UNIVERSITÄT PASSAU

Search-Based Software Engineering

Evolutionary Algorithms - Part II

Gordon Fraser
Lehrstuhl für Software Engineering II

Contents

- What is Evolution?
- History of Evolutionary Computation
- Evolutionary Algorithms

Components of an EA

```
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
```

Components of an EA

Representation

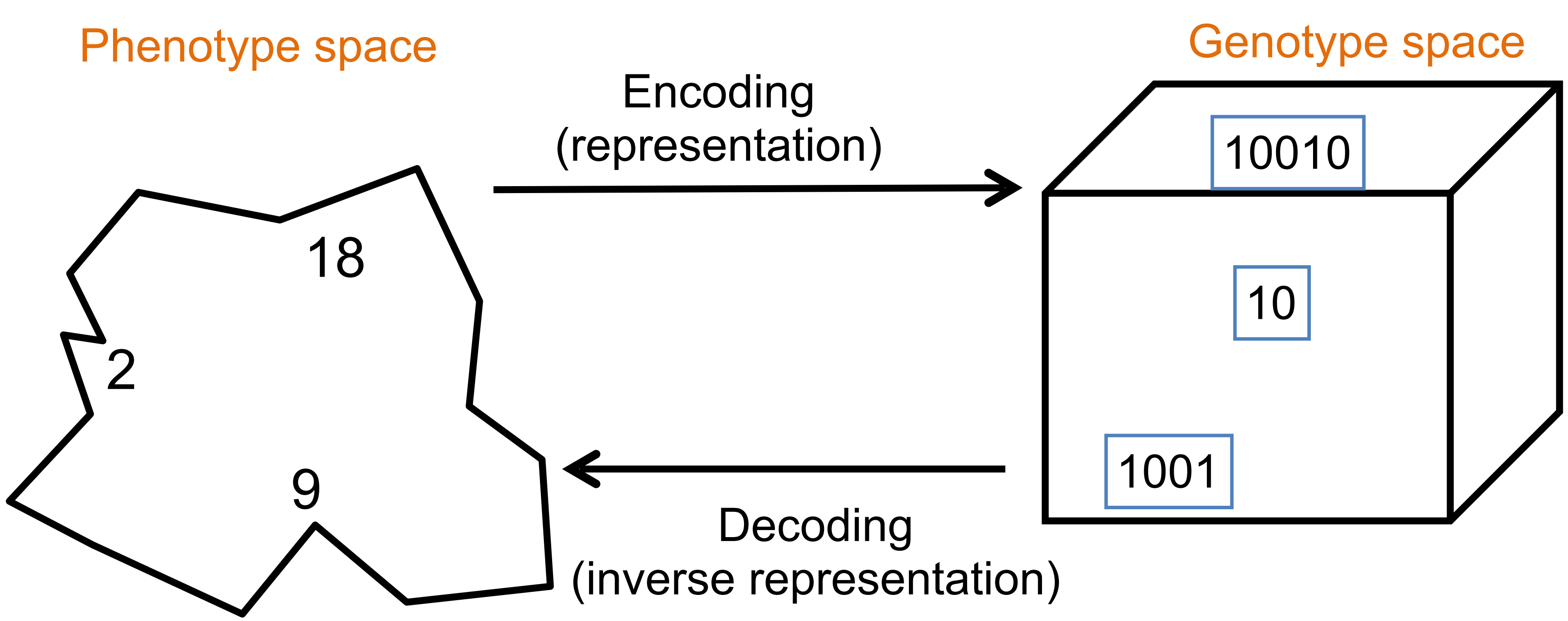
```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals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

Representation

- Encodes candidate solutions that can be manipulated by variation operators
- Leads to two levels of existence:
  - Phenotype: object in original problem context
  - Genotype: code to denote that object (chromosome)
- 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)

Binary Representation

- Example: Represent integer values by their binary code
- In order to find the global optimum, every feasible solution must be represented in genotype space



Integer Representation

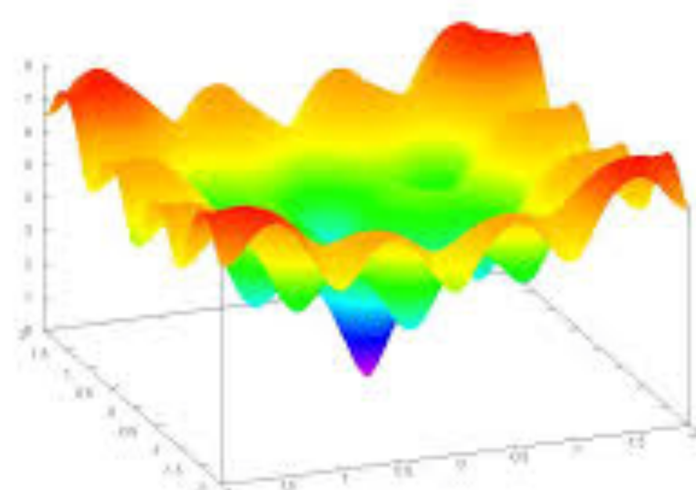
- Nowadays it is generally accepted that it is better to encode numerical variables directly (integers, floating point variables)
- Some problems naturally have integer variables, e.g. image processing parameters
- Others take categorical values from a fixed set e.g. {blue, green, yellow, pink}
- Requires different mutation and crossover operators

Real-Valued Representation

- Many problems occur as real valued problems, e.g. continuous parameter optimisation $f : \mathbb{R}^n \rightarrow \mathbb{R}$

- Illustration: Ackley's function (often used in EC)

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$

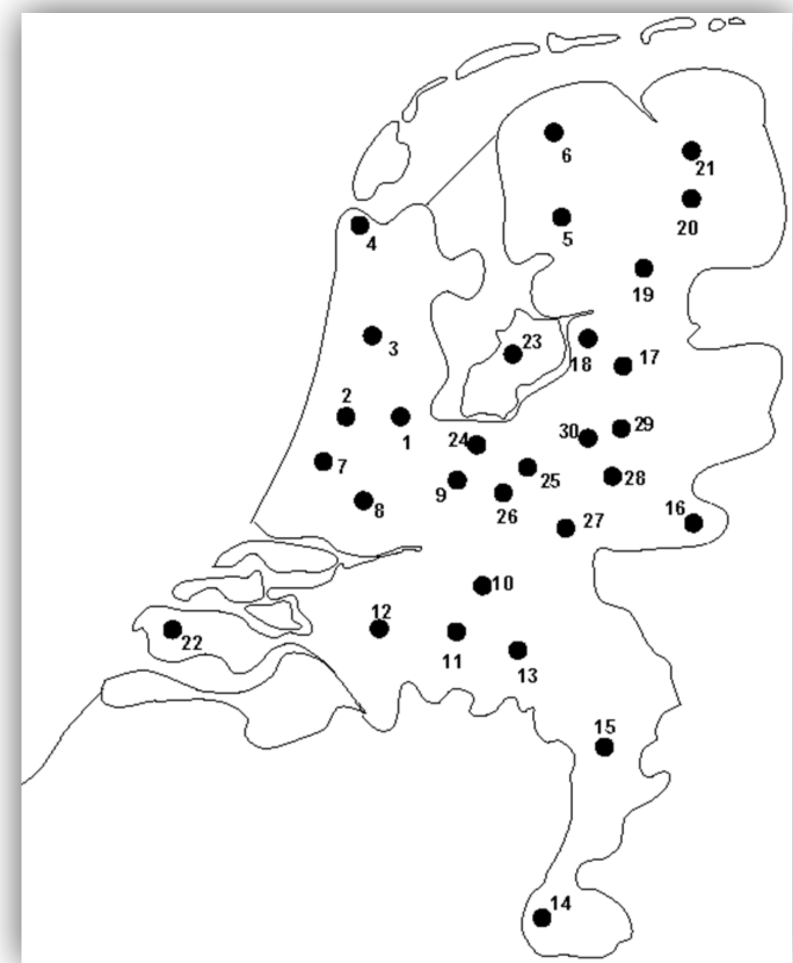


Permutation Representation

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

Example: TSP

- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , n
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG



Tree Representation

- Trees are a universal form, e.g. consider
- Arithmetic formula: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program:

```
i =1;
while (i < 20)
{
    i = i +1
}
```

…see lecture on Genetic Programming

Components of an EA

```
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
```

Evaluation Function

Components of an EA

```
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
```

Selection

Two Competing Forces

Increasing population diversity by genetic operators

- mutation
- recombination

Push towards novelty

Decreasing population diversity by selection

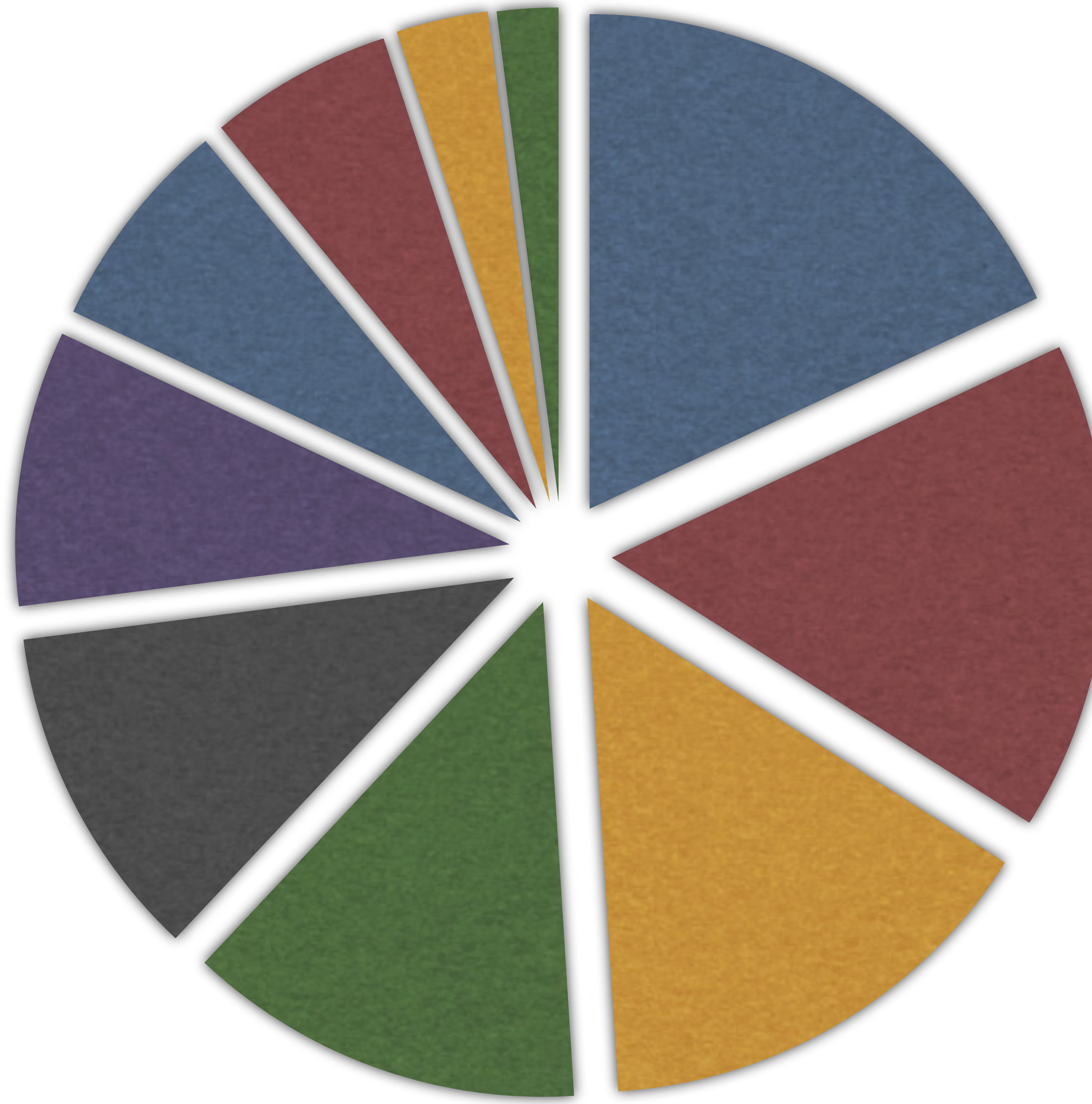
- of parents
- of survivors

Push towards quality

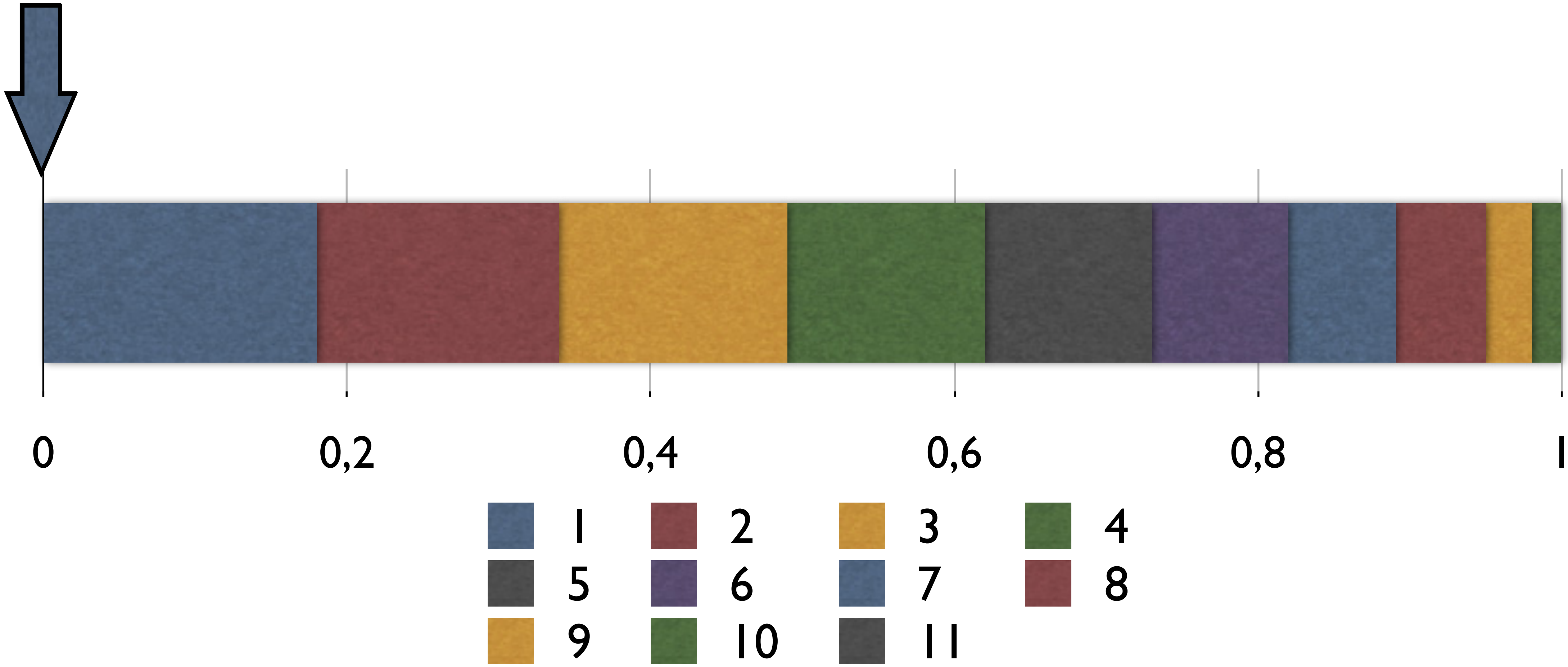
Selection Mechanism

- Identifies individuals
  - to become parents: Selection from current generation to take part in mating (parent selection)
  - to survive: Selection from parents + offspring to go into next generation (survivor selection)
- Pushes population towards higher fitness
- Selection operators are representation-independent
- 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

Individual	Fitness
1	2
2	1,8
3	1,6
4	1,4
5	1,2
6	1
7	0,8
8	0,6
9	0,4
10	0,2
11	0



- Chosen value: 0,81
- Chosen value: 0,32
- Chosen value: 0,01



Rank Selection

- The best individual (i=0) is given a fitness s, between 1 and 2
- The worst individual (i=N-1) is given 2 - s
- Intermediate strings' fitness values are given by interpolation (for position i, and population size N):

$$f(i) = s - \frac{2i(s-1)}{N-1}$$

- Since this prescription automatically gives an average fitness of 1, the fitness values translate directly as the expected number of reproductive opportunities.
- If s is set to 2, the worst string gets no chance of reproduction

Components of an EA

```
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
```

Variation operator: Mutation

Variation operator: Crossover

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

Components of an EA

```
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
```

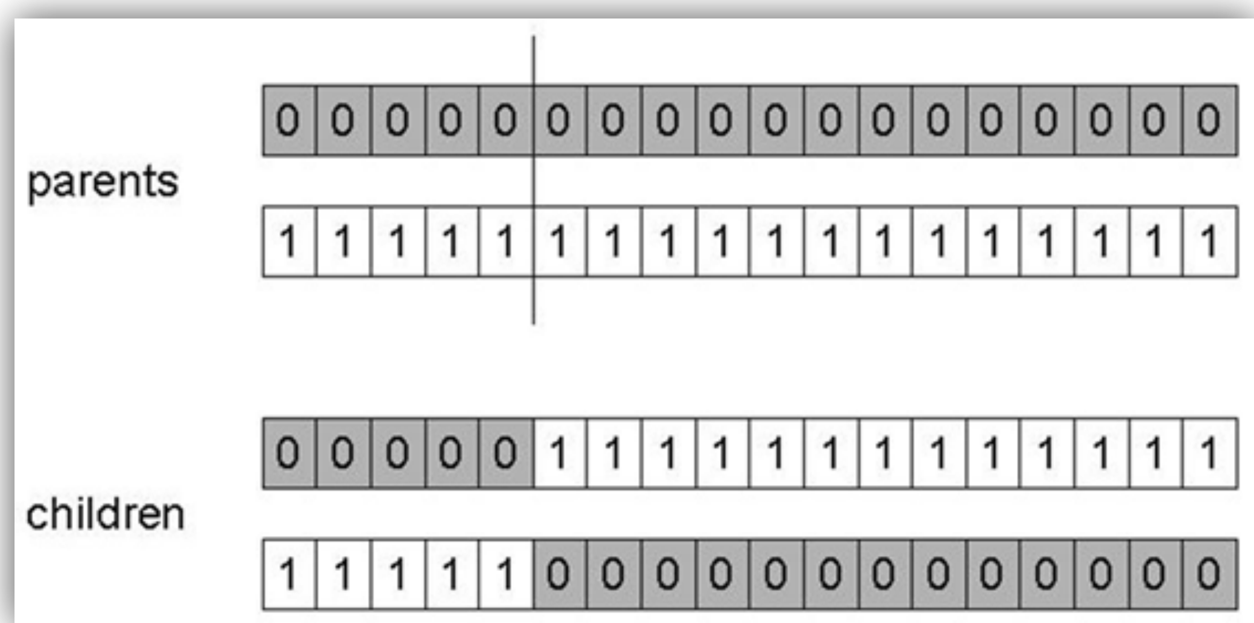
Variation operator: Crossover

Recombination

- 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

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)

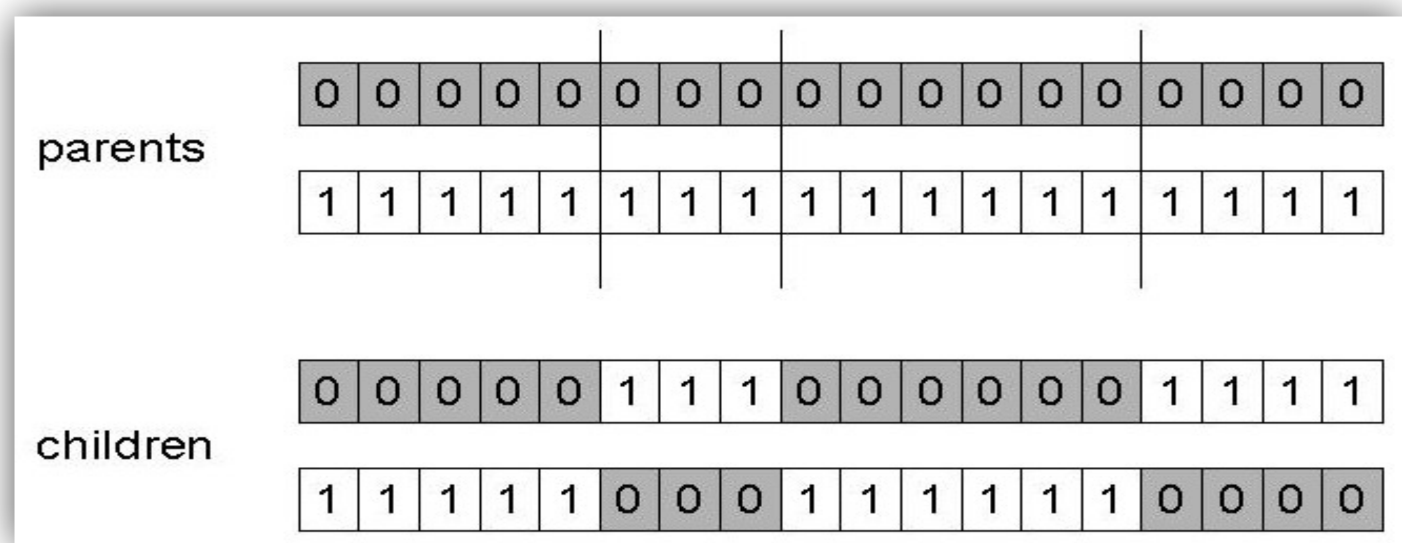


Alternative Crossover Operations

- 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

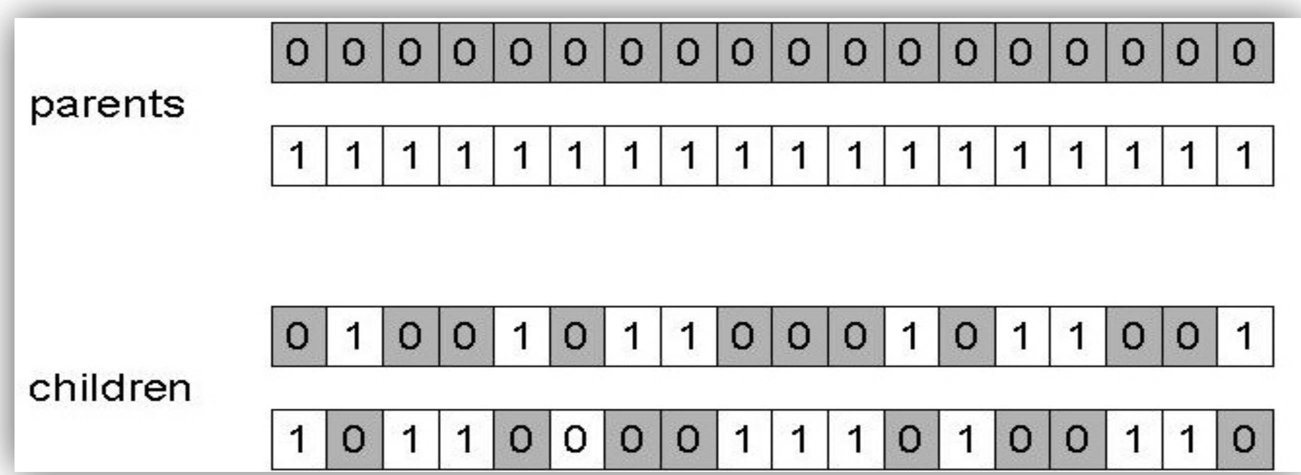
n-point Crossover

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



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



Real-Valued Representation

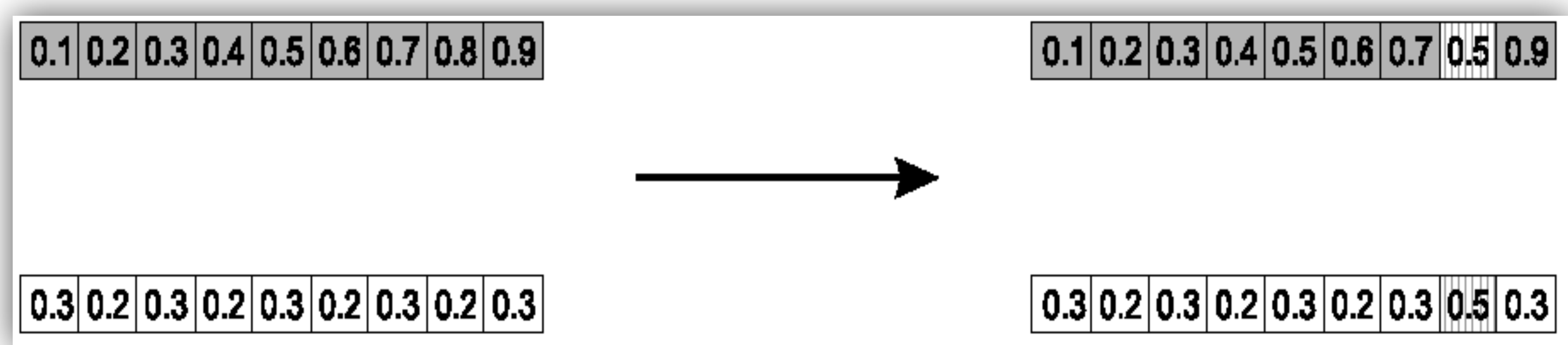
- Discrete:
  - Each allele value in offspring z comes from one of its parents (x,y) with equal probability: $z_i = x_i$ or y_i
  - Could use n-point or uniform
- Intermediate
  - Exploits idea of creating children “between” parents (hence a.k.a. arithmetic recombination)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $\alpha : 0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - constant: uniform arithmetical crossover
    - variable (e.g. depend on the age of the population)
    - picked at random every time

Single arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n\rangle$
- Pick a single gene (k) at random,
- $child_1$ is:

$$\left\langle x_1, \ldots, x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, \ldots, x_n \right\rangle$$

- Reverse for other child. e.g. with $\alpha = 0.5$

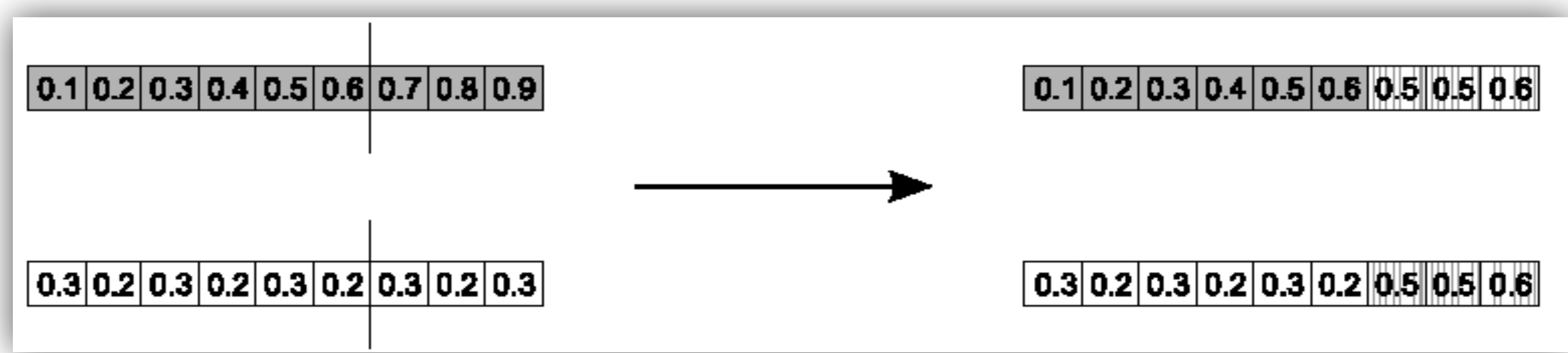


Simple arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n\rangle$
- Pick a random gene (k) after this point mix values
- child_1 is:

$$\left\langle x_1,\ldots,x_k,\alpha\cdot y_{k+1}+(1-\alpha)\cdot x_{k+1},\ldots,\alpha\cdot y_n+(1-\alpha)\cdot x_n\right\rangle$$

- reverse for other child. e.g. with $\alpha = 0.5$

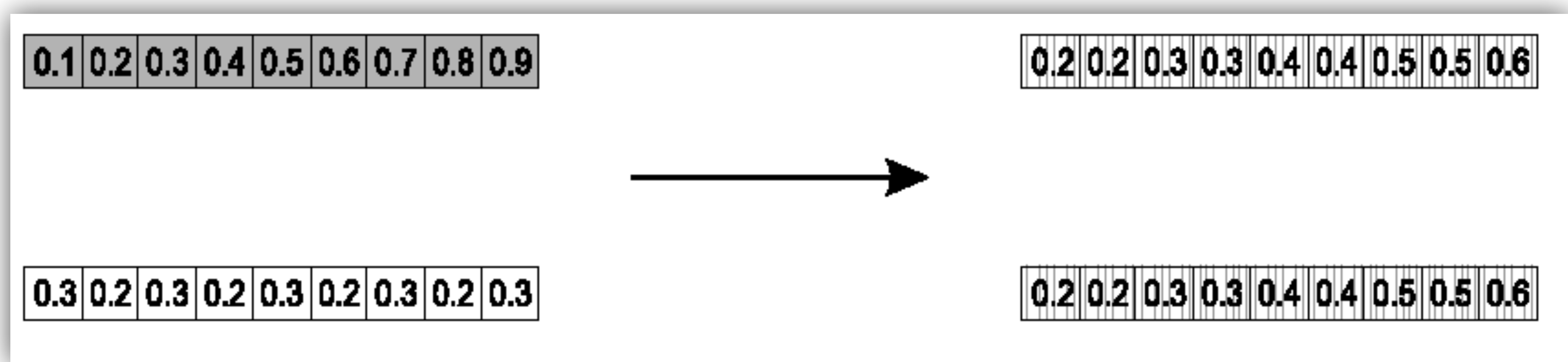


Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Child$_1$ is:

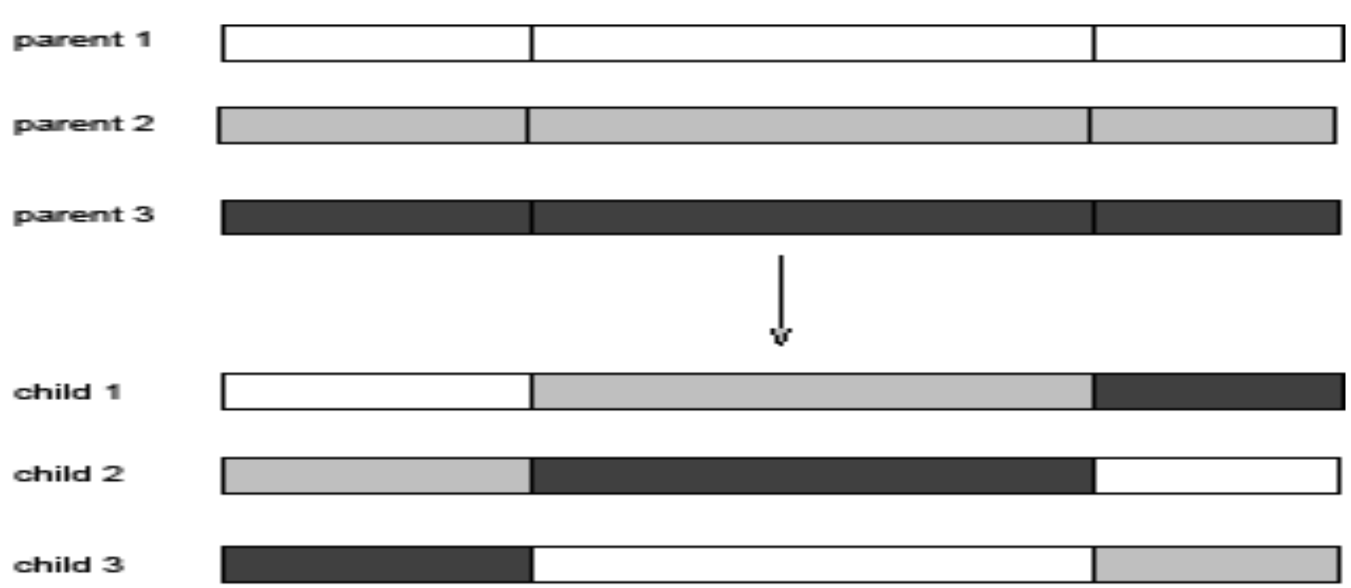
$$a \cdot \bar{x} + (1-a) \cdot \bar{y}$$

- reverse for other child. e.g. with α = 0.5



Multi-parent crossover

- Idea: segment and recombine parents
- Example: diagonal crossover for *n* parents:
  - Choose *n-1* crossover points (same in each parent)
  - Compose *n* children from the segments of the parents in along a “diagonal”, wrapping around

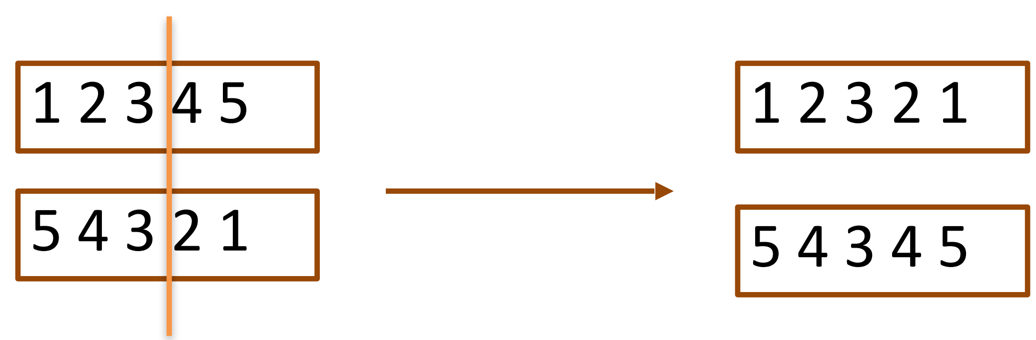


Multiple children

- This operator generalises 1-point crossover

Permutation Representation

- “Normal” crossover operators will often lead to inadmissible solutions



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

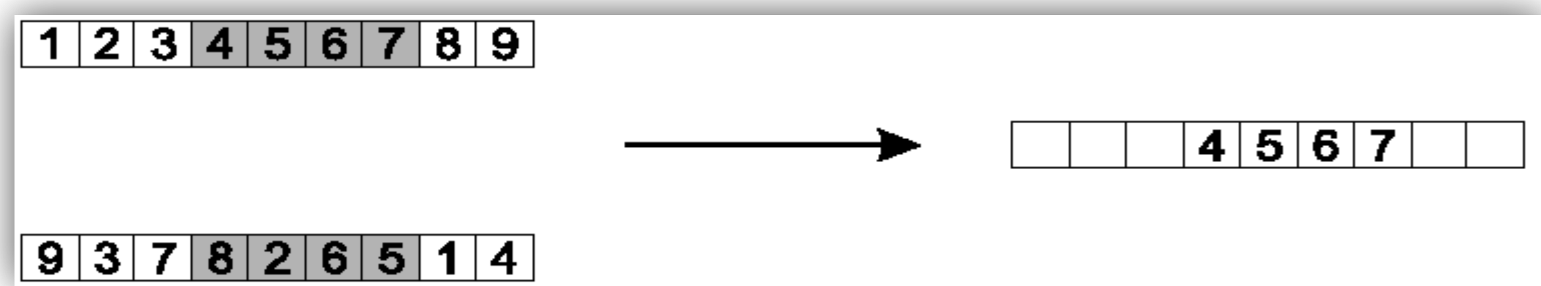
Order-1 Crossover

- Idea is to preserve relative order that elements occur
- Informal procedure:

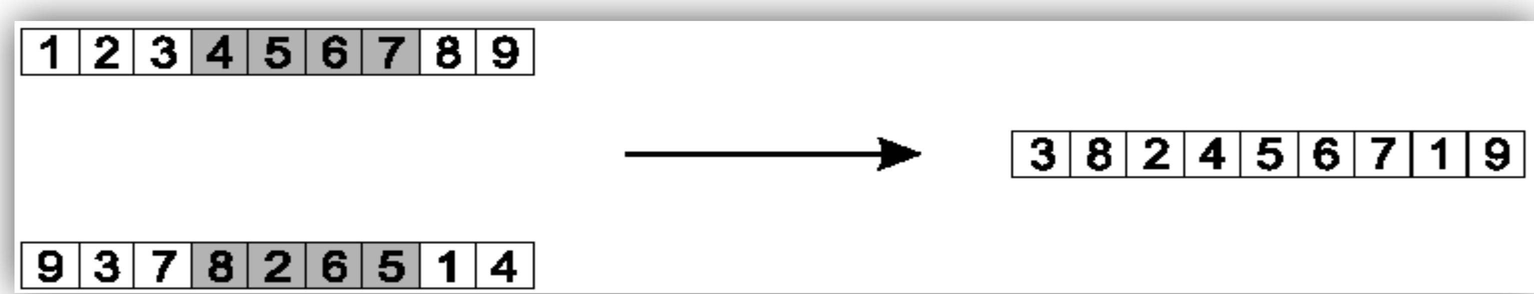
1. Choose an arbitrary part from the first parent
2. Copy this part to the first child
3. Copy the numbers that are not in the first part, to the first child:
    - starting right from cut point of the copied part,
    - using the order of the second parent
    - and wrapping around at the end
4. Analogous for the second child, with parent roles reversed

Order-1 Crossover

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2

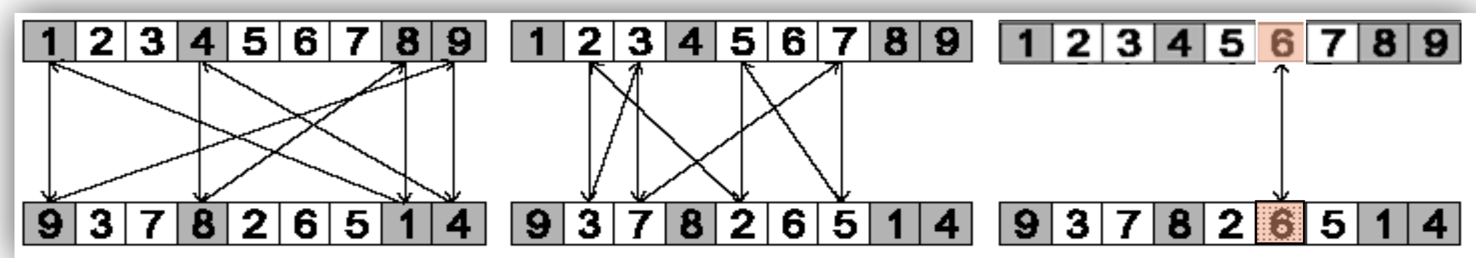


Cycle Crossover

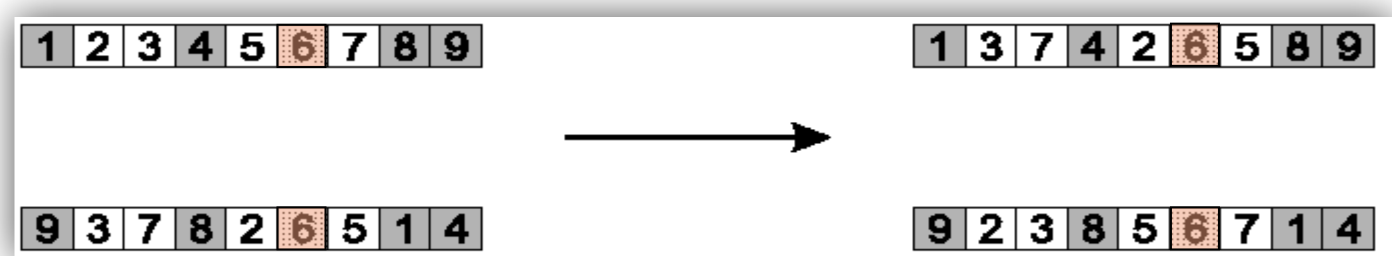
- Informal procedure:
- 1. Make a cycle of alleles from P1 in the following way.
    - (a) Start with the first allele of P1.
    - (b) Look at the allele at the same position in P2.
    - (c) Go to the position with the same allele in P1.
    - (d) Add this allele to the cycle.
    - (e) Repeat step b through d until you arrive at the first allele of P1.
- 2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
- 3. Take next cycle from second parent

Cycle Crossover

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Crossover or Mutation

- 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, xover-only-EA would not work
- Crossover is explorative, it makes a big jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, it creates random small diversions, thereby staying near (in the area of) the parent
- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)

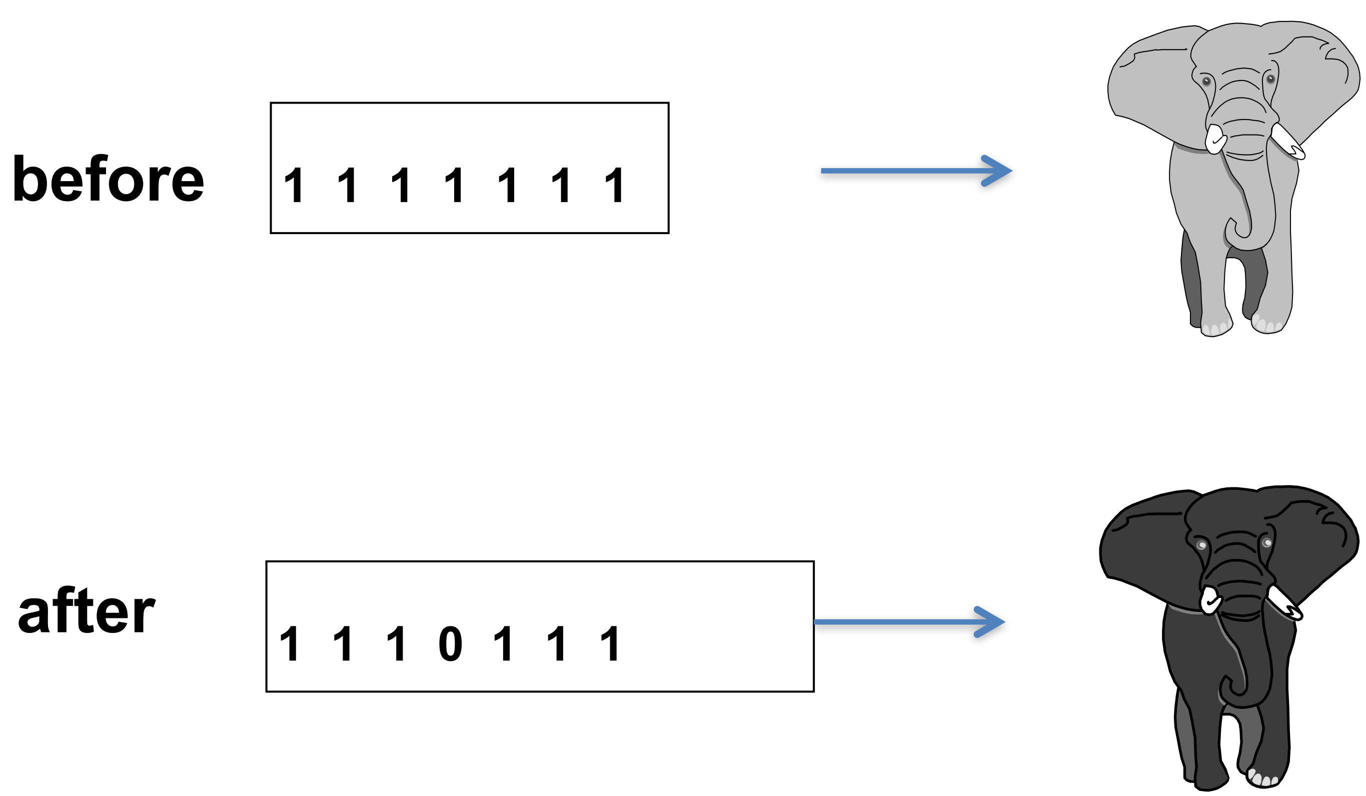
Components of an EA

```
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
```

Variation operator: Mutation

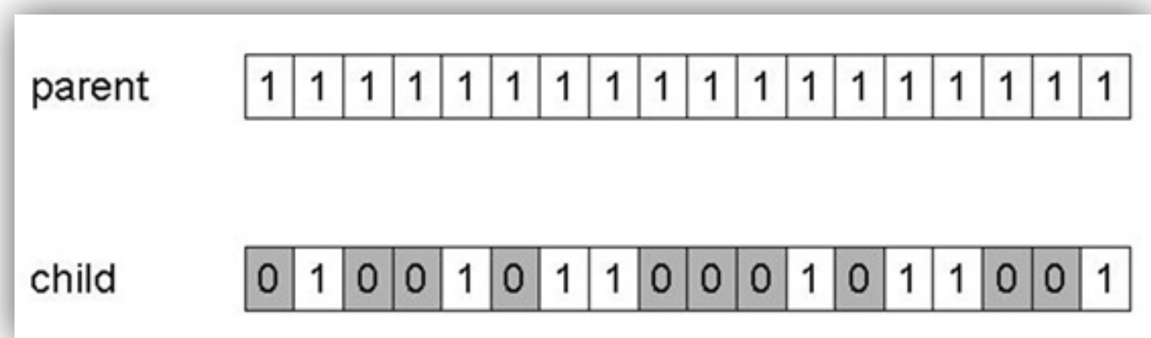
Mutation

- Causes small, random variance
- Acts on one genotype and delivers another
- Element of randomness is essential and differentiates it from other unary heuristic operators



Binary Representation

- Alter each gene independently with a probability P_m
- P_m is called the mutation rate
- Typically between 1/pop_size and 1/chromosome_length



- Mutation can cause variable effect (use gray coding)

Real-Valued Representation

- General scheme of floating point mutations

$$\bar{x} = \langle x_1, \; \ldots, x_l \rangle \rightarrow \bar{x}' = \langle x_1', \ldots, x_l' \rangle \qquad x_i, x_i' \in [LB_i, UB_i]$$

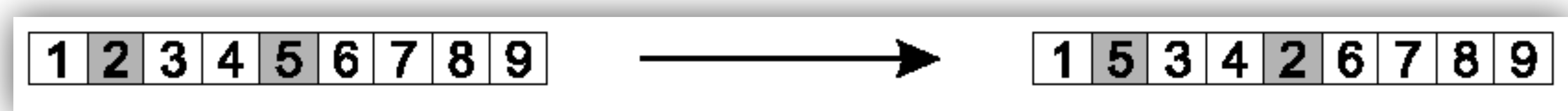
- Uniform Mutation
  - x'_i drawn randomly (uniform) from $[LB_i, UB_i]$
  - Analogous to bit-flipping (binary) or random resetting (integers)
- Non-uniform Mutation:
  - Add random deviate to each variable separately, taken from N(0, σ) Gaussian distribution and then curtail to range
    $x'_i = x_i + N(0,\sigma)$
  - Standard deviation σ, mutation step size, controls amount of change (2/3) of drawings will lie in range (-σ to +σ))

Permutation Representation

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

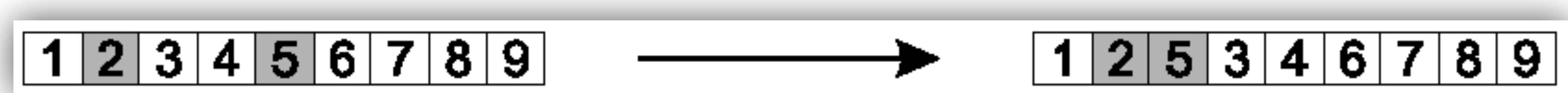
Swap Mutation

- Pick two alleles at random and swap their positions



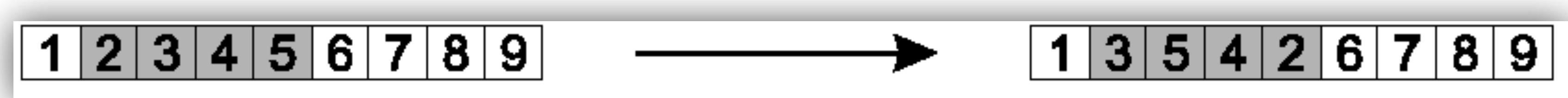
Insert Mutation

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



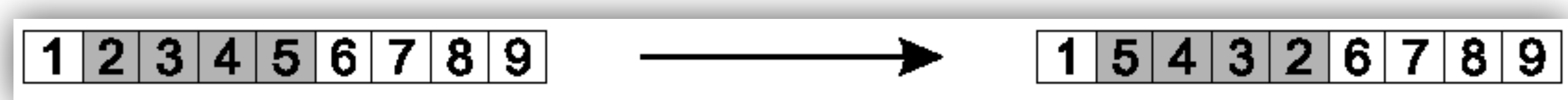
Scramble Mutation

- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions



Inversion Mutation

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



Components of an EA

Population

```
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
```

Population

- Holds the candidate solutions of the problem as individuals (genotypes)
- Formally, a population is a multiset of individuals, i.e. repetitions are possible
- Some sophisticated EAs also assert a spatial structure on the population e.g., a grid
- 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
- Diversity of a population refers to the number of different fitnesses / phenotypes / genotypes present (note: not the same thing)

Population

- Two different population management models exist:
  - Generational model
    - each individual survives for exactly one generation
    - the entire set of parents is replaced by the offspring
  - Steady-state model
    - one offspring is generated per generation
    - one member of population replaced
- Generation gap
  - The proportion of the population replaced
  - Parameter = 1.0 for GGA, = 1/pop_size for SSGA

Components of an EA

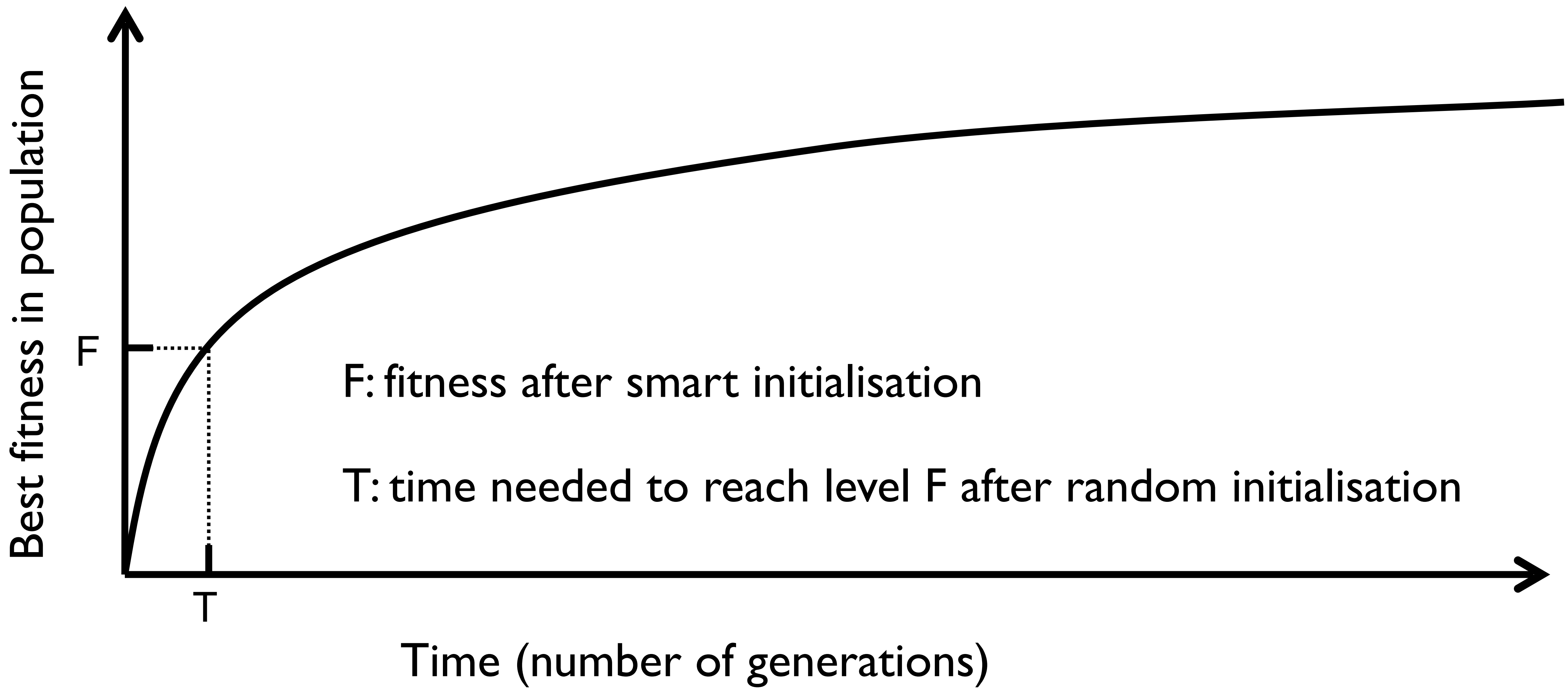
Initialisation

```
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
```

Initialisation

- 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

Typical EA behaviour:
Is it worth expending effort on smart initialisation?



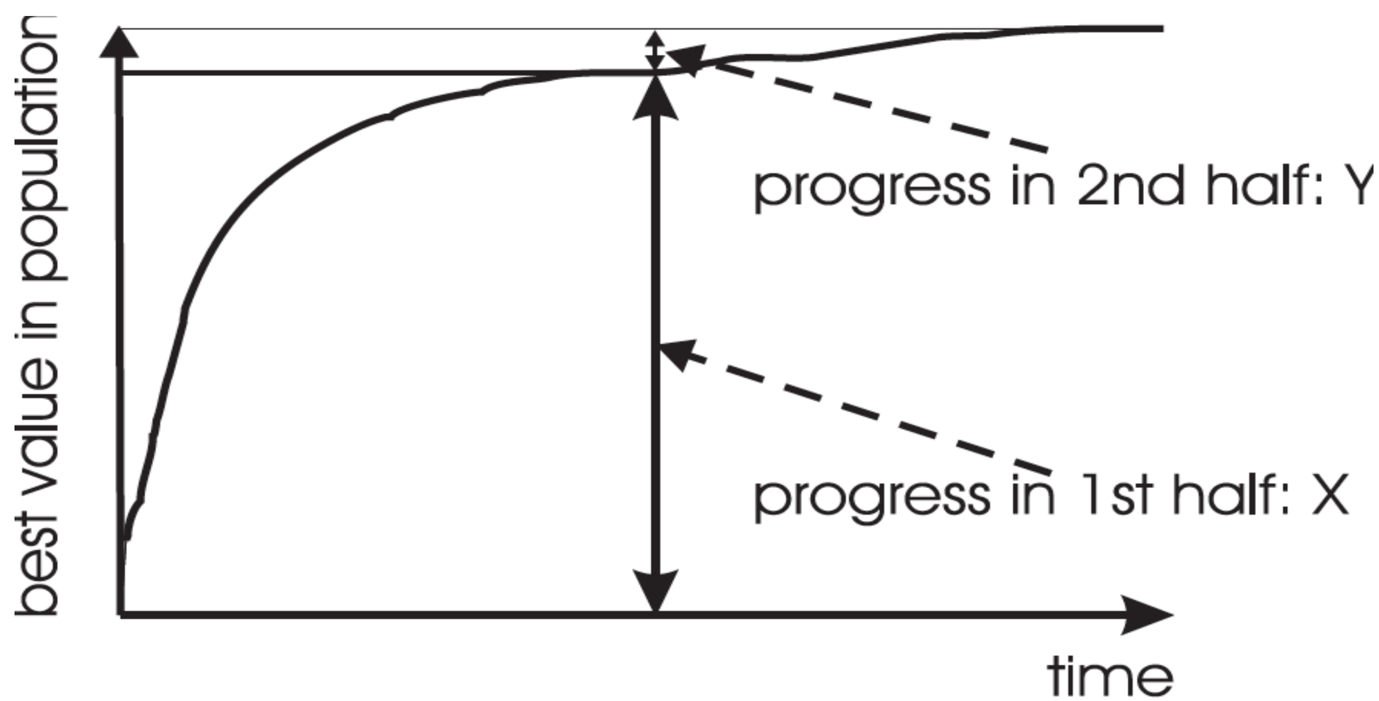
- Answer: it depends.
  - Possibly good, if good solutions/methods exist.

Components of an EA

Termination

```
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
```

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

Termination Condition

- 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

Contents

- What is Evolution?
- History of Evolutionary Computation
- Evolutionary Algorithms