Genetic Algorithms Various Operators

Sabah Sayed

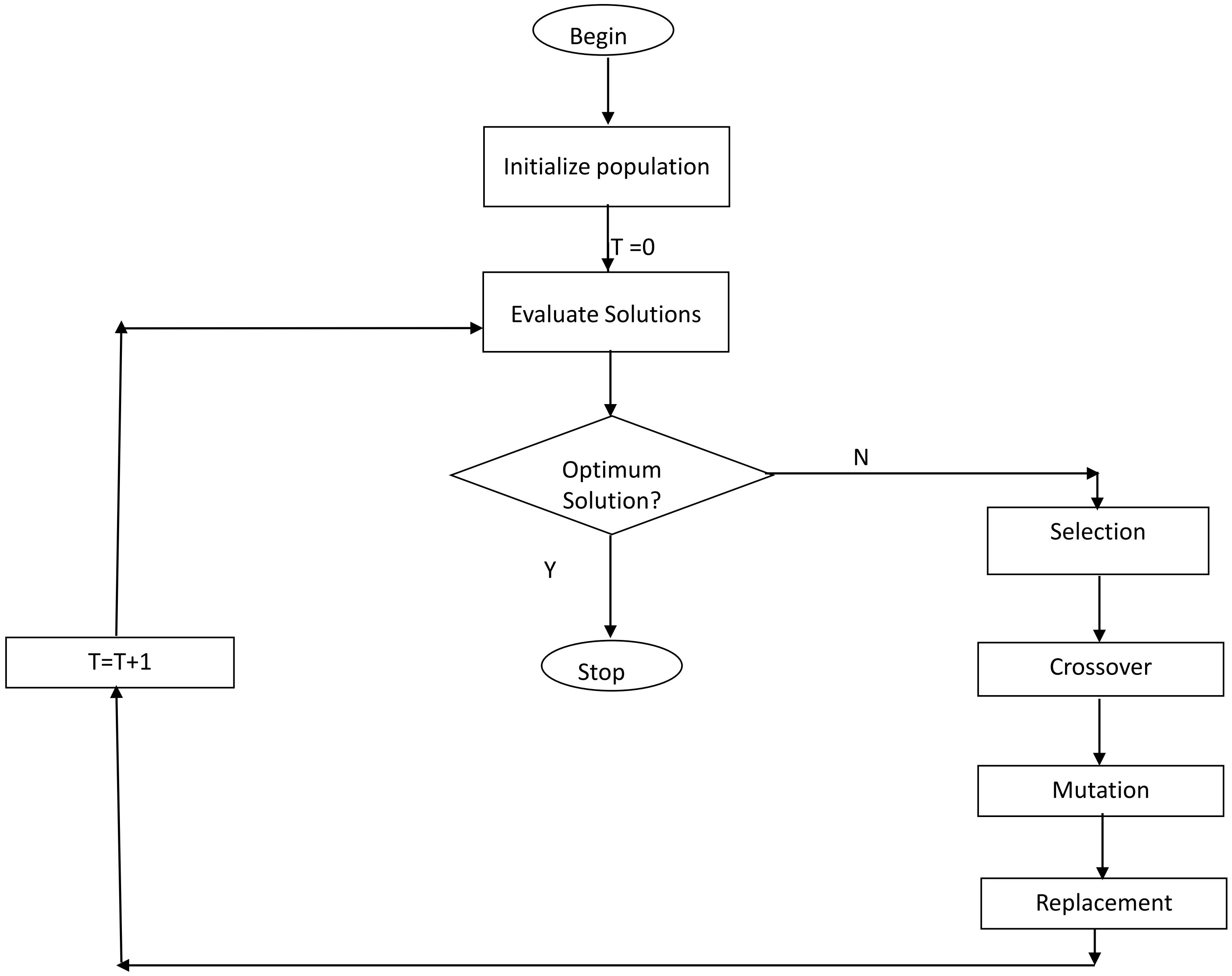
Department of Computer Science

Faculty of Computers and Artificial Intelligence

Cairo University

Egypt

Remember: Mechanism Of GAs



Various Strategies for the Genetic Operators

- Different GAs use different …….. strategies.
  - Representation (encoding/decoding)
  - Crossover
  - Mutation
  - Selection
  - Replacement

Various Representations

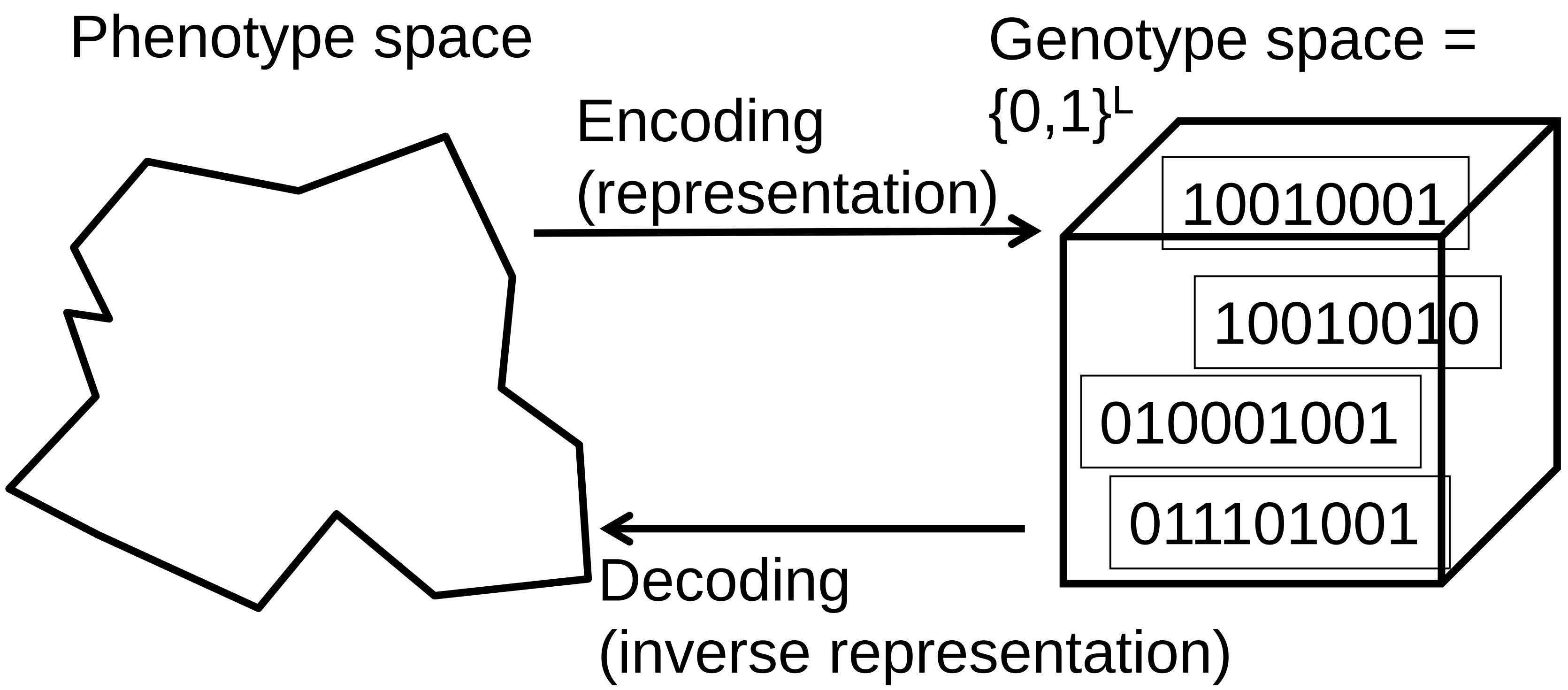
String Array?

Floating Point?

Character Array?

Integer?

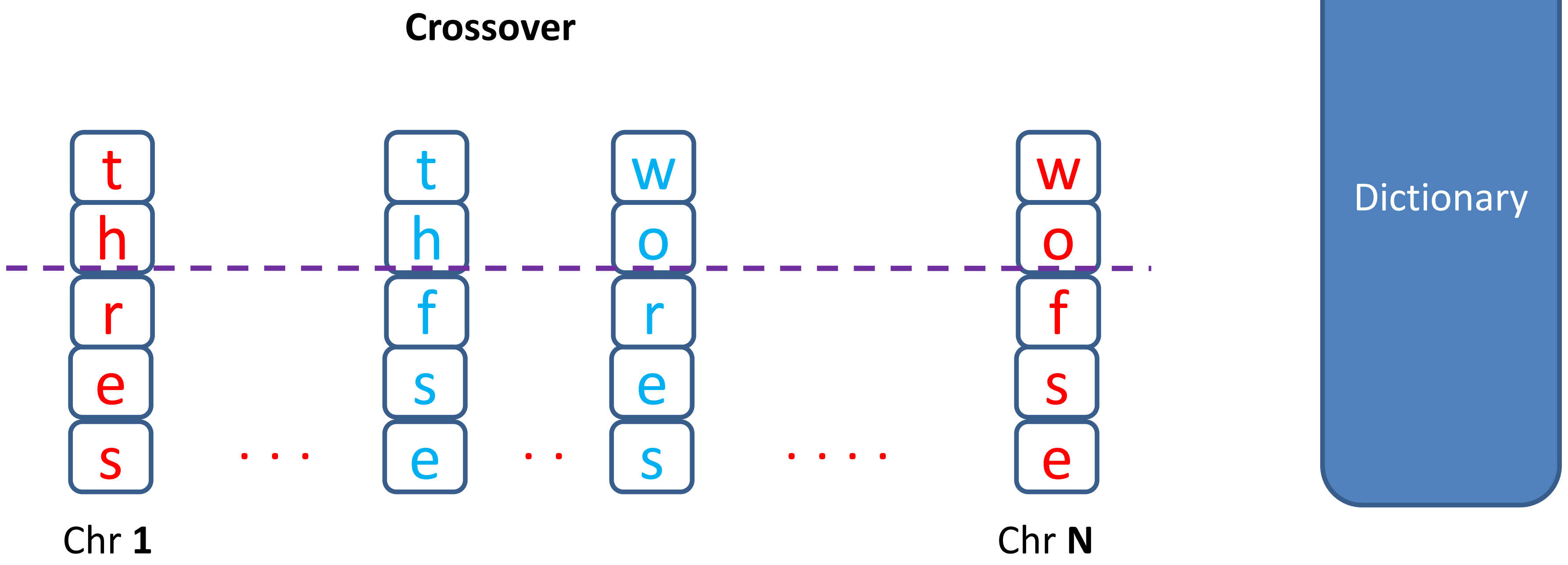
Permutation?



Phenotype space
Encoding (representation)
Genotype space = $\{0,1\}^L$
10010001
10010010
010001001
011101001
Decoding (inverse representation)

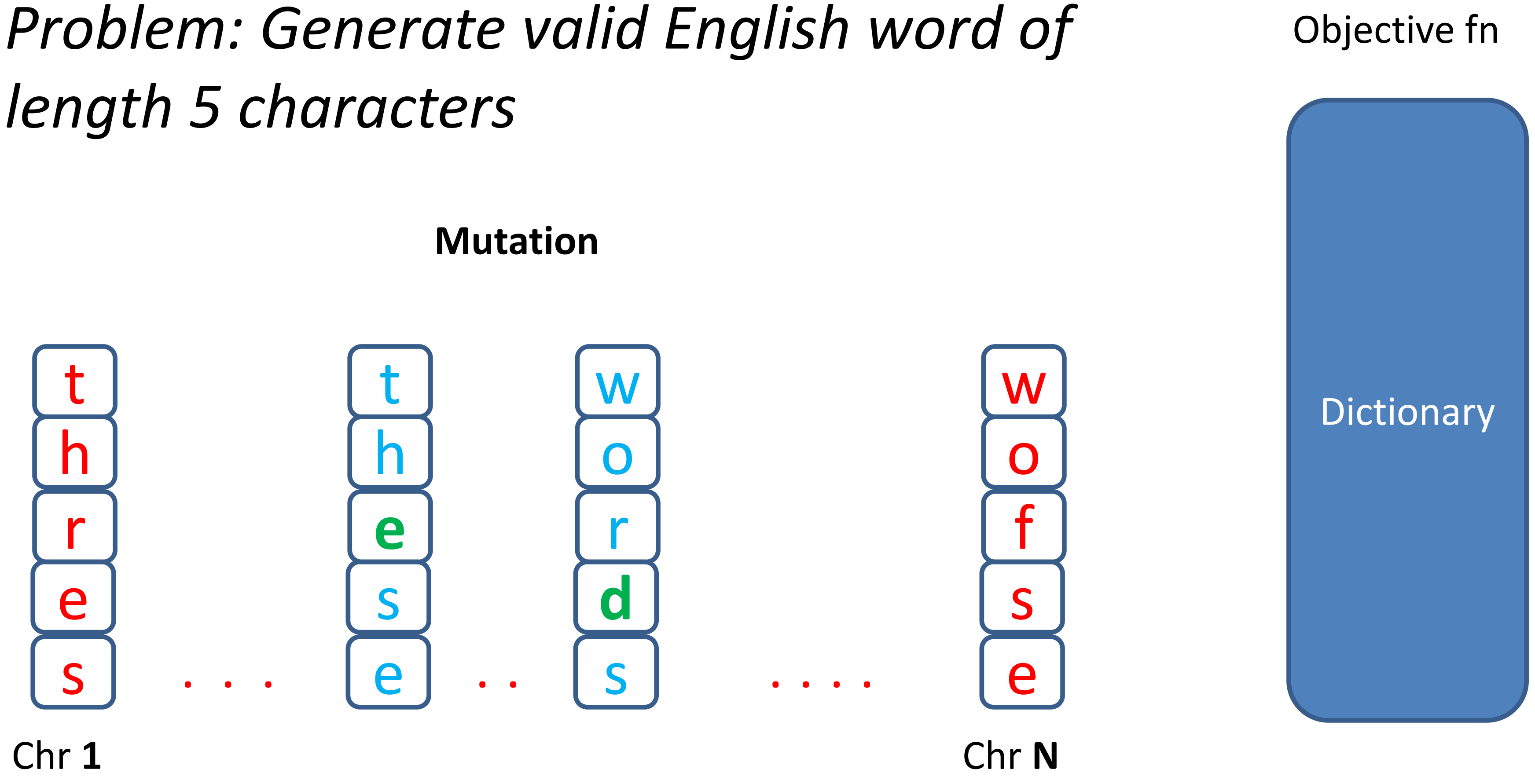
English Word Generation Example

Problem: Generate valid English word of length 5 characters



English Word Generation Example

Problem: Generate valid English word of length 5 characters



Integer Representation

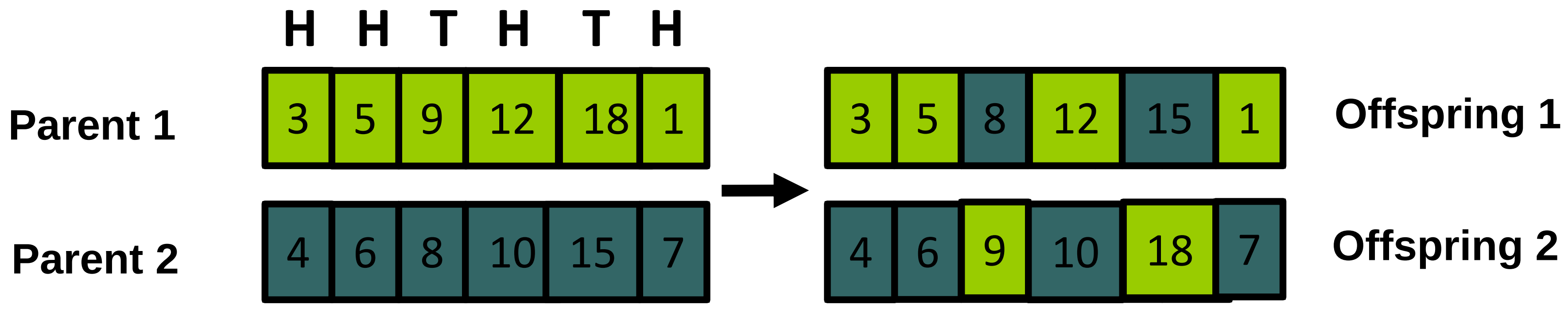
- Some problems naturally have integer variables, e.g. image processing parameters
- Others take *categorical* values from a fixed set e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make
    - "**creep**" i.e. more likely to move to similar value, For ordinal problems it is hard to know correct range for creep
    - **Random choice** (categorical variables)

Uniform Crossover for Integers

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child

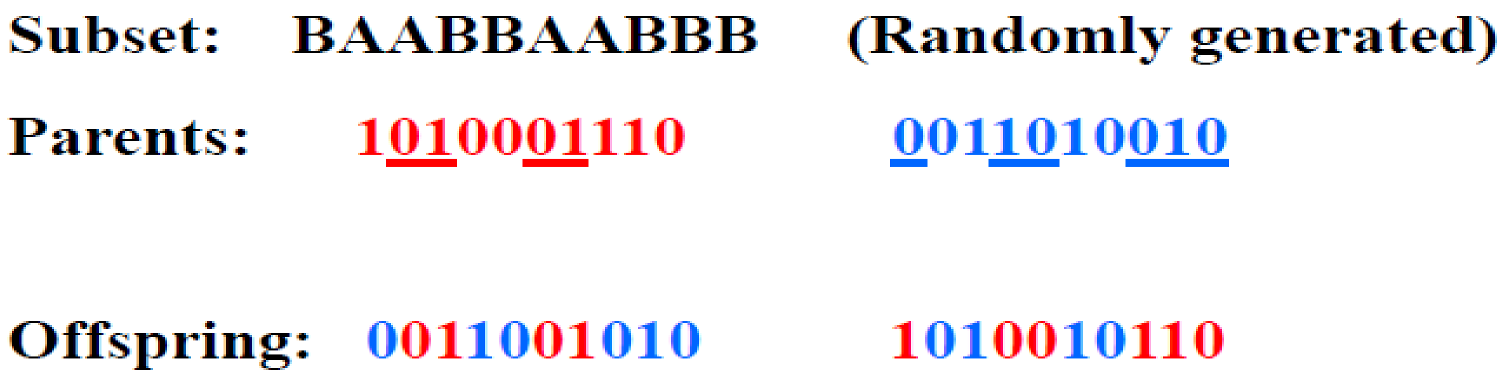
- Example:

Suppose H for Parent1 and T for Parent 2



Uniform Crossover for Integers

- Inheritance is independent of position
- Applicable for binary representation
- How to implement it (programming)
- N-point crossover ???



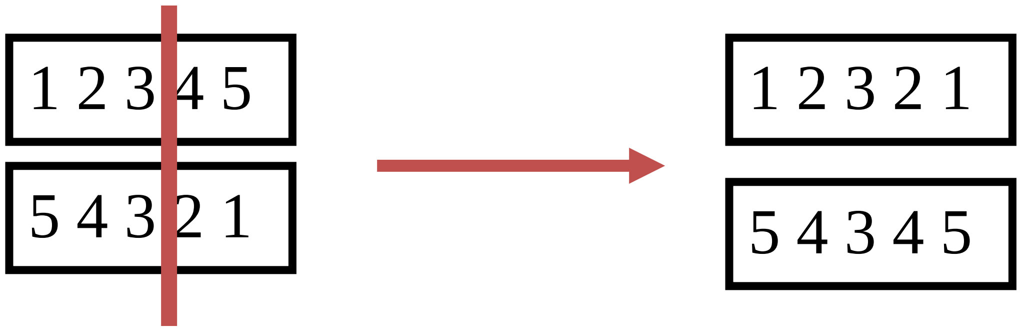
Subset:	BAABBAABBB	(Randomly generated)
Parents:	1010001110	0011010010
Offspring:	0011001010	1010010110

Permutation Representations

- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
  - Example: sort algorithm: important thing is which elements occur before others (order)
  - Example: Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (adjacency)
- These problems are generally expressed as a permutation:
  - if there are *n* variables then the representation is as a list of *n* integers, each of which occurs exactly once

Crossover for Permutations

- Normal crossover operators will often lead to inadmissible solutions



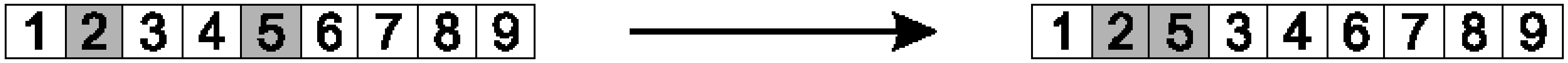
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Mutation operators for permutations

- Normal mutation operators lead to inadmissible solutions
  - e.g. bit-wise mutation : let gene *i* have value *j*
  - changing to some other value *k* would mean that *k* occurred twice and *j* no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

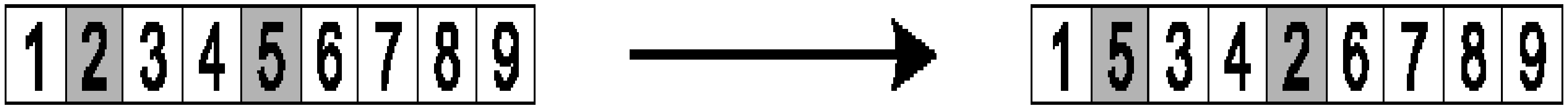
Insert Mutation for permutations

- Pick two allele (gene) values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information



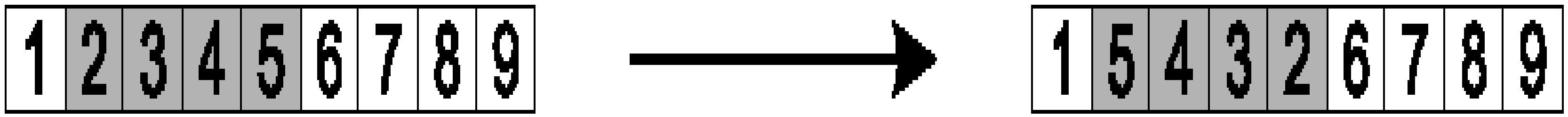
Swap mutation for permutations

- Pick two alleles (genes) at random and swap their positions
- Preserves most of adjacency information (4 links broken)
- disrupts order more



Inversion mutation for permutations

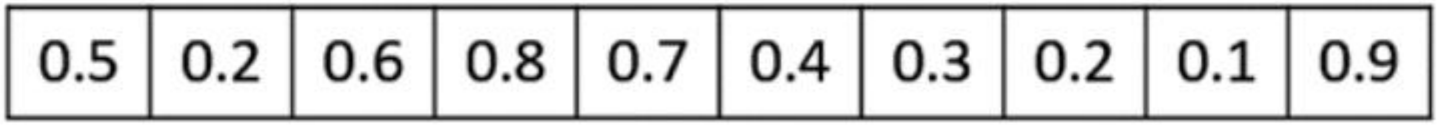
- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information



Floating-point (real values) Representation

- Many problems occur as real valued problems, e.g. continuous parameter optimization $f : \Re^n \rightarrow \Re$

- The chromosome will be an array of floating point variables

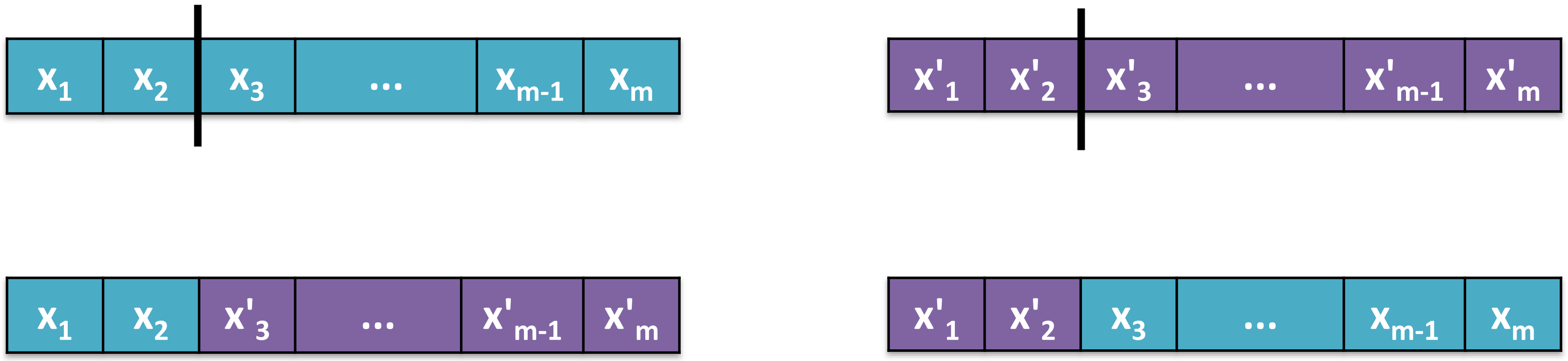


0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9

- This can serve in optimization of a multi-variate function $y=f(x_1, x_2, \ldots x_m)$

- Crossover is the same as in bit-string chromosomes.
- Mutation is different

Crossover over FP Chromosomes



Mutation over FP Chromosomes

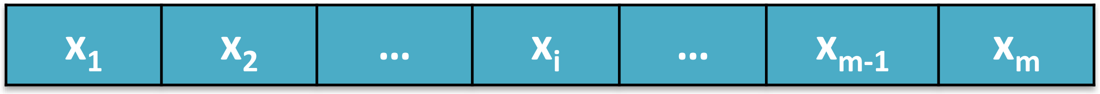
- General scheme of floating point mutations

$$\bar{x} = \left\langle x_1,\ \ldots, x_l \right\rangle \to \bar{x}' = \left\langle x_1', \ldots, x_l' \right\rangle$$
$$x_i, x_i' \in \left[LB_i, UB_i \right]$$

- Two kinds of FP mutations:
  - Uniform
  - Non-uniform

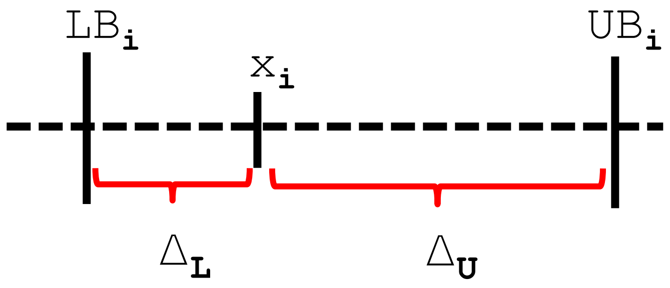
Uniform FP Mutation

$$x_i' \text{ drawn randomly (uniform) from } [LB_i, UB_i]$$



Given the above chromosome **X** in a particular generation **G**:

- Each gene (variable) has a range
- Gene x_i is a FP value inside chromosome **X** at generation **G**
- To mutate gene x_i
  1. Generate random number $r_{i1} \in [0, 1]$
     - $\Delta = \Delta_L$ if $r_{i1} \leq 0.5$
     - $\Delta = \Delta_U$ if $r_{i1} > 0.5$
     - This means equal chance to go left or right
  2. Generate random number $r_{i2} \in [0, \Delta]$
     - if $\Delta = \Delta_L$ then $x_{i\text{-new}} = x_i - r_{i2}$
     - if $\Delta = \Delta_U$ then $x_{i\text{-new}} = x_i + r_{i2}$



$$\Delta_L = x_i - LB_i$$

$$\Delta_U = UB_i - x_i$$

Non-uniform FP Mutation

Some methods proposed such as time-varying range of change

x_1	x_2	…	x_i	…	x_{m-1}	x_m

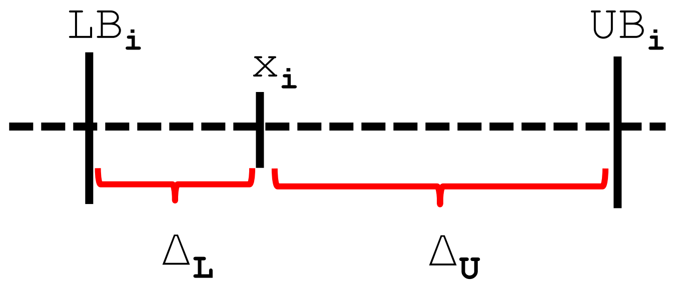
Given the above chromosome **X** in a particular generation **G**:

- Each gene (variable) has a range
- Gene x_i is a FP value inside chromosome **X** at generation **G**
- To mutate gene x_i
  1. Generate random number $r_{i1} \in [0, 1]$
     - $y = \Delta_L$ if $r_{i1} \le 0.5$
     - $y = \Delta_U$ if $r_{i1} > 0.5$
  2. Let $\Delta(t, y)$

     $=$ value of mutation at generation t

     $= y(1 - r^{(1-t/T)\wedge b})$

     where:
     - r = random number $\in [0, 1]$
     - t = current generation
     - T = maximum number of generations
     - b = dependency factor $\approx 1…5$



$$\Delta_L = x_i - LB_i$$

$$\Delta_U = UB_i - x_i$$

Non-uniform FP Mutation

- Analysis of Equation:

  Δ(t,y) = value of mutation at generation **t**

  $$= y(1-r^{(1-t/T)^b})$$

  where:

  - r = random number ϵ [0, 1]
  - t = current generation
  - T = maximum number of generations
  - b = dependency factor ≈ 1…5

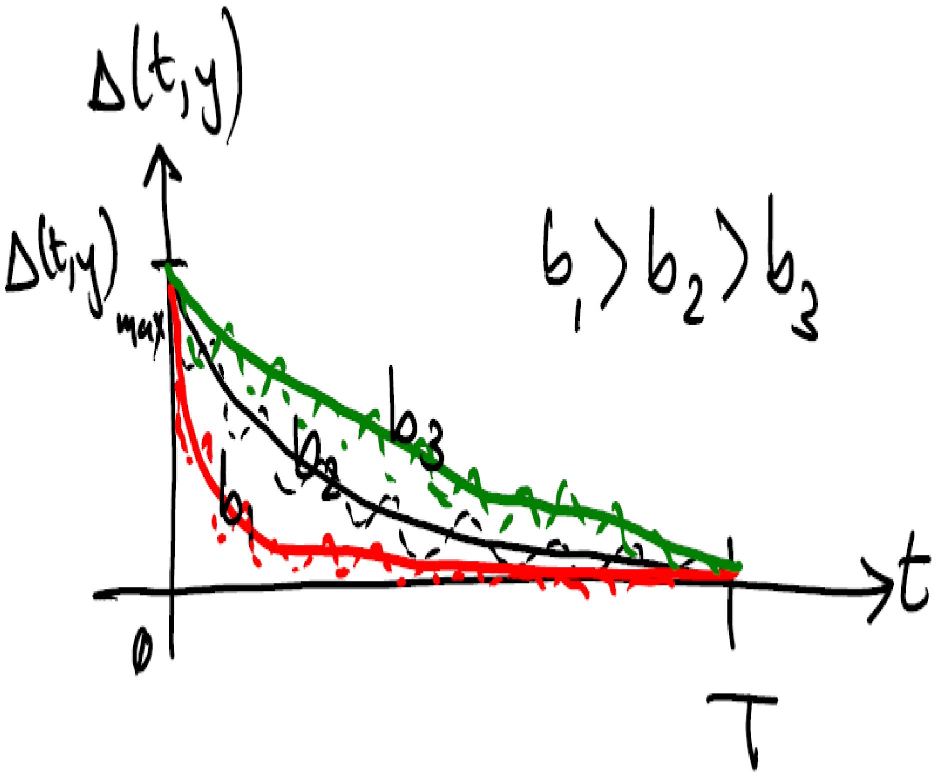
    (Controls the curve of mutation)

  - At t=0:

  Δ(t,y) = Maximum value of mutation

  - At t=T:

  Δ(t,y) = 0 (No mutation)



Crossover or Mutation?

- Decades of long debate: **which one is better or necessary?**

- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have different roles
  - mutation-only-GA is **possible**, crossover-only-GA **would not** work. Why??

Mutation

- Causes movement in the search space (local or global)
- Restores lost information to the population
- Mutation is necessary because some important genes might be missing from all the initial population.

Crossover

- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)

Crossover or Mutation?

- ✓ **Exploration:** Discovering promising areas in the search space, i.e. *gaining information* on the problem
- ✓ **Exploitation:** Optimising within a promising area, i.e. *using information*

There is co-operation AND competition between them

- **Crossover is explorative**: it makes a *big* jump to an area somewhere "in between" two (parent) areas
- **Mutation is exploitative:** it creates random *small* diversions, thereby staying near (in the area of) the parent

Crossover OR mutation?

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0’s on first bit in the population, ?% after performing *n* crossovers)
- To hit the optimum you often need a ‘lucky’ mutation