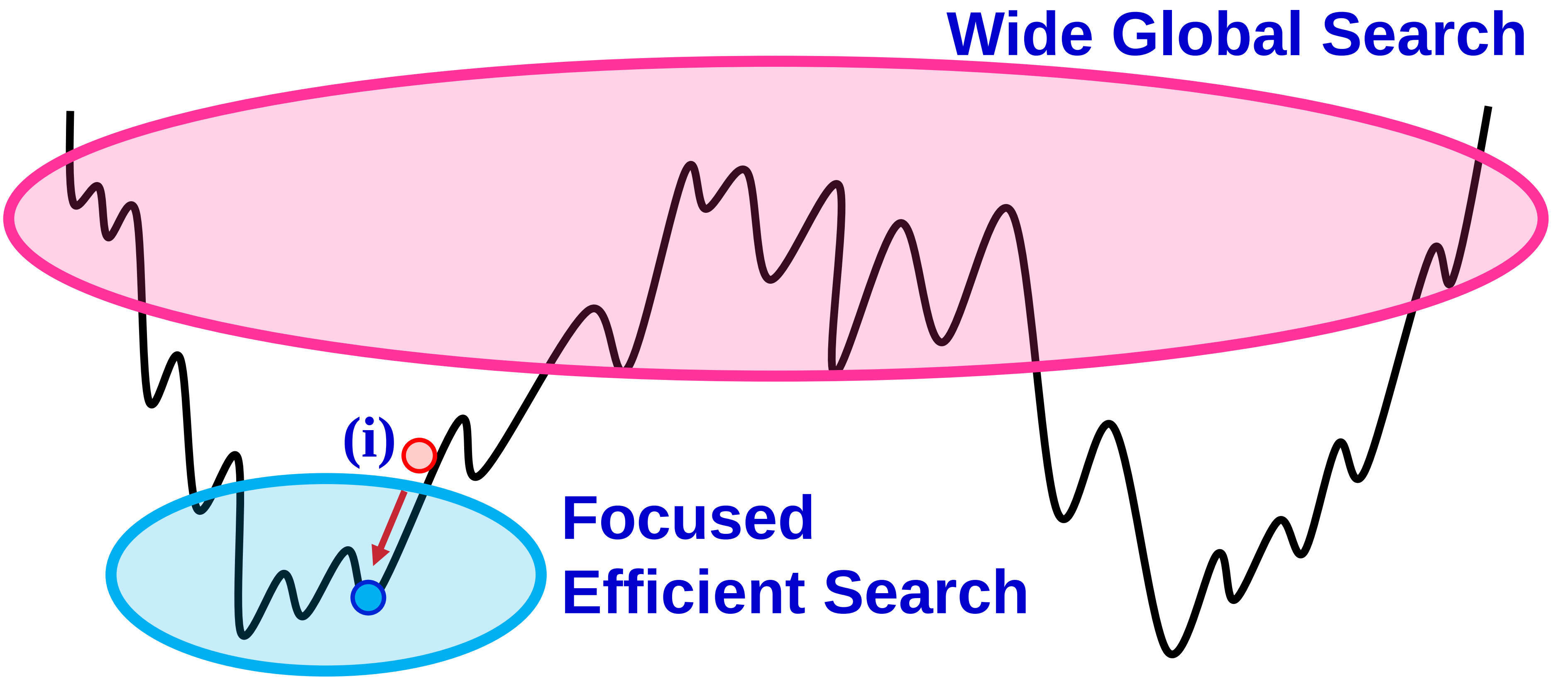
Optimization Methods

1. Introduction.
2. Greedy algorithms for combinatorial optimization.
3. LS and neighborhood structures for combinatorial optimization.
4. Variable neighborhood search, neighborhood descent, SA, TS, **EC**.
5. Branch and bound algorithms, and subset selection algorithms.
6. Linear programming problem formulations and applications.
7. Linear programming algorithms.
8. Integer linear programming algorithms.
9. Unconstrained nonlinear optimization and gradient descent.
10. Newton's methods and Levenberg-Marquardt modification.
11. Quasi-Newton methods and conjugate direction methods.
12. Nonlinear optimization with equality constraints.
13. Nonlinear optimization with inequality constraints.
14. Problem formulation and concepts in multi-objective optimization.
15. Search for single final solution in multi-objective optimization.
16: Search for multiple solutions in multi-objective optimization.

Optimization Algorithm Design:

Find a good balance between the wide global search and the focused efficient search (the good balance depends on the problem size and the available computation time)

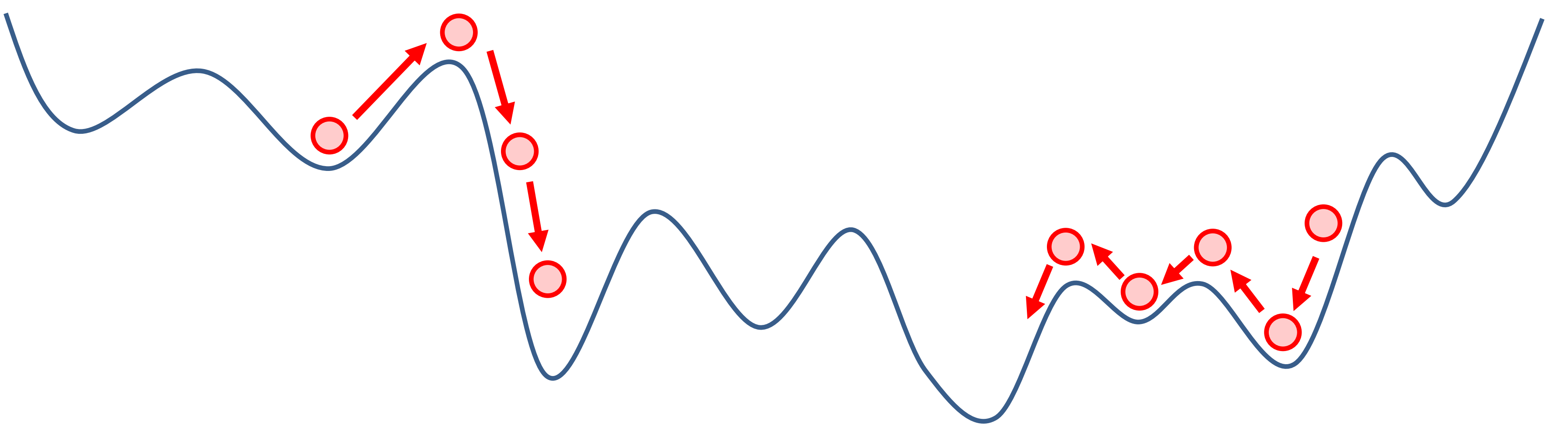


Move to a Better Solution

- Local Search (LS)
- Iterated Local Search (ILS)
- Variable Neighborhood Search (VNS)

Allow the Move to a Worse Solution

- Simulated Annealing (SA)
- Tabu Search (TS)



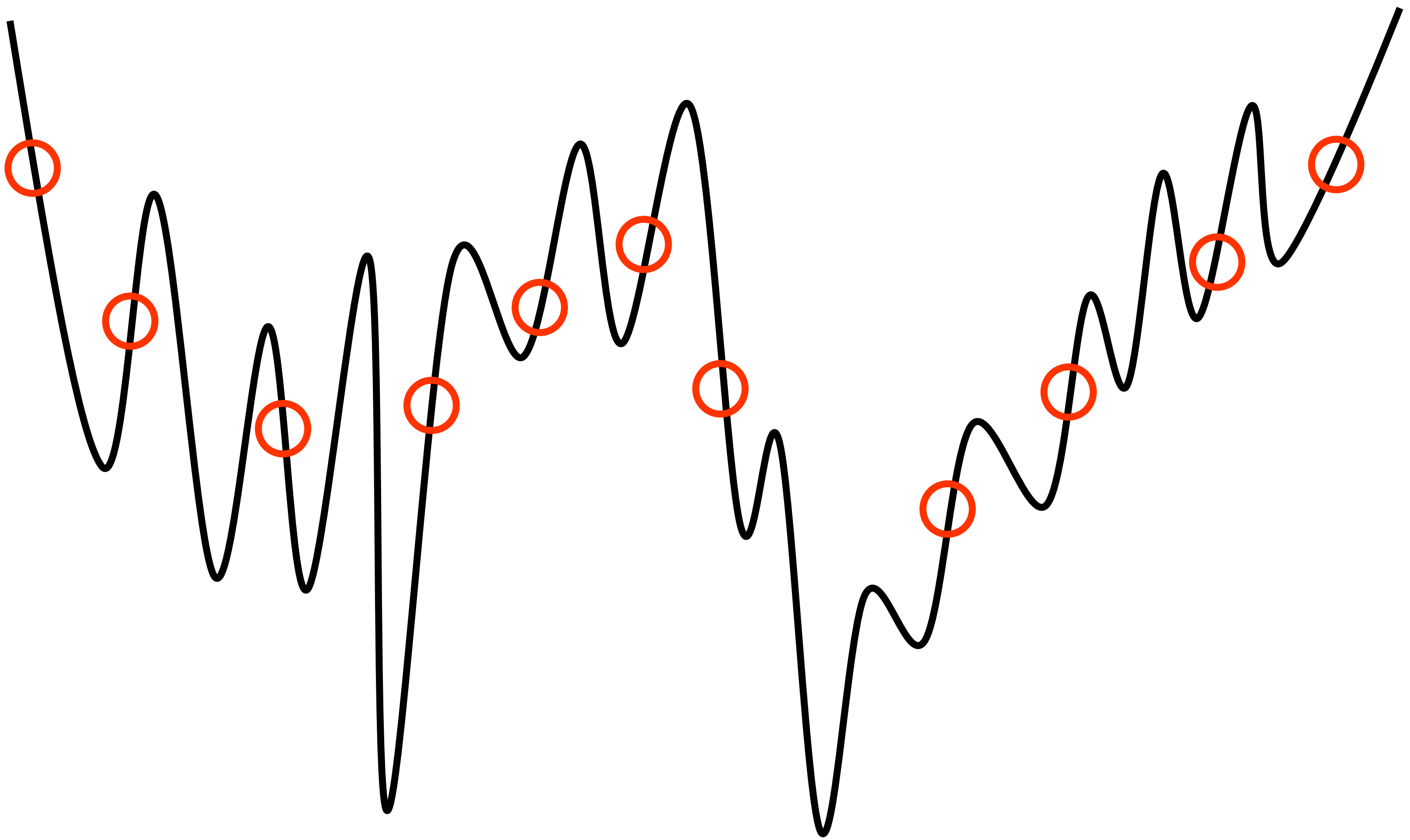
Point-based algorithms: Almost all algorithms

LS (Local Search), Iterated LS,

Variable Neighborhood Search, SA, TS

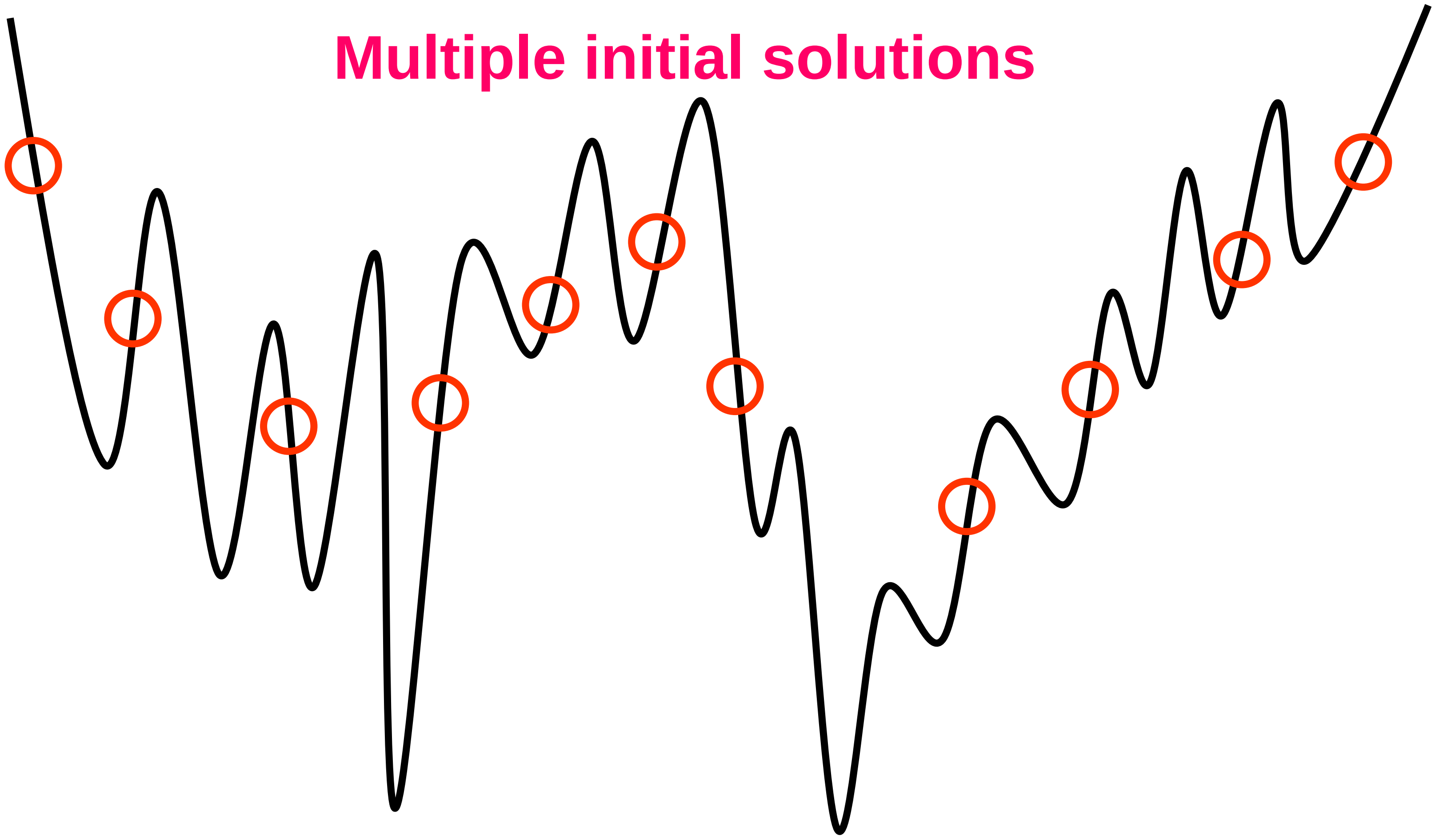
Genetic Algorithms: Population-based search

(multi-point search)



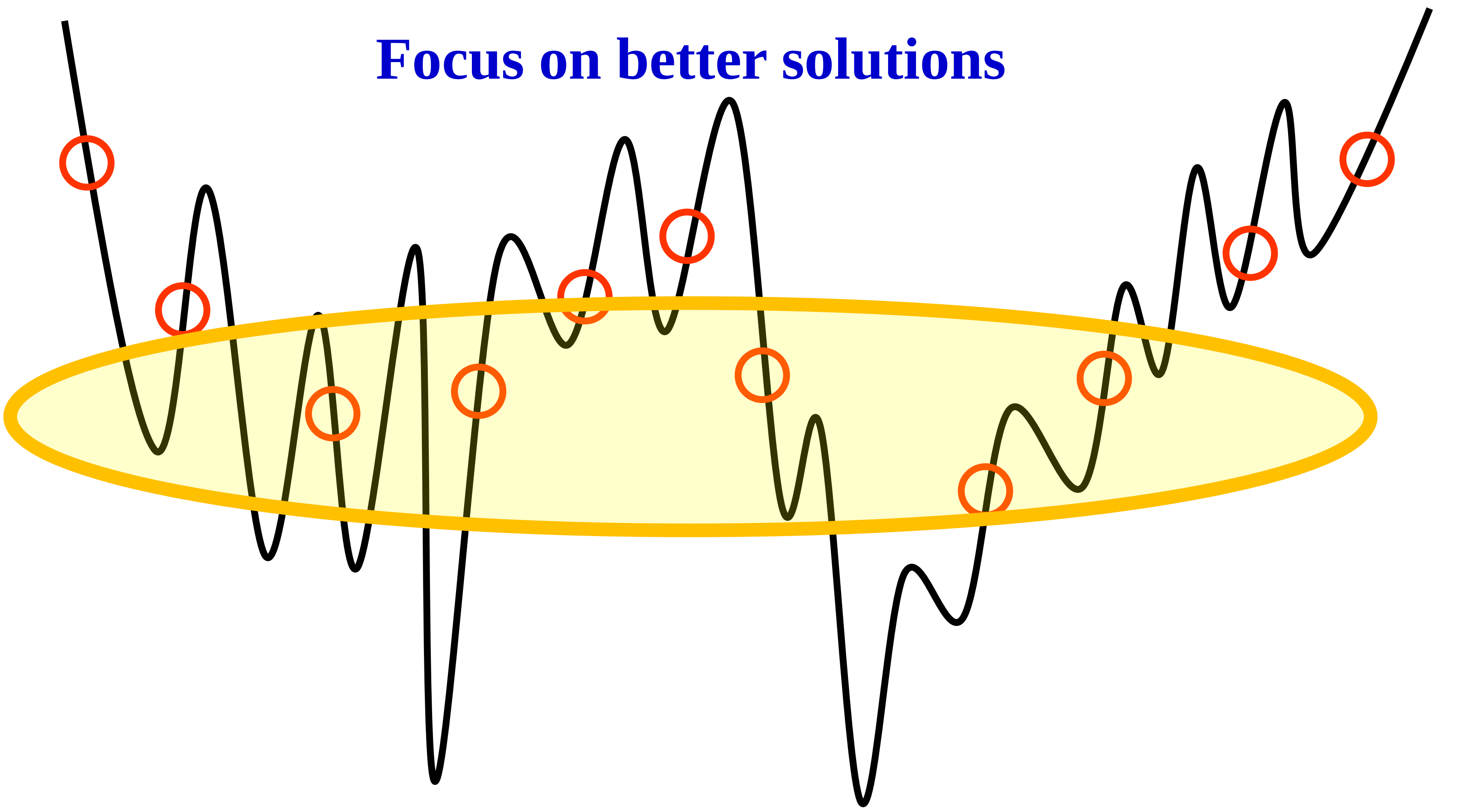
Evolutionary Algorithms (e.g., Genetic Algorithms)

- Population-based search algorithms
- Multi-point search algorithms



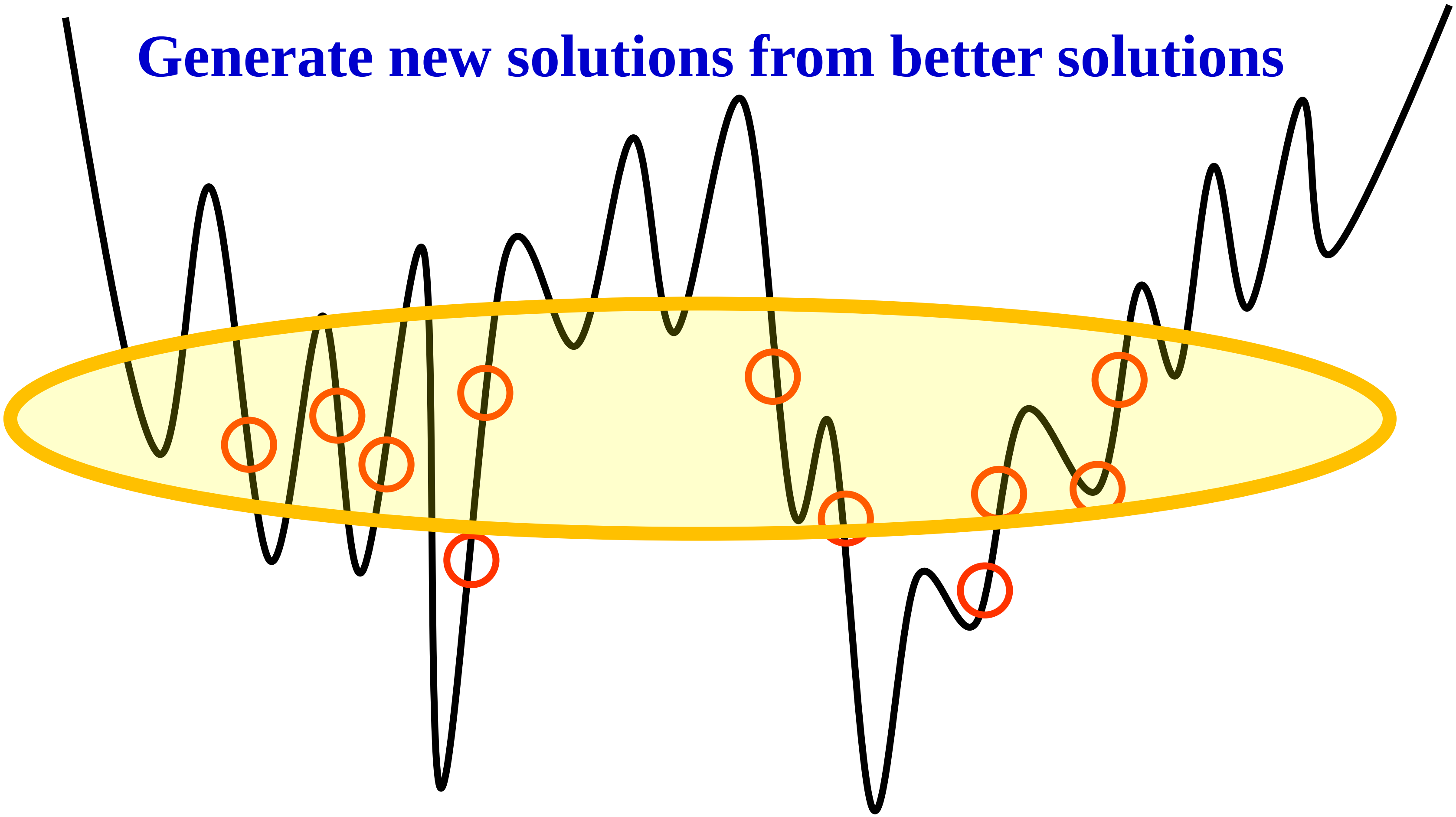
Evolutionary Algorithms (e.g., Genetic Algorithms)

- Population-based search algorithms
- Multi-point search algorithms



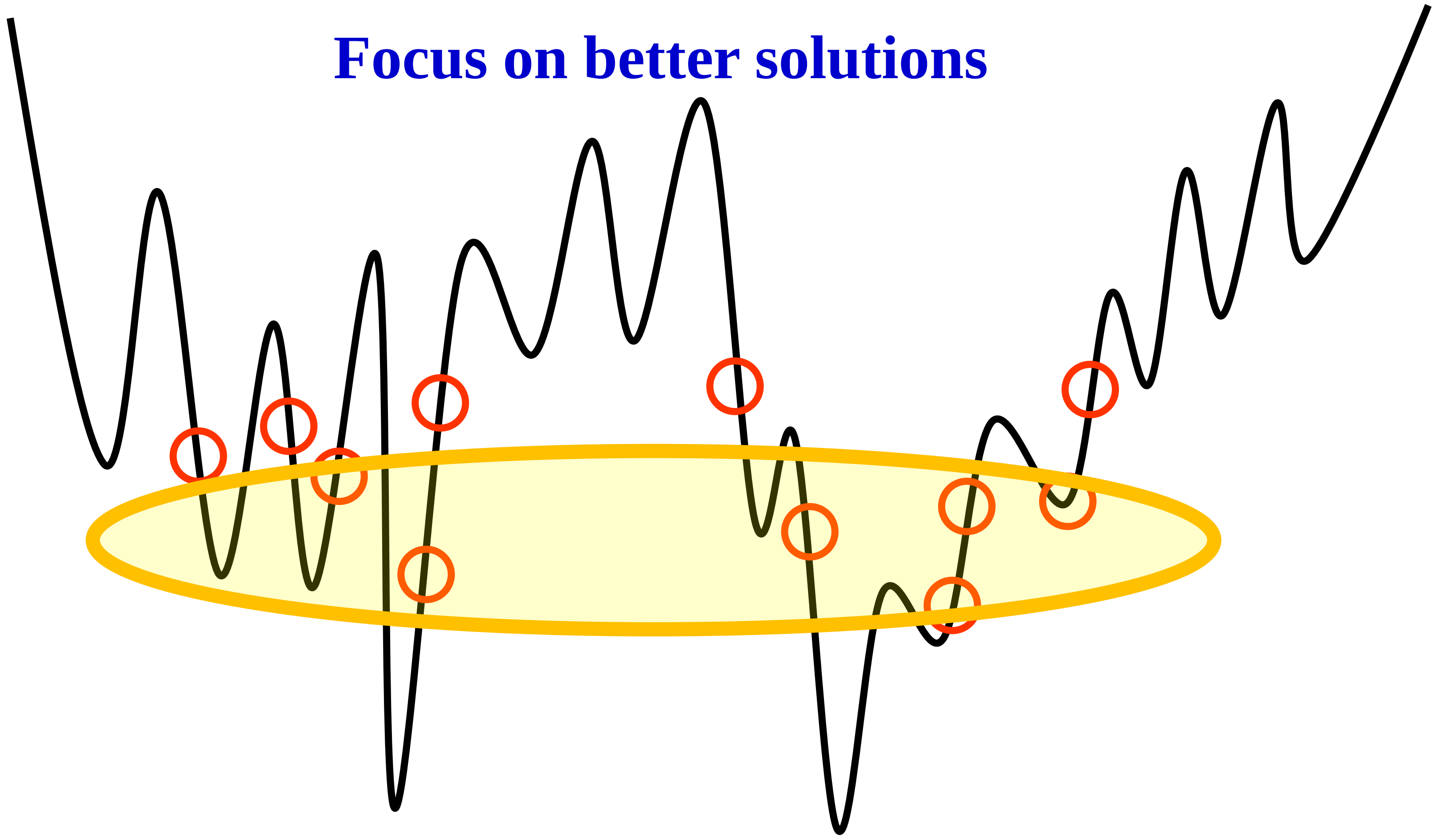
Evolutionary Algorithms (e.g., Genetic Algorithms)

- Population-based search algorithms
- Multi-point search algorithms



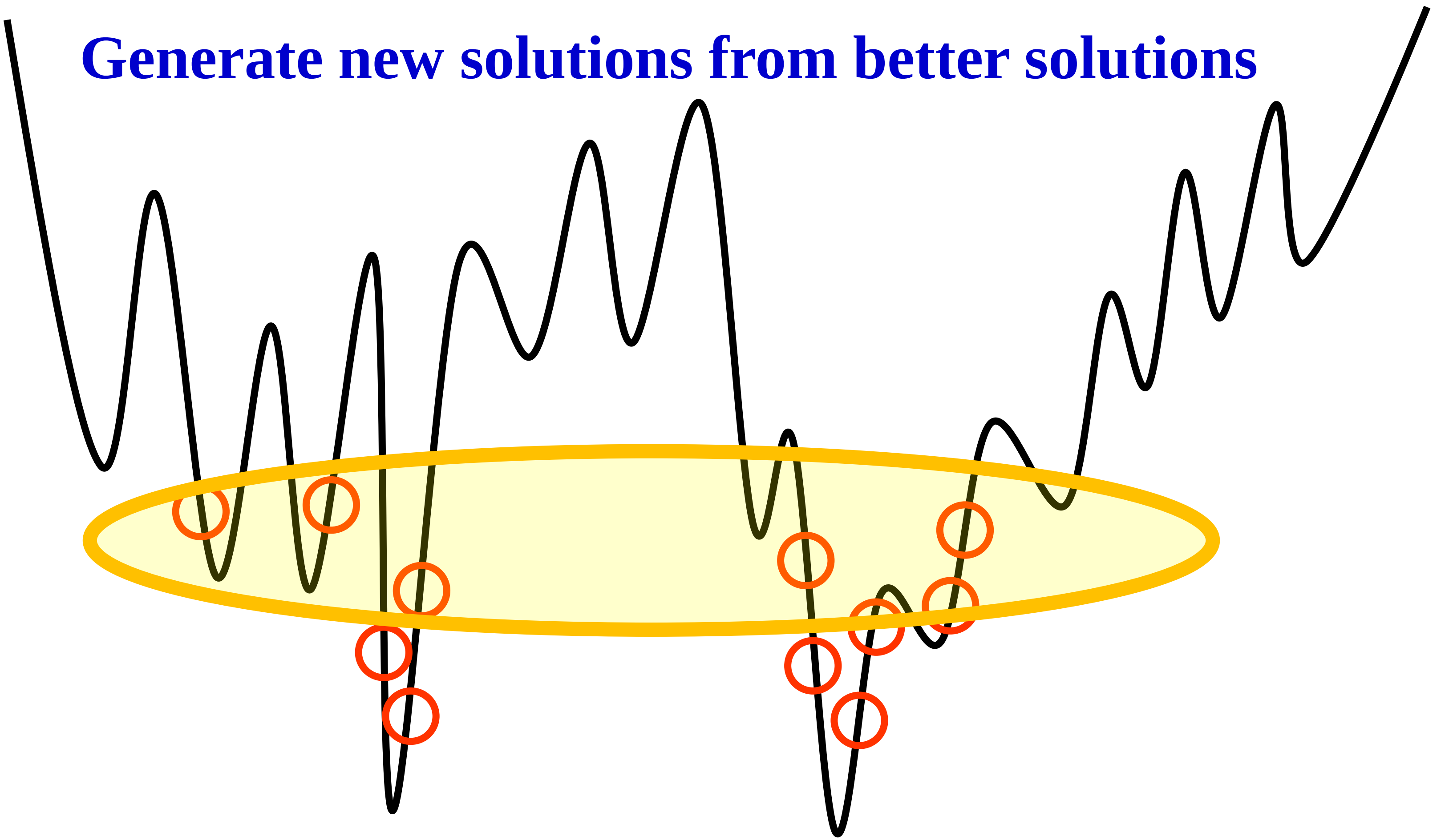
Evolutionary Algorithms (e.g., Genetic Algorithms)

- Population-based search algorithms
- Multi-point search algorithms



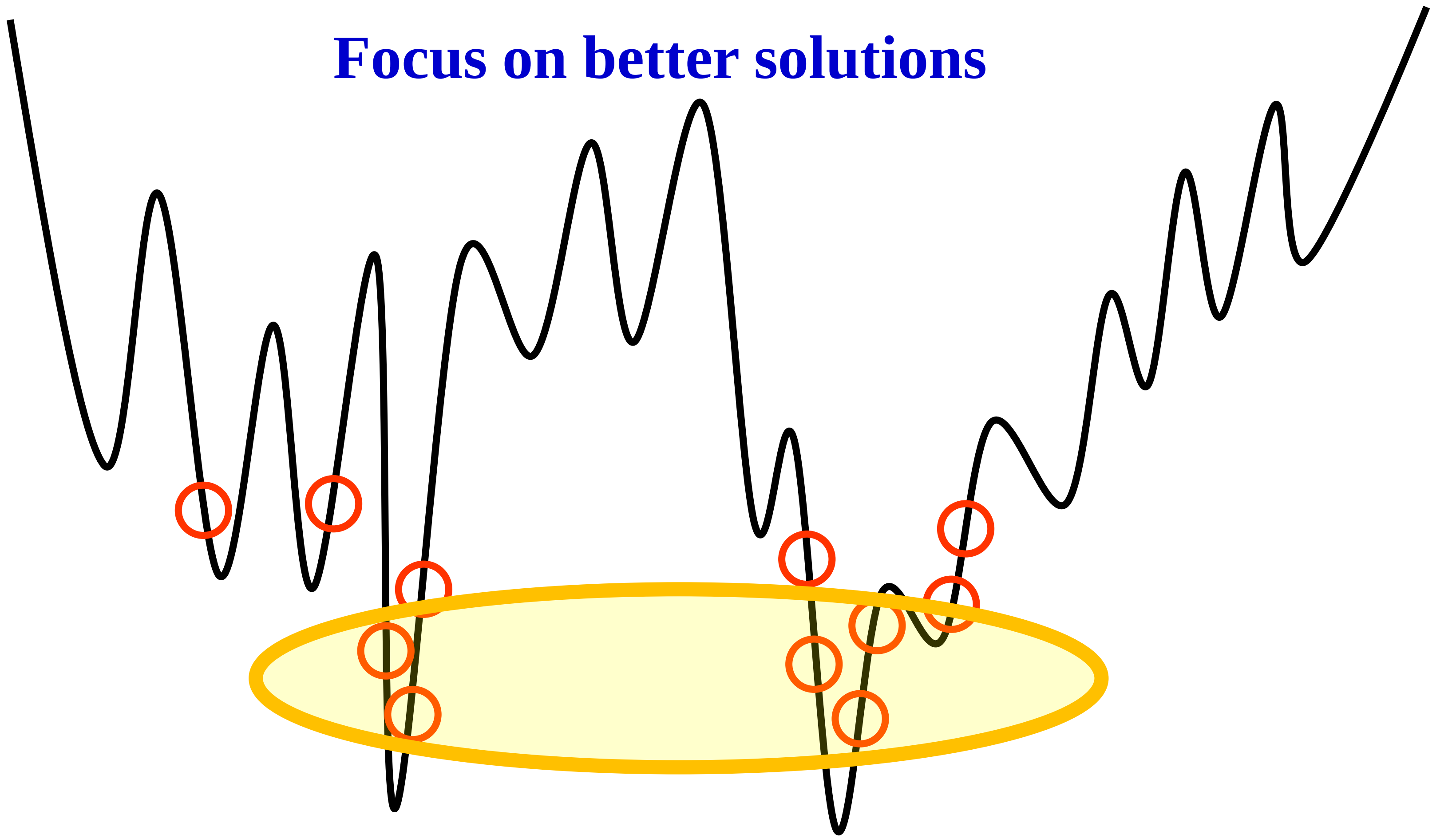
Evolutionary Algorithms (e.g., Genetic Algorithms)

- Population-based search algorithms
- Multi-point search algorithms



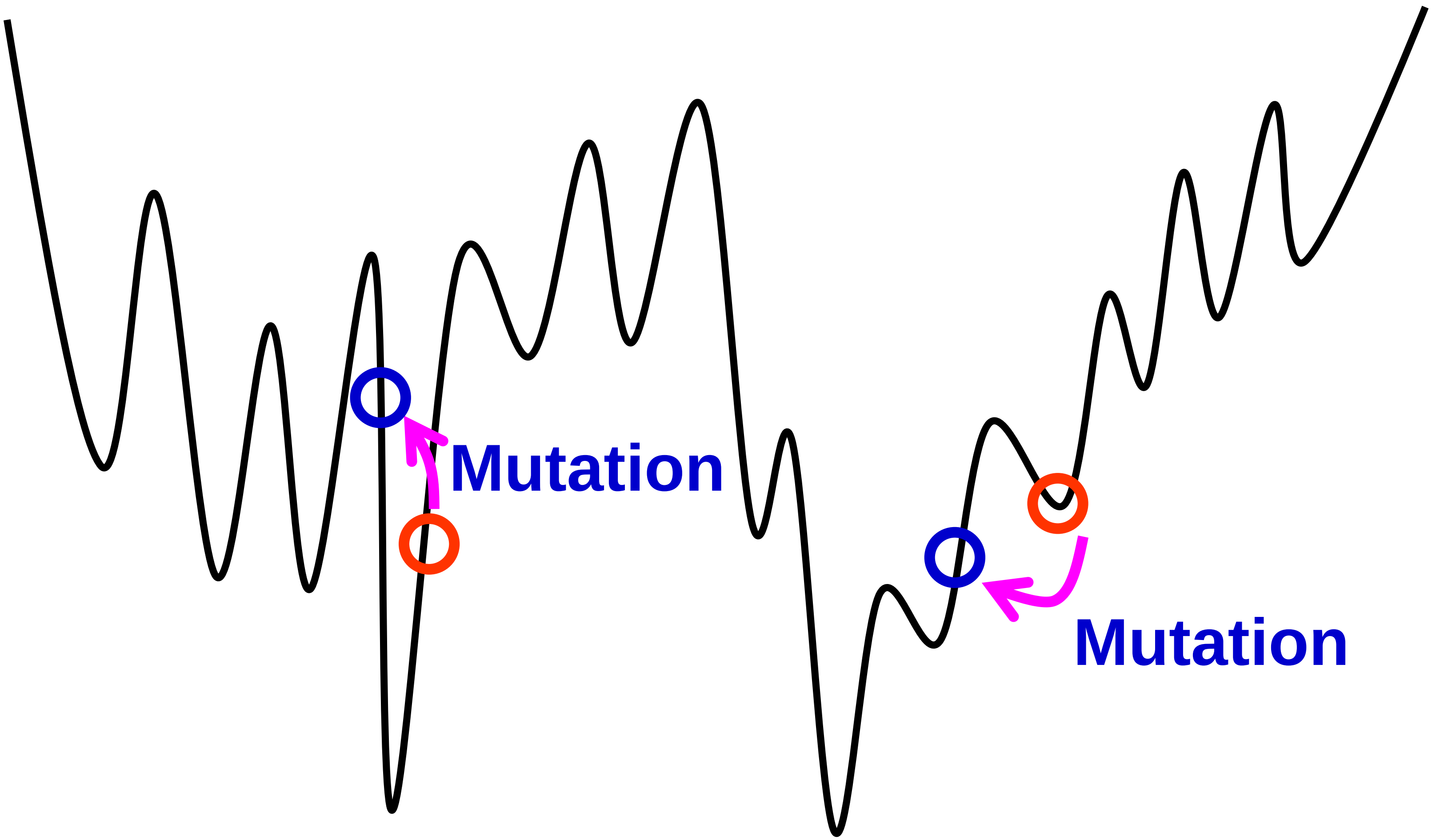
Evolutionary Algorithms (e.g., Genetic Algorithms)

- Population-based search algorithms
- Multi-point search algorithms



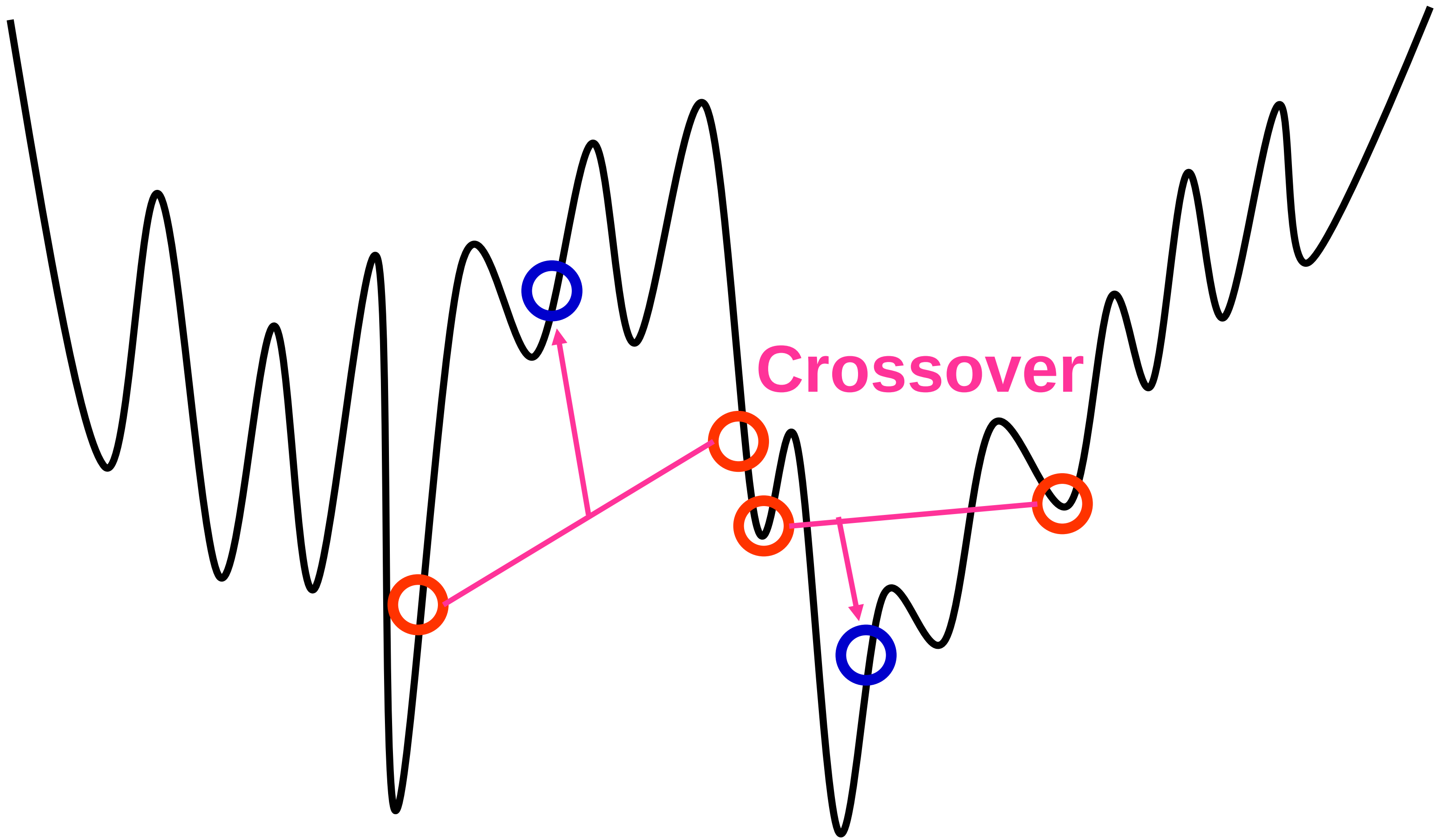
Two Operators for New Solution Generation

(1) Mutation (Random choice of a neighbor)
(2) Crossover (Recombination of two parents)

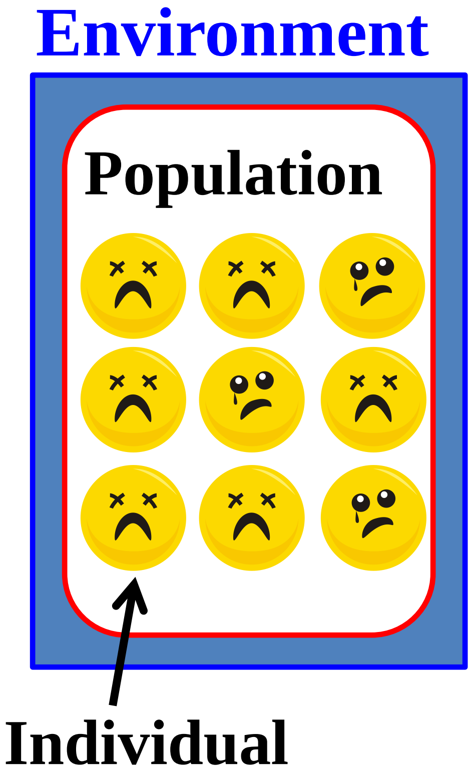


Two Operators for New Solution Generation

(1) Mutation (Random choice of a neighbor)
(2) Crossover (Recombination of two parents)

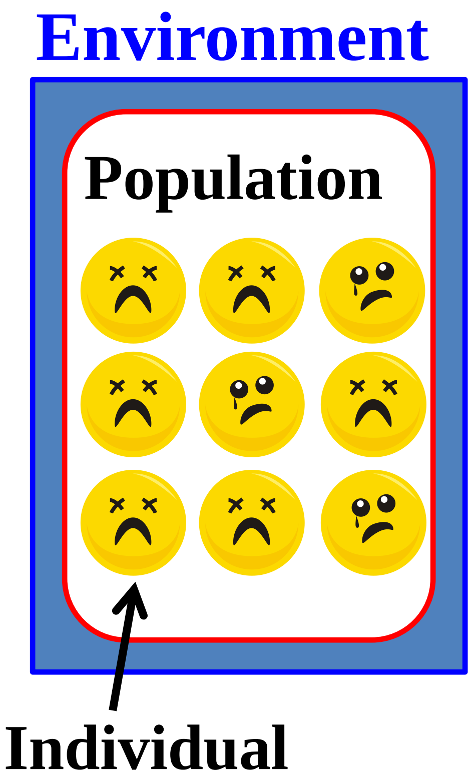


Basic Idea of Evolutionary Computation



(1) A population of individuals is randomly generated.
(2) Each individual is evaluated in the environment.

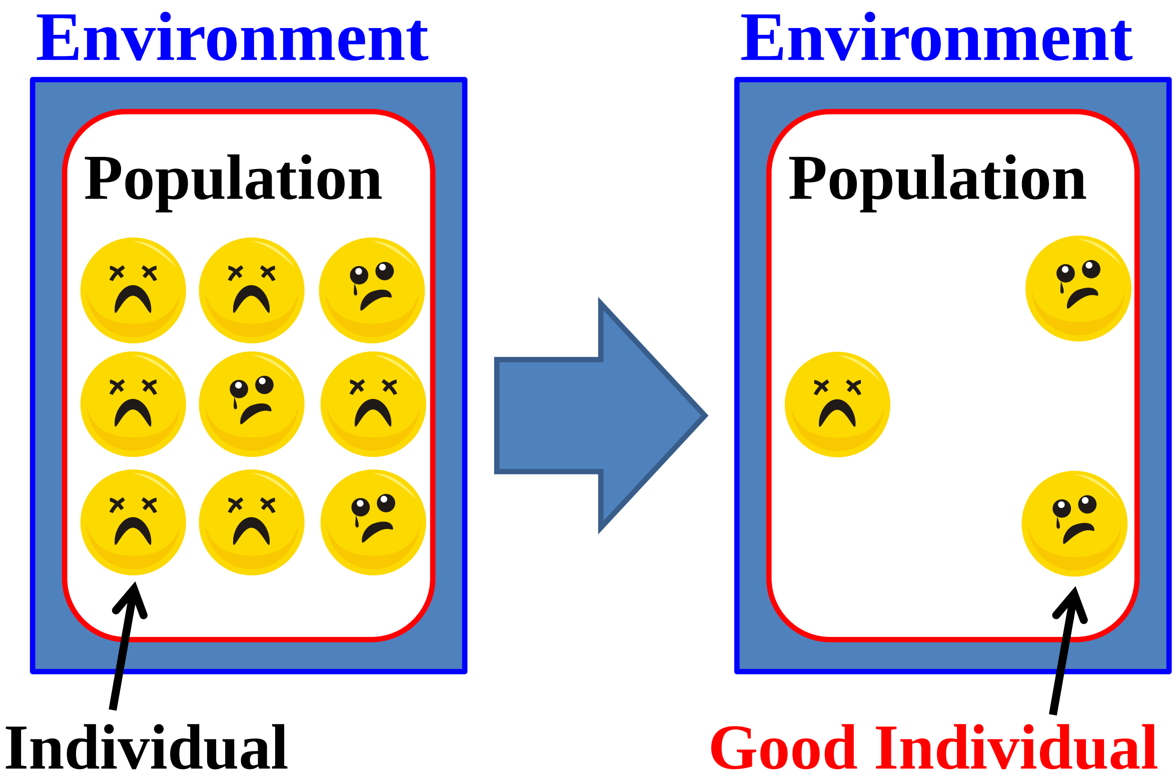
Basic Idea of Evolutionary Computation



The search ability of evolutionary computation can be significantly improved by using good initial solutions (whereas this is not emphasized).

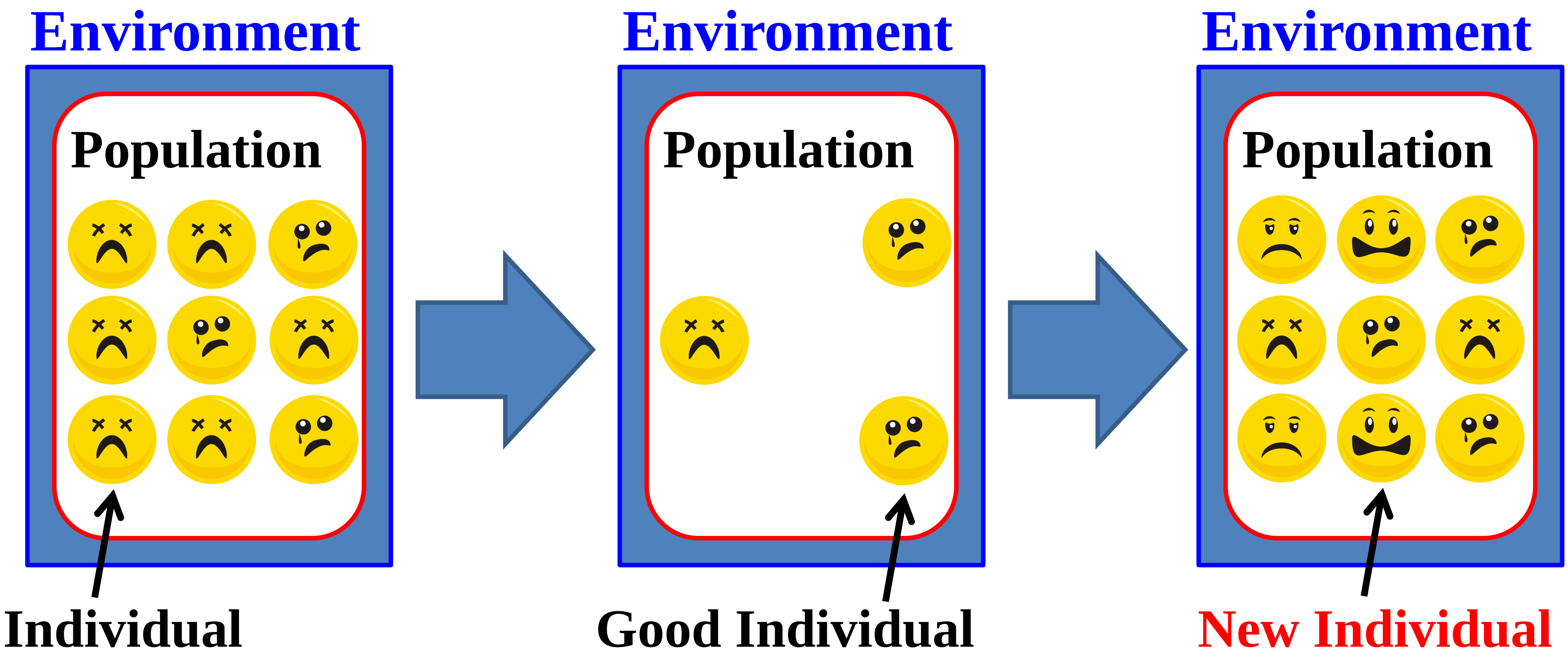
(1) A population of individuals is <u>randomly</u> generated.
(2) Each individual is evaluated in the environment.

Basic Idea of Evolutionary Computation



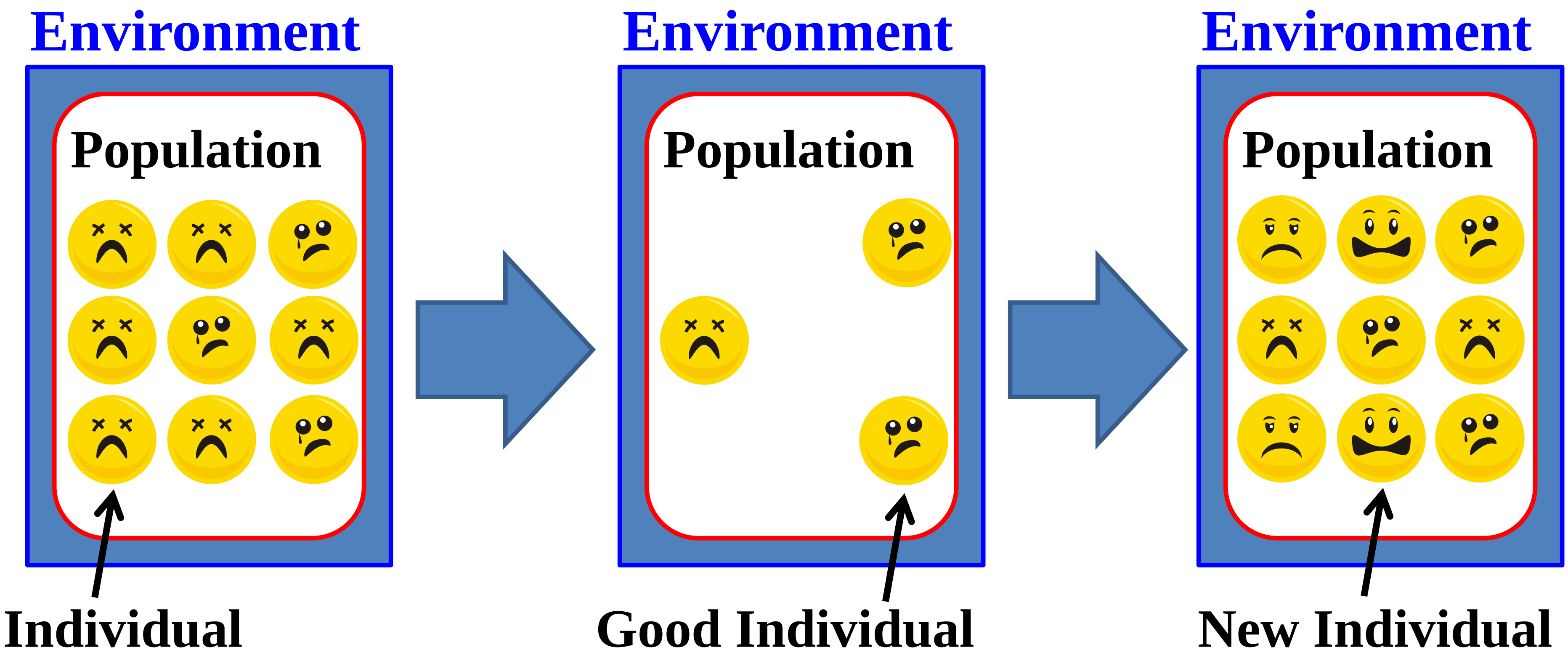
(1) A population of individuals is randomly generated.
(2) Each individual is evaluated in the environment.
(3) Good individuals survive probabilistically.

Basic Idea of Evolutionary Computation



(1) A population of individuals is randomly generated.
(2) Each individual is evaluated in the environment.
(3) Good individuals survive probabilistically.
(4) New individuals are generated from the good individuals.

Basic Idea of Evolutionary Computation



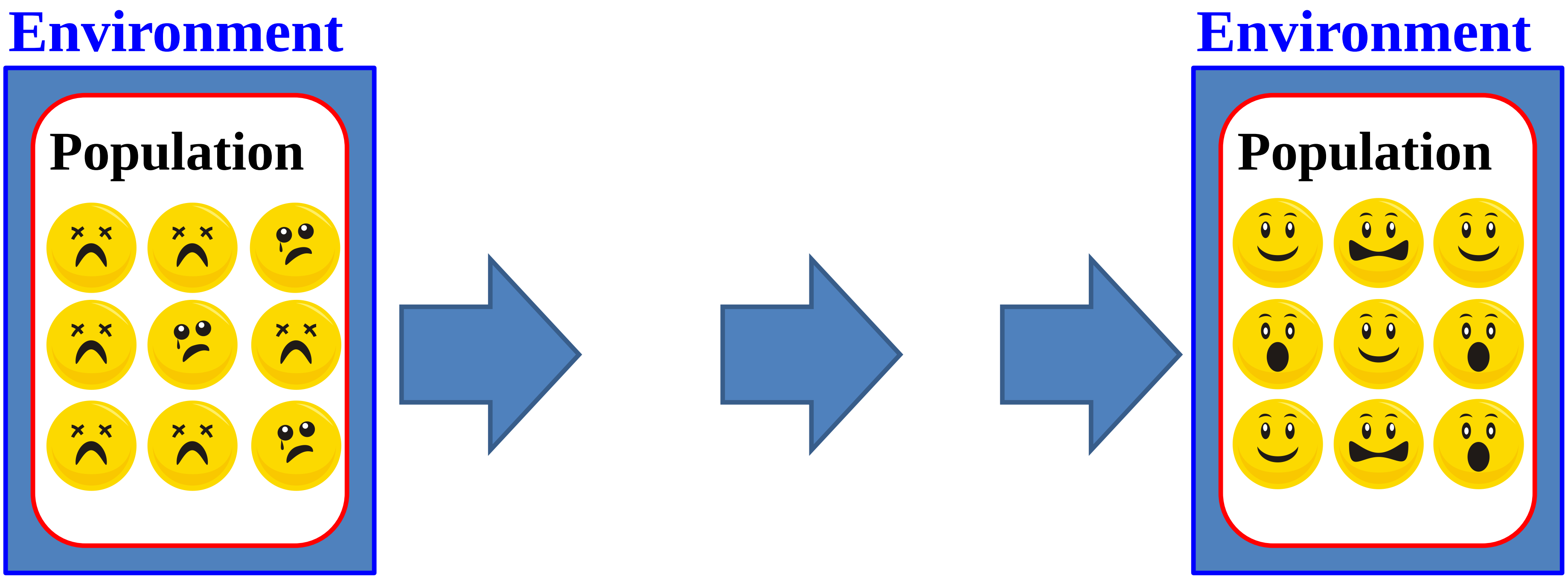
These steps (2)-(4) are iterated many times.

(2) Each individual is evaluated in the environment.

(3) Good individuals survive probabilistically.

(4) New individuals are generated from the good individuals.

Basic Idea of Evolutionary Computation



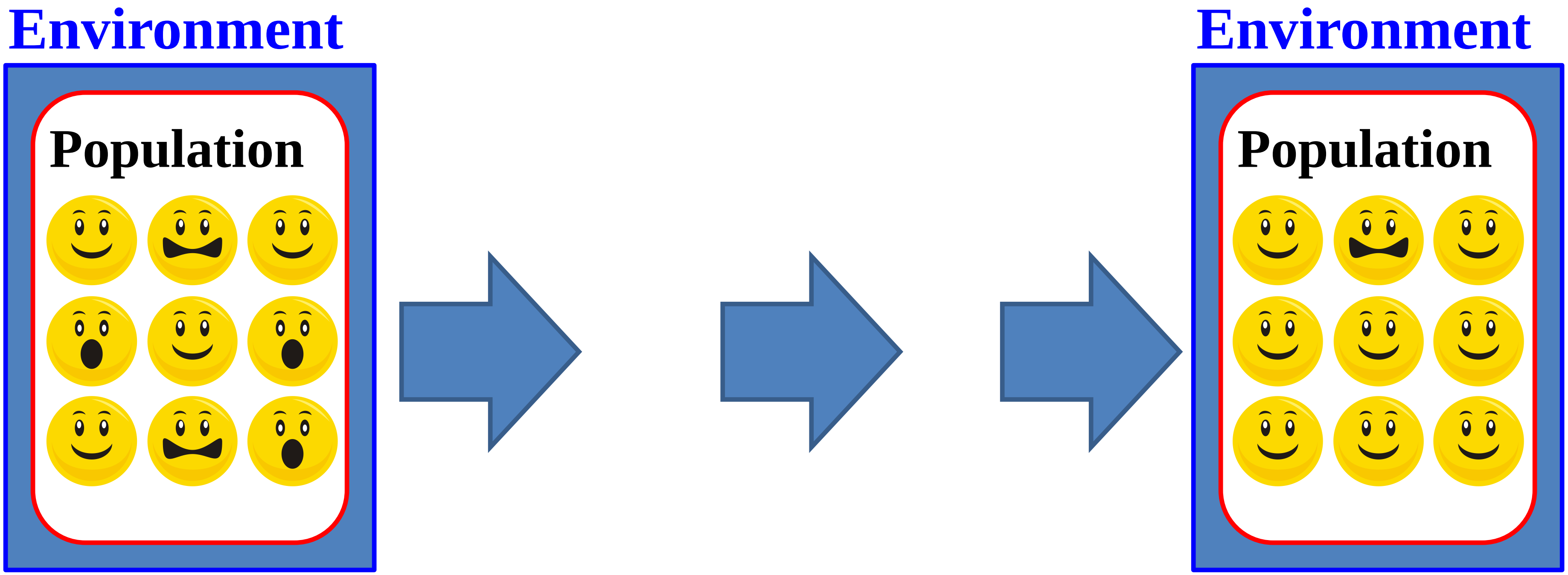
These steps (2)-(4) are iterated many times.

(2) Each individual is evaluated in the environment.

(3) Good individuals survive.

(4) New individuals are generated from the good individuals.

Basic Idea of Evolutionary Computation



These steps (2)-(4) are iterated many times.

(2) Each individual is evaluated in the environment.

(3) Good individuals survive.

(4) New individuals are generated from the good individuals.

After many generations, we may have good solutions.

Application of Evolutionary Computation

Design of High Speed Train

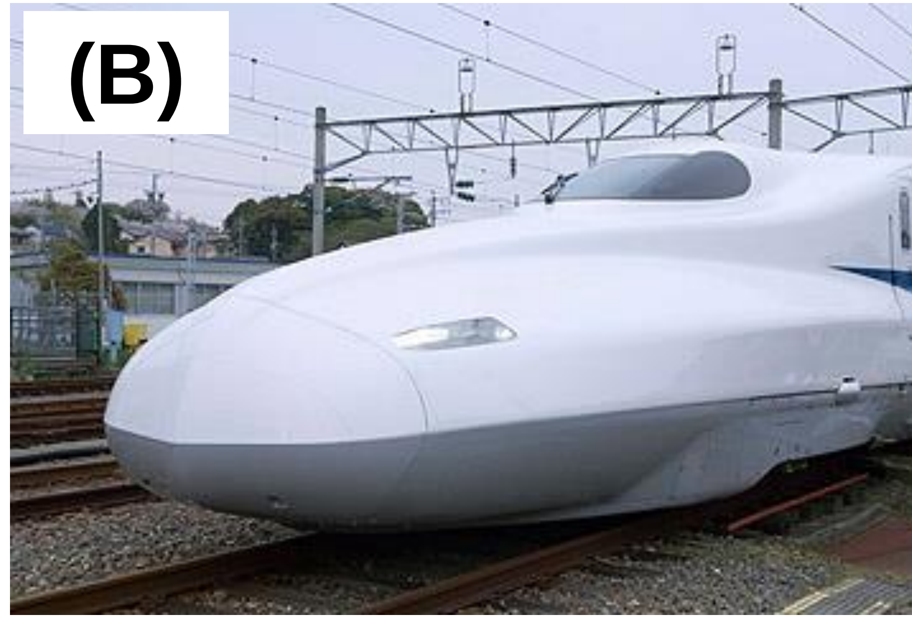
N700系フォトギャラリー

印刷

閉じる

東海旅客鉄道株式会社

Two Types of High Speed Trains



(A) Old Design by Human Experts.

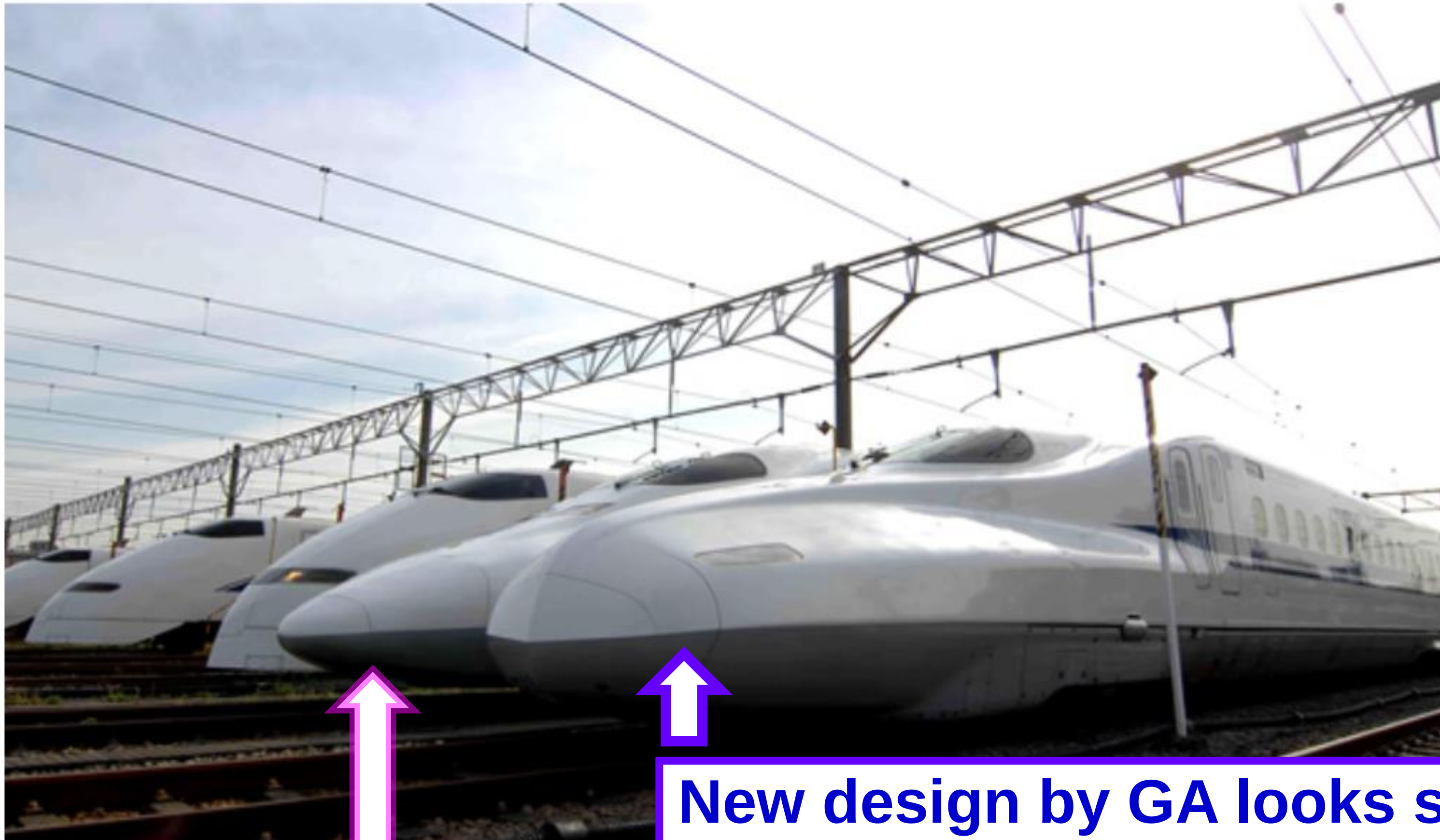
(B) New Design by Genetic Algorithms.

Old design looks better.

New design looks strange.

High Speed Trains in Japan

N700系フォトギャラリー



New design by GA looks strange

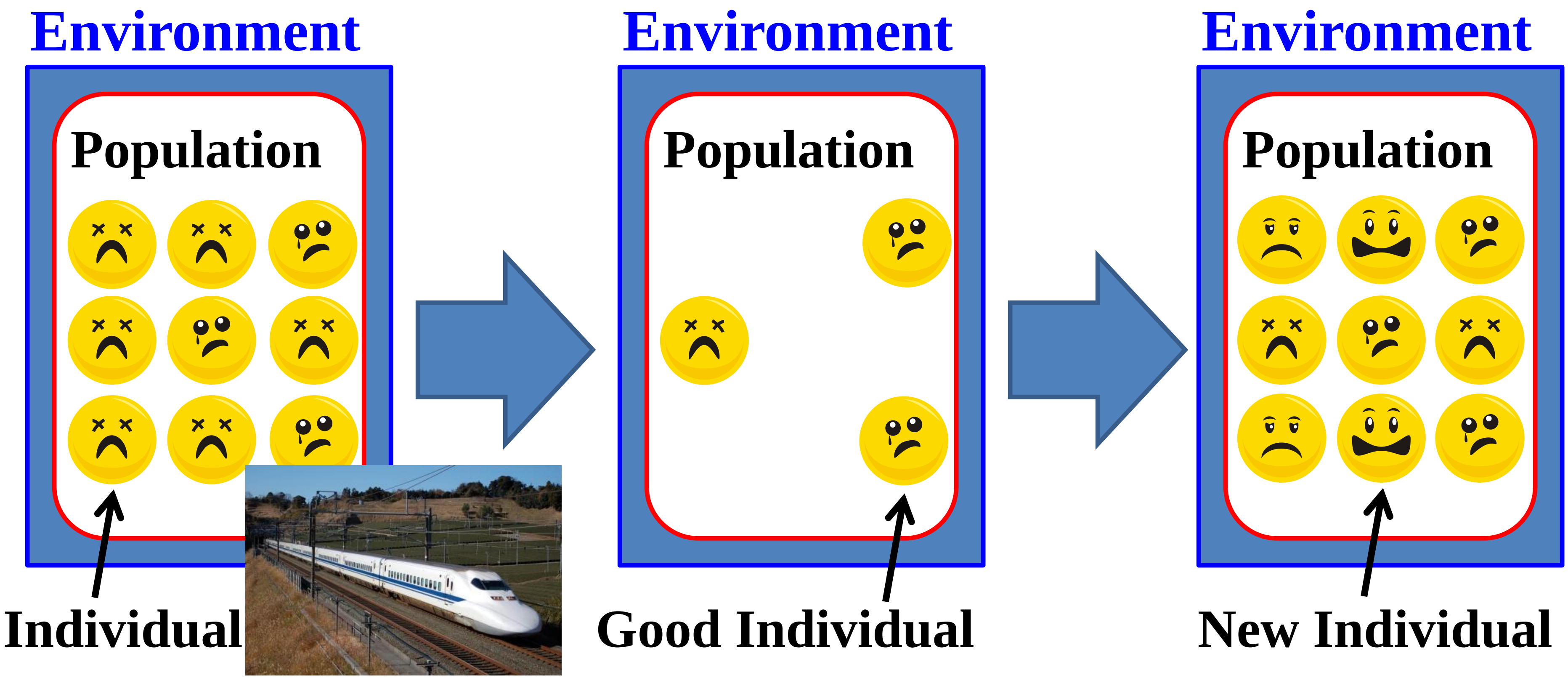
Old design by human experts looks nice !

印刷

閉じる

東海旅客鉄道株式会社

Basic Idea of Evolutionary Computation



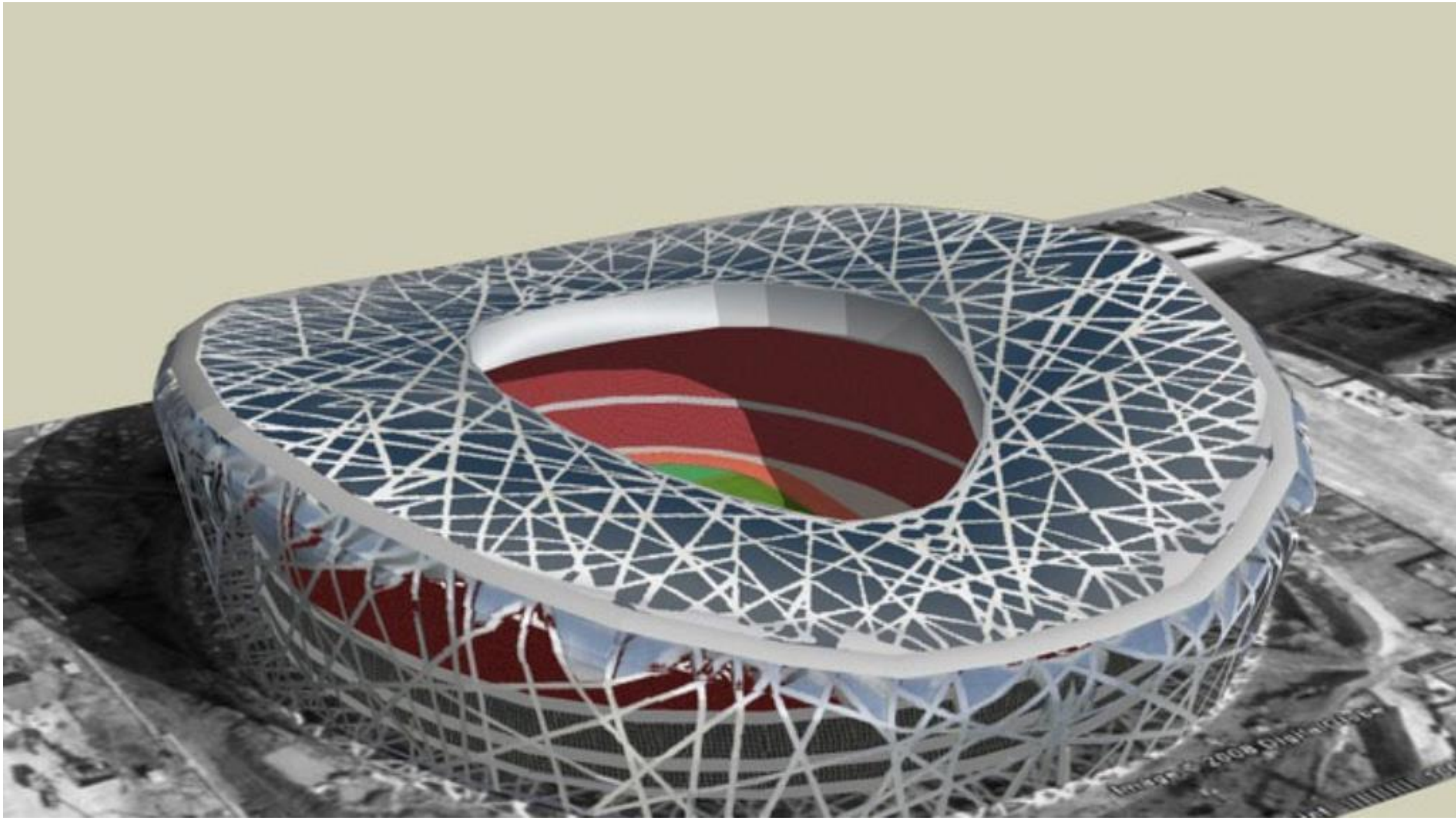
These steps (2)-(4) are iterated many times.

(2) Each individual is evaluated in the environment.

(3) Good individuals survive probabilistically.

(4) New individuals are generated from the good individuals.

Applications of Evolutionary Computation
Construction Planning



http://sketchup3dconstruction.com/skp/warehouse/stadiums/beijing-national-stadium.html

Applications of Evolutionary Computation

Construction Planning

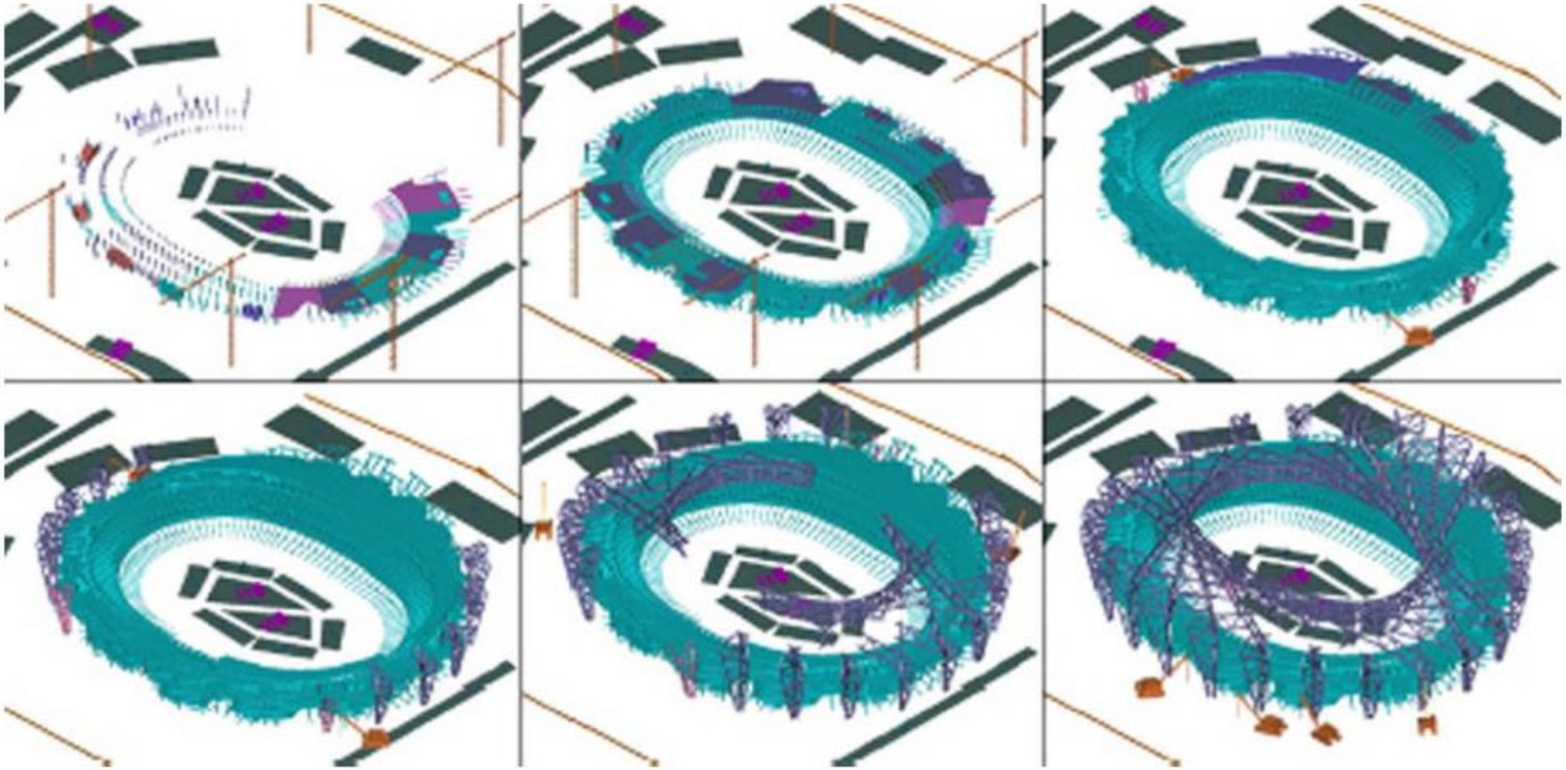
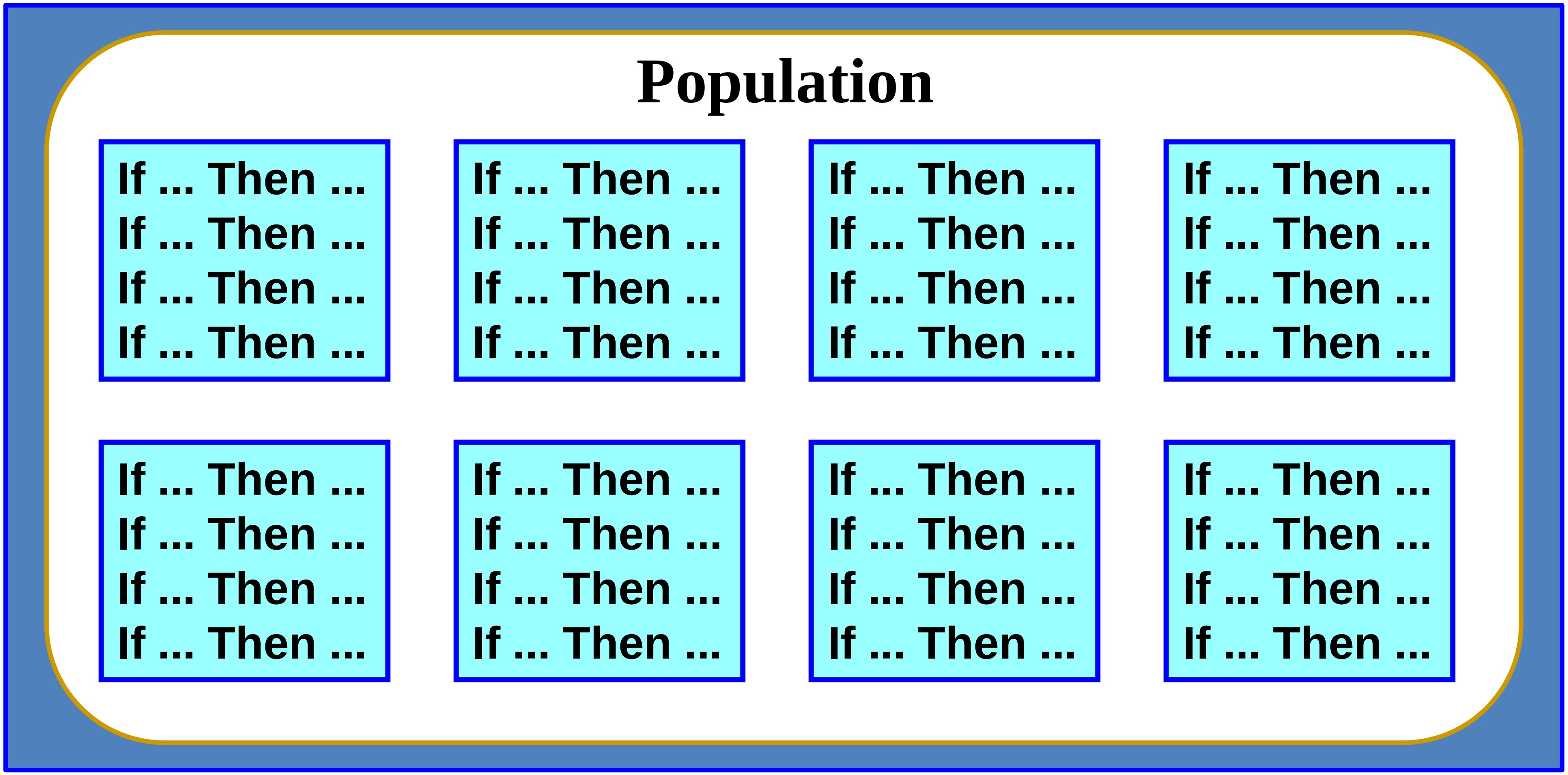


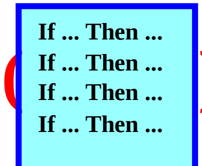
Fig. 4 4D screenshots of construction plan

J.ZHANG, Y. ZHANG, Z. HU, and M.LU: Construction Management Utilizing 4D CAD and Operations Simulation Methodologies, *TSINGHUA SCIENCE AND TECHNOLOGY*, Vol. 13, No. S1, pp. 241-247. Oct 2008.

Design of Rule-Based Systems

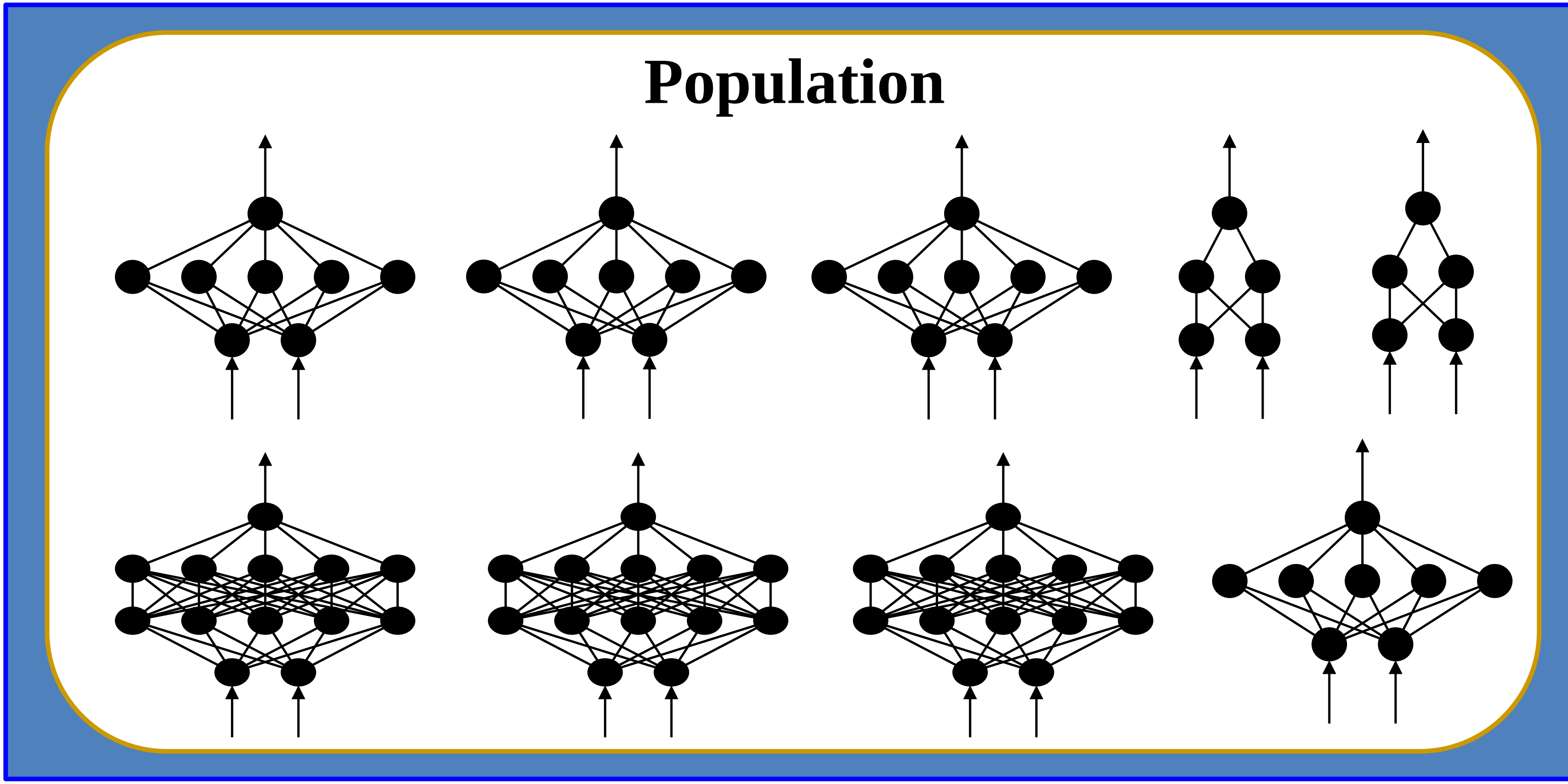
Environment

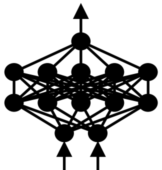


Individual = Rule-Based System ()

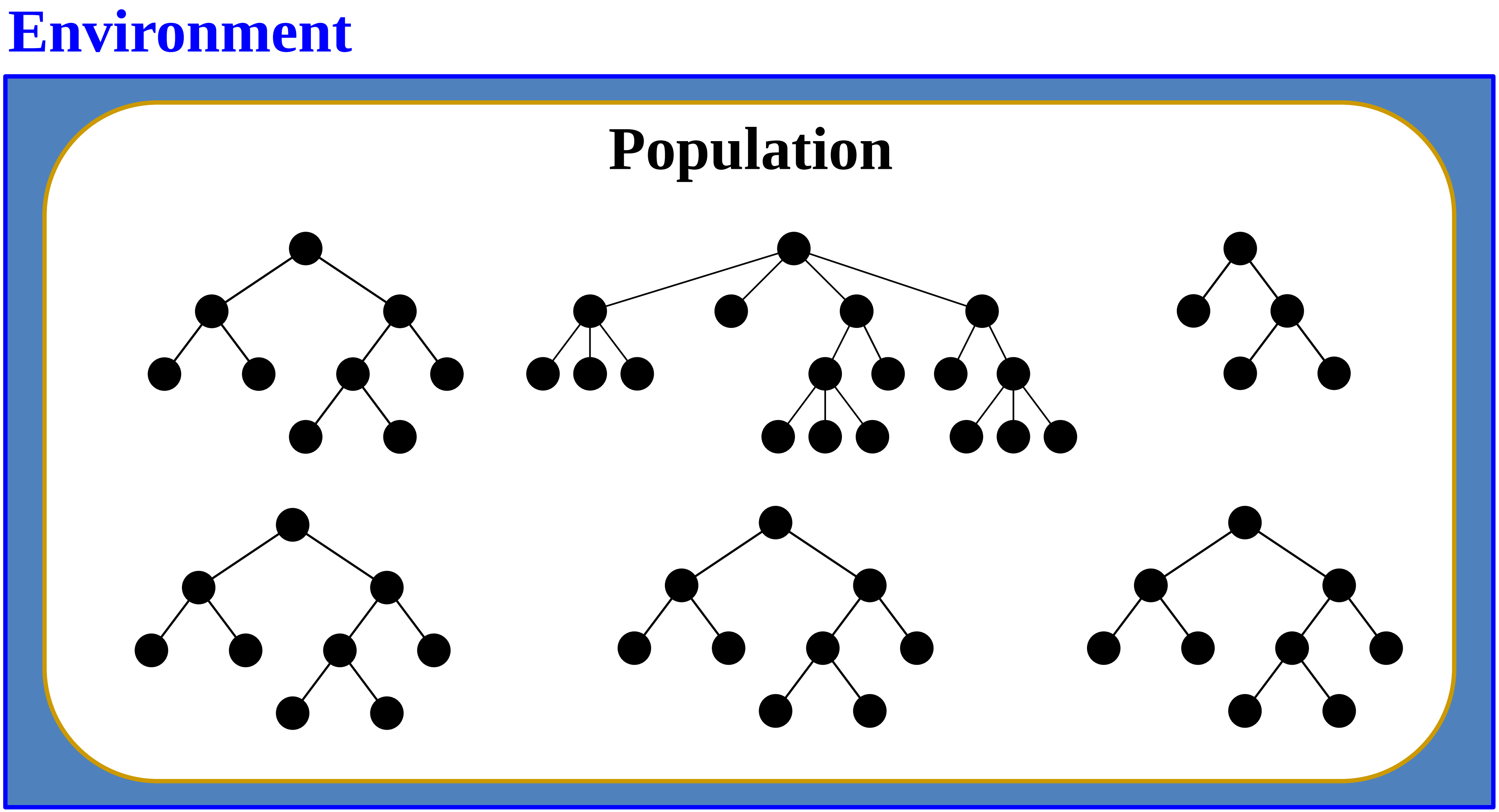
Design of Neural Networks

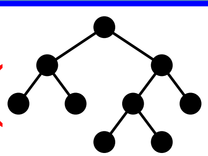
Environment



Individual = Neural Network ()

Design of Decision Trees



Individual = Decision Tree ()

How to Represent Each Solution for **Computer Simulation of Evolution**

Any string can be used

depending on the problem at hand

Any string can be used!

Knapsack Problem:

Binary String	1	1	1	0	1	0	0	0	0
Random Key	0.23	0.29	0.19	0.59	0.13	0.79	0.91	0.53	0.38
(Order)	3	4	2	7	1	8	9	6	5
Permutation	5	3	1	2	9	8	4	6	7

Random Key Genetic Algorithm (1994)

- Each solution is encoded as an array of *n* random keys

- A random key is a real number randomly generated in the interval [0, 1).

Any string can be used!

Knapsack Problem:

Binary String	1	1	1	0	1	0	0	0	0
Random Key	0.23	0.29	0.19	0.59	0.13	0.79	0.91	0.53	0.38
(Order)	3	4	2	7	1	8	9	6	5
Permutation	5	3	1	2	9	8	4	6	7

Random Key Genetic Algorithm (1994)

- Each solution is encoded as an array of *n* random keys

- A random key is a real number randomly generated in the interval [0, 1).

Example (Ascending Order as in the Above Example)

Random Key Coding: (0.46, 0.91, 0.33, 0.75, 0.51)

Permutation Coding: **?** => **?** => **?** => **?** => **?** **Send your answer**

Any string can be used!

Knapsack Problem:

Binary String	1	1	1	0	1	0	0	0	0
Random Key	0.23	0.29	0.19	0.59	0.13	0.79	0.91	0.53	0.38
(Order)	3	4	2	7	1	8	9	6	5
Permutation	5	3	1	2	9	8	4	6	7

Random Key Genetic Algorithm (1994)

- Each solution is encoded as an array of *n* random keys

- A random key is a real number randomly generated in the interval [0, 1).

Example (Ascending Order as in the Above Example)

Random Key Coding: (0.46, 0.91, 0.33, 0.75, 0.51)

Permutation Coding: 3 => 1 => 5 => 4 => 2

Any string can be used!

Knapsack Problem:

Binary String	1	1	1	0	1	0	0	0	0
Permutation	5	3	1	2	9	8	4	6	7
Random Key	0.23	0.29	0.19	0.59	0.13	0.79	0.91	0.53	0.38

TSP, Flowshop Scheduling:

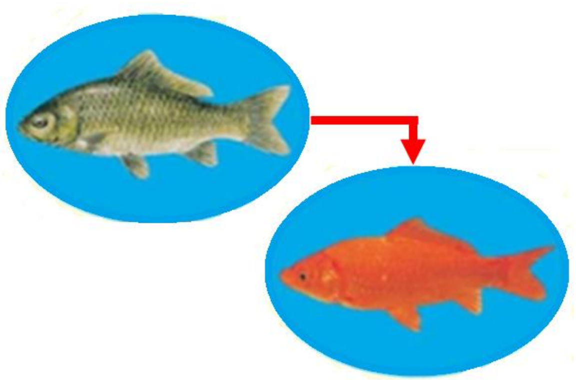
Permutation	5	3	1	2	9	8	4	6	7
Random Key	0.23	0.29	0.19	0.59	0.13	0.79	0.91	0.53	0.38

Function Optimization:

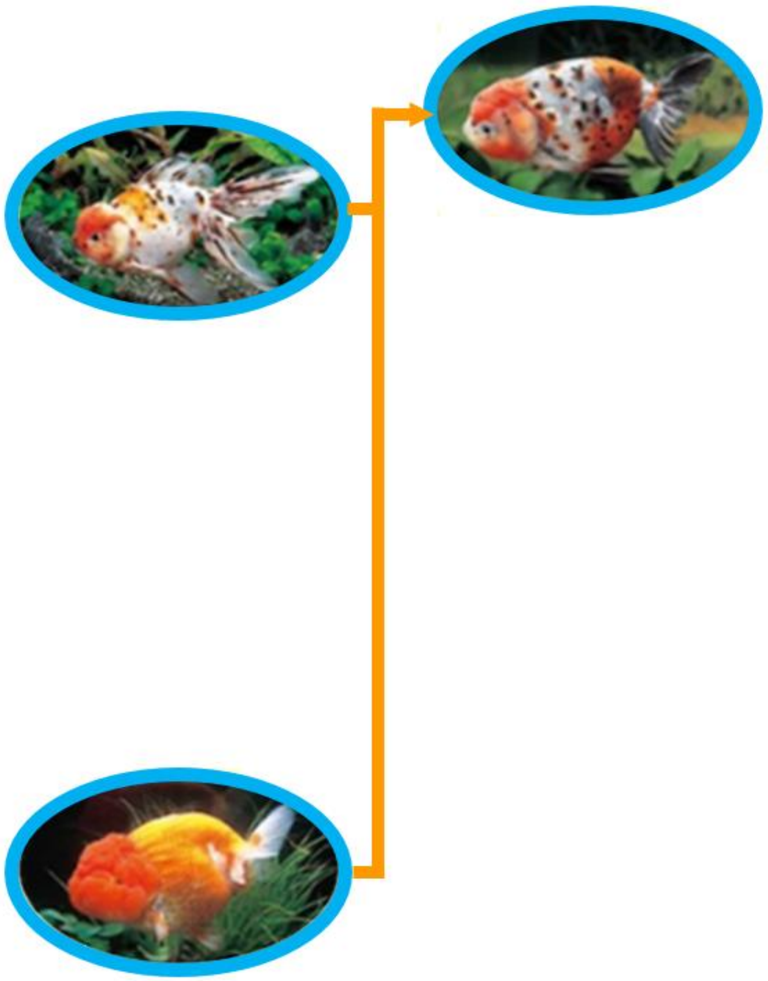
	x_1	x_2	x_3
Real number string	25.297	123.45	92.834
Binary String	1 1 1 0 1 0 0 0 0 1 1 1	0 1 0 0 0 0 1 0 0 1 1 0	0 0 0 1 1 1 0 1 0 0 0 0

How to generate new solutions

Mutation

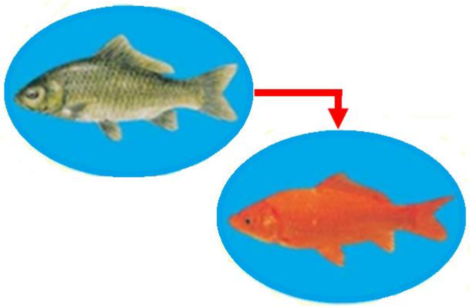


Crossover



Mutation

Random change of a part of a string



Random change of a part of a string

(Each value has the same mutation probability, e.g., 1/*n*, 2/*n*)

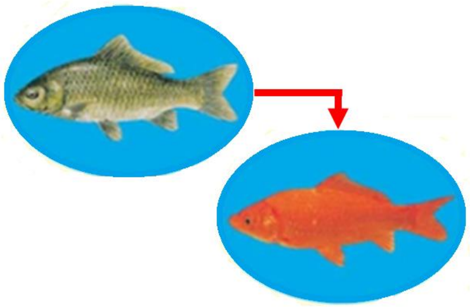
1	1	1	0	1	0	0	0	0	1

Binary String

1	0	1	0	1	0	0	1	0	1

Mutation

Random change of a part of a string



Random change of a part of a string

(Each value has the same mutation probability, e.g., 1/*n*, 2/*n*)

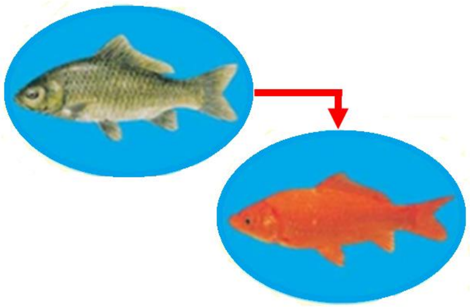
23	45	21	12	0	12	23	0	58	23

Integer String

23	45	21	12	0	12	23	0	26	23

Mutation

Random change of a part of a string



Random change of a part of a string

(Each value has the same mutation probability, e.g., 1/*n*, 2/*n*)

23.42	45.20	21.45	12.09	0.00	12.14	23.43	0.00	58.98	23.12

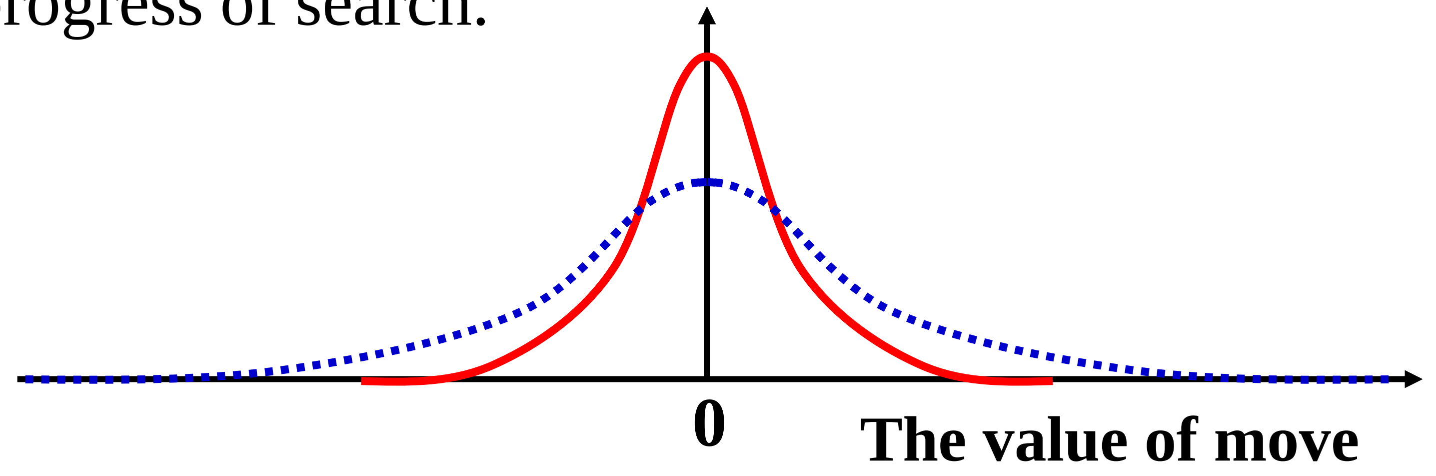
⇩ **Real Number String**

23.42	67.13	21.45	12.09	0.00	12.14	23.43	15.34	58.98	23.12

Mutation

For real number strings, a distribution can be used.

The spread of the distribution is a parameter, which can be a fixed parameter or an automatically adjustable parameter during the progress of search.



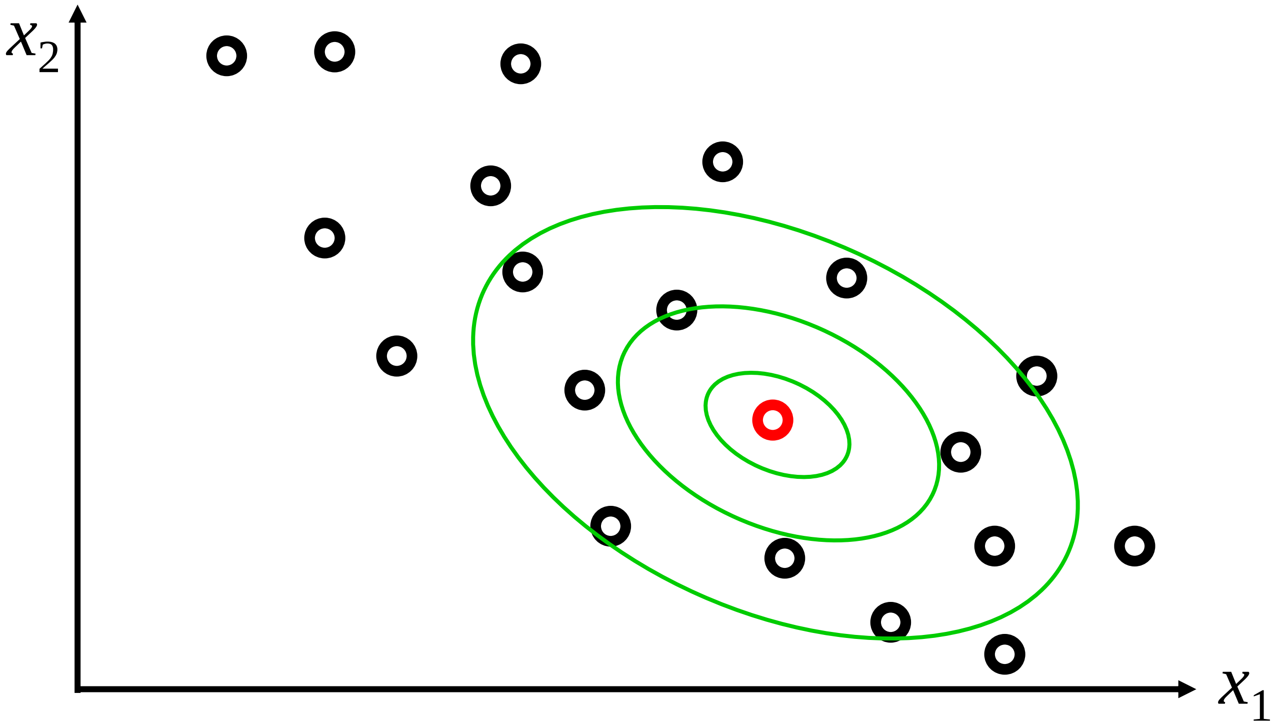
23.42	45.20	21.45	12.09	0.00	12.14	23.43	0.00	58.98	23.12

23.42	49.13	21.45	12.09	0.00	12.14	23.43	3.34	58.98	23.12

Mutation

A multi-dimensional distribution can be also used.

The distribution is usually automatically adjusted by the solutions in the current population (and the previous populations).



- All values of a solution are mutated.
- A mutation probability is assigned to each solution (not each value).

Mutation for Permutation Strings

Random change does not generate a permutation.

7	6	1	5	2	4	0	8	9	3

7	6	9	5	2	4	0	8	9	3

Two “9” and no “1”.

Mutation for Permutation Strings

A neighborhood structure is needed.

Adjacent two-position change

7	6	1	5	2	4	0	8	9	3

7	6	5	1	2	4	0	8	9	3

Two-position change

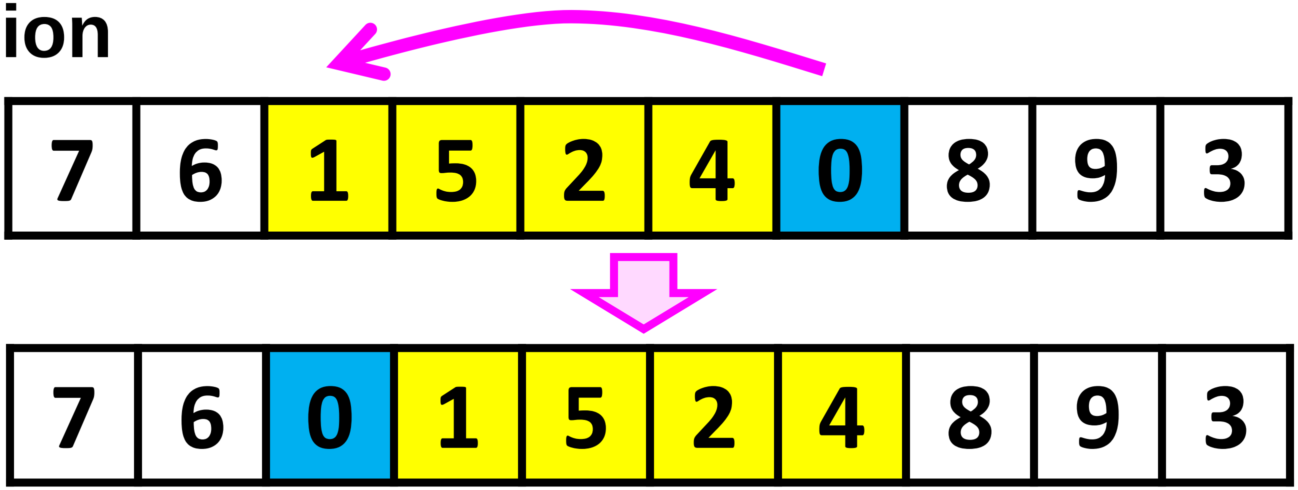
7	6	1	5	2	4	0	8	9	3

7	8	1	5	2	4	0	6	9	3

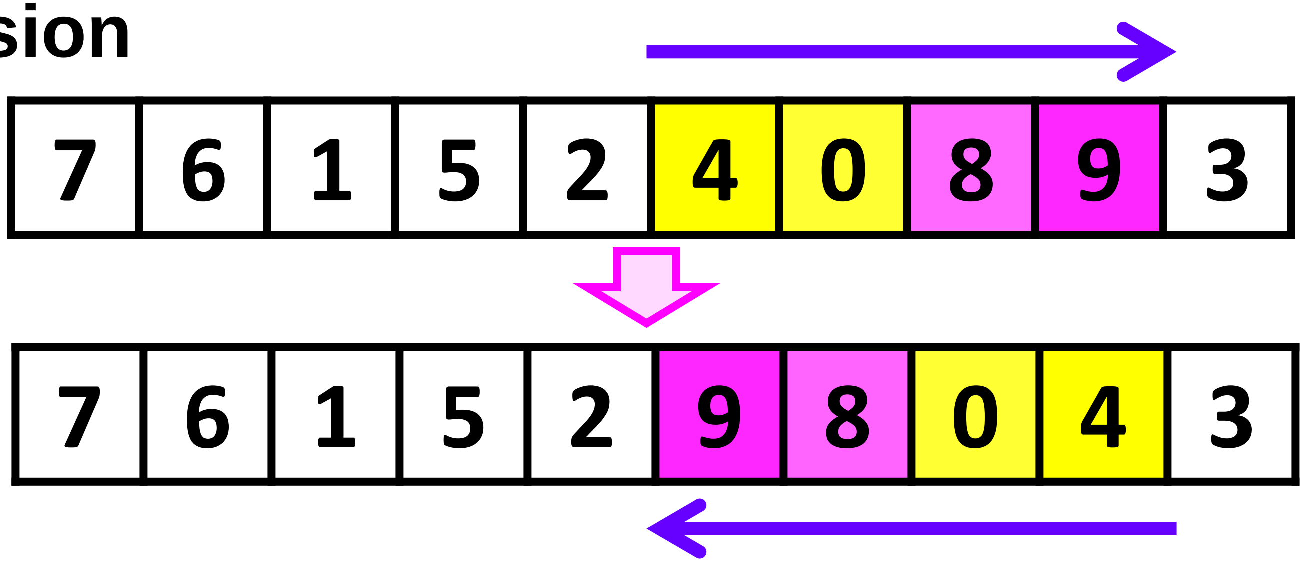
Mutation for Permutation Strings

A mutation probability is assigned to each string.

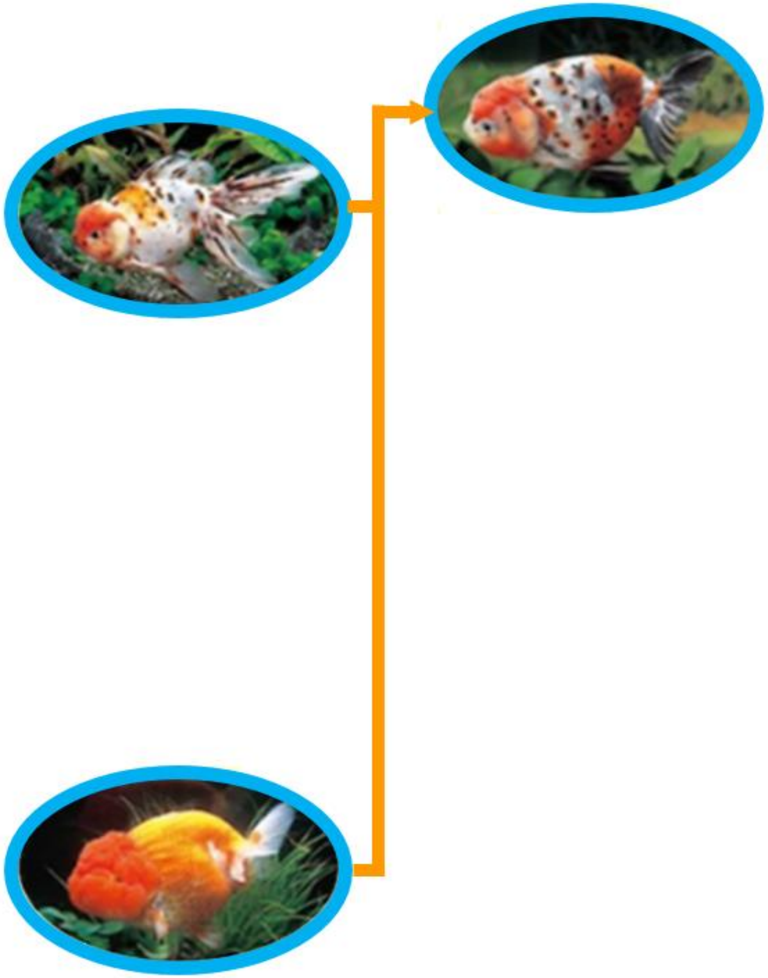
Insertion



Inversion



Crossover



Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

Parent A	1	1	1	0	1	0	0	0	0	1
Parent B	1	0	0	0	1	1	1	0	1	0

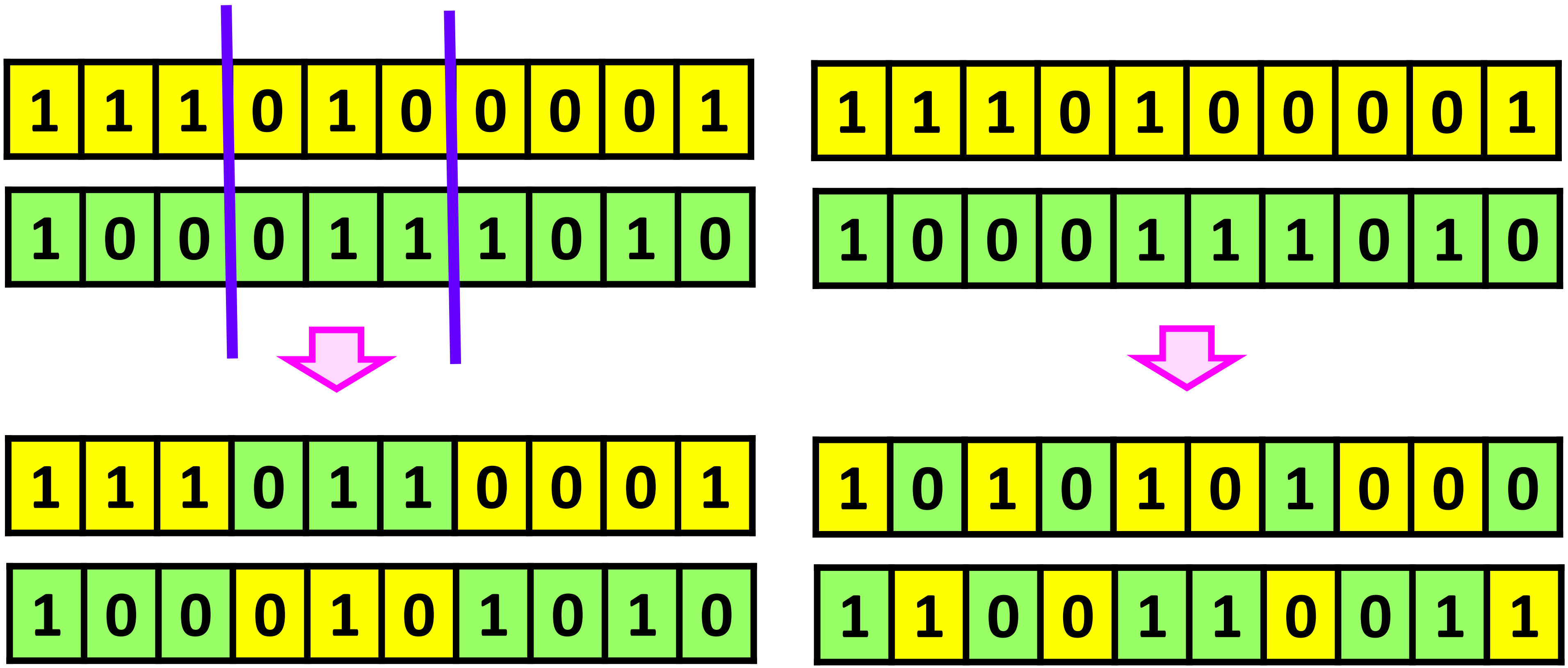
Child A	1	1	1	0	1	0	1	0	1	0
Child B	1	0	0	0	1	1	0	0	0	1

One-Point Crossover

Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

Any exchanges are usually OK.



Two-Point Crossover | **Uniform Crossover**

Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

Multiple parents can be used.

Parent A	1	1	1	0	1	0	0	0	0	1
Parent B	1	0	0	0	1	1	1	0	1	0
Parent C	1	0	1	0	1	1	0	1	0	1
Child	1	0	0	0	1	0	0	1	0	1

Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

Calculation can be used.

0.2 x	1.00	0.00	0.00	1.00	2.00	1.00	1.00	10.00	1.00	1.00

+

0.8 x	1.00	1.00	10.00	1.00	10.00	1.00	10.00	10.00	20.00	1.00

=	1.00	0.80	8.00	1.00	8.40	1.00	8.20	10.00	16.20	1.00

Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

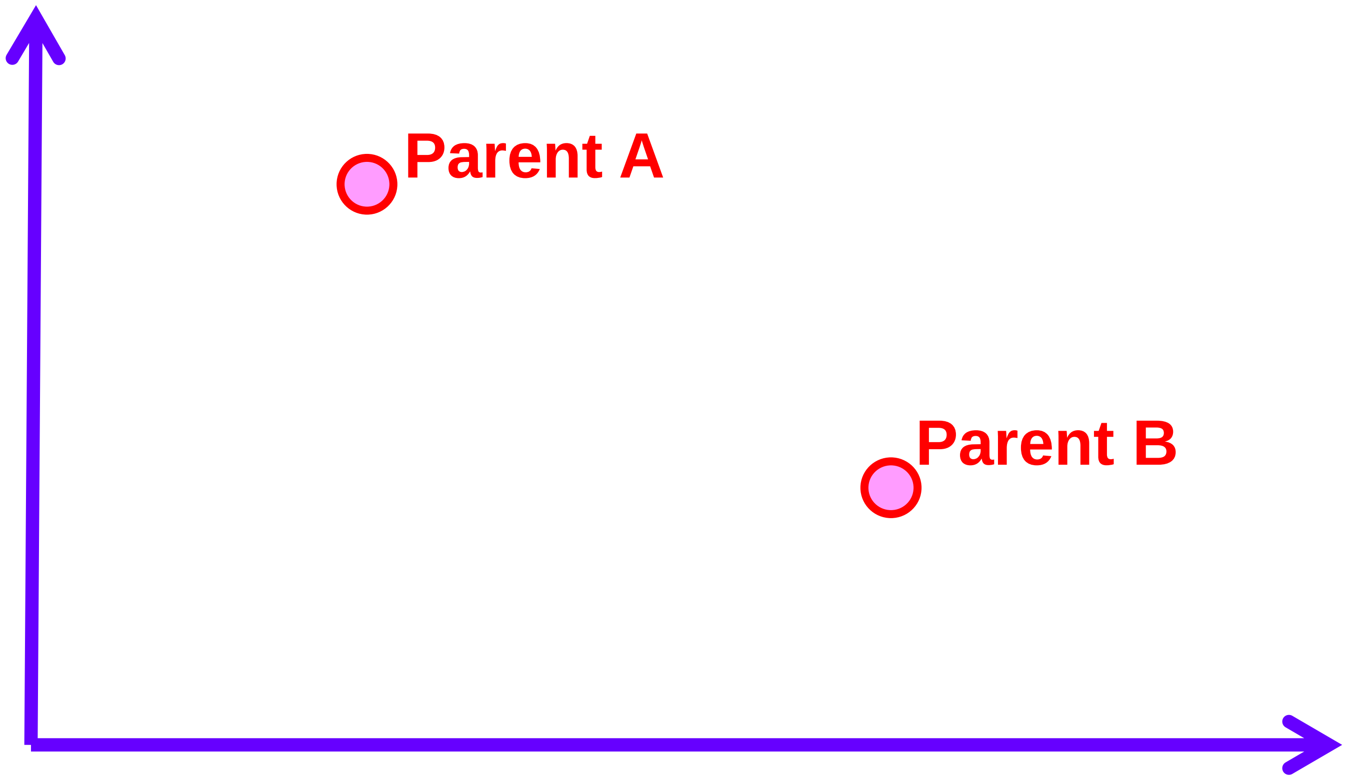
Multiple parents can be also used in calculation:

1.0 x	1.00	0.00	0.00	1.00	2.00	1.00	1.00	10.00	1.00	1.00
+										
0.6 x	1.00	1.00	10.00	1.00	10.00	1.00	10.00	10.00	20.00	1.00
-										
0.6 x	3.50	2.00	8.50	2.00	5.00	2.00	4.50	2.00	5.00	2.00
=										

Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

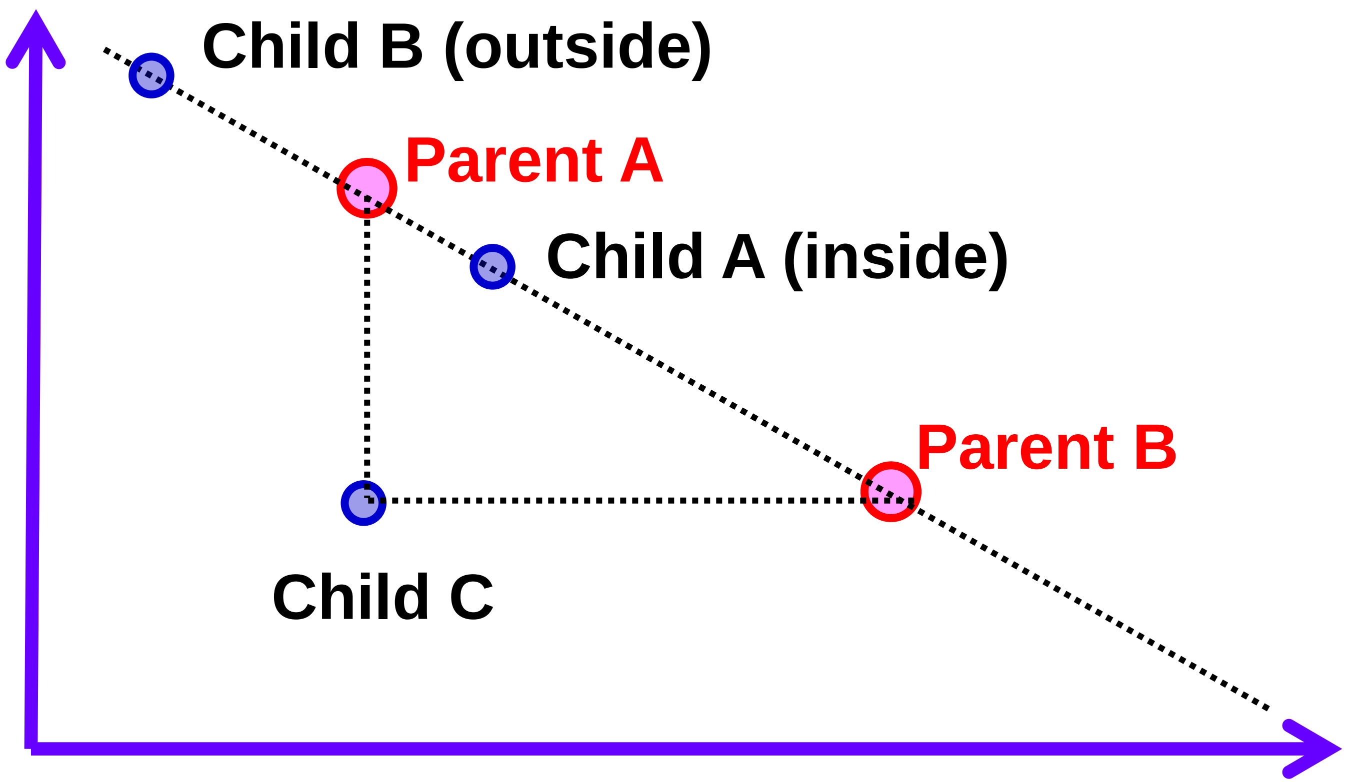
Crossover for real number strings of length 2 can be explained as a mechanism to generate a new point from the existing points in the 2D space.



Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

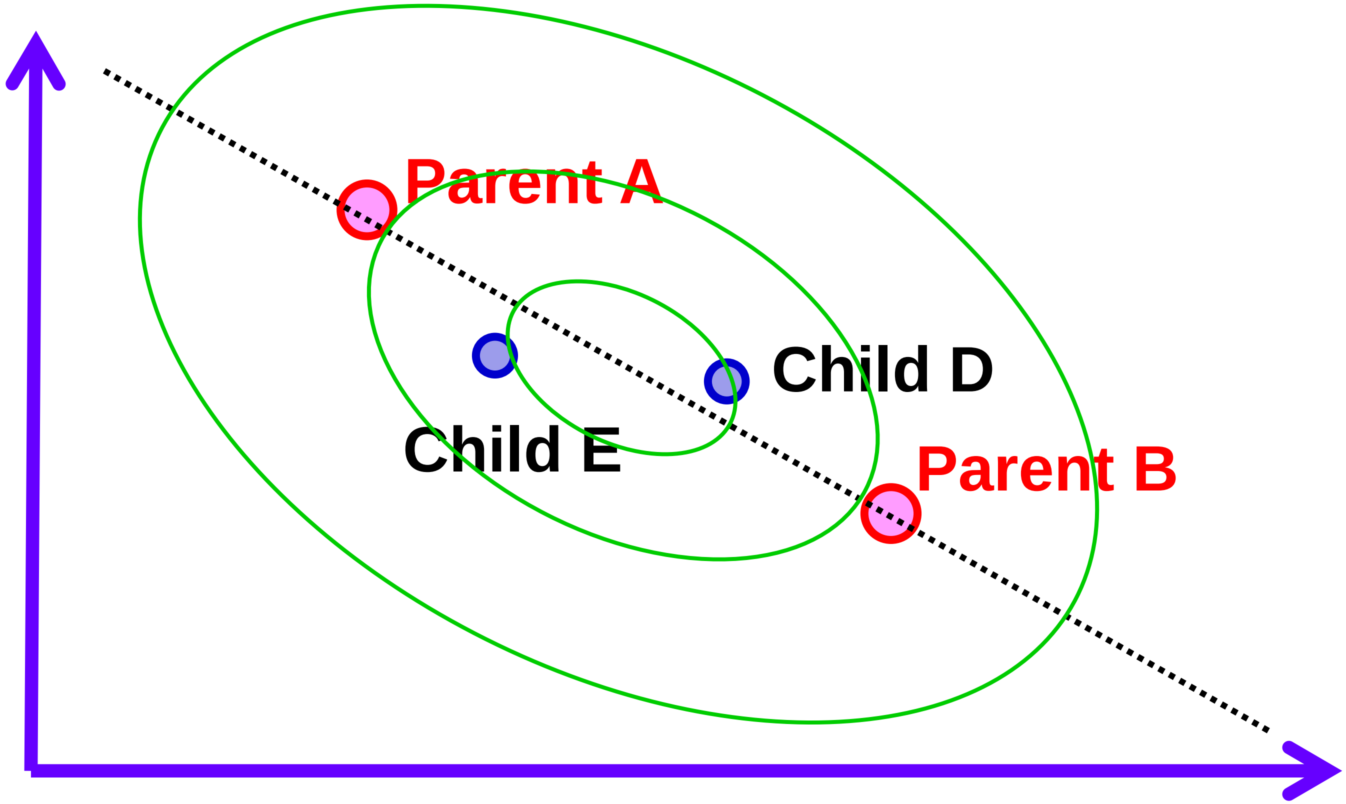
Crossover for real number strings of length 2 can be explained as a mechanism to generate a new point from the existing points in the 2D space.



Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

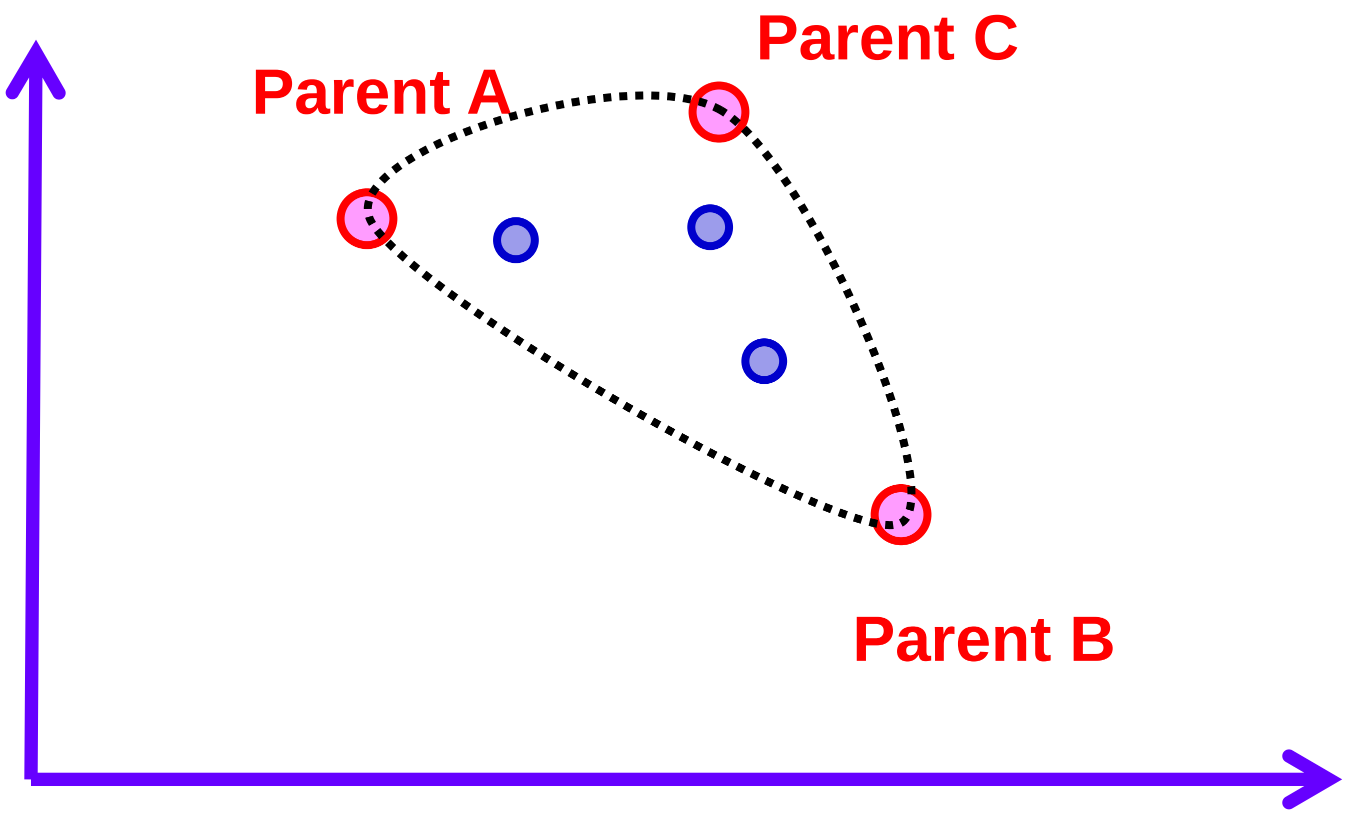
Crossover for real number strings of length 2 can be explained as a mechanism to generate a new point from the existing points in the 2D space.



Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

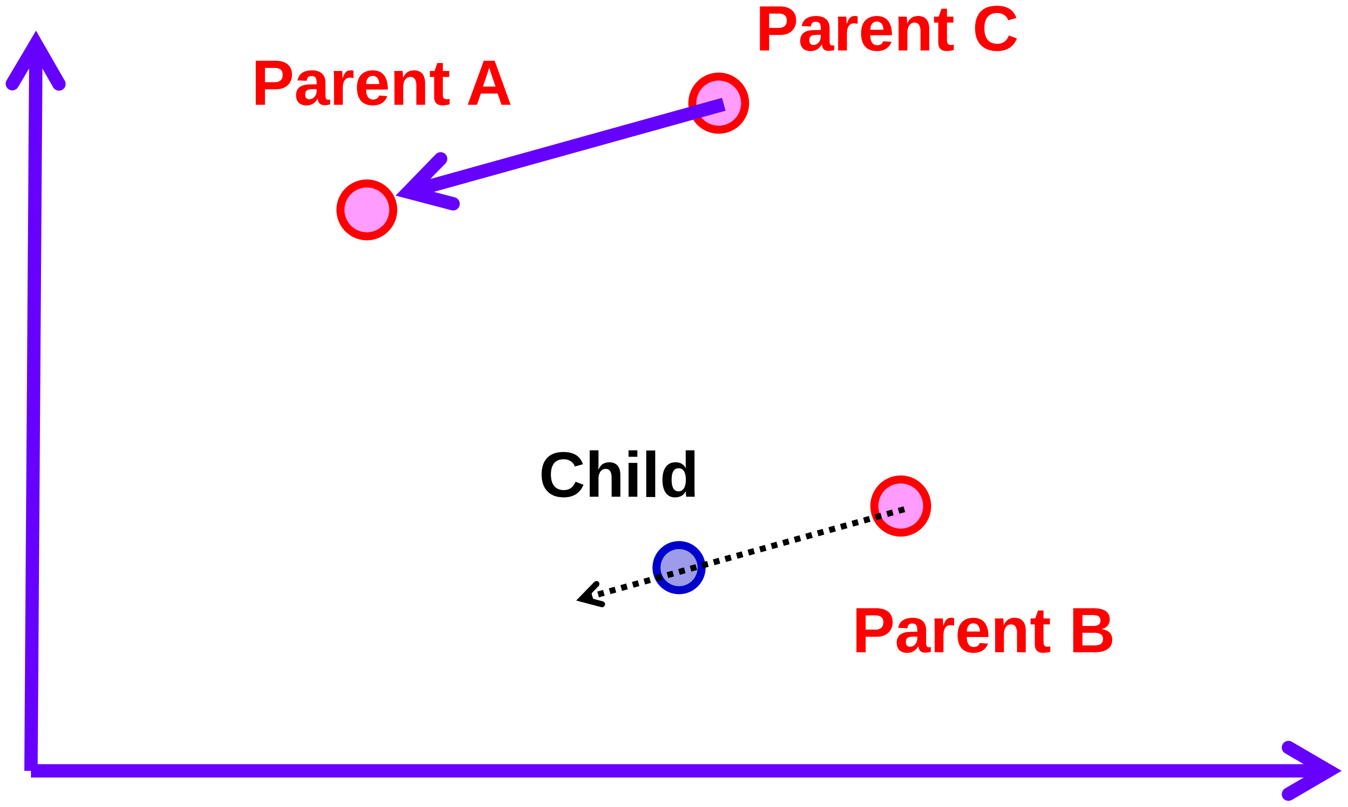
Crossover for real number strings of length 2 can be explained as a mechanism to generate a new point from the existing points in the 2D space.



Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

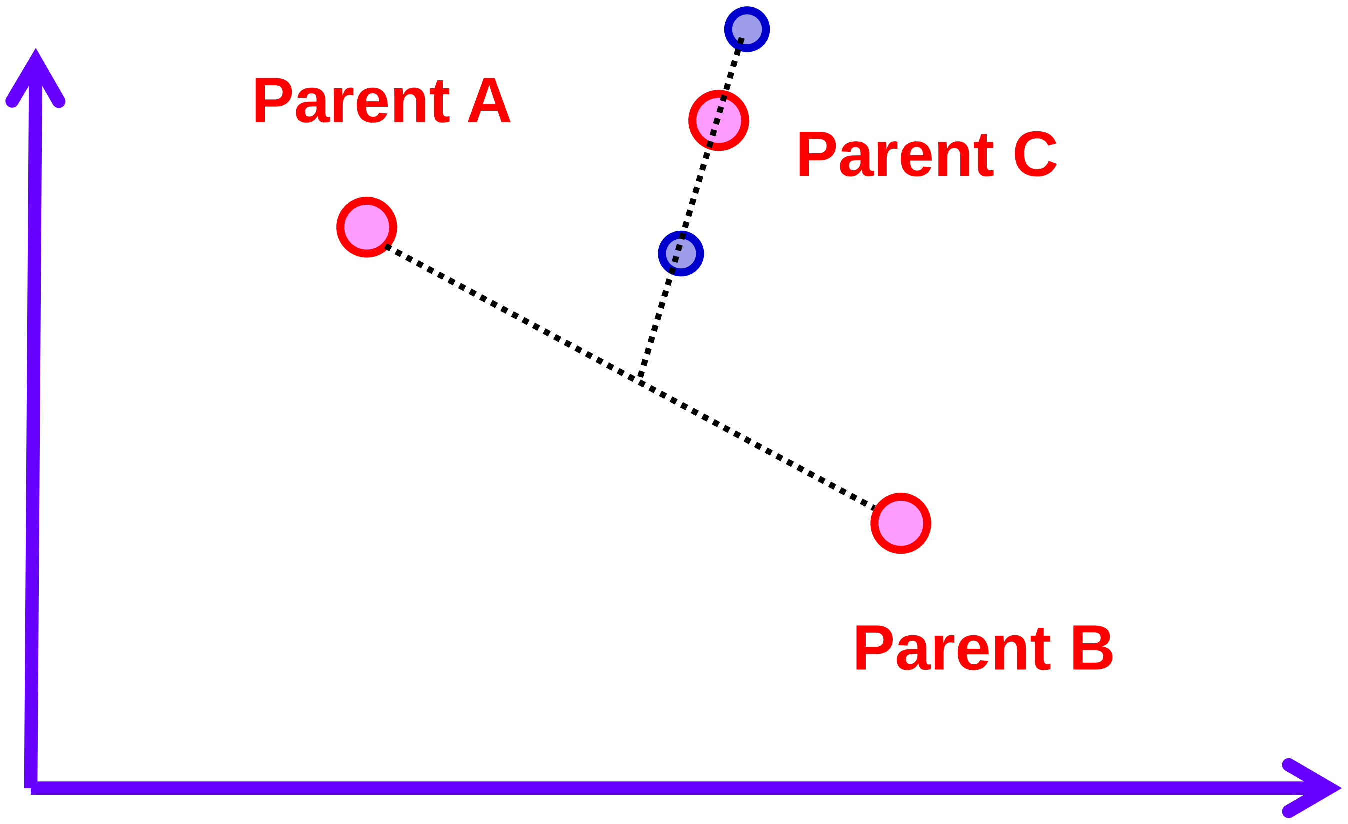
Crossover for real number strings of length 2 can be explained as a mechanism to generate a new point from the existing points in the 2D space.



Crossover: Exchange a part of strings

(with a crossover probability, e.g., 0.5, 0.8, 0.9, 1.0)

Crossover for real number strings of length 2 can be explained as a mechanism to generate a new point from the existing points in the 2D space.



Crossover for Permutation Strings

Simple crossover does not generate a permutation.

Parent A	0	1	2	3	4	5	6	7	8	9
Parent B	1	3	5	7	9	0	2	4	6	8
Child	0	1	2	3	4	5	2	4	6	8

Two “2”, two “4”, no “7”, and no “9”.

Your Idea

Parent A

0	1	2	3	4	5	6	7	8	9

Parent B

1	3	5	7	9	0	2	4	6	8

(1) Child

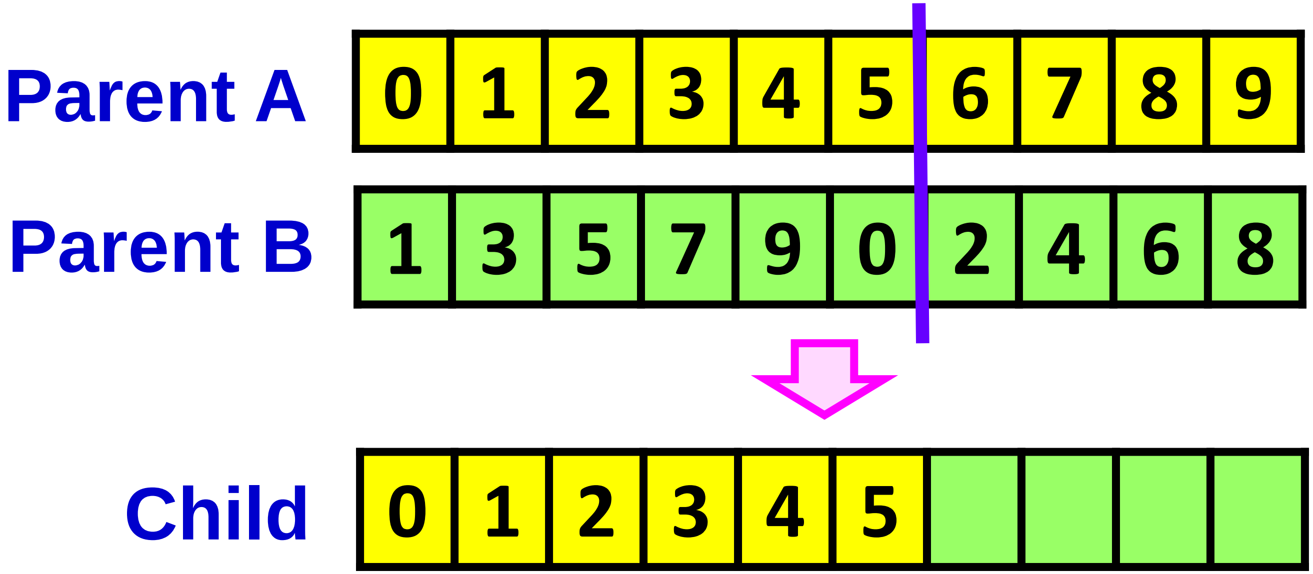
0	1	2	3	4	5	?	?	?	?

Send your answer

Crossover for Permutation Strings

Special Crossover for Permutation Strings:

One-Point Order Crossover



(1) A part of the child directly comes from Parent A.

Crossover for Permutation Strings

Special Crossover for Permutation Strings:

One-Point Order Crossover

Parent A	0	1	2	3	4	5	6	7	8	9
Parent B	1	3	5	7	9	0	2	4	6	8
Child	0	1	2	3	4	5	7	9	6	8

(1) A part of the child directly comes from Parent A.

(2) The other part of the child comes from Parent B in the order in Parent B.

Your Idea

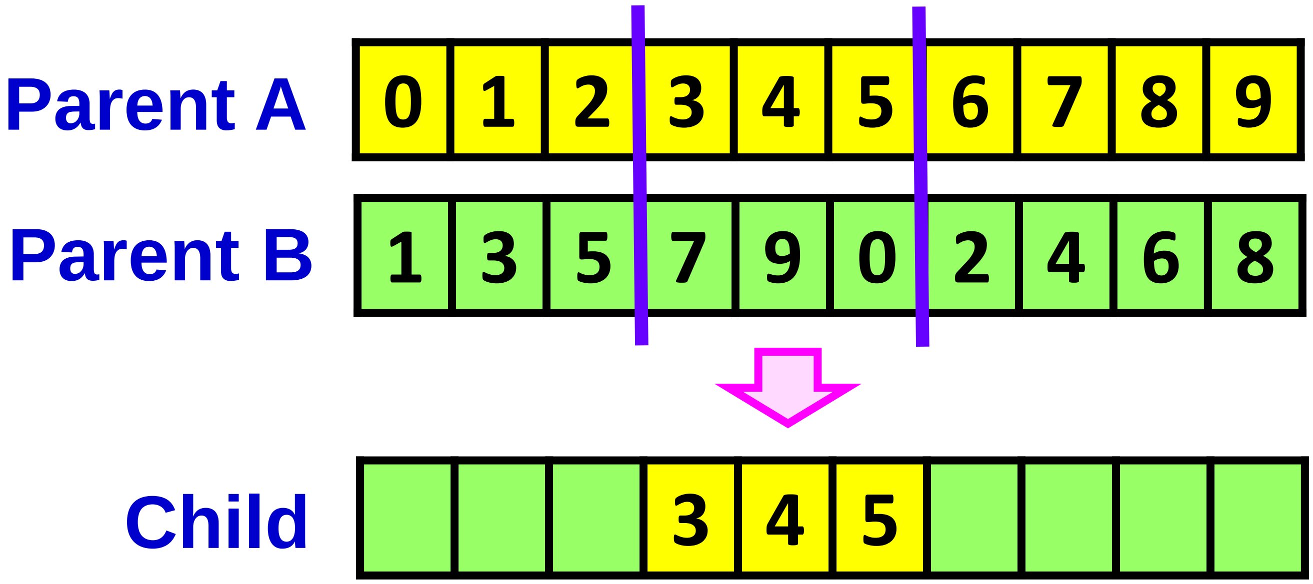
Parent A	0	1	2	3	4	5	6	7	8	9
Parent B	1	3	5	7	9	0	2	4	6	8
(2) Child	?	?	?	3	4	5	?	?	?	?

Send your answer

Crossover for Permutation Strings

Special Crossover for Permutation Strings:

Two-Point Order Crossover



(1) A part of the child directly comes from Parent A.

Crossover for Permutation Strings

Special Crossover for Permutation Strings:

Two-Point Order Crossover

Parent A	0	1	2	3	4	5	6	7	8	9
Parent B	1	3	5	7	9	0	2	4	6	8
Child	1	7	9	3	4	5	0	2	6	8

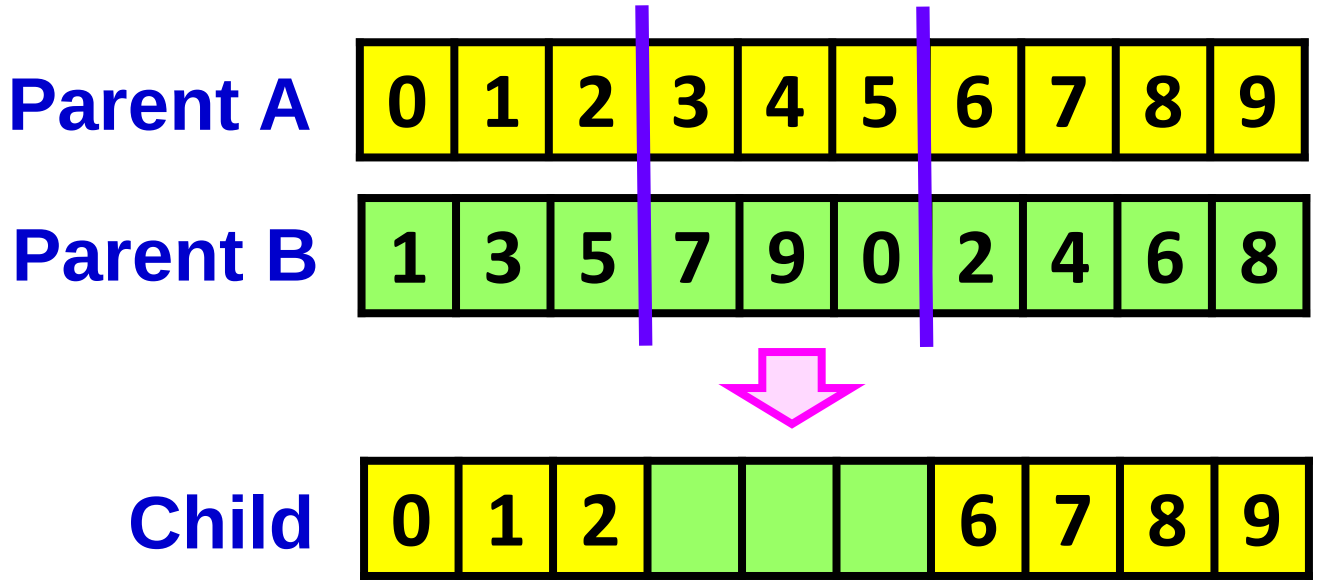
(1) A part of the child directly comes from Parent A.

(2) The other part of the child comes from Parent B in the order in Parent B.

Crossover for Permutation Strings

Special Crossover for Permutation Strings:

Two-Point Order Crossover

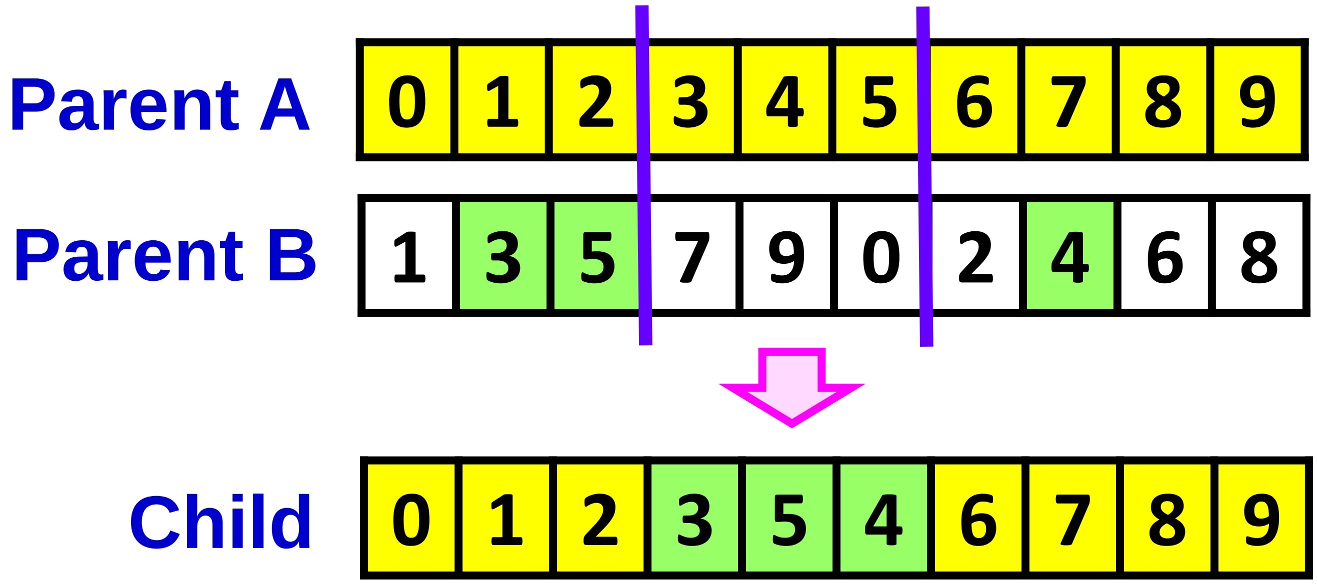


(1) A part of the child directly comes from Parent A.

Crossover for Permutation Strings

Special Crossover for Permutation Strings:

Two-Point Order Crossover



(1) A part of the child directly comes from Parent A.

(2) The other part of the child comes from Parent B in the order in Parent B.

How to evaluate each solution

Solution evaluation depends on the problem.

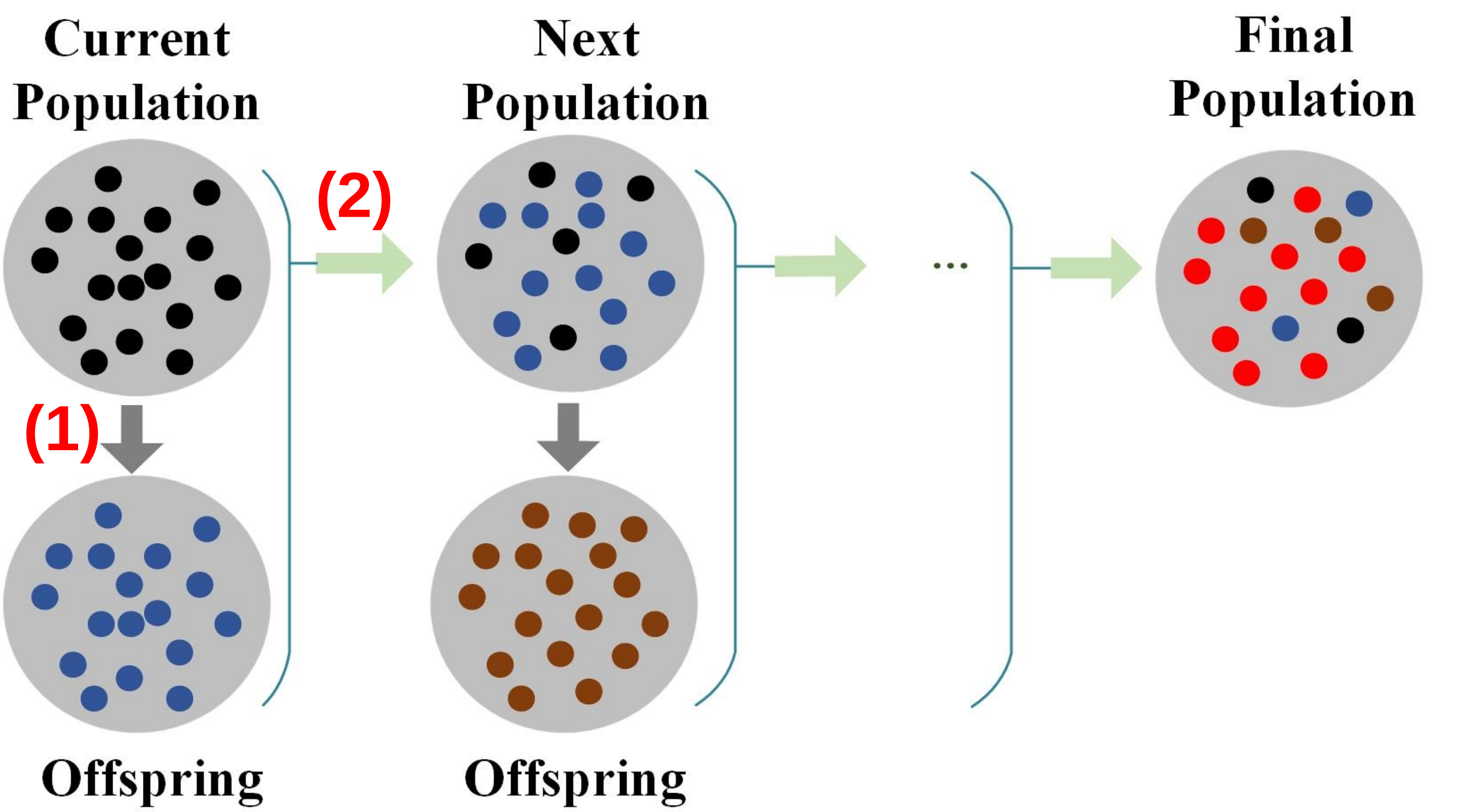
- **Profit maximization problem: Profit**
- **Cost minimization problem: Cost**
- **Distance minimization problem: Distance**
- **Error minimization problem: Error**
- **…**

Selection: How to select good solutions

Good solutions are selected through

(1) selection of parents from current population to generate new solutions

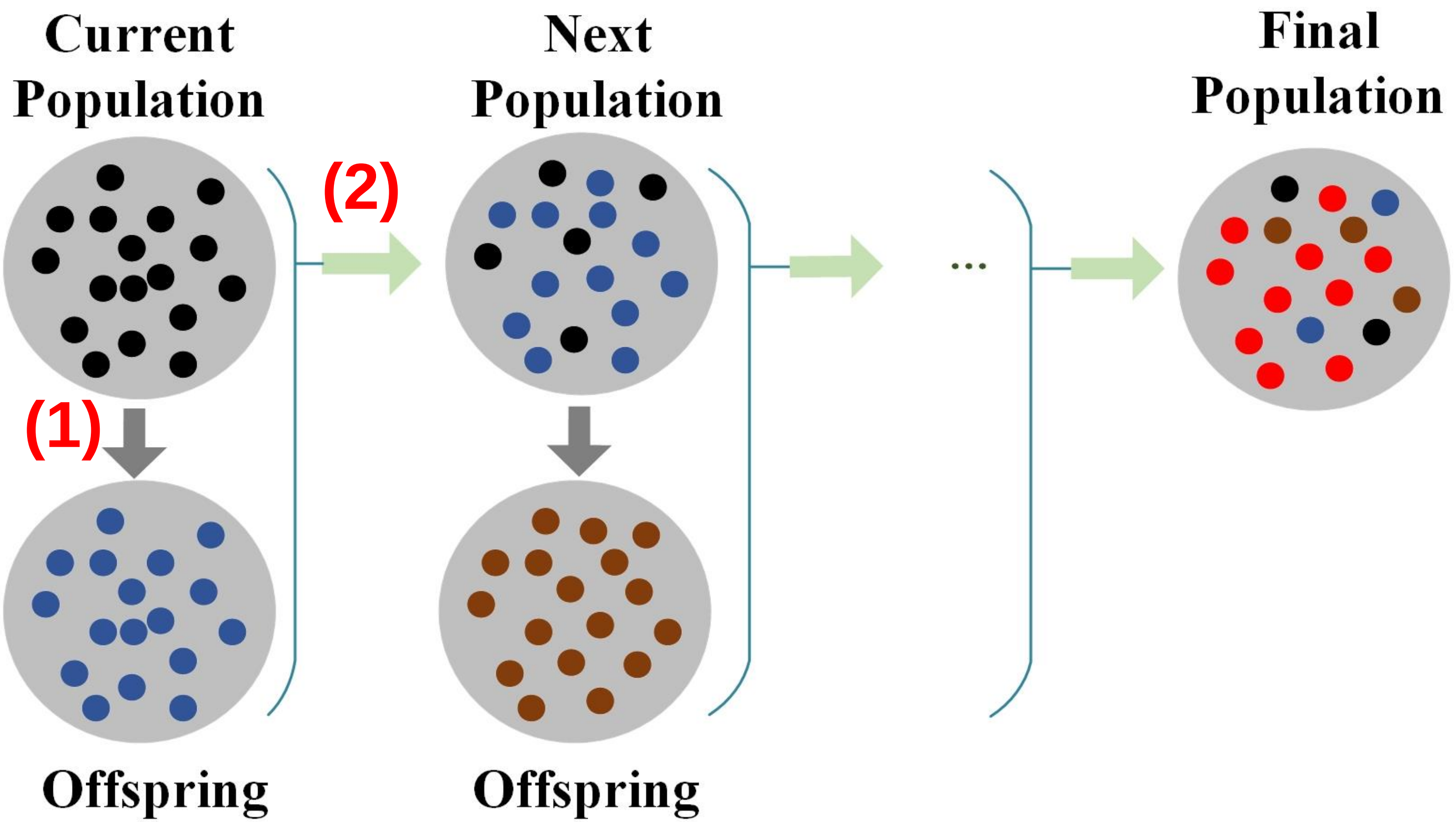
(2) selection of solutions for the next population from the current population and the offspring population.



Two Phases of Selection:

(1) Parent Selection (Mating Selection)

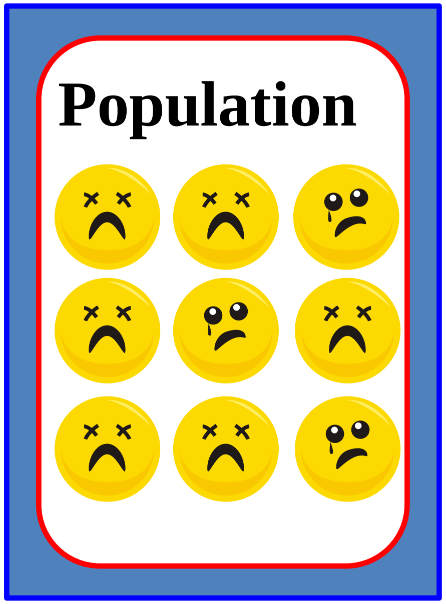
(2) Generation Update (Environmental Selection)



Parent Selection (Mating Selection)

Basic Idea of Parent Selection:

- **Good children will be generated more likely from good parents than poor parents.**
- **So, let us choose good solutions as parents.**



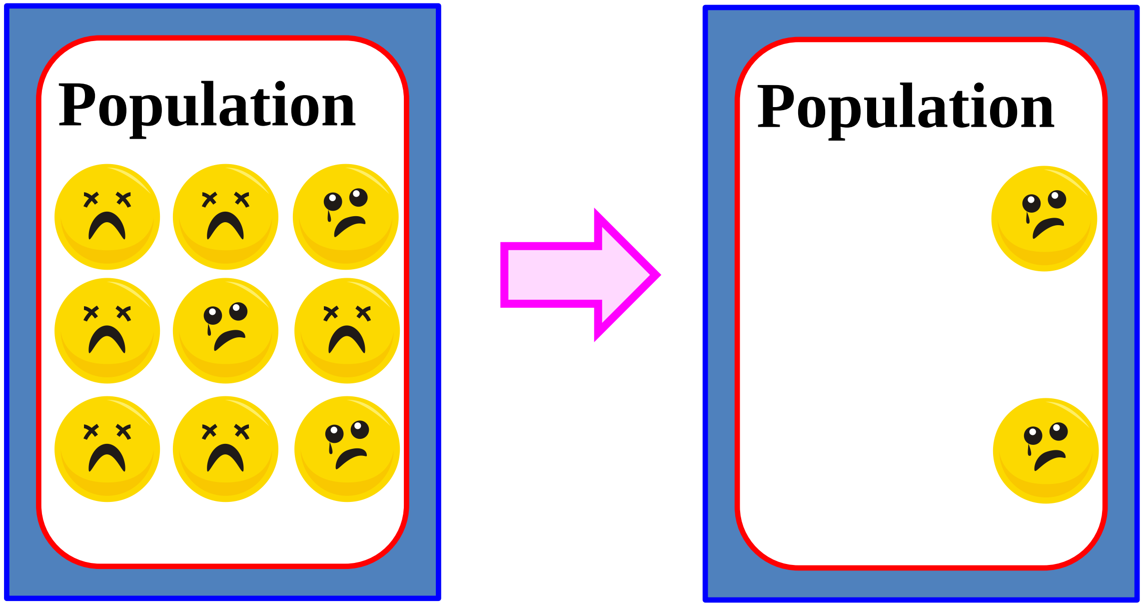
Parent Selection (Mating Selection)

Implementation

The basic idea is simple. However, there are a lot of possible implementations such as

- To choose the best two solutions as parents.

All children are generated from the two solutions.



Parent Selection (Mating Selection)

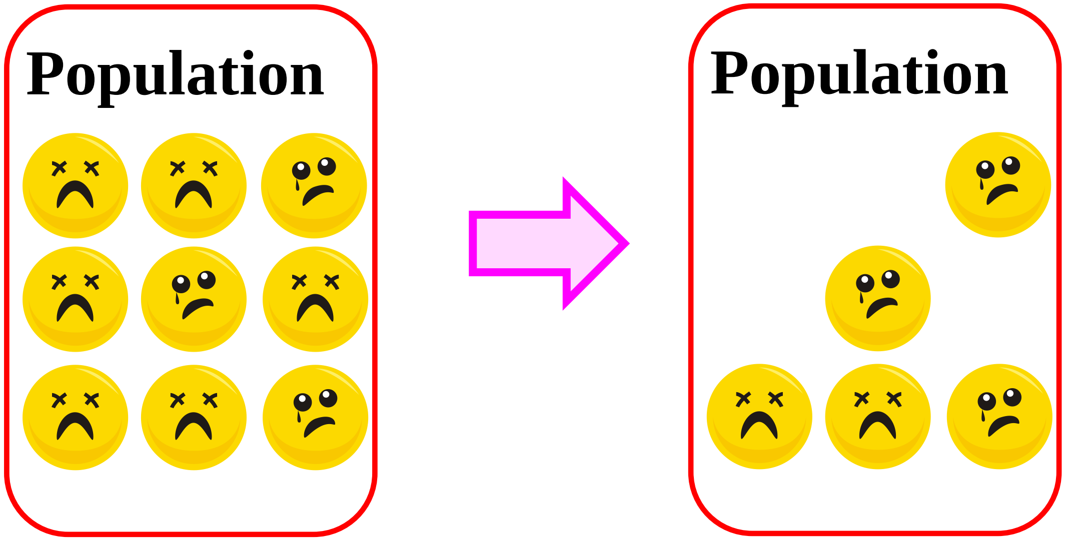
Implementation

The basic idea is simple. However, there are a lot of possible implementations such as

- To choose the best two solutions as parents.

All children are generated from the two solutions.

- To choose a half solutions as parents.



Parent Selection (Mating Selection)

Implementation

The basic idea is simple. However, there are a lot of possible implementations such as

- To choose the best two solutions as parents.

All children are generated from the two solutions.

- To choose a half solutions as parents.

- Assign a different selection probability to each solution depending on the fitness value.

Parent Selection (Mating Selection)

Roulette Wheel Selection (Fitness Proportional Selection)

Current Solutions: $x_1, x_2, ..., x_N$ (N: Population Size)

Their Fitness Values: $fitness(x_1), fitness(x_2), ..., fitness(x_N)$

Their Selection Probabilities: $p(x_1), p(x_2), ..., p(x_N)$

$$p(x_i) = fitness(x_i)/(fitness(x_1) + fitness(x_2) + \ldots + fitness(x_N))$$

Parent Selection (Mating Selection)

Roulette Wheel Selection (Fitness Proportional Selection)

Current Solutions: $x_1, x_2, \ldots, x_N$ (N: Population Size)

Their Fitness Values: $fitness(x_1), fitness(x_2), \ldots, fitness(x_N)$

Their Selection Probabilities: $p(x_1), p(x_2), \ldots, p(x_N)$

$$p(x_i) = fitness(x_i)/(fitness(x_1) + fitness(x_2) + \ldots + fitness(x_N))$$

Example (N = 10): Exercise

$fitness(x_1) = fitness(x_2) = fitness(x_3) = fitness(x_4) = fitness(x_5) = fitness(x_6) = 1$
$fitness(x_7) = fitness(x_8) = fitness(x_9) = 2,\ fitness(x_{10}) = 8$

$p(x_1) = p(x_2) = p(x_3) = p(x_4) = p(x_5) = p(x_6) = \boxed{?}$.
$p(x_7) = p(x_8) = p(x_9) = \boxed{?}$, $p(x_{10}) = \boxed{?}$.

Send your answer

Parent Selection (Mating Selection)

Roulette Wheel Selection (Fitness Proportional Selection)

Current Solutions: $x_1, x_2, ..., x_N$ (N: Population Size)

Their Fitness Values: $fitness(x_1), fitness(x_2), ..., fitness(x_N)$

Their Selection Probabilities: $p(x_1), p(x_2), ..., p(x_N)$

$$p(x_i) = fitness(x_i)/(fitness(x_1) + fitness(x_2) + . . . + fitness(x_N))$$

Example (N = 10): Exercise

$fitness(x_1) = fitness(x_2) = fitness(x_3) = fitness(x_4) = fitness(x_5) = fitness(x_6) = 1$
$fitness(x_7) = fitness(x_8) = fitness(x_9) = 2, \ fitness(x_{10}) = 8$

$p(x_1) = p(x_2) = p(x_3) = p(x_4) = p(x_5) = p(x_6) = 0.05$
$p(x_7) = p(x_8) = p(x_9) = 0.1, \quad p(x_{10}) = 0.4$

Rank Selection

Selection probability is based on the rank of each solution.

Example (N = 10)

$fitness(x_1) = 12$
$fitness(x_2) = 23$
$fitness(x_3) = 43$
$fitness(x_4) = 99$ (Best)
$fitness(x_5) = 11$
$fitness(x_6) = 12$
$fitness(x_7) = 24$
$fitness(x_8) = 56$
$fitness(x_9) = 79$ (3rd)
$fitness(x_{10}) = 92$ (2nd)

Example of Selection Probabilities (N = 10)

Best: 0.19, 2nd: 0.17, 3rd: 0.15, 4th: 0.13, 5th: 0.11
6th: 0.09, 7th: 0.07, 8th: 0.05, 9th: 0.03, 10th: 0.01

Rank Selection

Selection probability is based on the rank of each solution.

Example of Selection Probabilities (*N* = 10)

Best: 0.19, 2nd: 0.17, 3rd: 0.15, 4th: 0.13, 5th: 0.11
6th: 0.09, 7th: 0.07, 8th: 0.05, 9th: 0.03, 10th: 0.01

Example of Selection Probabilities (*N* = 10): Some special cases

Choose the best two solutions:

Best: 0.5, 2nd: 0.5, 3rd: 0, 4th: 0, 5th: 0
6th: 0, 7th: 0, 8th: 0, 9th: 0, 10th: 0

Choose the best half solutions:

Best: 0.2, 2nd: 0.2, 3rd: 0.2, 4th: 0.2, 5th: 0.2
6th: 0, 7th: 0, 8th: 0, 9th: 0, 10th: 0

Rank Selection

Selection probability is based on the rank of each solution.

Example of Selection Probabilities (*N* = 10)

Best: 0.19, 2nd: 0.17, 3rd: 0.15, 4th: 0.13, 5th: 0.11
6th: 0.09, 7th: 0.07, 8th: 0.05, 9th: 0.03, 10th: 0.01

Example of Selection Probabilities (*N* = 10): Some special cases

Choose the best two solutions:

Best: 0.5, 2nd: 0.5, 3rd: 0, 4th: 0, 5th: 0
6th: 0, 7th: 0, 8th: 0, 9th: 0, 10th: 0

Choose the best half solutions:

Best: 0.2, 2nd: 0.2, 3rd: 0.2, 4th: 0.2, 5th: 0.2
6th: 0, 7th: 0, 8th: 0, 9th: 0, 10th: 0

Your Idea: **Send your answer**

Best: **0.?**, 2nd: **0.?**, 3rd: **0.?**, 4th: **0.?**, 5th: **0.?**
6th: **0.?**, 7th: **0.?**, 8th: **0.?**, 9th: **0.?**, 10th: **0.?**

Tournament Selection

Choose the best from randomly selected *K* solutions.
(*K*: Tournament Size)

Iterate the following steps:

Step 1: Randomly select *K* solutions from the current population.
Step 2: Select the best solution among the *K* solutions.

- Selection pressure (probabilities) can be adjusted by *K*.

Large *K* ==> Only very good solutions can be parents.
Small *K* ==> Average solutions can have some probabilities
***K* = 1 ==> Random selection**

Tournament Selection

Choose the best from randomly selected *K* solutions.
(*K*: Tournament Size)

Iterate the following steps:

Step 1: Randomly select K solutions from the current population.
Step 2: Select the best solution among the K solutions.

==> Selection probability of each solution depends on its rank (and the value of *K*).

==> Tournament selection with a specific value of K can be implemented as rank selection by specifying the corresponding selection probability for each rank.

Tournament Selection

Choose the best from randomly selected *K* solutions.
(*K*: Tournament Size)

Iterate the following steps:

Step 1: Randomly select K solutions from the current population **(with duplication or without duplication).**

Step 2: Select the best solution among the K solutions.

==> Selection probability of each solution depends on its rank (and the value of *K*).

Lab Session Task 1:

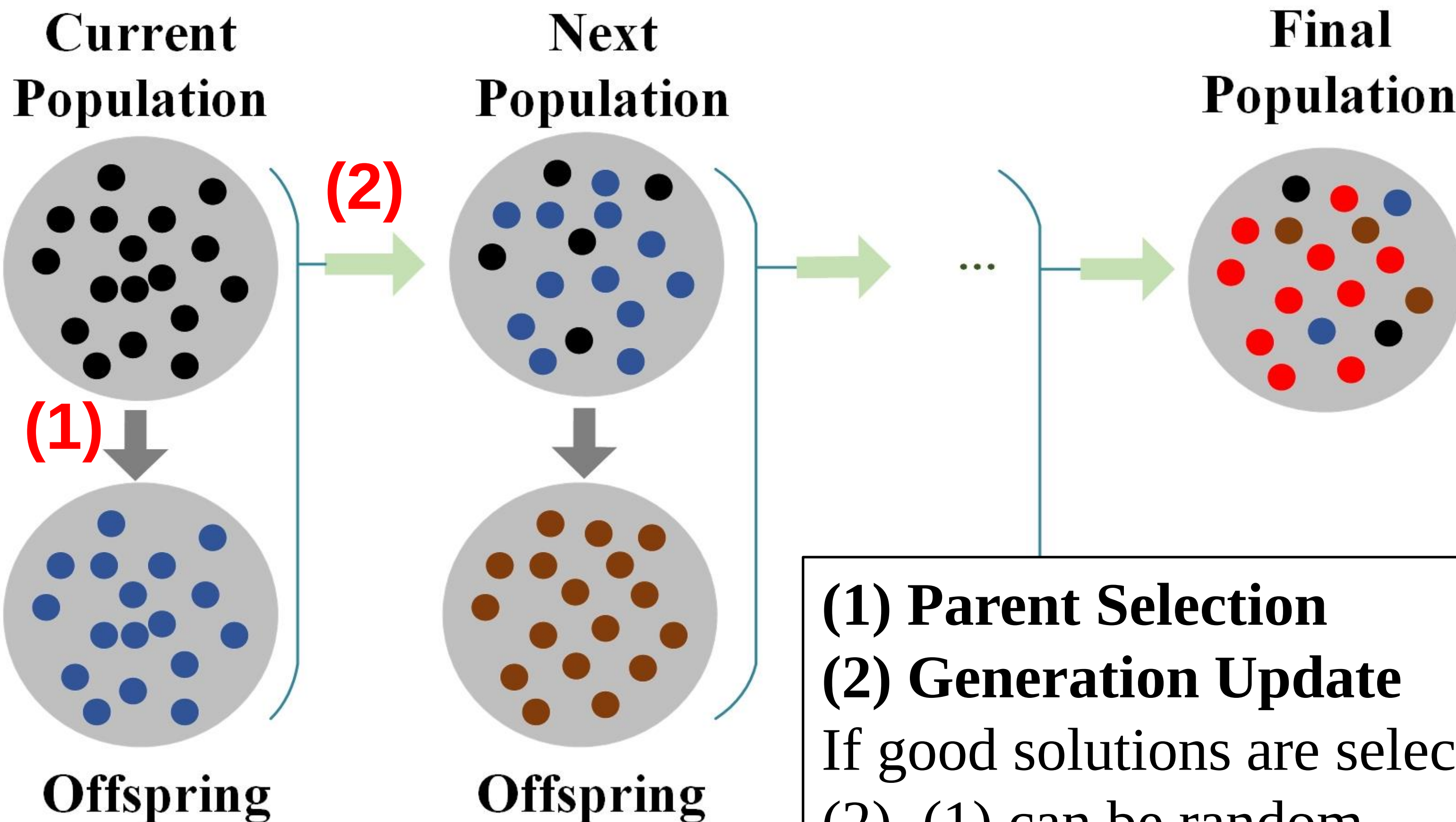
1. Calculate the selection probability of the worst solution for the case of N = 10 (population size) and K = 2 (tournament size) "with duplication".

2. Calculate the selection probability of the i-th worst solution for the case of N = 10 (population size) and K = 2 (tournament size) "with duplication".

3. Calculate the selection probability of the i-th worst solution for the case of N = 10 (population size) and K = 3 (tournament size) "with duplication".

Special Parent Selection Mechanism: Random Selection

Parent can be randomly selected from the current solution.

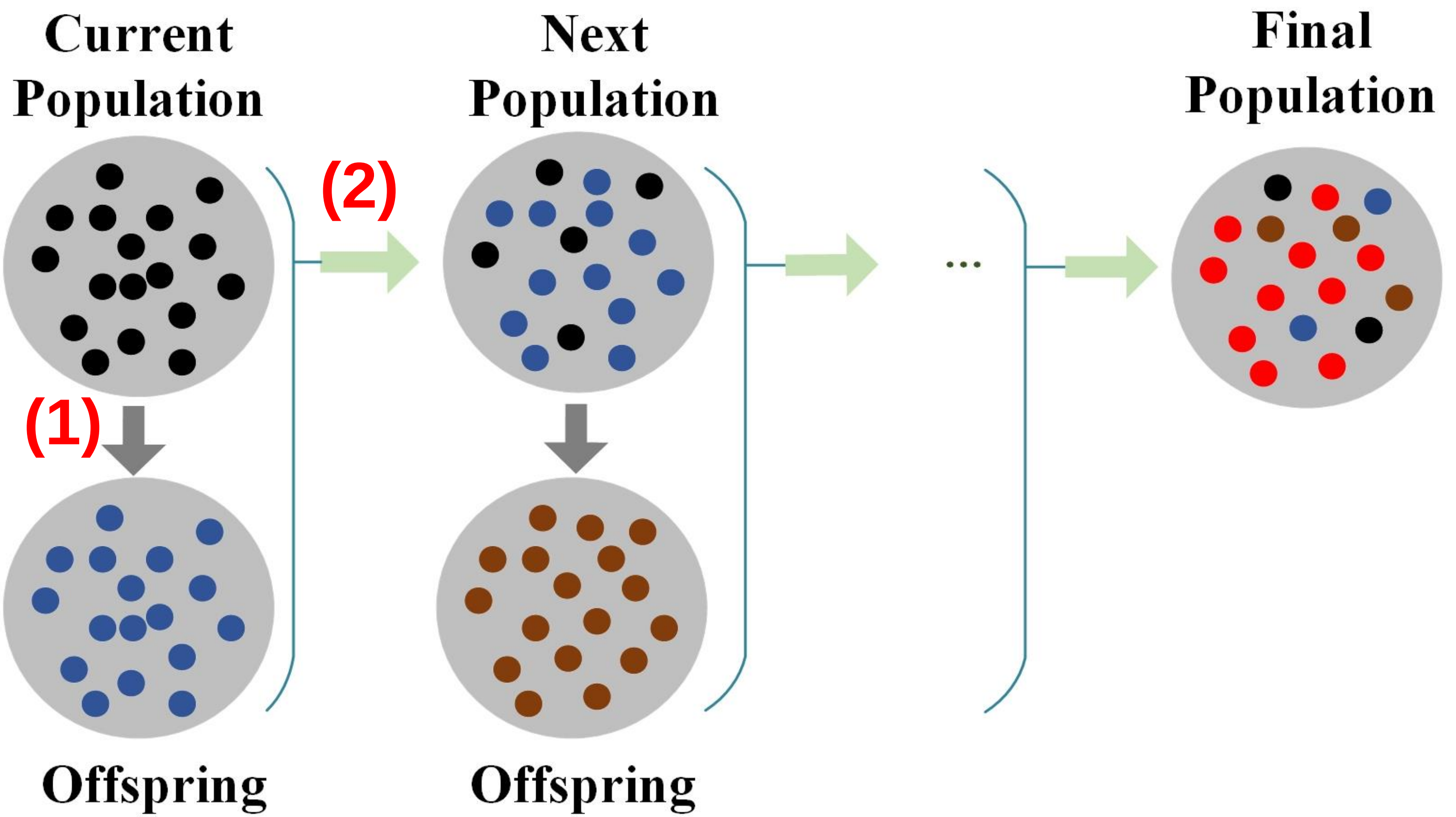


(1) Parent Selection
(2) Generation Update
If good solutions are selected in (2), (1) can be random.

Two Phases of Selection:

(1) Parent Selection (Mating Selection)

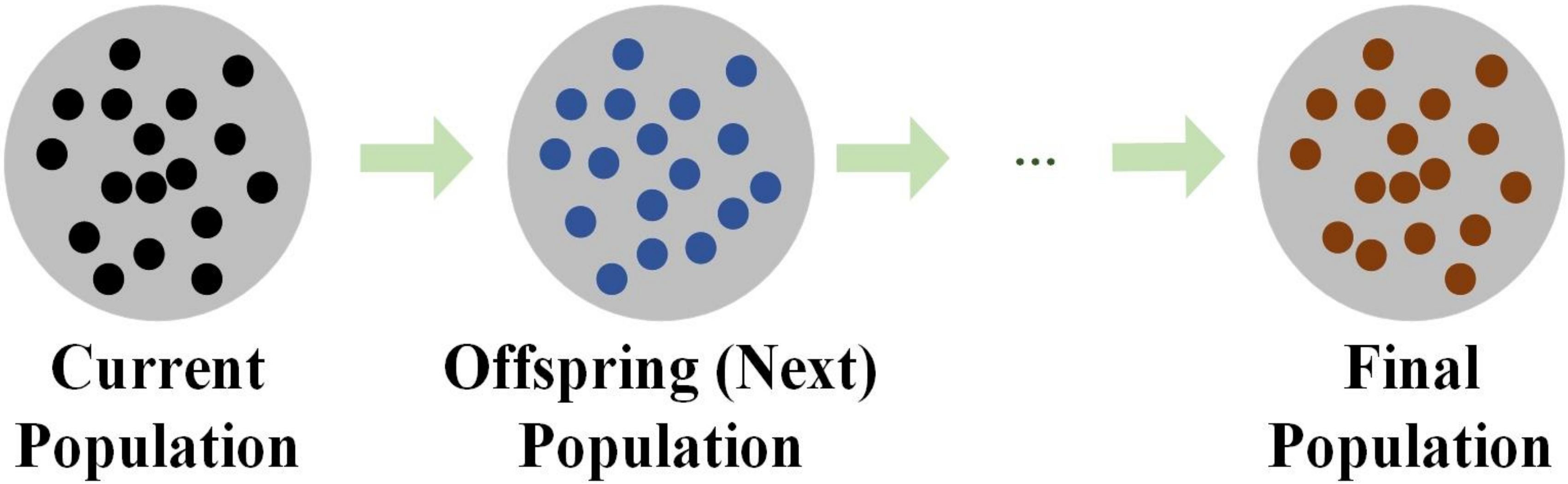
(2) Generation Update (Environmental Selection)



Generation Update (Environmental Selection)

Simplest Model: Next Population = Offspring Population

(This model is similar to evolution of many species in nature)



Advantage: Search can easily escape from local solutions.
Disadvantage: Good solutions cannot be efficiently utilized.

Strong selection pressure may be needed for parent selection.

Generation Update in ES (Evolution Strategies)

Plus Strategy: ($\mu + \lambda$)ES (Fast Convergence)

Select the best μ solutions from the ($\mu + \lambda$) solutions

Comma Strategy: (μ , λ)ES (Escape from Local Solutions)

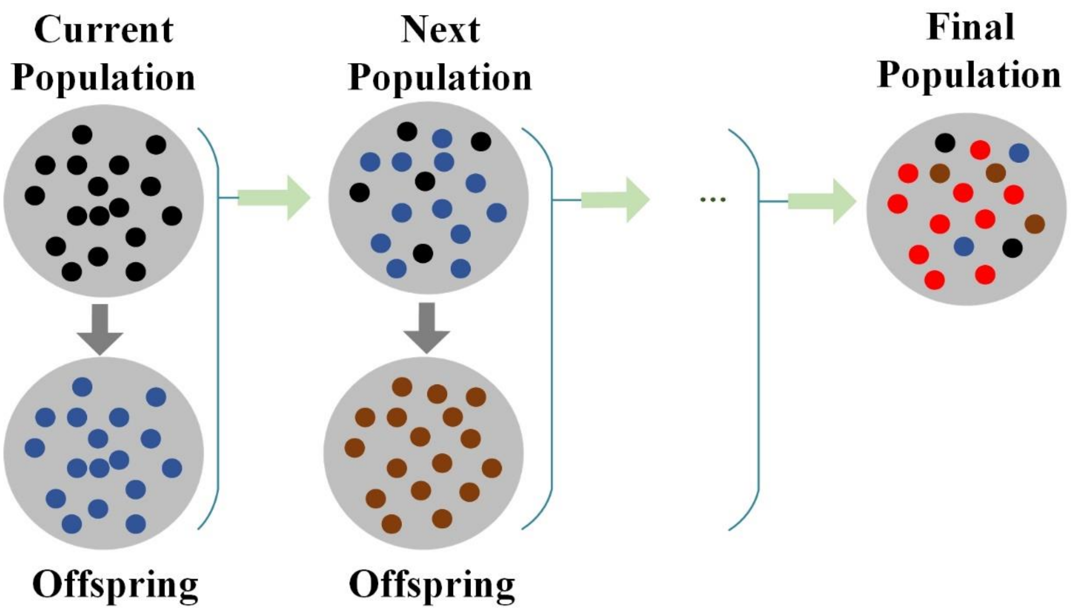
Select the best μ solutions from the λ offspring

m : Population size (Main population size)

l : The number of offspring (Offspring population size)

μ solutions

λ offspring



Generation Update in ES (Evolution Strategies)

Special Cases

(1 , 1)ES: Random search

(1 + 1)ES: First move local search

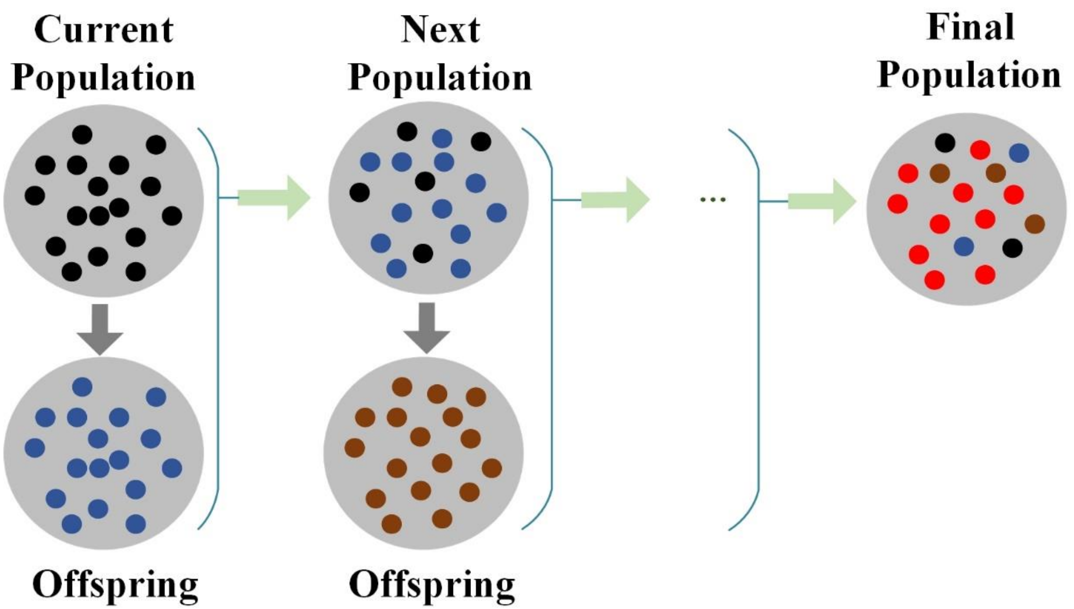
(1 + λ)ES: Local search

(μ + 1)ES: Steady state algorithm

(μ , μ)ES: The simplest model of generation update

μ solutions

λ offspring



Generation Update in GA (Genetic Algorithms)

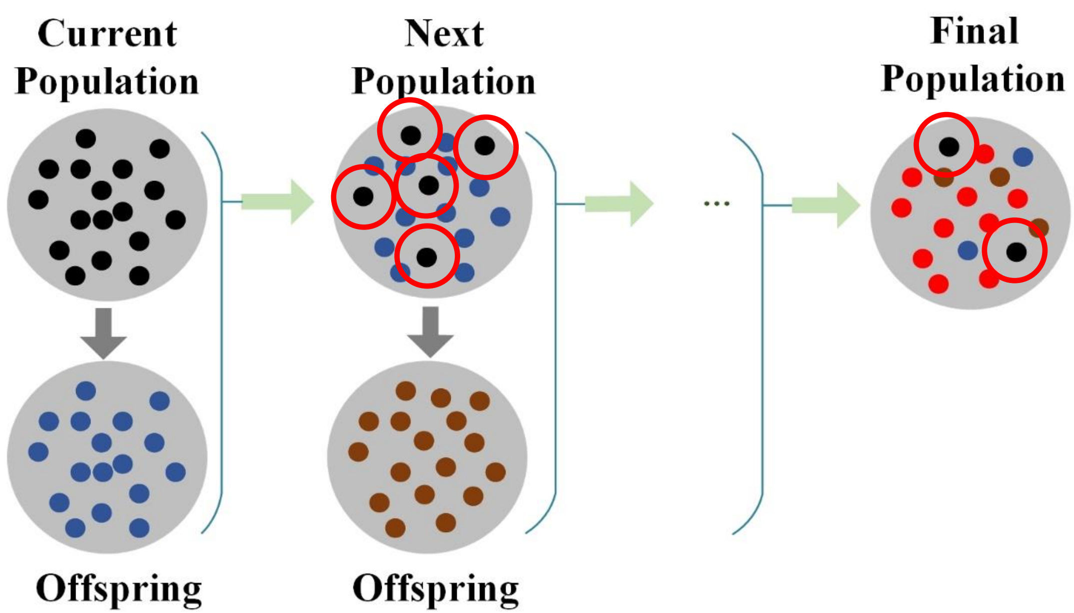
Elite Strategy (Elitist Strategy)

Next population with μ solutions:

- k best solutions in the current population (elite)
- (μ - k) new solutions (offspring)

Best k solutions

Other solutions



Tournament Selection

Choose the best from randomly selected *K* solutions.
(*K*: Tournament Size)

Iterate the following steps:

Step 1: Randomly select K solutions from the current population **(with duplication or without duplication).**

Step 2: Select the best solution among the K solutions.

==> Selection probability of each solution depends on its rank (and the value of *K*).

Lab Session Task 1:

1. Calculate the selection probability of the worst solution for the case of N = 10 (population size) and K = 2 (tournament size) “with duplication”.

2. Calculate the selection probability of the i-th worst solution for the case of N = 10 (population size) and K = 2 (tournament size) “with duplication”.

3. Calculate the selection probability of the i-th worst solution for the case of N = 10 (population size) and K = 3 (tournament size) “with duplication”.

Lab Session Task 2:

In evolutionary computation, which do you think the best generation update mechanism among the following mechanisms. Explain your idea with clear reasons.

- **$(1 , 1)$ES: Random search**
- **$(1 + 1)$ES: First move local search**
- **$(1 + \lambda)$ES: Local search ($\lambda > 1$ such as $\lambda = 5, 10, 50, 50, \ldots$)**
- **$(\mu + 1)$ES: Steady state algorithm ($\mu > 1$ such as $\lambda = 5, 10, 50, \ldots$)**
- **(μ , μ)ES: The simplest model of generation update ($\mu > 1$)**
- **$(\mu + \mu)$ES: The standard model of generation update ($\mu > 1$)**
- **(μ , λ)ES: The standard comma strategy ($\lambda > \mu > 1$ such as $(\mu , \lambda) = (10, 50), (50, 200), (100, 500), \ldots$)**