Evolutionary Algorithms

CPSC 472/572 – Intelligent Robotics

Brian Scassellati

The Problem

- How do we design a system when we have little idea of where to start?
- How can we do as little work as possible?

Random Selection

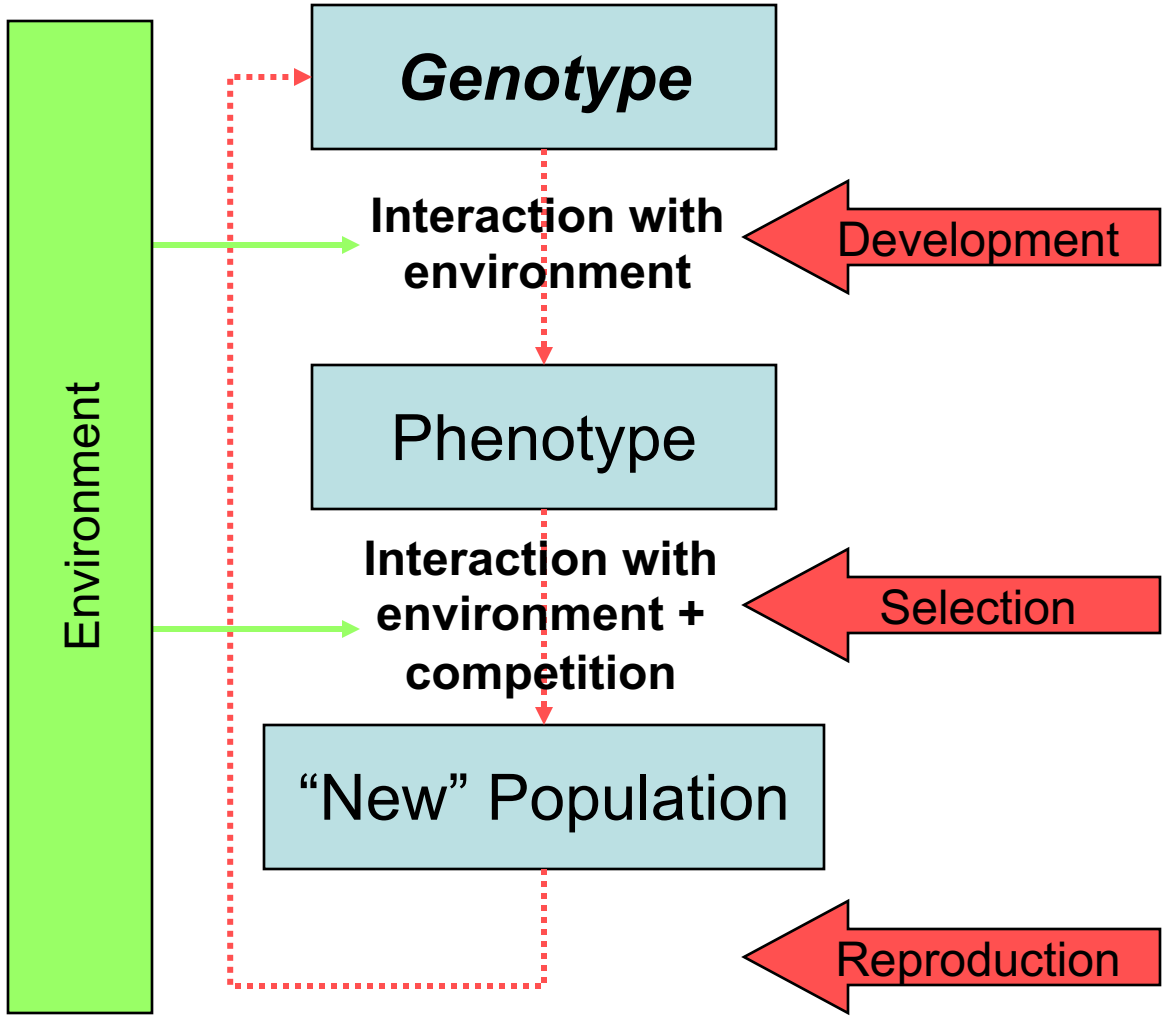
- A random sequence is selected at each instance
- Partial solutions are possible
- But will disappear in the next iteration
- Eventually something useful will result
- $27^{20} \approx 4 \times 10^{28}$ possible strings in even this simple example
- At one sample per second, that's $>10^{21}$ years

The Solution?

- Inspired by evolution
- Start with a set of solutions (represented by chromosomes) called a population.
- Select new solutions (offspring) based on their fitness.
- Repeat until satisfied.

Overview of the Process

1. ***[Start]***
2. [Fitness]
3. [New population]
   A. [Selection]
   B. [Crossover]
   C. [Mutation]
   D. [Accepting]
4. [Replace]
5. [Test]
6. [Loop] Go to step 2



Outline of the Basic Genetic Algorithm

1. [Start] Generate random population of *n* chromosomes.
2. [Fitness] Evaluate the fitness of each chromosome.
3. [New population] Create a new population by repeating:
   A. [Selection] Select two parent chromosomes based on their fitness.
   B. [Crossover] With a crossover probability cross over the parents to form new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
   C. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
   D. [Accepting] Place new offspring in a new population.
4. [Replace] Use new generated population for a further run of algorithm.
5. [Test] If the end condition is satisfied, **stop**, and return the best solution in current population.
6. [Loop] Go to step 2

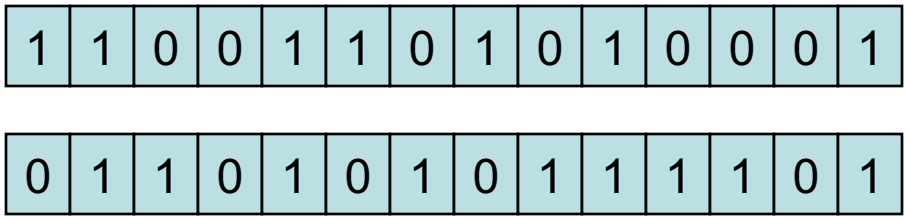
Parameters of a GA

- Population size
- Encoding choices
- Crossover probability
- Mutation probability
- Replacement strategies
- (and of course, the fitness function)

Encoding of a Chromosome

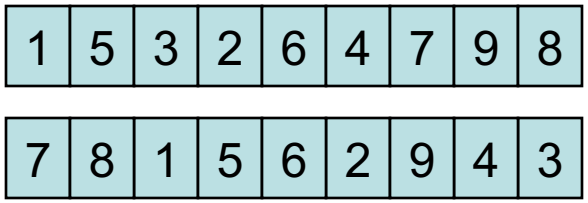
- The chromosome should in some way contain information about the solution which it represents
- Popular methods include
  - Binary encoding
  - Permutation encoding
  - Value encoding
  - Tree encoding

Encoding of a Chromosome: Binary Encoding



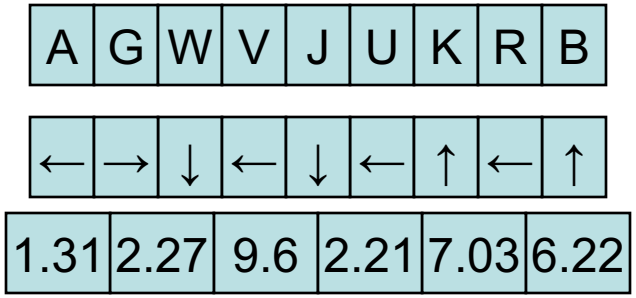
- Each chromosome has one binary string.
- Meaning of the string can vary
  - Each bit can represent some characteristic of the solution.
  - The whole string can represent a number.
- **Example**: Knapsack problem
  - There is a set of items, each with a given value and size.
  - The knapsack has given capacity.
  - **Task**: Select things to maximize the value of items in knapsack, but do not exceed capacity.
  - **Encoding**: Each bit says if the corresponding item is in knapsack.

Encoding of a Chromosome: Permutation Encoding



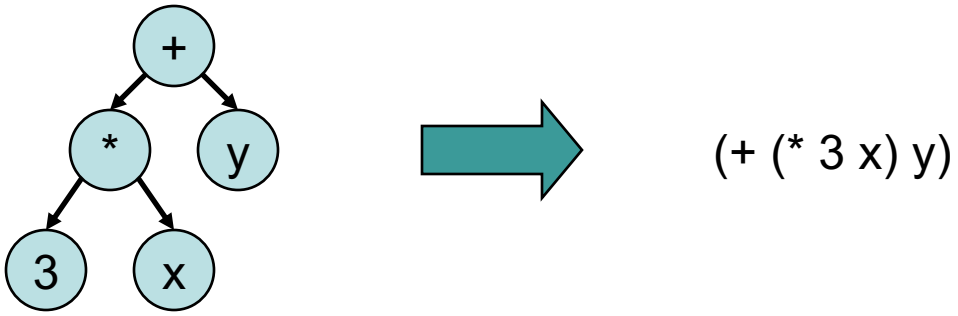
- Every chromosome is a string of numbers, which represent numbers in a sequence.
- Useful for ordering problems.
- **Example**: Traveling salesman problem (TSP)
  - There is a set of cities with given distances between them.
  - Traveling salesman must visit all cities
  - **Task**: Find a sequence of cities to minimize the distance travelled.
  - **Encoding**: Chromosome gives order of cities to visit

Encoding of a Chromosome: Value Encoding



- Every chromosome is a string of values.
- Values can be anything connected to the problem.
- **Example**: Finding weights for a neural network
  - Given a neural network with a specific architecture
  - **Task**: Find weights to train the network to a desired output.
  - **Encoding**: Real values in chromosomes represent corresponding weights for inputs.

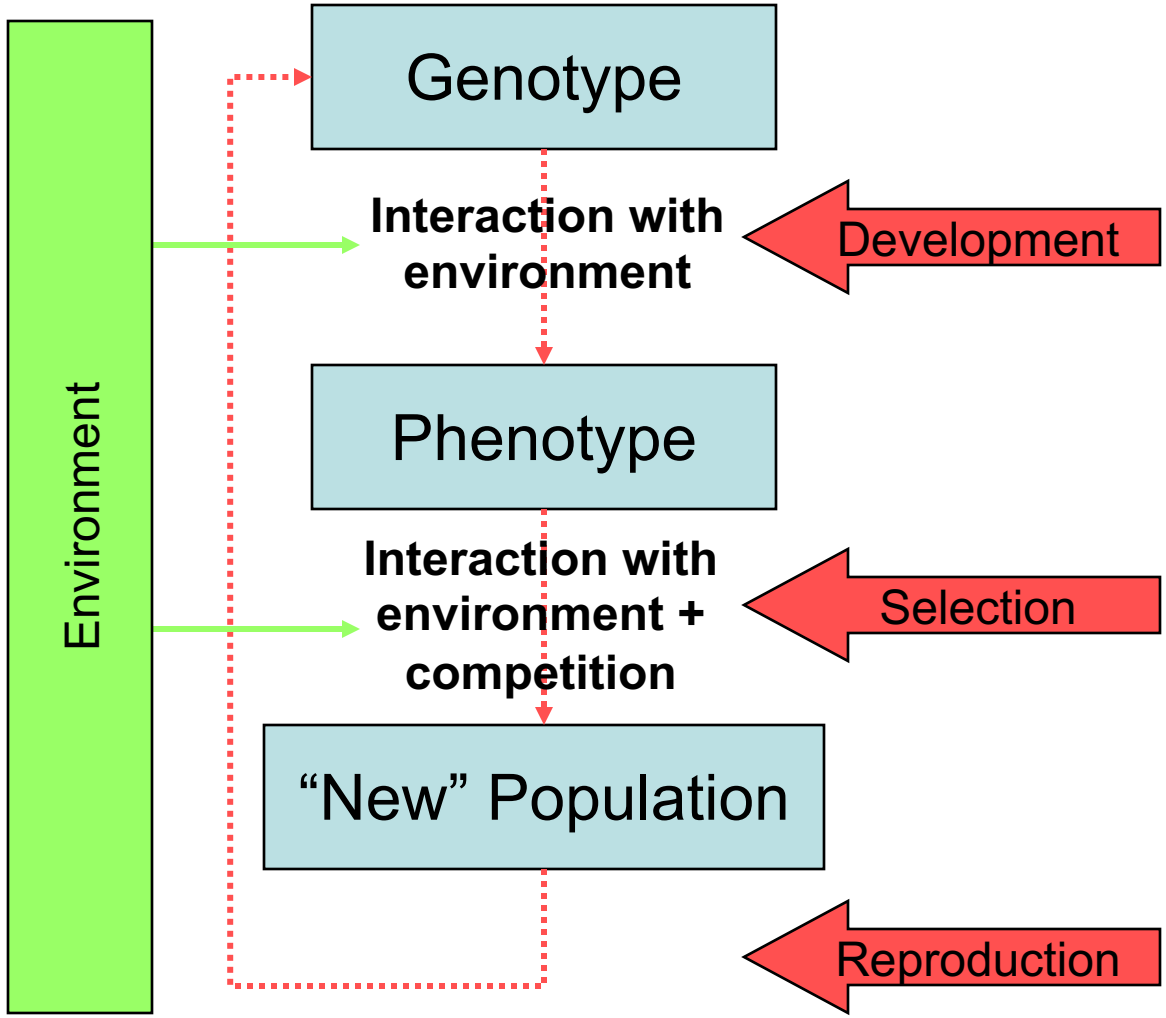
Encoding of a Chromosome: Tree Encoding



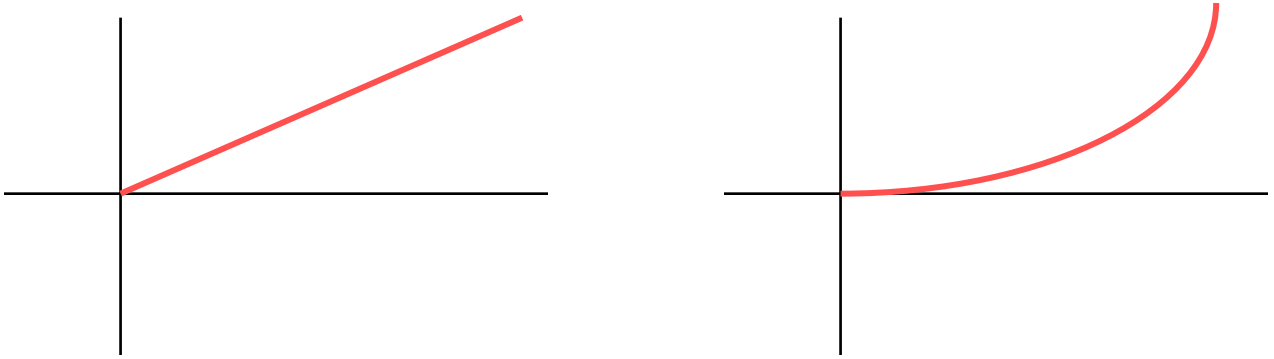
- Every chromosome is a tree of objects, such as functions or commands in a programming language.
- Useful mainly for evolving programs or expressions
- **Example**: Finding a function from given values
  - Some input and output values are given
  - **Task**: Find a function that will give the best (closest to desired) output to all inputs.
  - **Encoding**: Chromosome are functions represented in a tree.

Overview of the Process

1.[Start]
2.***[Fitness]***
3.[New population]
    A.[Selection]
    B.[Crossover]
    C.[Mutation]
    D.[Accepting]
4.[Replace]
5.[Test]
6.[Loop] Go to step 2



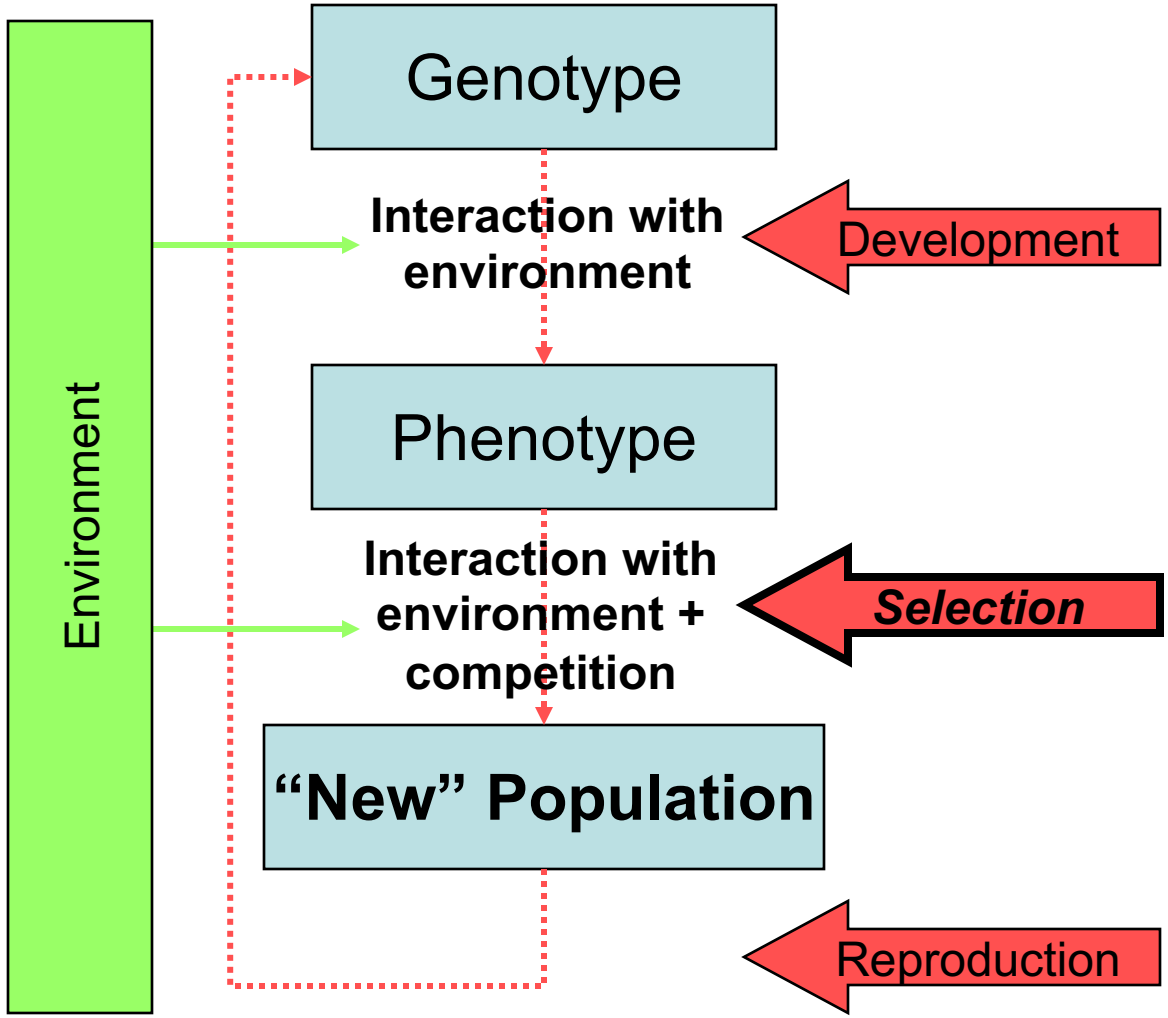
Defining a Fitness Function



- If the correct answer is known, the fitness is some distance metric toward the correct answer.
- If the correct answer is unknown, fitness must be an estimator of the value of the solution.
- Combinations of multiple goals into a single numeric function can be difficult.
- Evolution can only be as good as the fitness function.

Overview of the Process

1.[Start]
2.[Fitness]
3.*[New population]*
   A.***[Selection]***
   B.[Crossover]
   C.[Mutation]
   D.[Accepting]
4.[Replace]
5.[Test]
6.[Loop] Go to step 2

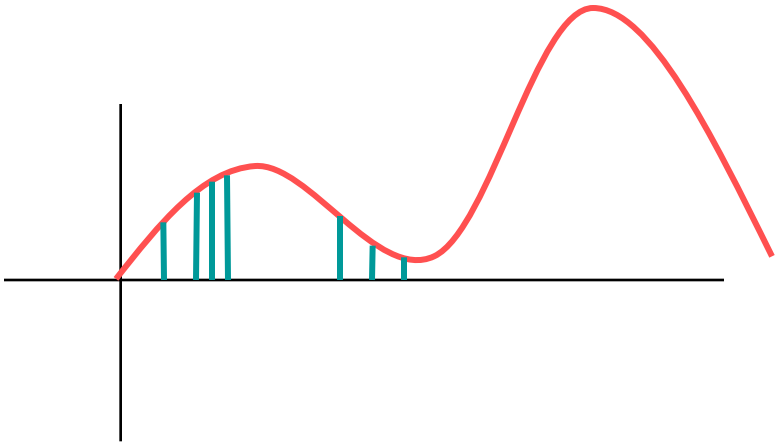


Selection Strategies

- Select a mating population from the current generation
- Popular methods include
  - Greedy selection
  - Roulette wheel selection
  - Rank selection
  - Elitism

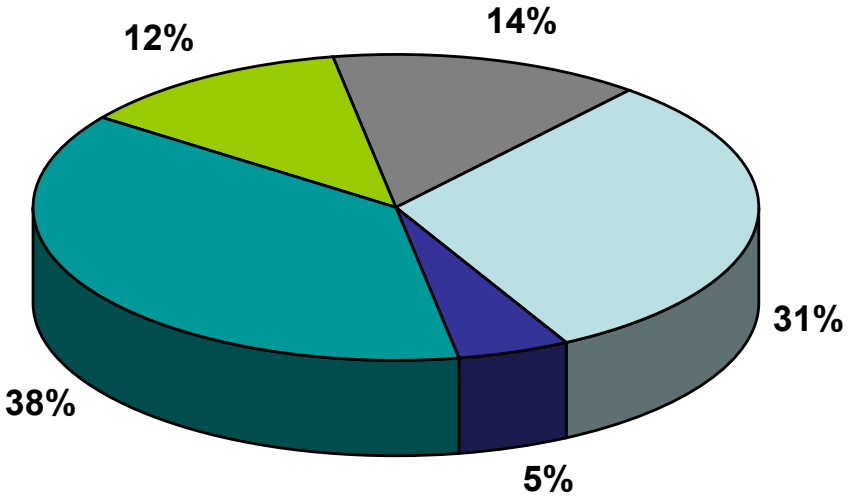
Greedy Selection

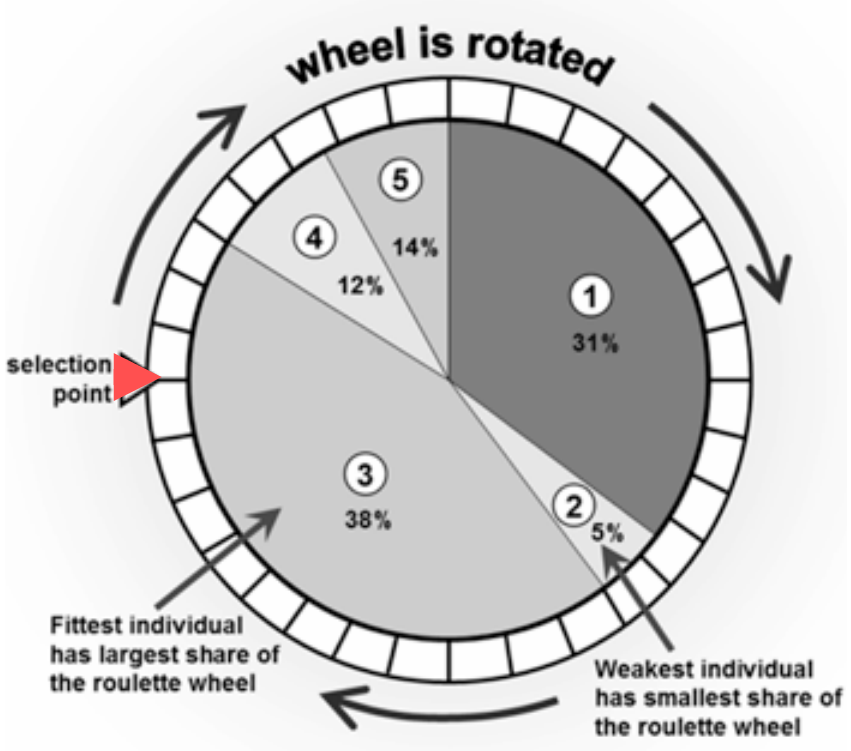
- The easy way:
  - Take the *n* most-fit individuals
- Why the easy way fails…
  - Loss of diversity
  - Stuck in local maxima



Population size: 7, *n*=3

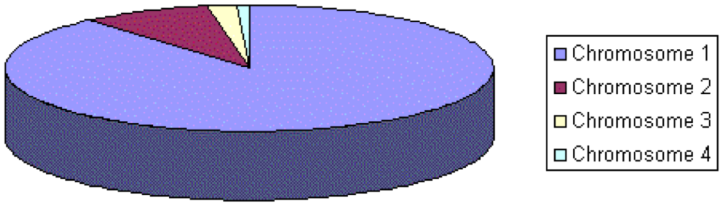
Roulette Wheel Selection



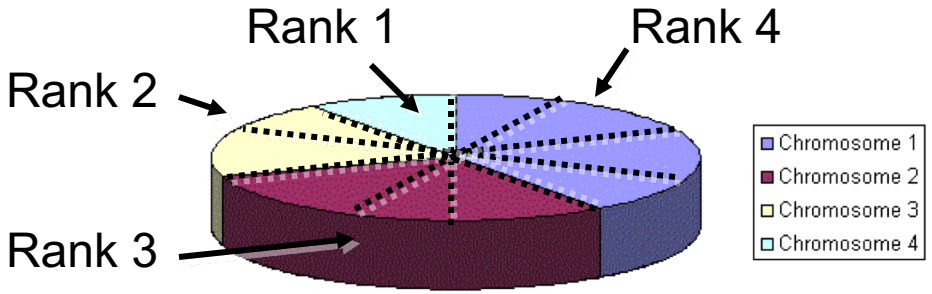


- Parents are allocated space on the wheel in proportion to their fitness.
- A marble is thrown and selects the chromosome.
  - Repeat *n* spins.
- Chromosomes with larger fitness values will be selected more times.

Rank Selection



Roulette Selection



Rank Selection

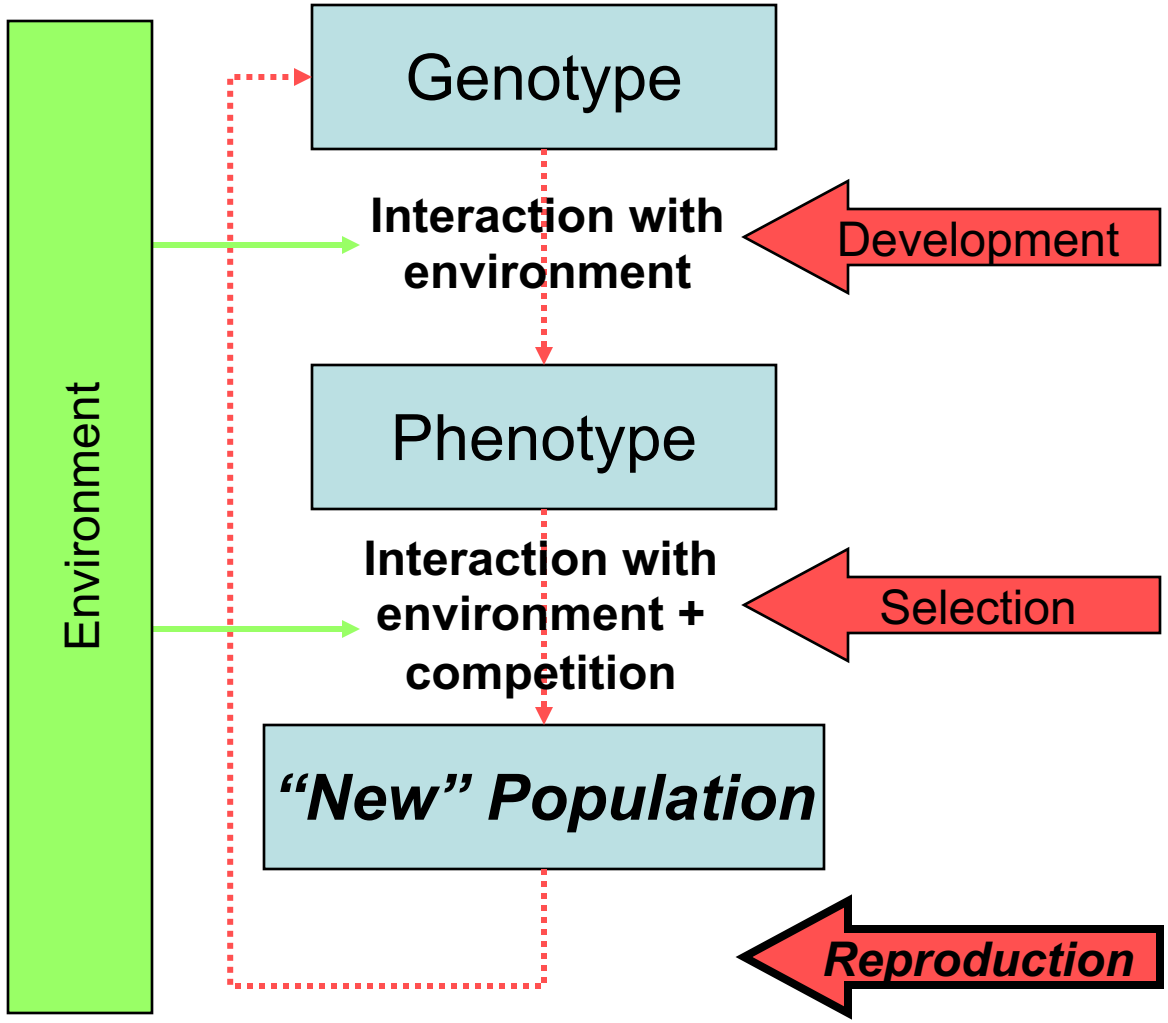
- Roulette selection has problems when the fitness values differ very much.
  - For example, if the best chromosome fitness is 90% of all the roulette wheel then the other chromosomes will have very few chances to be selected.
- Rank selection
  - Every chromosome receives fitness from its ranking.
  - The worst will have fitness 1.
  - The best will have fitness *n* (number of chromosomes in population).
- Can lead to slower convergence, because the best chromosomes do not differ very much from other ones.

Elitism

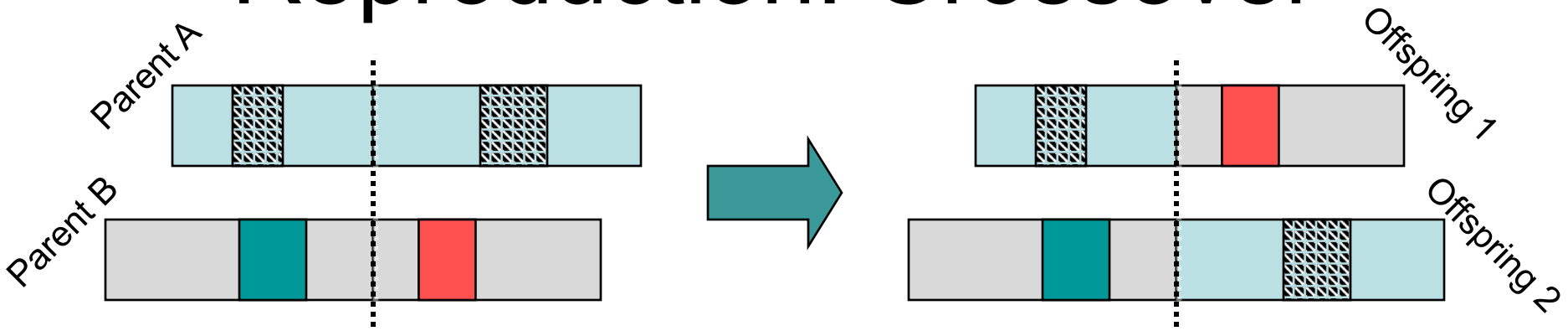
- When creating a new population by crossover and mutation, there is a good chance that the best chromosome will be lost.
- Elitism copies the best chromosome (or a few best chromosomes) from the previous generation into the new population.
- Selection proceeds according to any other selection method.
- Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution.

Overview of the Process

1.[Start]
2.[Fitness]
3.***[New population]***
   A.[Selection]
   B.***[Crossover]***
   C.***[Mutation]***
   D.[Accepting]
4.[Replace]
5.[Test]
6.[Loop] Go to step 2



Reproduction: Crossover



- **Single point crossover** :

  11001011 + 11011**111** → **11001111** + 11011011

- **Two point crossover** :

  110010**11** + 11**0111**11 → **11011111** + 11001011

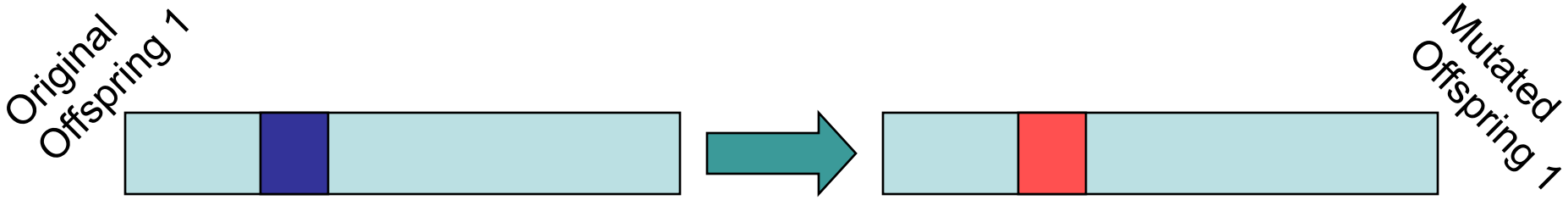
- **Uniform crossover** :

  1**10**010**11** + **1**10**111**01 → **11011111** + 11001001

- **Arithmetic crossover** : some arithmetic operation is performed to make a new offspring

  11001011 + 11011111 → 11001001 (AND)

Reproduction: Mutation



- In general, mutation depends on the encoding as well as the crossover.
- For binary encoding, mutation is simply bit inversion:

  11001001 → 10001001

- Mutation can prevent problems with local optima!

Crossover and Mutation with Permutation Encoding

- **Single point crossover**
  - One crossover point is selected
  - Copy sequence before crossover from first parent
  - Scan second parent for sequence of remaining

  (**1 2 3 4 5** 6 7 8) + (4 5 3 **6 8** 2 **7** 1) → (1 2 3 4 5 6 8 7)

- **Mutation**
  - Order changing : two numbers are selected and exchanged
    - (1 **2** 3 4 5 6 **8** 9 7) → (1 **8** 3 4 5 6 **2** 9 7)

Crossover and Mutation with Value Encoding

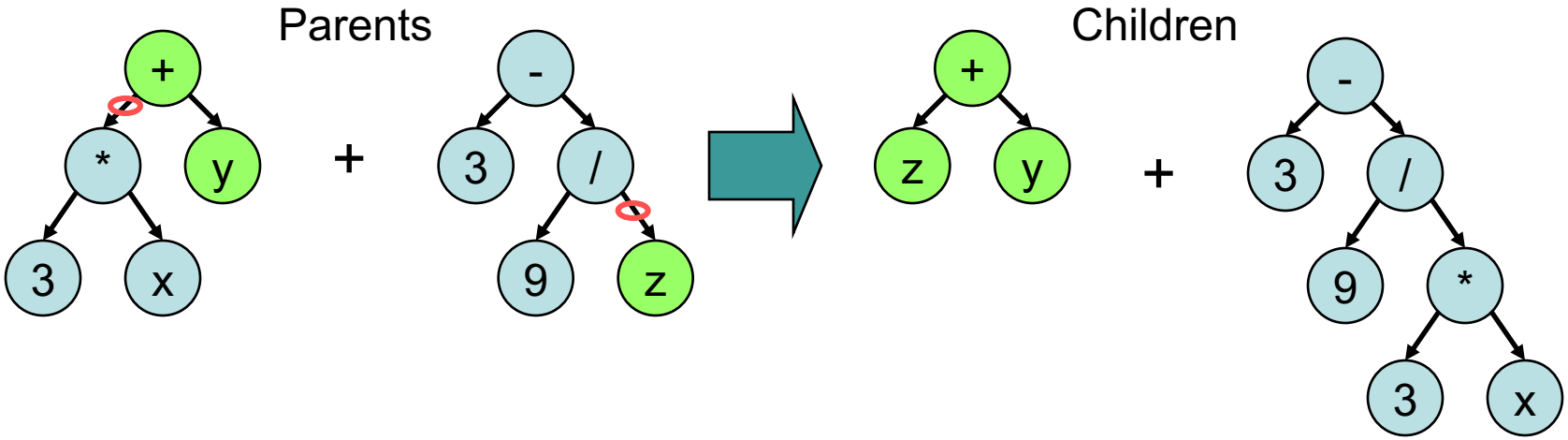
- **Crossover**
  - All crossovers from binary encoding can be used
- **Mutation**
  - Real numbers : add/subtract a small value

    (1.29 5.68 2.86 4.11) → (1.29 5.68 2.73 4.22)
  - Alphabet symbols : advance one place

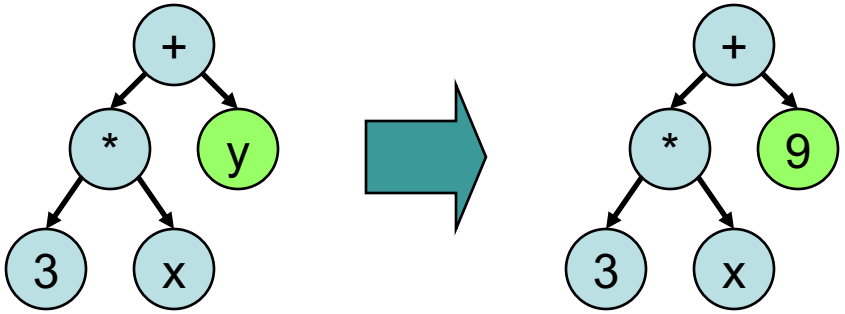
    (A F D E H) → (A G D E H)
  - Directions : rotate clockwise

Crossover and Mutation with Tree Encoding

- **Crossover**



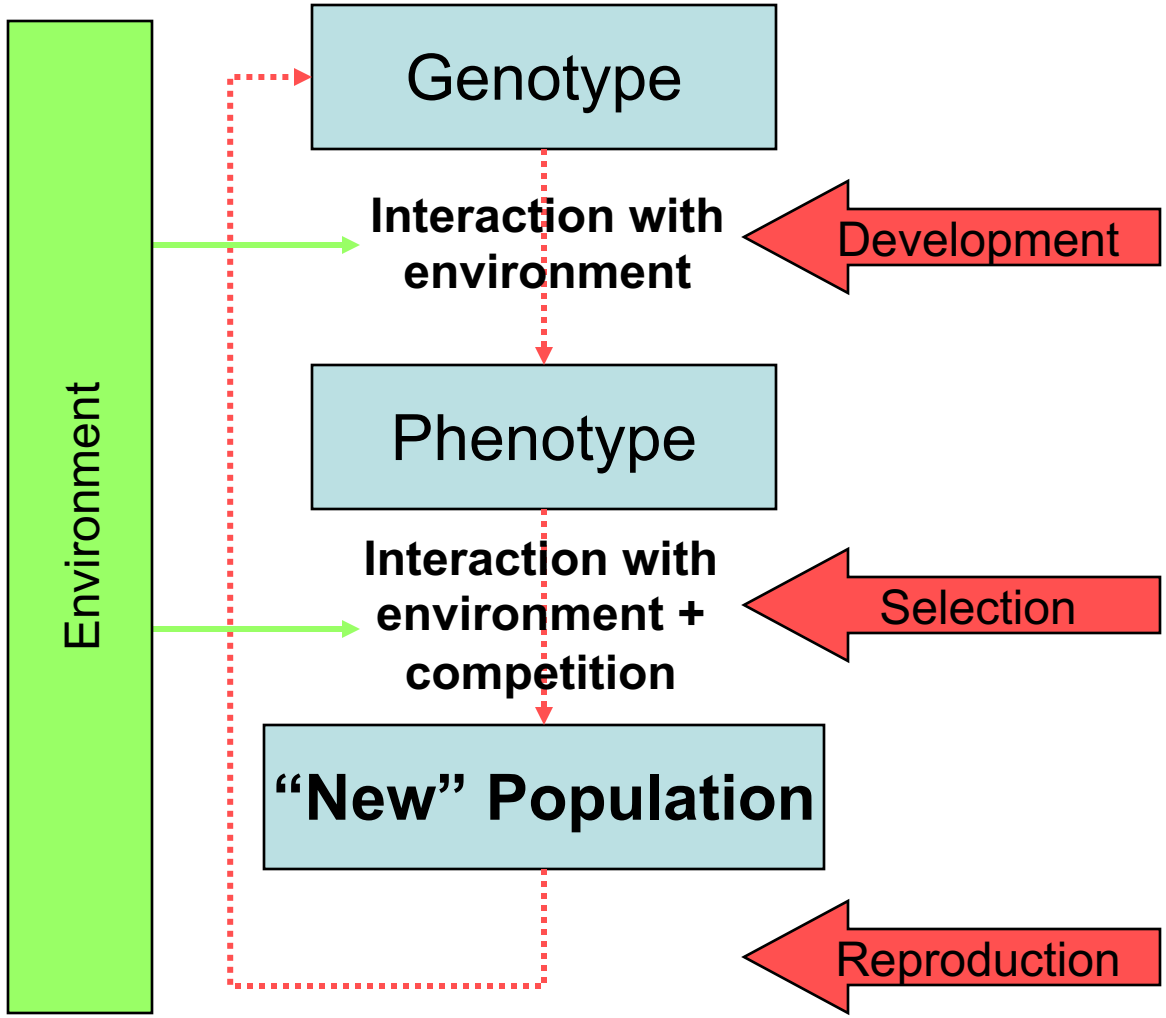
- **Mutation**



Note : We may need to distinguish between nodes based on the number of children to maintain a valid tree

Overview of the Process

1.[Start]
2.[Fitness]
3.[New population]
    A.[Selection]
    B.[Crossover]
    C.[Mutation]
    D.***[Accepting]***
4.***[Replace]***
5.[Test]
6.[Loop] Go to step 2

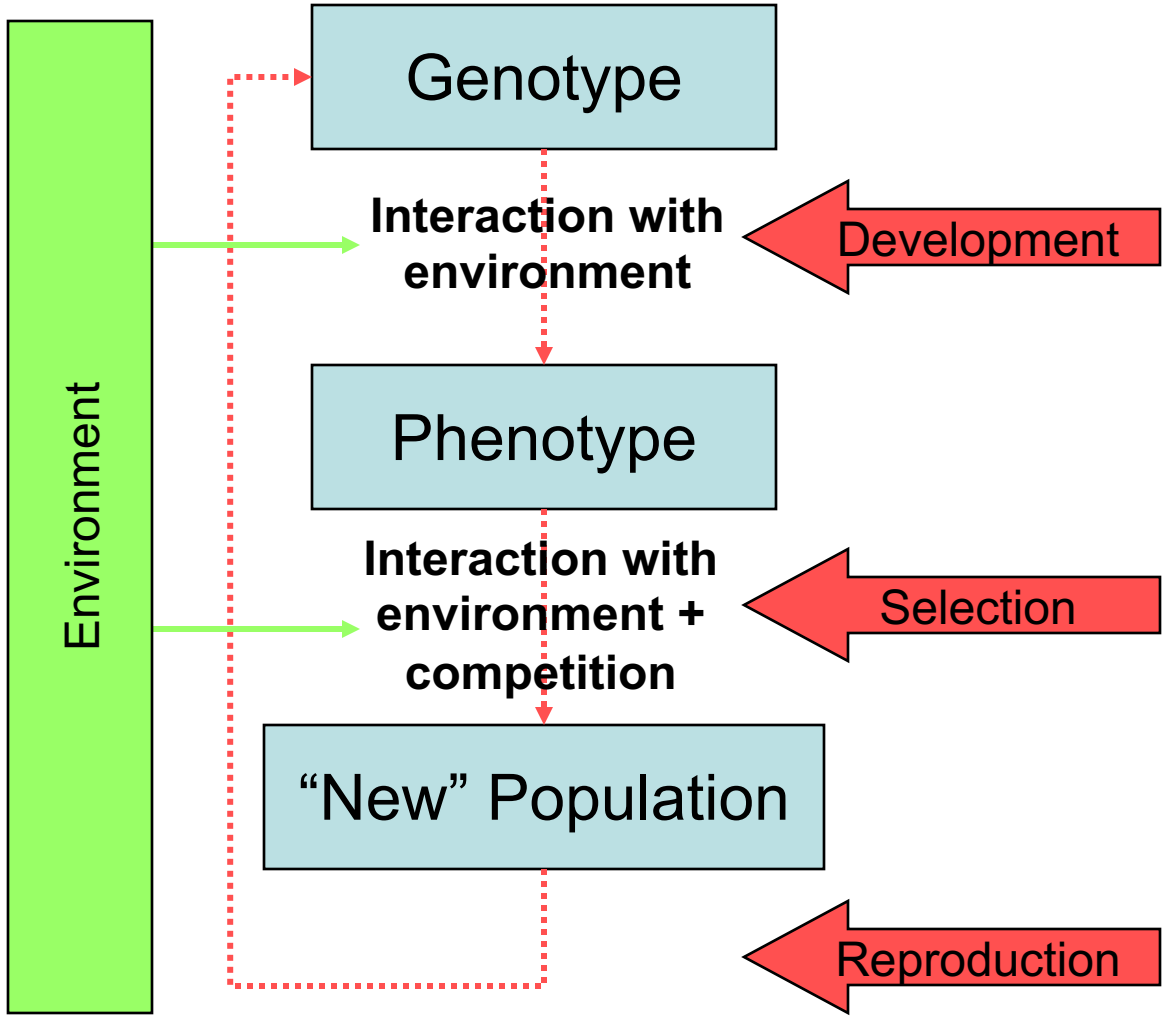


Replacement Strategies

- Generational Selection
  - All population members are removed on each generation.
- Overlapping Selection
  - In every generation, select a few high fitness chromosomes for creating new offspring.
  - Remove some low fitness chromosomes.
  - Replace these with new offspring.
  - Remainder of population remains
- Replace {worst, best, parent, random, most similar}

Overview of the Process

1.[Start]
2.[Fitness]
3.[New population]
    A.[Selection]
    B.[Crossover]
    C.[Mutation]
    D.[Accepting]
4.[Replace]
5.***[Test]***
6.***[Loop]*** Go to step 2



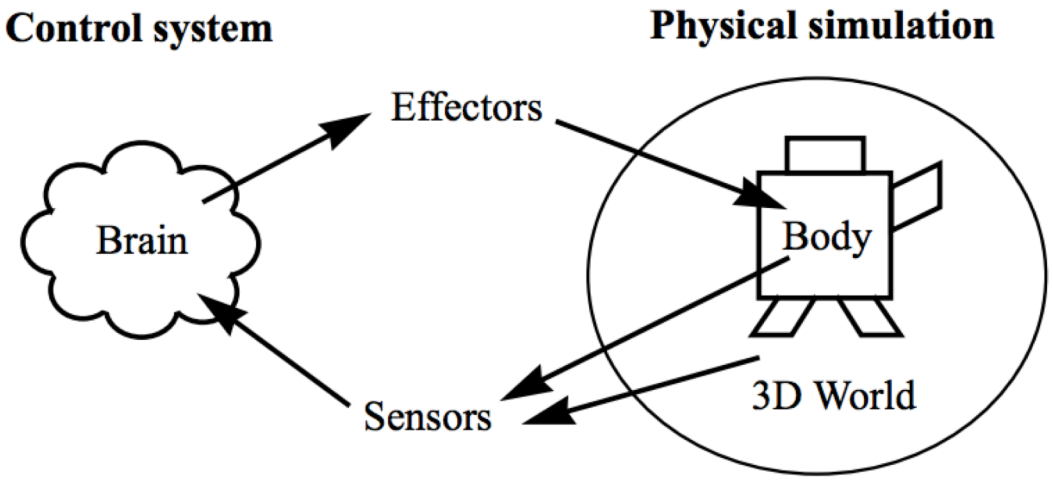
Advantages of GAs

- Parallelism
- Less likely to get stuck in local extrema than other optimization methods
- Relatively easy to implement
  - But choosing an encoding and fitness function can be difficult
- Disadvantages
  - Computational time
  - No guarantee on a solution
  - Strong dependence on parameters

Competitive Evolution in Simulation

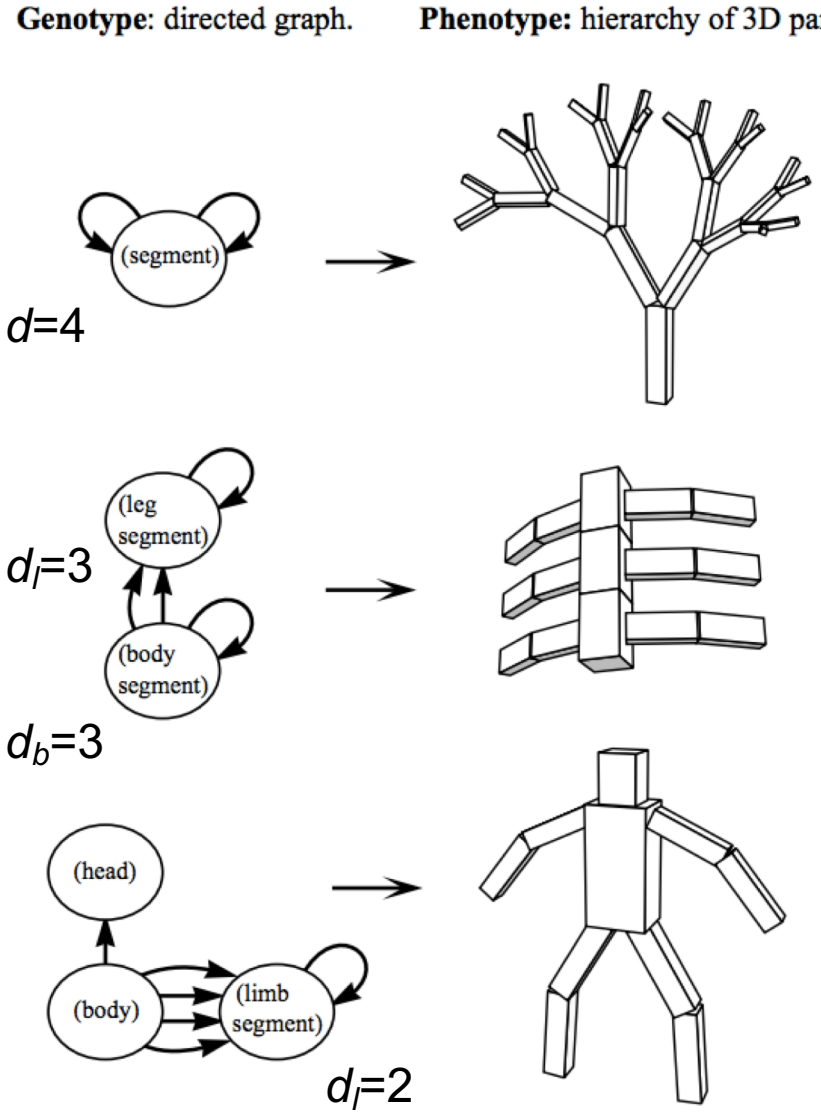
- **Karl Sims**
  - **MIT Media Lab**
  - **Thinking Machines**
  - **GenArts, Inc.**
- **“Evolving 3D Morphology and Behavior by Competition” (1994)**
- **2 big ideas**

Great Idea #1: Evolving Morphology with Control



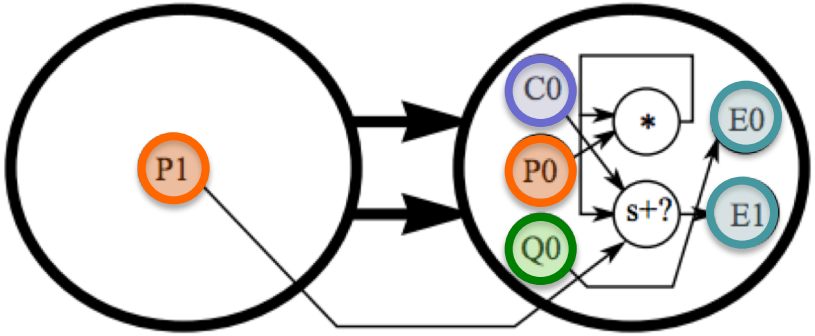
- **Evolve both the**
  - Body structure (morphology)
  - Brain structure (control)

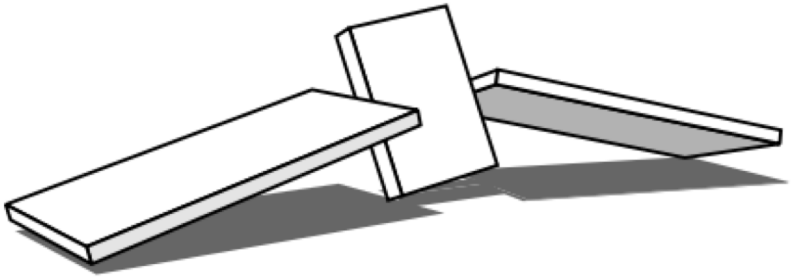
Physical Morphology Encoding: Genotypes and Phenotypes



- Genotype is composed of a directed graph structure where each node contains
  - Dimensions for a rigid part
  - Attachment points
  - Joint types at those points
  - Maximum recursion depth
- Phenotype is a hierarchy of 3-D parts in a physics-rich simulation

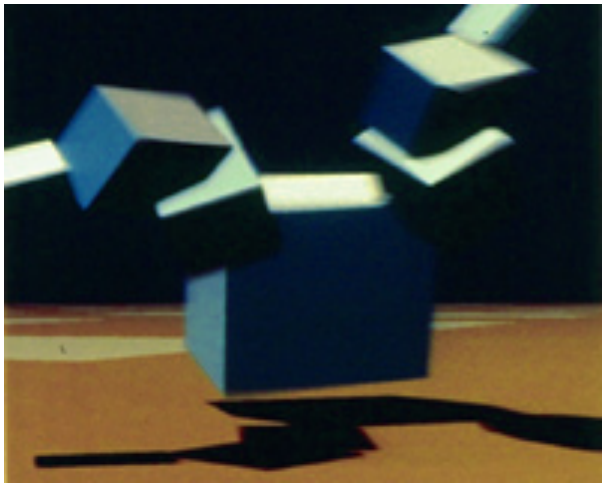
Control Structure Encoding (Genotype)





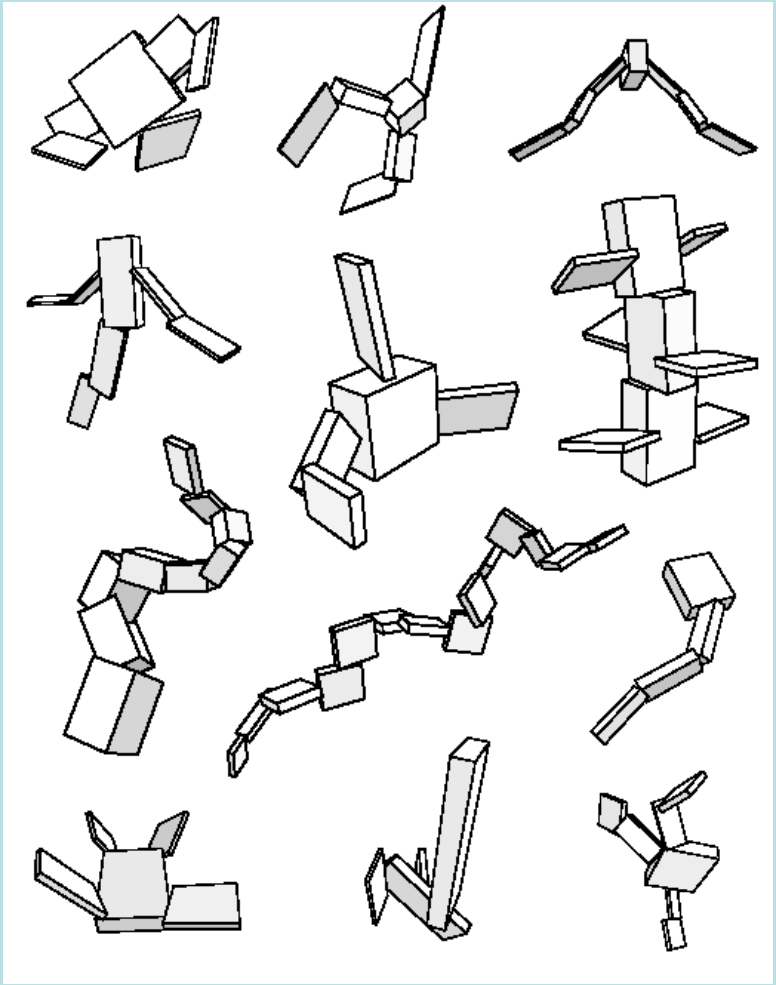
- Nested graphs genotype
  - Outer graph is morphology
  - Inner graph is neural circuitry
- Sensors for
  - Contact (C)
  - Photocells (P)
  - Joint angle sensors (Q)
- Computation elements (multiply and threshold sum)
- Effector outputs (E)

Fitness Functions for the Development of Locomotion

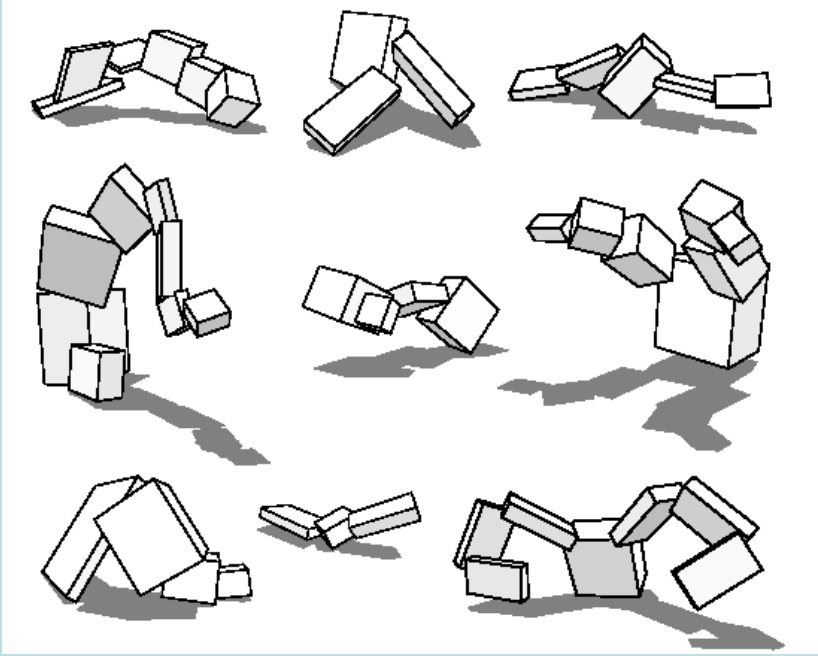


- Land and water environments
- Locomotion fitness function = speed
  - Distance traveled by the center of mass per unit time
  - Ignore the vertical component when on land!
- Jumping fitness function
  - Maximum height achieved by the lowest point in the creature

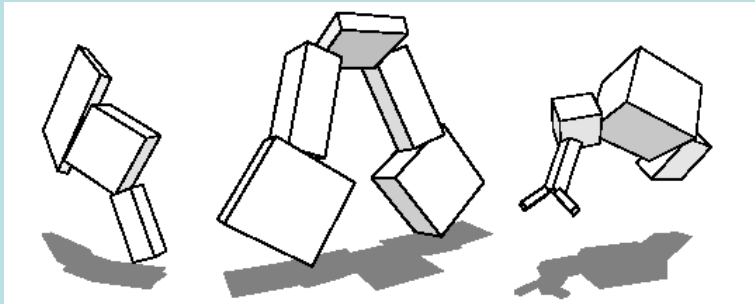
Creatures Evolved for Locomotion



Swimming

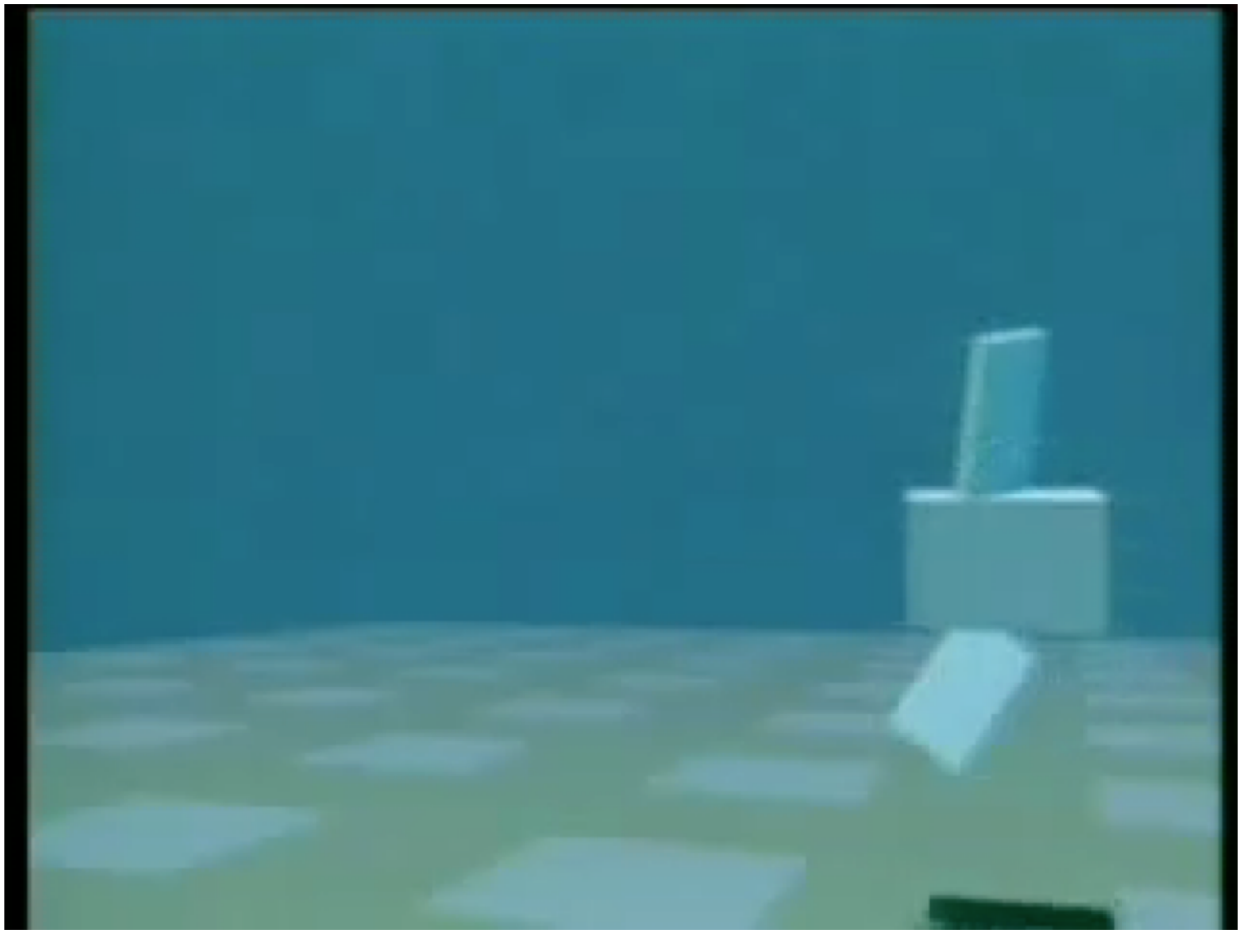


Walking



Jumping

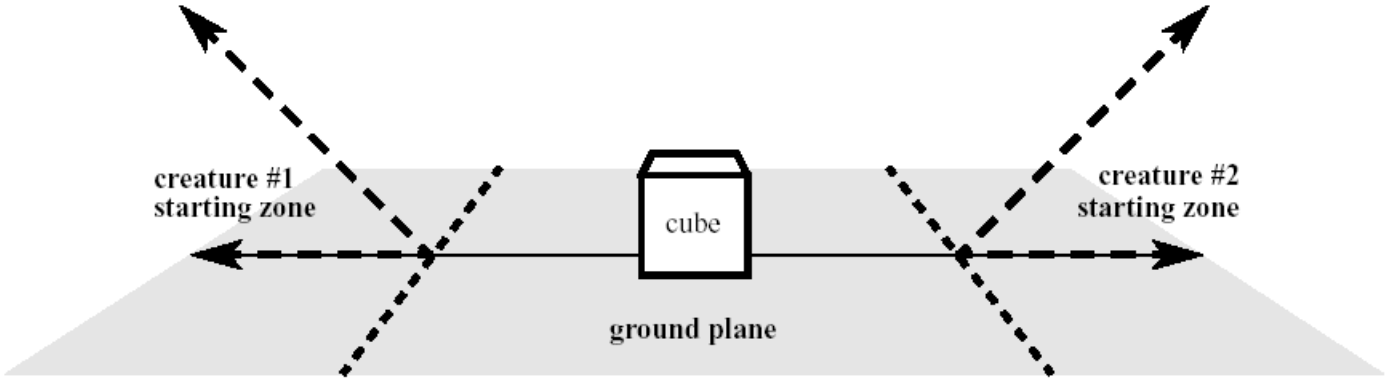
Demonstrations of Locomotion



- 100 generations of 300 individuals
- 3 hours on a 32-processor CM-5

Great Idea #2:
Evolution at times involves more than competition with the environment

Competition Arena



- Competition between two creatures
- Must start behind the line (and below a 45 degree ceiling)
- Objective is to get the cube for yourself (and thus keep the cube from your opponent)

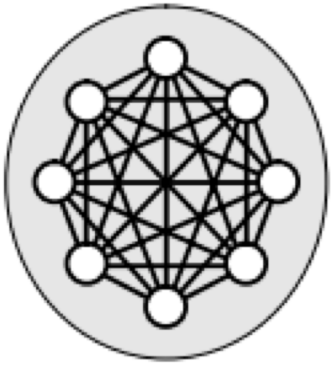
Fitness Function

- Let d_1 and d_2 be the final shortest distances of each creature to the cube
- Fitness for each creature:

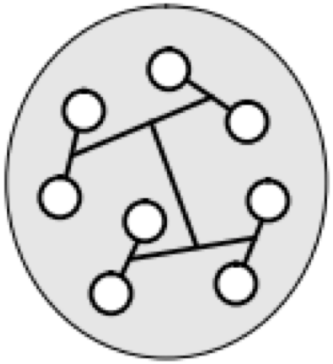
$$f_1 = 1.0 + \frac{d_2 - d_1}{d_1 + d_2} \qquad f_2 = 1.0 + \frac{d_1 - d_2}{d_1 + d_2}$$

- All fitness values in the range [0.0, 2.0]
- Scores always average to one
- Ties are permitted ($f_1 = f_2 = 1$)

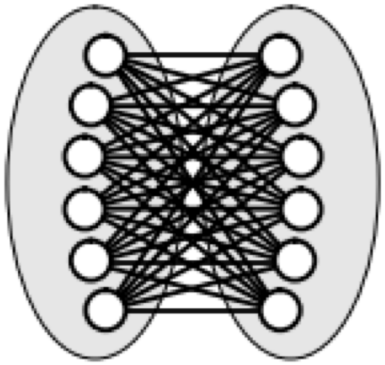
Pair-wise Competition Patterns



a. All vs. all, within species.

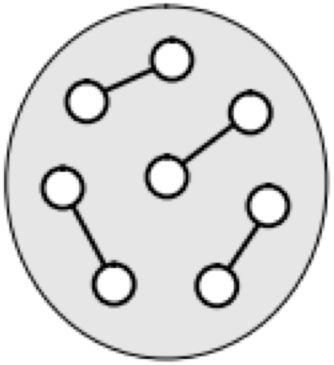


c. Tournament, within species.

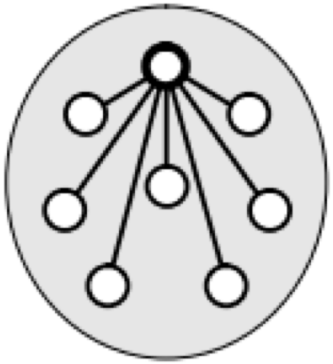


e. All vs. all, between species.

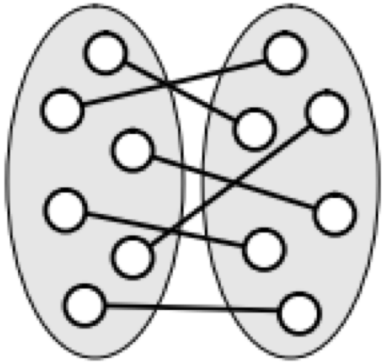
Empirically chosen to give the most interesting results



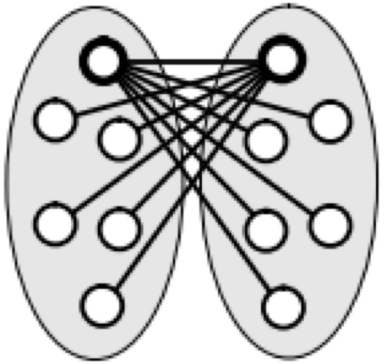
b. Random, within species.



d. All vs. best, within species.

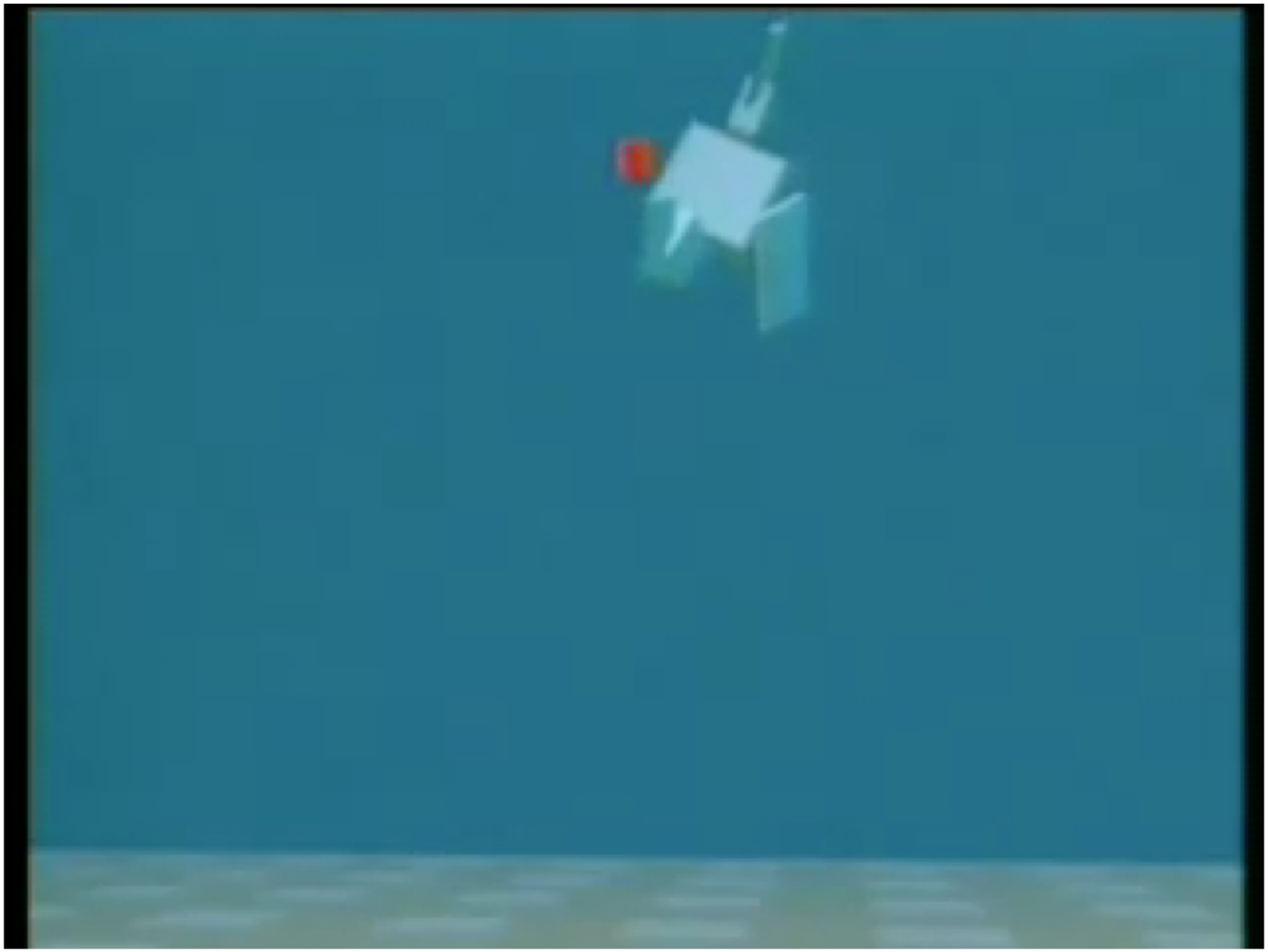


f. Random, between species.

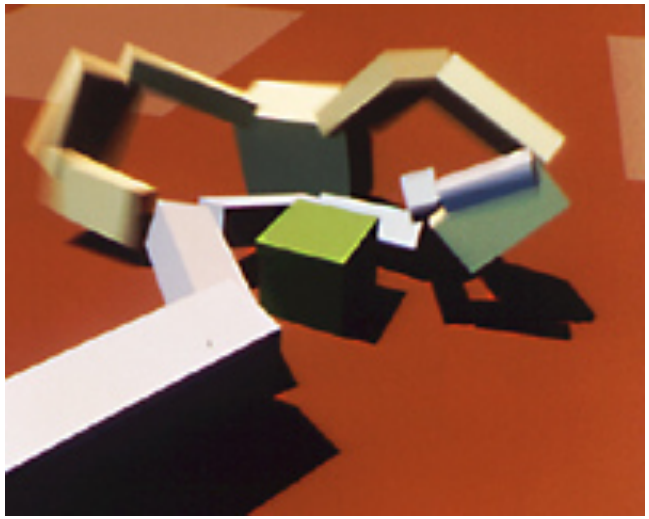


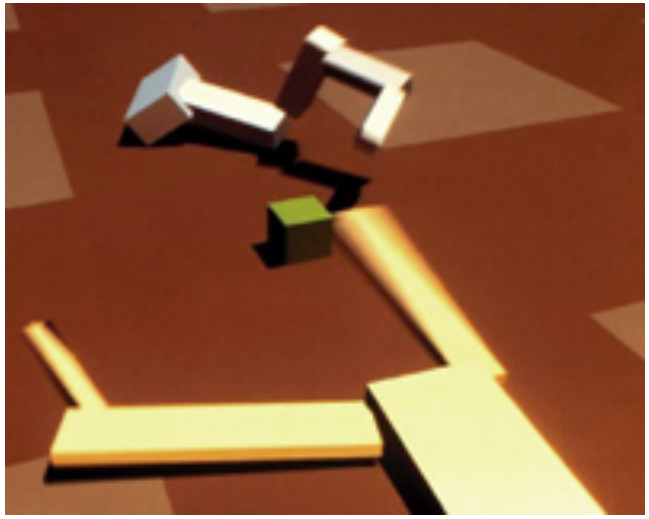
g. All vs. best, between species.

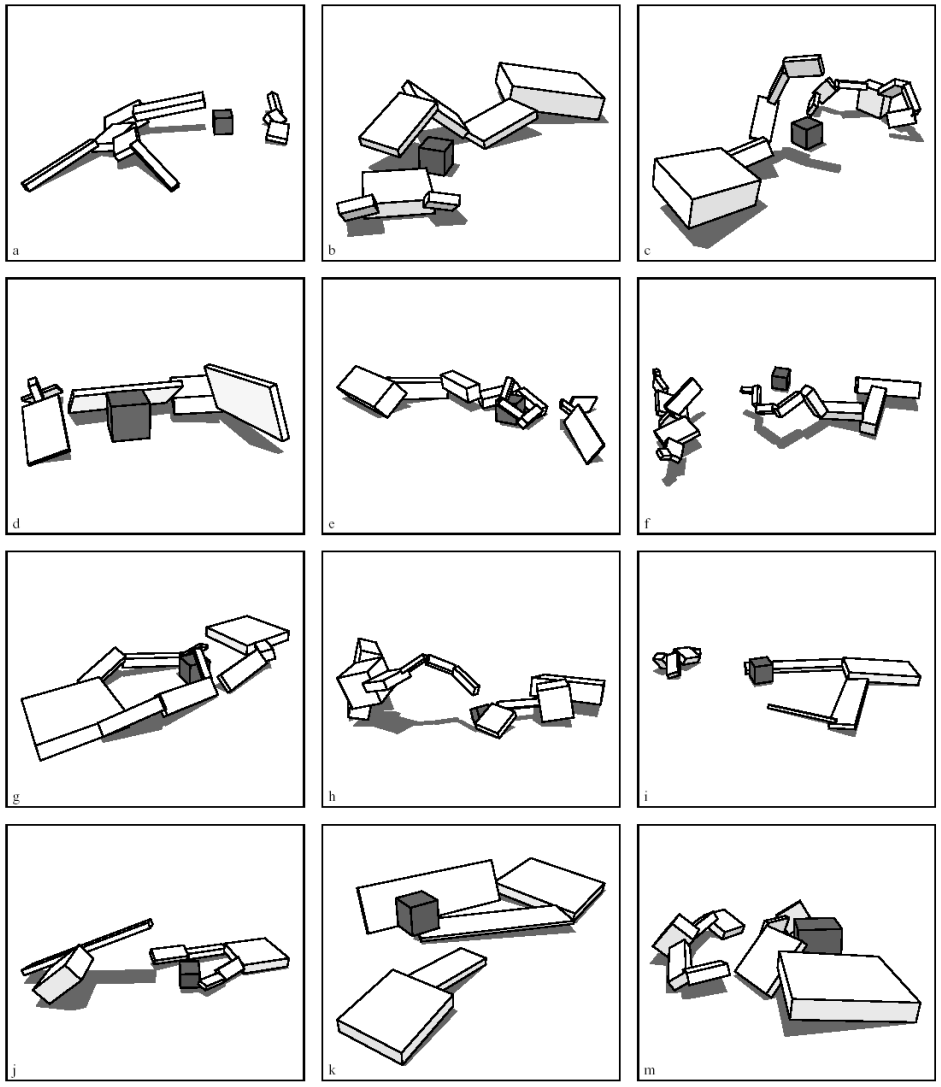
Evolved via Competition



Examples of Co-evolved Creatures





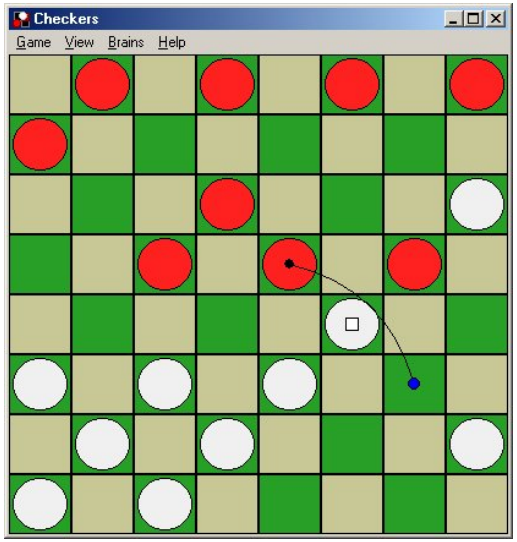


What are evolutionary algorithms good for?

Applications of GAs: Mechanical Design

- General Electric's custom GA *EnGEneous* used to design the Boeing 777's jet engines.
- GE had a viable six-stage compressor design in less than a week.
- The compressor solution yielded additional efficiencies on top of the design criteria.
  - required less metal
  - less weight
  - greater-than-anticipated decrease in fuel consumption

Applications of GAs: Game-Playing Systems



- Evolution of Master-level checkers player (David Fogel)
- But still unable to compete against Chinook, a hand-coded checkers player that can beat any human player

But is it *really* useful?

- “Neural networks are the second best way of doing just about anything…”
- “… and genetic algorithms are the third.”
  - Attributed to John Denker

Administrivia

- No class meeting on Friday… have a great break!

- Midterm exam scores released this week
- PS 5 is coming soon!