Genetic Algorithms

Guest Lecturer: **Alaa Khamis**, PhD, SMIEEE

Robotics and AI Consultant at MIO, Inc., InnoVision Systems and Sypron

Smart Mobility Group Coordinator in CPAMI at University of Waterloo

Associate Professor at Suez University and Adjunct Professor at Nile University, Egypt

http://www.alaakhamis.org/

Monday July 25, 2016

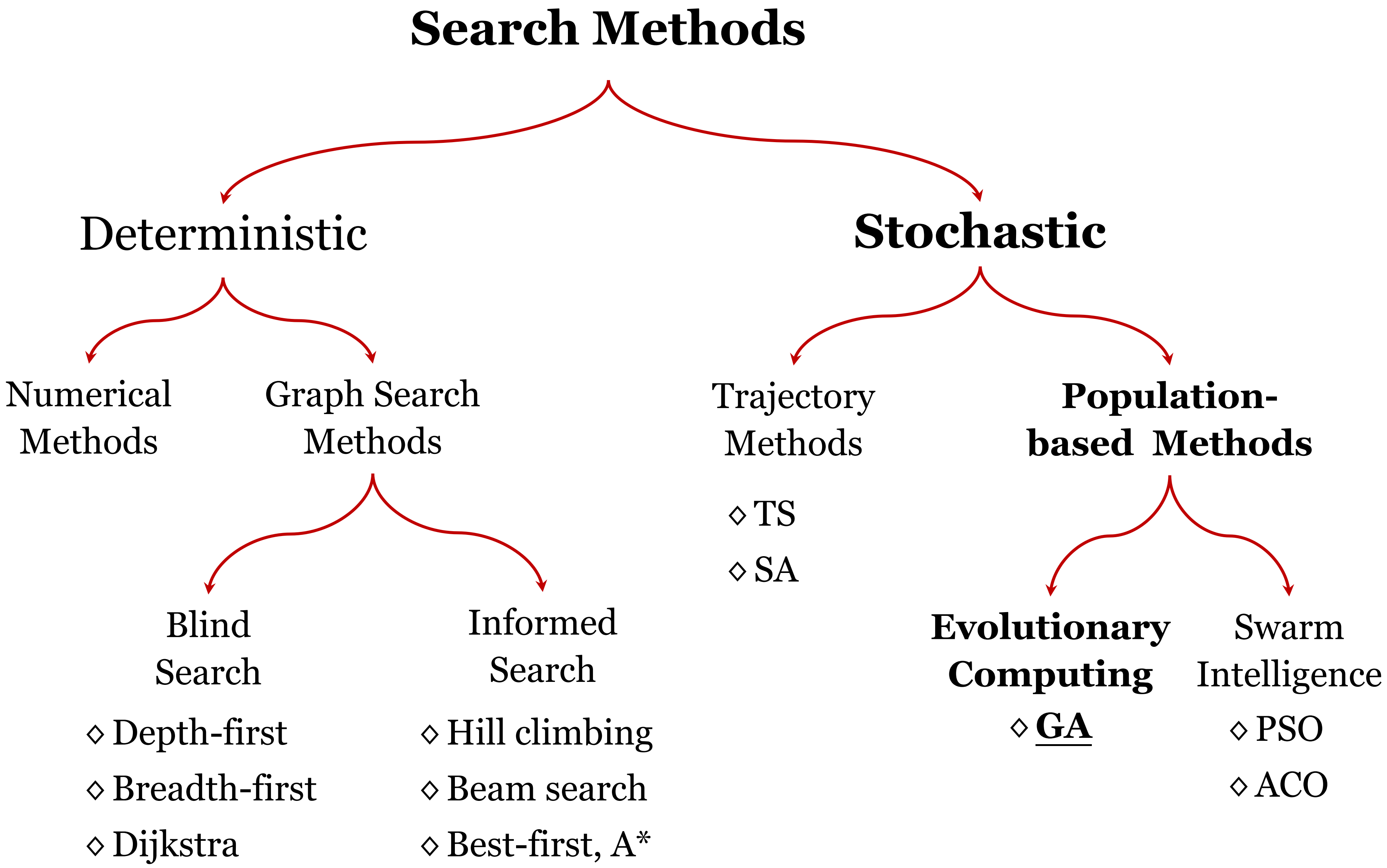
Outline

- Genetic Algorithm
- Fitness Function
- Representation Schemes
- Selection Operators
- Reproduction Operators
- Survivor Selection
- Real-valued GA
- Permutations GA

Outline

- **Genetic Algorithm**
- Fitness Function
- Representation Schemes
- Selection Operators
- Reproduction Operators
- Survivor Selection
- Real-valued GA
- Permutations GA

Genetic Algorithm



Search Methods

Deterministic

Numerical Methods

Graph Search Methods

Blind Search

◇ Depth-first

◇ Breadth-first

◇ Dijkstra

Informed Search

◇ Hill climbing

◇ Beam search

◇ Best-first, A*

Stochastic

Trajectory Methods

◇ TS

◇ SA

Population-based Methods

Evolutionary Computing

◇ **GA**

Swarm Intelligence

◇ PSO

◇ ACO

Genetic Algorithm

- **Definition:**

  The genetic algorithm is a **probabilistic search algorithm** that iteratively transforms a set (called a **population**) of mathematical objects (typically fixed-length binary character strings), each with an associated fitness value, into a new population of offspring objects using the **Darwinian principle of natural selection** and using operations that are patterned after naturally occurring genetic operations, such as **crossover (sexual recombination) and mutation**.

Genetic Algorithm

- Developed by John Holland, University of Michigan (1970's):
  - ◇ To understand the adaptive processes of natural systems.
  - ◇ To design artificial systems software that retains the robustness of natural systems.

John Holland
(1929-)
American scientist

- The original form of the GA, as illustrated by John Holland in 1975 had distinct features: **a bit string representation**, **proportional selection** and **cross-over** to produce new individuals.
- Several variations to the original GA have been developed using different representation schemes, selection and reproduction operators.

Genetic Algorithm

- **Basic Idea:**
  - ◊ A class of probabilistic optimization algorithms
  - ◊ Inspired by the biological evolution process
  - ◊ Uses concepts of "Natural Selection" and "Genetic Inheritance" (Darwin 1859).
  - ◊ Take a population of candidate solutions to a given problem.
  - ◊ Use operators inspired by the mechanisms of natural genetic variation.
  - ◊ Apply selective pressure toward certain properties.
  - ◊ Evolve a more fit solution.

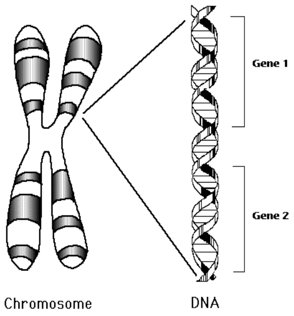
Genetic Algorithm

- **Genetic Algorithm vs. Nature:**

Nature	GA
Environment	Optimization Problem
Individual living in that environment	Feasible solutions
Individual's degree of adaptation to its surrounding environment	Solutions quality (fitness function)
A population of organisms (species)	A set of feasible solutions.
Selection, recombination and mutation in nature's evolutionary process	Stochastic operators
Evolution of populations to suit their environment	Iteratively applying a set of stochastic operators on a set of feasible solutions.

Genetic Algorithm

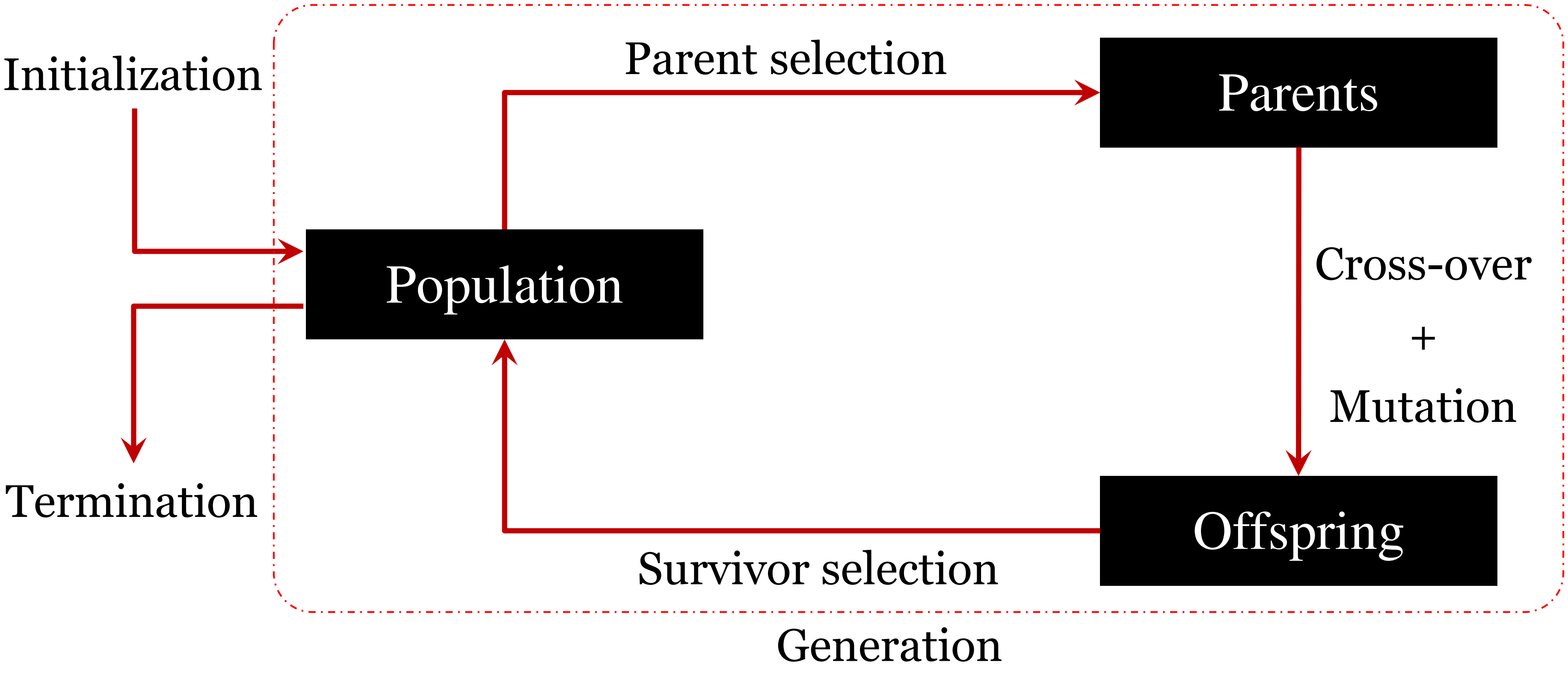
- **Genetic Algorithm vs. Natural Evolution:**



Natural Evolution		GA
Chromosome	⟺	Bitstring 1 0 0 1 sequence of on/off or the number 9
Gene	⟺	String position
Allele	⟺	Value at a certain position
Fitness	⟺	Objective function value
Recombination (cross-over)	⟺	Using a specific cross-over point 1 | 0 0 1 ↓ × 0 | 1 1 0 ↑ ⇒ 1 1 1 0 and 0 0 0 1
Mutation	⟺	0 0 0 1 ⇒ 0 1 0 1

Genetic Algorithm

- **The Algorithm:**



Genetic Algorithm

- **The Algorithm:**
  - ◇ Encoding the objectives or optimization functions;
  - ◇ Defining a fitness function or selection criterion;
  - ◇ Initializing a population of individuals;
  - ◇ Evaluating the fitness of all the individuals in the population;
  - ◇ Creating a new population by performing crossover, and mutation, fitness-proportionate reproduction etc.;
  - ◇ Evolving the population until certain stopping criteria are met;
  - ◇ Decoding the results to obtain the solution to the problem.

Genetic Algorithm

- **The Algorithm:**

```
Initialize population with random candidates,
Evaluate all individuals,
While termination criteria is not met
        Select parents,
        Apply crossover,
        Mutate offspring,
        Replace current generation,
end while
```

Genetic Algorithm

- **The Algorithm:**

  The **termination criteria** could be:

  ◇ A specified number of generations or fitness evaluations (100 or 150 generations),

  ◇ A minimum threshold reached,

  ◇ No improvement in the best individual for a specified number of generations,

  ◇ Memory/time constraints,

  ◇ Combinations of the above.

Genetic Algorithm

- **Main Features:**
  - ◇ Typically applied to:
    - highly multimodal functions,
    - discrete or discontinuous functions,
    - high-dimensionality functions, including many combinatorial ones, and
    - in cases of non-linear dependencies between parameters.
  - ◇ Traditionally emphasizes combining information from good parents (crossover).
  - ◇ Many variants, e.g., reproduction models, operators.
  - ◇ Not too fast.

Genetic Algorithm

- **Benefits of GA:**
  - ◇ Concept is easy to understand.
  - ◇ Modular, separate from application.
  - ◇ Supports multi-objective optimization.
  - ◇ Good for “noisy” environments.
  - ◇ Always an answer; answer gets better with time.
  - ◇ Inherently parallel; easily distributed.
  - ◇ Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained.

Genetic Algorithm

- **Benefits of GA (cont’d):**
  - ◇ Easy to exploit previous or alternate solutions.
  - ◇ Flexible building blocks for hybrid applications.
  - ◇ Substantial history and range of use.

Genetic Algorithm

- **Applications of GA:**

Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner’s dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph coloring and partitioning

Outline

- Genetic Algorithm
- **Fitness Function**
- Representation Schemes
- Selection Operators
- Reproduction Operators
- Survivor Selection
- Real-valued GA
- Permutations GA

Fitness Function

- As mentioned earlier, GAs mimic the **survival-of-the-fittest** principle of nature to make a search process.
- Therefore, GAs are naturally suitable for **solving maximization problems**.
- **Minimization problems** are usually **transformed into maximization problems** by suitable transformation.
- In general, a **fitness function** is first derived from the **objective function** and used in successive genetic operations.
- **Fitness** in biological sense is a **quality value** which is a measure of the **reproductive efficiency of chromosomes**.

[1]

Fitness Function

- In genetic algorithm, **fitness** is used to allocate reproductive traits to the individuals in the population and thus act as some **measure of goodness** to be maximized.
- This means that individuals with **higher fitness** value will have **higher probability** of being selected as candidates for further examination.
- Certain **genetic operators** require that the fitness function be **non-negative**, although certain operators need not have this requirement.
- For **maximization problems**, the **fitness function** can be considered to be the same as the **objective function**.

[1]

Fitness Function

- For **minimization problems**, to generate **non-negative values** in all the cases and to reflect the relative fitness of individual string, it is necessary to **map** the underlying natural **objective function** to **fitness function** form.

Minimization Problem → **Maximization Problems**

$$O(x) \longrightarrow f(x) = \frac{1}{1+O(x)}$$

This transformation does not alter the **location of the minimum**, but converts a minimization problem to an equivalent maximization problem.

[1]

Fitness Function

Minimization Problem → **Maximization Problems**

$$O(x) \longrightarrow f(x) = V - \frac{O(i) \times N}{\sum_{i=1}^{N} O(i)}$$

- ◇ where, *O(i)* is the objective function value of individual *i*,
- ◇ N is the population size and
- ◇ V is a large value to ensure non-negative fitness values.
- ◇ The value of V can be **the maximum value** of the **second term of the equation** so that the fitness value corresponding to maximum value of the objective function is **zero**.

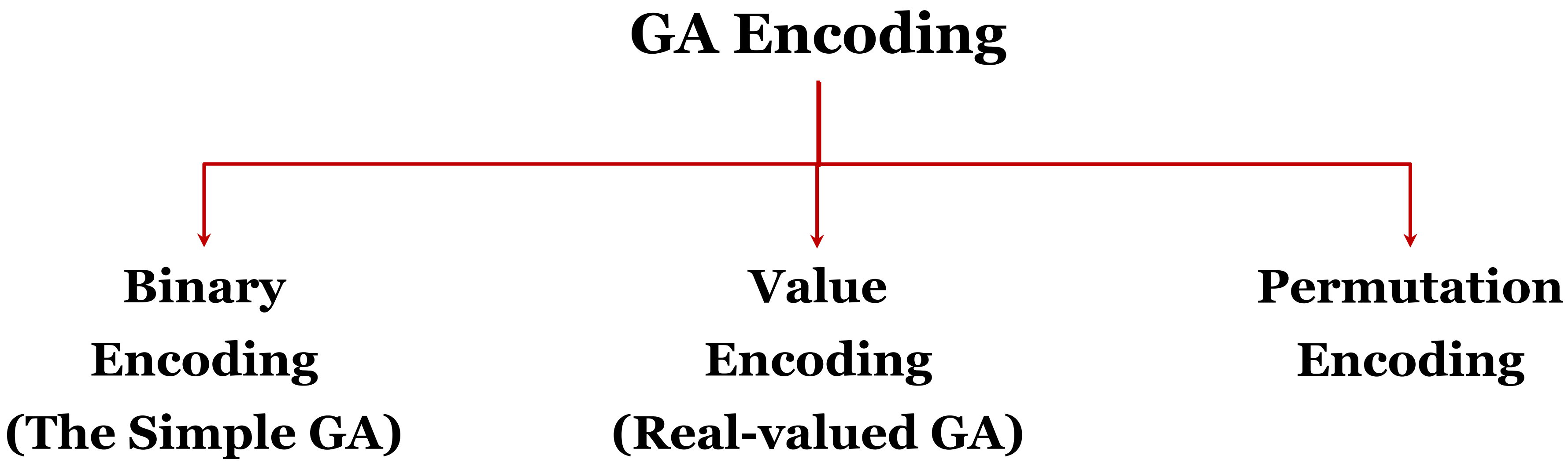
[1]

Outline

- Genetic Algorithm
- Fitness Function
- **Representation Schemes**
- Selection Operators
- Reproduction Operators
- Survivor Selection
- Real-valued GA
- Permutations GA

Representation Schemes

- **Parameters of the solution** (genes) are **concatenated** to form a **string** (chromosome).
- Encoding is a **data structure** for representing **candidate solutions**.
- Good **coding** is probably the **most important** factor for the performance of a GA.



GA Encoding

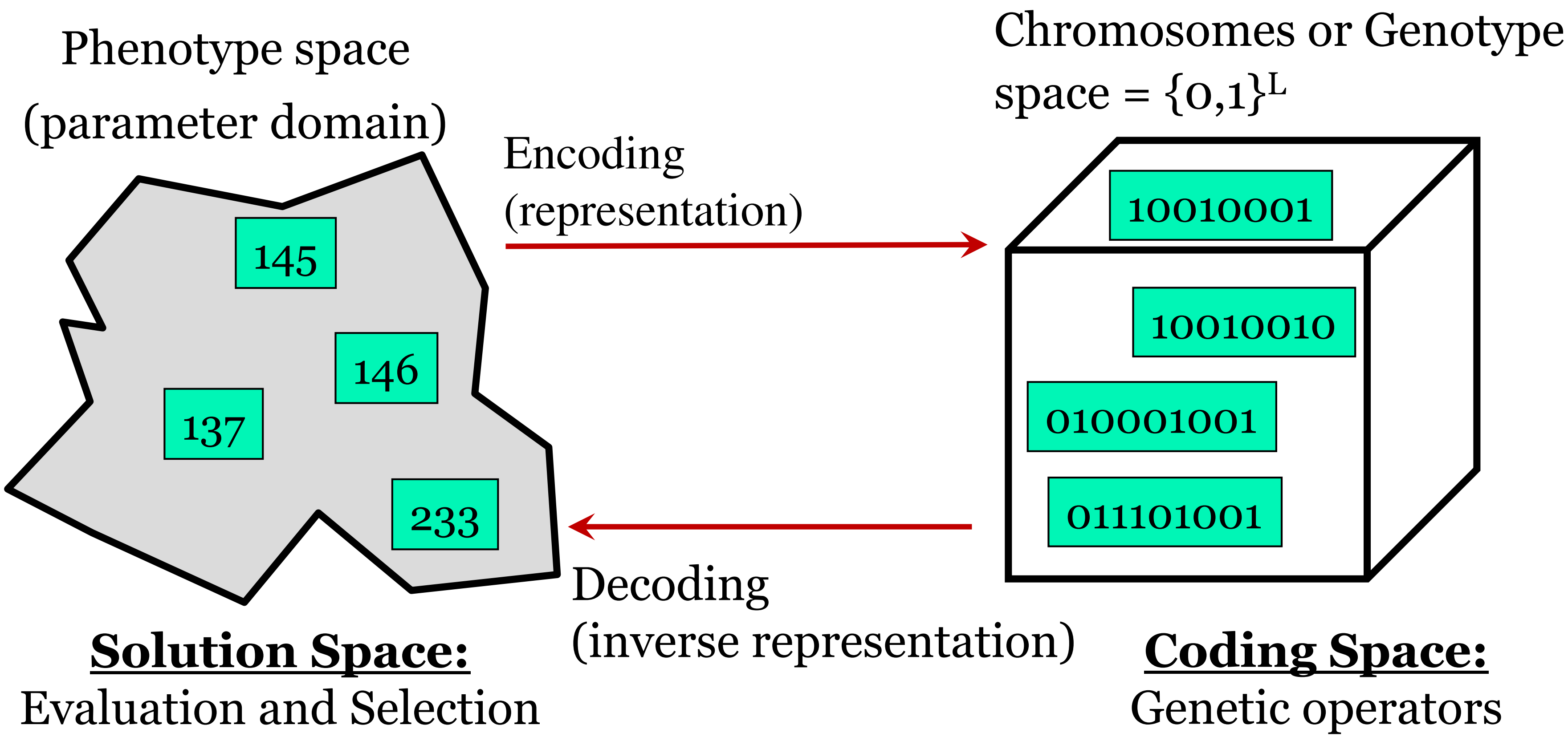
Binary Encoding (The Simple GA)

Value Encoding (Real-valued GA)

Permutation Encoding

Representation Schemes

- **Binary Encoding**



Phenotype space

(parameter domain)

Chromosomes or Genotype space = $\{0,1\}^L$

Encoding (representation)

Decoding (inverse representation)

Solution Space:
Evaluation and Selection

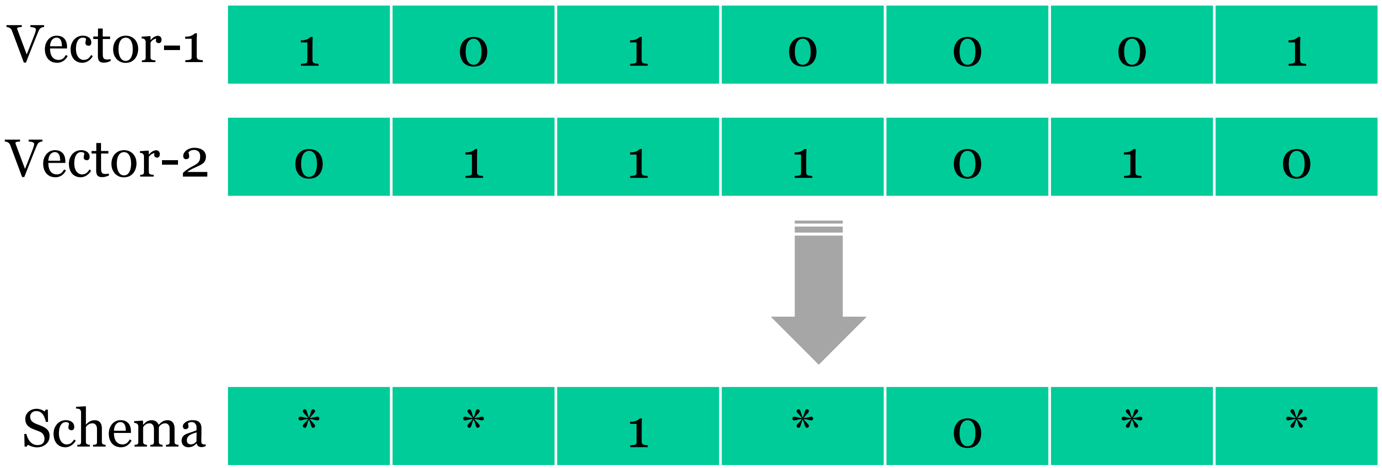
Coding Space:
Genetic operators

◇ A mapping is applied from the parameter domain to the binary domain. In SGA Genotype, is a string in the binary domain.

Representation Schemes

- **Binary Encoding: Schema Theorem**

Schema comes from Greek word ***echo*** that means **shape** or **to form**. Plural is **schemata**.



Vector-1	1	0	1	0	0	0	1
Vector-2	0	1	1	1	0	1	0
Schema	*	*	1	*	0	*	*

* represents a wild card, i.e. it can be replaced by a 0 or a 1.

Representation Schemes

- **Binary Encoding: Schema Theorem (cont'd)**

Schemata can be thought of defining **subsets** of similar chromosomes.

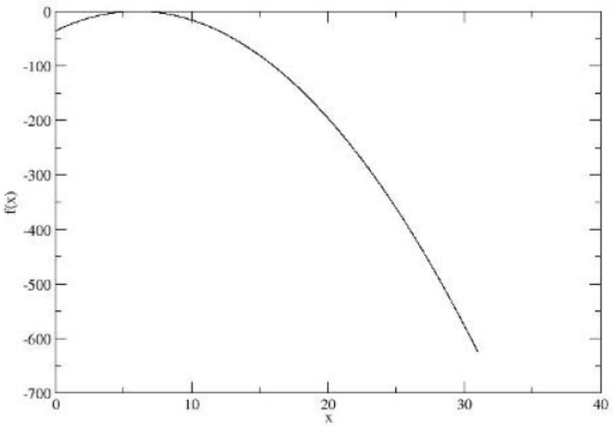
Schema	1	*	*	0	*	1

describes the set of all words of length 6 with **1's** at the **first** and **sixth positions** a **0** at the **fourth position**.

Representation Schemes

- **Example-1**

Optimization of a simple quadratic formula:



$$O(x) = -(x-6)^2 \qquad \text{for } 0 \le x \le 31$$

Using Simple Genetic Algorithm (SGA)

Encoding:

SGA features a **simple binary encoding**. In this example problem, the gene consists of a single integral number. Since there is only one parameter to be found, each chromosome consists only of a single gene.

The rather artificial constraint on the function given above allows us to use a **5-bit binary encoding** for our genes. Hence chromosomes are represented by bit strings of length 5. [2]

Representation Schemes

- **Example-1 (cont'd)**

Initial Population: $O(x) = -(x-6)^2$ for $0 \le x \le 31$

x	Bit string	*O(x)*
2	00010	-16
10	01010	-16
0	00000	-36
20	10100	-196
31	11111	-625

Note: In MATLAB, a uniform random number generator can be used to generate initial solutions.

```
Populations=31*rand(5,1)
```

Where, population size=5 & number of variable=1

In case you have multiple variables, `PopInitRange` in `GAOPTIMSET` can be used to set the initial range of each variables:

```
opts=gaoptimset(opts,'PopInit
Range',[-1 0;1 2]);
```

Population Size=5

[2]

Representation Schemes

- **Example-1 (cont'd): Population Size**
  - ◇ **Very big population size** usually **does not improve** performance of GA (in the sense of speed of finding solution).
  - ◇ **Good population size** is about **20-30**, however sometimes sizes **50-100** are reported as the best for problems with **large number of variables**.
  - ◇ Some research also shows, that the **best population size** depends on the **size of encoded string** (chromosomes).
  - ◇ It means that if you have chromosomes with **32 bits**, the population should be **higher than** for chromosomes with **16 bits**.

Representation Schemes

- **Example-1 (cont'd)**

Fitness Function: $O(x) = -(x-6)^2 \quad \text{for } 0 \le x \le 31$

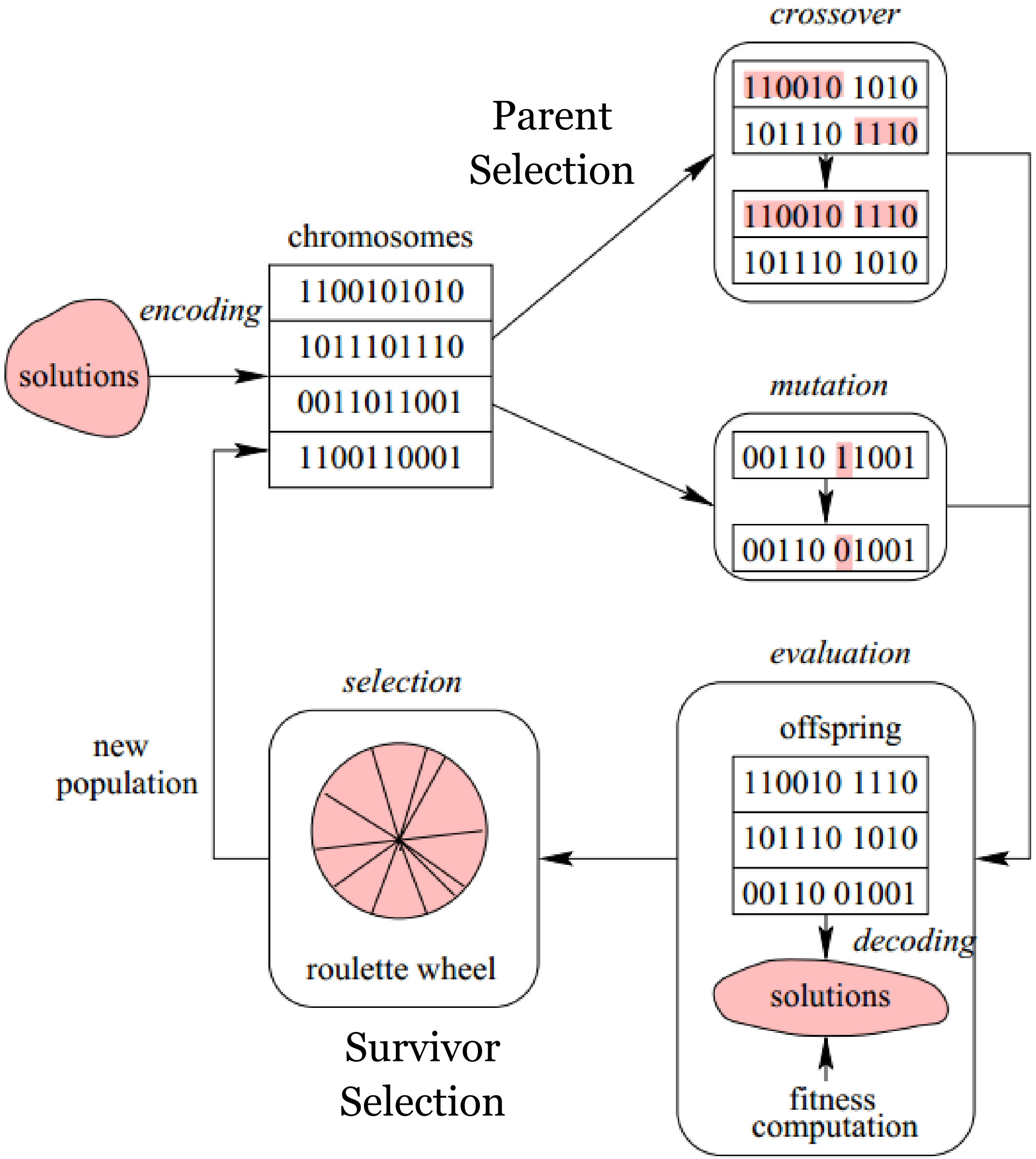
$$O(x) \Rightarrow f(x) = V - \frac{O(i) \times N}{\sum_{i=1}^{N} O(i)} \Rightarrow f(x) = \frac{-625 \times 5}{-889} - \frac{O(i) \times 5}{-889}$$

x	**Bit string**	***O(x)***	***f(x)***
2	00010	-16	3.4
10	01010	-16	3.4
0	00000	-36	3.3
20	10100	-196	2.4
31	11111	-625	0

Outline

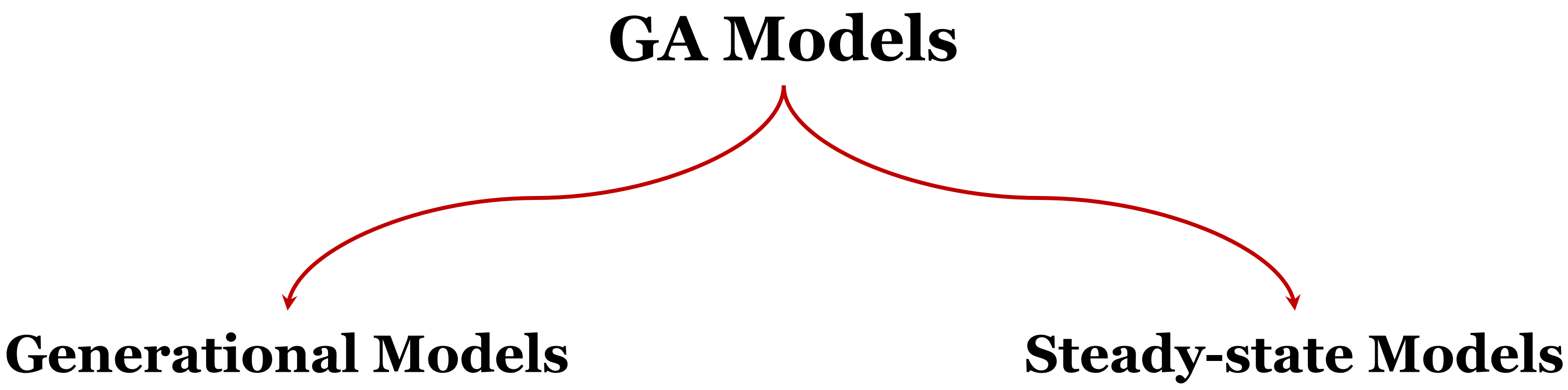
- Genetic Algorithm
- Fitness Function
- Representation Schemes
- **Selection Operators**
- Reproduction Operators
- Survivor Selection
- Real-valued GA
- Permutations GA

Selection Operators



Ref: Industrial and Systems Engineering, North Carolina State University

Selection Operators



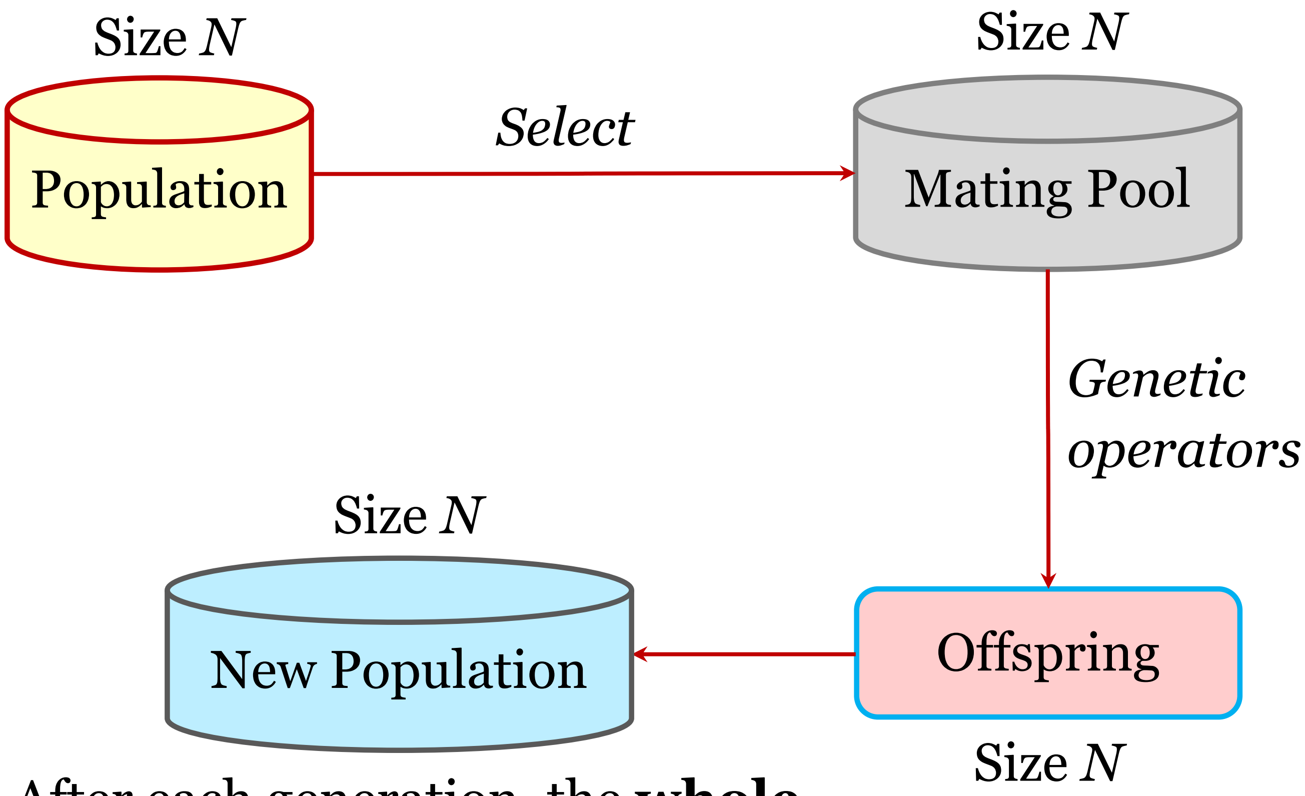
GA Models

Generational Models

Steady-state Models

Selection Operators

- **Generational GA Models**

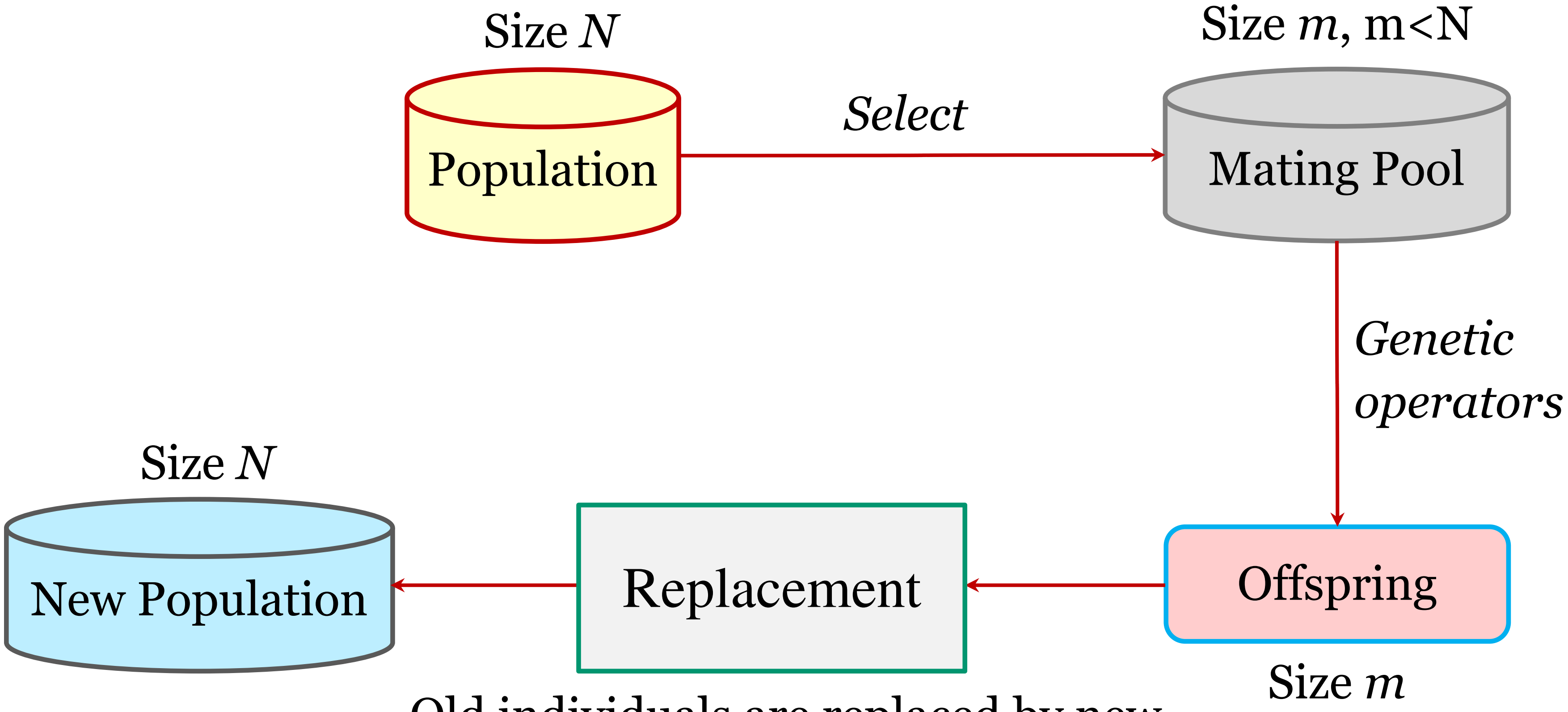


After each generation, the **whole population is replaced** by its offspring, which is called the "next generation"

[4]

Selection Operators

- **Steady-state GA Models**



Old individuals are replaced by new ones (*m* offspring)

m/N is called **generational gap G**

[4]

Selection Operators

- **Selection of the new population**
  - ◇ A **new population of candidate solutions** is selected at the **end of each generation** to serve as the population of the next generation.
  - ◇ The new population can be selected from only the offspring (**generational models**), or from both the parents and the offspring (**steady-state models**).
  - ◇ The selection operator should ensure that **good individuals** do survive to next generations.

[3]

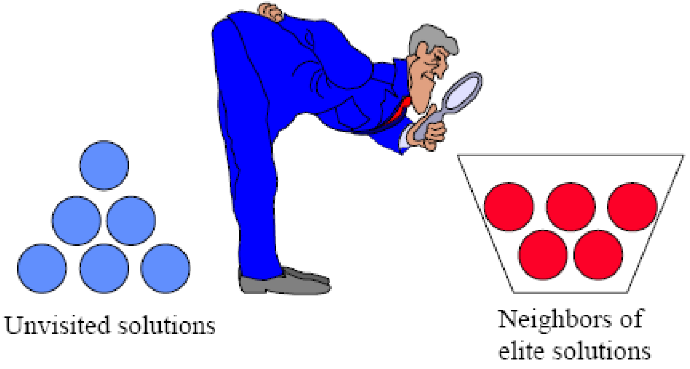
Selection Operators

- **Selective Pressure**
  - ◇ **Selection operators** are characterized by their **selective pressure** , also referred to as the **takeover time**, which relates to the time it requires to produce a uniform population.
  - ◇ **Selective pressure** indicates the **probability of the best individual being selected** compared to the average probability of selection of all individuals
  - ◇ It represents the **speed at which the best solution will occupy the entire population** by repeated application of the selection operator alone.

[3]

Selection Operators

- **Selective Pressure**
  - ◇ An operator with a **high selective pressure decreases diversity** in the population more rapidly than operators with a low selective pressure, which may lead to premature convergence to **suboptimal solutions**.
  - ◇ A **high selective pressure** limits **the exploration abilities** of the population.



Low selective pressure **High selective pressure**

[3]

Selection Operators

- **Survivor Selection Methods**
  - ◊ Random Selection
  - ◊ Fitness-Proportionate Selection (FPS)
  - ◊ Stochastic Universal Sampling
  - ◊ Rank-Based Selection
  - ◊ Elitism
  - ◊ Tournament Selection
  - ◊ Others: Boltzmann Selection, (μ, λ)- and (μ+λ)-selection and Hall of Fame

[3]

Selection Operators

- **Selection Methods: Random Selection**
  - ◊ Random selection is the **simplest selection operator**, where each individual has the **same probability of 1/N** (where N is the population size) to be selected.
  - ◊ **No fitness information is used**, which means that the best and the worst individuals have exactly the same probability of surviving to the next generation.
  - ◊ Random selection has the **lowest selective pressure** among the selection operators.

Selection Operators

- **Selection Methods: Fitness-Proportionate Selection**
  - ◊ Fitness-Proportionate Selection biases selection towards the **most-fit individuals**.
  - ◊ A probability distribution proportional to the fitness is created, and individuals are selected by sampling the distribution. Individual fitness assignment relative to the whole population can be calculated as follows:

$$F(x_i) = \frac{f(x_i)}{\sum_{l=1}^{N} f(x_l)}$$

where *f* is the solution represented by an individual chromosome.

[4]

Selection Operators

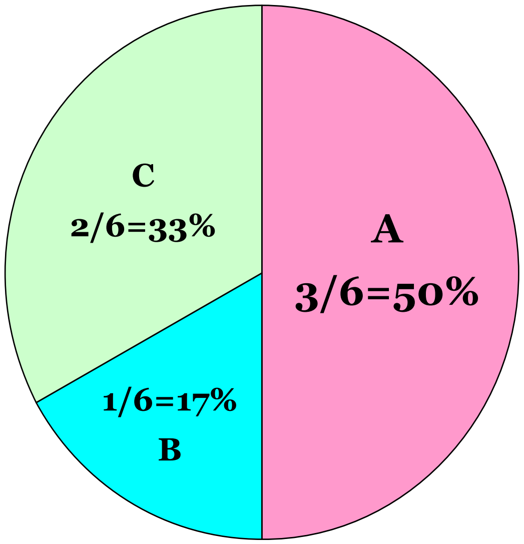
- **Selection Methods: Fitness-Proportionate Selection**
  - ◇ **Roulette wheel selection** is an example of Fitness-Proportionate Selection operator.
  - ◇ The size of each slice in the wheel is proportional to the **normalized selection probability** of each individual.

Fitness(A)=3

$f(x_i)$ Fitness(B)=1

Fitness(C)=2

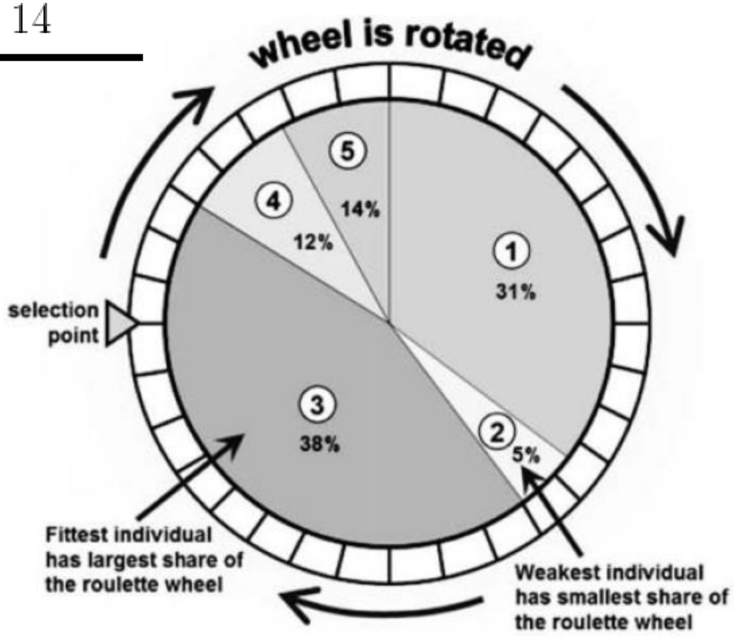
$$\sum_{l=1}^{N} f(x_l) = 6$$



Selection Operators

- **Selection Methods: Fitness-Proportionate Selection**
  - ◇ **Example:** $f(x) = x^2 + 2x + 5$

No. of Population	Chromosome	Base 10 Values	X	Fitness) (X)	Percentage
1	0001101011	107	1.05	6.82	31
2	1111011000	984	9.62	1.11	5
3	0100000101	261	2.55	8.48	38
4	1110100000	928	9.07	2057	12
5	1110001011	907	8087	3.08	14


wheel is rotated
5
14%
4
12%
1
31%
selection point
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

[6]

Selection Operators

- **Selection Methods: Fitness-Proportionate Selection**

◇ **For Example-1:**

▪ The roulette wheel can be constructed as follows.

1. Calculate the total fitness for the population.

$$F = \sum_{k=1}^{N} f(x_k)$$

x	Bit string	*f*(x)
2	00010	3.4
10	01010	3.4
0	00000	3.3
20	10100	2.4
31	11111	0

$F = 12.5$

Selection Operators

- **Selection Methods: Fitness-Proportionate Selection**
  - ◇ **For Example-1:**

    2. Calculate **selection probability *p_k*** for each chromosome x_k.

$$p_k = \frac{f(x_k)}{F}, \qquad k = 1,2,\ldots,N$$

x	Bit string	*f*(x)	p_k
2	00010	3.4	0.272
10	01010	3.4	0.272
0	00000	3.3	0.264
20	10100	2.4	0.192
31	11111	0	0

Selection Operators

- **Selection Methods: Fitness-Proportionate Selection**

◇ **For Example-1:**

3. Calculate **cumulative probability q_k** for each chromosome x_k.

$$q_k = \sum_{j=1}^{k} p_j, \qquad k = 1,2,\ldots,N$$

x	Bit string	$f(x)$	p_k	q_k
2	00010	3.4	0.272	0.272
10	01010	3.4	0.272	0.544
0	00000	3.3	0.264	0.808
20	10100	2.4	0.192	1
31	11111	0	0	1

Selection Operators

- **Selection Methods: Fitness-Proportionate Selection**
  - ◇ **For Example-1:**

4. Generate a random number *r* from the range [0,1].

5. If $q_1 \geq r$, then select the first chromosome x_1; else, select the kth chromosome x_k ($2 \leq k \leq N$) such that $q_{k-1} < r \leq q_k$.

x	**Bit string**	***f*(x)**	q_k
2	00010	3.4	0.272
10	01010	3.4	0.544
0	00000	3.3	0.808
20	10100	2.4	1
31	11111	0	1

Example: if r=0.125 → q_1, if r=0.43 → q_2 , if r=0.75 → q_3, etc.

Selection Operators

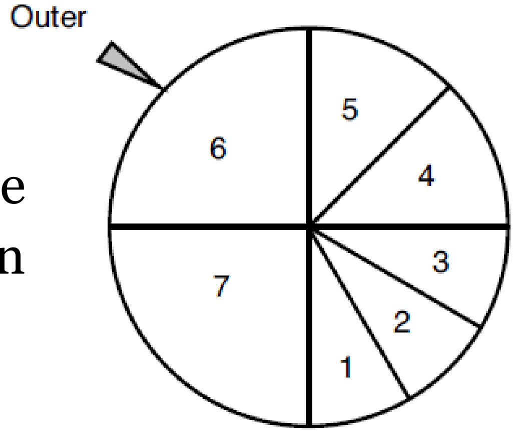
- **Selection Methods: Fitness-Proportionate Selection**
  - ◇ Because **selection is directly proportional to fitness**, it is possible that **strong individuals may dominate** in producing offspring, thereby limiting the diversity of the new population.
  - ◇ In other words, proportional selection has a **high selective pressure**.

Selection Operators

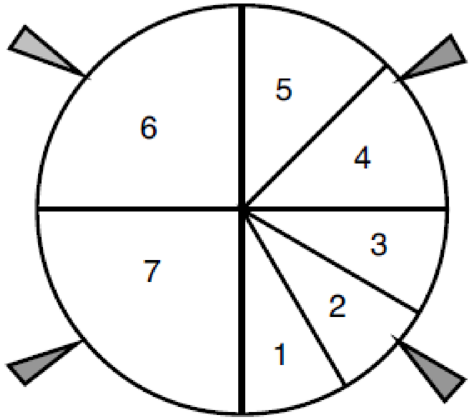
- **Selection Methods: Stochastic Universal Sampling**
  - ◇ To reduce the bias of the roulette selection strategy, the stochastic universal sampling may be used. An outer roulette wheel is placed around the pie with *N* equally spaced pointers. In the SUS strategy, a single spin of the roulette wheel will simultaneously select all the *N* individuals for reproduction.

Individuals	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Fitness	1	1	1	1.5	1.5	3	3

Roulette selection



Stochastic Universal Sampling



[5]

Selection Operators

- **Selection Methods: Rank-based selection**
  - ◊ Another way to remove problems of Fitness-Proportionate Selection (FPS) is to bias the selection probabilities on **relative rather than absolute fitness**.
  - ◊ Selection is therefore independent of actual fitness values, with the advantage that the **best individual will not dominate** the selection process.

Selection Operators

- **Selection Methods: Rank-based selection**
  - ◇ **Linear ranking** is an example for rank-based selection:

$$P(i) = \frac{2 - SP}{N} + \frac{r(i)(SP - 1)}{N(N-1)}$$

where

N is the size of the population;

SP is the selection pressure (1.0 < *SP* ≤ 2.0), and

r(*i*) is the rank associated with the individual *i*.

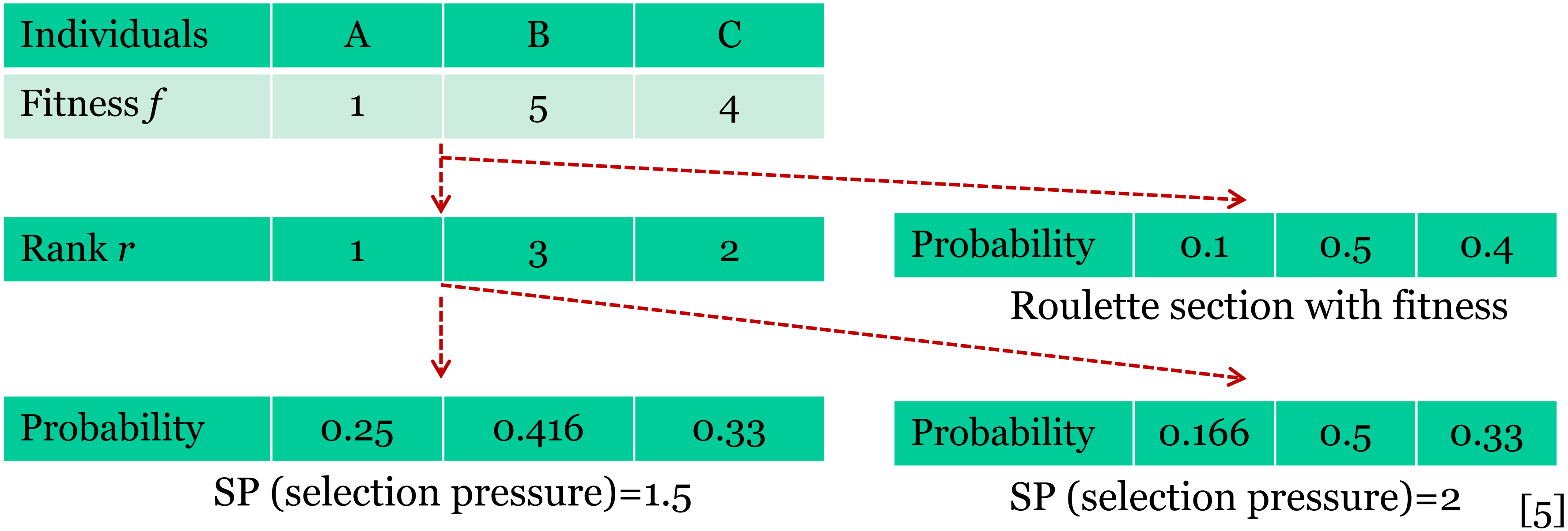
[5]

Selection Operators

- **Selection Methods: Rank-based selection**
  - ◇ **Linear ranking** is an example for rank-based selection:

$$P(i) = \frac{2-SP}{N} + \frac{r(i)(SP-1)}{N(N-1)}$$

Greater is the **selection pressure** *SP*, **more importance** to **better individuals** is given.



Individuals	A	B	C
Fitness *f*	1	5	4

Rank *r*	1	3	2

Probability	0.1	0.5	0.4

Roulette section with fitness

Probability	0.25	0.416	0.33

SP (selection pressure)=1.5

Probability	0.166	0.5	0.33

SP (selection pressure)=2

[5]

Selection Operators

- **Selection Methods: Rank-based selection**
  - ◊ **Another example of Linear ranking** in the text books

$$P(i) = 2 - SP + (SP - 1).\frac{(r(i) - 1)}{(N - 1)}$$

Some references use this form

$$P(i) = 2 - SP + 2.(SP - 1).\frac{(r(i) - 1)}{(N - 1)}$$

[6]

Selection Operators

- **Selection Methods: Rank-based selection**
  - ◇ **For Example-1:**

$$P(i) = 2 - SP + (SP - 1).\frac{(r(i) - 1)}{(N - 1)} = 0.5 + 0.5.\frac{(r(i) - 1)}{4}$$

				Rank-based Selection	Fitness-Proportionate Selection
x	**Bit string**	***f*(x)**	***r(i)***	***P(i)***	***Cumulative probability***
2	00010	3.4	1	0.5	0.272
10	01010	3.4	2	0.625	0.544
0	00000	3.3	3	0.75	0.808
20	10100	2.4	4	0.875	1
31	11111	0	5	1	1

Selection Operators

- **Selection Methods: Rank-based selection**
  - ◇ **Non-Linear ranking** permits higher selective pressures than the linear ranking method.

$$P(i) = \frac{N.X^{i-1}}{\sum_{i=1}^{N} X^{i-1}}$$

X is computed as the root of the polynomial:

$$(SP-N).X^{N-1} + SP.X^{N-2} + \ldots + SP.X + SP = 0$$

Non-linear ranking allows values of selective pressure in the interval [1, N - 2].

[6]

Selection Operators

- **Selection Methods: Elitism**
  - ◊ Elitism refers to the process of ensuring that the **best individuals** of the current population survive to the next generation.
  - ◊ The **best individuals** are copied to the new population without being mutated.
  - ◊ The **more individuals that survive** to the next generation, the **less the diversity** of the new population.

Selection Operators

- **Selection Methods: Elitism**
  - ◇ **For Example-1:**

x	Bit string	*f*(x)	Ranking
2	00010	3.4	1
10	01010	3.4	2
0	00000	3.3	3
20	10100	2.4	4
31	11111	0	5

*

Note that chromosomes x = 2 and x = 10 have equal fitness values, hence their relative ranking is an arbitrary choice.

Selection Operators

- **Selection Methods: Tournament Selection**

Tournament selection consists in **randomly selecting** k individuals; the parameter k is called the size of the tournament group.

A **tournament** is then applied to the k members of the group to select the best one.

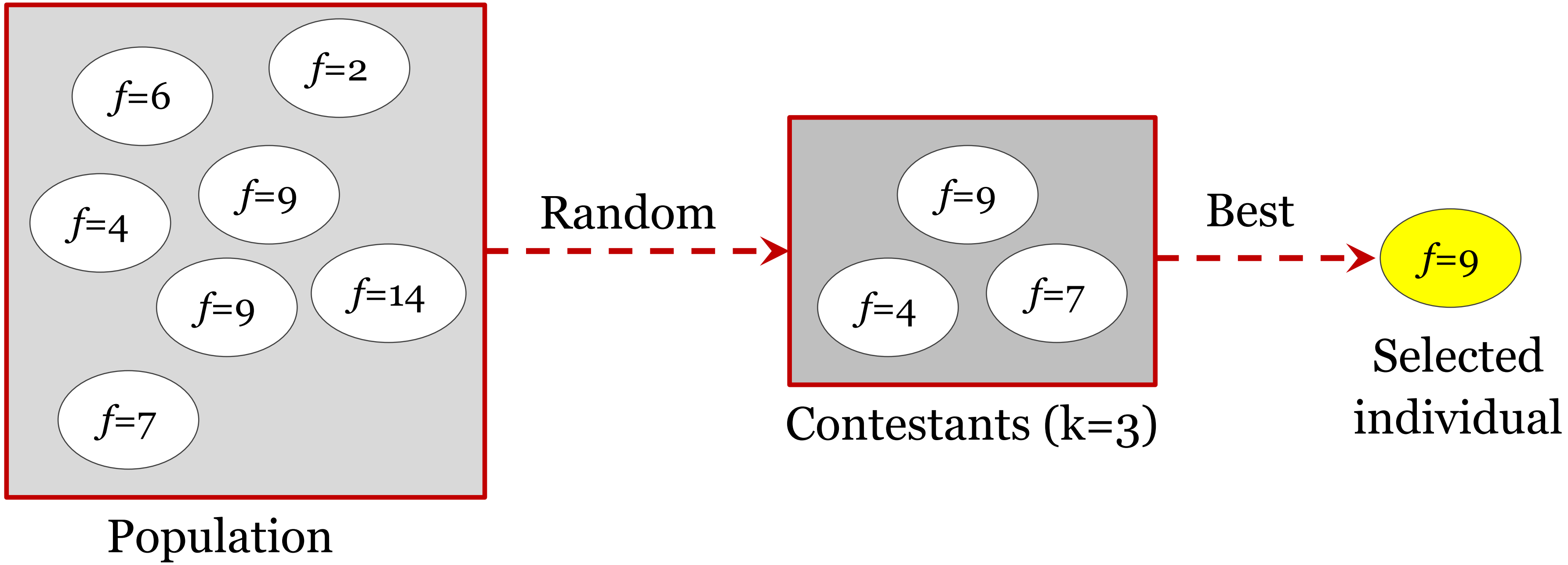
To select N individuals, the **tournament procedure** is then carried out N times.

[5]

Selection Operators

- **Selection Methods: Tournament Selection**

  Example: a tournament of size 3 is performed. Three solutions are picked randomly from the population. The best solution from the picked individuals is then selected.



[5]

Outline

- Genetic Algorithm
- Fitness Function
- Representation Schemes
- Selection Operators
- **Reproduction Operators**
- Survivor Selection
- Real-valued GA
- Permutations GA

Reproduction Operators

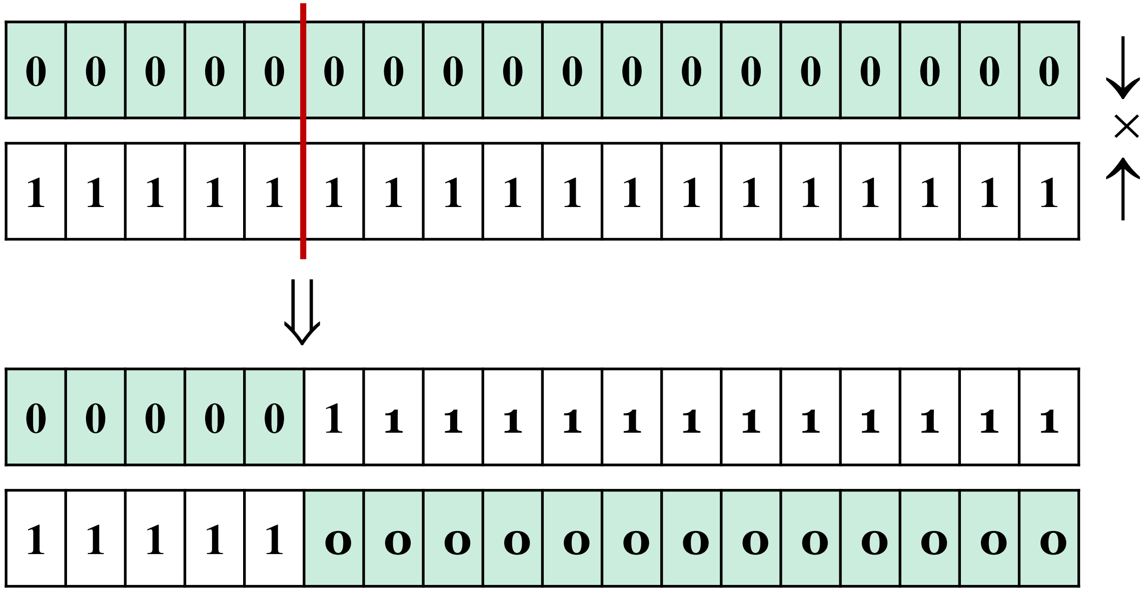
- **Crossover**
  - ◊ Crossover mimics biological recombination
    - ▪ Some portion of genetic material is **swapped** between chromosomes.
    - ▪ Typically the swapping produces an **offspring**.
  - ◊ Two parents produce two offspring.
  - ◊ In terms of crossover, **"superior" individuals** should have **more opportunities to reproduce** to ensure that offspring contain genetic material of the best individuals.
  - ◊ Crossover leads to effective combination of schemata (sub-solutions on different chromosomes)

Reproduction Operators

- **Crossover**
  - ◊ 1-point Crossover
  - ◊ n-point Crossover
  - ◊ Uniform Crossover

Reproduction Operators

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



0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0

1 1 1 1 1 | 1 1 1 1 1 1 1 1 1 1 1 1 1

↓ × ↑

⇓

0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0

Parents

Children or offspring

Reproduction Operators

- **1-point Crossover**
  - ◇ **MCQ:** Given a binary string **1101001100101101** and another binary string ***yxyyxyxxyyyxyxxy*** in which the values 0 and 1 are denoted by *x* and *y*.

    The offspring that results from applying **1-point crossover** on two strings at a randomly selected recombination point is

    a) *yxxyyyxyxxy*11010 and *yxyyx*01100101101

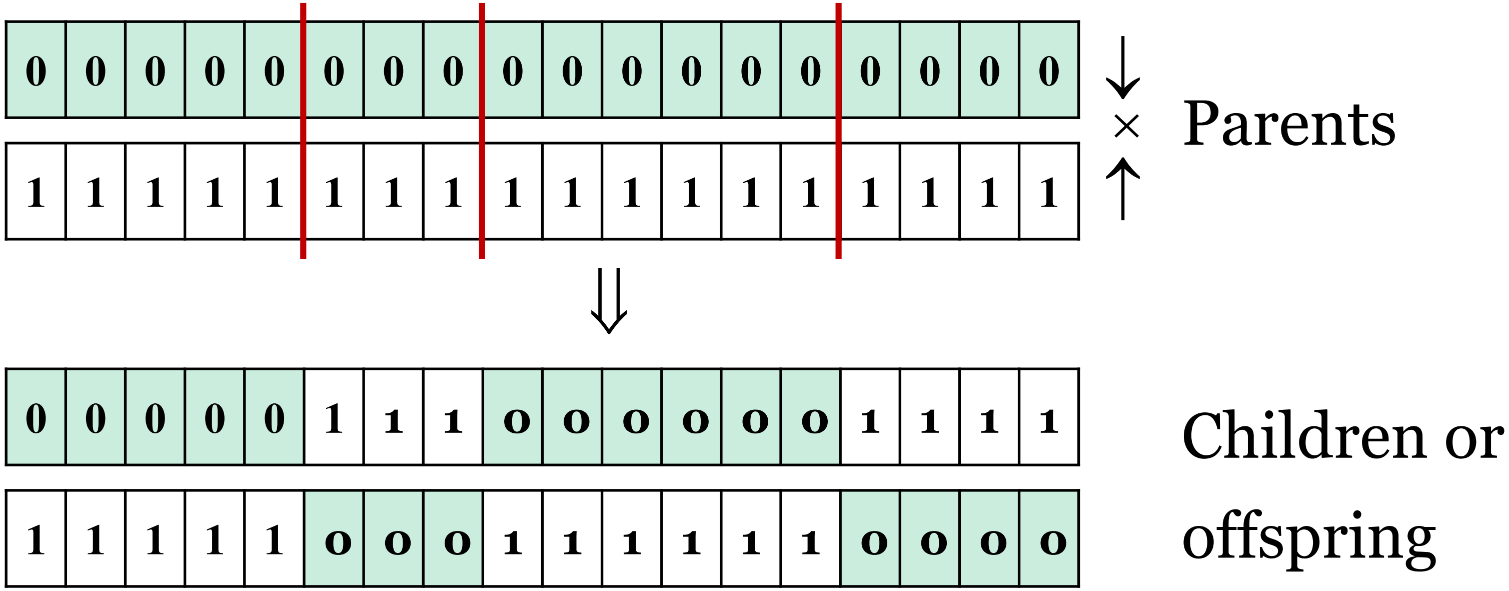
    [✓ b) 11010*yxxyyyxyxxy* and *yxyyx*01100101101]

    c) 11010*yxxyyyxyxxy* and 01100101101*yxyyx*

    d) None of the above

Reproduction Operators

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



0 0 0 0 0 | 0 0 0 | 0 0 0 0 0 0 | 0 0 0 0

1 1 1 1 1 | 1 1 1 | 1 1 1 1 1 1 | 1 1 1 1

0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0

Reproduction Operators

- **Uniform Crossover**
  - ◇ Corresponding bit positions are randomly exchanged between the two parents to produce two offspring.
  - ◇ 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.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	↓ ×	Parents
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	↑	
0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1		1st Child
1	0	1	1	0	1	0	0	1	1	1	0	1	0	0	1	1	0		2nd Child

Reproduction Operators

- **Crossover: For Example-1**

x	**Bit string**	***f*(x)**	**Ranking**
2	00010	3.4	1
10	01010	3.4	2
0	00000	3.3	3
20	10100	2.4	4
31	11111	0	5

Parents subject to crossover

◇ We randomly pair the **top four ranked** chromosomes.

◇ Choose a **crossover probability** p_c of 0.5, easily modeled by a coin toss. p_c is typically in range (0.6, 0.9).

◇ If p_c >r (random number from the range [0,1]) perform undergo crossover.

Reproduction Operators

- **Crossover: For Example-1**

x	Bit string	*f*(x)	Ranking
2	00010	3.4	1
10	01010	3.4	2
0	00000	3.3	3
20	10100	2.4	4
31	11111	0	5

Parents subject to crossover

◇ The random pairings selected are ranked **chromosomes (1,4) and (2,3)**.

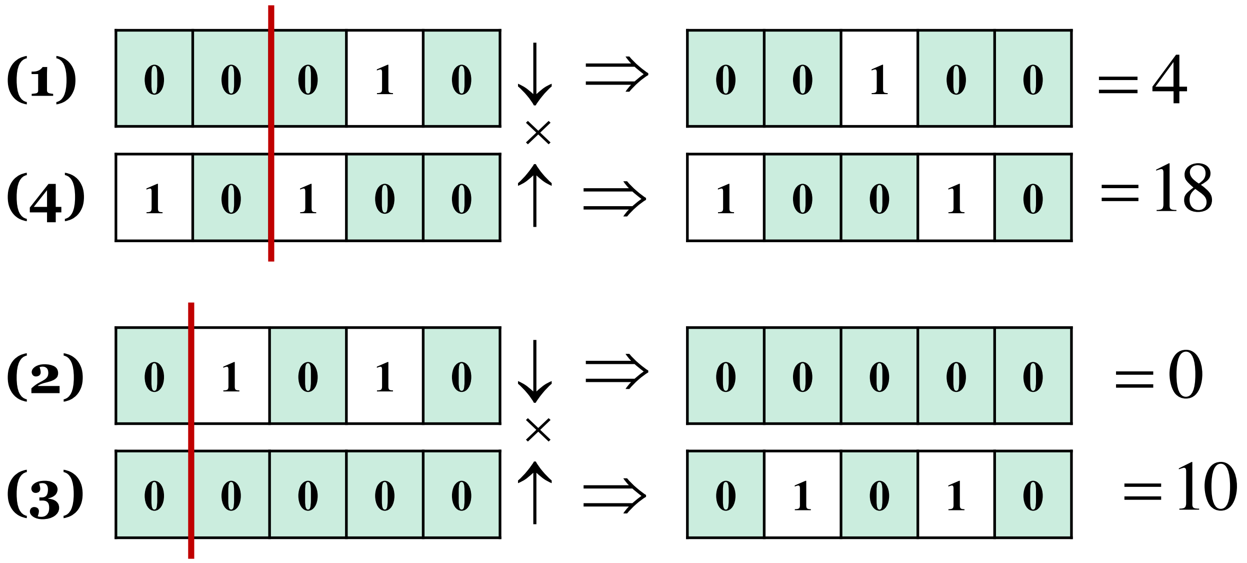
◇ Each pair of chromosomes will undergo a **single random point crossover** to produce two new chromosomes.

Reproduction Operators

- **Crossover: For Example-1**

x	Bit string	*f*(x)	Ranking
2	00010	3.4	1
10	01010	3.4	2
0	00000	3.3	3
20	10100	2.4	4
31	11111	0	5

Parents subject to crossover (1,4) & (2,3)



Note that in the case of the second crossover, because the first bit is identical in both strings the resulting chromosomes are the same as the parents. This is effectively equivalent to no crossover operation occurring.

Reproduction Operators

- **Crossover: For Example-1**
  - **New Population**

x	Bit string	*O(x)*	*f*(x)	Ranking
4	00100	-4	3.24	1
2	00010	-16	2.96	2
10	01010	-16	2.96	3
0	00000	-36	2.5	4
18	10010	-144	0	5

◇ We can see the population has **converged somewhat** toward the optimal answer (**x=6**).

◇ We now repeat the evaluation process with our new population.

Reproduction Operators

- **Mutation**
  - ◇ The aim of mutation is to **introduce new genetic material** into an existing individual; that is, **to add diversity** to the genetic characteristics of the population.
  - ◇ Mutation is used in support of crossover to ensure that the full range of allele is accessible for each gene.
  - ◇ In the case of mutation, selection mechanisms should focus on **“weak” individuals**.
  - ◇ The hope is that **mutation of weak individuals** will result in **introducing better traits** to weak individuals, thereby increasing their **chances of survival**.

Reproduction Operators

- **Mutation**
  - ◇ Alter **each gene independently** with a probability p_m
  - ◇ p_m is called the mutation rate: $\frac{1}{pop\_size} < p_m < \frac{1}{chromosome\_length}$
  - ◇ For each gene, generate a random number r:[0,1]
  - ◇ If p_m >r alter the gene.

parent	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

$$\Downarrow$$

child	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**

Reproduction Operators

- **Mutation: For Example-1**

x	Bit string	*O(x)*	*f*(x)	Ranking
4	00100	-4	3.24	1
2	00010	-16	2.96	2
10	01010	-16	2.96	3
0	00000	-36	2.5	4
18	10010	-144	0	5

Mutation

Crossover

◇ Again we preserve the best chromosome (x = 4) and remove the worst (x = 18).

◇ Our random pairings this time are ranked chromosomes (1, 2) and (3, 4). This time, only pair (3, 4) has been selected by a random process to cross over, and (1, 2) is selected for mutation.

Reproduction Operators

- **Mutation: For Example-1**

x	Bit string	*O*(*x*)	*f*(x)	Ranking	
4	00100	-4	3.24	1	Mutation
2	00010	-16	2.96	2	
10	01010	-16	2.96	3	Crossover
0	00000	-36	2.5	4	
18	10010	-144	0	5	

◇ It is worth noting that the **(1, 2)** pair had the potential to **produce the optimal solution x = 6** if it had undergone **crossover**.

◇ This missed opportunity is characteristic of the genetic algorithm's **non-deterministic nature**: the time taken to obtain an optimal solution cannot be accurately foretold.

Reproduction Operators

- **Mutation: For Example-1**

x	Bit string	*O(x)*	*f*(x)	Ranking
4	00100	-4	3.24	1
2	00010	-16	2.96	2
10	01010	-16	2.96	3
0	00000	-36	2.5	4
18	10010	-144	0	5

} Mutation

} Crossover

◇ The mutation of (2), however, **reintroduced some of the lost bit-string representation**. With no mutate operator the algorithm would no longer be capable of representing **odd values** (bit strings ending with a one).

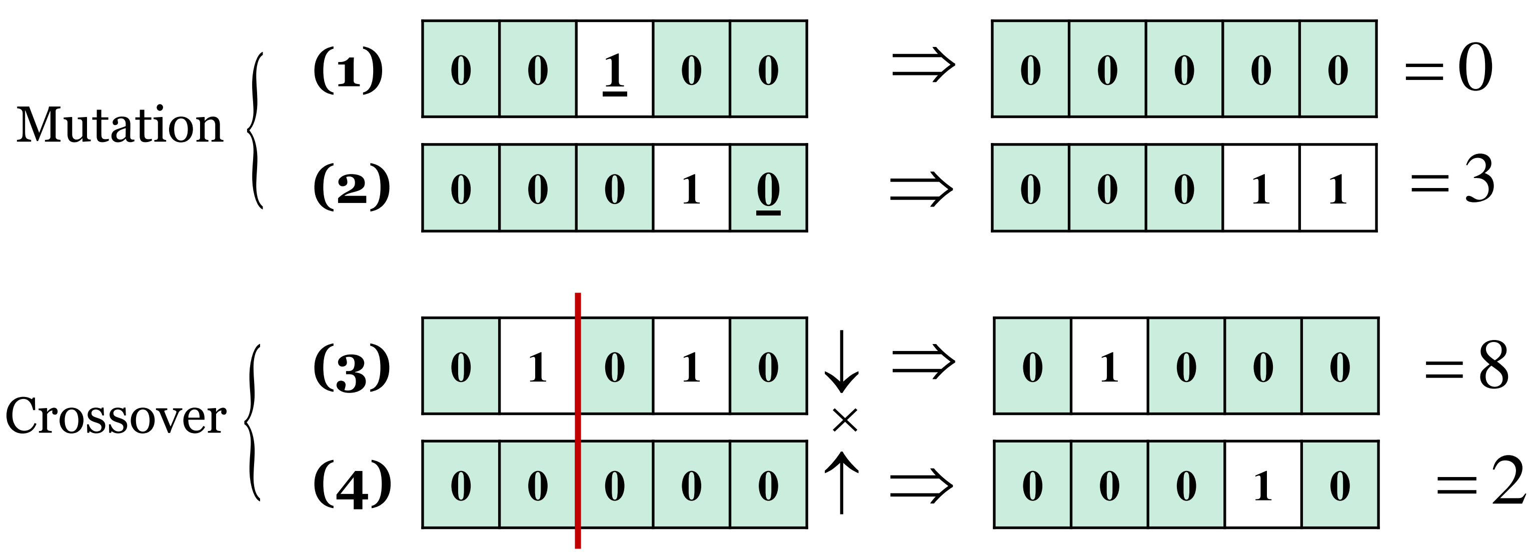
Reproduction Operators

- **Mutation: For Example-1**

x	Bit string	*O(x)*	*f*(x)	Ranking
4	00100	-4	3.24	1
2	00010	-16	2.96	2
10	01010	-16	2.96	3
0	00000	-36	2.5	4
18	10010	-144	0	5

Mutation (rankings 1–2)

Crossover (rankings 3–4)



Mutation

(1) 0 0 <u>1</u> 0 0 $\Rightarrow$ 0 0 0 0 0 $=0$

(2) 0 0 0 1 <u>0</u> $\Rightarrow$ 0 0 0 1 1 $=3$

Crossover

(3) 0 1 | 0 1 0 $\downarrow$ $\Rightarrow$ 0 1 0 0 0 $=8$

$\times$

(4) 0 0 | 0 0 0 $\uparrow$ $\Rightarrow$ 0 0 0 1 0 $=2$

Reproduction Operators

- **Mutation: For Example-1**
  - **New Population**

x	Bit string	*O(x)*	*f*(x)	Ranking
4	00100	-4	2.32	1
8	01000	-4	2.32	2
3	00011	-9	1.96	3
2	00010	-16	1.45	4
0	00000	-36	0	5

Crossover

◇ As before, chromosome x = 18 is removed and x = 4 is retained.

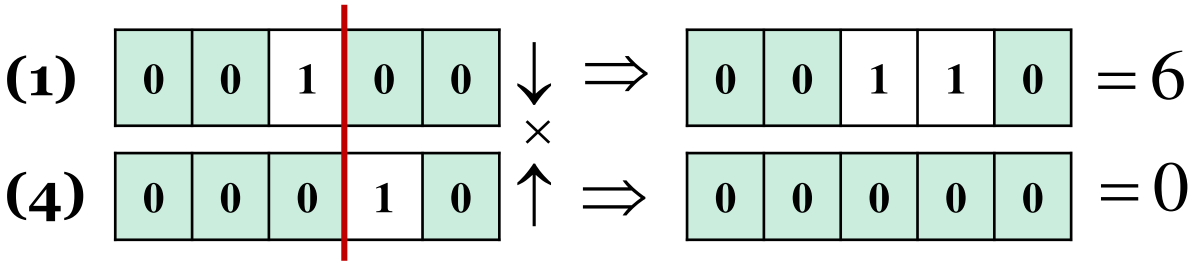
◇ The selected pairs for crossover are (1, 3) and (1, 4), of which only (1, 4) actually undergoes crossover.

Reproduction Operators

- **Mutation: For Example-1**
  - **New Population**

x	Bit string	*O(x)*	*f*(x)	Ranking
4	00100	-4	2.32	1
8	01000	-4	2.32	2
3	00011	-9	1.96	3
2	00010	-16	1.45	4
0	00000	-36	0	5

Crossover



(1) 0 0 1 | 0 0 $\downarrow$ $\Rightarrow$ 0 0 1 1 0 $= 6$

$\times$

(4) 0 0 0 | 1 0 $\uparrow$ $\Rightarrow$ 0 0 0 0 0 $= 0$

The optimal solution of **x = 6** has been obtained.

Reproduction Operators

- **Mutation: For Example-1**
  - ◊ At this point, we can stop the genetic algorithm because we know this is the optimal solution.
  - ◊ However, if we let the algorithm continue, it should eventually **completely converge to x = 6**. This is because the x = 6 chromosome is now persistent through subsequent populations due to its optimal nature.
  - ◊ When another chromosome is set to x = 6 through crossover, the chance of it being preserved through populations increases due to its increased presence in the population.

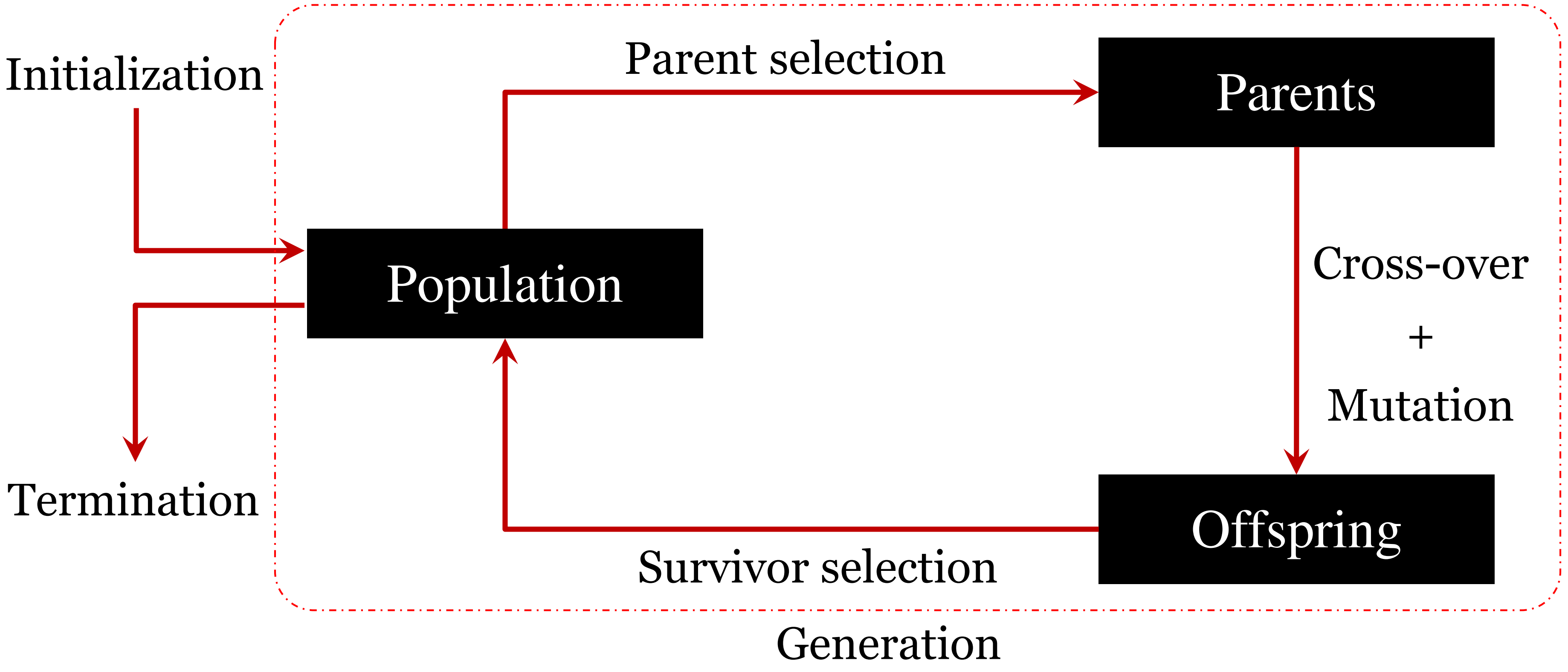
Reproduction Operators

- **Mutation: For Example-1**
  - ◇ This probability is proportional to the presence of the x = 6 chromosome in the population, and hence given enough iterations the whole population should converge.
  - ◇ The elitism operator, combined with the fact that there is only one maximum, ensures that the population will never converge to another chromosome.

Outline

- Genetic Algorithm
- Fitness Function
- Representation Schemes
- Selection Operators
- Reproduction Operators
- **Survivor Selection**
- Real-valued GA
- Permutations GA

Survivor Selection



Survivor Selection

- **Random Selection**: delete randomly.
- **Age-based Selection**: first-in-first-out (a.k.a. **delete-oldest**).
- **Fitness-based or Proportionate Selection (FPS)**:
  - ◇ Delete or replace based on inverse of fitness or
  - ◇ **Elitism**: always keep best individuals (at least one copy of the fittest solution so far)
  - ◇ **Genitor**: a.k.a. "**delete-worst**"
    - ▪ Can lead to rapid improvement and potential premature convergence.
    - ▪ use with large populations or "no duplicates" policy.

Survivor Selection

- **For Example-1**

x	Bit string	*f*(x)	Ranking
2	00010	3.4	1
10	01010	3.4	2
0	00000	3.3	3
20	10100	2.4	4
31	11111	0	5

◇ We preserve the best chromosomes (first four) and remove the worst (x = 31) ⇒ **Genitor** or "**delete-worst**".

Outline

- Genetic Algorithm
- Fitness Function
- Representation Schemes
- Selection Operators
- Reproduction Operators
- Survivor Selection
- **Real-valued GA**
- Permutations GA

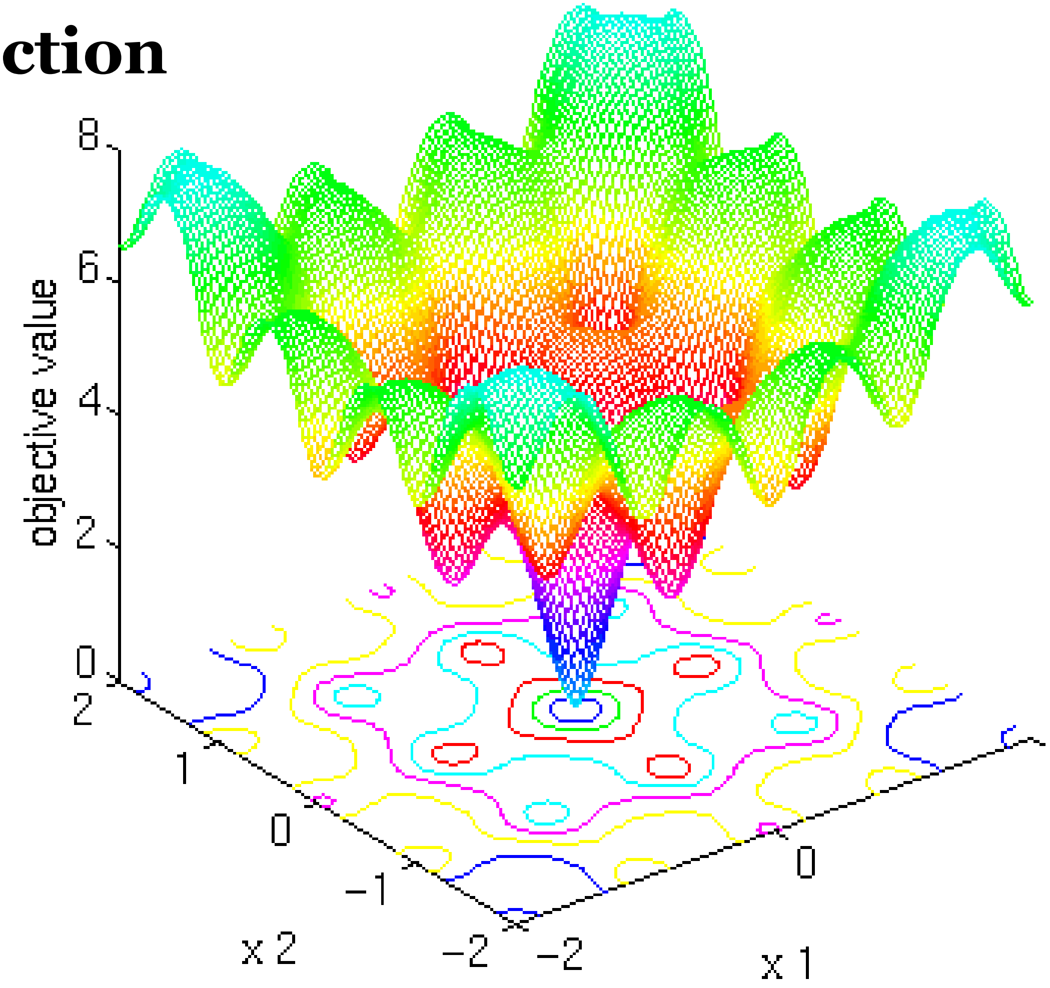
Real-valued GA

- Many problems occur as **real valued problems**, e.g. continuous parameter optimization $f : \Re^n \rightarrow \Re$

- Illustration: **Ackley's function**

$$f(\overline{x}) = -c_1 \cdot exp\left(-c_2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} cos(c_3 \cdot x_i)\right) + c_1 + 1$$

$$c_1 = 20,\ c_2 = 0.2,\ c_3 = 2\pi$$



Real-valued GA

- Many problems have real-valued parameters.
- If mapped to a binary space, **high precision solutions** would require **very long chromosomes** (slow evolution).

Example: 0≤X≤10 and Resolution=4

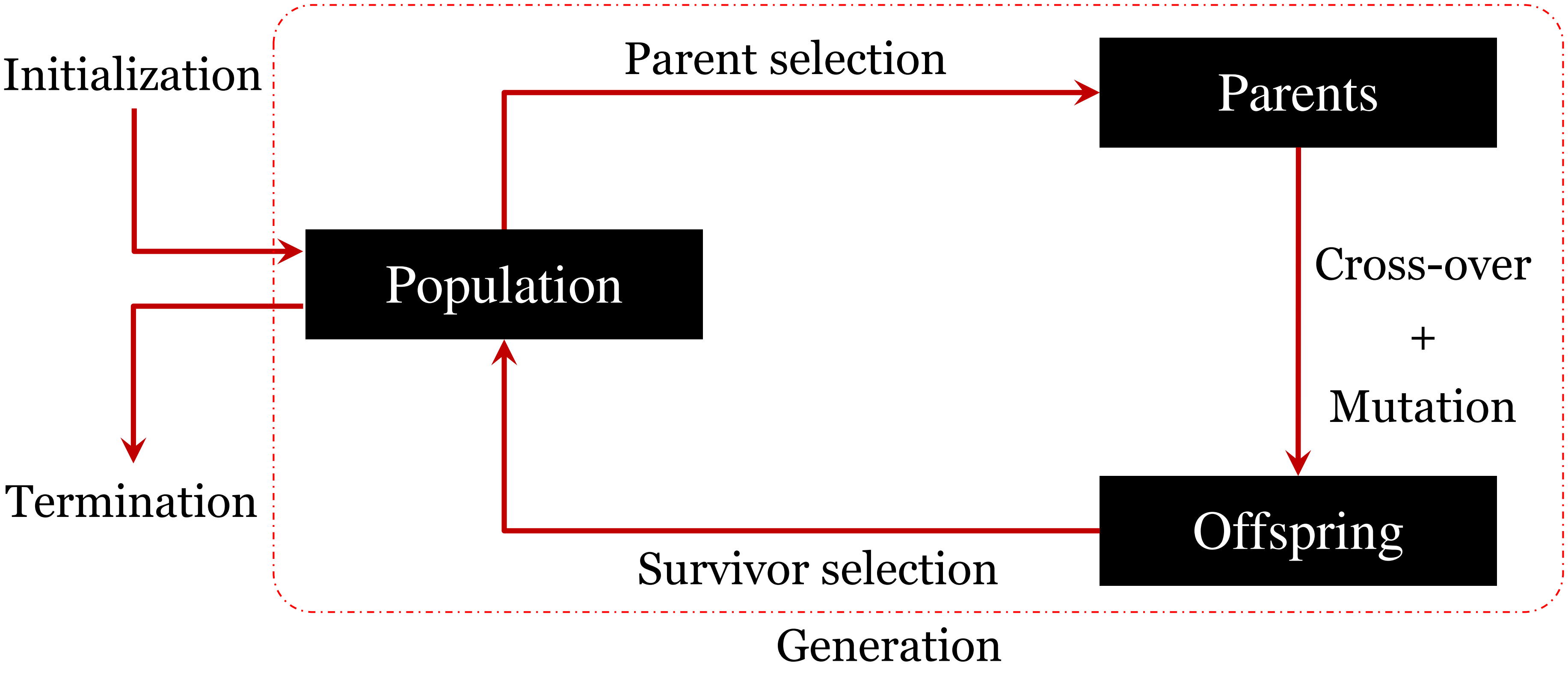
$$QI = floor\left(\log_2(\alpha)+1\right) = floor\left(\log_2(10)+1\right) = 4 \text{ bits}$$

$$QF = ceiling\left(\log_2\left(\frac{1}{0.0001}\right)\right) = 14 \text{ bits}$$

$$Q = QI + QF = 4 + 14 = 18 \text{ bits}$$

Ex. 3.1416 → 0011.00100100001111

Real-valued GA



Real-valued GA

- **Crossover Operators:**
  - ◇ **Discrete**: use any of the crossover operators identified before for binary representations.
    - ▪ 1-point Crossover
    - ▪ n-point Crossover
    - ▪ Uniform Crossover
  - ◇ **Intermediate (arithmetic)**:
    - ▪ Single Arithmetic,
    - ▪ Simple Arithmetic,
    - ▪ Whole Arithmetic.

[4]

Real-valued GA

- **Single Arithmetic Crossover:**

Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$,

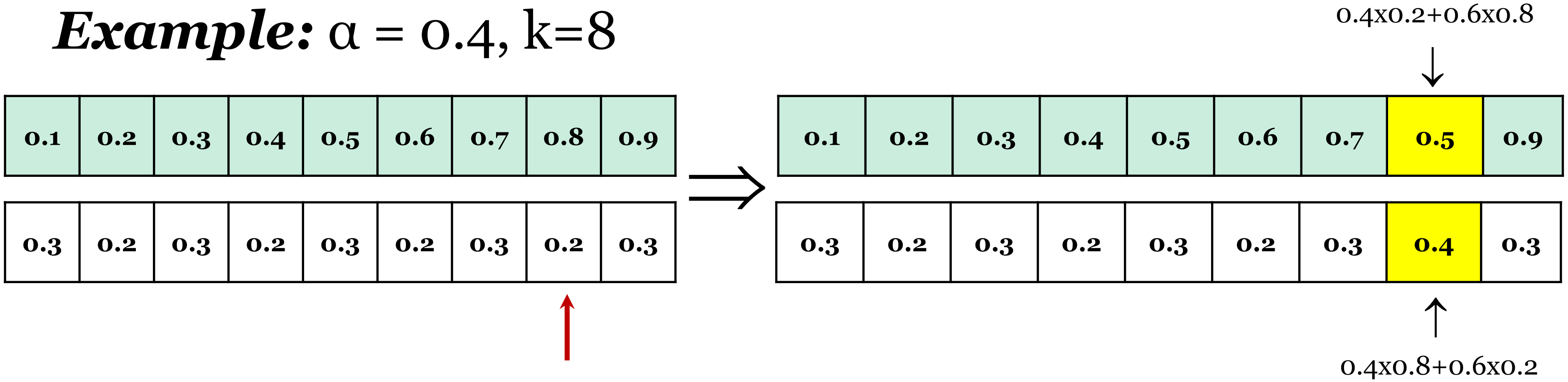
Pick random gene (k) at random,

Child 1 is:

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

Reverse for other child.

Example: α = 0.4, k=8



0.4x0.2+0.6x0.8

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	0.4	0.3

0.4x0.8+0.6x0.2

Real-valued GA

- **Simple Arithmetic Crossover :**

Parents: ⟨$x_1,\ldots,x_n$ ⟩ and ⟨$y_1,\ldots,y_n$⟩,

Pick random gene (k), after this point mix values,

Child 1 is:

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

Reverse for other child.

Example: α = 0.4, k=6

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

⇒

0.4x0.3+0.6x0.7

↓

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	0.4	0.4	0.5

↑

0.4x0.7+0.6x0.3

Real-valued GA

- **Whole Arithmetic Crossover:**

//Most commonly used

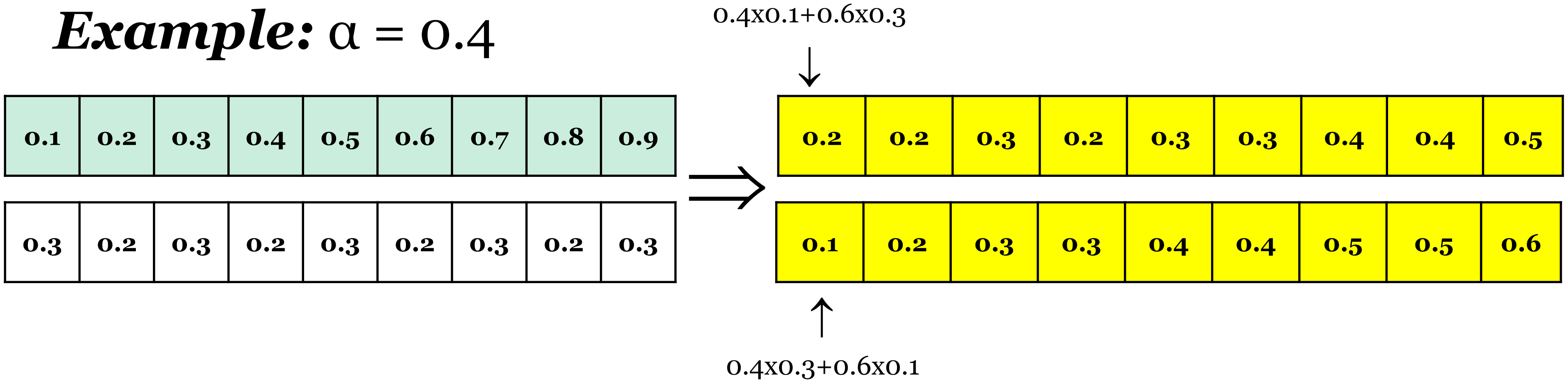
Parents: ⟨$x_1,\ldots,x_n$ ⟩ and ⟨$y_1,\ldots,y_n$⟩,

Child 1 is:

$$\alpha.\bar{x}+(1-\alpha)\bar{y}$$

Reverse for other child.

Example: α = 0.4



0.4x0.1+0.6x0.3

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

$\Rightarrow$

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

0.4x0.3+0.6x0.1

Real-valued GA

- **Mutation:**
  - ◇ **The general scheme for mutation** is:

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

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

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

  - ◇ Analogous to **bit-flipping mutation**.

[4]

Real-valued GA

- **Mutation:**
  - ◊ Another mutation scheme is to **add random noise**, for each gene:

    $$x_i' = x_i + N(0,\sigma)$$

    where N(0,σ) is a random **Gaussian number** with zero mean and standard deviation σ.

[4]

Outline

- Genetic Algorithm
- Fitness Function
- Representation Schemes
- Selection Operators
- Reproduction Operators
- Survivor Selection
- Real-valued GA
- **Permutations GA**

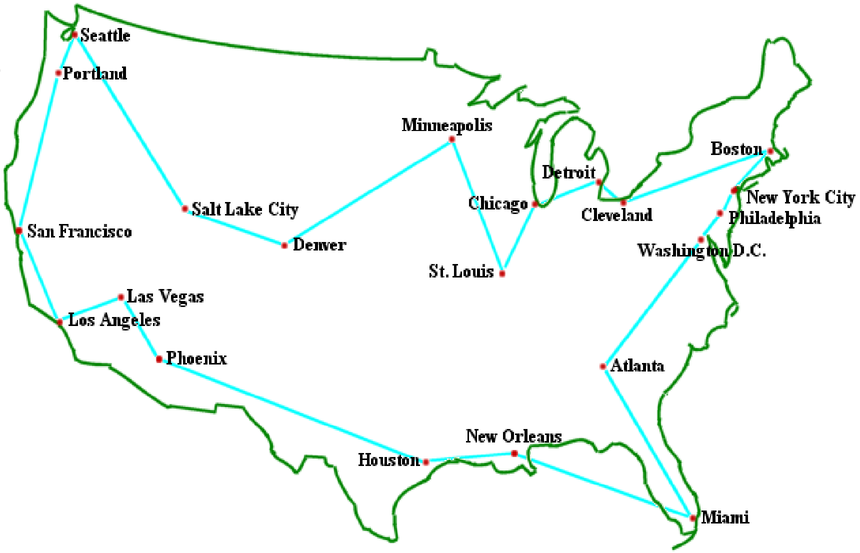
Permutations GA

- For problems that take the form of deciding on the **order** in which a **sequence of events** should occur,
- Two types of problems exist:
  - ◊ The events use **limited resources or time**. The order of events is important. e.g. **Job Shop Scheduling**,
  - ◊ The **adjacency of elements** is important. e.g. **Travelling Salesman Problem**.
- 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**.

$$[5 \quad 1 \quad 3 \quad 2 \quad 7 \quad 6 \quad 4]$$

Permutations GA

- **Travelling Salesman Problem**
  - ◇ Given **n** cities,
  - ◇ A travelling salesman must visit the n cities and return home, making a loop (roundtrip).



  - ◇ He would like to travel in the most efficient way (cheapest way or shortest distance or some other criterion).

- **Search Space**

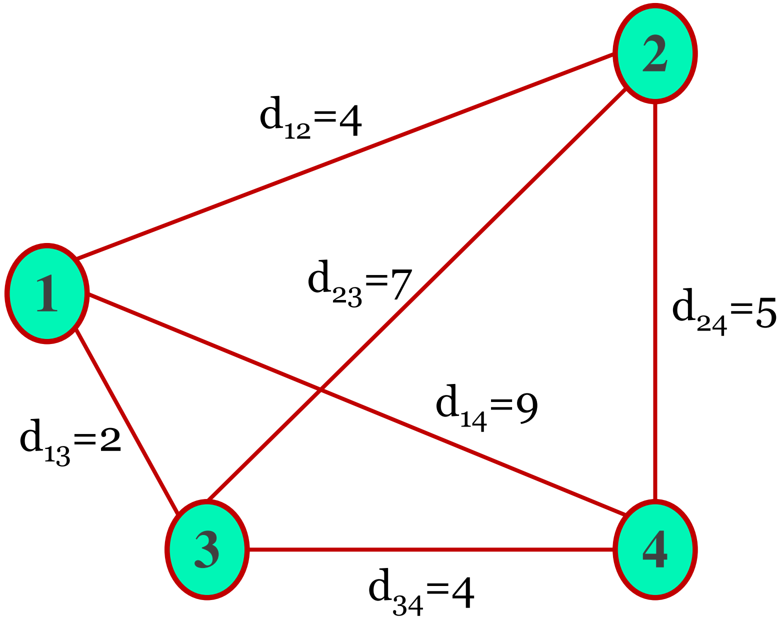
Search space is BIG:

for 21 cities there are 21! ≈ 51090942171709440000 **possible** tours. This is NP-Hard problem.

Permutations GA

- **Travelling Salesman Problem: Encoding**
  - ◇ Label the cities 1,2,…,n
  - ◇ One complete tour is one permutation

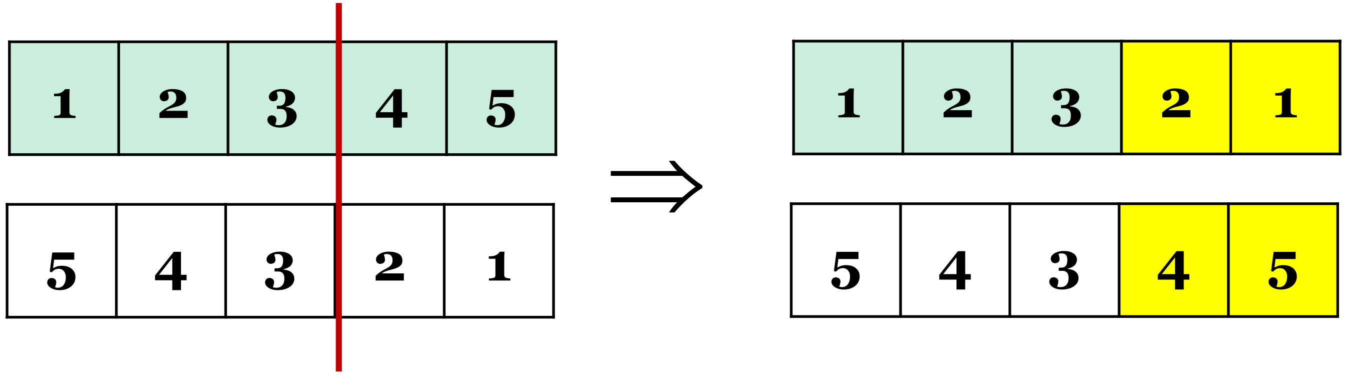
    (e.g. for n =4, [1,2,3,4], [3,4,2,1] are OK).



Permutations GA

- **Travelling Salesman Problem: Crossover**

  ◇ Previously defined crossover operators will often lead to **inadmissible solutions**.



1 2 3 | 4 5

5 4 3 | 2 1

⇒

1 2 3 2 1

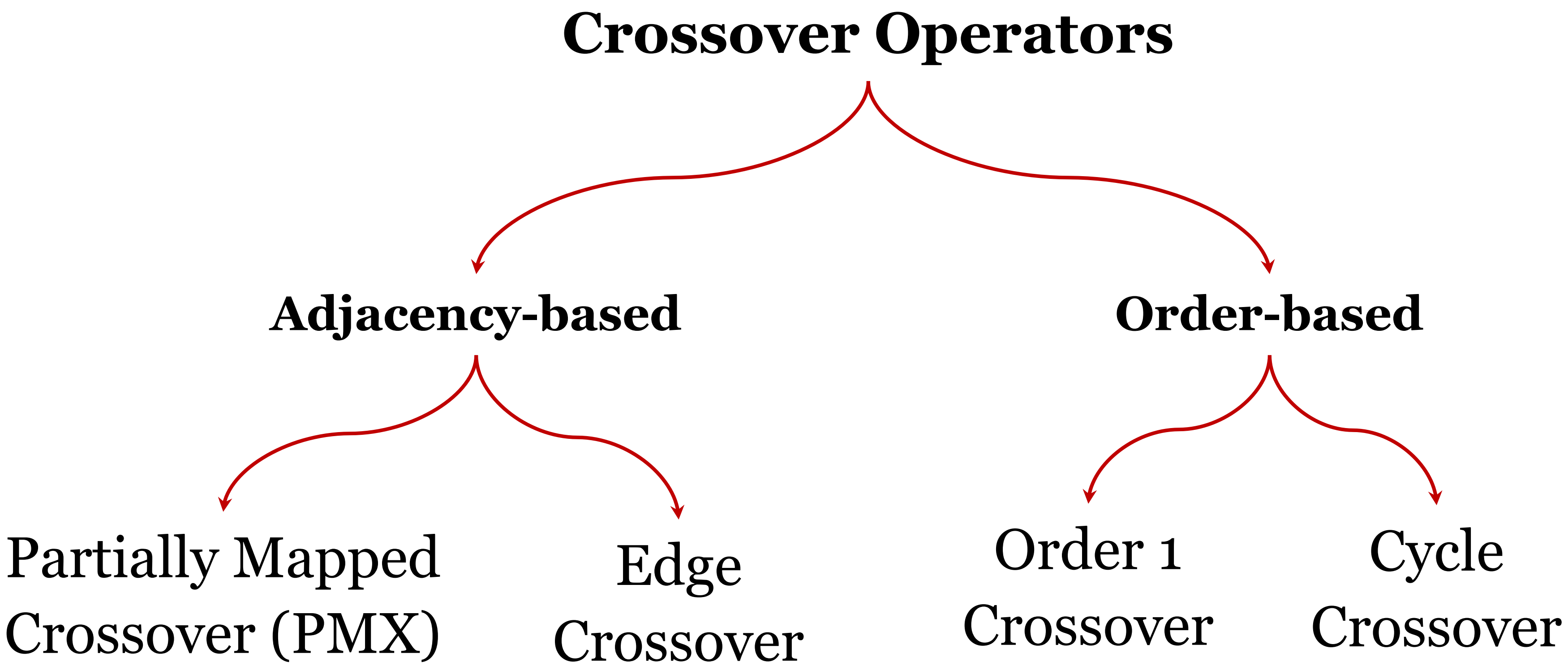
5 4 3 4 5

  ◇ Many specialized operators have been devised which focus on combining order or adjacency information from the two parents.

[4]

Permutations GA

- **Travelling Salesman Problem: Crossover**



Crossover Operators

Adjacency-based

Partially Mapped Crossover (PMX)

Edge Crossover

Order-based

Order 1 Crossover

Cycle Crossover

Permutations GA

• Partially Mapped Crossover (PMX)

Informal procedure for parents P1 and P2:

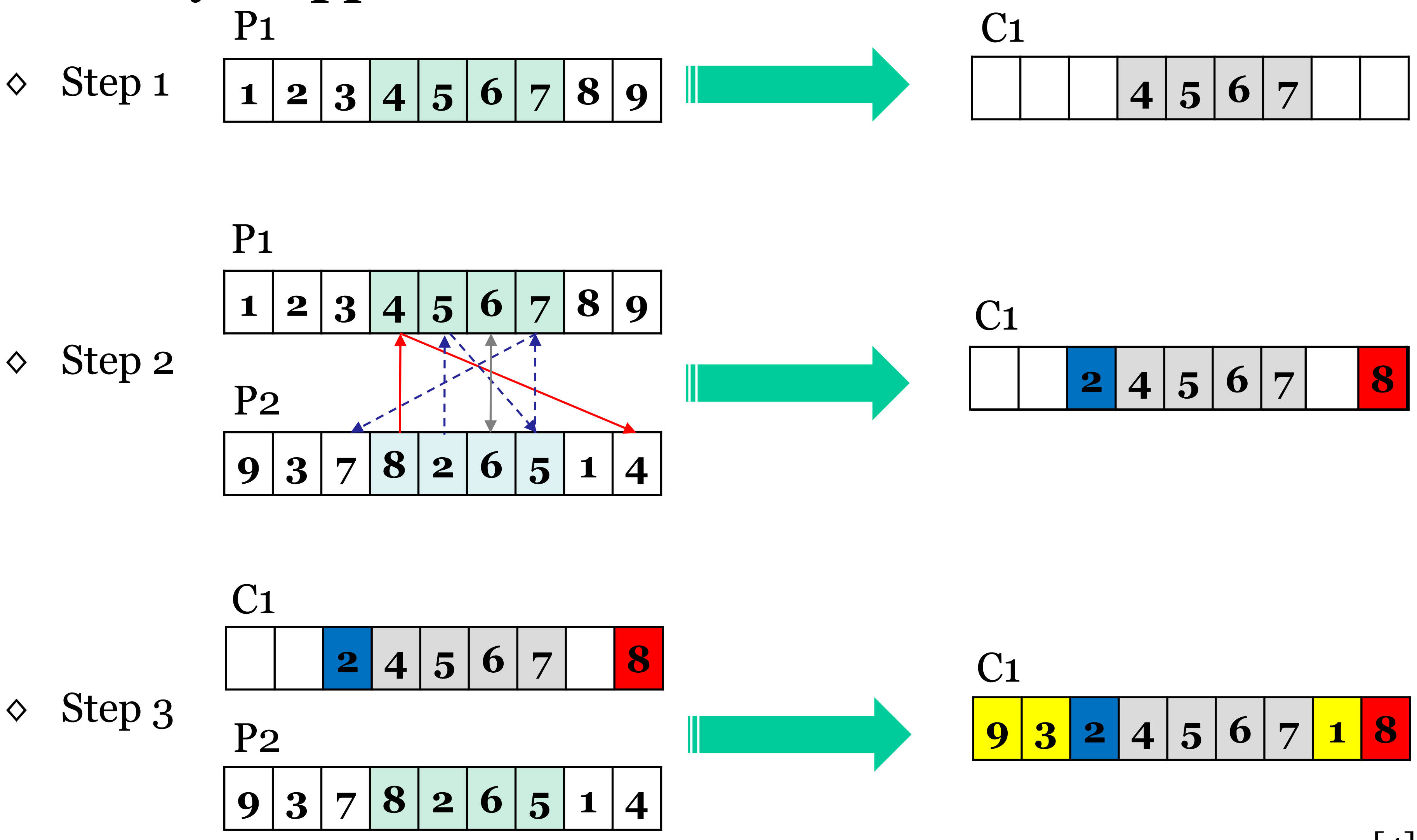
1. Choose two crossover points at random and copy the segment from P1 into child C1.
2. Starting from the first crossover point, look for elements in that segment of P2 that have not been copied.
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from 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.

[4]

Permutations GA

- **Partially Mapped Crossover (PMX)**



◇ Step 1

P1

1	2	3	4	5	6	7	8	9

C1

			4	5	6	7		

◇ Step 2

P1

1	2	3	4	5	6	7	8	9

P2

9	3	7	8	2	6	5	1	4

C1

		2	4	5	6	7		8

◇ Step 3

C1

		2	4	5	6	7		8

P2

9	3	7	8	2	6	5	1	4

C1

9	3	2	4	5	6	7	1	8

[4]

Permutations GA

- **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 parents.
  - ◇ In order to achieve this, an **edge table** (a.k.a **adjacency lists**) is constructed.

P1	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

P2	**9**	**3**	**7**	**8**	**2**	**6**	**5**	**1**	**4**

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

[4]

Permutations GA

- **Edge Crossover**

1. Construct edge table
2. Pick an initial element at random and out it in the offspring
3. Set the variable *current_element=entry*
4. Remove all references to *current_element* from the table
5. 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
6. In the case of reaching an empty list, the other end of the offspring is examined for extension, otherwise a new element is chosen at random.

[4]

Permutations GA

- **Edge Crossover**

P1	1	2	3	4	5	6	7	8	9
P2	9	3	7	8	2	6	5	1	4

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

Choices	Element Selected	Reason	Partial Results
All	1	Random choice	[1]
2,5,4,9	5	Shortest list	[1 5]
4,6	6	Common edge	[1 5 6]
7,2	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]

Permutations GA

- **Order 1 crossover**

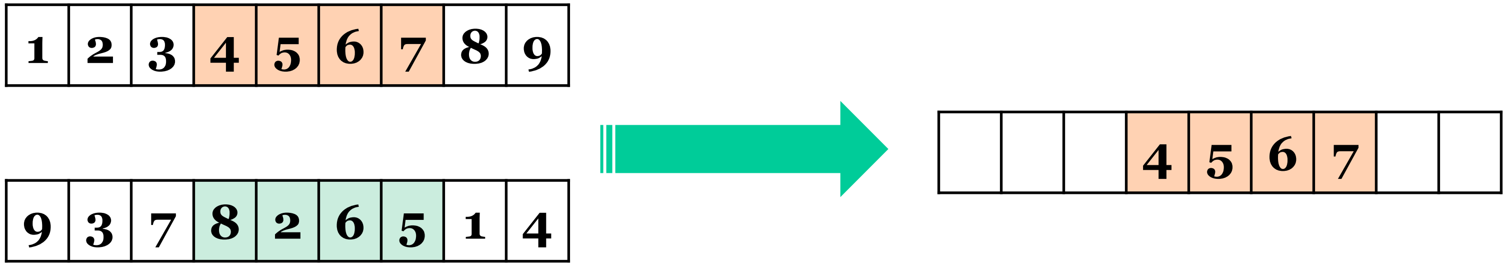
  The idea is to preserve relative order that elements occur.

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:
   - ◇ Start right from cut point of the copied part,
   - ◇ Use the order of the second parent,
   - ◇ Wrap around at the end.
4. Analogous for the second child, with parent roles reversed.

[4]

Permutations GA

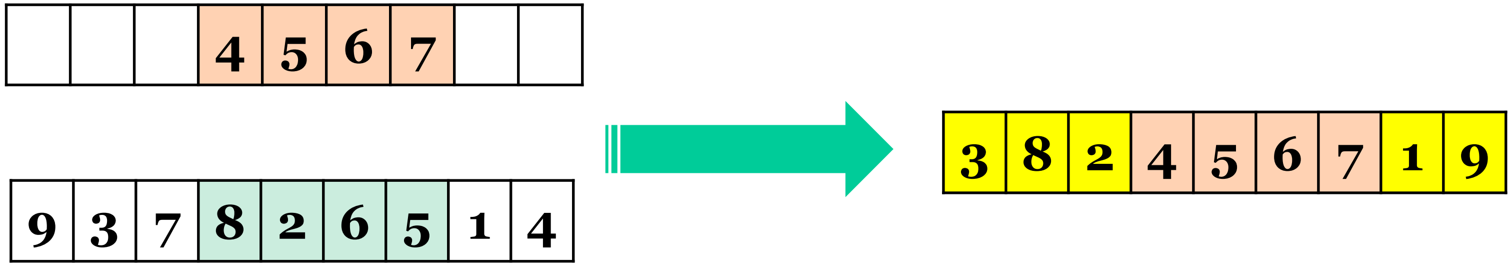
- **Order 1 crossover**
  - ◇ Copy randomly selected set from first parent.



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

			4	5	6	7		

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



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

3	8	2	4	5	6	7	1	9

[4]

Permutations GA

- **Cycle Crossover**

The operator works by diving 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 parents.

[4]

Permutations GA

- **Cycle Crossover**

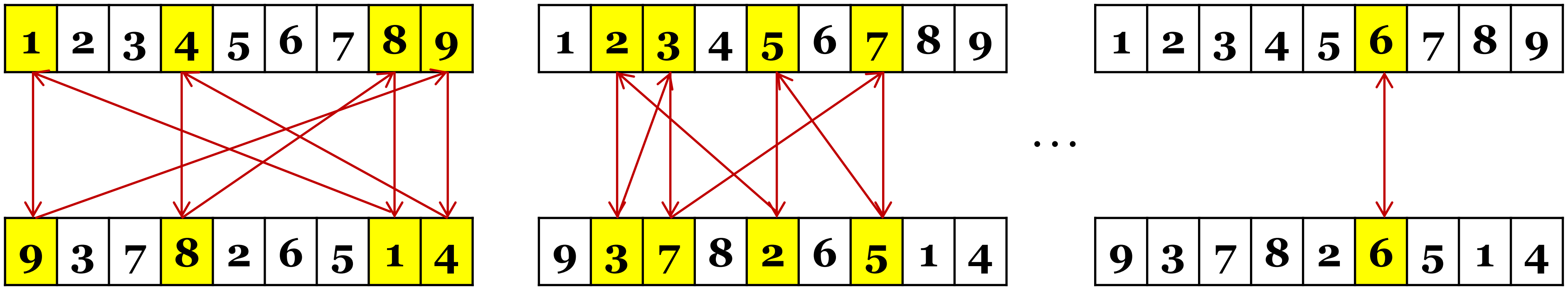
1. Start with the first unused position and allele of P1
2. Look at the allele in the *same position* in P2
3. Go to the position with the *same allele* in P1
4. Add this allele to the cycle
5. Repeat steps 2 through 4 until you arrive at the first allele of P1

[4]

Permutations GA

- **Cycle Crossover**

◇ Step 1: identification of cycles



1 2 3 4 5 6 7 8 9

9 3 7 8 2 6 5 1 4

1 2 3 4 5 6 7 8 9

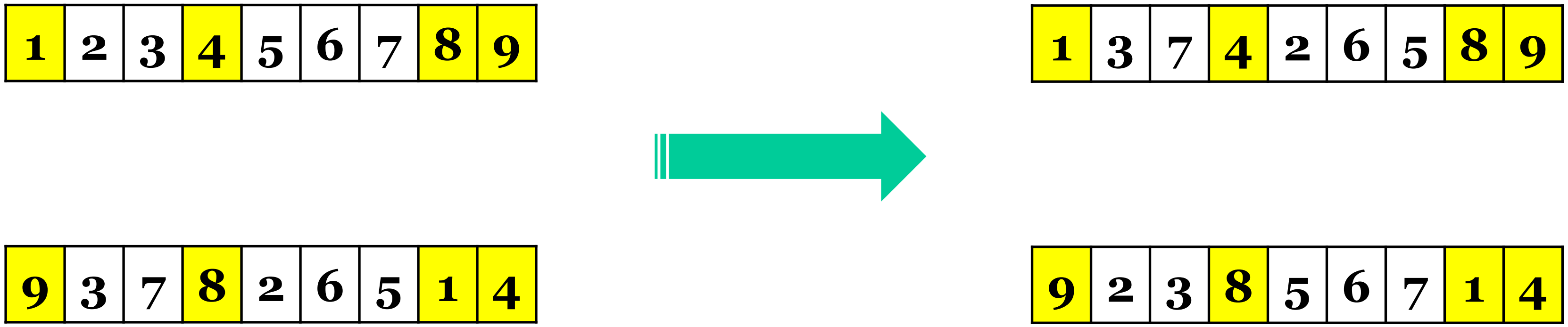
9 3 7 8 2 6 5 1 4

. . .

1 2 3 4 5 6 7 8 9

9 3 7 8 2 6 5 1 4

◇ Step 2: construction of offspring



1 2 3 4 5 6 7 8 9

9 3 7 8 2 6 5 1 4

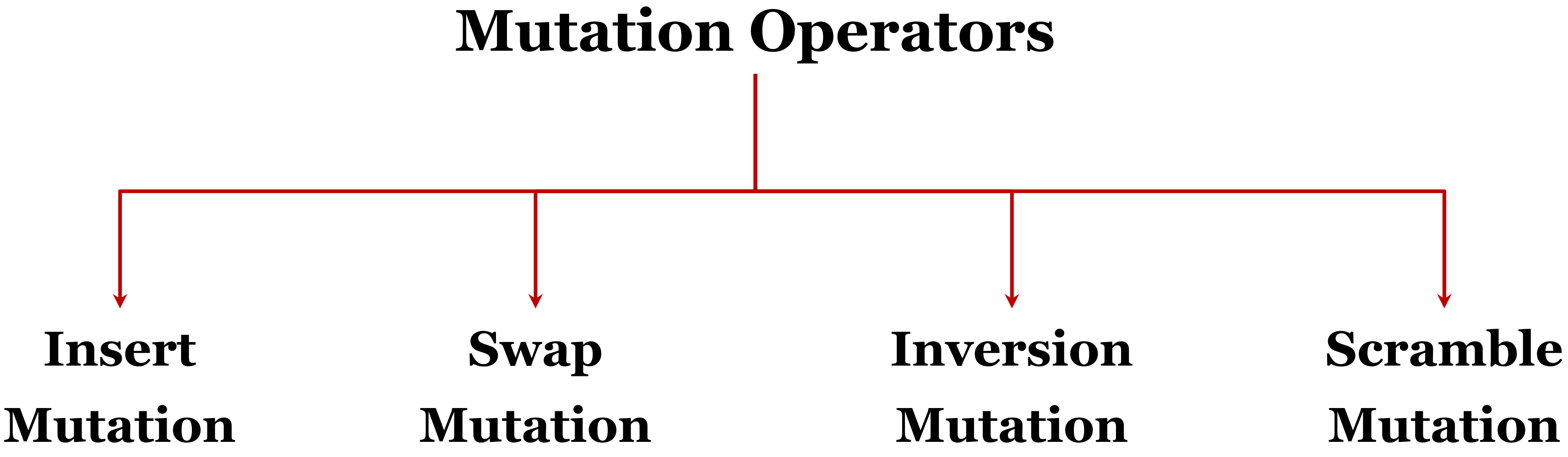
1 3 7 4 2 6 5 8 9

9 2 3 8 5 6 7 1 4

[4]

Permutations GA

- **Mutation:**



Mutation Operators

Insert Mutation

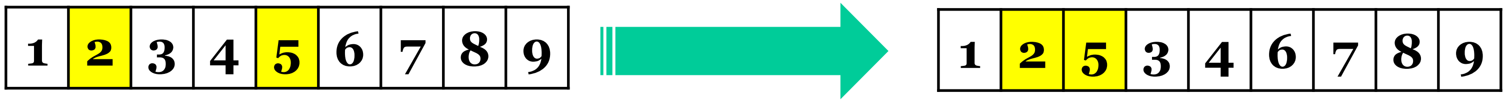
Swap Mutation

Inversion Mutation

Scramble Mutation

Permutations GA

- **Insert Mutation:**
  - ◇ Pick two genes values at random,
  - ◇ Move the second to follow the first, shifting the rest along to accommodate,
  - ◇ Preserves most of the order and the adjacency information.



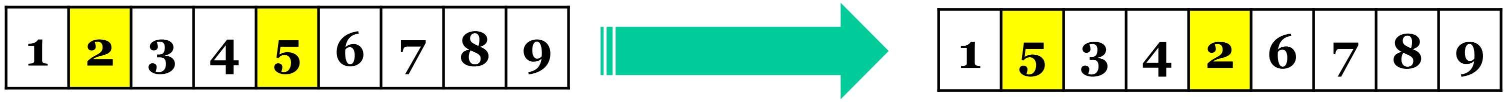
1	2	3	4	5	6	7	8	9

1	2	5	3	4	6	7	8	9

[4]

Permutations GA

- **Swap Mutation:**
  - ◇ Pick two genes at random and swap their positions,
  - ◇ Preserves most of adjacency information (4 links broken), disrupts order more.



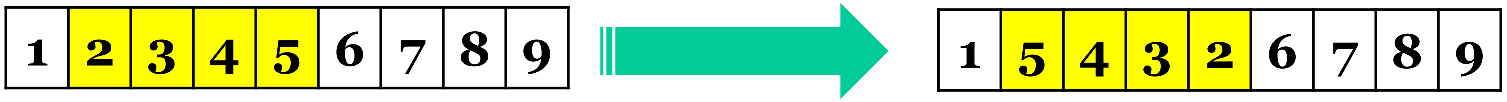
1	2	3	4	5	6	7	8	9

1	5	3	4	2	6	7	8	9

[4]

Permutations GA

- **Inversion Mutation:**
  - ◇ Pick two genes at random and then invert the substring between them,
  - ◇ Preserves most adjacency information (only breaks two links) but disruptive of order information.



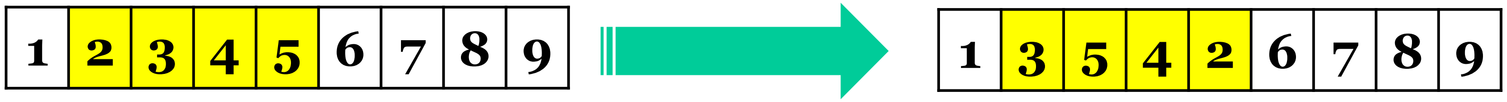
1	2	3	4	5	6	7	8	9

1	5	4	3	2	6	7	8	9

[4]

Permutations GA

- **Scramble Mutation:**
  - ◇ Pick two gene values at random,
  - ◇ Randomly rearrange the genes in those positions (note subset does not have to be contiguous)



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

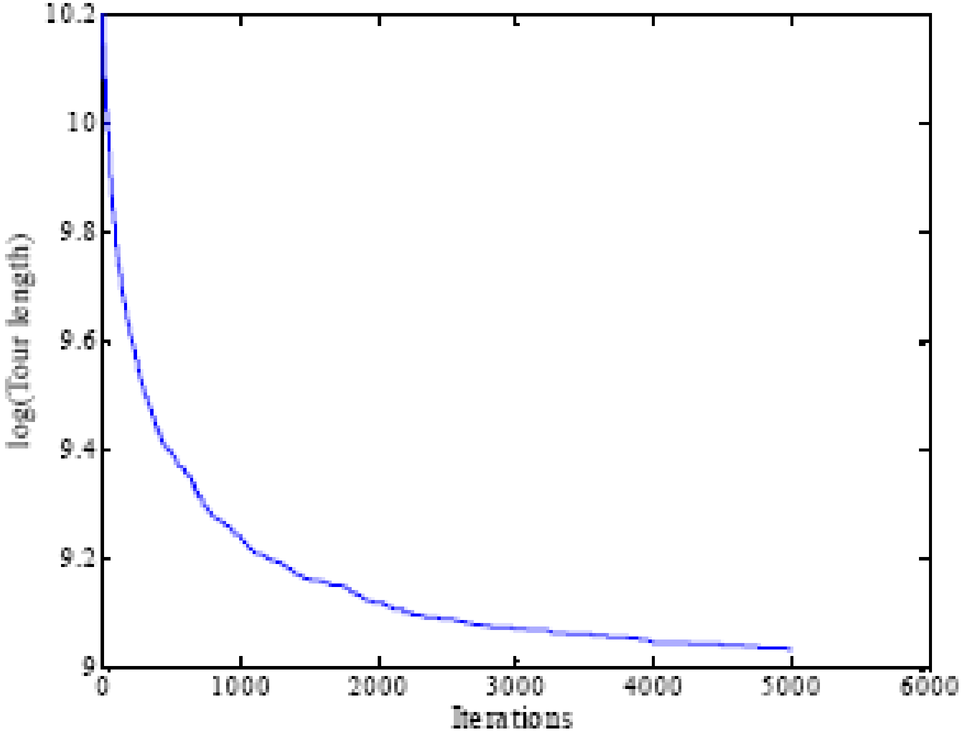
[4]

Permutations GA

- **Travelling Salesman Problem: GA for berlin52 TSP**
  - ◊ 20 individuals,
  - ◊ Using order 1 crossover producing only one child,
  - ◊ Crossover probability = 0.7,
  - ◊ Using swap mutation with one swap only,
  - ◊ Mutation probability = 0.05,
  - ◊ Using the elitism model while keeping only the best individual from one population to the next,
  - ◊ Using different number of iterations (500, 1000, 2000 and 5000).

Permutations GA

- **Travelling Salesman Problem: GA for berlin52 TSP**



Number of Iterations	Tour Length
500	12,135
1000	10,069
2000	8,986.2
5000	**8,730.3**

Optimal length= 8,236

References

1. Genetic Algorithms: Lecture notes in Traffic Engineering And Management, last accessed July 22, 2016.
2. Thomas Braunl. *Embedded Robotics*. Springer, 2006.
3. Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. 2nd Edition, John Wiley & Sons Ltd, 2007.
4. A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, 2007.
5. El-Ghazali Talbi. *Metaherusitcis: From Design to Implementation*. John Wiley, ISBN: 978-0-470-27858-1, 2009
6. Crina Grosan and Ajith Abraham. *Intelligent Systems: A Modern Approach*. Springer, 2011.

Useful Resources

Google Patent Search gives about **212,000 results** and Google Scholar gives **1,840,000 results** "Genetic Algorithm" as a keyword.

Interesting GA videos:

- Creating a flying snake using GA
- Genetic Algorithm - explained in 4 minutes
- GA helps Super Mario
- GA finds Waldo, Victoria McNally , Graduate Student Creates Algorithm To Beat Where's Waldo At His Own Game, February 8th 2015

Useful Resources

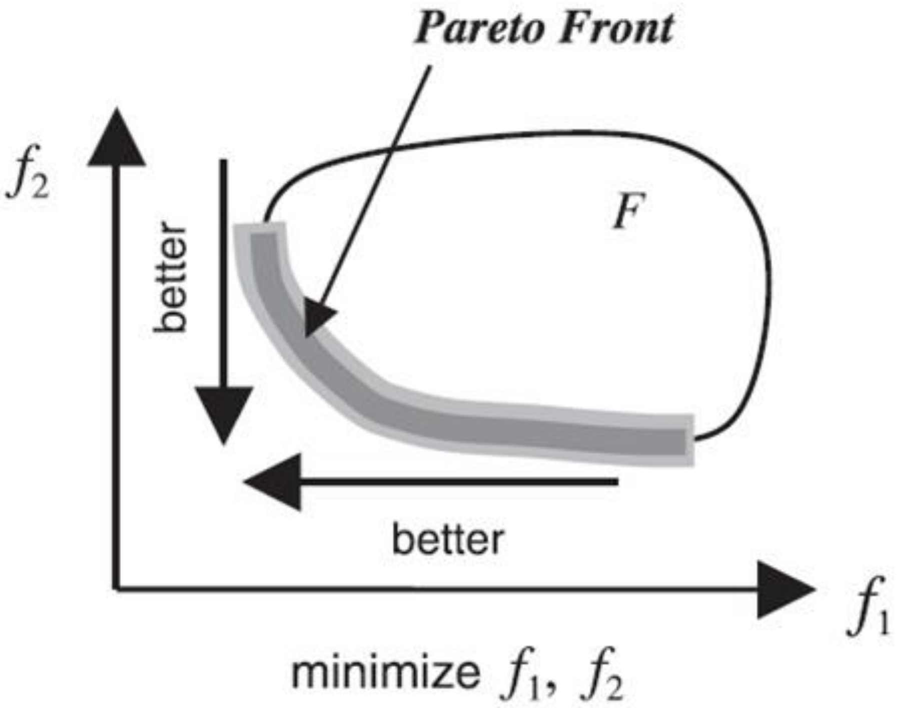
Interesting recent articles about GA:

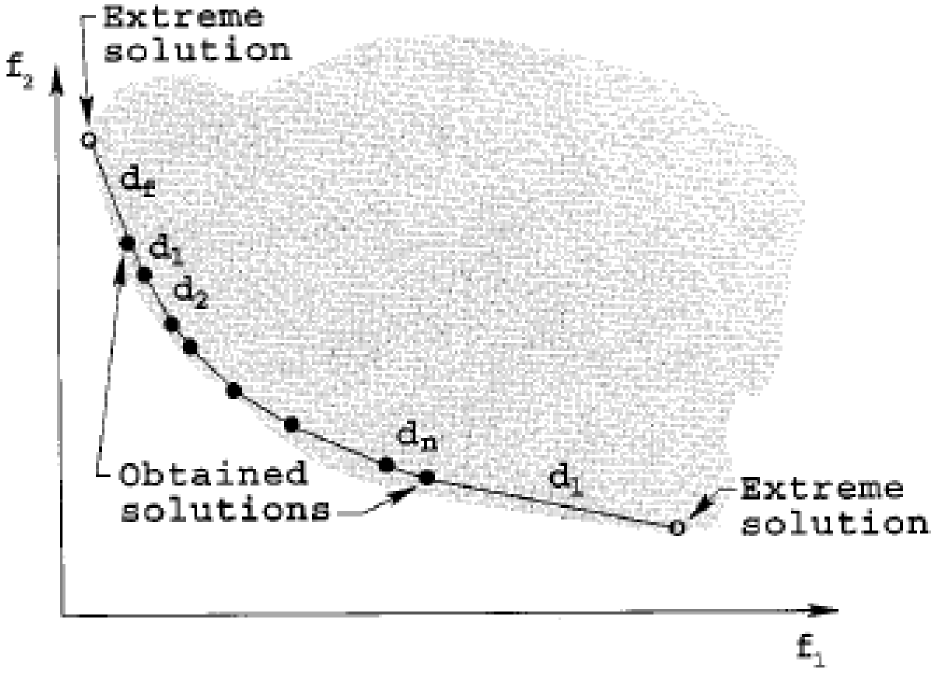
- Randal Olson, How I Found the Optimal Where's Waldo Strategy With Machine Learning, Wired, Feb. 4, 2015.
- Ritesh Anan, 'I Know First': An Advanced Self-Learning, Predictive Algorithm For Everyone, Jan. 14, 2015, benzinga
- Douglas McCormick, Fighting Buggy Code with Genetic Algorithms, Mar 10, 2014, IEEE Spectrum
- Jim Erickson, Know your enemy: Combating whooping cough requires informed vaccine booster schedules, Jan 19, 2015, Michigan News.
- Micah Wright. Can Hydrogen Now Be Created From Corn? June 2, 2015

Useful Resources

Multi-objective Optimization:

Nondominated sorting genetic algorithm II (NSGA-II)





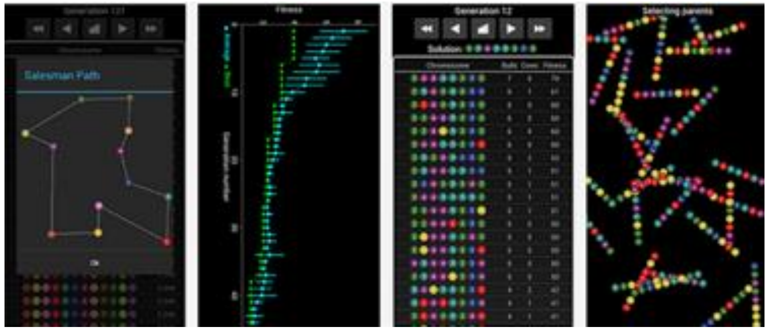
Paper: Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II”, IEEE Transactions on Evolutionary Computation, 6(2):182-197, April 2002

Code: NSGA - II: A multi-objective optimization algorithm

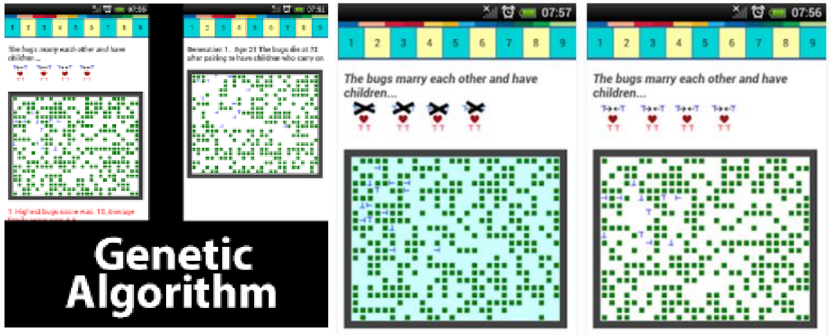
Useful Resources

Interesting Apps:

Android Genetic Algorithm Explorer



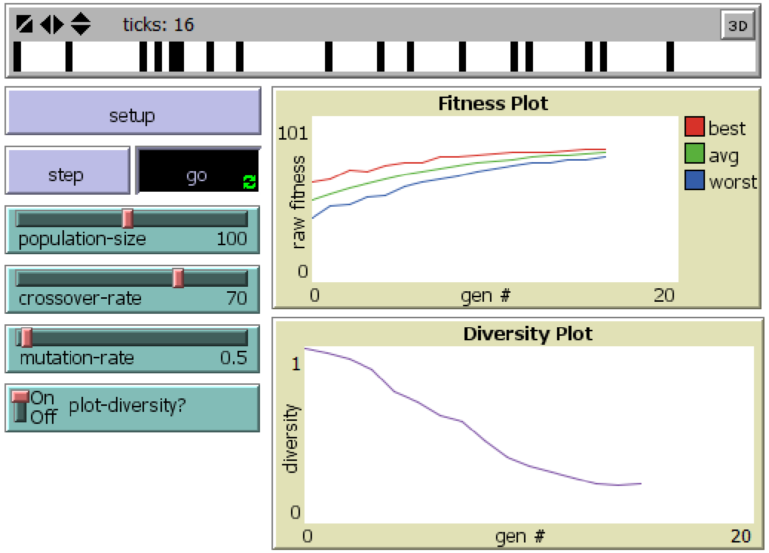
Android Genetic Algorithm



The bugs marry each other and have children...

NetLogo Simple Genetic Algorithm:

http://ccl.northwestern.edu/netlogo/models/SimpleGeneticAlgorithm



ticks: 16

population-size 100

crossover-rate 70

mutation-rate 0.5

On Off plot-diversity?

More in *Resources* :

http://alaakhamis.org/teaching/ECE457A_S2015/resources.html