Genetic Algorithms

Genetic Algorithms

- **Genetic Algorithms** (GAs) were **developed by Prof. John Holland** and his students at the University of Michigan during the 1960s and 1970s.

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional
Survival selection	Generational

Table 3.1. Sketch of the simple GA

Genetic Algorithms

- Maximizing the values of **x^2** for x in the range **0-31**.

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional
Survival selection	Generational

$Prob_i = f_i / \sum f_j$

Table 3.1. Sketch of the simple GA

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

Genetic Algorithms

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

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

Genetic Algorithms

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

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

20

625

25

729

27

400	784
2538	
634.5	

Representation of Individuals

1. **Binary Representations**

2. **Integer Representations**

3. **Real-Valued or Floating-Point**

Representation 4. Permutation Representations

Mutation

1. Mutation for Binary Representations

Mutation Operators for Integer Representations

2. Mutation Operators for Integer Representations

Random Resetting

• "Bit-flipping" mutation of binary encodings is extended to "random resetting"

• With probability Pm a new value is chosen at random from the set of permissible values in each position.

Creep Mutation

• Tended to make small changes relative to the range of permissible values.

• Designed for ordinal attributes and works by adding a small (positive or negative) value to each gene with probability p.

Mutation Operators for Floating-Point Representations

2. Mutation Operators for Floating-Point Representations

Change the allele value of **each gene randomly within its domain** given by a lower *Li* and upper *Ui* bound, resulting in the following transformation:

$$\langle x_1, \ldots, x_n \rangle \rightarrow \langle x'_1, \ldots, x'_n \rangle, \quad \text{where} \quad x_i, x'_i \in [L_i, U_i].$$

Uniform Mutation

• The values of x' are drawn **uniformly randomly** from *[Li, Ui]*

• Analogous to bit-flipping for binary encodings and the random resetting sketched for integer encodings.

• Position wise mutation probability

Mutation Operators for Floating-Point Representations

2. Mutation Operators for Floating-Point Representations

Change the allele value of **each gene randomly within its domain** given by a lower *Li* and upper *Ui* bound, resulting in the following transformation:

$$\langle x_1, \ldots, x_n \rangle \rightarrow \langle x'_1, \ldots, x'_n \rangle, \quad \text{where} \quad x_i, x'_i \in [L_i, U_i].$$

Non-uniform Mutation with a Fixed Distribution • Analogous to the creep mutation.

• Adding to the current gene value an amount drawn randomly from a Gaussian distribution with mean zero and user-specified standard deviation, and then curtailing the resulting value to the range [Li, Ui] if necessary.

Mutation Operators for Permutation Representations

- Swap Mutation • Insert Mutation

1 2 3 4 5 6 7 8 9 → 1 5 3 4

1 2 3 4 5 6 7 8 9 → 1 2 5 3 4 6 7 8 9

- Scramble Mutation

1 2 3 4 5 6 7 8 9 → 1 3 5 4 2 6 7 8 9

- Inversion Mutation

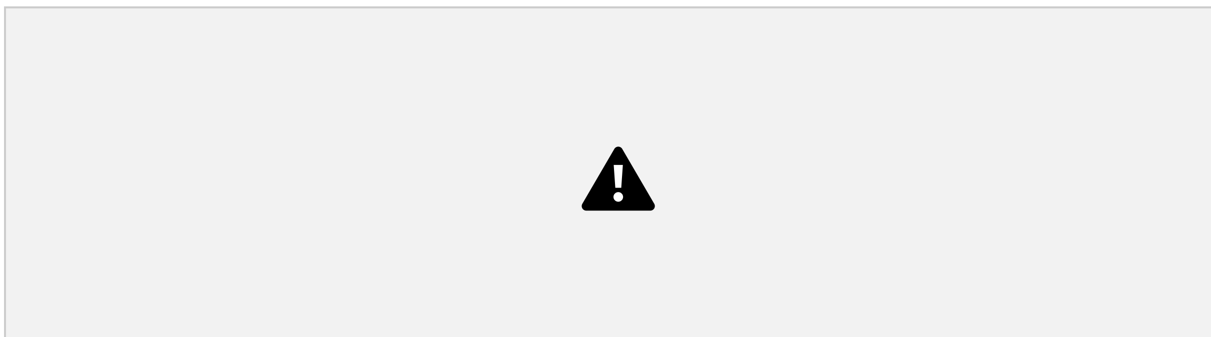
1 2 3 4 5 6 7 8 9 → 1 5 4 3 2 6 7 8 9

Recombination

- Recombination, the process whereby a new individual solution is created from the **information contained within two** (or more) **parent** solutions.

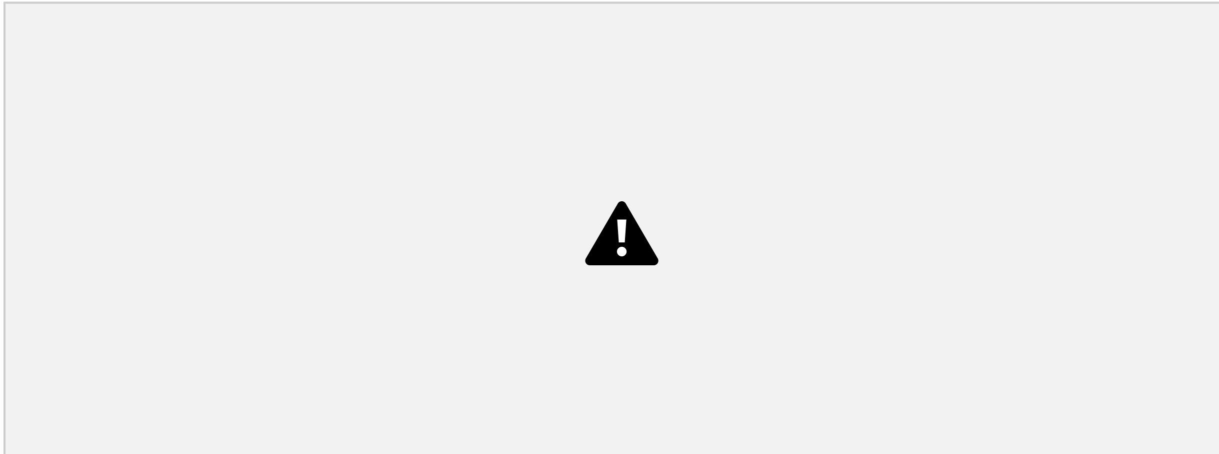
- **Recombination Operators for Binary Representations**

- One-Point Crossover



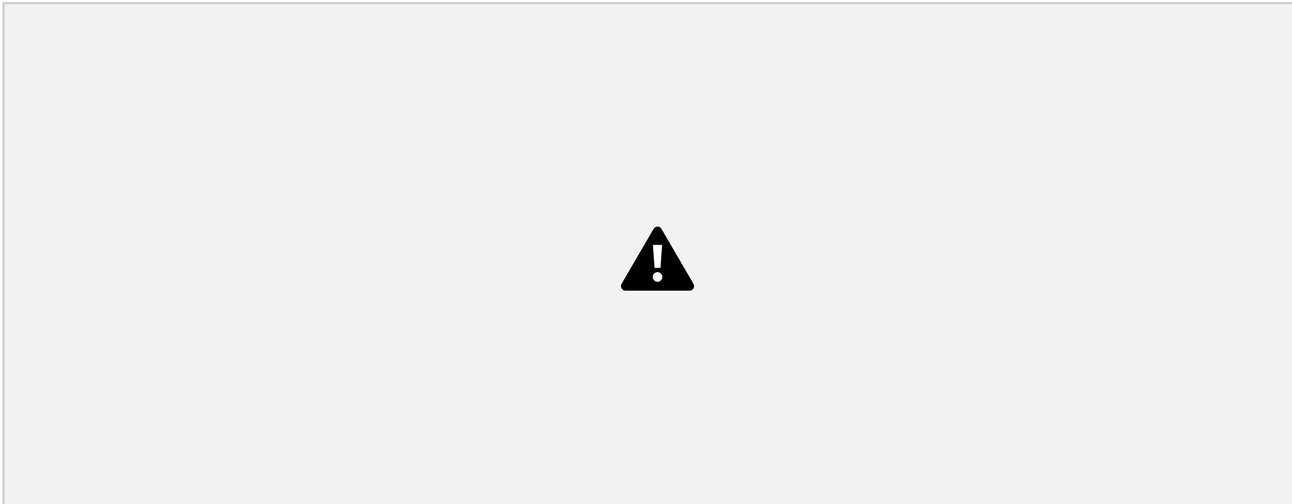
Recombination Operators for Binary Representations

- N-Point Crossover



Recombination Operators for Binary Representations

- Uniform Crossover
- In each position, if the **value is below a parameter p** (usually 0.5), the **gene is inherited from** the first parent; otherwise from the second. The second offspring is created using the **inverse mapping**.



Recombination Operators for Integer Representations

- Same set of operators as for binary representations.

Recombination Operators for Floating-Point Representations

- Arithmetic Recombination
  - Three types of arithmetic recombination

- Simple Recombination

Recombination Operators for Floating-Point Representations

- Single Arithmetic Recombination
  - Pick a **random allele k**. At that position, take the **arithmetic average of the two parents**.

Recombination Operators for Floating-Point Representations

- Whole Arithmetic Recombination

Recombination Operators for Permutation Representations

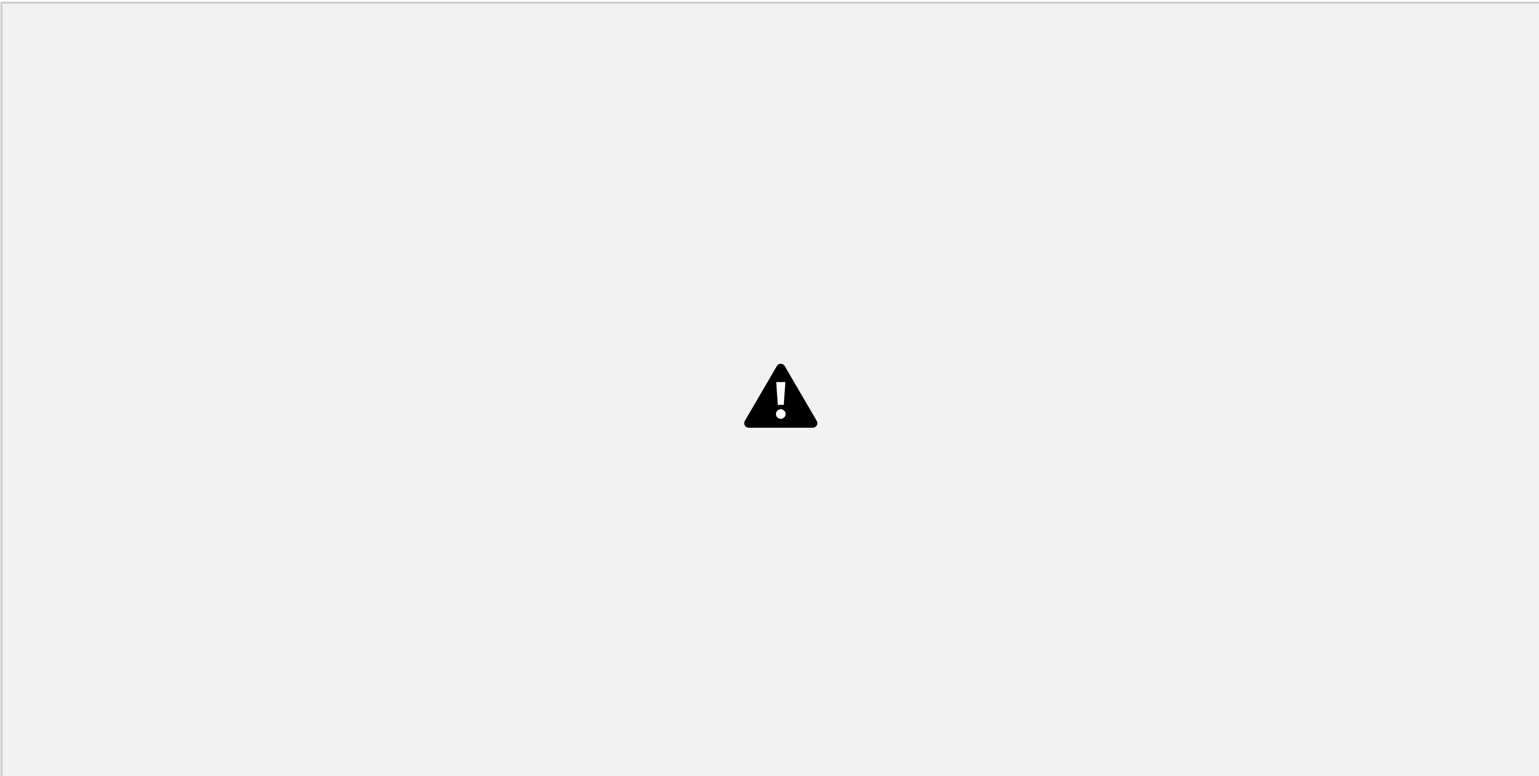
- Partially Mapped Crossover

Recombination Operators for Permutation Representations

- Partially Mapped Crossover

Recombination Operators for Permutation Representations

- Partially Mapped Crossover



Recombination Operators for Permutation Representations

- Partially Mapped Crossover

Recombination Operators for Permutation Representations

- **Edge Crossover**
- Edge crossover is based on the idea that an **offspring should be created as far as possible using only edges that are present in one or more parent**.

- Most commonly used version: edge-3 crossover after **Whitley**, which is designed to ensure that common edges are preserved.

Recombination Operators for Permutation Representations

- **Edge Crossover**

Recombination Operators for Permutation Representations

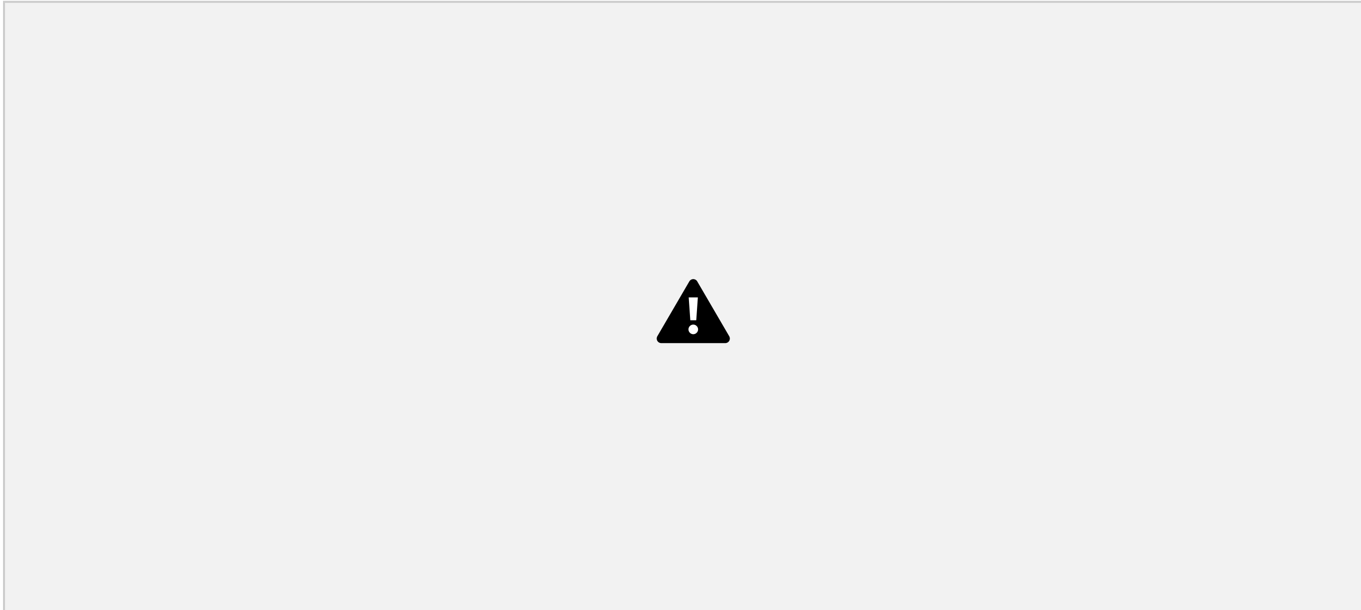
- **Edge Crossover**

  1. Let K be the empty list Let N be the first node of a random parent.

2. While Length(K) < Length(Parent):
    1. K := K, N (append N to K)
    2. Remove N from all neighbor lists
    3. If N's neighbor list is non-empty
    4. then let N* be the neighbor of N with the fewest neighbors in its list (or a random one, should there be multiple)
    5. else let N* be a randomly chosen node that is not in K
    6. N := N*

Recombination Operators for Permutation Representations

- **Edge Crossover** [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]



Recombination Operators for Permutation Representations

- **Edge Crossover** [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Recombination Operators for Permutation Representations

- **Edge Crossover CABDEF** and **ABCEFD**

Recombination Operators for Permutation

Representations

- **Edge Crossover CABDEF** and **ABCEFD**

Answer: ABDFCE

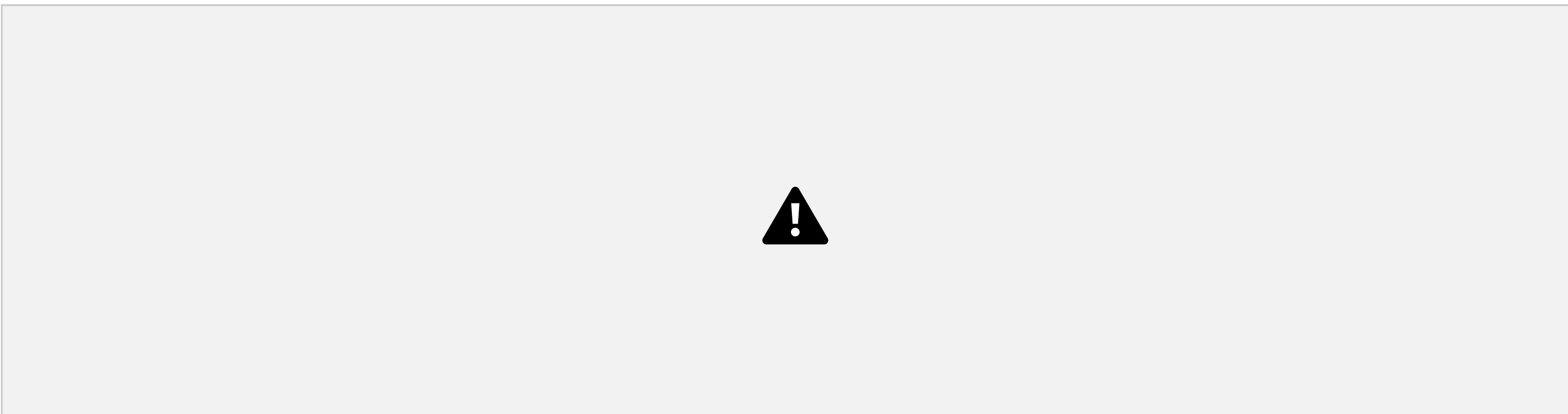
Recombination Operators for Permutation Representations

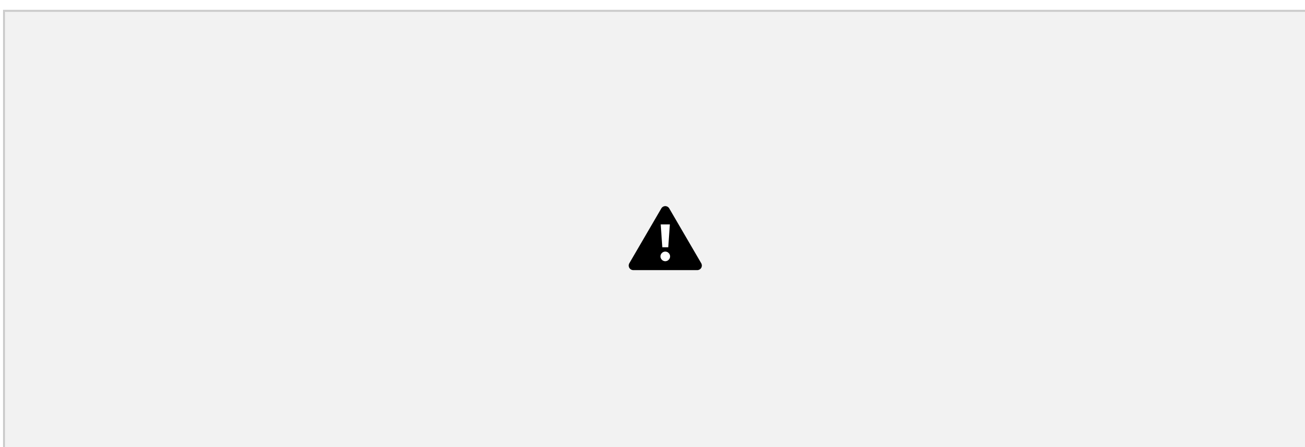
- **Order Crossover [D**esigned by Davis for order-based permutation]

- problems.]

Recombination Operators for Permutation Representations

- **Order Crossover**





Recombination Operators for Permutation Representations

- **Cycle Crossover**
- The operator works by **dividing the elements into cycles**. • A **cycle is a subset of elements** that has the property that **each element always occurs paired** with another element of the same cycle when the two parents are aligned.
- Having divided the permutation into cycles,
- The **offspring are created by selecting alternate cycles** from each parent.

Recombination Operators for Permutation Representations

- **Cycle Crossover**

- The procedure for constructing cycles is as follows:

Recombination Operators for Permutation Representations

- **Cycle Crossover**

3 8 5 6 7 1 4

1 2 3 4 5 6 7 8 9 9 3

1 3 7 4 2 6 5 8 9 9 2

7 8 2 6 5 1 4

Population Model

- **Generational model**

- In each generation we begin with a population of size from which a mating pool of parents is selected.
- Next, offspring are created from the mating pool by the application of variation operators, and evaluated.
- After each generation, the whole population is replaced by its offspring, which is called the "next generation" .

Population Model

- **Steady-state model**
- In the steady state model, the entire population is not changed at once, but rather a **part of it**.
- Select from **$(\lambda+\mu)$**

- Generational gap
  - If λ parents and μ offspring Generation gap = λ/μ

Parent Selection

- Fitness Proportional Selection
  - The selection probability depends on the absolute fitness value of the individual compared to the absolute fitness values of the rest of the population.
  - When fitness values are all very close together, there is almost no selection pressure.
  - Premature convergence.

Parent Selection

- Ranking Selection
  - It preserves a constant selection pressure by sorting the population on the basis of fitness, and then allocating selection probabilities to individuals according to their rank, rather than according to their actual fitness values.

Parent Selection

- Tournament Selection

Survivor Selection

- **Age-Based Replacement**

- Aged individuals will be changed
- **Fitness-Based Replacement**
  - fitness proportionate and tournament selection
  - Replace Worst (GENITOR)
  - Elitism

Credit Jason Lohn

NASA ST5 Mission had challenging requirements for antenna of 3 small spacecraft.

EA designs outperformed human expert ones and are nearly spacebound.

Credit Jason Lohn

Reference

- **Prof. Dr. A. E. Eiben, Dr. J. E. Smith auth. Introduction to Evolutionary Computing, Corrected second printing,Springer-ACM,2007**

Thank you