Evolutionary Computing

Genetic Algorithms

Dr. Petr Musilek

Department of Electrical and Computer Engineering
University of Alberta

Fall 2019

Genetic Algorithms

- search algorithms based on the mechanics of natural selection and natural genetics (Goldberg ’89)
- exploit the principle of the survival of the fittest and natural evolution to create a novel and innovative search strategy;
- generally characterized by:
  - Population-based optimization techniques
  - Stochastic reproduction
  - Fitness function

GAs Applications

- Scheduling problems (e.g, job-shop scheduling, traveling salesman problem)
- Optimization of network topologies
- Resource allocation over a distributed system
- Electronic circuit design
- Aircraft design
- Game playing
- Training of fuzzy systems or artificial neural networks

GAs: Terminology

Population of potential solutions: the collection of solution points (also called individuals)

Chromosome: an individual solution represented as an array or sequence of strings (also called genotype)

Gene: a string in the chromosome

Alleles: the values, or the states, that genes might have (0 or 1 in the case of binary encoding)

Locus: the position of a gene in a chromosome.

Main Components of a GA

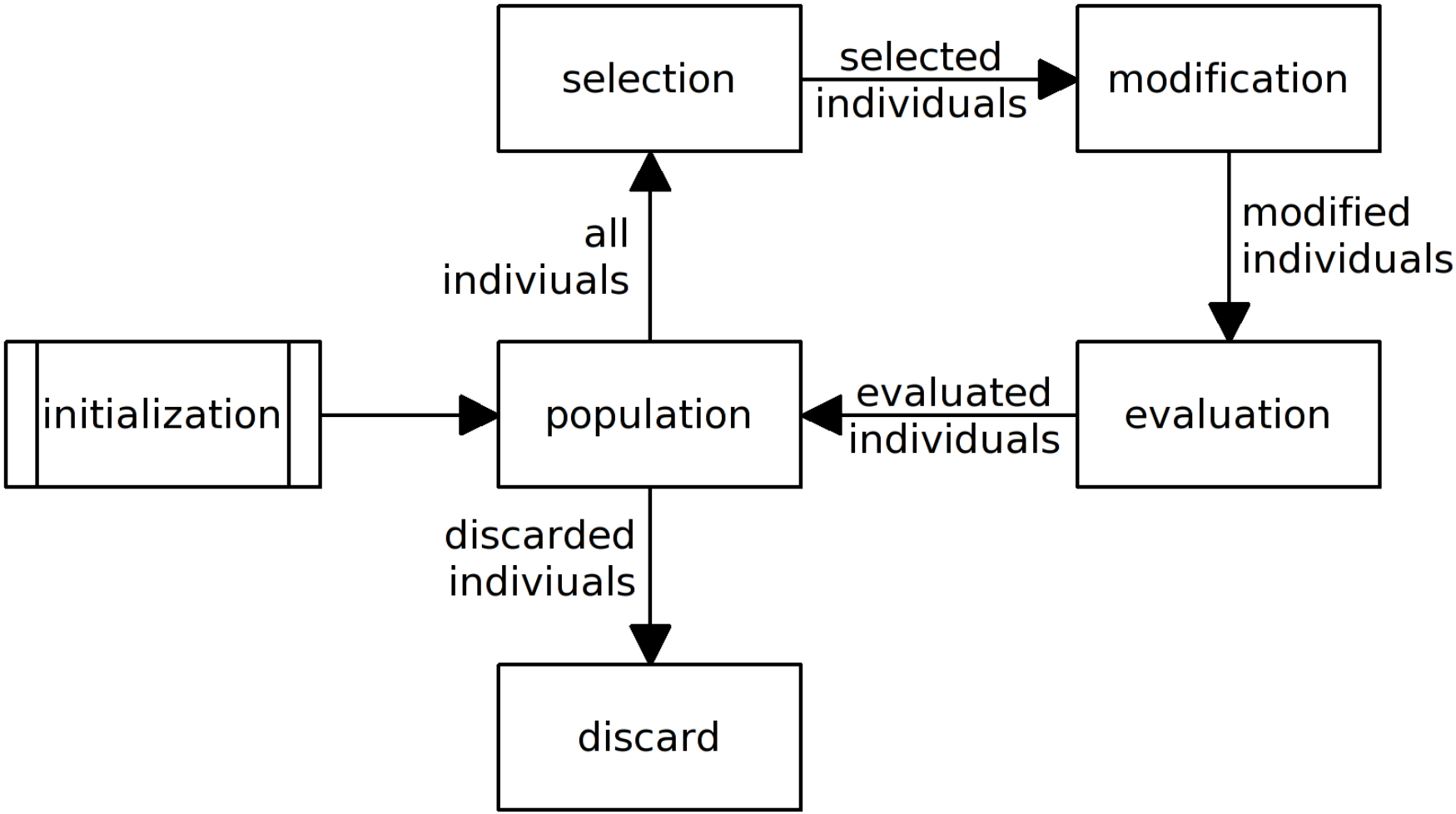
A problem to solve, and ...

- Encoding method (*genes, chromosomes*)
- Initialization procedure (*population creation*)
- Evaluation function (*fitness*)
- Selection of parents (*reproduction*)
- Genetic operators (*mutation, recombination*)
- Parameter settings (*practice and art*)

Simple Genetic Algorithm

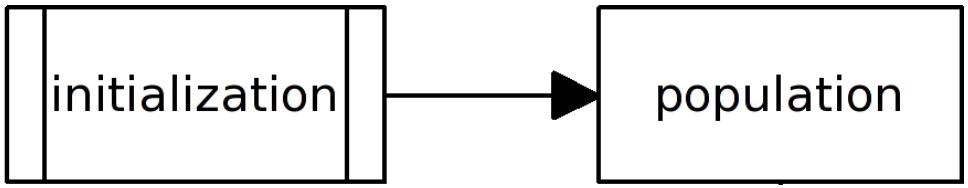
```
create initial population;
compute fitness of individuals;
REPEAT
   select ind. based on fitness;
   perform reproduction and mutation;
   update the current population;
UNTIL (stopping criteria)
```

GA - execution cycle



Population

At the beginning of the cycle, population is initialized; its mebers represent candidate solutions to the problem at hand

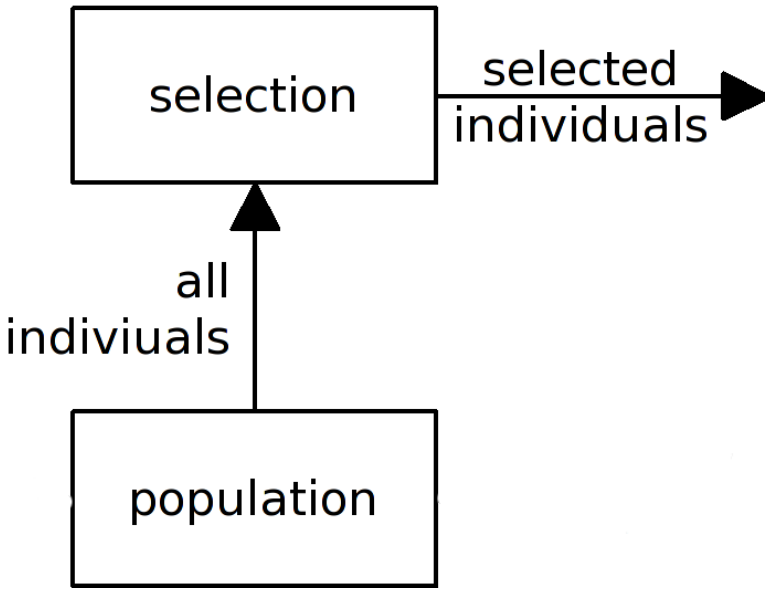


The solutions are represented in form of chromosomes, e.g.

- Bit strings
- Real numbers
- Permutations of element
- Lists of rules
- Program elements
- ... any data structure ...

Selection

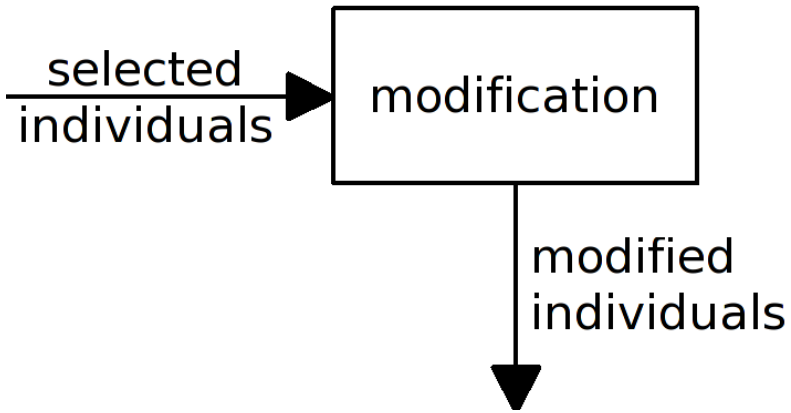
Parents to undergo modification (reproduction) are selected at random with chance biased in relation to their fitness (chromosome evaluation)



Selection methods include

- Fitness proportional selection (roulette wheel)
- Ranking
- Tournament

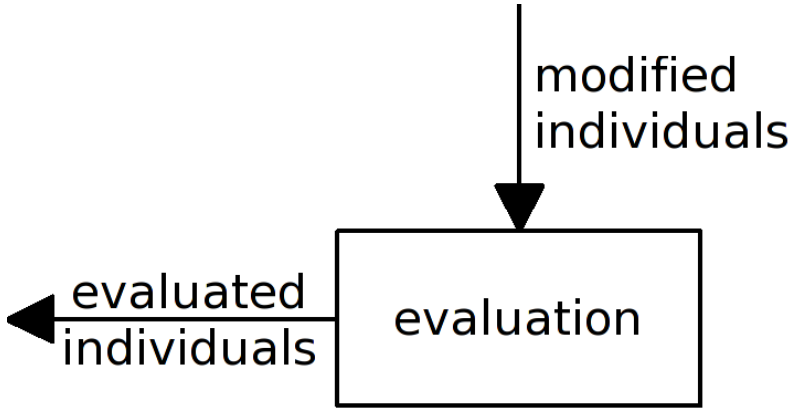
Modification



Modifications are stochastically triggered and based on the following (genetic) operators:

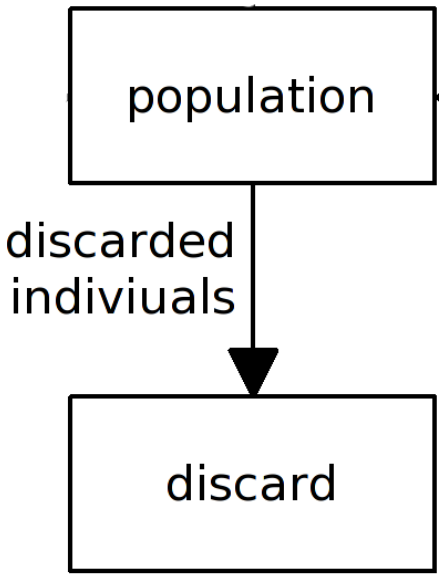
- Crossover (recombination)
- Mutation

Evaluation



- Evaluator decodes a chromosome and assigns it a fitness measure
- It is the only link between the GA and the problem it is solving (genotype–phenotype mapping)

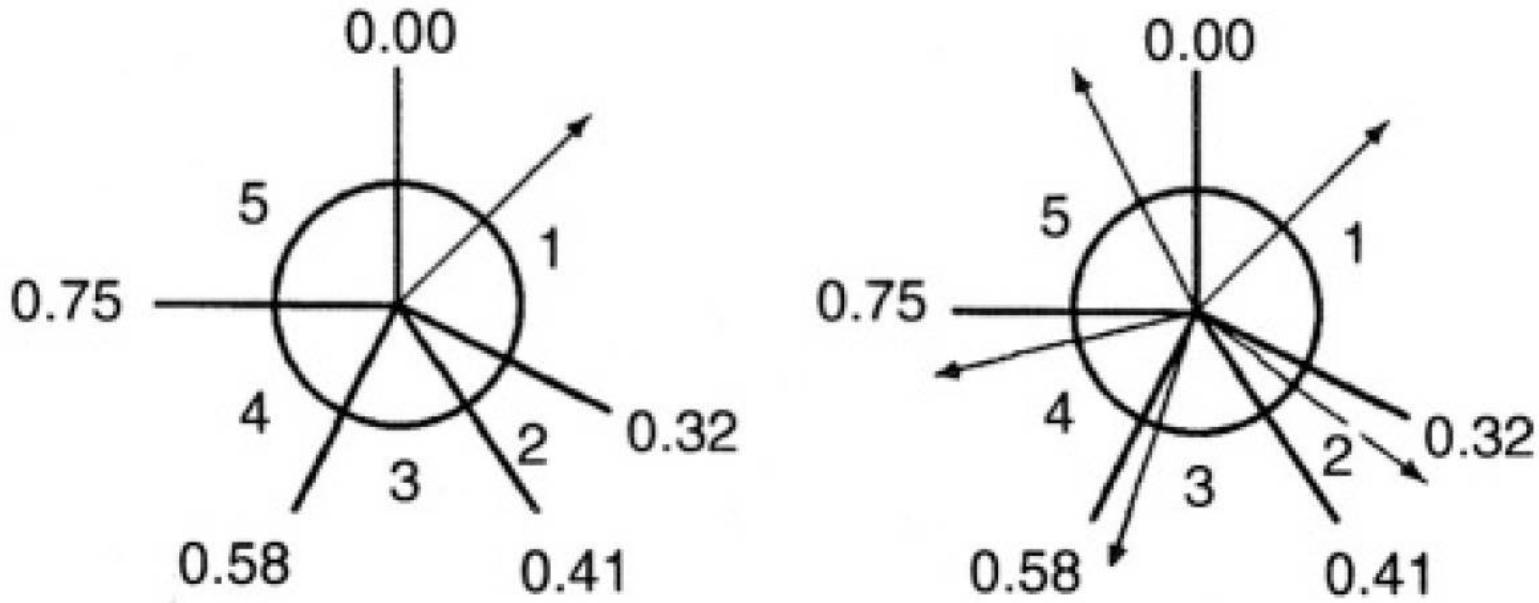
Deletion



- Low–fit members of the population are discarded to make space for the high–fit children while keeping the population size constant (so caled *steady–state GA*)
- In *generational GA*, entire population is replaced in each generation (the entire old population is discarded)

Selection – roulette wheel

Parents are selected randomly with bias corresponding to their fitness – similar to a biased roulette wheel with uneven sectors



Alternative to speed up the selection process: SUS (Stochastic Universal Selection) with multiple evenly–spaced pointers

Selection – alternative methods

Roulette wheel – simple and intuitive, but:

- assumes non-negative (positive or zero) fitness values
- assumes that fitness must be maximized
- requires computation of global statistic (total sum of fitness values)
- fitness–distribution related problems
  - a ”super–performer” will take over population leading to premature convergence
  - most individuals with about the same fitness values – selection becomes almost random

... alternative methods are needed

Ranking selection

Consists of two steps:

- all individuals are ranked according to their fitness and sorted (ascending or descending)
- selection is applied based on the rankings (similar to roulette wheel selection, but based only on relative performance of the individuals – absolute values of fitness are discarded)

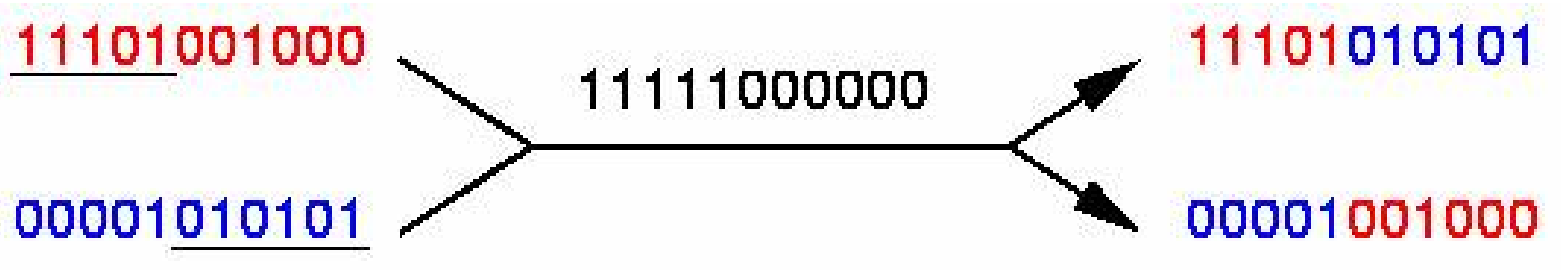
Tournament selection

Consists of two steps:

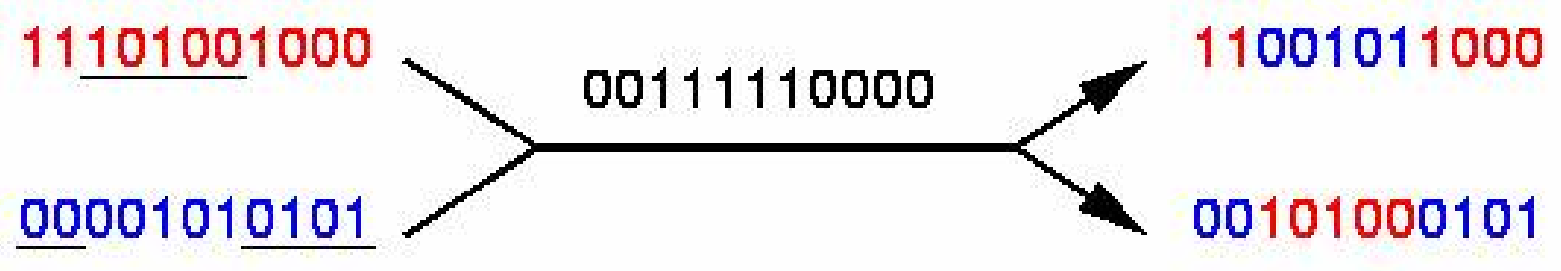
- random choice of k individuals from population (k - tournament size)
- performing tournament within the subpopulations
  - deterministic (more common)
  - fitness proportional (or ranked)

Crossover

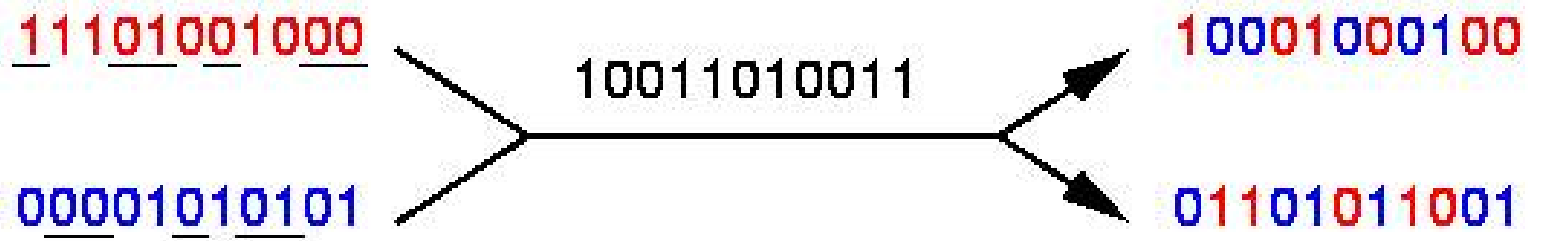
Single point crossover



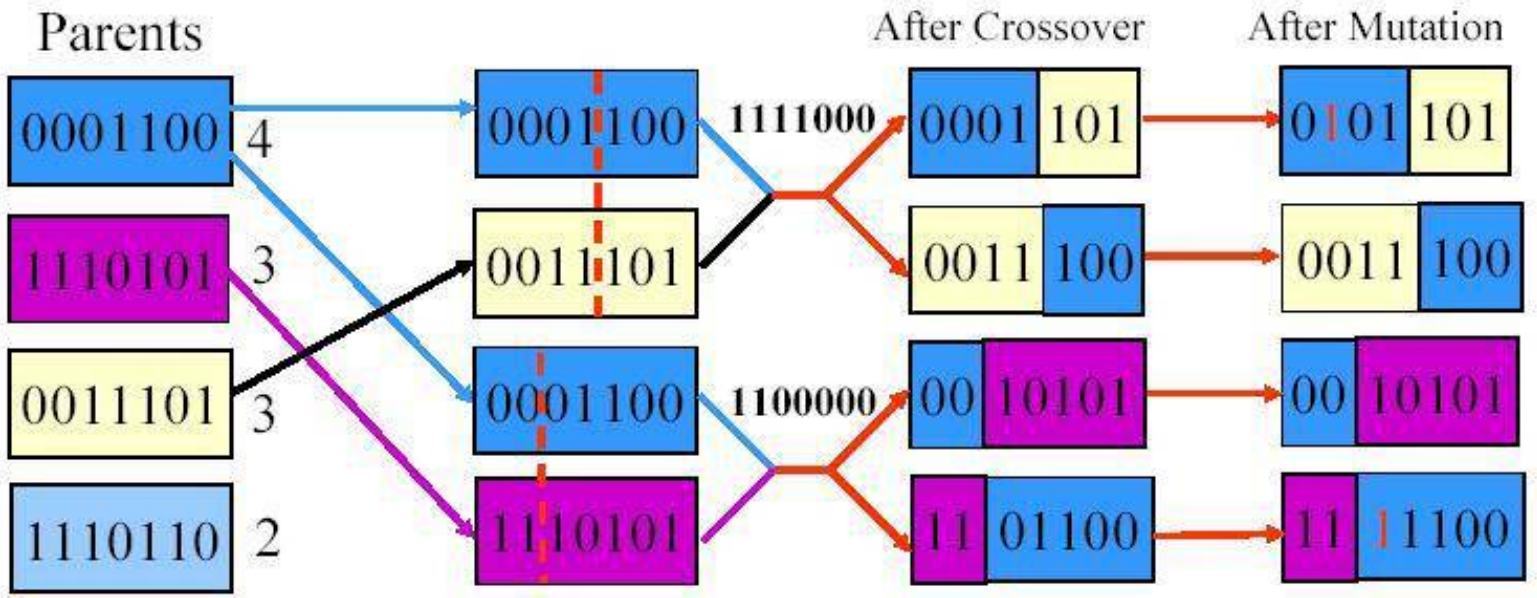
Two point crossover



Uniform crossover – applied at each locus with certain *p*



Example of selection and modifications



Example: Designing a can

Problem: Design a cylindrical can

- volume of at least 300ml
- minimal cost of can material

Consider 2 parameters: diameter d and height h
NLP problem:

Minimize	$f(d,h) = c\left(\frac{\pi d^2}{2} + \pi dh\right)$
subject to	$g(d,h) = \frac{\pi d^2 h}{4} \geq 300$
variable bounds	$d_{min} \leq d \leq d_{max}$, and $h_{min} \leq h \leq h_{max}$

In–class solved example ...

Representation of individuals

First and very important stage of developing GA

Right representation must be chosen for problem on hand:

- Binary representation: simplest and often used (sometimes improperly); suitable for problems with flags or Boolean decision variables;
- Integer representation: e.g. when finding optimal values for a set of integer variables
- Real–valued (floating–point) representation: often the most sensible choice
- Permutation representations: for order–based (e.g. scheduling) or adjacency–based problems (TSP); requirement of no repetition.

Mutation (binary representations)

Typically, each bit is considered separately and is allowed to flip with certain small probability p_m

Suitable value for the bitwise mutation rate p_m depends on problem. Typically, it is chosen such that on average between one gene per generation and one gene per offspring (child) is mutated.

Mutation (integer representations)

Two principal types of mutation which mutate each gene with probability p_m

Random reseting – bit flipping of binary mutation is extended to such that a new value (from valid set of integer values) is chosen with probability p_m (suitable for cardinal attributes)

$$1\ 2\ 3\ 5\ 7\ 9 \rightarrow 1\ 2\ 3\ 8\ 2\ 9$$

Creep mutation – adding a small value (positive or negative) to each gene with probability p_m (suitable for ordinal attributes); the magnitude of the value is drawn from a distribution around the current allele value at given position.

$$1\ 2\ 3\ 5\ 7\ 9 \rightarrow 1\ 2\ 3\ 6\ 6\ 9$$

Mutation (floating–point representations)

Allele value of each gene is changed randomly within its given domain (lower and upper bound). Two types are available according to the distribution from which gene values are drawn.

Uniform mutation – with position-wise probability p_m (analogous to bit–flipping and random resetting)

Nonuniform mutation – analogous to creep mutation for integer representation; Typically, Gaussian distribution is used. Sometimes, p_m is set equal to 1 and the amount of mutation is controlled by changing the standard deviation of the Gaussian (each gene gets mutated, but only some by a significant value due to small SD).

Mutation (permutation representations)

Genes cannot be considered independently; rather, the allele values are moved around in the chromosome. The mutation parameter p_m thus has the meaning of likelihood that the string as a whole undergoes mutation. There are four major types:

Swap mutation – alleles at two randomly chosen positions are exchanged

$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \rightarrow 1\ 5\ 3\ 4\ 2\ 6\ 7\ 8\ 9$$

Mutation (permutation representations - cont.)

Insert mutation – two alleles are randomly chosen and one is moved next to the other (others are shuffled to make the room)

$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \rightarrow 1\ 2\ 5\ 3\ 4\ 6\ 7\ 8\ 9$$

Scramble mutation – the values of the entire string, or its randomly chosen subset, have their positions scrambled

$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \rightarrow 1\ 3\ 5\ 4\ 2\ 6\ 7\ 8\ 9$$

Inverse mutation – two loci are randomly and the order in which the values appear between those positions are reversed (works well for adjacency based problems, because the other mutations could break a large number of links in such problems, e.g. TSP)

$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \rightarrow 1\ 5\ 4\ 3\ 2\ 6\ 7\ 8\ 9$$

Recombination (crossover) for binary representations

Three types of recombination (mentioned earlier) are used:

- One-point crossover

$$0\ 0\ 0\ 0 \mid 1\ 0\ 0\ 0\ 0 \rightarrow 0\ 0\ 0\ 0 \mid 0\ 0\ 0\ 0\ 1$$
$$1\ 1\ 0\ 1 \mid 0\ 0\ 0\ 0\ 1 \rightarrow 1\ 1\ 0\ 1 \mid 1\ 0\ 0\ 0\ 0$$

- *N*–point crossover (e.g. 2–point crossover)

$$0\ 0\ 0\ 0 \mid 0\ 0\ 0 \mid 0\ 0\ 0 \rightarrow 0\ 0\ 0\ 0 \mid 1\ 1\ 1 \mid 0\ 0\ 0$$
$$1\ 1\ 1\ 1 \mid 1\ 1\ 1 \mid 1\ 1\ 1 \rightarrow 1\ 1\ 1\ 1 \mid 0\ 0\ 0 \mid 1\ 1\ 1$$

Here, the position(s) to split both parents are randomly chosen from interval $[0, l-1]$, where l is the length of the string.

Recombination (crossover) for binary representations

- Uniform crossover

Here, each gene is considered separately to be inherited from either parent (random string of length l is generated; loci with sub-threshold values are inherited from first parent, otherwise from the second parent). For example

$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \rightarrow 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0$$
$$1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \rightarrow 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1$$

is the result of comparing random vector

$$\{0.35, 0.62, 0.18, 0.42, 0.83, 0.76, 0.39, 0.51, 0.36, 0.12\}$$

to threshold 0.5.

Recombination for integer representations

Operators analogous to those used for binary representations are used (no blending of allele values is desirable as it could lead to non–integer values)

- One-point crossover
- *N*–point crossover
- Uniform crossover

Recombination for floating-point representations

There are two classes of options:

Discrete recombination: analogous to the previous cases

- One-point crossover
- *N*–point crossover
- Uniform crossover

Arithmetic (intermediate) recombination: blending the alleles from both parents

- Simple arithmetic recombination
- Single arithmetic recombination
- Whole arithmetic recombination

Advantage: Introduction of new genetic information (not just moving existing values around)

Arithmetic (intermediate) recombination

In each case, parameter α is either set to 0.5, or chosen randomly from $[0, 1]$; parents are $\langle x_1, \ldots, x_l \rangle$ and $\langle y_1, \ldots, y_l \rangle$

- Simple arithmetic recombination (k is randomly chosen)
  Child 1: $\langle x_1, \ldots, x_k, [\alpha x_{k+1} + (1-\alpha) y_{k+1}], \ldots, [\alpha x_l + (1-\alpha) y_l] \rangle$
  Child 2: $\langle y_1, \ldots, y_k, [\alpha y_{k+1} + (1-\alpha) x_{k+1}], \ldots, [\alpha y_l + (1-\alpha) x_l] \rangle$
- Single arithmetic recombination
  Child 1: $\langle x_1, \ldots, x_{k-1}, [\alpha x_k + (1-\alpha) y_k], x_{k+1}, \ldots, x_l \rangle$
  Child 2: $\langle y_1, \ldots, y_{k-1}, [\alpha y_k + (1-\alpha) x_k], y_{k+1}, \ldots, y_l \rangle$
- Whole arithmetic recombination Child 1: $x_i^{\text{new}} = \alpha x_i^{\text{old}} + (1-\alpha) y_i^{\text{old}}$
  Child 2: $y_i^{\text{new}} = \alpha y_i^{\text{old}} + (1-\alpha) x_i^{\text{old}}, i = 1, 2, \ldots, k$

How do GAs work?

GAs work with very simple operations: string copying, substring exchange, bit (or gene) alteration – why does this work?

There is no straightforward general answer; however, for the simple genetic algorithm, there is the following theorem.

Schema theorem

Short, low order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

Schema

Schema is a similarity template describing a subset of strings with similarities at certain string positions; consider:

Alphabet	$V = \{0, 1, *\}$
Sample string	$*11*0**$ (length = 7)
Symbol	$*$ is a wildcard (don't care, either 0 or 1)

Ex 1: $S = *110111 \rightarrow 1110111, 0110111$ (2 strings)

Ex 2: (4 strings)

$S = 1*1011* \rightarrow 1010110, 1010111, 1110110, 1110111$

Ex 3: $S = 1***0** \rightarrow$ (? strings)

Schema description

Schema length: $d(S)$, the distance between the first and last specific string position

$$1*1011* \rightarrow d = 6 - 1 = 5$$
$$*11*0** \rightarrow d =$$

Schema order: $o(S) = l - m$, the number of not don't care positions

$$1*1011* \rightarrow o = 7 - 2 = 5$$
$$*11*0** \rightarrow o =$$

Schema evolution

The schema theory describes how each schema in the population evolves through time:

- Let $\Phi(S, g)$ be the number of instances representing schema S at generation g.
- What to expect of $\Phi(S, g + 1)$?.

Selection and fitness of schemata

Probability of selecting an individual i:

$$P_i(g) = \frac{f_i(g)}{\sum f_j(g)}, \text{ or } \frac{f_i(g)}{nf_{\text{ave}}(g)}$$

Average fitness value of the members of schema S can be defined in the following way

$$f_{\text{ave}}(S, g) = \frac{\sum_S f_i}{\Phi(S, g)},$$

where the sum goes through only the individuals covered by schema S.

Expectation for $g + 1$

The expected value of the number of instances in schema S at generation $g + 1$ is:

$$E[\Phi(S, g + 1)] = \frac{f_{\text{ave}}(S, g)}{f_{\text{ave}}(g)} \Phi(S, g),$$

which means the value is **proportional** to the average fitness of those individuals in schema S at generation g and **inversely proportional** to the average fitness of all individuals at this generation.

Schema growth equation

Assume that fitness of S remains constantly above average by β, i.e.

$$E[\Phi(S, g+1)] = \frac{f_{\text{ave}}(g) + \beta f_{\text{ave}}(g)}{f_{\text{ave}}(g)} \Phi(S, g),$$

which can be simplified as

$$E[\Phi(S, g+1)] = (1+\beta)\Phi(S, g).$$

This means that **better**/worse schemata will receive exponentially **increasing**/decreasing number of trials in the subsequent generations.

Schema theory illustrated

String Processing

String No.	Initial Population (Randomly Generated)	x Value (Unsigned Integer)	$f(x)$ x^2	pselect, $\frac{f_i}{\Sigma f}$	Expected count $\frac{f_i}{\bar{f}}$	Actual Count from Roulette Wheel
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4.0
Average			293	0.25	1.00	1.0
Max			576	0.49	1.97	2.0

Schema Processing

	schema	Before Reproduction: String Representatives	Before Reproduction: Schema Average Fitness $f(H)$
H_1	1 * * * *	2,4	469
H_2	* 1 0 * *	2,3	320
H_3	1 * * * 0	2	576