

Lecture 6

❖ Evolutionary Algorithms

- ✓ The general scheme of evolutionary algorithms
 - ✓ Components of evolutionary algorithms
 - ✓ Binary Genetic Algorithm

CENG 632- Computational Intelligence, 2024-2025, Spring
Assist. Prof. Dr. Osman GÖKALP



ORIGINS

Part I

Evolutionary computing: The origins

Darwinian Evolution (1/3): Survival of the fittest

- All environments have **finite resources**
(i.e., can only support a limited number of individuals)
- Life forms have basic instinct/ lifecycles geared **towards reproduction**
- Therefore some kind of **selection** is inevitable
- Those individuals that **compete** for the resources most effectively have increased chance of reproduction

Darwinian Evolution (2/3):

Diversity drives change

- Phenotypic traits:
 - **Behaviour / physical** differences that affect response to environment
 - Partly determined by **inheritance**, partly by factors during development
 - Unique to each individual, partly as a result of **random changes**
- If phenotypic traits:
 - Lead to **higher chances** of **reproduction**
 - Can be inherited

then they will tend to increase in subsequent generations, leading to new combinations of traits ...

Darwinian Evolution (3/3): Summary

- Population consists of diverse set of individuals
- Combinations of traits that are better adapted tend to increase representation in population

Individuals are “units of selection”

- Variations occur through random changes yielding constant source of diversity, coupled with selection means that:

Population is the “unit of evolution”

- Note **the absence of “guiding force”**

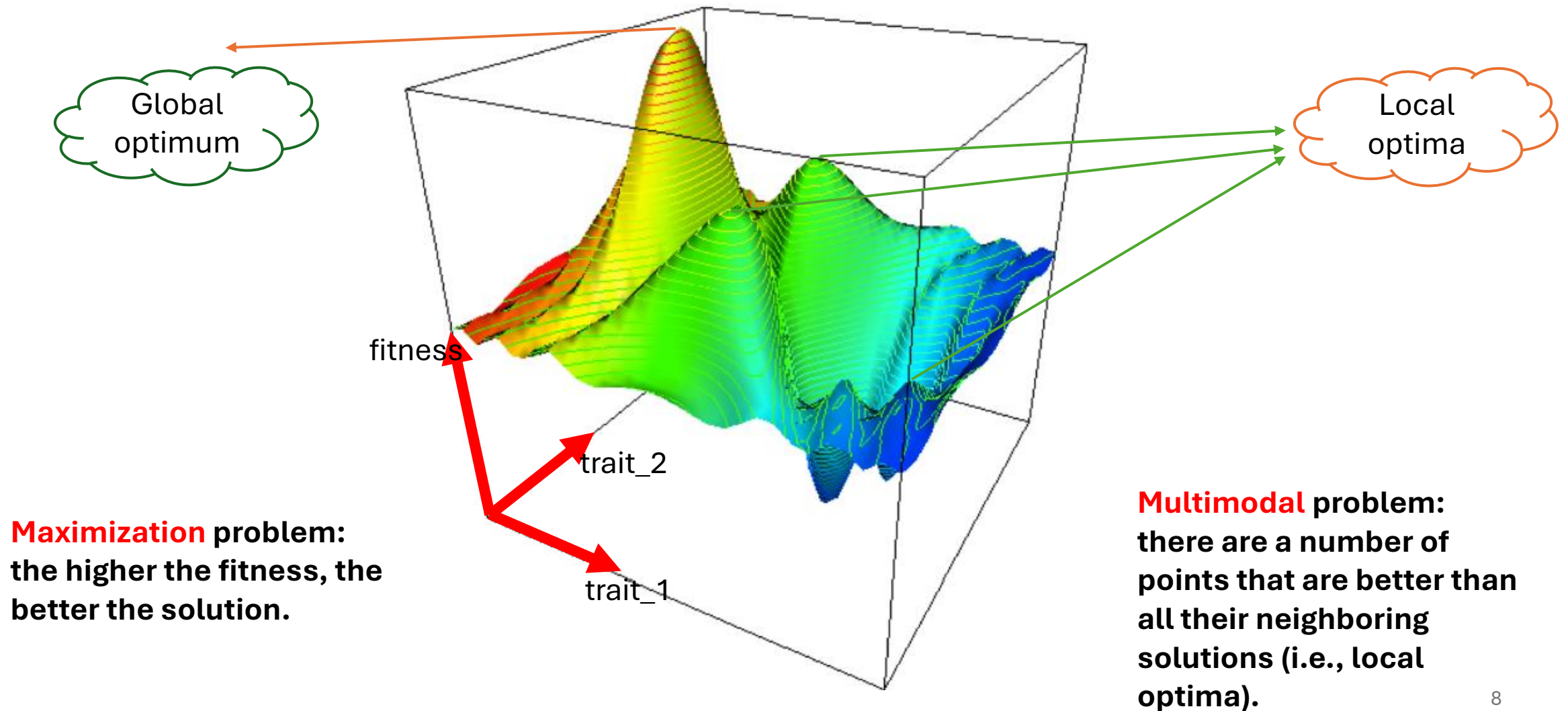
Analogies between “**biological evolution**” and “**problem solving**”

Evolution	Problem Solving
Environment	Problem
Individual	Candidate solution
Fitness	Solution Quality

Adaptive landscape metaphor (Wright, 1932)

- Can envisage population with **n traits** as existing in a **$n+1$ -dimensional space** (landscape) with height corresponding to **fitness**
- Each different individual (phenotype) represents a **single point** on the landscape
- **Population** is therefore a “cloud” of points, moving on the landscape over time as it evolves – adaptation

Adaptive landscape metaphor (Wright, 1932)



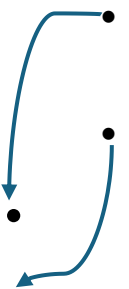
Genetics: Mutation

- **Occasionally** some of the genetic material **changes** very **slightly** during this process (**replication error**)
- This means that the child might have genetic material information **not inherited** from either parent
- This can be
 - catastrophic: offspring is not viable (most likely)
 - neutral: new feature does not influence fitness
 - advantageous: strong new feature occurs
- **Redundancy** in the genetic code forms a good way of **error checking**

Genetics: Recombination

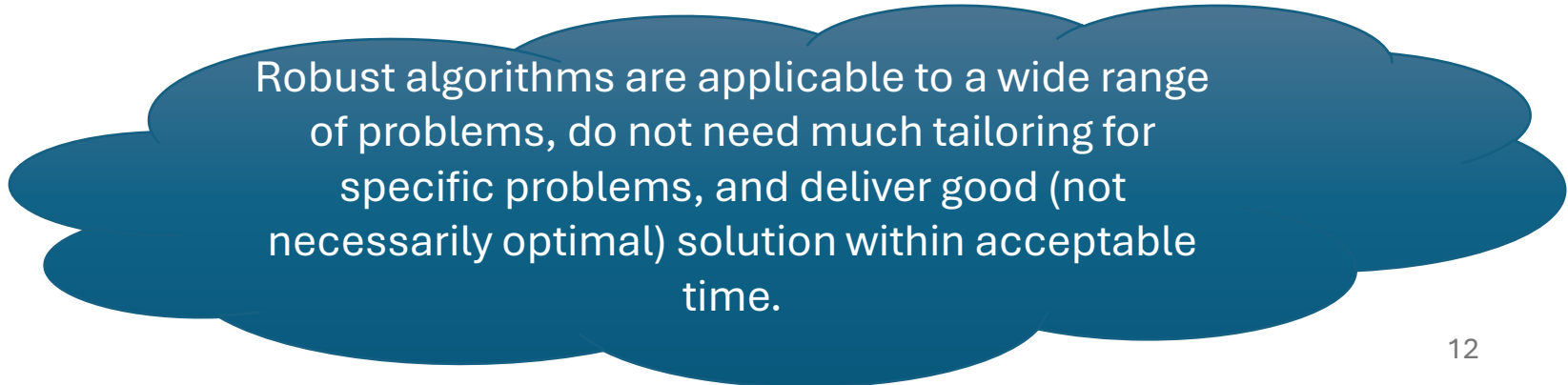
- During reproduction, **genetic material** from **parents** is **mixed** together to form a new combination in the child.
- This process, known as **recombination**, creates variation by shuffling existing genes in new ways.
- If there are **two parents**, it is called **crossover**.
- Recombination ensures **diversity** across generations and is a key mechanism in evolution.
- It increases the chances that offspring will **inherit** successful traits from parents.

Motivation for evolutionary computing (1/2)

- **Nature** has always served as a **source of inspiration** for engineers and scientists
 - The best problem solver known in nature is:
 - **the (human) brain** that created “the wheel, New York, wars and so on” (after Douglas Adams’ Hitch-Hikers Guide)
 - **the evolution mechanism** that created the human brain (after Darwin’s Origin of Species)
 - Answer 1 → **neurocomputing**
 - Answer 2 → **evolutionary computing**
- 

Motivation for evolutionary computing (2/2)

- Developing, analyzing, applying **problem solving methods** a.k.a. algorithms is a central theme in mathematics and computer science
- Time for thorough problem analysis and tailored algorithm design decreases
- Complexity of problems to be solved increases
- Consequence: **ROBUST PROBLEM SOLVING** technology needed



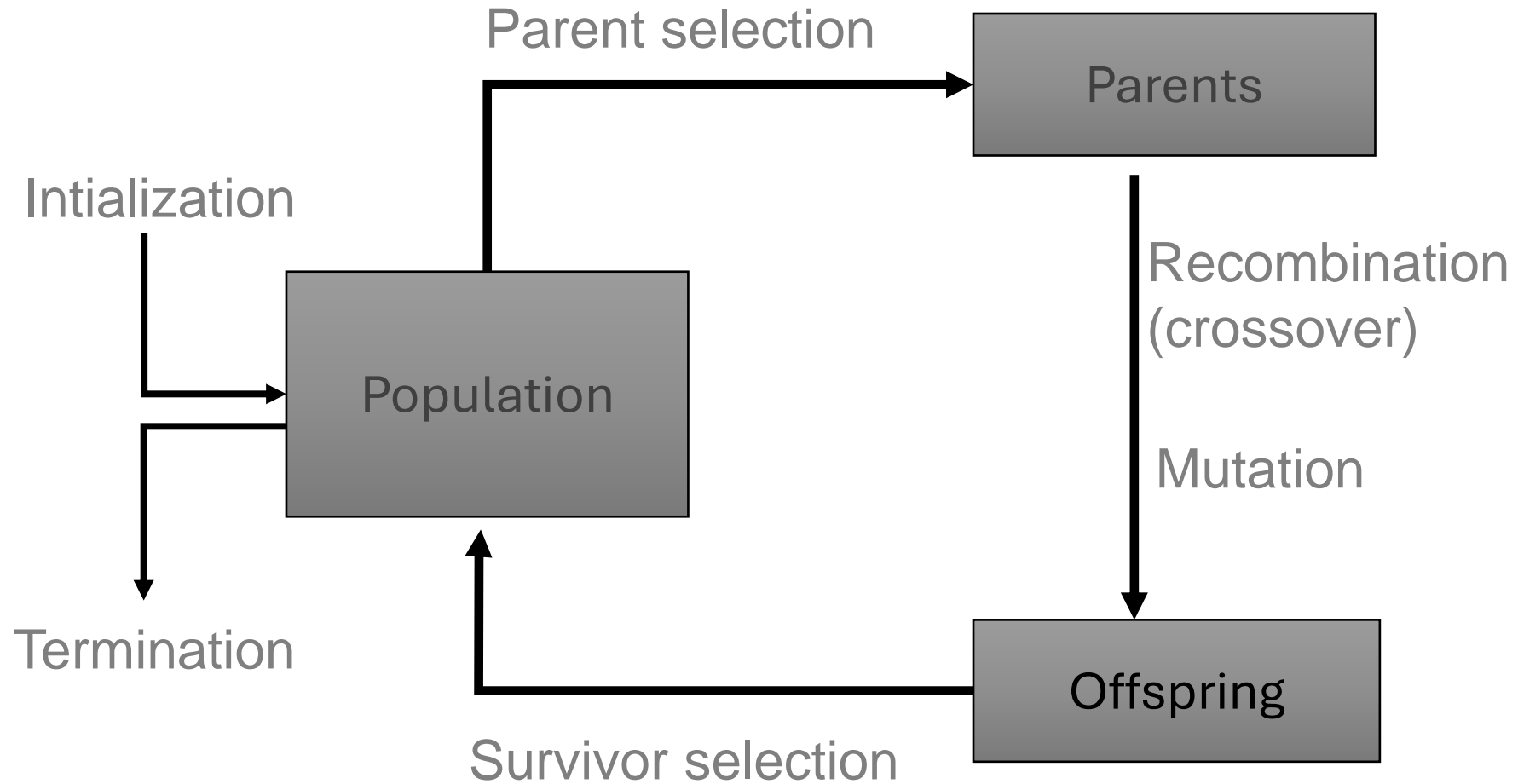
Robust algorithms are applicable to a wide range of problems, do not need much tailoring for specific problems, and deliver good (not necessarily optimal) solution within acceptable time.



Part II

The general scheme and components of evolutionary algorithms

General scheme of EAs



EA scheme in pseudo-code

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Scheme of an EA:

Common model of evolutionary processes

- **Population** of individuals
- Individuals have a **fitness**
- **Variation** operators: crossover, mutation
- **Selection** towards higher fitness
 - “survival of the fittest” and
 - “mating of the fittest”

Neo Darwinism (natural selection + genetic):

Evolutionary progress towards higher life forms

=

Optimization according to some fitness-criterion
(optimization on a fitness landscape)

Scheme of an EA:

Two pillars of evolution

There are two competing forces

Increasing population **diversity**
by genetic operators

- mutation
- recombination

Push towards **novelty**

Decreasing population **diversity** by
selection

- of parents
- of survivors

Push towards **quality**

Main EA components:

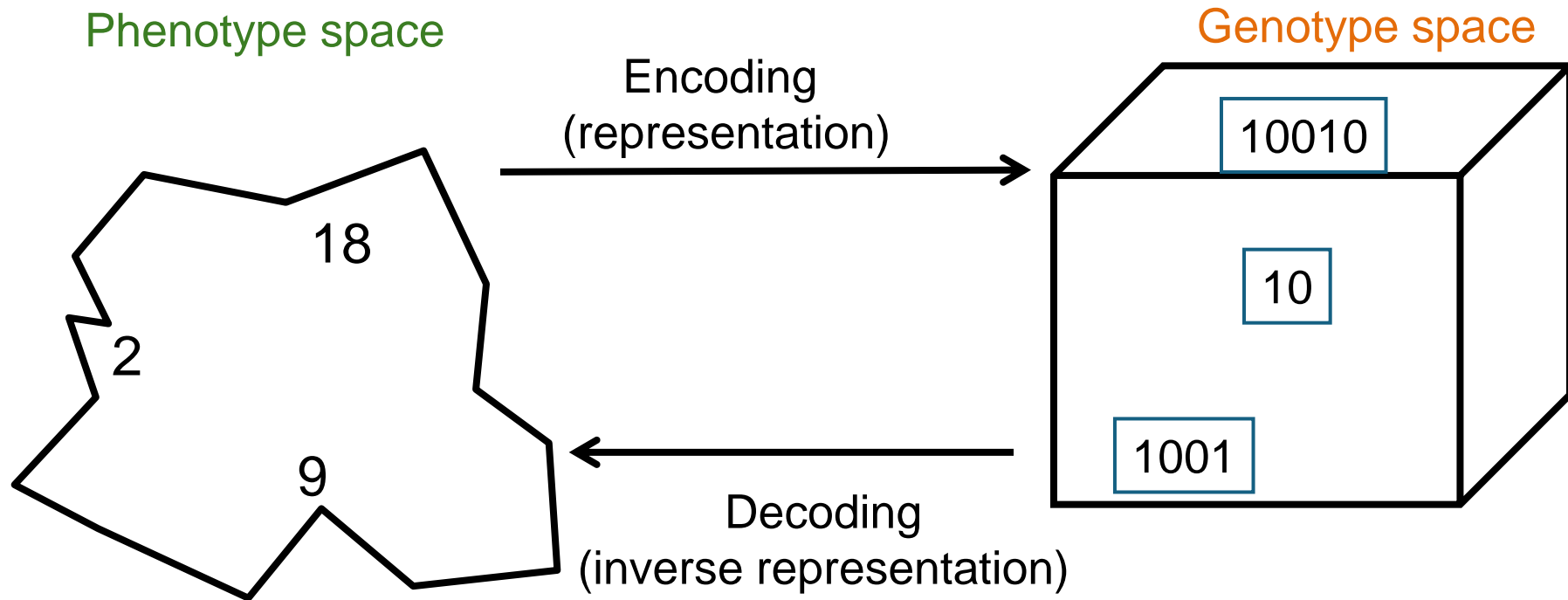
Representation (1/2)

- Role: provides code for **candidate solutions** that can be manipulated by variation operators
- Leads to two levels of existence
 - **phenotype: object** in original problem context, the outside
 - **genotype: code** to denote that object, the inside (chromosome, “digital DNA”)
- Implies two **mappings**:
 - **Encoding** : phenotype=> genotype (not necessarily one to one)
 - **Decoding** : genotype=> phenotype (must be one to one)
- Chromosomes contain **genes**, which are in (usually fixed) positions called **loci** (sing. locus) and have a value (**allele**)
 - Think of a gene as a "trait category" and an allele as a "specific option" for that trait.

Main EA components:

Representation (2/2)

Example: **represent integer values by their binary code**



In order to find the global optimum, **every feasible solution must be represented in genotype space**

Main EA components:

Evaluation (fitness) function

- Role:
 - **Represents** the **task** to solve, the requirements to adapt to (can be seen as “the environment”)
 - **Enables selection** (provides basis for comparison)
 - Operates on the **phenotype space**.
 - e.g., some phenotypic traits are advantageous, desirable, e.g. big ears cool better, these traits are rewarded by more offspring that will expectedly carry the same trait
- A.k.a. *quality* function or *objective* function
- Assigns a single real-valued fitness to each phenotype which forms the basis for selection
 - So the more discrimination (different values) the better
- Typically we talk about fitness being **maximised**
 - Some problems may be best posed as **minimisation** problems, but conversion is trivial

Main EA components:

Population (1/2)

- Role: holds the **candidate solutions** of the problem as individuals (genotypes)
- Formally, a population is a multiset of individuals, i.e. repetitions are possible
- Population is the basic **unit of evolution**, i.e., the population is evolving, not the individuals
- **Selection operators** act on **population** level
- **Variation operators** act on **individual** level

Main EA components:

Population (2/2)

- Some sophisticated EAs also assert a spatial structure on the population e.g., a grid
- Selection operators usually take whole population into account i.e., reproductive probabilities are *relative* to **current generation**
- **Diversity** of a population refers to the number of different fitnesses / phenotypes / genotypes present (note: not the same thing)

Main EA components:

Selection mechanism (1/3)

Role:

- Identifies individuals
 - ❖ to become **parents**
 - ❖ to **survive**
- Pushes population towards higher fitness
- Usually probabilistic
 - high quality solutions **more likely** to be selected than low quality
 - but **not guaranteed**
 - even worst in current population usually has non-zero probability of being selected
- This ***stochastic*** nature can aid **escape from local optima**

Main EA components:

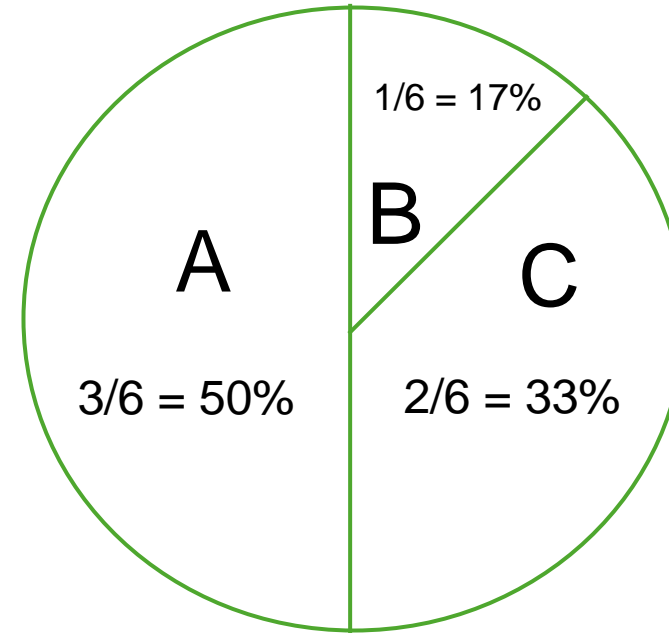
Selection mechanism (2/3)

Example: roulette wheel selection

$\text{fitness}(A) = 3$

$\text{fitness}(B) = 1$

$\text{fitness}(C) = 2$



In principle, any selection mechanism can be used for parent selection as well as for survivor selection

Main EA components:

Selection mechanism (3/3)

- Survivor selection A.k.a. ***replacement***
- **Most EAs use fixed population size** so need a way of going from (parents + offspring) to next generation
- **Often deterministic** (while parent selection is usually stochastic)
 - Fitness based : e.g., rank parents + offspring and take best
 - Age based: make as many offspring as parents and delete all parents
- Sometimes a combination of stochastic and deterministic (**elitism**)

Main EA components:

Variation operators

- Role: to generate **new candidate** solutions
- Usually divided into two types according to their **arity** (number of inputs):
 - Arity 1 : **mutation** operators
 - Arity >1 : **recombination** operators
 - Arity = 2 typically called **crossover**
 - Arity > 2 is formally possible, seldom used in EC
- There has been much debate about relative importance of recombination and mutation
 - Nowadays **most EAs use both**
 - Variation operators must match the given representation

Main EA components:

Mutation (1/2)

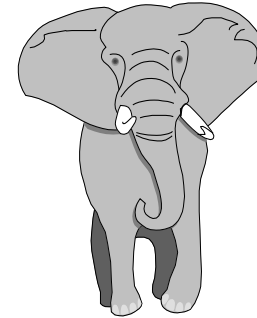
- Role: causes **small, random** variance
- Acts on one genotype and delivers another
- Element of randomness is essential and differentiates it from other unary heuristic operators
- Importance ascribed depends on representation and historical dialect:
 - Binary GAs – background operator responsible for preserving and introducing diversity
 - EP for FSM's / continuous variables – only search operator
 - GP – hardly used
- May guarantee connectedness of search space and hence **convergence** proofs

Main EA components:

Mutation (2/2)

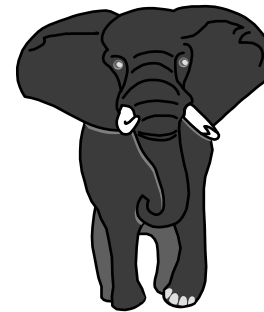
before

1 1 1 1 1 1 1



after

1 1 1 0 1 1 1



Main EA components:

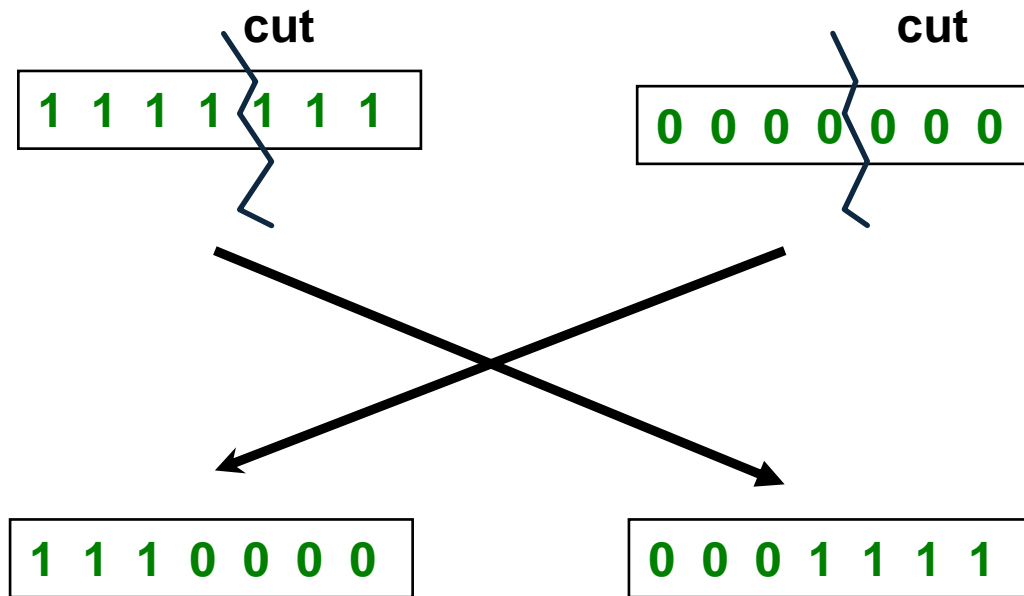
Recombination (1/2)

- Role: merges information from parents into offspring
- Choice of what information to merge is stochastic
- Most offspring may be worse, or the same as the parents
- Hope is that some are better by combining elements of genotypes that lead to good traits
- Principle has been used for millennia by breeders of plants and livestock

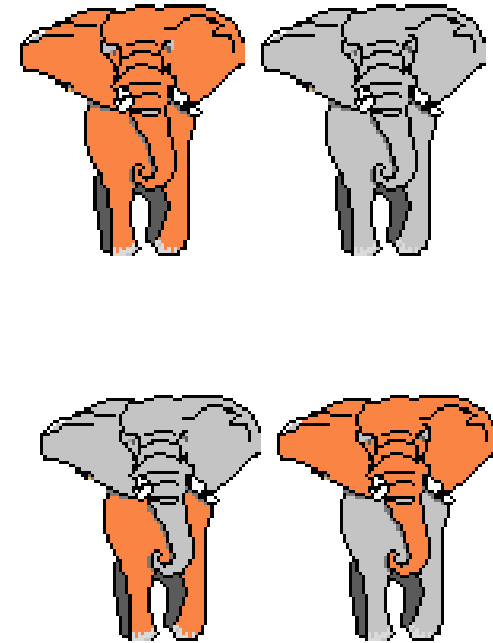
Main EA components:

Recombination (2/2)

Parents



Offspring



Main EA components:

Initialisation / Termination

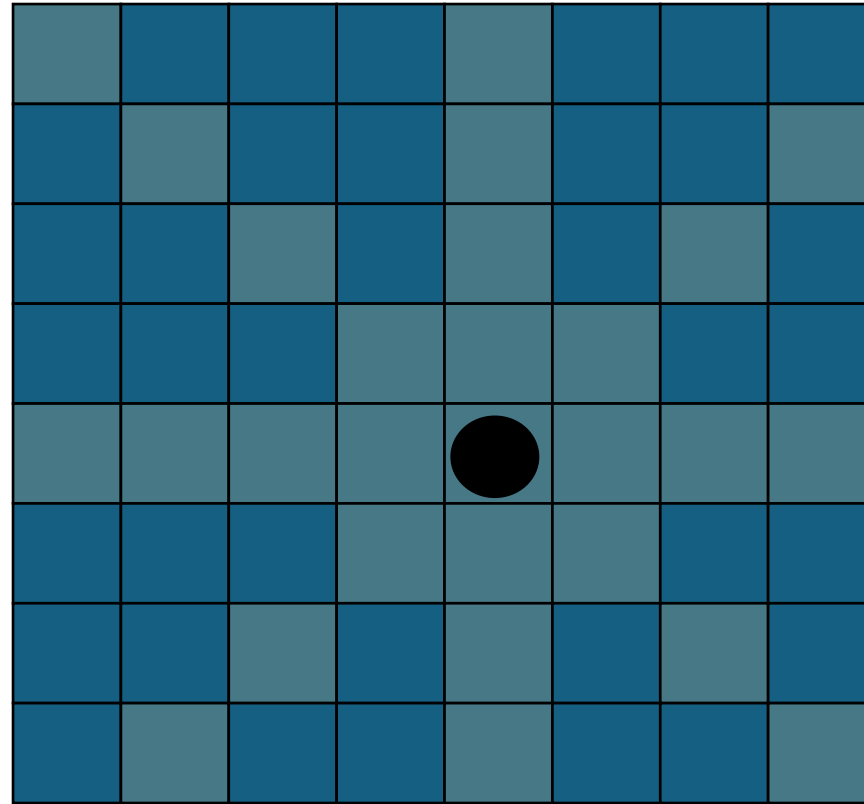
- **Initialisation** usually done at random,
 - Need to ensure even spread and mixture of possible allele values
 - Can include existing solutions, or use problem-specific heuristics, to “seed” the population
- **Termination condition** checked every generation
 - Reaching some (known/hoped for) fitness
 - Reaching some maximum allowed number of generations
 - Reaching some minimum level of diversity
 - Reaching some specified number of generations without fitness improvement

Main EA components:

What are the different types of EAs

- Historically different flavours of EAs have been associated with different data types to **represent** solutions
 - Binary strings, Permutations, Integers : Genetic Algorithms
 - Real-valued vectors : Evolution Strategies
 - Finite state Machines: Evolutionary Programming
 - LISP trees: Genetic Programming
- These differences are largely irrelevant, best strategy
 1. **choose representation** to suit problem
 2. **choose variation operators** to suit representation
- Selection operators only use fitness and so are independent of representation

Example: The 8-queens problem

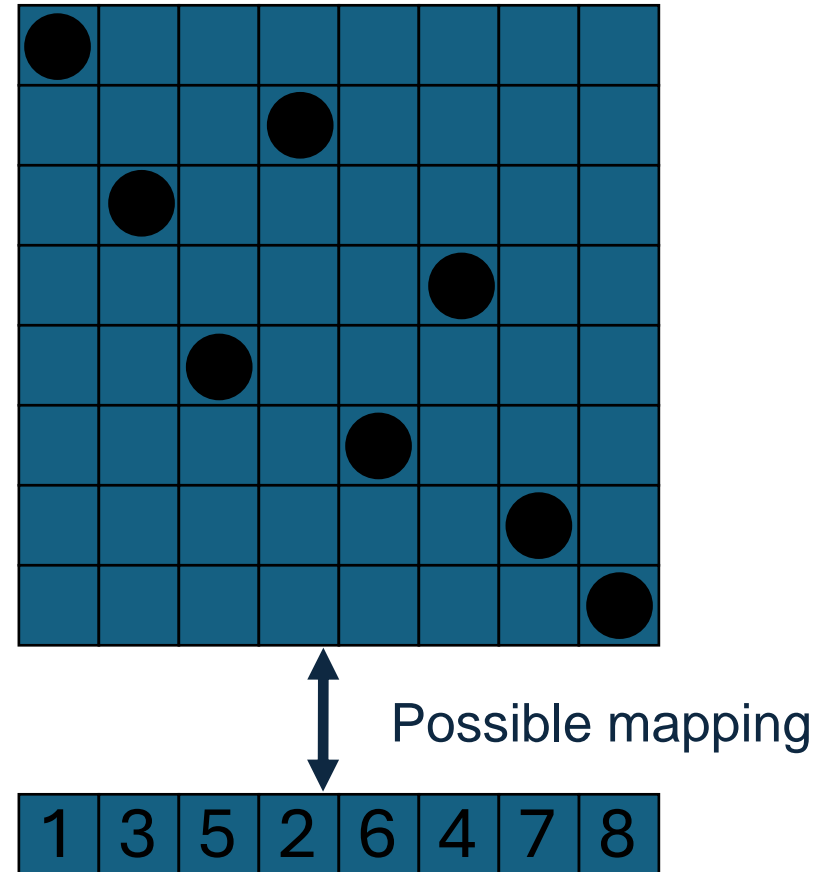


Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other

The 8-queens problem: Representation

Phenotype:
a board configuration

Genotype:
a permutation of
the numbers 1–8



The 8-queens problem:

Fitness evaluation

- **Penalty** of one queen: the number of queens she can check
- Penalty of a configuration: the sum of penalties of all queens
- Note: penalty is to be minimized
- **Fitness** of a configuration: inverse penalty to be maximized

The 8-queens problem:

Mutation

Small variation in one permutation, e.g.:

- **swapping** values of two randomly chosen positions,

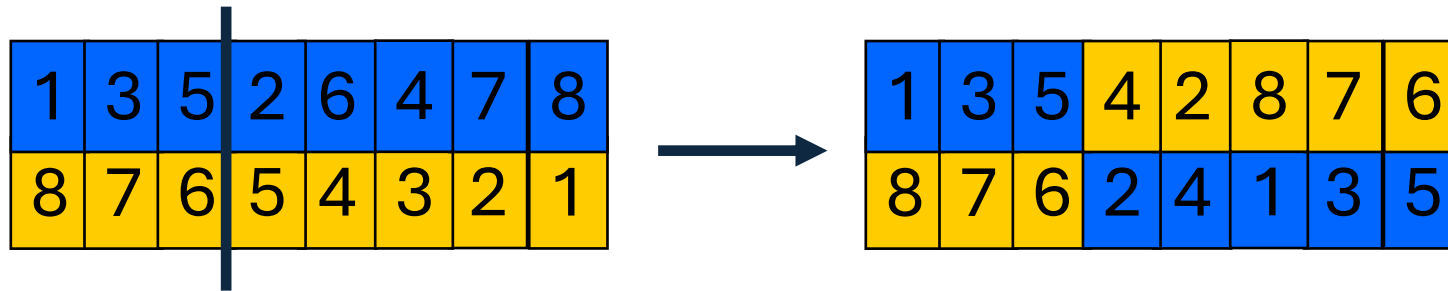


The 8-queens problem:

Recombination

Combining two permutations into two new permutations:

- choose **random crossover point**
- **copy first parts** into children
- create second part by inserting values from other parent:
 - in the **order** they appear there
 - **beginning** after crossover point
 - **skipping** values already in child



The 8-queens problem:

Selection

- Parent selection:
 - Pick 5 parents and take best two to undergo crossover
- Survivor selection (replacement)
 - When inserting a new child into the population, choose an existing member to replace by:
 - sorting the whole population by decreasing fitness
 - enumerating this list from high to low
 - replacing the first with a fitness lower than the given child

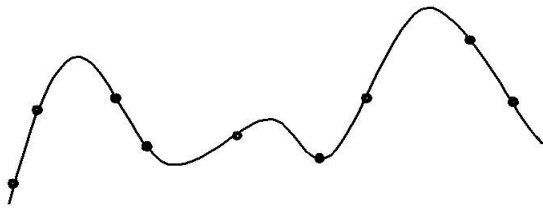
The 8-queens problem: Summary

Representation	Permutations
Recombination	“Cut-and-crossfill” crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluation

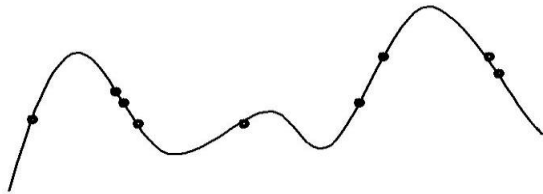
Note that there may be other
set of choices for operators and parameters

Typical EA behaviour: Stages

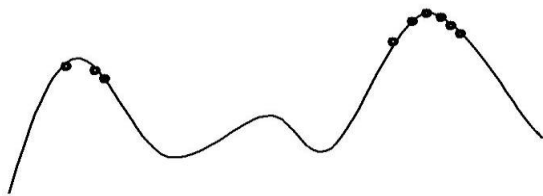
Stages in optimising on a 1-dimensional fitness landscape



Early stage:
quasi-random population distribution



Mid-stage:
population arranged around/on hills



Late stage:
population concentrated on high hills

Typical EA behaviour:

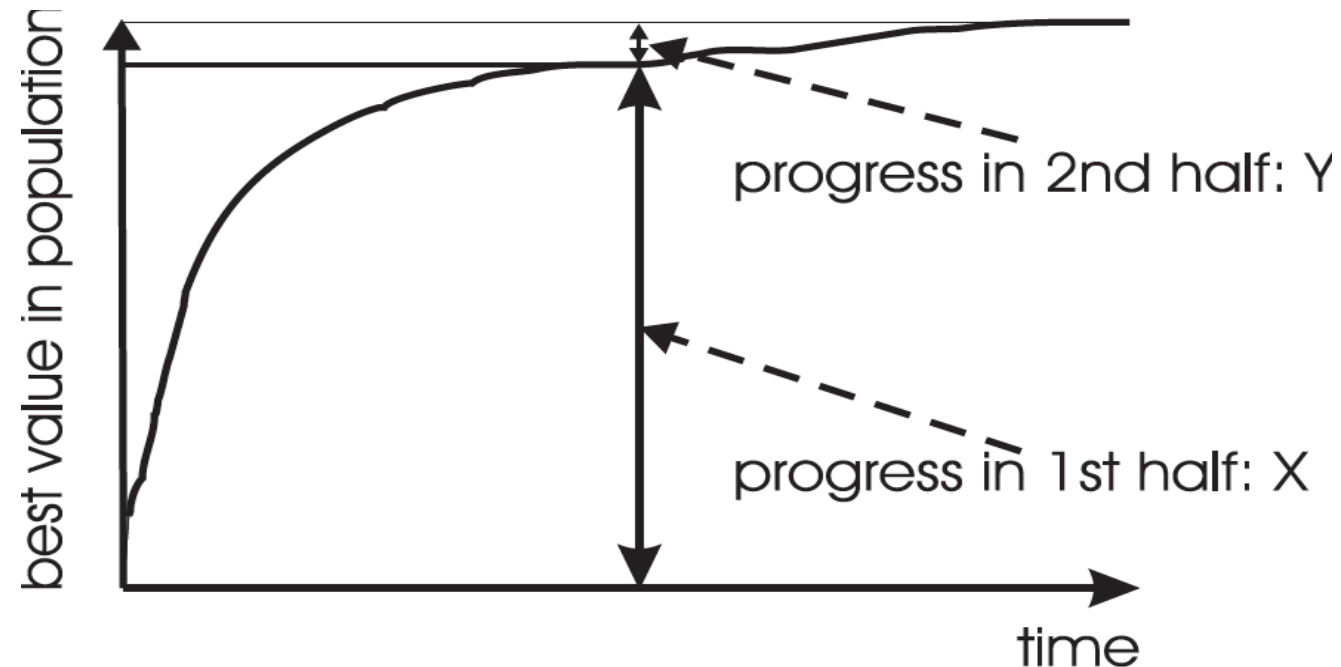
Typical run: progression of fitness



Typical run of an EA shows so-called “anytime behavior”

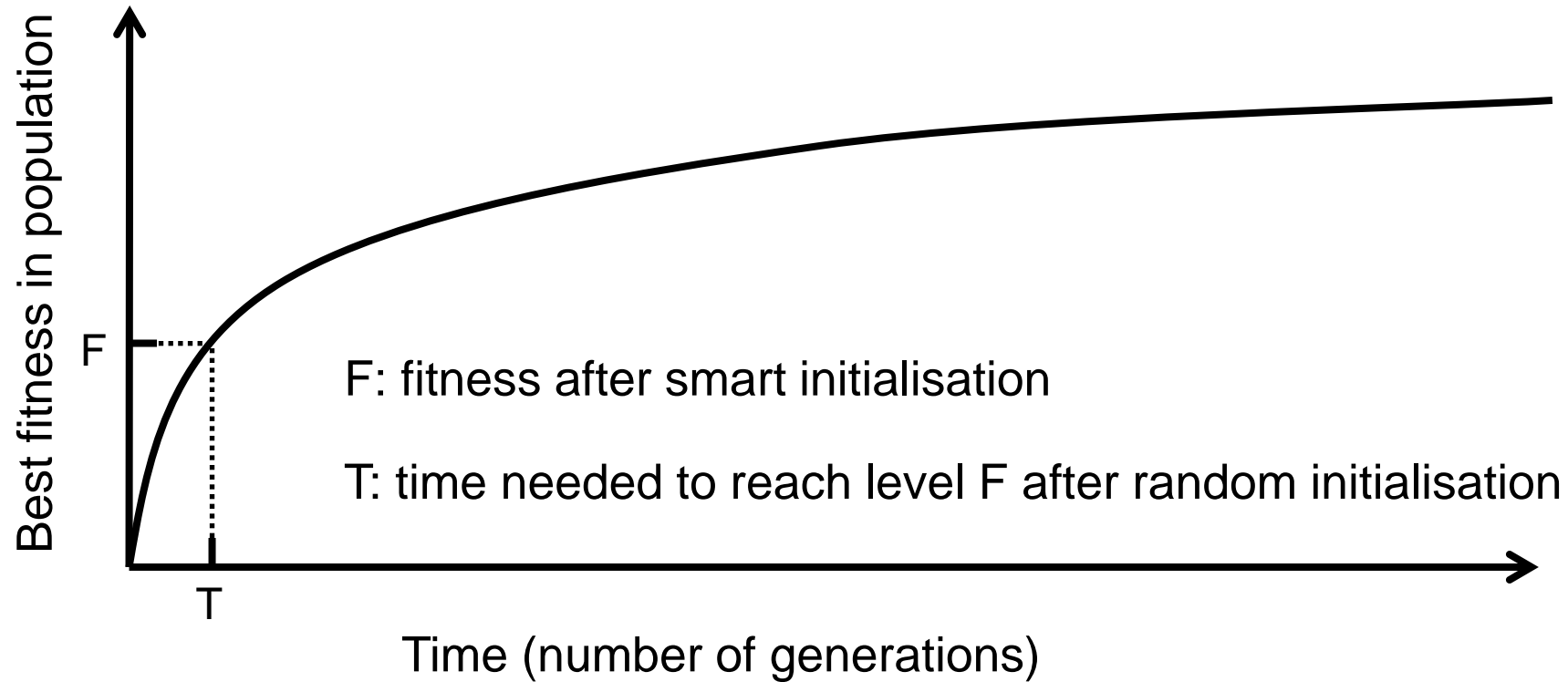
Typical EA behaviour: Are long runs beneficial?

- Answer:
 - It depends on how much you want the last bit of progress
 - May be better to do more short runs



Typical EA behaviour:

Is it worth expending effort on smart initialisation?



- Answer: it depends.
 - Possibly good, if good solutions/methods exist.
 - Care is needed, see chapter/lecture on hybridisation.

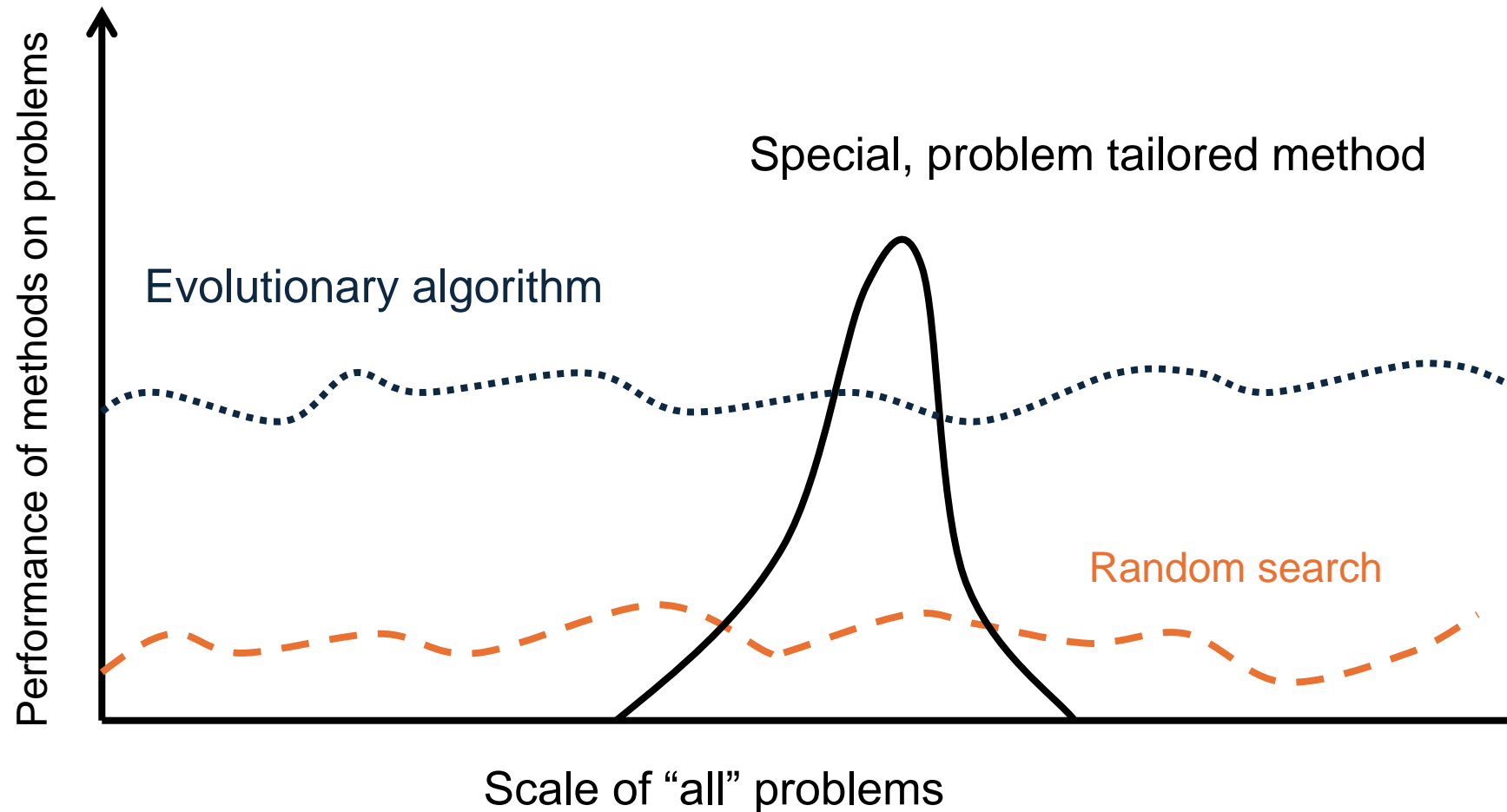
Typical EA behaviour:

Evolutionary Algorithms in context

- There are many views on the use of EAs as robust problem solving tools
- For most problems a problem-specific tool may:
 - perform better than a generic search algorithm on most instances,
 - have limited utility,
 - not do well on all instances
- Goal is to provide robust tools that provide:
 - evenly good performance
 - over a range of problems and instances

Typical EA behaviour:

EAs as problem solvers: Goldberg view (1989)

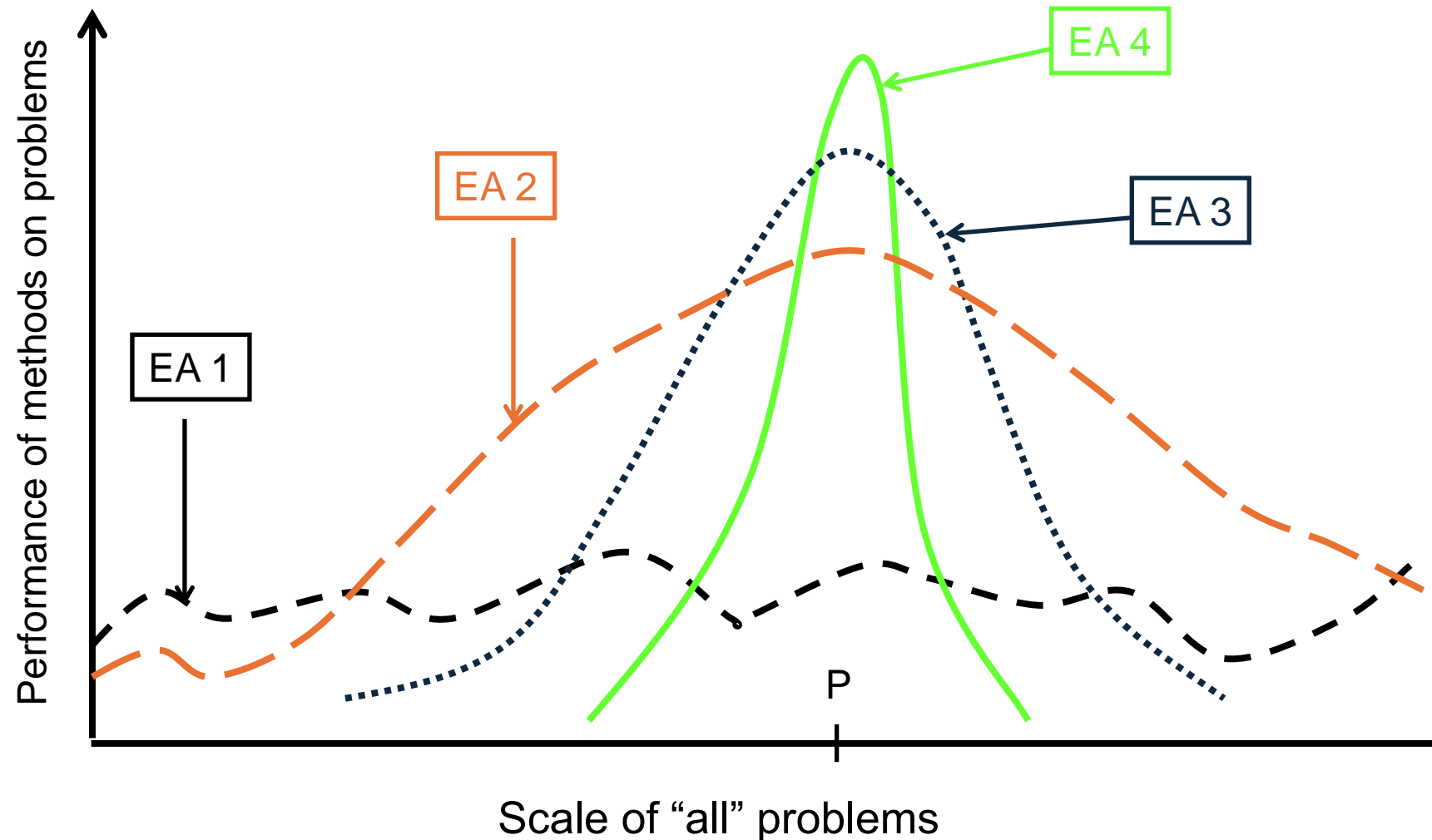


Typical EA behaviour: EAs and domain knowledge

- Trend in the 90's:
adding problem specific knowledge to EAs
(special variation operators, repair, etc)
- Result: EA performance curve “deformation”:
 - better on problems of the given type
 - worse on problems different from given type
 - amount of added knowledge is variable
- Recent theory suggests the search for an “all-purpose” algorithm may be fruitless

Typical EA behaviour:

EAs as problem solvers: Michalewicz view (1996)

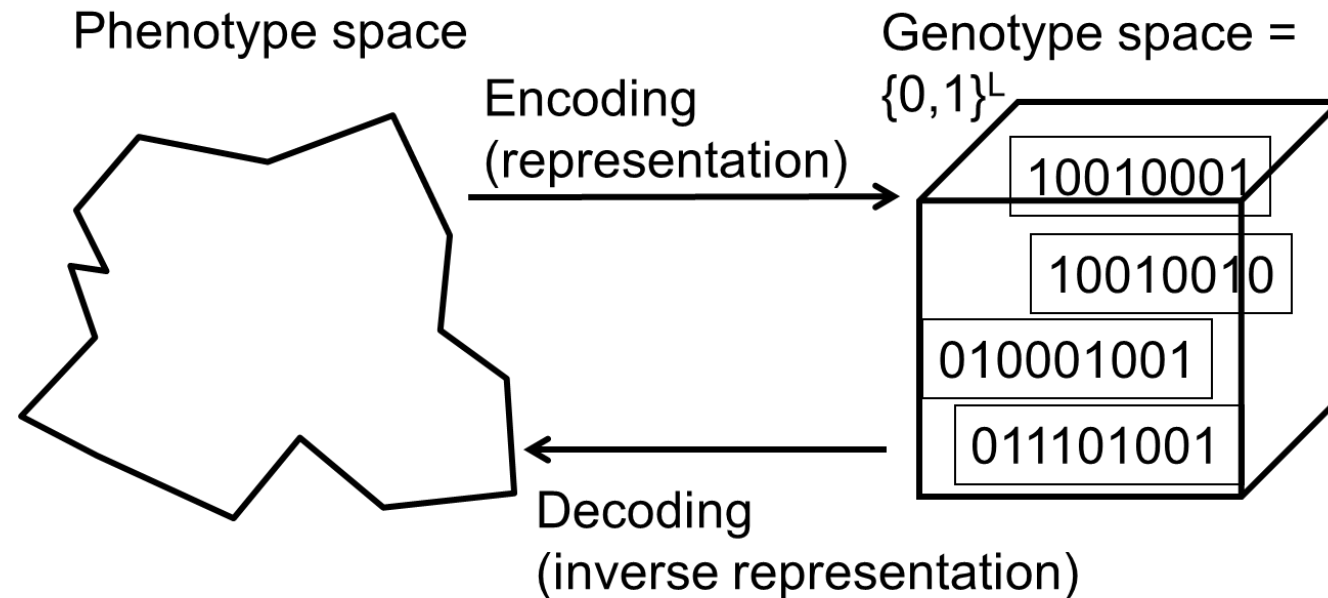




Binary Genetic Algorithm

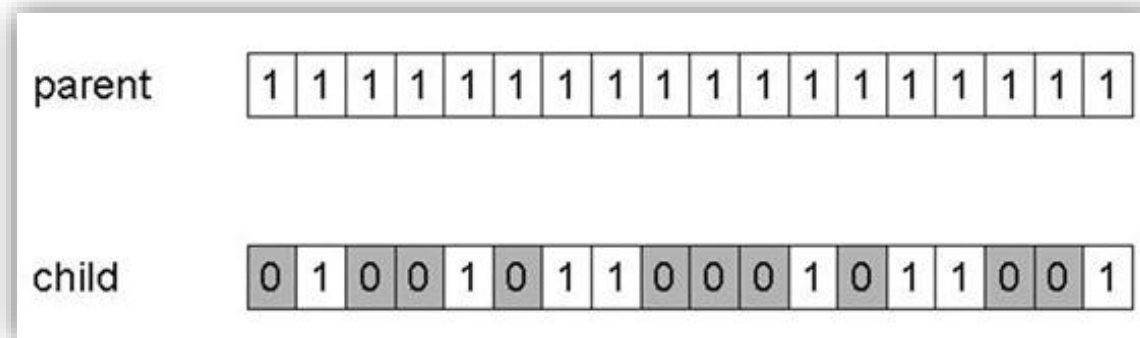
Binary Representation

- One of the earliest representations
- Genotype consists of a string of binary digits



Binary Representation: Mutation

- Alter each gene independently with a probability p_m (bit-flip mutation)
- p_m is called the mutation rate
 - Typically between $(1/\text{pop_size})$ and $(1/\text{chromosome_length})$

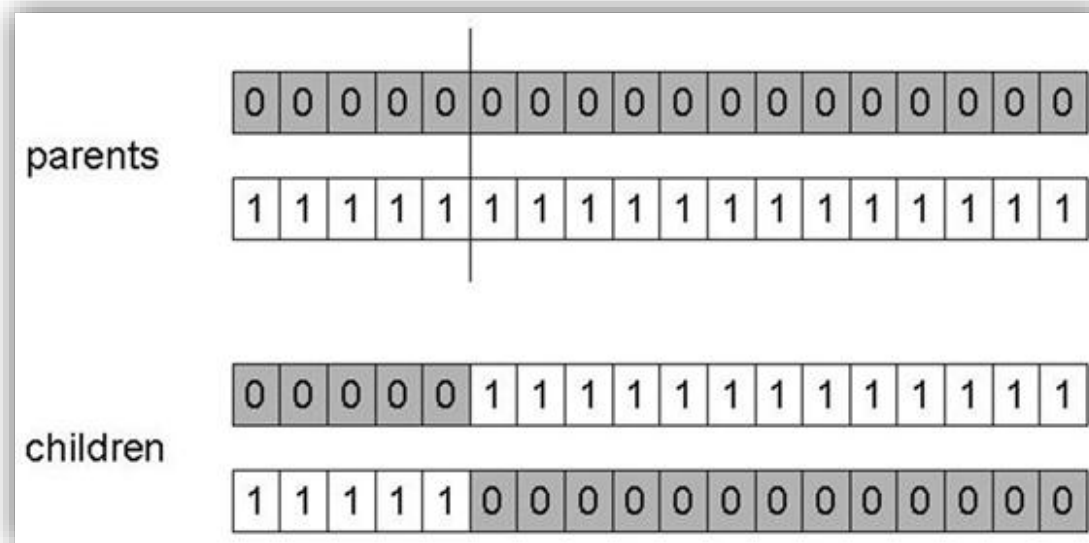


- Mutation can cause variable effect (use gray coding)

Binary Representation:

1-point crossover

- Choose a **random point** on the two parents
- **Split** parents at this crossover point
- Create children by **exchanging** tails
- P_c typically in range (0.6, 0.9)

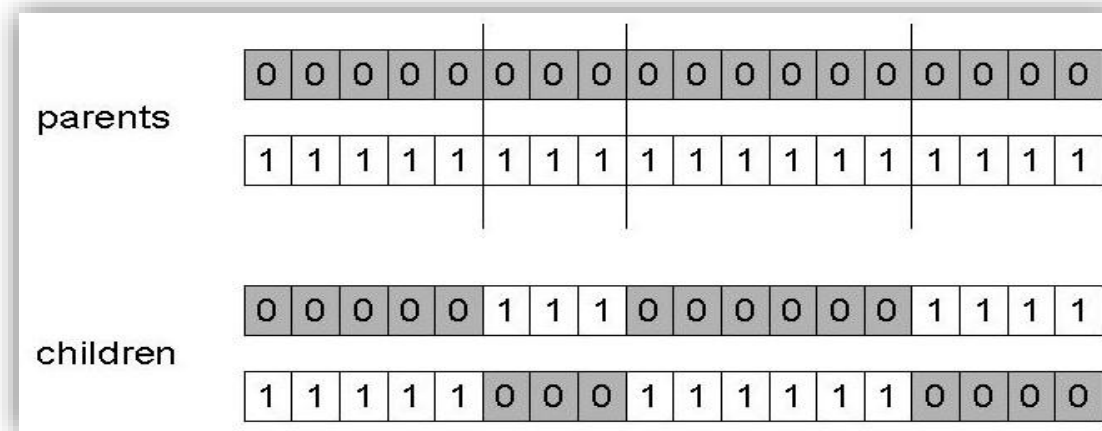


Binary Representation: Alternative Crossover Operators

- Why do we need other crossover(s)?
- Performance with 1-point crossover depends on the order that variables occur in the representation
 - More likely to keep together genes that are near each other
 - Can never keep together genes from opposite ends of string
 - This is known as **Positional Bias**
 - Can be exploited if we know about the structure of our problem, but this is not usually the case

Binary Representation: n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1-point (still some positional bias)



Binary Representation:

Uniform crossover

- 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
- Inheritance is independent of position

parents	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
children	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
	1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0

Binary Representation:

Crossover OR mutation? (1/3)

- Decade long debate: which one is better / necessary / main-background
- Answer (at least, rather wide agreement):
 - it **depends** on the problem, but
 - **in general, it is good to have both**
 - both have another role
 - **mutation-only-EA is possible**, crossover-only-EA would not work

Binary Representation:

Crossover OR mutation? (1/2)

- 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
- To hit the optimum you often need a '**lucky**' **mutation**

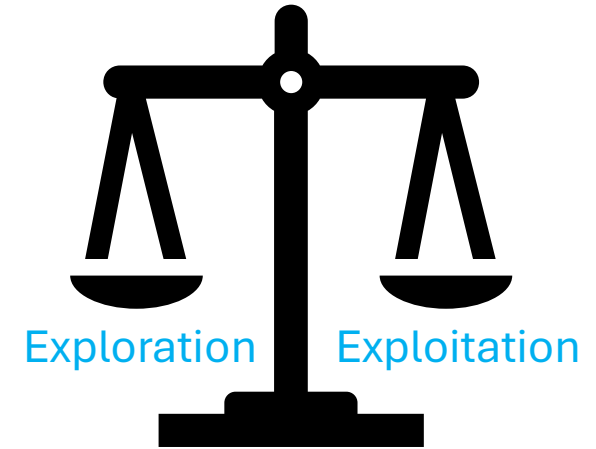
Exploration vs. Exploitation

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 of **randomly selected parents** is **explorative**, it makes a *big* jump to an area somewhere “in between” two (parent) areas. However, crossover using **high quality parents** (elitism) is **exploitative**.
- A **small mutation** is **exploitative**, it creates random *small* diversions, thereby staying near (in the area of) the parent. However, a **big mutation** is **explorative**.



An evolutionary cycle by Hand (only 1 gen.) – part I

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
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
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Table 3.1. The x^2 example, 1: initialisation, evaluation, and parent selection

Initial population generation: random

Genotype: binary bit strings of length 5

Phenotype: integer numbers [0, 31]

Fitness: maximize, assume black-box function $f(x)=x^2$

Crossover: 1-point crossover, 2 parents generate 2 children

Mutation: bit-flip (with a probability p_m)

Parent selection: fitness proportional

Survival selection: keep only new individuals

An evolutionary cycle by Hand (only 1 gen.) – part II

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 1	4	0 1 1 0 0	12	144
2	1 1 0 0 0	4	1 1 0 0 1	25	625
2	1 1 0 0 0	2	1 1 0 1 1	27	729
4	1 0 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Table 3.2. The x^2 example, 2: crossover and offspring evaluation

An evolutionary cycle by Hand (only 1 gen.) – part III

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

Table 3.3. The x^2 example, 3: mutation and offspring evaluation

An example binary GA: one-max problem

- "Download and run the '**Binary_GA_example.py**' Python program from the Teams course page. This code solves the **one_max problem**, which aims to maximize the number of 1s in a binary chromosome. You only need the NumPy library installed to run this example in a standard Python environment.
- There is also '**mystery**' function to be optimized. Just change the fitness function to use this mystery function and observe the best solution. What is the goal of this problem? Try changing the parameter values and observe the results. To make the problem harder or easier, you can adjust the **n_bits** parameter.
- You may also want to **enhance** this basic algorithm with different operators such as n-point crossover, uniform crossover, elitism in survival selection, etc. You can try to optimize hyper-parameters. You can also solve **other binary optimization** problems using the same code if you **adapt the fitness function** accordingly."