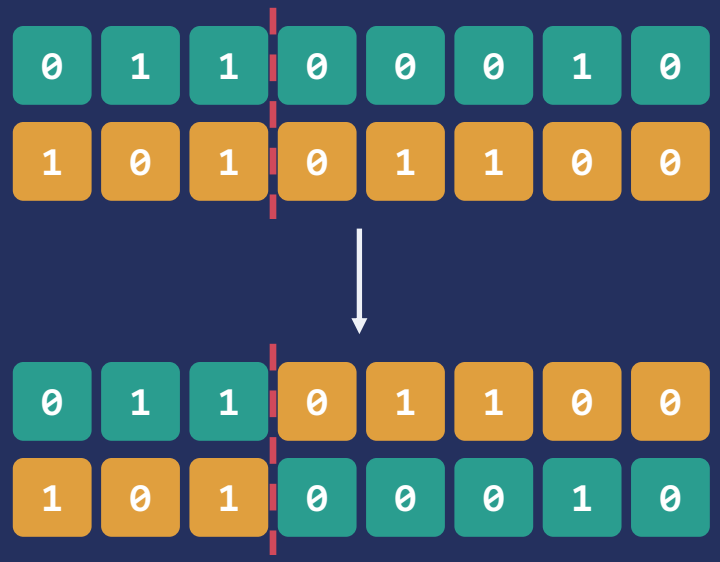
LECTURE 6 · SOFT COMPUTING

Genetic Algorithms

Optimization problems, the GA concept, and its operators: encoding, selection, crossover and mutation



parents → offspring

Prof. Dr Abdul Hadi M. Alaidi

What you will be able to do

- State an optimization problem: objective, constraints and design variables
- Explain how a GA borrows from genetics and natural selection
- Encode solutions as binary, real-valued, order and tree chromosomes
- Select parents by roulette wheel, rank, tournament and elitism
- Apply binary, real-coded and order crossover operators
- Mutate offspring, scale fitness and decide when a GA should stop

Today’s path: from a problem to a working GA

01 Optimization problems

Objective, constraints, types, and why classical methods struggle.

02 Concept of GA

Genetics and evolution as a search method; SGA and SSGA.

03 Encoding

Binary, real-valued, order and tree chromosomes.

04 Selection I

Fitness; canonical, roulette-wheel and rank selection.

05 Selection II

Tournament, steady-state, Boltzmann, elitism; pressure vs. diversity.

06 Crossover I

Binary crossover: one-point to three-parent.

07 Crossover II

Real-coded and order (permutation) crossover.

08 Mutation

Mutation, termination and fitness scaling.

01

Solving optimization problems

What an optimization problem is · its general form · types · classical methods and their limits

Optimization means finding the best value

Optimization

$$y = f(x)$$

Find x* where f(x*) is a minimum or a maximum: the smallest or largest value f can take over the allowed x.

Not optimization problems

$2x - 6y = 11$

An equation: once x is chosen, y is fixed. There is nothing to make best.

$3x + 4y \geq 56$

A constraint on its own: it allows many (x, y) but gives no way to rank them.

Every optimization problem has

f

Objective

what to minimize or maximize: cost, weight, distance, profit

g

Constraints

conditions every acceptable solution must satisfy

x

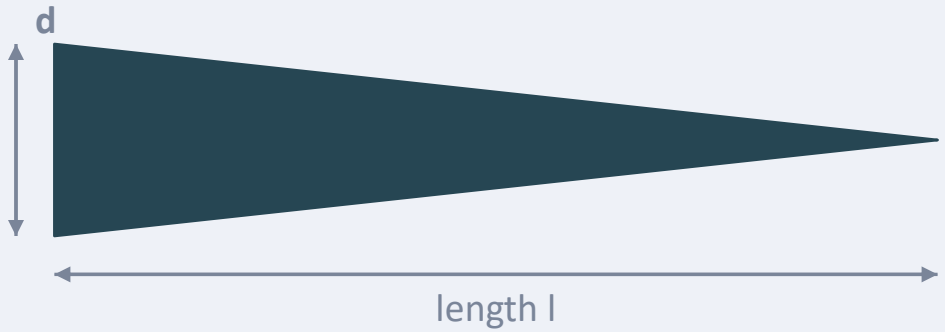
Design variables

the quantities we are free to choose

Search space: every combination of values the design variables can take.

Designing a pointer as an optimization problem

a cone of material with density ρ



- As light as possible: minimum mass M.
- Strong enough not to break: strength s = $f_2(d, l, \rho)$.
- Stiff enough: tip deflection δ = $f_1(d, l, \rho)$ must be negligible.

$$M = \frac{1}{3}\pi\left(\frac{d}{2}\right)^2 l\rho = \frac{1}{12}\pi d^2 l\rho$$

The formulation

Objective

$$\text{Minimize} \quad M = \frac{1}{12}\pi d^2 l\rho$$

Functional constraints

$$\delta \le \delta_{th}, \qquad s \ge s_{th}$$

Bounds on the design variables

$$d_{min} \le d \le d_{max}, \qquad l_{min} \le l \le l_{max}$$

The general form of an optimization problem

$$\text{Maximize (or minimize)} \quad y_i = f_i(x_1, x_2, \dots, x_n), \qquad i = 1,2, \dots, k, \;\; k \geq 1$$

$$\text{subject to} \quad g_i(x_1, x_2, \dots, x_n) \; \text{ROP}_i \; c_i, \qquad i = 1,2, \dots, j, \;\; j \geq 0$$

$$\text{and} \quad x_i \; \text{ROP} \; d_i, \qquad i = 1,2, \dots, n, \;\; n \geq 1$$

Reading the form

- ROP is a relational operator (≤, =, ≥); c_i and d_i are constants.
- k = 1: single objective. k > 1: multi-objective (a later lecture).
- j = 0: no functional constraint, so the problem is unconstrained.
- The last line bounds each design variable x_i.

Exercise: state these in the general form

- Travelling salesman problem
- 0-1 knapsack problem
- Graph colouring problem
- Job–machine assignment problem
- Coin change problem
- Optimal binary search tree construction

Types of optimization problems

Type	What defines it	Example
Unconstrained	No functional constraint, only bounds	$\min (x_1 - 5)^2 + (x_2 - 3)^3,\ x_1, x_2 \geq 0$
Constrained	One or more functional constraints	$\max f(x)$ subject to $g_i(x) \geq c_i,\ i = 1 \ldots k$
Integer	Every design variable is an integer	$\min 2x_1 + x_2,\ x_1 + x_2 \leq 3,\ 5x_1 + 2x_2 \leq 9$
Real-valued	Every design variable is real	the pointer: d and l are lengths
Mixed-integer	Some variables integer, the rest real	number of machines (integer) and their speed (real)
Linear	Objective and all constraints linear	$\max 2x_1 + x_2,\ x_1 + x_2 \leq 3,\ 5x_1 + 2x_2 \leq 10$
Non-linear	Objective or any constraint non-linear	$\max {x_1}^2 + 5{x_2}^3,\ {x_1}^4 + 3{x_2}^2 \leq 629,\ 2{x_1}^3 + 4{x_2}^3 \leq 133$

The labels combine: the integer example is also linear and constrained.

Classical optimization methods



Linear programming

- Graphical method
- Simplex method

Non-linear programming

Single variable

- Analytical: set derivatives to zero
- Elimination: exhaustive, Fibonacci, dichotomous, golden section
- Interpolation: quadratic, cubic, direct root

Multi-variable

- Constrained: Lagrangian method
- Unconstrained: random walk, univariate, pattern search, steepest descent, conjugate gradient, quasi-Newton, variable metric

Specialized algorithms

- Dynamic programming
- Branch and bound
- Greedy method
- Divide and conquer

The analytical method: derivatives locate the optimum

$$f'(x^*) = 0 \;\Rightarrow\; x^* \text{ is a candidate point}$$

Let n be the order of the first non-zero derivative at x*:

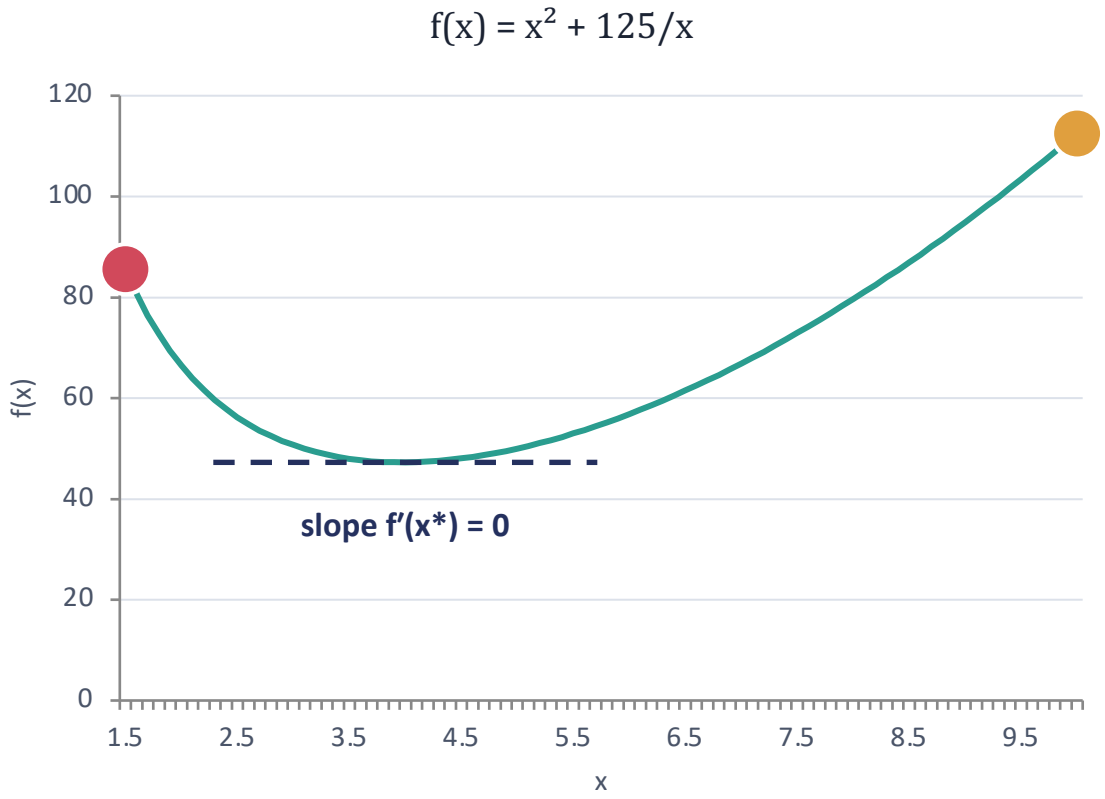
- n odd: an inflection (saddle) point, not an optimum
- n even and $f^{(n)}(x^*) > 0$: a local minimum
- n even and $f^{(n)}(x^*) < 0$: a local maximum

Exercise: $f(x) = x^2 + 125/x$

$$f'(x) = 2x - \frac{125}{x^2} = 0 \;\Rightarrow\; x^* = \sqrt[3]{62.5} = 3.97$$

$$f''(x^*) = 2 + \frac{250}{x^{*3}} = 6 > 0$$

Minimum at x* = 3.97 with f(x*) = 47.25



Works for a smooth f in one variable. Many variables, discontinuities or integer x break it.

Duality, and where classical methods fall short

Duality principle

A maximization problem becomes a minimization problem, and vice versa:

$$\max f(x) \Leftrightarrow \min y^* = -f(x)$$

$$\max f(x) \Leftrightarrow \min y^* = \frac{1}{f(x)}$$

The second form needs f(x) > 0. So one solver, or one GA, written for minimization handles both kinds.

Limits of the classical methods

1. Computationally expensive on large problems.
2. May fail when the objective is discontinuous.
3. Hard to run in parallel.
4. Discrete (integer) variables are difficult to handle.
5. Not adaptive: a method suits one problem type.

Evolutionary algorithms, such as the GA, were developed to address these limits.

02

Concept of genetic algorithms

Evolutionary computing · genetics and evolution · the GA loop · simple and steady-state GA

Algorithms inspired by nature

Biological behaviour

Genetics and evolution	→	**Genetic algorithm (GA)**
Ant colonies finding food	→	**Ant colony optimization (ACO)**
Bird flocks and fish schools	→	**Particle swarm optimization (PSO)**
The human nervous system	→	**Artificial neural network (ANN)**

Physical behaviour

Annealing: slow cooling of metal	→	**Simulated annealing (SA)**

Evolutionary algorithms: GA, ACO and PSO all improve a population of candidate solutions over many iterations.

A GA suits objectives that are

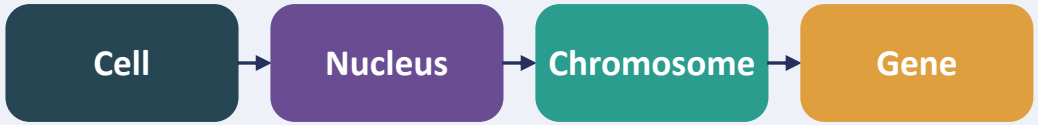
- very hard to model mathematically
- NP-hard (combinatorial) and expensive to search
- spread over many discrete and/or continuous parameters

Where it came from

- John Holland, University of Michigan: idea in the 1960s, book in 1975
- Genetics: Gregor Mendel (1865)
- Evolution by natural selection: Charles Darwin (1859)

Genetics in one slide

From cell to gene

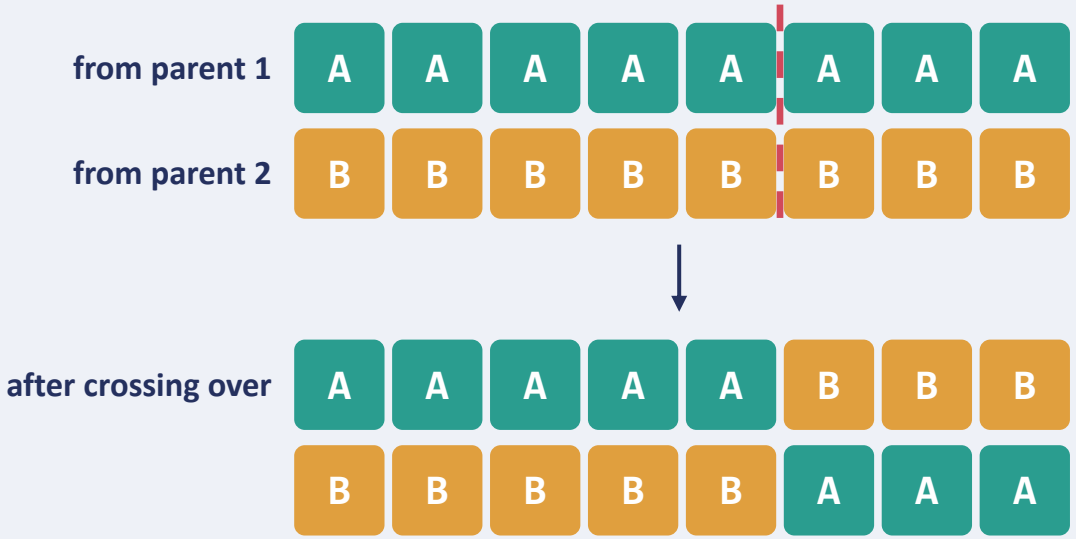


- Each cell carries genes, the basic units of heredity, on its chromosomes.
- Genes are written in DNA, a spiral helix unique to each organism.
- The number of chromosomes is fixed for a species.

Species	Mosquito	Frog	Human	Goldfish
Chromosomes	6	26	46	94

Reproduction and crossing over

Reproductive cells (gametes) are haploid: half the chromosomes. Two gametes combine into a diploid cell with a full set.



A random crossing-over point mixes the parents' information: endless diversity.

Evolution by natural selection, in GA terms

1 **Heredity**

Offspring inherit many characteristics of their parents.

2 **Diversity**

Characteristics vary from one individual to the next.

3 **Selection**

Only a small share of the offspring survive to adulthood.

4 **Ranking**

Which offspring survive depends on what they inherited.

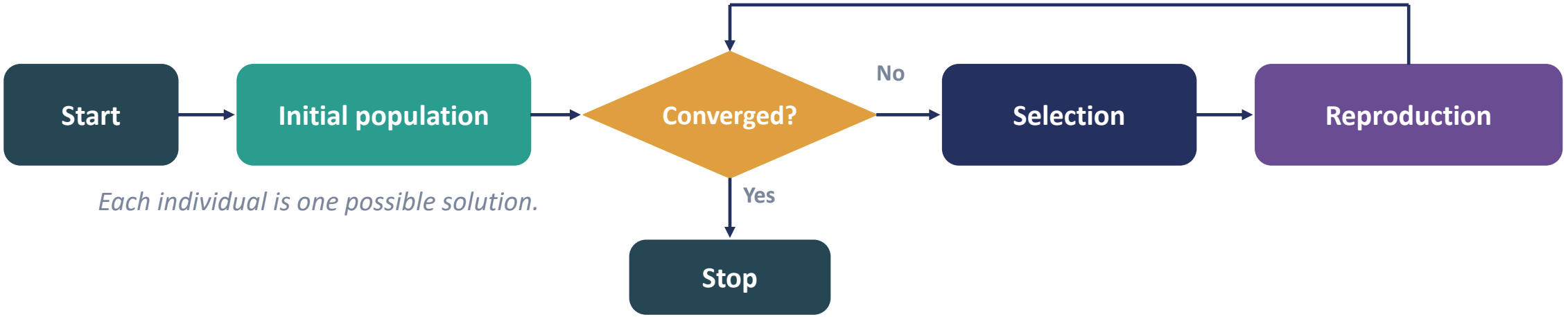
μ **Mutation**

Random changes keep the population changing when variation settles.

In nature	In a genetic algorithm
Gene	one design variable, or one bit of it
Chromosome	an encoded solution (a string)
Individual	one candidate solution
Population	the set of solutions at one iteration
Fitness	how good a solution is
Selection	choosing parents for the mating pool
Crossover	combining two parents into offspring
Mutation	a small random change to an offspring
Generation	one iteration of the GA loop

What a genetic algorithm is

A genetic algorithm is a population-based probabilistic search and optimization technique that works on the mechanisms of natural genetics and natural evolution.



Each individual is one possible solution.

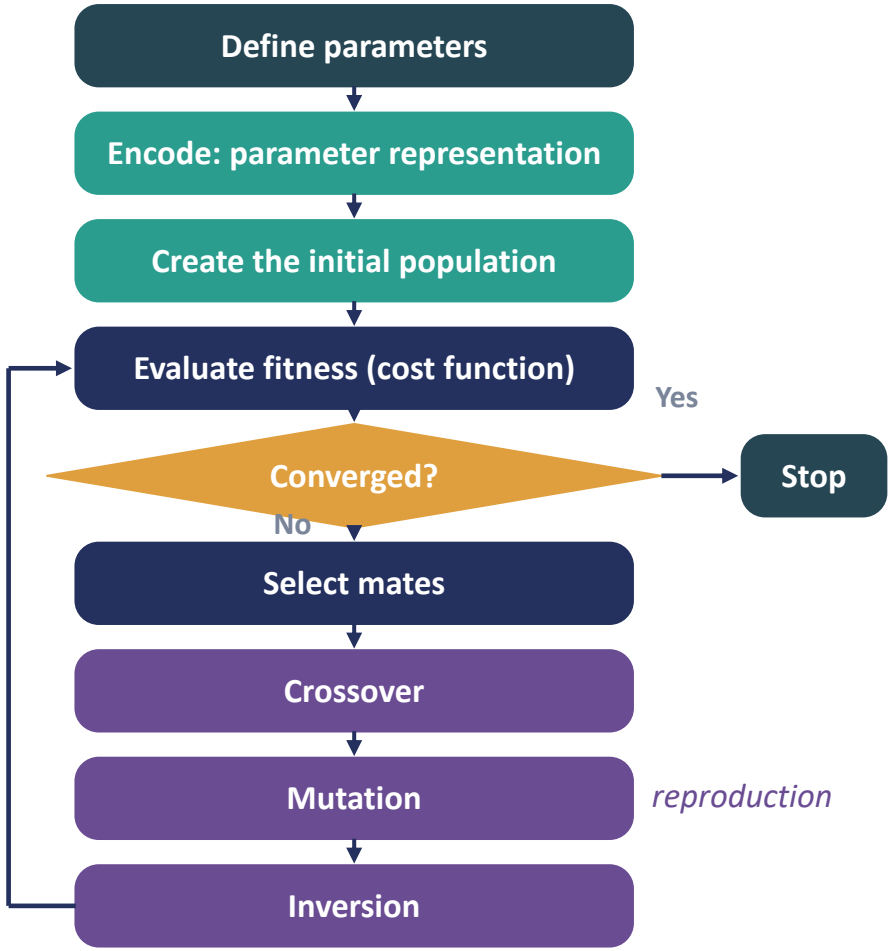
↻ Iterative: the same loop runs generation after generation.

? A search technique: it samples the search space, guided by fitness.

∞ Can run with or without a convergence test (e.g. a fixed number of generations).

≈ No guarantee of the global optimum: it usually ends at a good, possibly local, optimum.

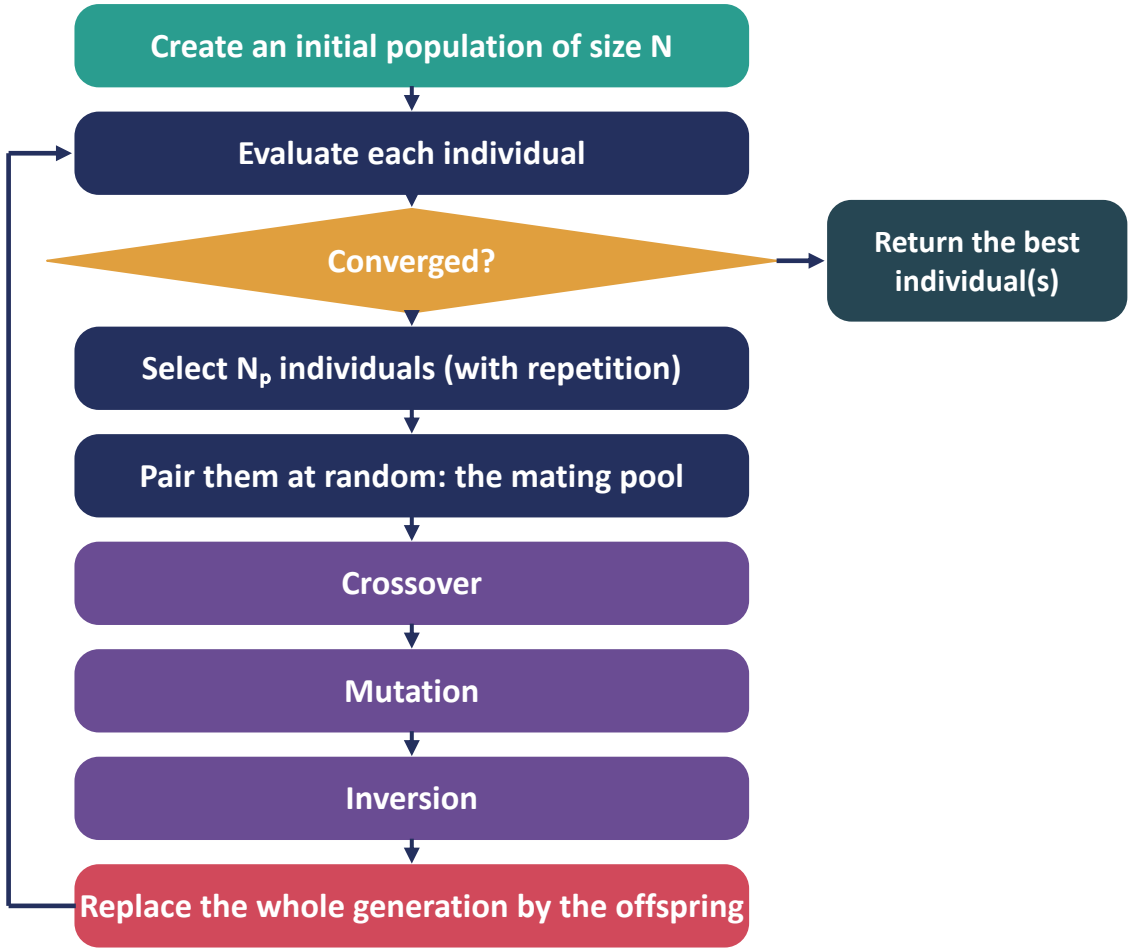
The GA framework in detail



GA operator	The question it answers
Encoding	How is a solution represented?
Fitness evaluation	How good is a solution?
Selection (mating pool)	Which solutions get to breed?
Crossover	How are two parents combined into new, diverse solutions?
Mutation	How do we explore solutions the parents cannot reach?
Inversion	How do we move from one optimum to another?
Convergence test	When do we stop?

Before running a GA, identify: the objective(s), the constraints, the input parameters, how fitness is evaluated (a formula or an algorithm), and how solutions are encoded and decoded.

Simple genetic algorithm (SGA)



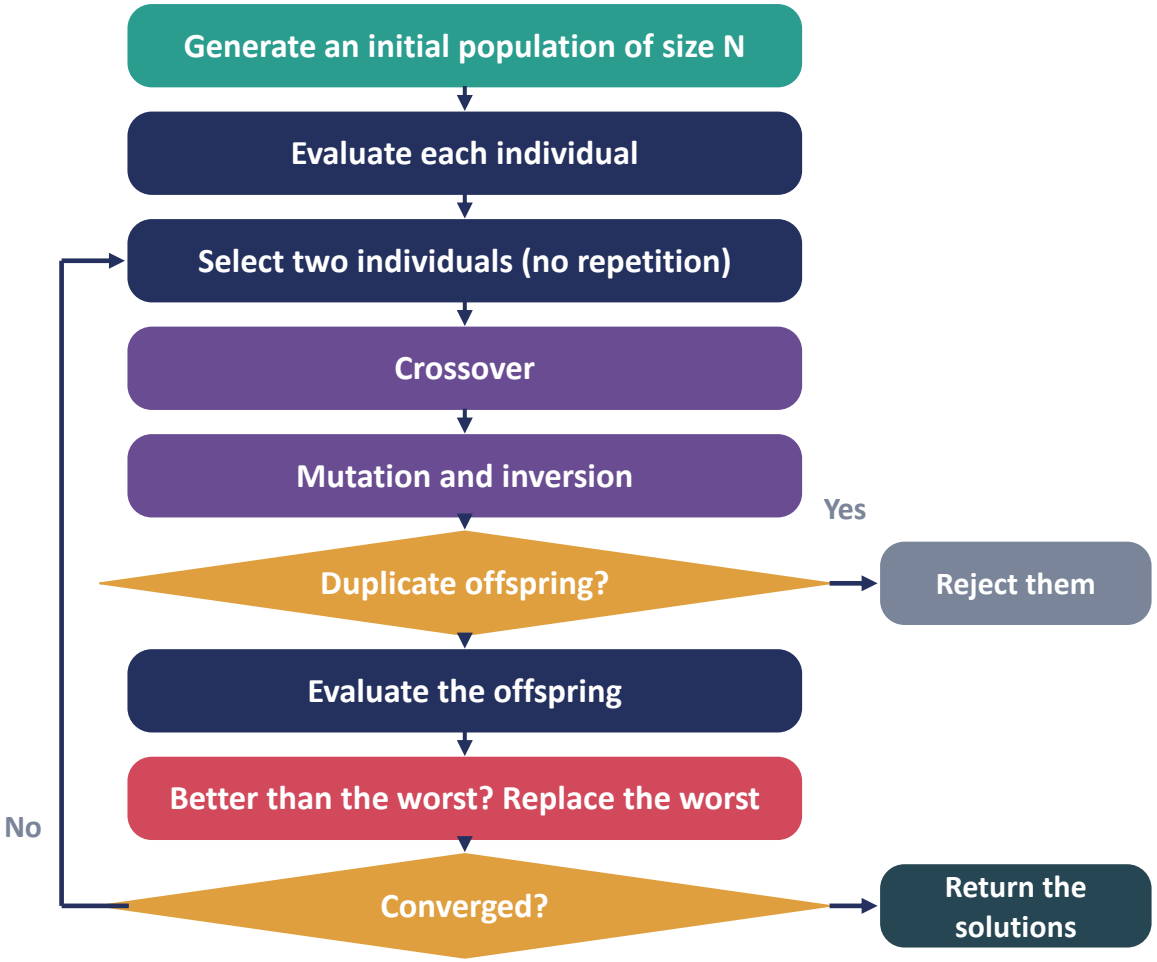
SGA parameters

Symbol	Meaning
N	initial population size
$N_p = p\%$ of N	size of the mating pool
δ	convergence threshold
μ, η, ρ	mutation, inversion and crossover rates

Features

- Generation gap near 1: most of the population is replaced each generation.
- Selection favours fit individuals, so the average fitness tends to rise.
- The best individual can appear in any generation: keep the best so far.
- Works well with a large population, but is computationally expensive.

Steady-state genetic algorithm (SSGA)



Features

- Small generation gap: only two offspring per generation.
- Suits a small population, long chromosomes, and evaluation that is cheap compared with duplicate checking.
- The population only ever loses its worst members, so the average fitness rises steadily.
- A third strategy, the messy GA, uses variable-length chromosomes (not covered).

Limitations

- Can stick at a local optimum if crossover, mutation and inversion do not diversify enough.
- Premature convergence.
- Stagnation: weak individuals vanish and many trials bring no gain.

03

GA operators: encoding

Genotype and phenotype · binary · real-valued · order · tree encoding

Individuals, populations and chromosomes

Two elements of the metaphor

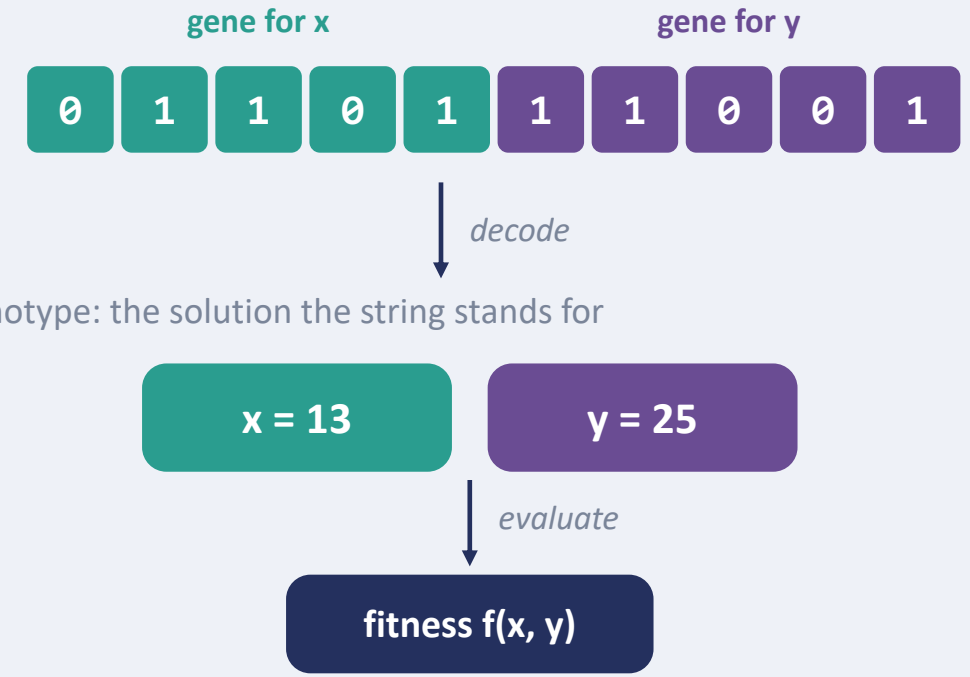
- Individual: a single solution, defined by its chromosome.
- Population: the set of individuals at one instant of the search.
- Gene: the GA's representation of one factor (design parameter). Its domain may be continuous, discrete or symbolic.

A population of four

individual 1	0	1	1	0	1	1	1	0	0	1
individual 2	1	1	0	0	1	0	0	1	1	1
individual 3	0	0	1	1	0	1	0	1	1	0
individual 4	1	0	1	0	0	0	1	1	0	1

Genotype and phenotype

Genotype: the coded string the operators work on



The genotype-to-phenotype mapping decides the speed and accuracy of the search.

Four ways to encode a solution

Binary encoding

Each gene is a string of bits.

1	0	1	1	0	0	1	1	1	0

→ Binary-coded GA

Real-value encoding

Each gene is a real number (or a symbol).

5.28	−475.36	0.07

→ Real-coded GA

Order (permutation) encoding

The chromosome is a sequence of distinct elements.

A	D	C	E	B

→ Order GA (permuted GA)

Tree encoding

The chromosome is a tree of objects, written compactly.

2	1	H	3	V

→ Tree-encoded GA

Example 1: A knapsack chromosome

Given

Three items: I_1 weighs 10 kg and is worth \$60, I_2 weighs 20 kg (\$100), I_3 weighs 30 kg (\$120).

The knapsack holds at most W = 50 kg. Each item is either taken whole or left behind.

Required

(a) Design a binary chromosome for a solution.

(b) List every non-empty candidate with its weight and value.

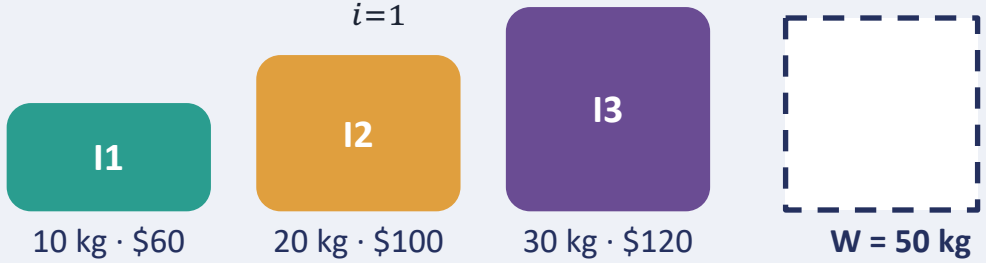
(c) Which candidates are feasible, and which one is optimal?

(d) How many candidates are there for n items?

Binary encoding: the 0-1 knapsack problem

$$\text{Maximize} \sum_{i=1}^{n} c_i\, x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i\, x_i \le W, \quad x_i \in \{0,1\}$$



One bit per item: 1 = take it, 0 = leave it.

$x_1\, x_2\, x_3$ = **0 1 1** take I_2 and I_3

n items give $2^n - 1$ non-empty candidates: 7 here, about 10^{30} for 100 items.

String	Items	Weight	Value	Feasible
001	I3	30 kg	$120	yes
010	I2	20 kg	$100	yes
011	**I2, I3**	**50 kg**	**$220**	**yes**
100	I1	10 kg	$60	yes
101	I1, I3	40 kg	$180	yes
110	I1, I2	30 kg	$160	yes
111	I1, I2, I3	60 kg	$280	no

Best feasible: 011, weight 50 kg, value $220. Taking all three items would weigh 60 kg.

Binary encoding of numeric variables

One integer variable

$$\text{Minimize } f(x) = x^2 + \frac{125}{x}, \quad 0 \le x \le 15,\ x \text{ integer}$$

16 values need 4 bits: 0000 = 0 … 1111 = 15.

x = 13: `1` `1` `0` `1` (1·8 + 1·4 + 0·2 + 1·1)

Two variables: concatenate the genes

$$\text{Maximize } f(x, y) = x^3 - x^2 y + x y^2 + y^3$$

$$x + y \le 10, \quad 1 \le x \le 10, \quad -10 \le y \le 10$$

x y

`0` `1` `1` `0` `1` `1` `1` `0` `0` `1`

Advantages

- Bit operations are fast, so every GA operator is fast.
- Any optimization problem has a binary-coded GA.

Limitations

- Values must be converted to and from binary.
- Accuracy depends on the number of bits.
- Neighbouring values can differ in many bits (section 06).

Real values on a binary string

Step 1: decide the precision

$$\varepsilon = \frac{X_U - X_L}{2^n - 1} \qquad n = \left\lceil \log_2 \left(\frac{X_U - X_L}{\varepsilon} + 1 \right) \right\rceil$$

ε, the obtainable accuracy, is the gap between neighbouring decoded values.

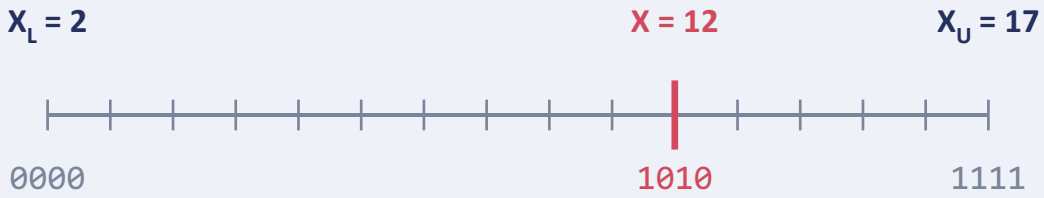
Step 2: decode

$$X = X_L + \frac{X_U - X_L}{2^n - 1} X_B$$

X_B is the integer value of the string: 00…0 gives X_L, 11…1 gives X_U.

Example: 1 ≤ x ≤ 16 with n = 6 bits gives ε = 15/63 = 0.238.

Worked: X_L = 2, X_U = 17, n = 4



16 strings, 15 equal gaps of ε = 1

string **1 0 1 0** X_B = 8 + 2 = 10

$$X = 2 + \frac{17 - 2}{2^4 - 1} \times 10 = 12$$

Encoding runs the other way: $X_B = (X - X_L)/\varepsilon$, rounded to the nearest integer.

Example 2: Precision and decoding

Given

A real variable X with $20.1 \le X \le 45.6$ is coded with an 8-bit binary string.

Required

(a) What accuracy ε can this representation obtain?

(b) Which string represents X = 34.35? How large is the coding error?

(c) Decode the string 10110011.

(d) How many bits would give an accuracy of at least 0.01?

Worked: 8 bits for 20.1 ≤ X ≤ 45.6

(a) Obtainable accuracy

$$\varepsilon = \frac{45.6 - 20.1}{2^8 - 1} = \frac{25.5}{255} = 0.1$$

Every decoded value is a multiple of 0.1 above 20.1.

(b) Encode X = 34.35

$$X_B = \frac{34.35 - 20.1}{0.1} = 142.5$$

Halfway between 142 = 10001110 (X = 34.30) and 143 = 10001111 (X = 34.40): the error is at most ε/2 = 0.05.

(c) Decode 10110011

$$X = 20.1 + 0.1 \times 179 = 38.0$$

The bits are worth 128 + 32 + 16 + 2 + 1 = 179: read them as an integer, then scale and shift.

(d) Bits for ε ≤ 0.01

$$n = \left\lceil \log_2 \left(\frac{25.5}{0.01} + 1 \right) \right\rceil = \lceil 11.32 \rceil = 12$$

12 bits give ε = 25.5/4095 = 0.0062.

Real-value encoding stores the numbers directly

The chromosome is the solution



- Best suited to continuous search spaces.
- Genes are the design parameters themselves, so there is no encoding or decoding step.
- Precision is that of the computer's floating-point numbers.
- Needs its own crossover and mutation (sections 07 and 08).

	Binary-coded	Real-coded
Gene	bits	real numbers
Precision	set by the number of bits n	floating point
Decoding	needed before every evaluation	none
Operators	bit operations: simplest, fastest	arithmetic: more costly
Weak spot	Hamming cliff (section 06)	parameters to tune (α, q)

Practice-sheet answer: binary coding gives the fastest execution; real coding gives the most direct representation.

Order encoding: the travelling salesman problem

Visit every city once, return home, travel least

$$\text{Minimize} \sum_{i=0}^{n-2} d\,(c_i, c_{i+1}) + d(c_{n-1}, c_0)$$

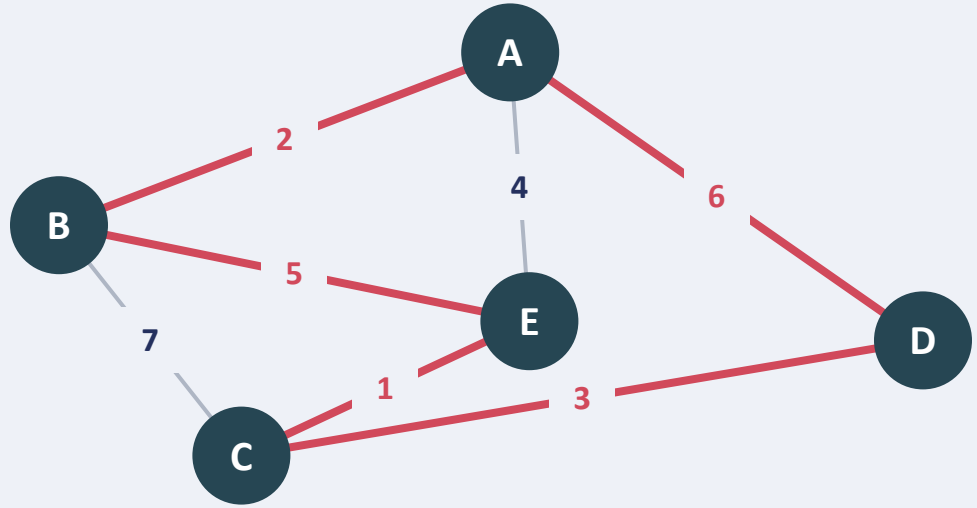
$$P = [c_0, c_1, \ldots, c_{n-1}], \quad c_i \neq c_j \text{ for } i \neq j$$

The chromosome is the visiting order, a permutation of the cities.

best tour A B E C D

cost 2 + 5 + 1 + 3 + 6 = 17

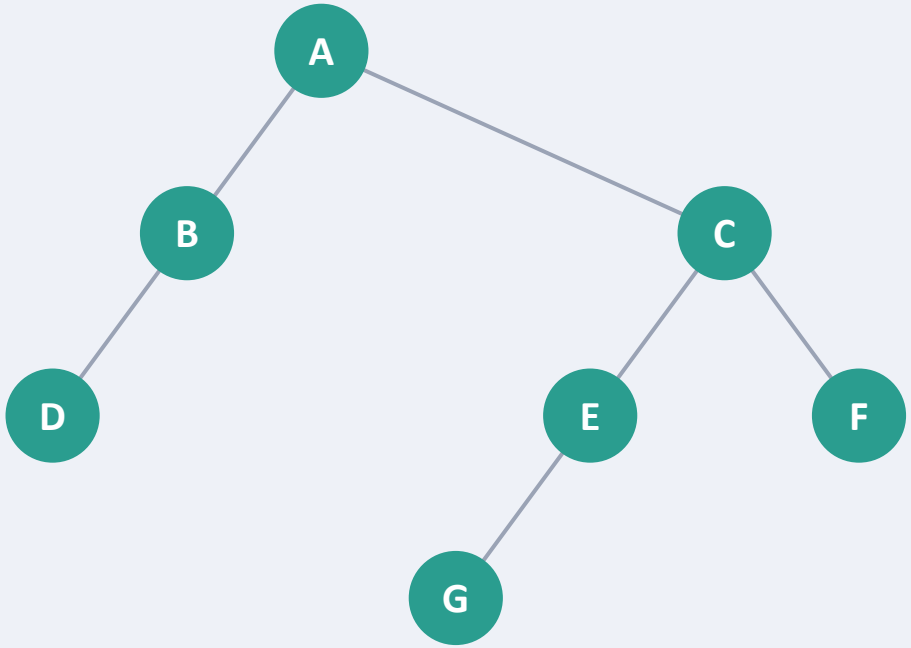
5 cities give (5 − 1)!/2 = 12 distinct tours; only 2 use existing roads. For n cities the count grows like n!.



d	A	B	C	D	E
A	0	2	∞	6	4
B	2	0	7	∞	5
C	∞	7	0	3	1
D	6	∞	3	0	∞
E	4	5	1	∞	0

Tree encoding: a tree written as a string

A binary tree



Traversal	Order	Result
In-order	left, root, right	D B A G E C F
Pre-order	root, left, right	A B D C E G F
Post-order	left, right, root	D B G E F C A

Post-order = Polish notation

$$a + b \div c \;\rightarrow\; a\ b\ c \div +$$

$$a + b - c \;\rightarrow\; a\ b + c\ -$$

A tree has exactly one post-order string, so the string can serve as the chromosome.

Floor planning: tree encoding in VLSI design

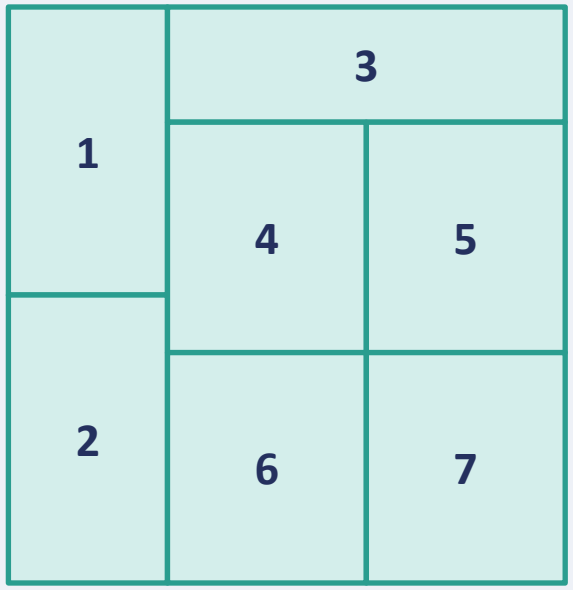
The problem

- Place n rectangular circuit blocks on a chip without overlap.
- Minimize area, wire length and delay.
- A slicing floor plan is a binary tree: leaves are blocks, inner nodes are cuts.

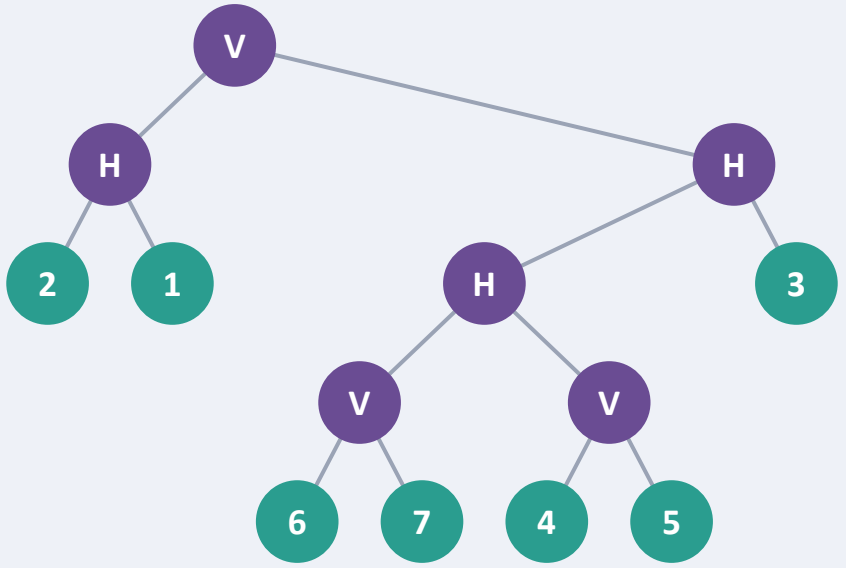
$$i\ j\ H\text{: } j \text{ on top of } i$$

$$i\ j\ V\text{: } i \text{ left of } j$$

Tree shapes alone grow as the Catalan numbers: $C_6 = 132$ shapes for 7 blocks, before choosing the cuts and the block order.



floor plan



slicing tree

Chromosome (Polish notation):

2	1	H	6	7	V	4	5	V	H	3	H	V

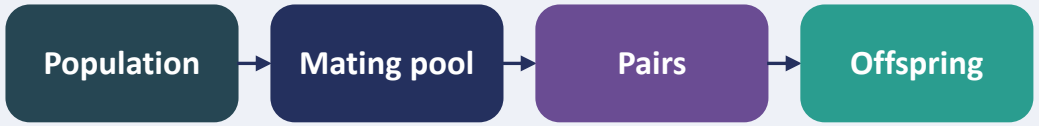
04

GA operators: selection I

Fitness evaluation · canonical · roulette-wheel · rank-based selection

Selection builds the mating pool

Breeding in a GA



- Selection copies individuals into the mating pool: survival of the fittest.
- Fitter parents are expected to have fitter offspring; random choice would ignore this.

Evaluating fitness

- One objective: fitness follows the objective value; for a minimization, use e.g. 1/cost.
- Several objectives: different scales and importance need extra care (a later lecture).

Tour	Cost	Fitness 1/cost	Rank
P1	11	0.091	4
P2	19	0.053	1
P3	16	0.063	2
P4	12	0.083	3
P5	**10**	**0.100**	**5**

Selection schemes in this lecture

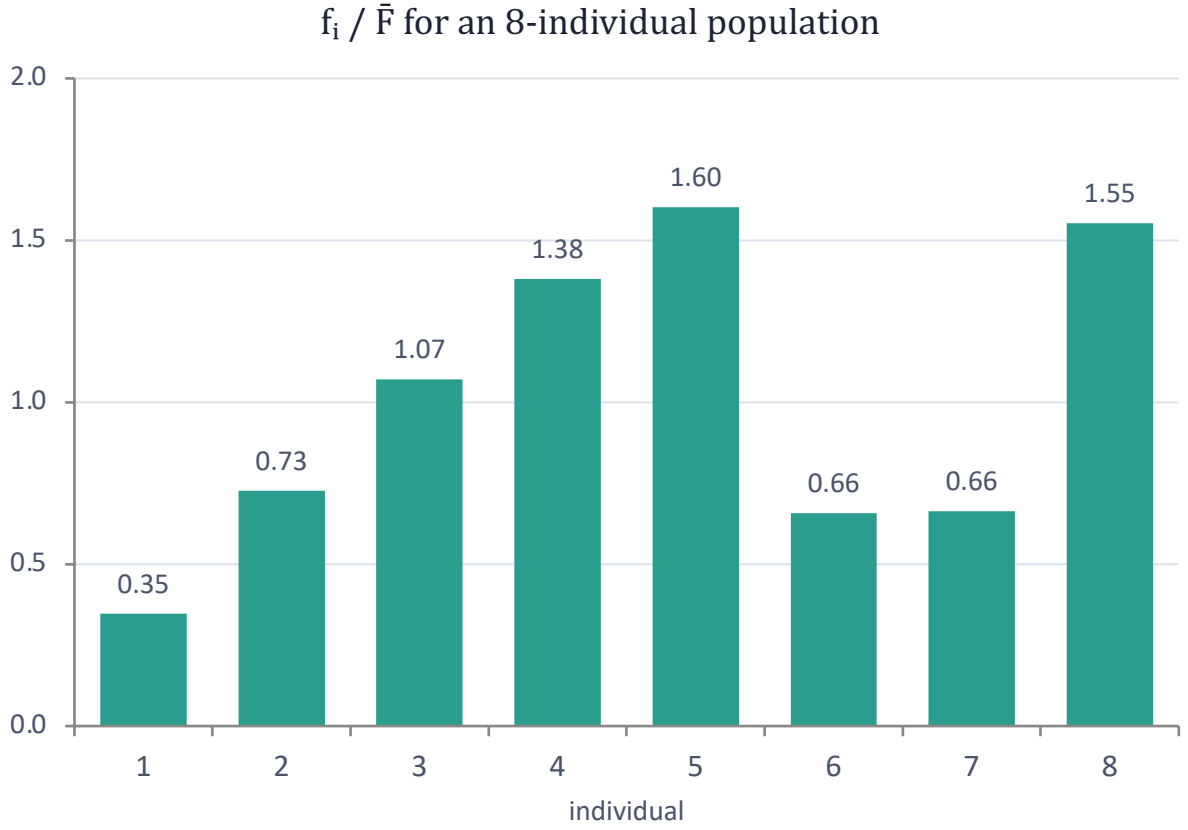
Section 04: canonical, roulette-wheel, rank-based

Section 05: tournament, steady-state, Boltzmann, elitism

Canonical (proportionate) selection

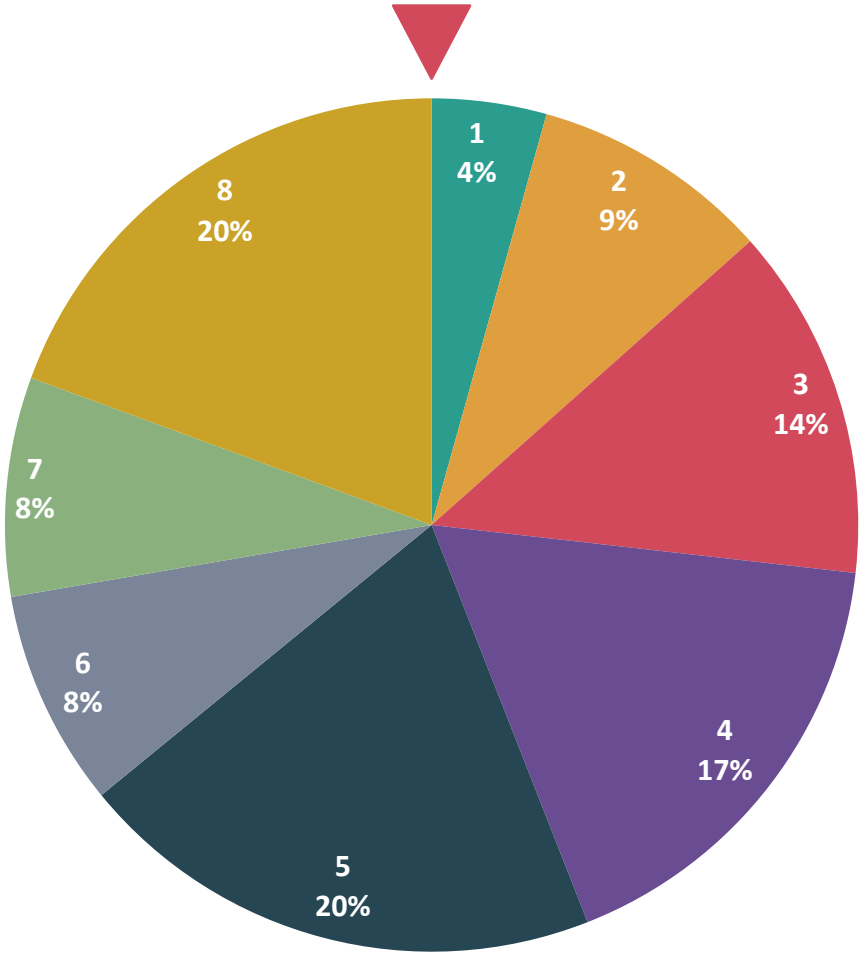
$$\text{fitness}(i) = \frac{f_i}{\bar{F}}, \qquad \bar{F} = \frac{1}{N}\sum_{i=1}^{N} f_i$$

- f_i is the evaluation of individual i; $\bar{F}$ is the population average.
- Individuals are copied into the mating pool in proportion to $f_i/\bar{F}$: a ratio of 1.6 means about 1.6 copies.
- The mating pool holds $N_p = p\% \times N$ individuals; the convergence rate depends on p.



$\bar{F}$ = 2.906. Bars above 1 expect more than one copy; bars below 1 expect less.

Roulette-wheel selection



$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

Each slice is proportional to fitness. Spin N_p times; the fixed pointer marks the winner.

1. Compute p_i for every individual.
2. Cumulative probability $P_i = p_1 + \ldots + p_i$ (so the last P_i is 1).
3. Draw a random number r in (0, 1).
4. Select individual j with $P_{j-1} < r \leq P_j$.
5. Repeat steps 3–4 until the pool holds N_p individuals.

An individual may be picked more than once. Convergence is fast.

Example 3: Roulette-wheel selection

Given

A population of N = 8 individuals has fitness values 1.01, 2.11, 3.11, 4.01, 4.66, 1.91, 1.93, 4.51.

Eight random numbers r in (0, 1): 0.26, 0.04, 0.48, 0.43, 0.09, 0.3, 0.61, 0.89.

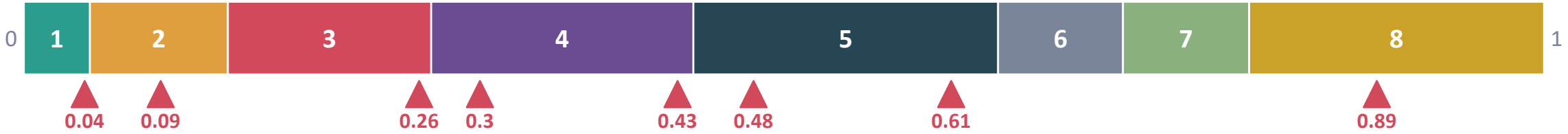
Required

(a) Find each selection probability p_i and the cumulative probability P_i.

(b) Spin the wheel once for each r and build a mating pool of 8.

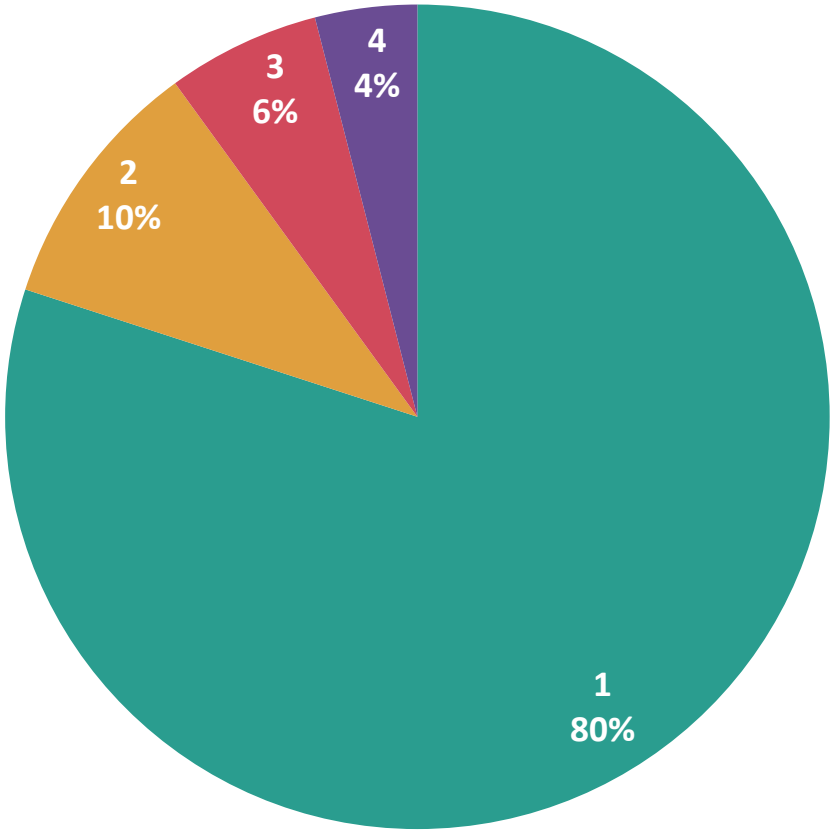
(c) Compare each individual's tally with its expected count $E_i = N \cdot p_i$.

Roulette wheel by cumulative probability



i	f_i	p_i	P_i	Interval $(P_{i-1}, P_i]$	r that land here	Tally	Expected $N \cdot p_i$
1	1.01	0.043	0.043	(0.000, 0.043]	0.04	1	0.35
2	2.11	0.091	0.134	(0.043, 0.134]	0.09	1	0.73
3	3.11	0.134	0.268	(0.134, 0.268]	0.26	1	1.07
4	4.01	0.172	0.440	(0.268, 0.440]	0.43, 0.3	2	1.38
5	4.66	0.200	0.641	(0.440, 0.641]	0.48, 0.61	2	1.60
6	1.91	0.082	0.723	(0.641, 0.723]	–	0	0.66
7	1.93	0.083	0.806	(0.723, 0.806]	–	0	0.66
8	4.51	0.194	1.000	(0.806, 1.000]	0.89	1	1.55

The roulette wheel's weakness: a super individual



i	f_i	p_i	Expected copies (N = 4)
1	0.40	80%	3.20
2	0.05	10%	0.40
3	0.03	6%	0.24
4	0.02	4%	0.16

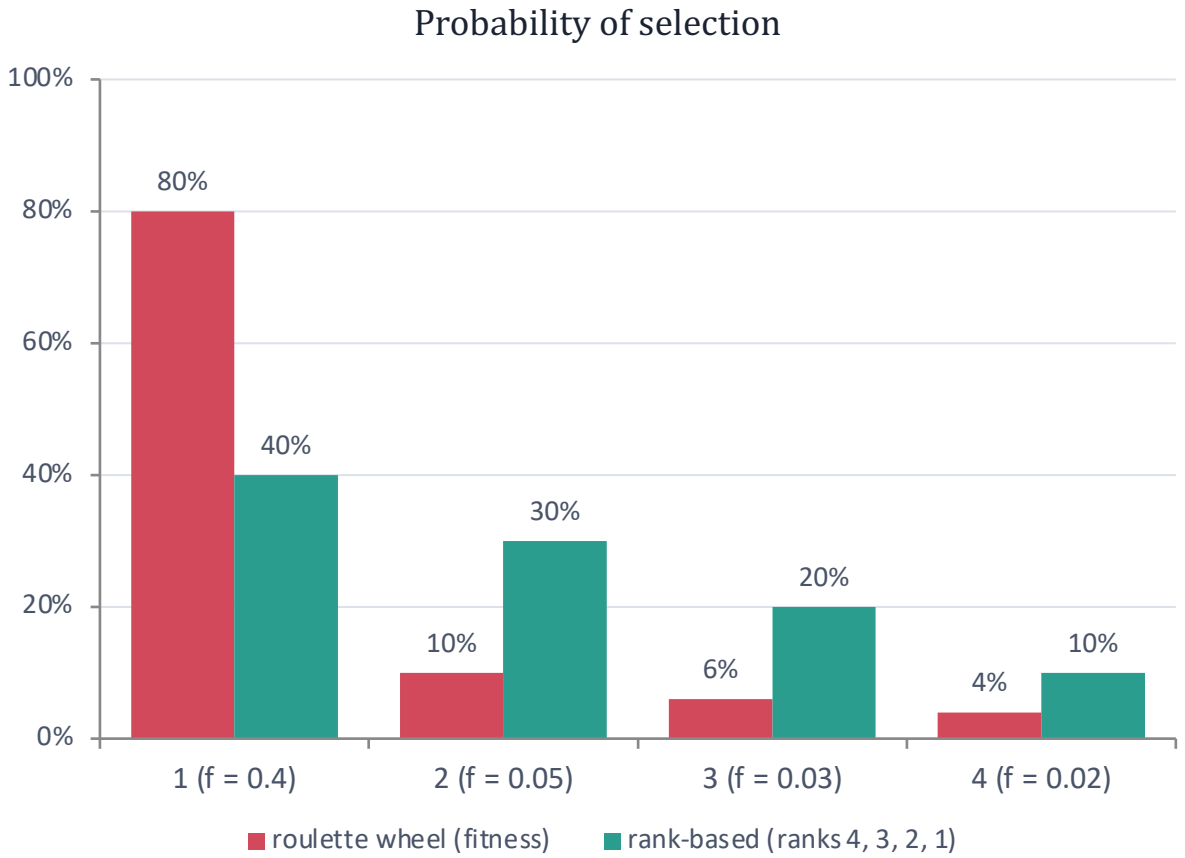
- Individual 1 is expected to fill 3 of the 4 places in the mating pool.
- The weaker individuals, and their genes, soon disappear: diversity falls.
- Few alternatives are explored, so the GA tends to converge early to a local optimum.

Rank-based selection evens out the wheel

1. Sort the individuals by fitness, in ascending order.
2. Rank them: the worst gets rank 1, the best rank N. Equal fitness values share a rank.
3. Spin a roulette wheel on the ranks instead of the fitness values.

$$p_i = \frac{r_i}{\sum_{j=1}^{N} r_j}$$

Only the order of the fitness values matters, not how far apart they are.



The best now expects 1.6 copies instead of 3.2, and the worst 0.4 instead of 0.16.

05

GA operators: selection II

Tournament · steady-state · Boltzmann · elitism · selection pressure versus diversity

Tournament selection

1. Pick N_u individuals at random ($N_u \leq N$, often 2 or 3).
2. The one with the highest fitness wins.
3. Copy the winner into the mating pool.
4. Repeat until the pool holds N_p individuals.

- One winner per tournament, so N_p tournaments fill the pool.
- A good individual can win several times.
- No sums or sorting: faster than roulette-wheel and rank selection.
- Ties are broken at random, by a coin toss.

1	2	3	4	5	6	7	8
1.0	2.1	3.1	4.0	4.6	1.9	1.8	4.5

Trial	Competitors	Their fitness	Winner
1	2 vs 4	2.1 vs 4.0	4
2	3 vs 8	3.1 vs 4.5	8
3	1 vs 3	1.0 vs 3.1	3
4	4 vs 5	4.0 vs 4.6	5
5	1 vs 6	1.0 vs 1.9	6
6	1 vs 2	1.0 vs 2.1	2
7	4 vs 2	4.0 vs 2.1	4
8	8 vs 3	4.5 vs 3.1	8

$N = 8$, $N_u = 2$, $N_p = 8$. Pool: 4, 8, 3, 5, 6, 2, 4, 8.

Steady-state and Boltzmann selection

Steady-state selection

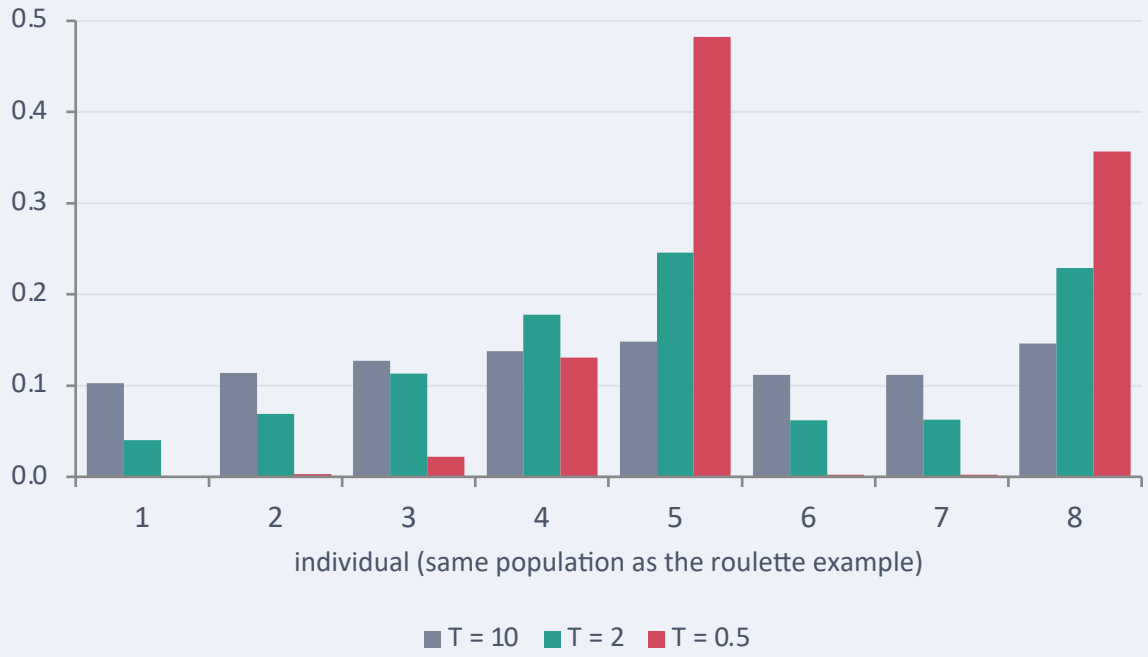
1. Choose $N_u < N$.
2. Copy the N_u fittest individuals into the mating pool.
3. Remove the N_u worst individuals from the sample.
4. Repeat until the pool is full.

Very simple and fast. Selection pressure is low and diversity shrinks slowly, so convergence is slow.

Boltzmann selection

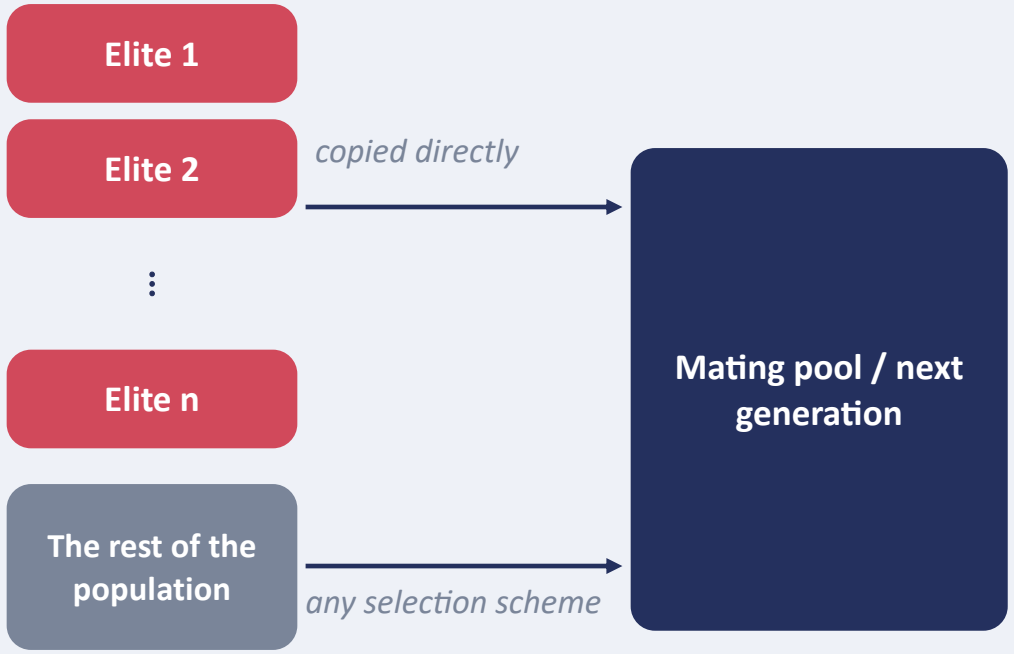
$$p_i = \frac{e^{f_i/T}}{\sum_j e^{f_j/T}}$$

T is a temperature, lowered as the run goes on, as in simulated annealing.



Elitism and the generation gap

Elitism: the best always survive



The best fitness found so far can never get worse.

Generation gap

$$G_p = \frac{p}{N}$$

p individuals out of N are replaced in each generation.

- Steady-state GA: p = 2, so $G_p \approx 0$ for a large population.
- Most other schemes: $G_p \approx 1$.

Ways to raise G_p:

1. Select by fitness, replace at random.
2. Select at random, replace by inverse fitness.
3. Select both parents, replace by fitness or inverse fitness.

Selection pressure versus population diversity



High pressure: exploitation

- The search focuses on the individuals that are good now.
- Population diversity is lost.
- Fast convergence, often premature, to a sub-optimal solution.

Low pressure: exploration

- The search may not be driven well: it can stagnate.
- Slow convergence: the GA takes a long time.
- More of the space is explored, so accuracy can improve.

Comparing the selection schemes

Scheme	Population diversity	Selection pressure	Use it when
Roulette wheel	Low: premature convergence, less accurate	High	fitness values are fairly evenly spread
Rank-based	High: explores more solutions	Low: slow convergence	a few individuals dominate the fitness
Tournament	Moderate	Very high, tunable through N_u	fitness values are very diverse; speed matters
Steady-state	Decreases gradually over generations	Too low: very slow convergence	a simple, fast scheme is acceptable
Boltzmann	High early, low late	Rises as T is lowered	you want to explore first, then exploit
Elitism	Unchanged (added to any scheme)	Slightly higher	the best solution must never be lost

Survival of the fittest, but not only the fittest: every scheme trades pressure for diversity.

06

GA operators: crossover I

Mating pool · single-, two- and multi-point · uniform · half-uniform · shuffle · matrix · three-parent

From mating pool to offspring

Before crossover

1. Random mating: draw $N_p/2$ pairs from the mating pool at random.
2. For each pair, toss a coin that lands heads with the crossover probability pc: heads, the pair crosses; tails, it is copied unchanged (a Monte Carlo step). Usually pc is close to 1.
3. Draw the crossover point(s) k as random integers between 1 and L − 1, for chromosome length L.
4. Exchange genes around k: two parents give two offspring.

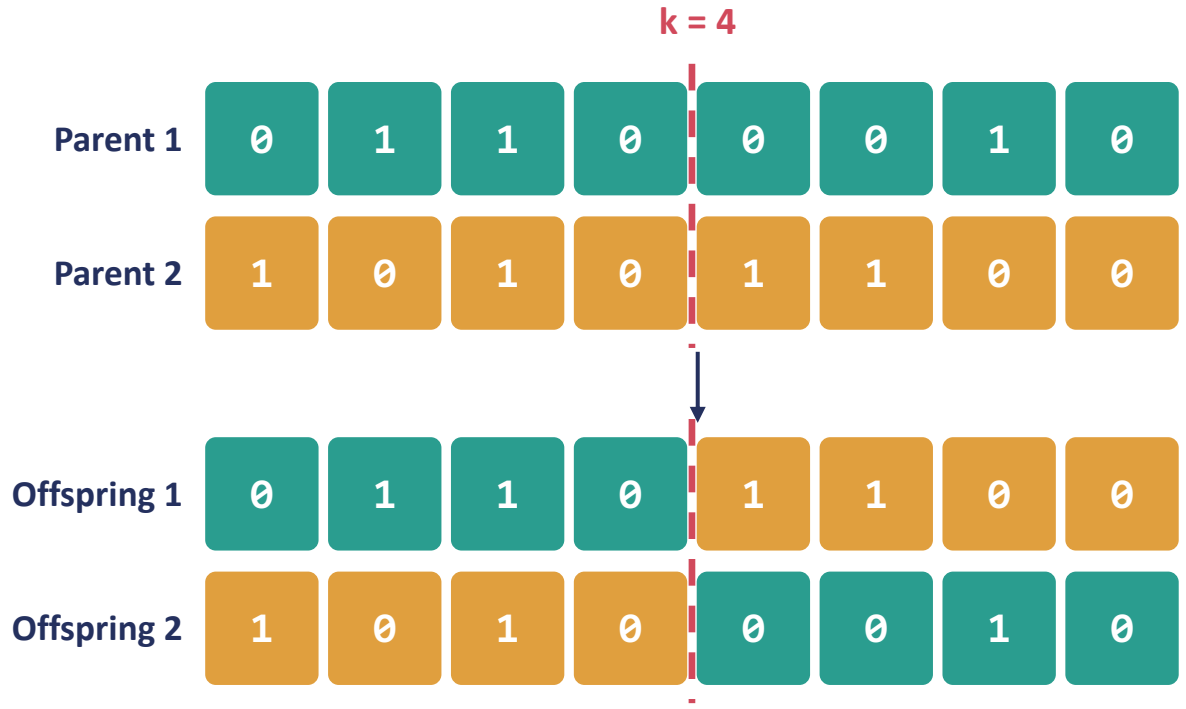
Encoding	Crossover	Mutation
Binary	single-, two-, multi-point, uniform, half-uniform, shuffle, matrix, three-parent	flipping, interchanging, reversing
Real-coded	linear, blend (BLX-α), simulated binary (SBX)	random, polynomial
Order	single- and two-point order, PPX, position-based, edge recombination	swap, inversion

Reproduction (crossover, mutation, inversion) depends on the encoding: each encoding has its own operators. This section: binary. Section 07: real-coded and order.

Single-point crossover

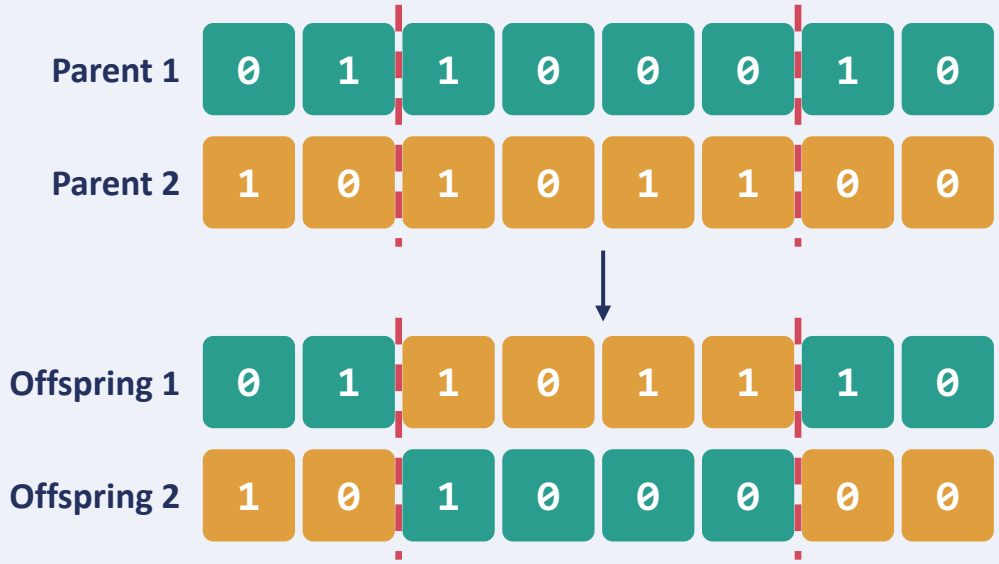
1. Draw one crossover point k between 1 and L − 1.
2. Cut both parents at k.
3. Swap everything to the right of k.
4. The two new strings are the offspring.

Teal genes come from parent 1, amber genes from parent 2. The dashed line is the crossover point.



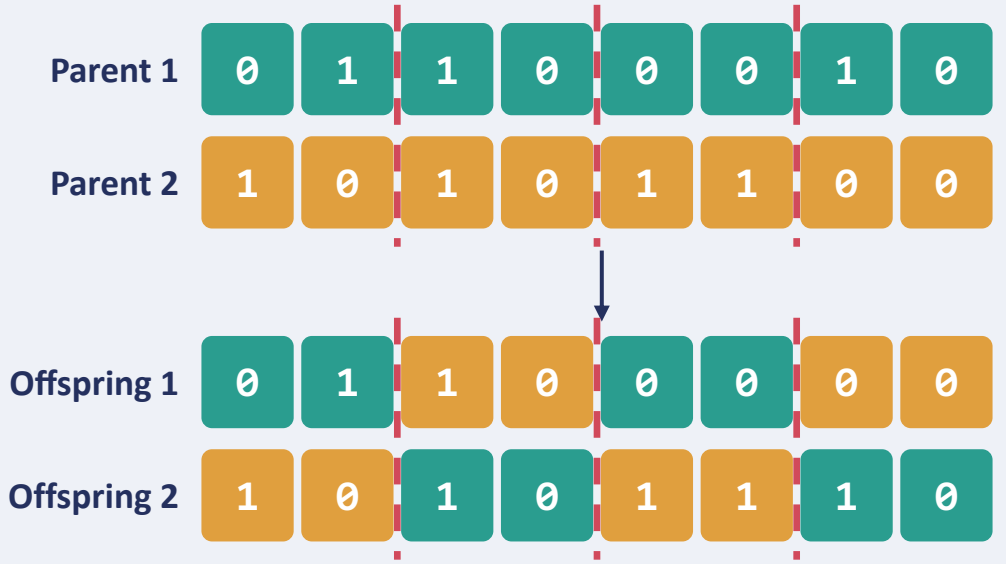
Two-point and multi-point crossover

Two-point: swap the middle



$k_1 = 2$, $k_2 = 6$: the genes between the two points are exchanged. Swapping the outer parts instead gives the same pair of offspring.

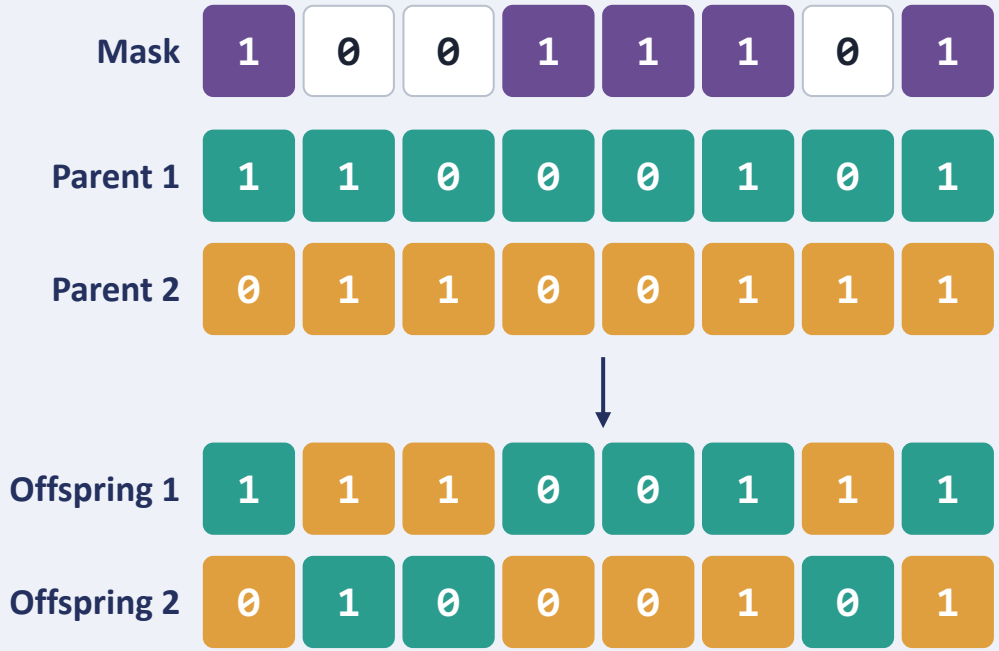
Multi-point: swap alternate segments



k = 2, 4, 6: segments 2 and 4 are exchanged. Uniform crossover takes this to the limit.

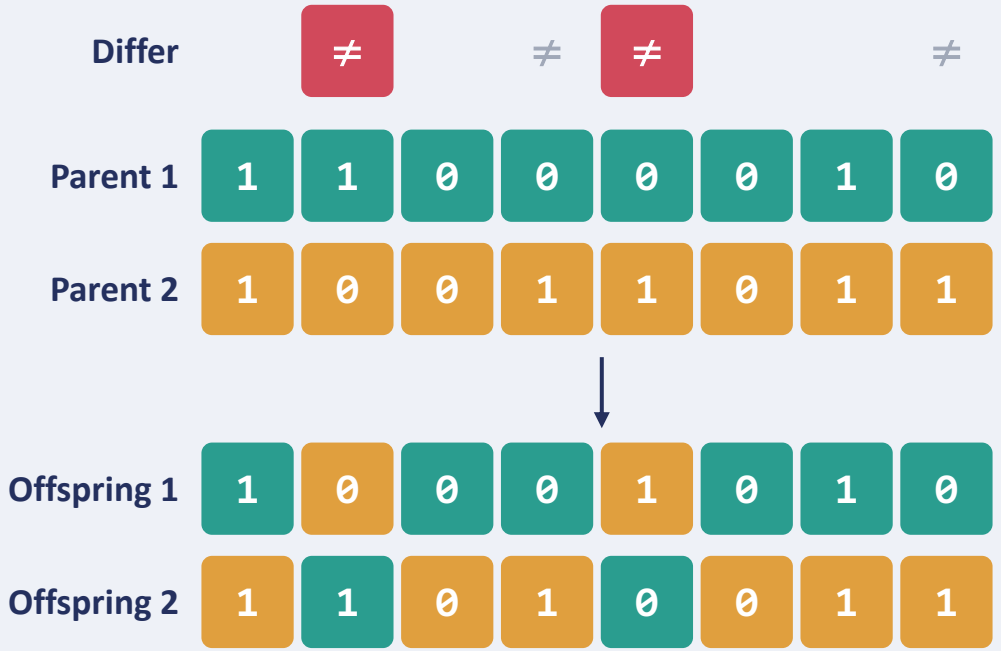
Uniform and half-uniform crossover

Uniform crossover (UX)



A random mask (one coin toss per gene): 1 copies the gene from parent 1, 0 from parent 2. Offspring 2 uses the opposite rule.

Half-uniform crossover (HUX)



The parents differ in h = 4 bits (Hamming distance). Exactly h/2 = 2 of them, chosen at random (red), are swapped.

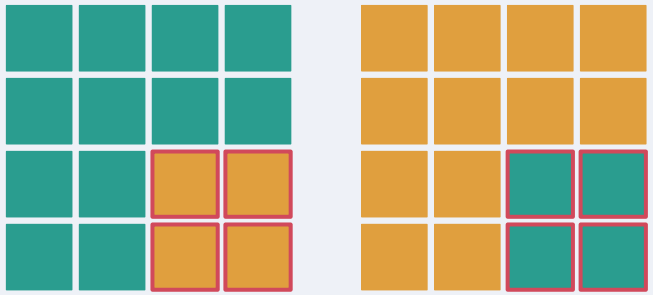
Shuffle, matrix and three-parent crossover

Shuffle crossover

- Shuffle the gene positions of both parents with the same random permutation.
- Apply single-point crossover.
- Un-shuffle the offspring.
- Genes far apart now stay together as often as neighbours: no positional bias.

Matrix crossover

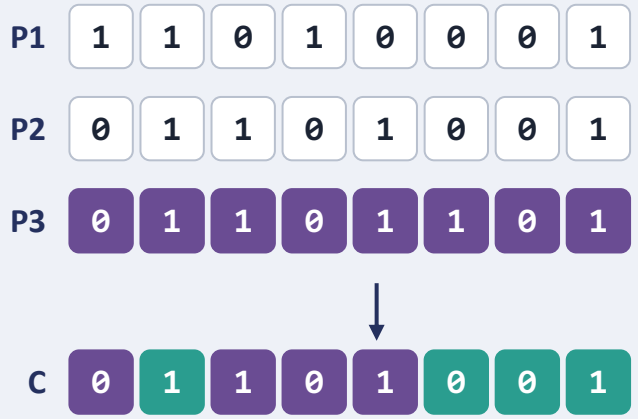
Each chromosome is written as a 2-D matrix; both matrices are cut into the same zones, and zones are swapped.



offspring 1 offspring 2

Here one zone (lower right) is exchanged.

Three-parent crossover

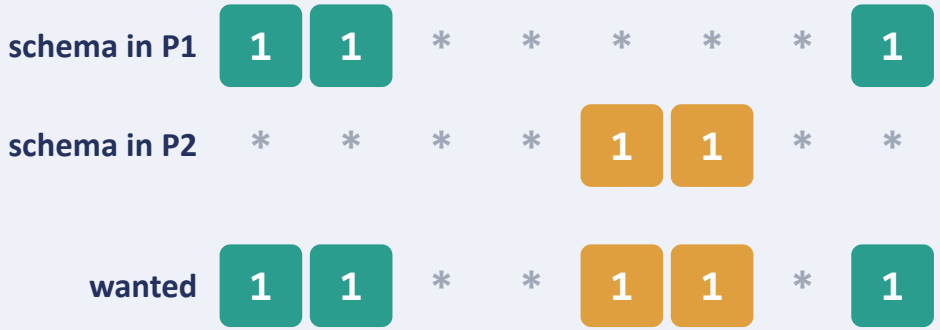


Where P1 and P2 agree, keep their bit (teal); where they differ, take P3's bit (purple).

P3 acts like a crossover mask.

Blind spots of binary crossover

Non-uniform variation



The wanted string needs both ends of P1 and the middle of P2. No single cut can produce it; two cuts (k = 4 and 6) can.

*Schema: a template; * matches 0 or 1. Good schemas are the building blocks a GA combines.*

Positional bias

Whether two genes are inherited together depends on how far apart they sit on the string.

End-point bias

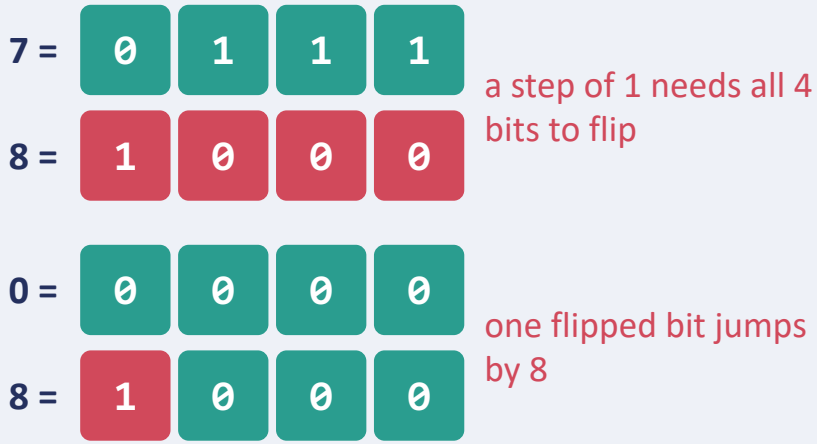
Single-point crossover always exchanges a segment containing an end of the string.

Remedies

Two- and multi-point crossover reduce both biases. UX and HUX remove them but disrupt the parents heavily.

The Hamming cliff, and Gray code as a fix

Hamming cliff



$$g = b \oplus (b \gg 1)$$

Gray code: each bit is XOR-ed with the bit to its left, so neighbouring integers differ in exactly one bit.

x	Binary	Bits changed	Gray	Bits changed
0	0000	–	0000	–
1	0001	1	0001	1
2	0010	2	0011	1
3	0011	1	0010	1
4	0100	3	0110	1
5	0101	1	0111	1
6	0110	2	0101	1
7	0111	1	0100	1
8	1000	4	1100	1

07

GA operators: crossover II

Real-coded: linear, blend, simulated binary · order: single- and two-point, PPX, position-based, edge recombination

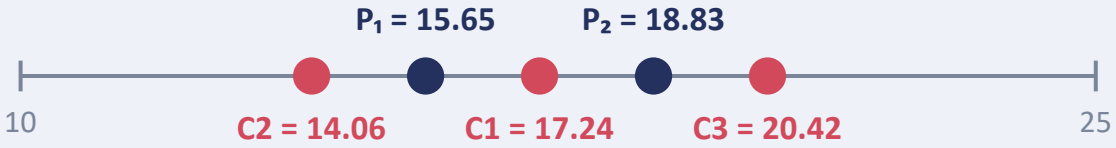
Linear crossover

$$C_i = \alpha_i P_1 + \beta_i P_2$$

Each child is a linear combination of the parents; one pair of parents can give many children.

Child	α_i	β_i	C_i
C1	0.5	0.5	17.24
C2	1.5	-0.5	14.06
C3	-0.5	1.5	20.42

α + β = 1 keeps the child on the line through the parents; α = β = 0.5 is the midpoint.



Advantages

- Simple and fast to compute.
- Many children from two parents.
- Wide control over the variation.

Limitations

- α_i and β_i must be chosen, which is hard for a beginner.
- Poor choices can trap the search at a local optimum.

Blend crossover (BLX-α)

With $P_1 < P_2$, children are drawn from the range

$$[P_1 - \alpha(P_2 - P_1),\ \ P_2 + \alpha(P_2 - P_1)]$$

using a random r in (0, 1):

$$\gamma = (1 + 2\alpha)\, r - \alpha$$

$$C_1 = (1 - \gamma)P_1 + \gamma P_2$$

$$C_2 = (1 - \gamma)P_2 + \gamma P_1$$

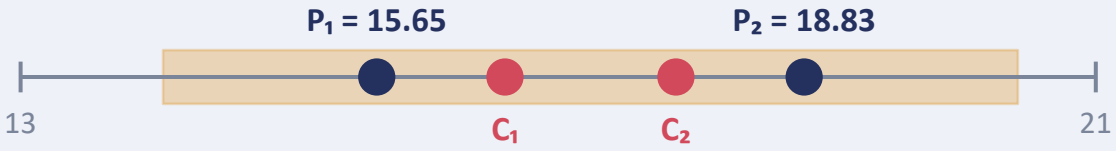
α sets how far children may land outside the parents; keep it small enough that children stay inside the variable's domain.

Worked: α = 0.5, r = 0.4

$$\gamma = (1 + 2{\times}0.5){\times}0.4 - 0.5 = 0.3$$

$$C_1 = 0.7{\times}15.65 + 0.3{\times}18.83 = 16.60$$

$$C_2 = 0.7{\times}18.83 + 0.3{\times}15.65 = 17.88$$



Shaded: the BLX-0.5 range [14.06, 20.42].

Simulated binary crossover (SBX)

$$\beta = \left|\frac{C_2 - C_1}{P_2 - P_1}\right|$$

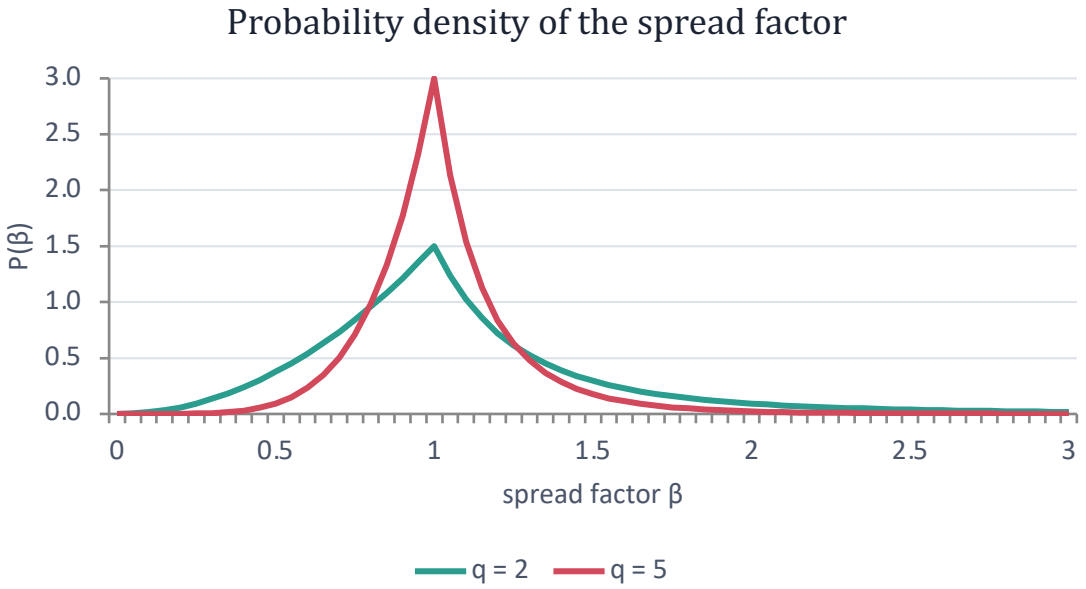
β < 1 contracting
β > 1 expanding
β = 1 stationary

$$P(\beta) = \begin{cases} 0.5(q+1)\,\beta^{q} & \beta \le 1 \\ 0.5(q+1)\,\dfrac{1}{\beta^{q+2}} & \beta > 1 \end{cases}$$

$$\beta' = \begin{cases} (2r)^{\frac{1}{q+1}} & r \le 0.5 \\ \left(\dfrac{1}{2(1-r)}\right)^{\frac{1}{q+1}} & r > 0.5 \end{cases}$$

$$C_{1,2} = 0.5[(P_1 + P_2) \mp \beta'|P_2 - P_1|]$$

q, the distribution index, is chosen by the user: a larger q keeps children closer to their parents.

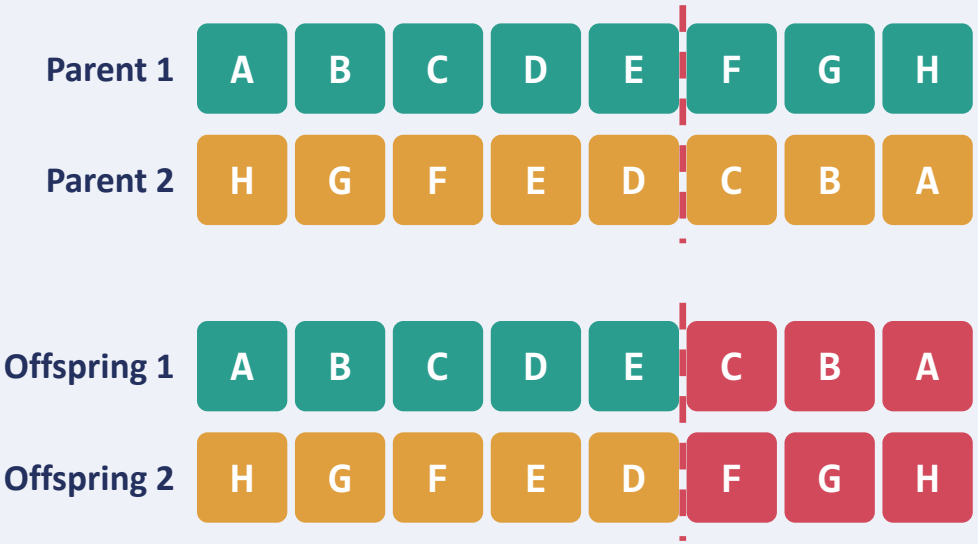


$$q = 2,\ r = 0.6: \quad \beta' = \left(\frac{1}{2\times 0.4}\right)^{1/3} = 1.0772$$

C_1 = 15.53, C_2 = 18.95: an expanding crossover around P_1 = 15.65, P_2 = 18.83

Order chromosomes need their own crossover

Binary single-point crossover on two tours



Red cities appear twice, and F, G, H (or C, B, A) are missing: neither offspring is a valid tour. Arithmetic real-coded crossover makes no sense on an order either.

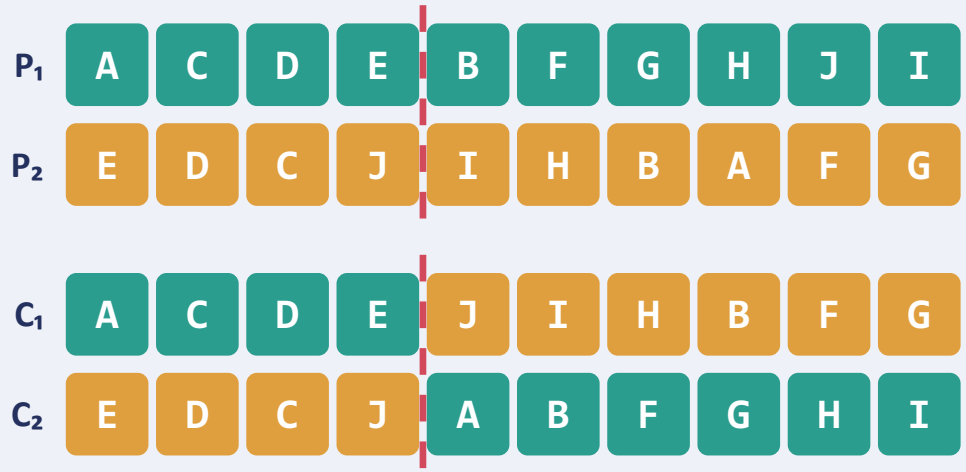
Order crossover operators

1. Single-point order crossover
2. Two-point order crossover
3. Partially mapped crossover (PMX)
4. Position-based crossover
5. Precedence-preservation crossover (PPX)
6. Edge recombination crossover

Notation: chromosome length L; parents P_1, P_2 from the mating pool; offspring C_1, C_2, initially empty.

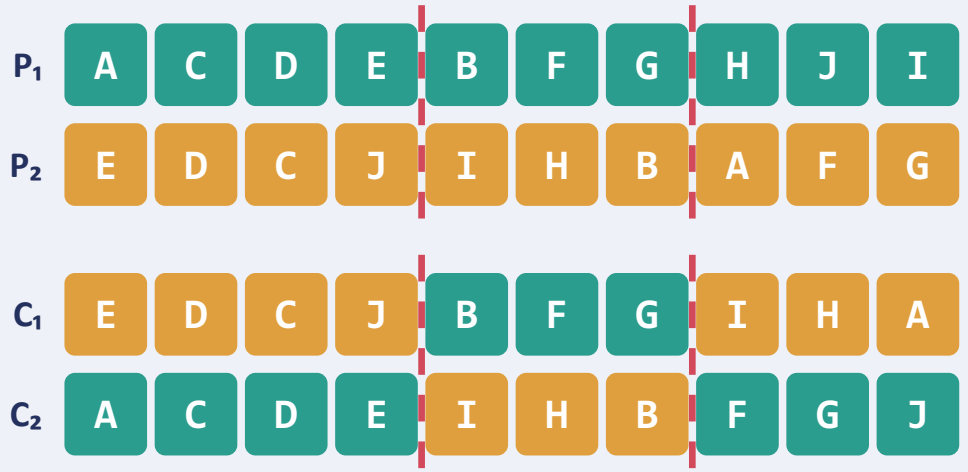
Single-point and two-point order crossover

Single-point (K = 4)



Copy P_1 left of K into C_1; fill the rest with P_2's genes in P_2's order, skipping genes already in C_1. Swap roles for C_2.

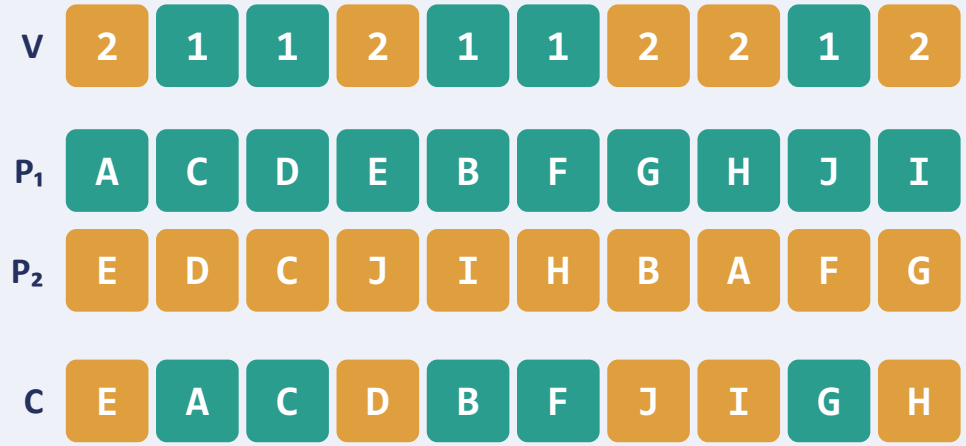
Two-point ($K_1 = 4$, $K_2 = 7$)



Copy P_1's middle segment into the same place in C_1; fill the free places, left to right, with P_2's genes in order, skipping duplicates.

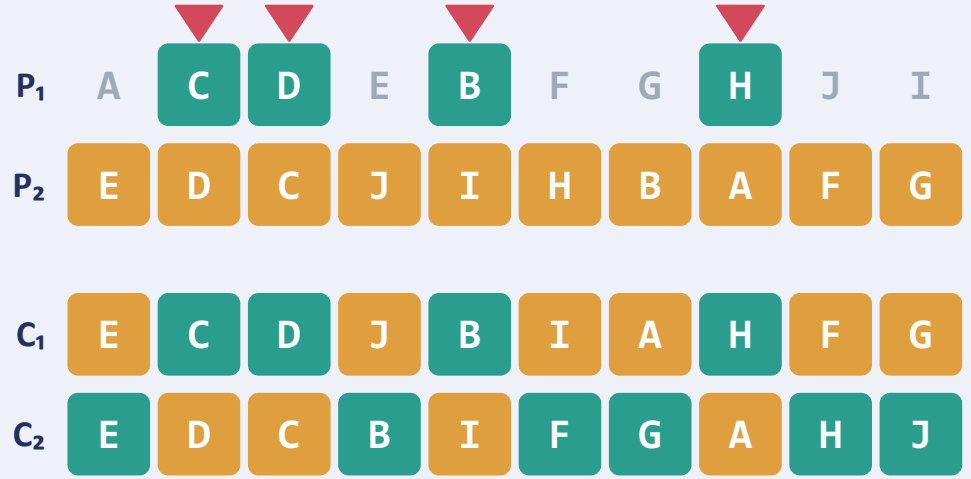
Precedence-preservation and position-based crossover

Precedence preservation (PPX)



Scan V left to right. A 1 takes the leftmost gene of P_1 not yet used, a 2 that of P_2; the gene is then deleted from both parents.

Position-based (positions 2, 3, 5, 8)



Keep P_1's genes at a few chosen positions; fill the other places with P_2's genes in P_2's order, skipping duplicates. Swap roles for C_2.

Edge recombination crossover

P_1	1	2	4	6	8	7	5	3
P_2	4	3	5	7	8	6	2	1

City	Neighbours in P_1 or P_2 (edge table)
1	2, 3, 4
2	1, 4, 6
3	1, 4, 5
4	1, 2, 3, 6
5	3, 7
6	2, 4, 8
7	5, 8
8	6, 7

1. Start C with P_1's first city X.
2. Delete X from every neighbour list.
3. Next city: X's neighbour with the fewest neighbours left (ties: here the lowest number).
4. If X has no neighbours left, pick any unvisited city. Repeat.

Current	Candidates (links left)	Next
1	2(2), 3(2), 4(3)	2
2	4(2), 6(2)	4
4	3(1), 6(1)	3
3	5(1)	5
5	7(1)	7
7	8(1)	8
8	6(0)	6

child C	1	2	4	3	5	7	8	6

08

GA operators: mutation

Binary and real-coded mutation · order mutation and inversion · termination · fitness scaling

Why mutate?

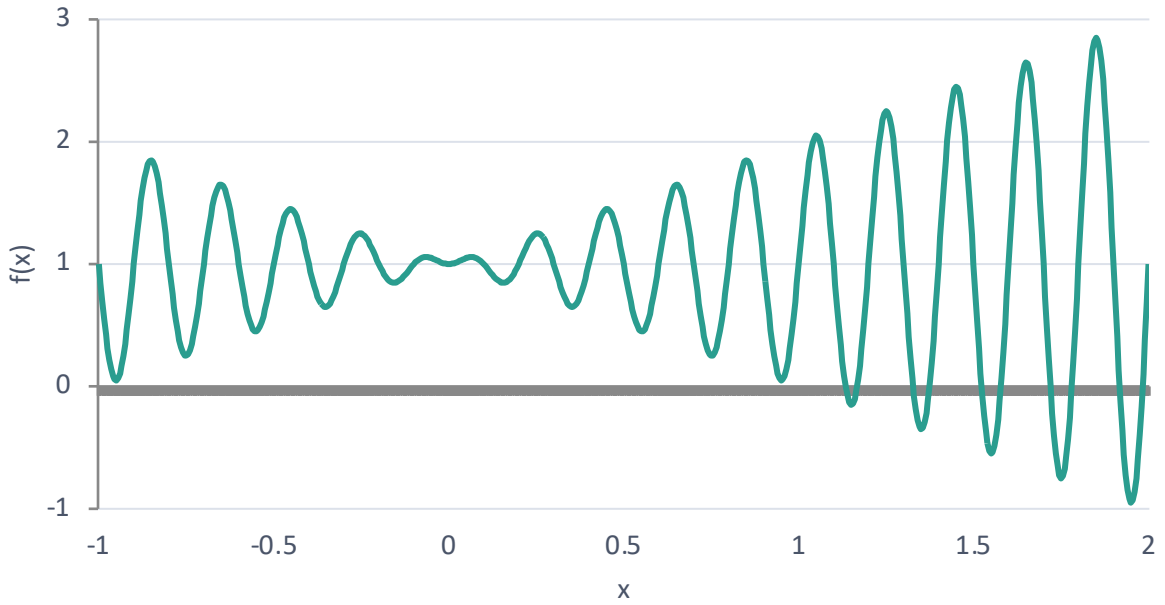
- Mutation keeps genetic diversity from one generation to the next, like biological mutation.
- It makes a small, local change to an offspring.
- Crossover only recombines genes the population already has; mutation can create genes nobody has, and so escape a local optimum.

Binary GA: mutation probability per bit

$$\frac{0.1}{L} \leq \mu_p \leq \frac{1}{L}$$

Kept low to avoid large jumps: about $L \cdot \mu_p \leq 1$ bit changes per chromosome.

f(x) = x sin(10πx) + 1 on [−1, 2]: many local optima



Encoding	Mutation operators
Binary	flipping, interchanging, reversing
Real-coded	random, polynomial
Order	swap, inversion

Binary mutation: flipping, interchanging, reversing

Flipping

offspring

1	0	1	1	0	0	1	0

mutation chromosome (1 = flip)

1	0	0	0	1	0	0	1

mutated

0	0	1	1	1	0	1	1

Each bit gets a 1 in the mutation chromosome with probability μ_p; those bits flip.

Interchanging

child: positions 2 and 6 chosen

1	0	1	1	0	1	0	0

mutated

1	1	1	1	0	0	0	0

Two random positions swap their bits. If both bits are equal nothing changes.

Reversing

child: position 6 chosen

0	1	1	0	0	1	0	1

mutated

0	1	1	0	0	1	1	1

A random position is chosen and the bit next to it is reversed (0 ↔ 1).

Real-coded mutation: random and polynomial

Random mutation

$$P_{mutated} = P_{original} + (r - 0.5)\,\Delta$$

$$15.6 + (0.7 - 0.5)\times 2.5 = 16.1$$

r is random in (0, 1); Δ is the largest perturbation the user allows.

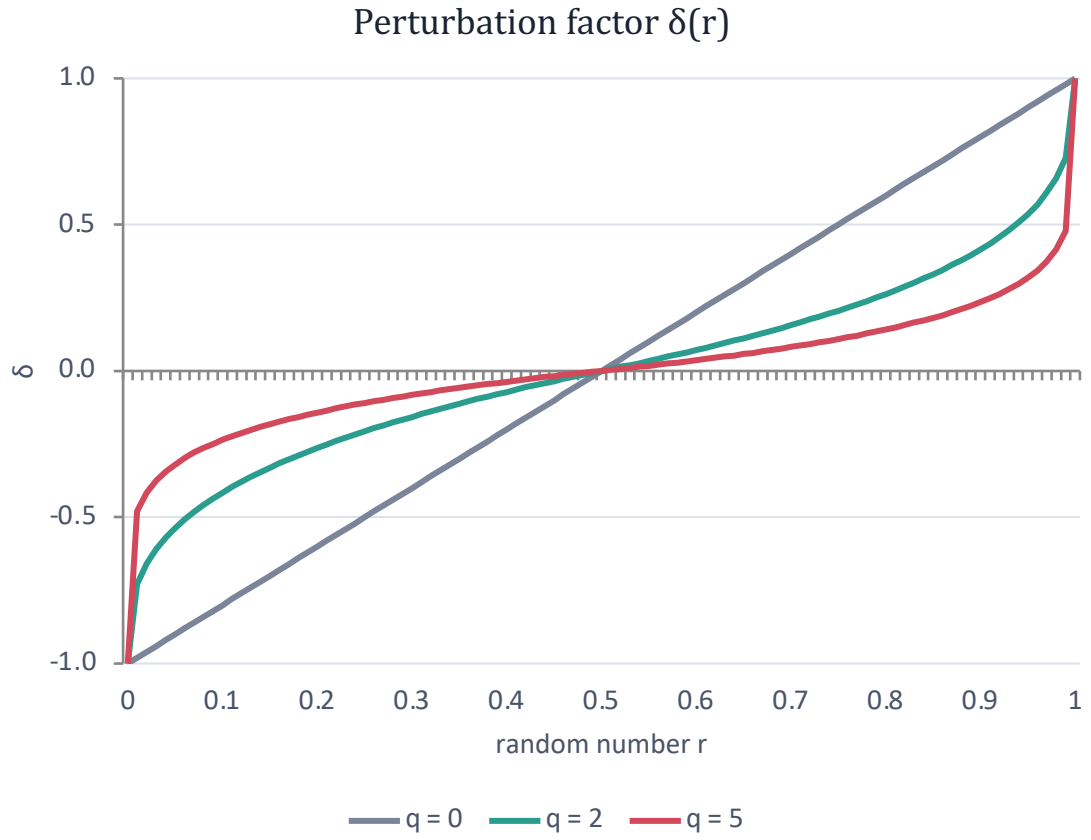
Polynomial mutation

$$\delta = \begin{cases} (2r)^{\frac{1}{q+1}} - 1 & r < 0.5 \\ 1 - [2(1-r)]^{\frac{1}{q+1}} & r \geq 0.5 \end{cases} \qquad P' = P + \delta\,\Delta$$

$$\delta = 1 - (2\times 0.3)^{1/3} = 0.1566$$

$$P' = 15.6 + 0.1566\times 1.2 = 15.788$$

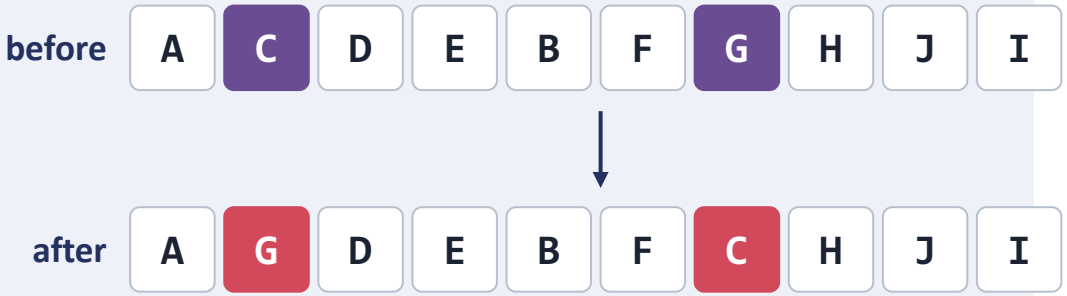
Example: P = 15.6, r = 0.7, q = 2 (distribution index), Δ = 1.2.



q = 0 spreads δ evenly over (−1, 1); a larger q makes small changes far more likely.

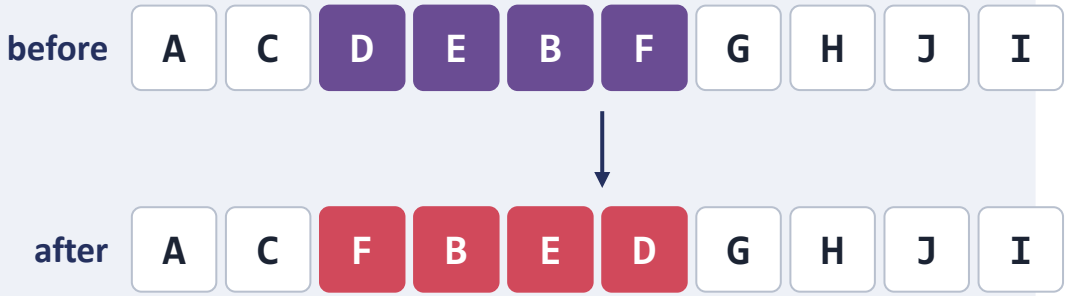
Order-GA mutation and inversion

Swap mutation



Two random positions exchange their genes. The chromosome stays a valid permutation.

Inversion



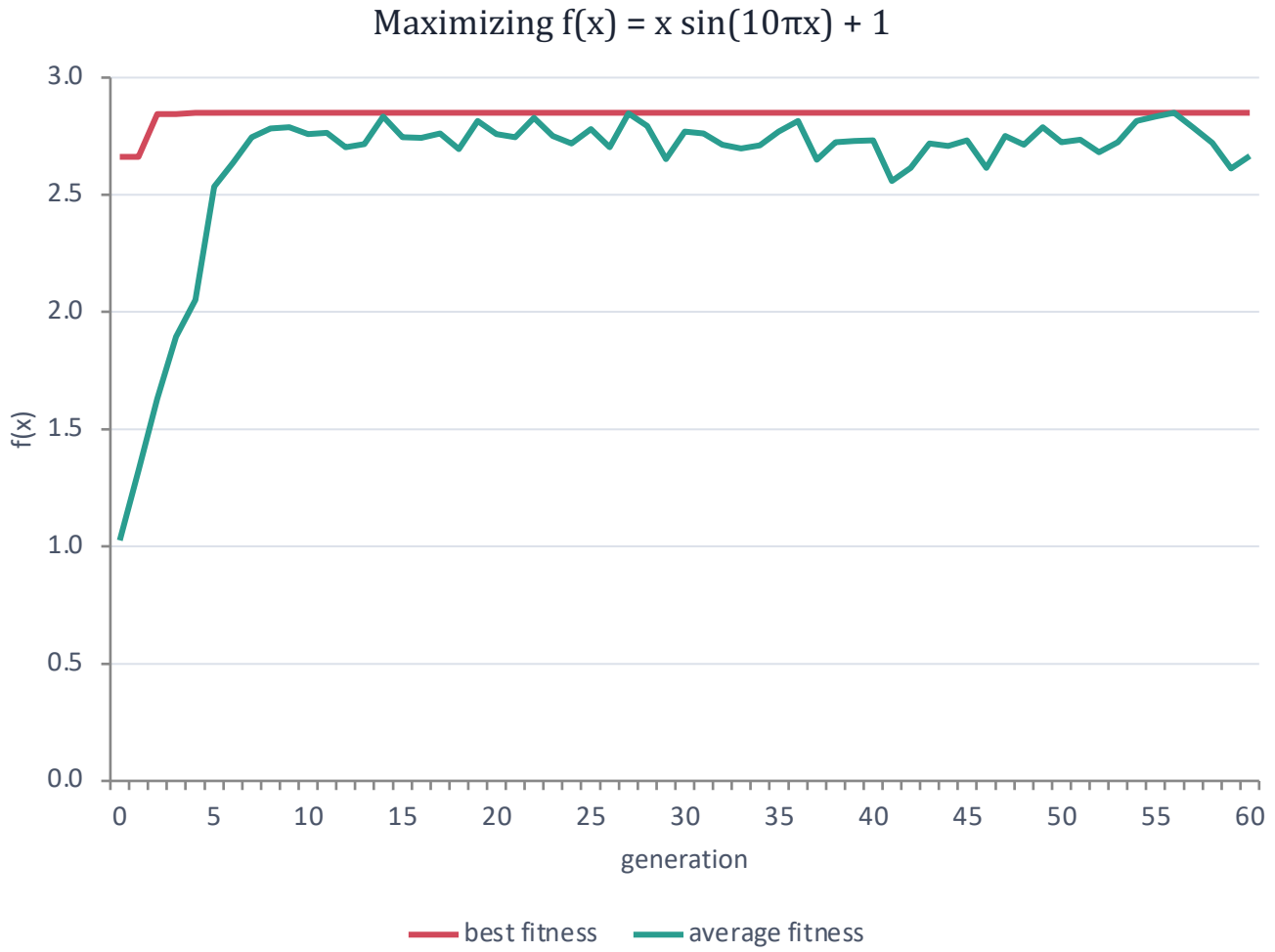
The genes between two random points are written in reverse order. NPTEL lists inversion as its own operator, used to move from one optimum to another. Binary strings can be inverted the same way.

One run of a simple GA

Settings

Operator	Choice
Encoding	22 bits on [−1, 2]
Population	N = 50, 60 generations
Selection	tournament, $N_u = 2$
Crossover	single-point, pc = 0.8
Mutation	flipping, $\mu_p = 0.01$
Elitism	the best string survives

Best: x = 1.8506, f(x) = 2.8503, the global maximum. The best value stops improving after generation 25.



When should a GA stop?

1 Goal reached

A solution meets the objective criteria.

2 Generation limit

A fixed number of generations has run.

3 Budget spent

The allocated computation time or evaluations are used up.

4 Plateau

The best fitness no longer improves over successive generations.

5 Manual inspection

A person decides the result is good enough.

6 A combination

e.g. a plateau or a generation limit, whichever comes first.

In the run on the previous slide the best fitness stopped improving after generation 25. A plateau rule that waits 15 generations would have stopped at generation 40, instead of running all 60.

Fitness scaling: values too far apart or too close

Too far apart

A few super individuals get many copies and the rest none. The population fills with similar chromosomes: premature convergence.

Too close together

Selection takes about one copy of everyone, so small differences cannot guide the search: stagnation and slow finishing.

Linear scaling

$$f' = af + b$$

$$a = \frac{(C-1)\bar{f}}{f_{max} - \bar{f}}$$

$$b = \frac{\bar{f}\,(f_{max} - C\bar{f})}{f_{max} - \bar{f}}$$

keeps the average and sets f'max = C·f̄ (C ≈ 1.2 to 2)

Sigma scaling

$$f' = \max\big(0,\ f - (\bar{f} - c\,\sigma)\big)$$

$$\sigma = \text{standard deviation of } f$$

c is usually 1 to 3; individuals below f̄ – cσ get 0

Power-law scaling

$$f' = f^{\,k}$$

k is a problem-dependent constant

Example 4: Linear fitness scaling

Given

Four individuals have raw fitness 0.4, 0.05, 0.03, 0.02.

Use linear scaling $f' = a \cdot f + b$ with scaling factor $C = 2$.

Required

(a) Find the roulette-wheel probabilities of the raw fitness values.

(b) Find a and b so that the average is kept and $f'_{max} = C \cdot \bar{f}$.

(c) Scale the fitness values and find the new probabilities.

Linear scaling tames a super individual

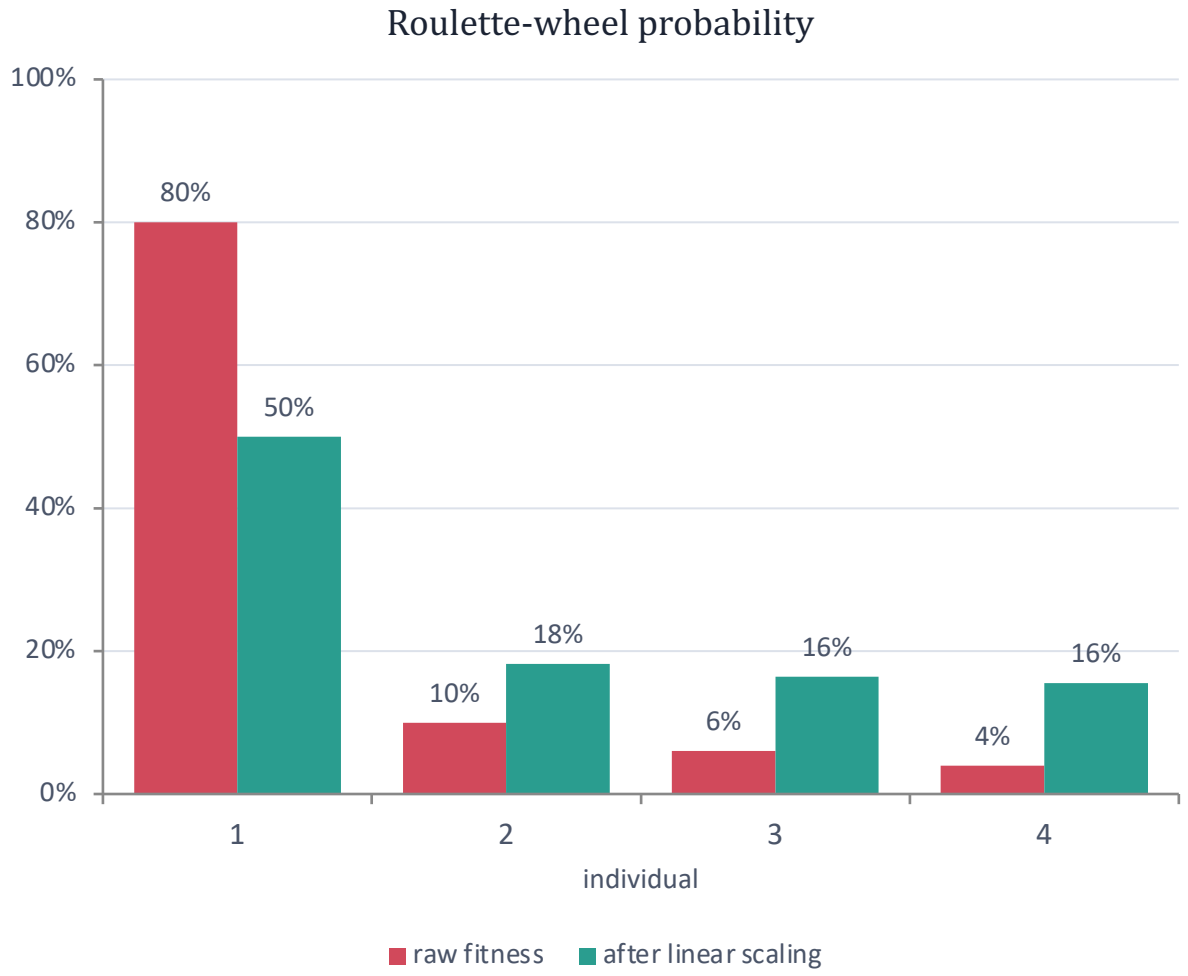
$$\bar{f} = 0.125, \quad f_{max} = 0.40, \quad C = 2$$

$$a = \frac{(2-1)\times 0.125}{0.40 - 0.125} = 0.4545$$

$$b = \frac{0.125\times(0.40 - 2\times 0.125)}{0.40 - 0.125} = 0.0682$$

i	f	f′ = af + b	p before	p after
1	0.40	0.2500	80%	50%
2	0.05	0.0909	10%	18.2%
3	0.03	0.0818	6%	16.4%
4	0.02	0.0773	4%	15.5%

The average is unchanged; the best now gets exactly C = 2 times the average.



Summary: the GA toolbox

	Binary-coded	Real-coded	Order (permutation)
Encoding	bit strings; $\varepsilon = (\text{range})/(2^n - 1)$	real numbers, no decoding	a permutation, e.g. a tour
Crossover	single-, two-, multi-point, UX, HUX, three-parent	linear, blend (BLX-α), SBX	order (1- and 2-point), PPX, position-based, edge recombination
Mutation	flipping, interchanging, reversing	random, polynomial	swap, inversion

Selection, for every encoding: canonical, roulette wheel, rank, tournament, steady-state, Boltzmann, plus elitism

Stopping: goal reached, generation limit, budget, plateau, inspection, or a combination

- A GA needs only an encoding and a fitness function: no derivatives, no continuity.
- Selection exploits good solutions; crossover and mutation explore. Balancing pressure and diversity is the art.
- The encoding decides which crossover and mutation are valid.

09

Questions

Review questions and practice problems

Review questions

1. Which two things must you define before a GA can run, and which two are not needed?

2. Is a GA guaranteed to find the global optimum? Will it return every optimum?

3. When does roulette-wheel selection go wrong, and how does rank-based selection help?

4. Why can’t binary single-point crossover be used on a TSP chromosome?

5. What is the Hamming cliff, and how does Gray code avoid it?

6. What happens to diversity and to the speed of convergence when selection pressure is high?

Practice problems

7 A variable lies in [0, 10] and needs an accuracy of 0.01. How many bits are needed, and what accuracy do they give?

8 Decode the 5-bit string 11010 for a variable in [−1, 2].

9 Fitness 12, 6, 30, 2. Find p_i and P_i, and select four individuals for r = 0.15, 0.55, 0.92, 0.40.

10 Same population: find the rank-based selection probabilities. Compare with problem 9.

11 Two-point crossover with $k_1 = 3$, $k_2 = 7$ on 1100110011 and 0101010101. Find both offspring.

12 Single-point order crossover, K = 3, on P_1 = 1 2 3 4 5 6 7 8 and P_2 = 8 6 4 2 7 5 3 1.

13 Blend crossover: $P_1 = 2.0$, $P_2 = 5.0$, $\alpha = 0.5$, r = 0.8. Find γ, C_1 and C_2, and the BLX range.

14 Polynomial mutation: P = 4.0, r = 0.2, q = 1, $\Delta = 0.5$. Find δ and the mutated value.

References

- D. Samanta, Introduction to Soft Computing, NPTEL, IIT Kharagpur, lectures 14–23 (concept of GA to mutation), with course slides and practice sheets GA-1 and GA-2.
- J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.
- D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.
- D. E. Goldberg and K. Deb, “A comparative analysis of selection schemes used in genetic algorithms,” Foundations of Genetic Algorithms 1, pp. 69–93, 1991.
- L. J. Eshelman and J. D. Schaffer, “Real-coded genetic algorithms and interval-schemata,” Foundations of Genetic Algorithms 2, pp. 187–202, 1993.
- K. Deb and R. B. Agrawal, “Simulated binary crossover for continuous search space,” Complex Systems, vol. 9, no. 2, pp. 115–148, 1995.
- Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed., Springer, 1996.