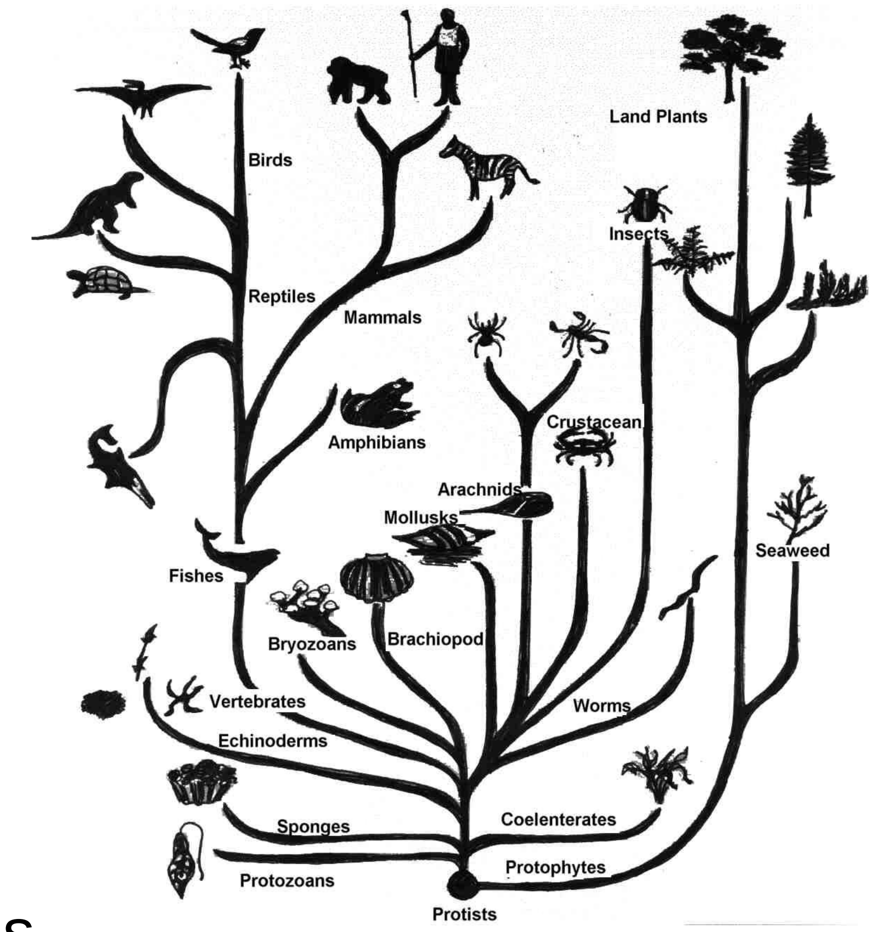
Theodosius Dobzhansky, geneticist and evolutionary biologist

Evolution: Nature's approach to design

Natural Evolution



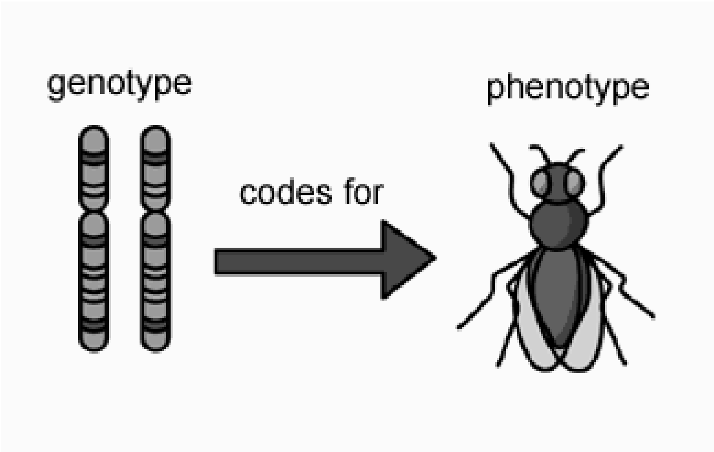
"All species derive from a common ancestor", Charles Darwin, "On the Origins of Species", 1859



The four pillars of Evolution:

1. **Population**: Evolution is based on <u>groups of individuals</u>
2. **Diversity**: Individuals in a population have <u>different characteristics</u> / <u>traits</u>
3. **Heredity**: The characteristics of an individual can be <u>transmitted over generations</u> through <u>reproduction</u>. Mechanisms involved in this process are <u>error-prone</u> → <u>Novel traits</u> can arise from <u>random</u> variations
4. **Selection**: Limited resources in the environment → <u>Not all individuals will survive and reproduce.</u> Better individuals (food gathering, mating) → Higher chance to survive and reproduce → Higher chance to find their characteristics in later generations → <u>Useful traits become more frequent</u> (innovation)

A bit of terminology



https://epfb.wordpress.com/genotype-vs-phenotype/

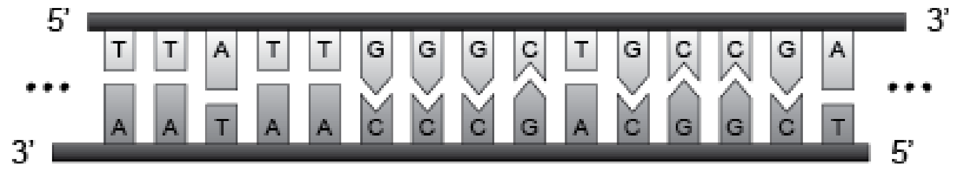
Genotype:

- "Blueprint" of an organism
- Individual's traits (observable features) are encoded there
- It is transmitted and manipulated by error-prone mechanisms (recombination, mutation) → Novel traits arise
- Through processes called translation and transcription, the genotype of an individual ultimately results in its…

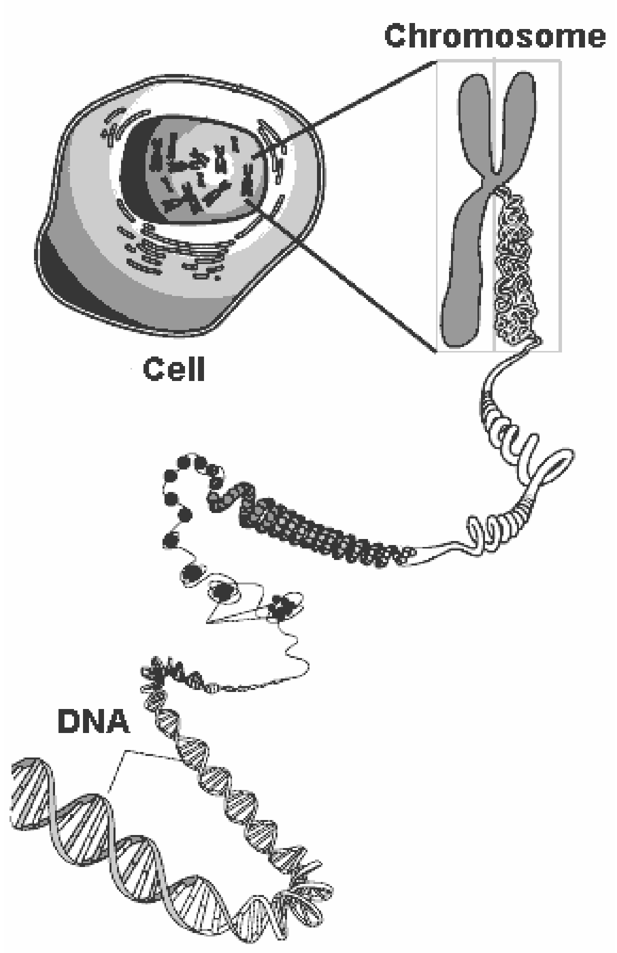
Phenotype:

- Observable features of an organism (physical appearance, behavior, …)
- Selection operates on the phenotype

A bit of terminology

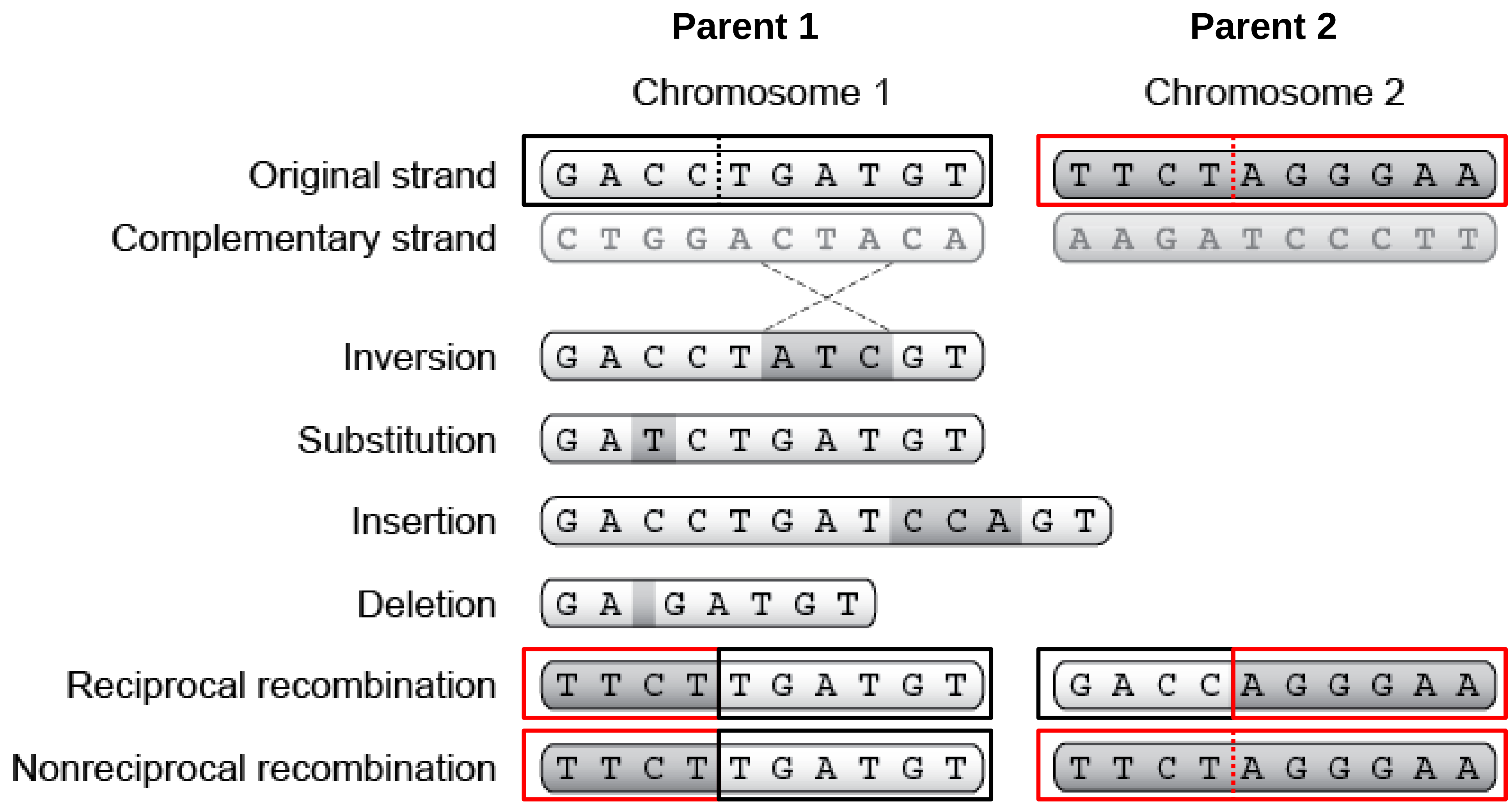


- In biology, the genetic material is based on the DNA
- DNA is organized in separated molecules → *Chromosomes*
- In sexual reproduction the genetic material of the parents is combined (*genetic recombination*)
- Genes: Functionally relevant sub-sequences of the DNA chain
- The characteristics encoded in genes ultimately result in specific phenotypic traits through a process called *gene expression*



Genetic mutation and recombination

<u>Error-prone</u> replication mechanisms
→ Novel traits arise from these <u>random variations</u>
→ Those that confer an <u>advantage</u> have more chances of being <u>selected</u>



From an algorithmic / engineering point of view…

Evolution can be thought of as a trial-and-error process, in which innovation is driven by the non-random selection (survival/reproduction) of random variations (genetic mutations)

In nature, this process is open-ended and non-goal-directed

Evolved biomechanics

Cheetah

Peregrine falcon

Manta ray

Evolution and adapation to the *ecological niche*

Adaptation to the environment: body coverings (mimicry), body parts, behaviors

Leaf-tailed gecko

Walking stick

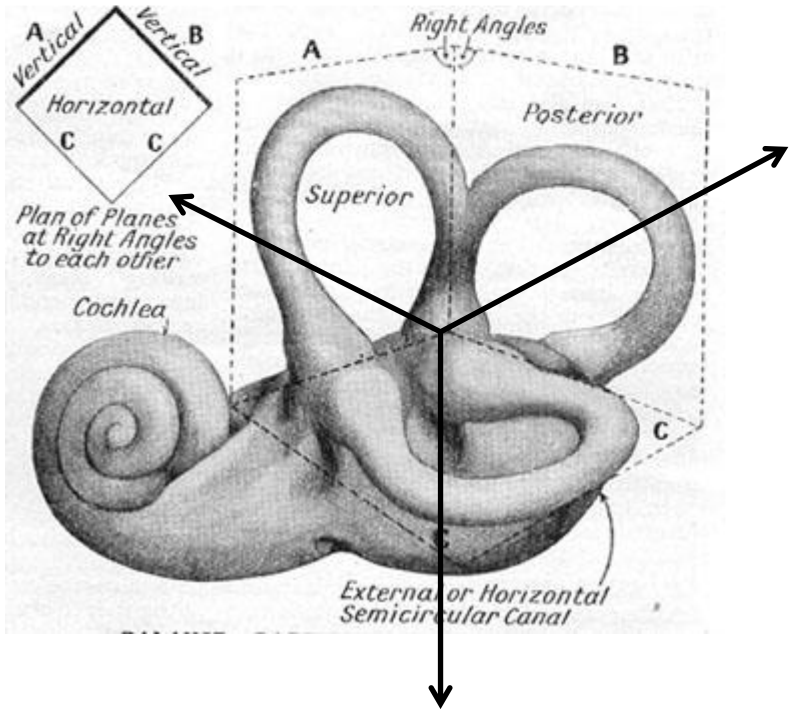
Green leaf Katydid

Chaetodon capistratus

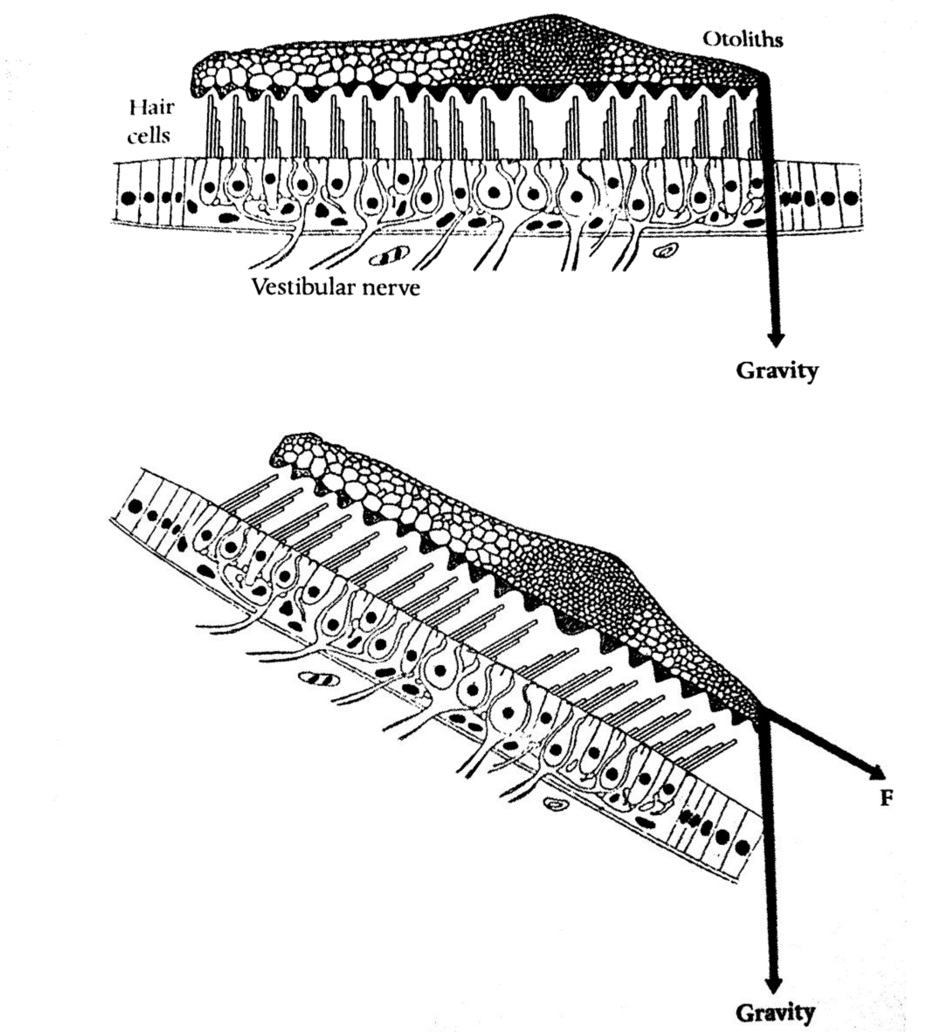
Non toxic butterfly mimics a toxic one

Evolved Sensors – vestibular system

Semicircular canals, detecting angular accelerations



Otoliths, detecting linear accelerations and tilting.
In some animals (e.g. insects) adapted to also detect vibrations

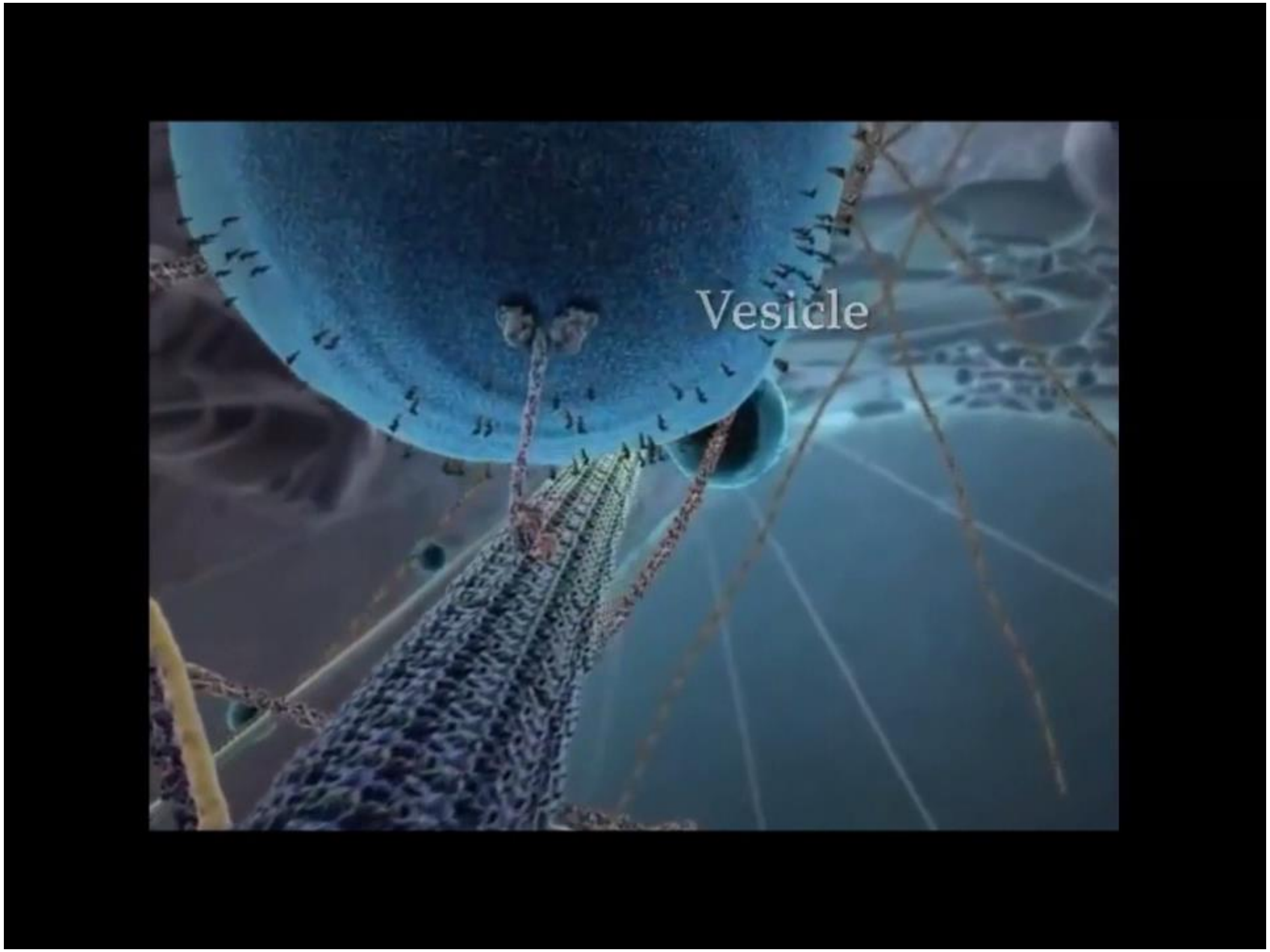


Remarkably sophisticated solutions

Evolved Complexity at the micro scale

***ATP Synthase** – a protein-based micro rotational motor*

Evolved Complexity at the micro scale



The inner life of the Cell - BioVisions, Harvard University – http://multimedia.mcb.harvard.edu

Another product of Evolution…

If we could replicate evolution in an artificial form, we could:

- Obtain a methodology to automatically design as many robots as we would like, for all possible tasks and environments

- Achieve similar levels of sophistication in the final solutions

→ Potentially outperforming human design skills

(machines are better than us already in many tasks)

→ Potentially outperforming bio-inspired designs

(artificial evolution would find a way to exploit the provided artificial substrate, i.e. our technology, instead of mimicking solutions arising from a biological one)

If we could replicate evolution in an artificial form, we could:

- Produce <u>**machines that can adapt**</u> to different tasks and environments, like the products of natural evolution (biological creatures) do → <u>Current robotic technology lacks of adaptivity</u>

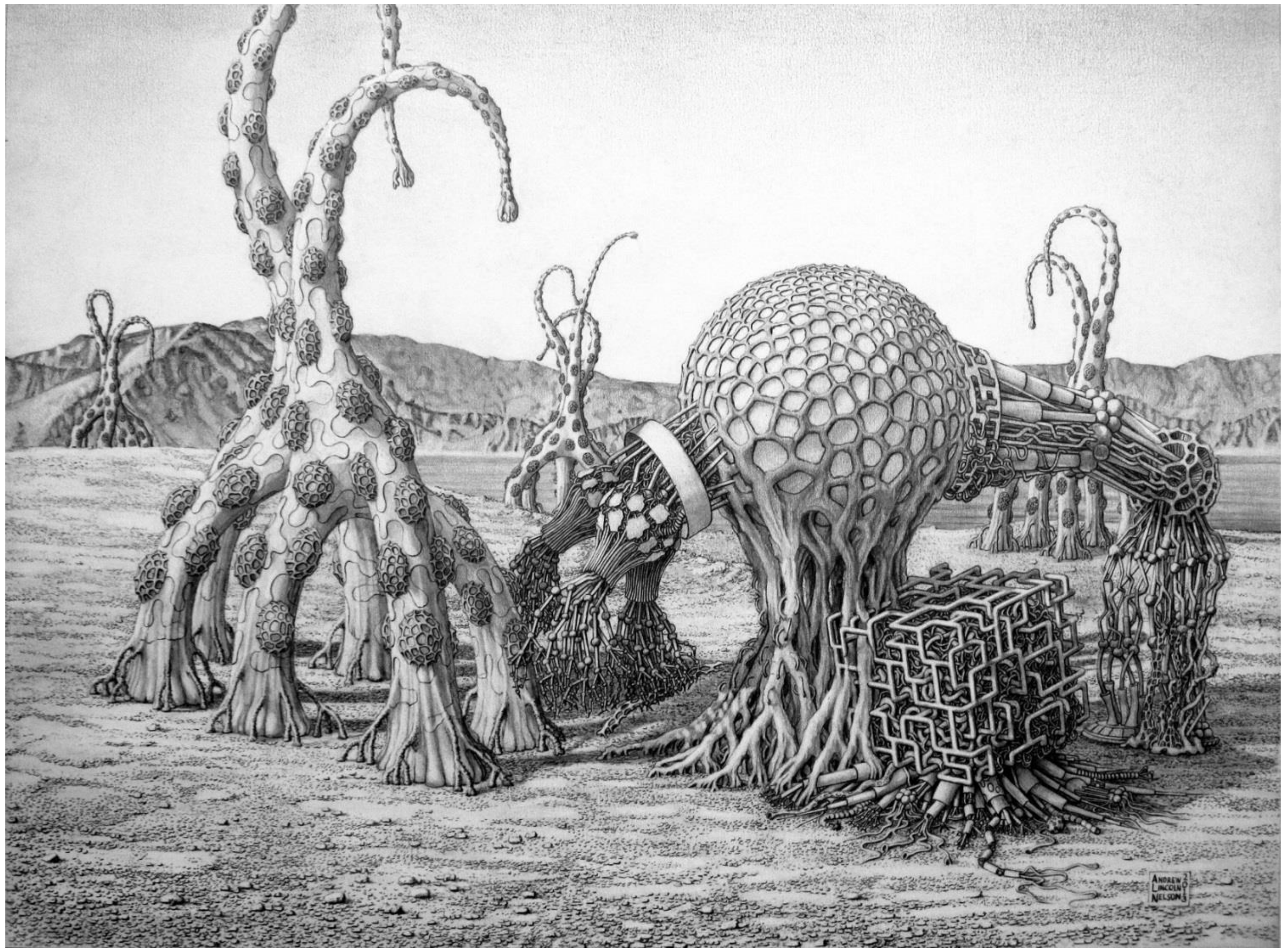
- Give rise to other desirable phenomena that natural evolution produced, that are difficult to comprehend and replicate in artificial form (e.g. <u>intelligence</u>)

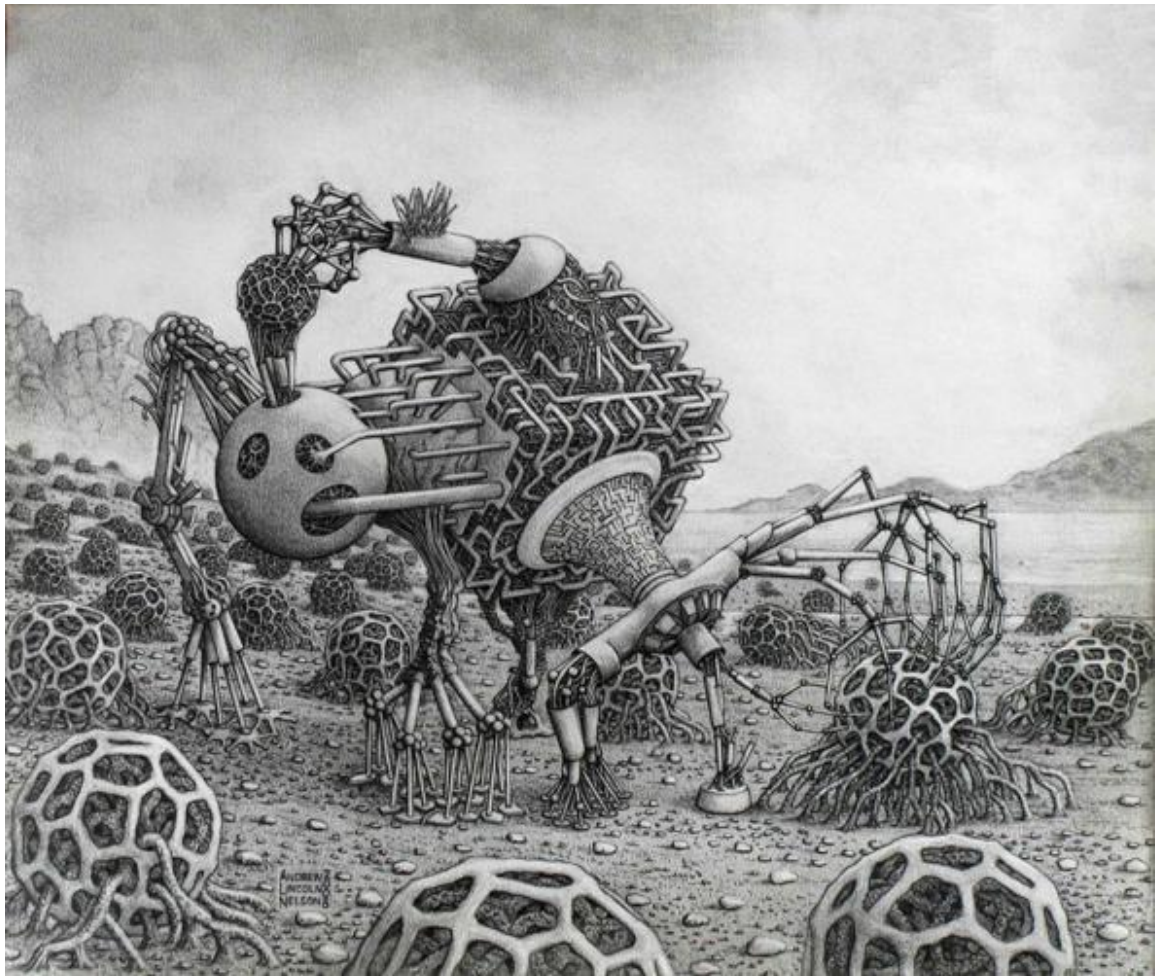
If we could replicate evolution in an artificial form, we could:

- Simulate many alternative evolutionary trajectories (e.g. What life on a different planet may look like, given the different environmental conditions?)

→ With computers we can simulate many possible worlds in a matter of hours

Christopher Langton, **Artificial Life (ALife)**:
study of life as it is… and as it could be





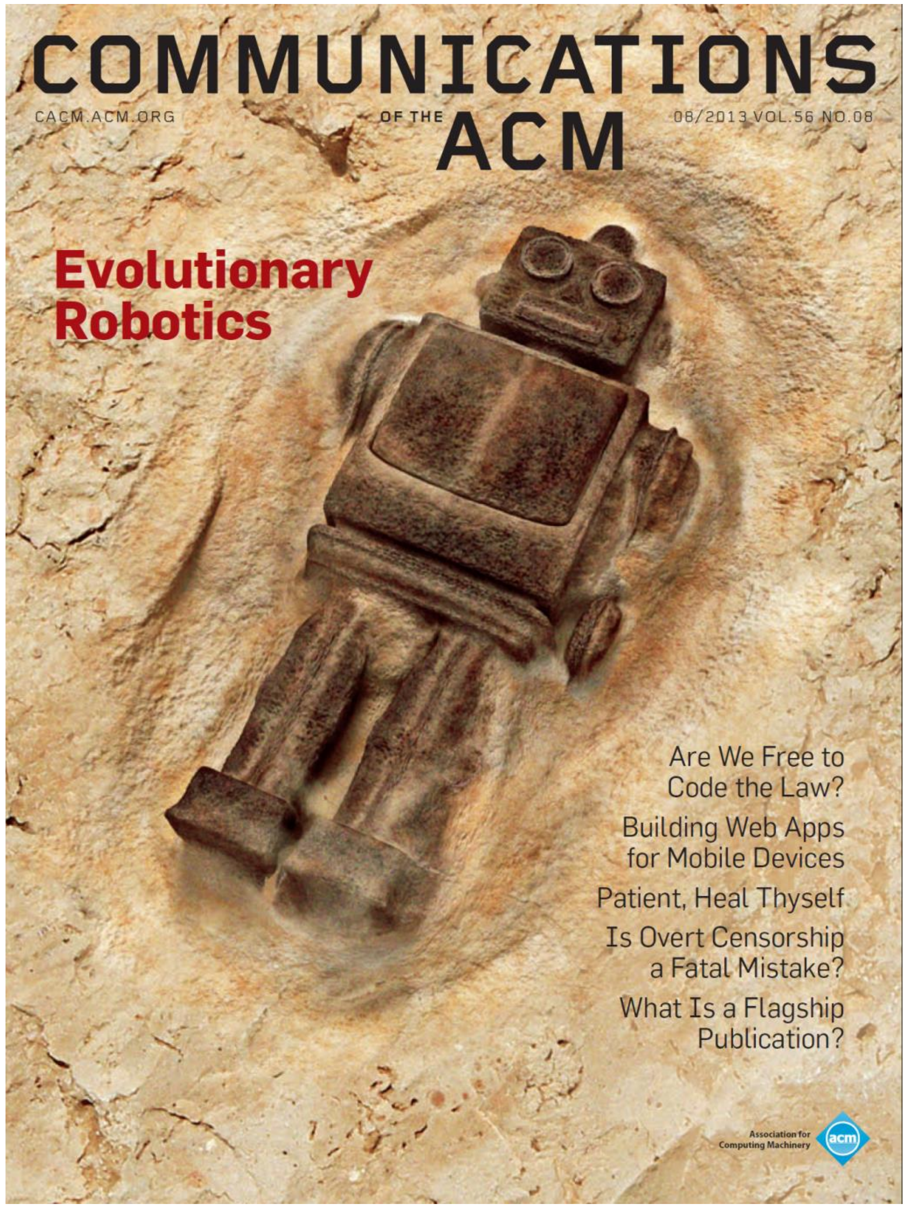
www.evolutionaryrobotics.org

Evolutionary robotics

What is Evolutionary Robotics?

Evolutionary Robotics is an interdisciplinary research field, at the intersection of:

- Robotics
- Artificial intelligence
- Cognitive sciences
- Computational and evolutionary biology
- Artificial Life
- …



Josh C. Bongard. 2013. "Evolutionary robotics". *Commun. ACM* 56, 8 (August 2013), 74-83

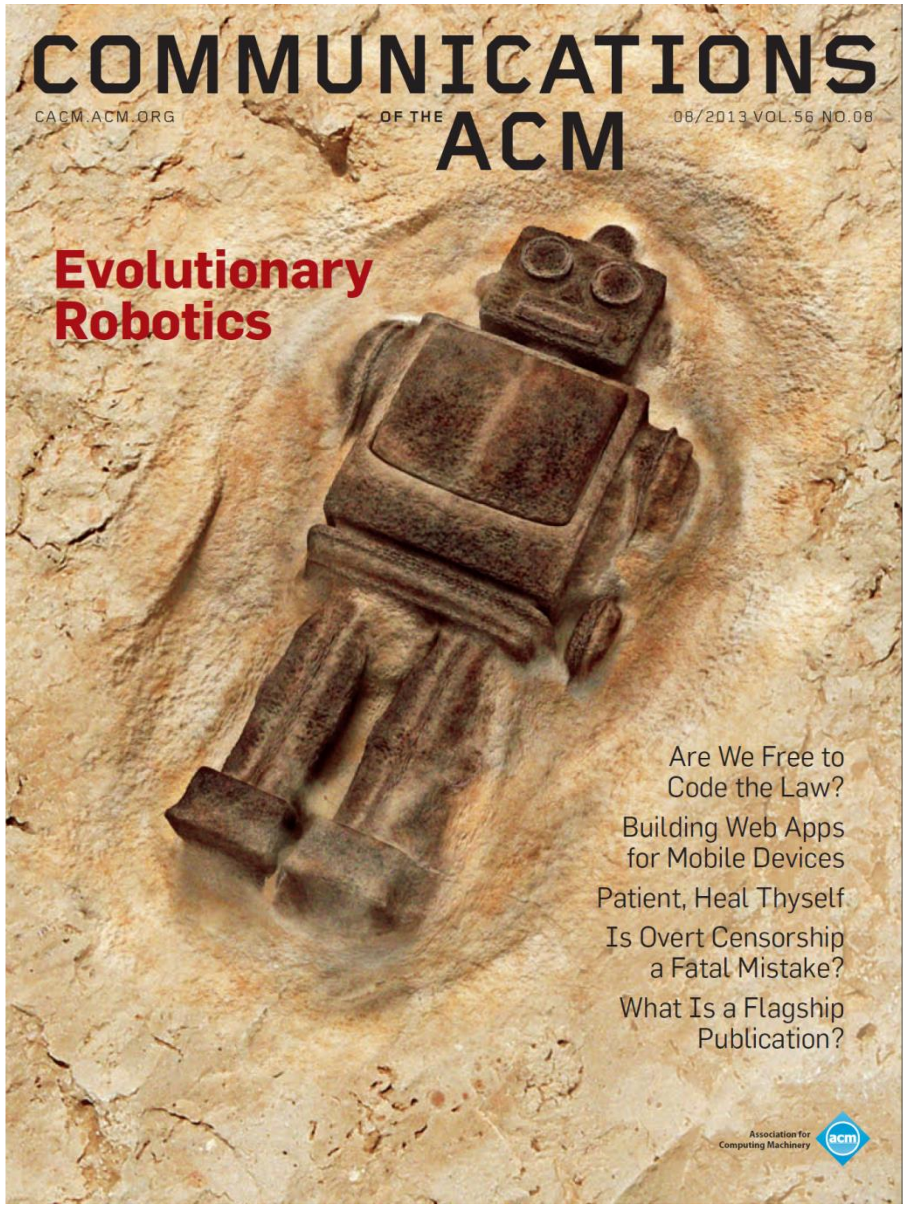
What is Evolutionary Robotics?

Core idea:

To apply optimization algorithms inspired by natural evolution in order to automatically design complete, adaptive and intelligent machines

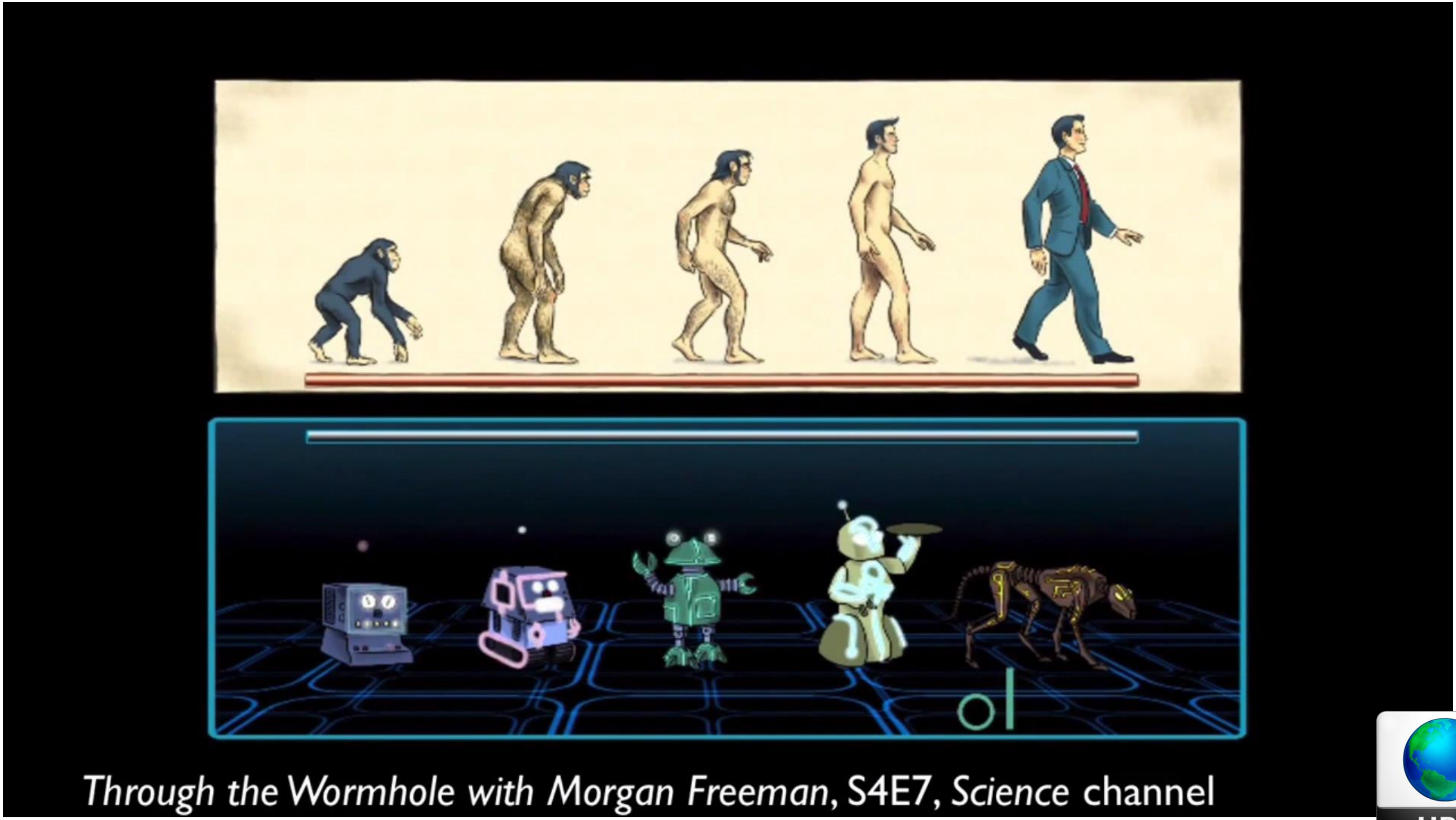
→ Paradigm shift:

Replicating natural *processes* instead of their *end-products*



Josh C. Bongard. 2013. "Evolutionary robotics". *Commun. ACM* 56, 8 (August 2013), 74-83

What is Evolutionary Robotics?



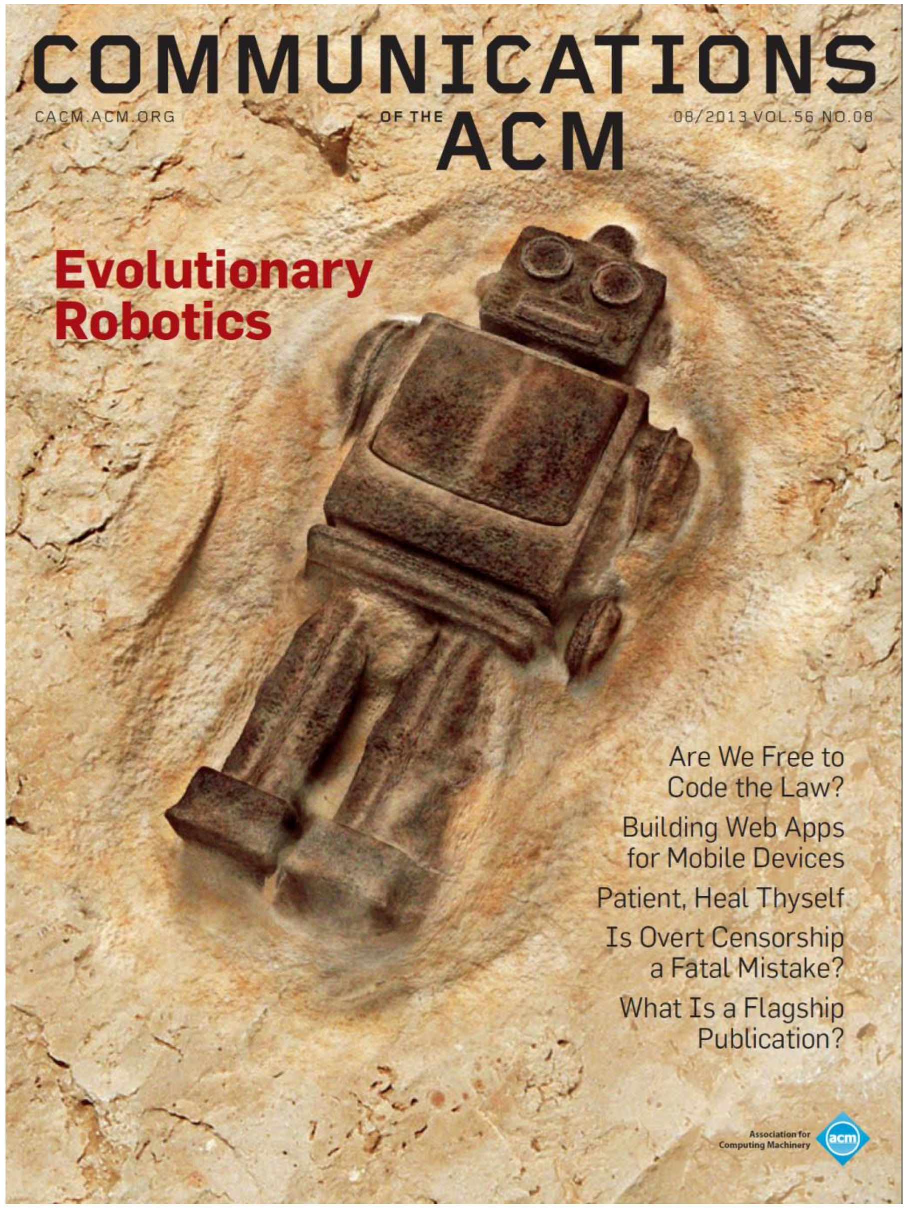
URL

What is Evolutionary Robotics?

Core idea:

To apply **optimization algorithms** inspired by natural evolution in order to *automatically* design *complete*, *adaptive* and *intelligent* machines

Although natural evolution does not have a goal, in evolutionary robotics **we usually have one**, which is formulated in the form of an **optimization problem**



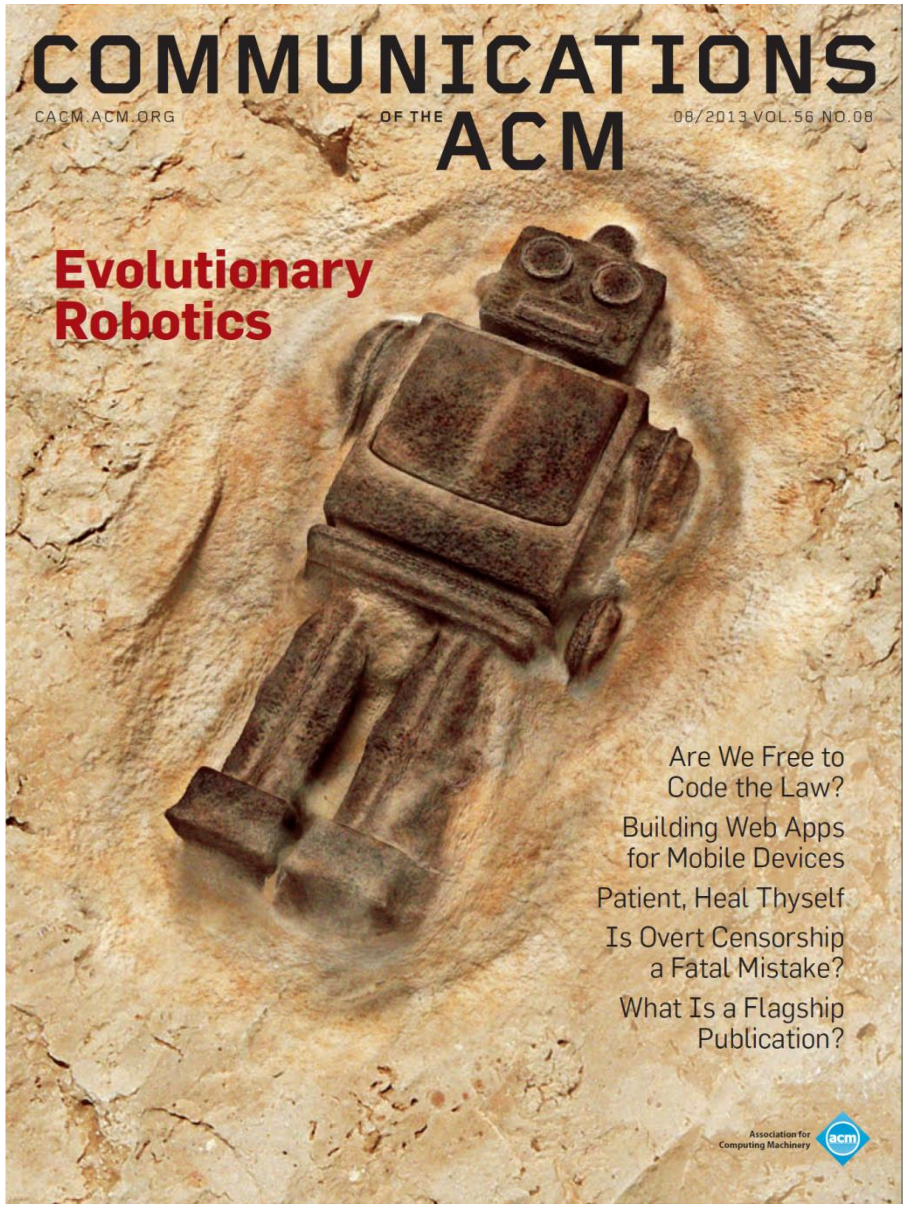
Josh C. Bongard. 2013. “Evolutionary robotics”. *Commun. ACM* 56, 8 (August 2013), 74-83

What is Evolutionary Robotics?

Core idea:

To apply **optimization algorithms** inspired by natural evolution in order to *automatically* design *complete*, *adaptive* and *intelligent* machines

Example: find the optimal morphology and controller for fast locomotion in a given environment

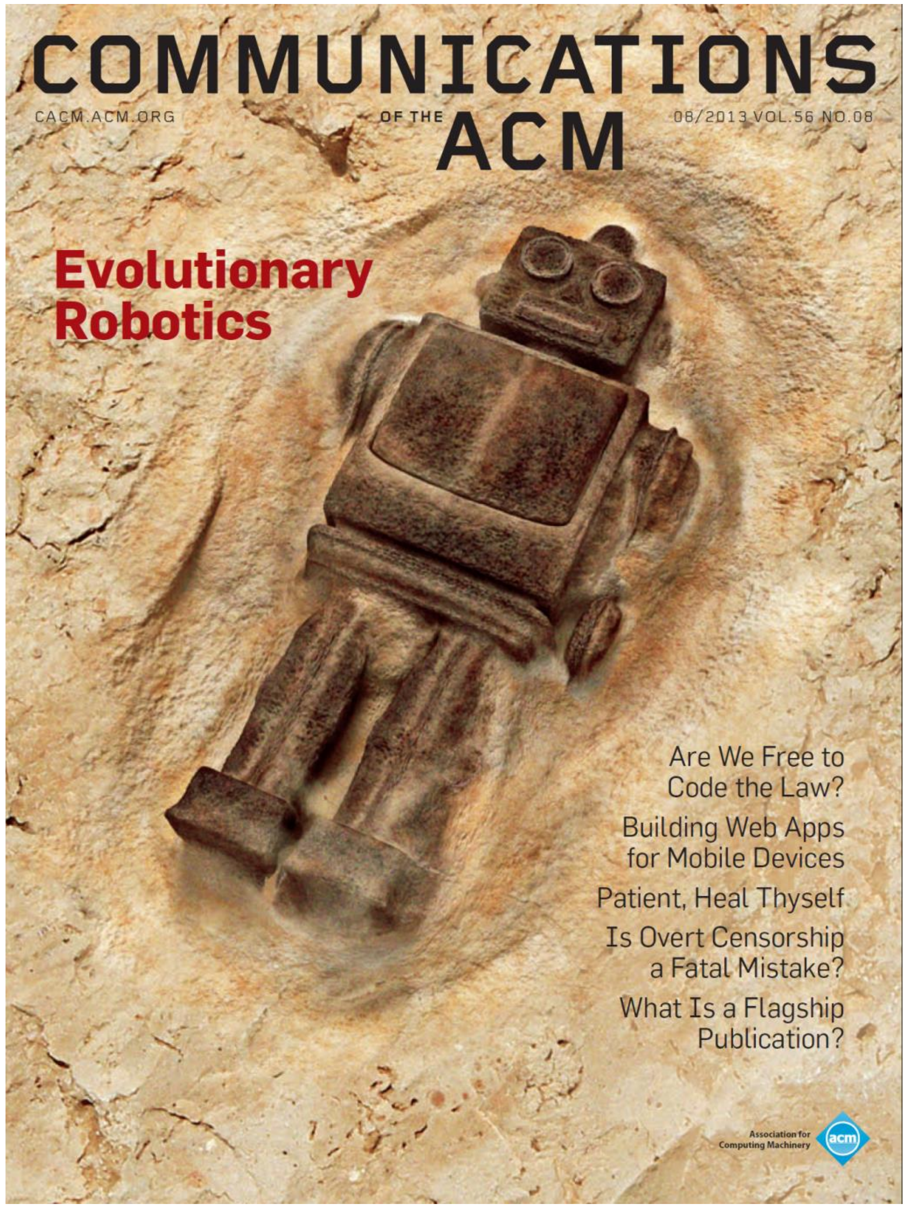


Josh C. Bongard. 2013. "Evolutionary robotics". *Commun. ACM* 56, 8 (August 2013), 74-83

What is Evolutionary Robotics?

Core idea:

To apply optimization algorithms inspired by **natural evolution** in order to *automatically* design *complete*, *adaptive* and *intelligent* machines



Josh C. Bongard. 2013. "Evolutionary robotics". *Commun. ACM* 56, 8 (August 2013), 74-83

Darwinian evolution:

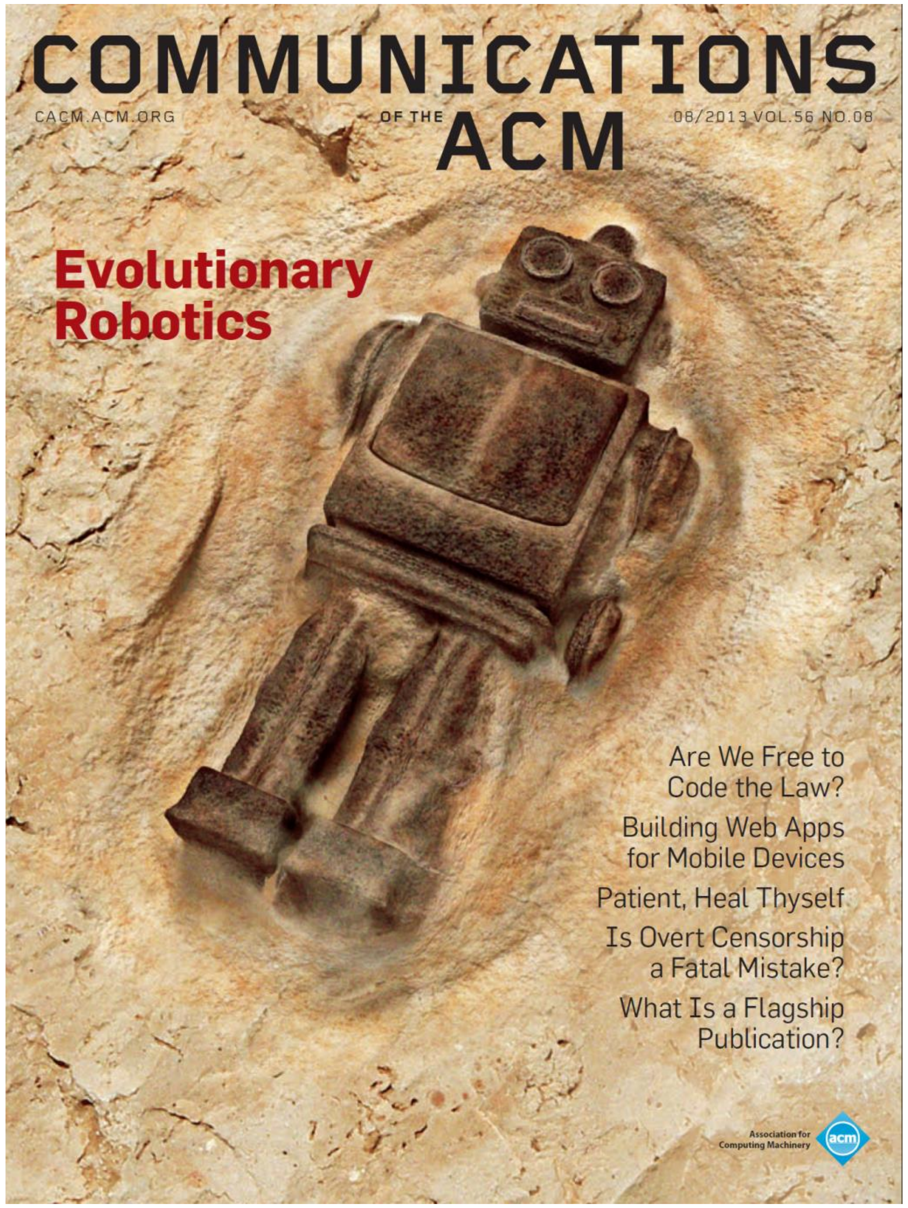
1. Descent with modification
2. Natural selection

What is Evolutionary Robotics?

Core idea:

To apply optimization algorithms inspired by natural evolution in order to ***automatically* design** *complete*, *adaptive* and *intelligent* machines

Design automation technique: once we set up the process, it requires no human intervention
→ When coupled with techniques such as 3D printing, potential for a completely automated design and fabrication pipeline



Josh C. Bongard. 2013. "Evolutionary robotics". *Commun. ACM* 56, 8 (August 2013), 74-83

What is Evolutionary Robotics?

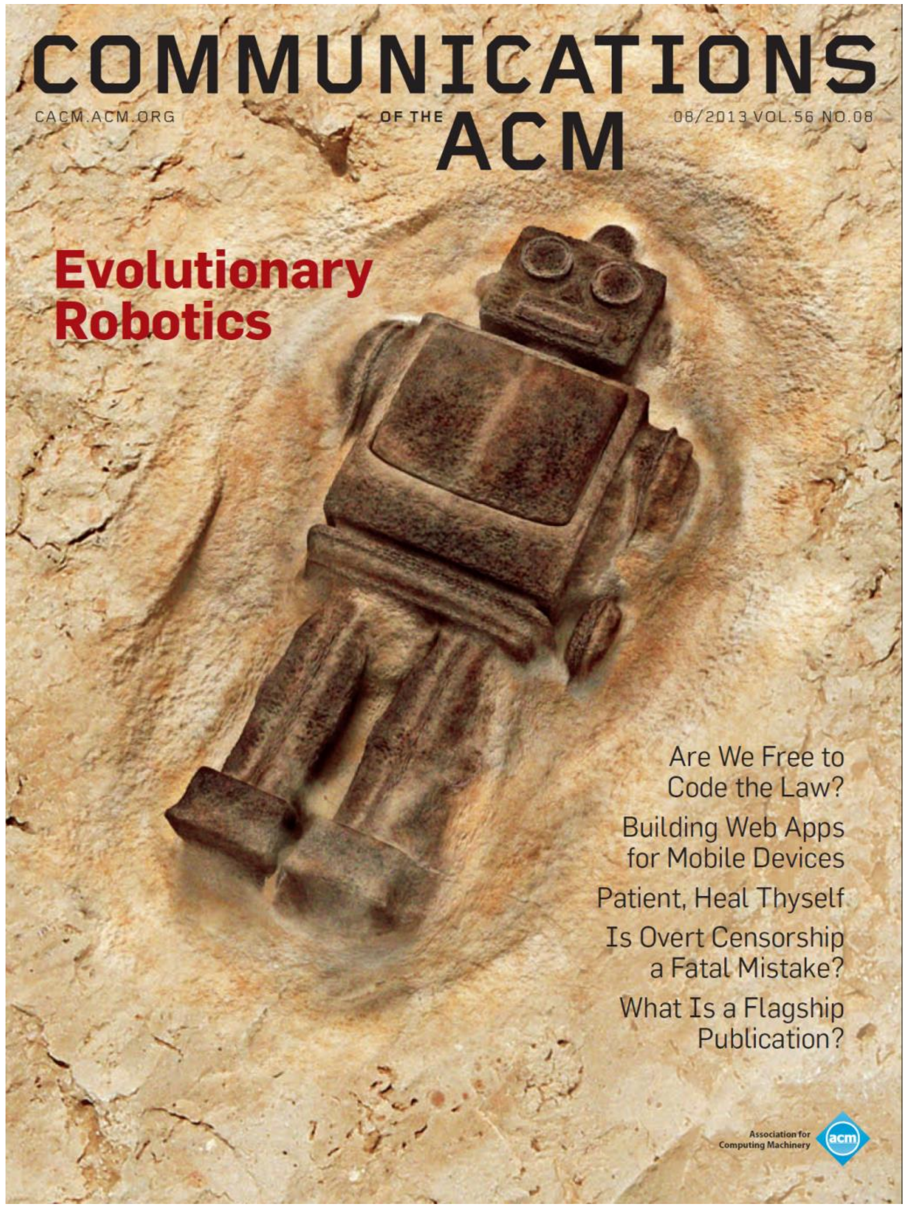
Core idea:

To apply optimization algorithms inspired by natural evolution in order to *automatically* design ***complete***, *adaptive* and *intelligent* machines

All aspects of a robot can be co-optimized at once:

1. **Morphology** (remember the passive dynamic walker)
2. Sensory and actuation systems
3. Controller

→ **Unique advantage of this type of technique**



Josh C. Bongard. 2013. "Evolutionary robotics". *Commun. ACM* 56, 8 (August 2013), 74-83

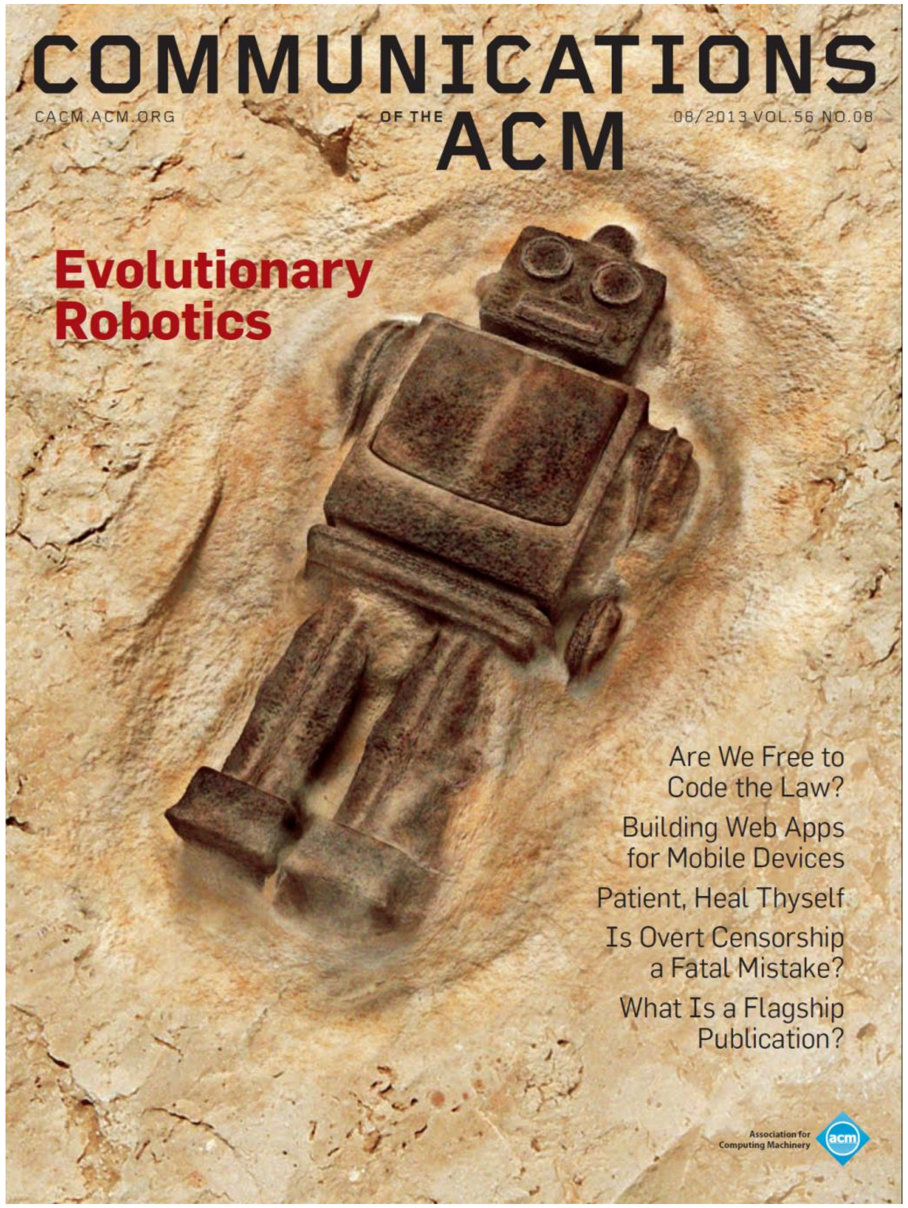
What is Evolutionary Robotics?

Core idea:

To apply optimization algorithms inspired by natural evolution in order to *automatically* design *complete*, ***adaptive*** and *intelligent* machines

The lack of adaptation of current robotic technology is a limiting factor to the widespread of robotics outside controlled environments

→ Evolving robots can adapt to unknown and possibly dynamic environments



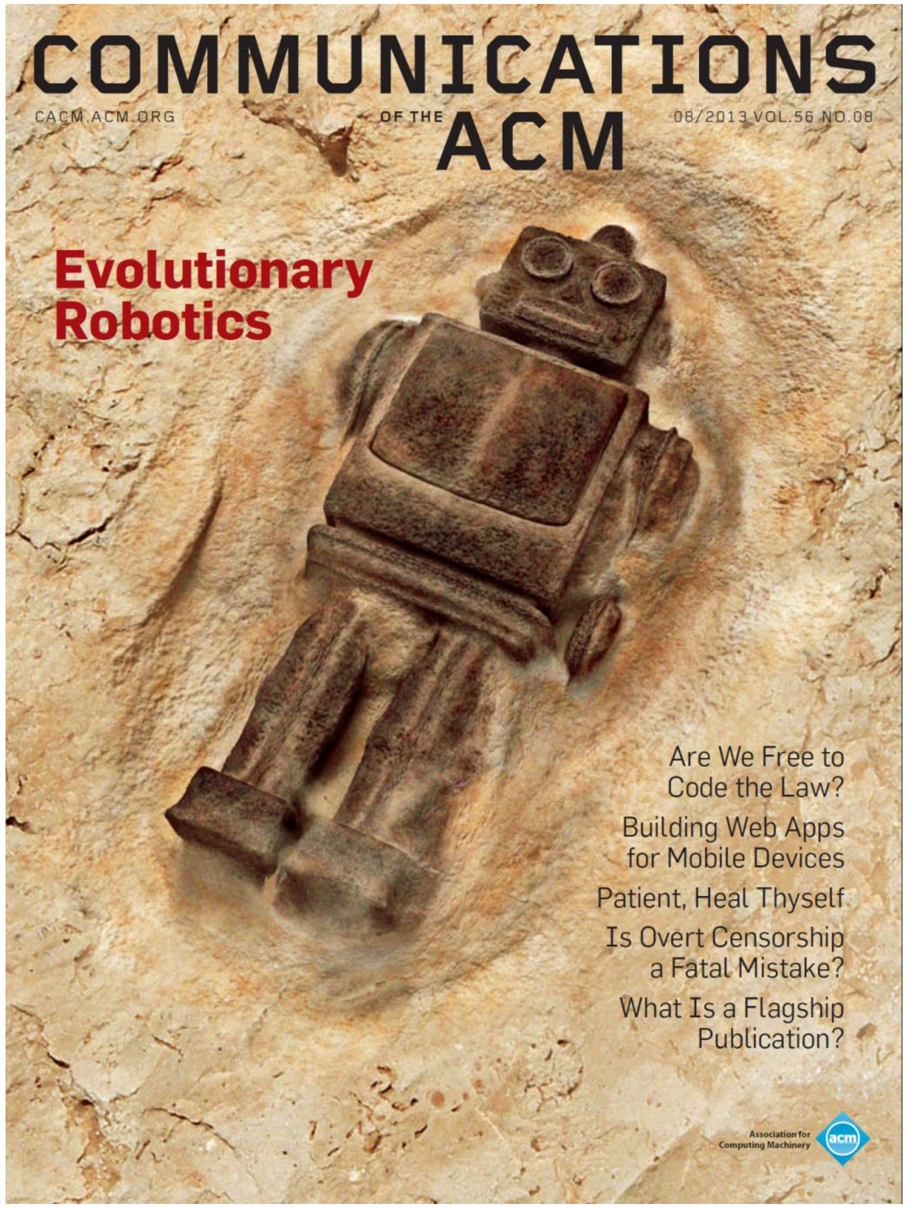
Josh C. Bongard. 2013. "Evolutionary robotics". *Commun. ACM* 56, 8 (August 2013), 74-83

What is Evolutionary Robotics?

Core idea:

To apply optimization algorithms inspired by natural evolution in order to *automatically* design *complete*, *adaptive* and ***intelligent*** machines

Biological intelligence was created by natural evolution: artificial evolution may be the most reasonable path to produce truly intelligent and cognitive machines



Josh C. Bongard. 2013. “Evolutionary robotics”. *Commun. ACM* 56, 8 (August 2013), 74-83

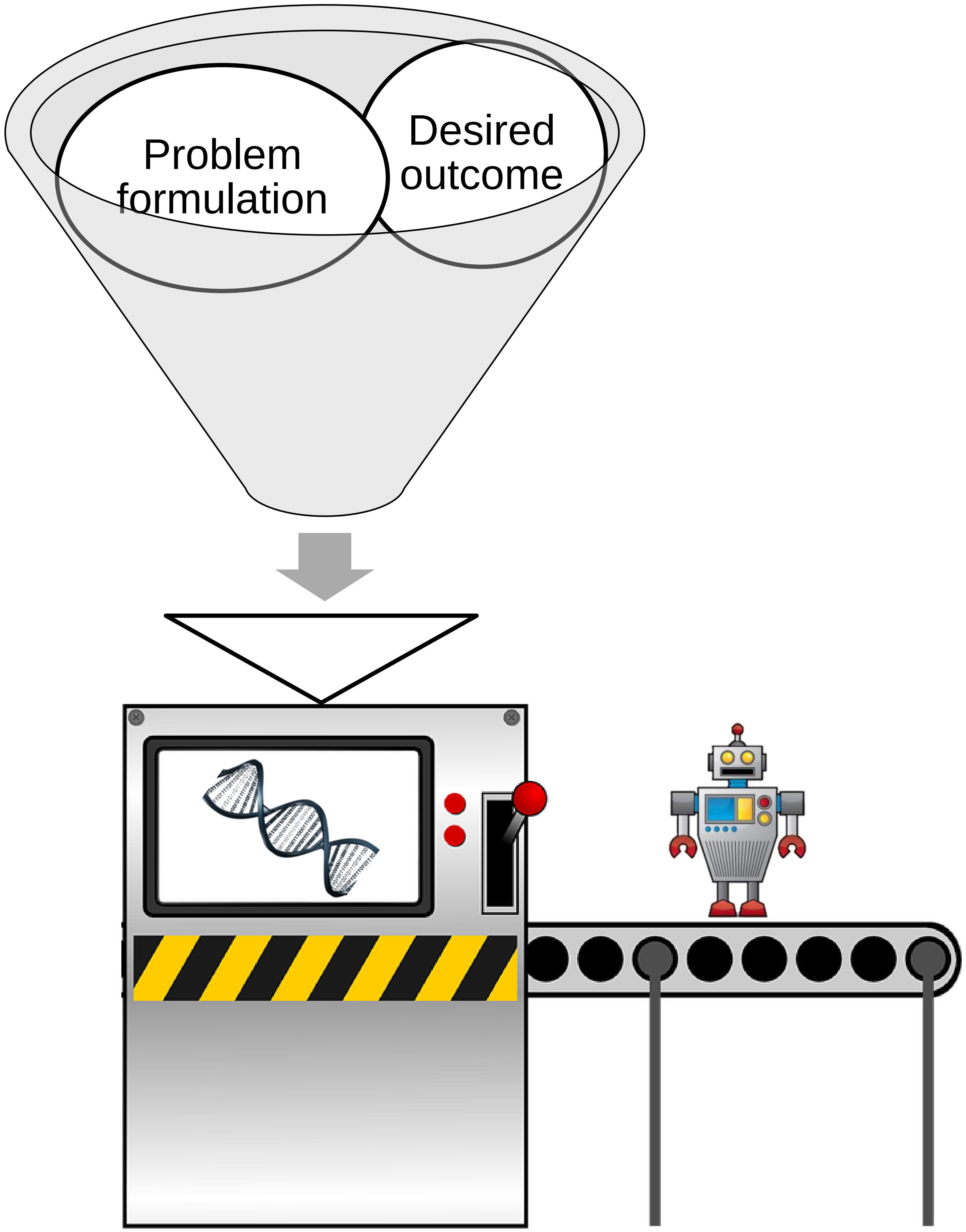
Anatomy of an Evolutionary Robotics experiment

Main components:

1. An *environment* (real/virtual)
2. A *task* that we want the robot to solve (e.g. walk)

   → 1+2: *"task environment"*

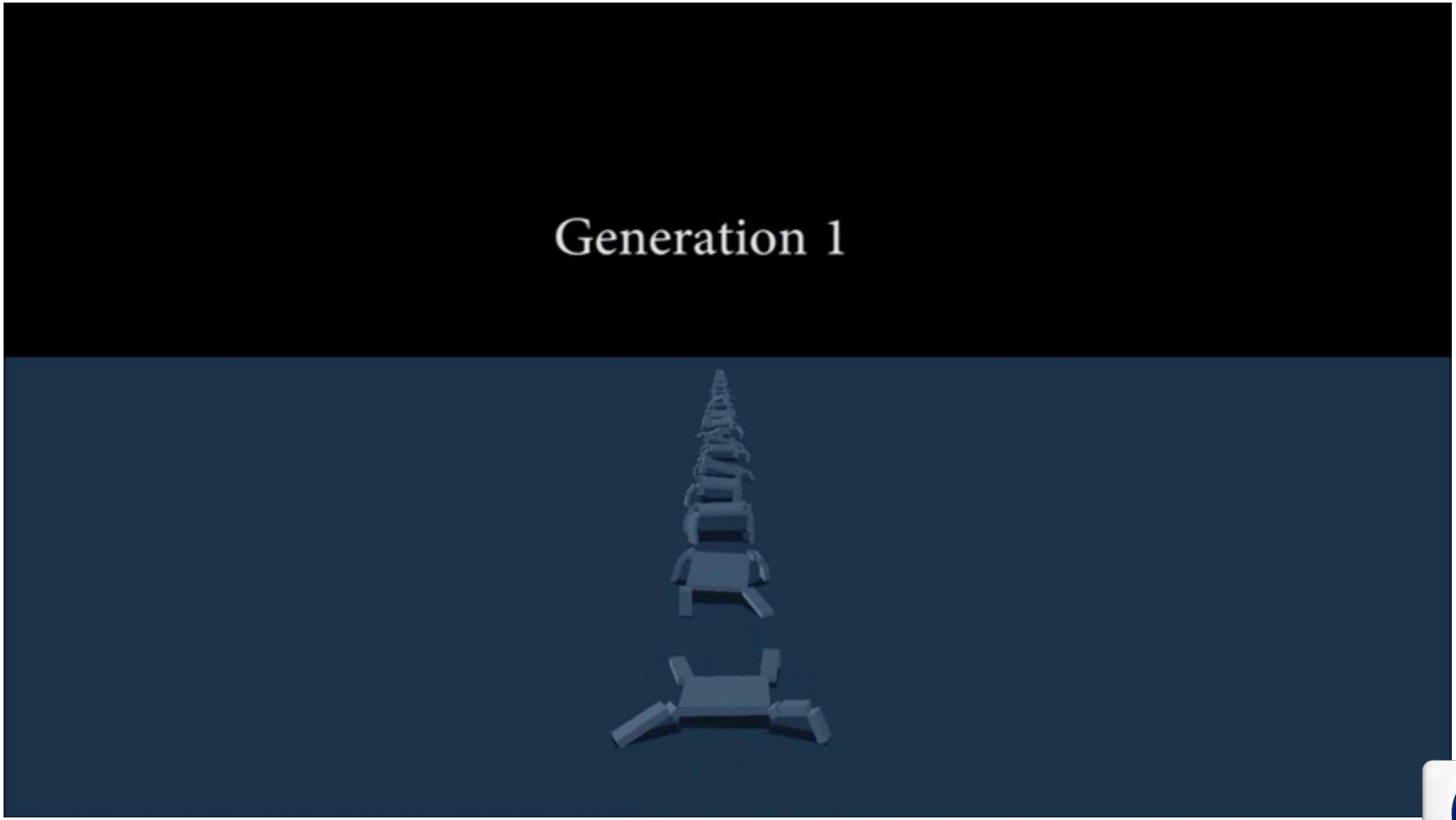
3. A *robot* (real/simulated), or (usually) a *population* of them, some aspects of which (e.g. morphology, control, both) should be optimized (→ they will be under evolutionary control)
4. A *fitness function*, measuring how well each robot performs (e.g. speed)
5. An *evolutionary algorithm*, trial-and error iterative procedure which optimizes the robot(s) over a number of *generations*, until a desired solution is found


Problem formulation
Desired outcome

The first attempt

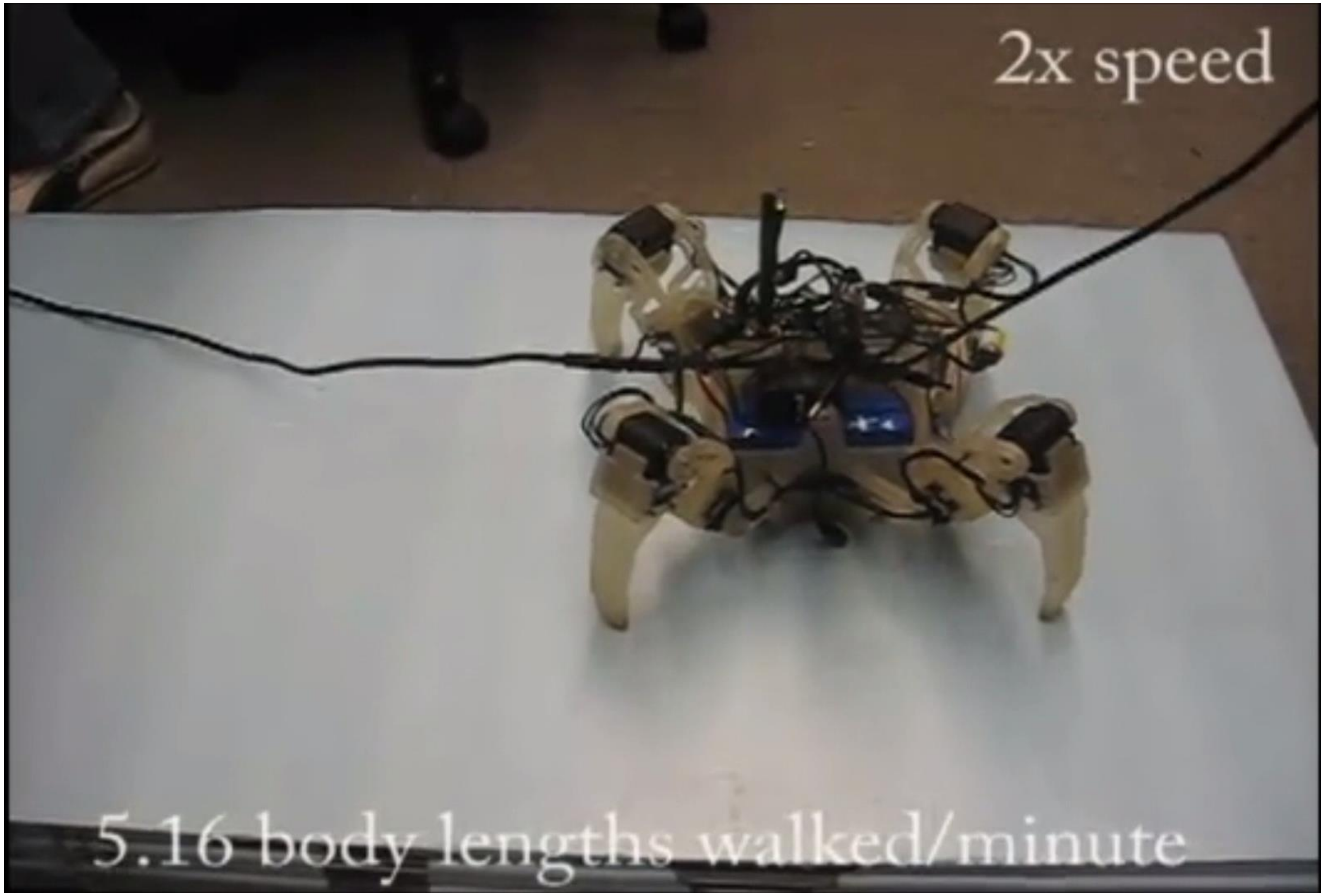
Sims, Karl. "Evolving virtual creatures." *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 1994.

Example I



From: YouTube (Arseniy Nikolaev, virtual spiders evolution)

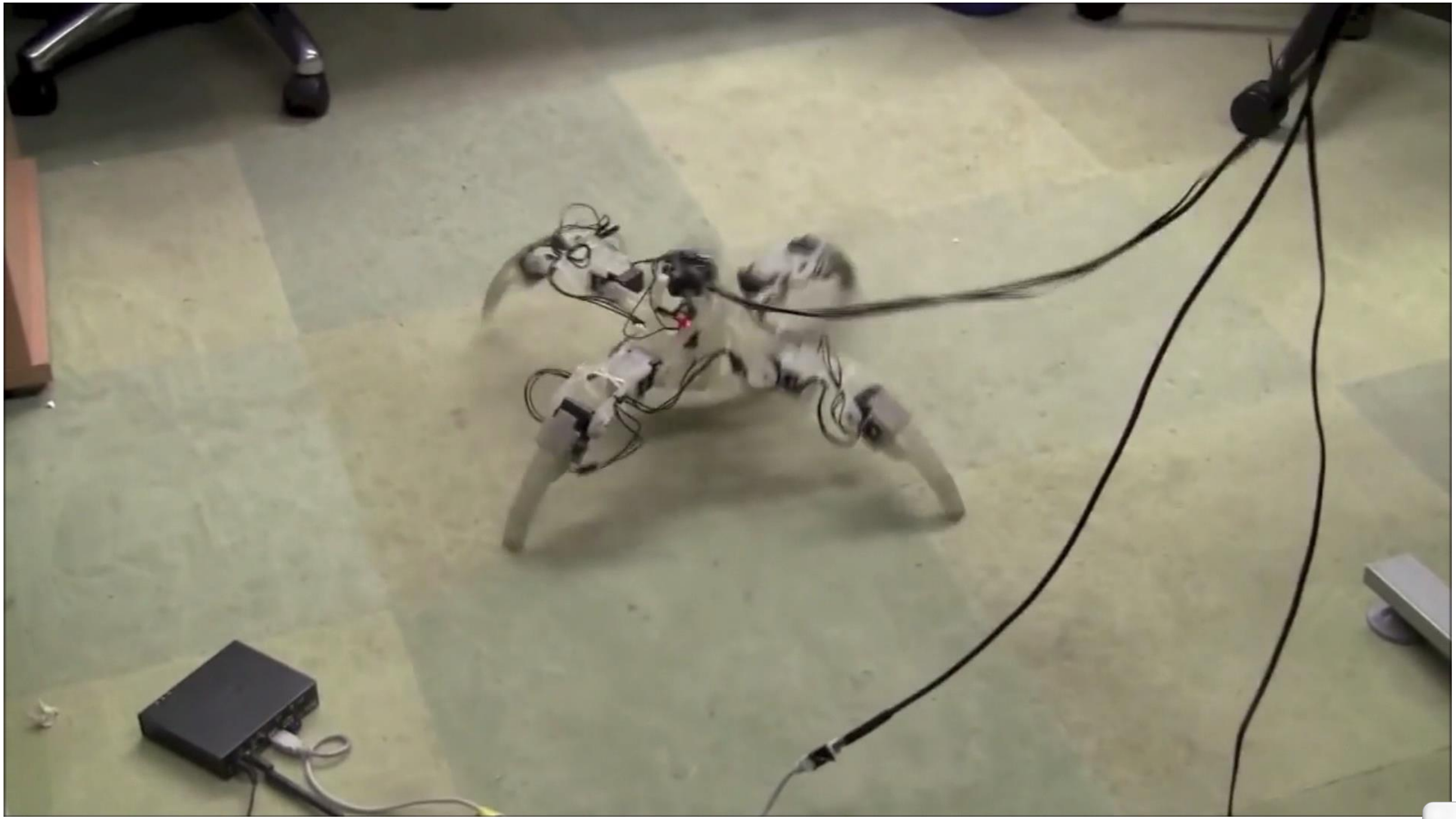
Example II



QuadraTot: A Learning Quadruped Robot Demo, Cornell University, Hidalgo, Nguyen, Yosinski

URL

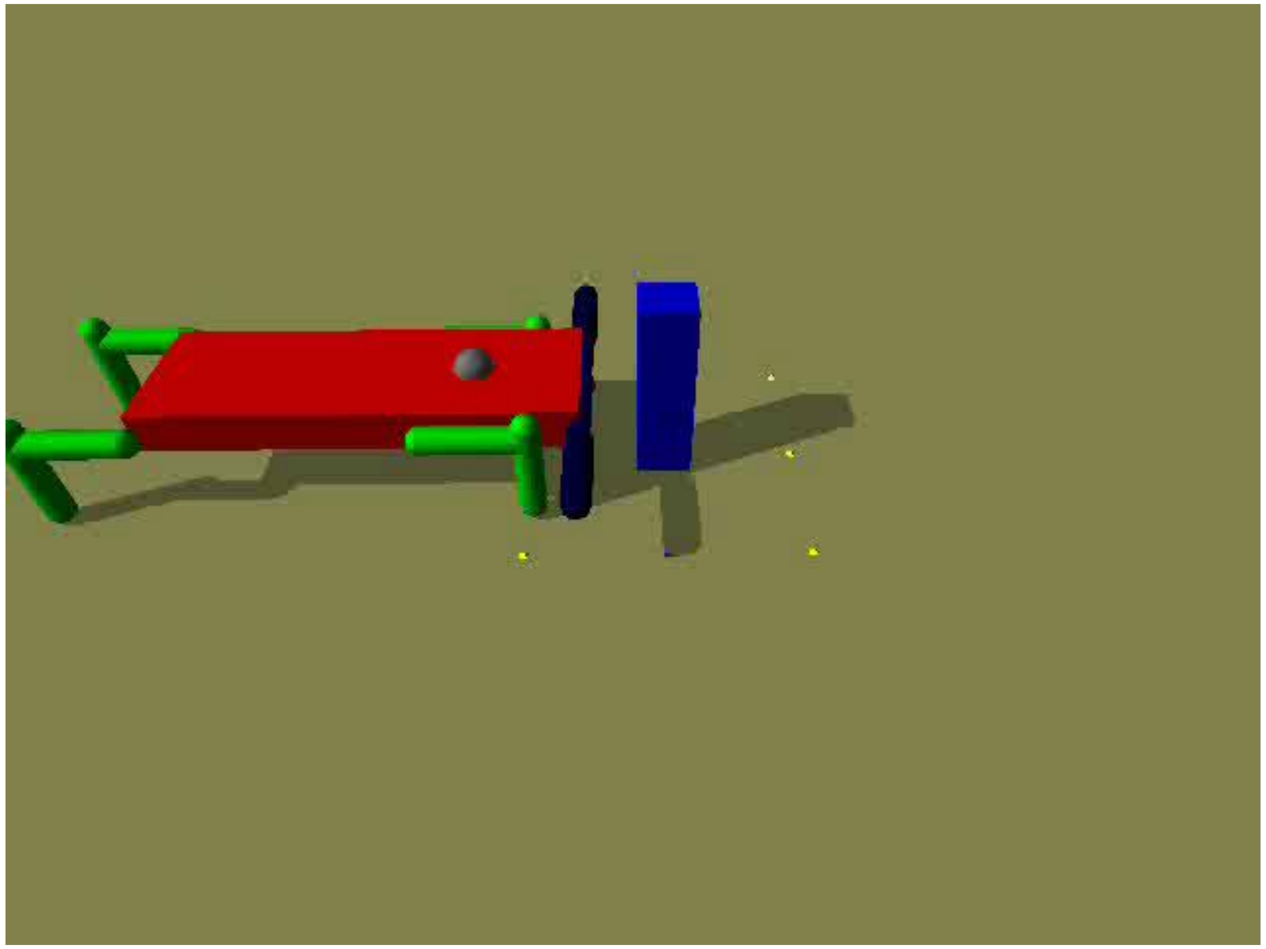
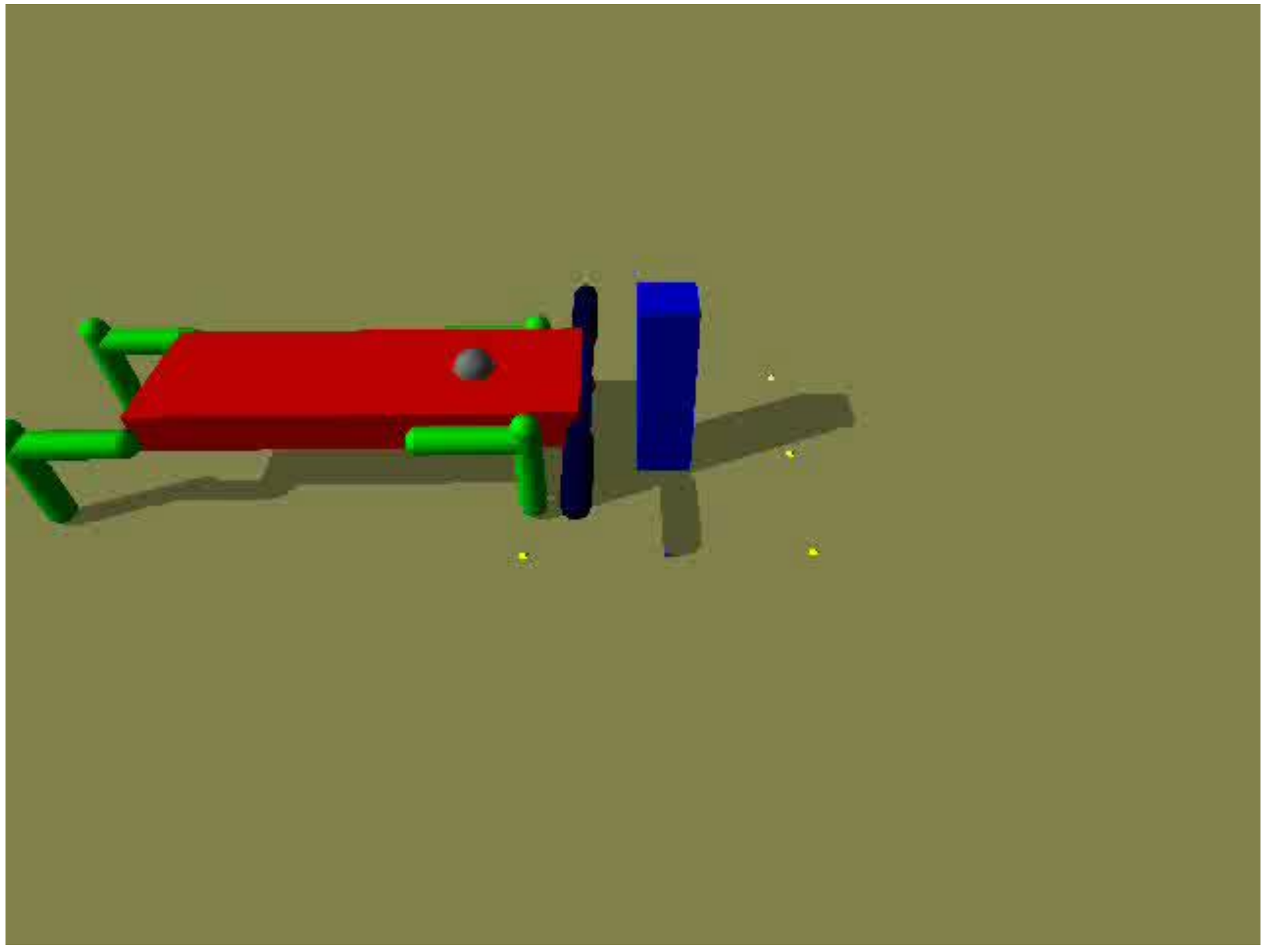
Example II (cont'd)



From: YouTube, Jeff Clune, Univ. of Wyoming, "Evolving Gaits for Legged Robots: Neural Networks with Geometric Patterns Perform Better"

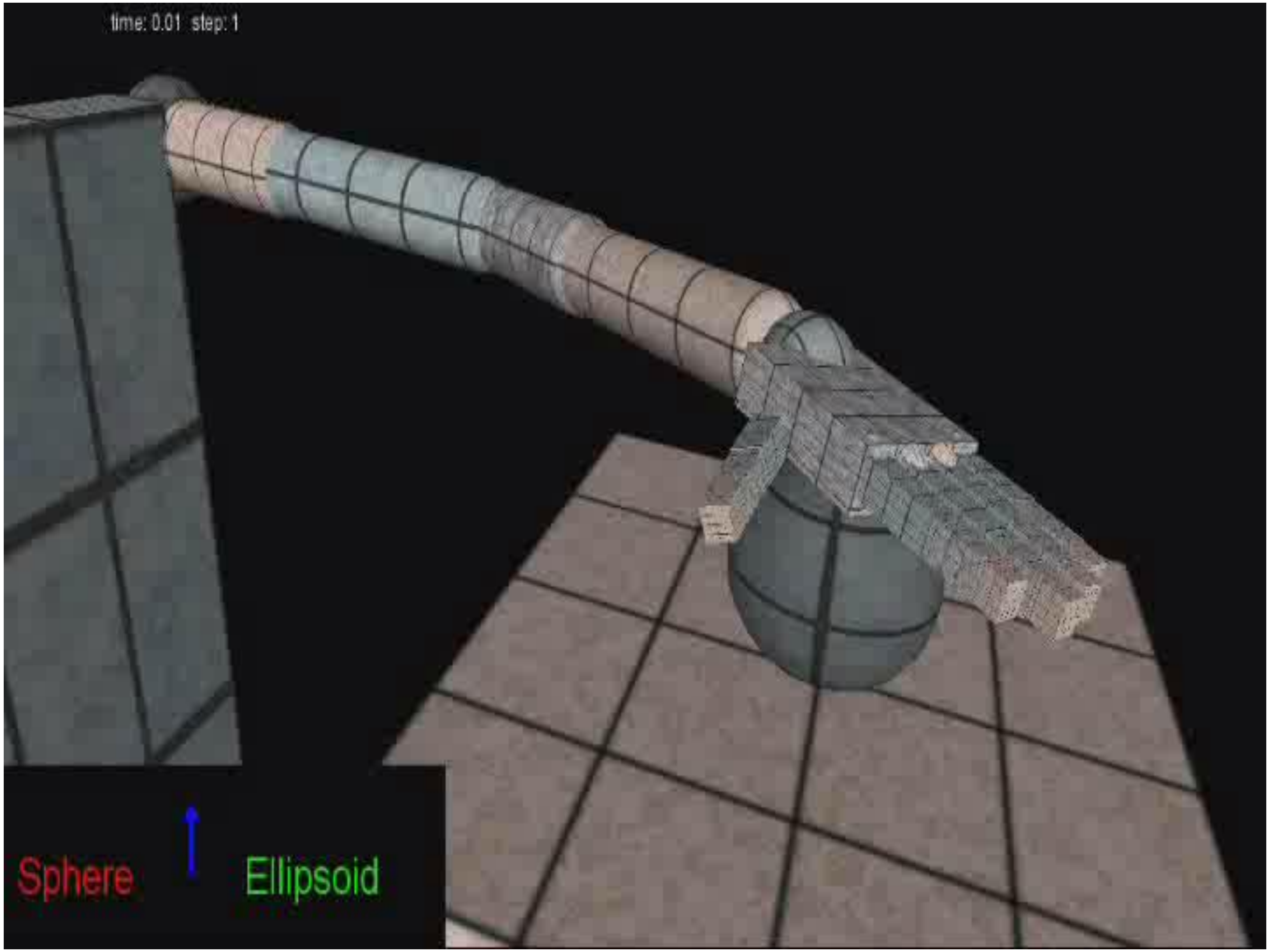
URL

Example III

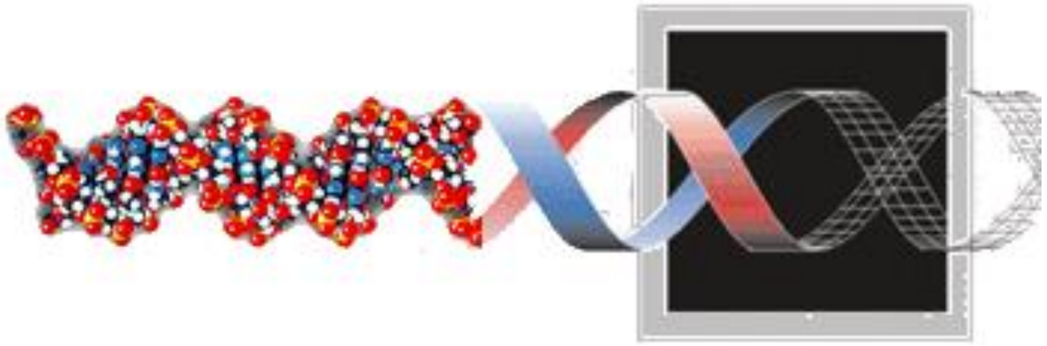
Bongard, J. (2008) Behavior Chaining: Incremental Behavior Integration for Evolutionary Robotics, Artificial Life XI, MIT Press, Cambridge, MA.

Example IV



Tuci et al., “Active categorical perception in an evolved anthropomorphic robotic arm“

Evolutionary Algorithms



1001101010001

Optimization, candidate solutions, fitness

- Evolutionary Algorithms are optimization algorithms
- Individuals in a population → *Candidate solutions* to an optimization problem
- Fitness: success of an individual in its environment (affects its reproduction rate) → Should capture what we want the robot to do

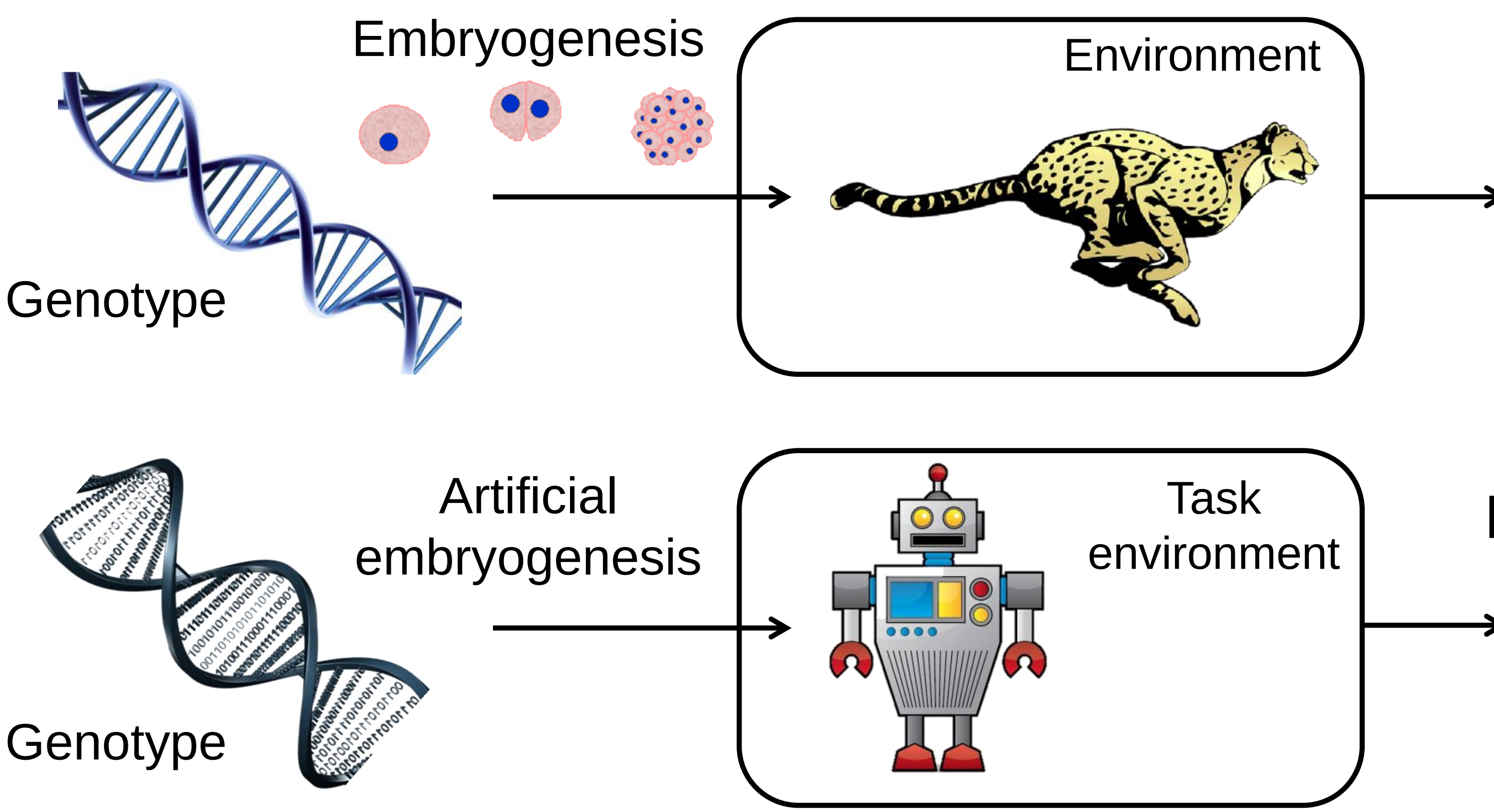
→ A function to be maximized/minimized (*objective*)

→ Simplest case: *single-objective*, real-valued function

→ More complex case: *multi-objective* (later)

Genetic representation, genotype-phenotype mapping

- Solutions (robots) must be *encoded* somehow in order to be manipulated by the algorithm → *Genetic representation*
- The genotype is then somehow expressed to form the phenotype (→ *Genotype-to-phenotype mapping*), that is evaluated *(fitness)* in the task environment



Fitness = number of offsprings

Fitness value = scalar quantifying how well the robot performed (→ will affect the probability of surviving to the next generation)

Genetic representation, genotype-phenotype mapping

- Complex encodings are often used in evolutionary robotics in order to efficiently encode both robot brains and bodies and increase evolvability → Indirect/generative/developmental encodings (later)

- **For the time being, imagine:**

  - Genotype: vector of N numbers (genes) (parameters)

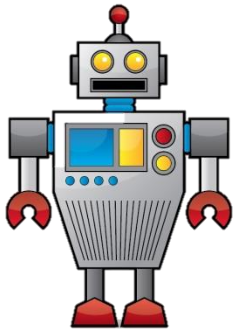
  - Direct encoding: 1:1 genotype-to-phenotype mapping, each gene represents a specific phenotypic trait

Example:

0.1	**1.5**	**0.01**	**3.1**	**2.7**

Genetic representation: vector of control (e.g. synaptic weights) and morphological (e.g. mass and length of each limb) parameters

→



Task environment: flat ground where the robot can walk

→

Fitness value = measured locomotion speed

Genetic representation, genotype-phenotype mapping

- Complex encodings are often used in evolutionary robotics in order to efficiently encode both robot brains and bodies and increase evolvability → Indirect/generative/developmental encodings (later)
- **For the time being, imagine:**
  - Genotype: vector of N numbers (genes) (parameters)
  - Direct encoding: 1:1 genotype-to-phenotype mapping, each gene represents a specific phenotypic trait

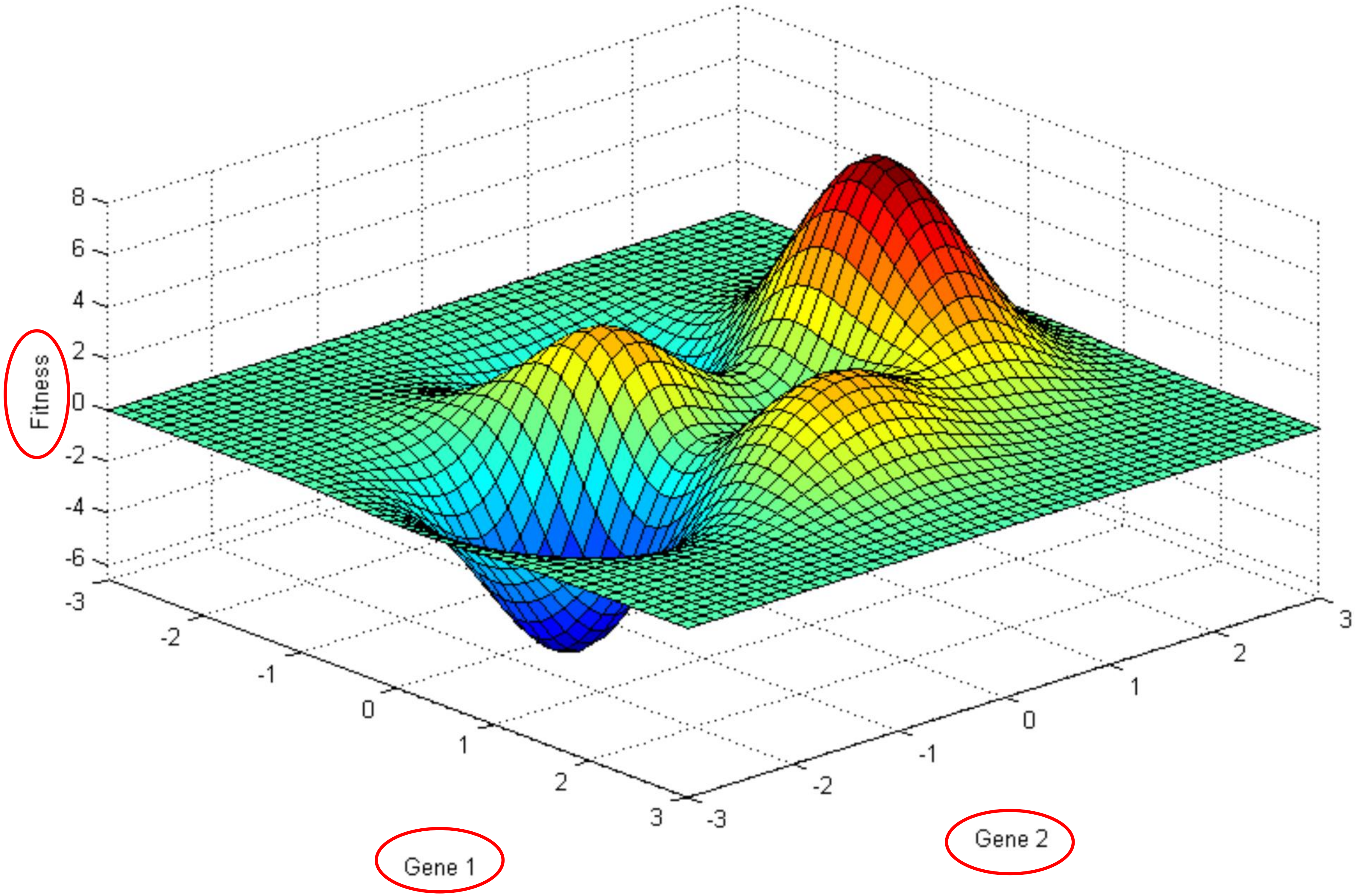
→ **The optimization algorithm will return a particular choice of values for the genes/parameters that maximizes/minimizes the fitness function (objective)**

Fitness landscape

- Represents the mapping between genotypes and fitness values*
- If the genotype is a vector of N real values, the fitness landscape is a multi-dimensional surface in $\Re^{N+1}$
- This mapping is usually unknown and non intuitive, which is why we need algorithms that can sample and navigate this surface until good solutions (ideally, global maxima/minima) are found
- Useful way to mentally visualize high-dimensional problems

* Keep in mind that the fitness is, however, always computed on the *phenotype* associated to a *genotype*

Fitness landscape

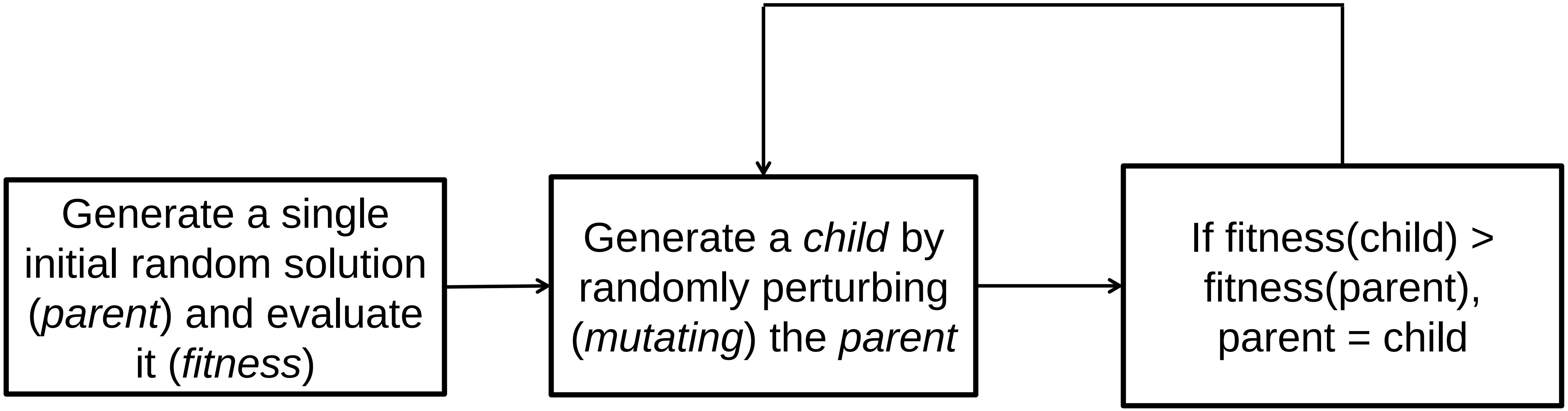


Fitness landscape – Assumptions

- Differently from other methods (e.g. gradient-based methods), evolutionary algorithms do not require a priori assumptions/knowledge regarding the properties of the function to be optimized (e.g. it doesn't have to be differentiable) → black box
- Drawback: they usually cannot guarantee convergence to a global optimum

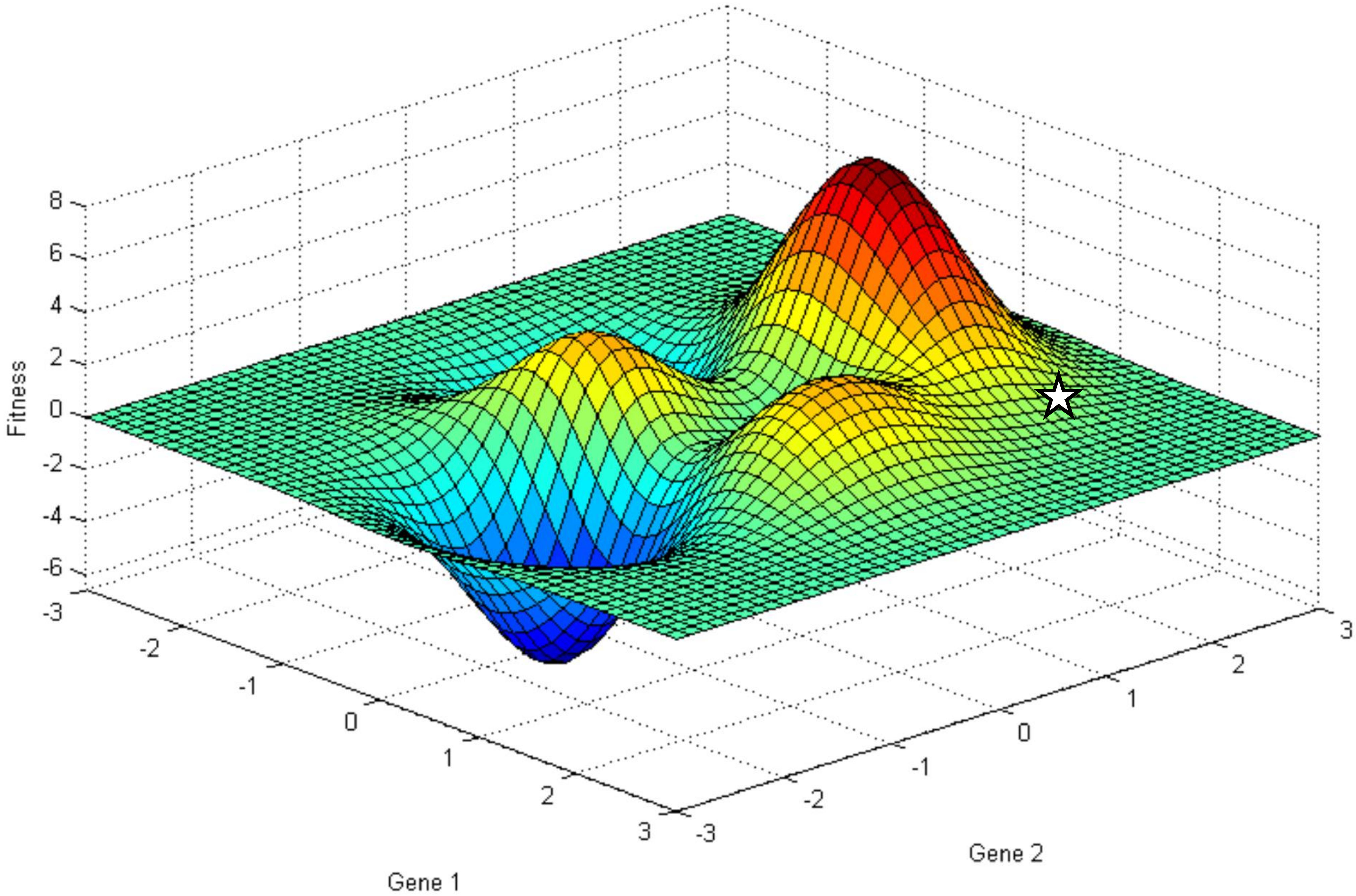
The simplest algorithm

<u>Recall</u>: "Evolution can be thought of as a <u>trial-and-error</u> process, in which innovation is driven by the <u>non-random selection of random variations</u>"

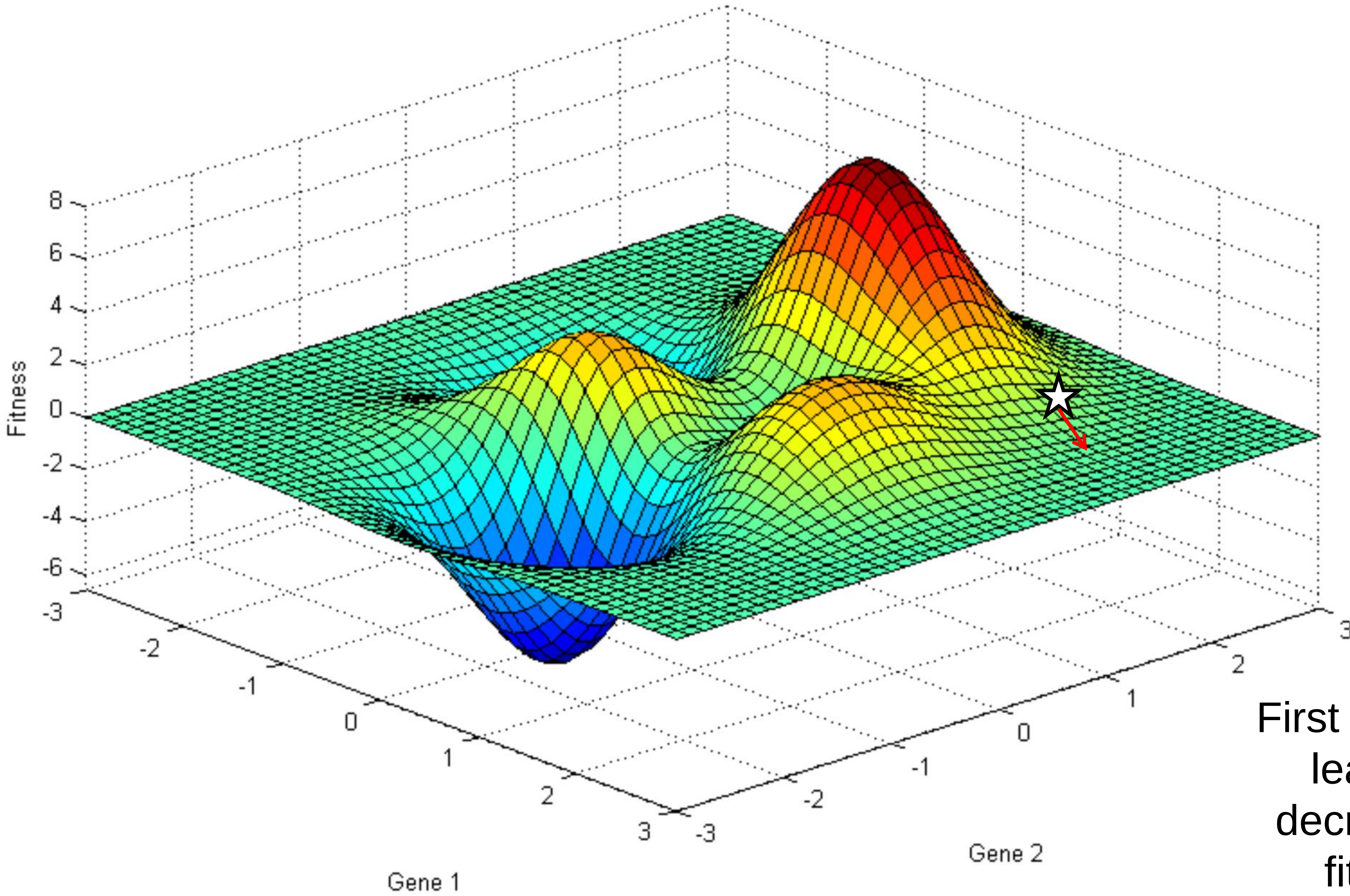


Repeat until sufficiently fit solution is found, or for a fixed number of iterations

How does it operate on the fitness landscape?

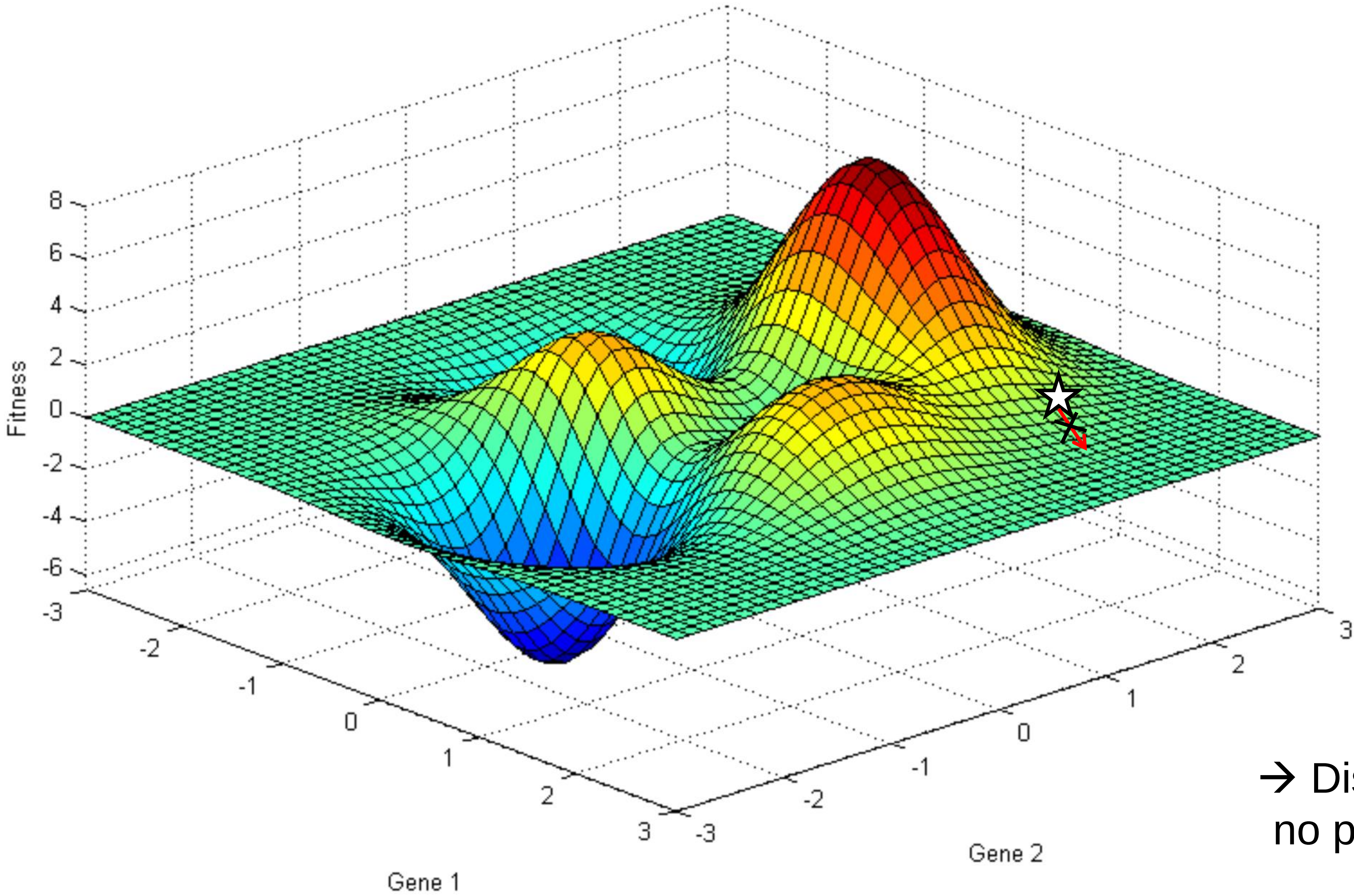


How does it operate on the fitness landscape?



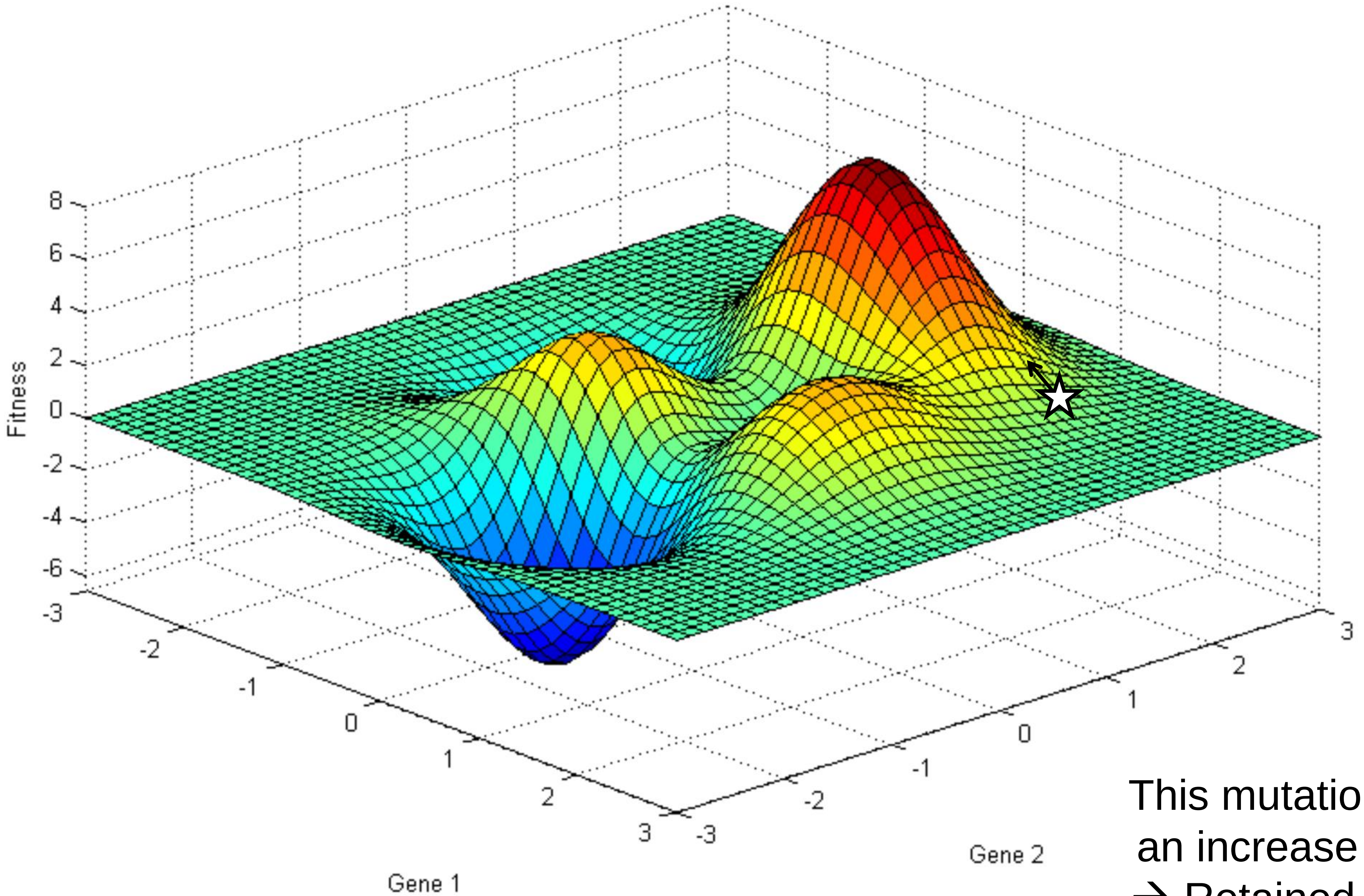
First mutation leads to decrease in fitness

How does it operate on the fitness landscape?



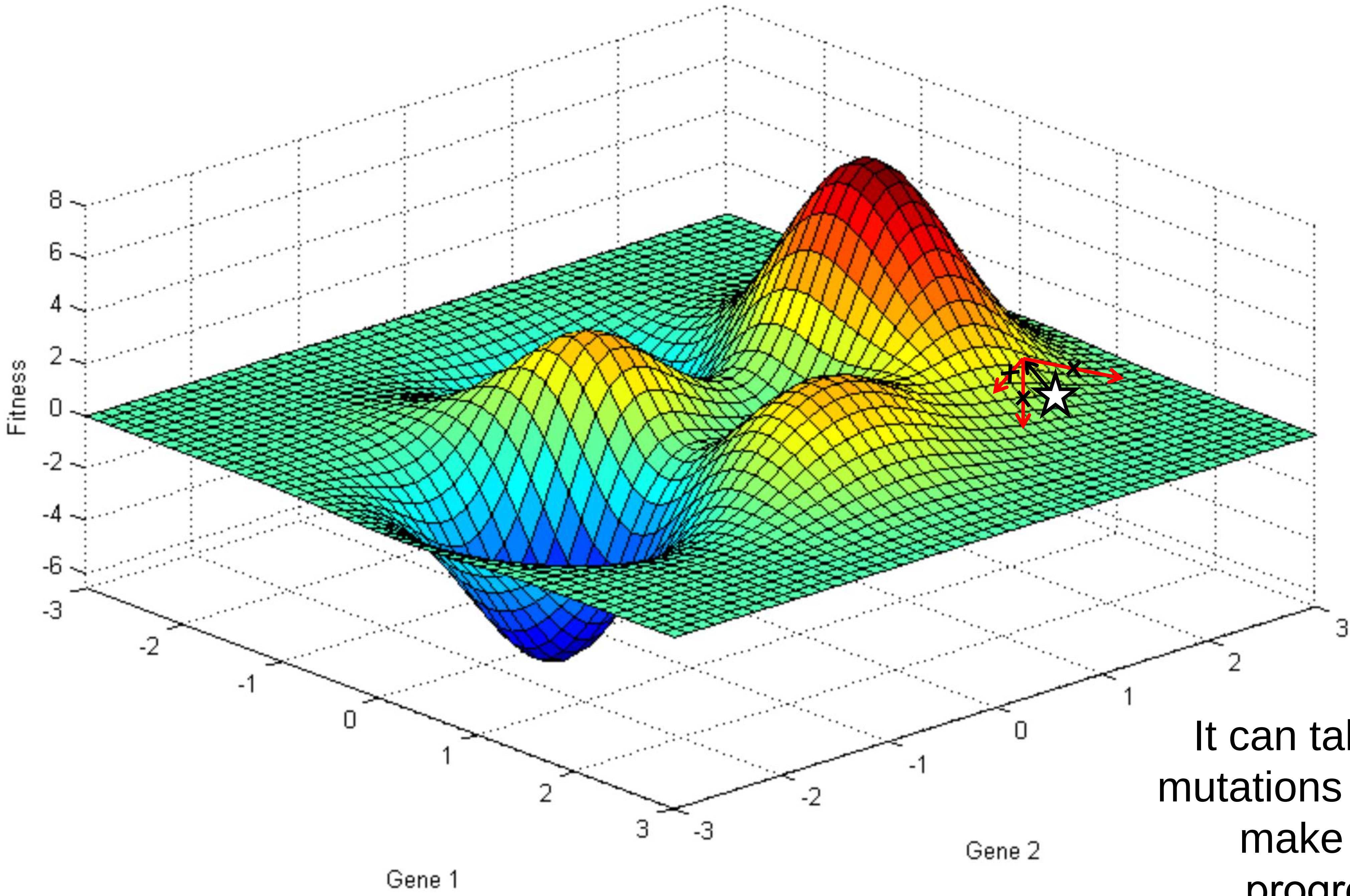
→ Discarded, no progress

How does it operate on the fitness landscape?



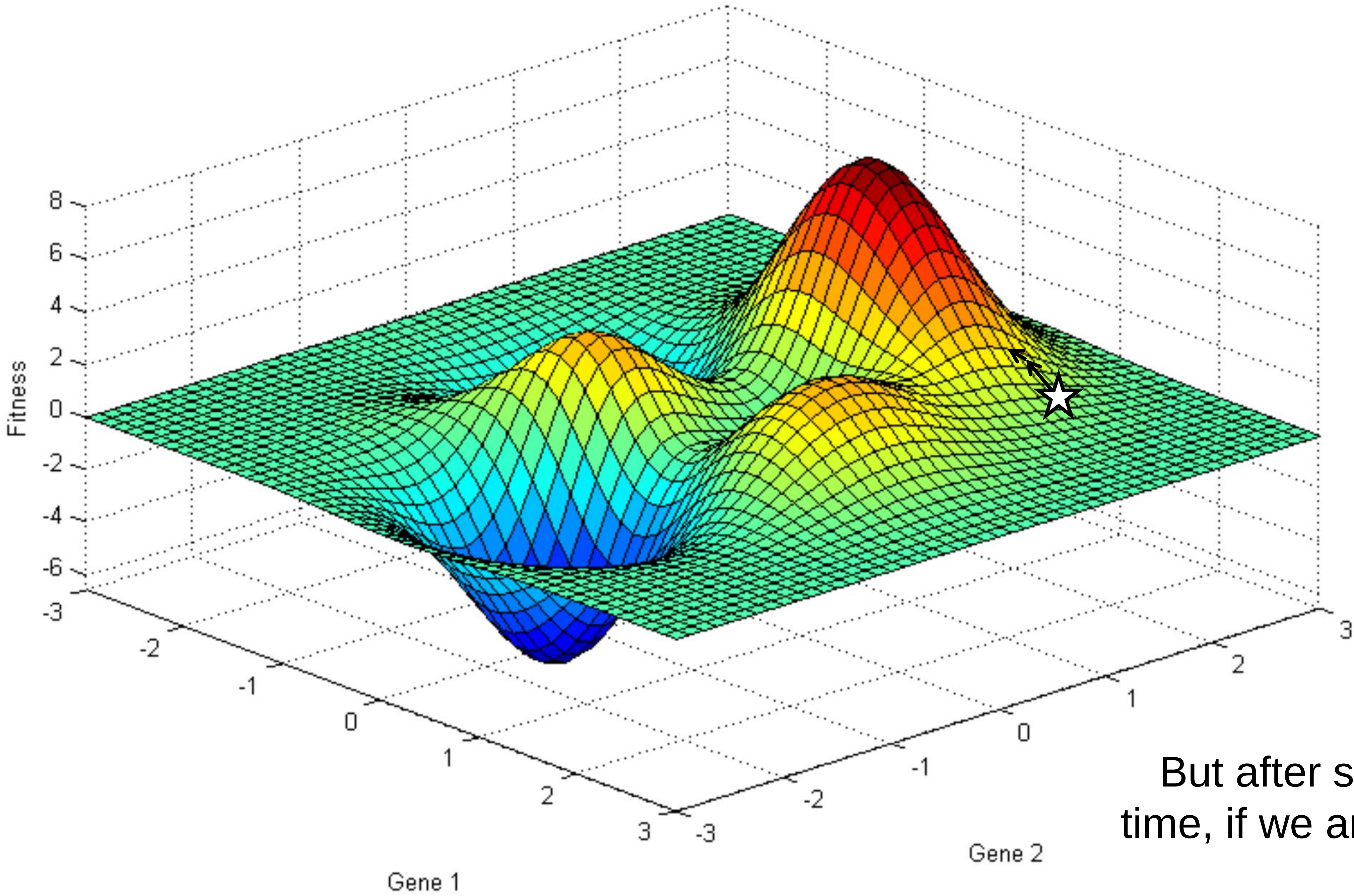
This mutation leads to an increase in fitness → Retained, progress

How does it operate on the fitness landscape?



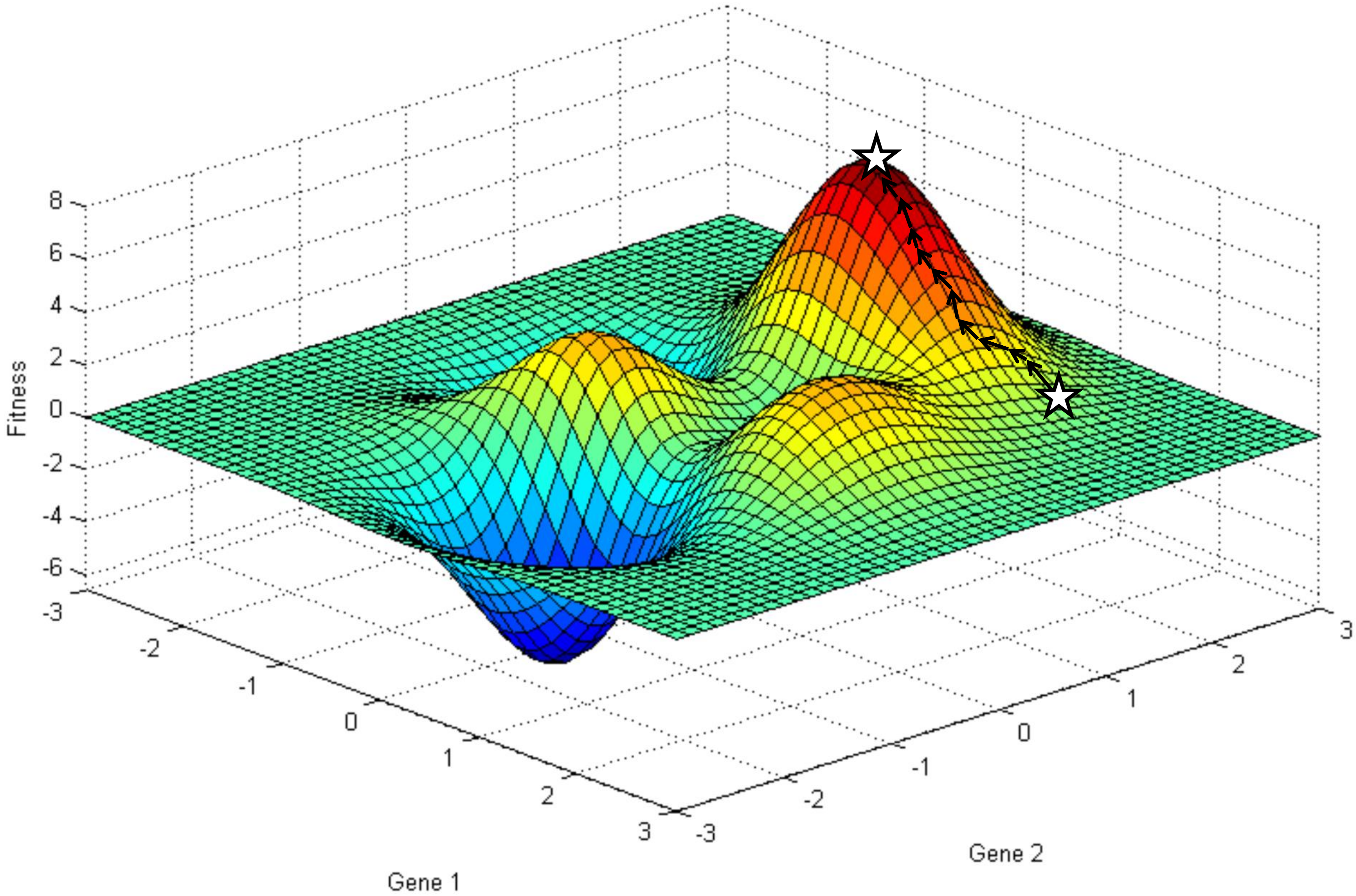
It can take many mutations before we make some progress…

How does it operate on the fitness landscape?

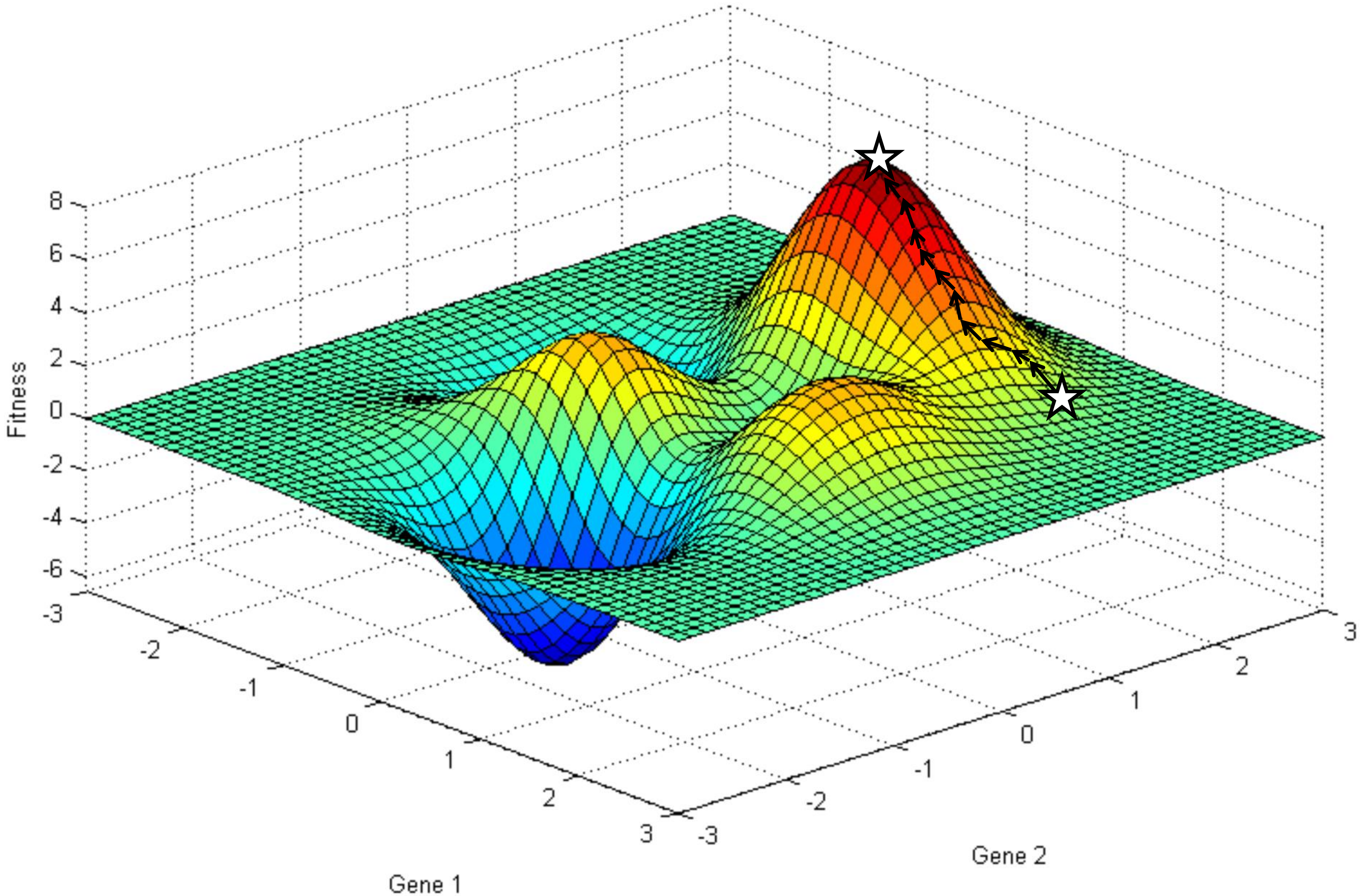


But after sufficient time, if we are lucky…

How does it operate on the fitness landscape?



→ The «Serial Hill Climber»



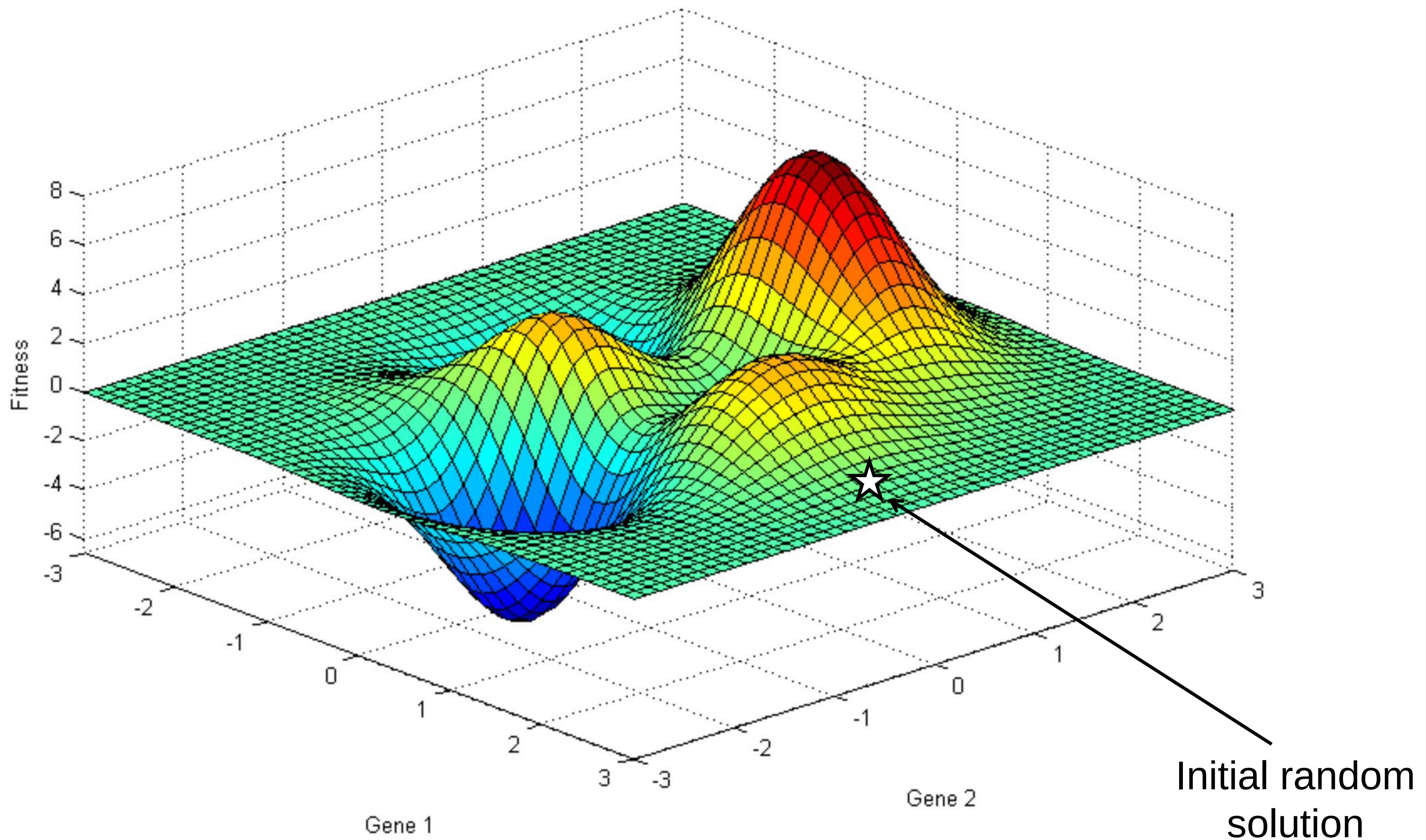
The Serial Hill Climber

Pro:

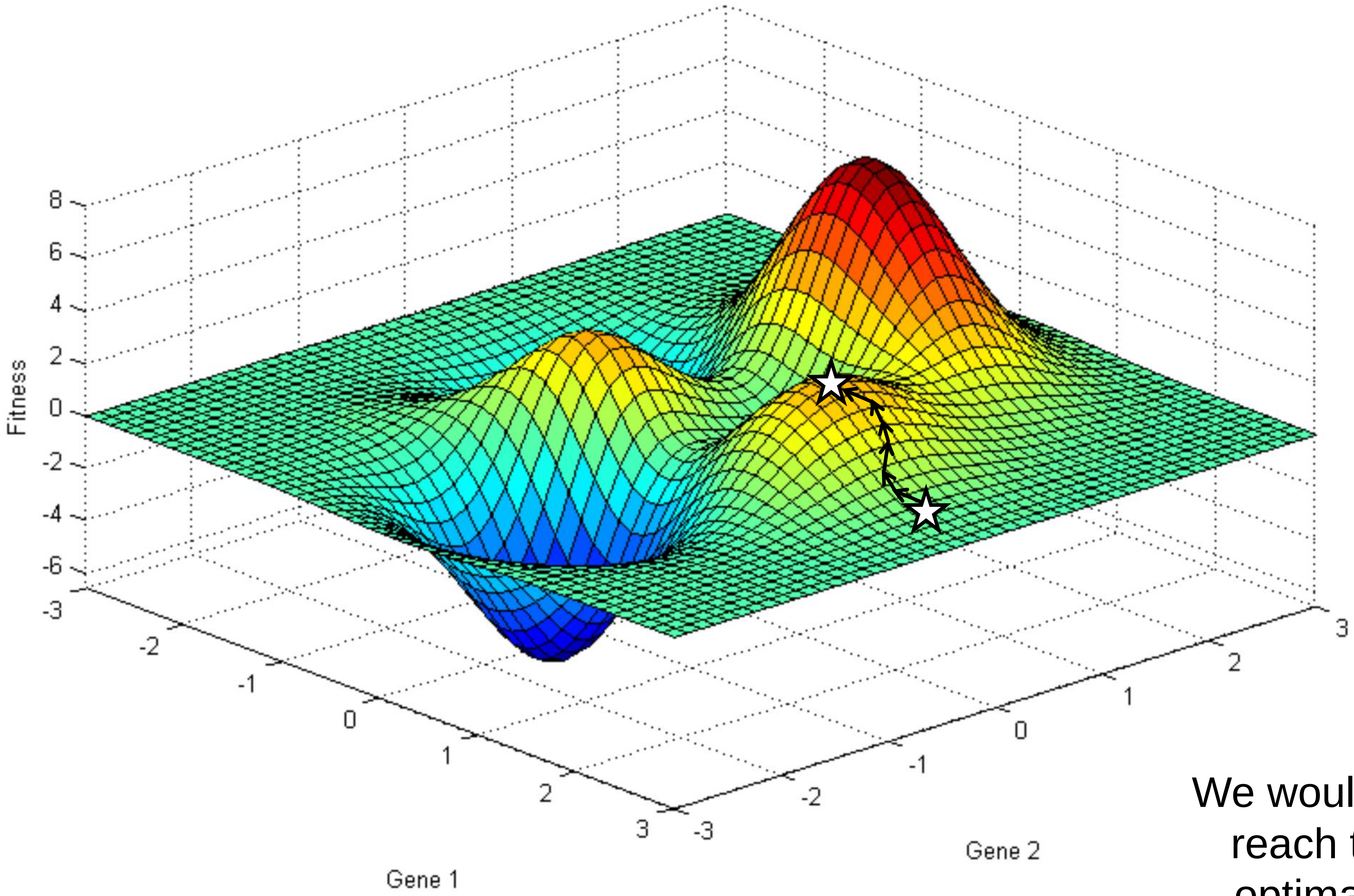
- Very simple to implement
- Only one hyperparameter: *mutation rate*

Cons: ?

Different initial condition?



Different initial condition



We would probably reach this local optima instead

The Serial Hill Climber

<u>Pro:</u>

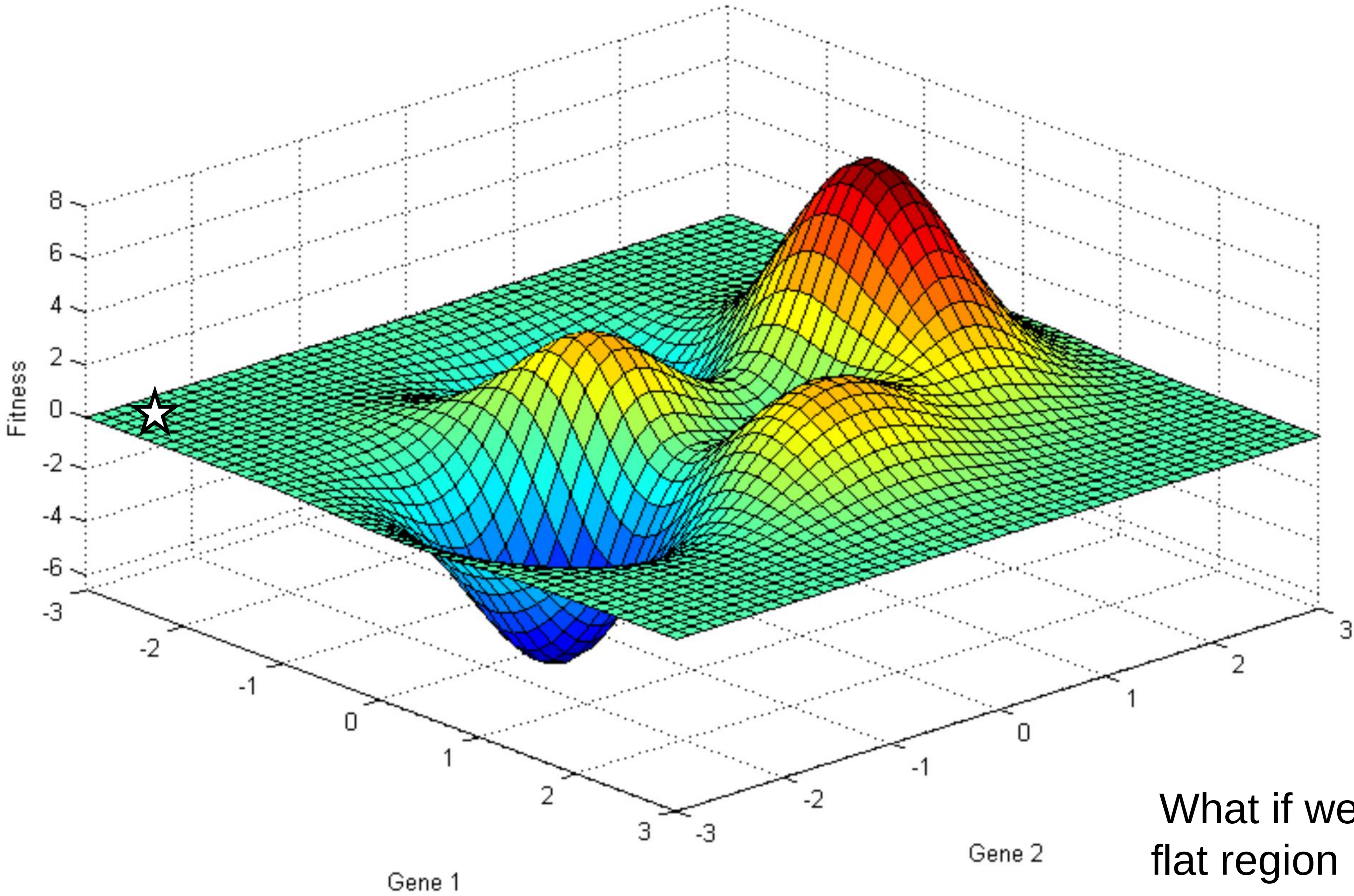
- Very simple to implement
- Only one hyperparameter: <u>*mutation rate*</u>

<u>Cons</u>:

- In general climbs the closest <u>local</u> peak

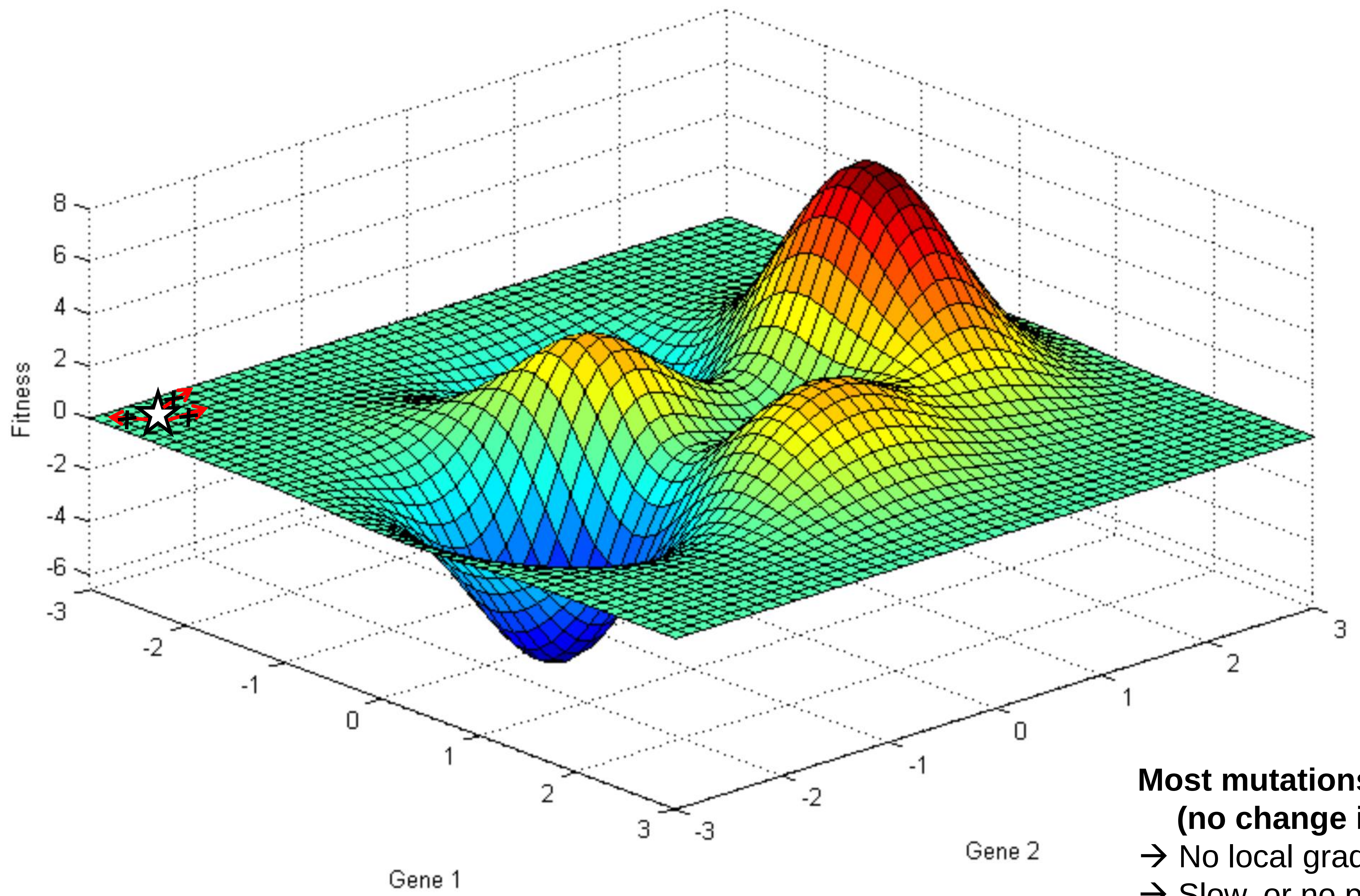
  → Very <u>sensitive to the starting condition</u>

Different initial condition



What if we start in a flat region (*plateau*)?

Different initial condition



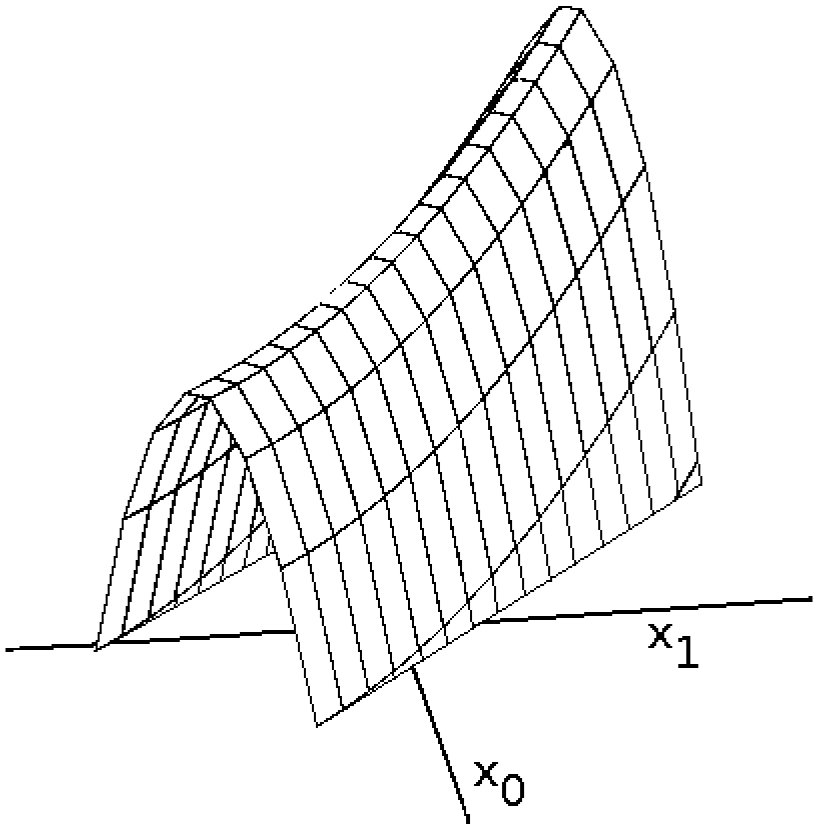
Most mutations are <u>neutral</u> (no change in fitness)

→ No local gradient to climb!

→ Slow, or no progress at all

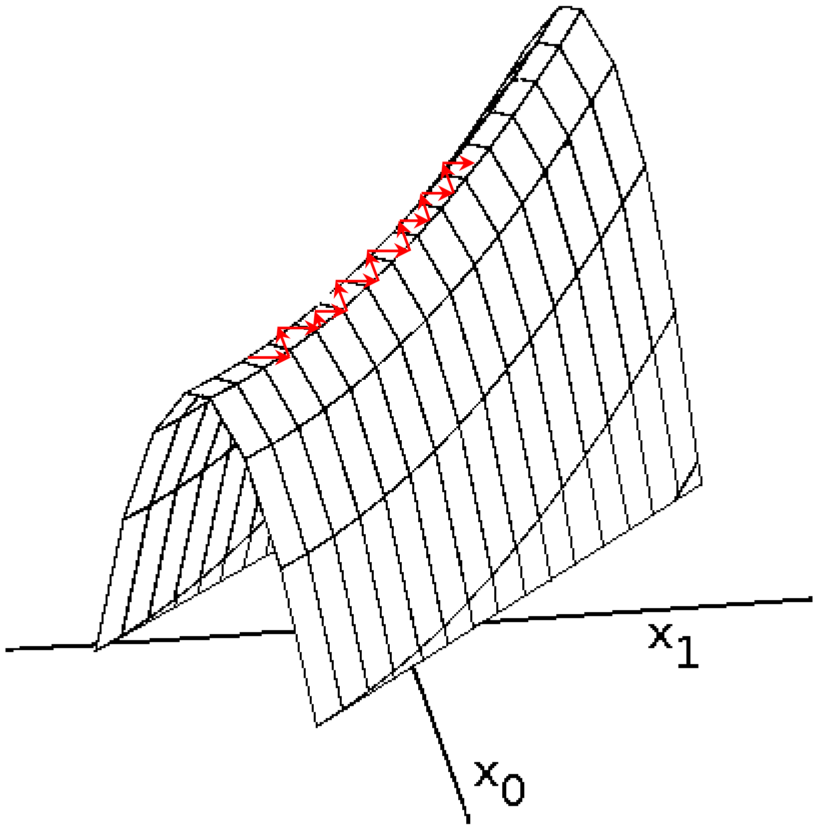
Different fitness landscape

What if the fitness landscape was like the one on the right (ridge) and our mutation operator only perturbs one gene (x0 or x1) at a time?



Different fitness landscape

What if the fitness landscape was like the one on the right (ridge) and our mutation operator only perturbs one gene (x0 or x1) at a time?



The Serial Hill Climber

Pro:

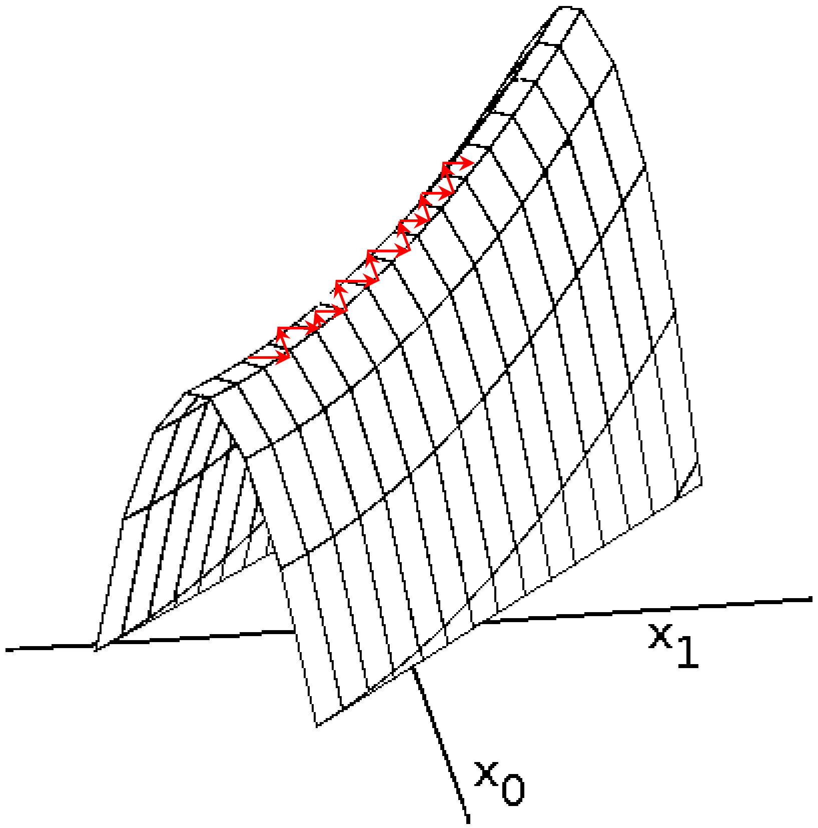
- Very simple to implement
- Only one hyperparameter: *mutation rate*

Cons:

- In general climbs the closest local peak

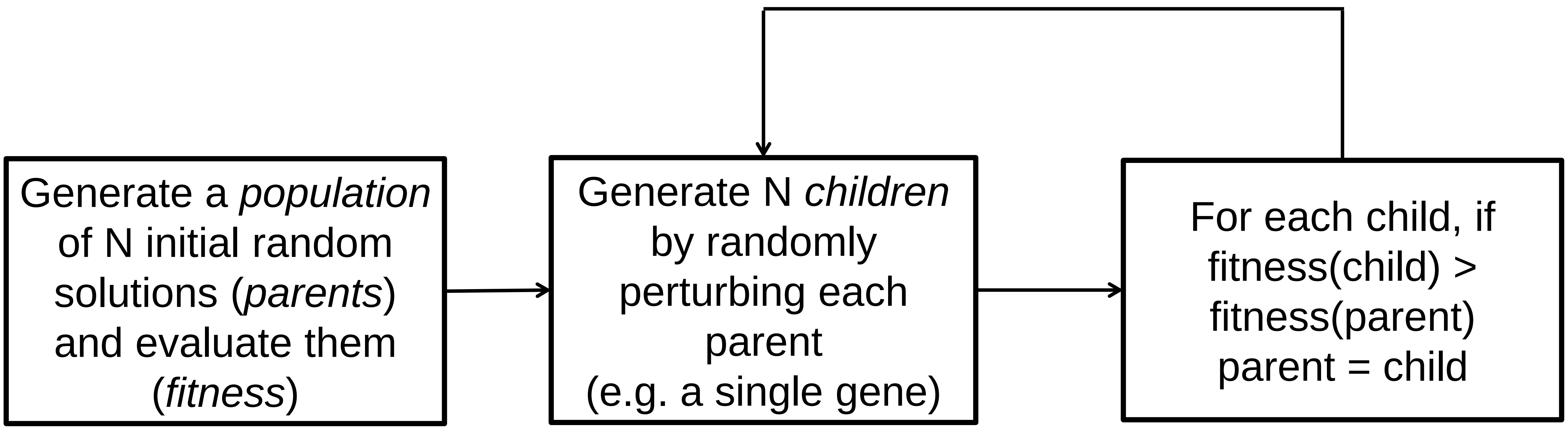
  → Very sensitive to the starting condition

- Difficulty in navigating *plateau* and ridges → Can improve very slowly, or not improve at all

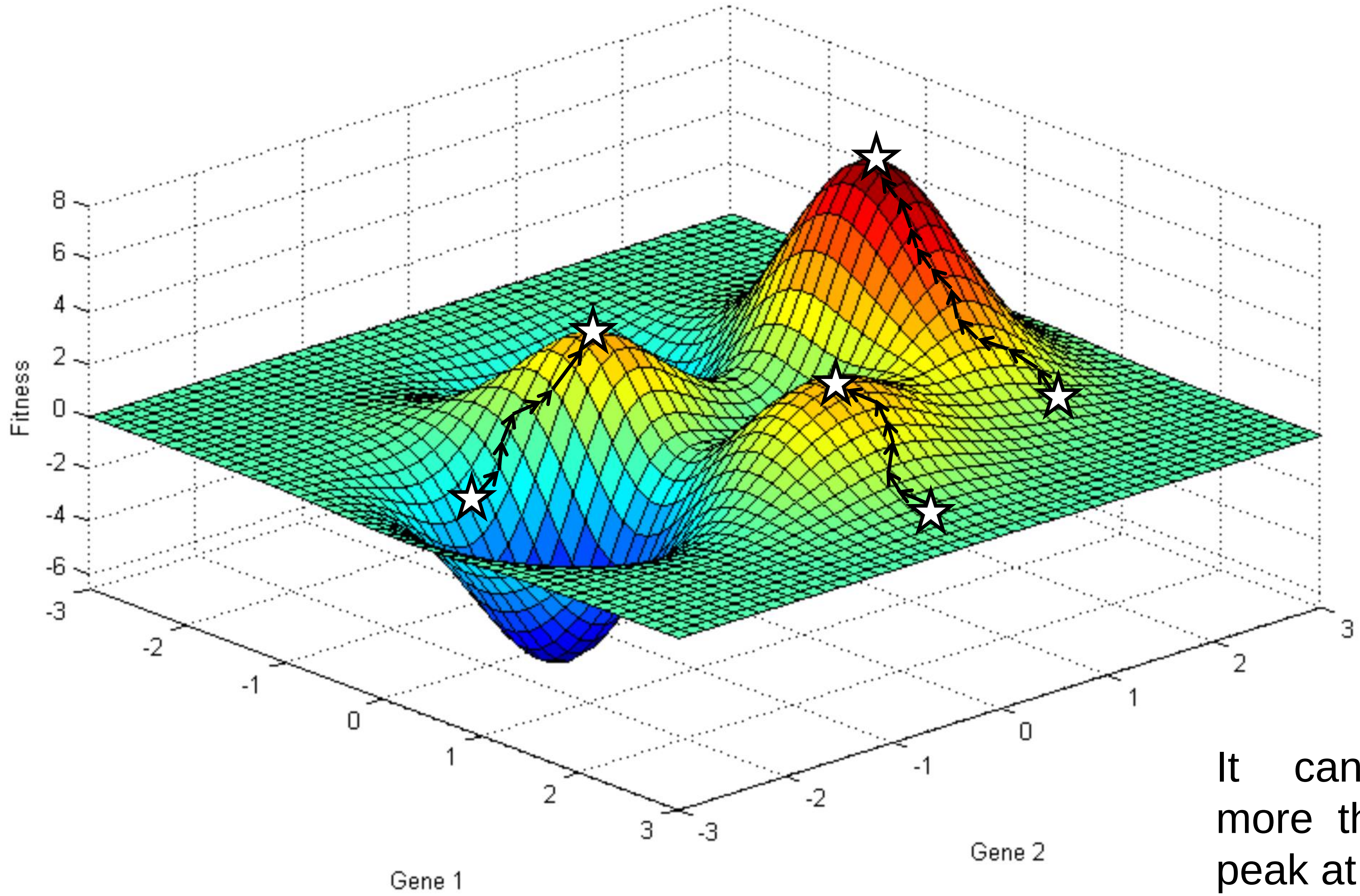


The Parallel Hill Climber

The Serial Hill Climber can be improved by executing several instances of it, in parallel → *<u>Parallel Hill Climber</u>*



The Parallel Hill Climber (ideally)



It can climb more than one peak at a time

Genetic Algorithms

- They work with *populations* of candidate solutions, too (popSize > 1)
- While hillclimbers only mimic genetic *mutations*, Genetic Algorithms (GA) mimic sexual *recombination* as well (*crossover*)

Genetic Algorithms

- They work with *populations* of candidate solutions, too (popSize > 1)
- While hillclimbers only mimic genetic *mutations*, Genetic Algorithms (GA) mimic sexual *recombination* as well (*crossover*)

→ *Recall: crossover entails a child getting part of the genetic material from one parent, and part from the other parent*

Any idea why this might be useful?

Genetic Algorithms – recombination, why?

	Genotype				
Optimal solution	1	1	1	1	1
Parent 1	1	1	1	0	0
Parent 2	0	0	0	1	1

- Children: partially good solutions
- It would take more than one mutation to reach the optimal solution (e.g. using a hillclimber)

Genetic Algorithms – recombination, why?

	Genotype				
Optimal solution	1	1	1	1	1
Parent 1	1	1	1	0	0
Parent 2	0	0	0	1	1
…	0	1	0	1	1
	1	1	0	1	1
	1	1	1	1	1

<u>At least</u> three mutations, usually more (mutations can disrupt correct bits, too)

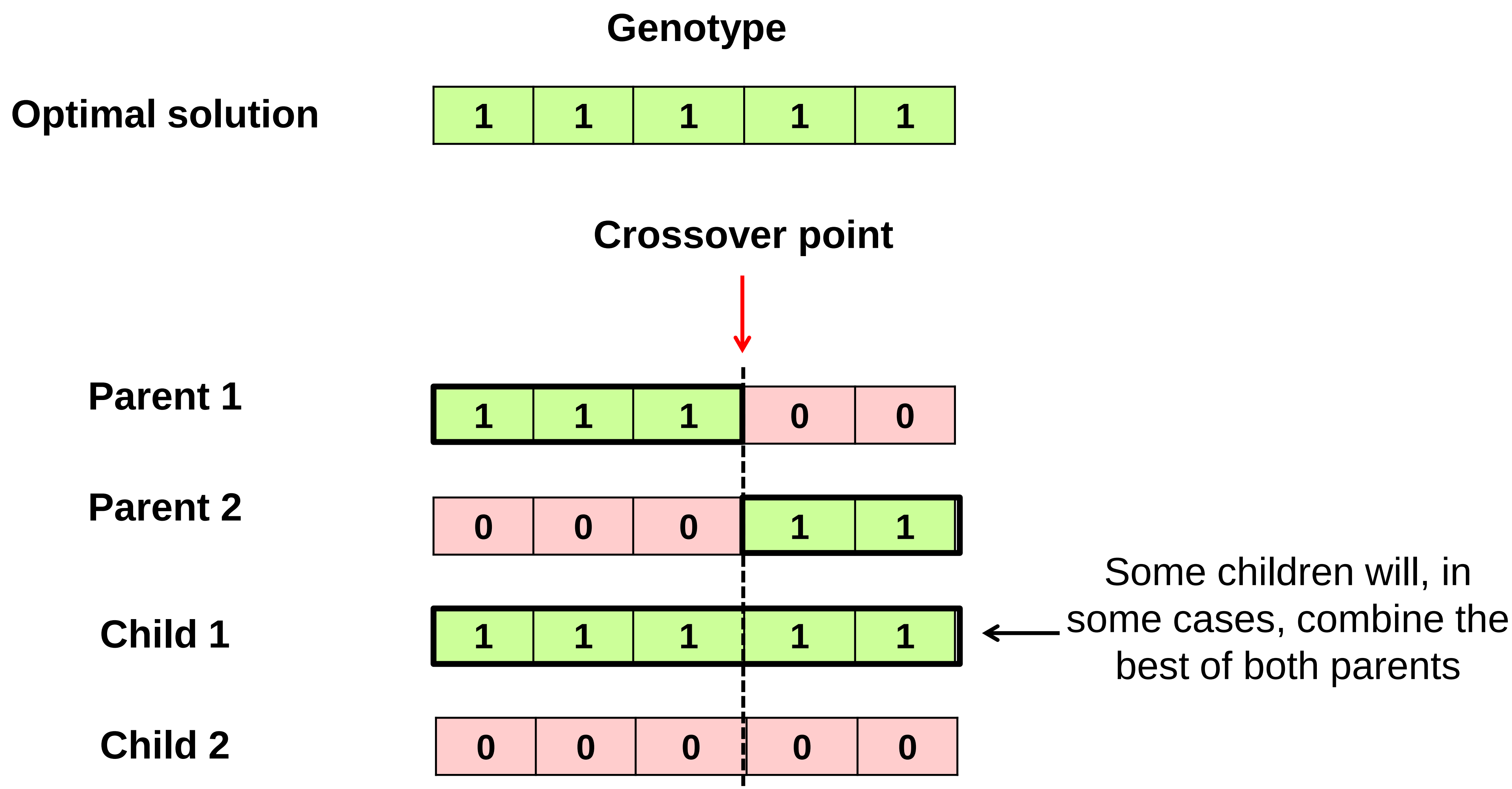
Genetic Algorithms – recombination, why?

	Genotype				
Optimal solution	1	1	1	1	1
Parent 1	1	1	1	0	0
Parent 2	0	0	0	1	1

- Children: partially good solutions
- It would take more than one mutation to reach the optimal solution (e.g. using a hillclimber)

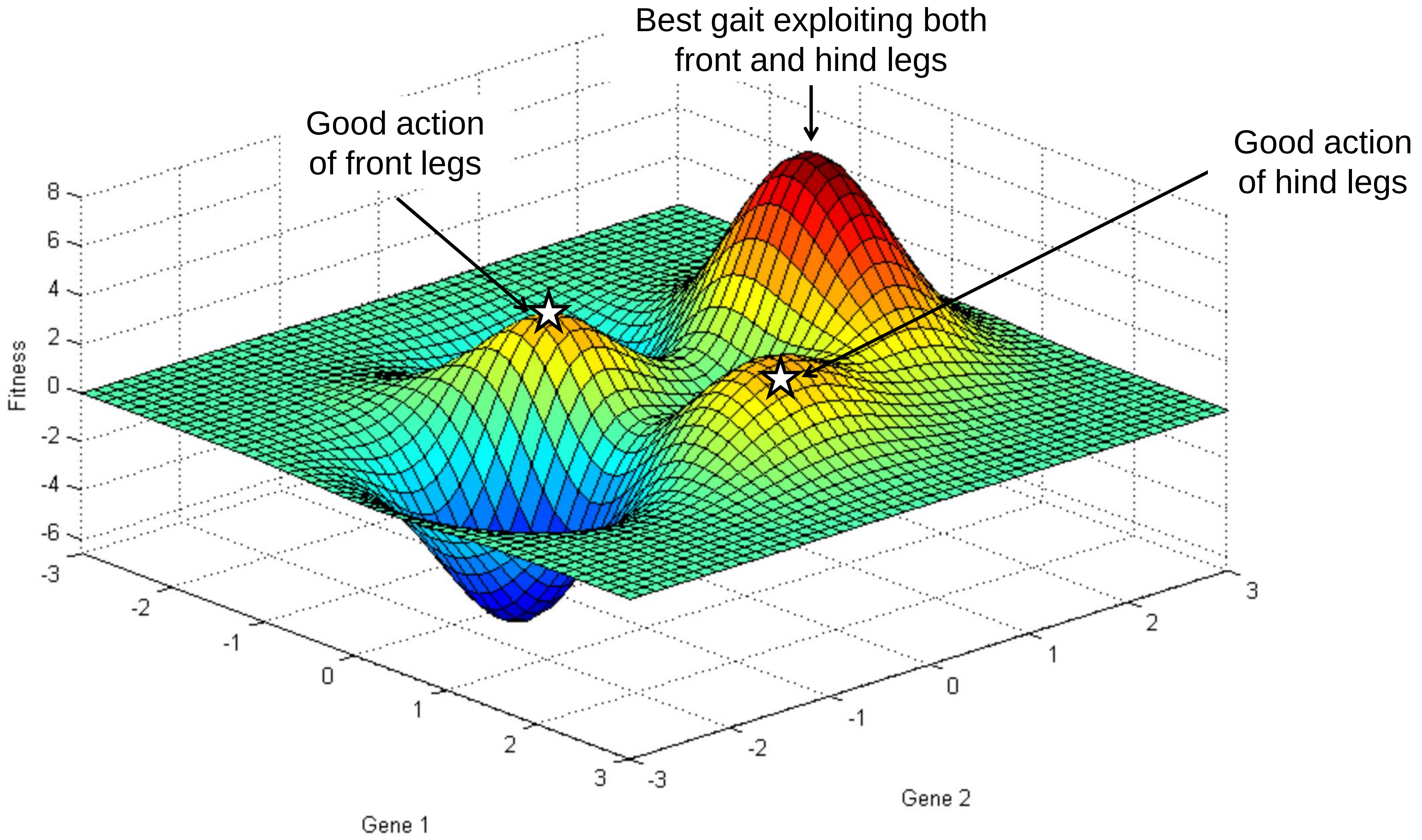
→ If we could recombine the two parents (mix their genetic material), a single operation could lead to the optimum instead

Genetic Algorithms – recombination, why?



Genetic Algorithms – recombination, why?

<u>(Very extreme) Example</u>: evolving control for a quadruped



Genetic Algorithms

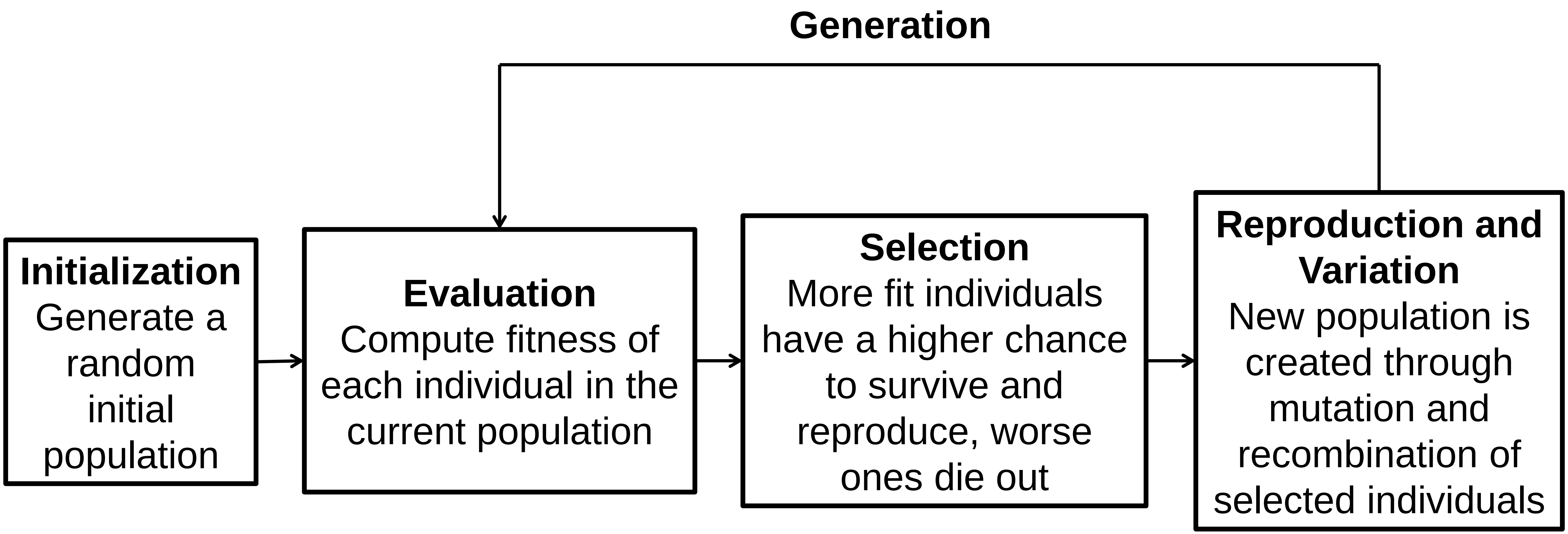
- **Underlying assumption: partial good solutions can be combined to form even better ones → "*building block hypothesis*"**

  → While this structure is present in many problems, crossover operators must be properly designed in order to exploit it. Their effect can be disruptive otherwise (similar to that of big mutations)

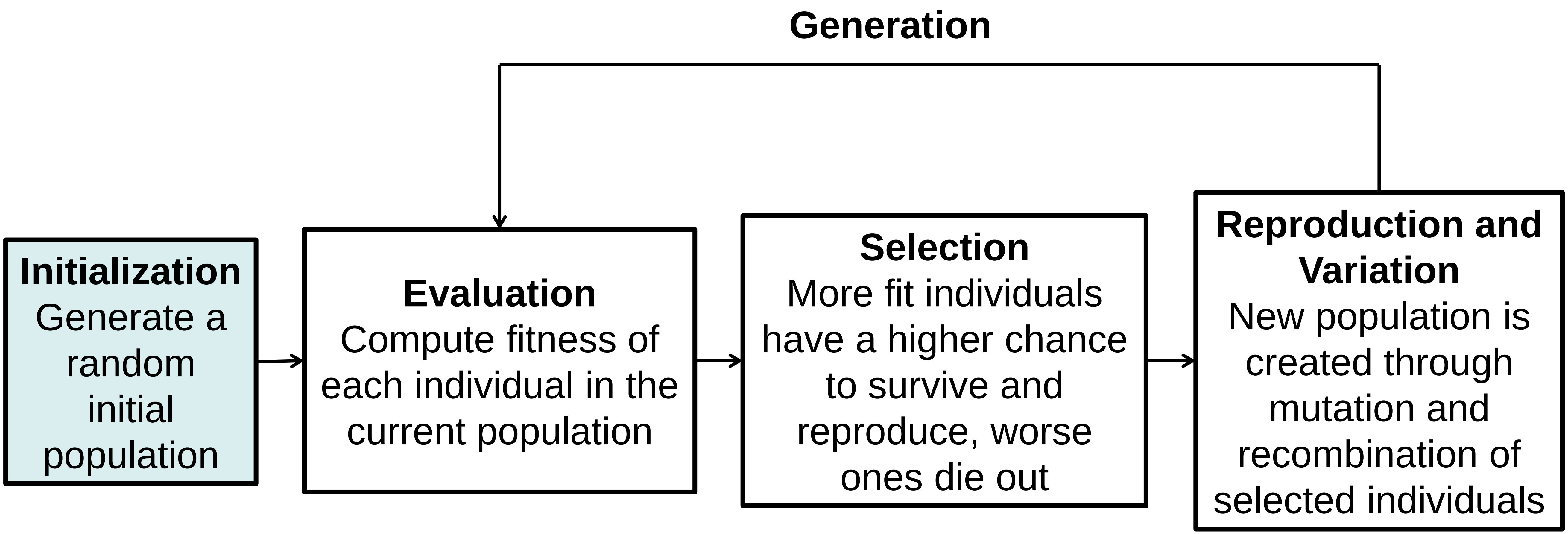
- Crossover + mutation should create a good tradeoff between global and local search

  → Exploration (test new solutions) vs exploitation (refining current ones)

Genetic Algorithms – overall scheme



Genetic Algorithms – overall scheme



Initialization – Population size

- Population size = size of the sample of the search space taken at every generation

Initialization – Population size

- <u>Population size</u> = <u>size of the sample of the search space</u> taken at every generation

What are the pros and cons of big / small populations?

Initialization – Population size

- Population size = size of the sample of the search space taken at every generation
- Smaller population → Small portion of the search space is sampled at each step → Easier to get stuck on local optima ("narrow eyefield")

  → "***Premature convergence***": fitness quickly reaches a *plateau*
- It is also related to the dimensionality of the search space (e.g. number of genes) → The higher, the bigger the population should be

→ **Let's work with extremely big populations then?**

Initialization – Population size

- Population size = size of the sample of the search space taken at every generation
- Smaller population → Small portion of the search space is sampled at each step → Easier to get stuck on local optima ("narrow eyefield")

  → "***Premature convergence"***: fitness quickly reaches a *plateau*
- It is also related to the dimensionality of the search space (e.g. number of genes) → The higher, the bigger the population should be
- But increasing the population size is **costly**:
  - Bigger population → more fitness evaluations → most computational-intensive part of the algorithm

    E.g. robot needs to behave (or to be simulated) for some time

Initialization – Population size (cont'd)

It is usually experimentally determined, trying to select the biggest possible value that allows the algorithm to return in reasonable time

E.g.

Max execution time is roughly $\mathrm{T} = \mathrm{N}_{\mathrm{generations}} \cdot T_{\mathrm{eval}} \cdot P_{\mathrm{size}}$

T_{eval} should be reduced as much as possible (e.g. efficient code, …)

Upper bound on T (how long you are willing to wait for the results)

→ A tradeoff between $\mathrm{N}_{\mathrm{generations}}$ and P_{size} is found accordingly

Initialization – How?

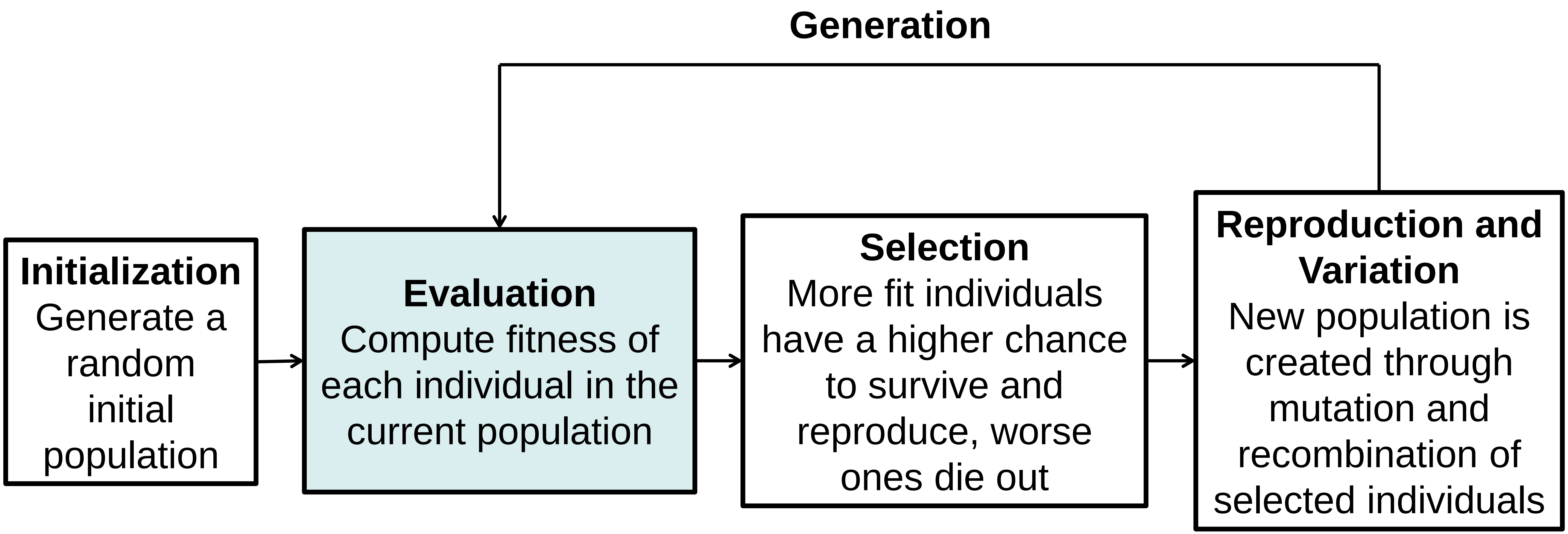
- The initial population is usually randomly generated (a random sample of the search space is taken)

  → Unbiased choice, promotes diversity (if all solutions are similar, again, we are looking at a very narrow portion of the search space)

- In some cases it is possible to *seed* the algorithm with an initial solution (hand crafted, result of another optimization technique, …)

  → In this case: Population = random variations of the seed individual

Genetic Algorithms – overall scheme



Fitness function

- It is a function associating a scalar (fitness value/*score*) to each phenotype
- Evaluating the fitness function is usually the most time-consuming part of an evolutionary algorithm
  - e.g. entails running a physics engine (computational intensive), or let a robot behave in the real world for some time
- The fitness usually quantifies individuals' *performance* (in terms of what you do want to optimize) → domain specific objective

Fitness function – aggregation

- In conventional EA there is a single optimization objective, which corresponds to a single fitness function
- E.g. speed of a walking robot, number of objects grasped by a manipulator, etc
- It is however possible to aggregate different quantities to be maximized/minimized in a single scalar value

E.g. evolving locomotion

$$fitness(f) = spaceTraveled_x$$

$$fitness(f) = 0.8 \cdot {spaceTraveled_x}^2 + 0.2 \cdot {spaceTraveled_y}^2$$

$$fitness(f) = \frac{0.8 \cdot {spaceTraveled_x}^2 + 0.2 \cdot {spaceTraveled_y}^2}{energySpent}$$

$$\ldots$$

Fitness function - Observation

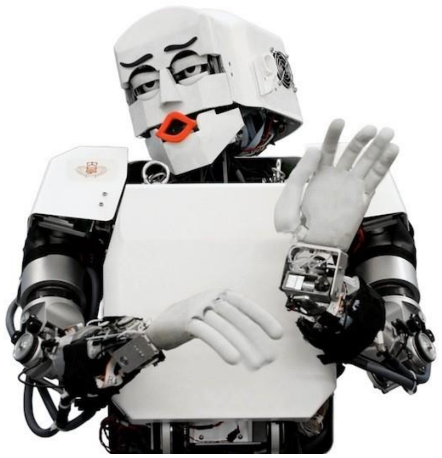
- The fitness function can be (and usually is) a <u>high-level performance metric</u>
- Low-level information regarding <u>how</u> the task should be solved is not necessary

  → Suitable for difficult problems for which we do not have an intuition
- You do not need to be able to <u>express</u> (or even be <u>aware</u> of) the <u>ingredients</u> that lead to high fitness
- As long as you can attribute a fitness score to each individual, the algorithm is able to maximize/minimize such a score
- In this field we just provide <u>the space of possible solutions</u> (task environment, encoding) and a <u>performance metric</u> to measure if a certain goal was met or not (fitness) → The algorithm is then free to find its way to reach the goal

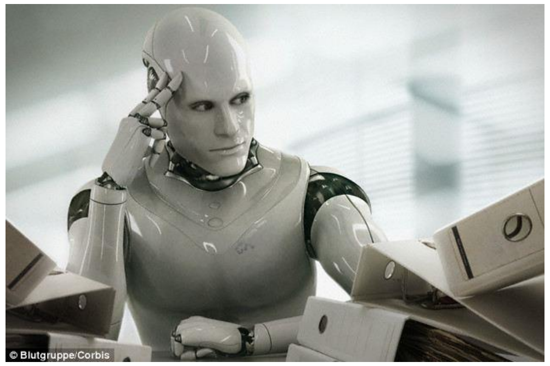
Fitness function - Observation

E.g. **Interactive evolution** (humans in the loop): a case in which fitness is usually high-level (and subjective)

- Fitness = User appreciation for an evolved picture/song (1 to 10) → May lead to an artistic agent, although we may not know how to mathematically define beauty, or how to produce a «beautiful» picture



- Fitness = Number of times the robot said/did something funny → May lead to a comedian robot, although we don’t know how to mathematically define what is funny and how to generate funny sentences



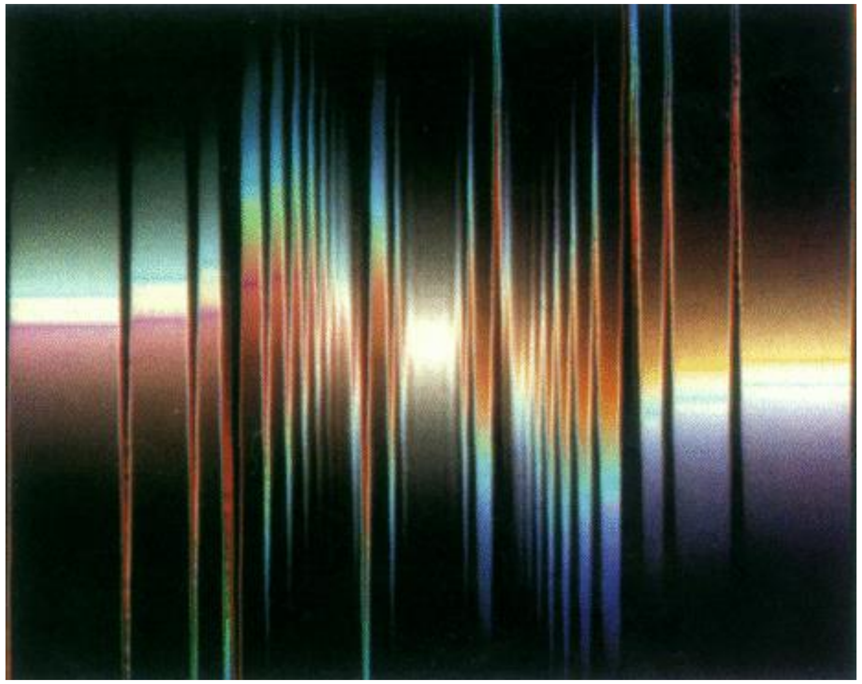
- «Number of times the robot appeared to behave intelligently» → May lead to an intelligent robot, although we don’t know how to define intelligence, nor how to implement it

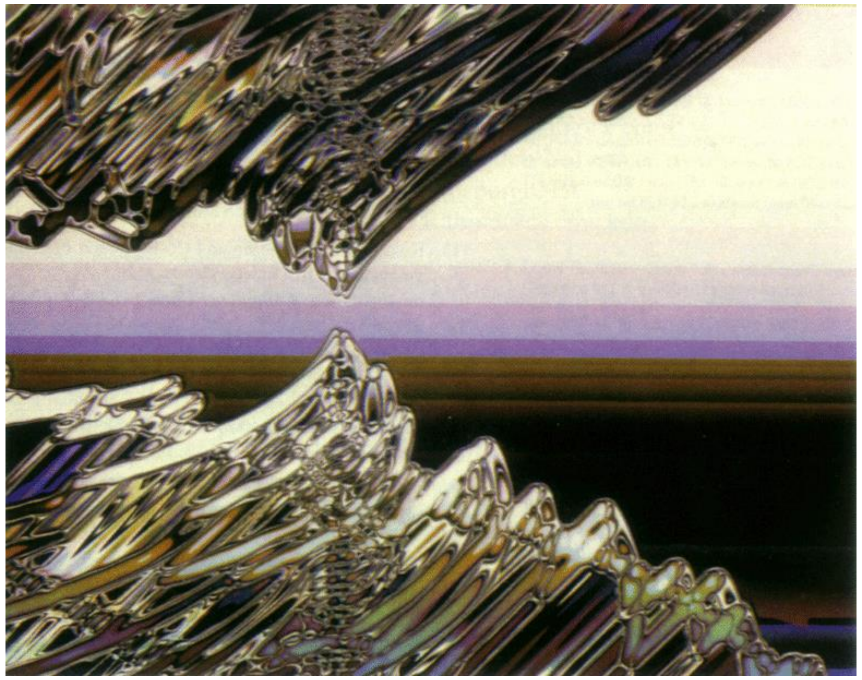
→ <u>Interesting, and general, from an AI standpoint</u> (creativity…)

One example: Interactive Evolution

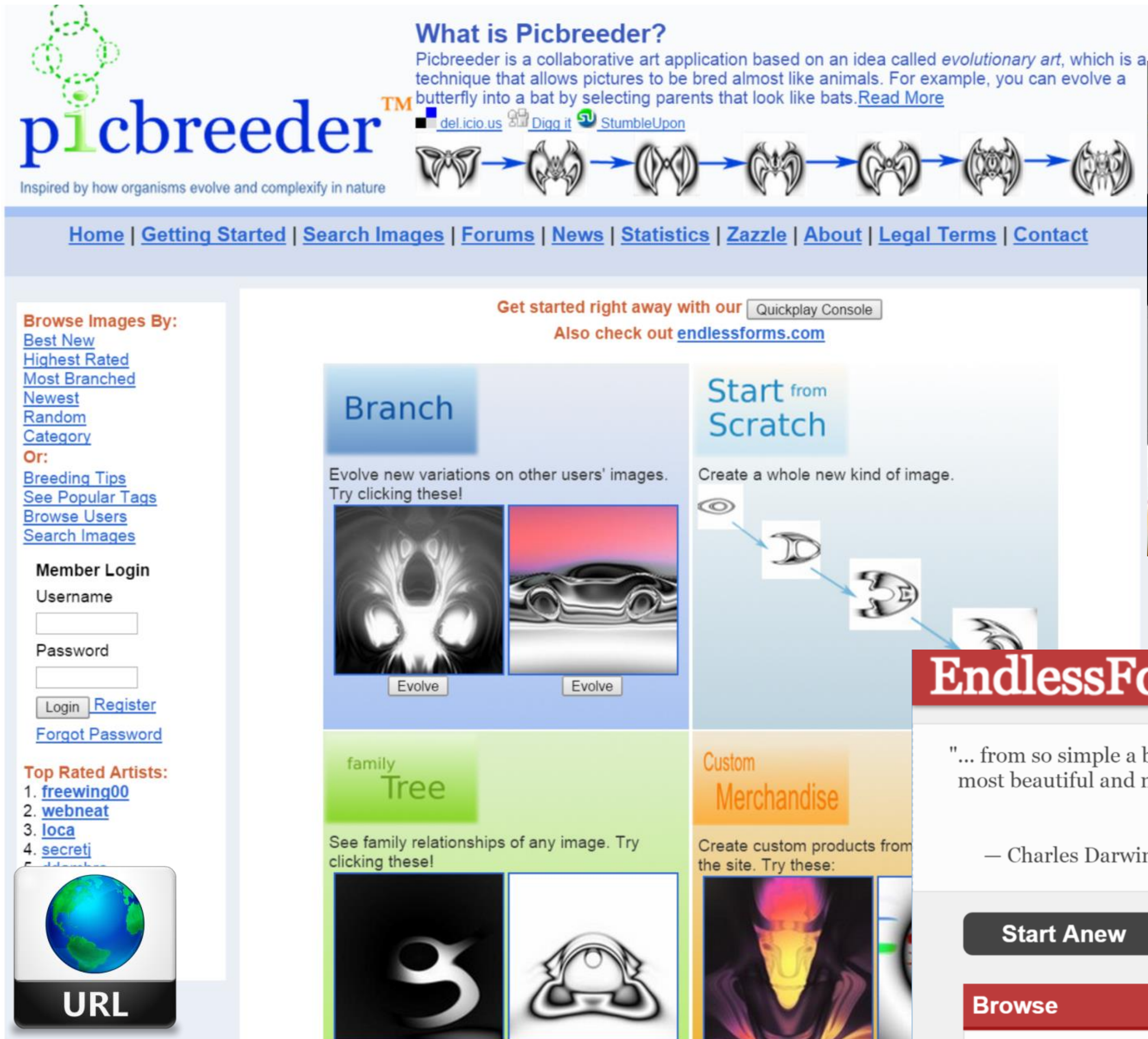
- *Karl Sim's "**Genetic Images**"* (1993) is a media installation in which visitors can interactively "evolve" abstract still images.
- A computer generates and displays 16 images on an arc of screens
- Pressure sensors are placed in front of each screen
- Fitness: how long people stand in front of each image → the longer, the higher the appreciation

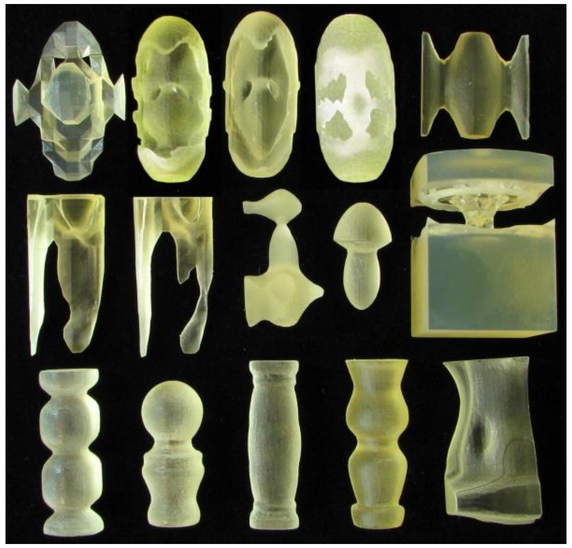
One example: Interactive Evolution

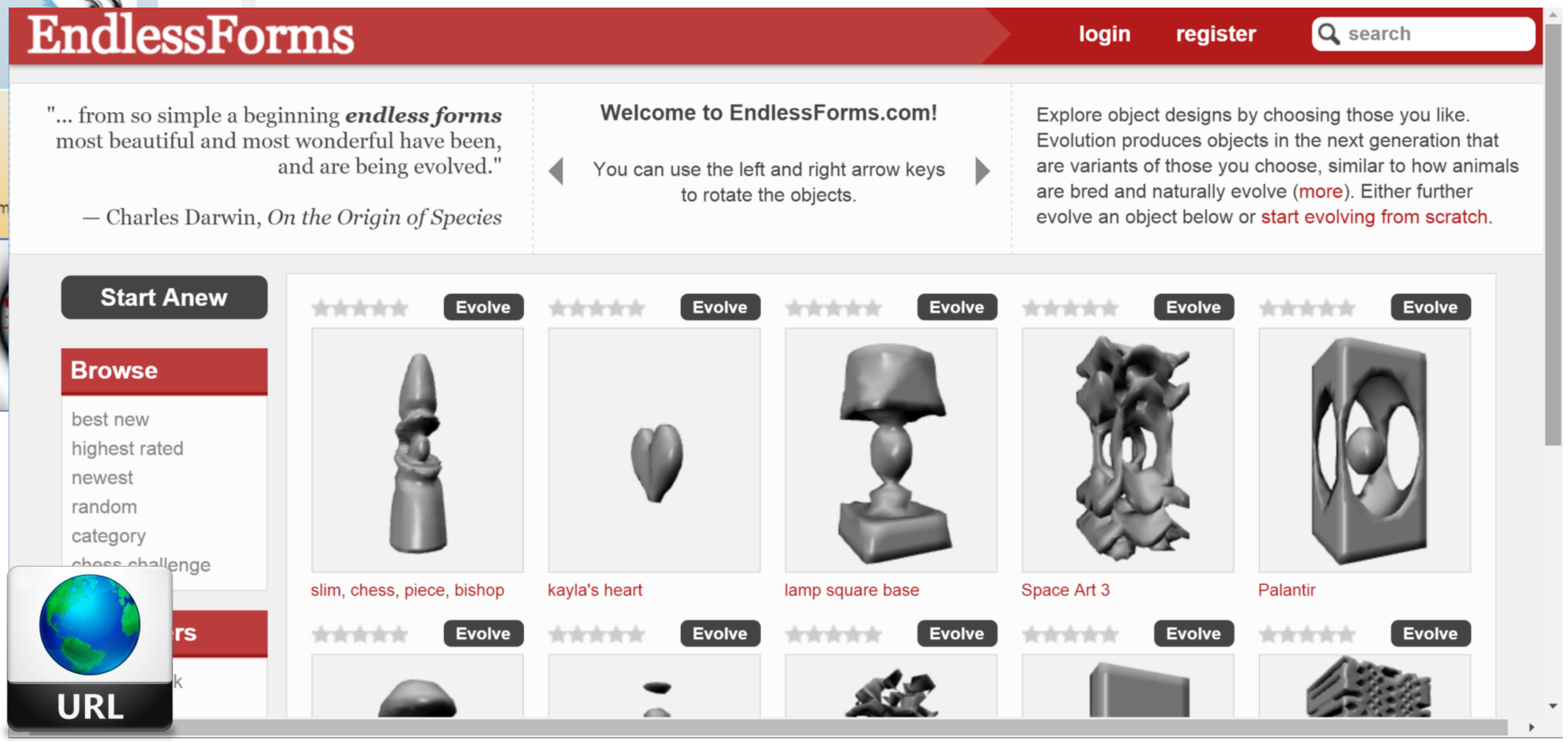


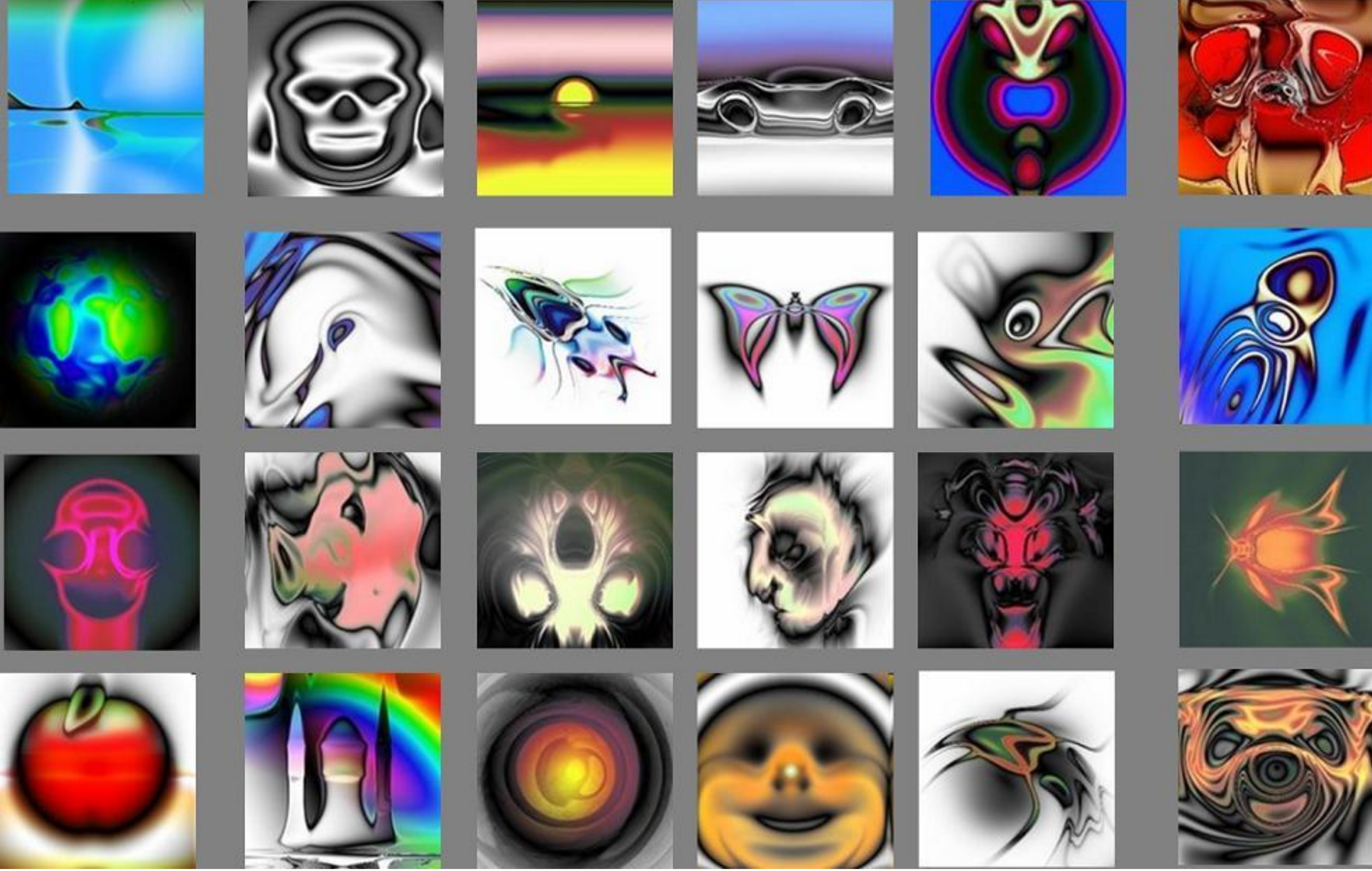


One example: Interactive Evolution



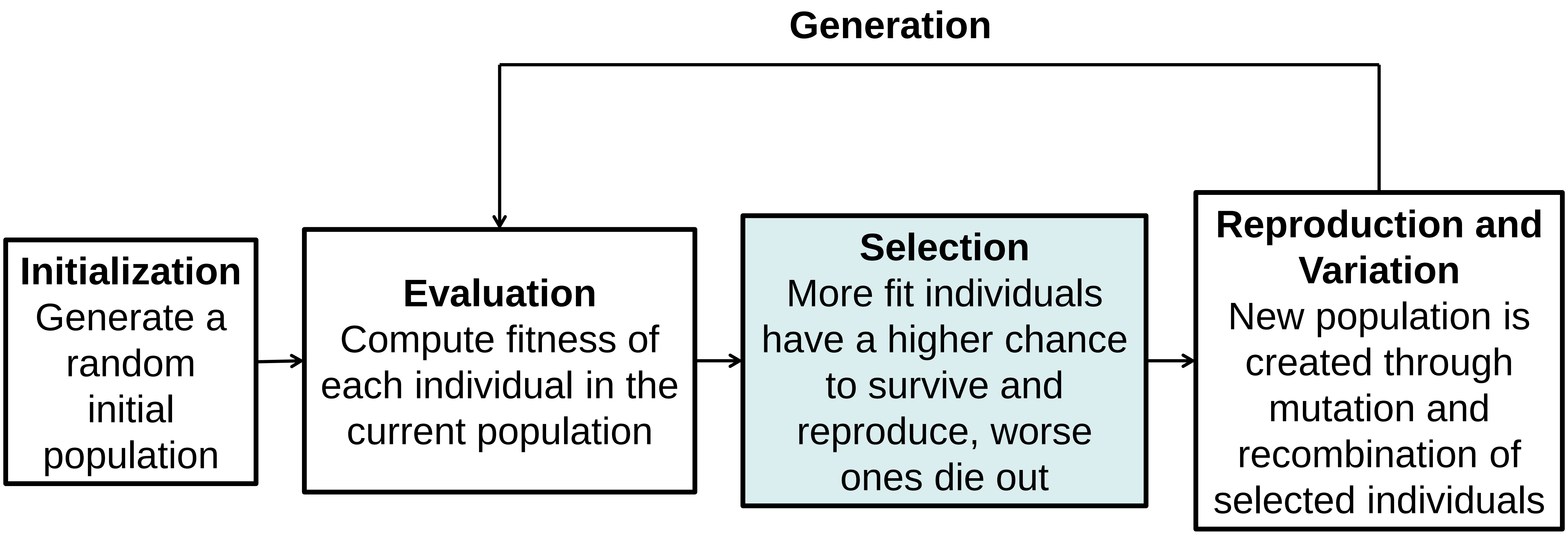






http://eplex.cs.ucf.edu http://picbreeder.org/

Genetic Algorithms – overall scheme



Selection and selection pressure

- Rationale: allocate a larger number of offsprings to the best performing individuals of the population
- Selection pressure: how difficult is for an individual to get a chance to reproduce
- High selection pressure: small % of individuals is selected for reproduction (e.g. only the very best ones)

  → **Rapid fitness improvement**, but **rapid loss of diversity**, risk of **premature convergence** to a local optimum

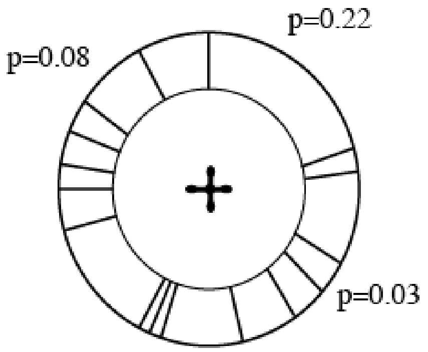
→ A **balance** is needed between selection pressure and factors that instead generate diversity (e.g. mutations)

- You should let less fit individuals reproduce too to maintain diversity
  - They may carry traits that will become successfull later on in evolution

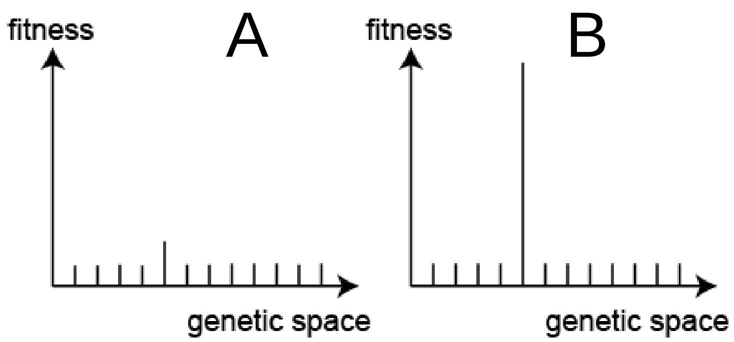
Proportionate selection (roulette wheel)

- The probability $p(i)$ of an individual i being selected for reproduction is proportional to its fitness relative to the overall population fitness (N is the population size):

$$p(i) = \frac{f(i)}{\sum_{k=1}^{N} f(k)}$$



- Like a *roulette wheel* where each slot corresponds to one individual of the population, and has a width that is proportional to $p(i)$ (and $f(i)$)
- To build the next generation, you spin the wheel N times (individuals can be selected several times)
- Works bad when: some individuals have remarkably bigger fitness than others (**selected almost every time → diversity loss, premature convergence**)



- **A solution**: ***fitness scaling*** (normalization)
- *Or…*

Rank-based selection

- Sort individuals based on their fitness value, from best to worst
- The place of an individual i in this sorted list is called **rank** $\boldsymbol{r(i)}$
- As in the *proportionate selection/roulette wheel*, but instead of the fitness value use the rank to determine the selection probability of individuals

→ Solves the problems mentioned for *proportionate selection*, given that the absolute value of the fitness does not directly determine the selection probability anymore

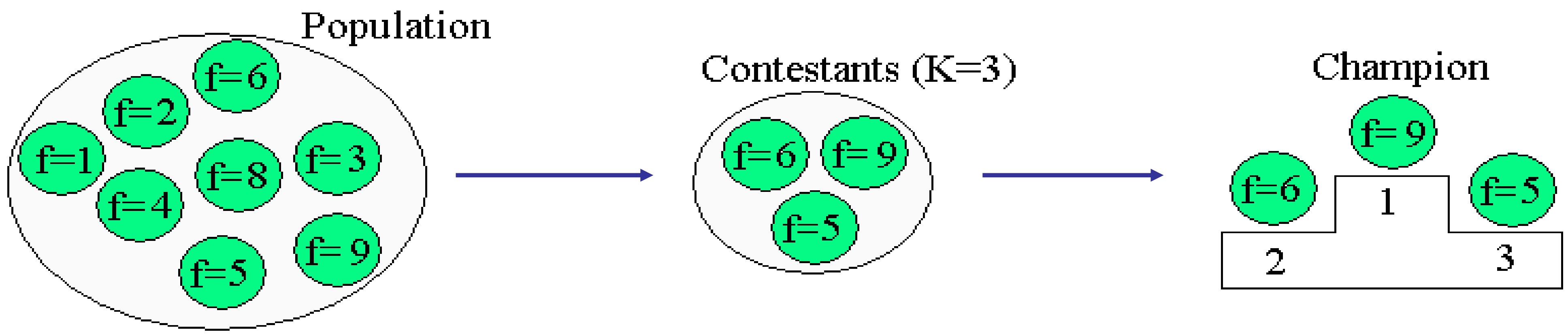
Truncated rank-based selection

- Select only the top n individuals based on their fitness
- Each of them will produce the same number of offsprings (N/n)
- E.g. $N = 100$, select top $n = 20$, $\frac{N}{n} = 5$ copies of each of the selected individuals will be used to form the next generation
- If n is not too small (would entail diversity loss → premature convergence), this method allows less fit individuals to produce the same number of offsprings as the fittest → maintains diversity

Tournament selection

- <u>For each new offspring to be generated:</u>
  - Randomly select a small subset of k individuals (*<u>contestants</u>*) of the current population
  - k is the *<u>tournament size</u>* parameter, the larger, the higher the selection pressure*)*
  - The individual that has the <u>best fitness among the contestants</u> wins and generates the new offspring
- Contestants can participate to multiple tournaments

→ <u>Good trade off between selection pressure and genetic diversity</u>



A glimpse of multi-objective optimization

- So far assumed <u>single-objective</u>
- Real optimization problems require, however, finding a <u>trade-off</u> between multiple (often antagonistic) objectives
  - <u>E.g.</u> maximize performances while minimizing energy expenditure
- When multiple antagonistic objectives are defined, there is <u>no single solution that optimizes all objectives at once</u>
  - <u>E.g.</u> fast but inefficient vs slow but efficient

→ Different solutions representing different trade-offs between the various objectives exist

A glimpse of multi-objective optimization – Pareto Optimality

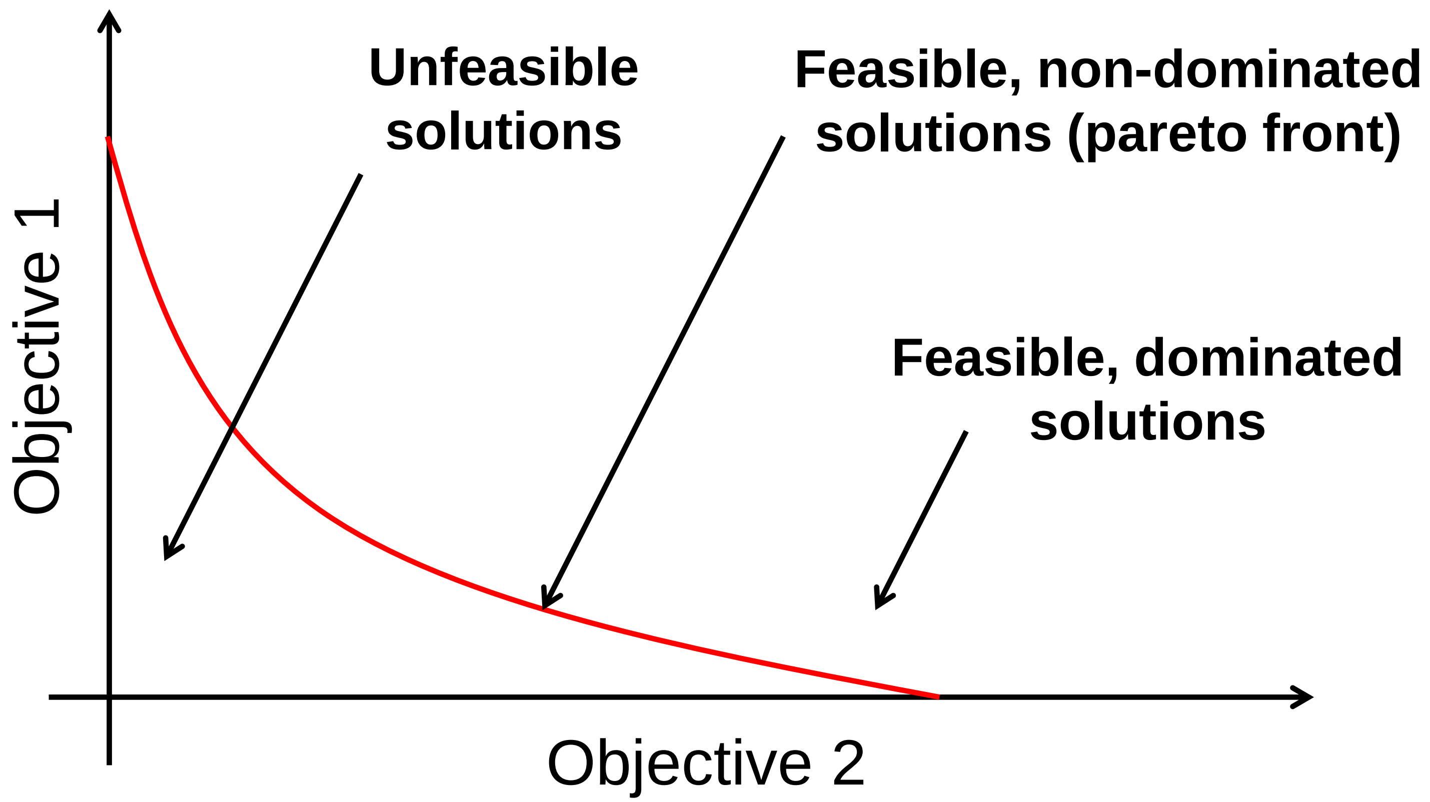
- A solution is pareto optimal (non-dominated) if there is no other solution in the search space that is better in all of the objectives
- Pareto front = set of non-dominated/pareto optimal solutions

→ Without additional subjective preferences, all solutions in the pareto front are to be considered equally good

- Rank = number of solutions that dominate S in one or more objectives

Example

Min objective1, objective2



A glimpse of multi-objective optimization – Pareto Optimality

- A solution is pareto optimal (non-dominated) if there is no other solution in the search space that is better in all of the objectives

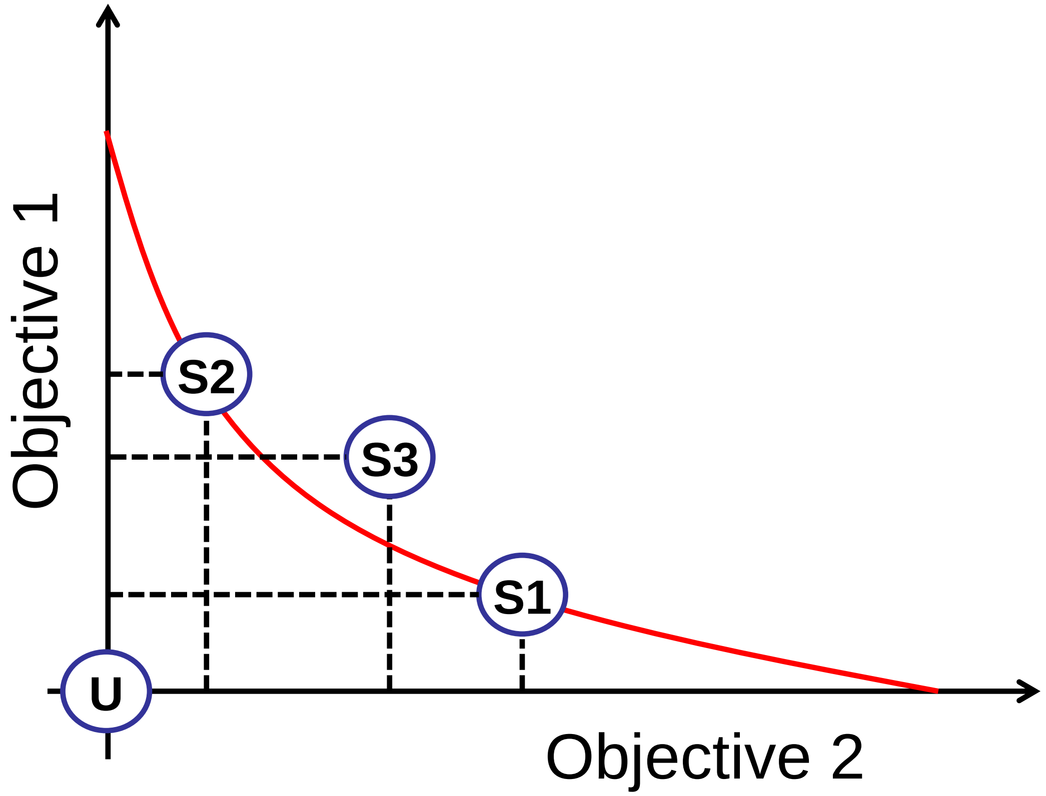
- Pareto front = set of non-dominated/pareto optimal solutions

→ Without additional subjective preferences, all solutions in the pareto front are to be considered equally good

- Rank = number of solutions that dominate S in one or more objectives

Example

Min objective1, objective2

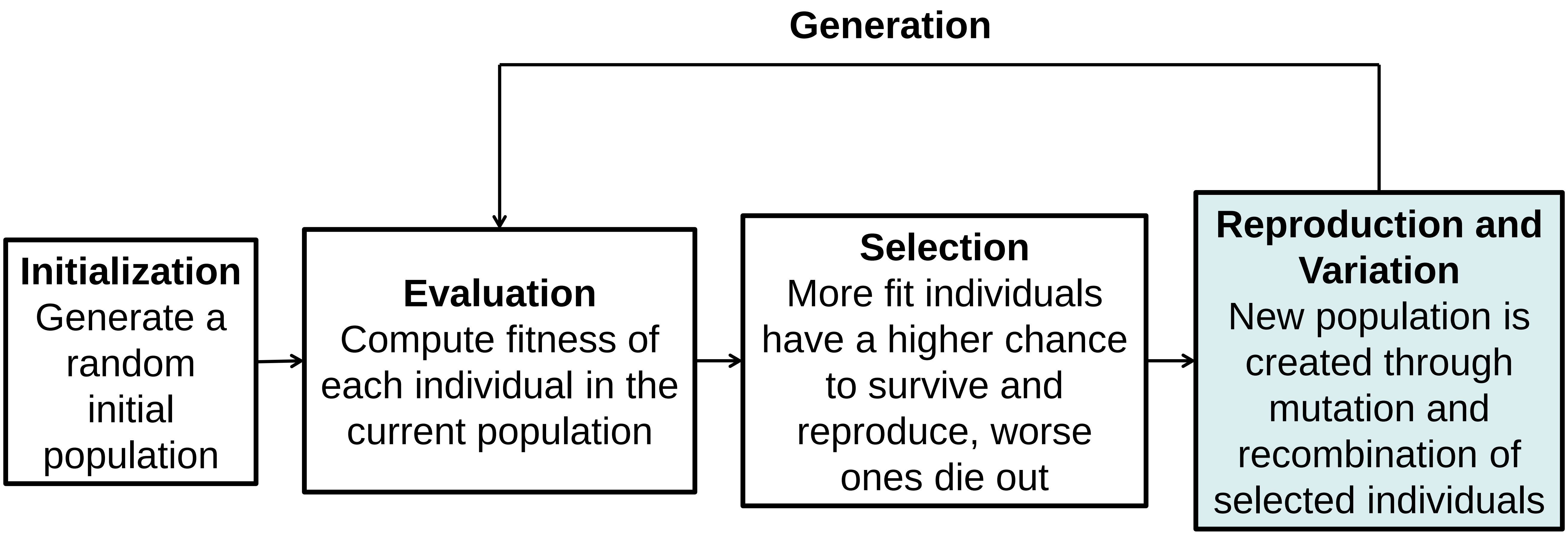


- U is what you would like ideally, but it is impossible to get («utopia»)
- Obj1(S1) < Obj1(S2), but Obj2(S2) < Obj2(S1)
  → Both are non-dominated, equally good → rank(S1) = rank(S2) = 0
- Obj2(S3) > Obj2(S2) → S3 is dominated by S2 in Obj2
- Obj1(S3) > Obj1(S1) → S3 is dominated by S1 in Obj1
- Rank(S3) is at least 2

A glimpse of multi-objective optimization

- The selection operators we have seen can be easily adapted to implement multi-objective EA
- E.g.
  - *Truncated rank-based selection: pareto rank* can be used instead of fitness-based rank to sort the population
    - Tries to let non-dominated individuals reproduce first (pareto front), then individuals with rank 1, 2, etc.
  - The winner in *tournament selection* can be determined based on the concept of *pareto dominance* instead of being just based on fitness

Genetic Algorithms – overall scheme



Genetic operators

- Capture the biological effect of *mutations* and *recombinations* on the genotype observed in the natural evolution
- Must match the genetic representation:
  - Genotype is binary → genetic operators must manipulate bitstrings
  - Genotype is based on networks → genetic operators must manipulate networks
  - Custom encoding/data structures → custom genetic operators are needed
- Introduce *diversity* and produce *innovation* by altering and combining individuals in the population
- Determine the tradeoff between exploration and exploitation

Genetic operators – Crossover/recombination

- Emulates the recombination of genetic material from two parents
- After selection, pairs of individuals are randomly formed…
- …and their genotypes are combined with a given probability p_c
- Crossover should allow to effectively merge partial solutions from the parents into an offspring that performs better than both of the parents with a probability $p > 0$

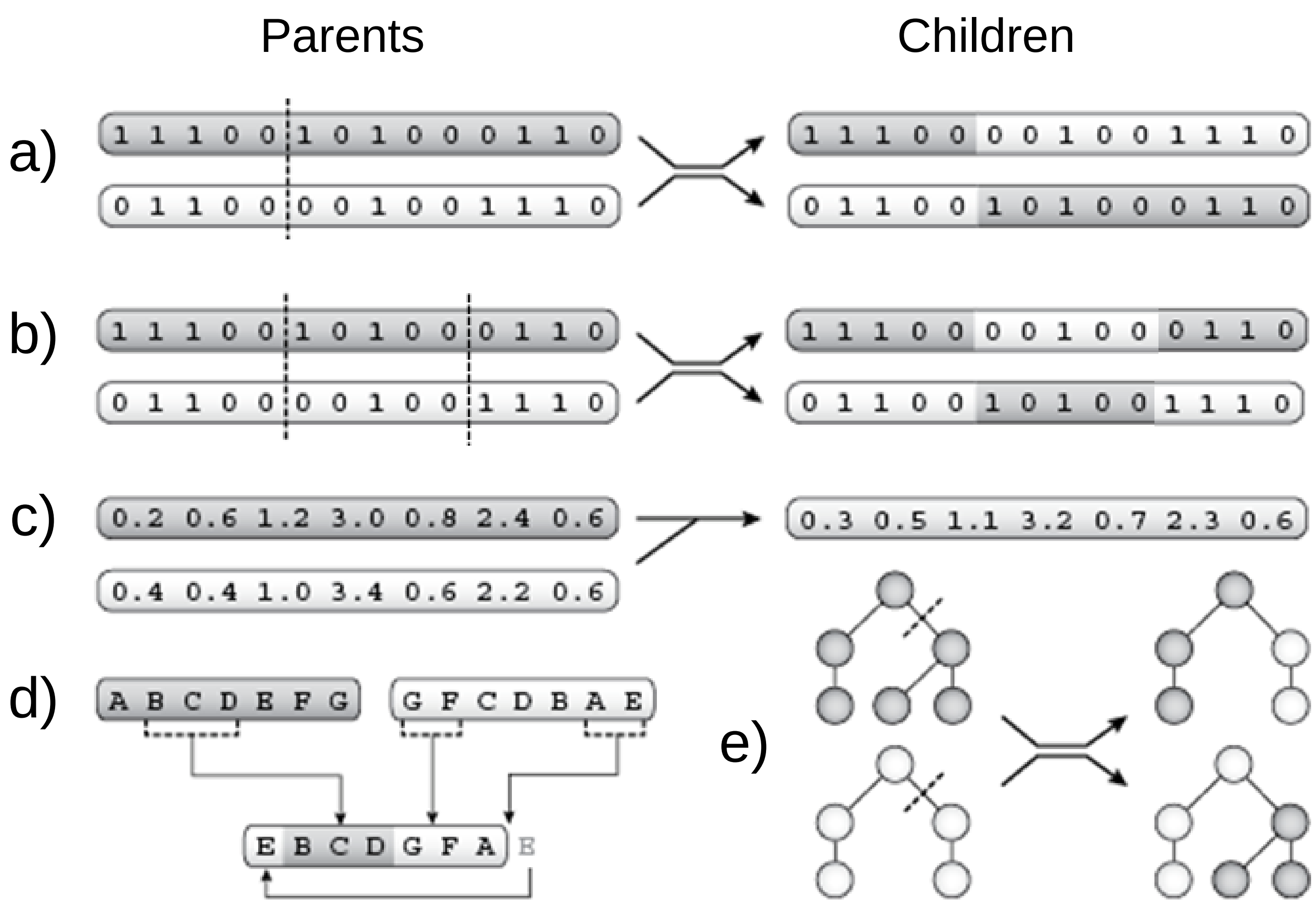
Genetic operators – Crossover/recombination

<u>Discrete/real valued encodings:</u>

a) **one-point**: randomly select a *<u>crossover point</u>* and swap chromosomes around that point

b) **multi-point**: as before, but selecting n *crossover points* (here $n = 2$)

c) **arithmetic**: creates a single offspring by combining the two genomes at n random positions (e.g. AND/OR for binary coded, average, or convex combination for real-coded, etc)



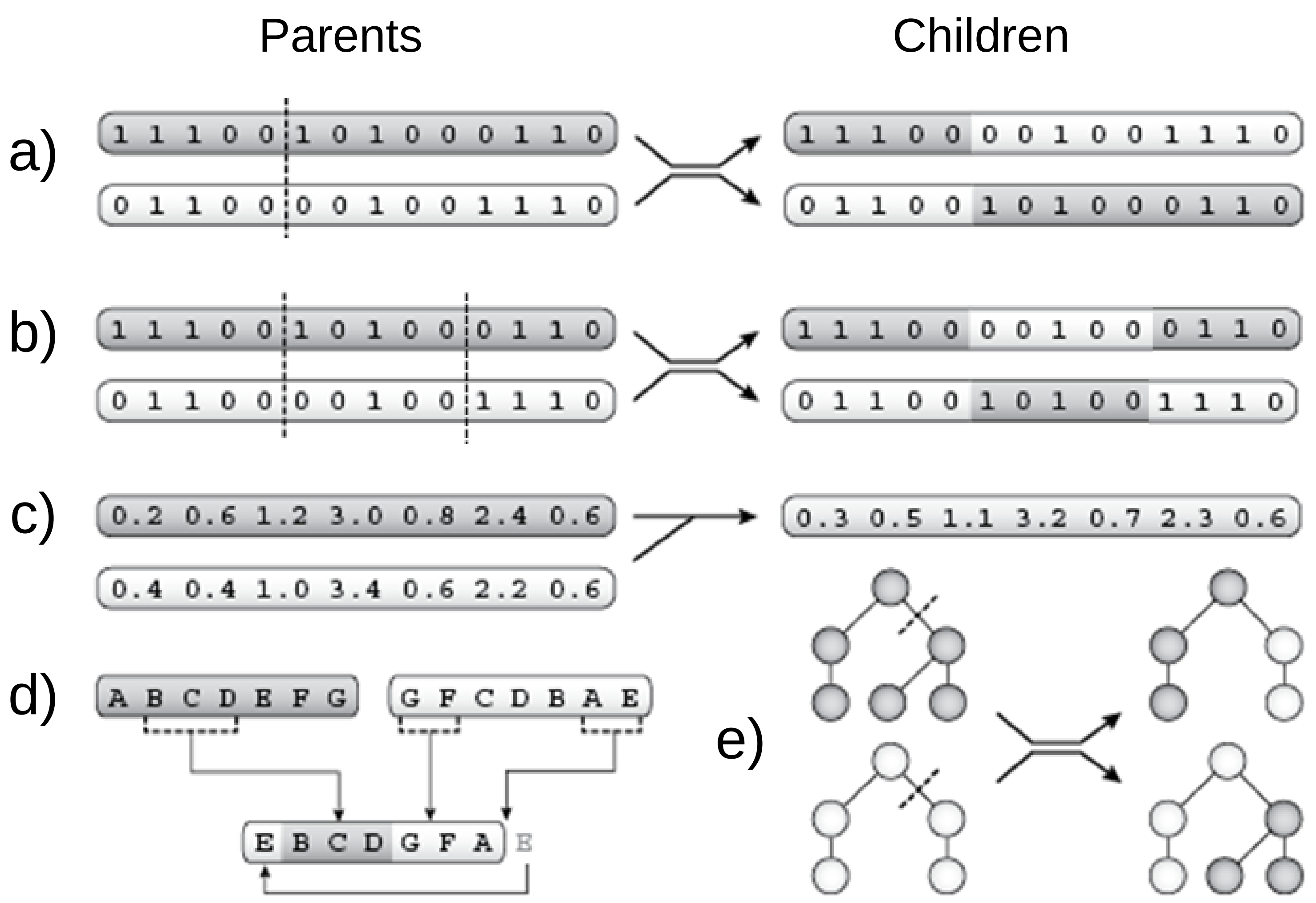
Genetic operators – Crossover/recombination

<u>Crossover for sequence encoding</u> *(all symbols must occur once and only once):*

d) Randomly copy a part of the sequence from one parent, then fill-in with remaining elements in the order in which they appear in the other parent

<u>Crossover for tree/network encoding:</u>

e) Randomly select a node of each parent, and exchange the two corresponding subtrees

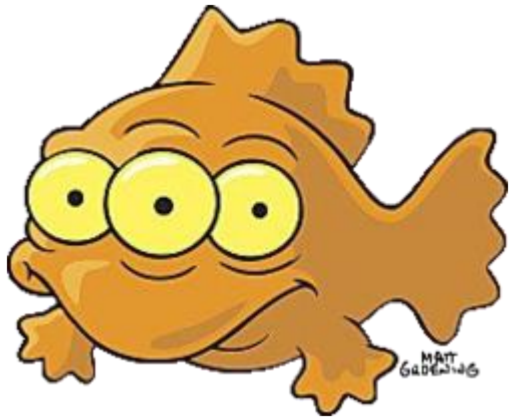


Genetic operators – Crossover/recombination

- Crossover is a non trivial operation: it should <u>isolate and recombine functionally-relevant chunks of the genome of the two parents</u>
  - Not easy to guarantee this property
  - When crossover does not work properly, <u>it can act as a very large mutation</u> → The effect of these mutations is usually detrimental (Fisher) → Poor evolvability

→ Statistically checking the effect of crossover on fitness can lead to insights regarding the behavior of the algorithm (e.g. % of fitness increase after crossover)

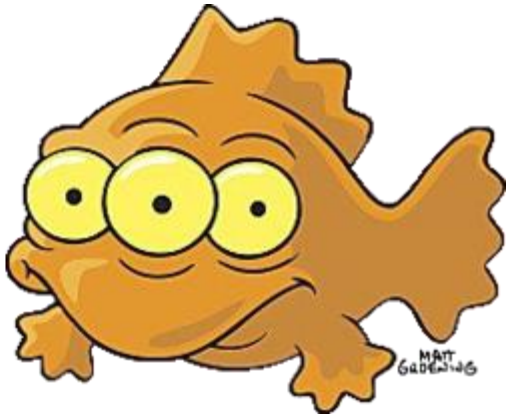
Genetic operators – Mutation



- Operates on a single individual at a time
- Applies small *random* modifications of the genotype

  → Fisher: probability of a mutation being beneficial is inversely proportional to its magnitude
- Allows evolution to explore *variations* with respect to the current solutions

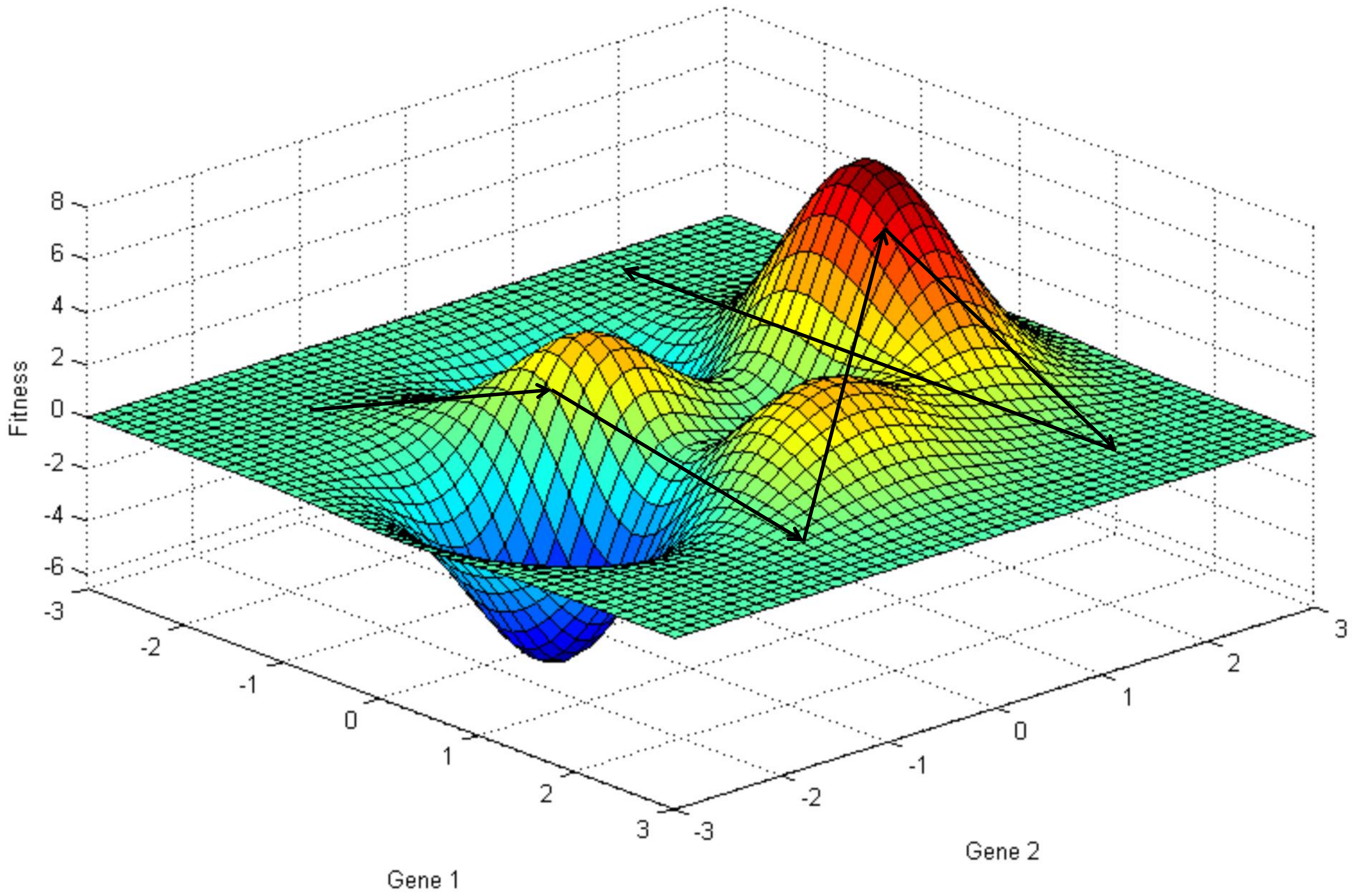
Genetic operators – Mutation



- **Mutations are useful to**:
  - Produce diversity
  - Promote exploration
- However, too disruptive mutations can slow down the search, as the algorithm can end up not benefiting from previously discovered solutions → random search

→ Proper tuning is necessary, analysis of the effect of mutations on the fitness can be useful (% of beneficial/detrimental mutations)

Bad mutation size



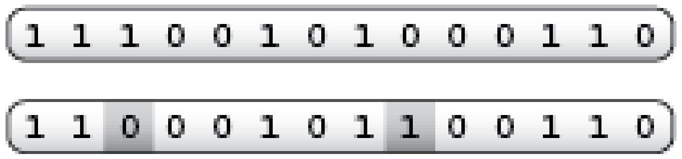
Genetic operators – Mutation

Mutation in simple encodings = change the content of each gene with probability p_m (e.g. $p_m = 0.01$)

a) Binary encoding: toggle bit values

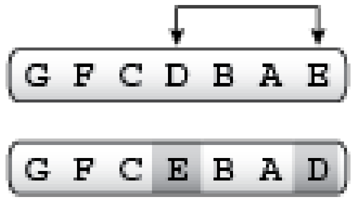
b) Real-valued encoding: add random noise (e.g. from a Gauss distribution $N(0, \sigma)$ → most mutations are small, few are big. Note that σ is an additional parameter)

a) Binary genotypes

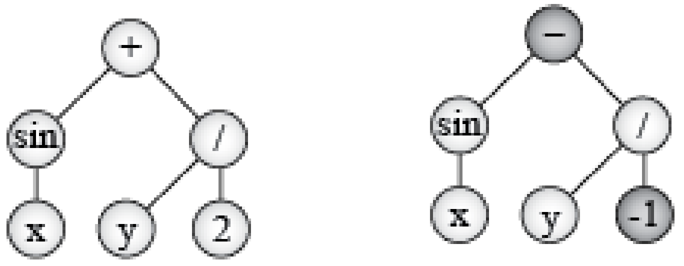


b) Real-valued genotypes

c) Sequence genotypes



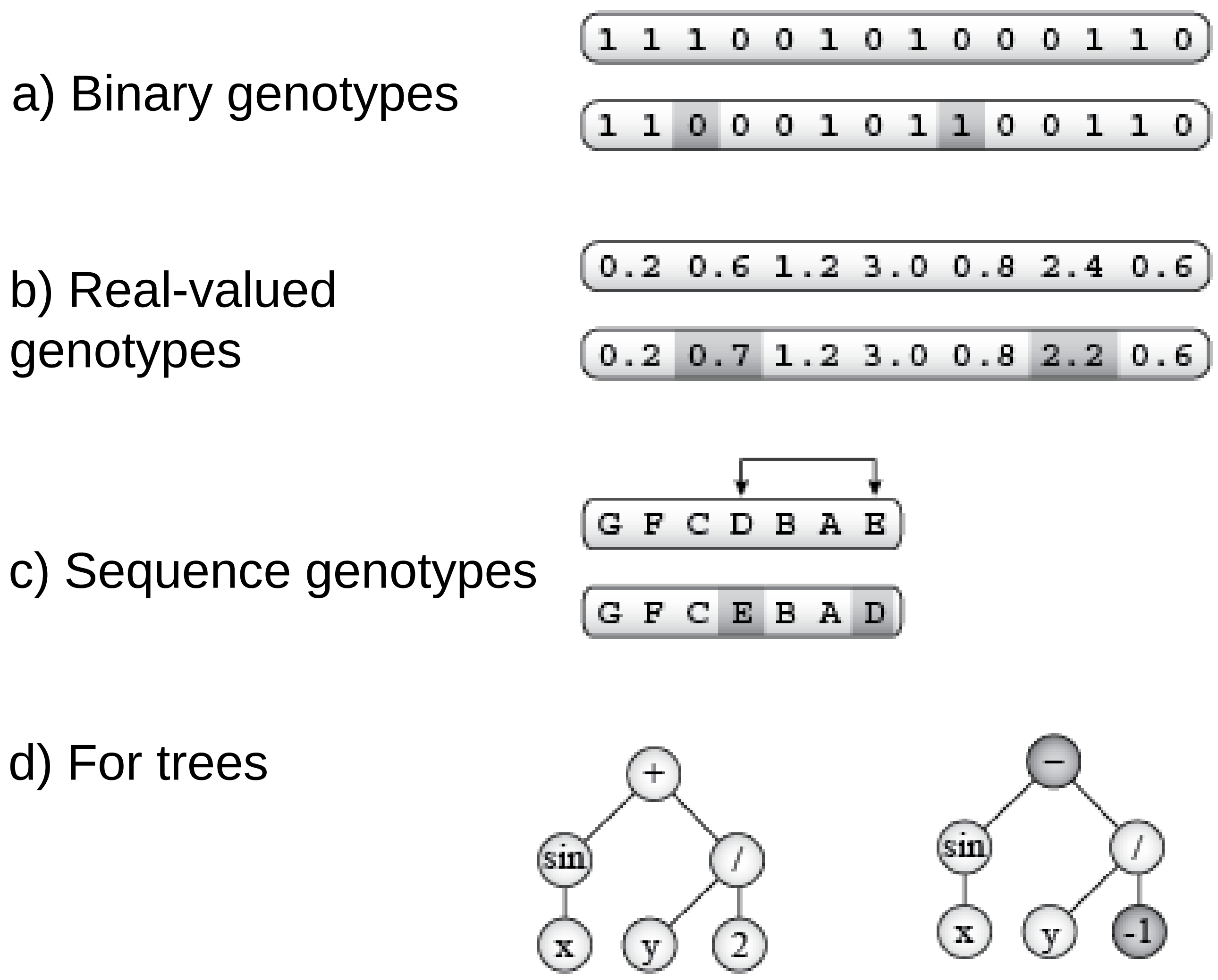
d) For trees



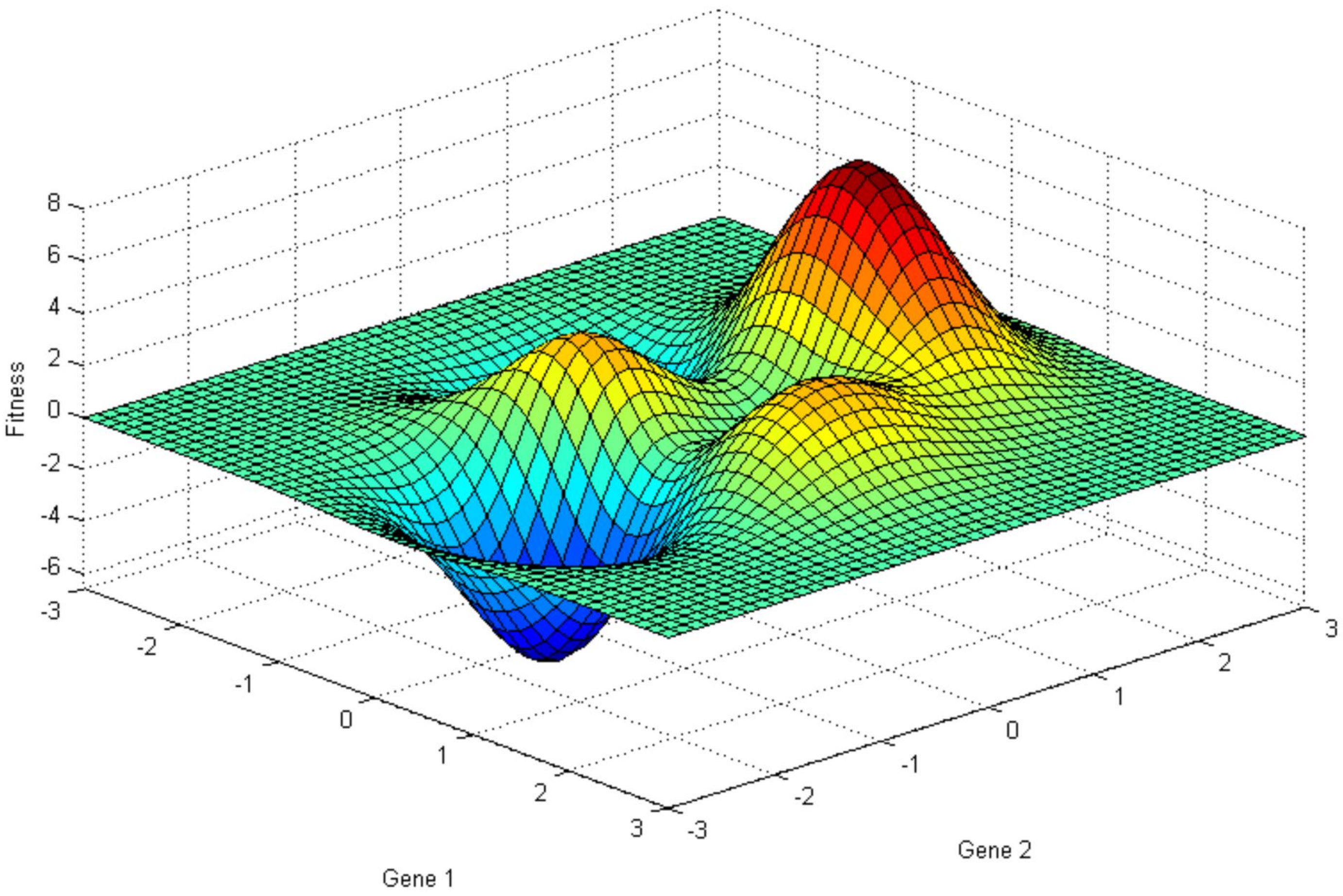
Genetic operators – Mutation

c) Sequence encoding: swap the contents of two randomly chosen genes

d) Tree-based/network encoding: change the value of a node with another from the same set (functions set/terminals set) with the same number of leaves → tree-structure unchanged



Genetic operators and fitness landscape



It is important to note that the fitness landscape is seen/navigated through genetic operators

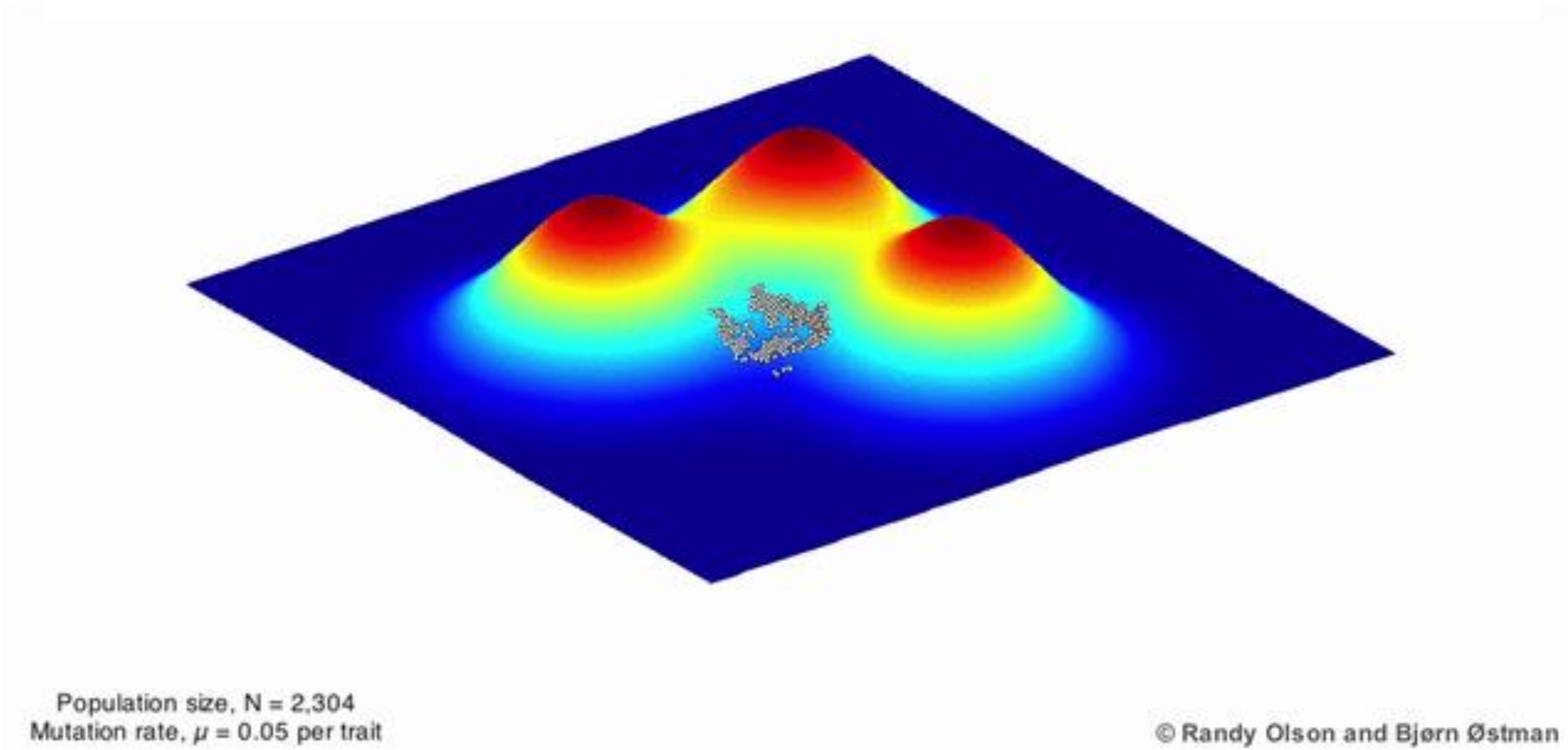
Paths towards optimal solutions might be less linear than we may think

A simple problem (smooth fitness landscape) can become difficult if genetic operators are not properly implemented/tuned

(we saw an example talking about mutation size)

Genetic Algorithms – redistribution of efforts

Differently from e.g. hill climbers, genetic algorithms can *redistribute efforts* (i.e. the costly fitness evaluations) during the search procedure as promising areas of the search space are discovered



How to conduct an evolutionary experiment

- Evolutionary algorithms involve some degree of randomness (random initial conditions, random mutations, probabilistic selection…)
- …but we don't need random observations (e.g. a particularly lucky or unlucky run is not very informative). We want to study actual phenomena

→ It is necessary to account for this stochasticity when conducting and analyzing evolutionary experiments

How to conduct an evolutionary experiment

Multiple runs:

- Consider your evolutionary experiment as a stochastic process
- In order to analyze it, you need multiple observations
- Computers simulate randomness with deterministic algorithms *(pseudo-random number generators)*
- Once provided with an initial "seed", the sequence of "random" events simulated by a computer is completely deterministic (and so is the history of your evolutionary algorithm)

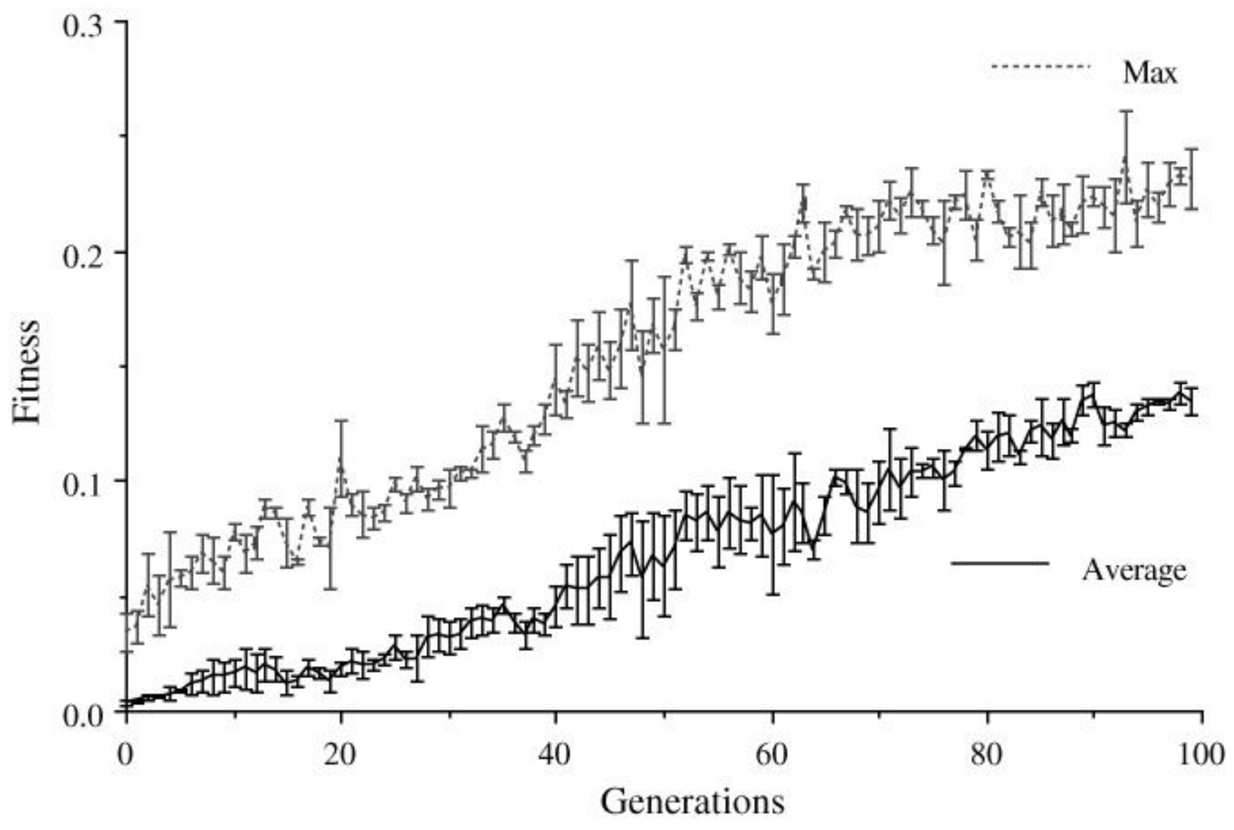
→ Perform different runs providing different seeds to your random number generator

→ Saving and loading the state of the random number generator allows reproducing results (crucial)

How to conduct an evolutionary experiment

Combining and reporting results:

- The first and most informative plot you should produce is the fitness plot, showing how fitness varies over evolutionary time (generations)
- At each generation you can report:
  - average of the best fitness in the population, across multiple runs
  - average of the average fitness in the population, across multiple runs
- Report variability plotting the standard deviation or the 95% confidence intervals

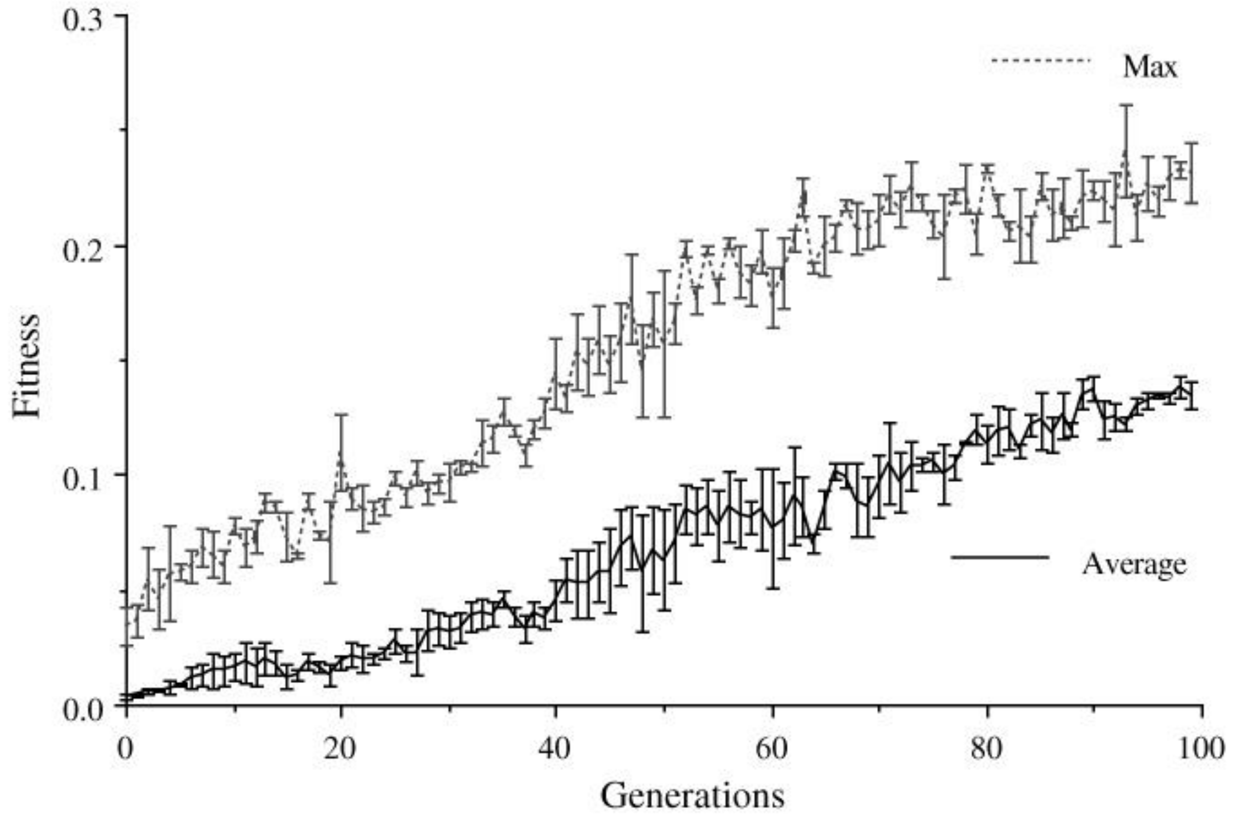


How to conduct an evolutionary experiment

<u>Statistical analysis:</u>

- Due to the aforementioned stochasticity, the <u>statistical significance</u> of every statement and comparison arising from your evolutionary simulations <u>should be tested</u>
- <u>p-value</u>: ~probability of your statement being not significant (at least $p < 0.05$ is usually required → 95% confidence)
- E.g.

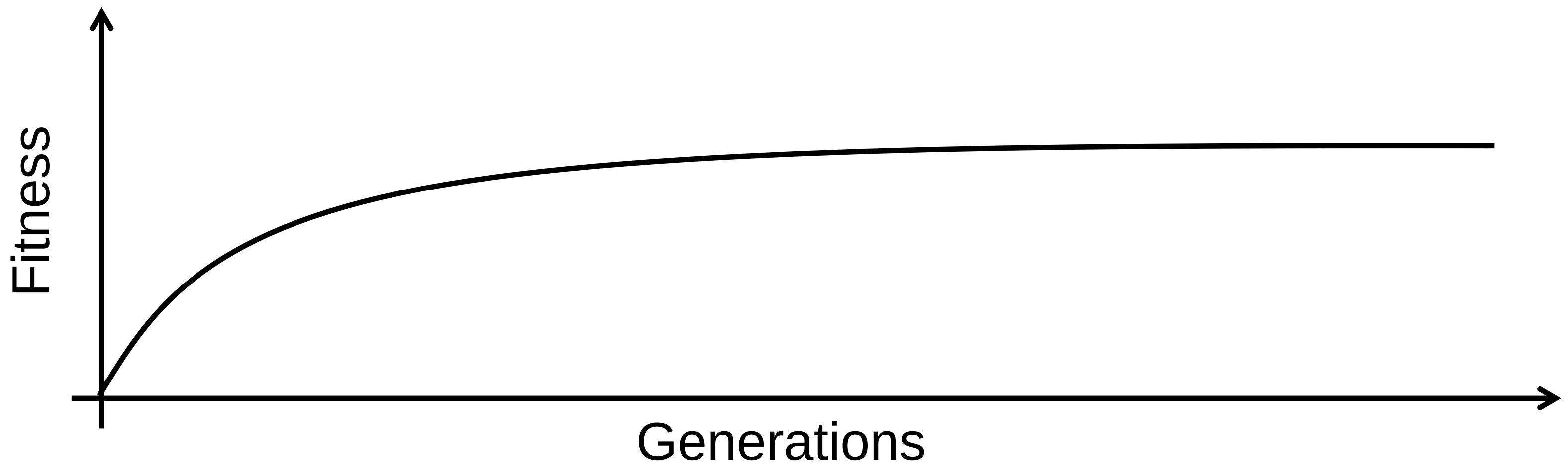
  «AlgorithmX outperforms AlgorithmY»: meanBestFitnessX > meanBestFitnessY with p<0.05
- Different methods to compute that



Convergence, stagnation, *neutral paths*

- The convergence of an evolutionary algorithm can be observed when the fitness plot reaches a *plateau*

  → "***Stagnation***", the fitness does not improve over generations, the algorithm is not making any progress



- Stagnation is often due to ***diversity loss*** (population becomes relatively homogeneous, genetic operators are not able to produce enough variation to produce innovation) → ***premature convergence*** to a sub-optimal solution
- **Diversity preserving** mechanisms can help avoiding this problem (e.g. some are based on injecting new random individuals every generation)

Types of evolutionary algorithms

- **Genetic Algorithms** (GA) - Holland, 1975
  Binary genotypes, crossover and mutation
- **Genetic Programming** (GP) - Koza, 1992
  Tree-based genotypes, crossover and mutations
- **Evolutionary Programming** (EP) - Fogel etal., 1966
  Real-valued genotypes, mutations, tournaments, gradual pop. replacement
- **Evolutionary Strategies** (ES) - Rechenberg, 1973
  As EP + mutation range encoded in genotype of individual
- **Island Models** – Whitley et al., 1998
  Parallel evolving populations with rare migration of individuals
- **Steady-State Evolution** – Whitley et al., 1988
  Gradual replacement: Best individuals replace worst individuals
- …

Some pros and cons

Cons:

- Weak theoretical basis
- No guarantees regarding the success and/or the time to get a solution
- Parameters tuning is needed
- Often computationally expensive

Some pros and cons

Pros:

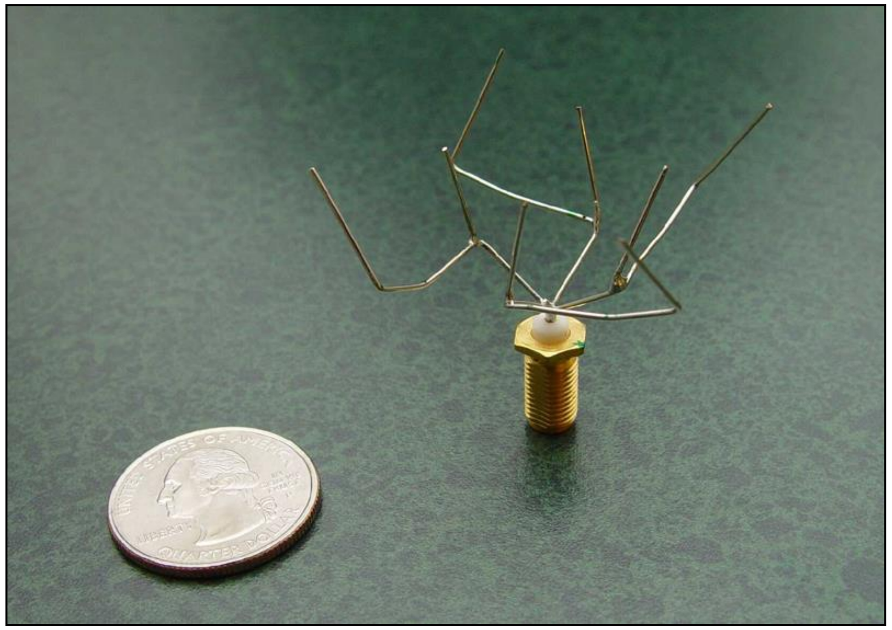
- Evolutionary algorithms can work where other optimization techniques cannot (e.g. discontinuous, noisy fitness functions) → robust
- Can be easily extended to deal with multi-objective, constrained problems
- Inherently parallel structure (evaluation of a population)
  → Easy to parallelize

Some pros and cons

Pros:

- They are able to solve <u>extremely difficult problems</u> requiring <u>very little knowledge and supervision</u>
- They often produce solutions that are <u>extremely effective (more effective than human-devised ones) yet completely counter-intuitive</u>
  → They "think" outside the box

Human-competitive design and «perverse instantiation»



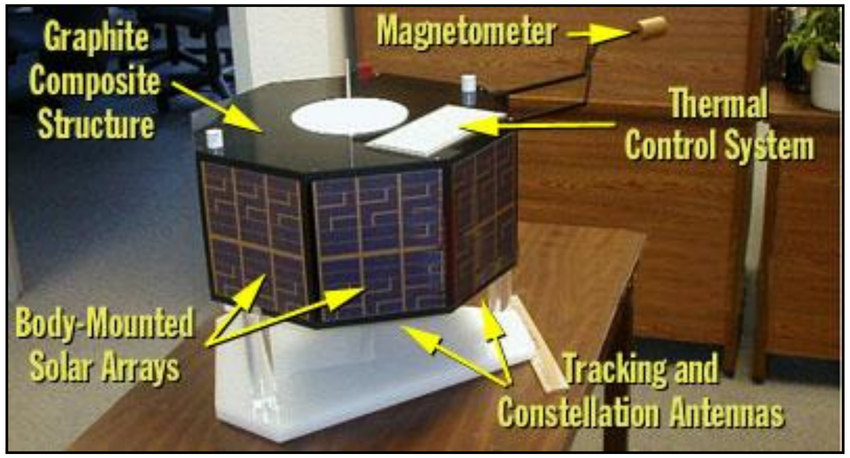
From «How The Body Shapes The Way We Think – A new view of Intelligence» (R. Pfeifer & J. Bongard)

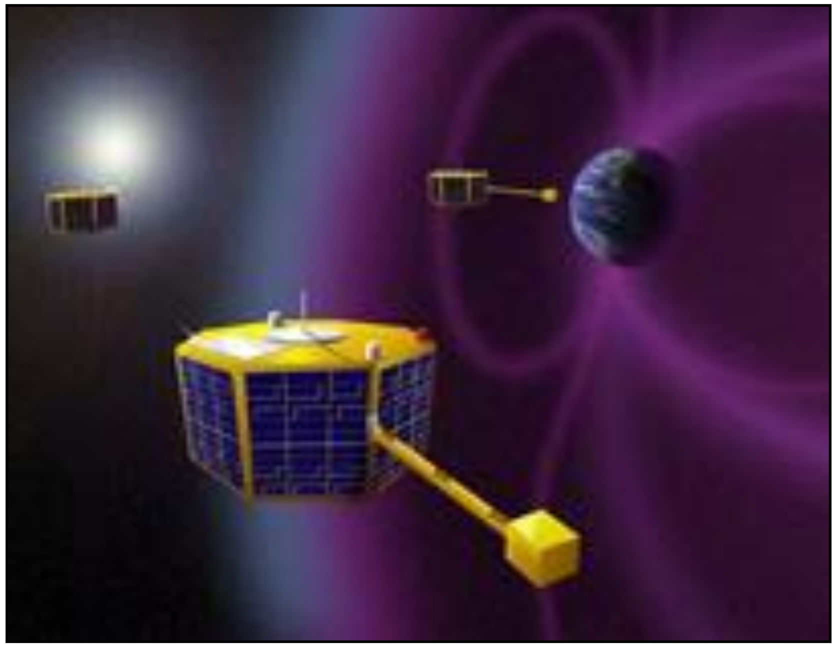
NASA's Antenna

Human-competitive design of an antenna for nanosatellites, NASA [Lohn, Hornby, Linden, 2004]

- Meeting several performance requirements (gain, sizes, operational frequencies, …) is very challenging for humans
- NASA automated its design by using evolutionary techniques

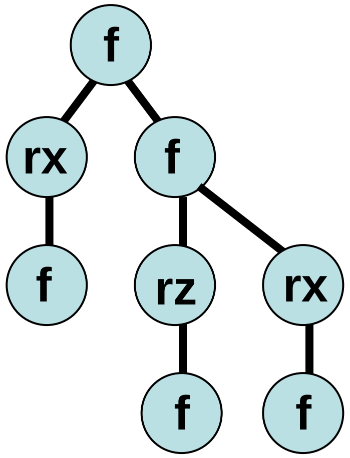
- Wiki page
- Paper

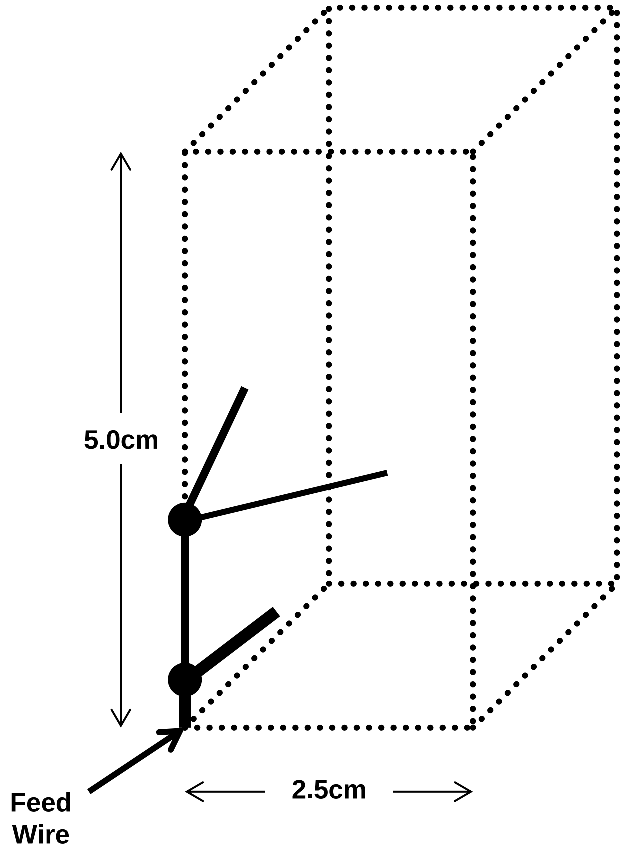




NASA's Antenna

- Tree-based encoding, instructions to "grow" (draw) an antenna
- Function set:
  - f=forward(length)
  - rx/y/z(angle)
  - Terminals: length, angles
- Technical specs tested in simulation
- Best designs were built and tested in the real world

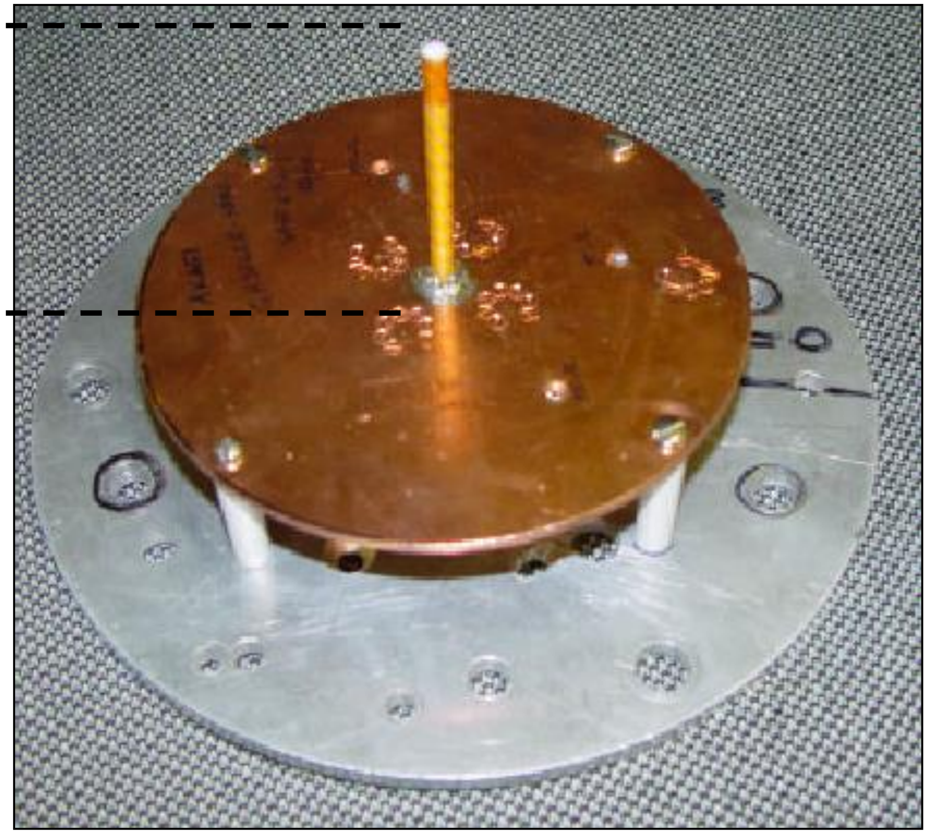




NASA's Antenna

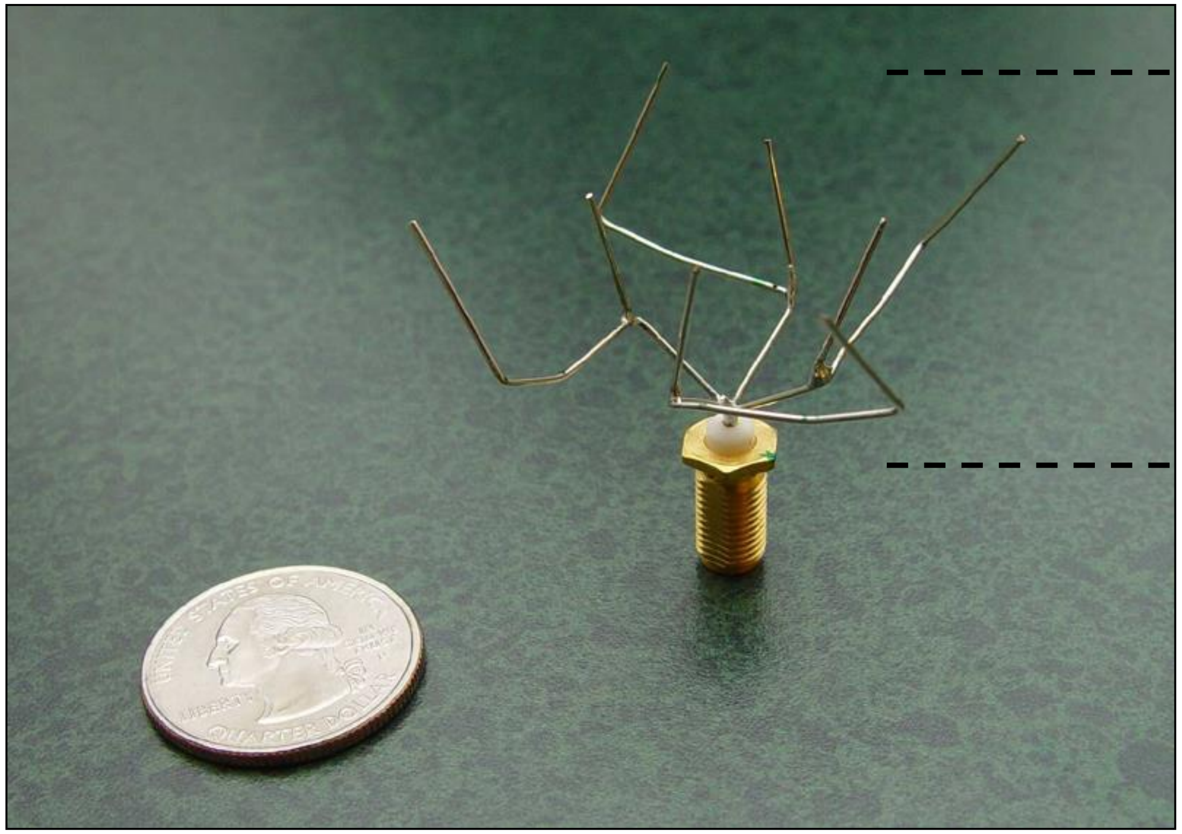
Human

~12 cm



Evolved

~ 5 cm



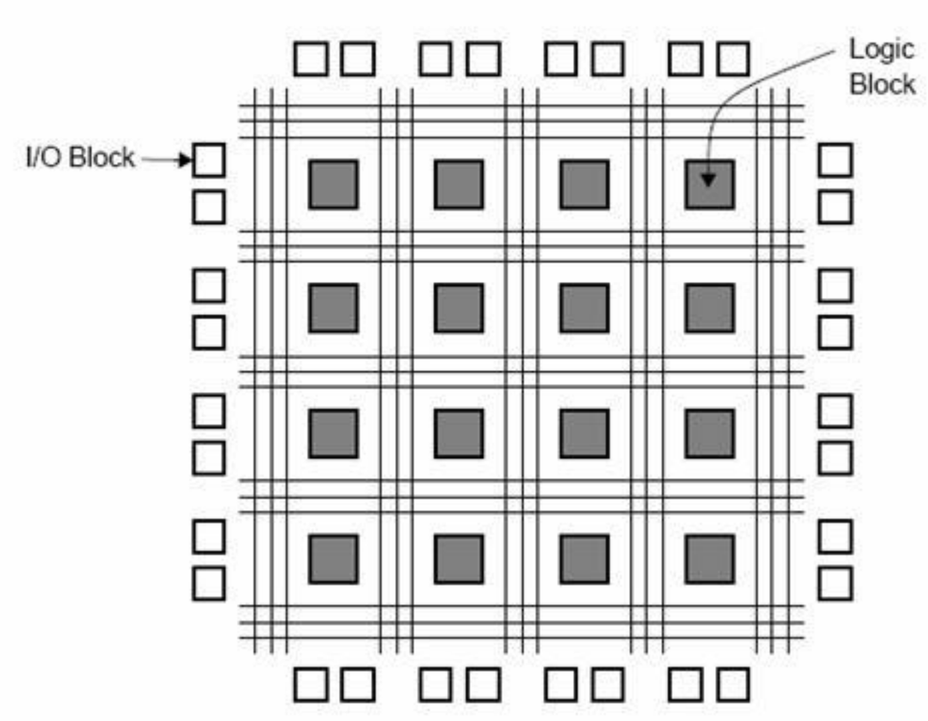
Evolved design is completely not intuitive, but <u>considerably smaller</u> and <u>far superior</u> in terms of performances

→ <u>Launched on board of the ST-5 satellite in 2006</u>

Evolvable hardware

Adrian Thompson, Sussex University, 1996

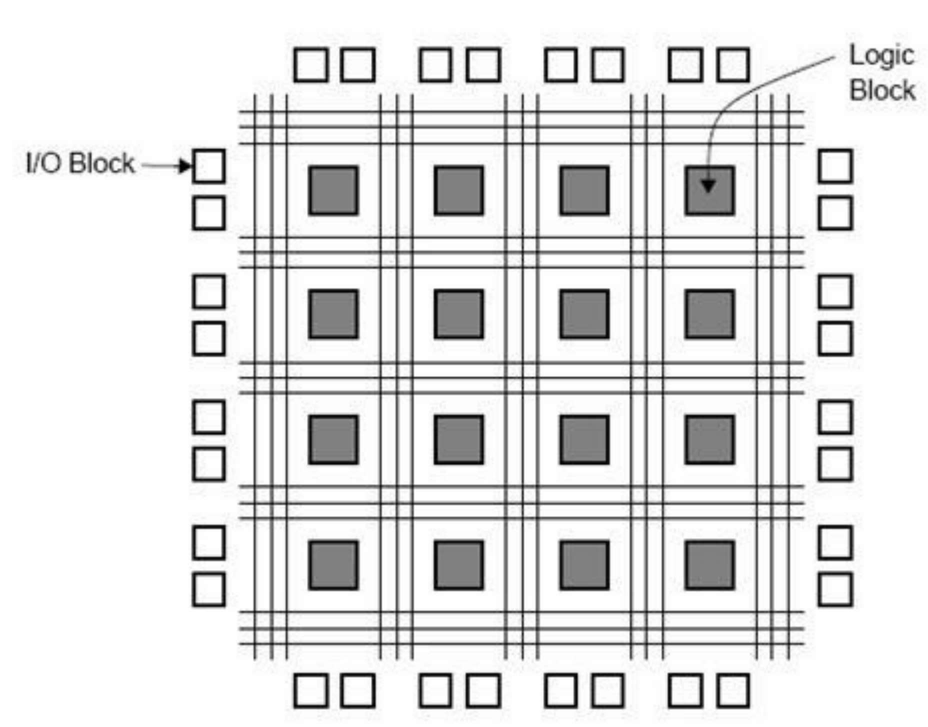
- Evolutionary algorithms designing FPGA-based circuits (modular programmable system)
- Goal: evolve a circuit to distinguish between a low and a high input sound
- Evolution in the real world
- An effective circuit was evolved, that worked properly but…



Evolvable hardware

Adrian Thompson, Sussex University, 1996

- Evolutionary algorithms designing FPGA-based circuits (modular programmable system)
- Goal: evolve a circuit to distinguish between a low and a high input sound
- Evolution in the real world
- An effective circuit was evolved, that worked properly but…
- **…once re-created on a custom chip, only considering the FPGA components connected in the design… → Did not work anymore**



Evolvable hardware

- It was found out that the original evolved circuit was exploiting weak electromagnetic interactions between the active components and the disconnected ones (that are usually assumed not to play any role)

- The solution devised by evolution broke the human-imposed modular design, exploiting to its benefit phenomena of the ecological niche that are usually regarded as undesired

Evolvable hardware

Another experiment in Sussex, by Jon Bird and Paul Layzell

- Goal: evolve a circuit producing an oscillatory signal without having an internal clock

Evolvable hardware

Another experiment in Sussex, by Jon Bird and Paul Layzell

- Goal: evolve a circuit producing an oscillatory signal without having an internal clock
- Evolved solution: instead of an oscillator, something like a *radio receiver* was evolved from scratch

?

Evolvable hardware

Another experiment in Sussex, by Jon Bird and Paul Layzell

- Goal: evolve a circuit producing an oscillatory signal without having an internal clock

- Evolved solution: instead of an oscillator, something like a *radio receiver* was evolved from scratch

→ **The evolved circuit was stealing the oscillating clock signal of a nearby computer**

- Another example of how artificial evolution finds clever ways to **exploit the ecological niche**

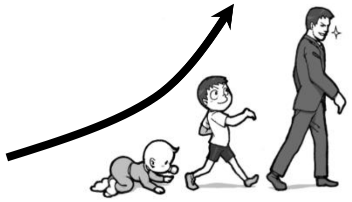
→ A new sensor modality was evolved from scratch!

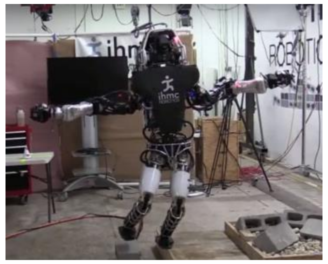
«Perverse instantiation»

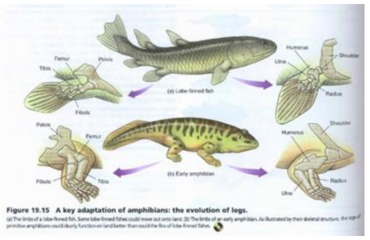
- Artificial Evolution sometimes <u>finds a way to solve the task</u> (optimize the fitness)…<u>but not the way you would want/expect to</u>
- <u>Usually undesired phenomenon</u> (solution is not admissible, i.e. exploits a glitch in the physics engine), but highlights <u>interesting properties</u> of evolutionary algorithms:
  - Ability to <u>go beyond our intuition</u> devising <u>non-intuitive solutions</u>, “thinking” <u>outside the box</u> → creativity?
  - <u>Adaptation</u> to the ecological niche: like biological organisms, evolved solutions <u>exploit all opportunities for survival</u>
- This phenomenon is related with the <u>*reality gap*</u> problem (later)

Developmental encodings and Artificial Evolution of brains and bodies

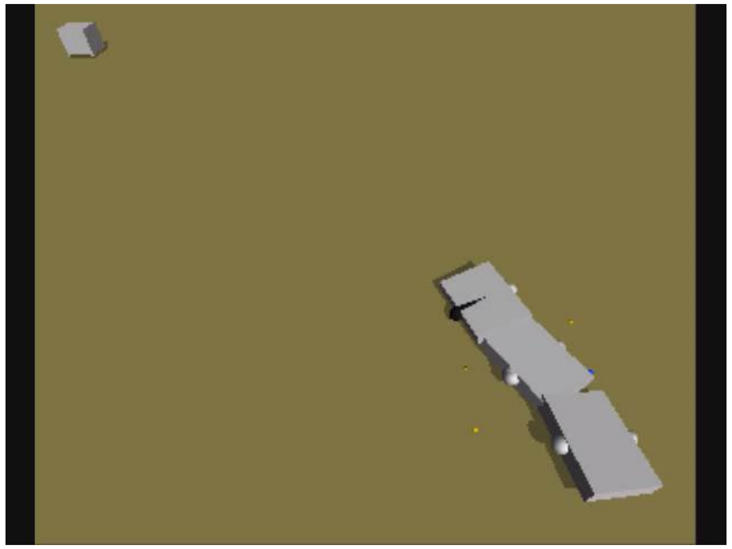
Why (co-)evolving brains and bodies?







- Biologically plausible: <u>brains and bodies evolve and develop together</u> (nature doesn't devise brains from scratch for already complex bodies)



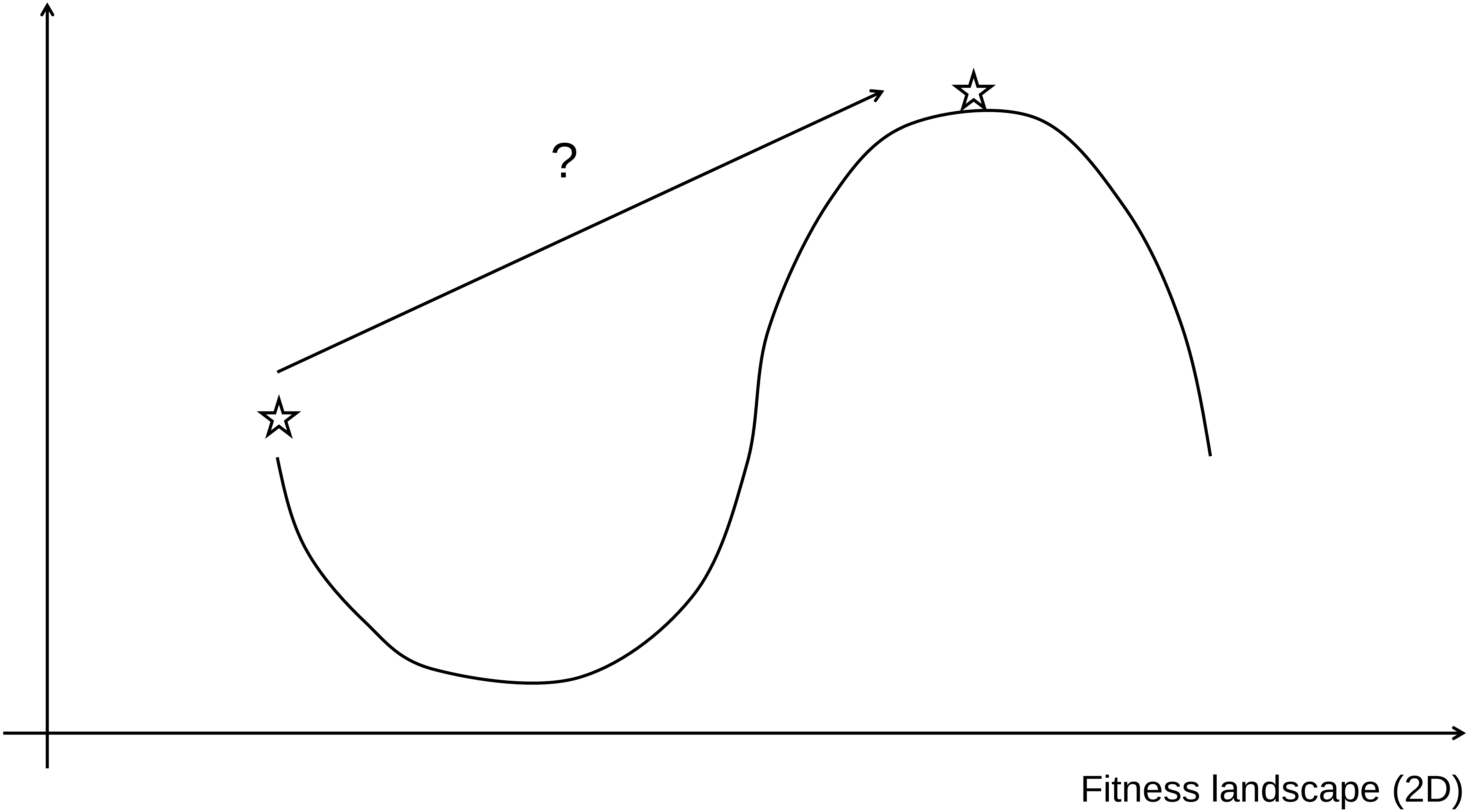
- <u>A suitable morphology can simplify control</u> (*embodied intelligence*, passive dynamic walker, etc) → <u>It should be evolved, rather than fixed</u>

  → Target: "<u>balanced" brain-body trade-off</u>

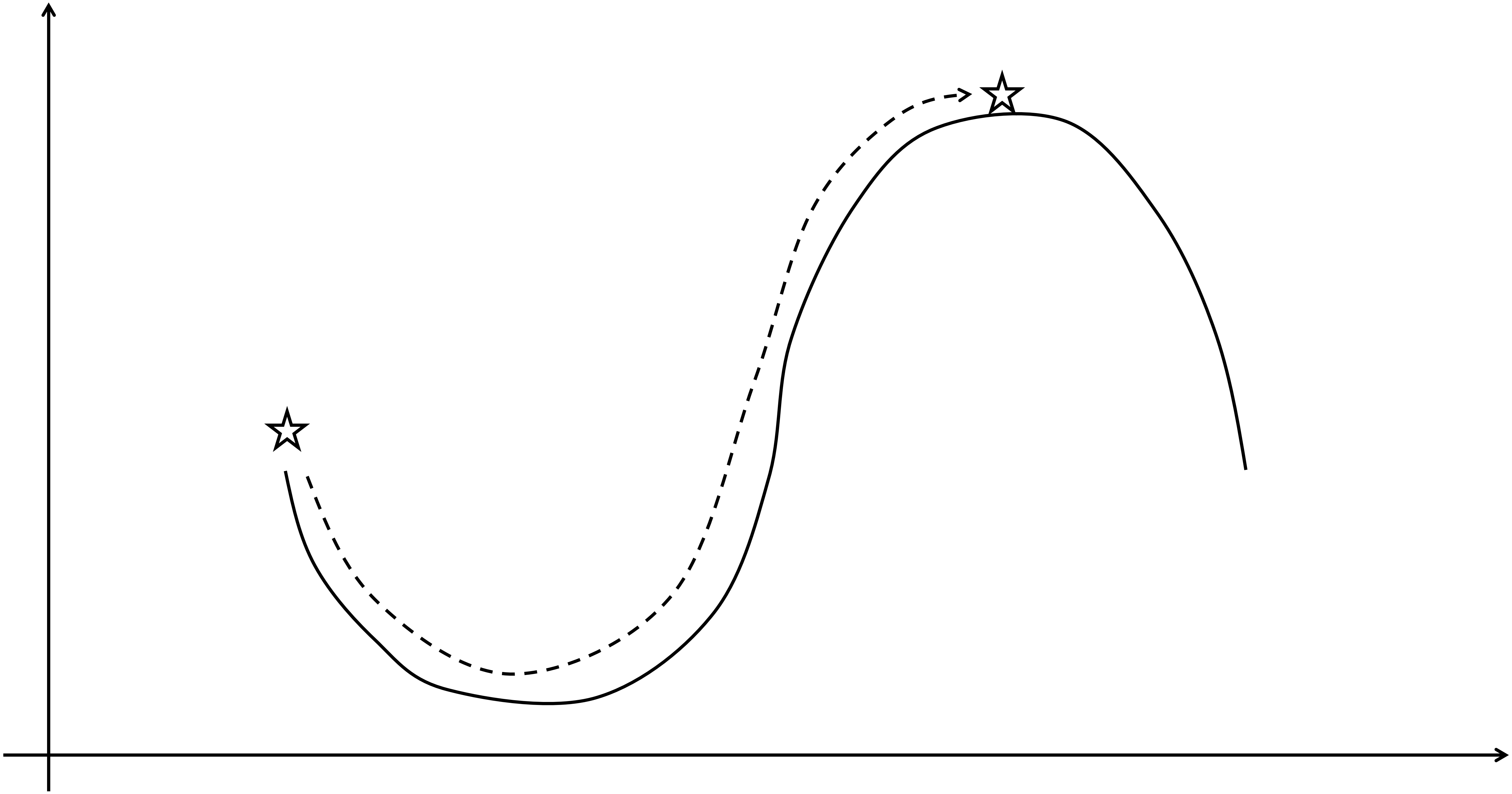
- Even if <u>the dimensionality of the search space increases</u> (more parameters), <u>the problem can actually become simpler</u>

  Bongard, Paul, "Making Evolution an Offer It Can't Refuse: <u>Morphology and the Extradimensional Bypass</u>"

Why (co-)evolving brains and bodies?

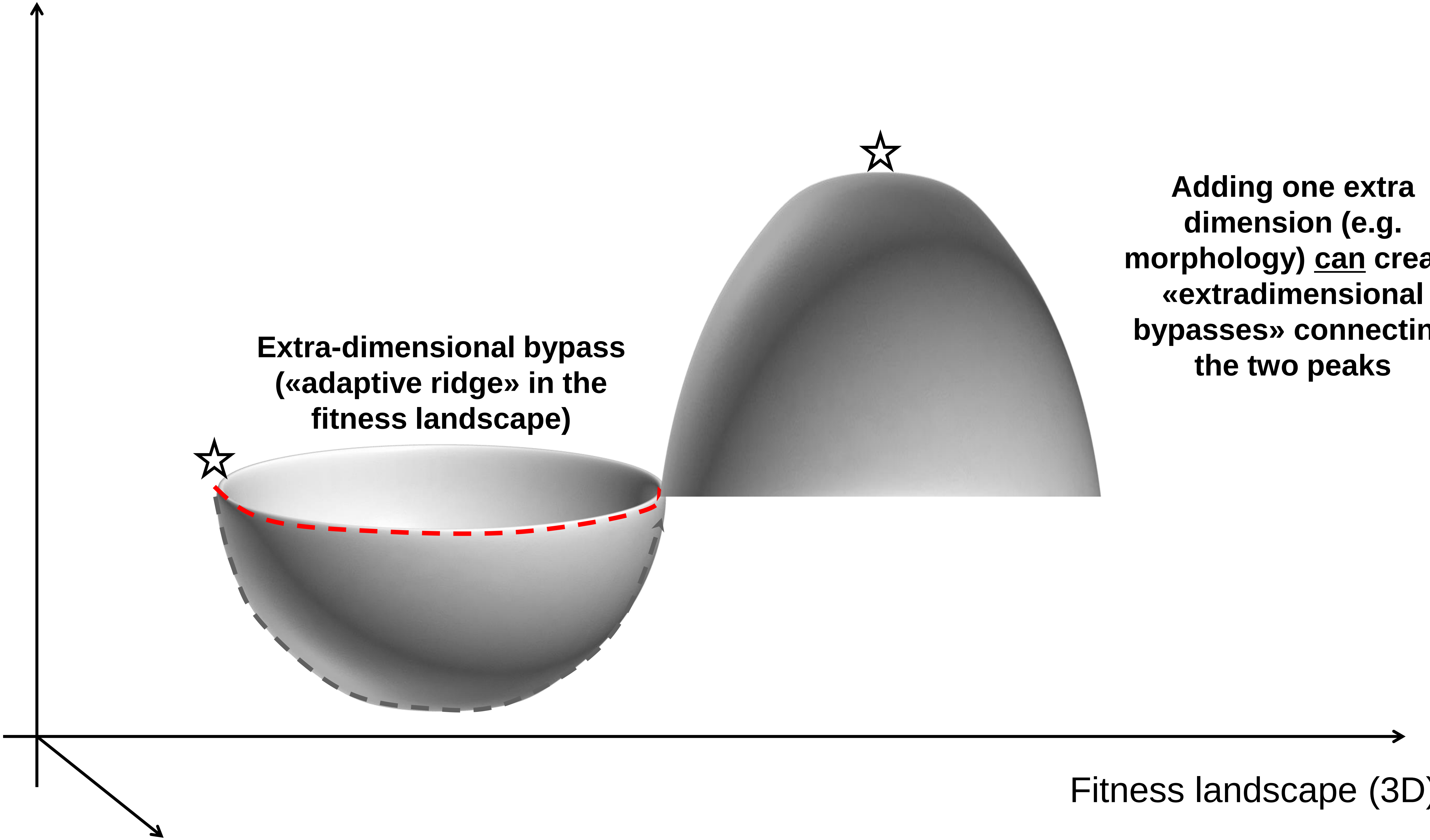


Why (co-)evolving brains and bodies?



Fitness landscape (2D)

Why (co-)evolving brains and bodies?



Adding one extra dimension (e.g. morphology) <u>can</u> create «extradimensional bypasses» connecting the two peaks

Direct encodings - limitations

- So far, for simplicity: assumed direct encoding, or, genotype-to-phenotype map (e.g. genotype = a vector of real numbers, 1:1 mapping between each number and a phenotypic trait)
- Not very general, nor scalable when co-evolving complex brains and bodies
  - Usually requires a priori assumptions, e.g. fix the structure of the phenotype (e.g. brain/body topology) and evolve parameters only (e.g. synapses of an ANN, length of limbs, …)
  - As the complexity of brains and bodies increases, so does the number of parameters to be evolved (the dimensionality of the search space can quickly explode)

    → *"Curse of dimensionality"*

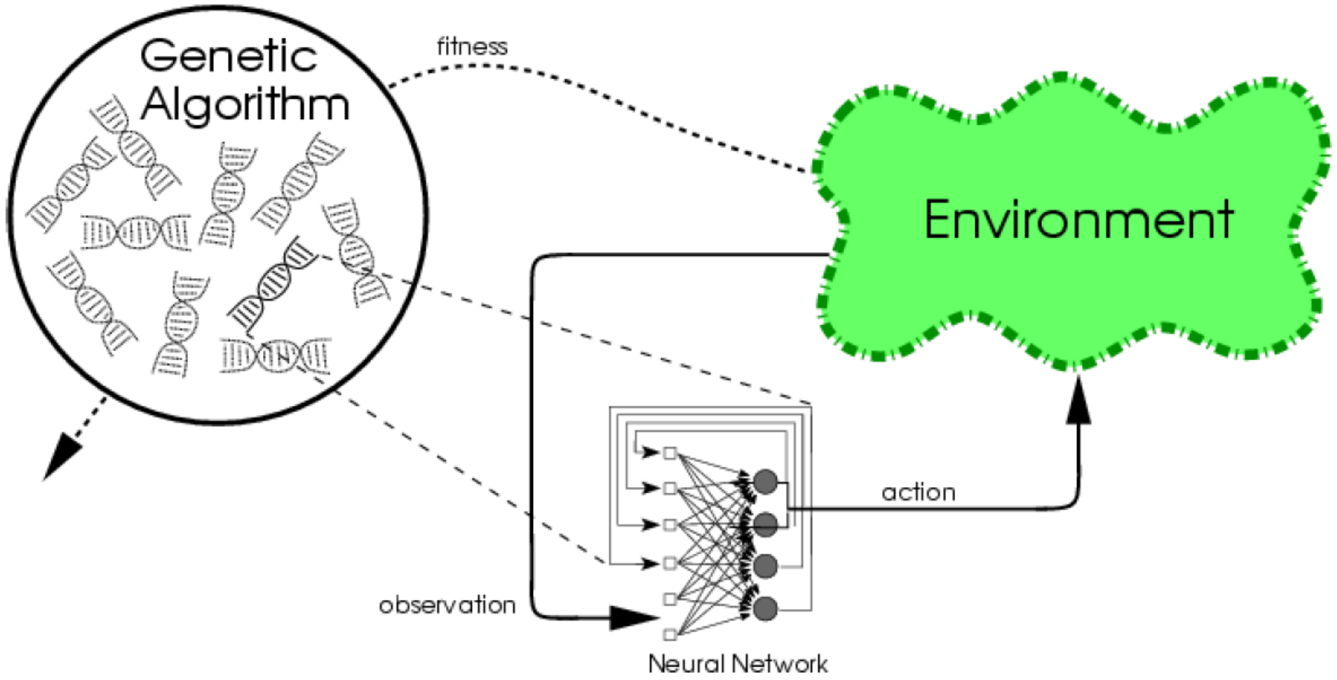
Genotype-to-phenotype map

- Genotype-to-phenotype map: function/algorithm that transforms the genotype into the phenotype
- Direct mapping: map = identity
- Indirect mapping:
  - Map is performed by an algorithm, whose parameters (genotype) are under evolutionary control
  - Each element of the genotype (optimization variables) potentially influences more than one phenotypic trait → scalability

→ Also known as **generative**, or **developmental** encodings (bring into play mechanisms inspired by *biological development*)

Example - Evolving brains - Conventional NeuroEvolution (CNE)

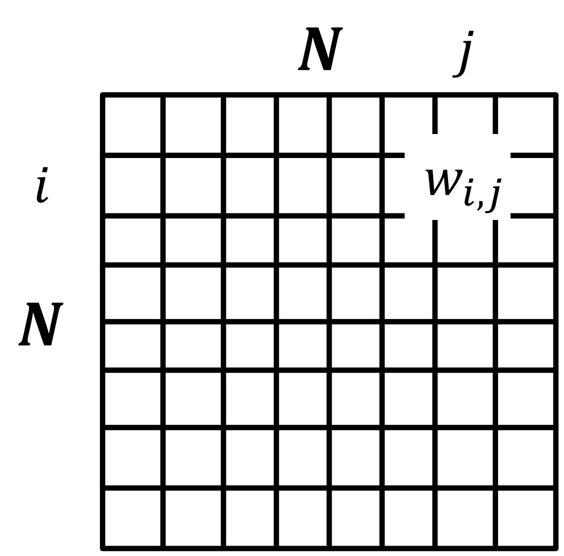
1. Fix the structure of the NN (usually fully connected)
2. Concatenate synaptic weights and biases into a genome → direct encoding
3. Use an evolutionary algorithm to evolve the network with respect to a given task
4. Fitness: evaluation of network's performance on a task

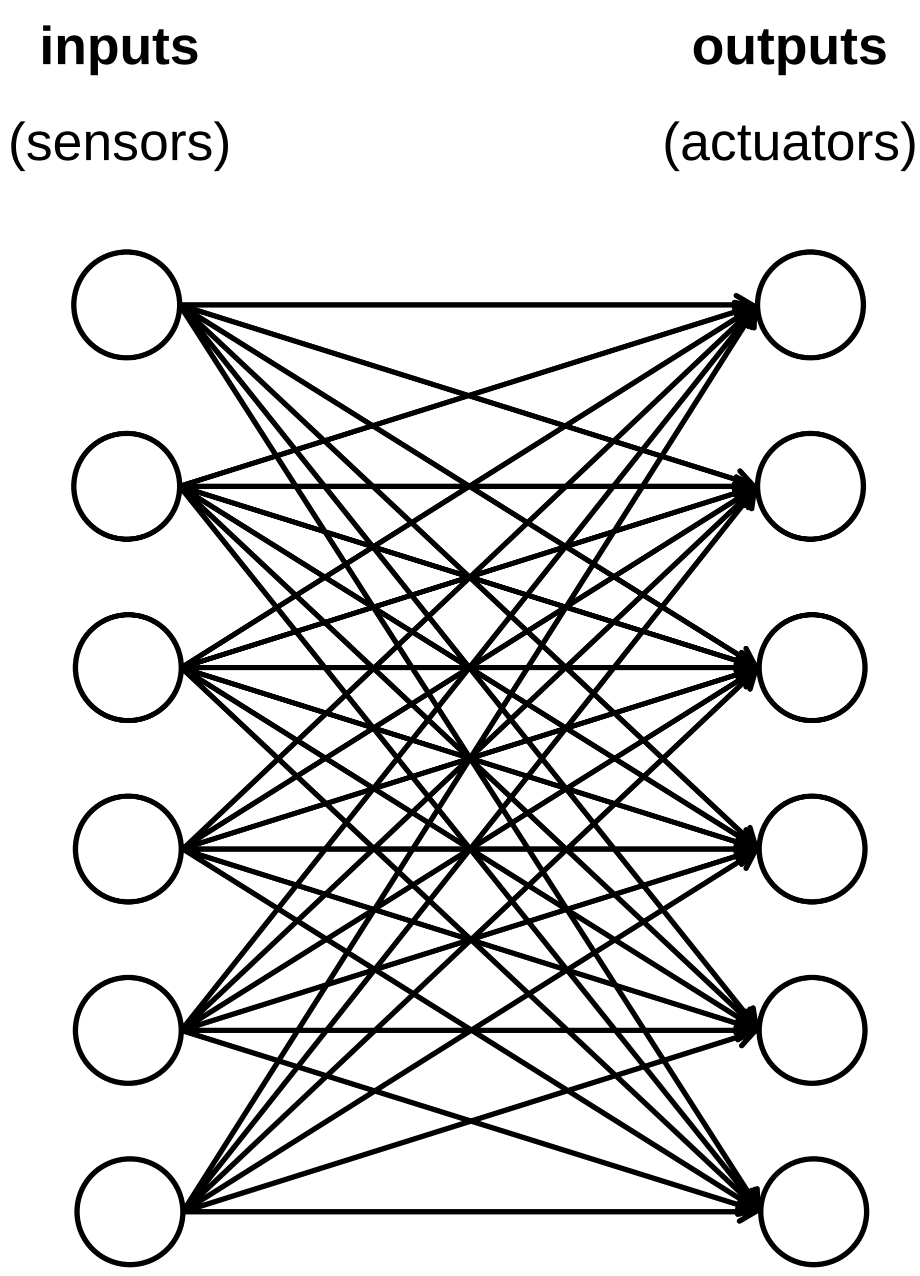


http://www.scholarpedia.org/article/Neuroevolution

Evolving an ANN with direct encoding

- Imagine having N sensors and N actuators (e.g. N = 100)
- Evolve a very simple **inputs → outputs map** to solve a task (sensors → motors)
- **Fully connected** network → $\boldsymbol{N^2}$ **synapses** (quadratic scaling) (matrix / picture) ∈ $[-1,1]$ (1: excitatory, 2: inhibitory)





- Direct encoding → **By trial and error, evolution must find out $\boldsymbol{N^2}$ numbers (e.g. 10000), independently**

  → Not considering hidden layers (scaling is even worse)

  → **Not scalable as body, brain, and task complexity go up**

Evolving an ANN with direct encoding

- **Also, imagine this ANN being embedded in a robot**

  → Can you think of other drawbacks of a direct encoding here?

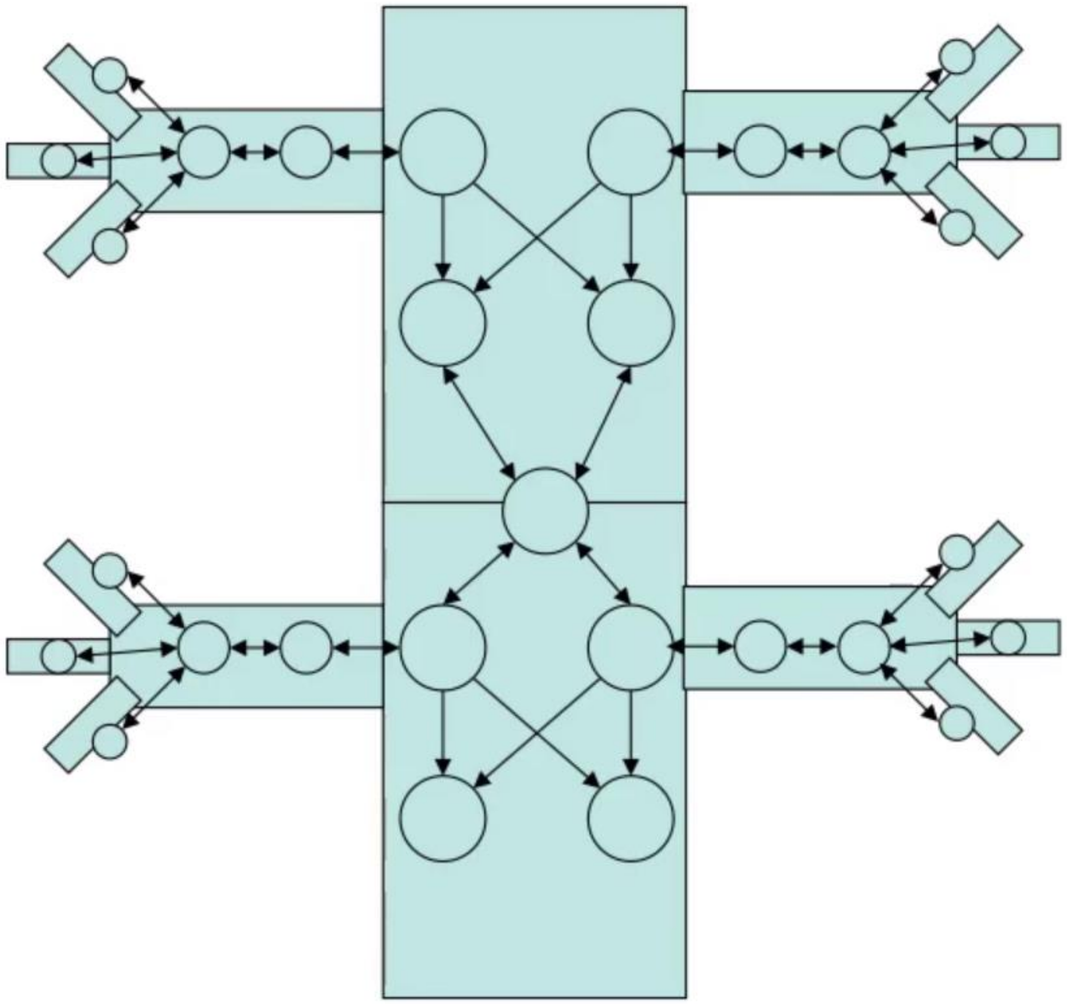
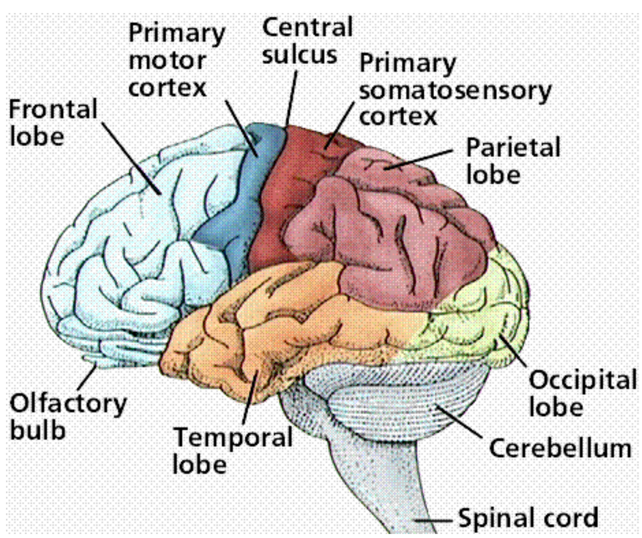


Image credits: Prof. Josh Bongard, UVM

Evolving an ANN with direct encoding

- Biological and artificial brains and bodies exhibit **regularities** (e.g. symmetry, repetition)
- Our evolved ANN **may benefit from mirroring some of these regularities**

Image credits:: https://www2.estrellamountain.edu/faculty/farabee/biobk/BioBookNERV.html



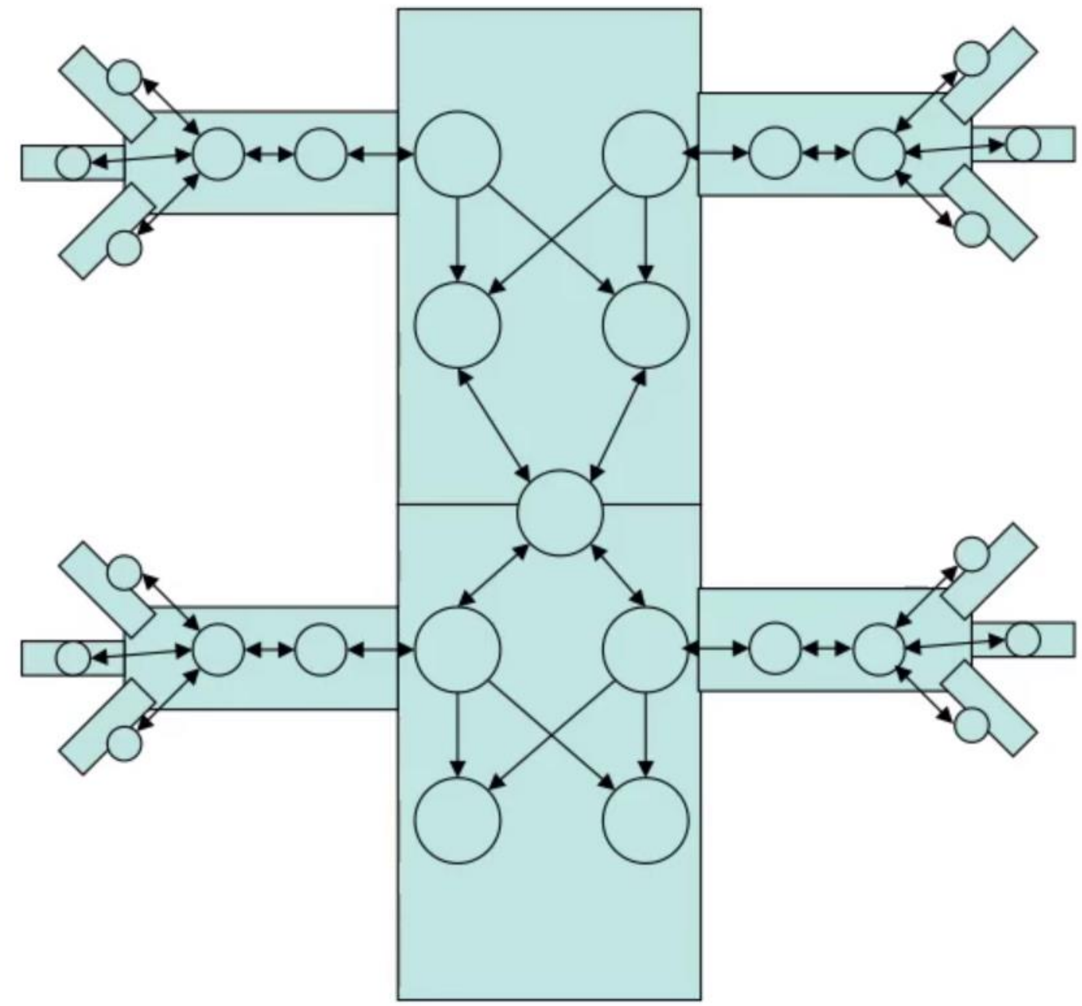


Image credits: Prof. Josh Bongard, UVM

Evolving an ANN with direct encoding

Many tasks (e.g. locomotion) **require coordinated control signals** (= regular networks)

→ Non coordinated controllers are most certainly bad

→ Makes often sense to bias the search towards regular solutions

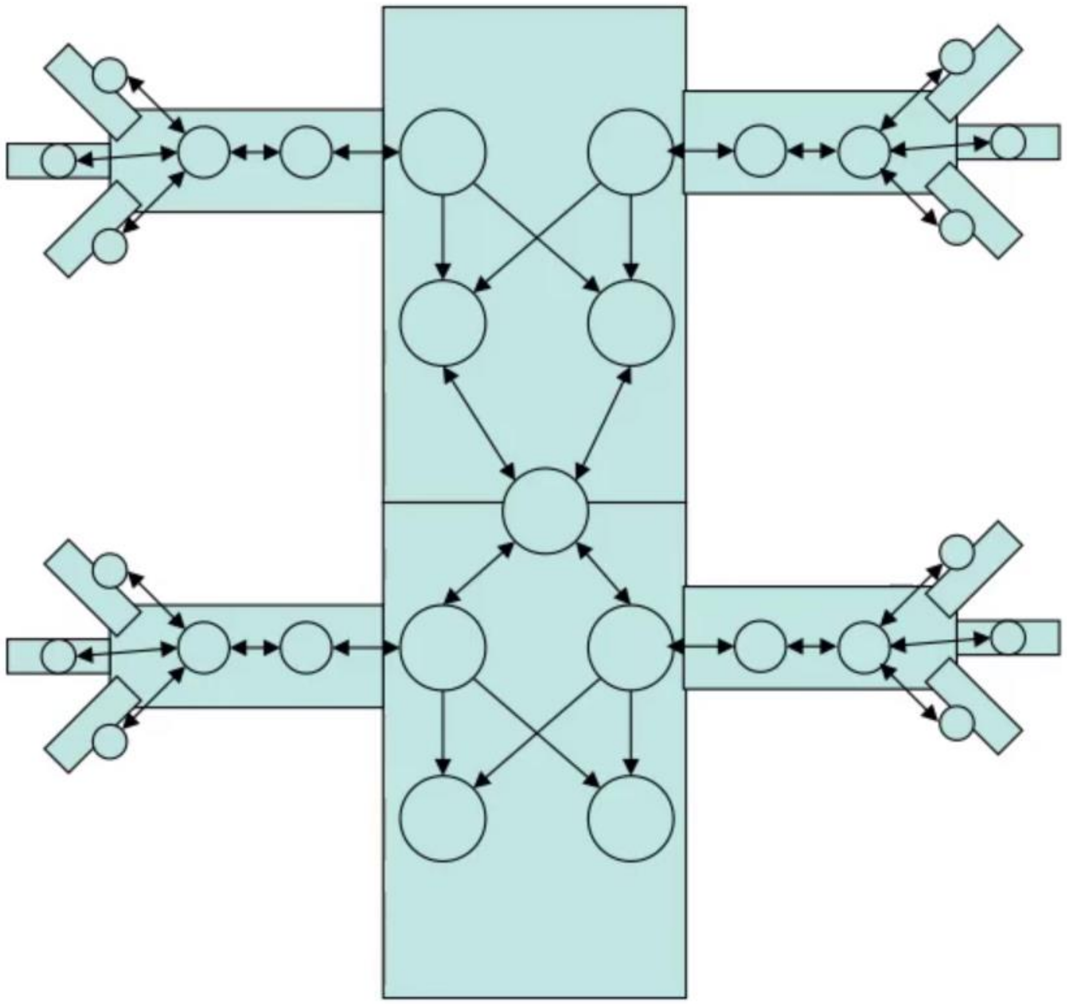
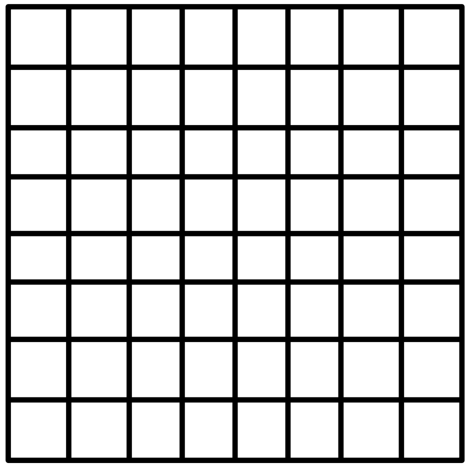


Image credits: Prof. Josh Bongard, UVM

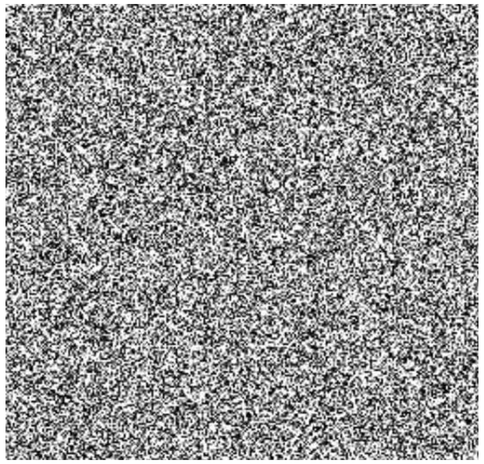
Evolving an ANN with direct encoding

With direct encoding, these regularities are overlooked: all the algorithm sees is a matrix of independent numbers to be optimized by trial and error

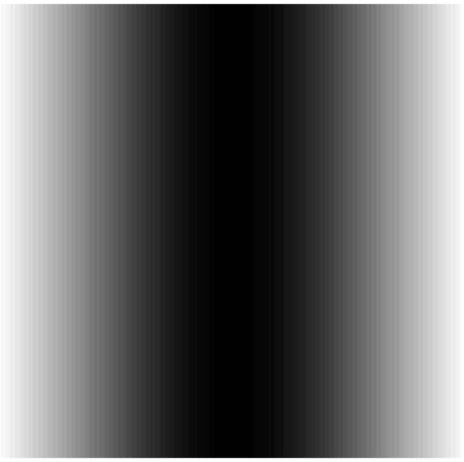
→ Regular structures **can be discovered, but are not enforced** (→ can be inefficient)



Matrix of synaptic weights



Random init with direct encoding



A potentially desirable regular solution, to be achieved changing one pixel at a time

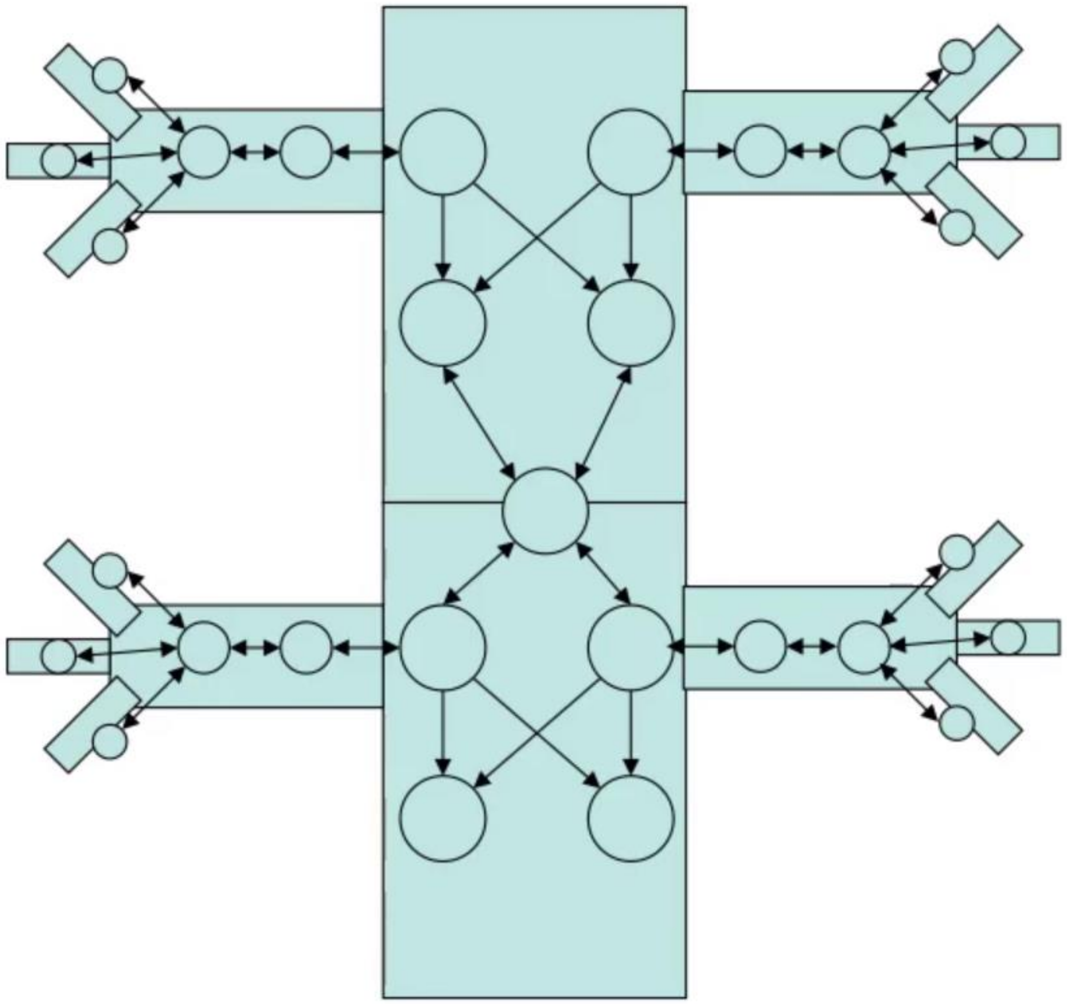


Image credits: Prof. Josh Bongard, UVM

Evolving an ANN with direct encoding

We could **manually enforce certain regularities for a specific task at hand** (e.g. enforce bilateral symmetry of synaptic weights when evolving locomotion)

Not general, though

→ **Need for general encodings that allow evolution to produce and select general regular patterns**

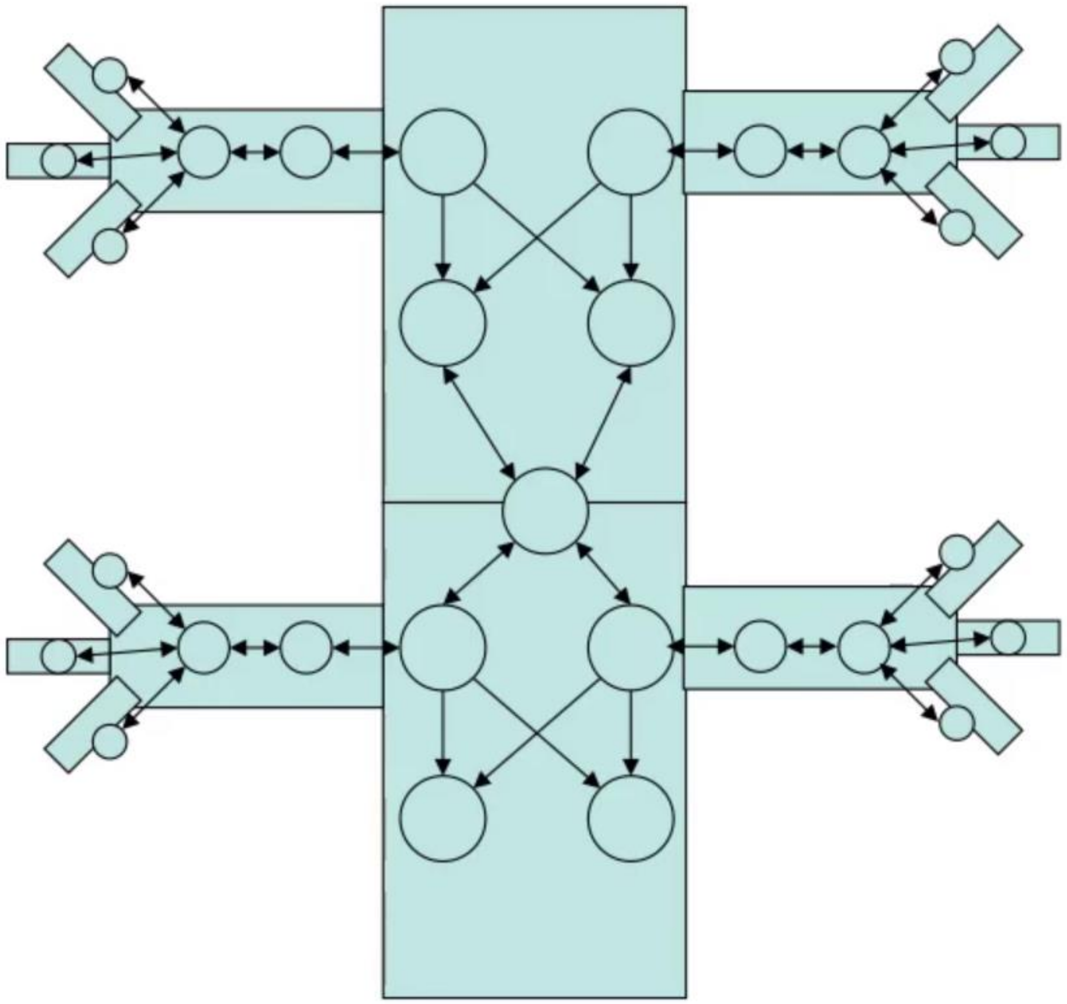
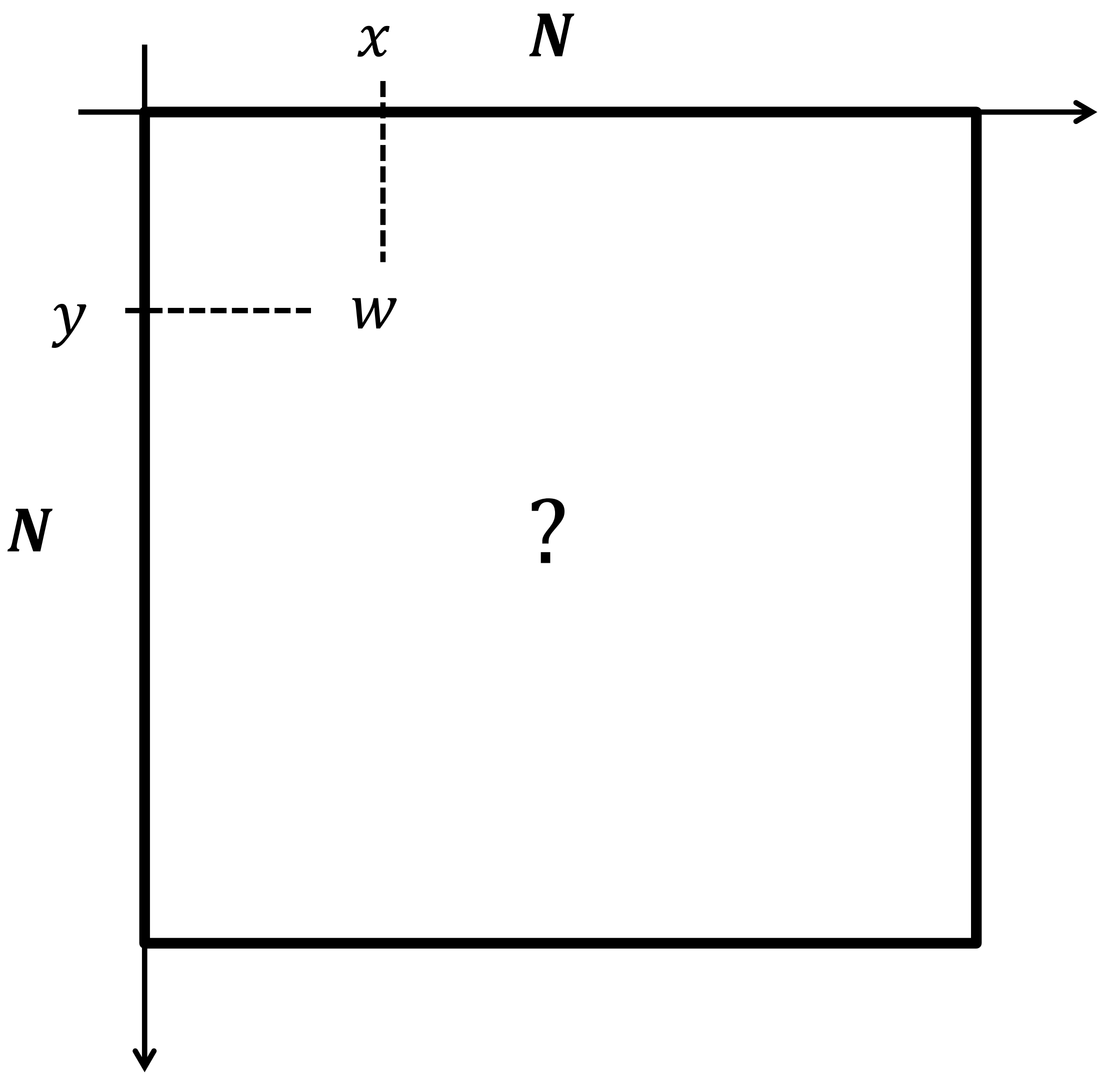


Image credits: Prof. Josh Bongard, UVM

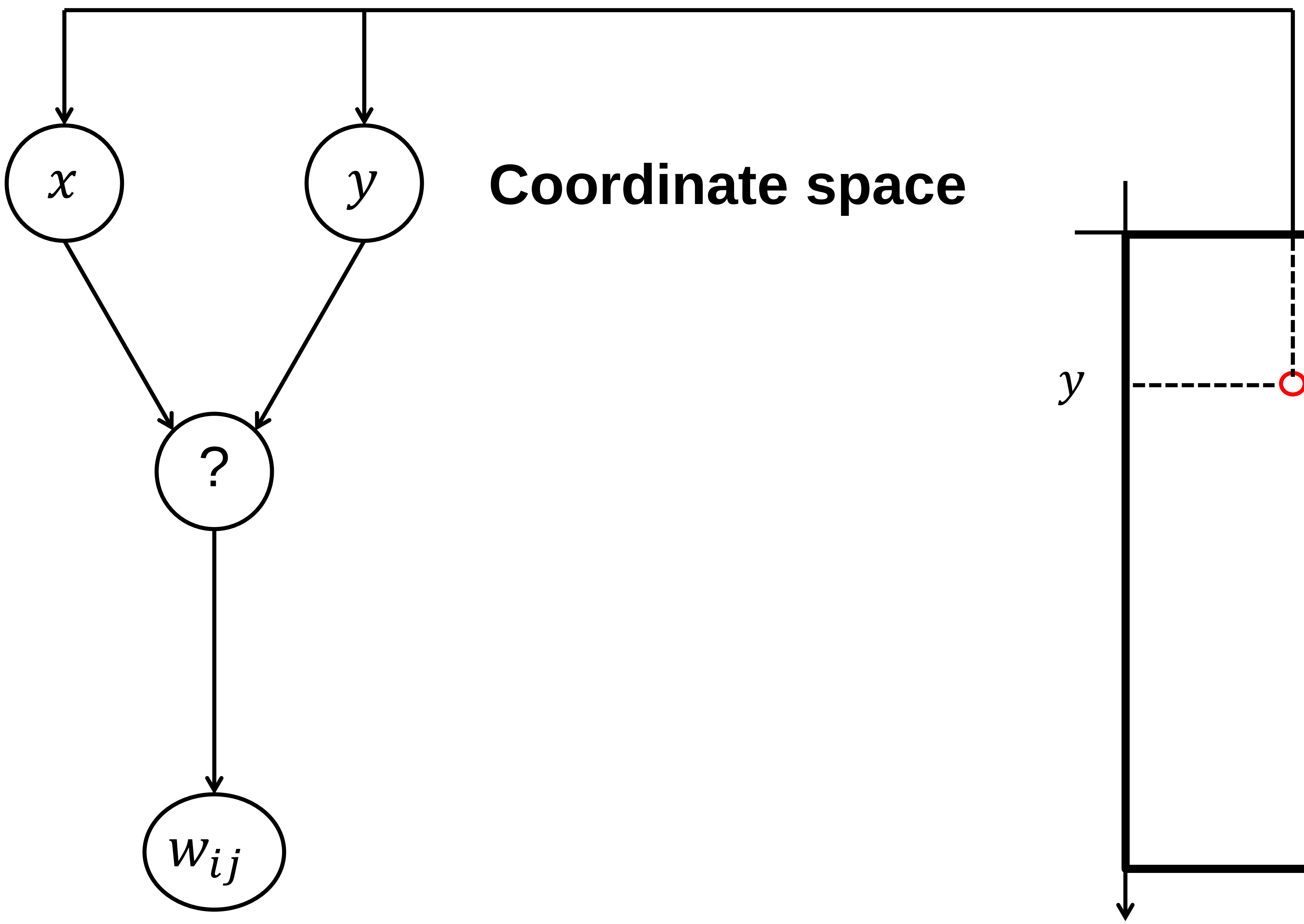
Evolving an ANN with indirect encoding

- Keep thinking about the phenotype (2D matrix of synaptic weights) as a picture within a 2D coordinate frame (x,y)
- Darkness of a pixel = synaptic weight
- For a given task, some specific patterns of synaptic weights will result in ANNs that perform well
- **Can you think of a compact way to encode regular patterns within such a coordinate frame?**

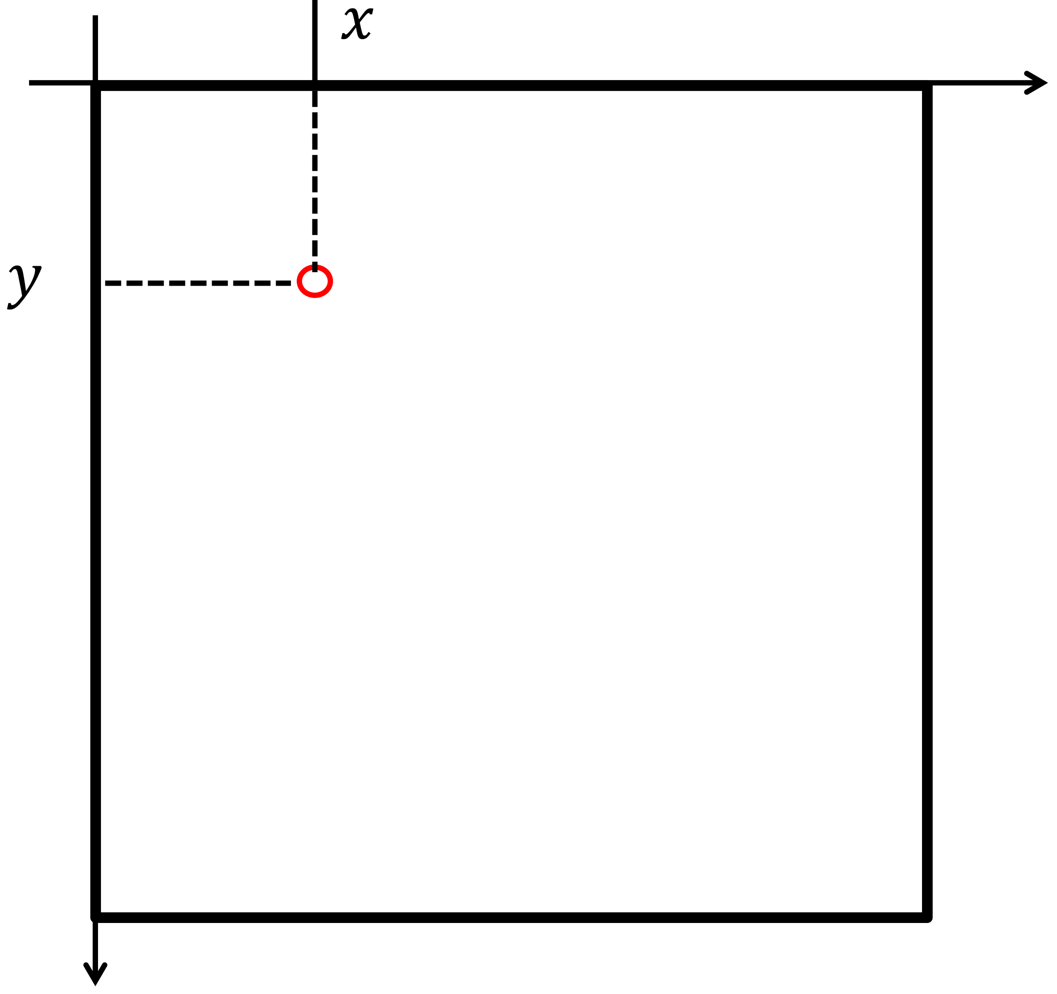
  i.e. a way to describe a 2D picture without listing the value of every single pixel



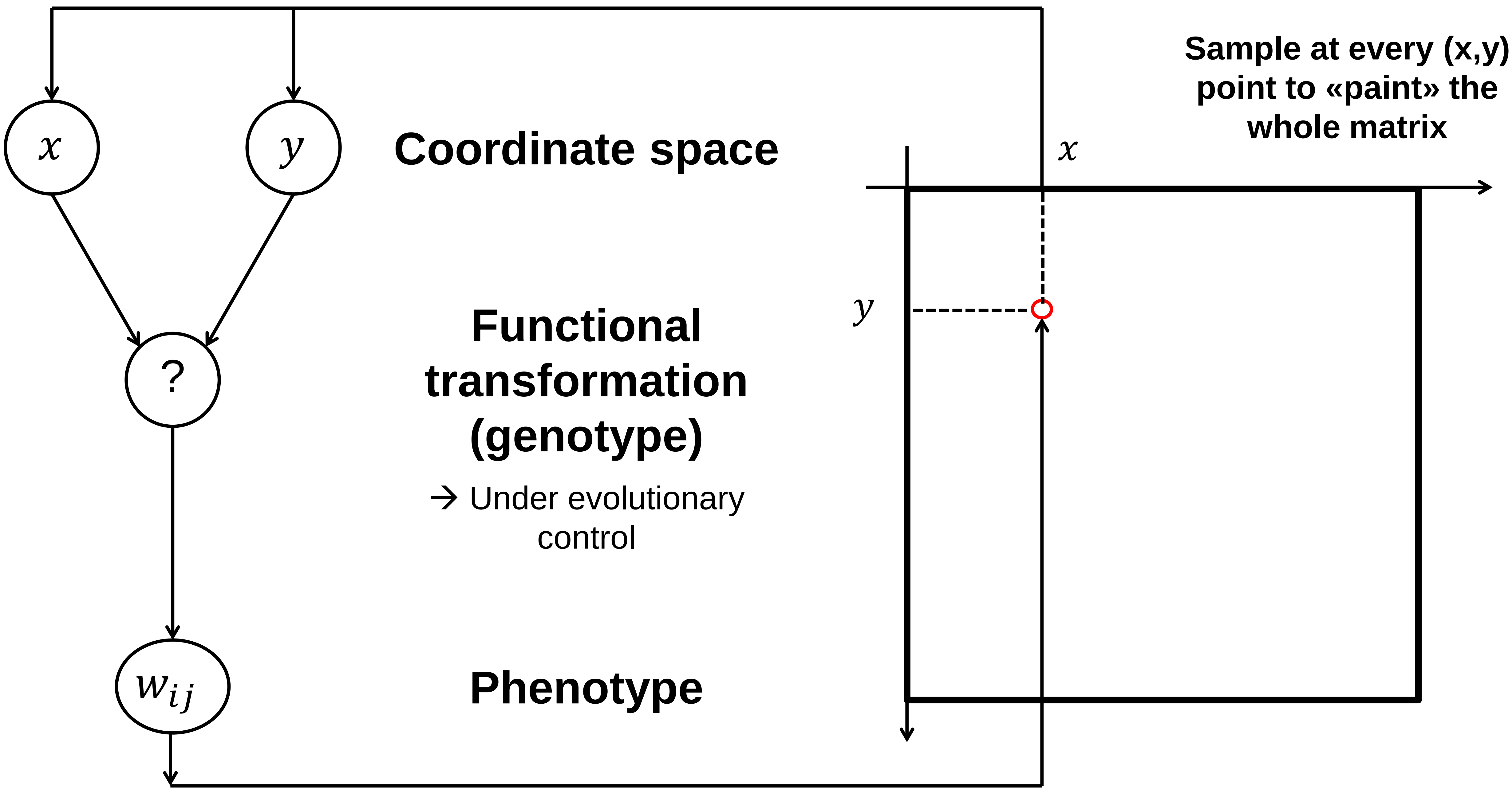
Evolving an ANN with indirect encoding



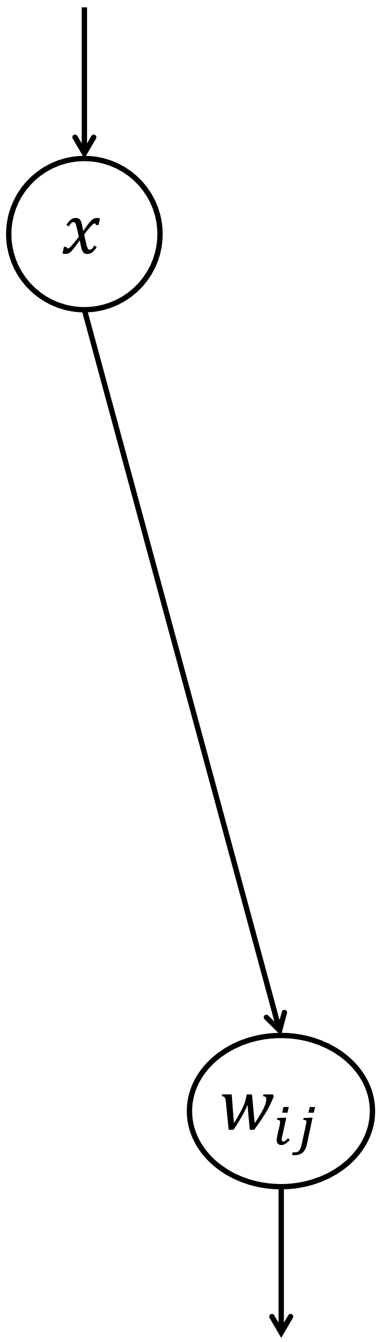
Coordinate space



Evolving an ANN with indirect encoding



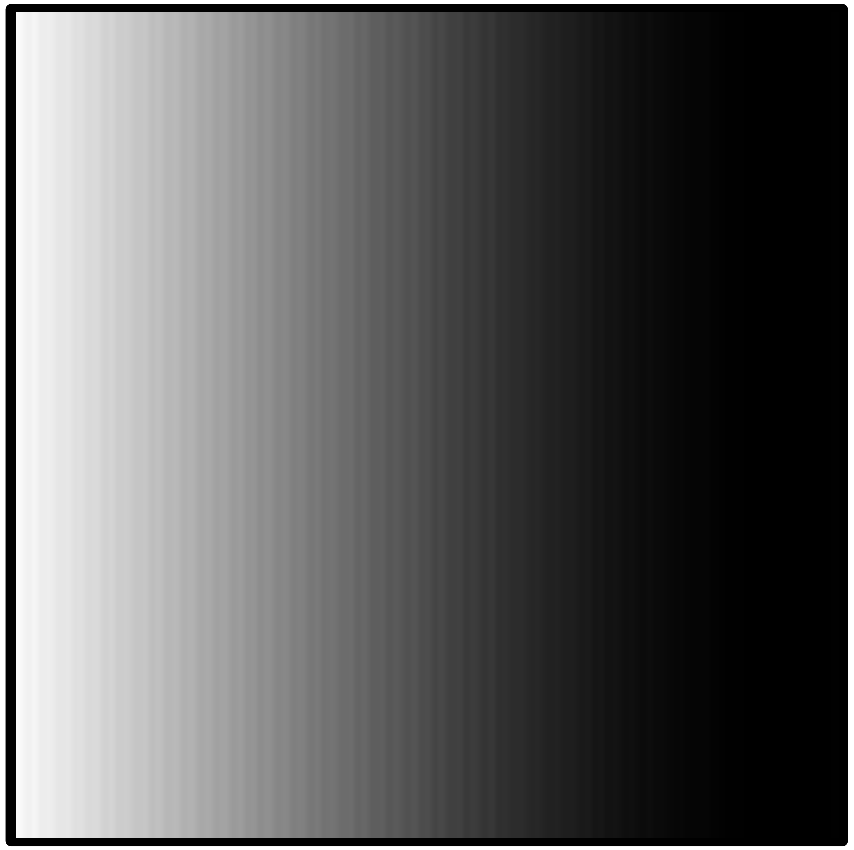
Evolving an ANN with indirect encoding



Coordinate space

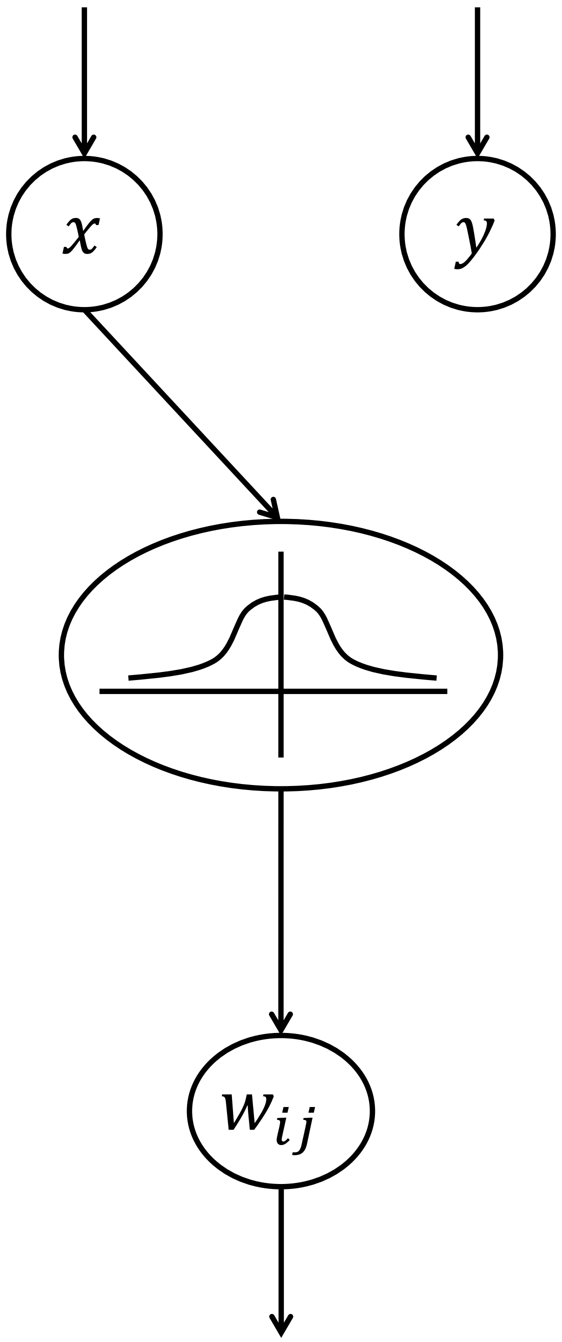
Functional transformation (genotype)

Phenotype



(Examples by N. Cheney)

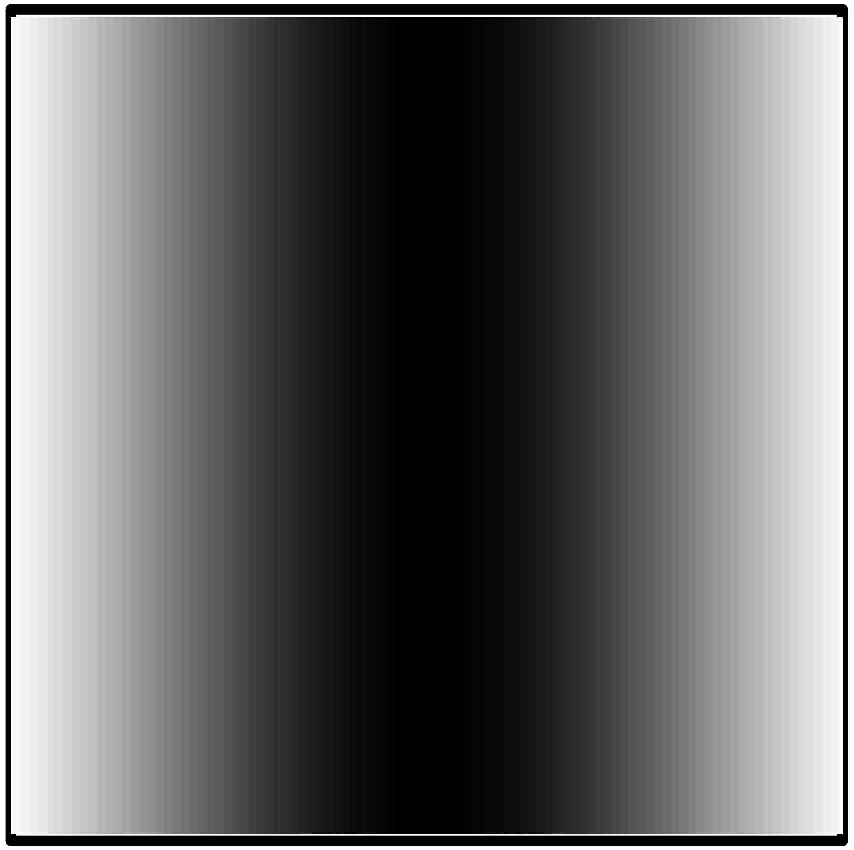
Evolving an ANN with indirect encoding



Coordinate space

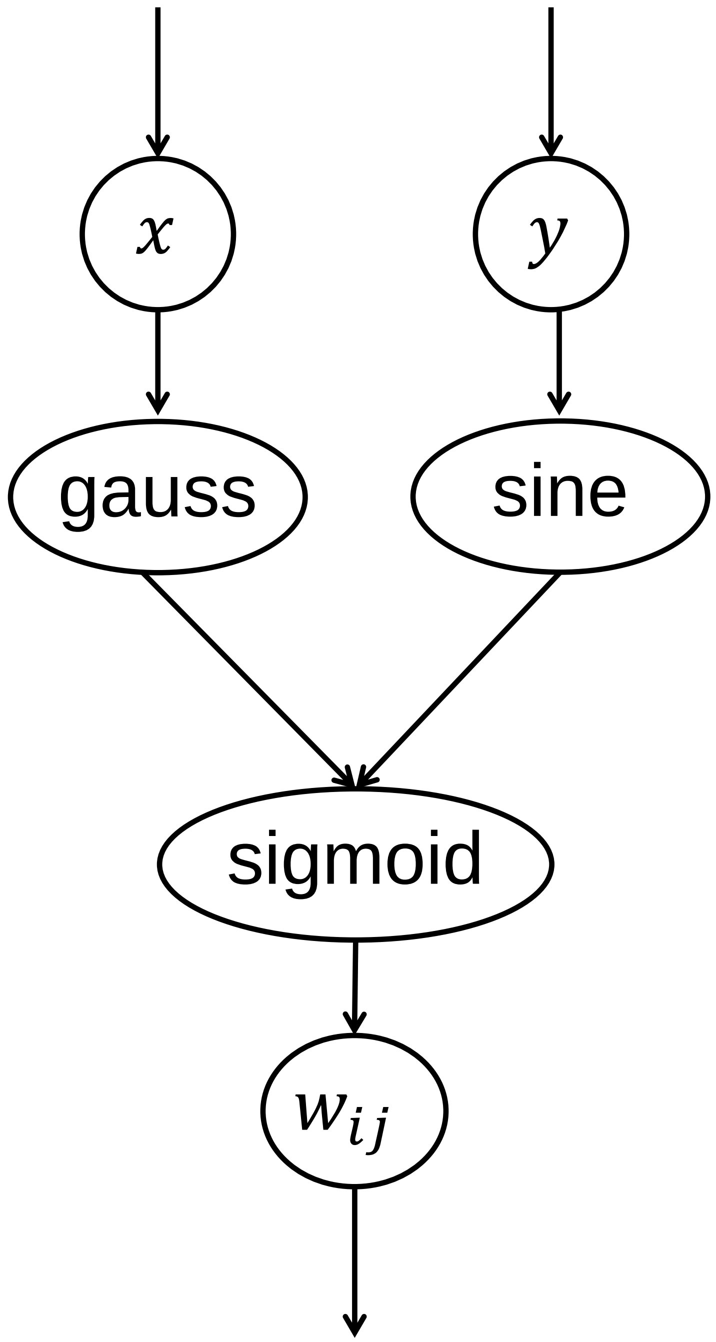
Functional transformation (genotype)

Phenotype



(Examples by N. Cheney)

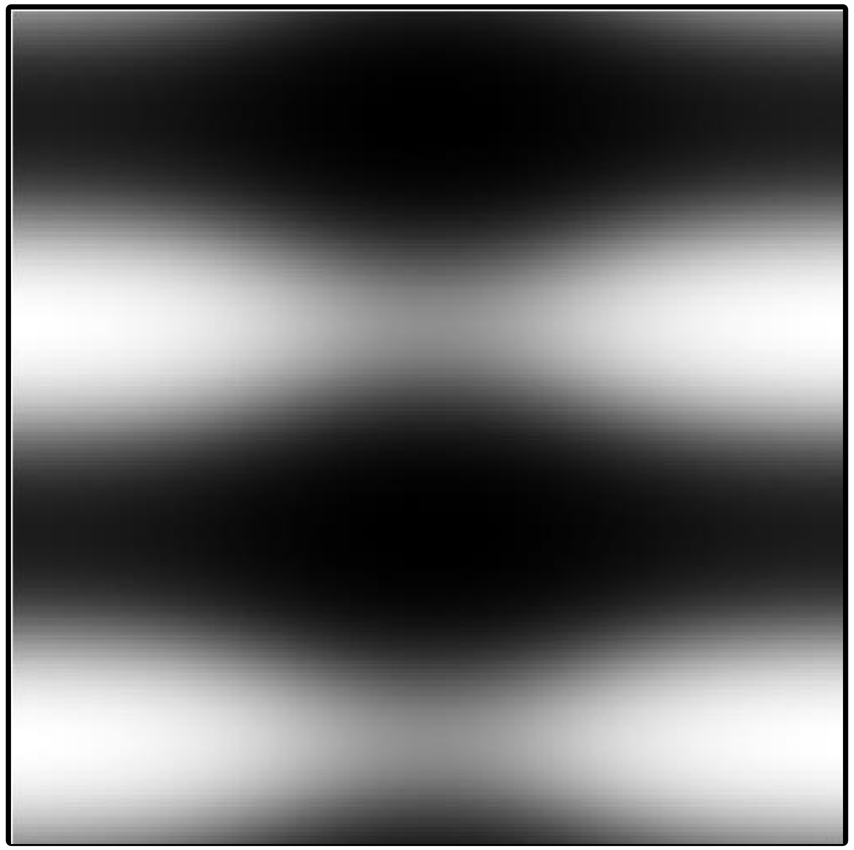
Evolving an ANN with indirect encoding



Coordinate space

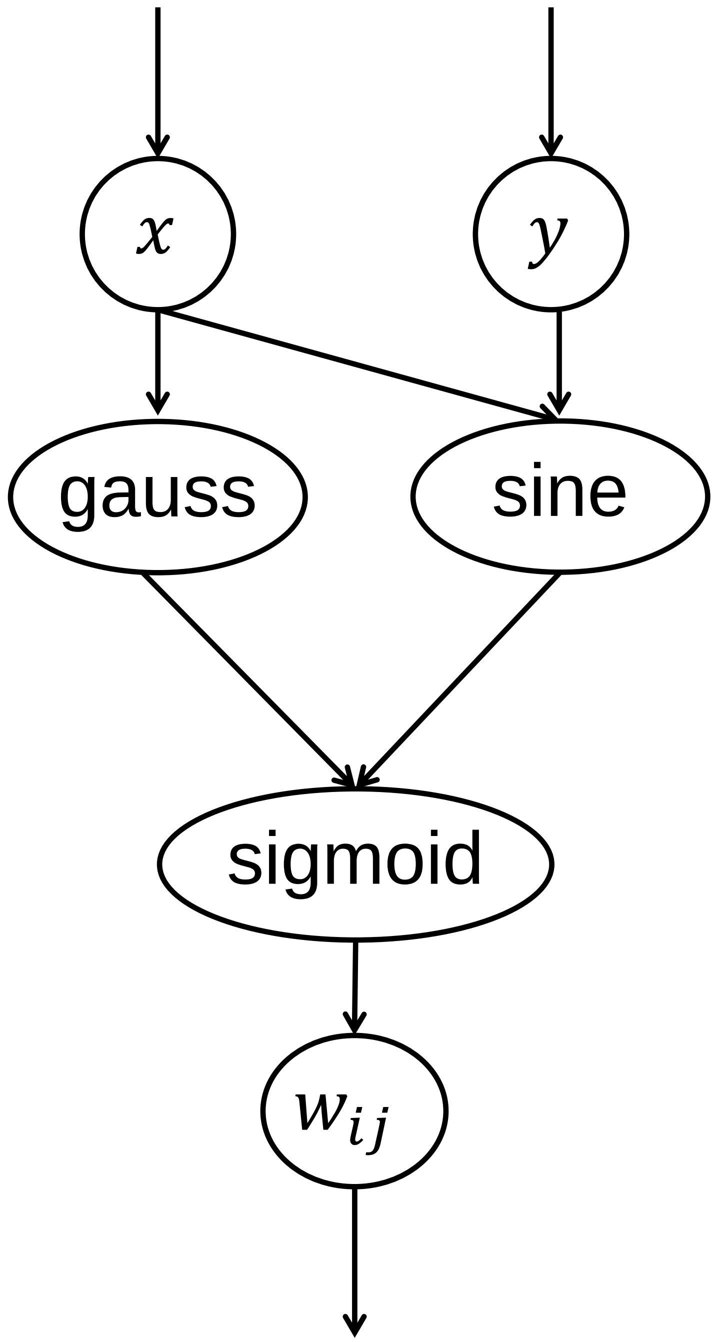
Functional transformation (genotype)

Phenotype



(Examples by N. Cheney)

Evolving an ANN with indirect encoding

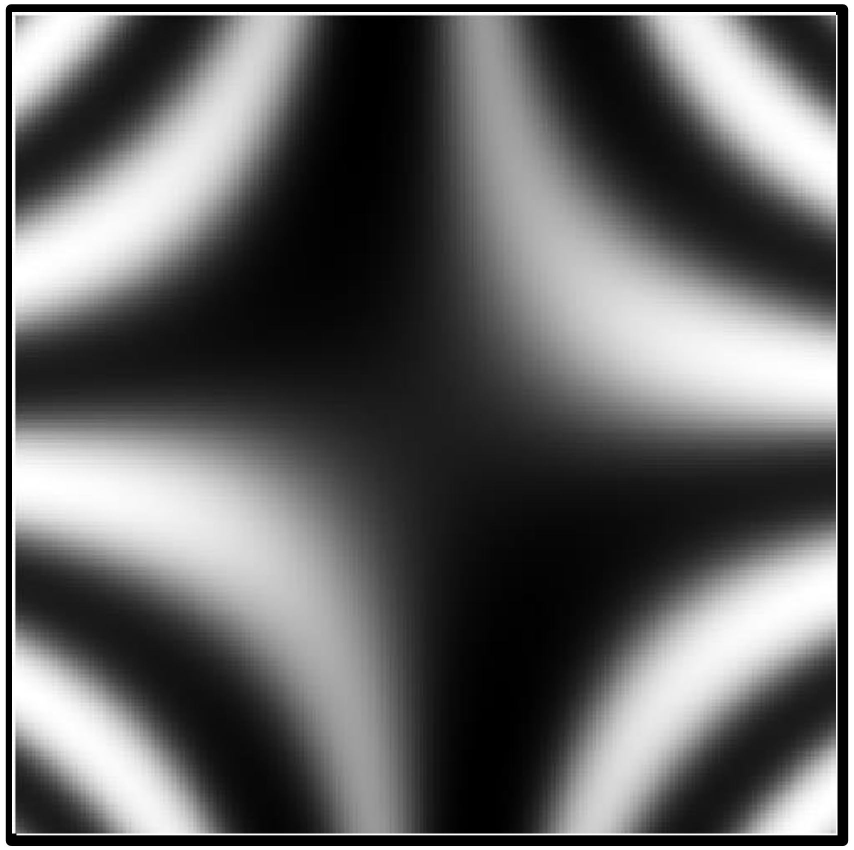


Coordinate space

Functional transformation (genotype)

Phenotype

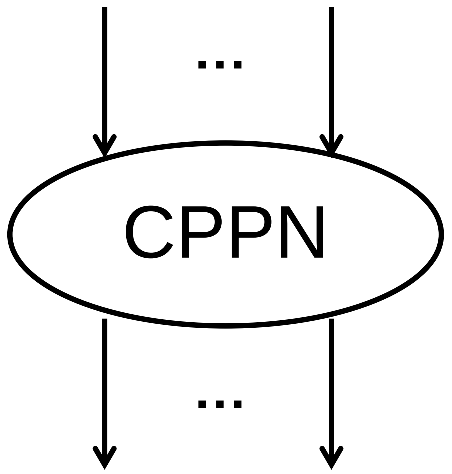
Complex pattern produced by a compact genotype



(Examples by N. Cheney)

→ Compositional Pattern Producing Networks (CPPN)

Inputs:
spatial features



Outputs:
Phenotype

Functional transformation (genotype)
Composition of regular base functions

K. Stanley, "Compositional Pattern Producing Networks: A Novel Abstraction of Development"

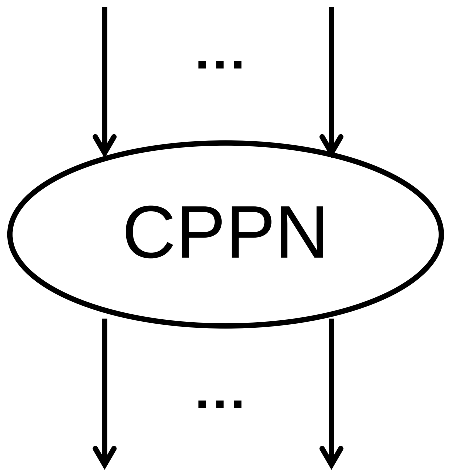
CPPNs are one example of indirect/generative/developmental encoding

- The <u>genotype is a network</u>
- Each node has an <u>activation function</u> chosen from a given <u>pool of regular base functions</u>
- Edges are <u>weighted</u>

What would you need to change in order to achieve an evolutionary algorithm that evolves CPPNs (w.r.t. one that evolves a matrix of synaptic weights with direct encoding)?

→ Compositional Pattern Producing Networks (CPPN)

Inputs:
spatial features



Outputs:
Phenotype

Functional transformation (genotype)
Composition of regular base functions

K. Stanley, "Compositional Pattern Producing Networks: A Novel Abstraction of Development"

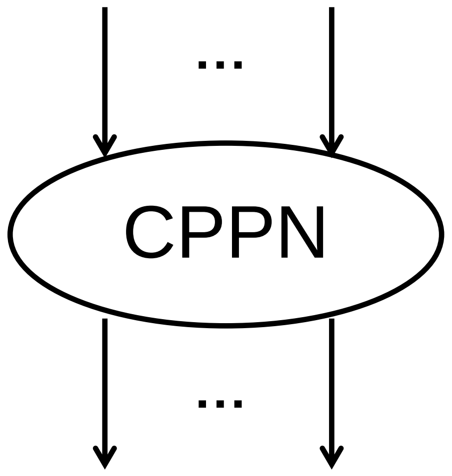
CPPNs are one example of indirect/generative/developmental encoding

- The <u>genotype is a network</u>
- Each node has an <u>activation function</u> chosen from a given <u>pool of regular base functions</u>
- Edges are <u>weighted</u>

→ **<u>Genetic operators</u>, example**:
<u>Mutation</u>: add/remove node/edge, randomly modify the weight of an existing edge, or the activation function of an existing node

→ Compositional Pattern Producing Networks (CPPN)

Inputs:
spatial features



Outputs:
Phenotype

Functional transformation (genotype)
Composition of regular base functions

K. Stanley, "Compositional Pattern Producing Networks: A Novel Abstraction of Development"

Features:

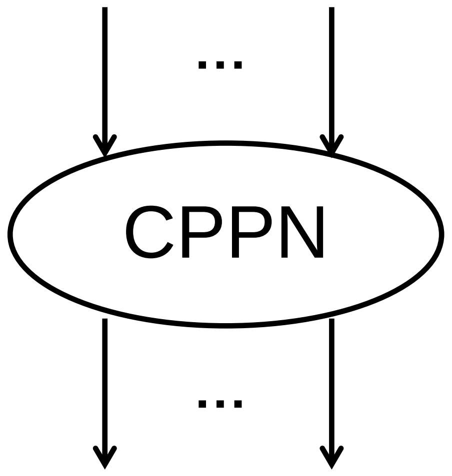
- Expressive: A compact genotype can generate a very large phenotype

  → **Scalability, evolvability**

- Composition of continuous functions → can be sampled at any desired resolution (phenotype can be continuous as well)
- Composition of regular functions → promotes regular phenotypic structures (useful both in brains and bodies)
- Starting from simple networks (= simple patterns), can encode phenotypes of increasing complexity («**complexification**»)

→ Compositional Pattern Producing Networks (CPPN)

Inputs:
spatial features



Outputs:
Phenotype

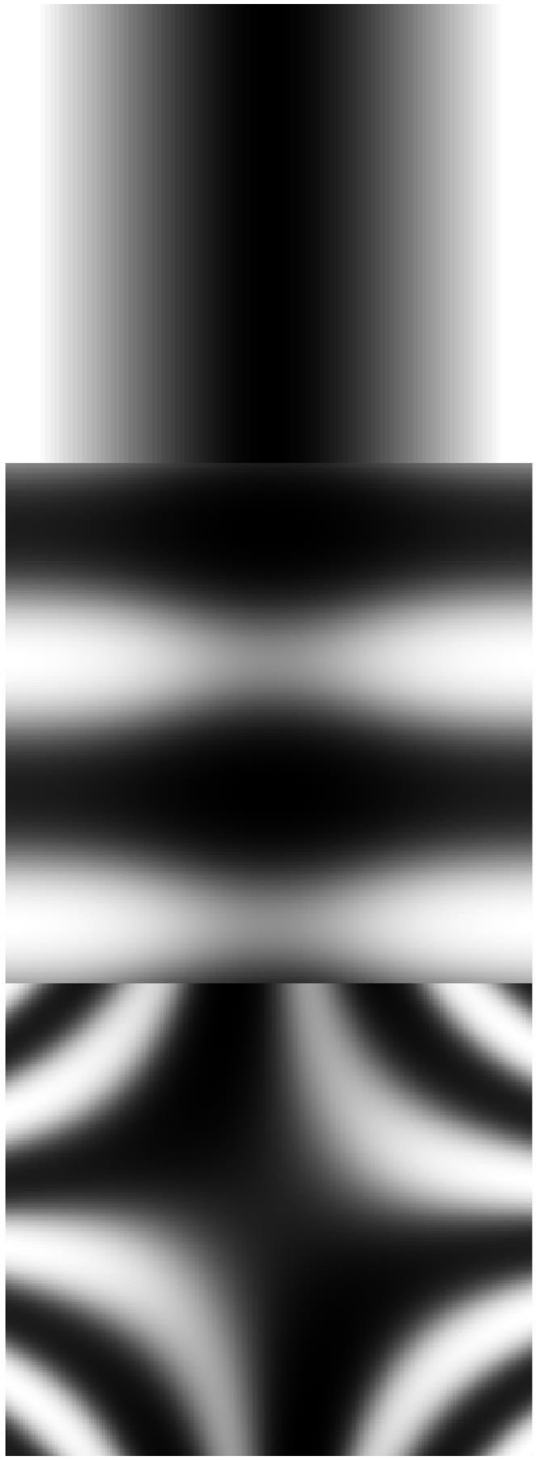
Functional transformation (genotype)
Composition of regular base functions

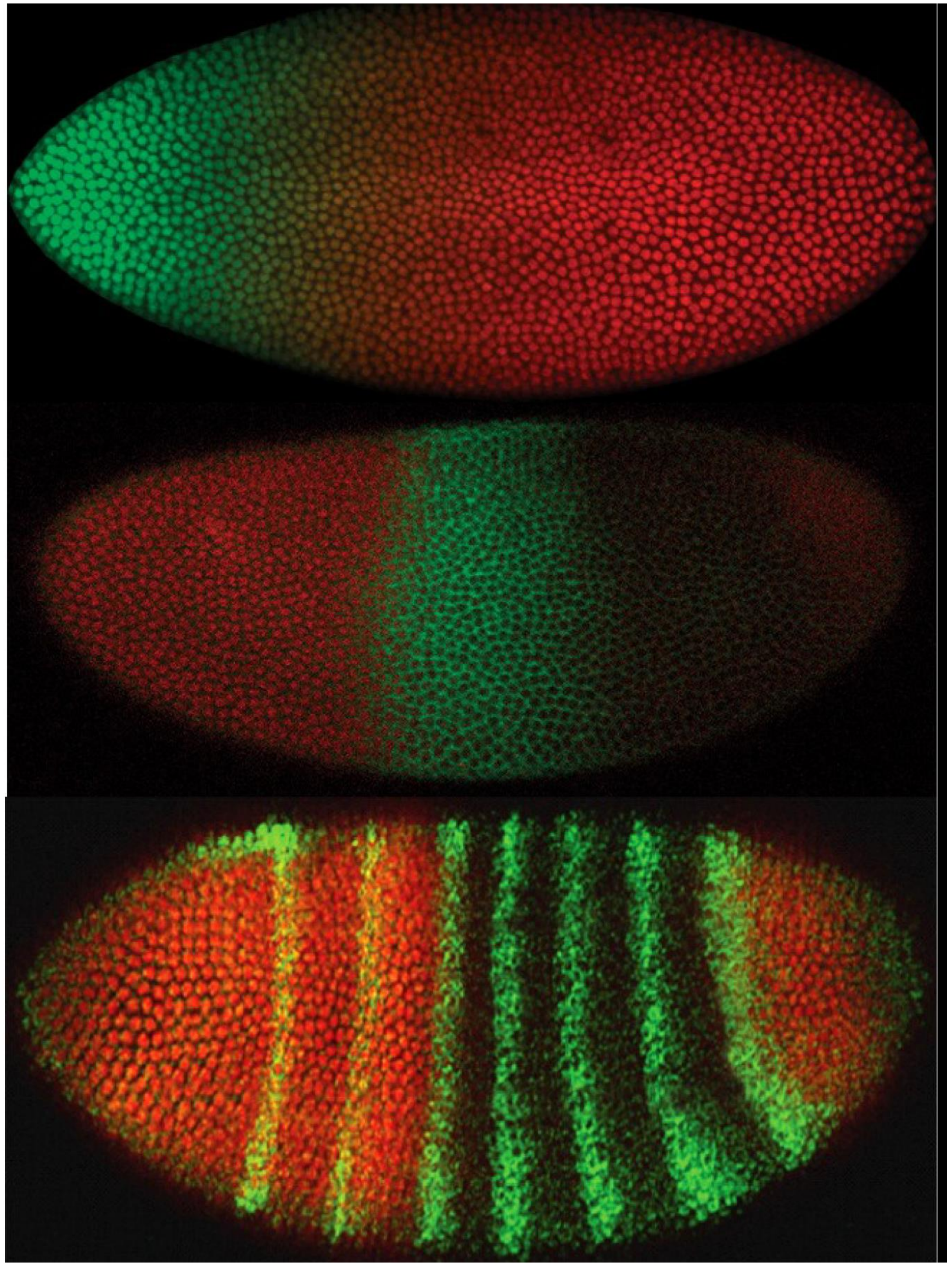
Features:

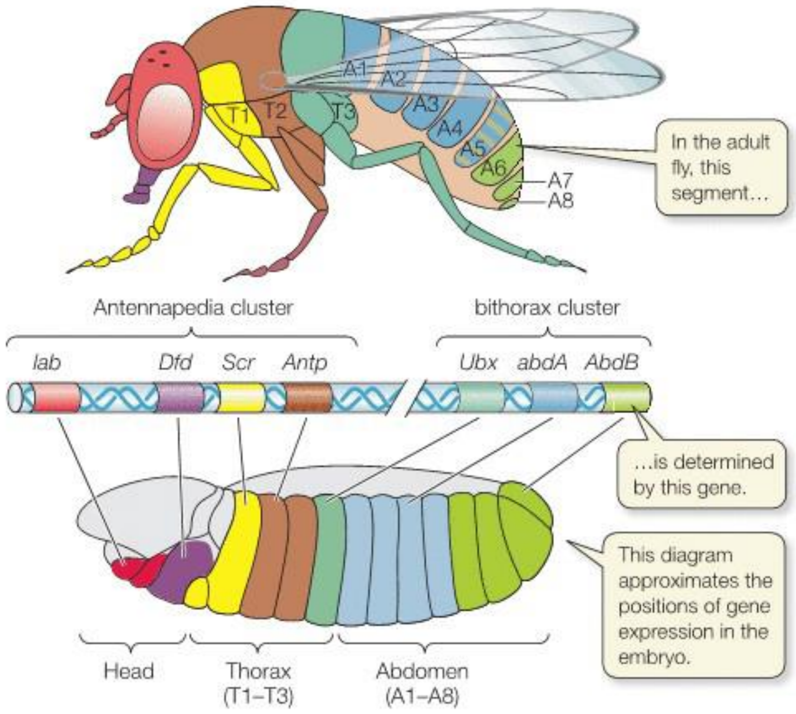
- More inputs (spatial features) and outputs (phenotypic traits) can be added to the same CPPN
  - Once discovered, the same pattern can be reused for more than one outputs (good)
  - But: pleiotropy: change to one gene affects multiple phenotypic traits (can be bad/disruptive)
- Otherwise, different traits can be genetically decoupled into different networks (greater modularity)

K. Stanley, "Compositional Pattern Producing Networks: A Novel Abstraction of Development"

Regularities without development

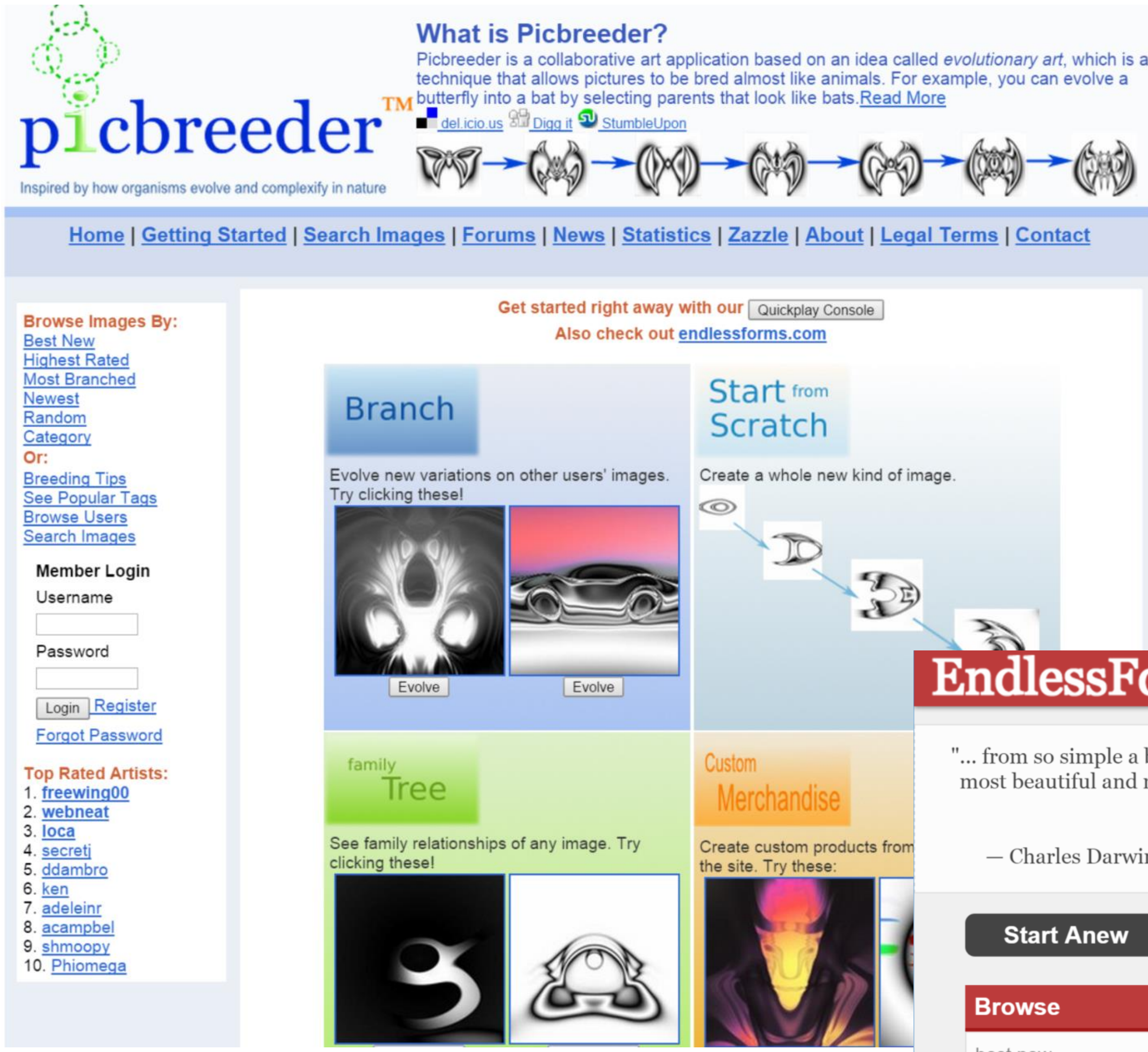


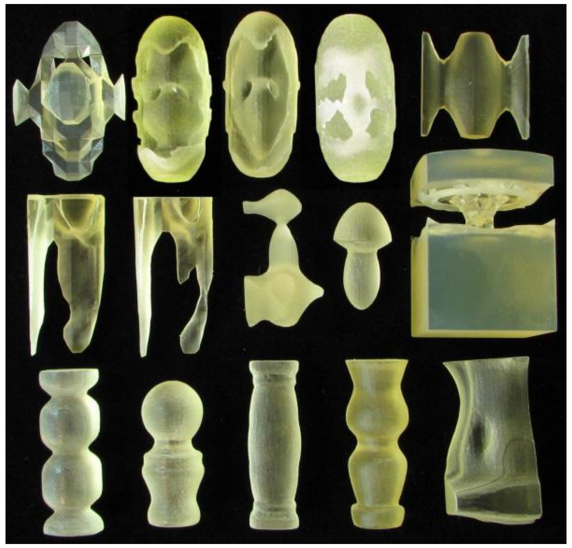


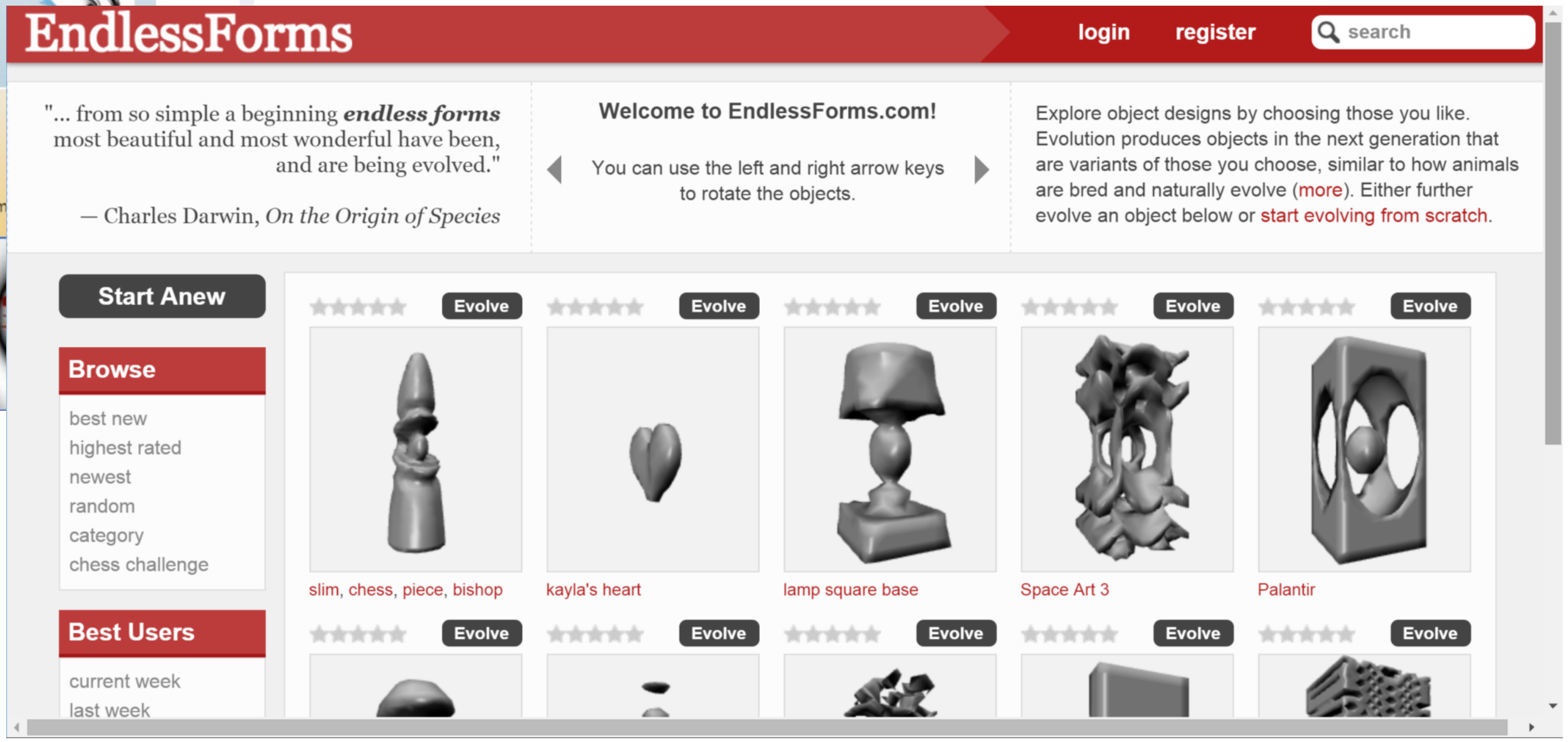


(Examples by N. Cheney)

Online interactive evolution platforms using CPPNs







Evolving brains - HyperNEAT

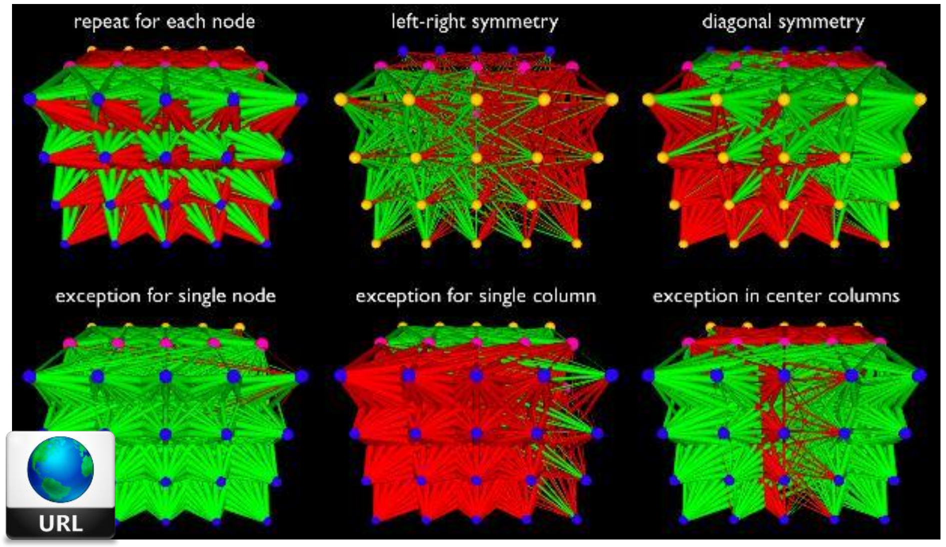
- The idea of <u>evolving CPPNs that paint the connectivity pattern of an ANN</u> is indeed at the core of a state of the art **<u>neuro-evolution</u>** technique:
- **HyperNEAT (Stanley et al., 2009)**
  - Encoding is based on CPPNs

    → <u>Scalability</u> (millions of connections)

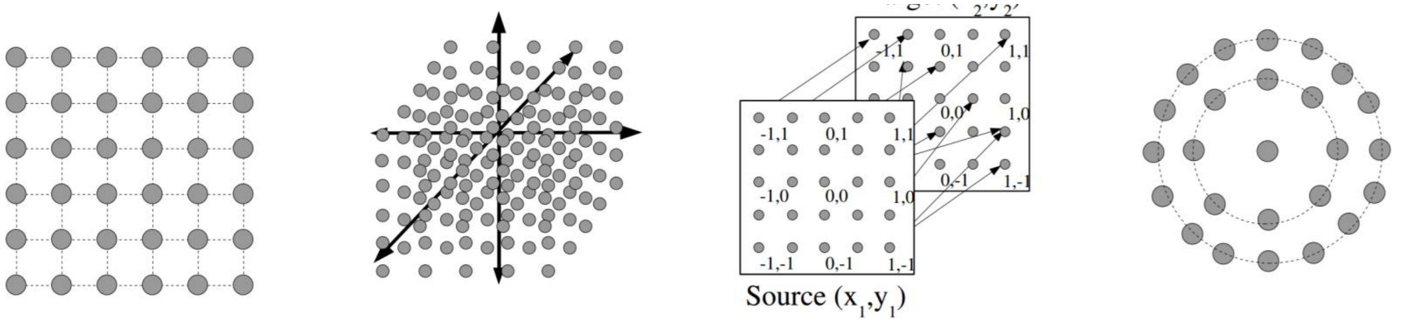
    → <u>Regular connectivity patterns</u>
  - ANN's nodes are arranged in a <u>substrate</u>, which has a certain <u>topology</u> (a square, a cube, etc.)

    → Different substrates are better suited for different tasks

    → <u>Substrate's structure can be evolved</u> itself (ES-HyperNEAT)



"Evolving Neural Networks That Are Both Modular and Regular", Huizinga, Mouret, Clune



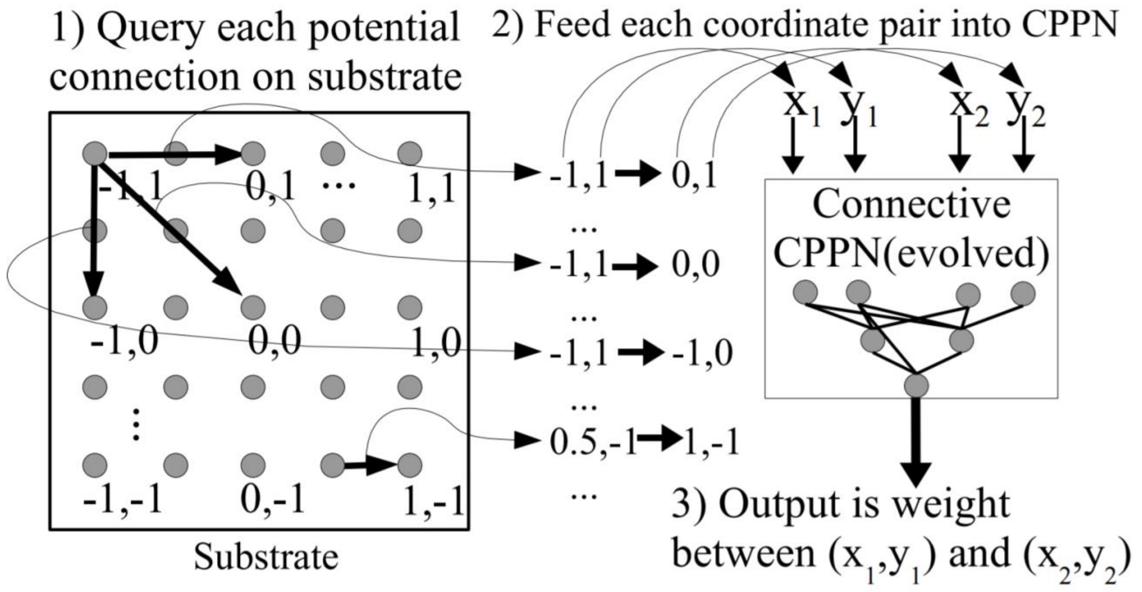
(a) Grid (b) Three-dimensional (c) Sandwich (d) Circular

Stanley et al 2009, "A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks"

Evolving brains - HyperNEAT

- The idea of evolving CPPNs that paint the connectivity pattern of an ANN is indeed at the core of a state of the art **neuro-evolution** technique:
- **HyperNEAT (Stanley et al., 2009)**
  - Evolved CPPN is queried for every potential connection to get the associated synaptic weight
    - 2D substrate (fig)
      → 4D CPPN (x1,y1,x2,y2)
    - 3D substrate
      → 6D CPPN (x1,y1,z1,x2,y2,z2)

  → "hypercube"



  - A connection is only expressed if the corresponding weight is above a given threshold → **ANN's topology is evolved** too (not only synaptic weights)

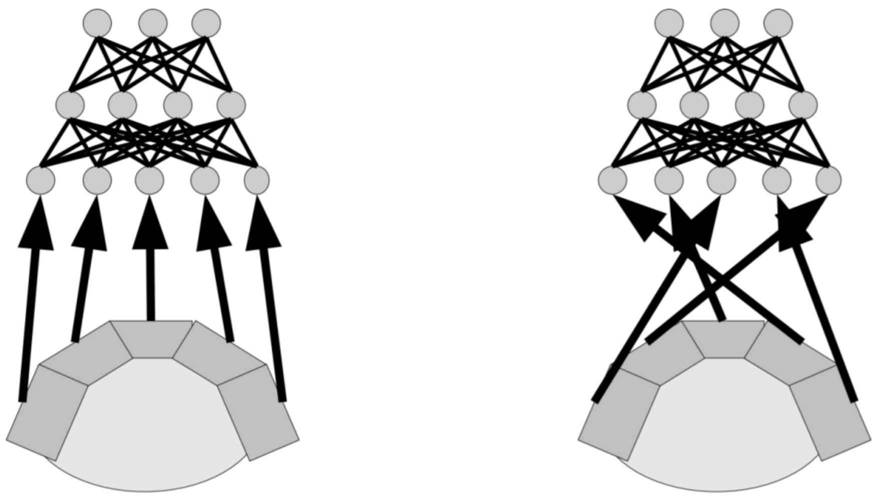
Evolving brains - HyperNEAT

- Consequence of topological arrangement of nodes:

  → <u>Correspondence between morphological, neurological, task topology is enforced</u>

- <u>E.g.</u>
  - Sensors that are close together are mapped to sensor neurons that are close together too
  - Same for outputs

  → <u>These regularities, usually neglected by learning algorithms, can simplify the learning problem</u>

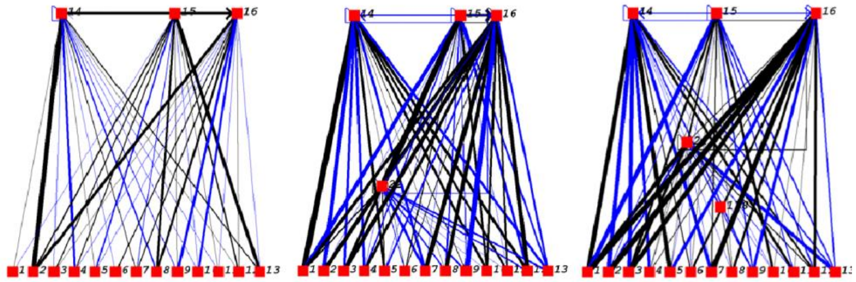


(a) Geometric Order (b) Random Order

Stanley et al 2009, "A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks"

NEAT

- In HyperNEAT, the CPPN that “paints” the connectivity pattern of the ANN is evolved with **NEAT (Neuro Evolution of Augmenting Topologies)**:
  - Evolves networks: originally ANNs, but with minor changes evolves CPPNs too, both topology and weights → CPPN-NEAT
  - “Complexification”: of networks and behaviors (simple → complex)
    - Biologically plausible
    - Helps reducing the search space (starts small, starts simple)
  - Highly evolvable: special crossover operator allows to effectively combine sub-functions computed by different ANNs
  - Speciation mechanism protects recent, possibly promising innovations from the unfair competition with already mature solutions



Stanley and Miikkulainen 2002, “Evolving Neural Networks through Augmenting Topologies”

Remarks on neuroevolution

- Different neuroevolution algorithms allow to evolve different aspects of ANNs
- Everything about an ANN can be evolved
  - Topology (which nodes, which connections)
  - Synaptic weights
  - Nodes activation functions
  - Local learning rules for lifetime learning (*evolution of learning*)
  - …

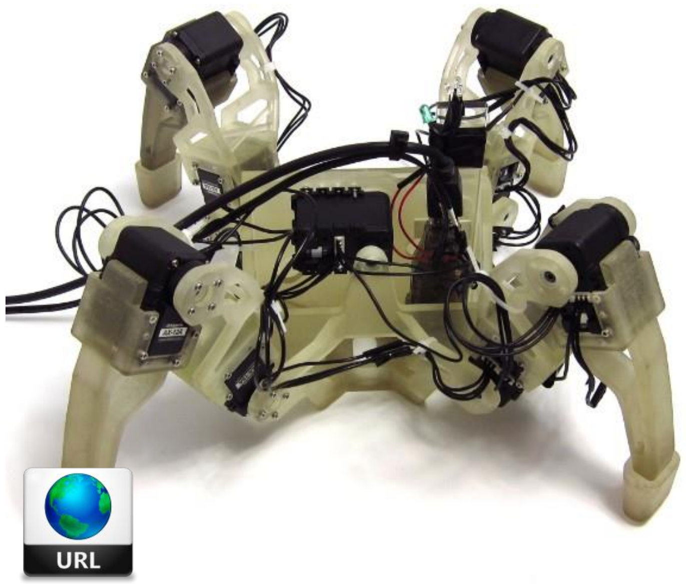
Remarks on neuroevolution

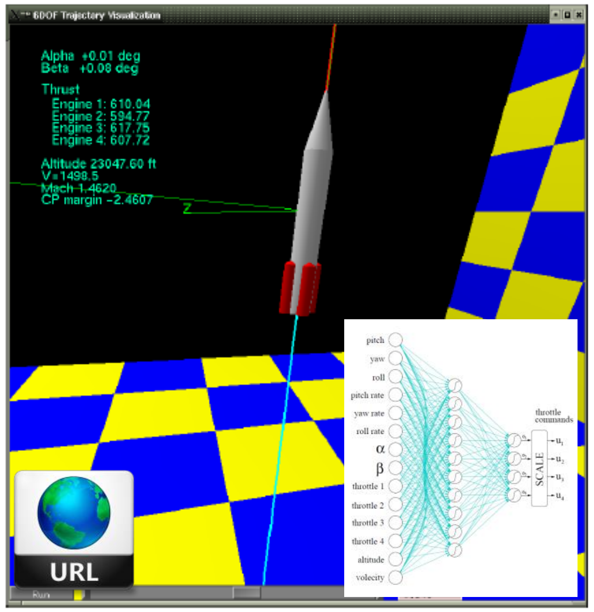
- Can be thought of as a method to <u>train ANNs in unsupervised settings</u>
  → <u>No input-output examples are provided</u>

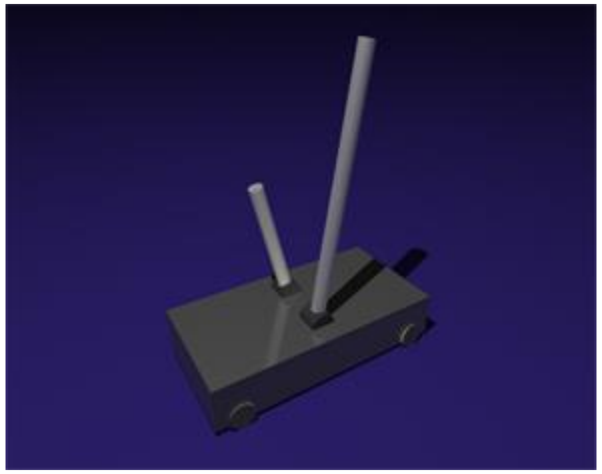
- Also very effective as <u>reinforcement learning</u> methods (learn agents' control policies to maximize cumulative reward in a given task environment), especially in <u>continuous and partially observable domains</u>

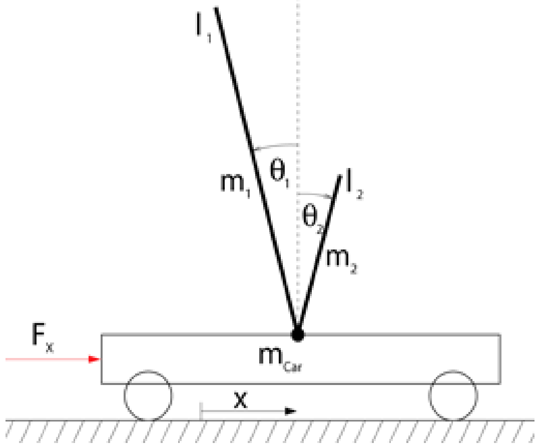
Neuroevolution – Some applications

- Proved to be effective in a variety of applications:
  - Adaptive nonlinear control of physical systems (robots, chemical plants, etc.)
  - Evolution of multi-modal cognitive architectures/behavior (e.g. human-like game play in videogames)
  - Evolution of large-scale brain-like structures



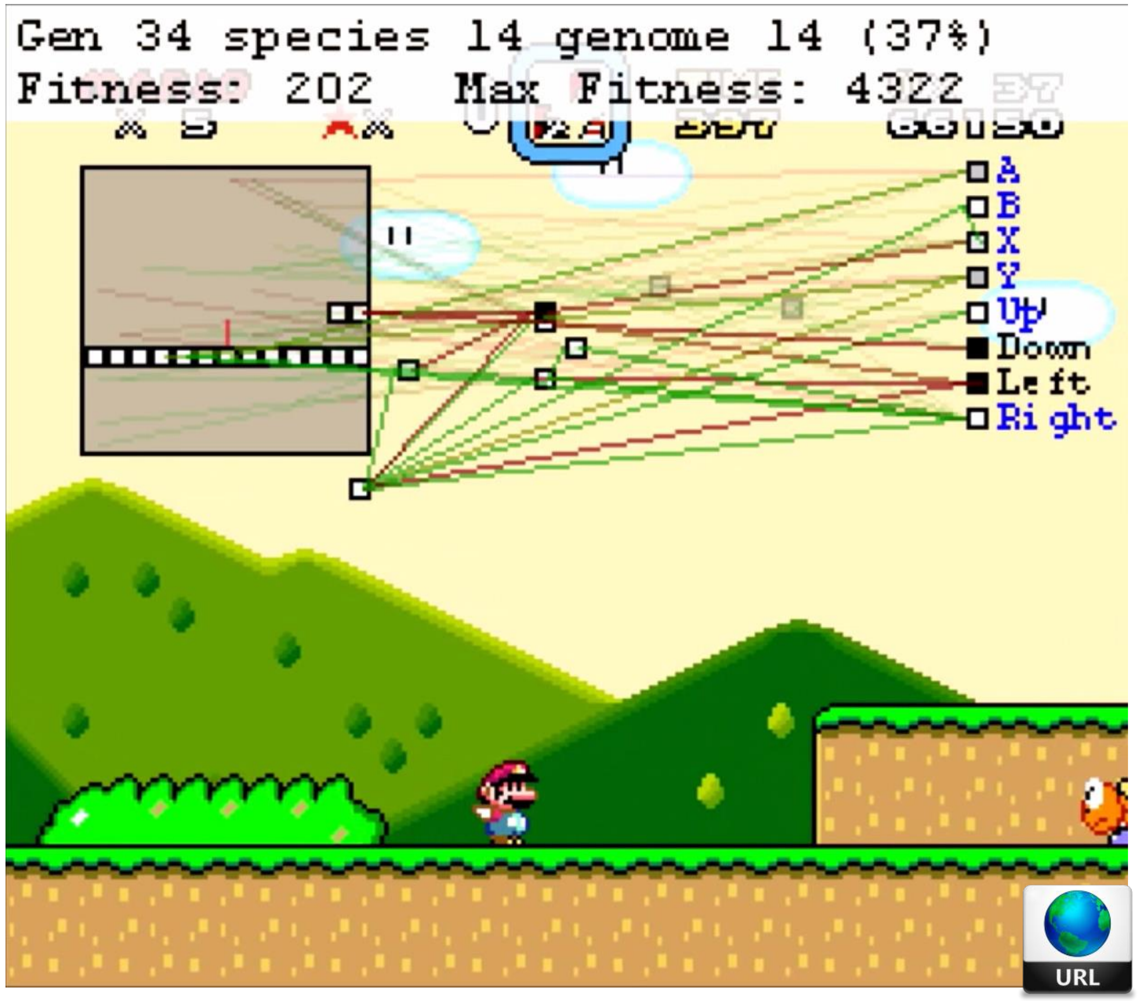






Inputs: θ_1, θ_2, x Output: F
Fitness: nr. of steps when (θ_1, θ_2, x) are within range

Neuroevolution – Some applications



Neuroevolved AI won 2012's BotPrize (goal: evolve human-like gameplay, playing against humans as well as other bots)

Rightmostvideo: Neuroevolved bot playing Unreal Tournament, judges viewpoint (aka: what is being killed by an AI like)

→ Neuroevolved AI broke the «human-like play barrier» for the first time (~Turing test, judged human > 50% of times)

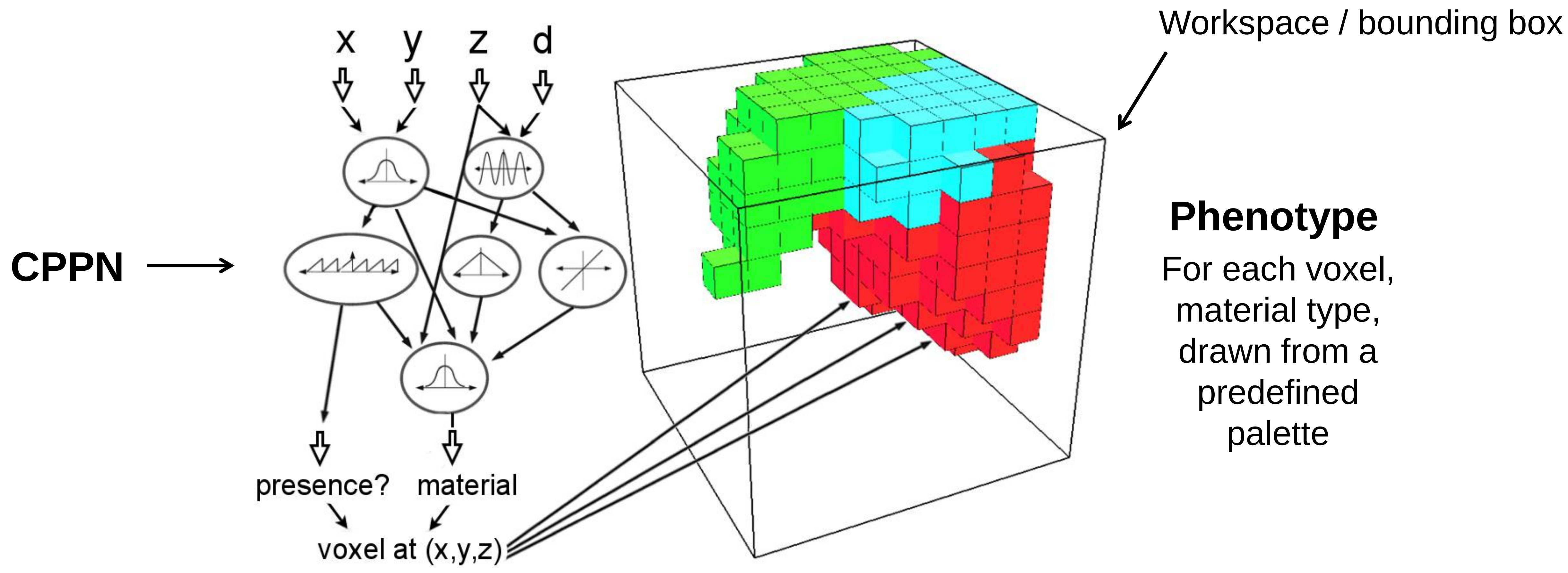
Remarks on neuroevolution

- **As most of the techniques we have seen, usage can be twofold:**
  - For practical applications, as <u>tools to solve complex problems</u> (e.g. robot control)
  - Given their biological inspiration, as <u>scientific tools</u> (e.g. investigate the evolution of brain-like structures → intelligence)

Back to CPPNs: Evolving bodies… with CPPNs

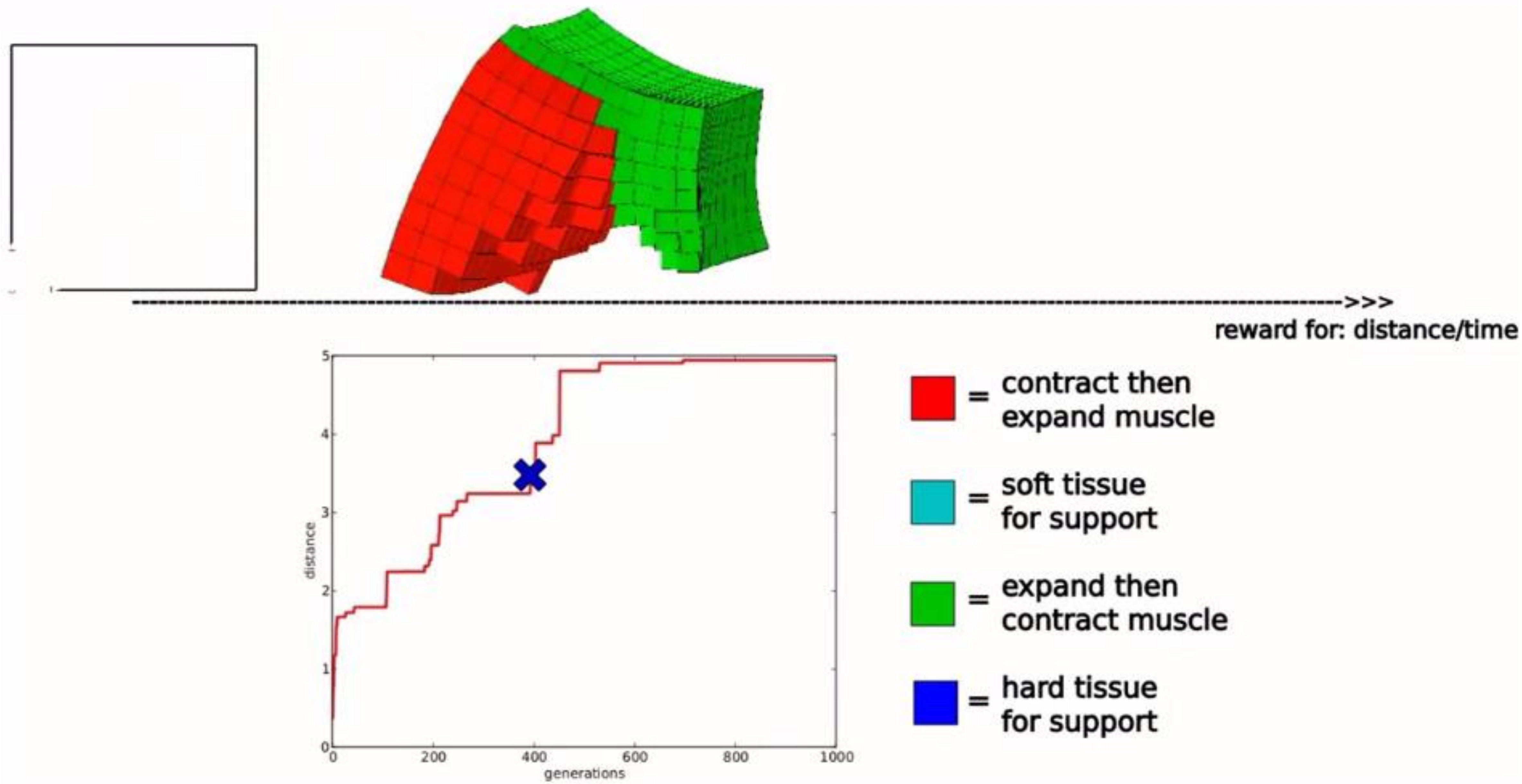
The same algorithms used to evolve regular ANNs' connection patterns (NEAT) can be used to evolve regular body plans

→ What changes is how CPPN outputs are interpreted



Cheney, Nick, et al. "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding." Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM, 2013.

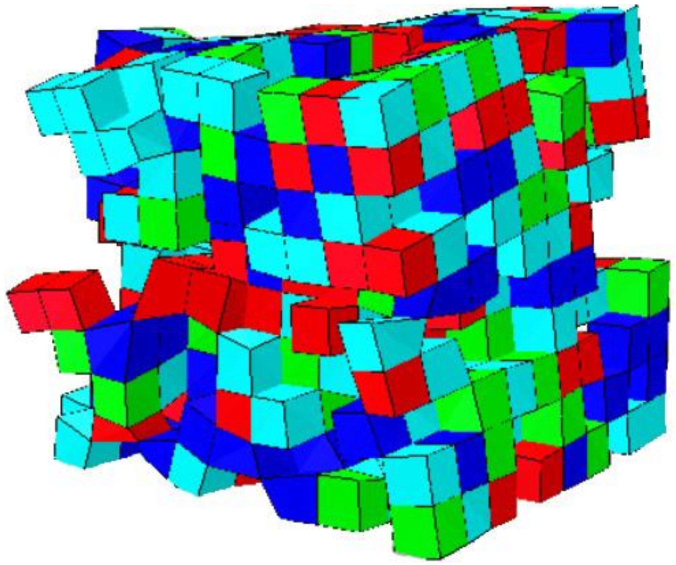
Evolving bodies… with CPPNs



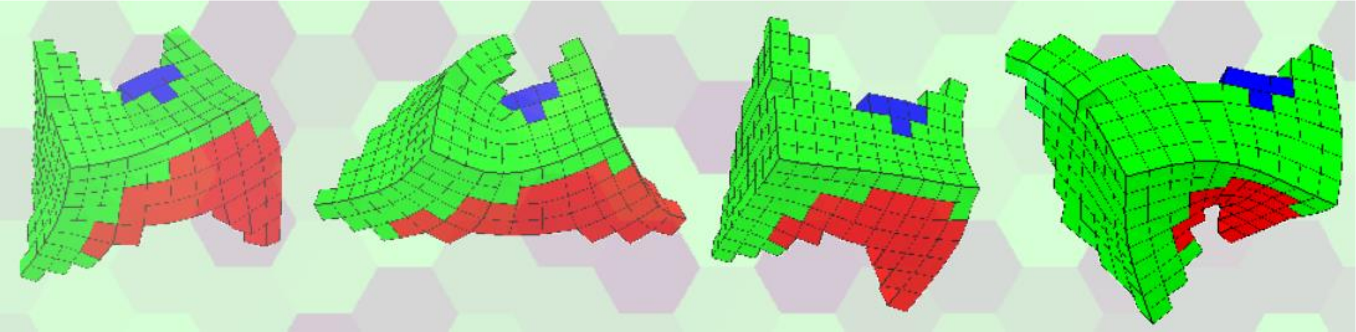
Cheney, Nick, et al. "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding." Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM, 2013.

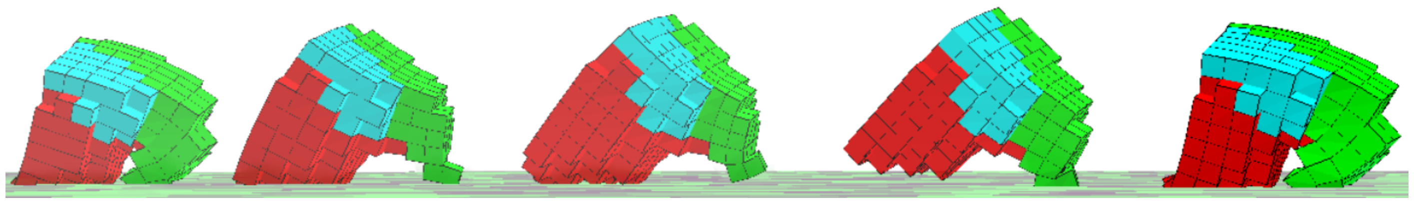
Evolving bodies… with CPPNs

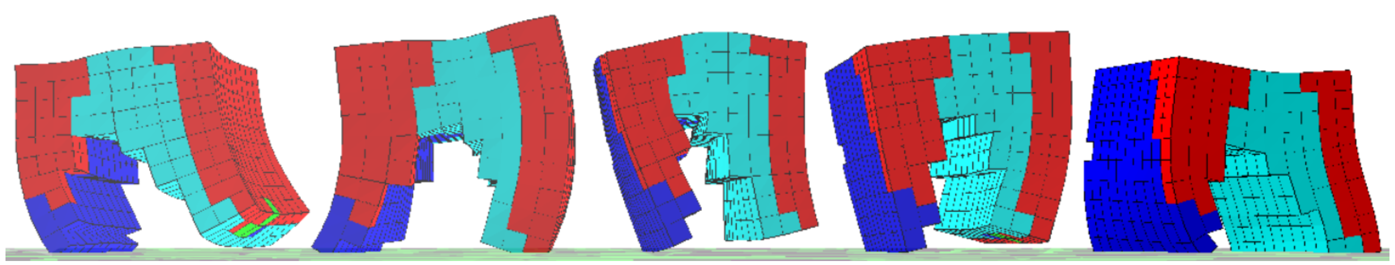
Sample morphology evolved with direct encoding → not regular



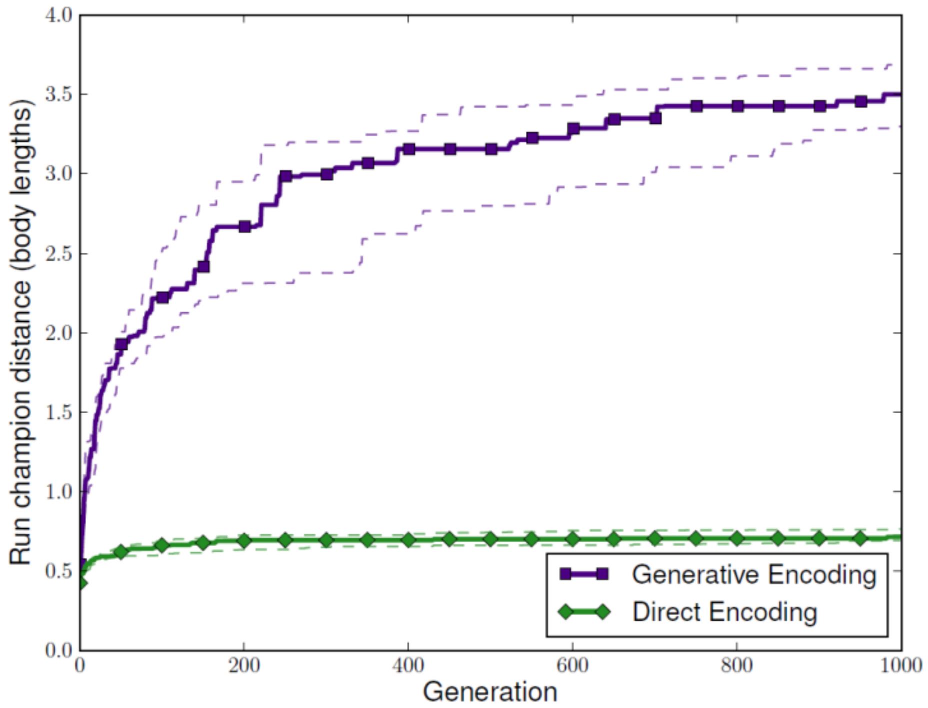
CPPN-NEAT evolved regular morphologies

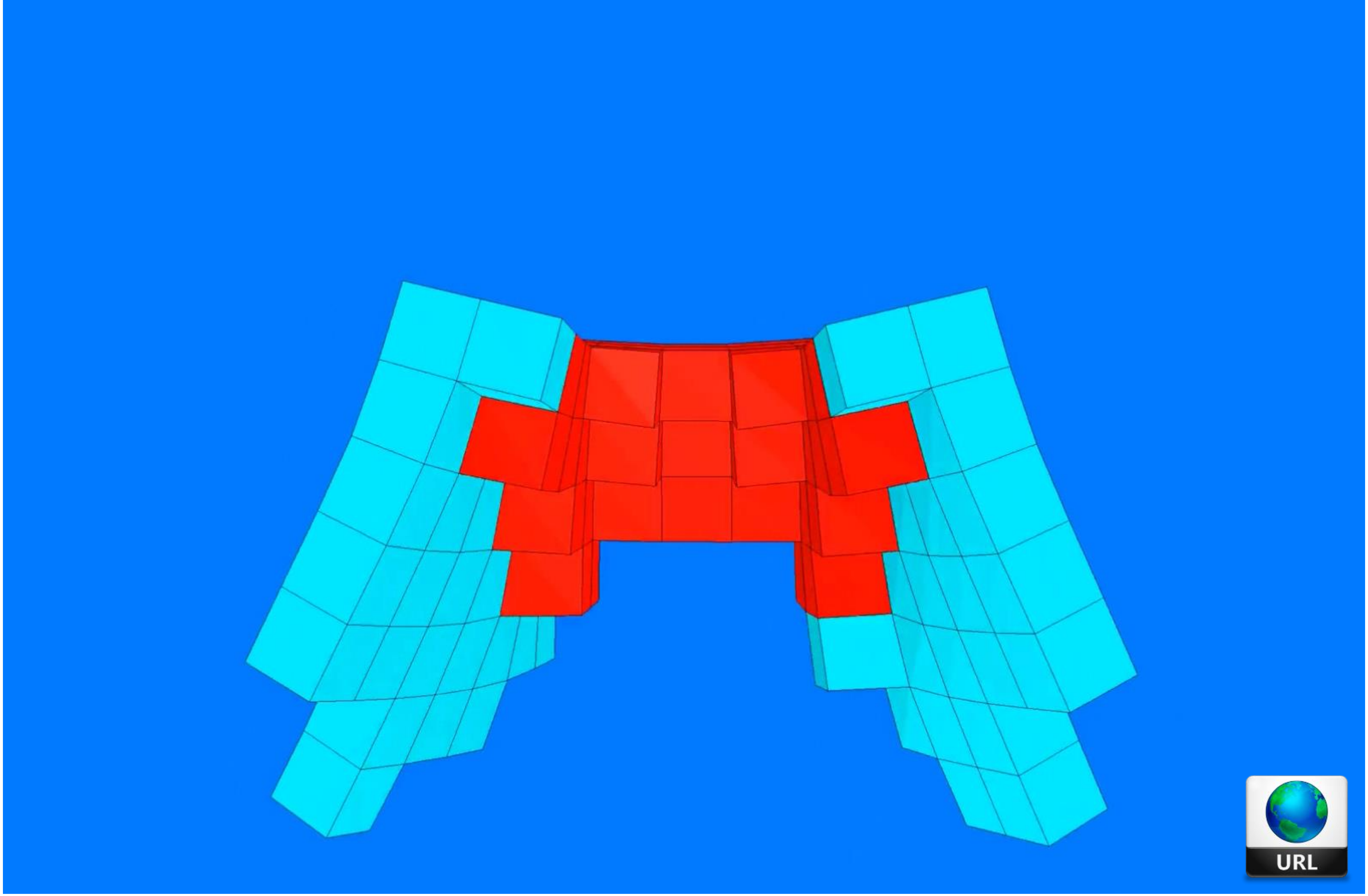






Regular morphologies
→ more effective robots

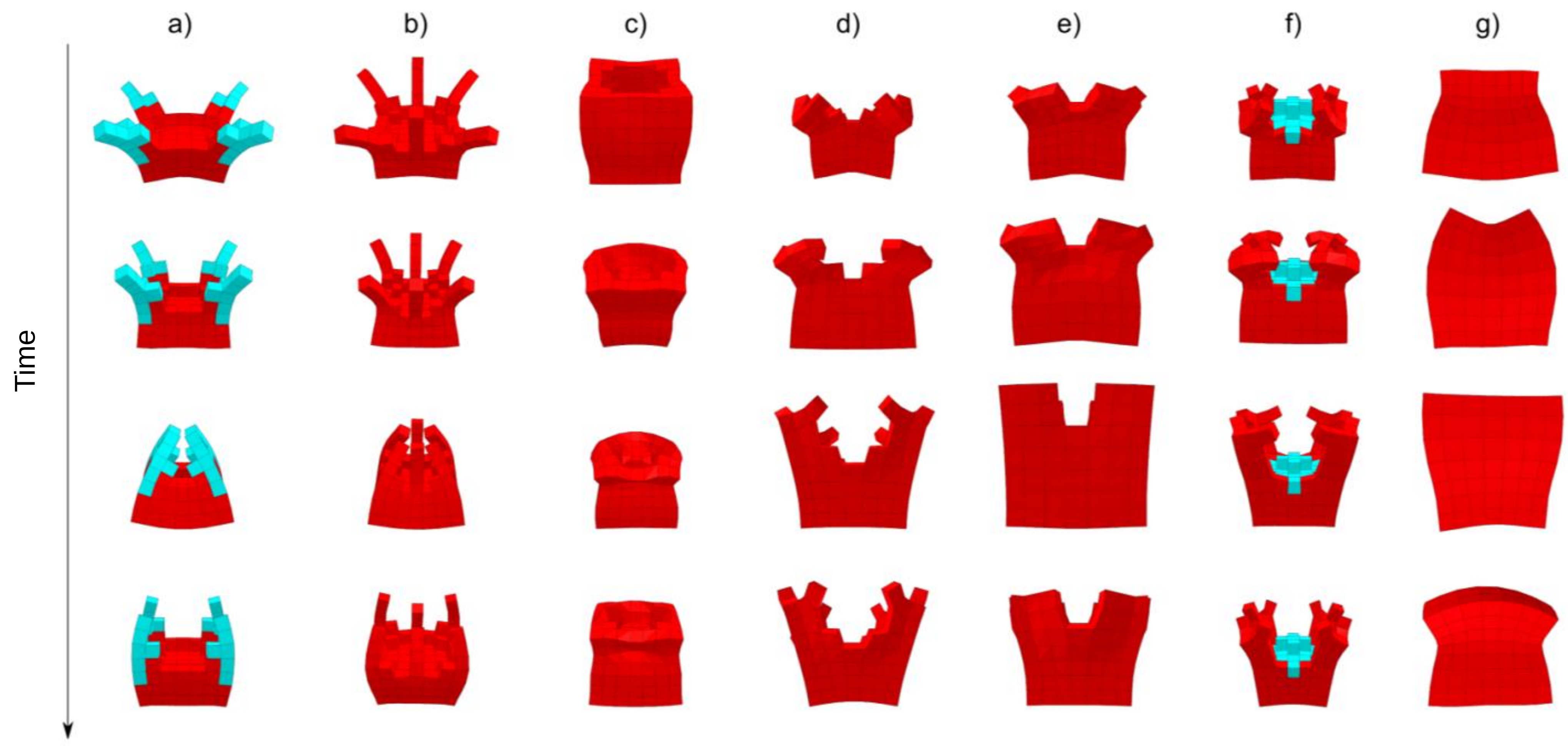




F. Corucci et al. "Evolving swimming soft-bodied creatures", ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, 2016 (late breaking abstract)

Evolving bodies… with CPPNs

Environment: simple fluid dynamics model is added (mesh-based resistive drag)
Evolved aspects: topology, active/passive material, actuation frequency and phase offset, stiffness distribution
Artificial Life experiment: investigating the effect of environmental transitions water ←→ land on evolved morphologies and behaviors



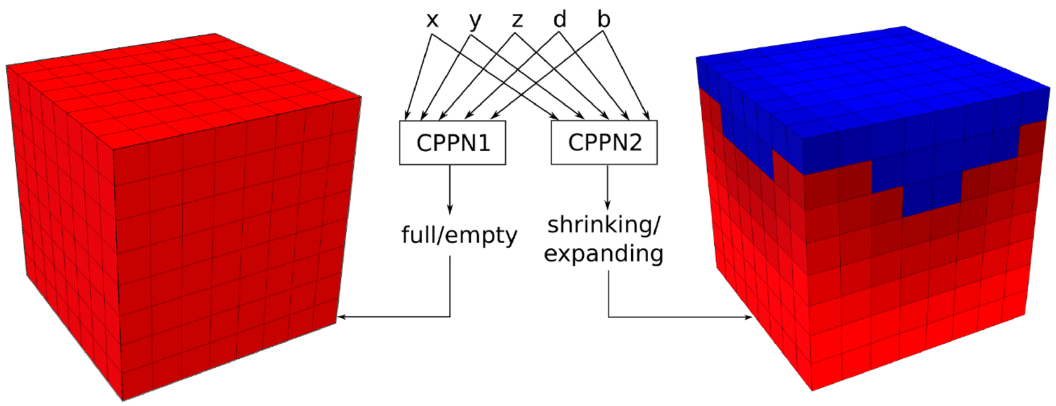
Note the **emergent morphological regularity** (modular appendages, symmetry)

→ Not explicitly rewarded

Partly due to the regular encoding, partly implicitly selected by evolution

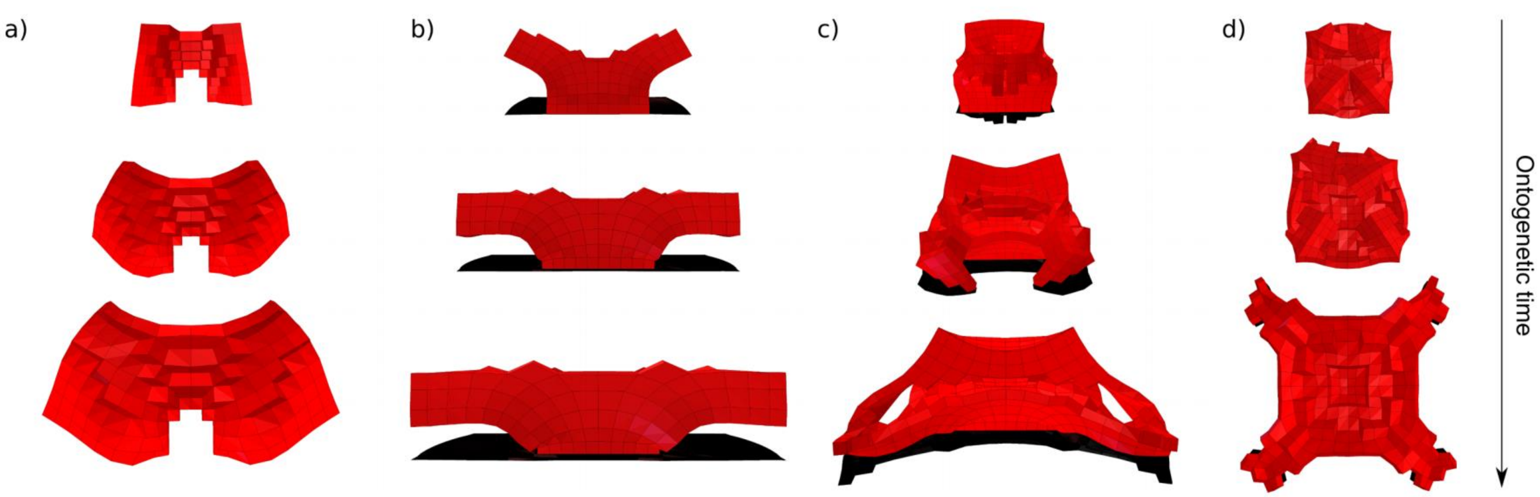
F. Corucci et al. "Evolving swimming soft-bodied creatures", ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, 2016 (late breaking abstract)

Evolution of development… with CPPNs

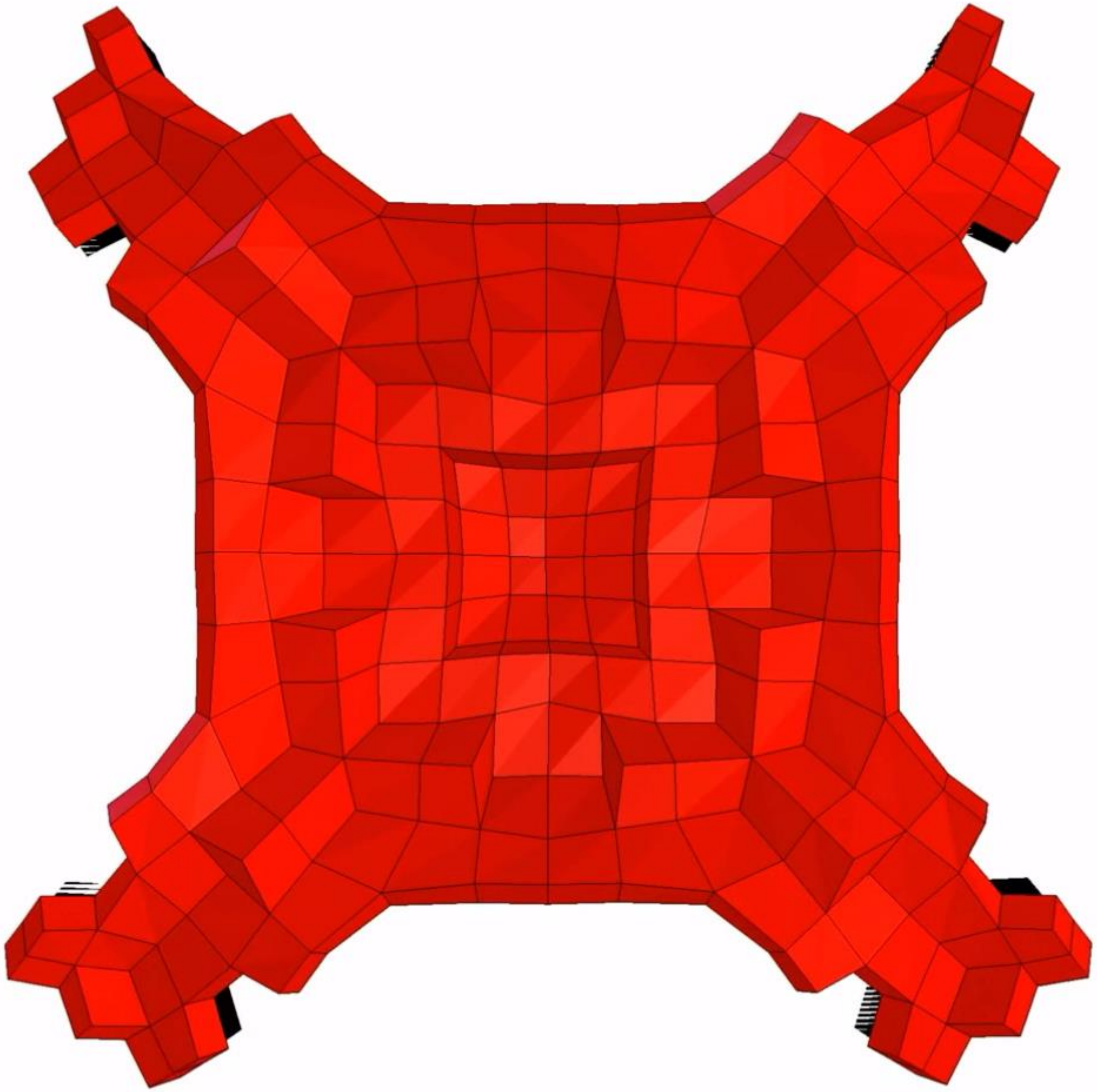


Evolved aspects: topology, parameters of a developmental process unfolding over time

Figure 2: Different attributes of the robot can be "painted" by different CPPNs. CPPN1 dictates the geometry of the robot, while CPPN2 determines its growth properties. In the current system red voxels expand in response to environmental stimuli, while blue ones shrink.



F. Corucci et al. *"Material properties affect evolution's ability to exploit morphological computation in growing soft-bodied creatures,"* ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, 2016



F. Corucci et al. *"Material properties affect evolution's ability to exploit morphological computation in growing soft-bodied creatures,"* ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, 2016

Evolution of development… with CPPNs

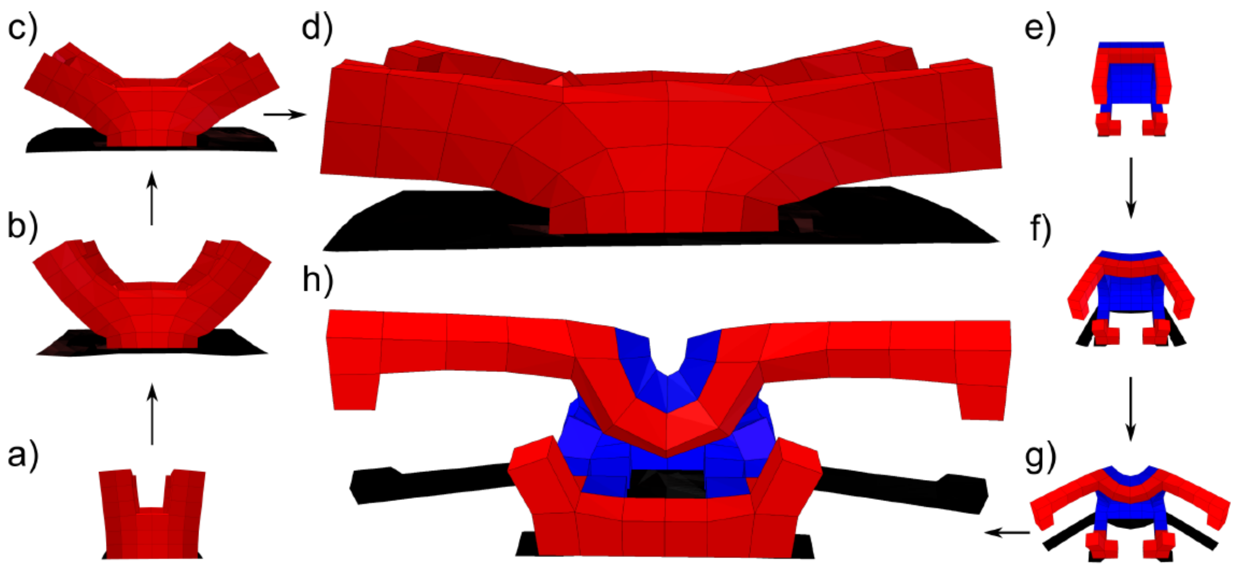


Figure 1: A soft (a-d) and stiff (e-h) robots evolved to grow towards two lateral light sources. Red voxels expand in response to environmental stimuli, blue ones shrink. While the soft robot only employs expanding voxels and effectively exploits morphological computation, passive dynamics, and the interaction with the environment to solve the task, the stiff one is prevented from doing so due to its unsuitable material properties, and had thus to evolve a more complex and active form of control in order to achieve the same result. See them in action at: `https://youtu.be/Cw2SwPNwcfM`

This setup pointed out **interesting relationships between material properties and the evolution of morphological computation in growing soft robots**

Unsuitable material properties can prevent evolution from discovering and exploiting morphological computation, which results in an automatic complexification of the controller

Morphological computation was quantified using information theoretic measures (Shannon entropy)
→ For a given fitness level, delta entropy between controllers

F. Corucci et al. *"Material properties affect evolution's ability to exploit morphological computation in growing soft-bodied creatures,"* ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, 2016

Evolution of development… with CPPNs

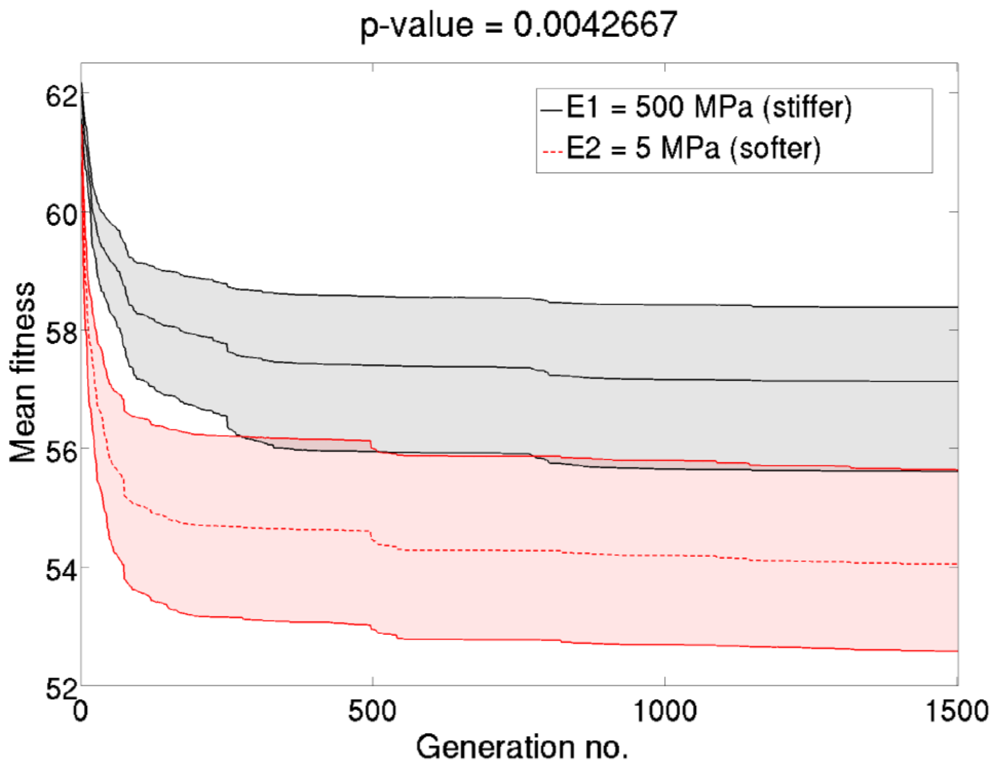


Figure 4: Average fitness over 20 independent runs. Softer robots have an evolutionary advantage over stiffer ones in this task/environment.

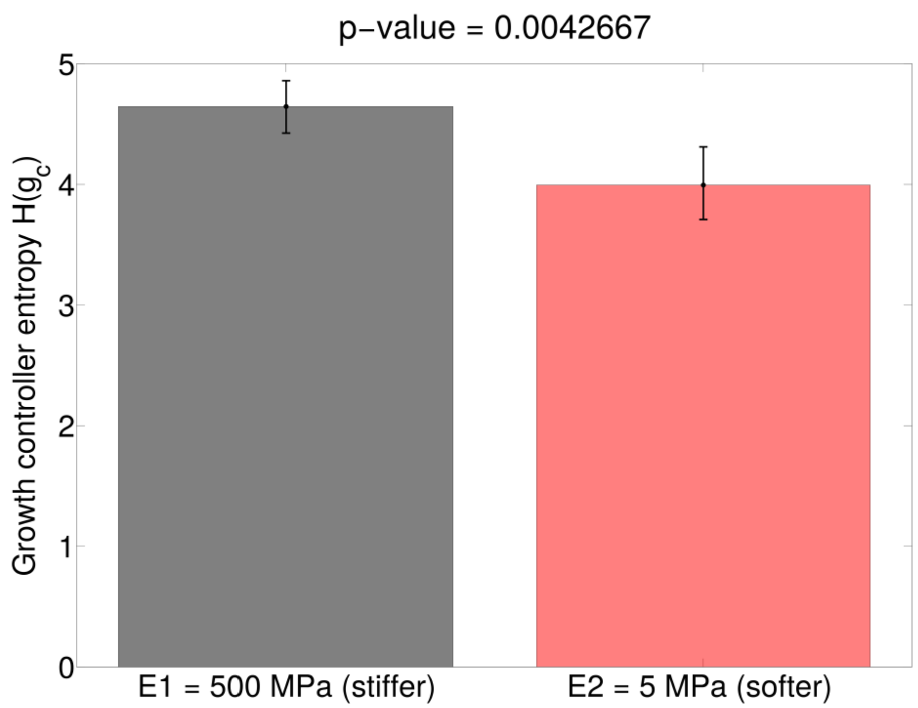


Figure 6: Stiffer robots exhibit more complex growth controllers than softer ones ($p < 0.005$), yet the latter achieve better performances (Fig. 4). It is argued that this difference is due to morphological computation, strictly connected to the material properties of robots in the two treatments.

For the task environment at hand, **softer robots achieved better performances with simpler controller**, thanks to morphological computation

F. Corucci et al. *"Material properties affect evolution's ability to exploit morphological computation in growing soft-bodied creatures,"* ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, 2016

Other indirect encodings – L-systems

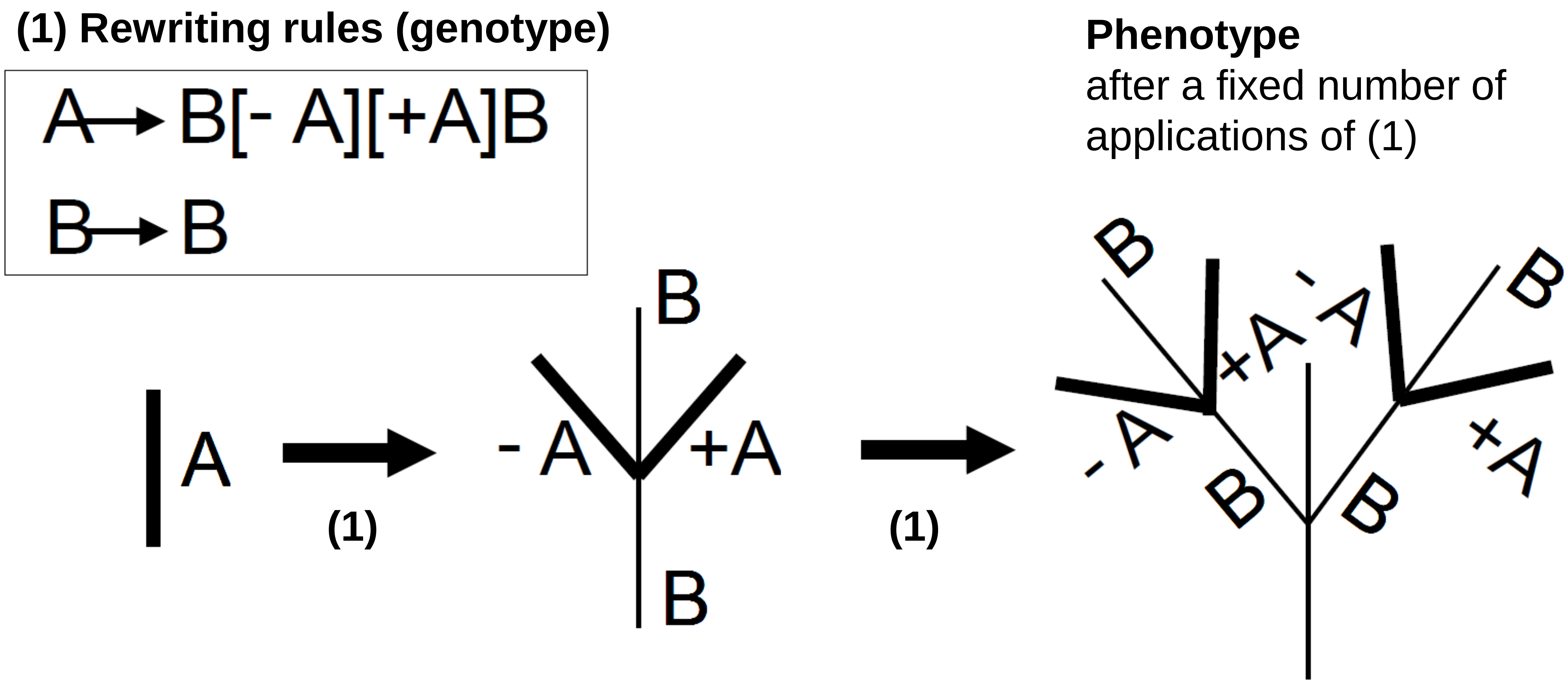
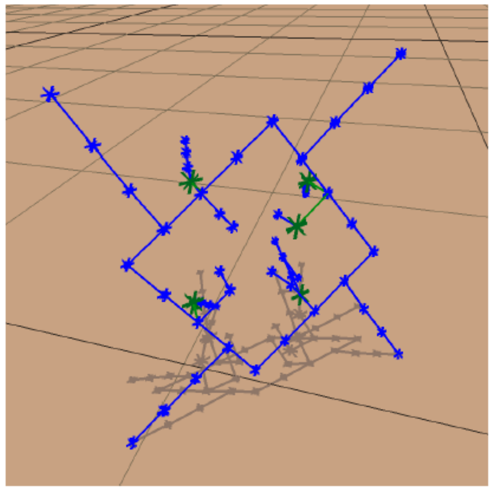


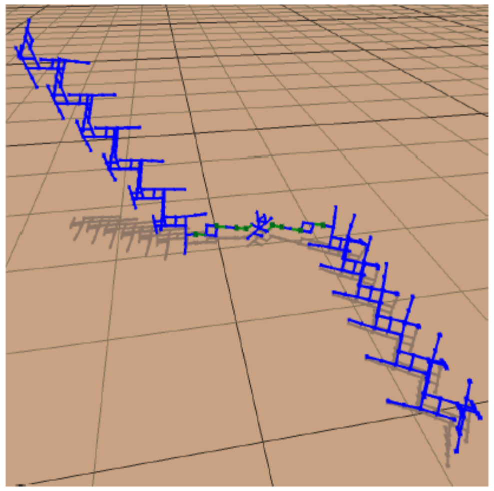
Figure 1: **Grammatical Approach Example (L-systems; Lindenmeyer 1968)..** The two rewrite rules (inset) describe the growth of a tree-like morphology. The symbol A, shown as a thick line in the tree, is the only symbol that is rewritten in this grammar. The symbol B, which does not expand, becomes a thin branch, and $-$ and $+$ determine relative angles of branches expanded from A symbols. This example illustrates how a few simple generative rules can encode a large structure with many components.

Stanley, Kenneth O., and Risto Miikkulainen. "A taxonomy for artificial embryogeny." *Artificial Life* 9.2 (2003): 93-130.

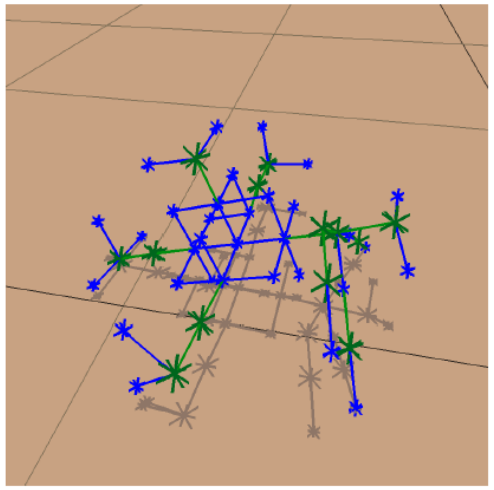
Other indirect encodings – L-systems



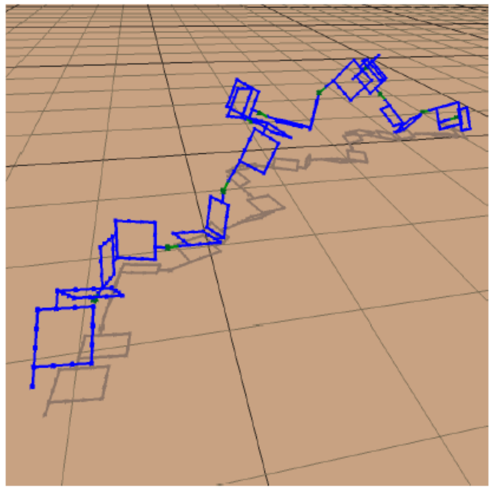
(a)



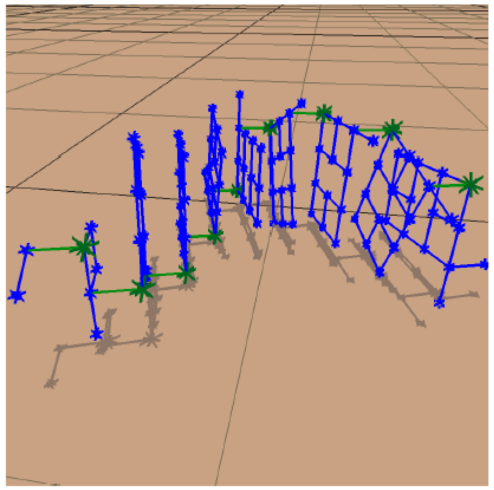
(b)



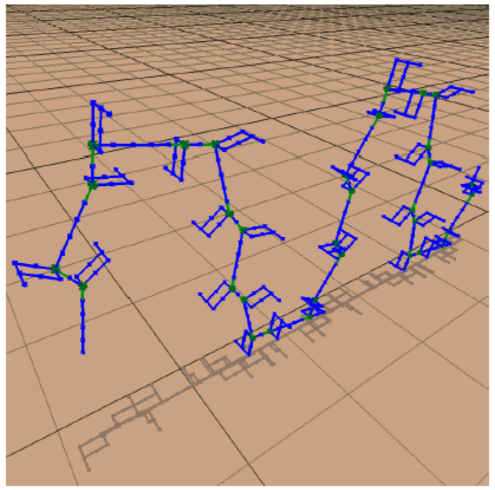
(c)



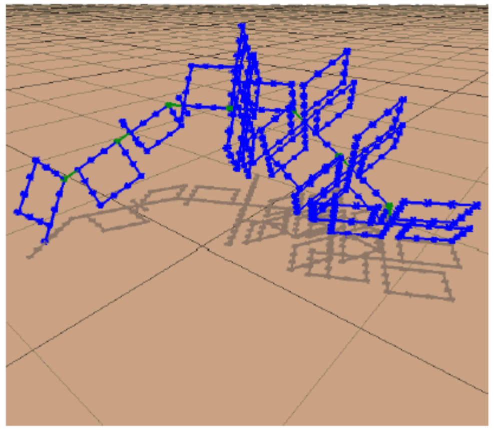
(d)



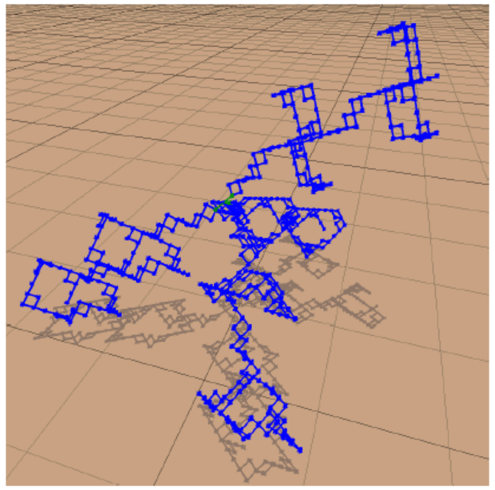
(a)



(b)



(c)



(d)

Same grammar-based approach is used to generate both <u>body</u> and <u>brains</u>

Hornby, Gregory S., and Jordan B. Pollack. "Body-brain co-evolution using L-systems as a generative encoding." Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). 2001.

Other indirect encodings – L-systems

Grammar-based approaches are particularly suitable to reproduce plant-like structures and **fractal** patterns

https://en.wikipedia.org/wiki/L-system

Other indirect encodings – Graph based

- Functionally **similar to grammar-based** approaches
- Similarly, they can encode both artificial <u>brains</u> and <u>bodies</u>

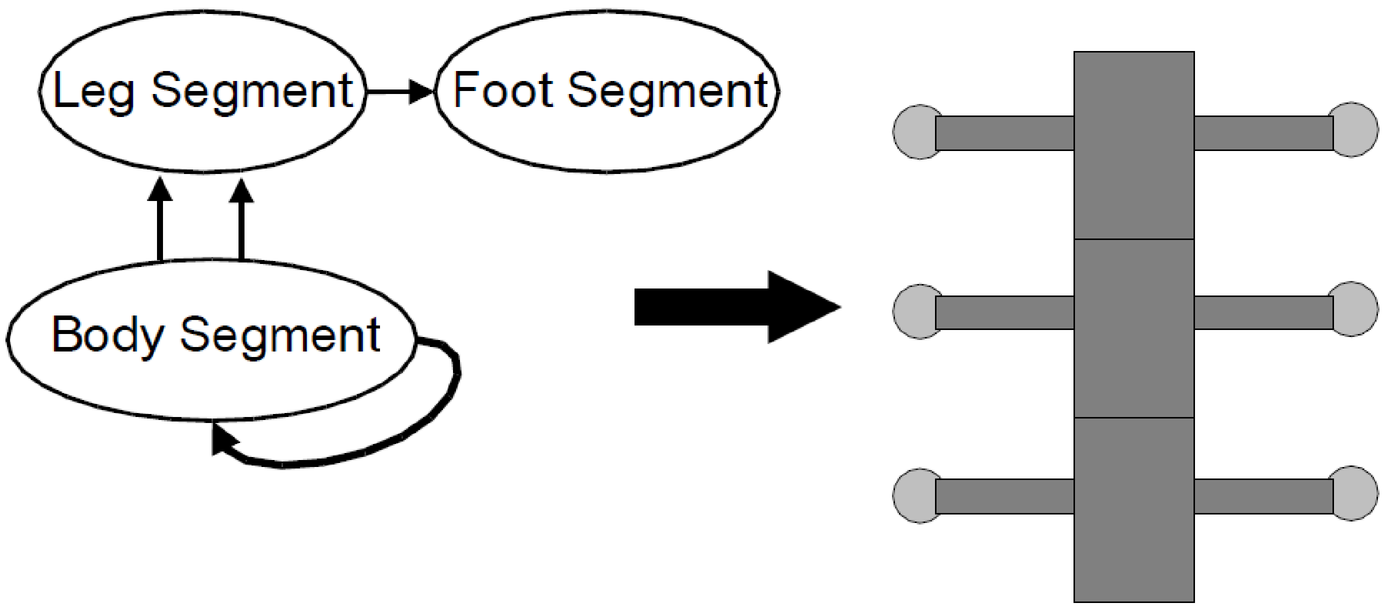


Figure 2: **Development of Body Morphology (Sims 1994).** The graph on the left specifies how the morphology on the right develops. The body segments repeat because of the recurrent loop on the "body segment" instruction node, which allows the reuse of genetic code. The number of repetitions is determined by an evolved parameter in the loop which is not shown. The final structure is a centipede-like creature with six legs and feet on each leg. Sims' work has inspired many AE researchers to explore body-brain evolution in simulated 3D environments.

Stanley, Kenneth O., and Risto Miikkulainen. "A taxonomy for artificial embryogeny." *Artificial Life* 9.2 (2003): 93-130.

Other indirect encodings – Graph based

Genotype: directed graph. **Phenotype:** hierarchy of 3D parts.

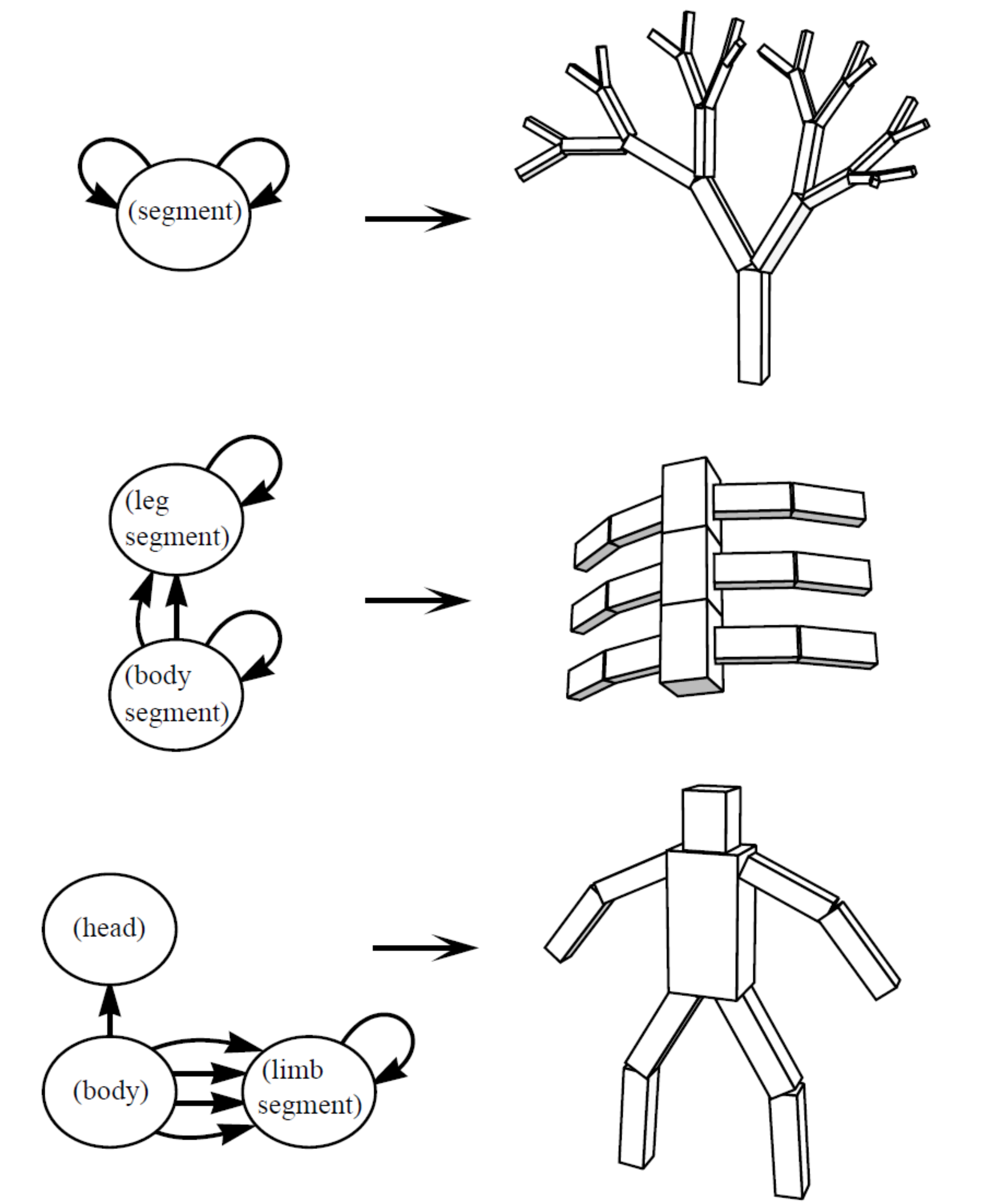


Figure 1: Designed examples of genotype graphs and corresponding creature morphologies.

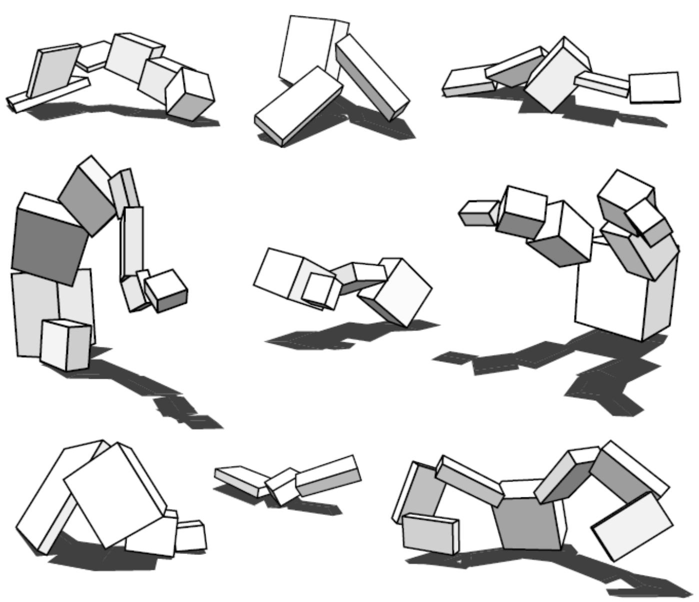


Figure 7: Creatures evolved for walking.

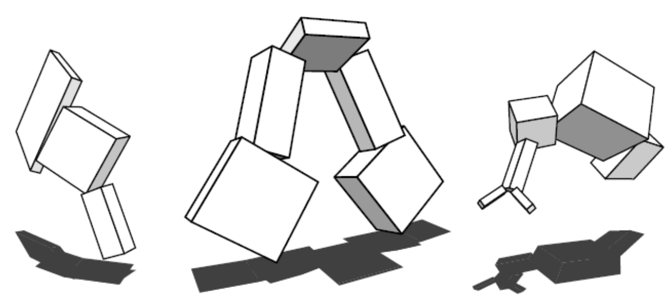


Figure 8: Creatures evolved for jumping.

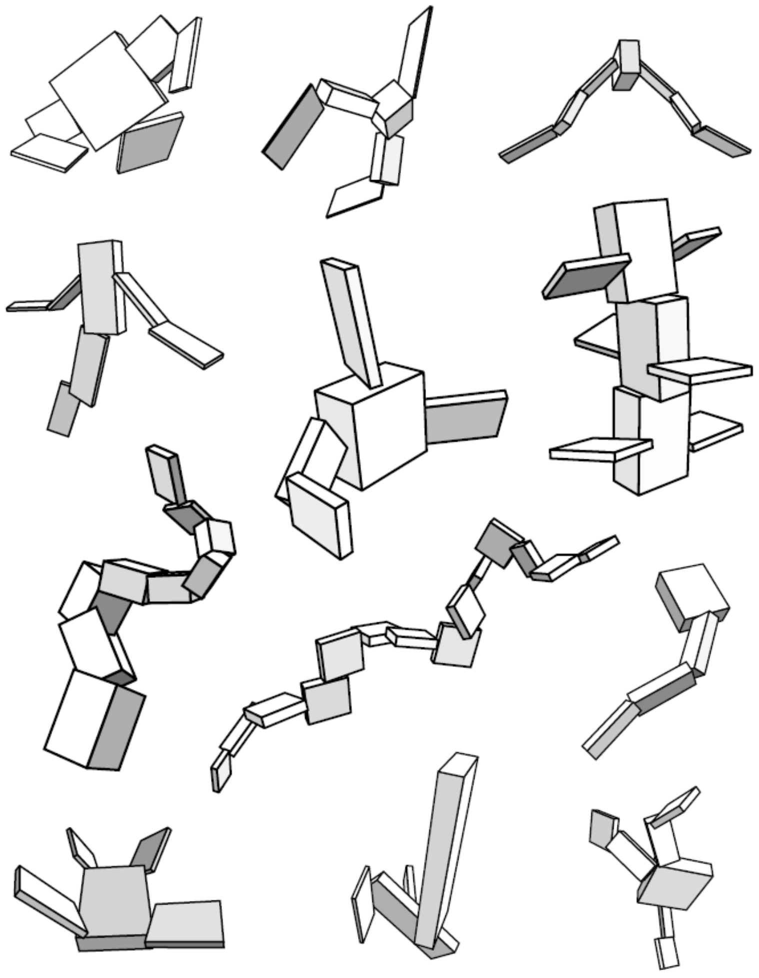


Figure 6: Creatures evolved for swimming.

Sims, Karl. "Evolving virtual creatures." Proceedings of the 21st annual conference on Computer graphics and interactive techniques. ACM, 1994.

Other indirect encodings – Gene Regulatory Networks

Networks that represent interactions governing gene expression

Genetic material composed of two parts:

- Coding region (White): instructions for a *gene product* (e.g. protein)

- Regulatory region (Gray): conditions under which the gene is expressed (i.e. product is created)

A,B,C,D: chemicals (e.g. proteins' concentration), triggering coding regions

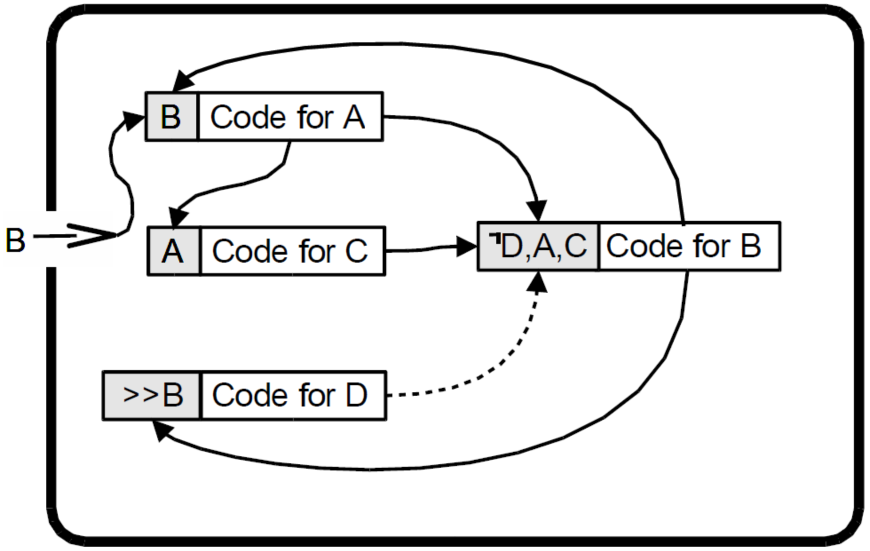


Figure 4: **Genetic Regulatory Network Example.** Each gene is modeled as a regulatory region and a coding region that codes a particular product (e.g. a protein in a natural cell). A simple network showing how different gene products of some genes regulate other genes is shown inside a cell, depicted as a rounded rectangle. The network describes a system that produces a number of products and then turns off when enough of product B is produced. The symbol ">>" means a large amount. The diagram shows that the entire network becomes activated when product B enters the cell from an external source. B causes A to be produced, which in turn causes C to be produced by another gene. A and C, without D in the cell cause more B to be created, which in turn feeds back into the production of A, further strengthening the cycle. Eventually, when a great deal of B is present, D is finally produced, stopping the generation of B and ending the feedback cycle. The GRN shows that interesting dynamics can result from the regulatory interactions of different genes. In an AE model, gene products might e.g. cause axons to grow or reduce neural thresholds.

GRN are usually embedded in artificial cells. Chemical gradients diffuse in the environment, influencing expression patterns.

Usually same GRN in all cells

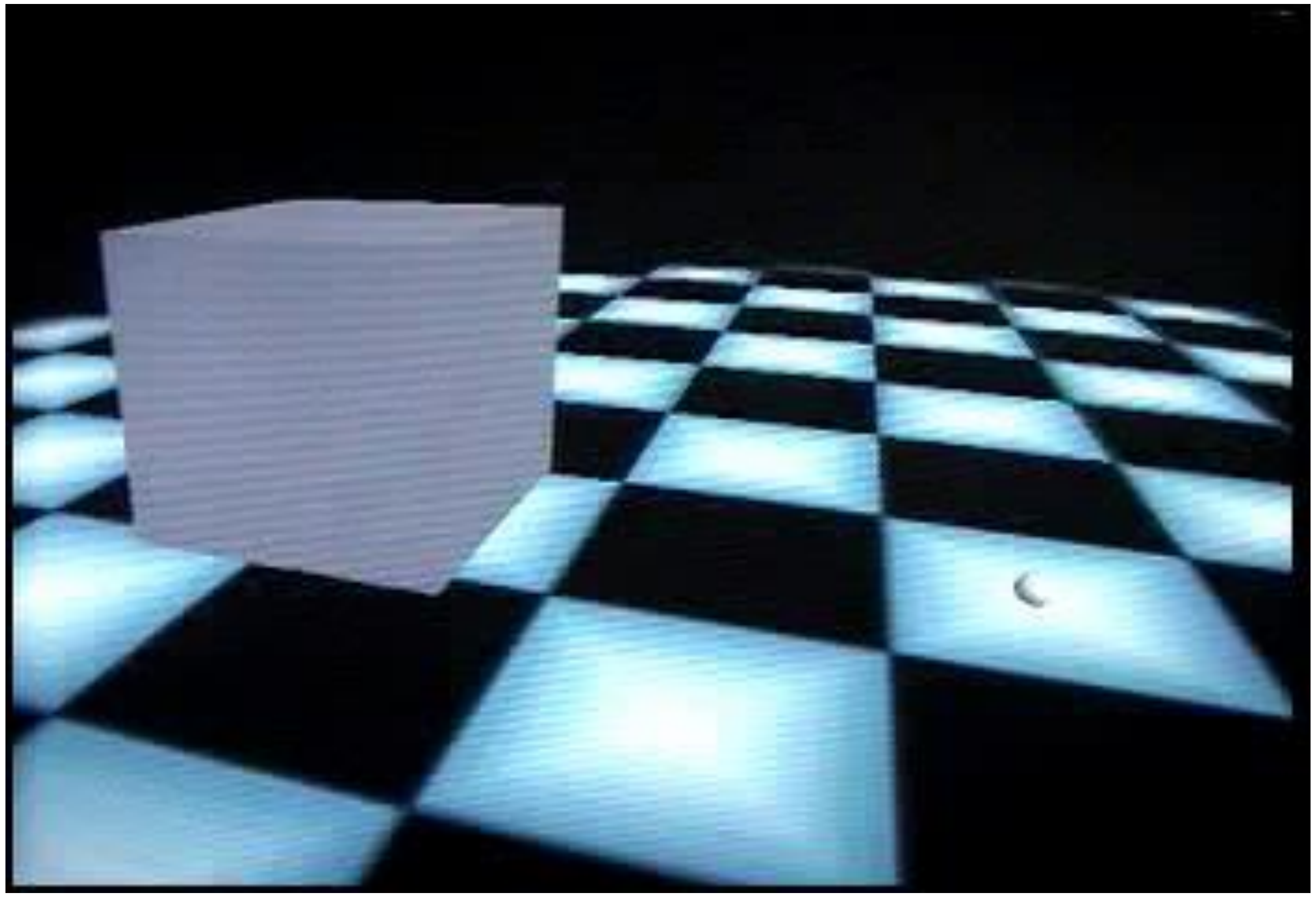
→ Different products based on local environment and feedback loops

As before, can implement both morphogenesis (development of morphology) and neurogenesis (development of neural system), depending on which gene products are defined.

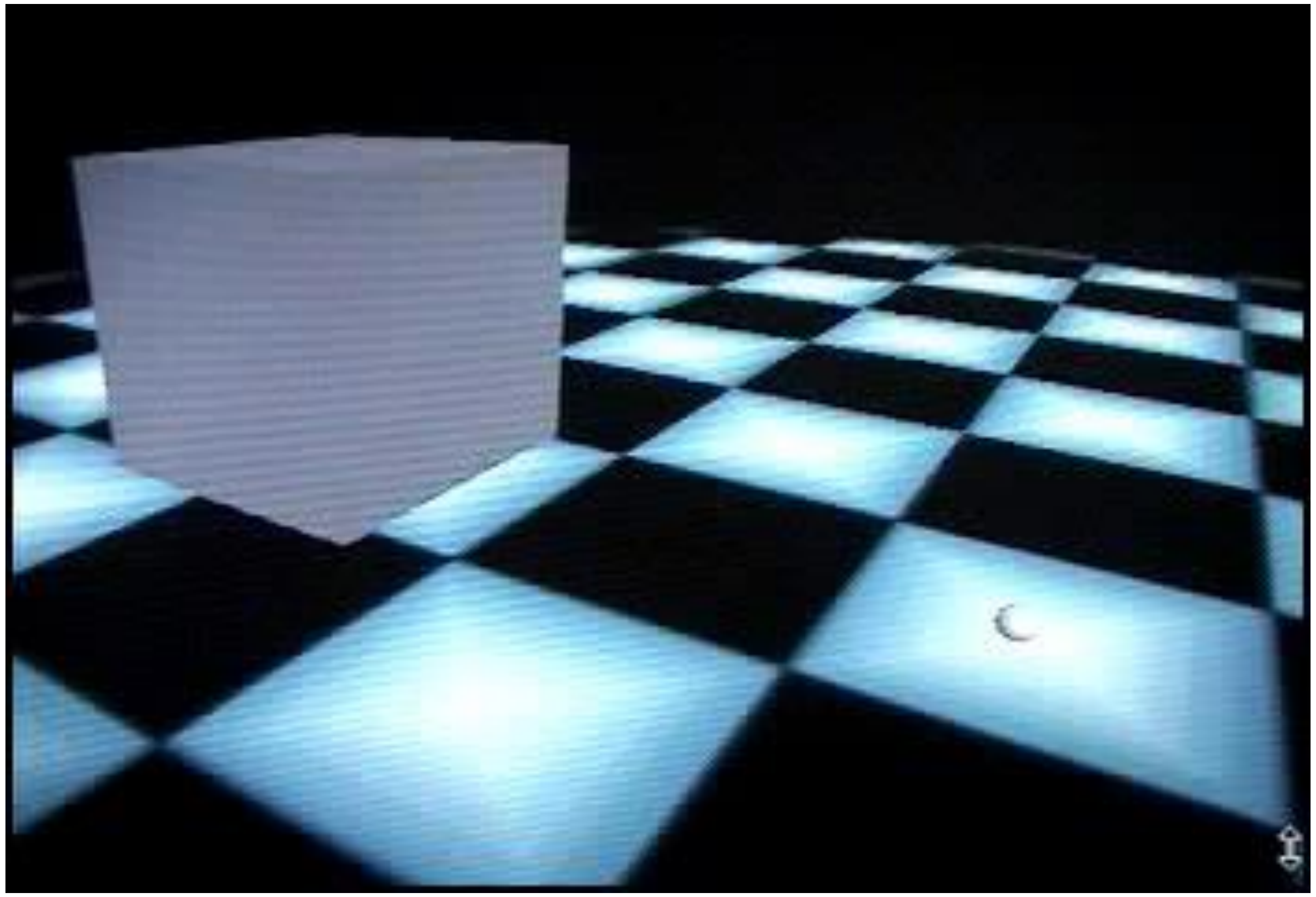
e.g. add/remove a neuron/synapsis, add/remove a cell, …

Stanley, Kenneth O., and Risto Miikkulainen. "A taxonomy for artificial embryogeny." *Artificial Life* 9.2 (2003): 93-130.

Other indirect encodings – Gene Regulatory Networks



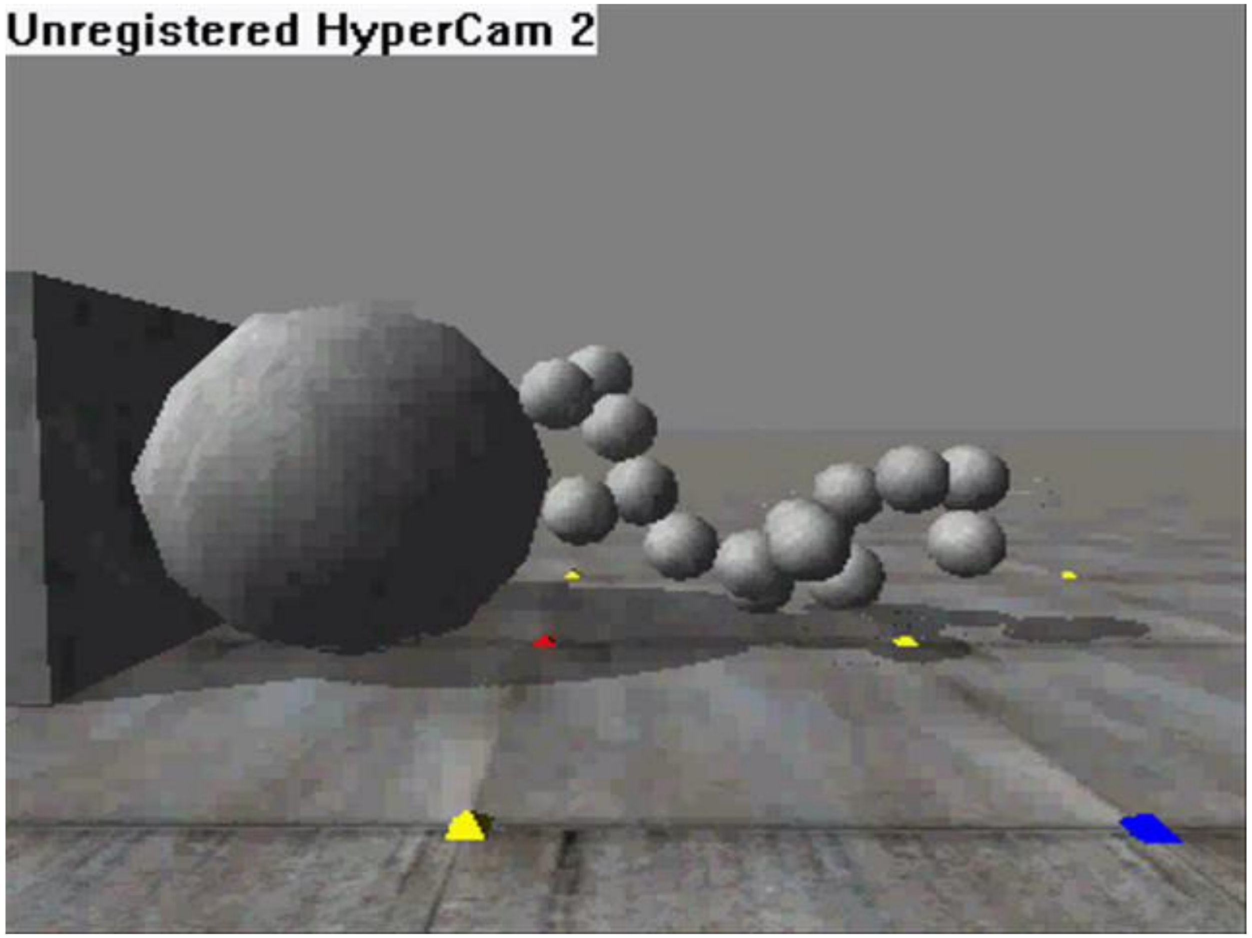
Evolution of Locomotion



Example of **exaptation** (peristaltic locomotion → manipulation) and evolution of size

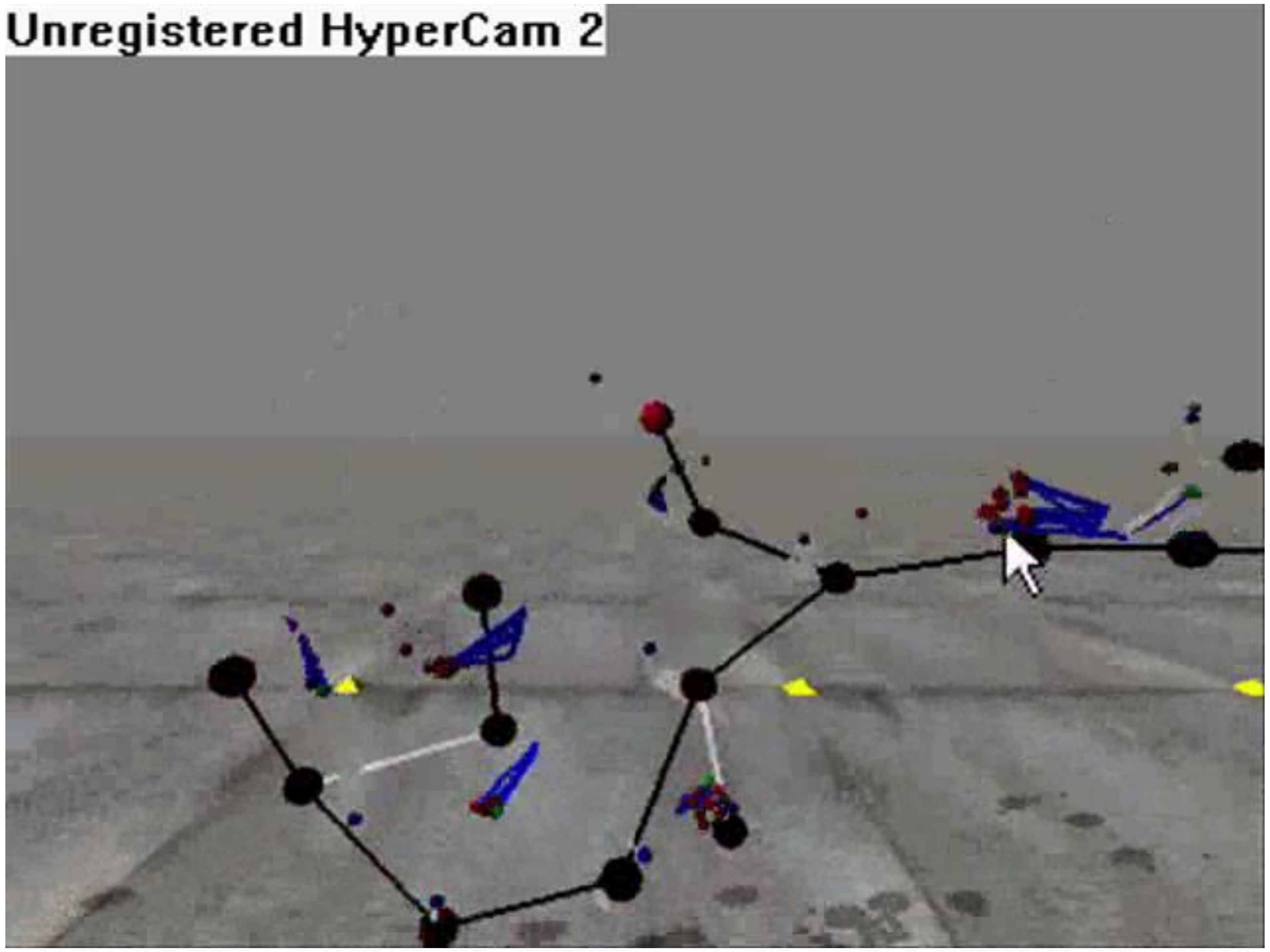
Bongard, Josh C., and Rolf Pfeifer. "Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny." Proceedings of the Genetic and Evolutionary Computation Conference. Vol. 829836. 2001.

Other indirect encodings – Gene Regulatory Networks



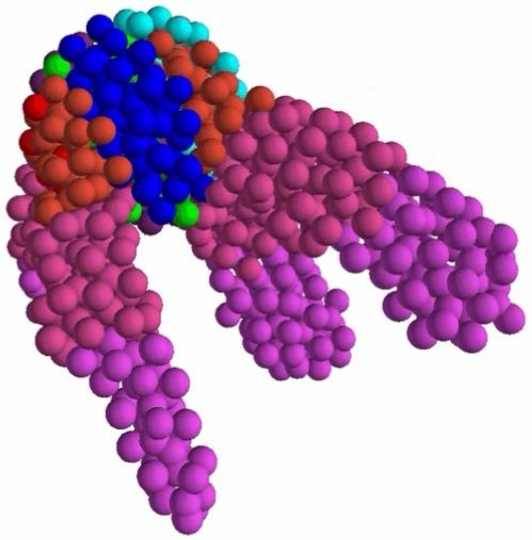
Growing morphology

Growing neural structure



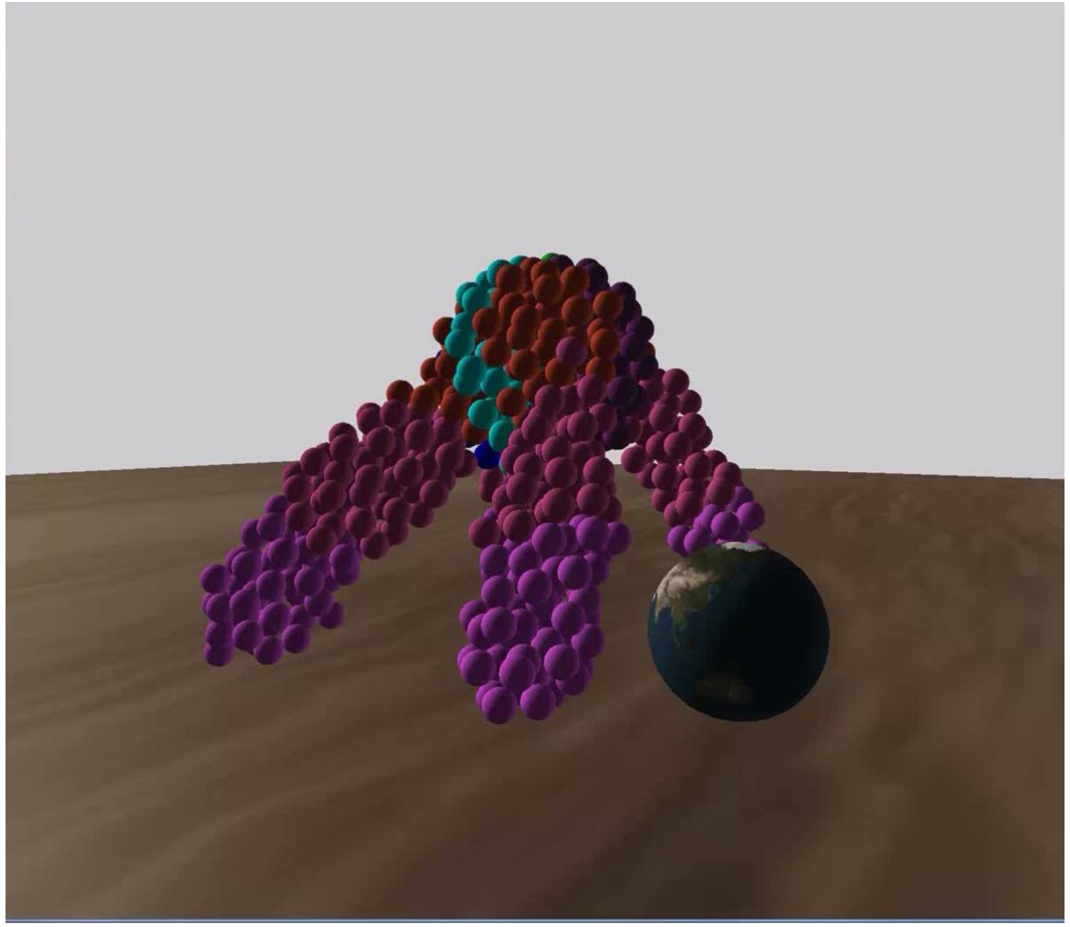
Bongard, Josh C., and Rolf Pfeifer. "Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny." Proceedings of the Genetic and Evolutionary Computation Conference. Vol. 829836. 2001.

Other indirect encodings – Gene Regulatory Networks



GRN are widely adopted in _morphogenetic engineering_ and artificial multicellular development

Programmable, self-organizing (“self-architecturing”) systems from low-level interactions



Doursat, R. & Sánchez, C. (2014) Growing fine-grained multicellular robots. Soft Robotics 1(2): 110-121

Conclusions

- Artificial brains and bodies can be **co-evolved using similar techniques**
- Different developmental encodings replicate different details of biological development (**different levels of abstraction are possible**)

Conclusions

- **Developmental encodings can empower artificial evolution**, bringing some aspects of **biological development** into play
- **Increased biological plausibility:** Morphological and neurological complexity are not produced by evolution alone in nature, **development plays a big role**
- Developmental processes are themselves a product of evolution: **evolution of development (*evo-devo)***
- Subtle **interactions exist between evolution and development** (active research area)

Conclusions

Why bother replicating developmental processes too?

- More comprehensive tools for Artificial Life and Computational Biology
- **Implications for evolutionary robotics:**
  - Developmental processes can help evolution

    → Can increase evolvability

    → Can increase scalability (**information is reused**)

    → More effective automated design of robot morphologies and controllers

The ***reality gap*** **problem**

The *reality gap* / transfer problem

In order to avoid technological limitations, evolutionary robotics methodologies are <u>often applied in simulation</u>

Evolutionary simulations allow to study many interesting phenomena, having <u>full access</u> to the evolving systems and <u>full control</u> over the environment in which they evolve

→ <u>Great tool for Artificial Life, Computational Biology, etc.</u>

The *reality gap* / transfer problem

However, one of the main goals of evolutionary robotics is to **<u>evolve real robots for real tasks</u>**.

<u>Two options:</u>

1. **Apply evolutionary techniques in the real world, using real robots**

→ Pros? Cons?

The *reality gap* / transfer problem

However, one of the main goals of evolutionary robotics is to **evolve real robots for real tasks**.

Two options:

1. **Apply evolutionary techniques in the real world, using real robots:** Possible, you "get physics for free", but some limitations (time → few evaluations → suboptimal results, hardware limitations and resilience, experimental setup must be designed)

The *reality gap* / transfer problem

However, one of the main goals of evolutionary robotics is to **evolve real robots for real tasks**.

Two options:

1. **Apply evolutionary techniques in the real world, using real robots:** Possible, you “get physics for free”, but some limitations (time → few evaluations → suboptimal results, hardware limitations and resilience, experimental setup must be designed)
2. **Apply evolutionary techniques in simulated worlds, then transfer the final products into the real world**

→ Pros? Cons?

The *reality gap* / transfer problem

However, one of the main goals of evolutionary robotics is to **evolve real robots for real tasks**.

Two options:

1. **Apply evolutionary techniques in the real world, using real robots:** Possible, you "get physics for free", but some limitations (time → few evaluations → suboptimal results, hardware limitations and resilience, experimental setup must be designed)

2. **Apply evolutionary techniques in simulated worlds, then transfer the final products into the real world:** Need to simulate the world, but can bypass technological limits, computational power → far more fitness evaluations are possible → more optimized designs

The *reality gap* / transfer problem

→ **Unfortunately, achieving a successful transfer of evolved solutions from simulation to reality has proven to be generally difficult: <u>reality gap / transfer problem</u>**

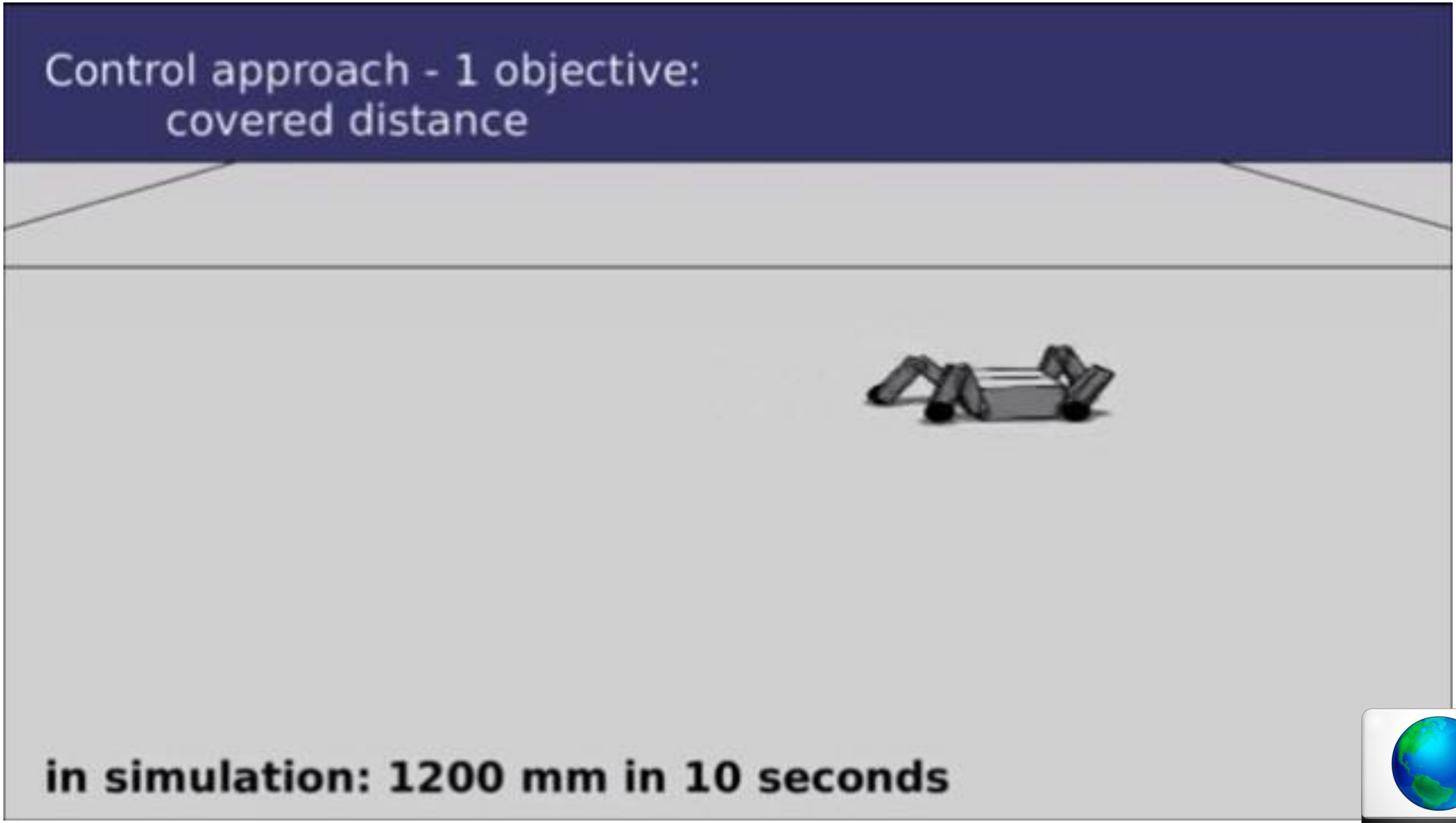
The *reality gap* / transfer problem

Why?

- The smallest discrepancy between the simulated environment and the real world can result in an unsuccessful transfer (e.g. approximated physics)
- …and no model is as rich as the physical reality: *"There is no better model of the world than the world itself"*, R. Brooks
- Also, evolutionary algorithms will especially try to exploit these discrepancies whenever this turns out to be beneficial in order to maximize fitness (*perverse instantiation*)

→ **Serious problem for evolutionary robotics**

An example of reality gap – [Koos et al, 2010]



The radical envelope-of-noise hypothesis

- **Jakobi (1997)**
  - Observation:
    - Evolution will create solutions that exploit details of the simulation
    - If those details do not exist in reality, the controller will fail to cross the reality gap
  - Hypotheses:
    - Properly adding noise to the simulation can prevent evolution from relying on those details → makes details unreliable

- Overall, the more complex the simulation, the more difficult to decide which aspects to "nosify" and how → Always try to create minimal simulations

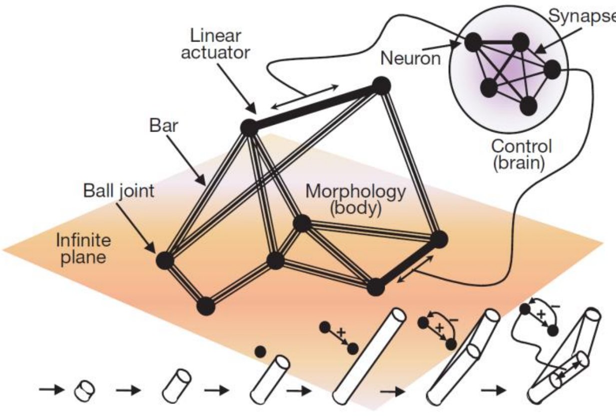
Crossing the gap – [Lipson and Pollack 2000]

First time the reality gap was crossed

Robot composed by a set of building blocks (bars, motors, neurons)

Brain-body co-evolution
+
3D printing

Noise added to the simulation



Crossing the gap – [Bongard and Lipson, 2006]

- Robot first **evolves a self-model** (simulator) that matches proprioceptive information from a **limited set of physical exploratory actions**

- The **self-model is then used to evolve a behavior** (gait)

- The **robot can detect a damage** by noticing that **predicted and actual sensory information do not match anymore**

- **Can evolve a new self-model, and then a new behavior**

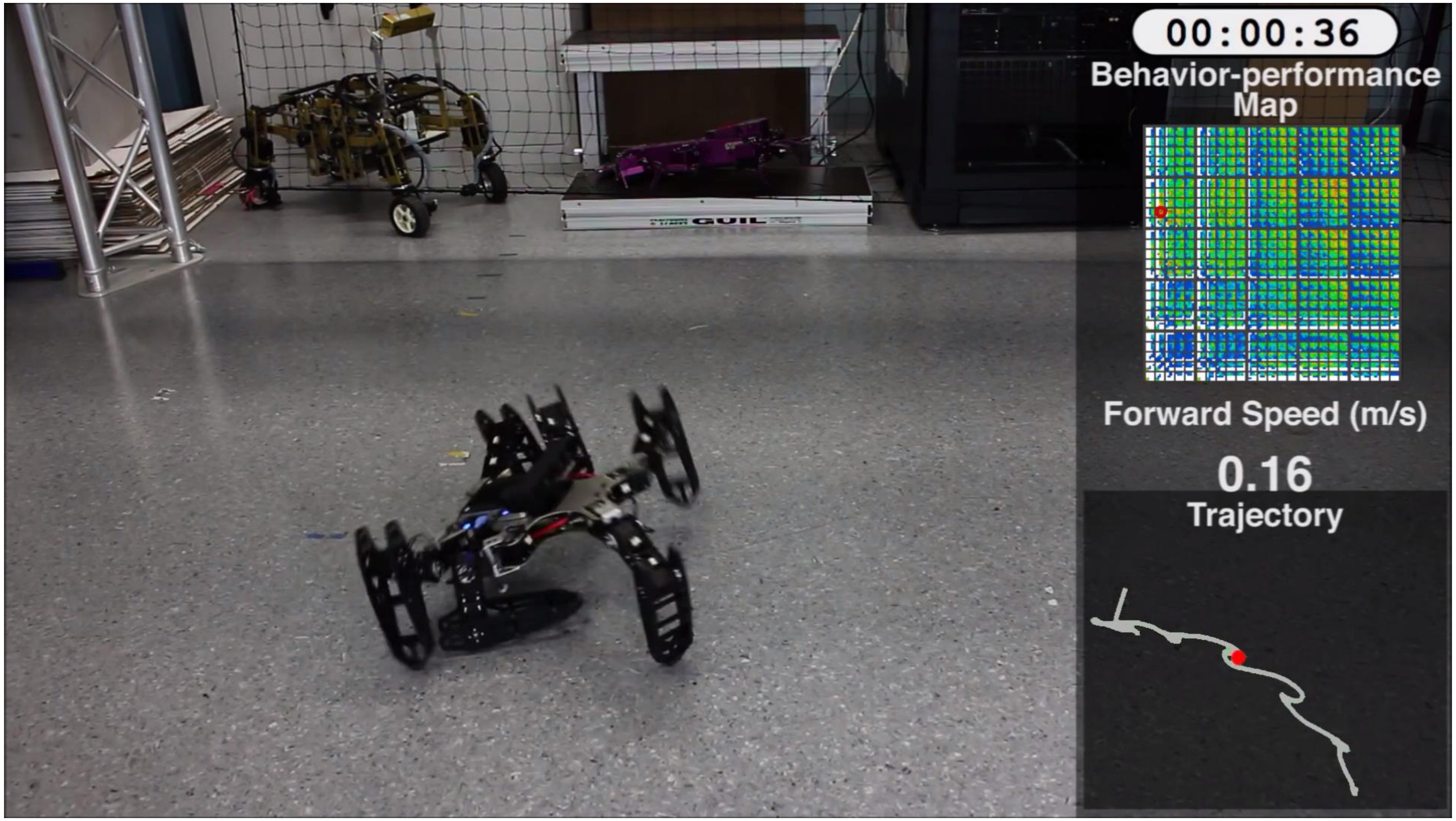
Crossing the gap – [Cully et al, 2015]

- Robot uses a fixed self-model to evolve (in simulation) a **map of different locomotion strategies and associated fitness** (*MAP-elites* algorithm)

- **Initially attributes low confidence to these behaviors** (only tested in simulation!)

- Uses an intelligent algorithm to **test a few behaviors in the real world and update its confidence level on the whole map**

- Can use this procedure to **find behaviors that transfer well**, as well as to **find behaviors that work in the face of a damage**



URL

MAP-elites

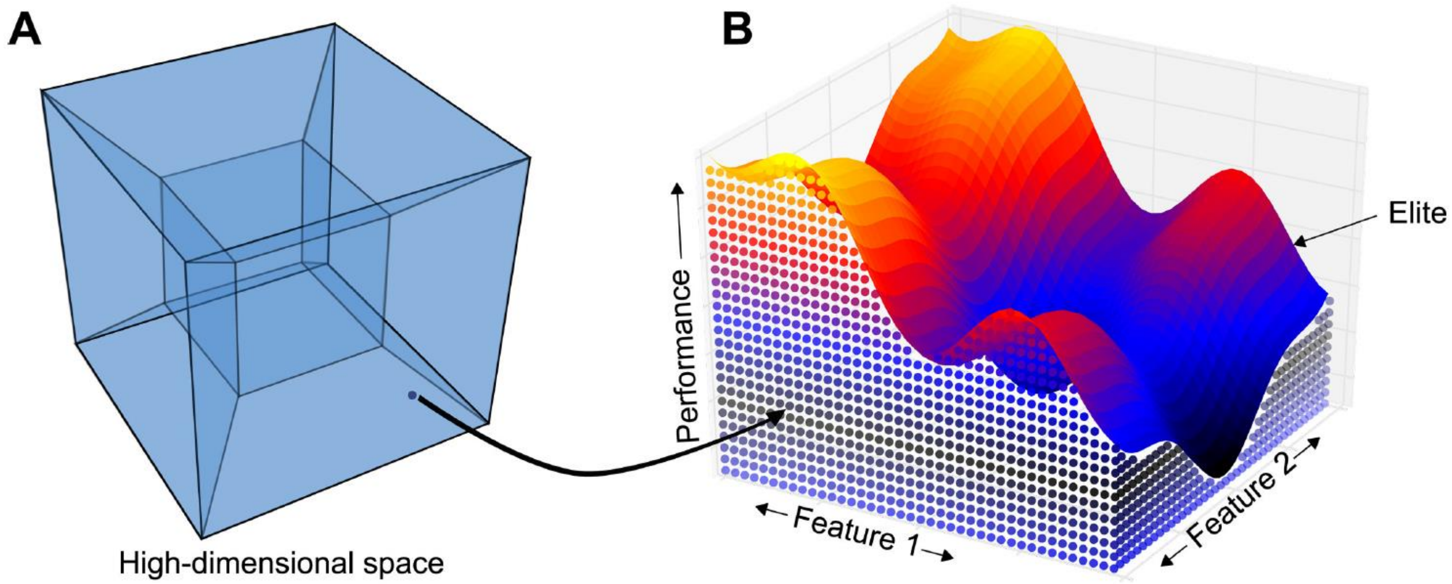


Fig. 1. The MAP-Elites algorithm searches in a high-dimensional space to find the highest-performing solution at each point in a low-dimensional feature space, where the user gets to choose dimensions of variation of interest that define the low dimensional space. We call this type of algorithm an “illumination algorithm”, because it illuminates the fitness potential of each area of the feature space, including tradeoffs between performance and the features of interest. For example, MAP-Elites could search in the space of all possible robot designs (a very high dimensional space) to find the fastest robot (a performance criterion) for each combination of height and weight.

Optimization vs “Illumination”:

Traditional optimization algorithm would try to find a single best performing solution (e.g. a specific gait, say, a tripod gait)

MAP-elites try instead to construct instead a whole *phenotype-fitness map* (!= finess landscape), which contains many behaviorally different high-performance solutions

Behavioral difference is quantified in a features space, which is task-dependent (e.g. for locomotion, % of time each foot is in contact with the ground)

Mouret, Clune, "Illuminating search spaces by mapping elites"

MAP-elites

procedure MAP-ELITES ALGORITHM (SIMPLE, DEFAULT VERSION)

$(\mathcal{P} \leftarrow \emptyset, \mathcal{X} \leftarrow \emptyset)$ ▷ *Create an empty, N-dimensional map of elites: {solutions $\mathcal{X}$ and their performances $\mathcal{P}$}*

for iter $= 1 \rightarrow I$ **do** ▷ *Repeat for I iterations.*

if iter $< G$ **then** ▷ *Initialize by generating G random solutions*

$\mathbf{x}' \leftarrow$ random_solution()

else ▷ *All subsequent solutions are generated from elites in the map*

$\mathbf{x} \leftarrow$ random_selection($\mathcal{X}$) ▷ *Randomly select an elite x from the map $\mathcal{X}$*

$\mathbf{x}' \leftarrow$ random_variation($\mathbf{x}$) ▷ *Create x', a randomly modified copy of x (via mutation and/or crossover)*

$\mathbf{b}' \leftarrow$feature_descriptor($\mathbf{x}'$) ▷ *Simulate the candidate solution x' and record its feature descriptor $\mathbf{b}'$*

$p' \leftarrow$performance($\mathbf{x}'$) ▷ *Record the performance p' of x'*

if $\mathcal{P}(\mathbf{b}') = \emptyset$ or $\mathcal{P}(\mathbf{b}') < p'$ **then** ▷ *If the appropriate cell is empty or its occupants's performance is $\leq p'$, then*

$\mathcal{P}(\mathbf{b}') \leftarrow p'$ ▷ *store the performance of x' in the map of elites according to its feature descriptor $\mathbf{b}'$*

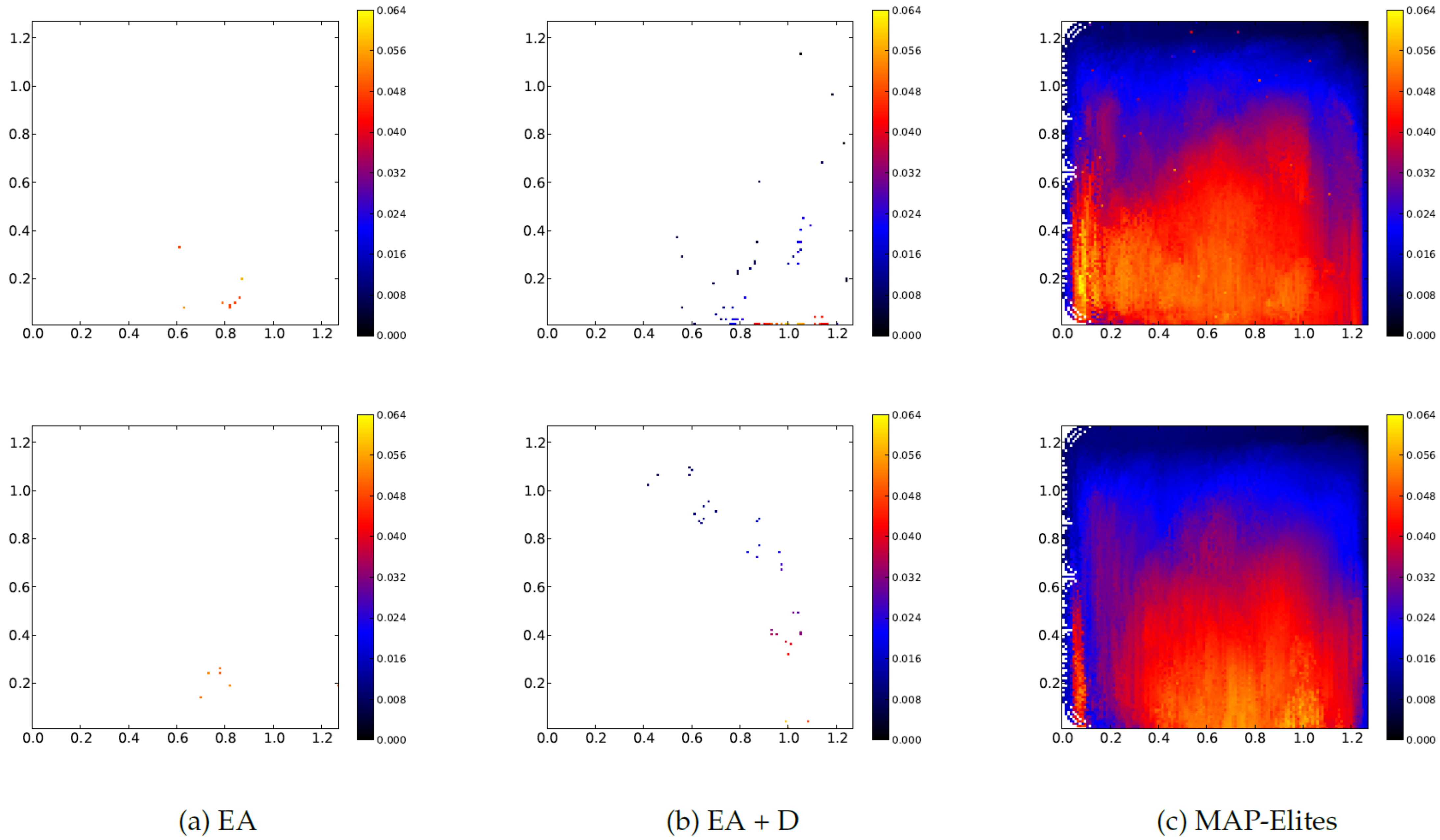
$\mathcal{X}(\mathbf{b}') \leftarrow \mathbf{x}'$ ▷ *store the solution x' in the map of elites according to its feature descriptor $\mathbf{b}'$*

return feature-performance map ($\mathcal{P}$ and $\mathcal{X}$)

Fig. 2. A pseudocode description of the simple, default version of MAP-Elites.

Mouret, Clune, "Illuminating search spaces by mapping elites"

MAP-elites



Comparison of phenotype-fitness map: optimization algorithms (first two columns) vs MAP-elites

Mouret, Clune, "Illuminating search spaces by mapping elites"

Beyond pure fitness: novelty search

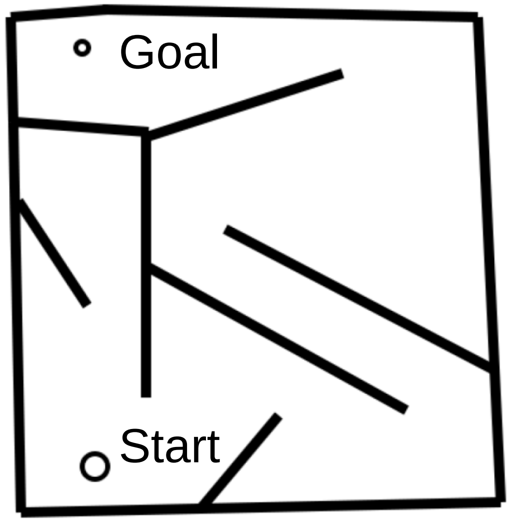
Beyond pure fitness

- MAP-elites is just one example of a recent trend in evolutionary computation
- Different approaches in which the role of pure performance-based fitness is revised
- A particularly radical one: **Novelty Search** (Lehman, Stanley, 2011)
  - Performances are completely ignored during search
  - Instead of directly rewarding better solutions, rewards novel solutions
  - Novelty is computed relative to an archive of observed behaviors (*novelty archive*) as well as w.r.t. the current population
  - Novelty score (actual fitness): Euclidean distance in a space of behavioral features (like those defined in MAP-elites)

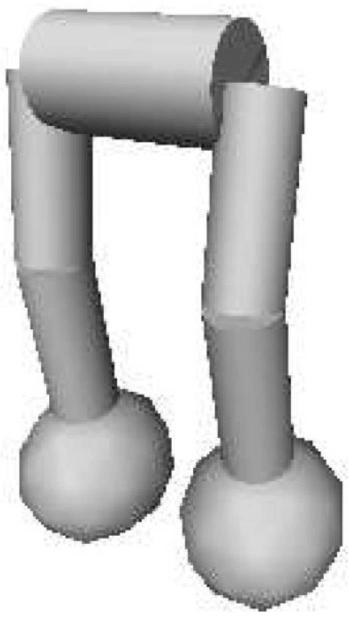
Lehman, Stanley, "Abandoning Objectives: Evolution through the Search for Novelty Alone"

Recent paradigm shift in evolutionary computation

- It has been shown that by doing so, it is actually possible to find **better** (even in a performance-oriented sense) solutions



- How so? Intuitions:
  - Fitness landscapes can be **deceptive**: greedily following fitness gradients can lead to bad local optima (e.g. deceptive maze)
  - Different, perhaps initially bad behaviors (e.g. falling) can be stepping stones for future effective ones (e.g. running is controlled falling: exploring different ways to fall can lead to discovering running instead of walking)

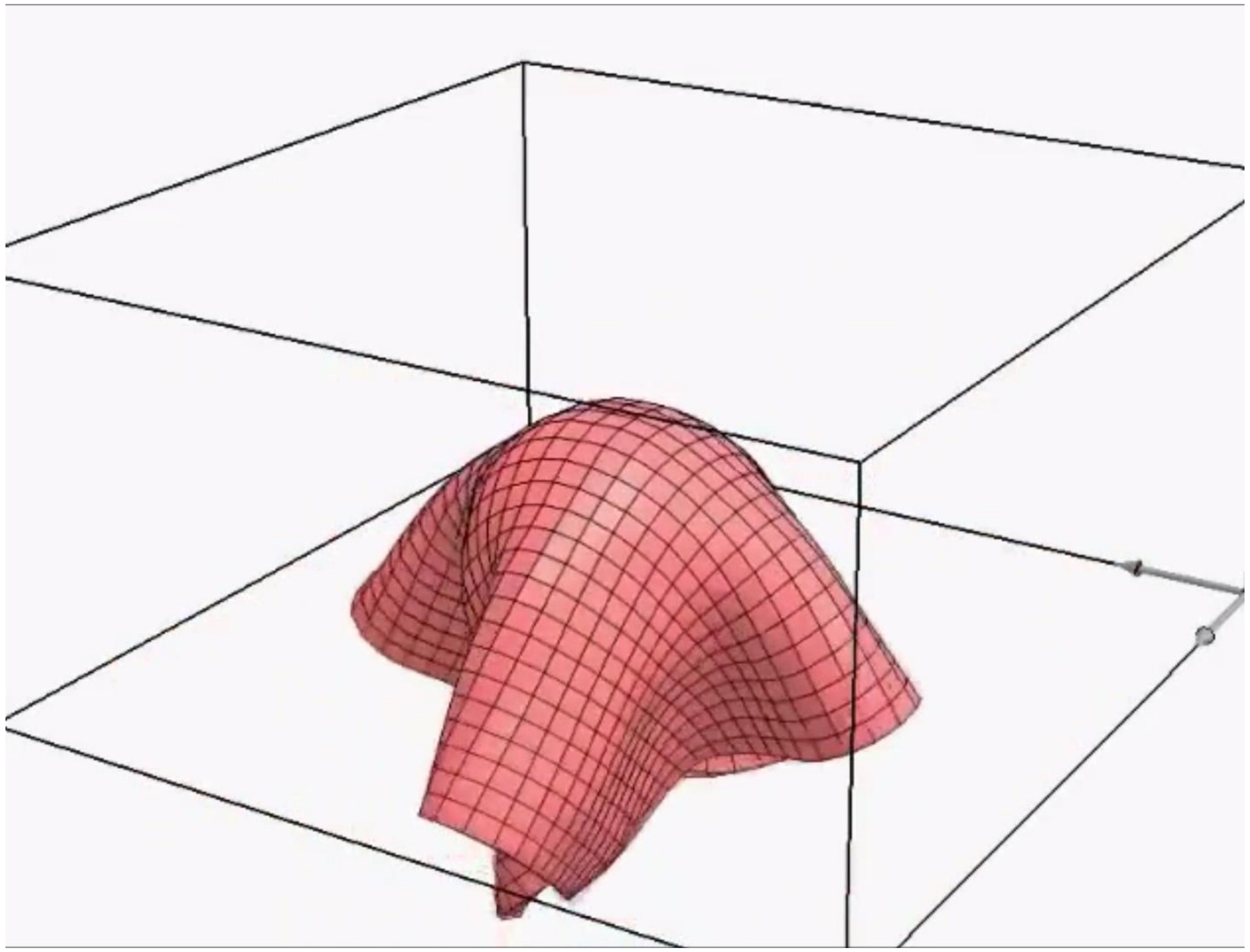


Lehman, Stanley, "Abandoning Objectives: Evolution through the Search for Novelty Alone"

Appendix: VoxCad simulator

VoxCad (Voxel Cad)

- Open-source voxel modeling and analyzing software (FEM)
- Originally developed by John Hiller (Cornell University)
- VoxCad: GUI
- Voxelyze: underlying 3D dynamics physics engine
  - Supports multiple-materials (soft-stiff) large deformations, collision detection, volumetric actuation, etc

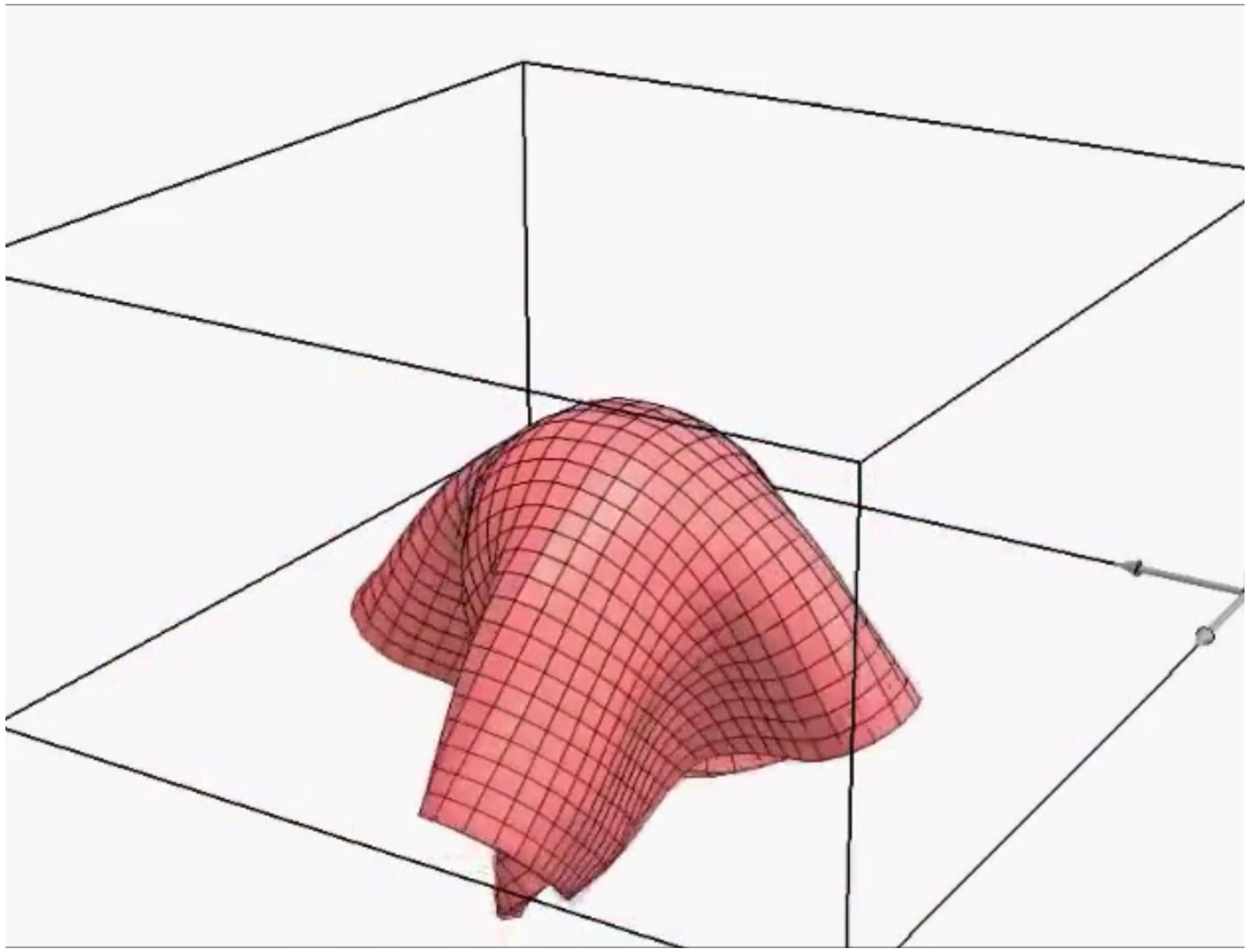


https://sites.google.com/site/voxcadproject/

VoxCad (Voxel Cad)

- Quickly adopted by researchers in *evolutionary soft robotics*, integrated with CPPN-NEAT
- Has been extended during the years with several features
  - Electrical actuation
  - Simplified fluid dynamics
  - Developmental processes

(some customized versions are yet to be published and are available within the lab)



https://sites.google.com/site/voxcadproject/

Overview of ongoing and past activities in this area

Ongoing and past activities

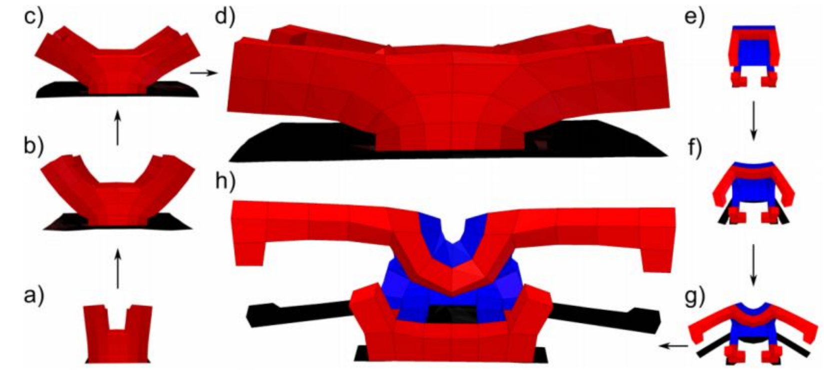
General idea:

To study and implement artificial evolutionary and developmental processes in order to investigate the emergence of adaptive and intelligent behavior in biological and artificial systems

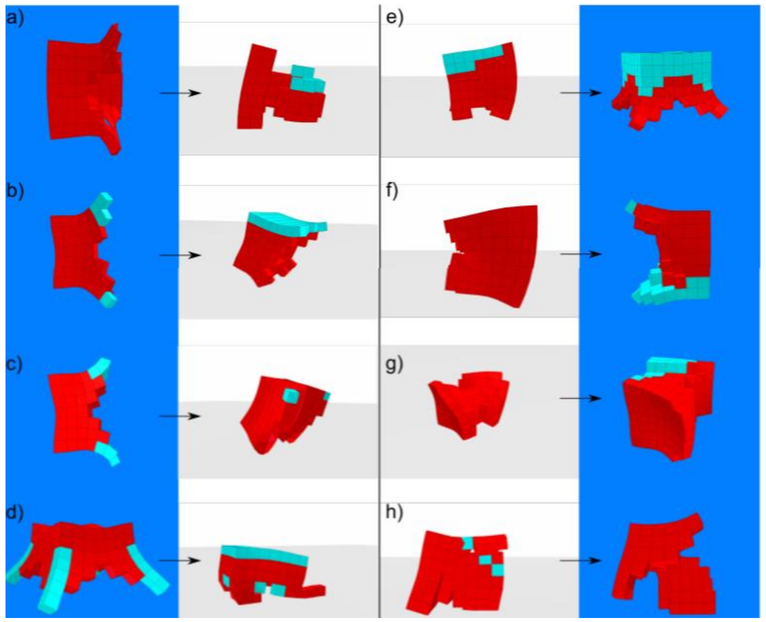
Particular attention to the role and implications of a soft morphology in this process

"Evolutionary Developmental Soft Robotics: Towards Adaptive and Intelligent Soft Machines Following Nature's Approach to Design", F. Corucci, Soft Robotics: Trends, Applications and Challenges, 111-116

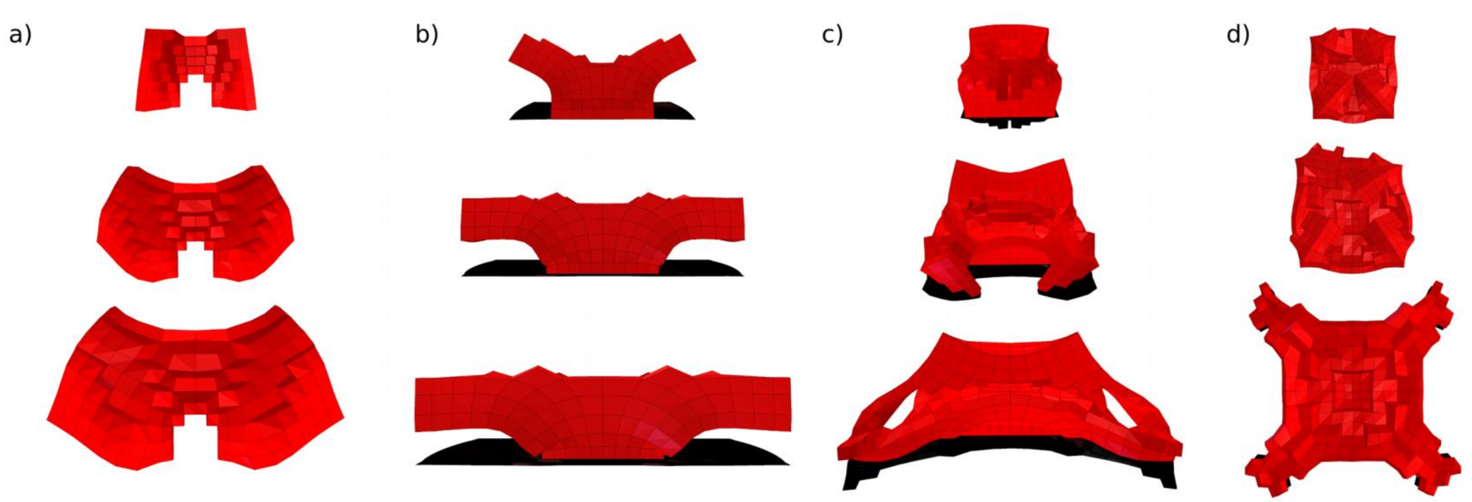
Ongoing and past activities

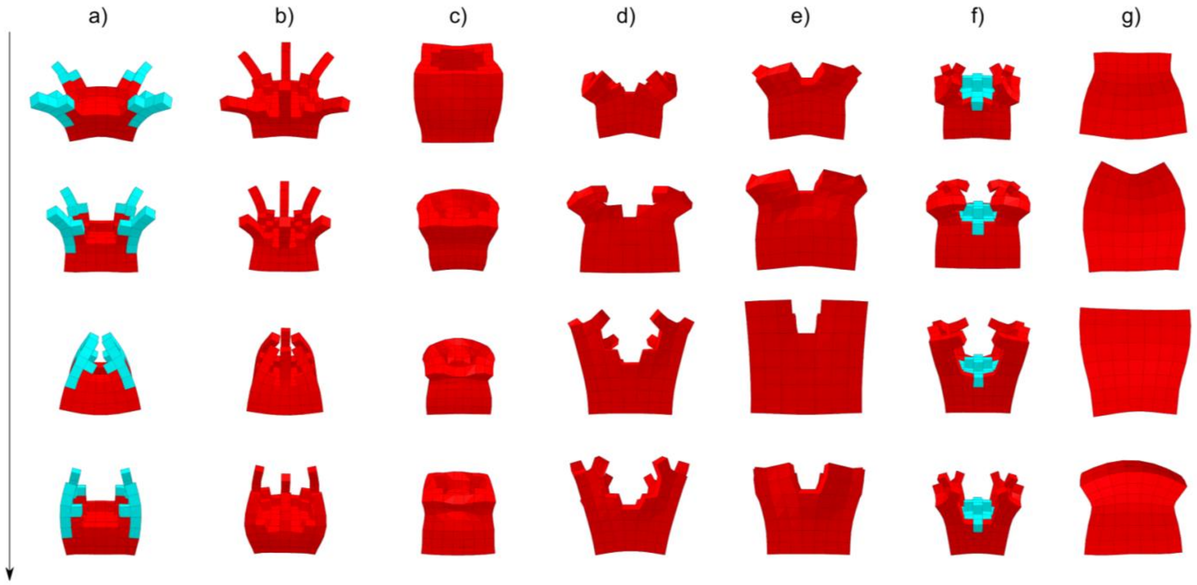


- Brain-body co-evolution of generic soft-bodied creatures/robots in different environments



- **Morphological developmental plasticity in soft bodied creatures**
- **Interaction between evolutionary and developmental processes (evo-devo)**

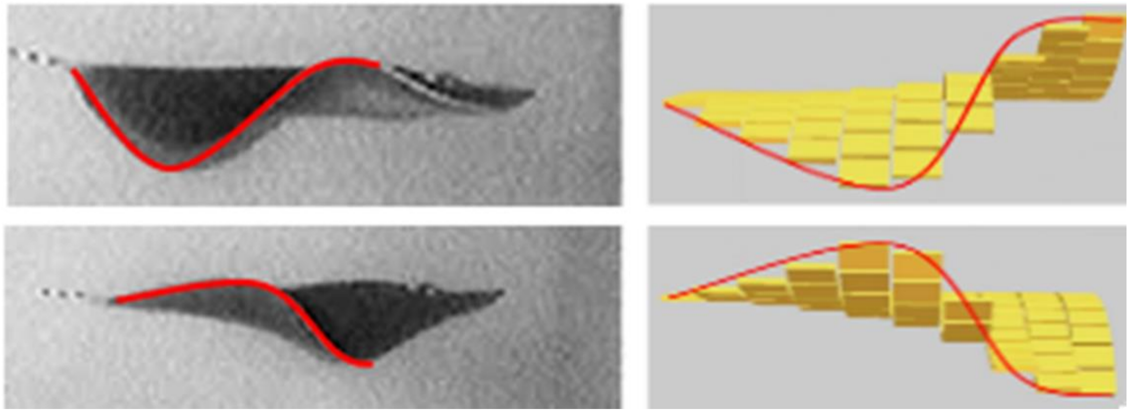


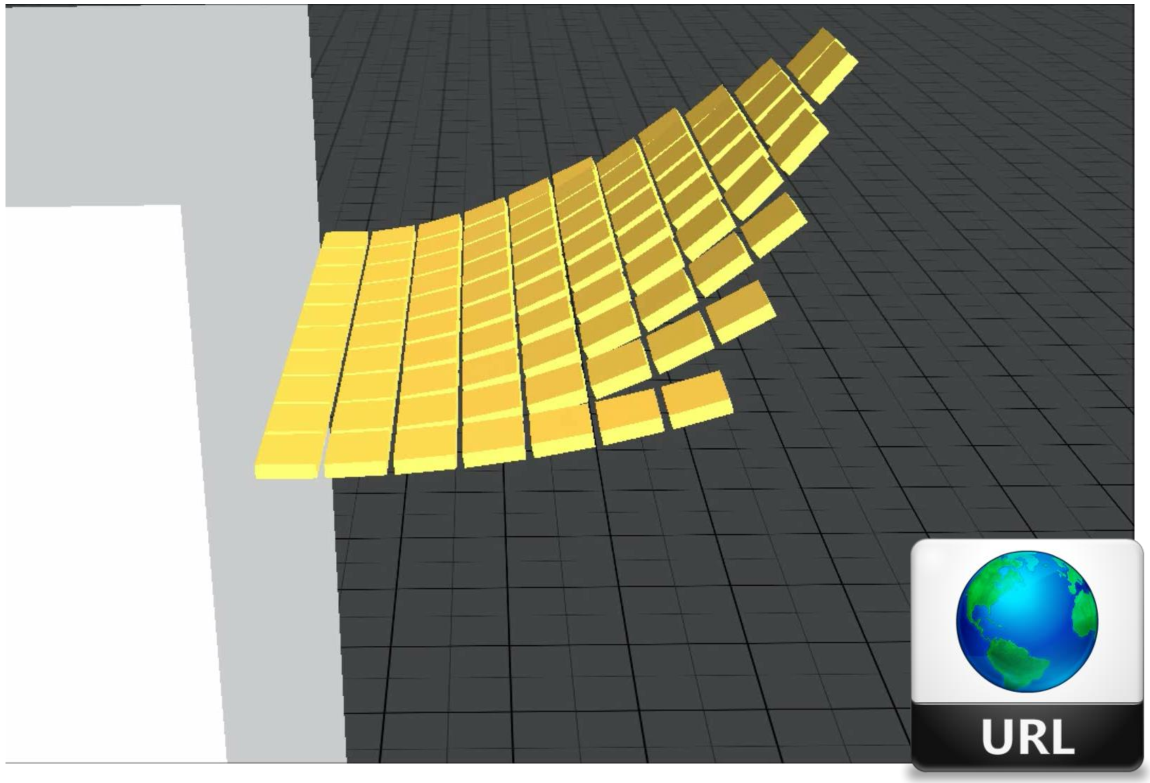


Ongoing and past activities

Using artificial evolution to study properties of specific animals

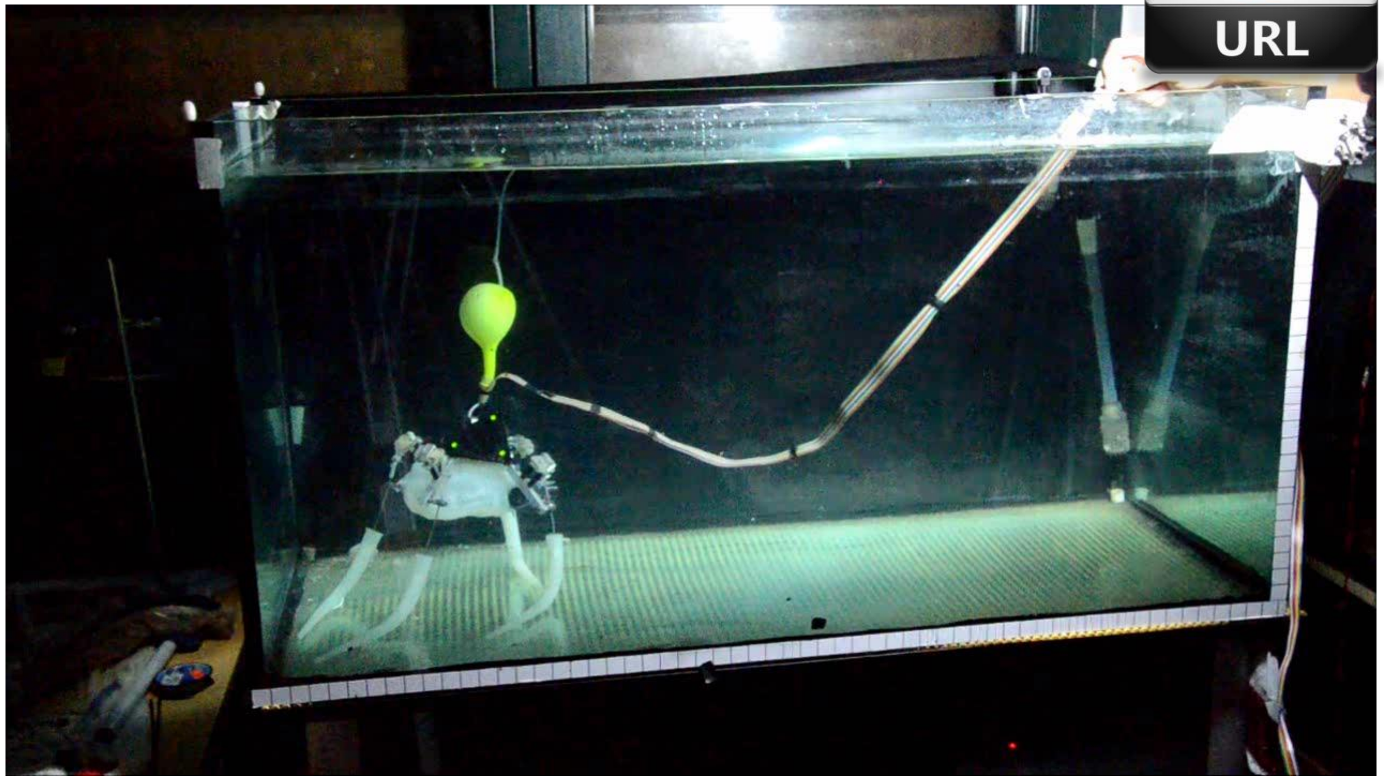
Example: Artificial Evolution and adaptation to different environments of a manta-like fin

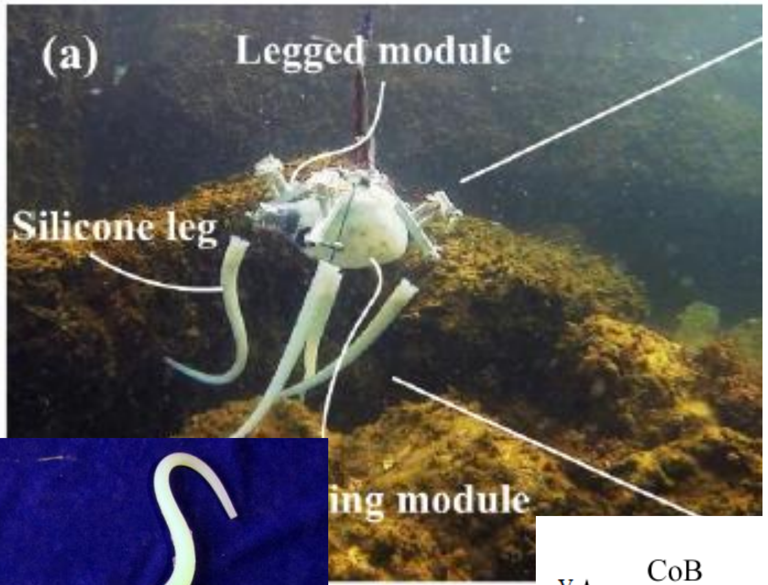


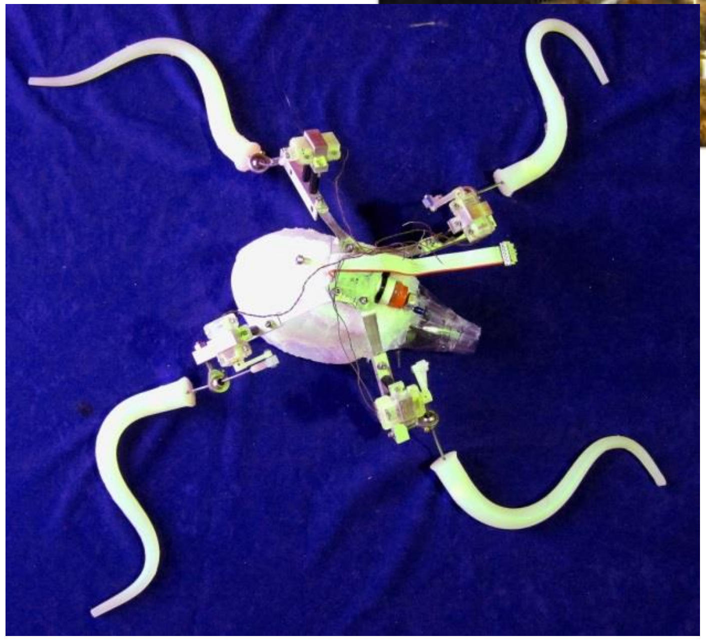


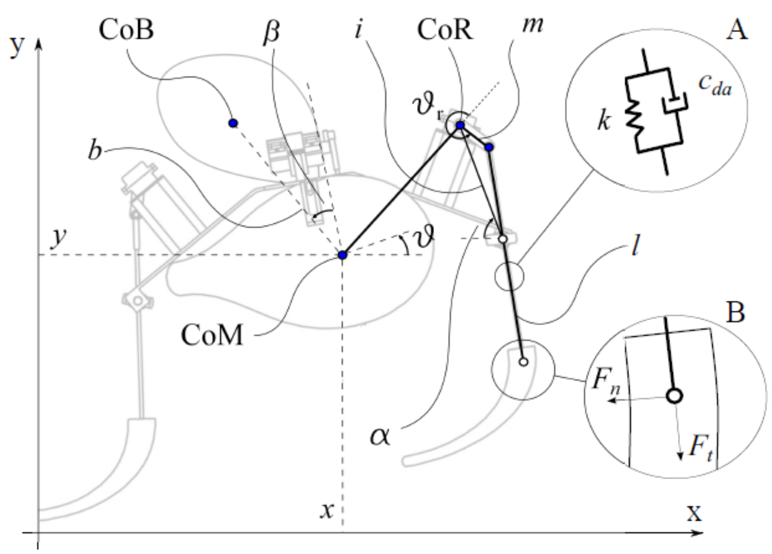
Ongoing and past activities

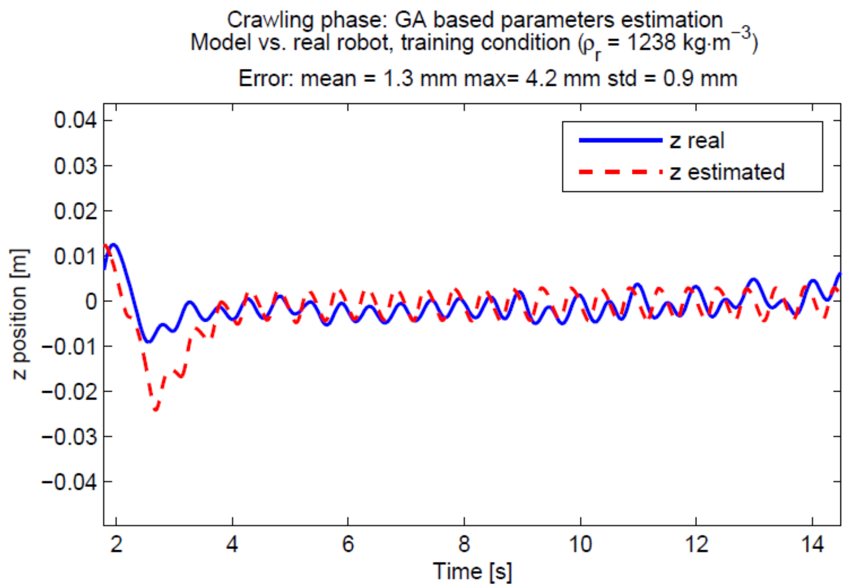
Applying <u>evolutionary design techniques</u> to improve soft bio-inspired robots







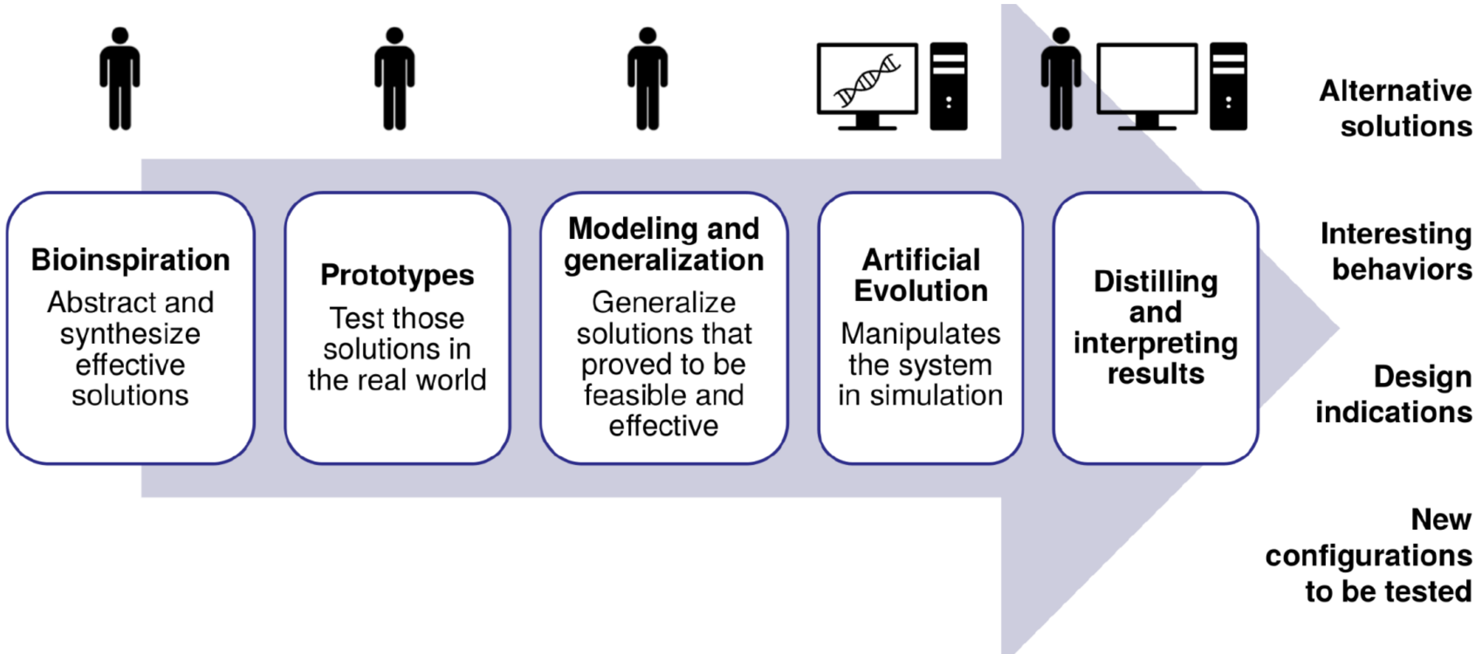


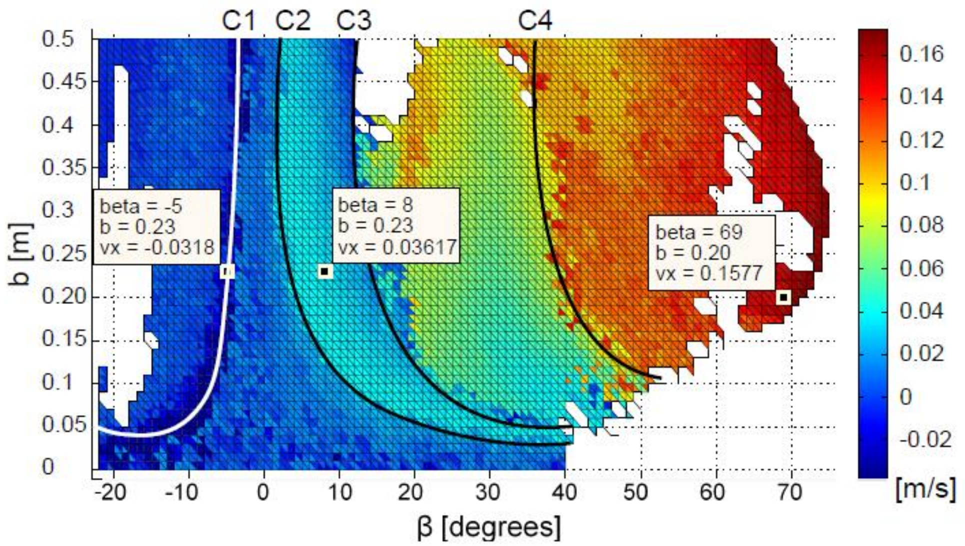


Ongoing and past activities

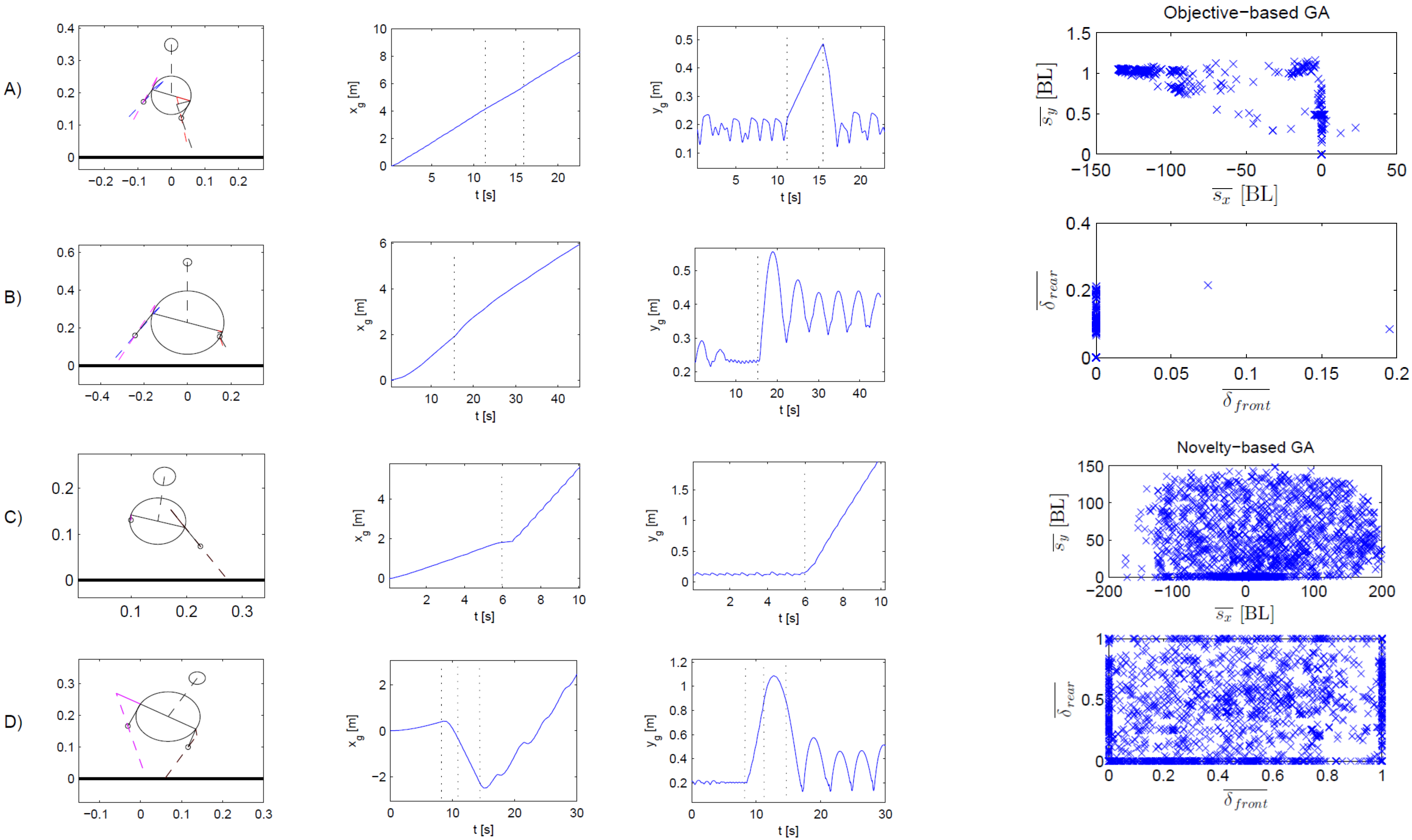
- Human-machine collaborative evolutionary design
  - Reconciliating bio-inspired design, human and artificial creativity (novelty)

- Investigating the feasibility and potential benefits of shape-changing robots (morphing/morphosis), i.e. robots that can reconfigure their body in order to adapt to different tasks/environmental conditions



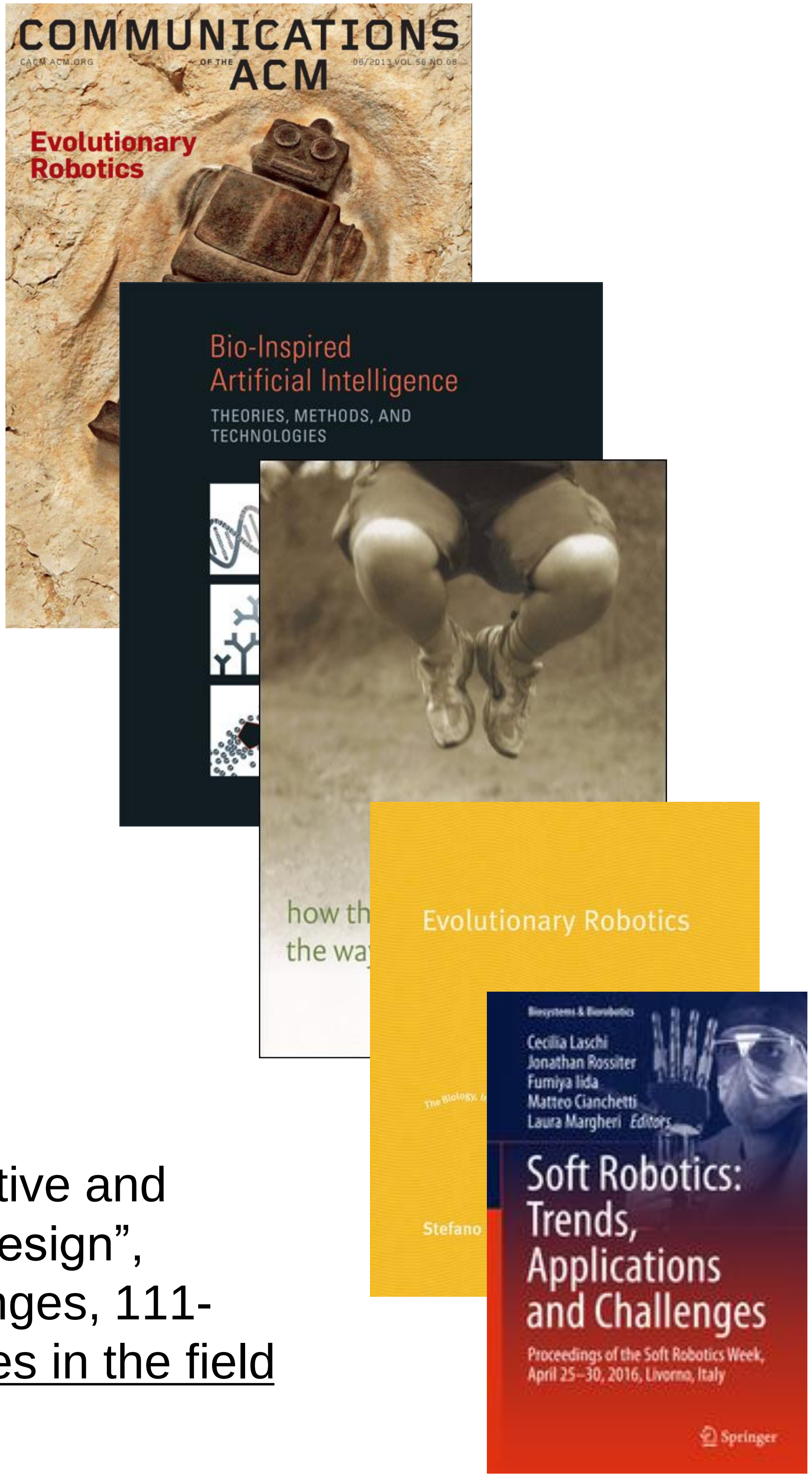


Ongoing and past activities



Suggested readings

1. **Josh C. Bongard. 2013. *“Evolutionary robotics*”. *Commun. ACM* 56, 8 (August 2013), 74-83**
   → Survey paper on Evolutionary Robotics
2. “Bio-Inspired Artificial Intelligence: Theories, Methods and Technology”
   Dario Floreano and Claudio Mattiussi
   → *Chapter 1*
3. “How the body shapes the way we think”
   Rolf Pfeifer and Josh Bongard
   → *Chapter 6*
4. “Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines”
   Stefano Nolfi, Dario Floreano
5. “Evolutionary Developmental Soft Robotics: Towards Adaptive and Intelligent Soft Machines Following Nature’s Approach to Design”,
   F. Corucci, Soft Robotics: Trends, Applications and Challenges, 111-116 → Book chapter, short overview of some of our activities in the field



For doubts, additional material,

projects and theses ideas:

f.corucci @ sssup.it

http://sssa.bioroboticsinstitute.it/user/1507