UiO : Department of Informatics
University of Oslo

INF3490 - Biologically inspired computing

Lecture 2: Eiben and Smith, chapter 1-4

Evolutionary Algorithms - Introduction and representation

Kai Olav Ellefsen

UiO : Department of Informatics
University of Oslo

Evolution

- **Biological evolution:**
  - Lifeforms adapt to a particular environment over successive generations.
  - Combinations of traits that are better adapted tend to increase representation in population.
  - Mechanisms: Variation (Crossover, Mutation) and Selection (Survival of the fittest).
- **Evolutionary Computing (EC):**
  - Mimic the biological evolution to optimize solutions to a wide variety of complex problems.
  - In every new generation, a new set of solutions is created using bits and pieces of the fittest of the old.

UiO : Department of Informatics
University of Oslo

Evolution in Nature

- A population of individuals exists in an environment with limited resources
- ***Competition*** for resources causes selection of ***fitter*** individuals that are better adapted to the environment
- These individuals act as seeds for the generation of new individuals through **recombination and mutation**
- Over time ***Natural selection*** causes a rise in the fitness of the population

11

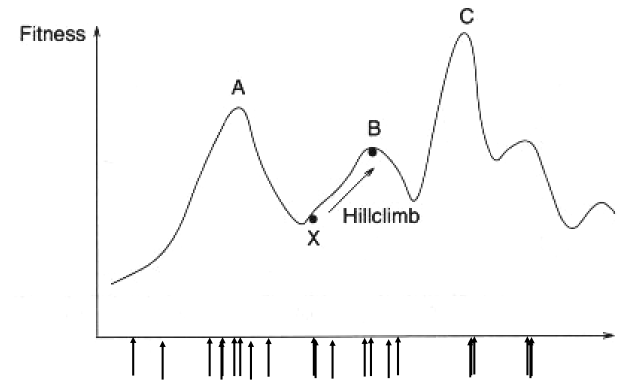
UiO : Department of Informatics
University of Oslo

Evolutionary Algorithms (EAs)

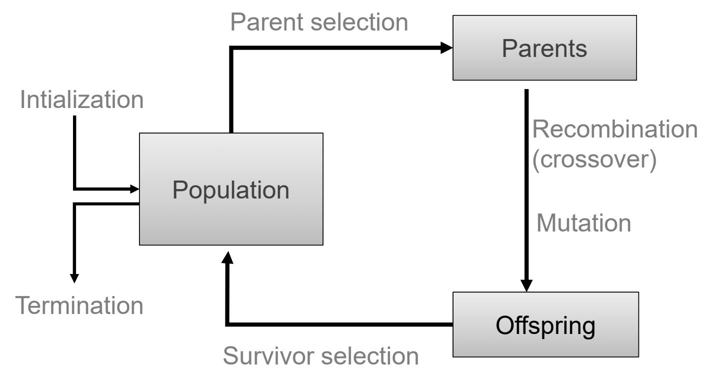
- EAs fall into the category of “generate and test” algorithms
- They are stochastic, population-based algorithms
- **Variation** operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty
- **Selection** reduces diversity and acts as a force pushing quality

12

Hillclimbing Problem in Search



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:
Two pillars of evolution

There are two competing forces

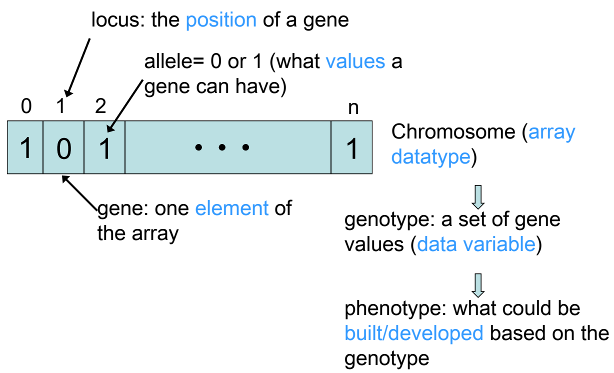
Increasing population **diversity** by genetic operators
- mutation
- recombination

Push towards **novelty**

Decreasing population **diversity** by selection
- of parents
- of survivors

Push towards **quality**

Representation: EA terms



UiO Department of Informatics
University of Oslo

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 : Genetic Algorithms (GA)
  - Real-valued vectors : Evolution Strategies (ES)
  - Finite state Machines: Evolutionary Programming (EP)
  - LISP trees: Genetic Programming (GP)
- These differences are largely irrelevant, best strategy
  - choose representation to suit problem
  - choose variation operators to suit representation

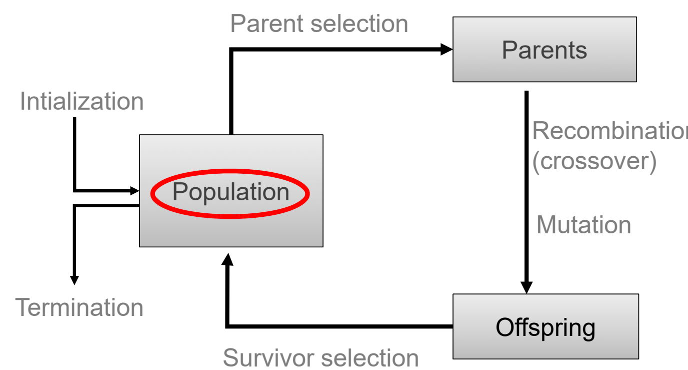
UiO Department of Informatics
University of Oslo

Main EA components: Evaluation (fitness) function

- Represents the task to solve
- Enables selection (provides basis for comparison)
- Assigns a single real-valued fitness to each phenotype

UiO Department of Informatics
University of Oslo

General scheme of EAs

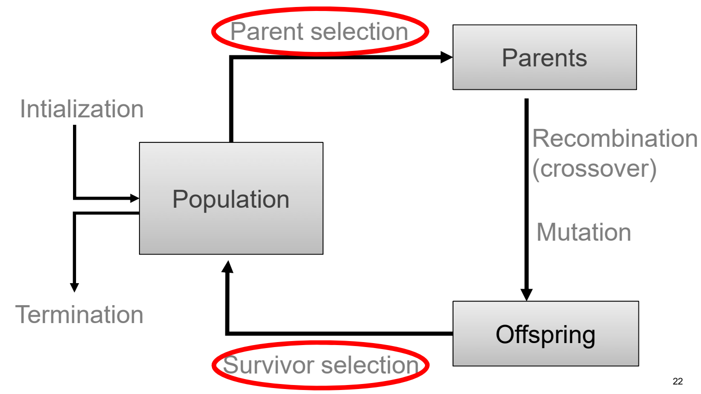


Main EA components: Population

- The **candidate solutions** of the problem
- A population is a *multiset* of individuals
- 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**

UiO Department of Informatics
University of Oslo

General scheme of EAs



UiO Department of Informatics
University of Oslo

Main EA components: Selection mechanism (1/3)

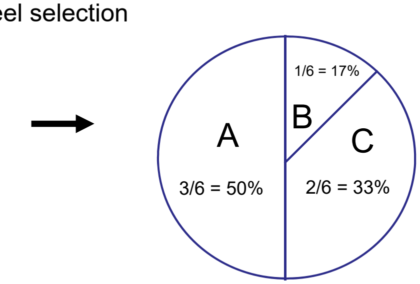
- Identifies individuals
  - to become parents
  - to survive
- Pushes population towards higher fitness
- Parent selection is usually probabilistic
  - high quality solutions more likely to be selected than low quality, but not guaranteed
  - This *stochastic* nature can aid escape from local optima

UiO Department of Informatics
University of Oslo

Main EA components: Selection mechanism (2/3)

Example: roulette wheel selection

fitness(A) = 3
fitness(B) = 1
fitness(C) = 2

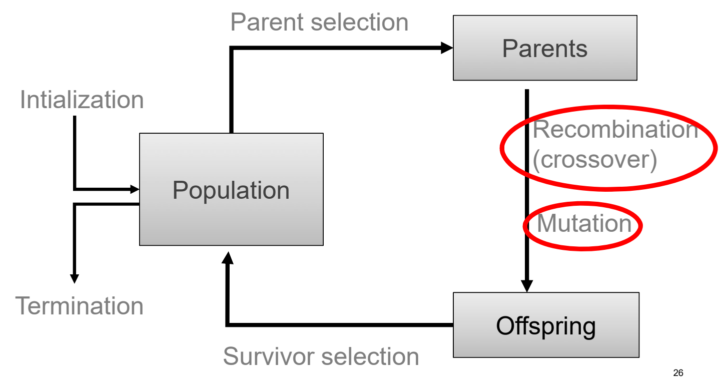


Main EA components: Selection mechanism (3/3)

Survivor selection:

- N parents + K offspring -> N individuals (new population)
- Often deterministic:
  - **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)

General scheme of EAs



Main EA components: Variation operators

- Role: to generate new candidate solutions
- Usually divided into two types according to their **arity** (number of inputs to the variation operator):
  - Arity 1 : **mutation** operators
  - Arity >1 : **recombination** operators
  - Arity = 2 typically called **crossover**
  - Arity > 2 is formally possible, seldom used in EC

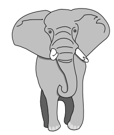
Main EA components: Mutation (1/2)

- Role: cause small, random variance to a genotype
- 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 – the only search operator
  - GP – hardly used

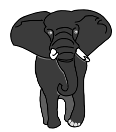
UiO Department of Informatics
University of Oslo

Main EA components: Mutation (2/2)

before 1 1 1 1 1 1 1



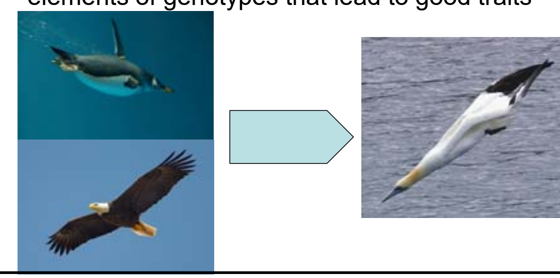
after 1 1 1 0 1 1 1



UiO Department of Informatics
University of Oslo

Main EA components: Recombination (1/2)

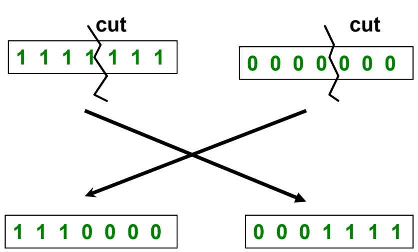
- Role: merges information from parents into offspring
- Choice of what information to merge is stochastic
- Hope is that some offspring are better by combining elements of genotypes that lead to good traits

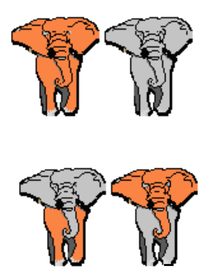


UiO Department of Informatics
University of Oslo

Main EA components: Recombination (2/2)

Parents

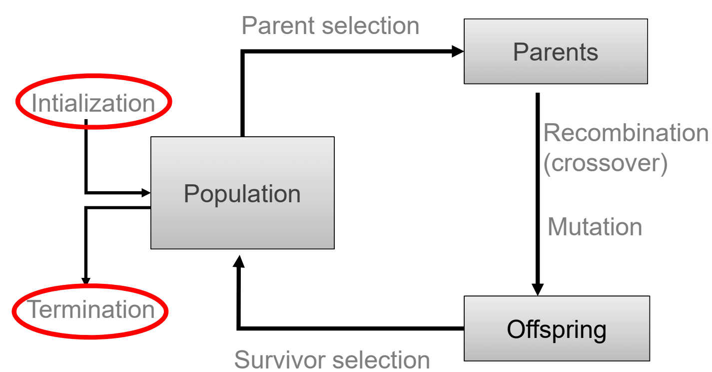




Offspring

UiO Department of Informatics
University of Oslo

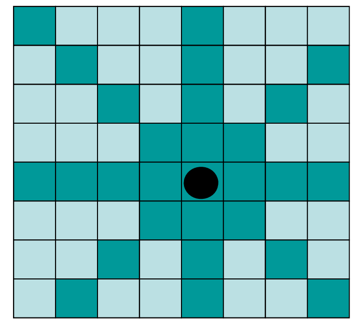
General scheme of EAs



Main EA components: Initialisation / Termination

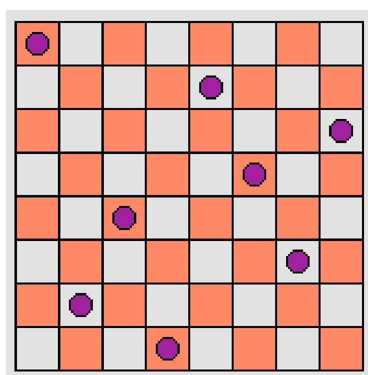
- Initialisation usually done at random,
  - Need to ensure even spread and mixture of possible allele values
  - Can include existing solutions, or use problem-specific heuristics, to “seed” the population
- Termination condition checked every generation
  - Reaching some (known/hoped for) fitness
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

Example: The 8-queens problem



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

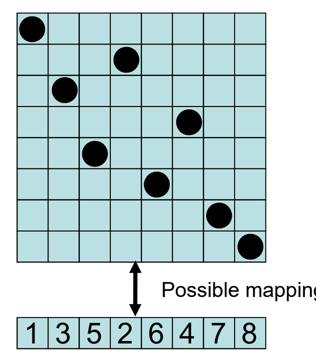
Example: The 8-queens problem – one solution



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

UiO : Department of Informatics
University of Oslo

The 8-queens problem: Mutation

Small variation in one permutation, e.g.:
- swapping values of two randomly chosen positions,

1 3 5 2 6 4 7 8 → 1 3 7 2 6 4 5 8

UiO : Department of Informatics
University of Oslo

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

1 3 5 | 2 6 4 7 8 → 1 3 5 4 2 8 7 6
8 7 6 | 5 4 3 2 1 → 8 7 6 2 4 1 3 5

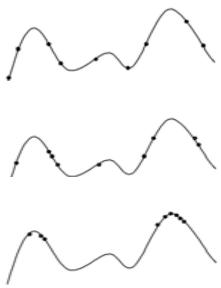
UiO : Department of Informatics
University of Oslo

The 8-queens problem: Selection

- Parent selection:
  - Pick 5 random parents and take best 2 to undergo crossover
- Survivor selection (replacement)
  - Merge old (parents) and new (offspring) population
  - Throw out the 2 worst solutions

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

41

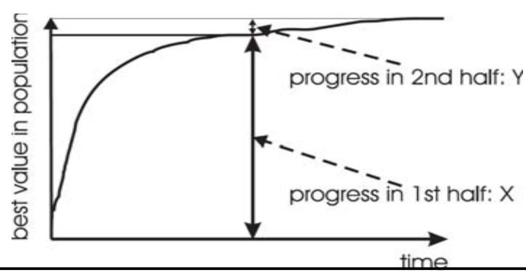
Typical EA behaviour: Typical run: progression of fitness

Typical run of an EA shows so-called "anytime behavior"

42

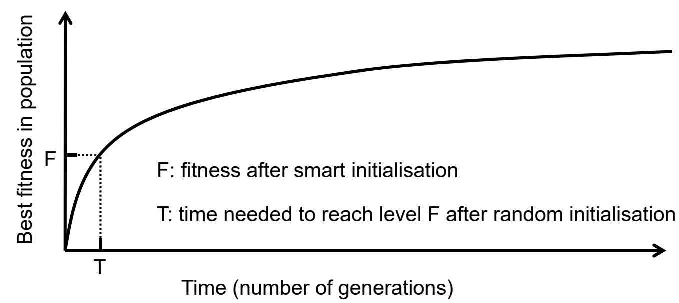
Typical EA behaviour: Are long runs beneficial?

- Answer:
  - It depends on how much you want the last bit of progress



43

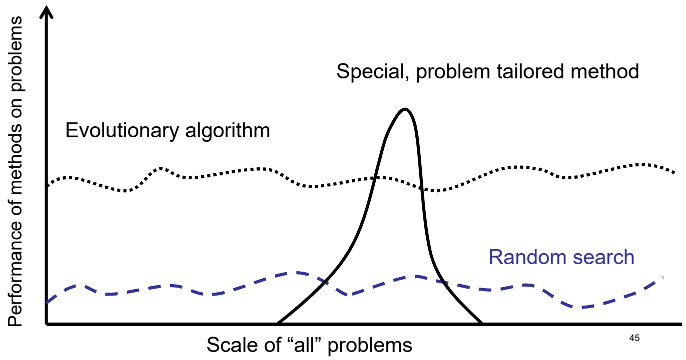
Typical EA behaviour: Is it worth expending effort on smart initialisation?



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

44

Traditional View on EA Performance



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

Chapter 4: Representation, Mutation, and Recombination

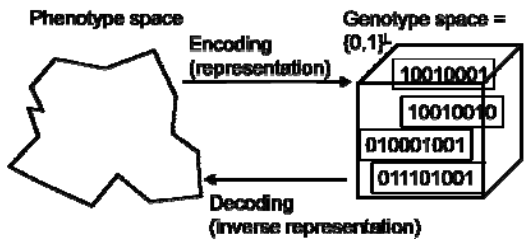
- Role of **representation** and **variation operators**
- Most common representation of genomes:
  - Binary
  - Integer
  - Real-Valued or Floating-Point
  - Permutation
  - Tree

Role of representation and variation operators

- First stage of building an EA and most difficult one: choose *right* representation for the problem
- Type of variation operators needed depends on chosen representation
- TSP problem
  - What are possible representations?

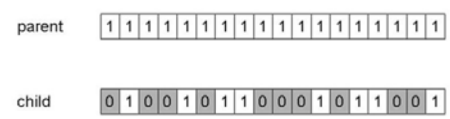
Binary Representation

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



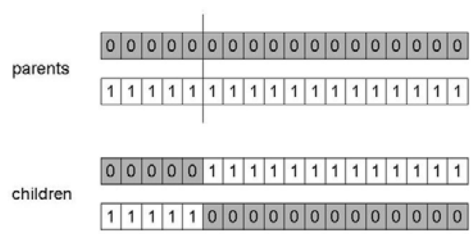
Binary Representation: Mutation

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



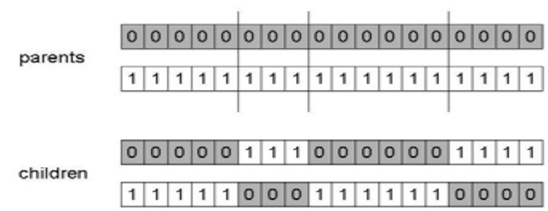
Binary Representation: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails



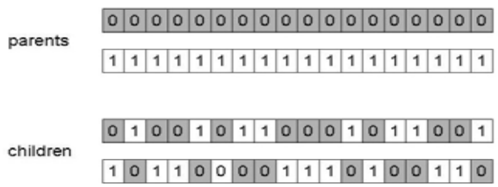
Binary Representation: n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents



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



UiO Department of Informatics
University of Oslo

Binary Representation: Crossover OR mutation? (1/3)

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

UiO Department of Informatics
University of Oslo

Binary Representation: Crossover OR mutation? (2/3)

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

UiO Department of Informatics
University of Oslo

Binary Representation: Crossover OR mutation? (3/3)

There is co-operation AND competition between them:

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

Integer Representation

- Some problems naturally have integer variables,
  - e.g. image processing parameters
- Others take categorical values from a fixed set
  - e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make:
  - "creep" i.e. more likely to move to similar value
    - Adding a small (positive or negative) value to each gene with probability p.
  - Random resetting (esp. categorical variables)
    - With probability p_m a new value is chosen at random

Real-Valued or Floating-Point Representation: Uniform Mutation

- General scheme of floating point mutations

$$\bar{x} = \langle x_1, \ldots, x_l \rangle \rightarrow \bar{x}' = \langle x'_1, \ldots, x'_l \rangle$$

$$x_i, x'_i \in [LB_i, UB_i]$$

- **Uniform Mutation**

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

  - Analogous to bit-flipping (binary) or random resetting (integers)

Real-Valued or Floating-Point Representation: Nonuniform Mutation

- Non-uniform mutations:
  - Most common method is to 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))

Real-Valued or Floating-Point Representation: Crossover operators

- Discrete recombination:
  - 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 recombination:
  - 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: $\alpha = 0.5$ -> uniform arithmetical crossover
    - variable (e.g. depend on the age of the population)
    - picked at random every time

Real-Valued or Floating-Point Representation: Simple arithmetic crossover

- Parents: $\langle x_1,\dots,x_n \rangle$ and $\langle y_1,\dots,y_n \rangle$
- Pick a random gene (k) after this point mix values
- child$_1$ is:

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

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

→

0.1	0.2	0.3	0.4	0.5	0.6	0.5	0.5	0.6
0.3	0.2	0.3	0.2	0.3	0.2	0.5	0.5	0.6

Real-Valued or Floating-Point Representation: Single arithmetic crossover

- Parents: $\langle x_1,\dots,x_n \rangle$ and $\langle y_1,\dots,y_n \rangle$
- Pick a single gene (k) at random,
- child$_1$ is: $\left\langle x_1, \dots, x_k, \alpha \cdot y_k + (1-\alpha)\cdot x_k, \dots, x_n \right\rangle$
- Reverse for other child. e.g. with $\alpha = 0.5$

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

→

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.5	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.5	0.3

Real-Valued or Floating-Point Representation: Whole arithmetic crossover

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

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

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

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

→

0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.6
0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.6

Permutation Representations

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

Permutation Representation: TSP example

- 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:

  for 30 cities there are 30! ≈ 10^{32} possible tours

Permutation Representations: Mutation

- Normal mutation operators lead to inadmissible solutions
  - Mutating a single gene destroys the permutation
- 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

Permutation Representations: Swap mutation

- Pick two alleles at random and swap their positions

1 2 3 4 5 6 7 8 9 → 1 5 3 4 2 6 7 8 9

Permutation Representations: 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

1 2 3 4 5 6 7 8 9 → 1 2 5 3 4 6 7 8 9

Permutation Representations: Scramble mutation

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

1 2 3 4 5 6 7 8 9 → 1 3 5 4 2 6 7 8 9

Permutation Representations: 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

1 2 3 4 5 6 7 8 9 → 1 5 4 3 2 6 7 8 9

Permutation Representations: Crossover operators

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

1 2 3 | 4 5 → 1 2 3 2 1

5 4 3 | 2 1 → 5 4 3 4 5

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

Permutation Representations: Partially Mapped Crossover (PMX) (1/2)

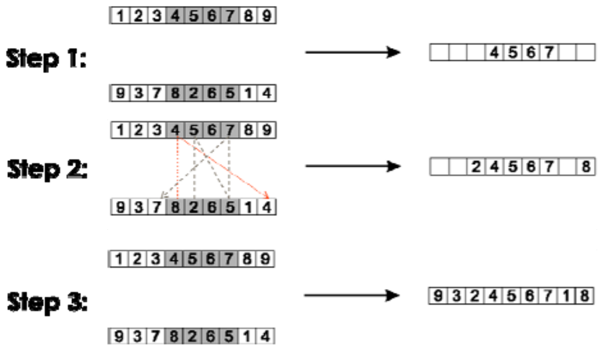
Informal procedure for parents P1 and P2:

1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied *j* in P2, since we know that we will not be putting *j* there (as is already in offspring)
5. If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *i* in the position occupied by *k* in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

UiO Department of Informatics
University of Oslo

Permutation Representations: Partially Mapped Crossover (PMX) (2/2)



73

UiO Department of Informatics
University of Oslo

Permutation Representations: Edge Recombination (1/3)

- Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark with a +
- e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

74

UiO Department of Informatics
University of Oslo

Permutation Representations: Edge Recombination (2/3)

Informal procedure: once edge table is constructed

1. Pick an initial element, *entry*, at random and put it in the offspring
2. Set the variable *current element* = *entry*
3. Remove all references to *current element* from the table
4. Examine list for current element:
   - If there is a common edge, pick that to be next element
   - Otherwise pick the entry in the list which itself has the shortest list
   - Ties are split at random
5. In the case of reaching an empty list:
   - a new element is chosen at random

75

UiO Department of Informatics
University of Oslo

Permutation Representations: Edge Recombination (3/3)

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Choices	Element selected	Reason	Partial result
All	1	Random	[*1*]
2,5,4,9	5	Shortest list	[*1 5*]
4,6	6	Common edge	[*1 5 6*]
2,7	2	Random choice (both have two items in list)	[*1 5 6 2*]
3,8	8	Shortest list	[*1 5 6 2 8*]
7,9	7	Common edge	[*1 5 6 2 8 7*]
3	3	Only item in list	[*1 5 6 2 8 7 3*]
4,9	9	Random choice	[*1 5 6 2 8 7 3 9*]
4	4	Last element	[*1 5 6 2 8 7 3 9 4*]

UiO Department of Informatics
University of Oslo

Permutation Representations: Order crossover (1/2)

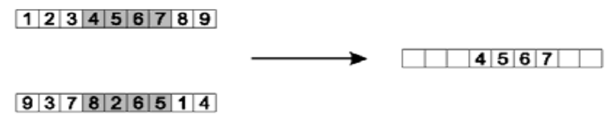
- Idea is to preserve relative order that elements occur
- Informal procedure:
  - 1. Choose an arbitrary part from the first parent
  - 2. Copy this part to the first child
  - 3. Copy the numbers that are not in the first part, to the first child:
    - starting right from cut point of the copied part,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

77

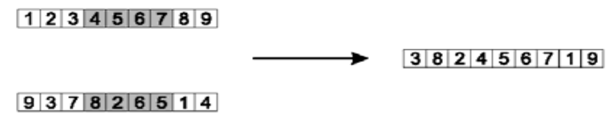
UiO Department of Informatics
University of Oslo

Permutation Representations: Order crossover (2/2)

- Copy randomly selected set from first parent



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



78

UiO Department of Informatics
University of Oslo

Permutation Representations: Cycle crossover (1/2)

Basic idea:

Each allele comes from one parent *together with its position*.

Informal procedure:

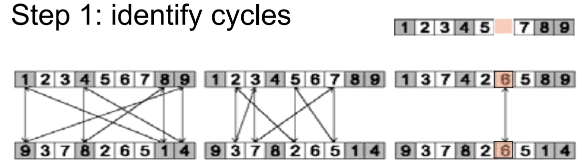
1. Make a cycle of alleles from P1 in the following way.
   - (a) Start with the first allele of P1.
   - (b) Look at the allele at the *same position* in P2.
   - (c) Go to the position with the *same allele* in P1.
   - (d) Add this allele to the cycle.
   - (e) Repeat step b through d until you arrive 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

79

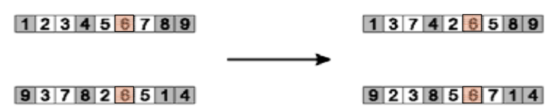
UiO Department of Informatics
University of Oslo

Permutation Representations: Cycle crossover (2/2)

- Step 1: identify cycles



- Step 2: copy **alternate** cycles into offspring



80

UiO Department of Informatics
University of Oslo

Tree Representation (1/5)

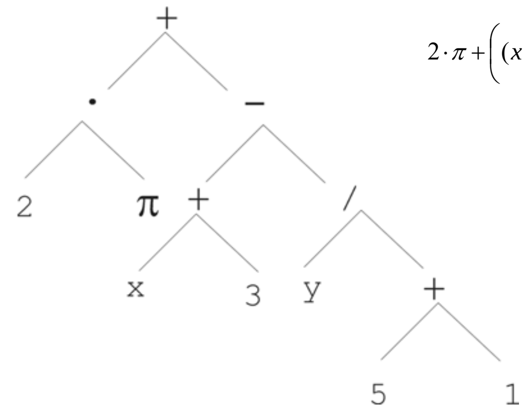
- Trees are a universal form, e.g. consider
- Arithmetic formula: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula: (x ∧ true) → ((x ∨ y) ∨ (z ↔ (x ∧ y)))
- Program:

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

81

UiO Department of Informatics
University of Oslo

Tree Representation (2/5)

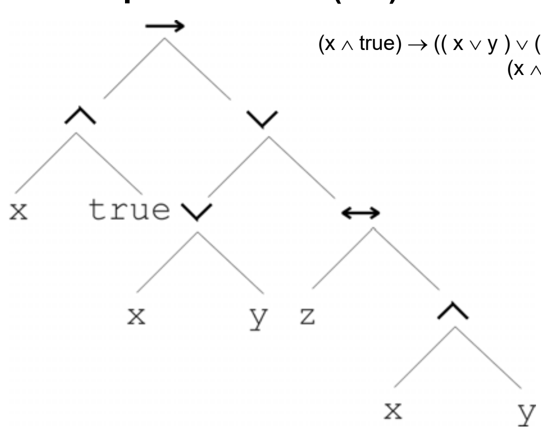


$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$

82

UiO Department of Informatics
University of Oslo

Tree Representation (3/5)

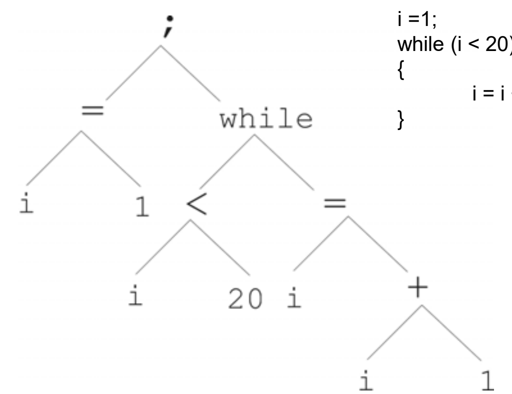


(x ∧ true) → ((x ∨ y) ∨ (z ↔
(x ∧ y)))

83

UiO Department of Informatics
University of Oslo

Tree Representation (4/5)




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

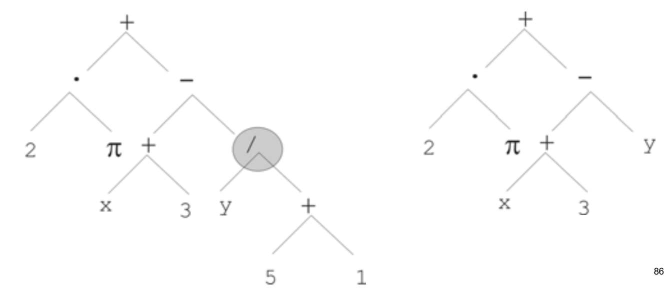
84

Tree Representation (5/5)

- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations)
- Tree shaped chromosomes are non-linear structures
- In GA, ES, EP the size of the chromosomes is fixed
- Trees in GP (Genetic Programming) may vary in depth and width

Tree Representation: Mutation (1/2)

- Most common mutation: replace randomly chosen subtree by randomly generated tree



Tree Representation: Mutation (2/2)

- Mutation has two parameters:
  - Probability p_m to choose mutation
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

Tree Representation: Recombination (1/2)

- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents

UiO Department of Informatics
University of Oslo

Tree Representation: Recombination (2/2)