<u>Evolutionary algorithms</u>

- **Simple genetic algorithms**
- **Evolutionary Strategies**
- **Genetic Programming**

Partially based on slides by Thomas Bäck

Heuristic Search

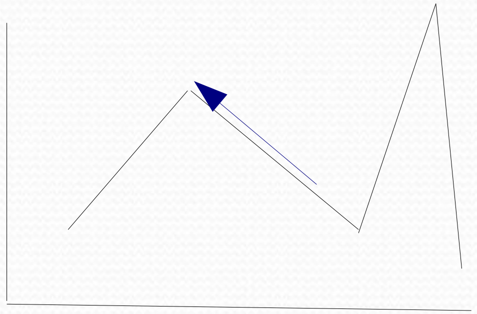
- SAT solvers, CP solvers, ILP solvers:
  - find exact solutions to discrete constraint optimization problems
  - can be time consuming
- Heuristic solvers:
  - employ “heuristics”: guidelines for finding good solutions quickly
  - don't find exact solutions
  - can be much faster
  - can deal with problems that are numerical and not in a “nice” form (eg., linear)

Examples in Fuzzy Logic

- When learning a fuzzy classifier from training data we need to find:
  - Parameters of membership functions
  - Attributes to put in rules

- When finding the parameters that maximize the output of a fuzzy system, we need to find numerical values

Hill-Climbing



- Hill-climbing is arguably the simplest heuristic algorithm

1. S = arbitrary candidate solution
2. S' = solutions in the neighborhood of S
3. **if** best solution in S' is not better than S **then** stop
4. let S be the best solution in S'
5. go to 2.

Neighborhood Search

- Important choice in hill-climbing: which neighborhoods to consider
  - Add a small value to each coordinate? Substruct a small value from each coordinate?

$$(x_1, x_2, \ldots, x_n) \rightarrow \begin{cases} (x_1 + \epsilon, x_2, \ldots, x_n) \\ (x_1 - \epsilon, x_2, \ldots, x_n) \\ (x_1, x_2 + \epsilon, \ldots, x_n) \\ (x_1, x_2 - \epsilon, \ldots, x_n) \\ \vdots \end{cases}$$

Large Neighborhood Search

- Iteratively select a random subset of variables of limited size, find an optimal assignment for these variables, assuming the others are fixed
  - Requires the availability of an algorithm to solve the intermediate problems optimally (linear programming, CP, ..)

$$(x_1, x_2, \ldots, x_n)$$
$$\downarrow$$
$$(?, ?, x_3, \ldots, x_n)$$
$$\downarrow$$
$$(x'_1, x'_2, x_3, \ldots, x_n)$$
$$\downarrow$$
$$(x'_1, x'_2, ?, x_4, \ldots, x_n)$$
$$\downarrow$$
$$(x'_1, x'_2, x'_3, x_4, \ldots, x_n)$$

Other Well-known Heuristic Search Strategies

- Simulated annealing
- Tabu search
- Evolutionary algorithms
  - genetic algorithms
  - genetic programming
  - evolutionary strategies
- Artificial ants
- Particle swarms

Advantages of GAs

- Evolution and natural selection has proven to be a robust method

- A “black box” approach that can easily be applied to many optimization problems

- GAs can be easily parallelized and run on multiple machines

Some definitions

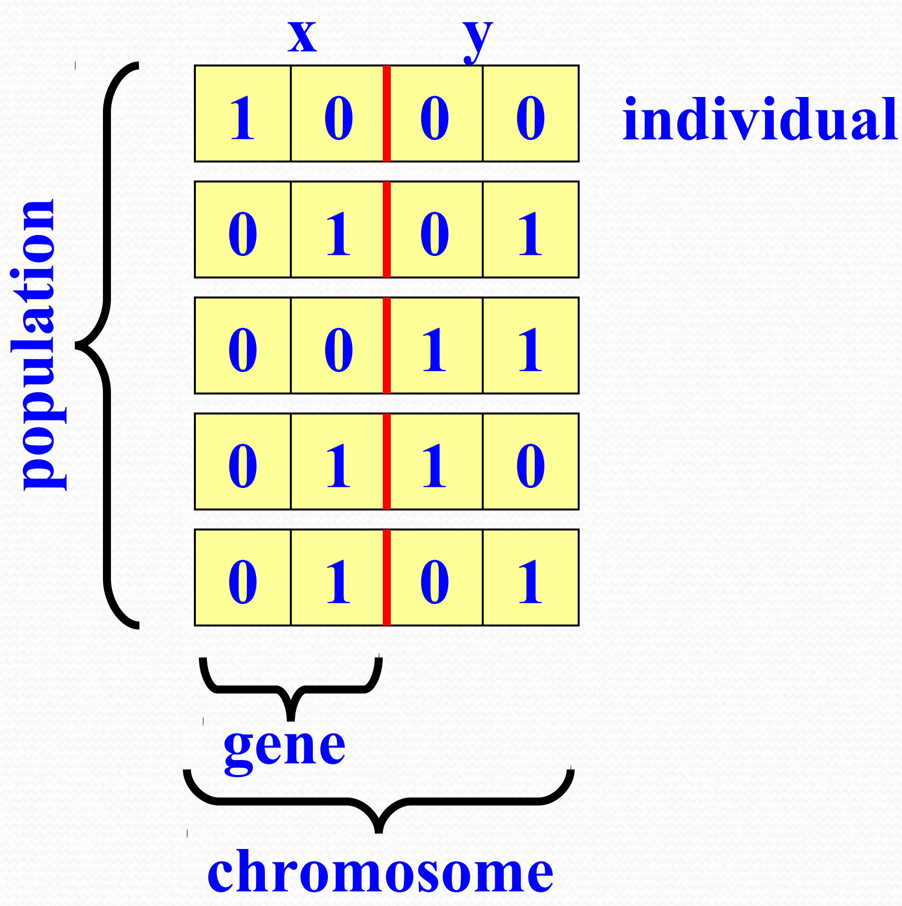
- **Population**: a collection of solutions for the studied (optimization) problem
- **Individual**: a single solution in a GA
- **Chromosome (genotype)**: representation for a single solution
- **Gene**: part of a chromosome, usually representing a variable as part of the solution

Some definitions

- **Encoding**: conversion of a solution to its equivalent representation (chromosome)
- **Decoding**: conversion of a chromosome (**genotype**) to its equivalent solution (phenotype)
- **Fitness**: scalar value denoting the suitability of a solution

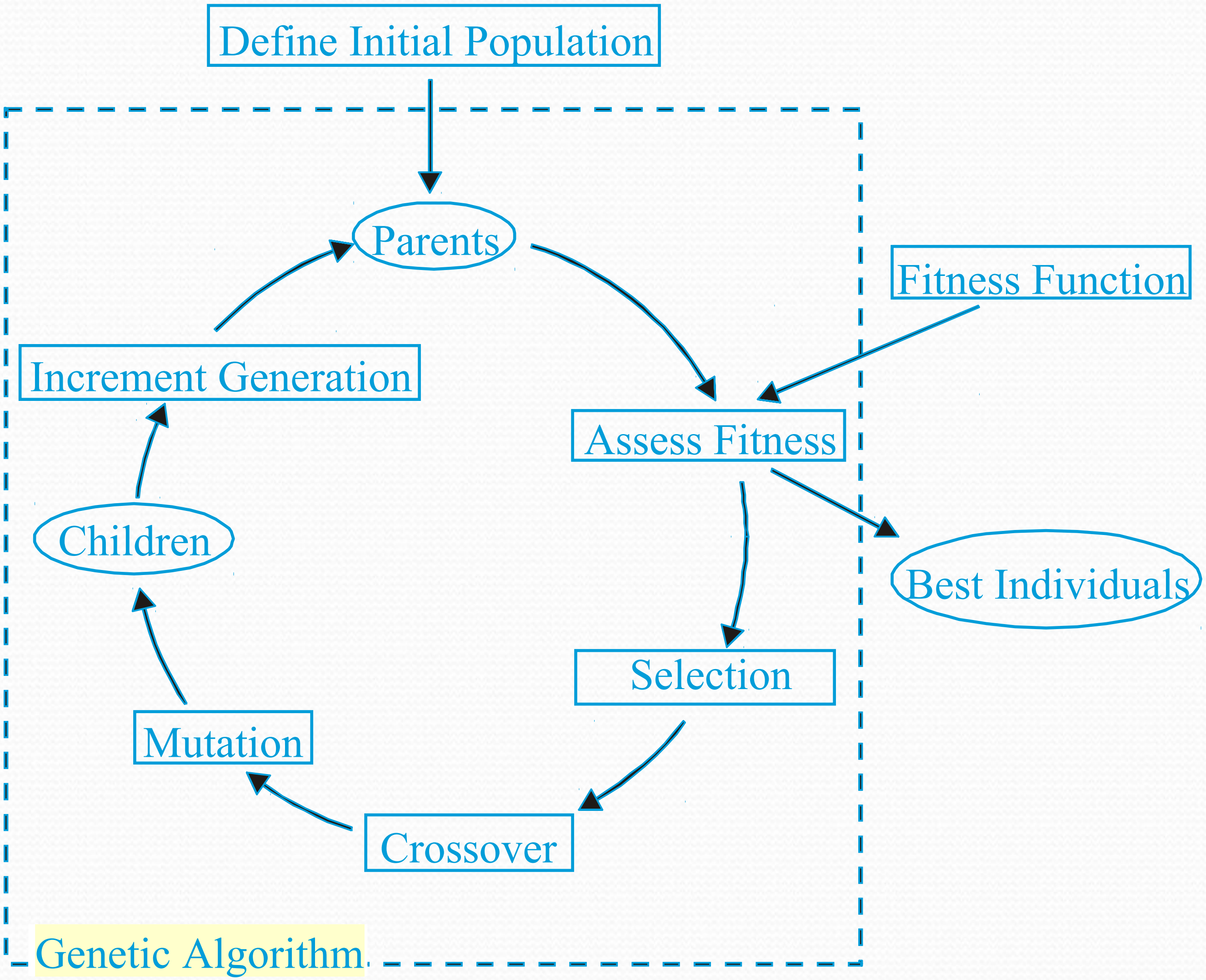
GA terminology

Generation t



solution	fitness
(2,0)	4
(1,1)	2
(0,3)	3
(1,2)	3
(1,1)	2

Genetic algorithm



Pseudo code

- Initialize population P:
  - E.g. generate random p solutions
- Evaluate solutions in P:
  - determine for all $h \in P$, Fitness(h)
- **While** terminate is FALSE
  - Generate new generation P using genetic operators
  - Evaluate solutions in P
- **Return** solution $h \in P$ with the highest Fitness

Termination criteria

- Number of generations
  (restart GA if best solution is not satisfactory)
- Fitness of best individual
- Average fitness of population
- Difference of best fitness (across generations)
- Difference of average fitness (across generations)

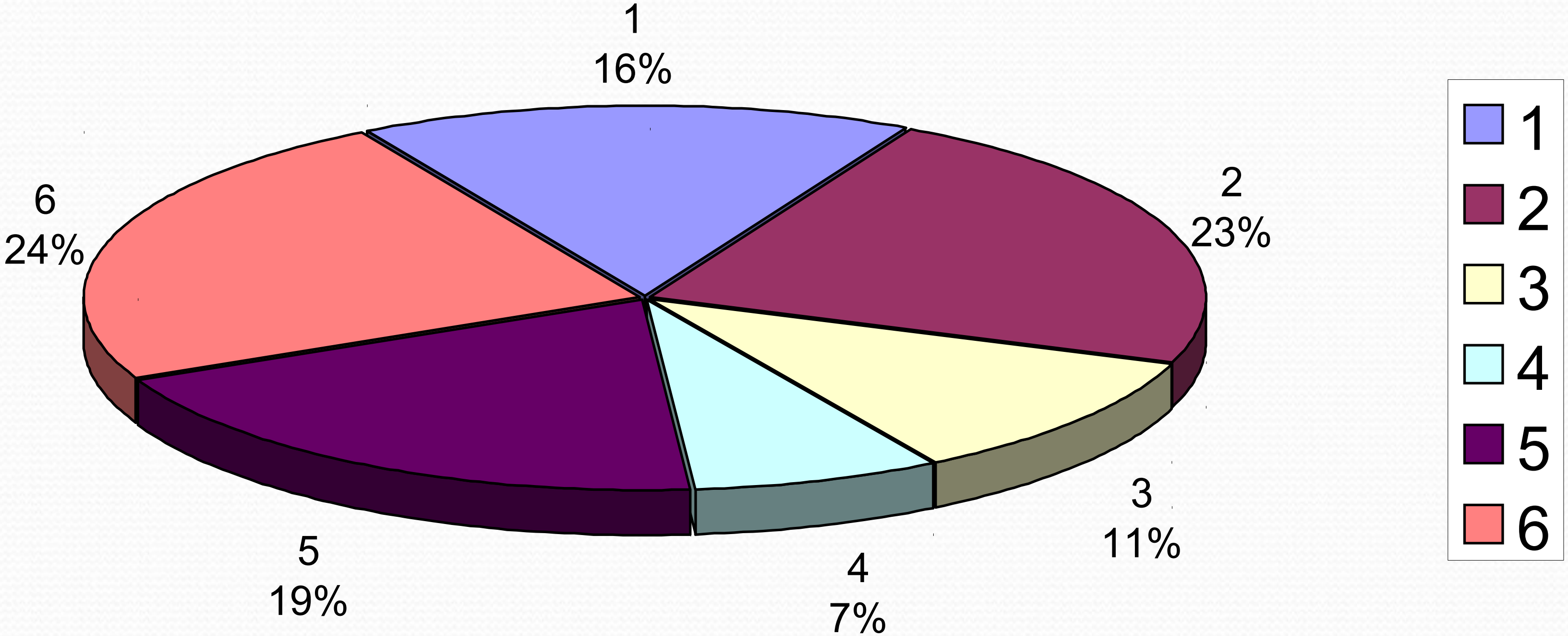
Reproduction

Three steps:

- Selection
- Crossover
- Mutation

In GAs, the population size is often kept constant. User is free to choose which methods to use for all three steps.

Roulette-wheel selection



Roulette-wheel selection

individuals fitness

$$\begin{array}{l} p = 0.16 \\ p = 0.23 \\ p = 0.11 \\ p = 0.07 \\ p = 0.19 \\ p = 0.24 \end{array} \begin{bmatrix} 01100 \\ 10001 \\ 11010 \\ 00111 \\ 11000 \\ 10110 \end{bmatrix} \begin{bmatrix} 34 \\ 48 \\ 23 \\ 15 \\ 41 \\ 50 \end{bmatrix} \xrightarrow{\text{selection}} \begin{bmatrix} 01100 \\ 10001 \\ 10001 \\ 11000 \\ 10110 \\ 10110 \end{bmatrix}$$

Sum = 211

Cumulative probability: 0.16, 0.39, 0.50, 0.57, 0.76, 1.00

Tournament selection

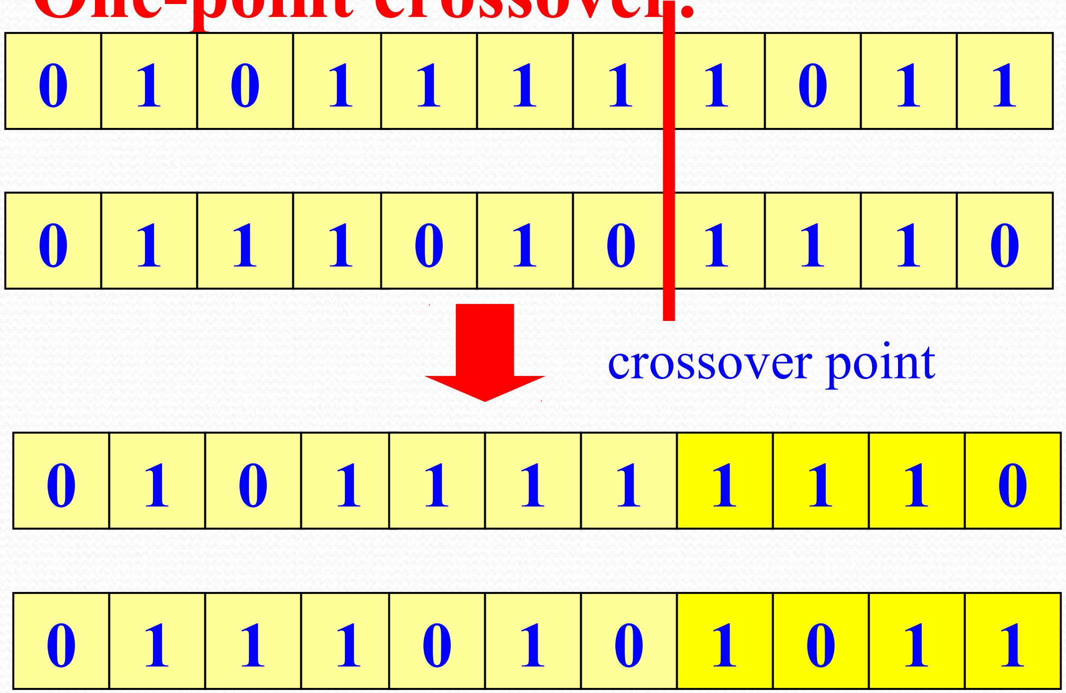
- Select pairs randomly
- Fitter individual wins
  - deterministic
  - probabilistic
    - constant probability of winning
    - probability of winning depends on fitness

It is also possible to combine tournament selection with roulette-wheel

Crossover

- Exchange parts of chromosome with a crossover probability (p_c is usually about 0.8)
- Select crossover points randomly

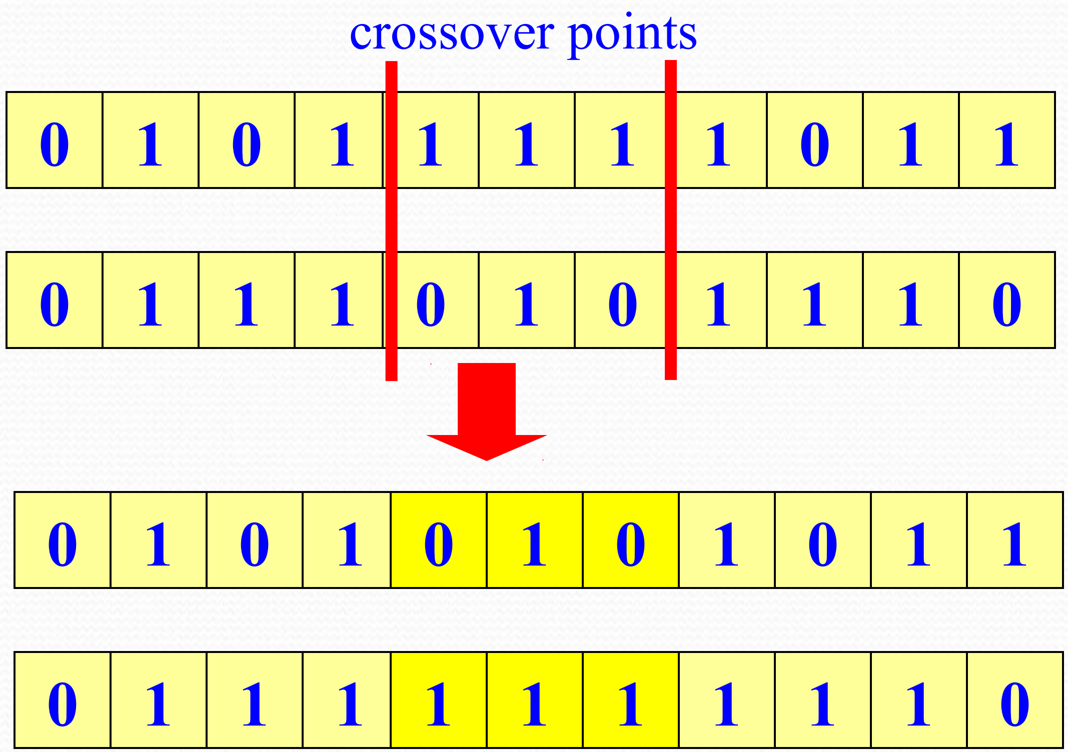
One-point crossover:



N-point crossover

- Select N points for exchanging parts
- Exchange multiple parts

Two-point crossover:



Uniform crossover

- Exchange bits using a randomly generated mask

0	1	0	1	0	0	1	0	0	1	1	mask

0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	0	1	0	1	1	1	0

0	1	0	1	1	1	0	1	0	1	0
0	1	1	1	0	1	1	1	1	1	1

Mutation

- Crossover is used to search the solution space
- Mutation is needed to escape from local optima
- Introduces genetic diversity
- Mutation is rare (p_m is about 0.005)

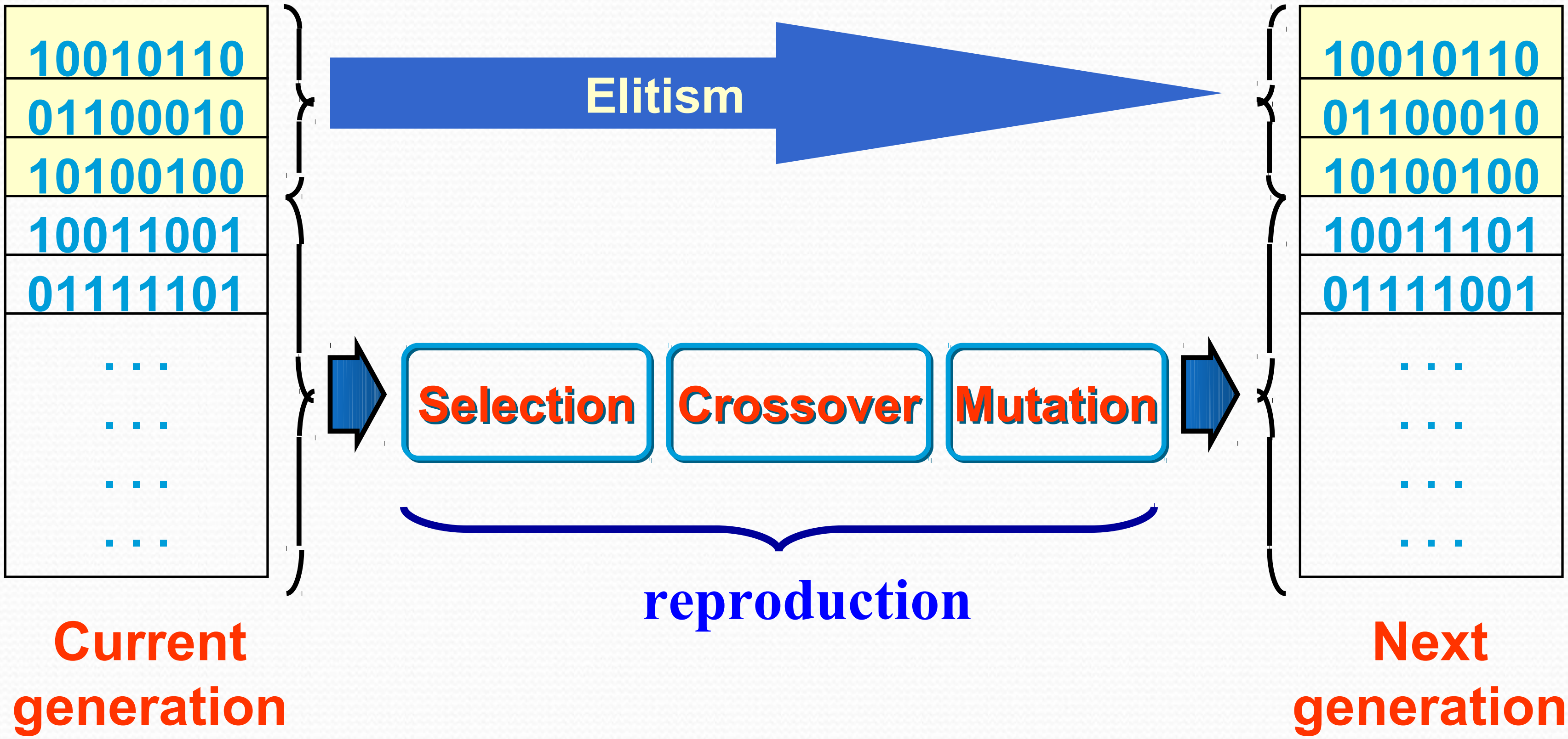
Uniform mutation:

0	1	0	1	1	1	1	1	1	1	0

0	1	0	1	1	1	0	1	1	1	0

mutated bit

GA iteration



Encoding and decoding

- Common coding methods
  - “standard” binary integer coding
  - Gray coding (binary)
  - real valued coding (*evolutionary strategies*)
  - tree structures (*genetic programming*)

Gray Coding

- Aim: binary coding of integers such that integers x and y for which $|x-y|=1$ only differ in one bit

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

Gray Coding

- Codes for n=1: (i.e., integers 0, 1)

  0 1
- Codes for n=2: (i.e., integers 0, 1, 2, 3)

  Reflected entries for n=0:

  1 0

  Prefix old entries with 0:

  00 01

  Prefix reflected entries with 1:

  11 10

  Codes hence:

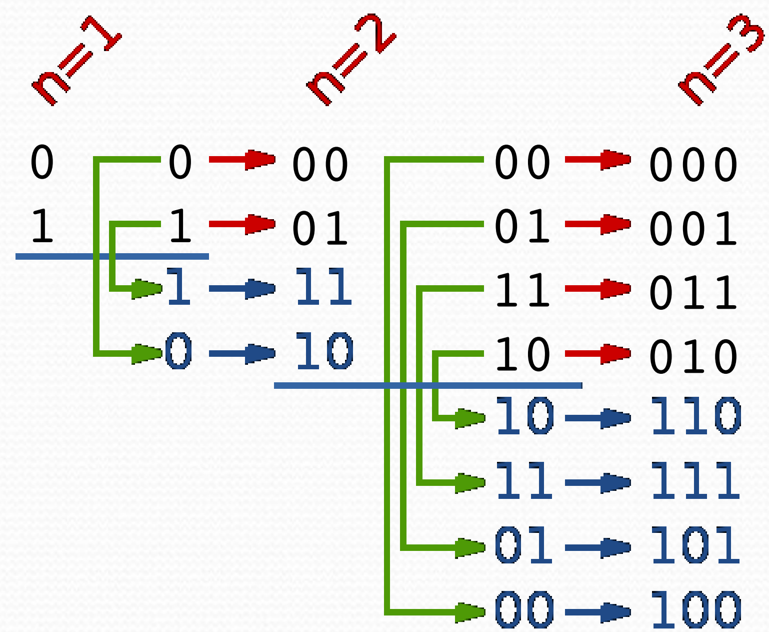
  00 01 11 10
- Codes for n=3: (i.e., integers 0, 1, 2, …, 7)

  Reflected entries for n=2:

  10 11 01 00

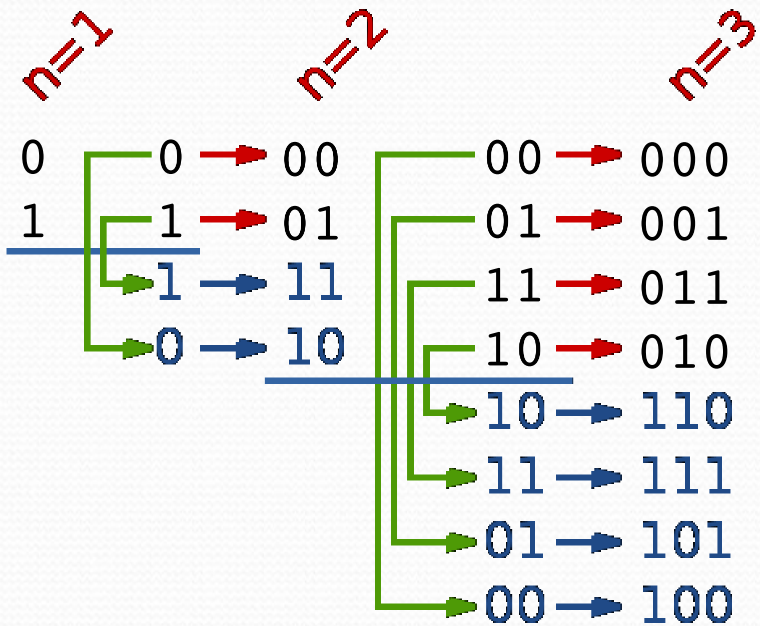
  Codes hence:

  000 001 011 010 110 111 101 100



Gray Coding

- Given a "normal" bit representation, how to calculate the Gray code?



000
001
010
011
100
101
110
111

bitstring → Gray
10100 → 11110
10101 → 11111
10110 → 11101
11001 → 10101

A bit flips in the Gray code iff the bit before it has value 1 in the original code.

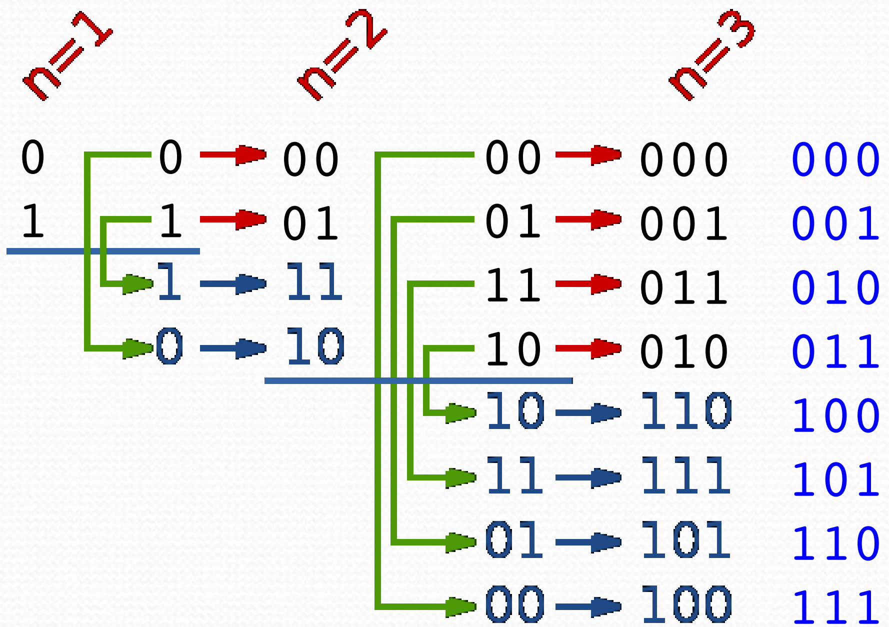
Gray Coding

- Source code in Python for calculating Gray code:

```
def binaryToGray(num):
        return (num >> 1) ^ num
```

Gray Coding

- Given a Gray code, how to calculate a “normal” bit representation?



bitstring → Gray
10100 → 11110
10101 → 11111
10110 → 11101
11001 → 10101

A bit flips in the “normal” code (as compared to the Gray code) iff the bit before it has value 1 in the “normal” code.

Gray Coding

- Gray coding does not avoid that integers far away from each other can have similar codes
  00000=0
  10000=31
  → Mutation can still change numbers a lot

- Gray coding only ensures that there always is a one-bit mutation to transform integer x into integer $x+1$ or $x-1$.

Constraints

- Examples:
  - "A string of numbers should represent a permutation" (1,2,3) is valid; (1,1,3) is not
  - "The sum of numbers should not be lower than a threshold"
- Possibility 1: fitness function modification
  - setting fitness of unfeasible solutions to zero (search may be very inefficient due to unfeasible solutions)
  - penalty function (negative terms for violated constraints)
  - barrier function (already penalty if "close to" violation)

Constraints

- Possibility 2 (preferred method): special encoding
  - GA searches always through allowed solutions
  - smaller search space
  - ad hoc method, may be difficult to find
- Example: permutations (see AI course)