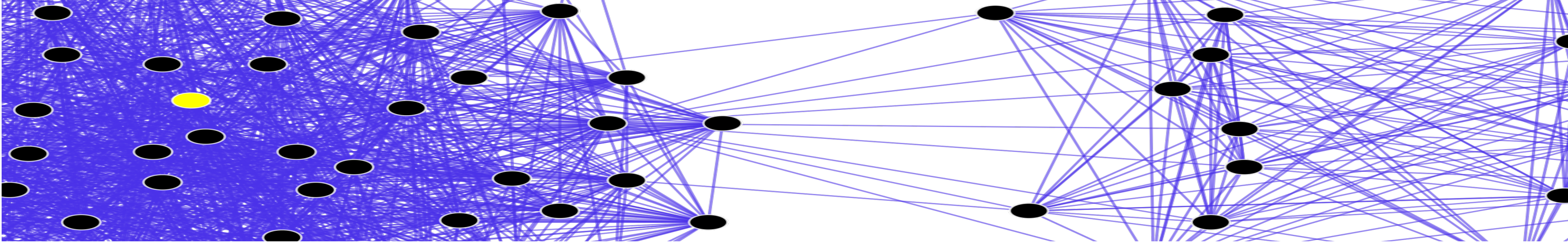


Computational Intelligence

Master in Artificial Intelligence

2019-20

Lluís A. Belanche & René Alquézar

Introduction to Genetic Algorithms

Soft Computing Research Group

Outline of various techniques

- Genetic Algorithms (bitstrings)
- Evolutionary Programming (finite-state automata)
- Evolution Strategies (real-valued vectors)
- Genetic Programming (computer programs)
- Classifier Systems (rules)

Genetic Algorithms

Genetic Algorithms are a class of Evolutionary Algorithms for which:

- Individuals are bitstrings of length *n*
- Strategy is usually (μ , λ) with $\mu = \lambda$
- Selection is usually proportional to fitness
- Mutation is seen as “transcription error” (secondary discovery force)
- Recombination is called “crossover” (primary discovery force)

Genetic Algorithms

In solving a given optimization problem with a GA, the first task would be **devising an appropiate description of a solution** in terms of a **bitstring**:

- How many loci? (chromosome length *n*)
- What exactly are the genes? Do they come in groups?
- Do we allow polyploidy? (dominant/recessive traits)
- Are we using a bijective code (or, at least, injective)?
- Are we using feasible recombination & mutation operators (sound, complete, scalable)?

Selection – first ideas

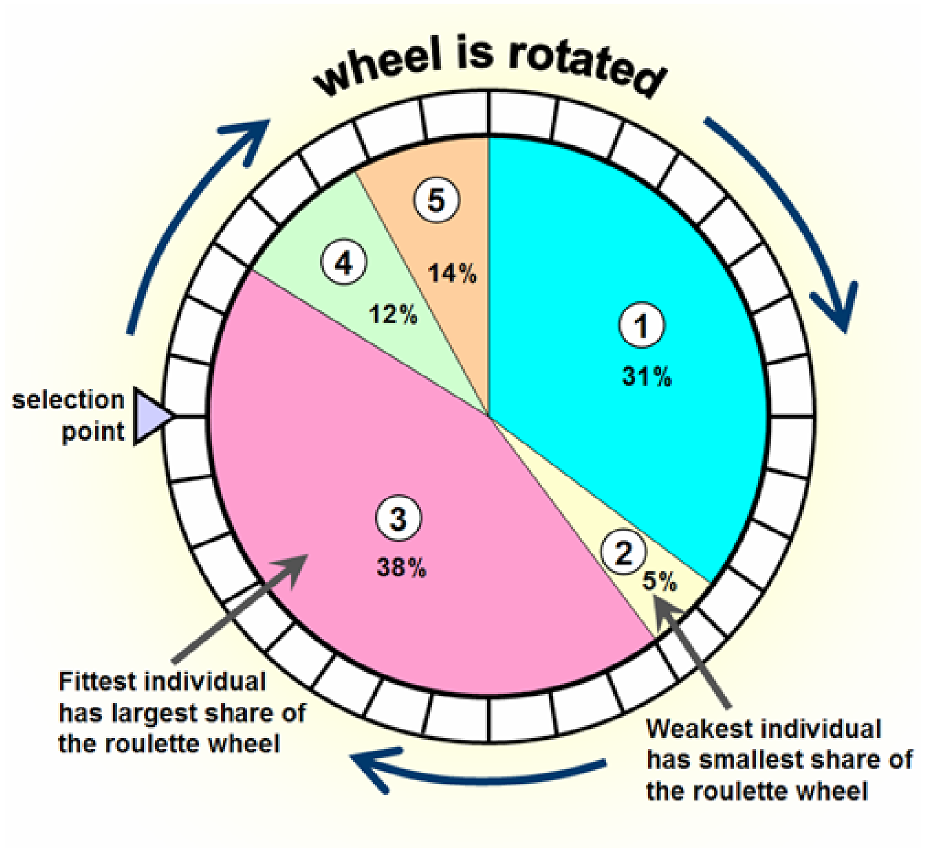
- Selection completely at **random**
  (no benefit for being better than others)
  → little or no **exploitation**

- Always select (only) the **best**
  (very high selective pressure)
  → little or no **exploration**

- **Stochastic** selection (better fitness → higher chance of reproduction)
  (leads to fitness-proportional diversity)
  → more balanced **exploitation-exploration** searches

Selection – The Roulette Wheel

- Each solution gets a region on a roulette wheel according to its fitness
- Spin the wheel, select solution marked by roulette pointer →
- **Stochastic selection** (better fitness = higher chance of reproduction)


wheel is rotated
selection point
5
14%
4
12%
1
31%
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php

Selection – alternatives to Roulette Wheel

Standard Roulette Wheel (SRW) may be very sensitive to the “details” of the fitness function

- Spin wheel once with as many equally-spaced pointers as individuals (*stochastic universal sampling*), or
- Give a sure number of copies for the above-average individuals and perform standard SRW with remainder (*remainder stochastic sampling*), or
- *Rank selection* (probability of selection is proportional to *rank*), which can be used to cope with:
  - High selective pressure in the first few generations
  - High selective pressure due to a decrease in fitness variance as the search proceeds

Selection/Replacement - Elitism

The best A individuals from the last B generations are *kept unchanged* for the next generation

Example:

- keep the best individual of past population (A=1,B=1)
- “unrealistic” but ensures best fitness of a generation never decreases
- entails a decrease in diversity (more EXPLOITATION)
- they can be subject to specific local improvement
    - Hill-climbing
    - Large(r) mutation steps

Selection - Tournament

- Randomly select *k* individuals (with replacement)
- Select the individual with best fitness among these *k* individuals
- **Example**:
  - randomly pick two individuals and keep the best of the two; do this $\mu-1$ times (these parents will then undergo recombination & mutation)
  - use **elitism** (once) to restore population size μ

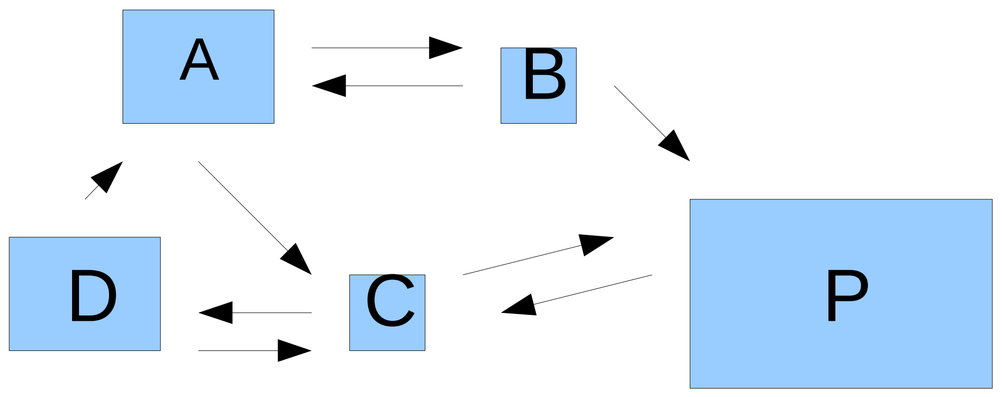
Selection - Niching

Start with a single population P

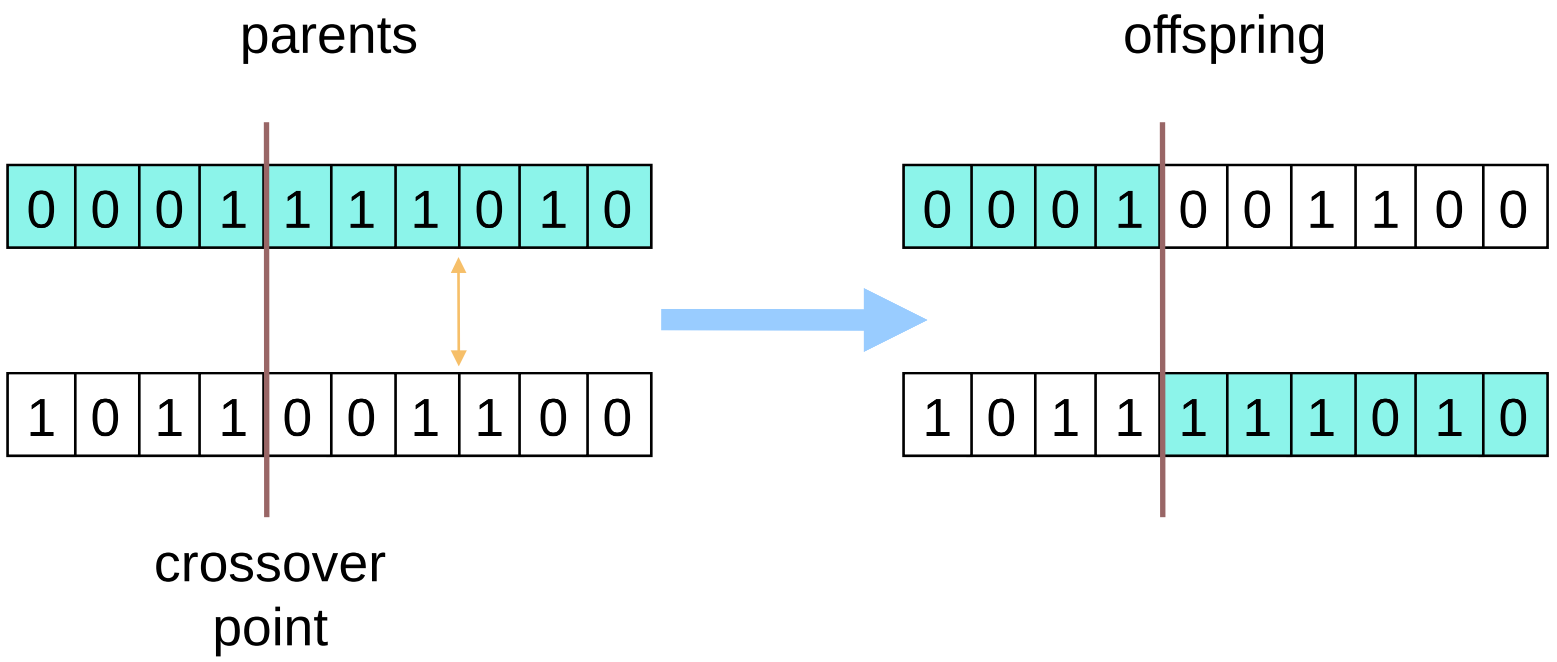
Split it into parts (niches) around promising parts of the search space (A,B,C,D)

Reproduction is within each niche, but

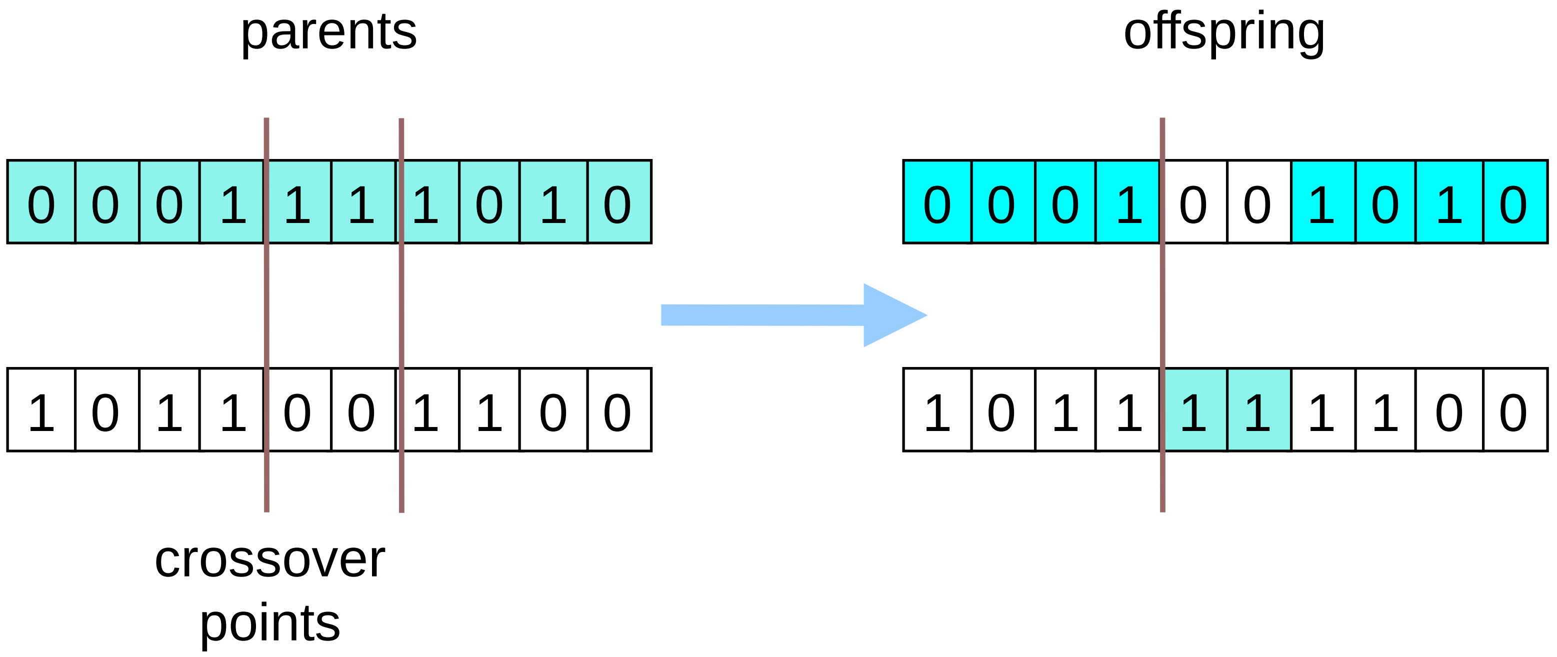
let the parts interchange information and merge back:



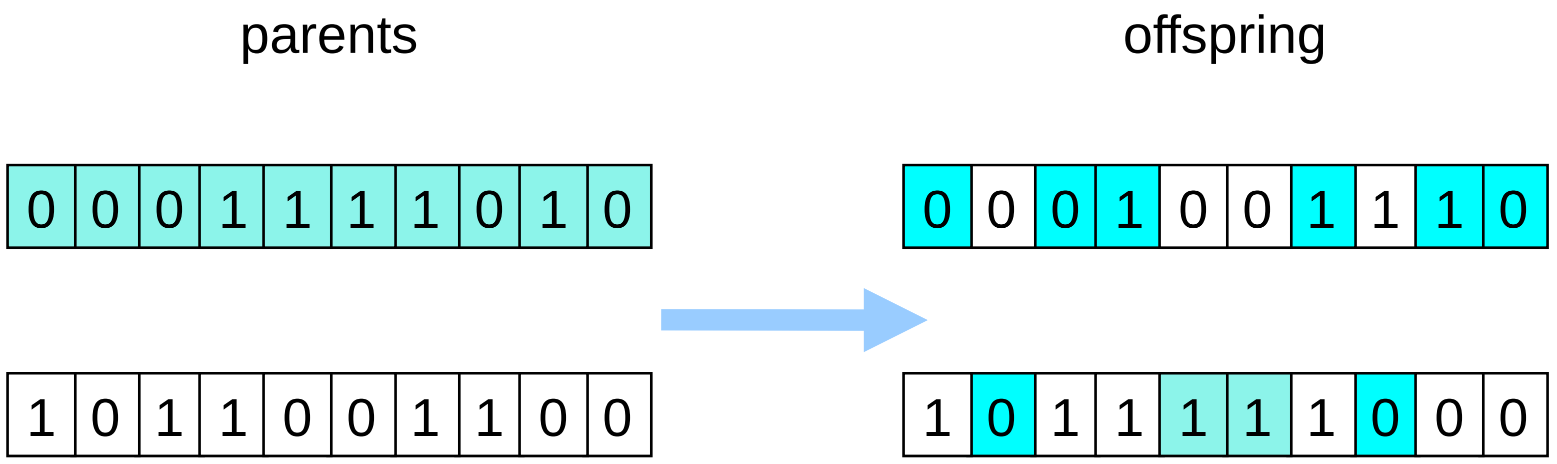
Crossover: One-Point Crossover (1P)



Crossover: Two-Point Crossover (2P)



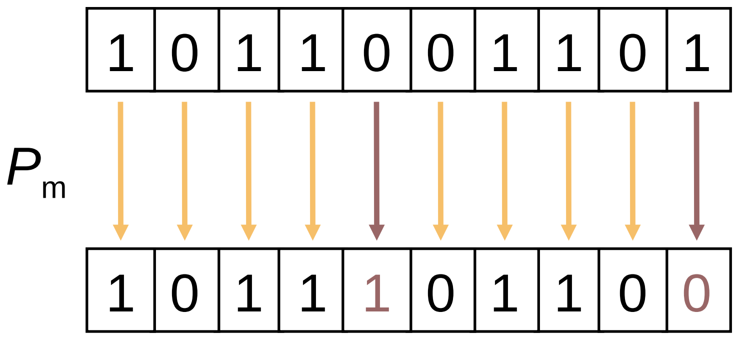
Crossover: Uniform Crossover (UX)



UX evaluates each bit in the parent strings for exchange with a probability of 0.5

HUX (Half Uniform Crossover) is as UX, but exactly half of the non-matching bits are swapped

Mutation



Binomial = independent Bernoulli “transcription errors”

Mutations occur with some small probability every time a chromosome is duplicated

In humans, this probability (the chance that a given nucleotide is mutated in a generation) is around 10^{-8}

Knowing when to stop

- Limit the **number of** generations or fitness evaluations
- Detect **convergence**:
    - No relative improvement in mean/max fitness
    - No change in best
    - Uniformity of fitness evaluations or individuals
    - Fitness evaluation beyond a predefined value
    - Closeness to optimum (if the optimal fitness is known)

Replacement strategies (1)

- **Generational** genetic algortihms (GGA):
    - replace all parents with their offspring, (μ , λ) with $\mu = \lambda$
    - no overlap between populations of different generations
- **Steady-state** genetic algorithms (SSGA):
    - immediately after offspring is created and mutated, it is used to replace some parents in the old generation
    - some overlap exists between populations of different generations

Replacement strategies (2)

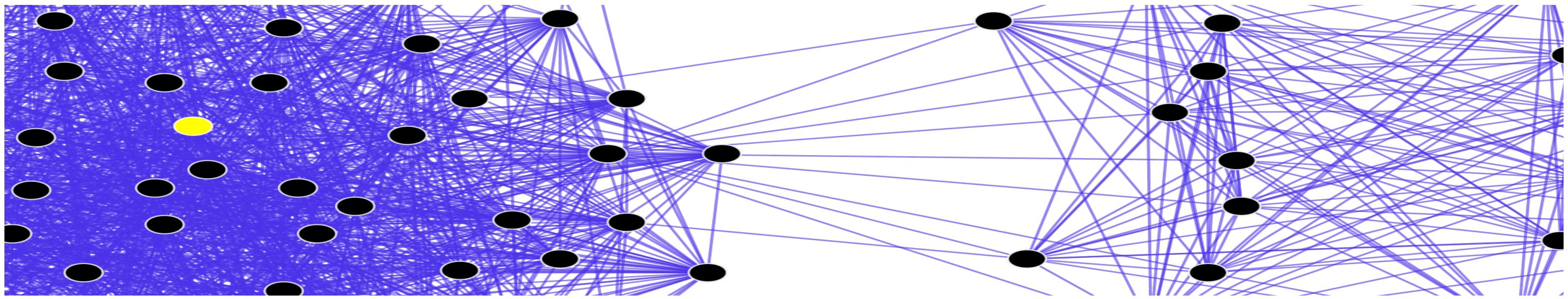
- Replacement strategies for SSGA:
  - the offspring replace the **worst** individuals of the current population (elitism)
  - the offspring replace **random** individuals of the current population
  - the individuals to be replaced are selected using «reverse» **tournament selection** (now for the worst one)
  - the offspring replace the **oldest** individuals of the current population

Schools of "thought"

In GAs, originally there were two different views of what a population represents:

- The **Michigan** view: individuals represent parts of the solution; the whole population represents a full solution
- The **Pittsburgh** view: each individual represents a full solution (this view has prevailed and is the common approach)

- **Example**: evolving neural networks



Binary Representations

1010101110011110100000101000100110101001010

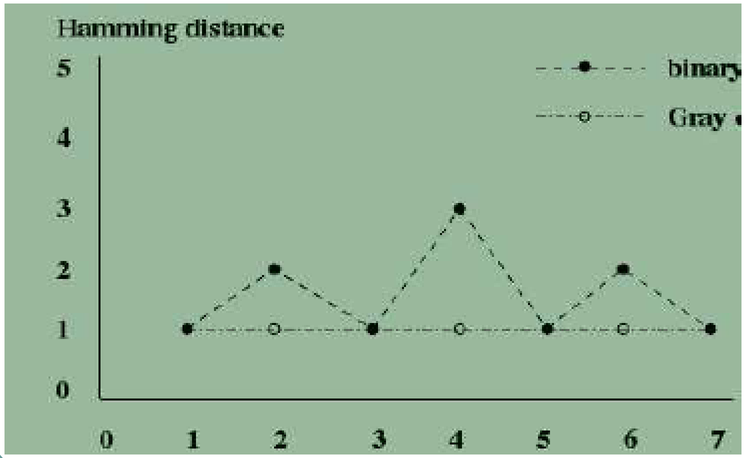
The (variables in the) optimization problem may:

- admit a natural binary representation (boolean variables)
- be given by nominal-valued variables (e.g. type of product)
- be given by a discrete structure (tree, graph, …)
- be given by integer or ordinal-valued variables
- be given by continuous information (real numbers)

Binary Representations

Problems using a binary representation:

- maximum attainable precision may be less than needed to represent the optimum (for real numbers)
- introduction of «Hamming cliffs»



Hamming cliff is formed when two numerically adjacent values have bit representations that are far apart

$7_{10} = 0111_2$ vs $8_{10} = 1000_2$

Large change in solution is needed for small change in fitness

Gray Coding

In a Gray coding, the Hamming distance between the *representation* of consecutive numerical values is 1

Example for n=3:

	Binary	*Gray*
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Converting binary strings to Gray bit strings:

$$\begin{aligned} g_1 &= b_1 \\ g_l &= b_{l-1}\overline{b}_l + \overline{b}_{l-1}b_l \end{aligned}$$

where b_l is bit l of the binary number

$$b_1 b_2 \cdots b_{n_b}$$

How a GA *might* work: schemata theory

A **schema** S is a template describing a subset of individuals:

*	*	*	1	0	*	1	*	*	*

Don't care symbol: *

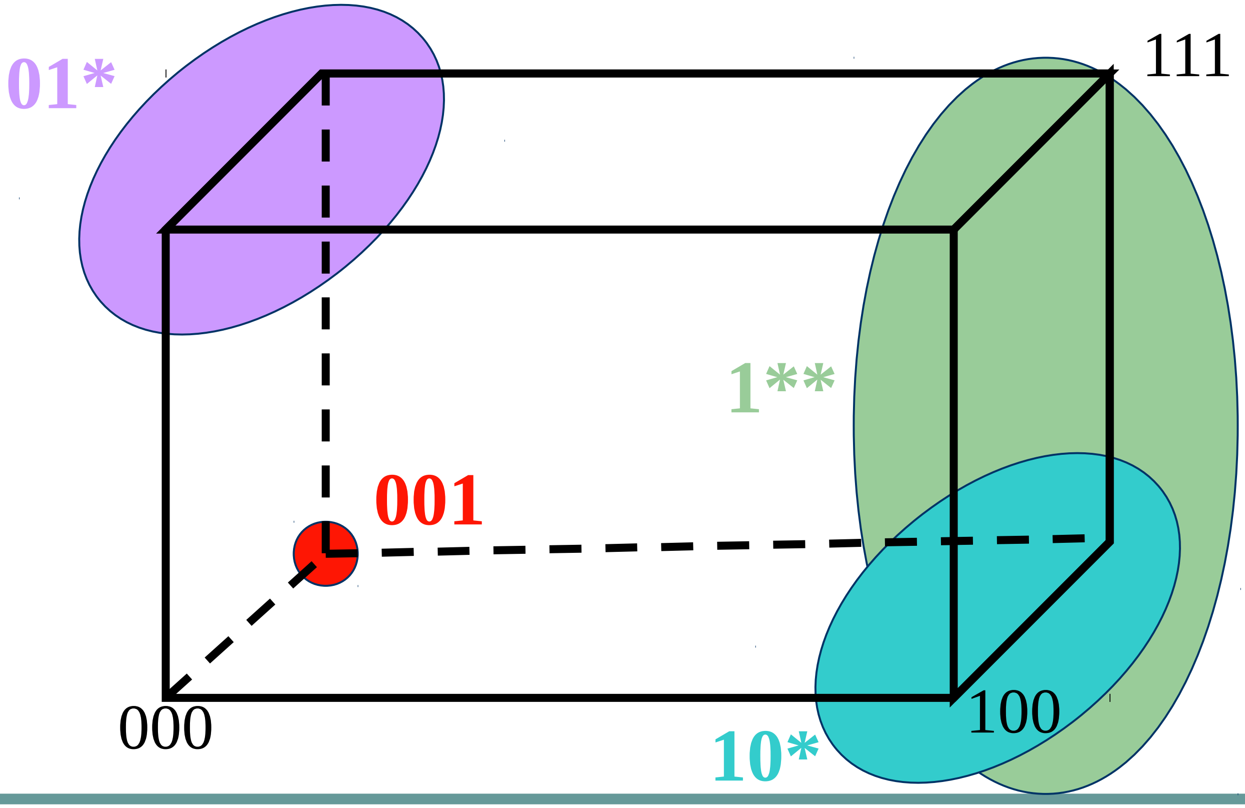
order $o(S)$ = # fixed positions

defining length $\delta(S)$ = distance between first and last fixed positions

- there are 3^n possible schemata
- there are (δ over o)$\cdot 2^o$ schemata of order o and defining length δ
- a schema S includes (represents) $2^{n - o(S)}$ individuals
- an individual of length n belongs to (i.e. represents) 2^n schemata
  (this result does not depend on the alphabet being binary)

Schemata as hyperplanes

A **schema** (*pl.* schemata) is a string in the ternary alphabet {0,1,*} representing a hyperplane in the search space



Schemata interpretations

- **Geometric** view: hyperplanes in $\{0,1\}^n$
- **Evolutionary** view: what survives is not the individuals, but their genetic stuff (the schemata)

Thus the GA is a **schema processor**

The Schema Theorem

Theorem. Short, **low-order**, **above-average** schemata receive **exponentially increasing** trials (individuals represented by the schema) in subsequent generations

Schema theory is (generally) accepted to be (essentially) correct, providing an intuitive explanation for the good (and bad) performance observed in practical GA applications

Theoretical work on GAs includes:

- Markov chains as an alternative formalism
- Synthetic search problems (e.g. Walsh functions)
- Studies on (kinds of) deceptive problems

The Building Blocks Hypothesis

Closely related to the Schema Theorem is the “Building Block Hypothesis”:

“Genetic Algorithms work by discovering and exploiting **building blocks** --groups of closely interacting genes-- and then combining these blocks (via crossover) to produce successively larger blocks until the problem is solved.”

D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Boston, MA, USA 1989.

Deception

A problem is said to be **deceptive** if the building blocks identified actually lead the GA away from the global objective.

f(000) = 28	f(001) = 26
f(010) = 22	f(100) = 14
f(110) = 0	f(011) = 0
f(101) = 0	f(111) = 30

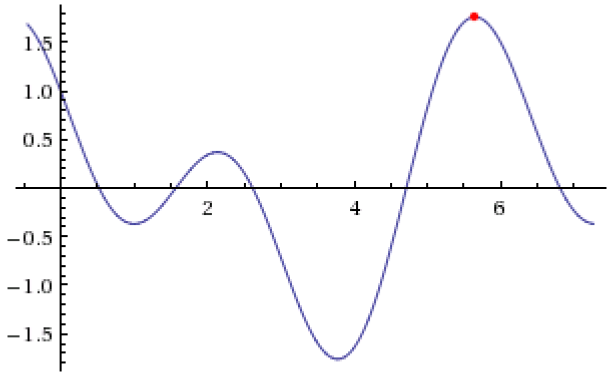
F(0**) > f(1**)	F(00*) > f(11*), f(01*), f(10*)
F(*0*) > f(*1*)	F(0*0) > f(1*1), f(0*1), f(1*0)
F(**0) > f(**1)	F(*00) > f(*11), f(*01), f(*10)

(global optimum is 111; 000 is suboptimal)

Example: Function optimisation (1)

Let f(x) = cos(x) – sin(2x)

(to be maximised in [0, 2π])

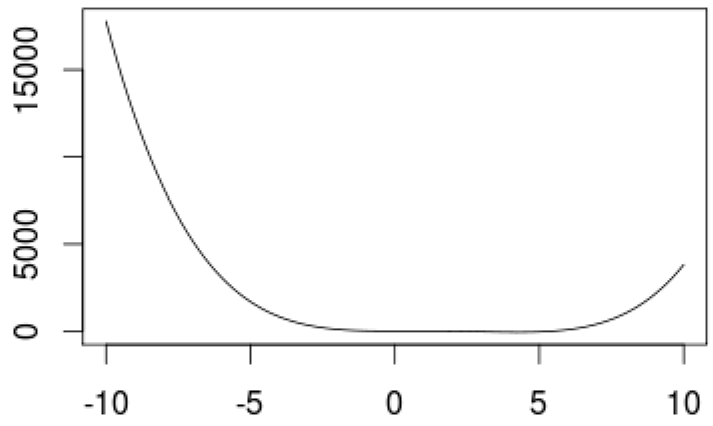


Example: Function optimisation (2)

Finding the roots of a polynomial p(x) in an interval

Fitness function: maximize $F(x) = -p(x) \cdot p(x)$

Example: $p(x) = x^4 - 7x^3 + 8x^2 + 2x - 1$ in [0,10]

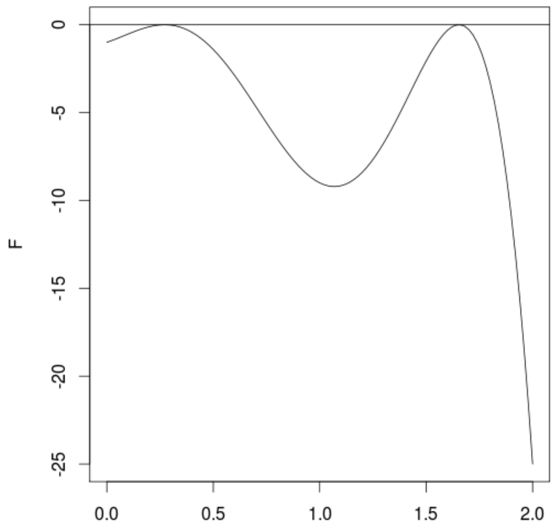


Example: Function optimisation (3)

Most methods need a **range** for the possible values

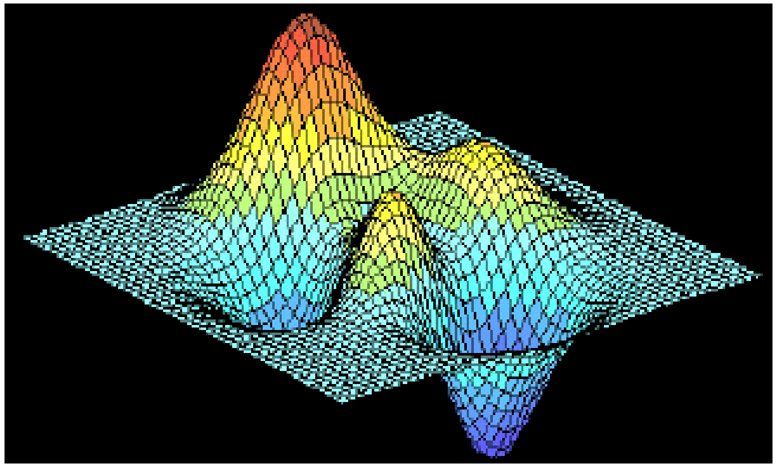
Example: $p(x) = x^4-7x^3+8x^2+2x-1$ in [0,10] can be constrained to [1/9, 9]:

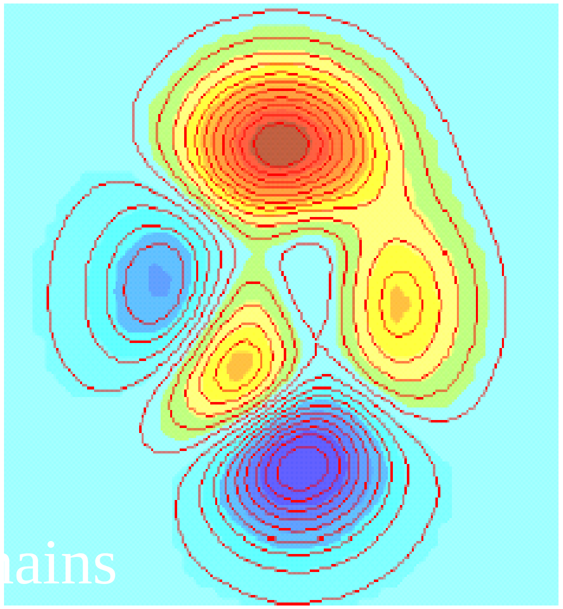
- $x_1 \approx 0.27190$
- $x_2 \approx 1.65434$



Example: Function optimisation (4)

- z = f(x, y) = 3*(1-x)^2*exp(-(x^2) - (y+1)^2) - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) -1/3*exp(-(x+1)^2 - y^2).

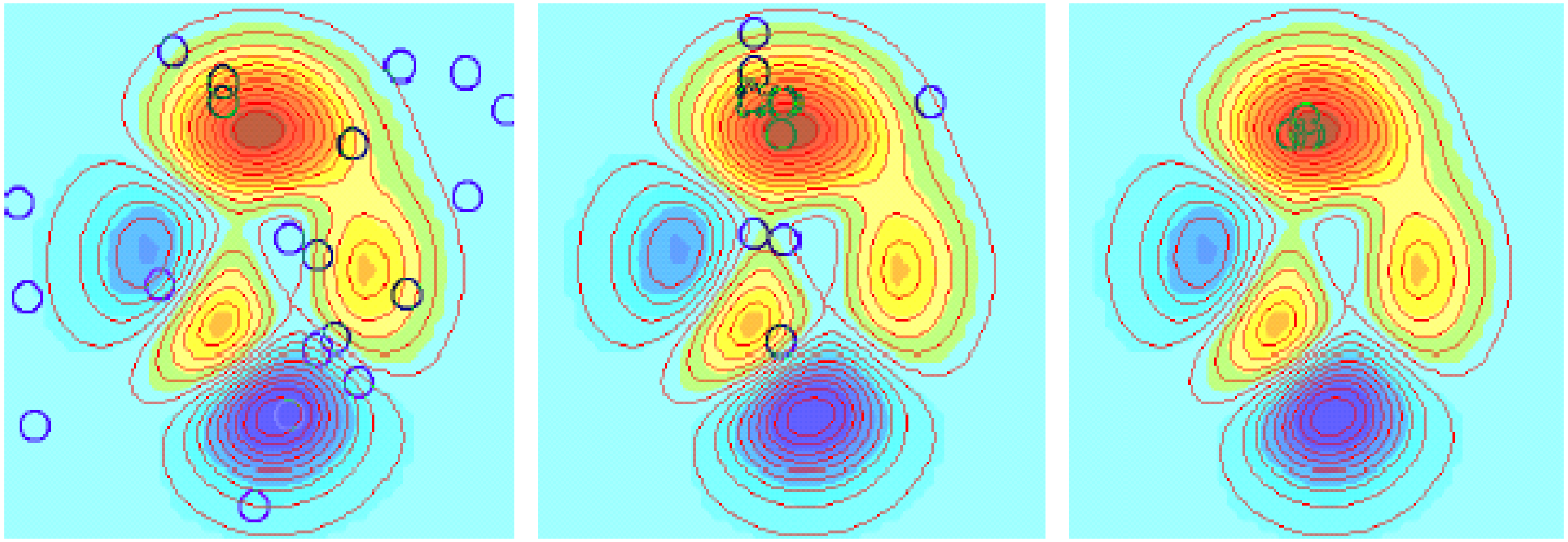




ains

Example: Function optimisation (5)

- GA process:


Initial population 5th generation 10th generation

Example: the (iterated) Prisoner's Dilemma

- The 'Prisoner's dilemma game was invented by Merrill Flood & Melvin Dresher in the 50s
- Albert Tucker formalized the game with prison sentence rewards and gave it the name "prisoner's dilemma"
- Studied in game theory, economics, political science, cold war
- The story
  - Paula and Bruno arrested, no communication between them
  - They are offered a deal:
    - If only **one** person accuses the other then he/she gets suspended sentence while the other gets 5 years in prison
    - If **both** confess & testify against the other, they both get 4 years
    - If **none** of them confesses then they both get 2 years
  - What is the best strategy for maximising one's own payoff?

Example: the (iterated) Prisoner's Dilemma

- Humans play against each other with the goal of maximizing one's payoff in the long run
- A winner strategy is TIT-FOR-TAT*:
  - Cooperates as long as the other player does not defect
  - Defects on defection until the other player begins to cooperate again
  - Can a GA discover this strategy?
  - Can a GA evolve a better strategy?

(*) actions done intentionally to punish other people because they have done something unpleasant to you (http://dictionary.cambridge.org)

Example: the (iterated) Prisoner's Dilemma

Define a representation of cases:

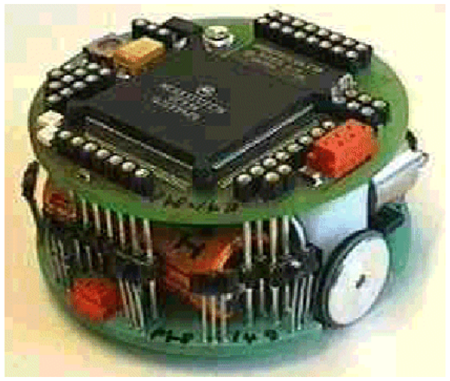
Case 1: CC	C	
Case 2: CD	D	$\longleftarrow$ encoding of TIT FOR TAT
Case 3: DC	C	
Case 4: DD	D	

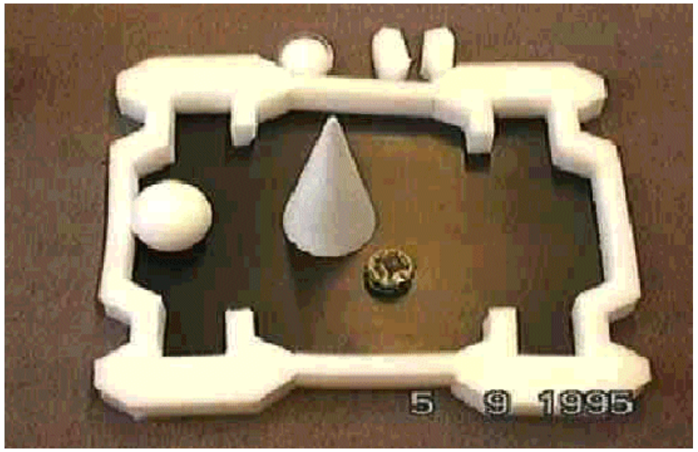
- Results
  - For a environment formed by 8 human-designed strategies, a GA found a better strategy (i.e., highly adapted to *that* environment)

    [memory of 3 previous games, 40 runs, 50 generations each, population of 20, fitness: average score over all games played]
  - Experiments in *changing* environments: the evolving strategies played against each other: found strategies similar to the winner human-designed strategies
  - Idealized model of evolution & co-evolution

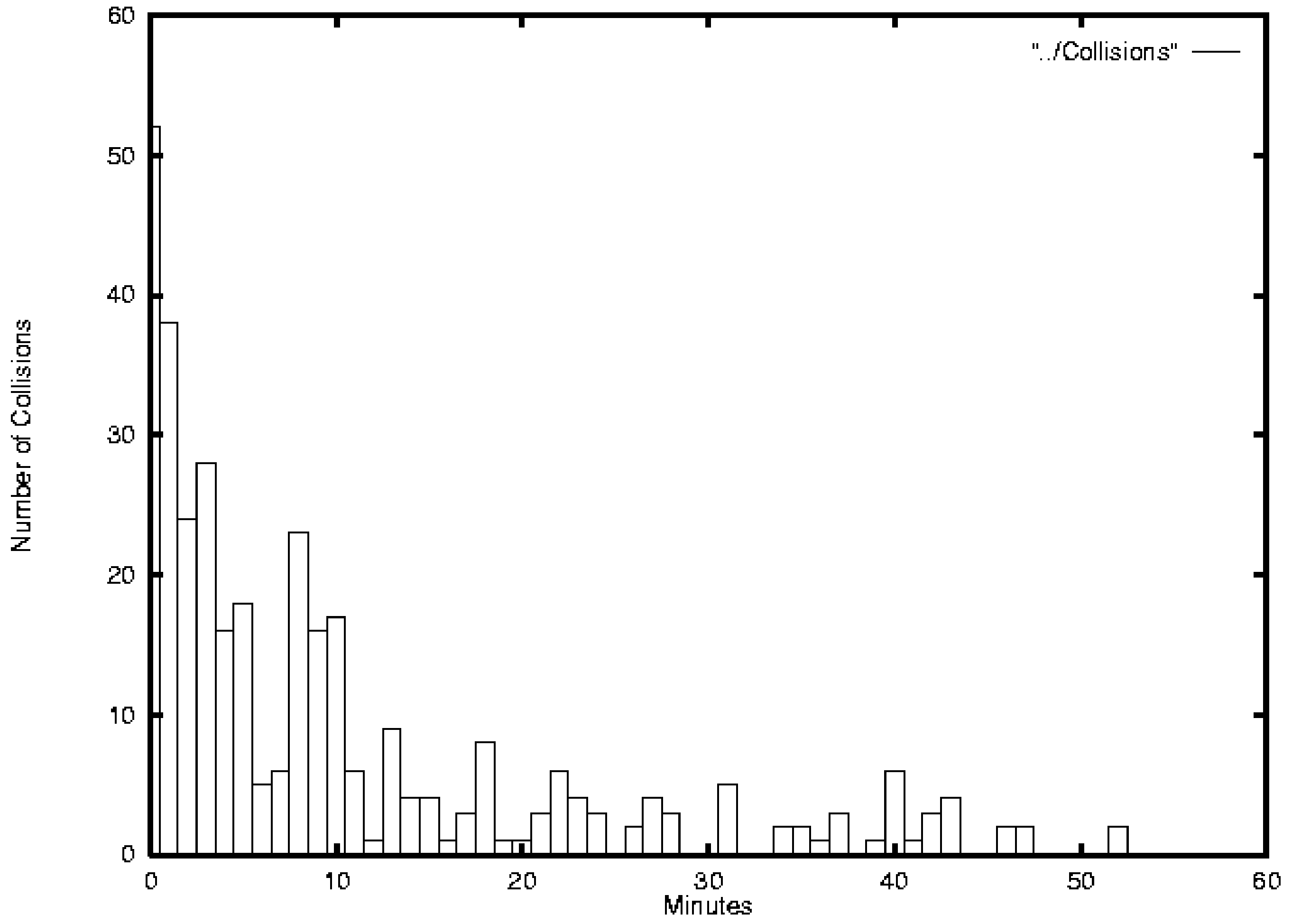
Example: Evolving programs for a small mobile robot

- **Goal**: obstacle avoidance
- Inputs from eight sensors on robot $\{s_1,\ldots,s_8\}$ with values in $\{0,\ldots,1023\}$ (higher values mean closer obstacle)
- Output to two motors (speeds) with values in $\{0,\ldots,15\}$

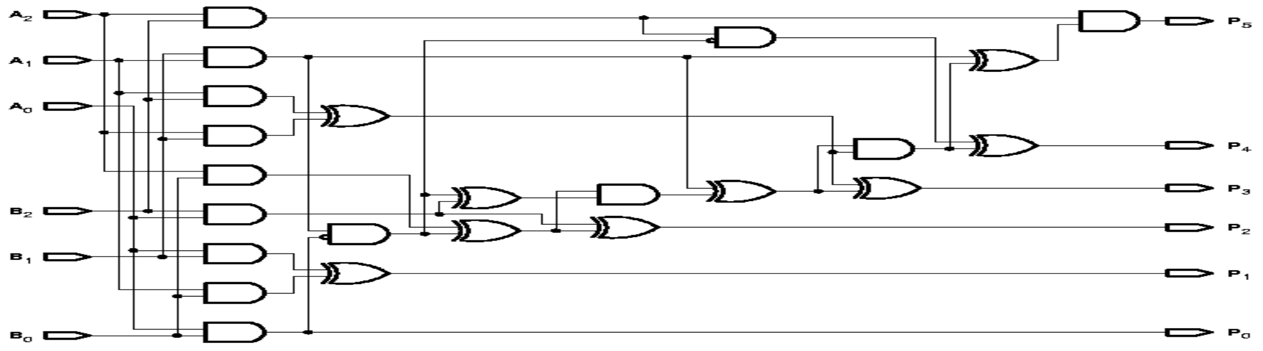




Results



Example: Electronic circuit design



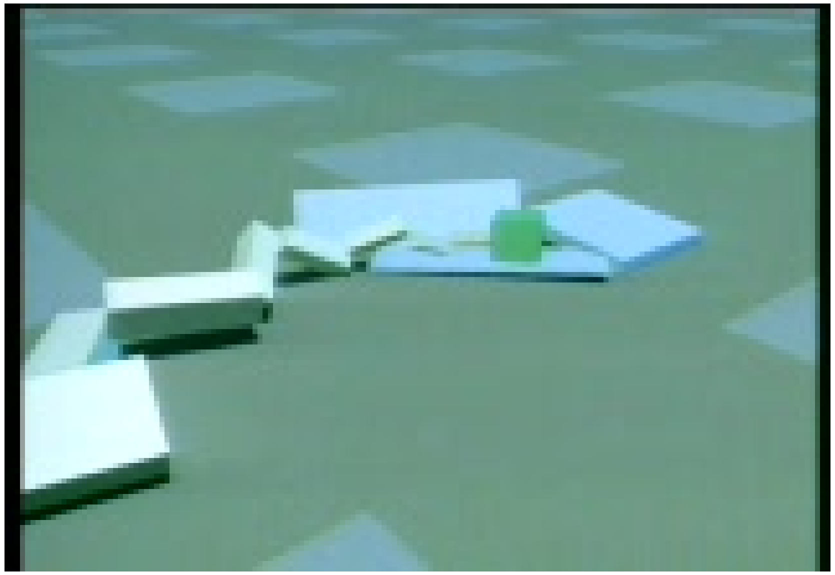
• Individuals are programs that transform an initial circuit to a final circuit by adding/subtracting components and connections

• Fitness: computed by simulating the circuit

• Population of 640,000 individuals run on a parallel processor

• After only 137 generations, some discovered circuits exhibit performance competitive with best human designs

Example: Karl Sims' virtual creatures



http://www.karlsims.com/evolved-virtual-creatures.html

Key ideas (1)

- The encoding should <u>respect the schemata</u>: it must allow discovery of small building blocks from which larger, more complete solutions can be formed
- The encoding should <u>reflect functional interactions,</u> as proximity on the genome (*linkage bias*)
- You should devise <u>appropriate genetic operators:</u>
  - all individuals correspond to feasible solutions and vice versa
  - genetic operators preserve feasibility

Key ideas (2)

- High flexibility and adaptability because of many options:
  - Problem representation
  - Genetic operators with parameters
  - Mechanism of selection
  - Size of the population
  - Fitness function
- These decisions are highly problem-dependent
- Parameters are not independent: you cannot optimize them one by one
- Parameters can be adaptable, e.g. high in the beginning (more exploration), going down (more exploitation), or even be subject to **evolution** themselves!