5-13-2003

Evolutionary Algorithms for Beginners

Marek Perkowski
Portland State University

Citation Details

Perkowsji, Marek, "Evolutionary Algorithms for Beginners," Data Mining Seminar in Materials Science at KAIST, 2003.

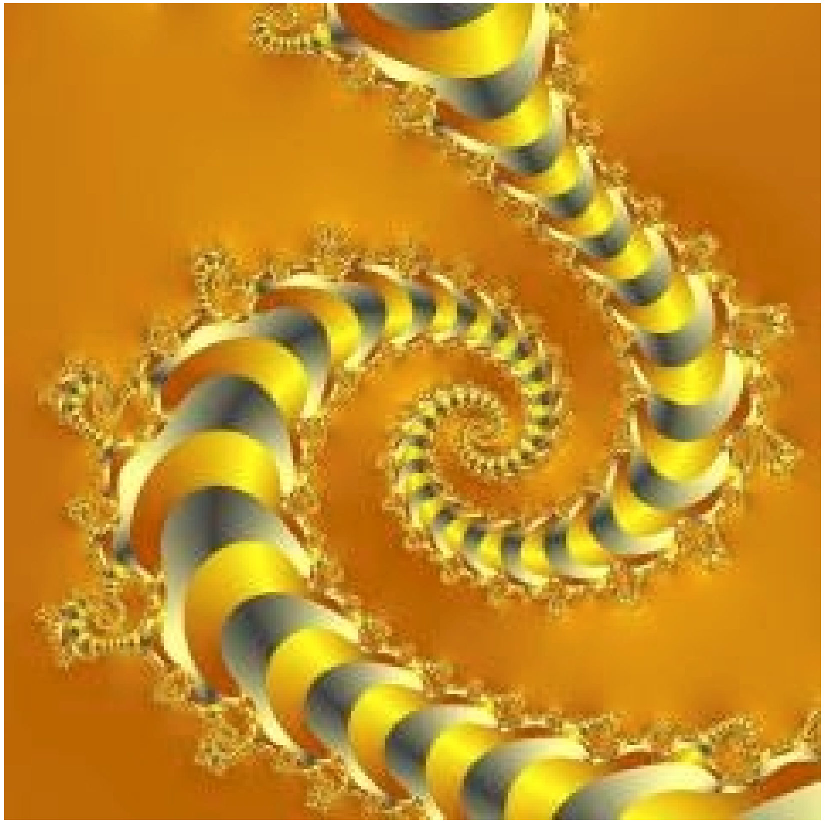
Evolutionary Algorithms for Beginners

Data Mining Seminar in Materials Science at KAIST

May 13, 2003

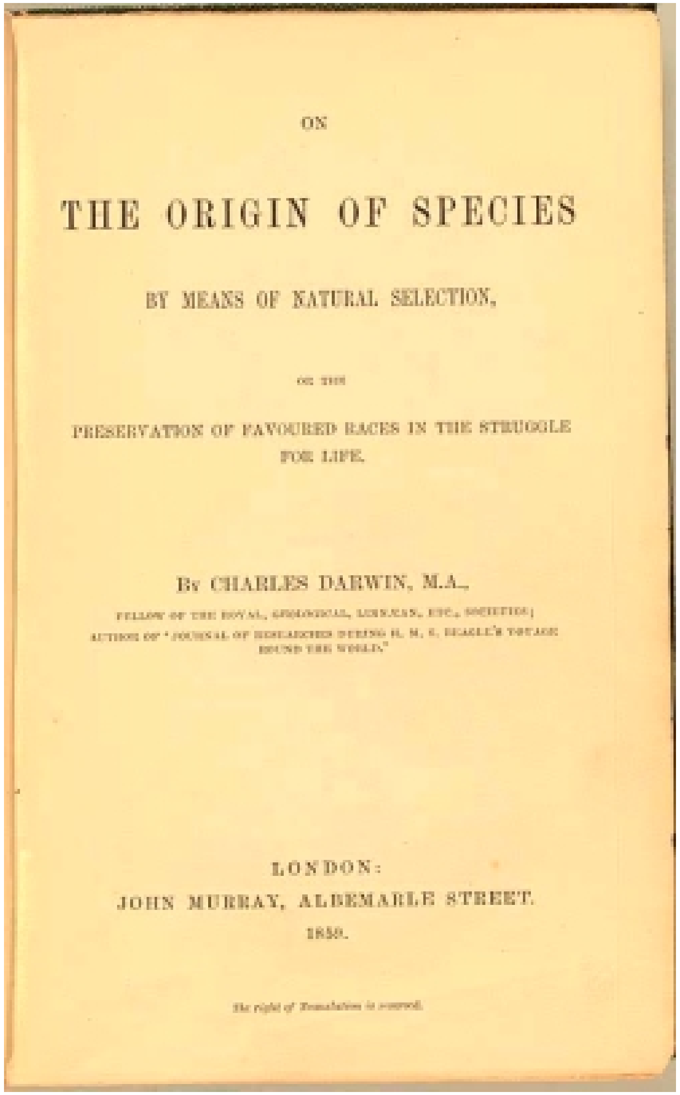
Marek Perkowski

Exiting stuff...



Evolutionary Ideas in Computing

Theory of Evolution - Darwin



ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RACES IN THE STRUGGLE
FOR LIFE.

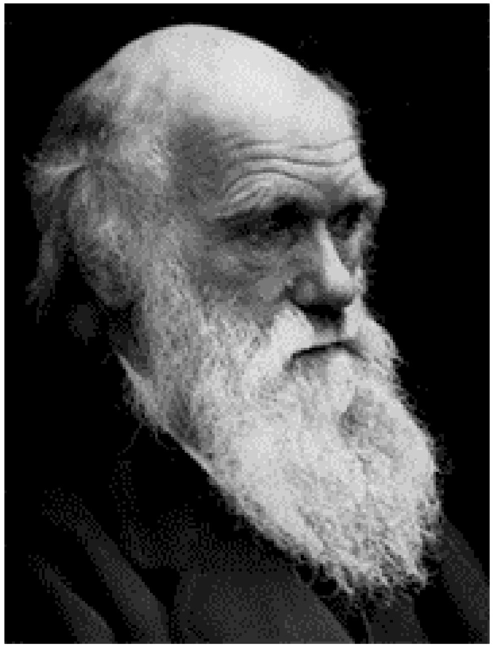
By CHARLES DARWIN, M.A.,

FELLOW OF THE ROYAL, GEOLOGICAL, LINNÆAN, ETC., SOCIETIES;
AUTHOR OF 'JOURNAL OF RESEARCHES DURING H. M. S. BEAGLE'S VOYAGE
ROUND THE WORLD.'

LONDON:
JOHN MURRAY, ALBEMARLE STREET.
1859.

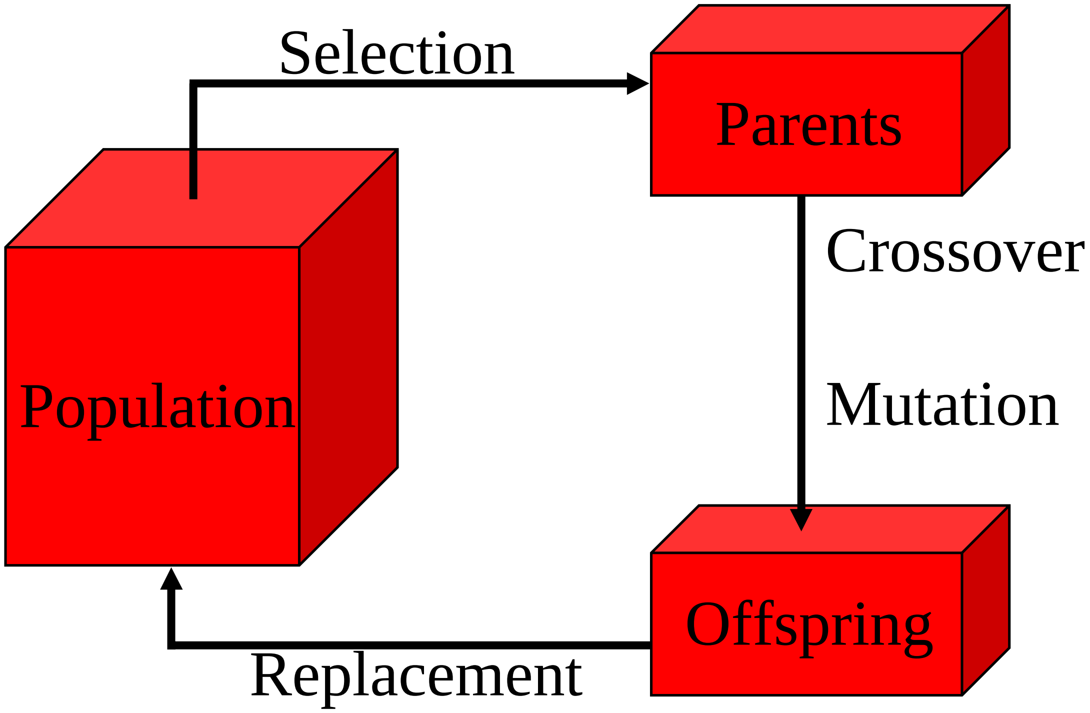
The right of Translation is reserved.

1859



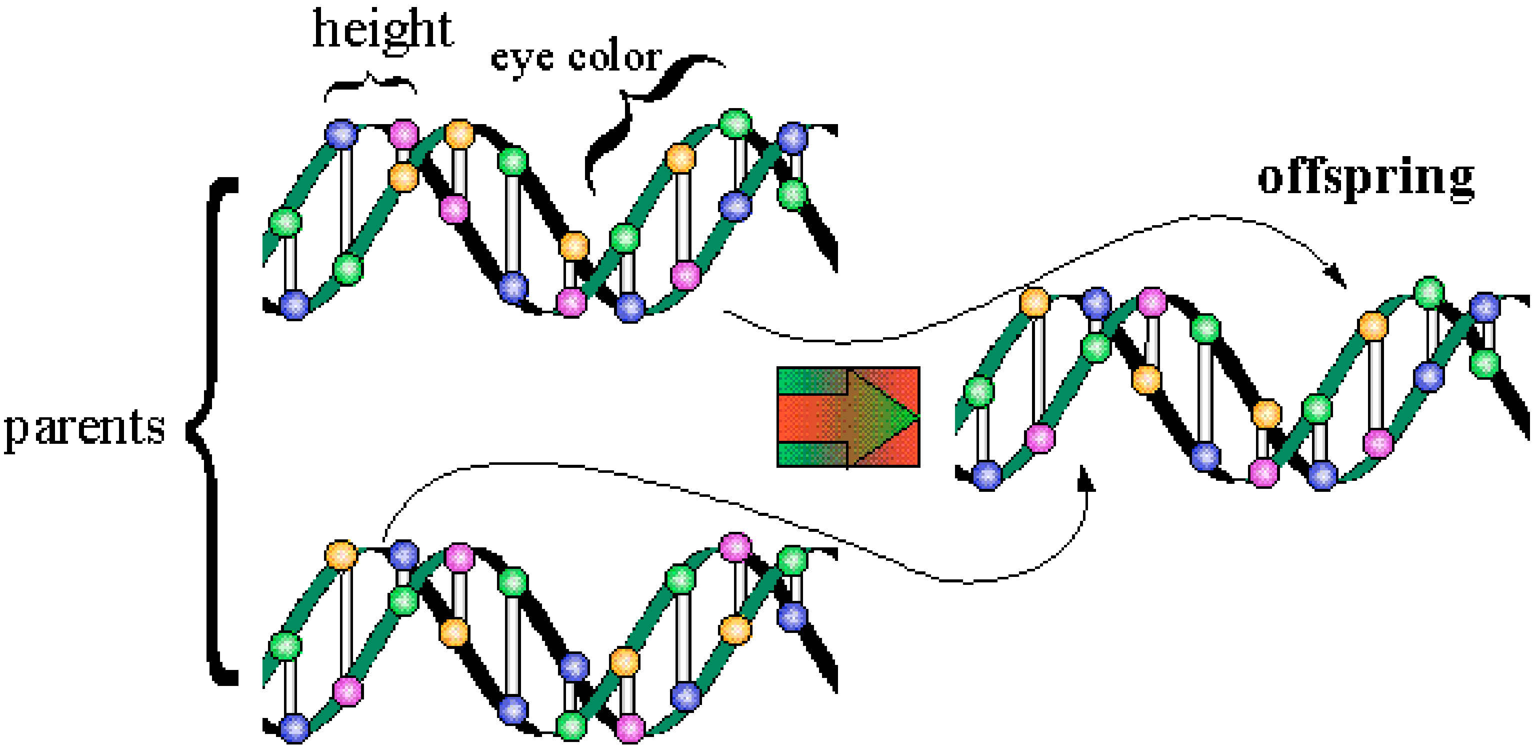
Charles Darwin
(1809 - 1882)

General Principle



Concepts from Genetics I

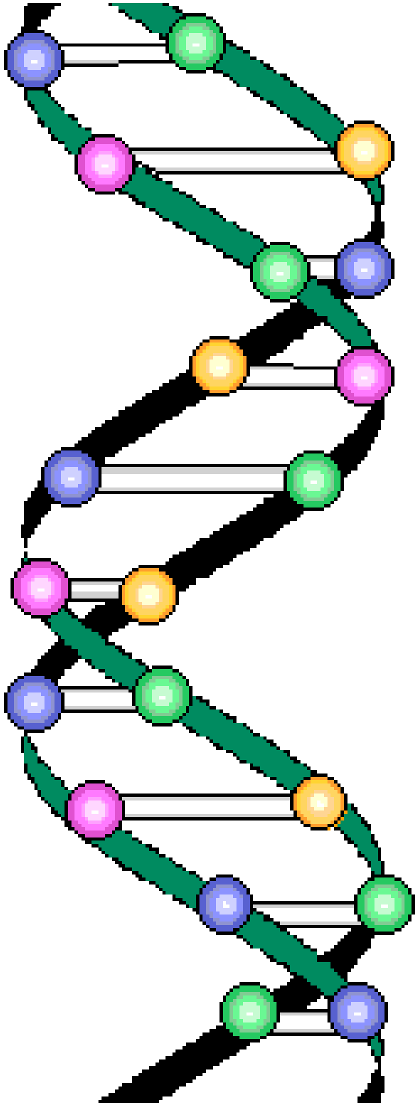
- A gene is a short length of a chromosome which controls a characteristic of an organism.
- The gene can be passed on from parent to offspring, e.g. a gene for eye-color.



Concepts from Genetics II

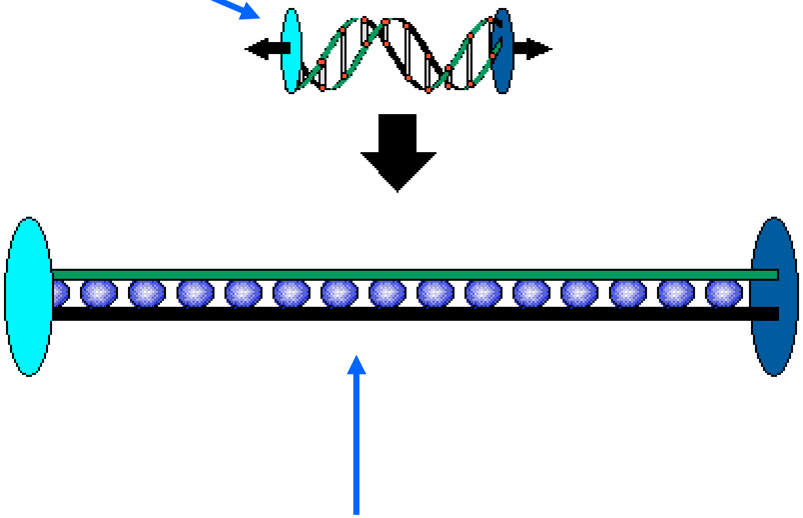
- A chromosome is a chain of genes

- Each living object has a particular number of chromosomes, e.g. human beings have 46 chromosomes.



GA Structure

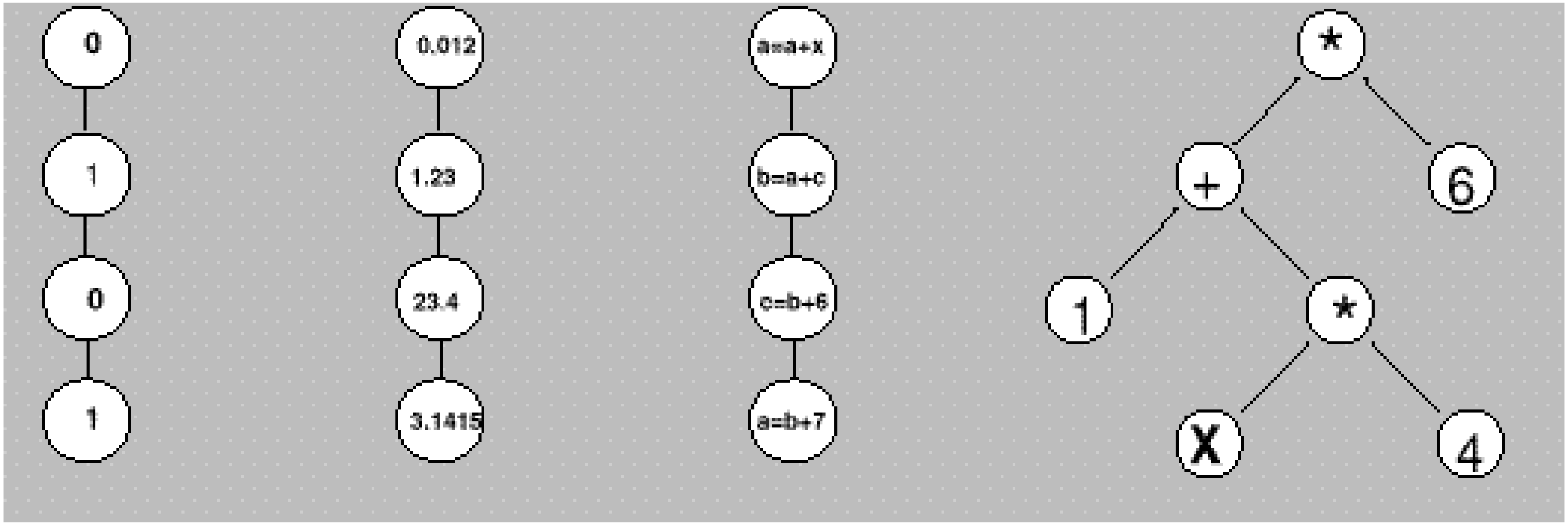
- Given a problem that in some way involves a search, a genetic algorithm begins with chromosome which represents a solution (usually a binary string).
- We will use non-binary strings also



Human Chromosome

Linear collection of bits (GA Chromosome)

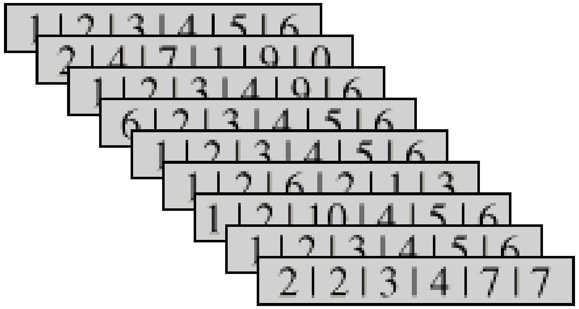
Representation of individuals in various EAs

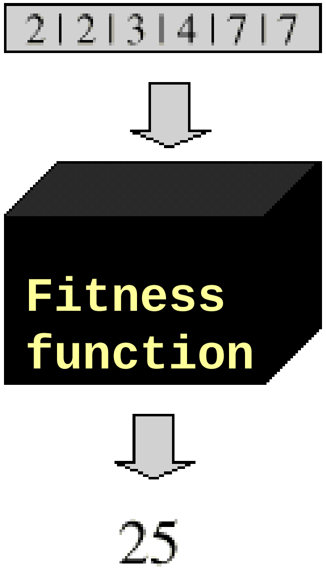


GA Components

- **Population** - consists of individuals who may be able to solve the given problem

- **Fitness function** – a function which determines how well each individual solves the problem





Contents of the Three Lectures on Evolutionary Algorithms

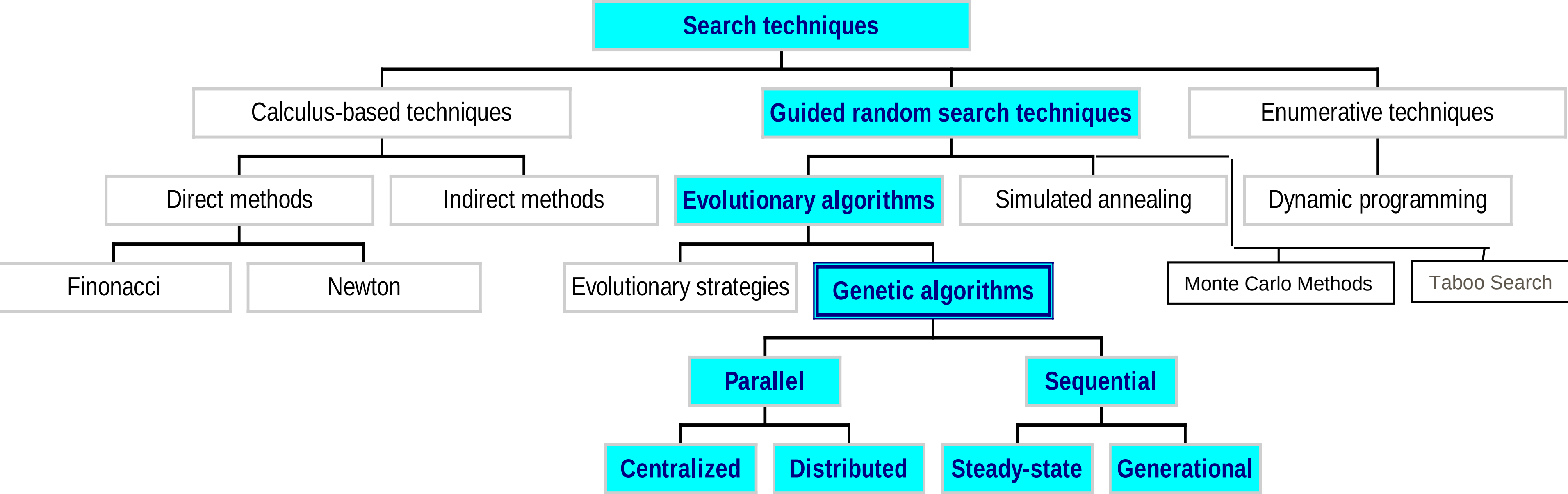
- Taxonomy and History;
- Evolutionary Algorithms basics;
- Theoretical Background;
- Outline of the various techniques: plain genetic algorithms, evolutionary programming, evolution strategies, genetic programming;
- Practical implementation issues;
- Evolutionary algorithms and soft computing;
- Selected applications;
- Evolutionary Hardware;
- Summary and Conclusions.

Bibliography

- Th. Bäck. ***Evolutionary Algorithms in Theory and Practice*.** Oxford University Press, 1996
- L. Davis. ***The Handbook of Genetic Algorithms*.** Van Nostrand & Reinhold, 1991
- D.B. Fogel. ***Evolutionary Computation*.** IEEE Press, 1995
- D.E. Goldberg. ***Genetic Algorithms in Search, Optimization and Machine Learning*.** Addison-Wesley, 1989
- J. Koza. ***Genetic Programming*.** MIT Press, 1992
- Z. Michalewicz. ***Genetic Algorithms + Data Structures = Evolution Programs*.** Springer Verlag, 3rd ed., 1996
- H.-P. Schwefel. ***Evolution and Optimum Seeking*.** Wiley & Sons, 1995
- J. Holland. ***Adaptation in Natural and Artificial Systems*.** MIT Press 1995

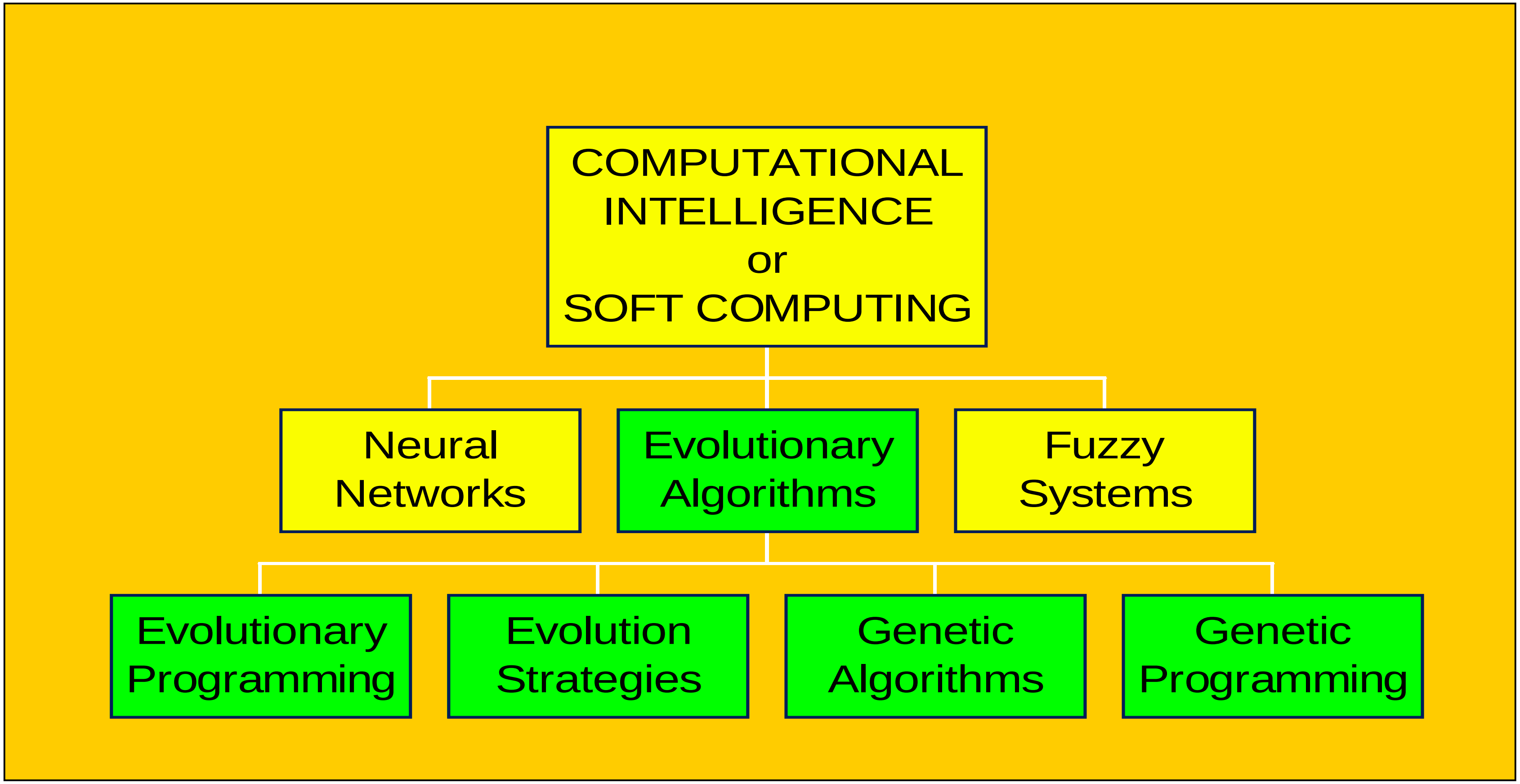
Taxonomy of Evolutionary Algorithms (1)

Evolutionary computing is a family of stochastic search techniques that mimic the natural evolution.



Taxonomy of Evolutionary Algorithms (1)

Evolutionary algorithms as a subdivision of soft computing:

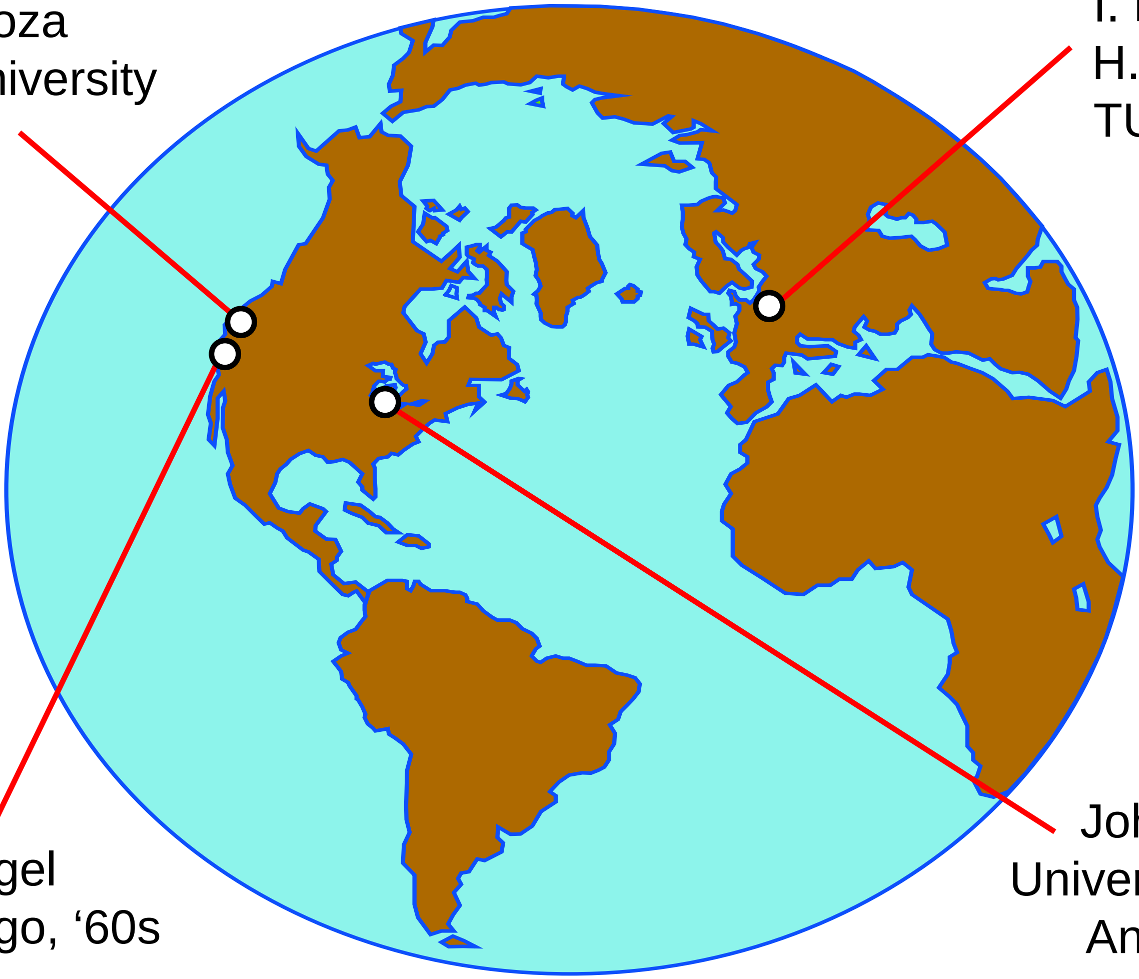


Taxonomy of Evolutionary Algorithms (2)

Distinctive Properties of Evolutionary Algorithms

- verification of correctness of solution ;
- consideration of instances in the population of candidate solutions ;
- deriving solutions from solutions ;
- Probabilistic transition rules

History (1)



John Koza
Stanford University
‘80s

I. Rechenberg,
H.-P. Schwefel
TU Berlin, ‘60s

L. Fogel
UC S. Diego, ‘60s

John H. Holland
University of Michigan,
Ann Arbor, ‘60s

History (2)

1859 Charles Darwin: inheritance, variation, natural selection

1957 G. E. P. Box: random mutation & selection for optimization

1958 Fraser, Bremermann: computer simulation of evolution

1964 Rechenberg, Schwefel: mutation & selection

1966 Fogel et al.: evolving automata - “evolutionary programming”

1975 Holland: crossover, mutation & selection - “reproductive plan”

1975 De Jong: parameter optimization - “genetic algorithm”

1989 Goldberg: first textbook

1991 Davis: first handbook

1993 Koza: evolving LISP programs - “genetic programming”

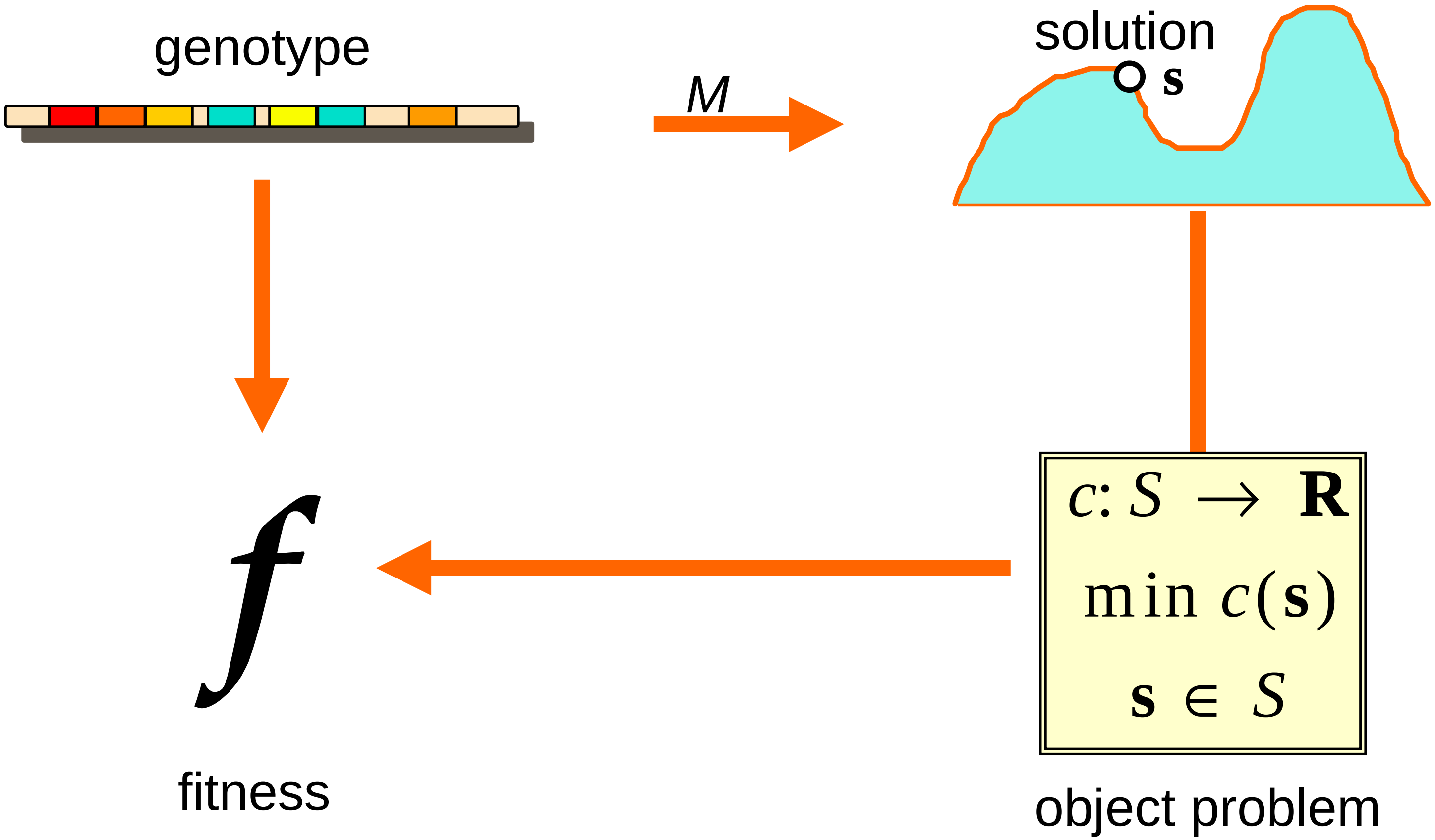
Evolutionary Algorithms Basics

- what an EA is (the Metaphor)
- object problem and fitness
- the Ingredients
- schemata
- implicit parallelism
- the Schema Theorem
- the building blocks hypothesis
- deception

A metaphore

EVOLUTION	PROBLEM SOLVING
Environment	Problem to solve
Population	Set of alternative solutions
Generation	Iteration step
Individual	Candidate solution
Parents	Individuals chosen for reproduction
Adaptation measure	Fitness function
Genotype	String of characters
Phenotype	Decoded solution (circuit,plant)
Metagenesis	Process of transition from the current to the next generation

Object problem and Fitness



genotype

M

solution

$\mathbf{s}$

$$c\colon S \to \mathbf{R}$$

$$\min c(\mathbf{s})$$

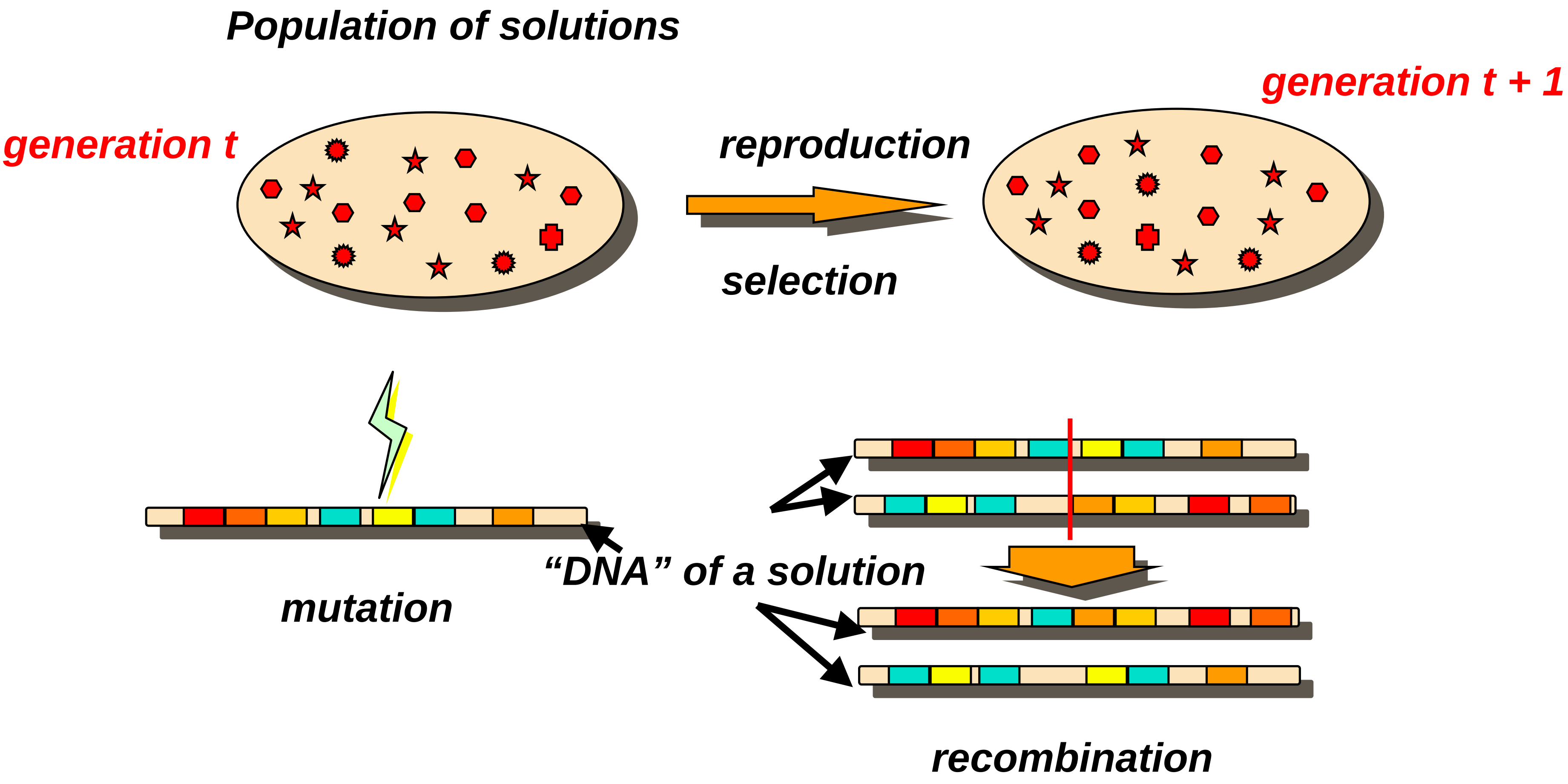
$$\mathbf{s} \in S$$

object problem

f

fitness

Ingredients of an evolutionary algorithm



Population of solutions

generation t

reproduction

selection

generation t + 1

mutation

“DNA” of a solution

recombination

Let us go into more details.....

- EA Structure
- EA Operators
- EA Applications
- Specific types of Eas: GA, GP, ES...
- EA Examples

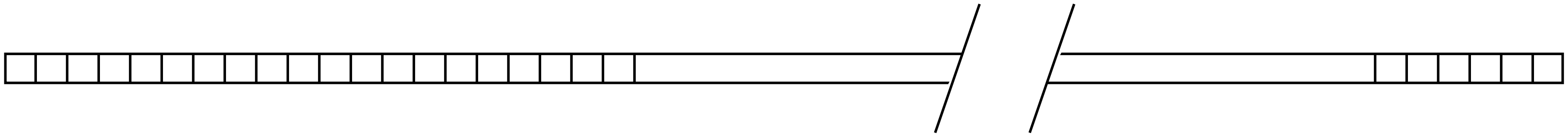
EA Structure

Example 1: Genetic Algorithm for MAXONE problem

- The MAXONE problem
- Genotypes are bit strings
- Fitness-proportionate selection
- One-point crossover
- Flip mutation (transcription error)

The MAXONE Problem

Problem instance: a string of l binary cells, $\gamma \in \{0, 1\}^l$:



Fitness:

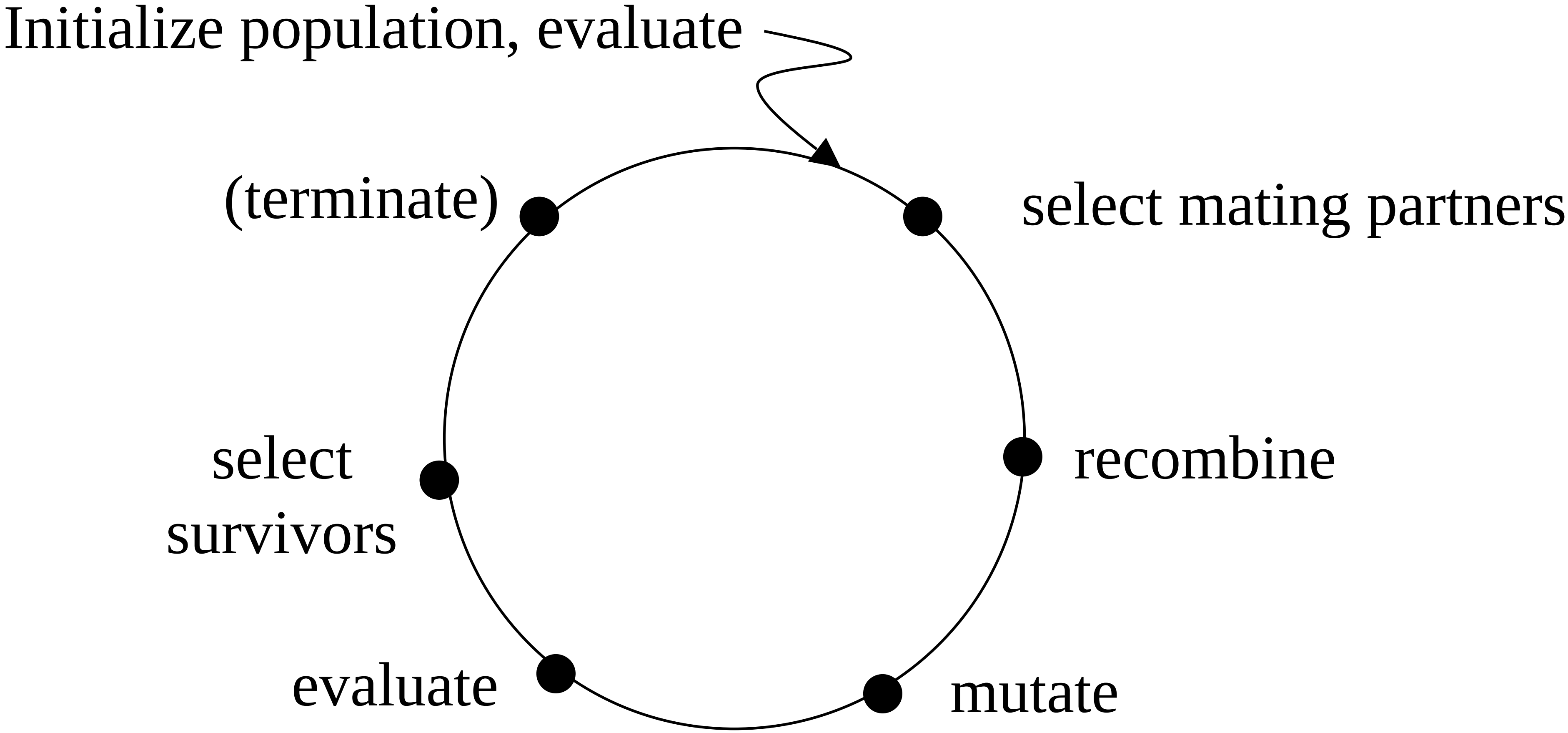
$$f(\gamma) = \sum_{i=1}^{l} \gamma_i$$

Objective: maximize the number of ones in the string.

Genetic Algorithm Inspired by natural evolution

- Population of individuals
  - Individual is feasible solution to problem
- Each individual is characterized by a Fitness function
  - Higher fitness means a better solution
- Based on their fitness, parents are selected to reproduce offspring for a new generation
  - Fitter individuals have more chance to reproduce
  - New generation has same size as old generation; old generation dies
- Offspring has combination of properties of two parents
- If well designed, population will converge to optimal solution

Evolutionary Computing



Initialize population, evaluate

select mating partners

recombine

mutate

evaluate

select survivors

(terminate)

Example 2: Discrete Representation of various type of data in Genetic Algorithms

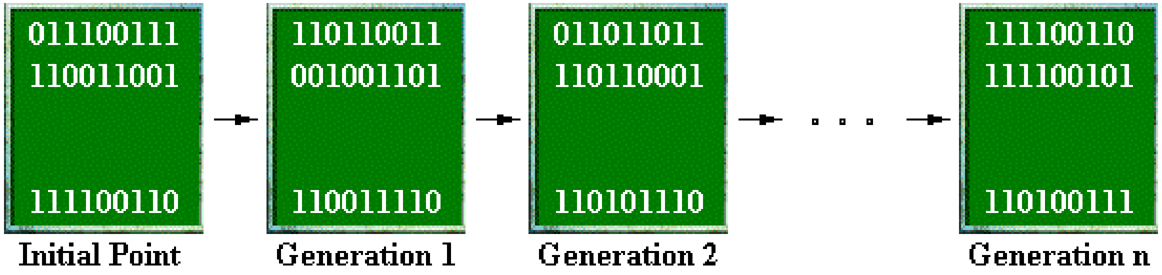
- Genotype: 8 bits

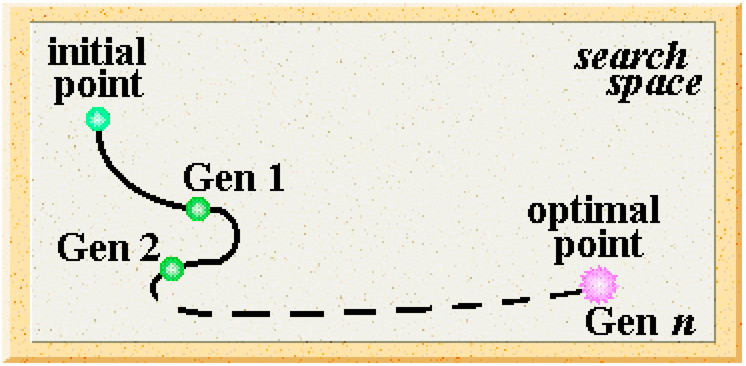
1	0	1	0	0	0	1	1

- Phenotype:
  - integer
    $1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 163$
  - a real number between 2.5 and 20.5
    $2.5 + 163/256\ (20.5 - 2.5) = 13.9609$
  - schedule

GA Process

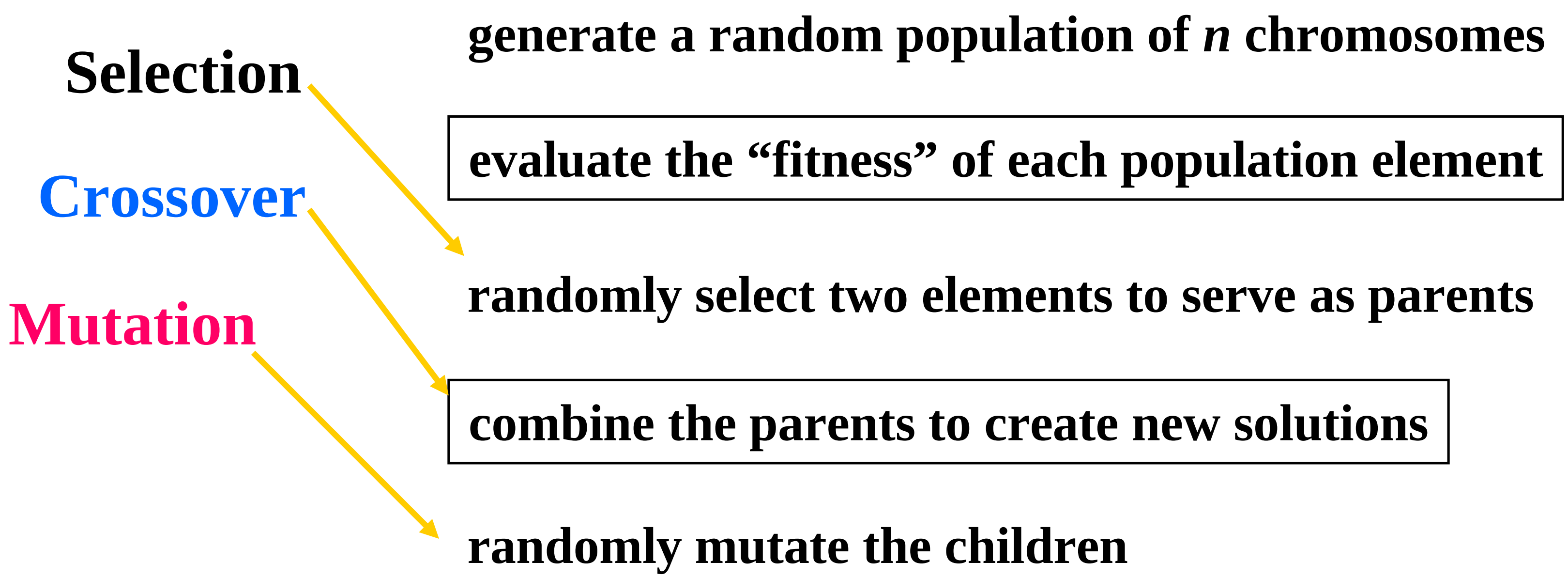
- GAs will cycle through populations of binary strings until a solution is found or a maximum number of generations have been created
- Various stopping criteria





GA Steps

- A GA will implement the following:



Selection

Crossover

Mutation

generate a random population of *n* chromosomes

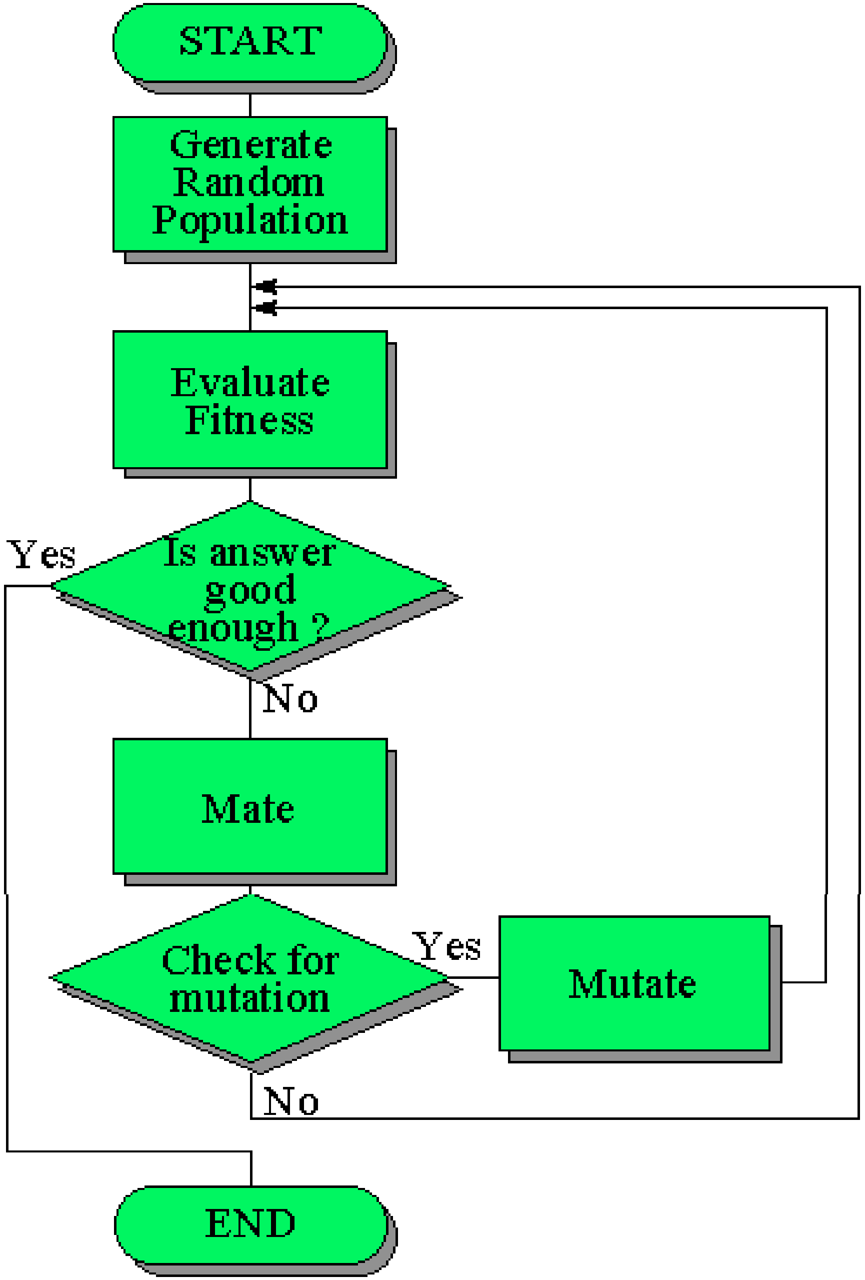
evaluate the “fitness” of each population element

randomly select two elements to serve as parents

combine the parents to create new solutions

randomly mutate the children

GA FLOW CHART



Pseudocode of EA

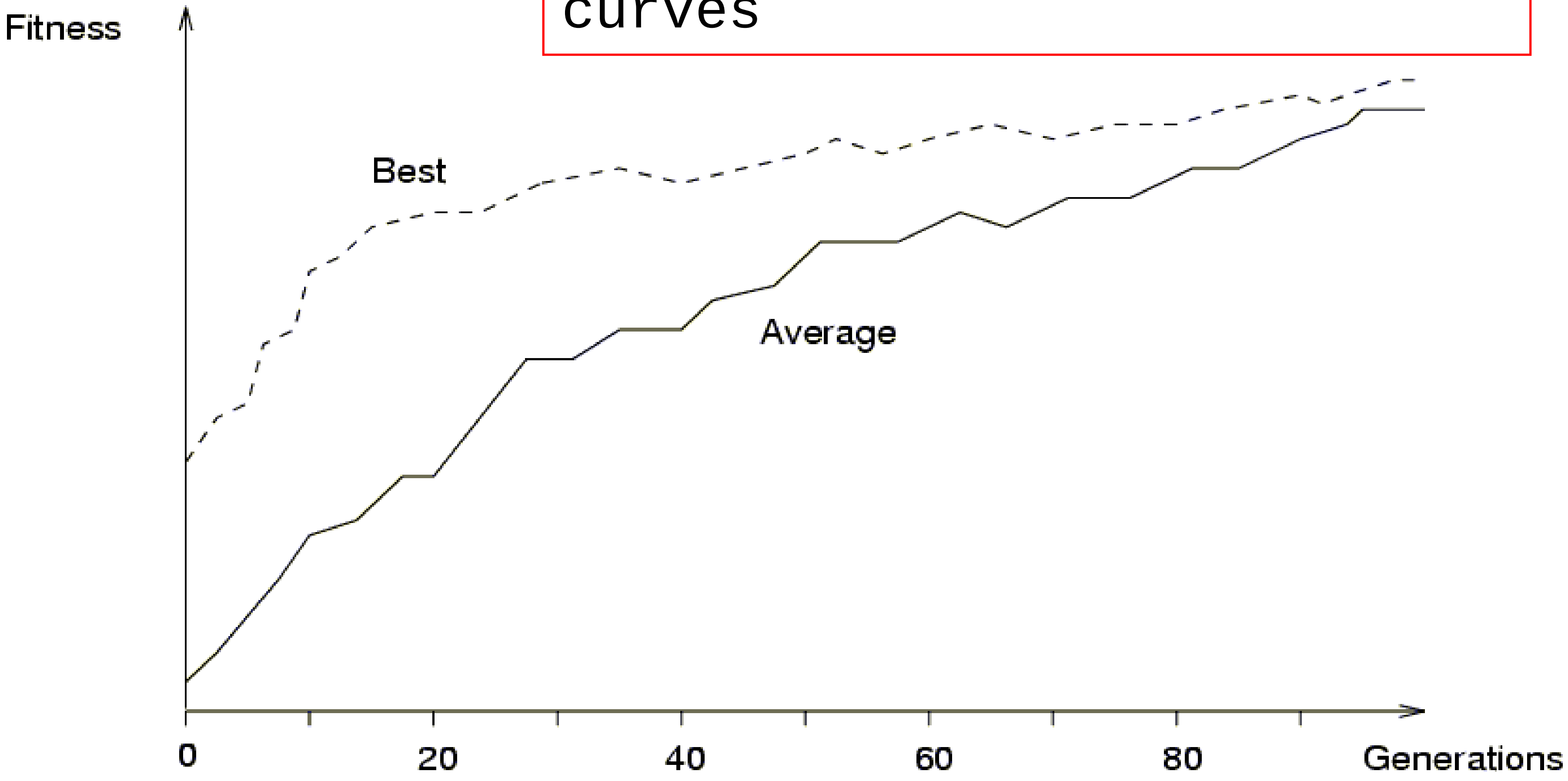
```
generation = 0;
SeedPopulation(popSize);   // at random or from a file
while(!TerminationCondition())
{
    generation = generation + 1;
    CalculateFitness();        // ... of new genotypes
    Selection();               // select genotypes that will reproduce
    Crossover($p_{cross}$);    // mate $p_{cross}$ of them on average
    Mutation($p_{mut}$);       // mutate all the offspring with Bernoulli
                               // probability $p_{mut}$ over genes
}
```

Pseudocode of EA (another variant)

```
BEGIN
  Generate initial population;
  Compute fitness of each individual;
  REPEAT /* New generation /*
    FOR population_size / 2 DO
      Select two parents from old generation;
       /* biased to the fitter ones */
      Recombine parents for two offspring;
      Compute fitness of offspring;
      Insert offspring in new generation
    END FOR
  UNTIL population has converged
END
```

Example of convergence

Usefulness of analyzing such curves



Link between GA and a Problem

- Reproduction mechanisms have no knowledge of the problem to be solved
- Link between genetic algorithm and problem:
  - Coding
  - Fitness function

Basic principles: genotype and phenotype

- An individual is characterized by a set of parameters: Genes
- The genes are joined into a string: Chromosome

- The chromosome forms the genotype
- The genotype contains all information to construct an organism: the phenotype

- Reproduction is a “dumb” process on the chromosome of the genotype
- Fitness is measured in the real world (‘struggle for life’) of the phenotype

Coding

- Design alternative → individual (chromosome)
- Single design choice → gene
- Design objectives → fitness

Each of us is a design alternative, thanks to our parents

Chromosomes are strings of bits, symbols, lists of atoms, etc,

Coding

- Parameters of the solution (genes) are concatenated to form a string (chromosome)
- All kind of alphabets can be used for a chromosome (numbers, characters), but generally a binary alphabet is used
- Order of genes on chromosome can be important
- Generally many different codings for the parameters of a solution are possible
- Good coding is probably the most important factor for the performance of a GA
- In many cases many possible chromosomes do not encode feasible solutions

Example of problem fomulation

- Problem
  - Schedule *n* jobs on *m* processors such that the maximum span is minimized.

Design alternative: job i (i=1,2,…n) is assigned to processor j (j=1,2,…,m)

Individual: A n-vector **x** such that $x_i = 1, \ldots, or\ m$

Design objective: minimize the maximal span

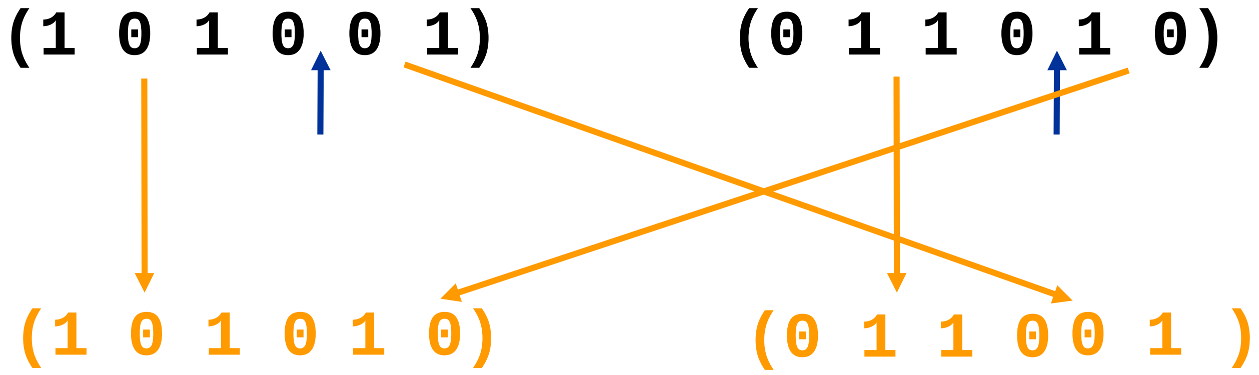
Fitness: the maximal span for each processor

Reproduction

- Reproduction operators
  - Crossover
  - Mutation

Example

- Assume the parents selected are:



(1 0 1 0 0 1)

(0 1 1 0 1 0)

(1 0 1 0 1 0)

(0 1 1 0 0 1)

These are the two children which are now part of the next generation

Find a random crossover point

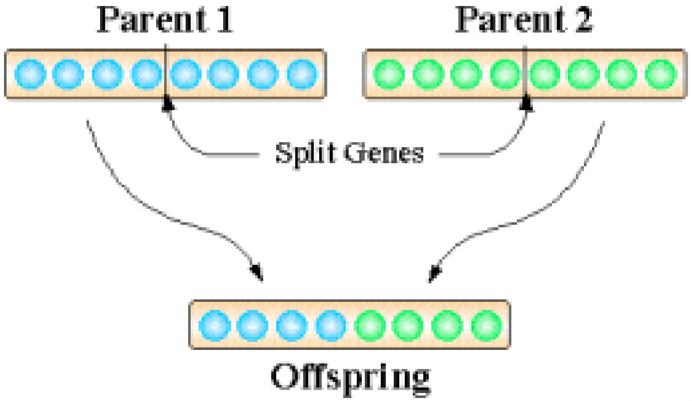
Swap the bits after the crossover point

Crossover algorithm

- Select two random parents

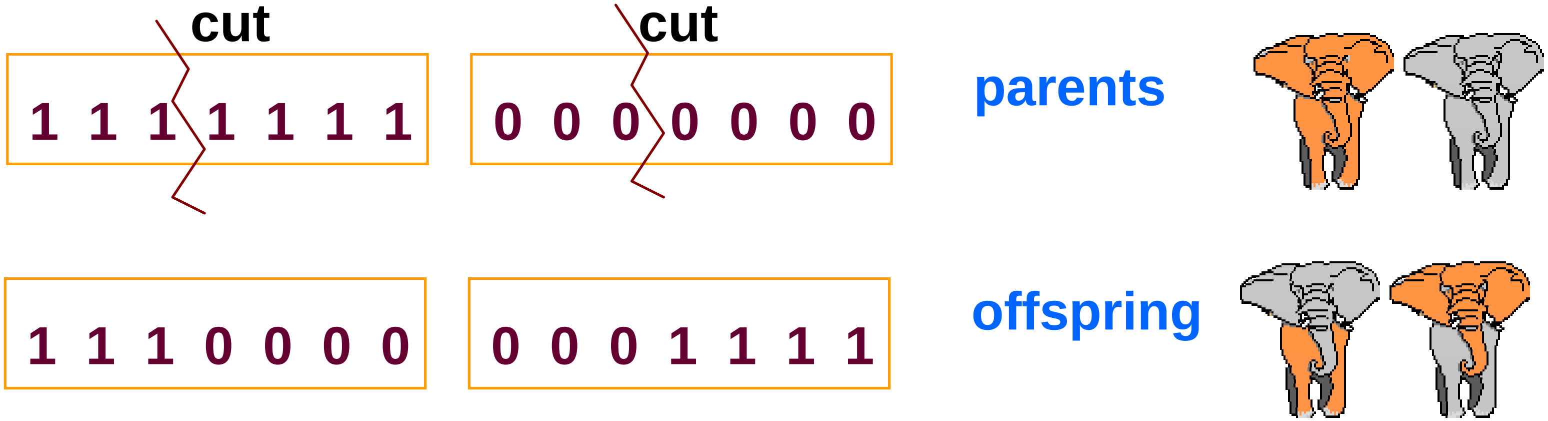
- Using the preset probability of a crossover, p_c, throw a random number, r.
  - if $r < p_c$ then perform a crossover operation on the two parents
  - otherwise pass both parents on to the next generation

- Repeat this process until the next generation is full
- Two parents produce two offspring
  - There is a chance that the chromosomes of the two parents are copied unmodified as offspring
  - There is a chance that the chromosomes of the two parents are randomly recombined (crossover) to form offspring
  - Generally the chance of crossover is between 0.6 and 1.0



Recombination: genotype versus phenotype

- Each chromosome is cut into 2 pieces which are recombined



cut

cut

1 1 1 1 1 1 1

0 0 0 0 0 0 0

parents

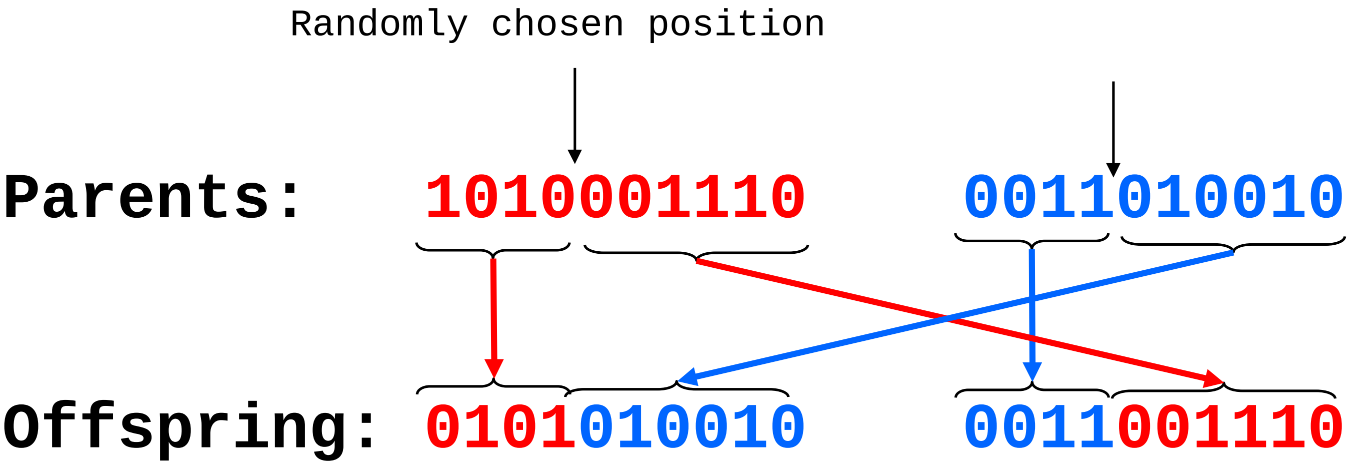
1 1 1 0 0 0 0

0 0 0 1 1 1 1

offspring

One-point crossover 1

- Randomly one position in the chromosomes is chosen
- Child 1 is head of chromosome of parent 1 with tail of chromosome of parent 2
- Child 2 is head of 2 with tail of 1



Randomly chosen position

Parents: 1010001110 0011010010

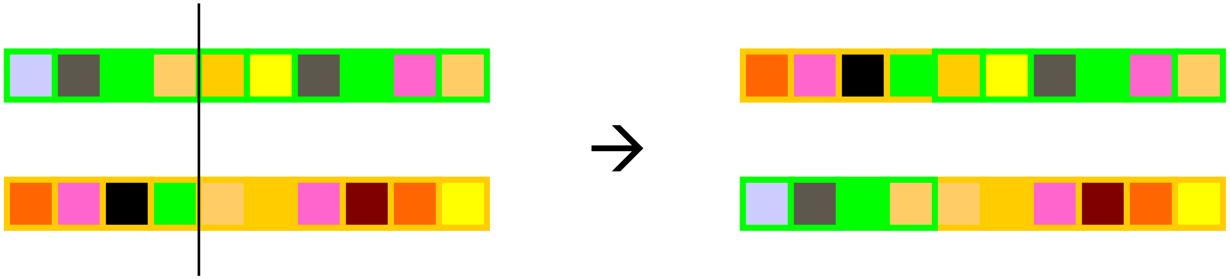
Offspring: 0101010010 0011001110

Generating offspring from two selected parents

1. Single point crossover
2. Two point crossover (Multi point crossover)
3. Uniform crossover

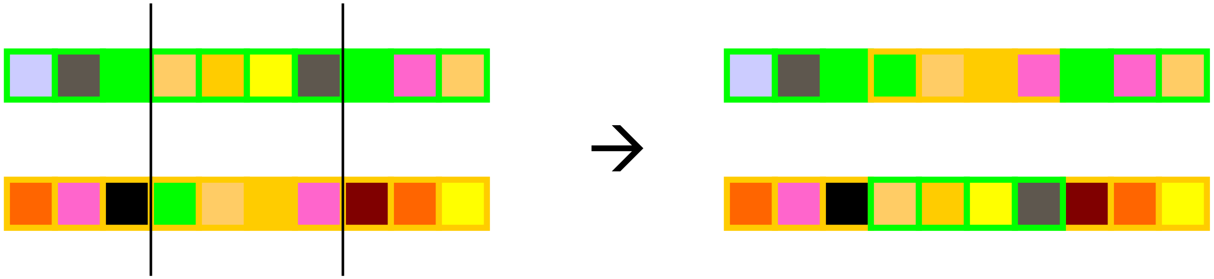
Crosover Operators comparison

- Single point crossover



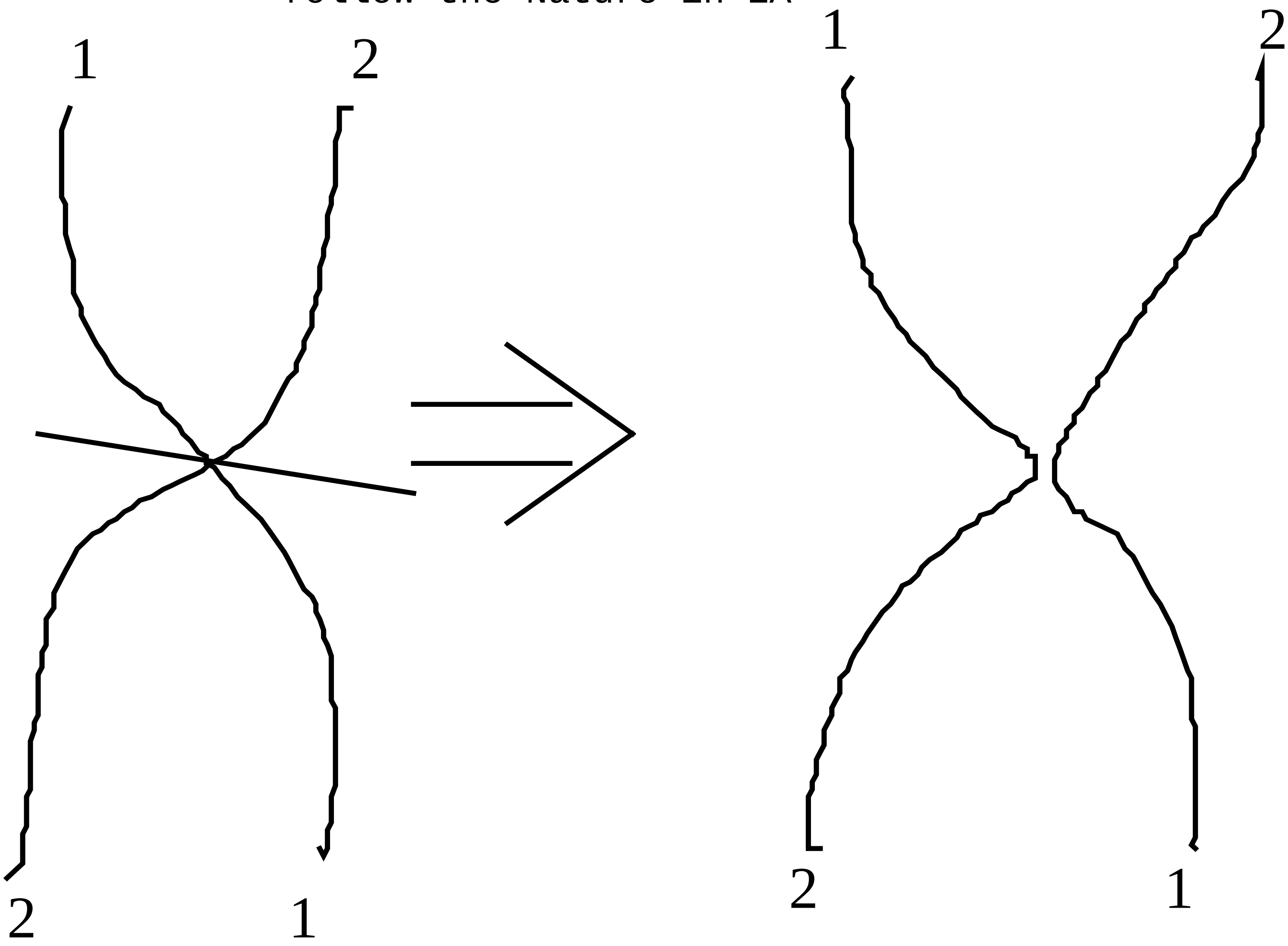
Cross point

- Two point crossover (Multi point crossover)



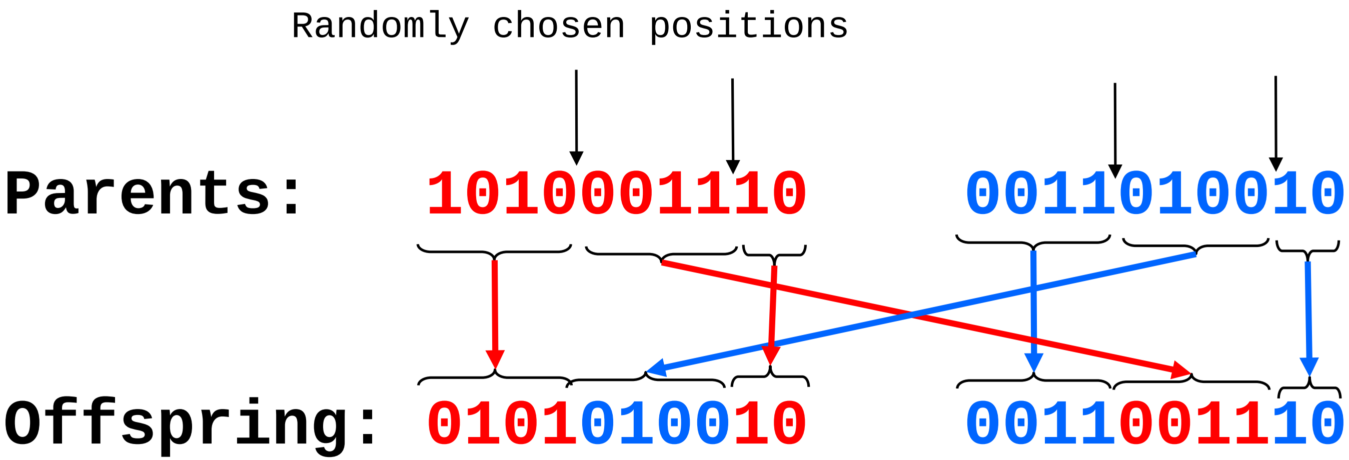
One-point crossover - Nature

But we do not have to
follow the Nature in EA



Two-point crossover

- Randomly two positions in the chromosomes are chosen
- Avoids that genes at the head and genes at the tail of a chromosome are always split when recombined



Randomly chosen positions

Parents: 1010001110 0011010010

Offspring: 0101010010 0011001110

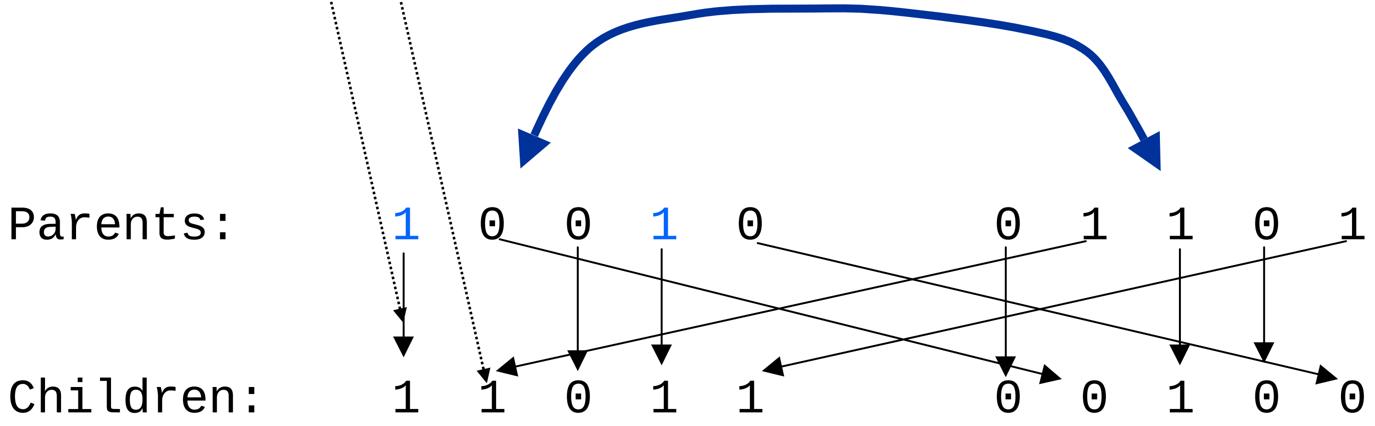
Remember the elephants example?

Uniform Crossover

- **PROCESS**:
  - each bit in the first offspring is selected sequentially from <u>parent 1</u> with probability p and from <u>parent 2</u> with probability (1-p),
  - the second offspring receives the bit not selected for the first offspring.
  - Probabilities of next bits can be constrained on previous values and choices

- EXAMPLE:
- P(parent1) = 0.9
- P(parent1) = 0.3

These choices correspond to create dynamically a mask for both parents



Parents: 1 0 0 1 0 0 1 1 0 1

Children: 1 1 0 1 1 0 0 1 0 0

Uniform crossover (other variant)

- 1. A **random mask** is generated
- 2. The mask determines which bits are copied from one parent and which from the other parent
  - Bit density in mask determines how much material is taken from the other parent (takeover parameter)
- This mask may be biased - not totally random but user-influenced.

Mask:	**0110011000**	**(Randomly generated)**
Parents:	**1010001110**	**0011010010**
Offspring:	**0011001010**	**1010010110**

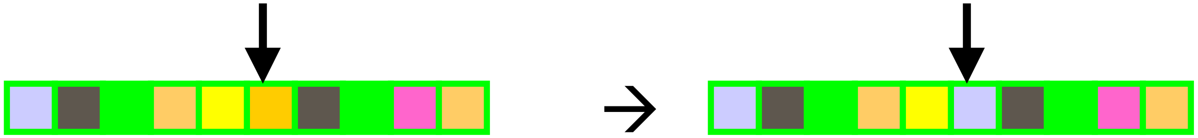
- Is uniform crossover better than single crossover point?
  - Trade off between
    - Exploration: introduction of new combination of features
    - Exploitation: keep the good features in the existing solution

Problems with crossover

- Depending on coding, simple crossovers can have high chance to produce illegal offspring
  - E.g. in **TSP** with simple binary or path coding, most offspring will be illegal because not all cities will be in the offspring and some cities will be there more than once
- Uniform crossover can often be modified to avoid this problem
  - E.g. in TSP with simple path coding:
    - Where mask is 1, copy cities from one parent
    - Where mask is 0, choose the remaining cities *in the order of the other parent*

Mutation

- Generating new offspring from single parent



- This operator helps maintaining the diversity of the individuals
  - Crossover can only explore the combinations of the current gene pool
  - Mutation can “generate” new genes

Mutation (cont)

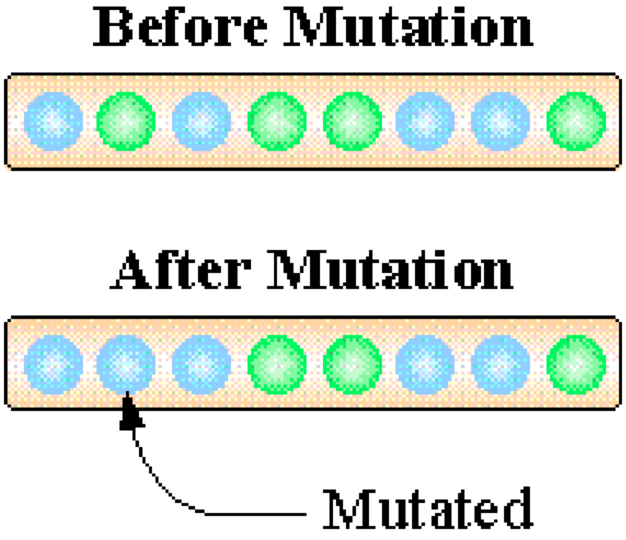
- As each chromosome is added to the next generation, it is examined bit by bit
  - each time a bit is examined, a random number is thrown, r
  - if $r < P_m$ then that bit is complemented otherwise it is left unchanged

- The whole cycle begins again - it will stop when either a solution is found or the maximum number of generations have been produced

There is a chance that a gene of a child is changed randomly
Generally the chance of mutation is low (e.g. 0.001)

Example: Binary Mutation

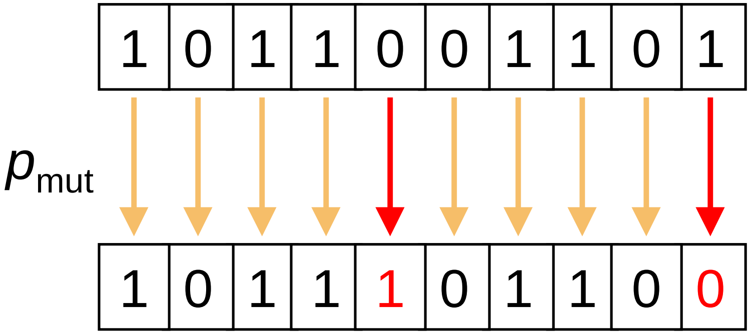
- A bit in a child is changed (from 1 to 0 or from 0 to 1) at random



This is a usually small probability event but in some approaches it may be quite high

The effect is to prevent a premature convergence to a local minimum or maximum

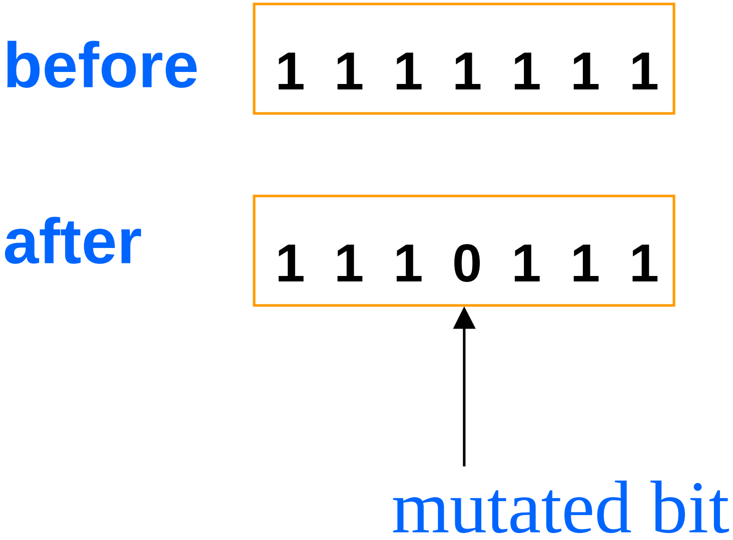
Mutation



independent Bernoulli transcription errors

Sometimes there is high mutation rate and its probability is generated for each gene, dynamically with analyzing previous changes. Many bits can change in one generation.

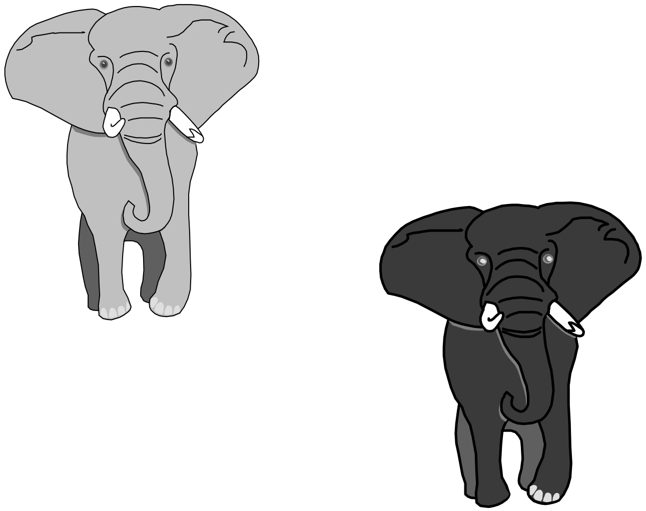
Example: Binary Mutation



before 1 1 1 1 1 1 1

after 1 1 1 0 1 1 1

mutated bit



Mutation happens with probability p_m for each bit

Reproduction Operators

- **Control parameters:** **population size, crossover / mutation probability**
  - These parameters are problem specific
  - P1. Increase population size
    - Increase diversity and computation time for each generation
  - P2. Increase crossover probability
    - Increase the opportunity for recombination but also disruption of good combination
  - P3. Increase mutation probability
    - Closer to randomly search
    - Help to introduce new gene or reintroduce the lost gene
  - P4. Vary the population
    - Usually using crossover operators to recombine the genes to generate the new population, then using mutation operators on the new population

EA Performance: Diversity

- Increasing diversity by genetic operators
  - mutation
  - Recombination

- Decreasing diversity by selection
  - of parents
  - of survivors

GA: crossover OR mutation?

If we define distance in the search space as Hamming distance then:

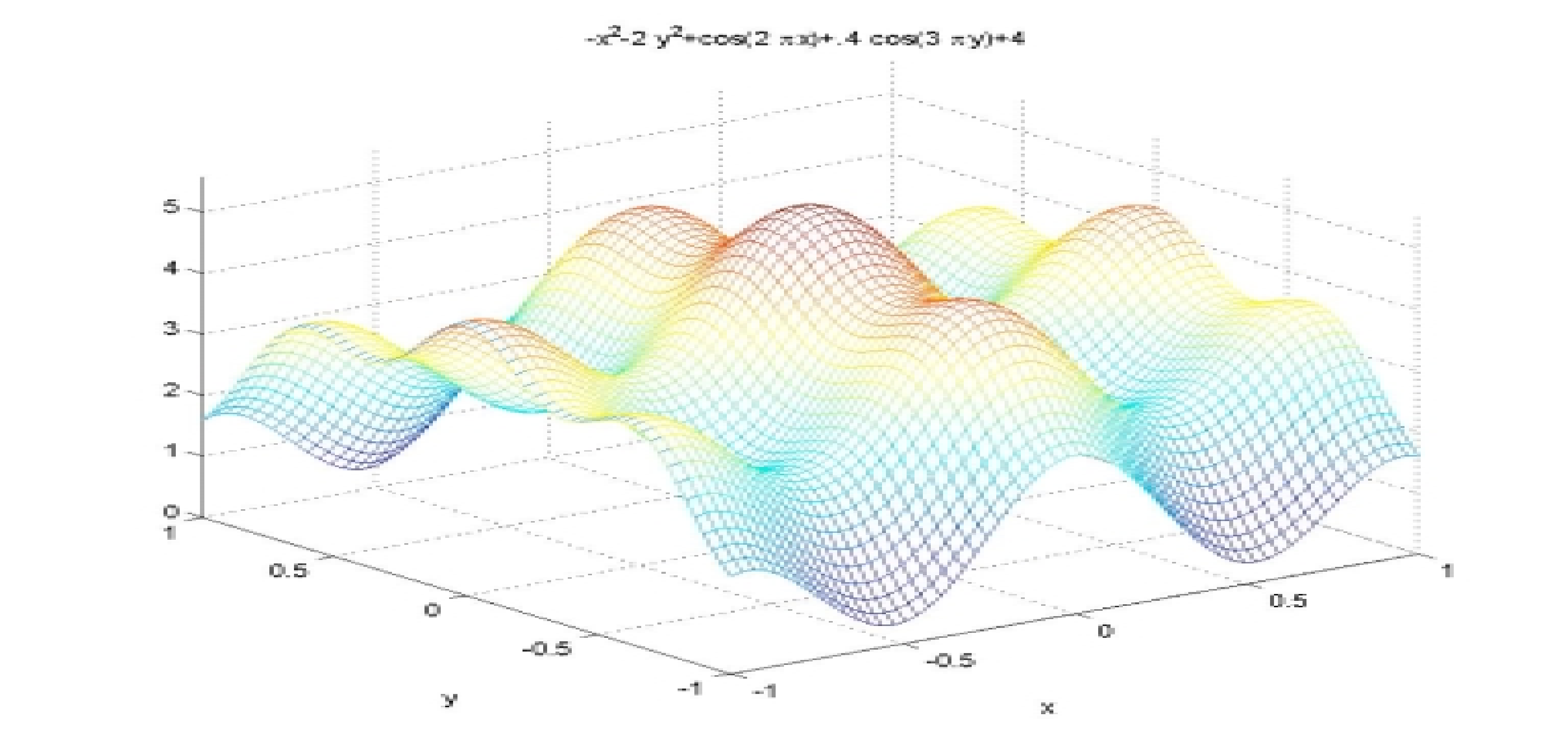
- Crossover is **explorative**, it makes a *big* jump to an area somewhere ‘in between’ two (parent) areas.
- Mutation is **exploitative**, it creates random *small* variations, thereby staying near the parent.
- To hit the optimum you often need a lucky mutation.
- GA community: crossover is mission critical.

Genetic Algorithm for function optimization

Numerical Example:

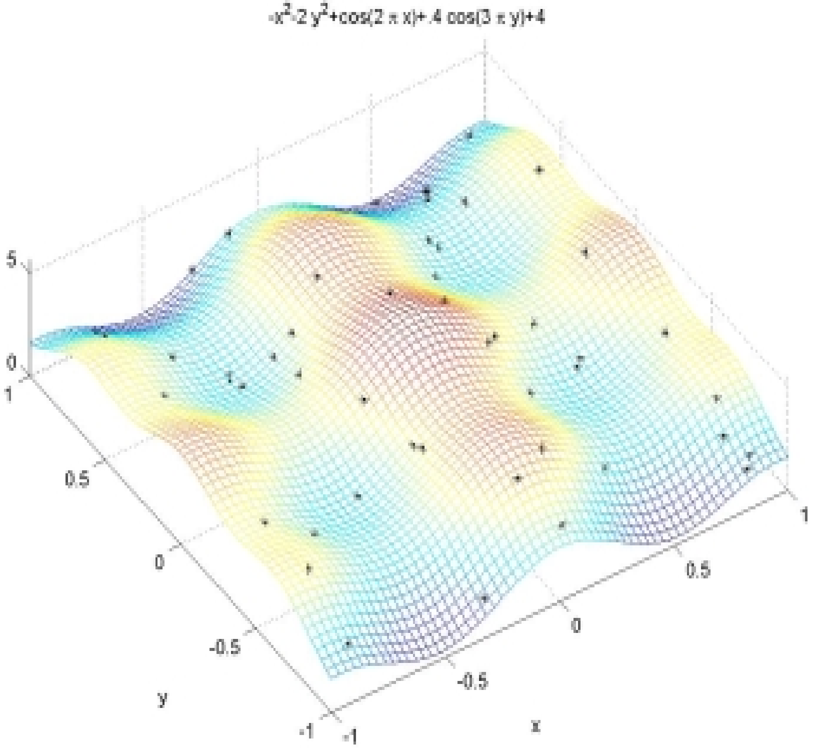
$$F(x,y) = -x^2 - 2y^2 + \cos(3\pi x) + 0.3\cos(4\pi y) + 4$$

$$-1 \le x \le 1 \qquad -1 \le y \le 1$$

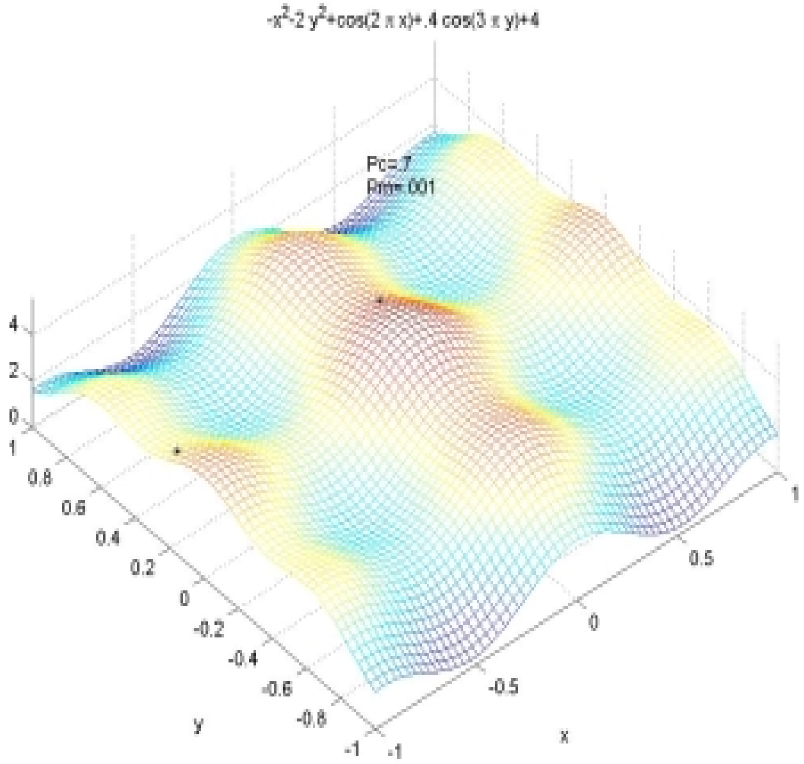


Genetic Algorithm for function optimization

$$s_i = [\underbrace{1\quad 0\quad 0\quad 1\quad 0\quad 1\quad 1\quad 1}_{x}\ \underbrace{0\quad 0\quad 1\quad 1\quad 1\quad 0\quad 1\quad 1}_{y}]$$

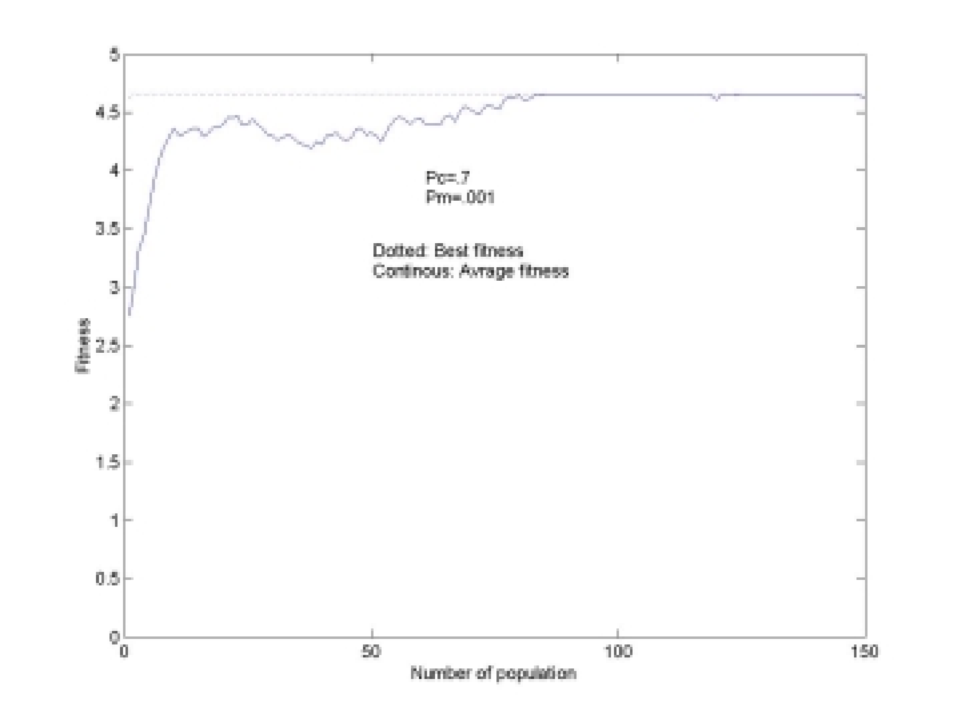


First Generation of 50



After 150 Generation

Genetic Algorithm for function optimization



Evolution of average fitness

Parent selection

- The process in which individual strings in the population are selected to contribute to the next generation is called **parent selection**
    - based on fitness
    - strings with a high fitness have a higher probability of contributing one or more offspring to the next generation

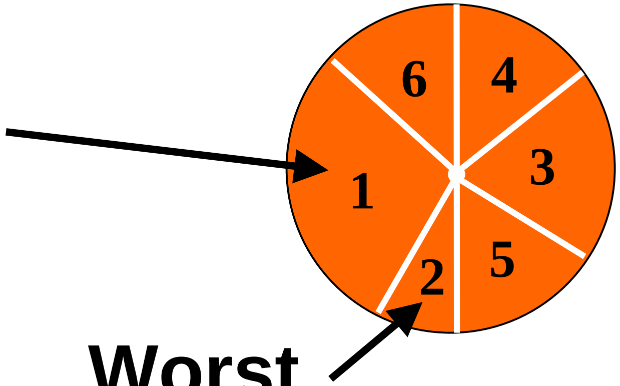
- Example: Biased **Roulette Wheel** Selection

Specifically: Fitness proportionate selection

- Expected number of times chromosome with f_i is selected equals f_i ***/ average fitness***

When you spin the wheel, items 1 and 5 have the greatest chance of coming up while item 2 has the smallest

Best individuals obtain this much

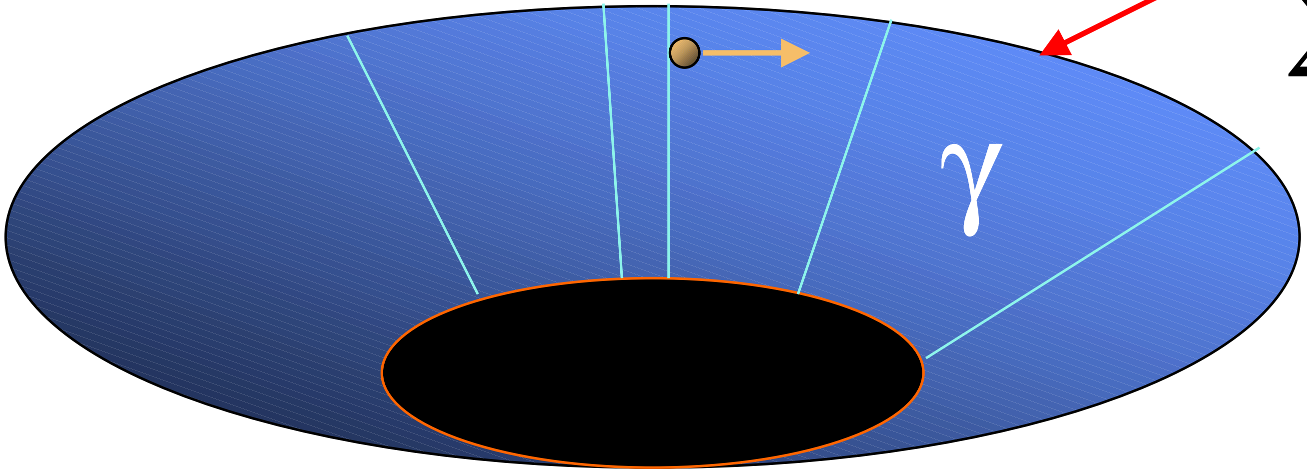


Worst

Fitness Proportionate Selection

Probability of γ being selected: $$P(\gamma) = \frac{f(\gamma)}{\sum f}$$

Implementation: "Roulette Wheel"



$$2\pi \frac{f(\gamma)}{\sum f}$$

Example: Selection for MAXONE

Number of bits 1
in the string

Cumulative f

Probability of
being selected

0111011011	f = 7	Cf = 7	P = 0.125	●●
1011011101	f = 7	Cf = 14	P = 0.125	
1101100010	f = 5	Cf = 19	P = 0.089	●●
0100101100	f = 4	Cf = 23	P = 0.071	●
1100110011	f = 6	Cf = 29	P = 0.107	●
1111001000	f = 5	Cf = 34	P = 0.089	
0110001010	f = 4	Cf = 38	P = 0.071	●●
1101011011	f = 7	Cf = 45	P = 0.125	●●
0110110000	f = 4	Cf = 49	P = 0.071	
0011111101	f = 7	Cf = 56	P = 0.125	

hits

Random sequence: 43, 1, 19, 35, 15, 22, 24, 38, 44, 2

Example continued: Recombination & Mutation

0111011011 → 0111011011 → 0111**1**11011 f = 8

0111011011 → 0111011011 → 0111011011 f = 7

110|1100010 → 1100101100 → 1100101100 f = 5

010|0101100 → 0101100010 → 0101100010 f = 4

1|100110011 → 1100110011 → 1100110011 f = 6

1|100110011 → 1100110011 → 1**0**00110011 f = 5

0110001010 → 0110001010 → 0110001010 f = 4

1101011011 → 1101011011 → 1101011011 f = 7

011000|1010 → 0110001011 → 0110001011 f = 5

110101|1011 → 1101011010 → 1101011010 f = 6

TOTAL = 57

Total increased in
next generation

Parent selection: another Example

Roulette wheel

- Sum the fitness of all chromosomes, call it T
- Generate a random number N between 1 and T
- Return chromosome whose fitness added to the running total is equal to or larger than N
- Chance to be selected is exactly proportional to fitness

Chromosome:	1	2	3	4	5	6
Fitness:	8	2	17	7	4	11
Running total:	8	10	27	34	38	49
N (1 ≤ N ≤ 49):			23			
Selected:			3			

Parent/Survivor Selection

- Strategies
  - Survivor selection:
    - 1. Always keep the best one,
    - 2. Elitist: deletion of the K worst
    - 3. Probability selection: inverse to their fitness
    - 4. Etc.

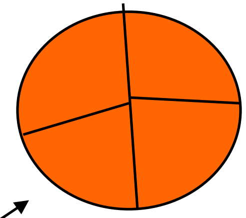
Parent/Survivor Selection

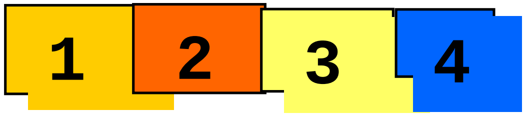
- Too strong fitness selection bias can lead to sub-optimal solution
- Too little fitness bias selection results in unfocused and meandering search

Give examples
from biology and
society

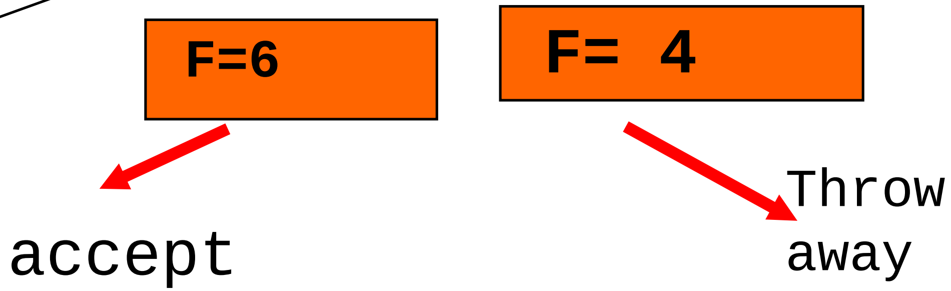
Parent selection

- In Roulette Wheel the chance to be selected as a parent are proportional to fitness
- If values change by little, the mechanism becomes near random.
- Thus other methods may be used, based on ranking proportional (linear selection) or on tournament



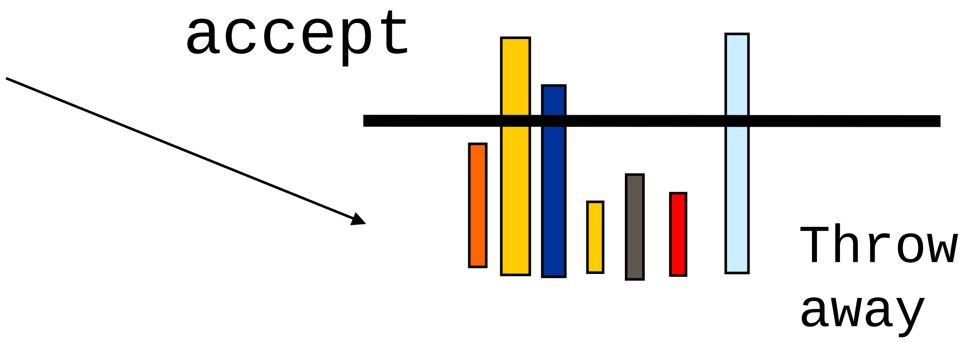


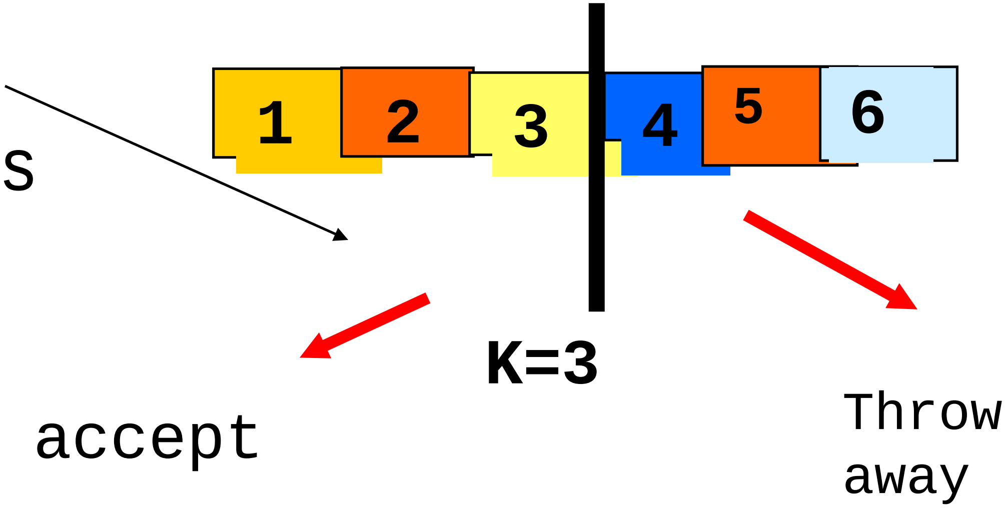
Select two randomly



Parent selection

- **Threshold method** - select those who have fitness above some value T
- Elitist method - kill all but k best fit individuals





Tournament Selection Methods

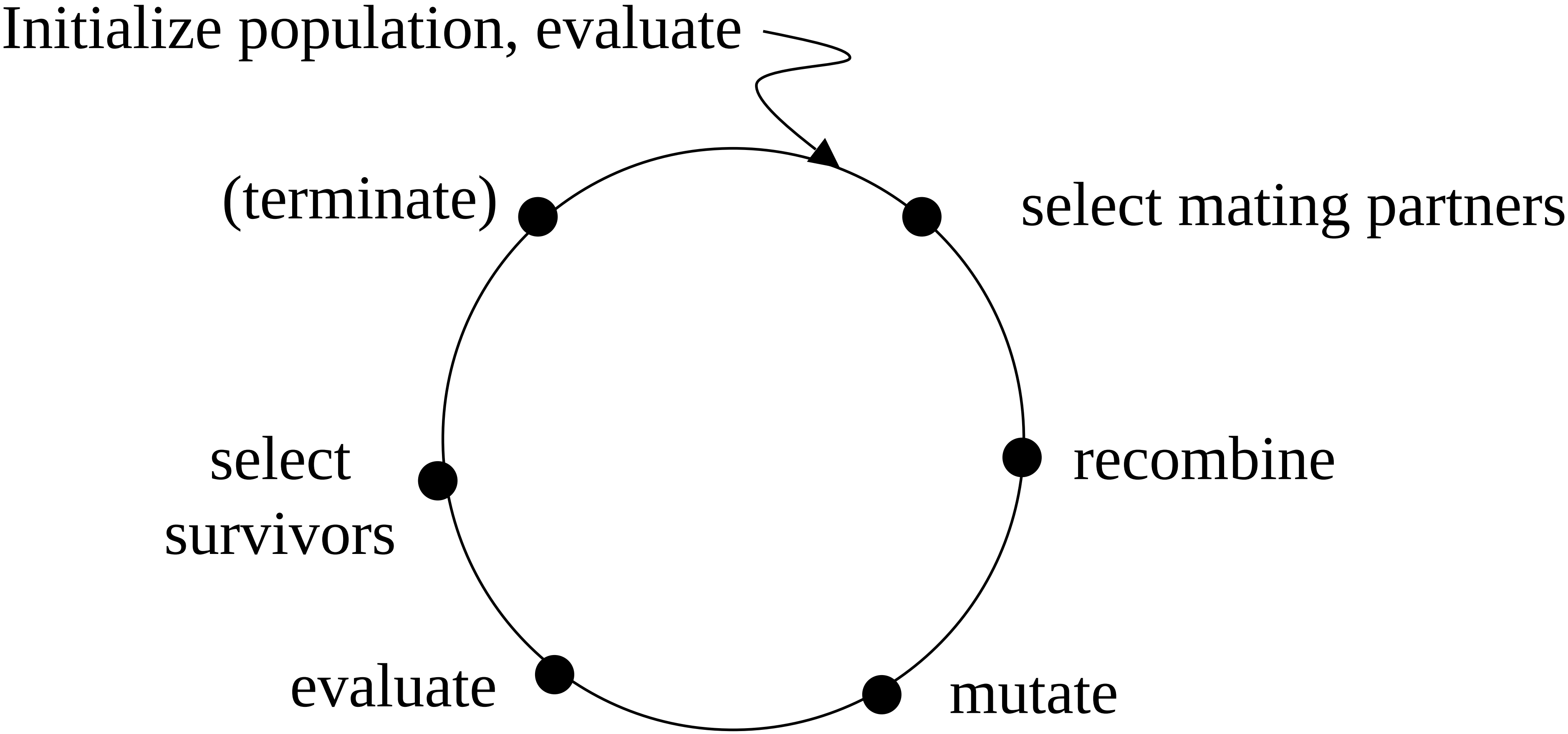
- Binary tournament
  - Two individuals are randomly chosen; the fitter of the two is selected as a parent
- Probabilistic binary tournament
  - Two individuals are randomly chosen; with a chance p, $0.5<p<1$, the fitter of the two is selected as a parent
- Larger tournaments
  - n individuals are randomly chosen; the fittest one is selected as a parent

- By changing n and/or p, the GA can be adjusted dynamically

Mixed Parent/Survivor Selection Strategies

- Strategies
  - Parent selection
    - Uniform randomly selection
    - Probability selection : proportional to their fitness
    - Tournament selection (Multiple Objectives)
      - Build a small comparison set
      - Randomly select a pair with the higher rank one beats the lower one
        - Non-dominated one beat the dominated one
        - **Niche count**: the number of points in the population within certain distance, higher the niche count, lower the rank.
    - Etc.

Evolutionary Computing



Initialize population, evaluate

select mating partners

recombine

mutate

evaluate

select survivors

(terminate)

Fitness Function

- A key component in GA
- Time/quality trade off
- Multi-criterion fitness

Purpose

- Parent selection
- Measure for convergence
- For Steady state: Selection of individuals to die

- Should reflect the value of the chromosome in some "real" way
- Next to coding the **most critical** part of a GA

Problems with fitness range

- **Problem #1.** Premature convergence
  - ΔFitness too large
  - Relatively superfit individuals dominate population
  - Population converges to a local maximum
  - Too much exploitation; too few exploration
- **Problem # 2.** Slow finishing
  - ΔFitness too small
  - No selection pressure
  - After many generations, average fitness has converged, but no global maximum is found; not sufficient difference between best and average fitness
  - Too few exploitation; too much exploration

Thus we want ΔFitness to be not too small, not too large

What are the solutions to solve these problems with fitness range?

- Use tournament selection
  - Implicit fitness remapping
- Adjust fitness function for roulette wheel
  - Explicit fitness remapping
    - Method 1: Fitness scaling
    - Method 2: Fitness windowing
    - Method 3: Fitness ranking

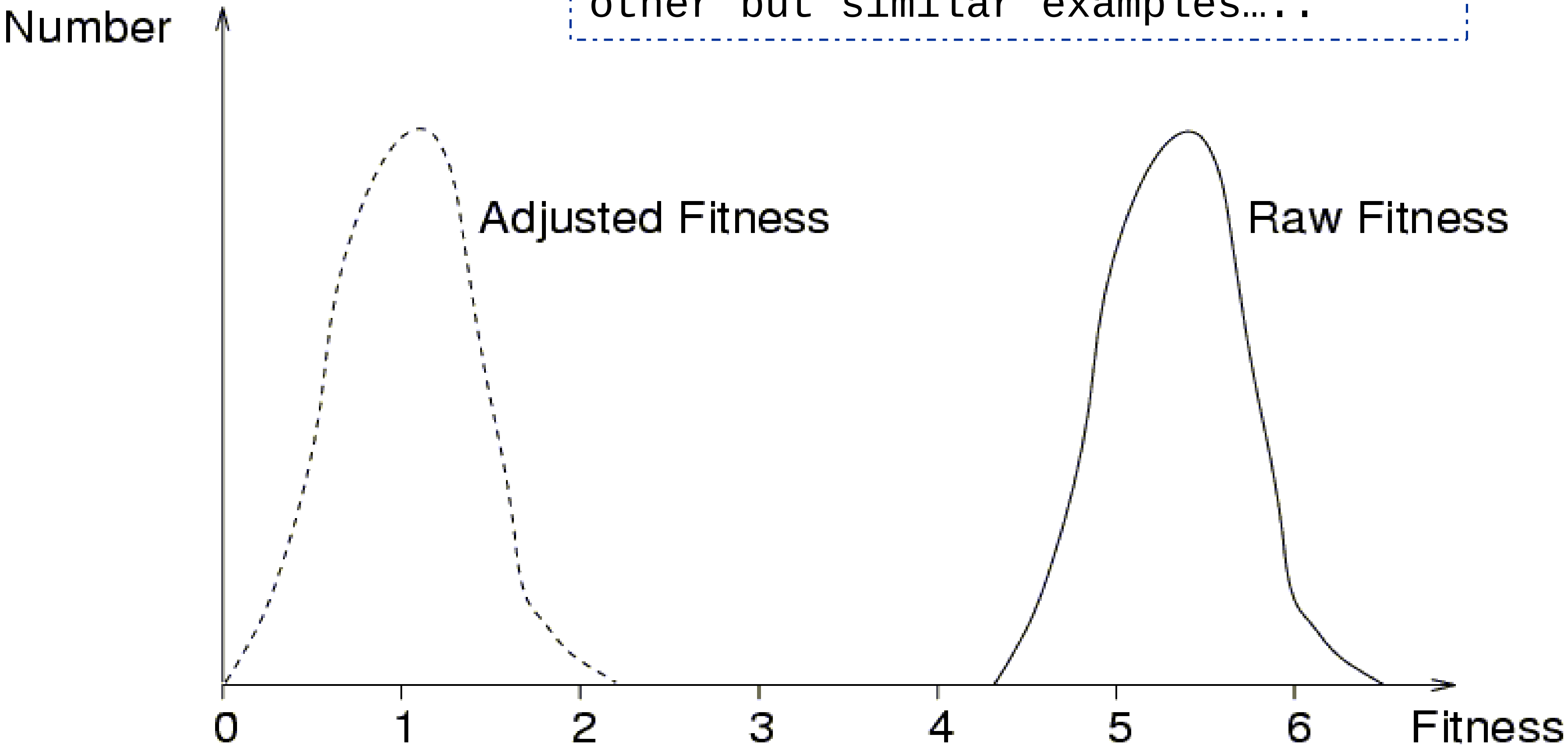
Will be explained below

Method 1: Fitness scaling

- Fitness values are scaled by subtraction and division
  - so that worst value is close to 0 and the best value is close to a certain value, typically 2
    - Chance for the most fit individual is 2 times the average
    - Chance for the least fit individual is close to 0
- There are problems when the original maximum is very extreme (super-fit)
  - or when the original minimum is very extreme (super-unfit)
    - Can be solved by defining a minimum and/or a maximum value for the fitness

Example of Fitness Scaling

… this is a kind of normalization of shape that allows to make decisions based on statistics of other but similar examples…..



Method 2: Fitness windowing

Other popular method to deal with fitness function range

- The method is the same as window scaling, except the amount subtracted is the minimum *observed in the n previous* generations.
- For instance, take $n = 10$
- There exist the same problems as with scaling

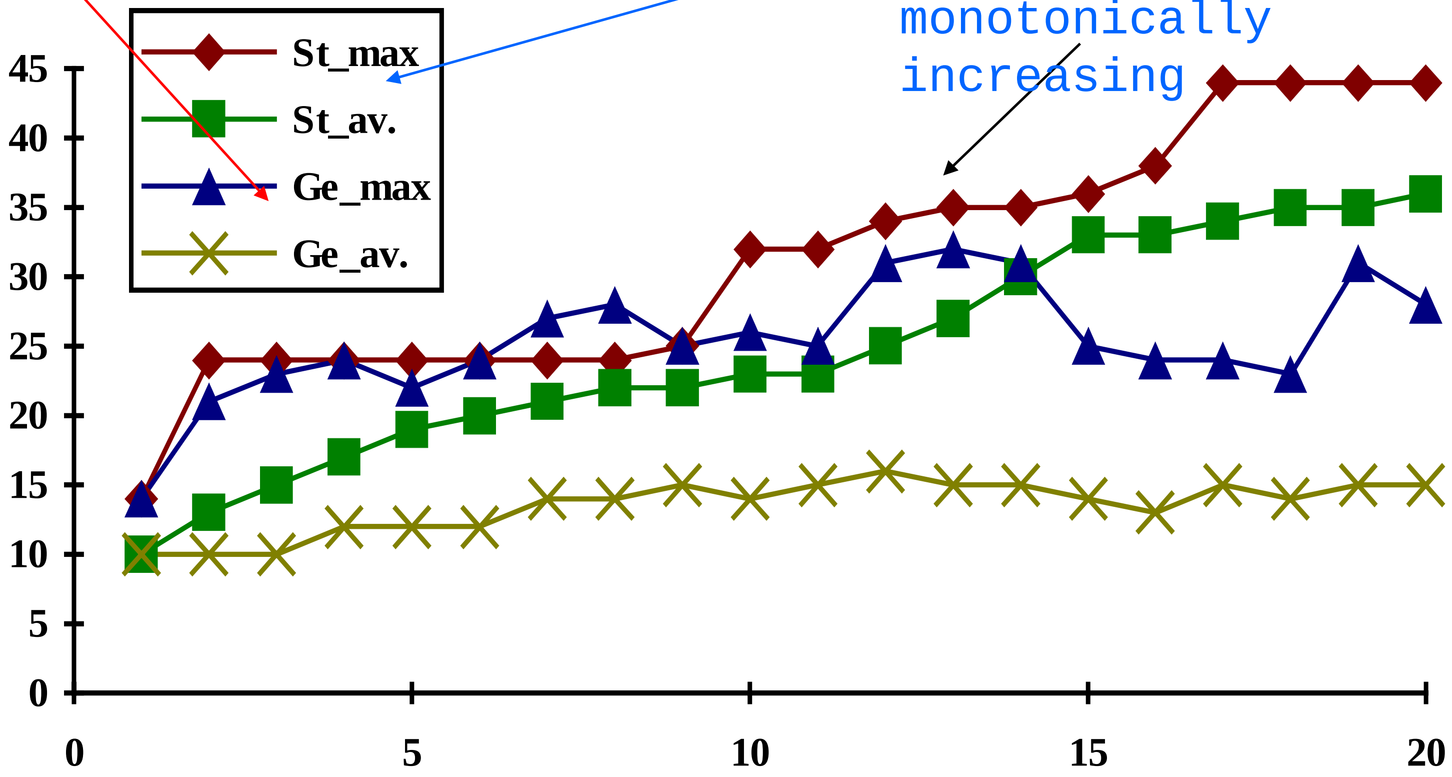
Method 3: Fitness ranking

- Individuals are numbered in order of increasing fitness
- The rank in this order is the adjusted fitness
- Starting number and increment can be chosen in several ways, and they influence the results

- No problems with super-fit or super-unfit
- This method is often superior to scaling and windowing

Observe the fitness in generations: Example run

Maxima and Averages of steady state (St) and generational (Ge) replacement



This line is monotonically increasing

Fitness Landscapes

Landscapes are related to selection and fitness functions

- Selection moves organisms over a landscape to find peaks

- Peak indicates high level of fitness

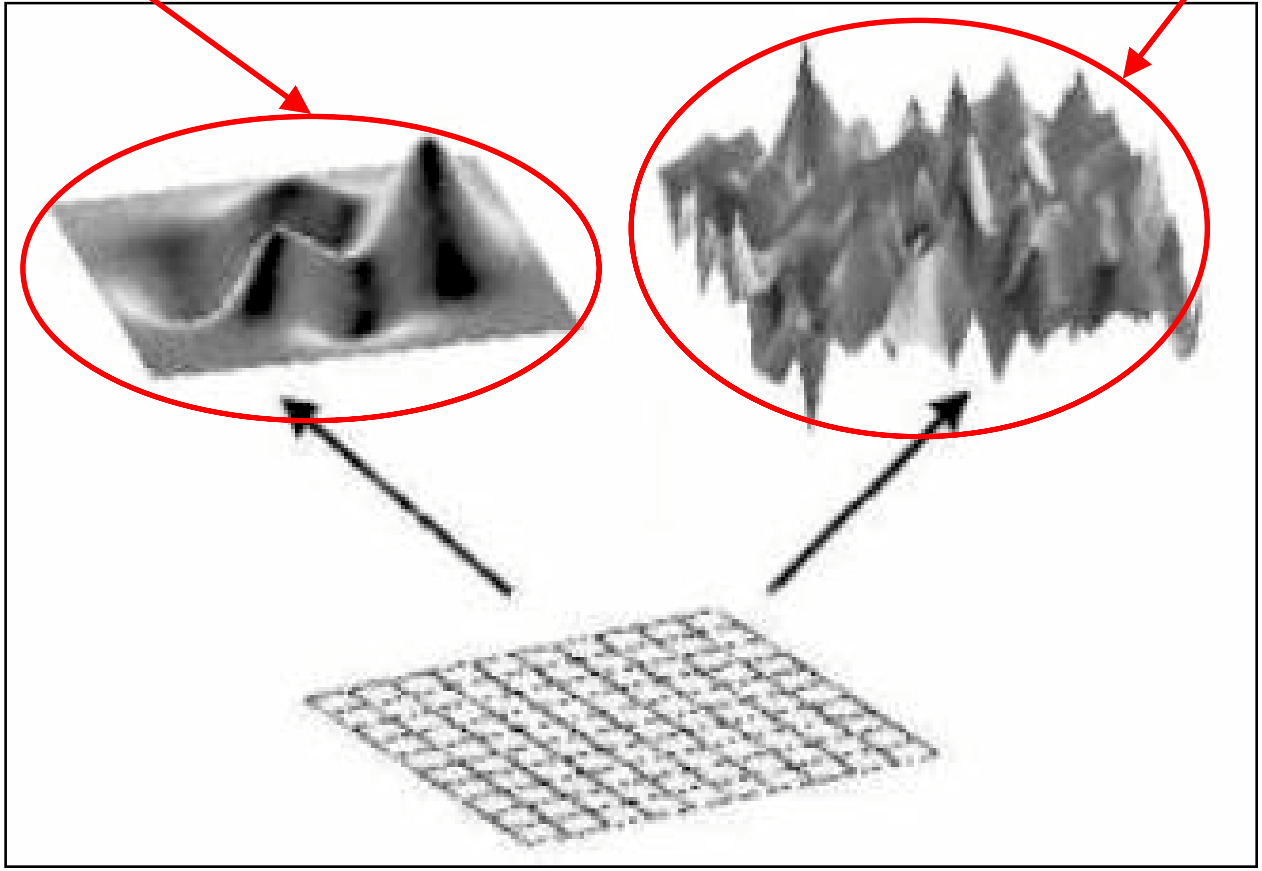
- Some numbers about the space of such a landscape:

  - image number of genes = 7

  - then 2^7 = 128 different *genotypes*

  - E. Coli has about 3,000 genes, thus 10^{900} possible genotypes

Selection - Fitness Landscapes



Smooth landscape

Rugged landscape

Discuss analogy to travellers who want to find highest peak in Himalaya Mountains

Selection - Fitness Landscapes

Complex organisms -> complex fitness landscape

Problem!

- Random fitness landscape
- Adaptive walk
- Correlated landscapes
- theoretical model -> very nice!

Multi-Criterion Fitness

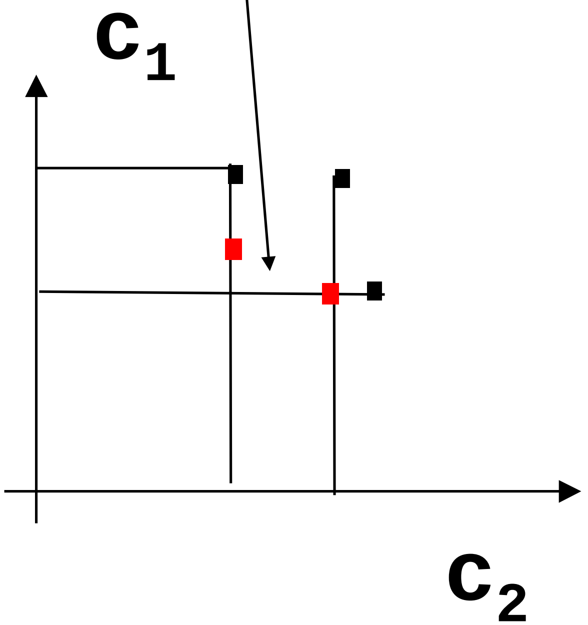
. . . Sometimes there is no single best solution criterion. . . .

- Pareto Optimal Set
  - If there exists no solution in the search space which dominates any member in the set *P*, then the solutions belonging the the set *P* constitute a global Pareto-optimal set.
  - Pareto optimal front
- Dominance Check

Minimizing

c_1 and c_2

red are Pareto Points



Multi-Criterion Fitness

- Dominance and indifference
  - For an optimization problem with more than one objective function (f_i, $i=1,2,\ldots n$)
  - given any two solution X_1 and X_2, then
    - X_1 dominates X_2 ($X_1 \succ X_2$), if

      $f_i(\boldsymbol{X_1}) >= f_i(\boldsymbol{X_2})$, *for all* $i = 1,\ldots,n$
    - X_1 is indifferent with X_2 ($X_1 \sim X_2$), if X_1 does not dominate X_2, and X_2 does not dominate X_1

Maximizing f

Minimization and maximization problems can be easily converted to one another

Multi-Criterion Fitness

- Weighted sum
  - $F(\mathbf{x}) = w_1 f_1(x_1) + w_2 f_2(x_2) + \ldots + w_n f_n(x_n)$
  - *Problems?*
    - *Convex and convex Pareto optimal front*

      Sensitive to the shape of the Pareto-optimal front
    - *Selection of weights?*

      Need some pre-knowledge

      Not reliable for problem involving uncertainties

Multi-Criterion Fitness

- Optimizing single objective
    - *Maximize:* $f_k(\mathbf{X})$

      Subject to:

      $f_i(\mathbf{X}) <= K_i, \quad i <> k$

      $\mathbf{X}$ *in F where F is the solution space*.

Multi-Criterion Fitness

- Weighted sum
  - $F(\mathbf{x}) = w_1f_1(x_1) + w_2f_2(x_2) + \ldots + w_nf_n(x_n)$
  - *Problems?*
    - *Convex and convex Pareto optimal front*

      Sensitive to the shape of the Pareto-optimal front
    - *Selection of weights?*

      Need some pre-knowledge

      Not reliable for problem involving uncertainties

Multi-Criterion Fitness

- Preference based weighted sum (**ISMAUT** ***Imprecisely Specific Multiple Attribute Utility Theory***)
  - $F(\mathbf{x}) = w_1 f_1(x_1) + w_2 f_2(x_2) + \ldots + w_n f_n(x_n)$
  - Preference
    - Given two known individuals **X** and **Y,** if we prefer **X** than **Y**, then

      $F(X) > F(Y),$

      that is

      $w_1(f_1(x_1)-f_1(y_1)) + \ldots + w_n(f_n(x_n)-f_n(y_n)) > 0$

Multi-Criterion Fitness

- All the preferences constitute a linear space $W_n=\{w_1,w_2,\ldots,w_n\}$

  $w_1(f_1(x_1)-f_1(y_1)) +\ldots+w_n(f_n(x_n)-f_n(y_n)) > 0$

  $w_1(f_1(z_1)-f_1(p_1)) +\ldots+w_n(f_n(z_n)-f_n(p_n)) > 0$, *etc*

- For any two new individuals ***Y'*** and **Y''**, how to determine which one is more preferable?

Multi-Criterion Fitness

$$Min : \mu = \sum_{k} w_k [\, f_k(\mathbf{Y}')) - f_k(\mathbf{Y}'')]$$

$$s.t.: \qquad W_n$$

$$Min : \mu' = \sum_{k} w_k [\, f_k(\mathbf{Y}'')) - f_k(\mathbf{Y}')]$$

$$s.t.: \qquad W_n$$

Where W_n is defined in previous slide

Multi-Criterion Fitness

Then,

$$\mu > 0 \Rightarrow \mathbf{Y}' \succ \mathbf{Y}''$$

$$\mu' > 0 \Rightarrow \mathbf{Y}'' \succ \mathbf{Y}'$$

Otherwise,

$$\mathbf{Y}' \sim \mathbf{Y}''$$

Construct the dominant relationship among some indifferent ones according to the preferences.

Perceptrons, logic decomposition or Neural Nets can be used to create cost functions for these kinds of problems. Linking EA and other Machine Learning/AI methods

Other parameters of GA

- Initialization:
  - Population size
  - Random initialization
  - Dedicated greedy algorithm (smart) to initialize
- Reproduction:
  - Generational: as described before (insects)
  - Generational with elitism: fixed number of most fit individuals are copied unmodified into new generation
  - Steady state: two parents are selected to reproduce and two parents are selected to die; two offspring are immediately inserted in the pool (mammals)

Other parameters of GA (cont)

- Stop criterion:
  - Number of new chromosomes
  - Number of new and unique chromosomes
  - Number of generations
- Measure:
  - Best of population
  - Average of population
- Duplicates
  - Accept all duplicates
  - Avoid too many duplicates, because that degenerates the population (inteelt)
  - No duplicates at all

Other issues

- Global versus Optimal
- Parameter Tuning: hand versus automatic
- Parallelism: software, versus hardware, versus evolvable hardware
- Random number generators; quality of software and hardware realizations

EA Applications

EA Applications

- Numerical, Combinatorial Optimization
- System Modeling and Identification
- Planning and Control
- Engineering Design
  (logic and architectural synthesis)
- Data Mining
- Machine Learning
- Artificial Life (Brain Building)

Evaluation of EA algorithms

- Acceptable performance at acceptable costs on a wide range of problems
- Intrinsic parallelism (robustness, fault tolerance)
- Superior to other techniques on complex problems with
  - lots of data, many free parameters
  - complex relationships between parameters
  - many (local) optima

Advantages of EA algorithms

- No presumptions with respect to problem space
- Widely applicable
- Low development & application costs
- Easy to incorporate other methods
- Solutions are interpretable (unlike NN)
- Can be run interactively, accomodate user proposed solutions
- Provide many alternative solutions

Disadvantages of EA algorithms

- No guarantee for optimal solution within finite time
- Weak theoretical basis
- May need parameter tuning
- Often computationally expensive, i.e. slow

Outline of various techniques

- Plain Genetic Algorithms
- Evolutionary Programming
- Evolution Strategies
- Genetic Programming

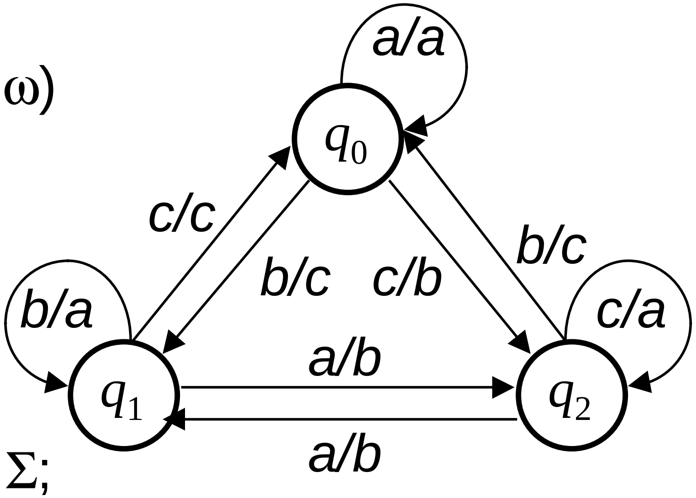
Evolutionary Programming

- Individuals are finite-state automata
- Used to solve prediction tasks
- State-transition table modified by uniform random mutation
- No recombination
- Fitness depends on the number of correct predictions
- **Truncation** selection

Evolutionary Programming: Individuals

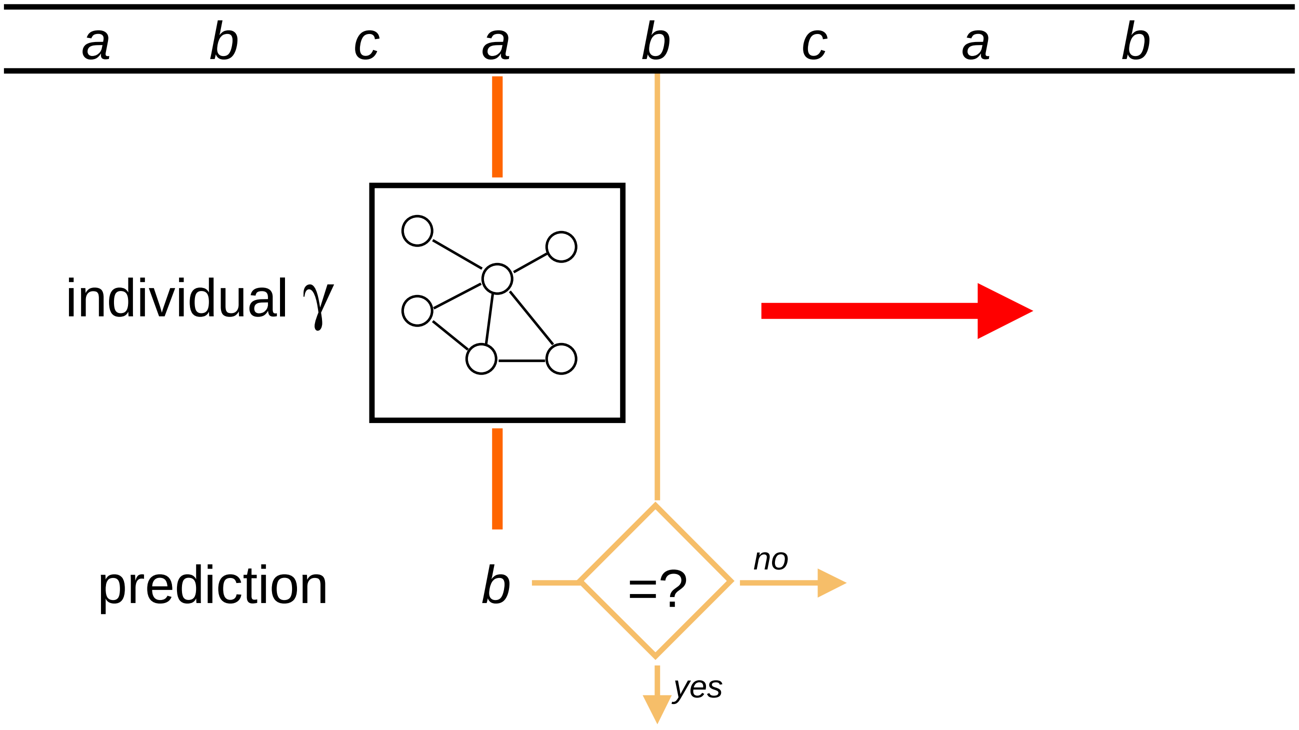
Finite-state automaton: $(Q, q_0, A, \Sigma, \delta, \omega)$

- set of states Q;
- initial state q_0;
- set of accepting states A;
- alphabet of symbols Σ;
- transition function δ: $Q \times \Sigma \to Q$;
- output mapping function ω: $Q \times \Sigma \to \Sigma$;



state / input	q_0	q_1	q_2
a	q_0 *a*	q_2 *b*	q_1 *b*
b	q_1 *c*	q_1 *a*	q_0 *c*
c	q_2 *b*	q_0 *c*	q_2 *a*

Evolutionary Programming: Fitness

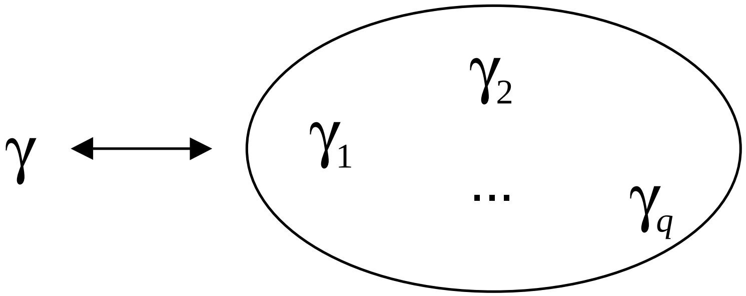


$$f(\gamma) = f(\gamma) + 1$$

Fitness depends on the number of correct predictions

Evolutionary Programming: Selection

Variant of stochastic *q*-tournament selection:



score(γ) = $\#\{\gamma_i \mid f(\gamma) > f(\gamma_i) \}$

Order individuals by decreasing score
Select first half (Truncation selection)

Evolution Strategies

- Individuals are *n*-dimensional vectors of real numbers
- Fitness is the objective function
- Mutation distribution can be part of the genotype

  (standard deviations and covariances evolve with solutions)
- Multi-parent recombination *(more than two parents)*
- Deterministic selection (truncation selection)

GA Examples

Example 1: coding for TSP

Travelling Salesman Problem

- Binary coding
  - Cities are binary coded; chromosome is string of bits
    - Most chromosomes code for illegal tour
    - Several chromosomes code for the same tour
- Path coding
  - Cities are numbered; chromosome is string of integers
    - Most chromosomes code for illegal tour
    - Several chromosomes code for the same tour
- Ordinal
  - Cities are numbered, but code is complex (permutative coding)
  - All possible chromosomes are legal and only one chromosome for each tour
- Several others

Example 2: Function Optimization

- Problem: Find the maximum value of a two-variable function

$$F(x_1,x_2) = 21.5 + x_1\sin(4\pi x_1) + x_2\sin(20\pi x_2)$$

x_1 is in the range [-3.0,12.1] and
x_2 is in the range [4.1,5.8]

There are several ways to solve this problem: Classical/Search/GA . . .

Example 2: (cont)

Representation of a real number

- The first problem is one of representation
  - how do you create a binary representation of a real number?

- SOLUTION: determine a level of precision and divide the range into equal sized intervals
  - if p is the level of precision and the interval is L ***then*** the number of bits required is the number of bits in the binary representation of L x 10^p
    - for x_1 the interval is -3 to 12.1 so L = 15.1

      1 decimal place of precision requires 8 bits (15.1 x 10^1)

      2 decimal places of precision require 11 bits (15.1 x 10^2)

Example 2: (cont)

Conversion

- To convert a binary number b to a real number x use:

$$x = d + i\left(\frac{L}{R}\right)$$

Where

i is the integer value of b
d is the lower value of the range
L is the length of the range
R is the range of binary integers

For example, x_1 with 4 digits of precision (18 bits), is found to be (in binary): 010101110000110010

i is 89136

L is 15.1

R is 2^{18}-1

d is -3

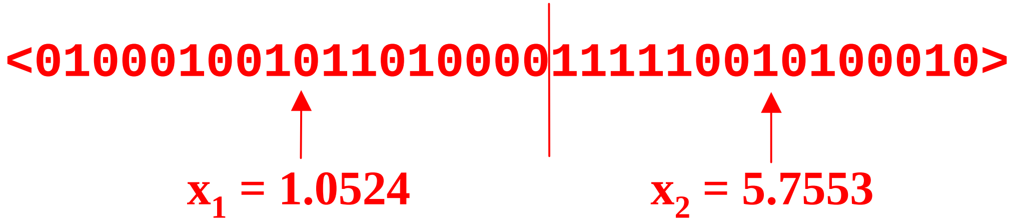
$$x = -3.0 + 89136\left(\frac{15.1}{2^{18}-1}\right) = 2.1344$$

Example 2: (cont)

Complete Representation

- x_2 has a range of 1.7, so for 4 decimal places of precision it requires 15 bits
- Hence, a single chromosome for both variables requires 18 + 15 = 33 bits
  - the first 18 bits represent x_1
  - the last 15 bits represent x_2

For example, this chromosome represents two real numbers



<010001001011010000111110010100010>

$x_1 = 1.0524$

$x_2 = 5.7553$

Example 2: (cont)

Start a GA

- Determine a population size, n
- Using a random number generator, construct n binary strings
- Determine the fitness of each random solution

Example 2: (cont)

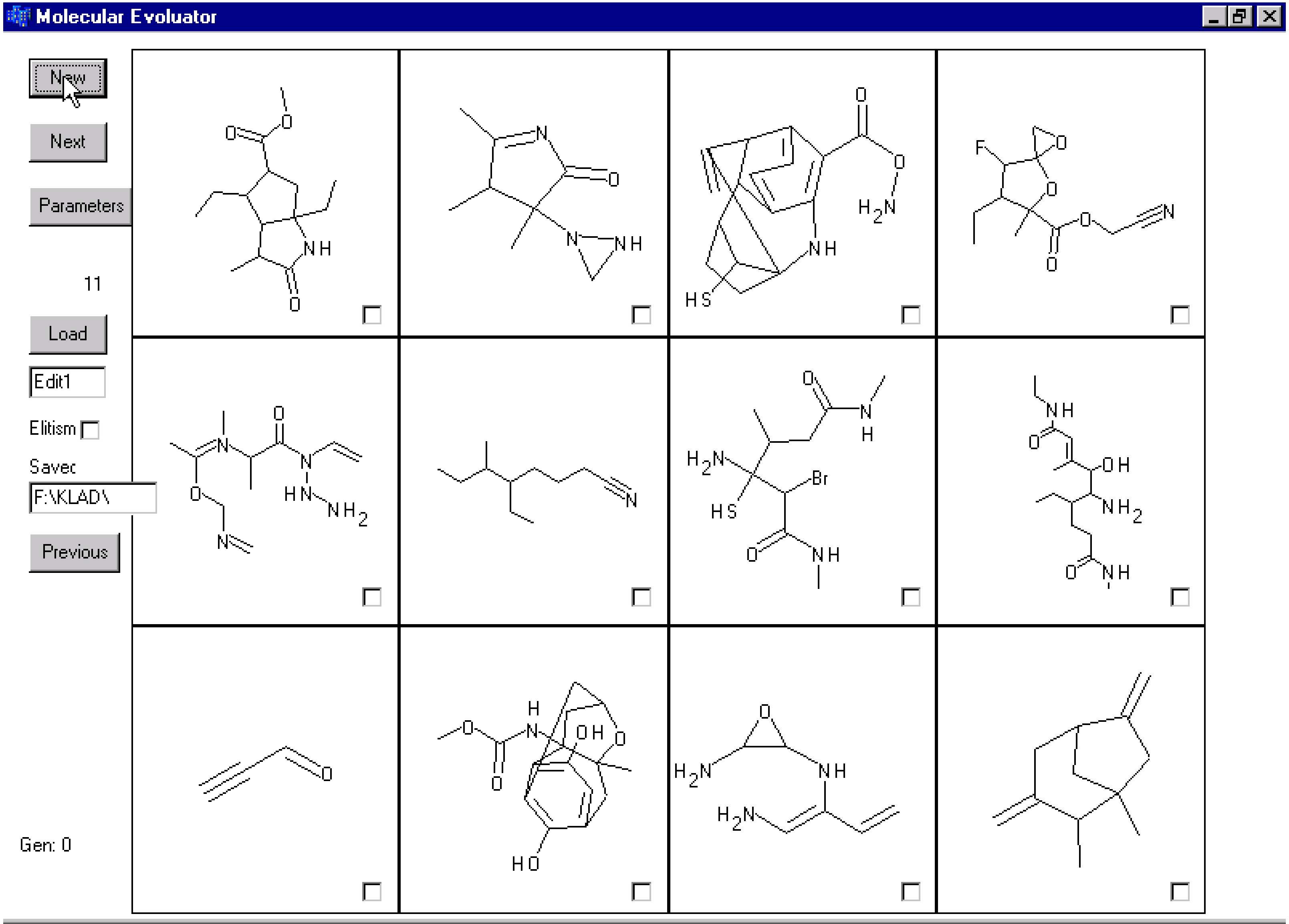
Example Run

Population size is 20, fitness is the function value

Best Element (so far)

Point	Binary Representation	Value
v_1	100110100000001111111010011011111	26.0196
v_2	111000100100110111001010100011010	7.5800
v_3	000010000011001000001010111011101	19.5263
v_4	100011000101101001111000001110010	17.4067
v_5	000111011001010011010111111000101	25.3411
v_6	000101000010010101001010111111011	18.1004
v_7	001000100000110101111011011111011	16.0208
v_8	100001100001110100010110101100111	17.9597
v_9	010000000101100010110000000101100	16.1277
v_{10}	000001111000110000011010000111011	21.2784
v_{11}	011001111110110101100001101111000	23.4106
v_{12}	110100010111101101000101010000000	15.0116
v_{13}	111011111010001000110000001000110	27.3167
v_{14}	010010011000001010100111100101001	19.8762
v_{15}	111011111011100001000111110111110	30.0602
v_{16}	110011110000011111100001101001011	23.8672
v_{17}	011010111111001111010001101111101	13.6961
v_{18}	011101000000001110100111110101101	15.4142
v_{19}	000101010011111111110000110001100	20.0959
v_{20}	101110010110011110011000101111110	13.6669

Example: Combinatorial Chemistry



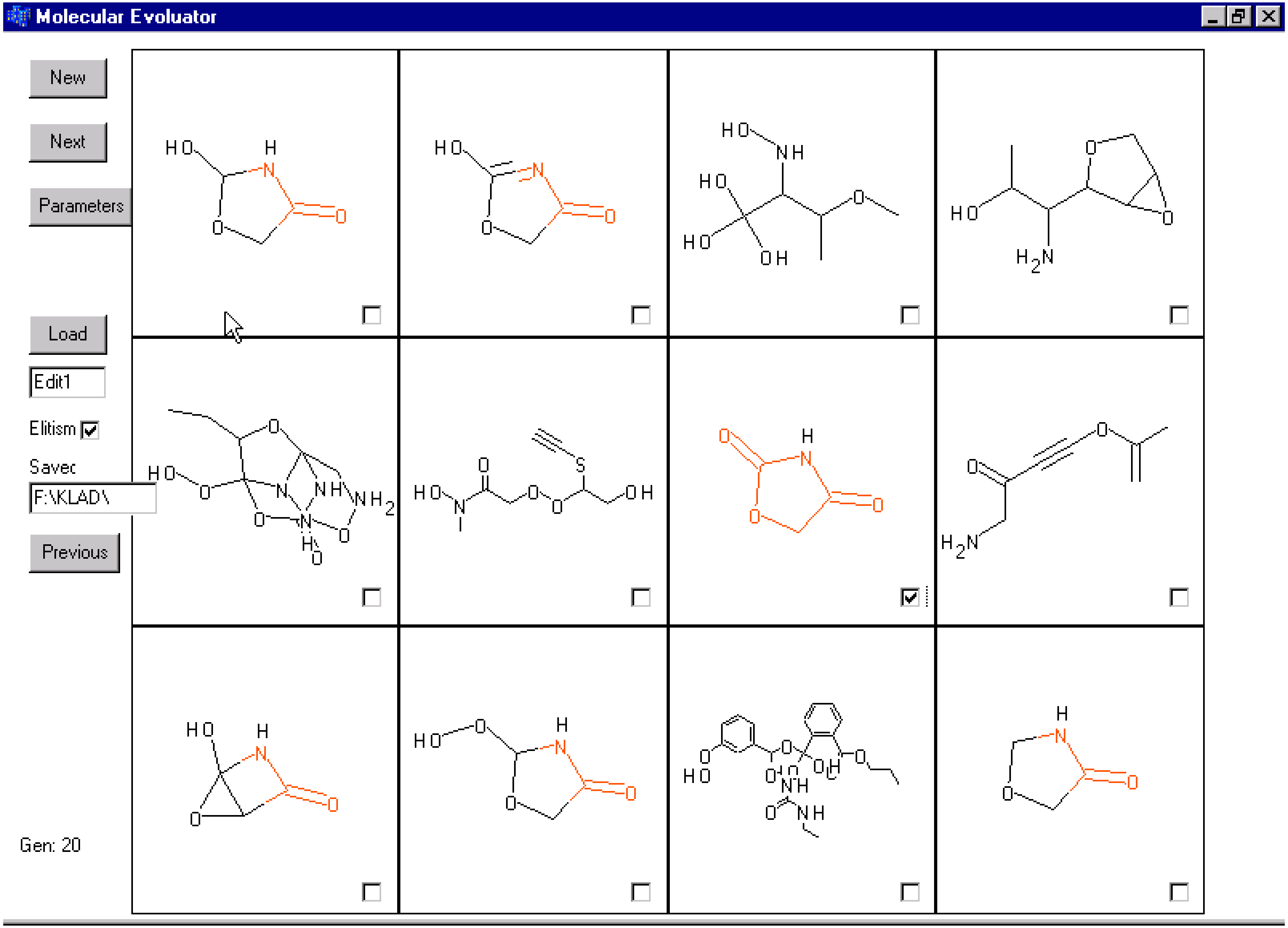
11

Edit1

Elitism ☐

Savec F:\KLAD\

Gen: 0



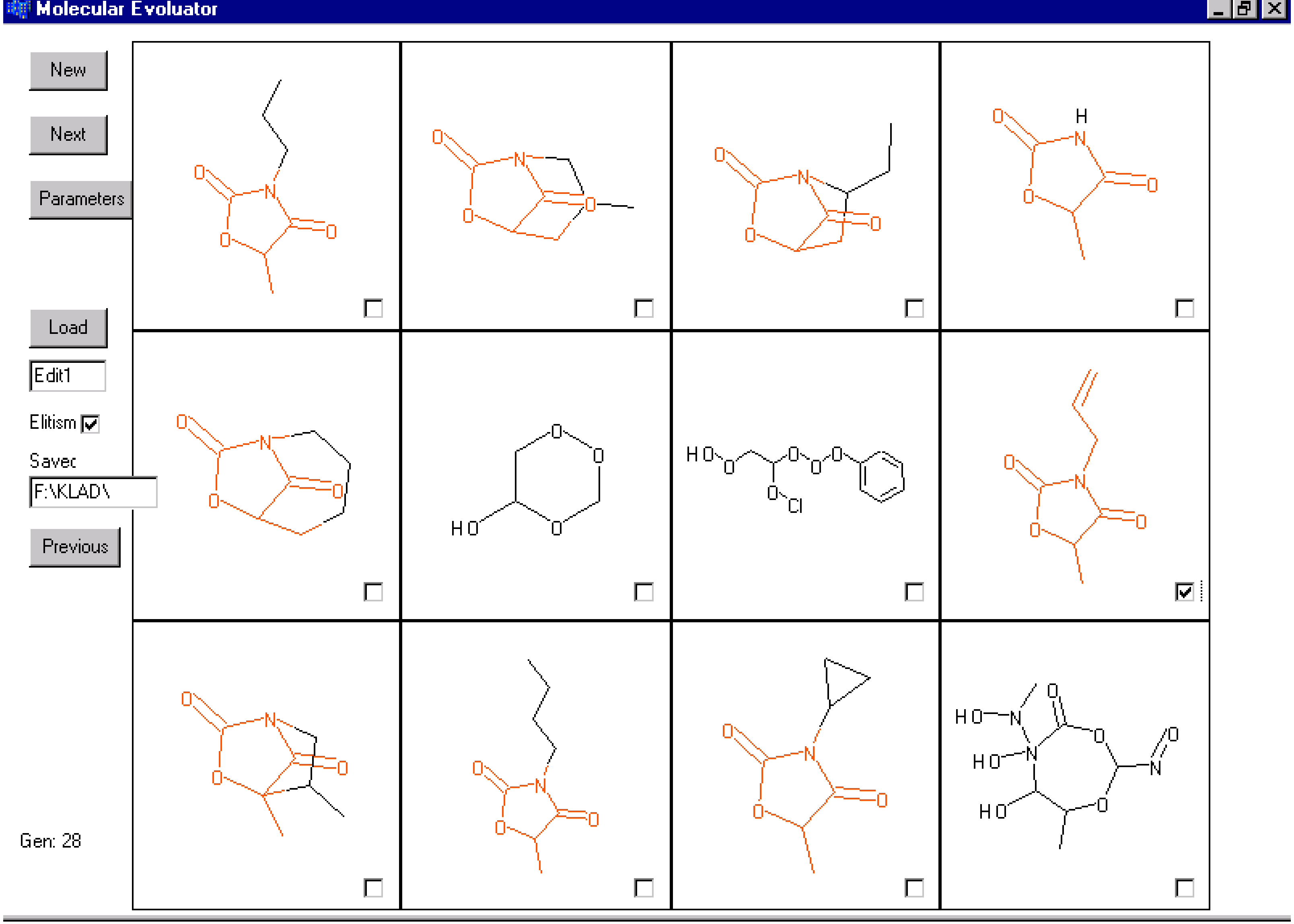
Edit1

Elitism ☑

Savec F:\KLAD\

Gen: 20

Constraints, rules and partial cost functions



Edit1

Elitism ☑

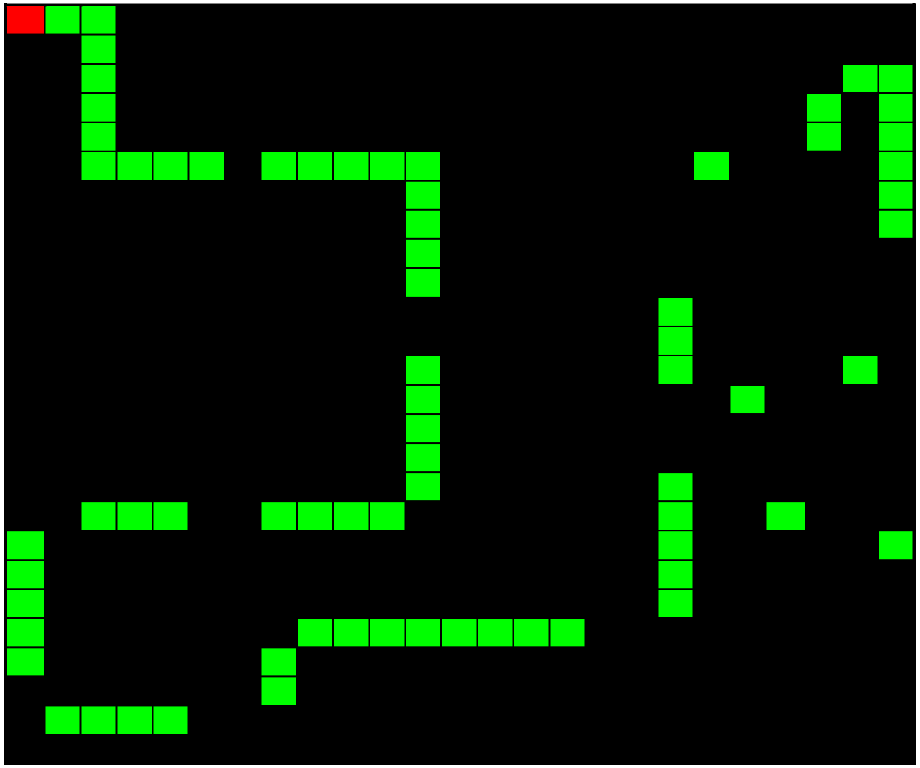
Savec F:\KLAD\

Gen: 28

User part of the loop, subjective cost functions, GA-based Computer Aided Design, Computer-Aided Art,etc.

Example GP Problem

- Santa-Fe Trail Problem



Fitness: How much food collected

Individual program on the previous slide generated on 7th generation solved the problem completely

Example GP Problem (cont)

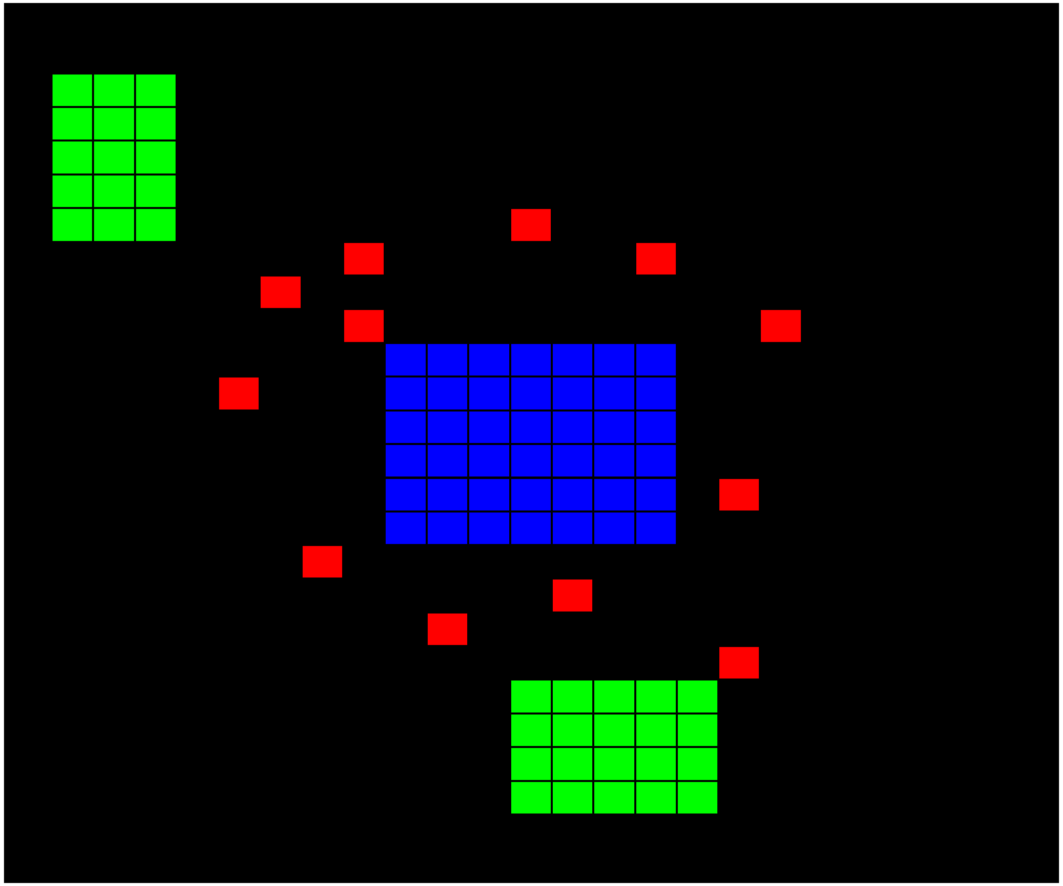
Examples: Artificial Ant Problem. Given a set environment with a trail of food, goal is to get as most of the food as possible in a given timeframe

Functions: IF-FOOD, PROGN
Terminals: ADVANCE, TURN LEFT, TURN RIGHT

After 51 generations with population of 1000, following individual emerged that got all the food:

```
(If-Food (Advance)
        (Progn (turn-right)
                (If-Food (Advance) (Turn-Left))
                (Progn (Turn-left)
                        (If-Food (Advance) (Turn-Right))
                                (Advance))))
```

Variant of Ants - Emergent Collective Behavior



Fitness: Food collected by all ants and returned to nest in given time period

Programs evolved to demonstrate collective intelligent behavior, lay pheromone trails

Other known Examples, some of them you will find on my WebPage

- Evolutionary Art
- Nozzle
- Best Sorter
- ESOP synthesis
- Decision Trees and Diagrams, all kinds of circuits and architectures
- Reversible logic (KAIST/PSU project)
- Quantum Logic
- Brain Building
- DNA Computing
- Nano-Technologies

Optimization Techniques

- Mathematical Programming
- Network Analysis
- Branch & Bound
- Genetic Algorithm
- Simulated Annealing
- Tabu Search

Belong to the *machine learning* part of AI

But Genetic Algorithm is also a representative of Evolutionary Computing, which is a general problem solving paradigm taken from Nature

Sources

Martijn Schut, schut@cs.vu.nl,

Jaap Hofstede, Beasly, Bull, Martin

Andrea G. B. Tettamanzi

Joost N. Kok and Richard Spillman

KMT Software, Inc. -- http://www.kmt.com

Dan Kiely
Ran Shoham
Brent Heigold William H. Hsu, Department of Computing and Information Sciences, KSU

- **A.E. Eiben, Free University Amsterdam**
- **EvoNet Training Committee and its "Flying Circus"**
- **Peter Nordin, Physical Resource Theory, Complex Systems**
- **Mehrdad Dianati**
- **Insop Song**
- **Mark Treiber**
- **Nathalie Japkowicz**