Introduction to Machine Learning and Evolutionary Robotics

2nd part – 3 CFU

Eric Medvet

A.Y. 2023/2024

Contents

- Evolutionary computation
- Its application to robotics: evolutionary robotics

Section 1

Evolutionary computation

Goal: optimization

- User has a broad idea of the universe of possible
- User has a way for measuring the quality
- "Computer: search the possible for a good solution!"

How?

Goal: optimization

- User has a broad idea of the universe of possible
- User has a way for measuring the quality
- "Computer: search the possible for a good solution!"

How?

A significant case:

- universe of possible: "physics"
- quality: ability of being alive
- computer: nature
- how? natural evolution

(Who is the user?)

Evolution

A general and basic scheme:

- a population of individuals compete for limited resources
- the population is dynamic: individuals die and are born
- fittest individual survive and reproduce more than the others
- offspring inherit some characters from parents (they are similar but not identical)

Evolution

A general and basic scheme:

- a population of individuals compete for limited resources
- the population is dynamic: individuals die and are born
- fittest individual survive and reproduce more than the others
- offspring inherit some characters from parents (they are similar but not identical)

On/by/for computers? Evolutionary computation (EC)

EC: a bit of history

1930s	first ideas
1960s	ideas development using first computers
1970s	exploration
1980s	exploitation
1990s	unification
2000s+	mature expansion

Communities

At least three communities:

- biologists: simulate/understand real evolution
- computer scientists/engineers: build interesting artifacts
- artificial-life researchers: build/study artificial worlds

Result:

- some duplications
- different vocabularies
- strong habits

Kenneth A De Jong. *Evolutionary computation: a unified approach*. MIT press, 2006

What can be taught/learned?

Here:

- general scheme
- terminology
- some significant variants
- general usage guidelines

Not here:

- (variant) details
- detailed motivation (“theory”)
- specific tools

General scheme

- a population of individuals compete for limited resources
- the population is dynamic: individuals die and are born
- fittest individual survive and reproduce more than the others
- offspring inherit some characters from parents (they are similar but not identical)

Some questions:

- what is an individual?
- what is a population? what are resources?
- how individuals compete?
- how fitness is measured?
- how do individual reproduce?

Individual

A candidate solution for the considered problem:

- a program in a given programming language
- a set of numerical parameters
- a picture
- ...

Internally represented as:

- itself (program, set, picture, ...)
- some well defined data structure:
  - a fixed/variable-length string of bits
  - an abstract syntax tree
  - ...

Individual

A candidate solution for the considered problem: (**phenotype**)

- a program in a given programming language
- a set of numerical parameters
- a picture
- ...

Internally represented as: (**genotype**)

- itself (program, set, picture, ...)
- some well defined data structure:
  - a fixed/variable-length string of bits
  - an abstract syntax tree
  - ...

Individual: why genotype/phenotype?

- To resemble nature
- To ease manipulation
  - how two programs should reproduce?
  - how two images should reproduce?
- To allow reuse, hence enabling actual usage of EC
  - someone found a good way of making bits strings reproduce
  - user "just" need to decide how to transform (**genotype-phenotype mapping**) a bits string to his/her solution form (e.g., numerical parameters)

Population and competition for resources

Mainstream:

- a population is a set of individuals with a fixed (max) size
- "limited resources" is a place in the population

The population is dynamic:

- when a new individual is born, some individual must leave the population (die): which one?

Population dynamics

How/when individuals are replaced? (**generational model** or replacement strategy)

Underlying (and common) assumptions:

- individuals life is instantaneous
  - given the genotype, the phenotype (if any) and the fitness are immediately known
- time flowing is determined by births (and deaths)

Generational model: general scheme

Parameters:

- a population of m parents
- a population of n offspring (built from parents; how? later)
- a Boolean flag (overlapping vs. non-overlapping)

(Recall: population size is fixed)

Overlapping generational model

At each time tick:

1. build n offspring from the m parents
2. obtain an $n + m$ population by merging parens and offspring
3. select m individuals to survive

Non-overlapping generational model

At each time tick:

1. build n offspring from the m parents (assume $n \geq m$)
2. select m individuals to survive among the n offspring

All parents die!

Common cases

- $n = m$, overlapping
- $n = m$, non-overlapping
- $n = 0.8m$, overlapping
- $n = 1$, overlapping (steady state)

Common cases

- $n = m$, overlapping
- $n = m$, non-overlapping
- $n = 0.8m$, overlapping
- $n = 1$, overlapping (steady state)

Problem:

- different degrees of dynamicity in the single time tick
  - makes different variants comparison difficult

Solution:

- measure time flowing as number of births referred to population size m
- a **generation** occurs each m births

Selection criteria

How to

- select individuals to survive?
- select parents to reproduce?

Many options:

- uniform (neutral) selection
- fitness-proportional selection
- rank-proportional selection
- truncation selection
- tournament selection
- ...

Fitness/rank-proportional

Fitness-proportional:

1. given the numerical fitness of each individual
2. randomly pick one individual with probability proportional to the fitness (the better, the larger probability)

Rank-proportional:

1. given the rank of each individual in a fitness-based ranking
2. randomly pick one individual with probability proportional to the rank (the better, the larger probability)

(Can be applied to a non-numerical fitness, in principle)

Uniform and truncation

Uniform:

1. pick randomly an individual (with uniform probability)

Truncation:

1. pick the best individual (**elitism**)

(Deterministic)

Tournament selection

Given a parameter n_{size} (size of the tournament):

1. randomly (with uniform probability) pick n_{size} individuals
2. from them, choose the one with the best fitness

Selection criteria differences

Is criterion A better than criterion B? *Just* measure!

Criteria differ in how strongly they tend to prefer fit vs. unfit individuals:

- uniform selection: no preferences
- truncation selection: strong preference of fit individuals
- tournament: $n_{\text{size}} \to 1$: no preference, $n_{\text{size}} \to m$: strong preference

Selecting fit/unfit individuals

Strong preference (or selective/evolutionary pressure):

- population tends to converge to fittest individuals
- evolution concentrates in improving most promising solutions (**exploitation**)
- risk of "falling" in local optimum

Weak preference (or selective/evolutionary pressure):

- population includes also unfit individuals
- evolution investigates many different (maybe not promising) solutions (**exploration**)
- risk of not finding a good solution

Exploration/exploitation trade-off is hard to rule!

Selectors: common cases

- Reproduction: tournament of n_{size}
  - e.g., $m = n_{\text{pop}} = 500$, $n_{\text{size}} = 5$
- Survival: truncation

- Reproduction: fitness proportional
- Survival: truncation

Reproduction

Build n offspring from the m parents. How?
General scheme:

- given one or more parents, an offspring is generated by applying a unary or binary **genetic operator** on parent genotypes
  - unary (**mutation**): $f : \mathcal{G} \mapsto \mathcal{G}$
  - binary (recombination or **crossover**): $f : \mathcal{G}^2 \mapsto \mathcal{G}$

Then:

- given n and a set of weighted operators, generate offspring with operators according to their weights (deterministically or stochastically)
- *or* generate offspring by applying n (or $\frac{n}{2}$) times the crossover and then the mutation on the resulting individual(s)

Choice of operators

Operators:

- crossover for generating 80% of offspring
- mutation for generating 20% of offspring

Deterministically:

1. for $0.8n$ times
   - 1.1 select 2 parents (with reproduction selection criterion)
   - 1.2 apply crossover to genotypes
2. for $0.2n$ times
   - 2.1 select 1 parent (with reproduction selection criterion)
   - 2.2 apply mutation to genotype

Choice of operators

Operators:

- crossover for generating 80% of offspring
- mutation for generating 20% of offspring

Stochastically:

1. for n times
   - 1.1 randomly choose between mutation/crossover with 20/80 probability
   - 1.2 select 1 or 2 parents (with reproduction selection criterion) accordingly
   - 1.3 apply operator to genotype(s)

Mutation for bits string genotypes

Most classical option: probabilistic bit flip mutation

1. copy parent genotype g_p as child genotype g_c
2. for each bit in the in g_c, flip it ($0 \to 1$ or $1 \to 0$) with p probability

Commonly, $p = 0.01$

$$g_p = 00101001110101010110010010 1$$
$$g_c = 00101011110101010110110010 1$$

Crossover for (bits) string genotypes

Many options:

- one-point crossover
- two-points crossover
- n-points crossover
- uniform crossover
- ...

One-, two-, n-points crossover

Assume parents with equal genotype size:

1. choose randomly one (two, n) *cut points* in the genotype (indexes i such that $i < |g_{p_1}| = |g_{p_2}|$)
2. child bits before the cut point comes from parent 1, child bits after the cut point comes from parent 2

In general, jth bit comes from parent 1 iff closest larger cut point is even, from 2, otherwise.

One-, two-, n-point crossover

One-point:

$$g_{p_1} = 0010100111010101010|1100100101$$
$$g_{p_2} = 1110101010100101010|0101110111$$
$$g_c = 0010100111010101010\ 0101110111$$

Two-points:

$$g_{p_1} = 0010100|1110101010|1100100101$$
$$g_{p_2} = 1110101|0101001010|0101110111$$
$$g_c = 0010100\ 0101001010\ 1100100101$$

Uniform crossover

A cut point is placed at each index with $p = 0.5$ probability

Crossover with variable length (bits) string genotype

Many variants:

- one-, two-points crossover
  - cut points may be different within parents
  - child genotype size may be larger or smaller than parents sizes
- ...

One-point:

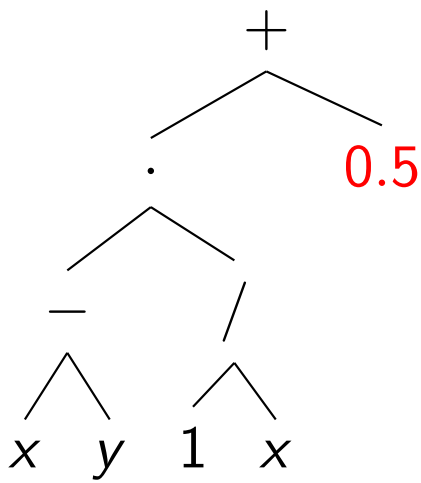
$$g_{p_1} = 0010100111010101010|1100100101$$
$$g_{p_2} = 111010101|0100101001011101111$$
$$g_c = 0010100111010101010\ 0100101001011101111$$

Genotype-phenotype mapping must allow for variable length genotypes!

Mutation (trees)

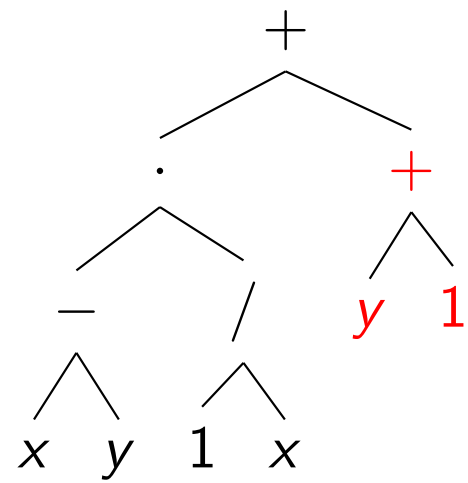
Parent

$$(x - y)\frac{1}{x} + 0.5$$



Child

$$(x - y)\frac{1}{x} + 1 + y$$



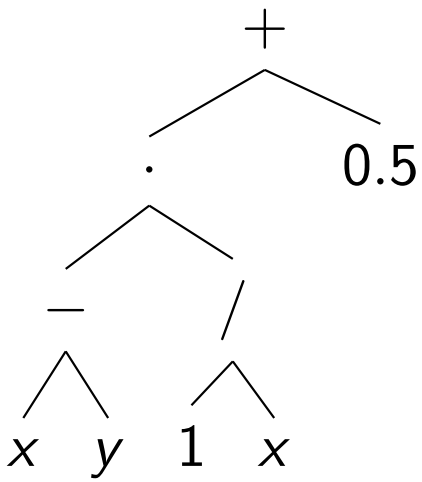
1. choose a random subtree
2. replace with a randomly generated subtree

Usually, constraints on depth

Crossover (trees)

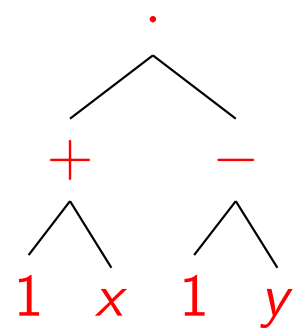
Parent 1

$$(x - y)\frac{1}{x} + 0.5$$



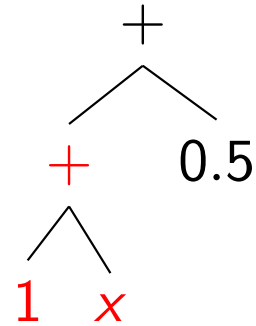
Parent 2

$$(1 + x)(1 - y)$$



Child

$$1 + x + 0.5$$



1. choose a random subtree in parent 1
2. choose a random subtree in parent 2
3. swap subtrees (child is copy of parent)

Usually, constraints on depth

Mutation for real-valued vectors ($\mathcal{G} = \mathbb{R}^p$)

Gaussian mutation

- parent g_p, child g_c
- for each $i \in 1, \ldots, p$, $g_c^i = g_p^i + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma)$
- σ is a parameter representing the mutation strength
  - large $\sigma \rightarrow$ exploration
  - small $\sigma \rightarrow$ exploitation

. . . and many similar variants

Crossover for real-valued vectors ($\mathcal{G} = \mathbb{R}^p$)

Besides all suitable for string genotypes, also (aka geometric crossover):

- parents g_{p_1}, g_{p_1}, child g_c
- for each $i \in 1, \ldots, p$, $g_c^i = g_{p_1}^i + \lambda(g_{p_2}^i - g_{p_1}^i)$, with $\lambda \sim \mathcal{U}(0, 1)$

Lacks the ability of explore out of the hyperrectangle enclosing the population

Role of operators

Mutation (x)or crossover?

- mutation $\rightarrow$ exploitation
- crossover $\rightarrow$ exploration

But the EC community is still debating about this point...

Population initialization

- Totally random
- More specific approaches, dependent on genotype form

Fitness

Fitness of an individual = ability to solve the problem of interest

- errors on several fitness cases by execution/simulation/application

Common cases:

- one numerical index
- more than one numerical indexes
- ...

Closely related with selectors

Many indexes: multiobjective

$$f(i) = \langle f_1(i), \ldots, f_n(i) \rangle$$

How to compare individuals i_1, i_2 ?

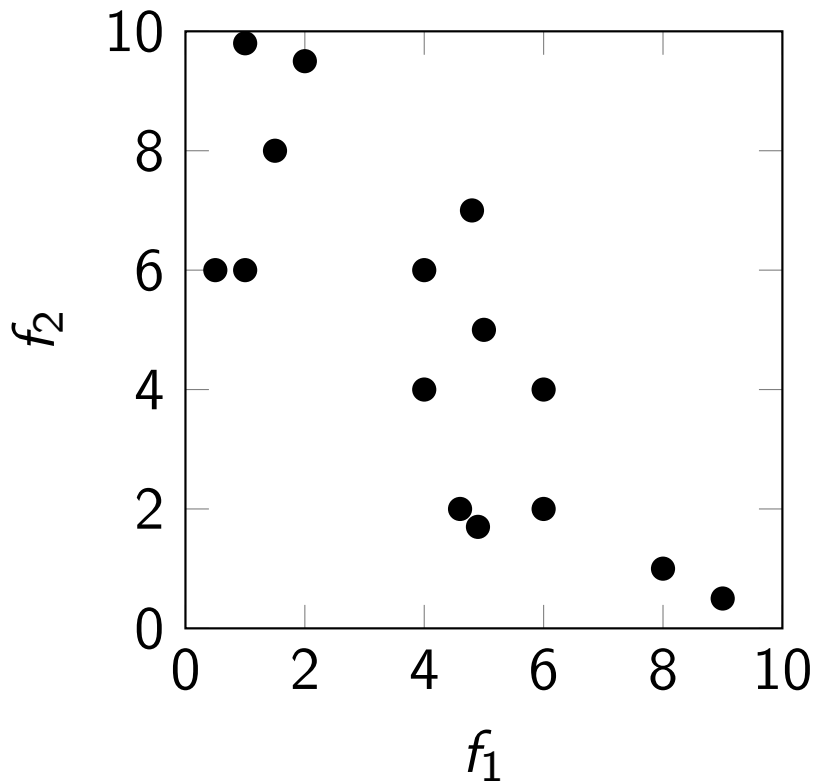
- linearization
  - $f(i) = \alpha_1 f_1(i) + \cdots + \alpha_n f_n(i)$
- lexicographical order
  - compare $f_1(i_1) \overset{?}{>} f_1(i_2)$; if tie, $f_2(i_1) \overset{?}{>} f_2(i_2)$; ...
- Pareto dominance
- ...

Q: with which selectors?

Pareto dominance

i_1 dominates i_2 iff:

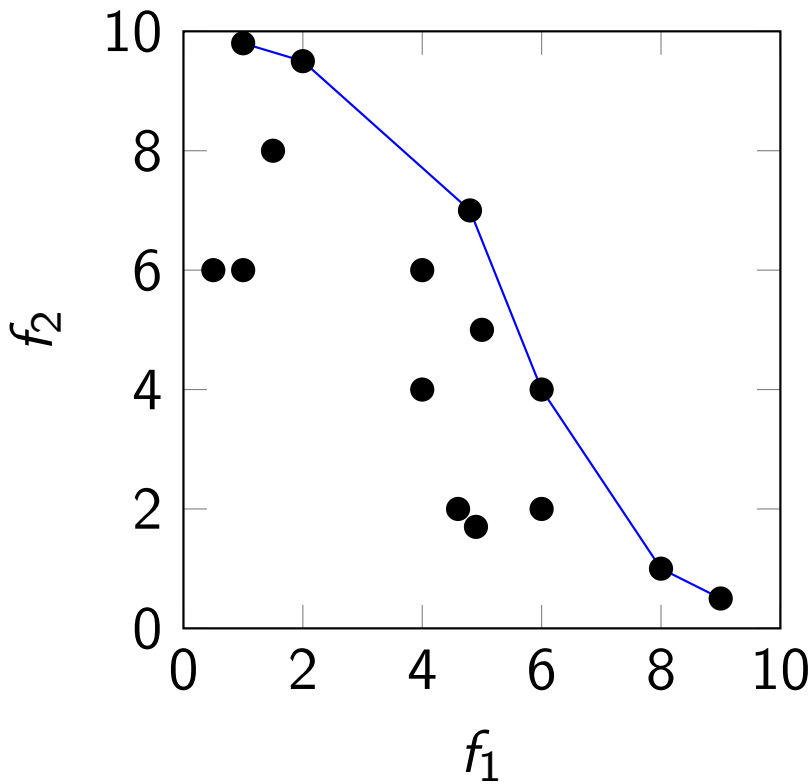
$$\forall j, f_j(i_1) \geq f_j(i_2) \wedge \exists k, f_k(i_1) > f_k(i_2)$$



Pareto dominance

i_1 dominates i_2 iff:

$$\forall j, f_j(i_1) \geq f_j(i_2) \wedge \exists k, f_k(i_1) > f_k(i_2)$$

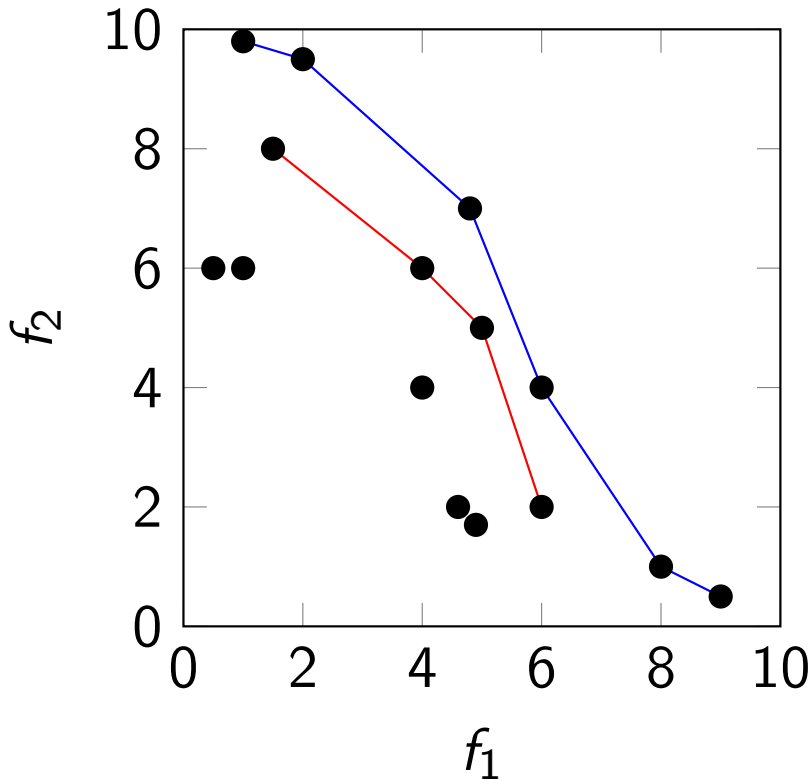


- 1st Pareto front: undominated solutions

Pareto dominance

i_1 dominates i_2 iff:

$$\forall j, f_j(i_1) \geq f_j(i_2) \wedge \exists k, f_k(i_1) > f_k(i_2)$$

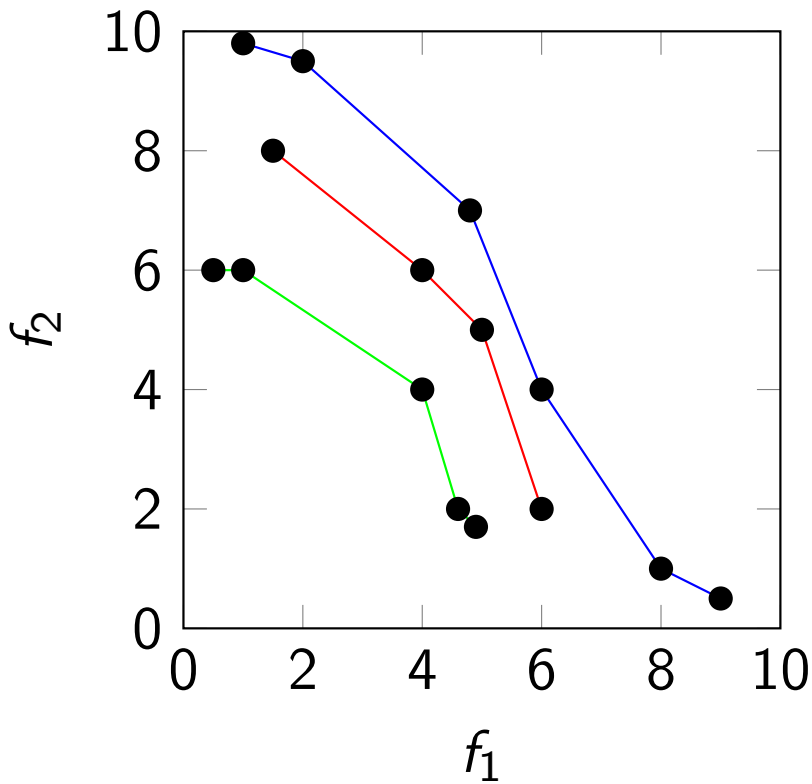


- 1st Pareto front: undominated solutions
- 2nd Pareto front: undominated solutions, while not considering 1st front

Pareto dominance

i_1 dominates i_2 iff:

$$\forall j, f_j(i_1) \geq f_j(i_2) \wedge \exists k, f_k(i_1) > f_k(i_2)$$



- 1st Pareto front: undominated solutions
- 2nd Pareto front: undominated solutions, while not considering 1st front
- ...

An example EA

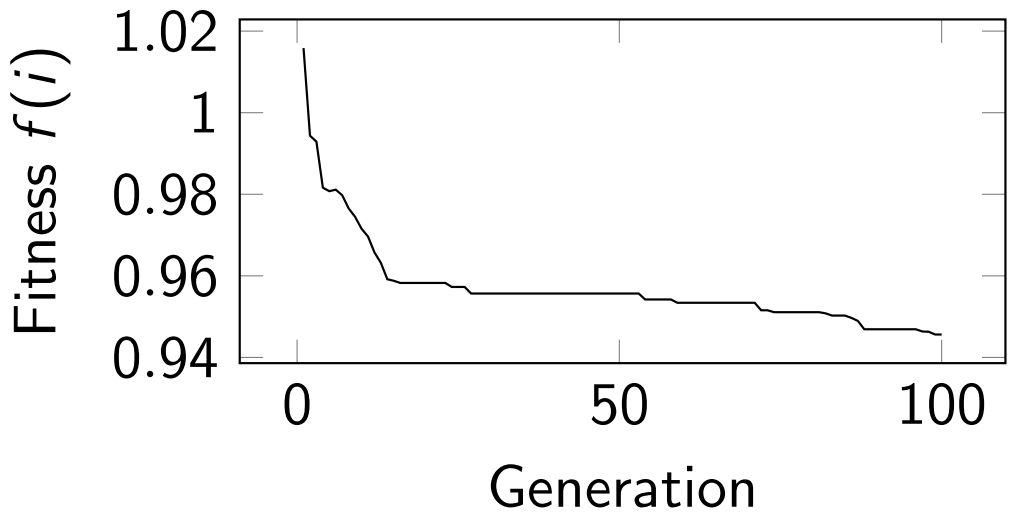
$b \leftarrow 0$
$I = \text{Initialize}()$
while $b \leq n_{\text{pop}} n_{\text{gen}}$ **do**
  $I' = \emptyset$
  for all $i \in \{1, \ldots, n_{\text{pop}}\}$ **do**
    $(g_{p_1}, p_{p_1}, f_{p_1}) \leftarrow \text{SelTournament}(I, n_{\text{tour}})$
    $(g_{p_2}, p_{p_2}, f_{p_2}) \leftarrow \text{SelTournament}(I, n_{\text{tour}})$
    $g_c \leftarrow o_m(o_c(g_{p_1}, g_{p_2}))$
    $I' \leftarrow I' \cup \{(g_c, \Phi(g_c), f(\Phi(g_c))\}$
    $b \leftarrow b + 1$
  end for
  $I \leftarrow I \cup I'$
  while $|I| > n_{\text{pop}}$ **do**
    $I \leftarrow I \setminus \text{SelWorst}(I)$
  end while
end while

In practice

- Is my EA working?
- When to stop evolution?
- How to choose value for parameter X?

In practice

- Is my EA working?
- When to stop evolution?
- How to choose value for parameter X?



On many (≥ 30) runs!

Issues

- Diversity
- Variational inheritance
- Expressiveness
- ...

Diversity

Is the population diverse enough?

- "No" $\rightarrow$ too much exploitation $\rightarrow$ local minimum
- "Yes" $\rightarrow$ in principle, no drawbacks
  - how to measure diversity?
  - how to enforce/promote diversity?

Giovanni Squillero and Alberto Tonda. "Divergence of character and premature convergence: a survey of methodologies for promoting diversity in evolutionary optimization". In: *Information Sciences* 329 (2016), pp. 782–799

Variational inheritance

Are children similar but not identical to parents?

- "Too much similar" $\rightarrow$ too much exploitation $\rightarrow$ local minimum, no/slow evolution
- "Too much different" $\rightarrow$ no exploitation, just coarse exploration (random walk)

- How to measure? (locality, redundancy, degeneracy, uniformity, ...)
- How to tackle? Operators, mapping, both?

Expressiveness

Is the representation (phenotype) expressive enough?

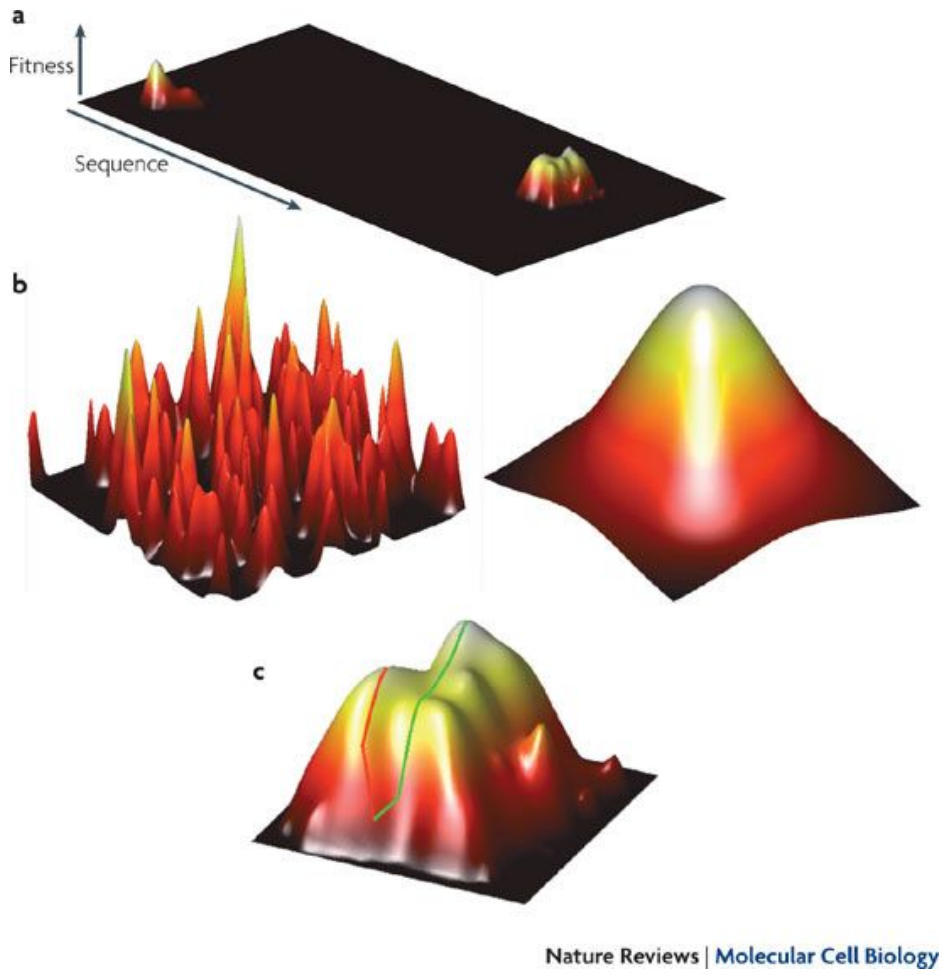
- "Low expressiveness" $\rightarrow$ good/optimal solution might not be representable, or might not be reachable
- "Large expressiveness" $\rightarrow$ large search space $\rightarrow$ very long or infiniti convergence time

Fitness landscape

- How are genotype and fitness spaces related?
- What does a small step on one correspond to on the other?

Q: is phenotype space relevant?

Fitness landscape



Philip A Romero and Frances H Arnold. "Exploring protein fitness landscapes by directed evolution". In: *Nature reviews Molecular cell biology* 10.12 (2009), p. 866

EC in action

https://youtu.be/4pdiAneMMhU

Genetic Algorithms (GA)

- Genotype = phenotype = bits string
- $m = n \approx 1000$, no overlapping
- Fitness-proportional selection, or multiobjective (Pareto-based) selection

- Most widely used/studied
- Genotypes often encodes numerical parameters

Genetic Programming (GP)

Focus: individuals are programs

- Genotype = phenotype = tree (tree-based GP) or list of instructions (linear GP)
- $m = n \approx 1000$, overlapping
- Tournament selection

- Syntactic/semantic validity?

Grammatical Evolution (GE)

A form of GP based on GA, given a context-free grammar $\mathcal{G}$

- Genotype = bits string, phenotype = string $\in \mathcal{L}(\mathcal{G})$, by means of a mapping procedure
- steady state ($m \approx 500, n = 1$, overlapping) or $m = n$, overlapping
- Tournament selection

```
<expr> ::= ( <expr> <op> <expr> ) | <var> | <num>
<op> ::= + | - | * | /
<var> ::= x | y
<num> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Properties of the representation

Given:

- genotype space $\mathcal{G}$, phenotype space $\mathcal{P}$, fitness space $\mathcal{F}$
  - a (partial) order $\prec$ existing in $\mathcal{F}$
- genotype-pheotype mapping: $\Phi : \mathcal{G} \mapsto \mathcal{P}$
  - often $\Phi : \mathcal{G} \mapsto \mathcal{P} \cup \perp$, where $\perp$ represents an *invalid* solution
- fitness function: $f : \mathcal{P} \mapsto \mathcal{F}$
- mutation $o_m : \mathcal{G} \mapsto \mathcal{G}$ and crossover $o_c : \mathcal{G}^2 \mapsto \mathcal{G}$
- distances $d_{\mathcal{G}} : \mathcal{G}^2 \mapsto \mathbb{R}^+$ and $d_{\mathcal{P}} : \mathcal{P}^2 \mapsto \mathbb{R}^+$

some properties of the representation can be defined

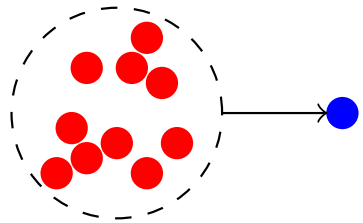
Invalidity

The tendency of generating invalid phenotypes:

$$\text{invalidity} = \frac{|\{g \in \mathcal{G} : \Phi(g) = \perp\}|}{|\mathcal{G}|}$$

or, experimentally, with G

Degeneracy



The degree to which different genotypes are mapped to the same phenotype:

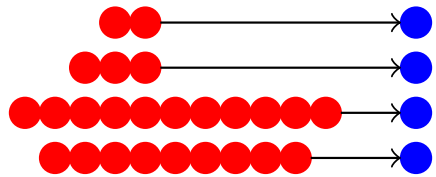
$$\text{degeneracy} = 1 - \frac{|\mathcal{P}|}{|\mathcal{G}|}$$

or, experimentally, with G and $P = \{\Phi(g), g \in G\}$

Notes:

- often called redundancy
- assuming $\mathcal{P}$ is the range of Φ (i.e., $\mathcal{P} = \{\Phi(g), g \in \mathcal{G}\}$)

Uniformity of degeneracy



The degree to which the sizes of different sets of genotypes mapping to the same phenotype differ:

$$\mathcal{G}_i = \{g \in \mathcal{G} : \Phi(g) = p_i\}, \forall p_i \in \mathcal{P}$$
$$S = \{|\mathcal{G}_1|, |\mathcal{G}_2|, \ldots, |\mathcal{G}_{|\mathcal{P}|}|\}$$
$$\text{non-uniformity} = \frac{\sigma_S}{\mu_S} \quad \text{(coefficient of variation)}$$

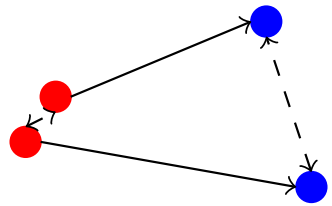
or, experimentally, with G and $P = \{\Phi(g), g \in G\}$

Redundancy

The degree to which parts of the genotype do not concur in the mapping process

- how to measure depends on the representation
- is a source of degeneracy: genotypes which differ in redundant part are (likely) mapped to the same phenotype

Locality



The degree to which close genotypes are mapped to close phenotype:

$$D_{\mathcal{G}} = \{d_{\mathcal{G}}(g_i, g_j), i, j \in \{1, \ldots, |\mathcal{G}|\}\}$$
$$D_{\mathcal{P}} = \{d_{\mathcal{P}}(\Phi(g_i), \Phi(g_j)), \{i, j \in 1, \ldots, |\mathcal{G}|\}\}$$
$$\text{locality} = \text{cor}(D_{\mathcal{G}}, D_{\mathcal{P}})$$

or, experimentally, with G and $P = \{\Phi(g), g \in G\}$; or simpler versions for discrete spaces (no need for distance, nor for correlation)

Evolvability

The likelihood of obtaining a better individual after the application of a genetic operator

- involves the operator and the fitness

$$\text{evolvability}_{\text{mutation}} = P(f(\Phi(o_m(g_p)) \prec f(\Phi(g_p)))$$

$$\text{evolvability}_{\text{crossover}} = P\left(\begin{array}{c} f(\Phi(o_c(g_{p_1}, g_{p_2})) \prec f(\Phi(g_{p_1})) \\ \wedge \\ f(\Phi(o_c(g_{p_1}, g_{p_2})) \prec f(\Phi(g_{p_2})) \end{array}\right)$$

More in general: the tendency of an evolutionary system to improve solutions

Why properties matter: the GE case

$g = 01101001\ 00001101\ 01011000\ 00000011\ 11000110\ 01111101$ (bits)

$= 105\ 13\ 88\ 3\ 198\ 125$ (integers)

i	g_i	$\lvert r_s \rvert$	j	w	Phenotype p
					`<expr>`
0	105	3	0	0	`( <expr> <op> <expr> )`
1	13	3	1	0	`( <var> <op> <expr> )`
2	88	2	0	0	`( x <op> <expr> )`
3	3	4	3	0	`( x / <expr> )`
4	198	3	0	0	`( x / ( <expr> <op> <expr> ) )`
5	125	3	2	0	`( x / ( <num> <op> <expr> ) )`
0	105	10	5	1	`( x / ( 5 <op> <expr> ) )`
1	13	4	1	1	`( x / ( 5 - <expr> ) )`
2	88	3	1	1	`( x / ( 5 - <var> ) )`
3	3	2	1	1	`( x / ( 5 - y ) )`

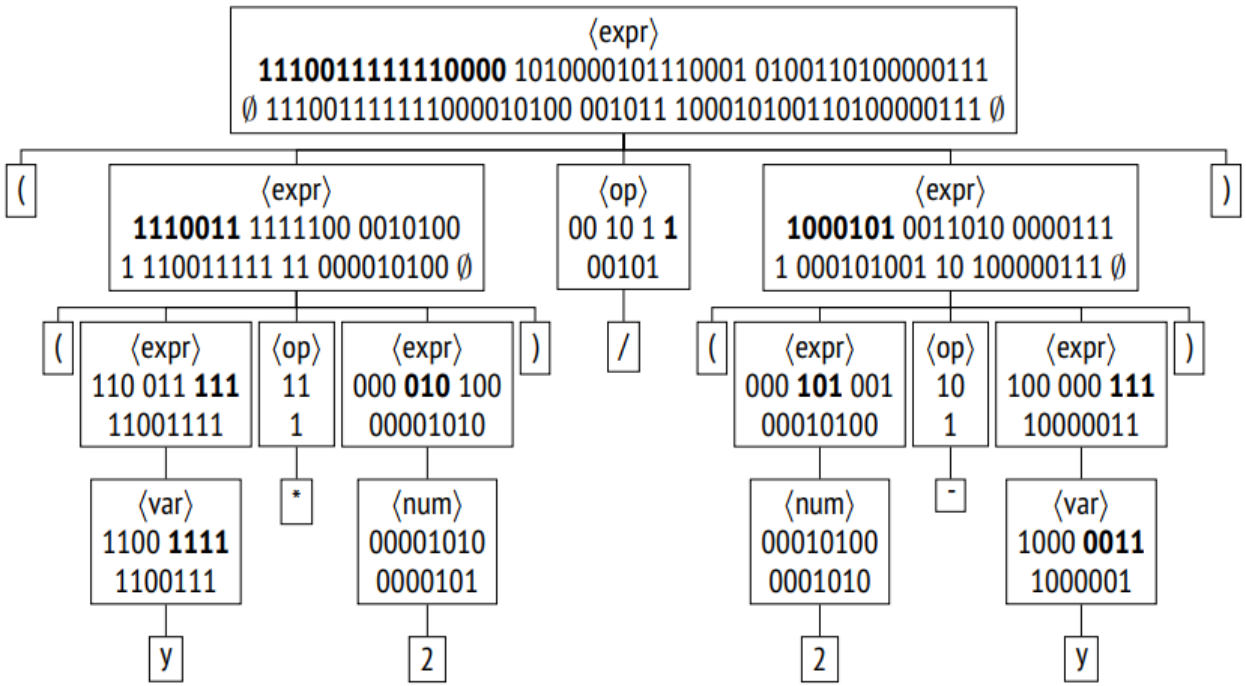
```
<expr> ::= ( <expr> <op> <expr> ) | <var> | <num>
<op> ::= + | - | * | /
<var> ::= x | y
<num> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Q: which properties are affected?

An alternative: WHGE genotype-phenotype mapping

$$g = 1110011111111000010100001011100010100110100000111$$

$$p = ((y * 2)/(2 - y))$$



Alberto Bartoli, Mauro Castelli, and Eric Medvet. "Weighted Hierarchical Grammatical Evolution". In: *IEEE transactions on cybernetics* (2018)

Lab: Build your one EA ($\approx 3\,\mathrm{h}$)

Build an EA that can be used for solving the problem of finding a given target string. *(A toy problem!)*

1. decide solution representation
2. decide fitness function
3. decide EA components
   - generational model
   - genetic operators
   - selection criteria
4. implement
5. investigate impact of other design choices

Section 2

Evolutionary robotics

What is Evolutionary Robotics?

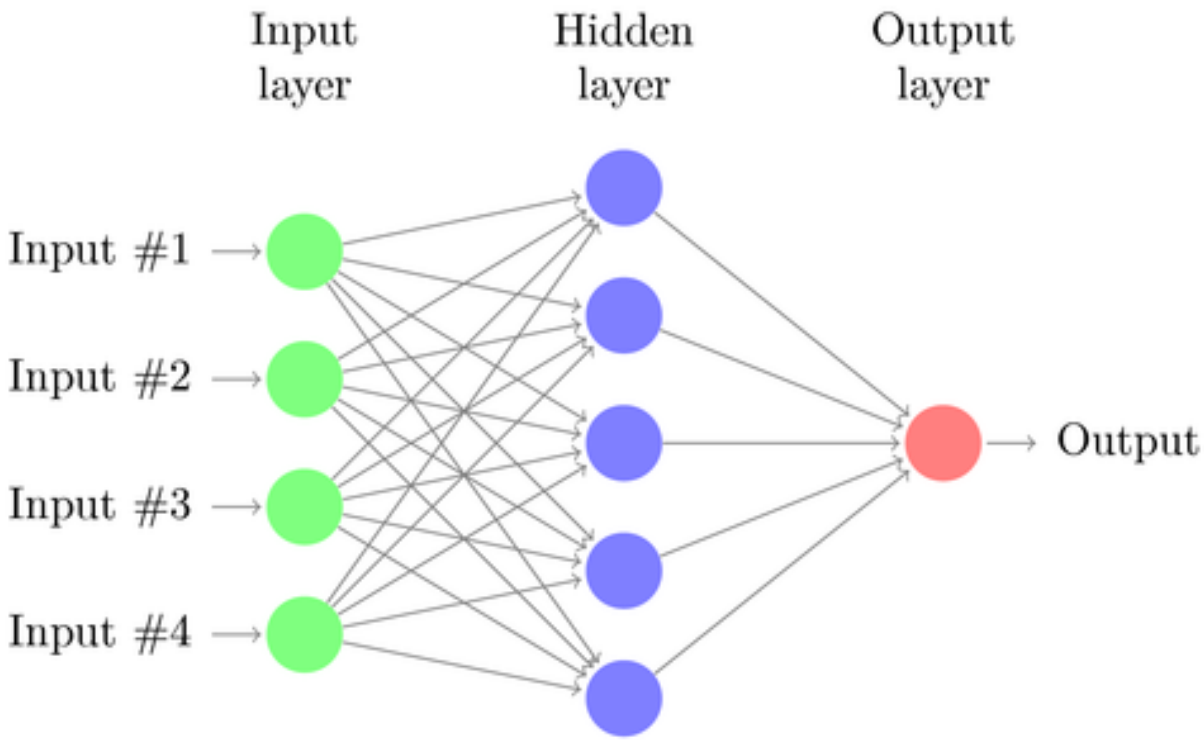
Definition

Evolutionary Robotics is the application of the techniques, methods, and principles of *Evolutionary Computation* for the automatic design of the *body* (morphology) and the *mind* (controller) of autonomous robotic agents.

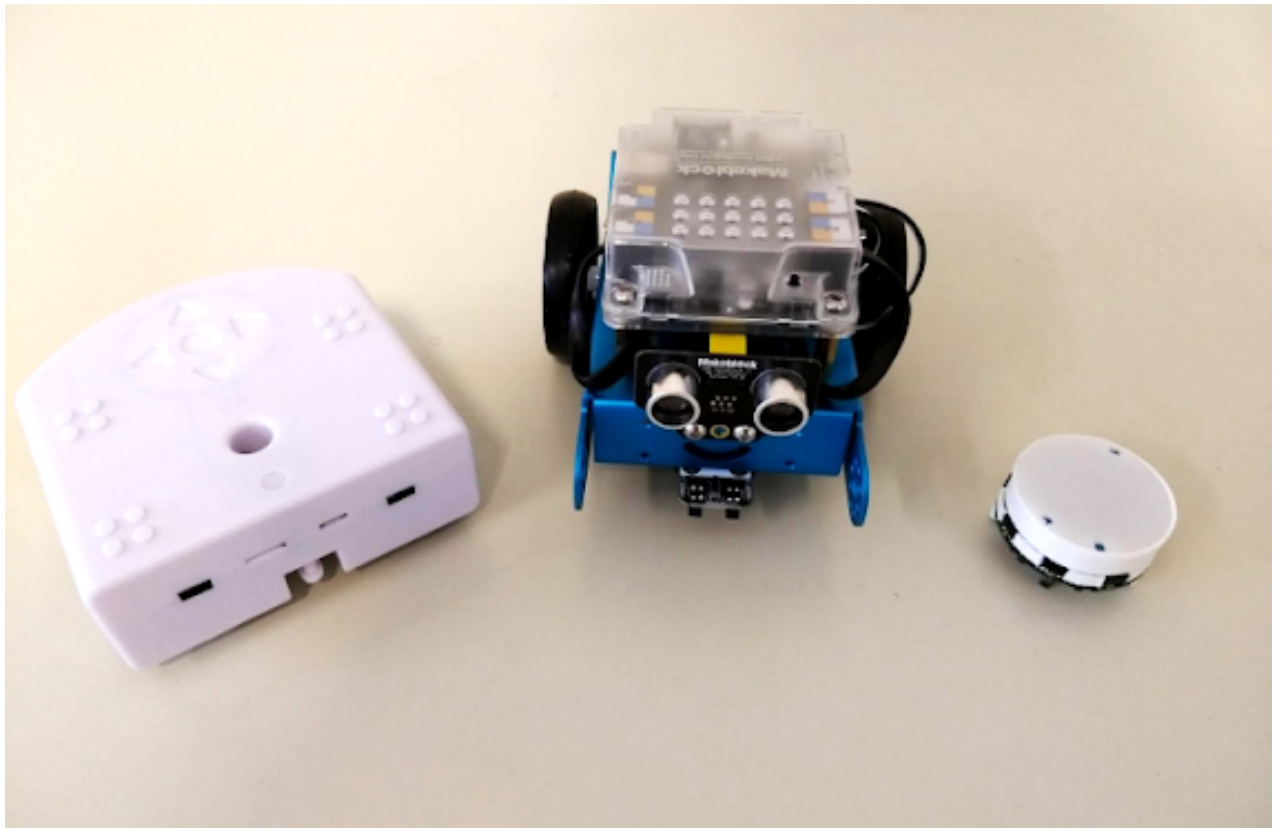
Subsection 1

Evolving a neural network

What is artificial neural network (ANN)?



ANN for controlling (small) robots



E.g., Thymio-II robot:

- inputs: 7 IR proximity sensors (and others)
- outputs: 2 motors (wheels)

Designing a neural network

Choose:

- topology
- weights (θ)
- other "details": e.g., activation function

Just the weights

A possible EC-based approach:

- phenotype: ANN with pre-fixed topology
- genotype: $\theta \in \mathbb{R}^p$, p depending on the topology
- genetic operators suitable for $\mathcal{G} = \mathbb{R}^p$
- other representation-independent parameters (e.g., selection criteria, generational model)
- fitness (mainly problem-related)

How to choose the topology?

- usually, input and output size are predefined, so...
- ...how to choose how many hidden layers and how many neurons per layer? (expressiveness)

Subsection 2

NEAT

An alternative to "just the weights"

Kenneth O Stanley and Risto Miikkulainen. "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2 (2002), pp. 99–127

Key ideas:

- evolve topology and weights together (TWEANN)
- starting with simple topology and then add complexity
- "protect" innovation

Representation for TWEANN

How to represent the set of ANN with different topologies?

- direct: genotype $\approx$ phenotype
  - genotype specifies nodes, connections, and weights
- indirect: genotype $\neq$ phenotype
  - genotype specifies how to build a phenotype

Key question: how to meaningfully do crossover?

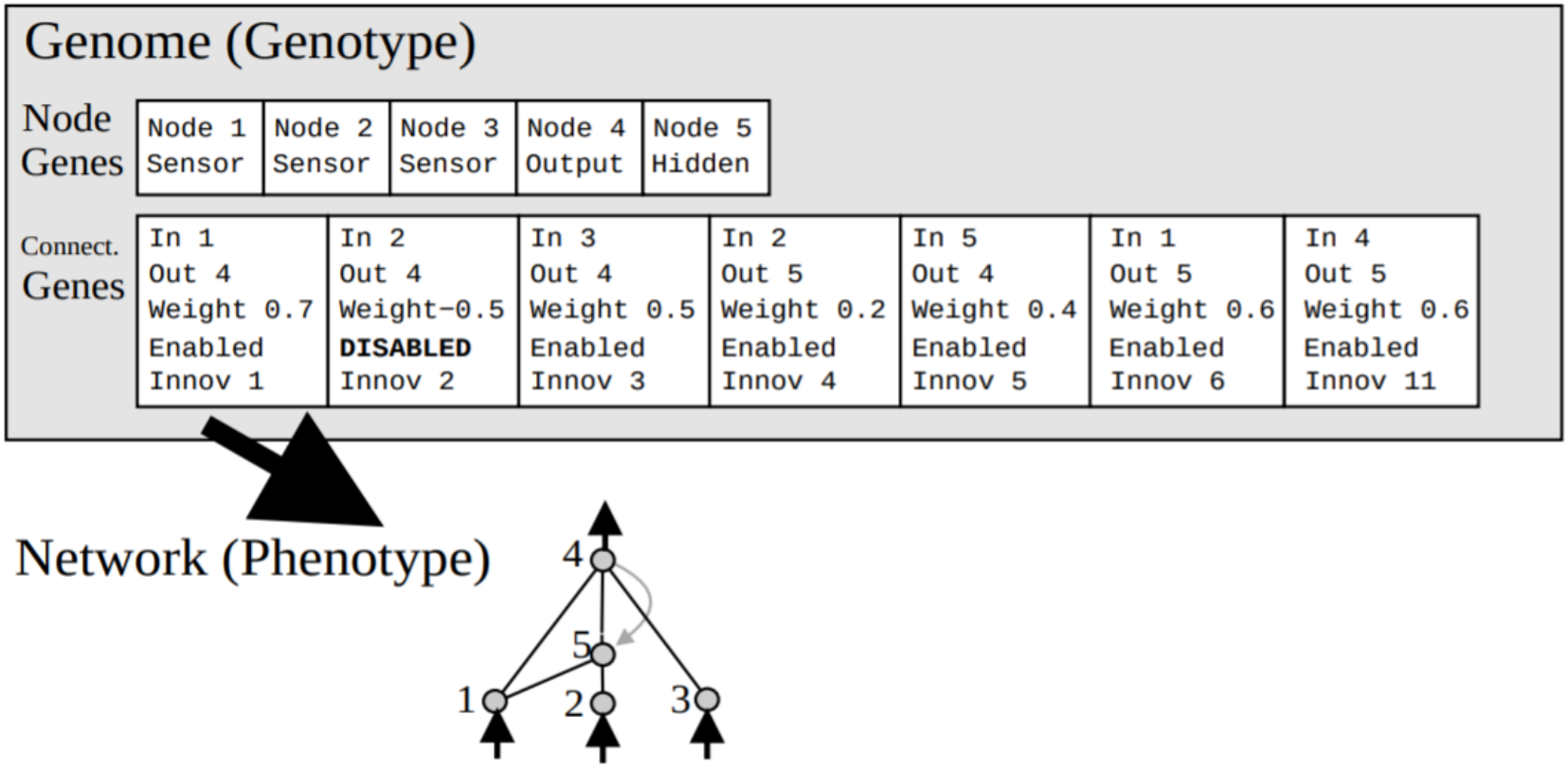
- meaningfully $\rightarrow$ variational principle
- trivial solution: avoiding crossover

Crossover

How to meaningfully do crossover?

- networks (genotypes) of different size
- competing conventions (degeneracy)
  - many genotypes for the same network, how to align components?

NEAT representation



Key component: the innovation number!

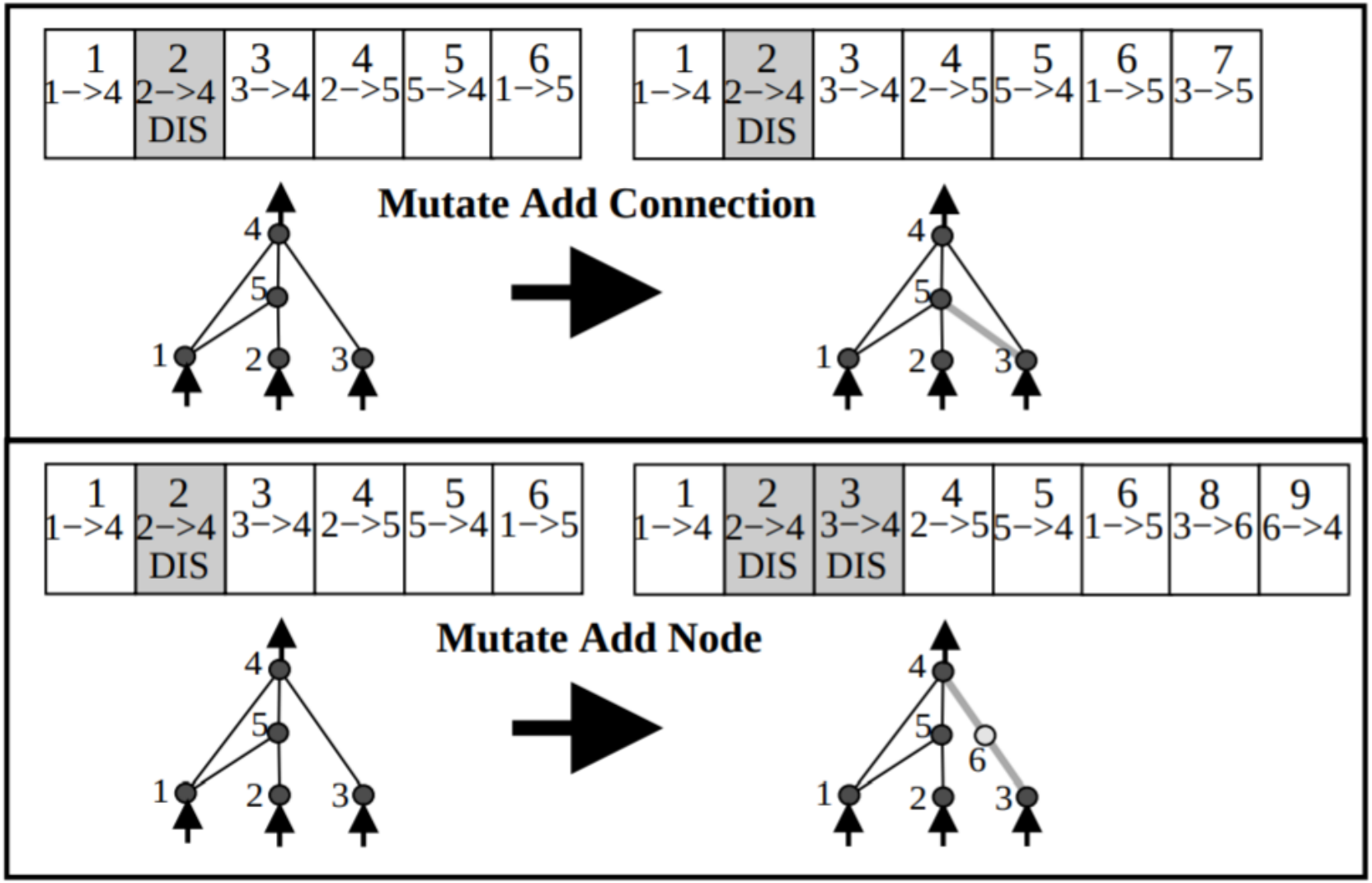
- a *global* counter assigned to any new created connection, on whichever individual

Mutation(s)

Three variants:

- perturb a weight
- add a connection with random weight (new gene!)
- add a node (new gene!)
  - "in the middle" of a connection (w_{old}): the existing connection is disabled and replaced with two connections connecting the new node ($w^{\text{in}}_{\text{new}} = 1$, $w^{\text{out}}_{\text{new}} = w_{\text{old}}$)

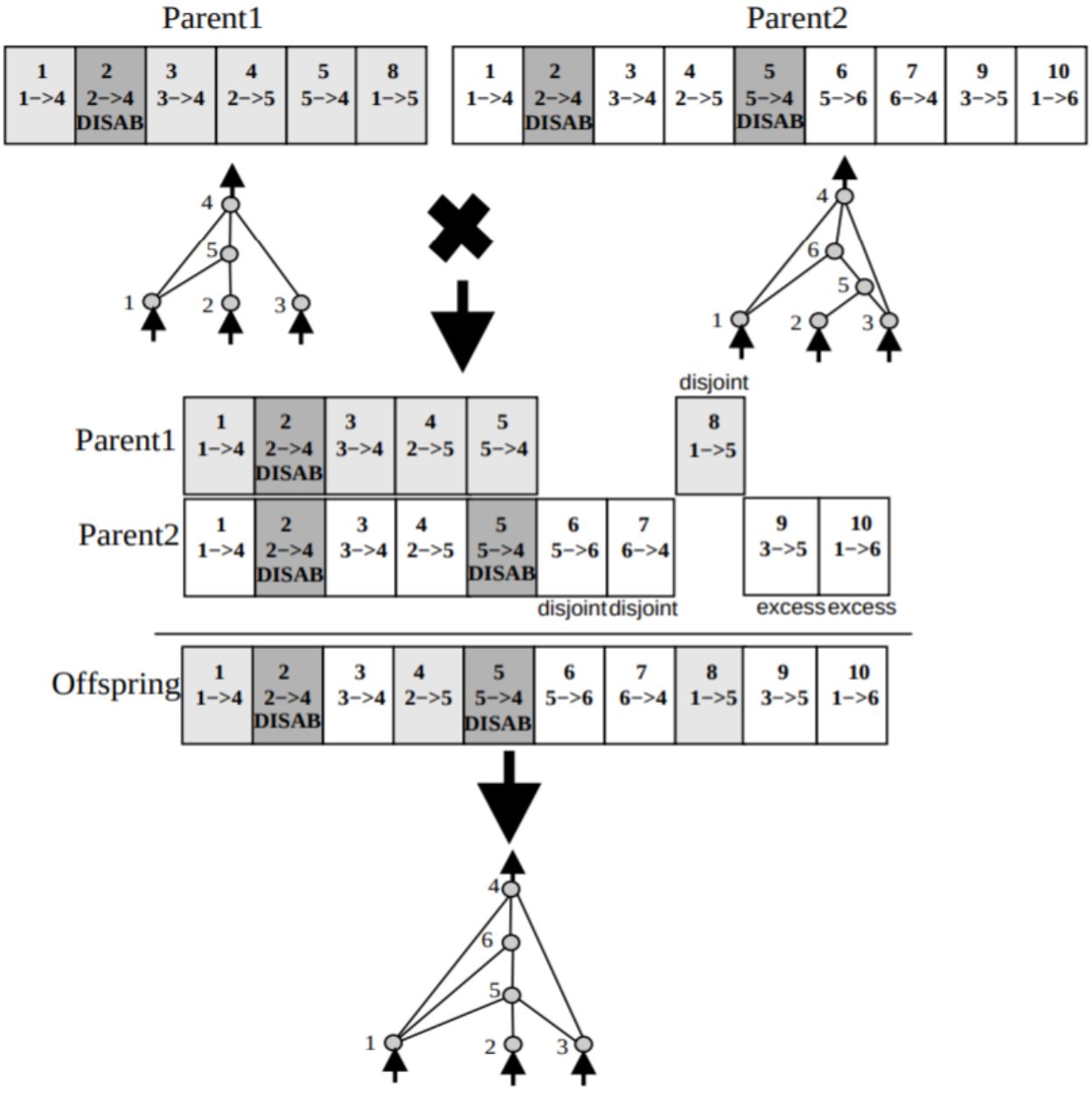
Mutation(s)


1 1->4
2 2->4 DIS
3 3->4
4 2->5
5 5->4
6 1->5
1 1->4
2 2->4 DIS
3 3->4
4 2->5
5 5->4
6 1->5
7 3->5
Mutate Add Connection
4
5
1
2
3
4
5
1
2
3
1 1->4
2 2->4 DIS
3 3->4
4 2->5
5 5->4
6 1->5
1 1->4
2 2->4 DIS
3 3->4 DIS
4 2->5
5 5->4
6 1->5
8 3->6
9 6->4
Mutate Add Node
4
5
1
2
3
4
5
6
1
2
3

Crossover

1. Align genes by innovation number
2. *Matching* genes are inherited randomly from one of the parents
3. The remaining (disjoint or excess) are inherited from the more fit parent

Crossover


Parent1
1 1->4
2 2->4 DISAB
3 3->4
4 2->5
5 5->4
8 1->5
Parent2
1 1->4
2 2->4 DISAB
3 3->4
4 2->5
5 5->4 DISAB
6 5->6
7 6->4
9 3->5
10 1->6
disjoint
disjoint
disjoint
excess
excess
Offspring

Complexification

Instead of starting with a random population, including possibly complex individuals, start with only simple individuals (ANN with no hidden layer) $\rightarrow$

- search space is "never" too large, but
- expressiveness is still virtually high

Protecting innovations

Despite representation and operators attempt to reduce the risk, structural modification might negatively affect the fitness
Idea: protect the innovative individuals

Protect new individuals

Instead of competing against the full population, individuals compete in a limited niche (*speciation*):

- a specie is a set of similar individuals
- in NEAT, similarity is based on number matching nodes and weights
- individuals in the same niche share the fitness: $f'(x_i) = \frac{f(x_i)}{|P_j|}$

NEAT works!

- Experimentally evaluated when introduced:
  - two tasks: XOR network, two poles on cart balancing
  - the three components are all useful
- Still widely used
- Extended in many ways

Subsection 3

How to choose the fitness?

Premise

EC as a *black box* optimizer: I do not know anything about the problem, I just want a solution

In ER:

- I want the robot to perform a task
- I do not know how

A priori knowledge

"I do not know how" $\rightarrow$ often, it is not fully true!

Extreme cases:

- high knowledge: the desired fine behavior is known
- no knowledge: just the aggregate goal is known

$\rightarrow$ how to design the fitness function?

Fitness function design

Mohammad Divband Soorati and Heiko Hamann. "The effect of fitness function design on performance in evolutionary robotics: The influence of a priori knowledge". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 153–160

Scenario (simulated):

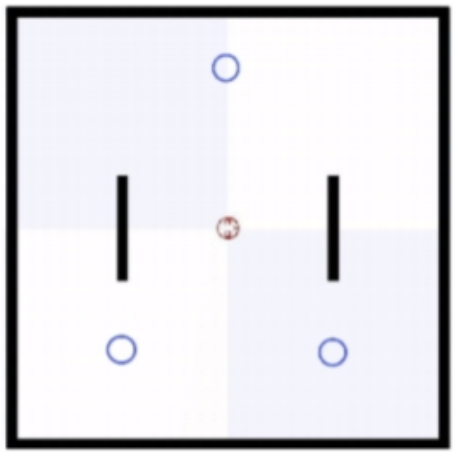
- small robots with 2 wheels, 6 front and 2 back proximity sensors
- moving in an arena of $2\,\mathrm{m} \times 2\,\mathrm{m}$
- ANN controller, evolved using NEAT

Tasks

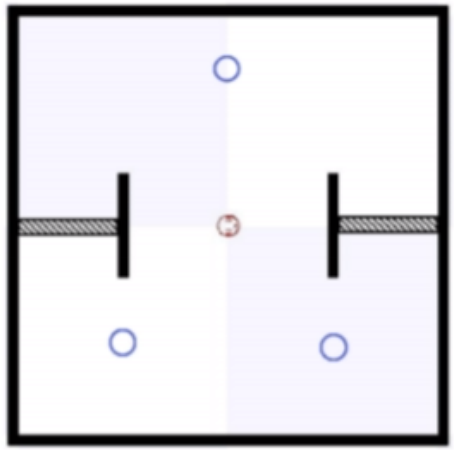
Three tasks of increasing complexity:

- obstacle avoidance: move w/o colliding with obstacles
- goal homing: move towards a light, avoiding collisions
- periodic goal homing: periodically move towards to and away from a light, avoiding collisions

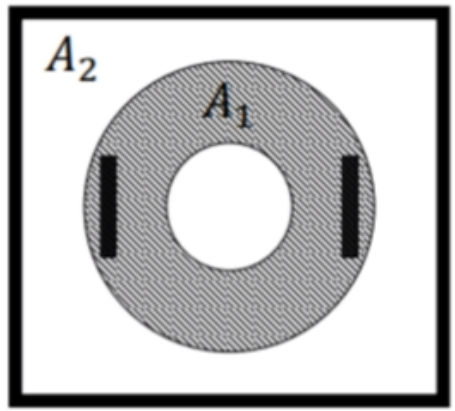
Arenas



(a) 1st robot arena



(b) 2nd robot arena



(c) light intens. areas

- first arena used for evolution
- second arena for testing adaptation

Fitness functions

Three degrees of a priori knowledge incorporation:

- Aggregate (AFF): low a priori knowledge, what is achieved
- Behavioral (BFF): medium, how it is achieved
- Tailored (TFF): combination (product) of AFF and BFF
- Functional incremental (FIFF): sequence (over the evolution) of fitness functions, AFF, then BFF

AFF, low a priori knowledge

- obstacle avoidance: move w/o colliding with obstacles

$$f = d \qquad \text{(traveled distance)}$$

- goal homing: move towards a light, avoiding collisions

$$f = \sum_t I(t) \qquad \text{(light intensity)}$$

- periodic goal homing: periodically move towards to and away from a light, avoiding collisions

$$f = \sigma_{I(t)} \tag{1}$$

BFF, medium a priori knowledge

- obstacle avoidance: move w/o colliding with obstacles

$$f = \underbrace{\left(\frac{\overline{v}_l + \overline{v}_r}{2}\right)}_{\text{move fast}} \underbrace{(1 - \sqrt{|\overline{v}_l - \overline{v}_r|})}_{\text{move straight}} \underbrace{\left(\frac{1}{\theta} \min_t(\min_i s_i(t))\right)}_{\text{maximize dist from obstacles}}$$

BFF, medium a priori knowledge

- goal homing: move towards a light, avoiding collisions

$$f = \sum_t |cI(t) - v(t)| \tag{2}$$

Notes:

- after collision, $I(t)$ is halved for some time
- idea: move fast when far the light, move slowly (stop) when in the light

BFF, medium a priori knowledge

- periodic goal homing: periodically move towards to and away from a light, avoiding collisions

$$f = \sum_t \varphi(t) v(t) \tag{3}$$

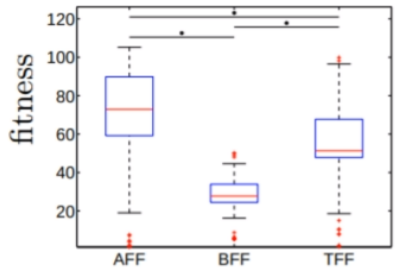
Notes:

- $\varphi(t) = \begin{cases} -1 & \text{in medium light} \\ +1 & \text{otherwise} \end{cases}$
- evaluation stopped upon collision
- idea: move fast when in medium light, slowly otherwise

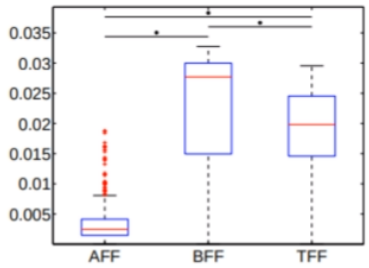
And then?

- lot of tests in many cases...
- also for testing adaptation (evolve on one arena, then resume evolution on another arena)

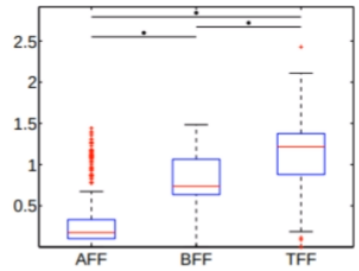
And then?



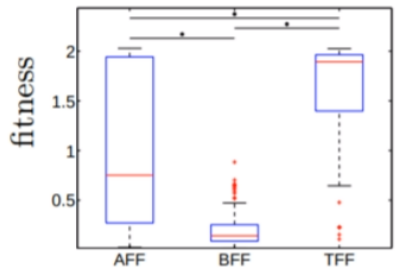
(d) OA, fitness AFF.



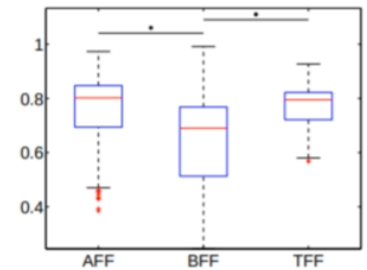
(e) OA, fitness BFF



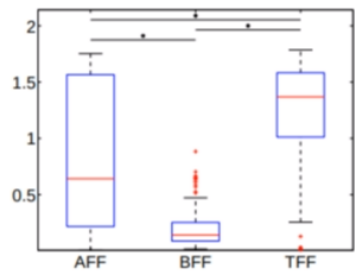
(f) OA, fitness TFF



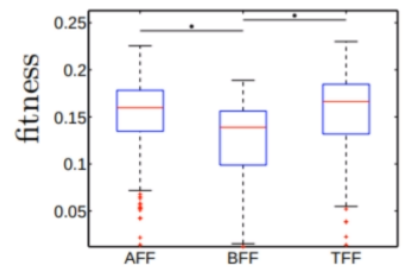
(g) GH, fitness AFF



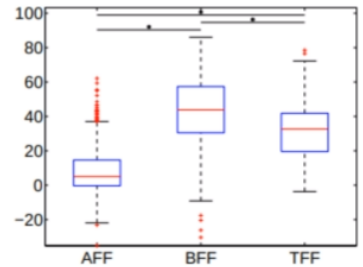
(h) GH, fitness BFF



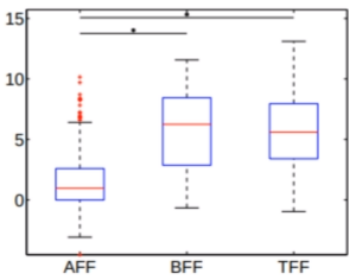
(i) GH, fitness TFF



(j) PGH, fitness AFF



(k) PGH, fitness BFF



(l) PGH, fitness TFF

And then?

"Fitness functions that include a high degree of a priori knowledge (here TFF and BFF) influence the performance positively. They help to simplify the evolution of a successful controller and hence simplify the chosen task. However, [...] foils the dominant idea of evolutionary computation"

And what if the knowledge is wrong? What if it incorporates a strong bias about how to solve the task?

Subsection 4

Soft robots

Soft robots

https://www.youtube.com/watch?v=A7AFsk4ONGE

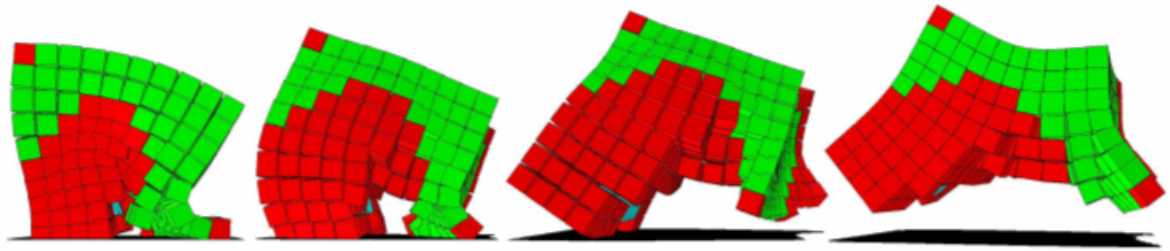
Soft robots

- w.r.t. to rigid-elements counterparts, soft robots may exhbit "infinite" degrees of freedom
- more complex to design and control

Voxel-based Soft robots (VSRs)

An aggregation of elastic cubic building blocks (voxels):

- morphology (body): how the voxels are aggregated
- controller (mind): how the voxel volumes change over the time



Sam Kriegman et al. "Simulating the evolution of soft and rigid-body robots". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM. 2017, pp. 1117–1120

Embodied cognition paradigm

Rolf Pfeifer and Josh Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006

"both bodies and brains combine to produce complex behaviors, in contrast to the traditional view that the only seat of intelligence is the brain"

- VSRs composed of many simple parts
- easy to put complexity in the body, rather than in the brain

Evolving a controller for a VSR

Sam Kriegman, Nick Cheney, and Josh Bongard. "How morphological development can guide evolution". In: *Scientific reports* 8.1 (2018), p. 13934

- VSR with pre-defined morphology ($4 \times 4 \times 3$ voxels)
- actual goal: investigating if development is beneficial to evolution
- task: locomotion

Representation

Each voxel volume varies with sin function:

$$s_i(k) = s_i^0 + a \sin(2\pi f k \Delta t + \phi_i) \tag{4}$$

Amplitude and frequency equal for all the voxels; phase ϕ_i and resting volume s_i^0) are subjected to evolution.

- individual is $\theta_{\mathrm{NS}} = (s_1^0, \phi_1, \ldots, s_n^0, \phi_n)$
- with bilateral symmetry for resting volumes

EA

$P \leftarrow \emptyset$
for all $i \in \{1, \ldots, n_{\text{pop}}\}$ **do**
    $P \leftarrow P \cup (\text{random}(), 0)$
end for
for all $i \in \{1, \ldots, n_{\text{gen}}\}$ **do**
    $P' \leftarrow \emptyset$
    for all $(\theta, a) \in P$ **do**
        $\theta' \leftarrow \text{mutate}(\theta)$
        $P' \leftarrow P' \cup (\theta, a + 1)$
        $P' \leftarrow P' \cup (\theta', a + 1)$
    end for
    $P' \leftarrow P' \cup (\text{random}(), 0)$
    $P \leftarrow \text{select}(P', n_{\text{pop}})$
end for

EA: some details

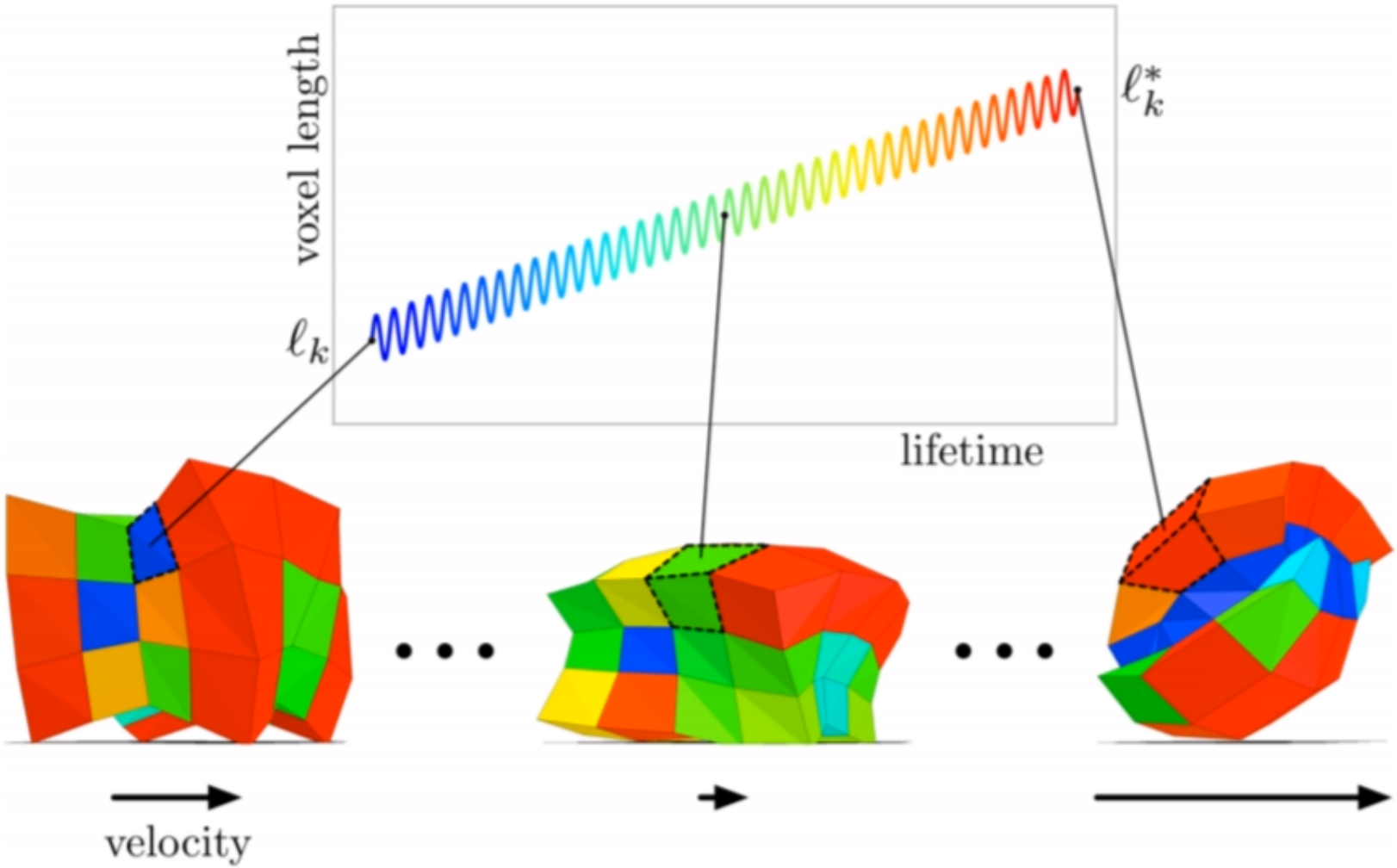
- mutation-only
- Pareto-dominance based optimization: maximize fitness, minimize age $\rightarrow$ favor diversity
- overlapping and truncation selection: high selective pressure

Evo+devo

Evo: evolution, devo: development

- evo only: params do not change during the "life" of the robot (simulation)
- evo+devo: params linearly change over the life

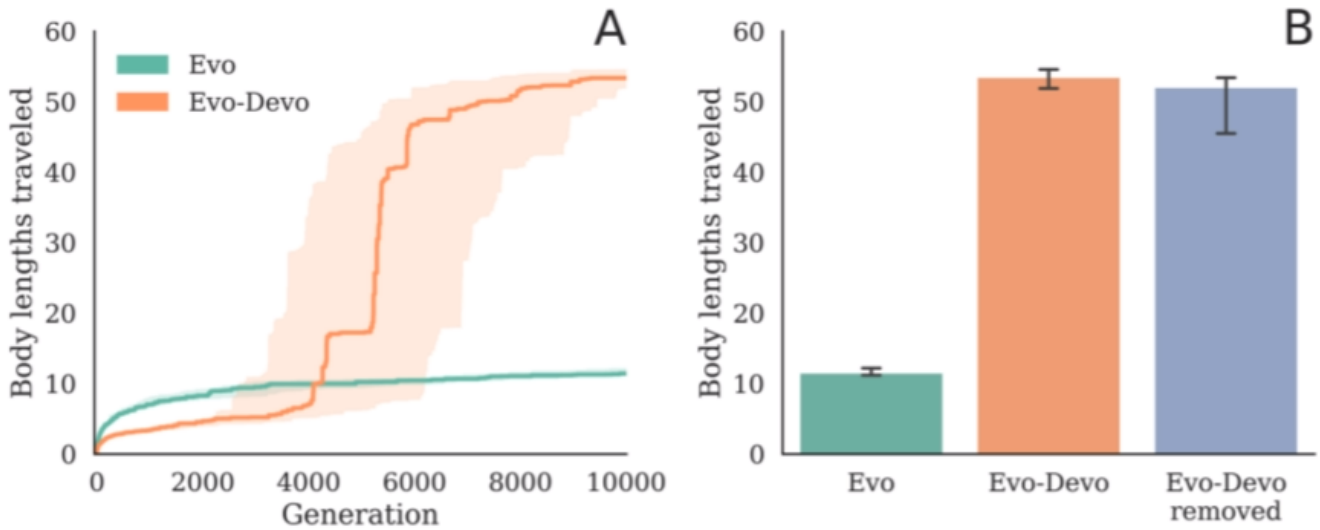
Evo+devo



Results

It works!

- controllers evolved with evo+devo are better even before learning $\rightarrow$ development increase evolvability
- side effect: evo+devo improve robustness to noise on parameters



Evo+devo VSR controllers

https://www.youtube.com/watch?v=Ee2sU-AZWC4

Subsection 5

Evolving the body of a VSR

Evolving the body of a VSR

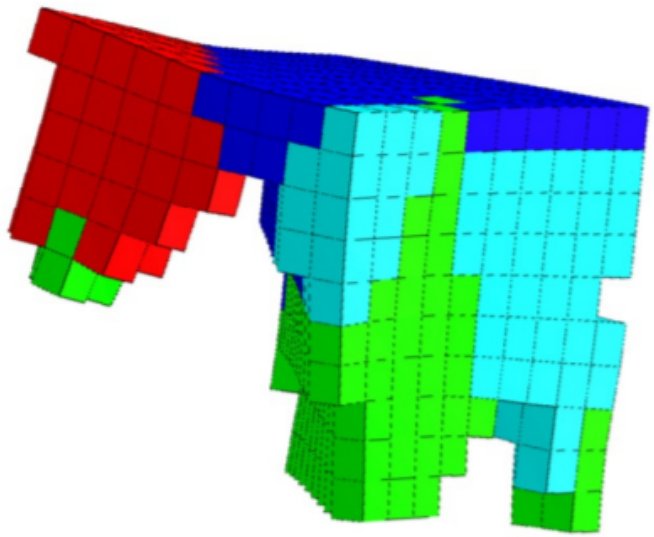
Nick Cheney et al. "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding". In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM. 2013, pp. 167–174

- focus on the representation: a generative encoding is better than a direct encoding
- combine different materials
- complexity in the body, rather than in the brain
- task: locomotion

Representation

Phenotype:

- a connected volume within a $10 \times 10 \times 10$ box of voxels
- voxels can be of different materials



Representation

Genotype?

- direct encoding: fixed length string ($g \in M^{10^3}$); then take just the largest connected volume
- indirect (generative) encoding

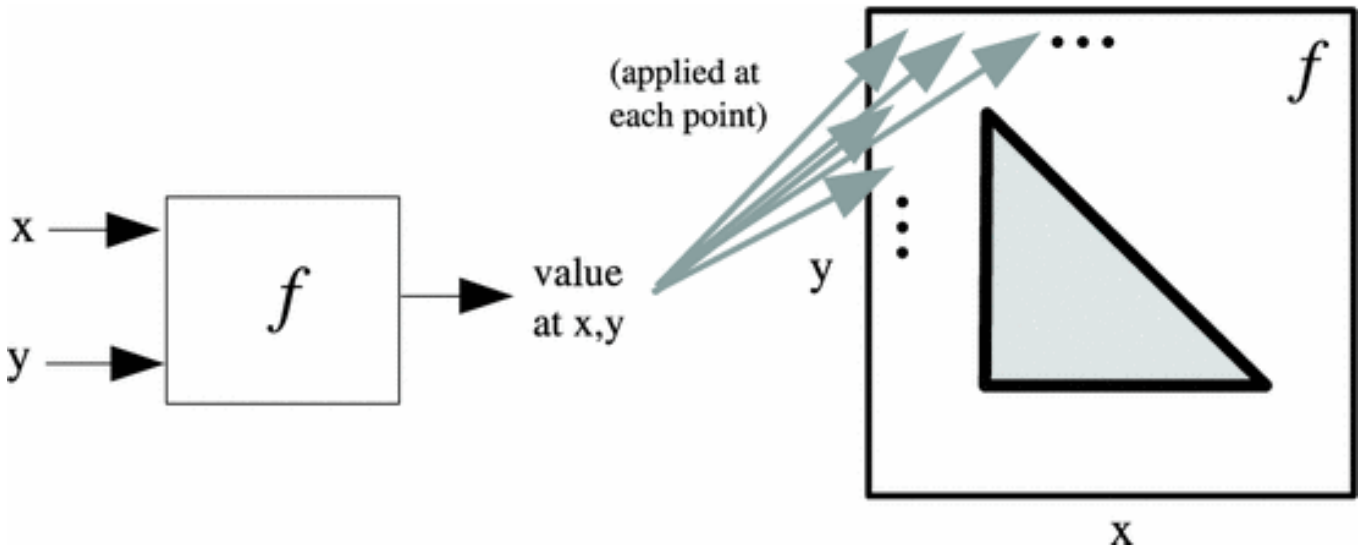
Generative encoding

CPPN: compositional pattern-producing network

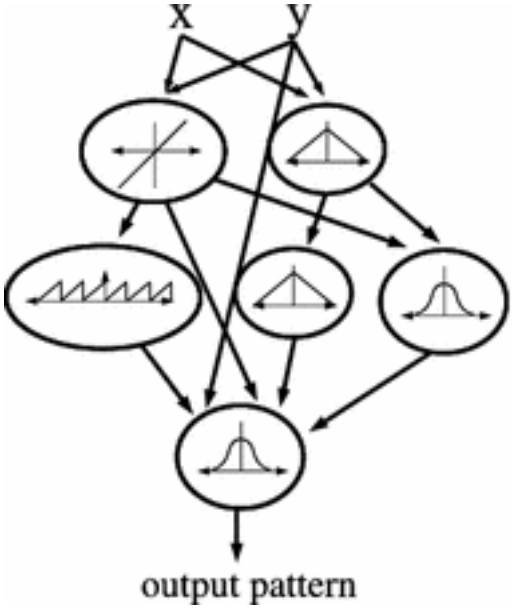
- a network of basic mathematical functions
- a number of inputs equal to the dimensionality of the pattern (here 3)
- one or more (here 2 for the material) output encoding pattern value

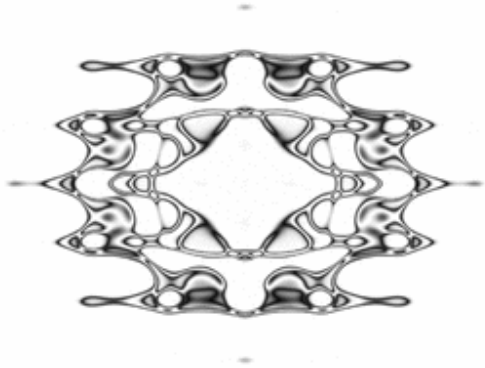
Kenneth O Stanley. "Compositional pattern producing networks: A novel abstraction of development". In: *Genetic programming and evolvable machines* 8.2 (2007), pp. 131–162

CPPN

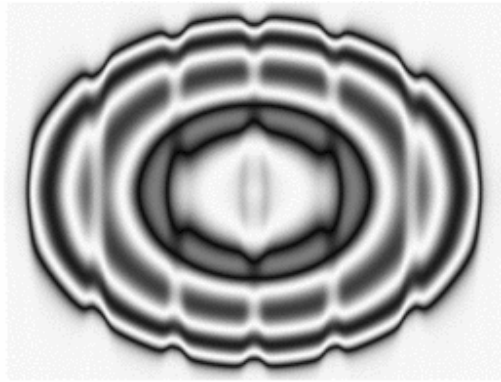


CPPN: properties





(a) Circular cell pattern



(b) Circularity selected for further

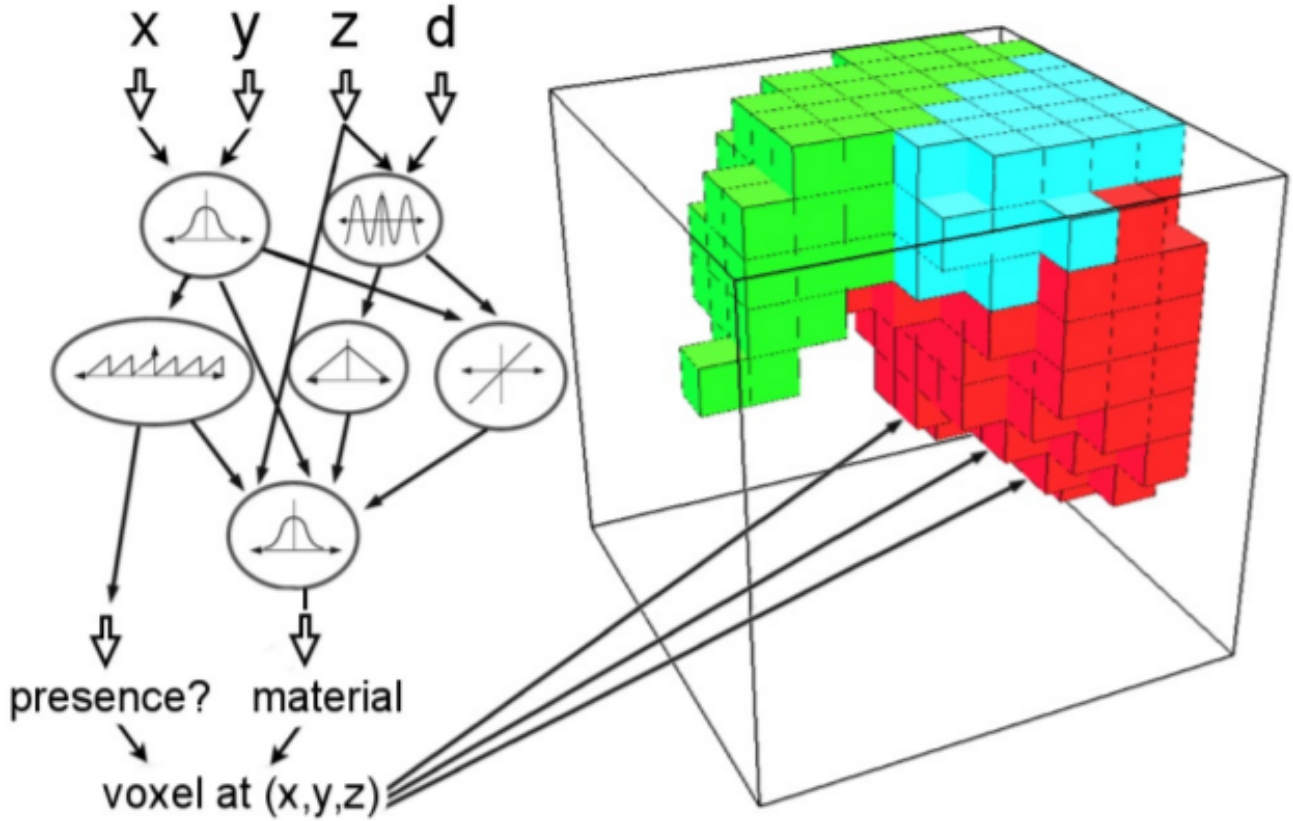
- Patterns resemble nature: repetitions and symmetry
- Work at any scale

How to build a CPPN?

With NEAT!

- node genes encode also the function

VSR representation with CPPN



Controller

Voxel of different materials contract and expands differently:

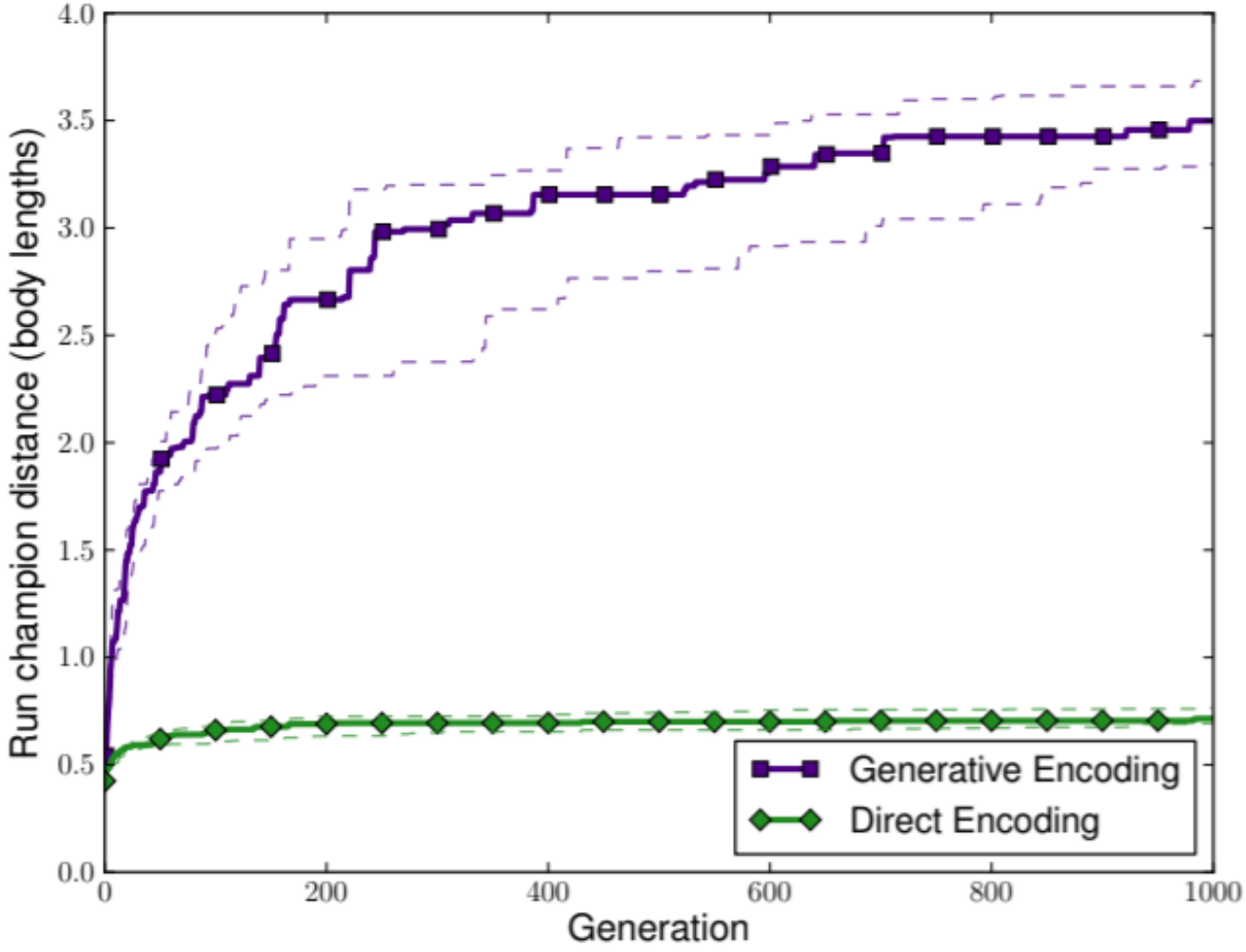
- soft passive tissue
- hard passive tissue
- contract/expand active tissue
- expand/contract active tissue ($\Delta\phi = \pi$)

Controller is simple!

- no feedback from the environment

Results

Indirect vs. direct encoding



Results

Indirect vs. direct encoding

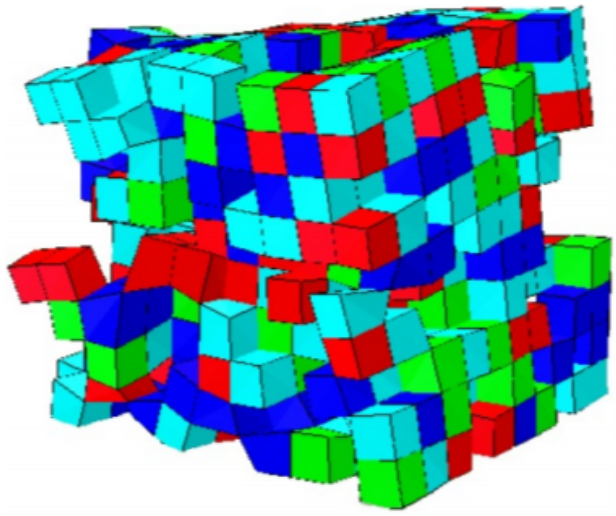
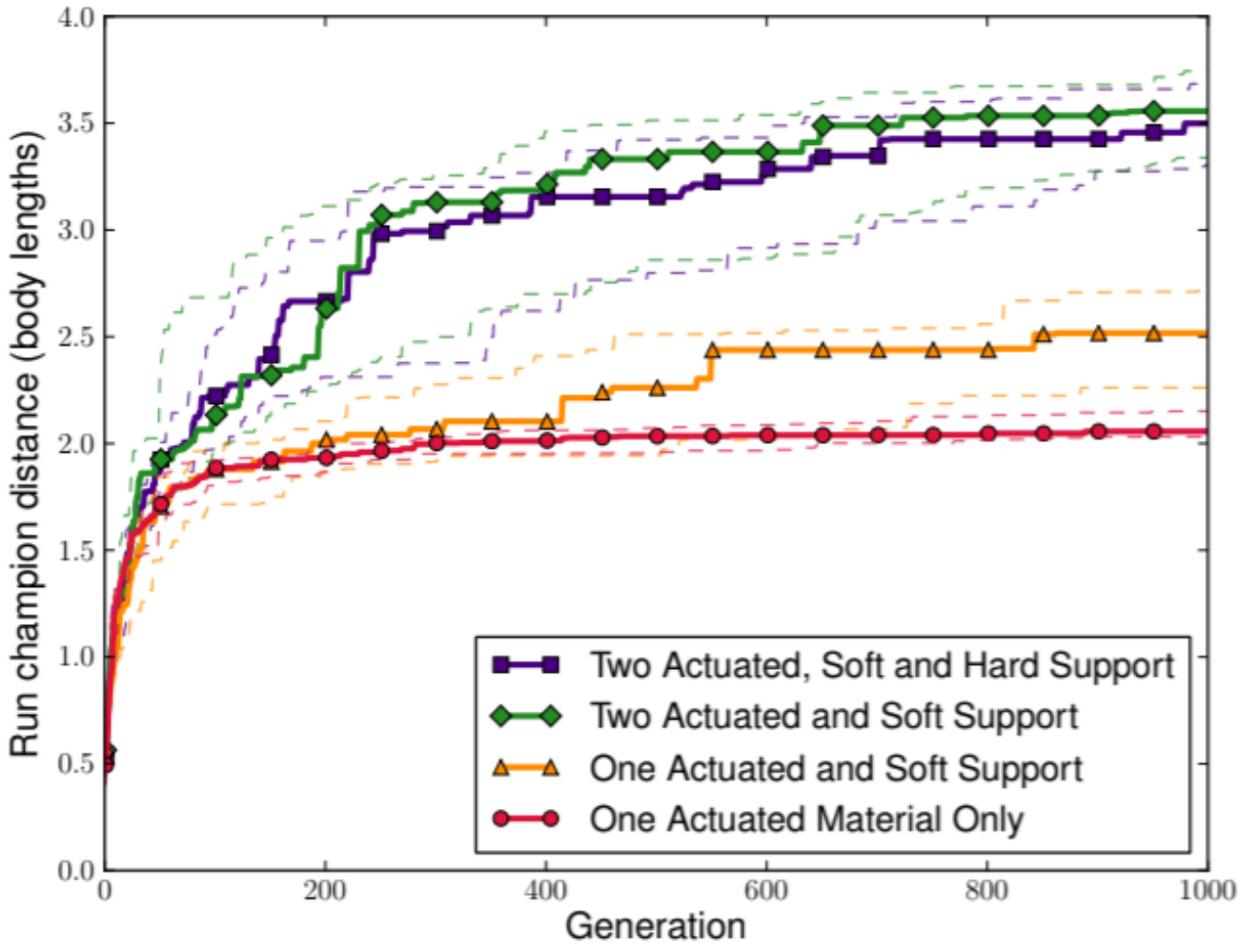


Figure 7: A representative example of a soft robot evolved with a direct encoding. Note the lack of regularity and organization: there are few contiguous, homogeneous patches of one type of voxel. Instead, the organism appears to be composed of randomly distributed voxels . The resolution is the default 10^3.

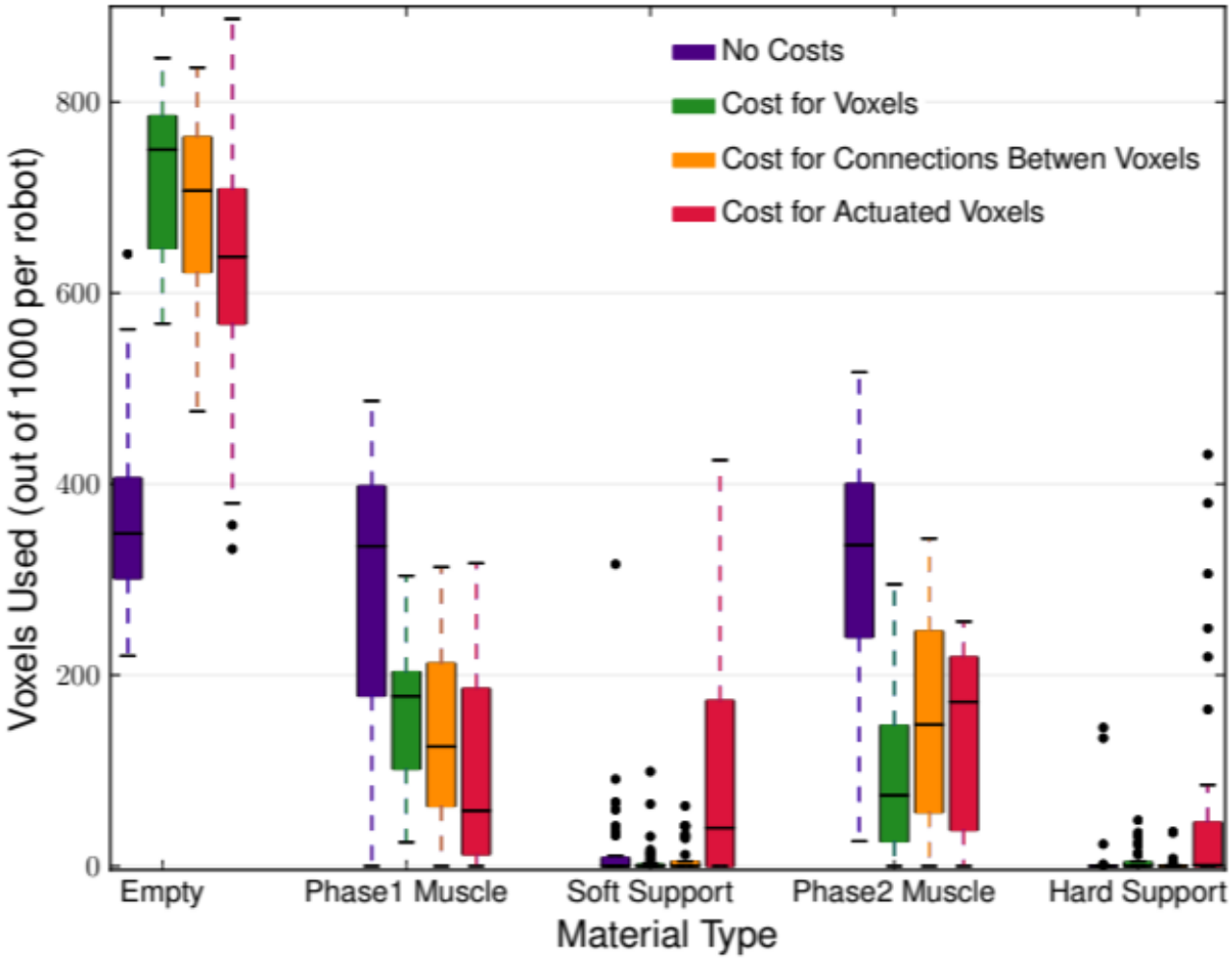
Results

How many materials?



Results

Material-based penalties (act on the fitness)



Results

https://www.youtube.com/watch?v=z9ptOeByLA4

Other similar approaches

Nicholas Cheney, Jeff Clune, and Hod Lipson. "Evolved electrophysiological soft robots". In: *Artificial Life Conference Proceedings 14*. MIT Press. 2014, pp. 222–229

- how to transfer control signals across the body?
- `https://www.youtube.com/watch?v=HgWQ-gPIvt4`

Francesco Corucci et al. "Evolving soft locomotion in aquatic and terrestrial environments: effects of material properties and environmental transitions". In: *Soft robotics* 5.4 (2018), pp. 475–495

- different environment, also during the evolution
- `https://www.youtube.com/watch?v=4ZqdvYrZ3ro`

Subsection 6

Reality gap

Reality gap

Good solutions found in simulations may be inefficient in reality!

- they "exploit badly modeled phenomena to achieve high fitness values with unrealistic behaviors"

Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. "The transferability approach: Crossing the reality gap in evolutionary robotics". In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), pp. 122–145

Transferability

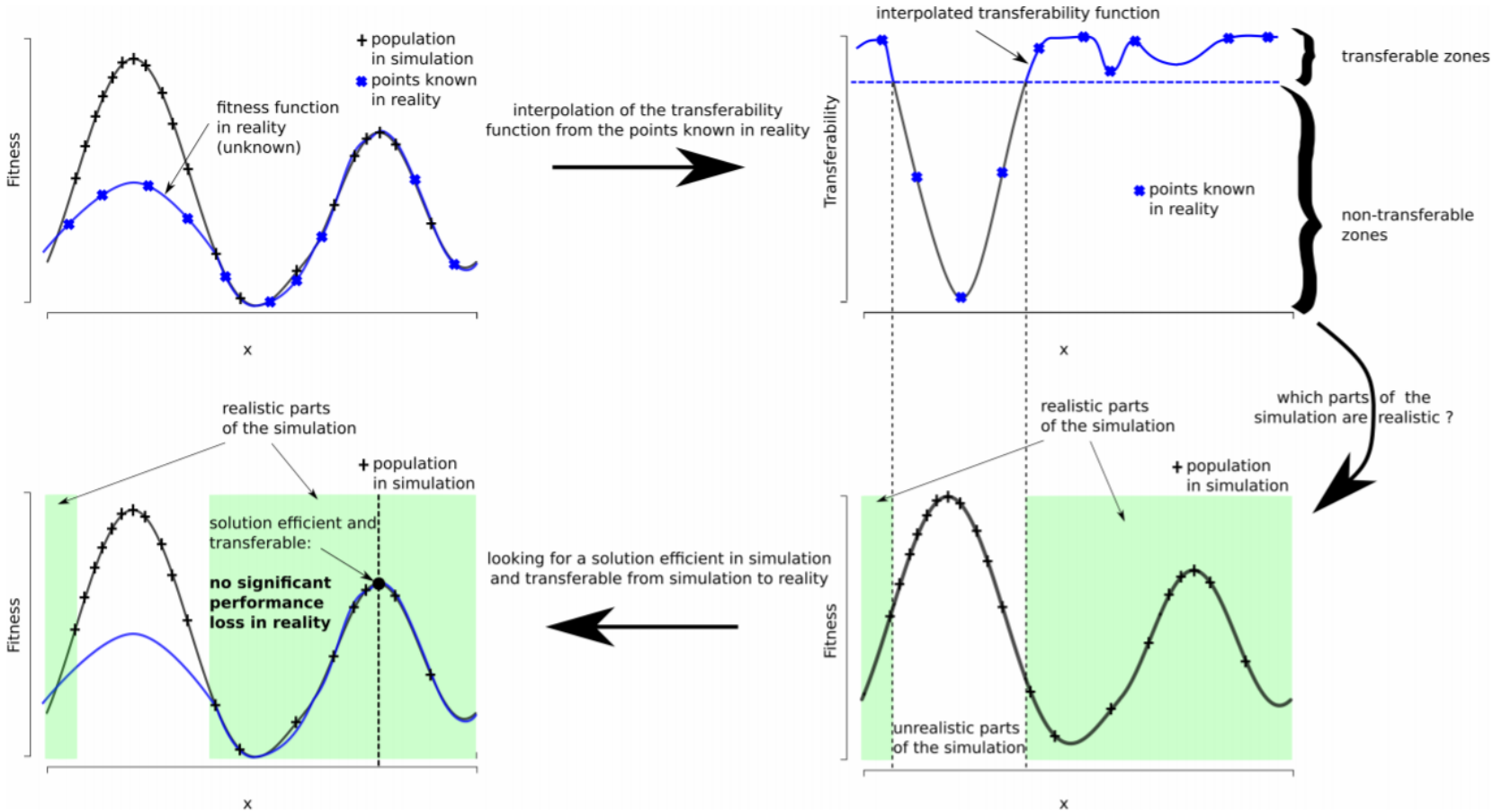
Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. "The transferability approach: Crossing the reality gap in evolutionary robotics". In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), pp. 122–145

Idea: fitness measures

- how well the task is solved
- how easily the controller can be transferred in reality
- Pareto-based optimization

Good, yet unrealistic controllers are not favored

Transferability



Measuring the transferability

In principle:

1. observe the behavior in simulation
2. observe the behavior in reality
3. compute their distance: the larger, the lower the transferability

But:

- requires testing each controller in reality (pointless!)
- how to measure behavior similarity?

Estimating the transferability

Idea:

- sometimes, test a controller in reality
- build an estimator of the transferability and refine it

Still requires a lot of domain knowledge for representing the behavior

Experiment 1: conditioned homing

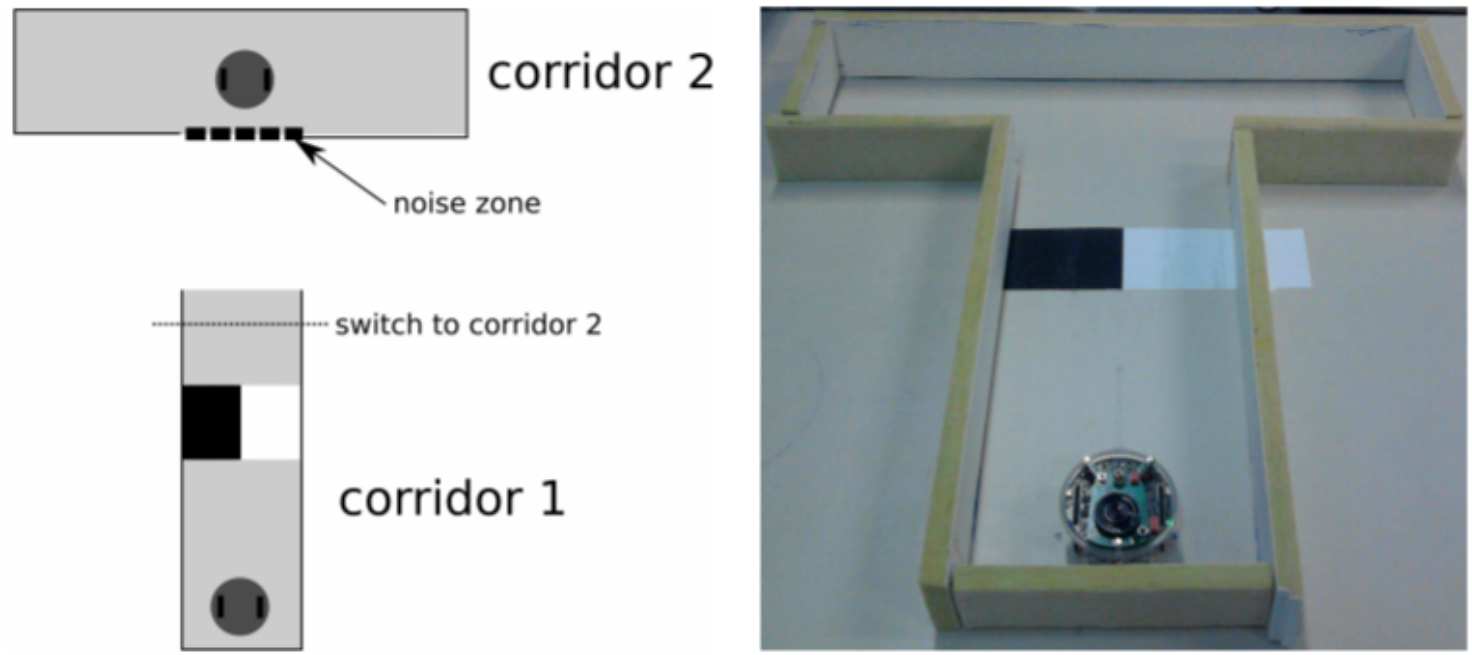
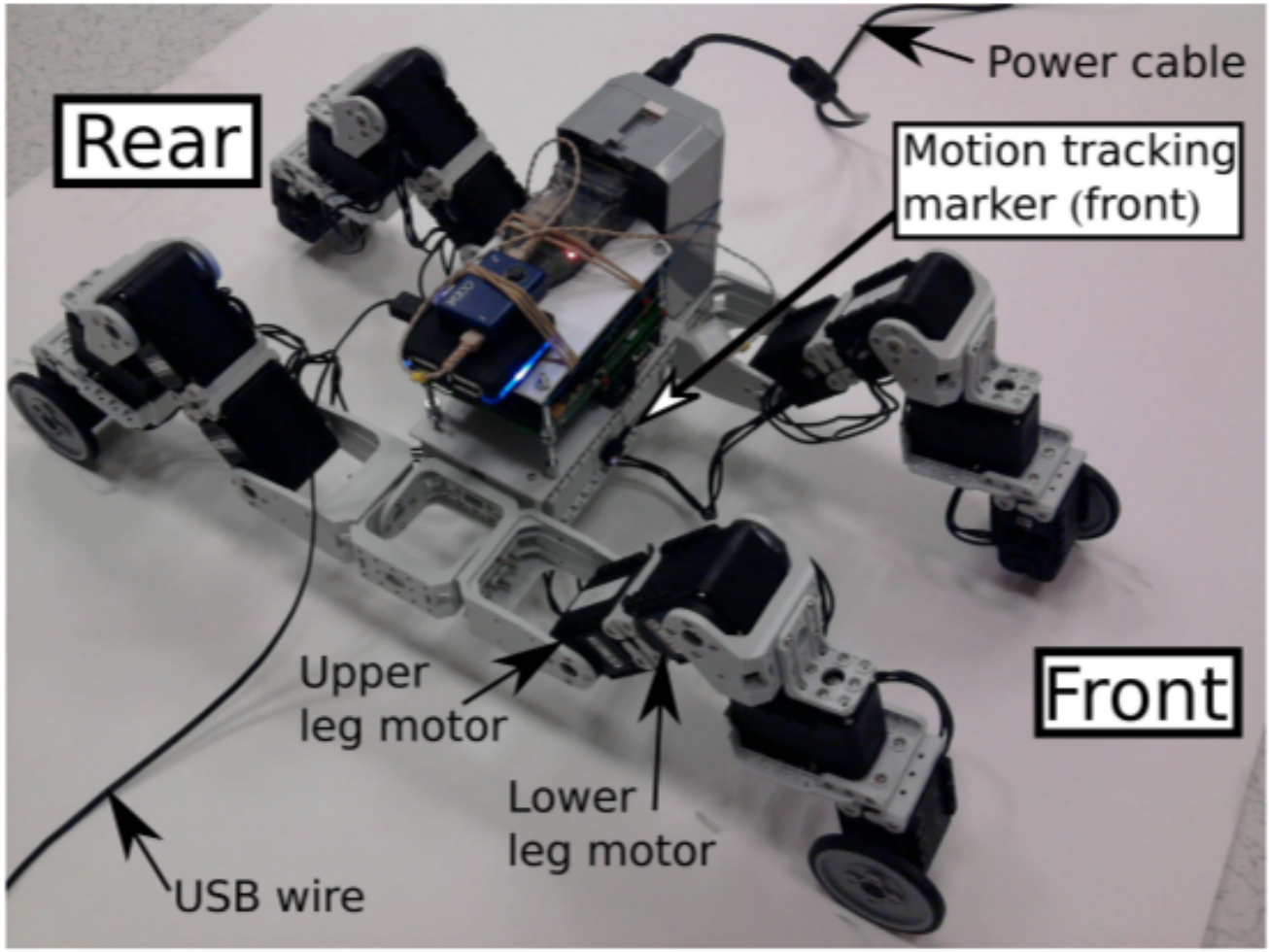


Fig. 9. Left, representation of the T-maze used as minimal simulation; right, photography of the real T-maze with both color patterns

Conditioned homing

- Representation of the controller: fixed simple topology ANN, direct encoding
- Fitness: traveled distance (with bonus if correct target is reached) (optimum is known)
- Behavior representation: robot trajectory

Experiment 1: 8-DOF quadrupedal locomotion



8-DOF quadrupedal locomotion

- Representation of the controller: two control parameters
- Fitness: traveled distance (unknown optimum)
- Behavior representation: robot trajectory (just distance from the starting point)

8-DOF quadrupedal locomotion

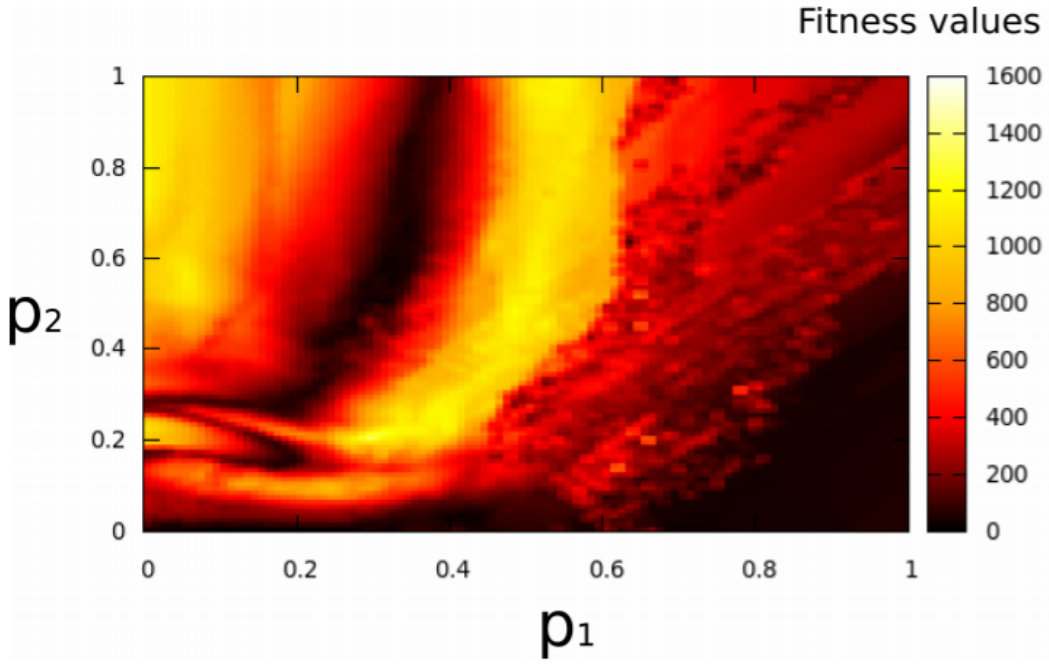


Fig. 14. Fitness landscape in simulation of the application II (covered distance, mm). This picture shows the covered distances in simulation computed for all the possible individuals on the whole control space (about $2.6 \cdot 10^5$ individuals). The control parameters p_1 and p_2 respectively are in x-coordinates and in y-coordinates. The color denotes the strength of the fitness value.