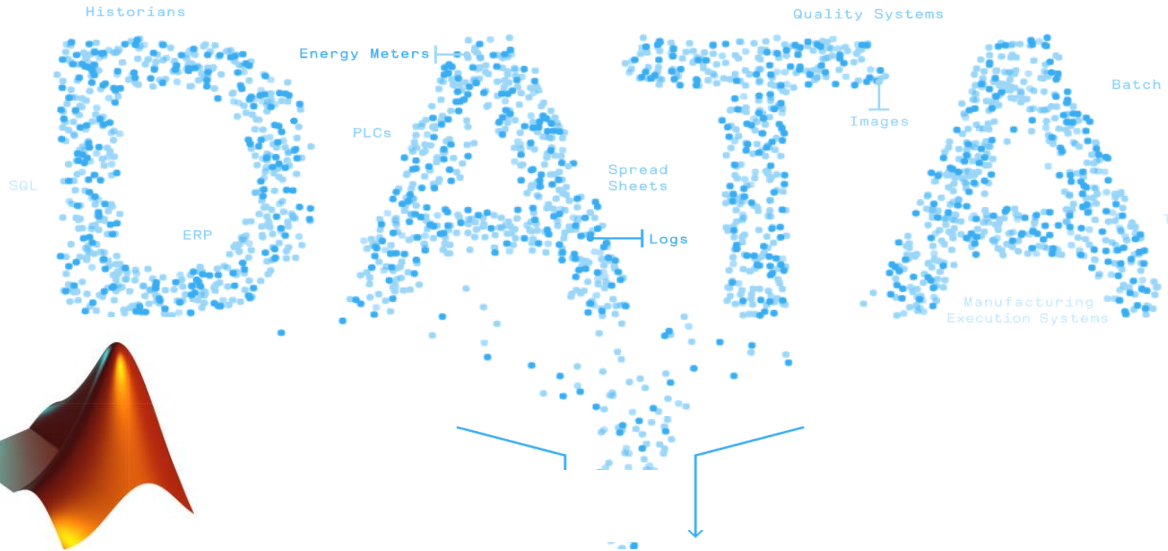
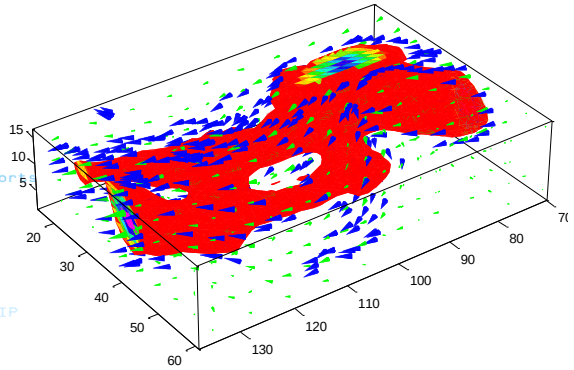
Introduction to Artificial Intelligence -02-02 Genetic Algorithms

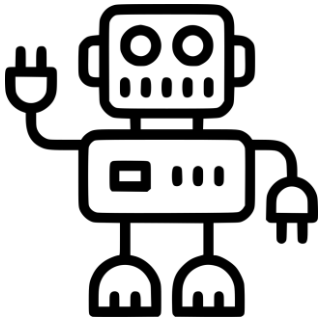




Dr Leo Chen
leo.chen@ieee.org
23/Feb/2023

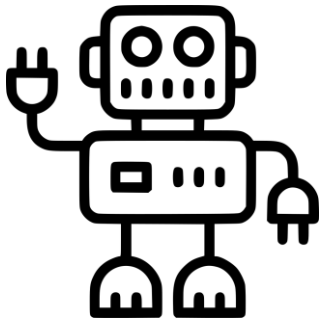
Module Contents

1. Introduction
2. Evolutionary Computation
3. Artificial Neural Network
4. Fuzzy Logic and Fuzzy Systems
5. More AI Subsets
6. AI and Industry 4.0
7. AI Applications
8. Labs
9. Courseworks



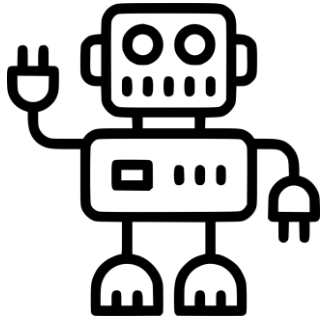
Chapter Contents

1.Introduction
2.Genetic Algorithms (GA)
3.Genetic Programming (GP)
4.Evolution Strategies (ES)
5.Evolutionary Programming (EP)
6.Related Topics



Section Contents

- **What is Genetic Algorithms**
- **History of Genetic Algorithms**
- **Basic Structure and Workflow**
- **Terminology**
- **‘New’ Genetic Algorithms**
- **Genetic Algorithms Tools for MATLAB**
- **Case Studies**



https://github.com/SGALAB/What-is-Genetic-Algorithms

What is Genetic Algorithms [4-8]

- Genetic algorithms (GAs) are a family of computational methods inspired by **Darwin's evolutionary theory** (natural selection), which are based on a biological metaphor-They view learning as a **competition among a population** of evolving candidate problem solutions.

- In GAs, a population of candidate solutions (individuals , or **phenotypes**) to an optimisation problem is **evolved toward better** solutions.

What is Genetic Algorithms [4-8]

- Each candidate solution has a set of properties (its chromosomes or genotype) which can be **mutated** and **altered**.
- Traditionally, solutions are represented in **binary** as strings of 0s and 1s, but other **encodings** are also possible.
- GAs are commonly used to generate high-quality solutions to optimisation and search problems by relying on biologically inspired **operators** such as, *mutation, crossover and selection*.

What is Genetic Algorithms [4-8]

- A **'fitness' function evaluates** each solution to decide whether it will contribute to the next generation of solutions. *In each generation*, the fitness of every individual in the population is evaluated;
- the **fitness** is usually the value of the **objective** function in the optimisation problem being solved.
- The more fit individuals are **selected** from the current population, and each individual's genome is **modified** (by **operators** to form a new generation.

Section Contents

- What is Genetic Algorithms
- History of Genetic Algorithms
- Basic Structure and Workflow
- Terminology
- ‘New’ Genetic Algorithms
- Genetic Algorithms Tools for MATLAB
- Case Studies

https://github.com/SGALAB/What-is-Genetic-Algorithms

History of Genetic Algorithms

- As early as **1962**, John Holland(1926-2015)'s work on **adaptive systems** laid the foundation for GAs' later developments.
- Holland was also the first to explicitly propose **crossover** and other recombination **operators**.
- *GAs were formally introduced in the **1975** by John Holland* at University of Michigan by the book '**Adaptation in Natural and Artificial Systems**'

History of Genetic Algorithms

- In 1975, this book was the **first** to **systematically** and rigorously present the concept of adaptive digital systems using mutation, selection and crossover, simulating processes of biological evolution, as a problem-solving strategy.

- The book also attempted to put GAs on a firm **theoretical footing** by introducing the notion of **schemata**.

History of Genetic Algorithms

- Also in 1975, Kenneth De Jong's important dissertation established the potential of GAs by showing that they could perform well on a wide variety of **test functions**, including noisy, discontinuous, and multimodal search landscapes.
  - Further GAs development, some of them will be listed in the Section **'New' Genetic Algorithms**, including:

    muGA, Mendel GA, NPGA, NSGA, etc.

LSA UNIVERSITY OF MICHIGAN

DEPARTMENT OF Psychology

CONTACT US | MAP | WELCOME

HOME | GRADUATE PROGRAM | UNDERGRADUATE PROGRAM | PROGRAM AREAS | PEOPLE

HOME : PEOPLE : DIRECTORY : FACULTY PROFILE — JOHN HOLLAND

DIRECTORY
FACULTY RESEARCH INTERESTS
LABS
FACULTY POSITIONS
JOB OPENINGS
FACULTY MEDIA CONTACTS
ROOM SCHEDULES

QUICKLINKS

faculty in the news
alumni & friends

FACULTY PROFILE — JOHN HOLLAND

http://www.lsa.umich.edu/psych/people/directory/profiles/faculty/?uniquename=jholland

Professor of Psychology and Electrical Engineering & Computer Science
Ph.D. University of Michigan
Area: Cognition & Perception

Contact Information

Office: 1255 East Hall
Phone: 763-3648
Email: jholland@umich.edu

Research and Teaching Interests

Study of cognitive processes and complex adaptive systems in general, using mathematical models and computer simulation.

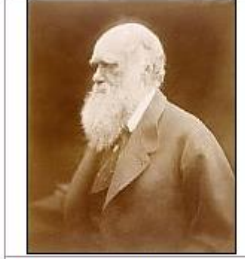
http://www.lsa.umich.edu/psych/people/directory/profiles/faculty/?uniquename=jholland

Related Links

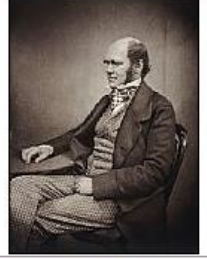
- Dept. of Electrical Engineering & Computer Science
- EECS Faculty Profile

http://www.lsa.umich.edu/psych/people/directory/profiles/faculty/?uniquename=jholland

Richmond - Charles Darwin J980057

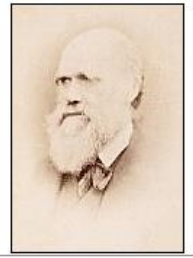


Charles Darwin K980352

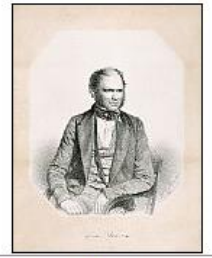


Charles Darwin K970247

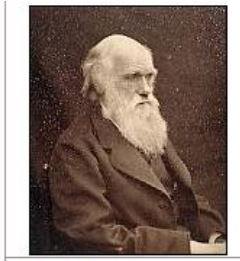
Charles Darwin on the verandah at Down House K970226



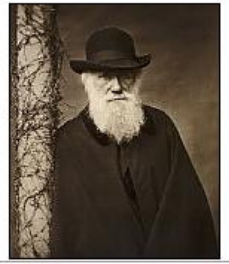
Charles Darwin K970234



Charles Darwin K970239



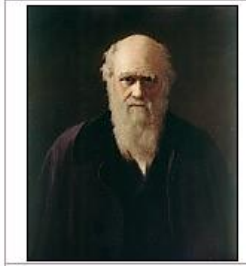
Charles Darwin K970215



Charles Darwin K980123

Charles Darwin on horseback K970217

Laurence - Charles Darwin J970202



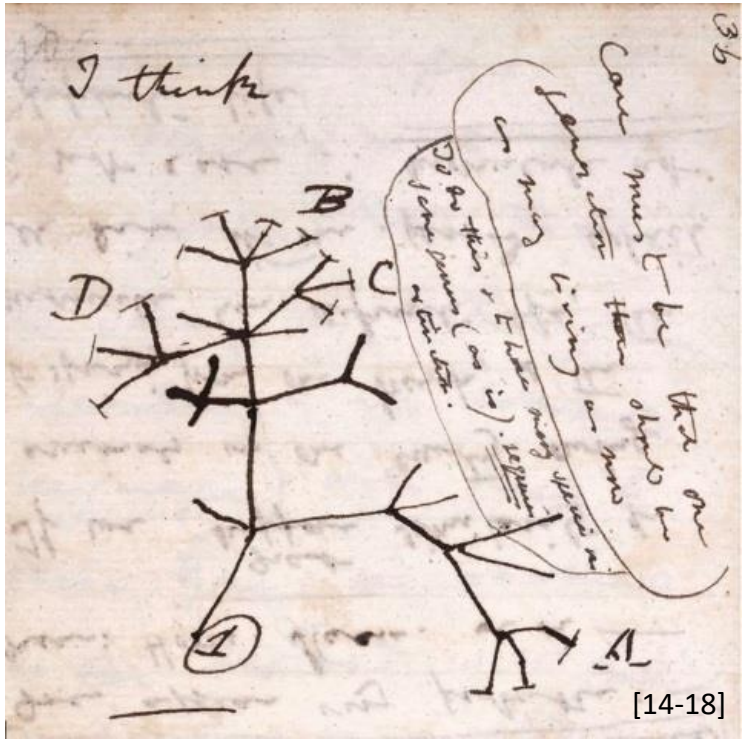
Reilly - Charles Darwin J970164

Evstafieff - Darwin in his Study J970178

Charles Darwin and his son N990002

Charles Darwin K970235

Sharples - Charles Darwin (aged six) and Catherine K971925



Charles Darwin (1809 - 1882)

01 - Charles Darwin A Short Biography (4:46mins)

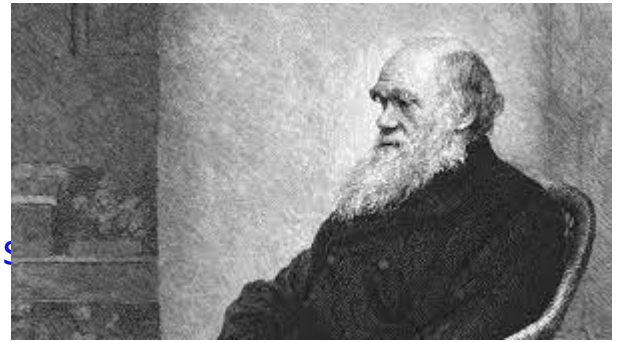
02 - The Truth About Charles Darwin (9:55mins)

03 - BBC News-Theory of Evolution How did Darwin come up with it (5:23mins)

04 - Richard Dawkins - The Genius of Charles Darwin - Part 1_ Life, Darwin & Everything (48:10mins)

05 - What is Natural Selection (9:18mins)

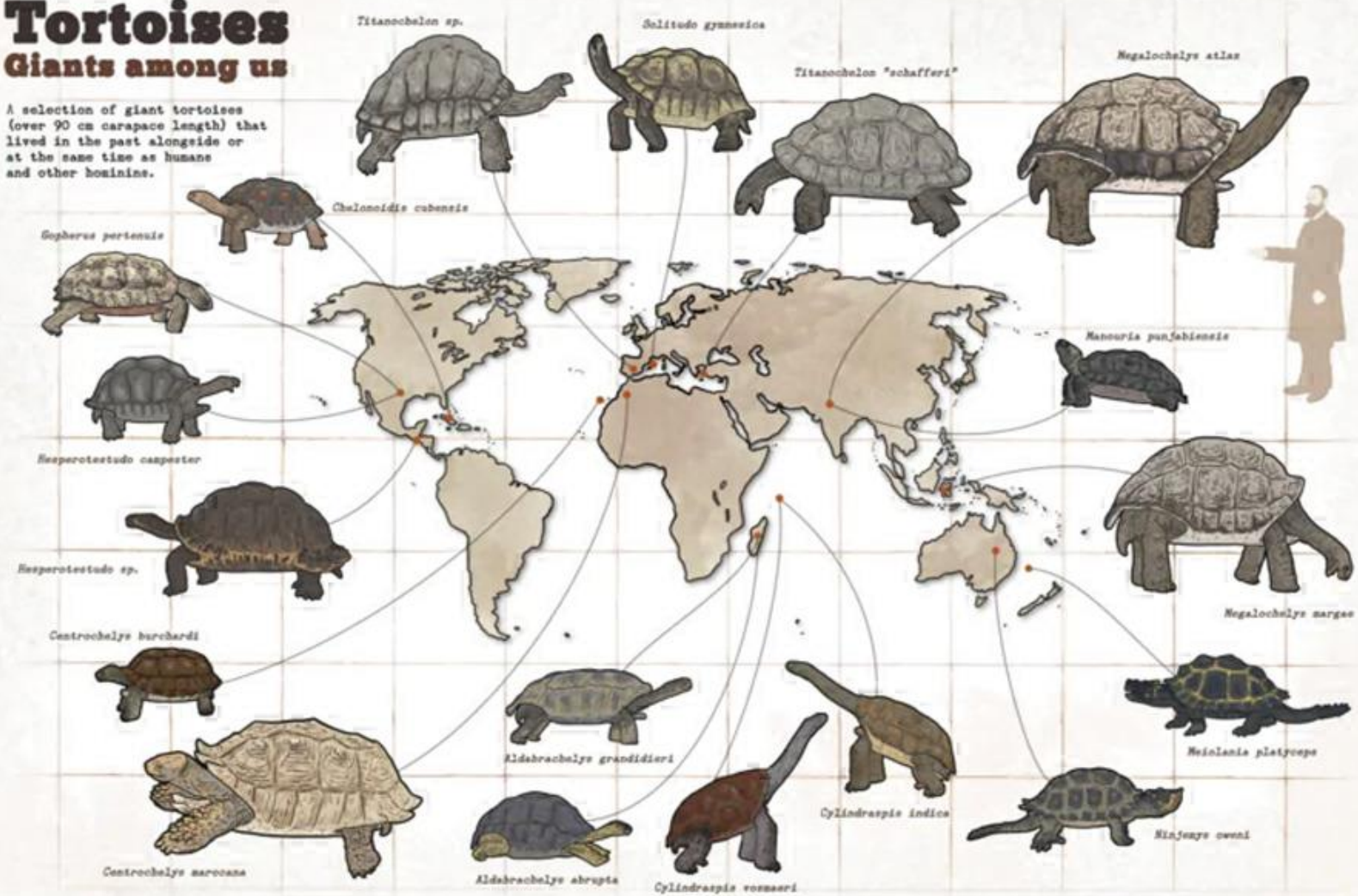
06 - Simulating Natural Selection (10:00mins



Tortoises

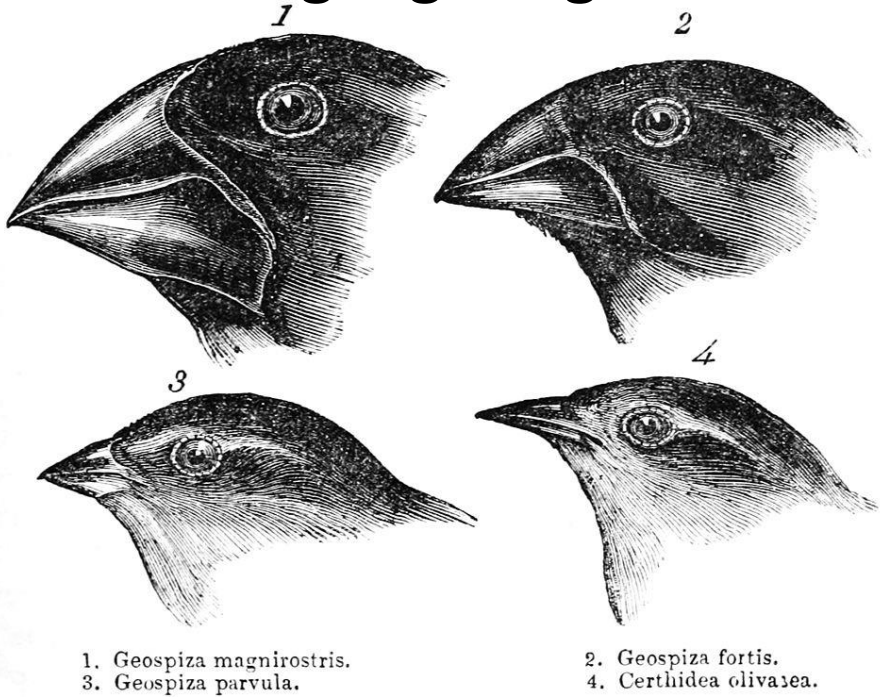
Giants among us

A selection of giant tortoises (over 90 cm carapace length) that lived in the past alongside or at the same time as humans and other hominins.



Competition in Life

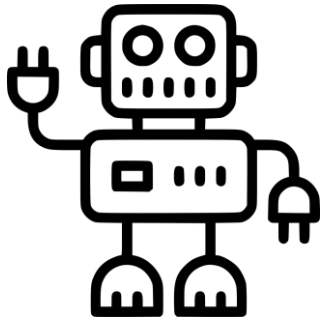
- From crawling to standing
- Standing higher gets more



Darwin's finches by John Gould

Section Contents

- What is Genetic Algorithms
- History of Genetic Algorithms
- Basic Structure and Workflow
- Terminology
- ‘New’ Genetic Algorithms
- Genetic Algorithms Tools for MATLAB
- Case Studies



https://github.com/SGALAB/What-is-Genetic-Algorithms

Basic Structure and Workflow

- **Structure of Genetic Algorithms**
- **Basic Steps**
- **Basic Workflow**
- **Major Advantages**

5 basic components


genetic representation


population


evaluation


genetic operators


parameter values

Structure of Genetic Algorithms - 5 basic components [19]

1. A genetic **representation** of potential solutions to the problem. (via encoding and decoding)
2. A way to create a **population** (an initial set of potential solutions).
3. An evaluation (objective) function rating solutions in terms of their **fitness**.
4. **Genetic operators** that alter the genetic composition of offspring (*selection, crossover, mutation,* etc.).
5. **Parameter values** that genetic algorithm uses (*population size, probabilities of applying genetic operators,* etc.).

Dr Leo Chen, leo.chen@ieee.org

Basic Steps

1 Genetic Representation and Initialisation:

- The genetic algorithm maintains a **population** $P(t)$ of **chromosomes** or **individuals** $v_k(t)$, k=1, 2, …, *popSize* for generation t.

- Each chromosome represents a potential solution to the problem at hand.

Dr Leo Chen, leo.chen@ieee.org

Basic Steps

2 Genetic Operators:

- Some chromosomes undergo stochastic transformations by means of genetic operators to form new chromosomes, *i.e.,* **offspring**.

- **Selection**, a new population is formed by selecting **the more fit chromosomes** from the *parent* population and the *offspring* population.

Dr Leo Chen, leo.chen@ieee.org

Basic Steps

- There are two kinds of transformation:
  - **Crossover**, which creates new chromosomes by combining parts from two chromosomes.
  - **Mutation**, which creates new chromosomes by making changes in a single chromosome.
- New chromosomes, called **offspring** $C(t)$, are then evaluated

Dr Leo Chen, leo.chen@ieee.org

Basic Steps

3 **Evaluation**:

- Each chromosome is evaluated to give some measure of its **fitness** $eval(v_k)$.

4 Termination Condition:

- The termination condition of a GA is important in determining when a GA run will **end**.
- The GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the **improvements are very small**.

Basic Steps

5 Best solution:

After several generations, the algorithm converges to **the best chromosome**, which hopefully represents an optimal or suboptimal solution to the problem.

Dr Leo Chen, leo.chen@ieee.org

GAs Basic Workflow

Max : $F(x_i) = \{ f_1(x_i), f_2(x_i), \cdots, f_k(x_i) \}$

Subject to: $g_j(x_i) \le 0$

$i = 1,2,\cdots,m$

$j = 1,2,\cdots,n$

(1) Object design based on engineering application

Start (2) Variable/Fitness function design

initial solution

x(1) x(2) x(n)

population

(10.4554)(74545.4)...(5.4345)

(11.0230)(75690.3)...(5.2344)

(9.99234)(74545.9)...(5.2322)

(10.4544)(74544.3)⋯(5.123)

Fitness/Cost Evaluation

No

En-coding

x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)

population

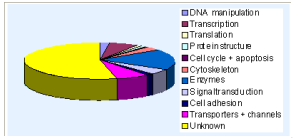
10001010101010100...0101

01010000010100101...1101

01000101000101010...1110

00101000101010101...1000

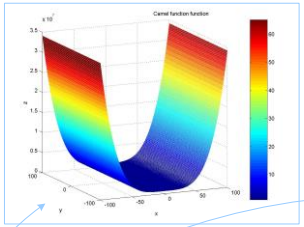
Selection

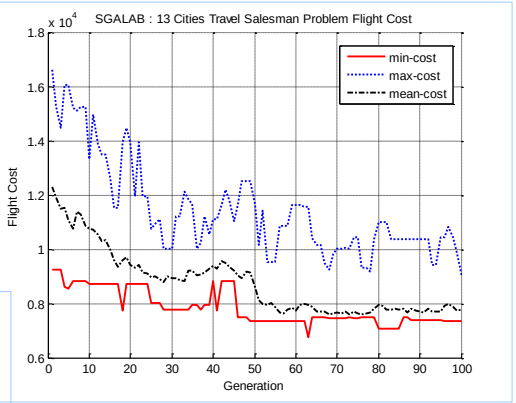


Crossover

Yes

De-coding





x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)

population

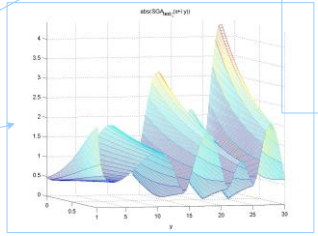
10001010101000101...1101

01010000010110100...0101

00101000101**101010...1110**

01000101000010101...1000

Mutation



Stop

Result & Analysis

x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)

population

10001010**1**110**00101...1101**

0101100010010110100...0101

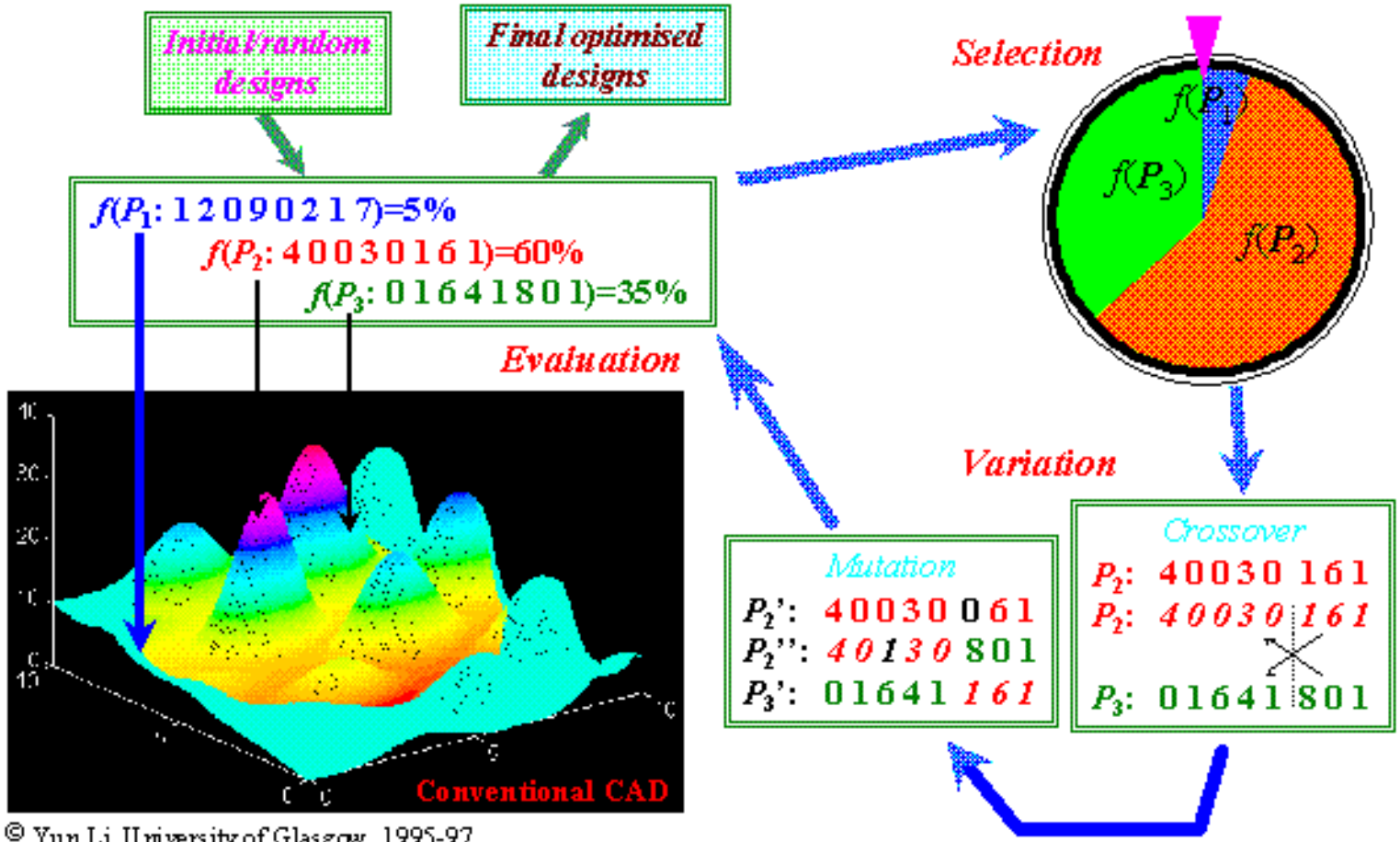
00101000011**101010...110**0

010001010100100**0**1...1000

GAs Basic Workflow (Pseudo Code)

```
Begin (1)
  t = 0 ;
  Initialize P(t);
  Evaluate P(t);
    While ( Not termination-condition) do
      Begin (2)
       { t = t+1;
            Select P(t) from P(t-1)
            Crossover P(t);
            Mutation P(t);
         Evaluate P(t); }
       End (2)
 End (1)
```

Computer-Automated Design by Artificial Evolution



Major Advantages (1/3)

1 Genetic algorithms do not have much mathematical requirements about the optimisation problems.

- Due to their evolutionary nature, genetic algorithms will search for solutions without regard to the specific inner workings of the problem.
- Genetic algorithms can handle any kind of fitness (objective) functions and any kind of constraints, *i.e.*, linear or nonlinear, defined on discrete, continuous or mixed search spaces.

Dr Leo Chen, leo.chen@ieee.org

Major Advantages (2/3)

2 The ergodicity of evolution operators makes genetic algorithms very effective at performing global search (in probability).

- The traditional approaches perform local search by a convergent stepwise procedure, which compares the values of nearby points and moves to the relative optimal points.
- Global optima can be found only if the problem possesses certain convexity properties that essentially guarantee that any local optima is a global optima.

Dr Leo Chen, leo.chen@ieee.org

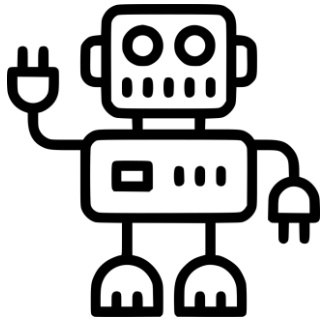
Major Advantages (3/3)

3 Genetic algorithms provide us a great flexibility to hybridise with domain dependent heuristics to make an efficient implementation for a specific problem.

Dr Leo Chen, leo.chen@ieee.org

Section Contents

- What is Genetic Algorithms
- History of Genetic Algorithms
- Basic Structure and Workflow
- Terminology
- ‘New’ Genetic Algorithms
- Genetic Algorithms Tools for MATLAB
- Case Studies



https://github.com/SGALAB/What-is-Genetic-Algorithms

Terminology [1-3,20]

1. Fitness Functions
2. Fitness Values and Best Fitness Values
3. Individuals
4. Populations
5. Chromosomes, Gene, Allele
6. Generations
7. Parents and Children
8. Diversity

Terminology

9. Genetic Representation
10. Genetic Operators
11. Evaluation
12. Phenotype and Genotype
13. Decoding and Encoding
14. Termination Condition

1. Fitness Function

- The ***fitness function*** is the function you want to optimise.
- For standard optimisation algorithms, this is known as the ***objective function***.
- The GAs are trying to find the maximum or minimum of the fitness function.
- When coding, write the fitness function as a standard MATLAB function, and pass it as a function handle input argument to the main GA function.

2. Fitness Values and Best Fitness Values

- The *fitness value* of an individual is the value of the fitness function **for** that individual.
- Because the Gas toolbox software finds the **maximum** or **minimum** of the fitness function, so the ***best*** fitness value for a population is the **biggest** or **smallest** fitness value for any individual in the population.

3. Individuals

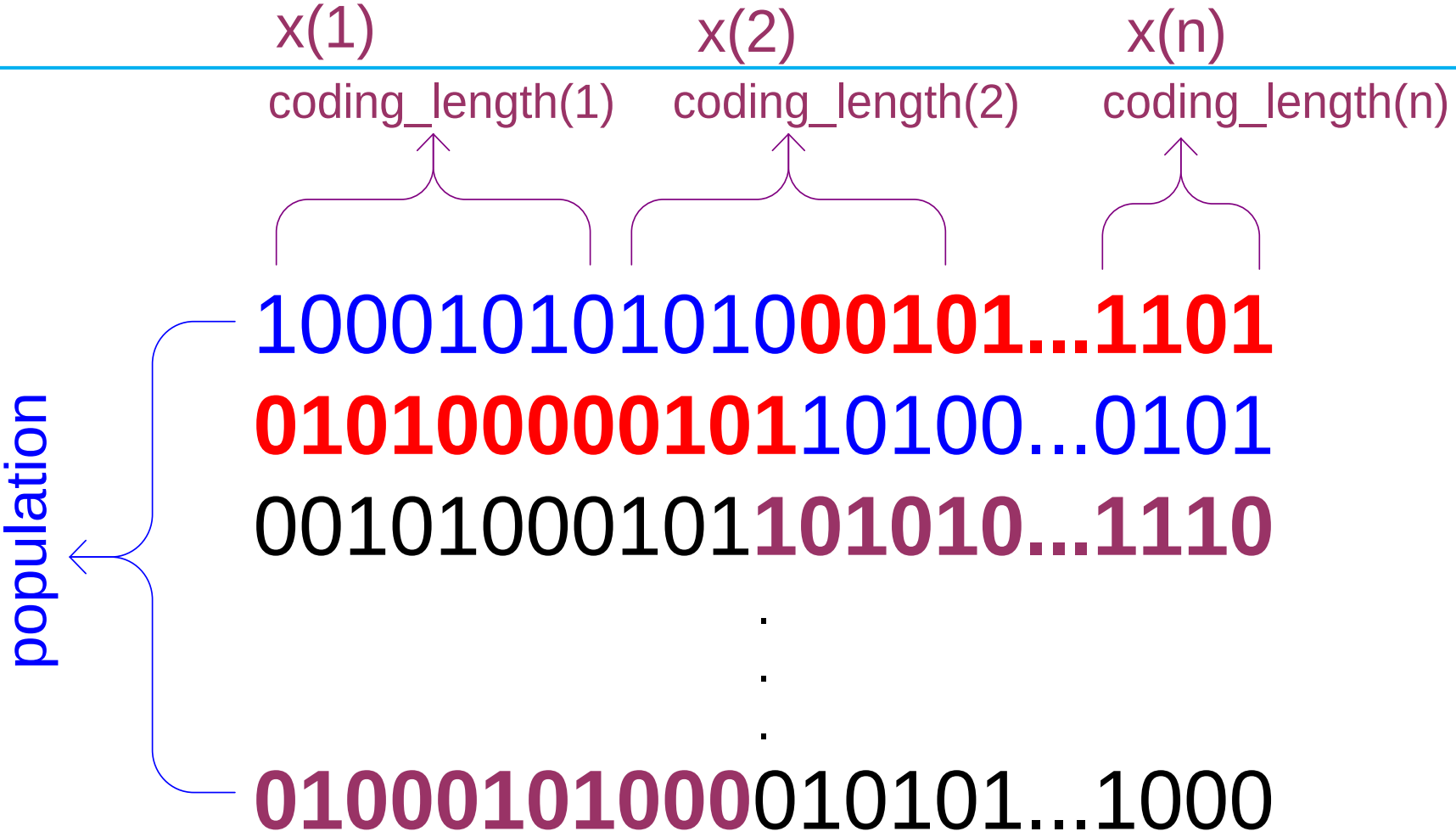
- An *individual (X)* is any point to which you can apply the fitness function f(X). The value of the fitness function for an individual is its score.
- For example, if the fitness function ***f(X)*** is

$$f(x_1, x_2, x_3) = (2x_1 + 1)^2 + (3x_2 + 4)^2 + (x_3 - 2)^2$$

- *The individual X* = (2, -3, 1)
- The fitness value (score) of the individual (2, –3, 1) is *f*(2, –3, 1) = 51.
- An individual is sometimes referred to as a *genome* and the vector entries of an individual as *genes*.

4. Populations

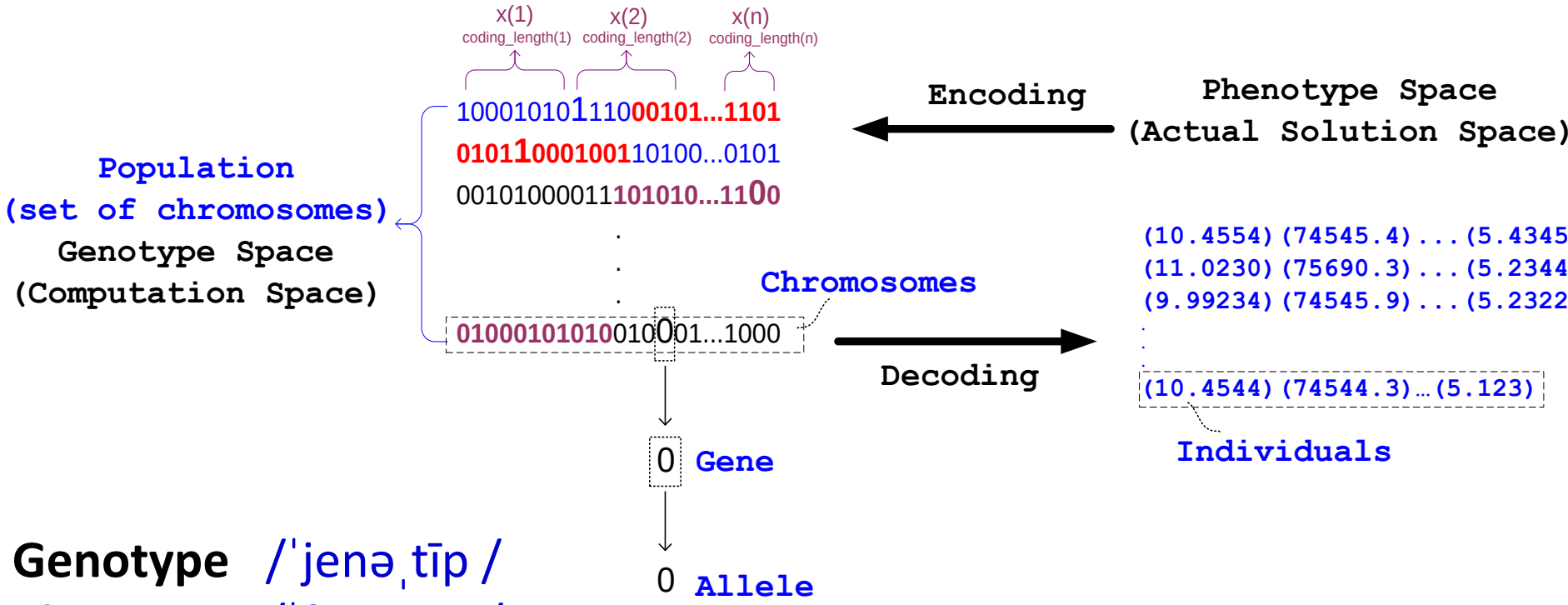
- A population is an **array** of individuals.
- **For example**, if the size of the population is 100 and the number of variables in the fitness function is 3, you represent the population by a **100-by-3** matrix.
- The same individual can appear **more than once** in the population.
- **For example**, the individual (2, -3, 1) can appear in more than one row of the array.


x(1)
x(2)
x(n)
coding_length(1)
coding_length(2)
coding_length(n)
population
1000101010100010 1...1101
0101000001011010 0...0101
0010100010110101 0...1110
0100010100001010 1...1000

5. Chromosomes, Gene, Allele

- **Chromosomes** /ˈkrōməˌsōm/ – A chromosome is one such solution to the given problem.
- **Gene** /jēn/ – A gene is one element position of a chromosome.
- **Allele** /əˈlēl/ – It is the value a gene takes for a particular chromosome.

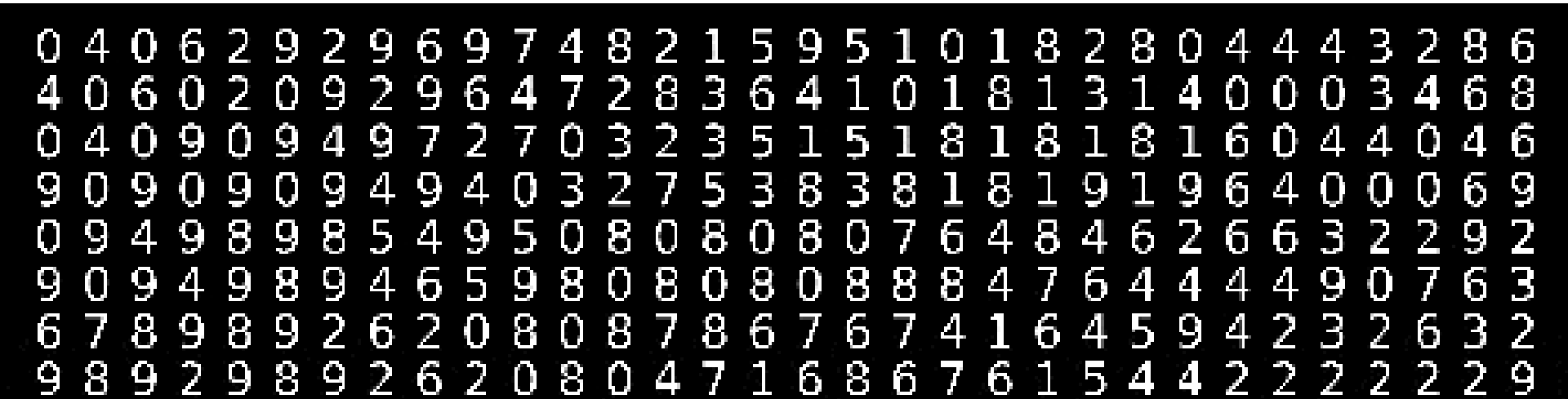
Encoding and Decoding



Genotype /ˈjenəˌtīp/
Phenotype /ˈfēnəˌtīp/

6. Generations

- At each iteration, the genetic algorithm performs a series of computations on the **current population** to produce a **new population**.
- Each successive population is called a new *generation*.

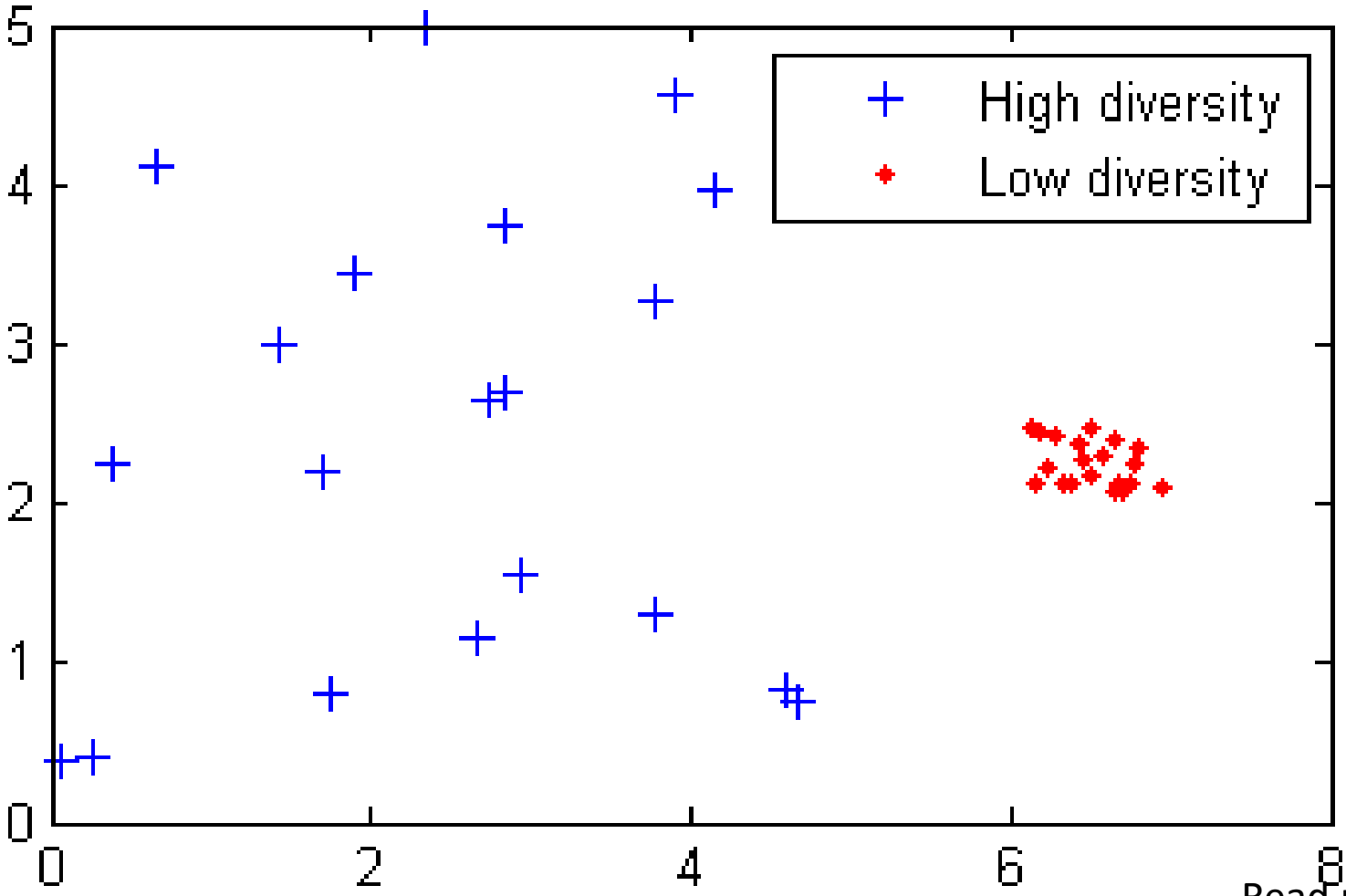


7. Parents and Children

- To create the next generation, the GA selects certain individuals in the **current** population, called ***parents***, and uses them to create individuals in the next generation, called ***children***.
- New chromosomes, called **offspring**
- Typically, the algorithm is more likely to select parents that have **better fitness** values.

8. Diversity

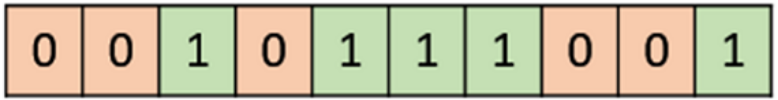
- Diversity refers to the **average** distance between individuals in a population.
- A population has high diversity if the **average** distance is **large**; otherwise it has low diversity.
- For example, in the following figure, the population on the **left** has **high** diversity, while the population on the **right** has **low** diversity.
- Diversity is essential to the GAs because it enables the algorithm to search a larger region of the space.



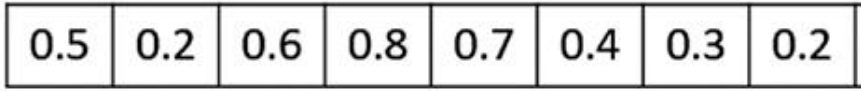
Read more [25]

9. Genetic Representation

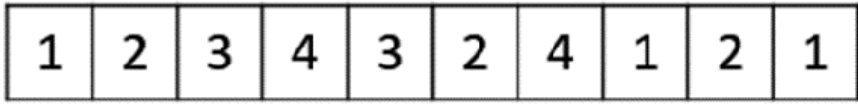
1) Binary encoding method



2) Real valued encoding method



3) Integer encoding method



4) Permutation encoding method

5) Gray encoding method

6) DNA encoding method

7) Messy encoding method

1) Binary encoding method

- The domain of x_j is $[a_j, b_j]$ and the required precision is five places after the decimal point.
- The **precision requirement** implies that the range of domain of each variable should be divided into at least $(b_j - a_j)$ 10^5 size ranges
- The **required bits** (denoted with m_j) for a variable is calculated as follows:

$$2^{m_j - 1} < (b_j - a_j) \times 10^5 \leq 2^{m_j} - 1$$

1) Binary encoding method

- The mapping from a **binary** string to a **real** number for variable x_j is completed as follows:

$$x_j = a_j + \text{decimal}(substring_j) \times \frac{b_j - a_j}{2^{m_j} - 1}$$

Binary String Encoding

- The precision requirement implies that the range of domain of each variable should be divided into at least (b_j - a_j) 10^5 size ranges.

- The required bits (denoted with m_j) for a variable is calculated as follows:

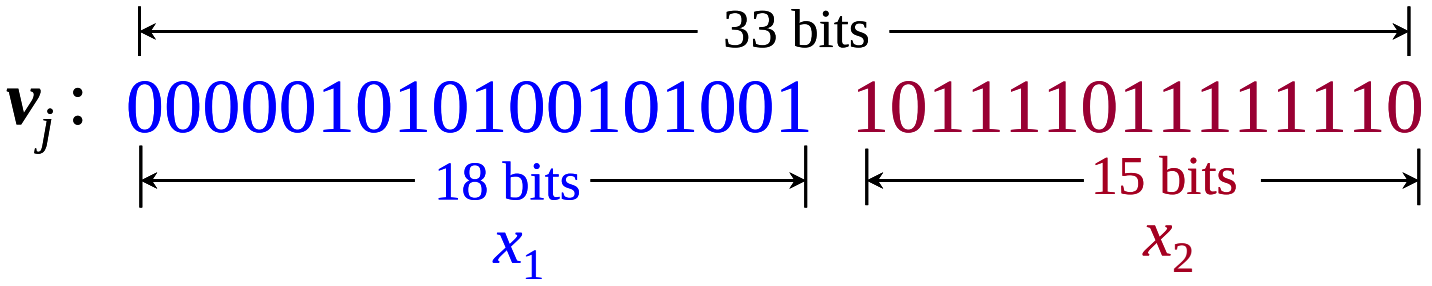
x_1 : (12.1-(-3.0)) 10,000 = 151,000

$2^{17} < 151{,}000 < 2^{18}$, m_1 = 18 bits

x_2 : (5.8-4.1) 10,000 = 17,000

$2^{14} < 17{,}000 < 2^{15}$, m_2 = 15 bits

precision requirement: $m = m_1 + m_2 = 18 + 15 = 33$ bits



Procedure of Binary String Encoding

input: domain of $x_j \in [a_j, b_j]$, (j=1,2)

output: chromosome v

step 1: The **domain** of x_j is $[a_j, b_j]$ and the required precision is five places after the decimal point.

step 2: The **precision requirement** implies that the range of domain of each variable should be divided into at least $(b_j - a_j)$ 10^5 size ranges.

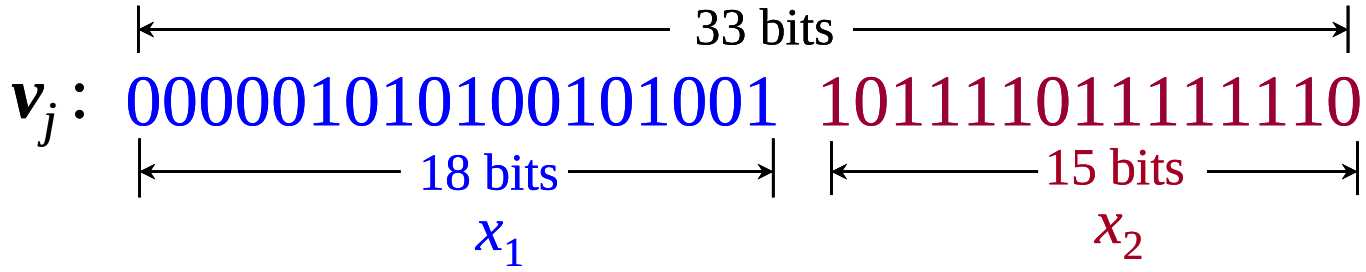
step 3: The **required bits** (denoted with m_j) for a variable is calculated as follows:

$$2^{m_j - 1} < (b_j - a_j) \times 10^5 \leq 2^{m_j} - 1$$

step 4: A chromosome v is randomly generated, which has the number of genes m, where m is sum of m_j (j=1,2).

Binary String Decoding

- The mapping from a binary string to a real number for variable x_j is completed as follows:



	Binary Number	Decimal Number
x_1	0000010101001010 01	5417
x_2	1011110111111 10	24318

$$x_j = a_j + \text{decimal}(substring_j) \times \frac{b_j - a_j}{2^{m_j} - 1}$$

$$x_1 = -3.0 + 5417 \times \frac{12.1-(-3.0)}{2^{18} - 1}$$

$$= -2.687069$$

$$x_2 = 4.1 + 24318 \times \frac{5.8 - 4.1}{2^{15} - 1}$$

$$= 5.361653$$

Procedure of Binary String Decoding

input: substring$_j$

output: a real number x_j

step 1: Convert a substring (a binary string) to a decimal number.

step 2: The mapping for variable x_j is completed as follows:

$$x_j = a_j + \text{decimal}(substring_j) \times \frac{b_j - a_j}{2^{m_j} - 1}$$

2) Real valued encoding method

- **Real valued** encoding is a method for representing data using real numbers instead of binary values.
- It is commonly used in **GA** and **neural networks** to represent data as numerical values that can be used as inputs to a neural network.
- The data is often represented as a **vector** of real numbers and is used to train neural networks.

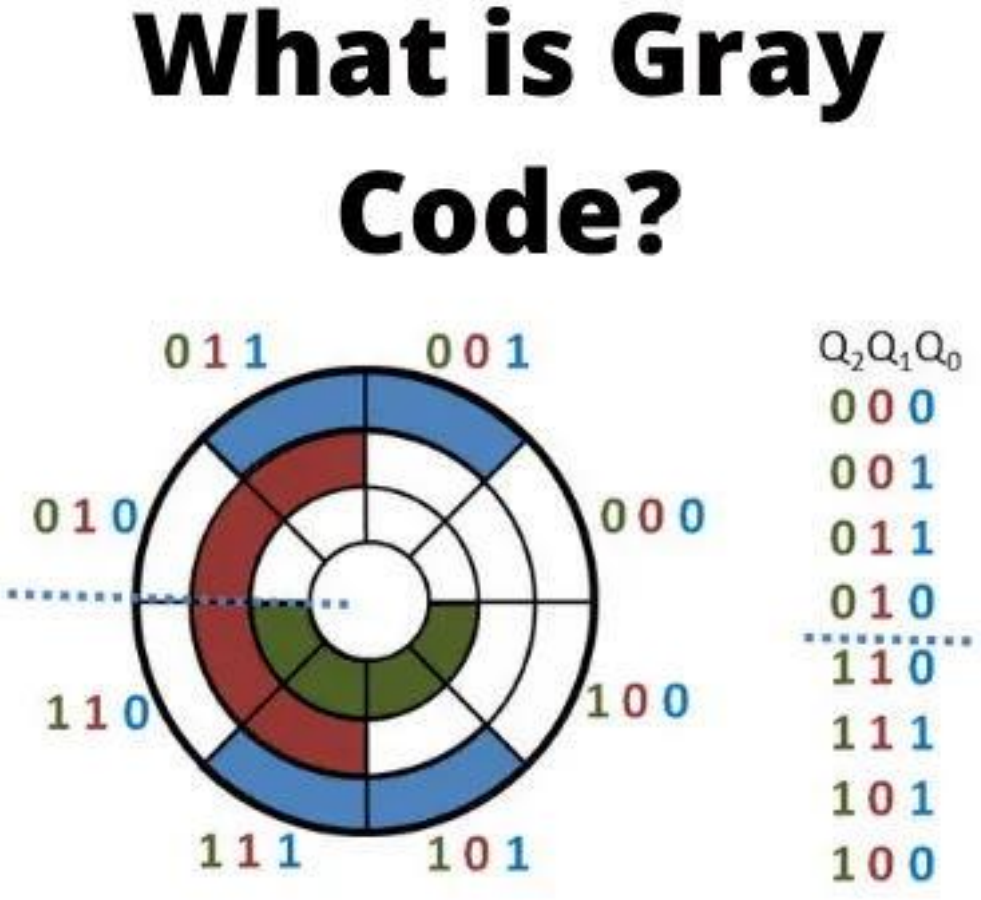
3) Integer encoding method

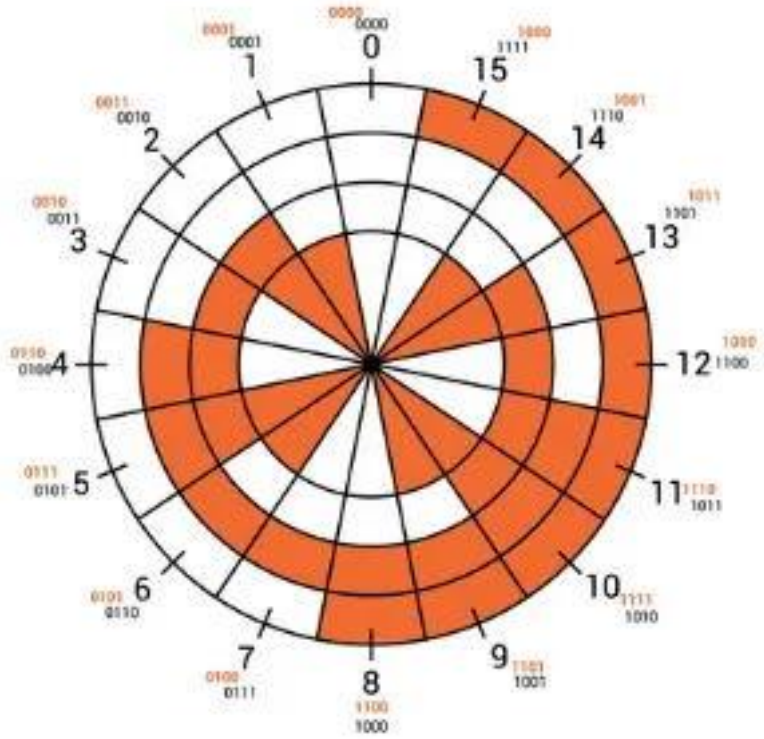
- Integer encoding is a type of encoding method used to represent categorical variables as **integer** numbers.
- This method helps to **reduce** the **size** of the dataset and improve the model performance.
- It assigns a **unique integer** value to each category in the dataset.
- This is particularly helpful for GA and **machine learning** algorithms that cannot process categorical data.

4) Permutation encoding method

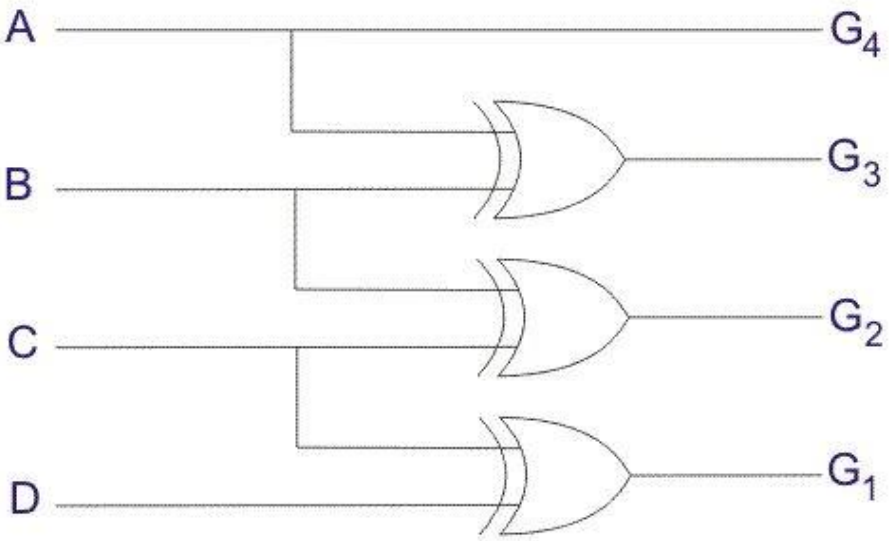
- Permutation encoding is a type of encoding technique used to secure data by **changing the order** of characters in the data.
- This technique is used to secure data from **unauthorized access** by applying a series of mathematical operations on the data.
- The order of the **characters** is changed **randomly** to **form** an **encrypted** text.
- This encrypted text can then be **decrypted** using the same mathematical operations.

5) Gray encoding method[38]





5) Gray encoding method[38]



Logic Circuit for Binary to Gray Code Converter

Decimal Number	4 bit Binary Number ABCD	4 bit Gray Code $G_1G_2G_3G_4$
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

6) DNA encoding method

- DNA encoding represents solutions as a **sequence** of symbols or values (A-T, C-G).

7) Messy encoding method

- Messy encoding is a type of encoding allows for more **flexibility** in representing solutions than traditional encoding methods like binary or real encoding.
- In messy encoding, each gene in the chromosome represents a "**messy**" variable that can take on one of several values.
- The possible values for each messy variable are **not** fixed, but are determined by a set of rules or constraints that define the valid values for that variable.

7) Messy encoding method

- For example, suppose we are using messy encoding to represent a set of objects, where each object has a colour and a shape.
- The colour of each object can be **red**, **green**, or **blue**, and the shape can be **square**, **circle**, or **triangle**.

7) Messy encoding method

- We might represent each object with a messy variable that has 9 possible values:
  - (R, S)
  - (R, C)
  - (R, T)
  - (G, S)
  - (G, C)
  - (G, T)
  - (B, S)
  - (B, C)
  - (B, T)

- The downside of using messy encoding is that it can be more **difficult** to design **fitness** functions that effectively evaluate the quality of solutions, since the possible values for each variable are not fixed.
- Therefore, messy encoding is typically used in **combination** with other techniques, such as fuzzy logic, to improve the performance of genetic algorithms.

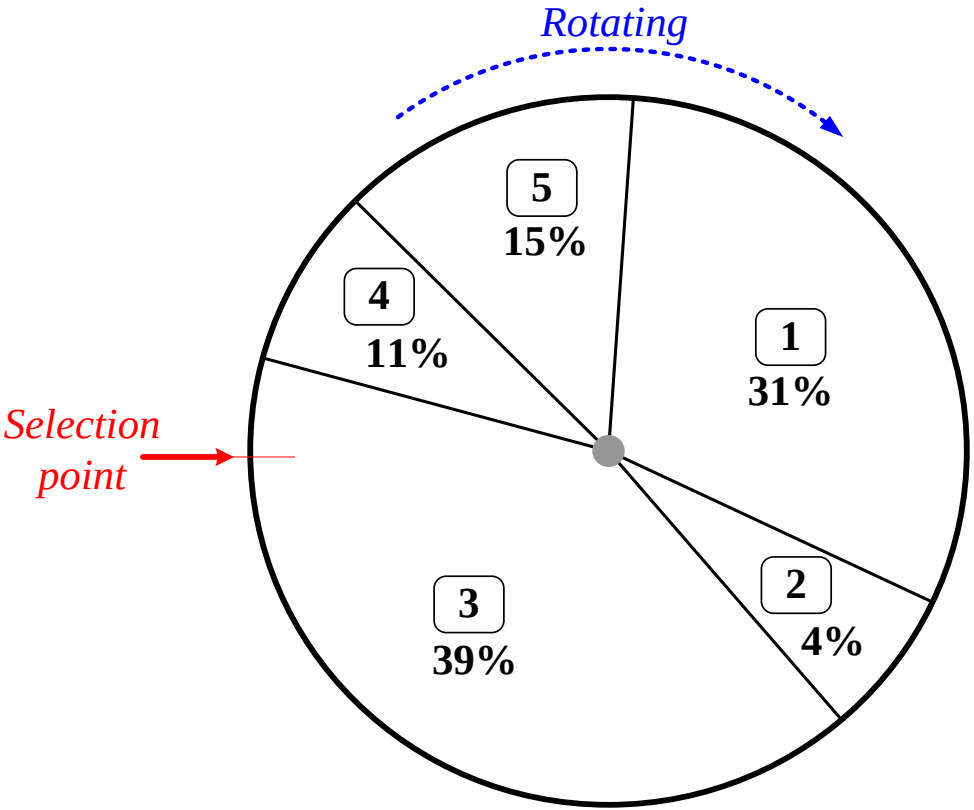
Hamming Distance

- The **Hamming distance (HD)** is the **number** of positions for which the corresponding genes are **different**.
- The Hamming distance between (0 1 1 1) and (1 0 0 0) is 4, which is the largest Hamming distance in the field of 4 binary code.
- try this:
  - (0 0 0 0) and (1 0 0 0) , then HD = 1
  - (0 0 0 1) and (1 0 0 0) , then HD = 2

10. Genetic Operators

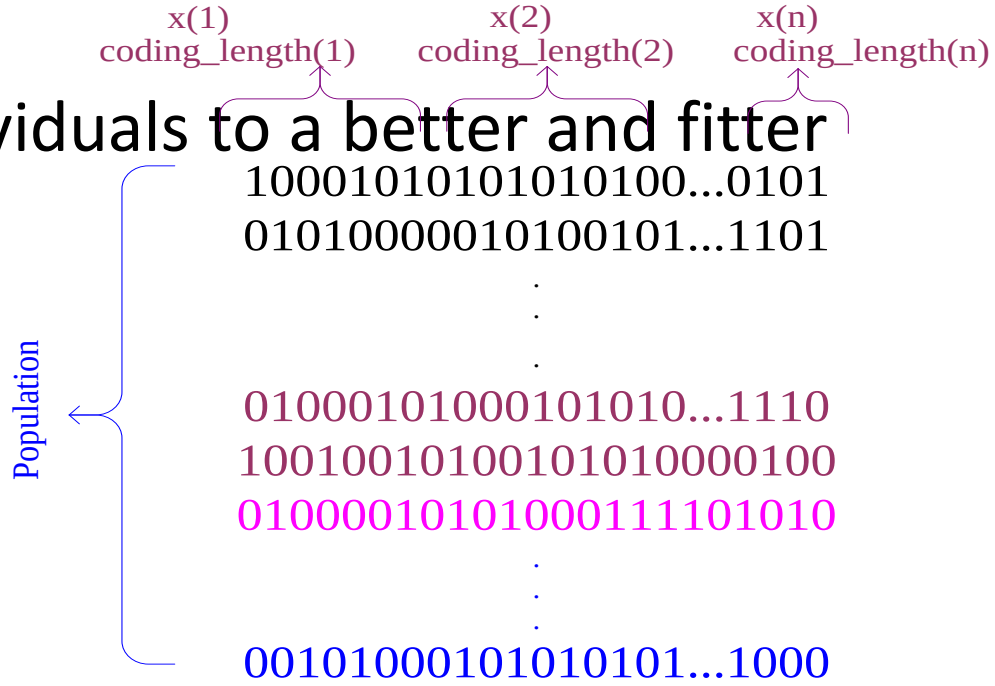
These alter the genetic composition of the offspring, including:

- **Selection**
- **Crossover**
- **Mutation**



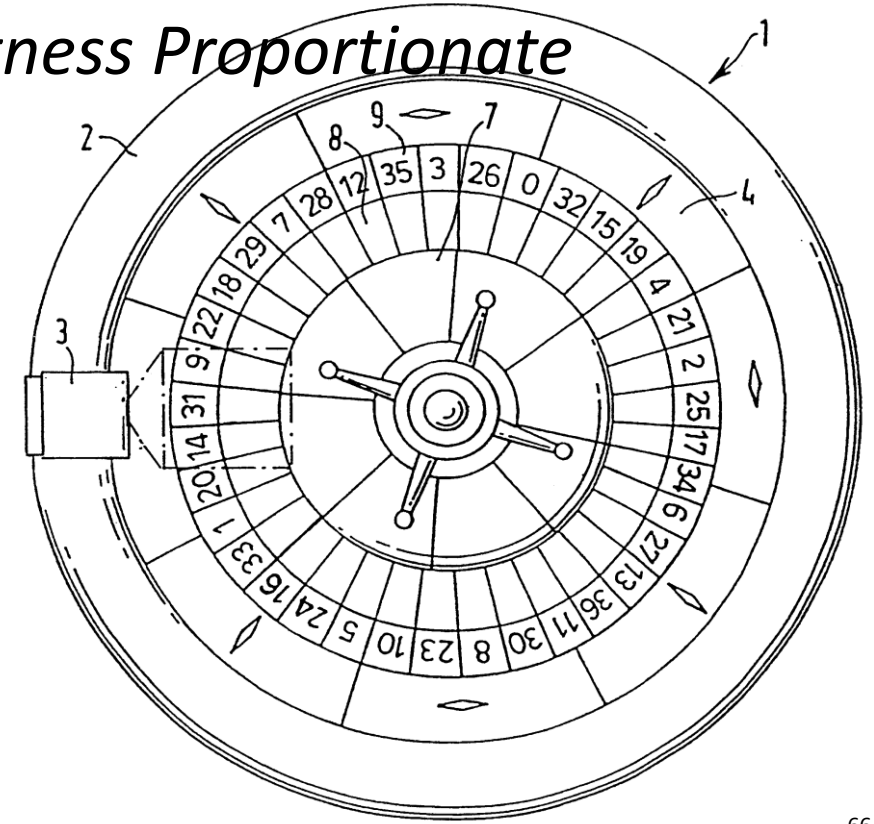
Selection

- Selection is the **process** of selecting parents which mate and recombine to create **off-spring** for the next generation.
- Good parents drive individuals to a better and fitter solutions.



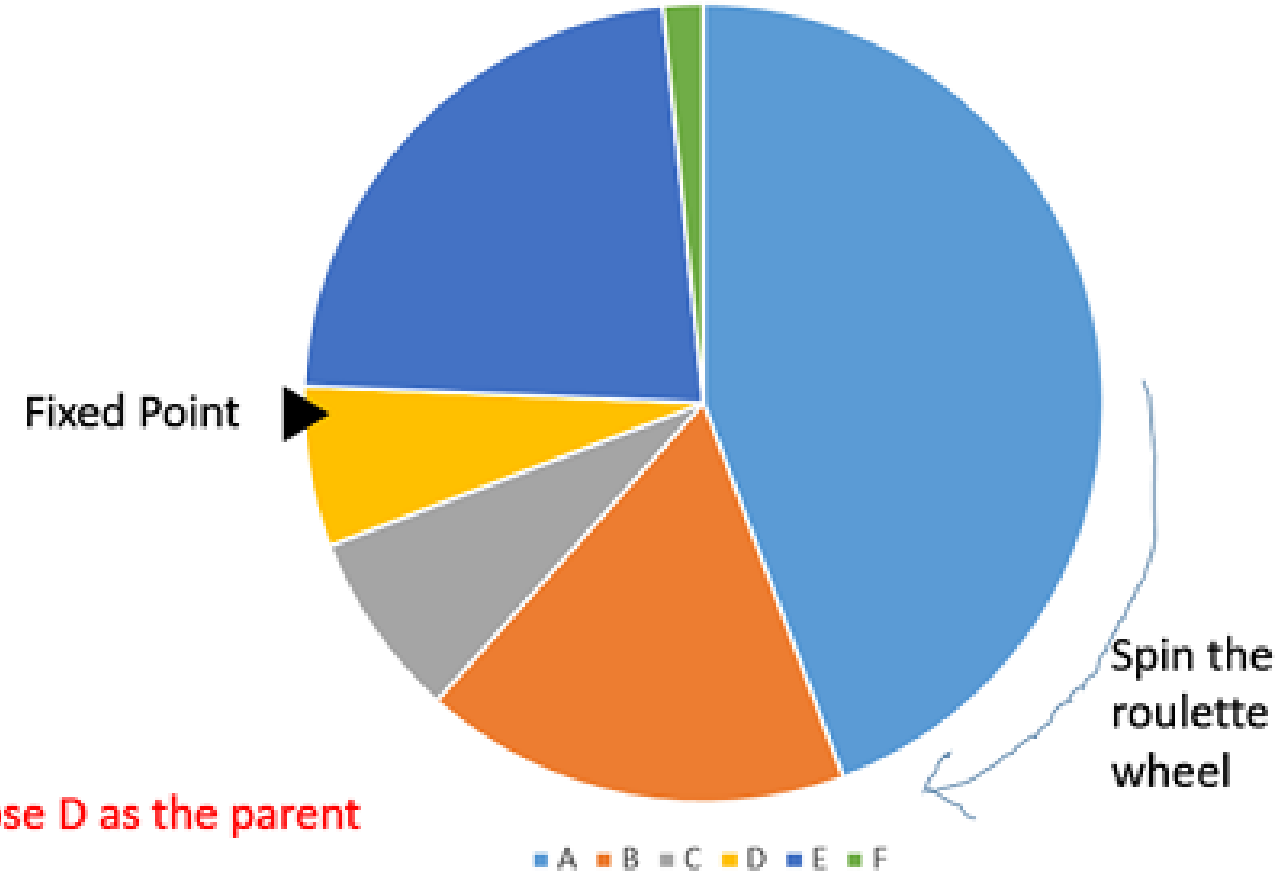
Selection

1) *Roulette Wheel Selection (Fitness Proportionate Selection)*
2) *Truncation Selection*
3) *Tournament Selection*
4) *Rank Selection*
5) *Random Selection*



1) Roulette Wheel Selection

Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3



Choose D as the parent

Fitness Value	Chromosome
1	Q
5	A
9	Z
8	W
7	S
4	X
2	E
3	F
6	R
2	T
2	Y
1	U
0	I

Select
K chromosomes
at random

A

E

T

Pick the best
as parent

A

Selection Probability

- **Fitness scaling** has a two-fold intention:
  - ✓ To maintain a reasonable differential between relative fitness ratings of chromosomes.
  - ✓ To prevent a too-rapid takeover by some supper chromosomes in order to meet the requirement to limit competition early on, but to stimulate it later.

Selection Probability

- Suppose that the **raw** fitness f_k (e.g. objective function value) for the k-th chromosomes, **the scaled fitness f_k'** is:

$$f_k' = g(f_k)$$

- Function $g(\cdot)$ may take **different form** to yield different **scaling methods**.

2) Truncation Selection

- It assigns a probability of **selection proportional** to an individual's **fitness** value, in that only the **top *n* individuals** with the **highest** fitness values are selected for reproduction, where n is a pre-determined parameter.
- The basic idea behind truncation selection is to **preserve** the **best solutions** found so far and **discard the rest**.
- This can be useful in situations where the goal is to converge quickly towards an optimal solution or to **maintain** a **diverse** population.

3) Tournament Selection

- The selection process begins by randomly selecting a subset of **k individuals** from the population, where k is a pre-determined parameter.
- These individuals then compete in a tournament, where the individual with the **highest fitness value** is selected as the **winner** and advances to the next round.
- This process is repeated **until** the desired number of individuals have been selected.

4) Rank Selection

- In rank selection, individuals are **sorted based on** their fitness values, and each individual is **assigned** a rank based on its position in the sorted list.
- The **best** individual is given a **rank of 1,** the second-best is given a rank of 2, and so on, until all individuals are ranked.
- The selection probabilities assigned by **rank selection** have the property that **better** individuals have a **higher** probability of being selected than **worse** individuals, but the probabilities are not proportional to fitness.

4) Rank Selection

- After ranking the individuals, a selection probability is assigned to each individual based on its rank.
- The selection probability for an individual with rank **i** is calculated as follows:
- **P(i) = (c - (c-1) * (i-1) / (N-1)) / S**
- where **c** is a constant, typically set to 2, **N** is the population size, and **S** is the sum of the probabilities for all individuals in the population.

5) Random Selection

- It is to **maintain genetic diversity** within the population and to ensure that individuals with lower fitness values still have a chance to be selected for the next generation.
- The process involves selecting individuals at random from the population, with the probability of **selection proportional** to their fitness value.
- Individuals with **higher fitness values** have a higher chance of being selected, but individuals with lower fitness values are not excluded entirely.

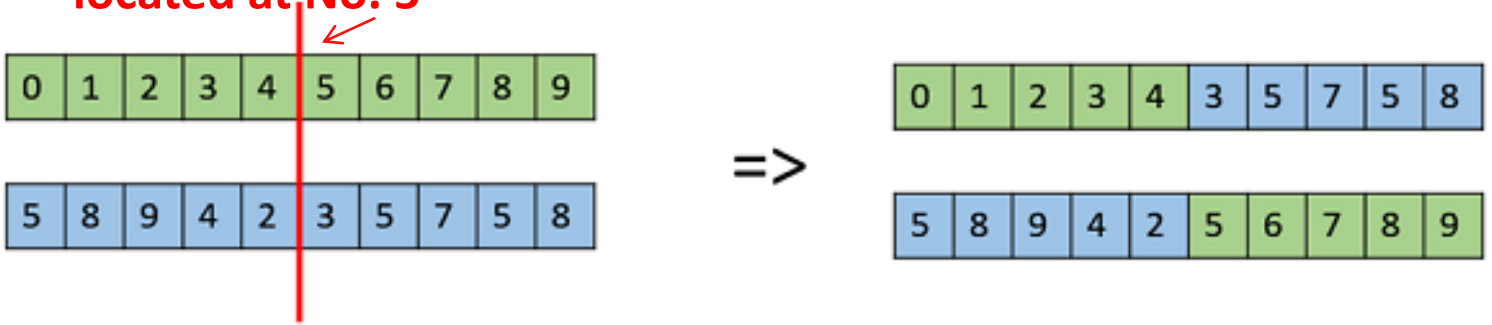
Crossover

- The crossover operator is **analogous** /əˈnaləgəs/ to reproduction and **biological crossover**.
- In this **more than one** parent is selected and one or more off-springs are produced using the genetic material of the parents.
- Crossover is usually applied in a GA with a high **probability** ($\boldsymbol{p_c}$).

Crossover

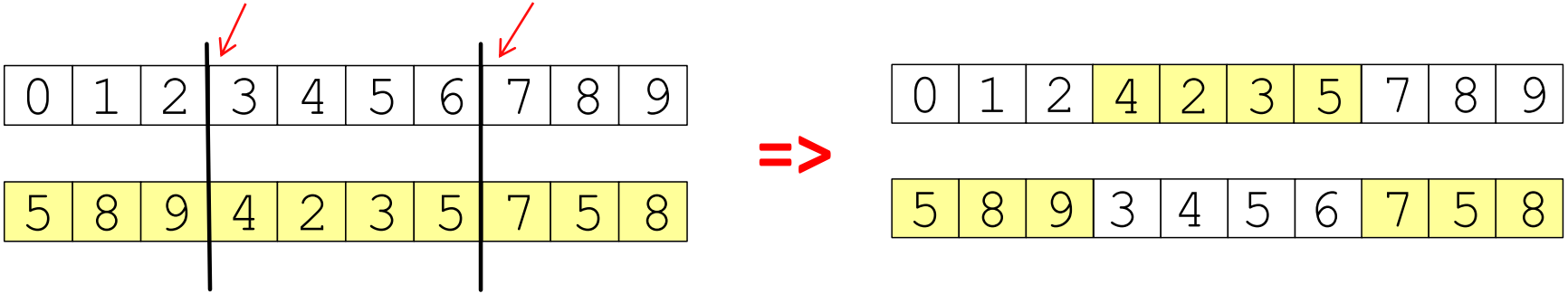
- **One Point Crossover**

located at No. 5



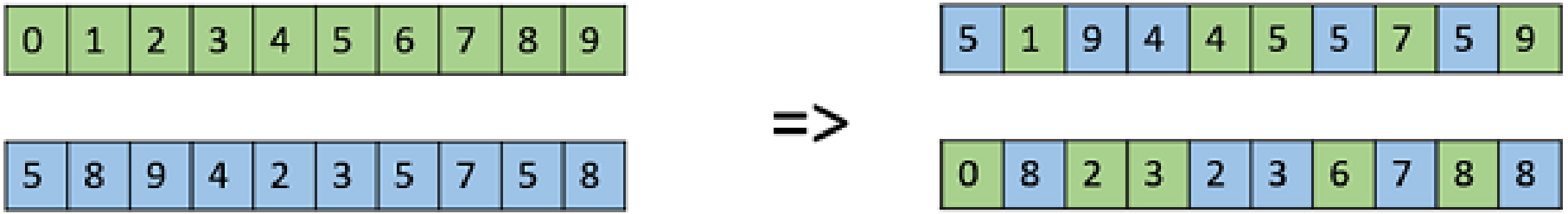
- **Multi Point Crossover**

located at No. 3 and No. 7

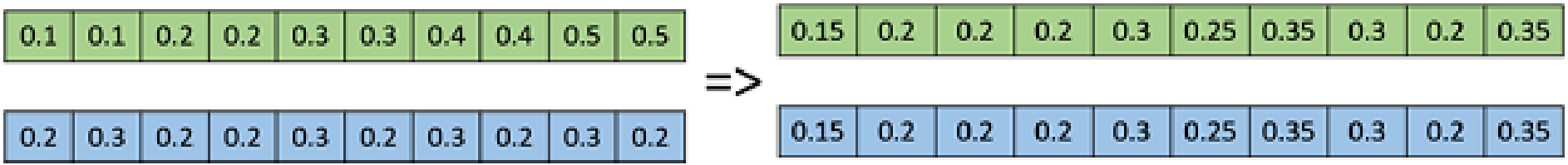


Crossover

- **Uniform Crossover**



- **Whole Arithmetic Recombination**



More crossover operators

- Davis’ Order Crossover (OX1)
- Partially Mapped Crossover (PMX)
- Order based crossover (OX2)
- Shuffle Crossover
- Ring Crossover

Mutation

- Mutation is a **small random** tweak in the chromosome, to get a new solution, which is used to maintain and introduce **diversity** in the genetic population and is usually applied with a low probability($\boldsymbol{p_m}$).
- **If** the probability is very high, the GA gets **reduced** to a random search.
- Mutation is the part of the GA which is related to the "**exploration**" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is **not**.

Mutation

- **Bit Flip Mutation**

select one or more random bits and flip them.

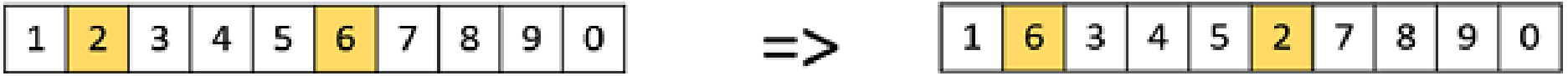
- **Random Resetting**

Random Resetting is an extension of the bit flip for the integer representation. In this, a **random value** from the set of permissible values is assigned to **a randomly chosen gene**.

Mutation

Swap Mutation

select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings



Scramble Mutation

Scramble mutation is popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled **randomly**.

Mutation

- **Inversion Mutation**

select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely **invert** the entire string in the subset.

11. Evaluation

Genotype /ˈjenəˌtīp /
Phenotype /ˈfēnəˌtīp /

- The process of evaluating the fitness of a chromosome consists of the following three steps:

input: chromosome v_k, k = 1, 2, ..., *popSize*
output: the fitness *eval*(v_k)

step 1: **Convert** the chromosome's **genotype to** its **phenotype**, *i.e.*, convert binary string into relative real values $x_k = (x_{k1}, x_{k2})$, k = 1,2, …, *popSize*.

11. Evaluation

step 2: **Evaluate** the **objective** function $f(\boldsymbol{x}_k)$, $k = 1,2, \ldots, popSize$.

step 3: **Convert** the value of objective function into

For the maximization problem, the fitness is simply equal to the value of objective function:

$eval(\boldsymbol{v}_k) = f(x_k)$, $k = 1,2, \ldots, popSize$.

11. Evaluation

$$eval(v_k) = f(x_i) \qquad (k = 1, 2, \cdots, popSize)$$

$$(i = 1, 2, \cdots, n)$$

$$f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x2 \cdot \sin(20\pi x_2)$$

Example: (x_1=-2.687969, x_2= 5.361653)

$$eval(\boldsymbol{v}_1) = f(-2.687969, 5.361653) = 19.805119$$

11. Evaluation

- An evaluation function plays **the role of the environment**, and it rates chromosomes in terms of **their fitness**.
- The fitness function values of above chromosomes are as follows on the next page
- It is clear that chromosome v_4 is the strongest one and that chromosome v_3 is the weakest one.

$eval(\boldsymbol{v}_1) = f\,(-2.687969, 5.361653) = 19.805119$

$eval(\boldsymbol{v}_2) = f\,(0.474101, 4.170144) = 17.370896$

$eval(\boldsymbol{v}_3) = f\,(10.419457, 4.661461) = 9.590546$

$eval(\boldsymbol{v}_4) = f\,(6.159951, 4.109598) = 29.406122$

$eval(\boldsymbol{v}_5) = f\,(-2.301286, 4.477282) = 15.686091$

$eval(\boldsymbol{v}_6) = f\,(11.788084, 4.174346) = 11.900541$

$eval(\boldsymbol{v}_7) = f\,(9.342067, 5.121702) = 17.958717$

$eval(\boldsymbol{v}_8) = f\,(-0.330256, 4.694977) = 19.763190$

$eval(\boldsymbol{v}_9) = f\,(11.671267, 4.873501) = 26.401669$

$eval(\boldsymbol{v}_{10}) = f\,(11.446273, 4.171908) = 10.252480$

12. Phenotype and Genotype

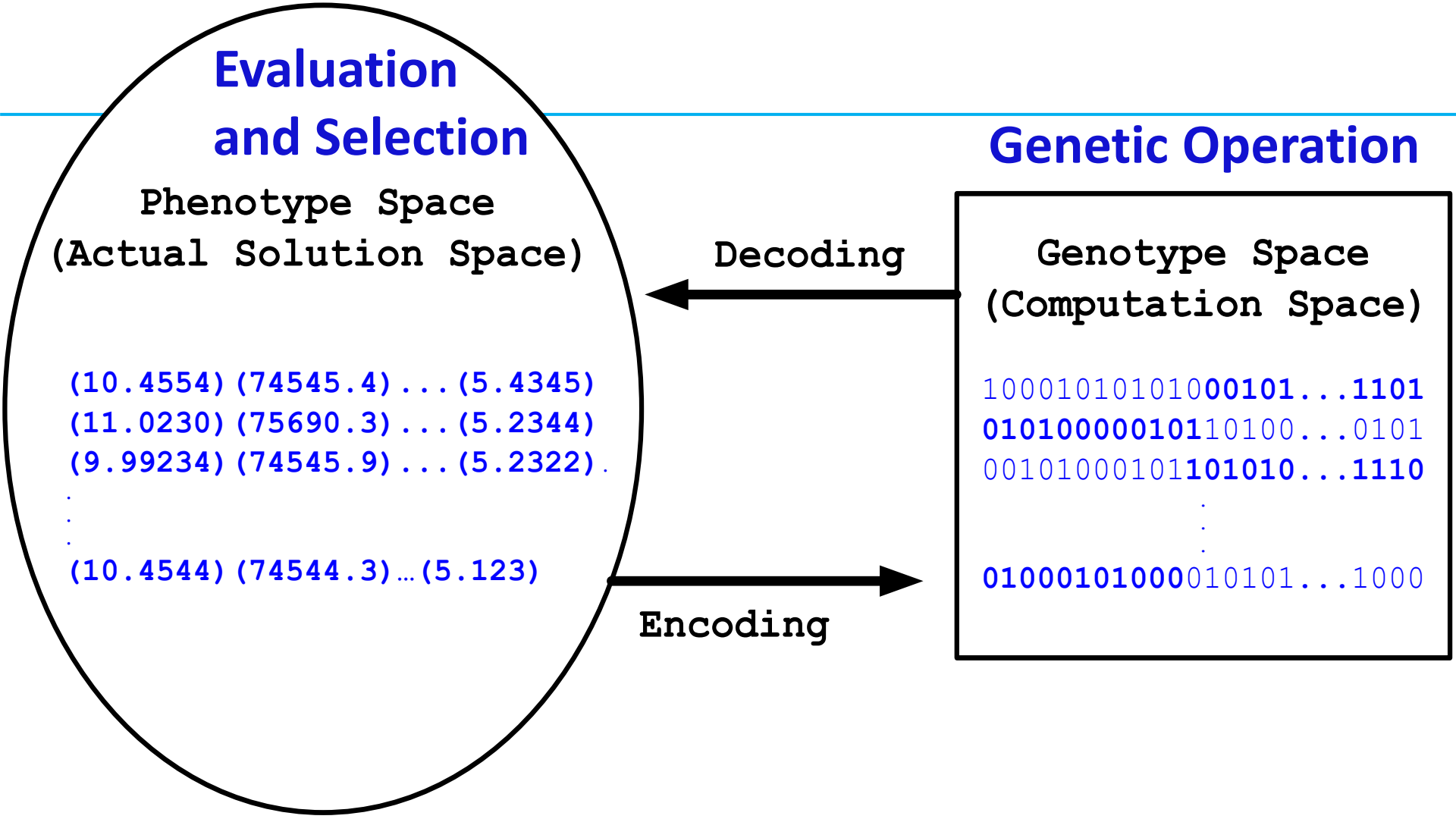
Genotype /ˈjenəˌtīp /
Phenotype /ˈfēnəˌtīp /

- **Genotype** – Genotype is the population in the **computation space**. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a **computing system**.
- 0000 0001 0010 0011
- **Phenotype** – Phenotype is the population in the actual **real world** solution space in which solutions are represented in a way they are represented in real world situations.
- 0 1 2 3

13. Decoding and Encoding

- **Decoding** is a process of transforming a solution **from** the *genotype* **to** the *phenotype* space; Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

- **Encoding** is a process of transforming **from** the phenotype **to** genotype space.

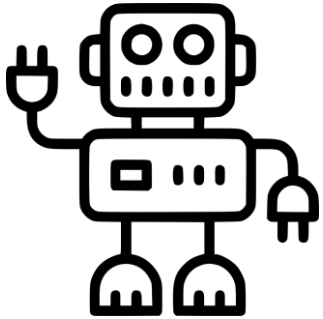

Evaluation
and Selection
Phenotype Space
(Actual Solution Space)
(10.4554) (74545.4) . . . (5.4345)
(11.0230) (75690.3) . . . (5.2344)
(9.99234) (74545.9) . . . (5.2322)
(10.4544) (74544.3) … (5.123)
Decoding
Encoding
Genetic Operation
Genotype Space
(Computation Space)
1000101010100101...1101
0101000010110100...0101
0010100010101010...1110
0100010100010101...1000

14. Termination Condition

- *When there has been **no improvement** in the population for X iterations f(x).*
- *When we reach an absolute **number of generations**.*
- *When the objective function value has reached a certain **pre-defined value***

Section Contents

- What is Genetic Algorithms
- History of Genetic Algorithms
- Basic Structure and Workflow
- Terminology
- ‘New’ Genetic Algorithms
- Genetic Algorithms Tools for MATLAB
- Case Studies

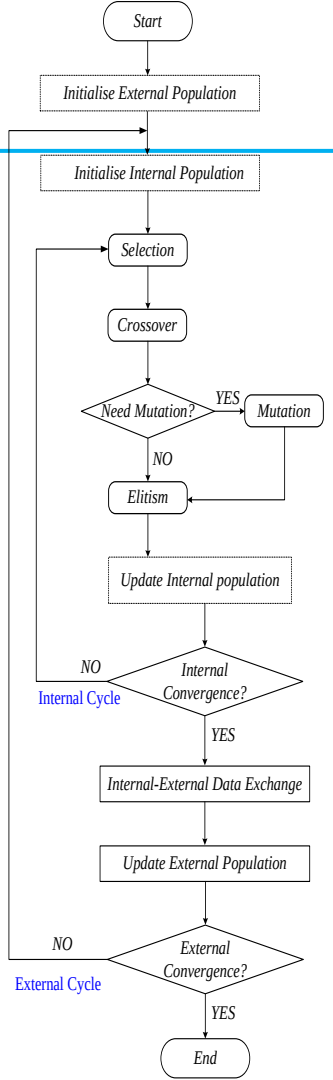


https://github.com/SGALAB/What-is-Genetic-Algorithms

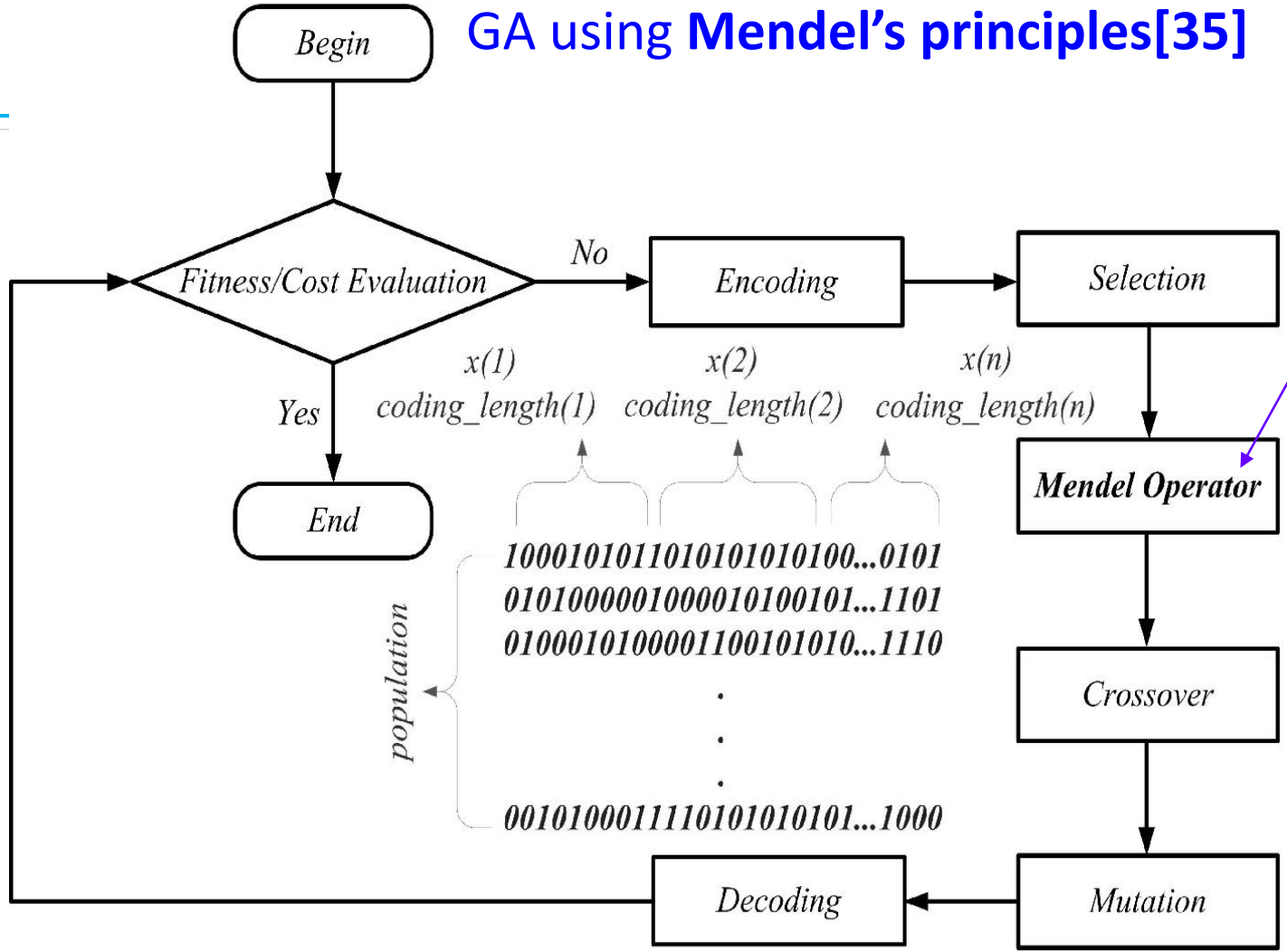
‘New’ Genetic Algorithms

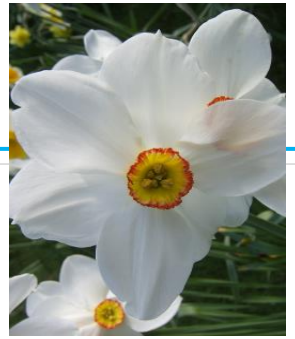
- MicroGA (MuGA)
- Mendel GA
- NPGA
- NSGA/ NSGAII

MicroGA (MuGA)[34]



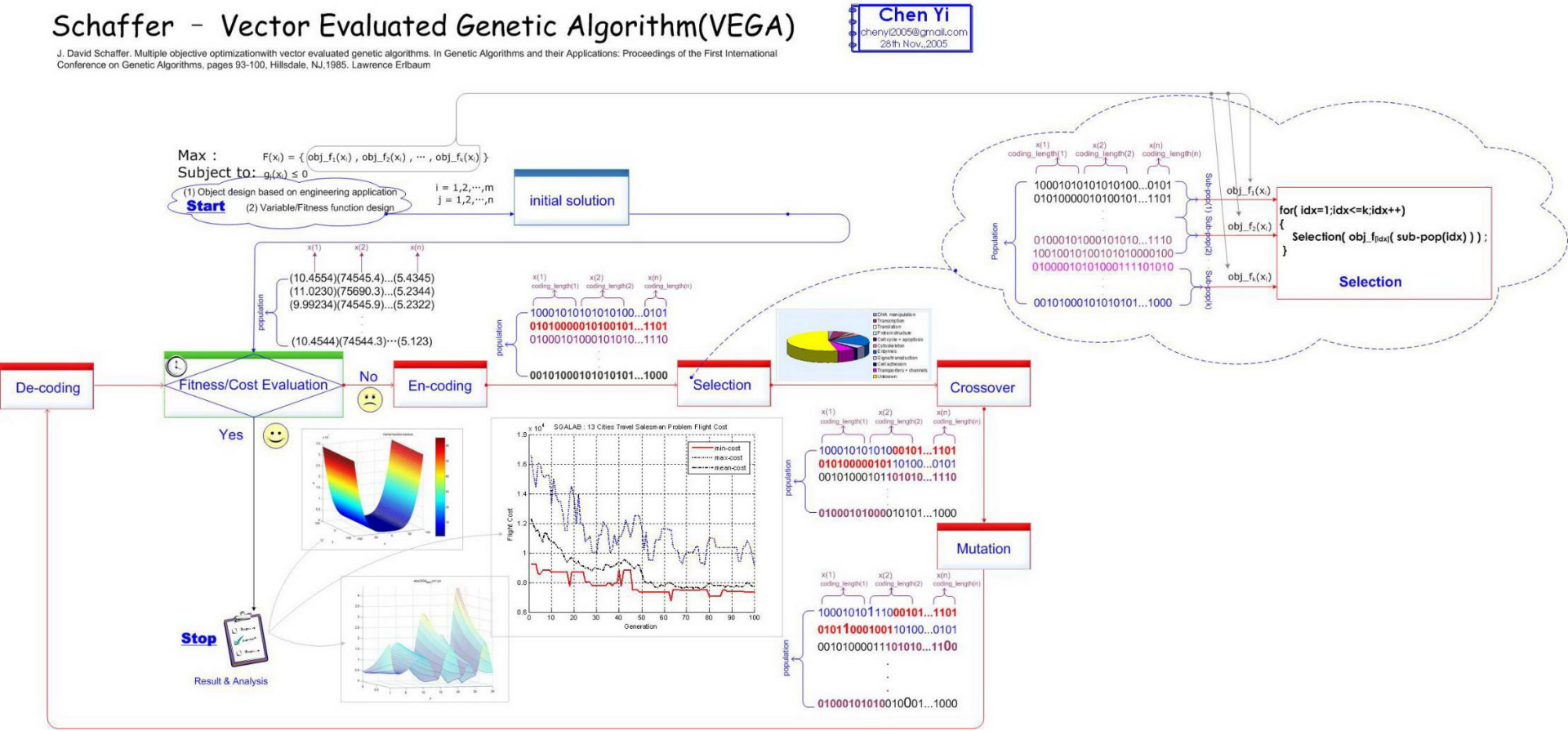
GA using **Mendel's principles[35]**




	D	***H***	***R***
D	D 100%	D 50% H 50%	H 100%
H	D 50% H 50%	D 25% H 50% R 25%	H 50% R 50%
R	H 100%	H 50% R 50%	R 100%

Vector Evaluated Genetic Algorithm [Schaffer (1984) PhD Thesis, Schaffer (1985)]



Multiobjective Genetic Algorithm (MOGA)

Chen Yi
chenyi2005@gmail.com
28th Nov.,2005

Fonseca, C. M. & P. J. Fleming: "Genetic algorithms for multiobjective optimization: formulation, discussion and generalization", *Proc. of the 5th Inter. Conf. on GAs*, pp. 416-423, 1993

Multiobjective Genetic Algorithm (MOGA:Fonseca & Fleming 1993)

1. Fitness **Pareto ranking** -- for each chromosome *ci* calculate the number of chromosomes that dominate it (*ni*). Let rank *ri*=1+*ni* ;
2. **Fitness assignment** -- Assign fitness to *ci* using linear interpolation so that *ci* with lowest rank has maximum fitness;
3. Fitness **Average** fitnesses of chromosomes with same rank and assign as the new fitness;
4. Fitness **Sharing** -- Use niching methods to distribute chromosomes over the Pareto-optimal region.

The rest of the algorithm is a standard GA.

Max : $F(x_i) = \{ obj\_f_1(x_i) , obj\_f_2(x_i) , \cdots , obj\_f_k(x_i) \}$

Subject to: $g_i(x_i) \le 0$

(1) Object design based on engineering application

Start (2) Variable/Fitness function design

$i = 1,2,\cdots,m$

$j = 1,2,\cdots,n$

initial solution

x(1) x(2) x(n)

population

(10.4554)(74545.4)...(5.4345)
(11.0230)(75690.3)...(5.2344)
(9.99234)(74545.9)...(5.2322)
(10.4544)(74544.3)⋯(5.123)

x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)

population

1000101010101010100...0101
0101000001010010 1...1101
0100010100010101 0...1110
00101000101010101...1000

De-coding

Fitness/Cost Evaluation

No

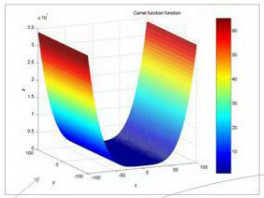
En-coding

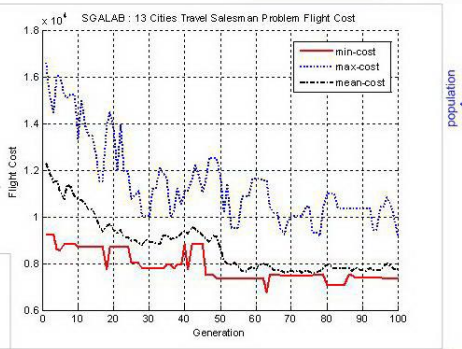
MOGA Block

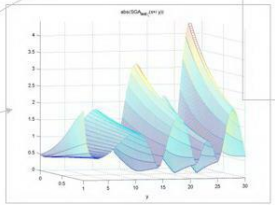
Selection

Crossover

Yes







Stop

Result & Analysis

x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)

population

1000101010100 0101...1101
0101000001011 0100...0101
0010100010110 1010...1110
0100010100010 101...1000

Mutation

x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)

population

1000101011110 0101...1101
0101100010011 0100...0101
0010100001110 1010...1100
0100010101001 0001...1000

Niched Pareto Genetic Algorithm (NPGA)

Horn, J. and N. Nafpliotis (1993). Multiobjective optimization using the niched pareto genetic algorithm. IlliGAL Report 93005, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, Champaign.

Chen Yi
chenyi2005@gmail.com
28th Nov.,2005

Max : $F(x_i) = \{ obj\_f_1(x_i), obj\_f_2(x_i), \cdots, obj\_f_k(x_i) \}$

Subject to: $g_j(x_i) \le 0$

(1) Object design based on engineering application

Start (2) Variable/Fitness function design

$i = 1,2,\cdots,m$

$j = 1,2,\cdots,n$

initial solution

population

x(1) x(2) x(n)

(10.4554)(74545.4)...(5.4345)
(11.0230)(75690.3)...(5.2344)
(9.99234)(74545.9)...(5.2322)
(10.4544)(74544.3)···(5.123)

Niche count $m_i = \sum_{j=1}^{population} Sh[d(i,j)]$

i

j = 1 ~ population

De-coding

Fitness/Cost Evaluation

No

En-coding

Yes

x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)

1000101010101010...0101
0101000001010010 1...1101
0100010100010101 0...1110
0010100010101010 1...1000

This technique might also be called **flat** fitness sharing, since the same effect is produced in normal (single-attribute) fitness sharing when two individuals have the exact **same fitness**. When fi = fj (i.e. the fitness function is flat for individuals i and j), then their shared fitnesses, fi/mi and fj/mj respectively, are ordered solely by their niche counts mi and mj. This situation would arise on a plateau of constant Fitness, where the radius of the plateau was much larger than sigma-share. Although we found no studies that analyzed or reported the behaviour of fitness sharing in such regions, we speculate that the population would spread out across the plateau. We should expect to find individuals sigma-share units apart from one Another, assuming a sufficiently large population to cover the plateau, and assuming crossover and mutation do not remove excessive numbers of individuals from the plateau each generation.

Stop

Result & Analysis

Selection (tournament)

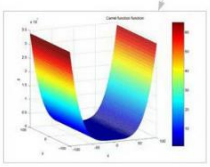
Flat fitness sharing

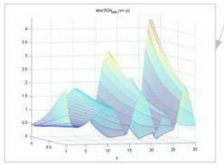
Crossover

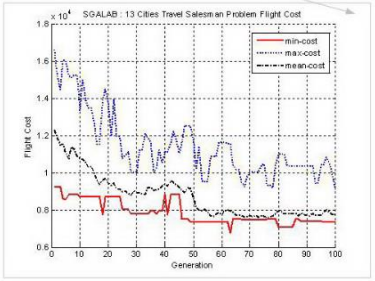
1000101010100010 1...1101
0101000001011010 0...0101
0010100010110101 0...1110
0100010100001010 1...1000

Mutation

1000101011100010 1...1101
0101100010011010 0...0101
0010100001101010 1...1100
0100010101001000 1...1000







Non-dominated Sorting Genetic Algorithm(NSGA)

Srinivas, N. and K. Deb (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation* 2(3),221–248.

Chen Yi
chenyi2005@gmail.com
28th Nov.,2005

Max : $F(x_i) = \{ obj\_f_1(x_i) , obj\_f_2(x_i) , \cdots , obj\_f_k(x_i) \}$

Subject to: $g_j(x_i) \le 0$

(1) Object design based on engineering application

Start (2) Variable/Fitness function design

$i = 1,2,\cdots,m$
$j = 1,2,\cdots,n$

initial solution

population
x(1) x(2) x(n)
(10.4554)(74545.4)...(5.4345)
(11.0230)(75690.3)...(5.2344)
(9.99234)(74545.9)...(5.2322)
(10.4544)(74544.3)···(5.123)

Goldberg' s method
Input : chromosomes x(i), I = 1,2,···,population;
Output : rank of fitness

Step 1 : rank 1 to the non-dominated chromosomes;
Step 2 : removing ranked chromosomes from input chromosomes;
Step 3: go on next round, rank 2 will be assigned to nondominated fitness among rest chromosomes;
Step 4: until all the fitness are ranked

MOGA method(MOGA:Fonseca and Fleming, 1993)
Input : chromosomes x(i), I = 1,2,···,population;
Output : rank of fitness

step 1: rank 1 is given to the nondominated chromosomes ;
step 2: removing them from contention;
step 3: finding the next non-dominated chromosomes, removing them from contention, rank equally to the number of its dominating individuals plus one ;
step 4: process continues until the entire population is ranked.

De-coding → Fitness/Cost Evaluation → No → En-coding → Pareto-ranking

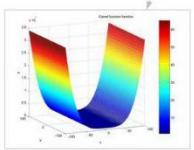
Yes → Stop → Result & Analysis

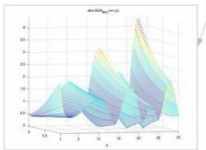
population
x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)
1000101010101010...0101
0101000001010101...1101
0100010100010101...1110
00101000101010101...1000

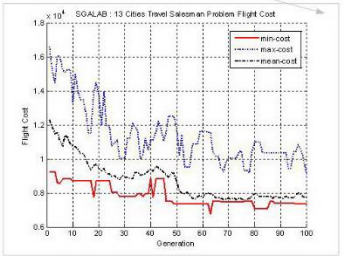
Non-dominated Sorting GA (NSGA: Deb, 1995)
Input : chromosomes x(i), I = 1,2,···,population;
Output : fitness

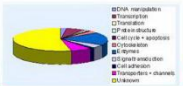
step 1: The non-dominated individuals are first identified and assigned a large/small **dummy** fitness value;
step 2: To maintain diversity in the population, these individuals are shared with their dummy fitness values;
step 3: After **sharing**, these non-dominated individuals are ignored temporarily;
step 4: The second non-dominated **front** in the rest of the population is identified and assigned a dummy fitness value that is kept smaller than the minimum shared dummy fitness of the previous front;
step 5: This process is continued until the entire population is classified into several fronts.

NSGA Block → Selection (stochastic remainder proportionate selection) → Crossover → Mutation





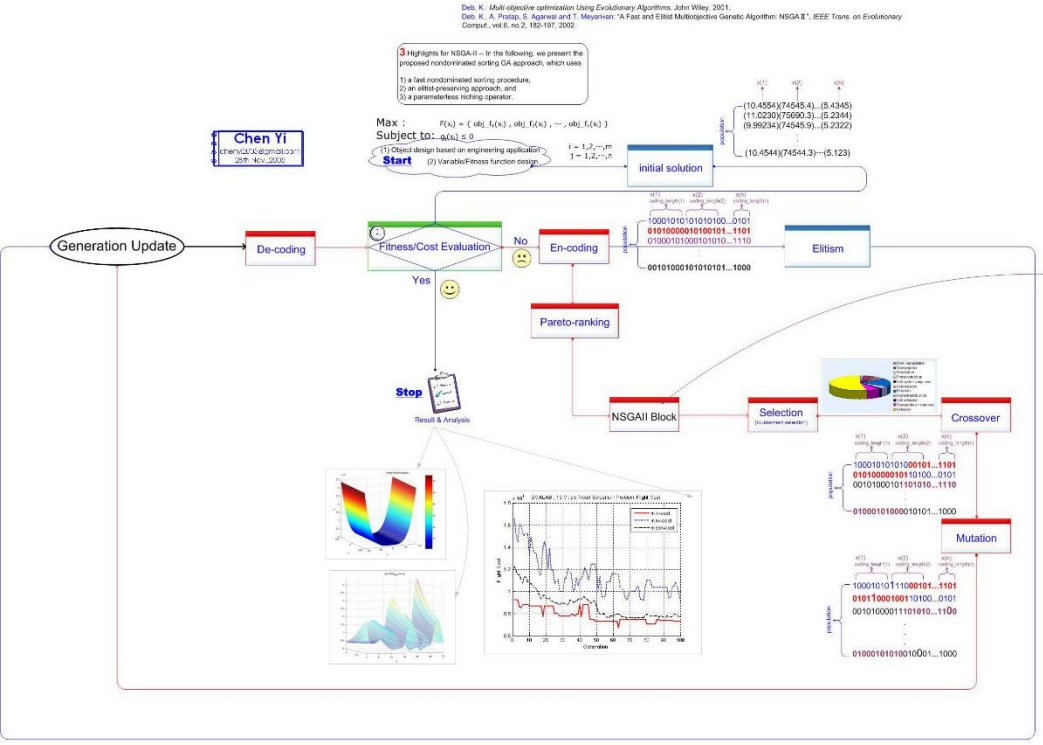




population
x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)
1000101010100010...1101
0101000001010100...0101
0010100010110101...1110
0100010100010101...1000

population
x(1) coding_length(1) x(2) coding_length(2) x(n) coding_length(n)
1000101011100010...1101
0101100010010100...0101
0010100001110101...1100
0100010101001000...1000

Non-dominated Sorting Genetic Algorithm-II(NSGA-II)




```
Fast non-dominated sort ()
{
Step 1) pareto rank
Step 2) get fronts
Step 3) sort population into different groups according to fronts
Step 4) crowding distance assignment
}
```

NSGAII goes as following steps:

Step 1: Fast non-dominated sort

Step 2: crowding distance assignment

Step 3 :crowded-comparison operator

```
Crowding_distance_assignment(l)
/* in current front , there are m fitness functions , n = 1,2,…,m*/
{
(1) Set initialization distance to 0 ,
     Dist(2:n-1) = 0,
     Dist(1)     = Dist(n) = inf

(2) MAX & MIN sort objective value ,namely,
    MAX & MIN sort( fitness )

(3) get each distance by following function:

for  idx = 2 : (n-1)
:
Dist(idx) = Dist(idx)_last + (fitness(idx+1)_idx − fitness(idx−1)_idx) / (fitness^MAX_idx − fitness^MIN_idx)
:
}
```

```
crowded-comparison operator( ranks, crowded_distances )
{                  i < j
  if ( rank( i ) < rank( j ) )
     or
     ( rank( i ) == rank( j ) )&& distance( i ) > distance( j )

        get j;
  else

        get i;
}
```

2) Crowded-Comparison Operator: The crowded-comparison operator ($\prec_n$) guides the selection process at the various stages of the algorithm toward a uniformly spread-out Pareto-optimal front. Assume that every individual i in the population has two attributes:

1) nondomination rank (i_{rank});
2) crowding distance (i_{distance}).

We now define a partial order $\prec_n$ as

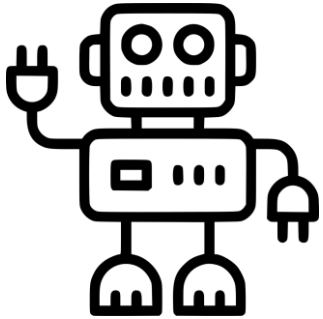
$$i \prec_n j \quad \text{if } (i_{\text{rank}} < j_{\text{rank}}) \text{ or } ((i_{\text{rank}} = j_{\text{rank}}) \text{ and } (i_{\text{distance}} > j_{\text{distance}})).$$

That is, between two solutions with differing nondomination ranks, we prefer the solution with the lower (better) rank. Otherwise, if both solutions belong to the same front, then we prefer the solution that is located in a lesser crowded region.

With these three new innovations—a fast nondominated sorting procedure, a fast crowded distance estimation procedure, and a simple crowded comparison operator, we are now ready to describe the NSGA-II algorithm.

Section Contents

- What is Genetic Algorithms
- History of Genetic Algorithms
- Basic Structure and Workflow
- Terminology
- ‘New’ Genetic Algorithms
- Genetic Algorithms Tools for MATLAB
- Case Studies

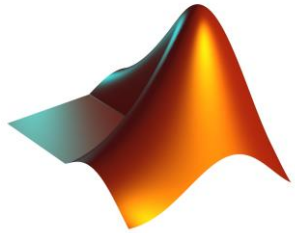


https://github.com/SGALAB/What-is-Genetic-Algorithms

Genetic Algorithms Tools for MATLAB

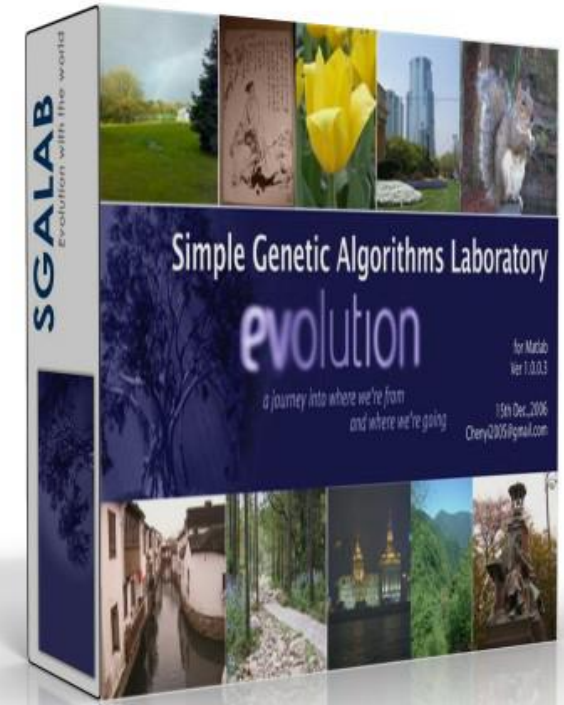
- Global Optimization Toolbox (Mathworks)
- Simple Genetic Algorithms Laboratory (SGALAB)
- Genetic and Evolutionary Algorithm Toolbox (GEATbx)
- Genetic Algorithm Toolbox for use with MATLAB (GATBX)
- Genetic Algorithm Optimization Toolbox (GAOT)

Global Optimization Toolbox (by Mathworks)[28]



- **Genetic Algorithm**
- Particle Swarm
- Simulated Annealing
- Surrogate Optimization
- Pattern Search
- Multiobjective Optimization

Simple Genetic Algorithms Laboratory (SGALAB)[29]



SGALAB

Evolution with the world

SxLAB Family Member

- Single and Multiobjective Problems
- Multidisciplinary Applications:
  - engineering design
  - intelligent control
  - material design
  - spatial-tempo analysis
  - econometrics analysis

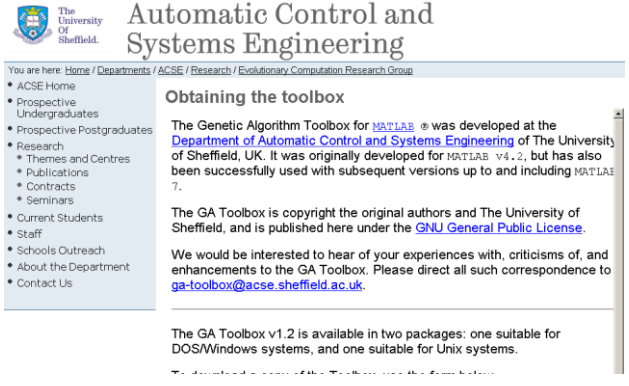
Genetic and Evolutionary Algorithm Toolbox (GEATbx) [30]

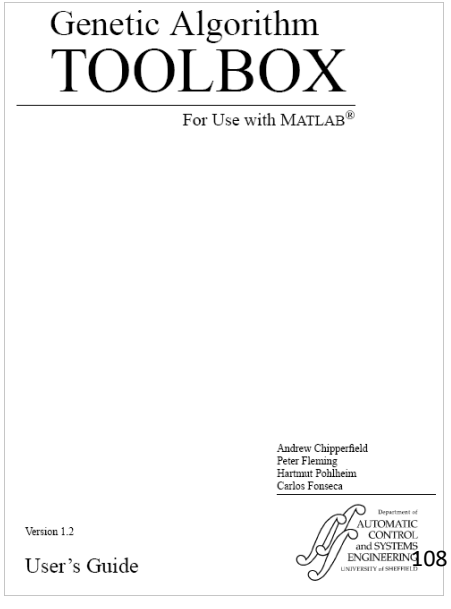
- high level functions to all operators
- multi-objective optimization
- real, integer and binary (linear and logarithmic scaling, gray coding) variable representation
- can be completely compiled into C/C++ Code using the Matlab Compiler.

GEATbx.com

Genetic Algorithm Toolbox for use with MATLAB (GATBX)[31]

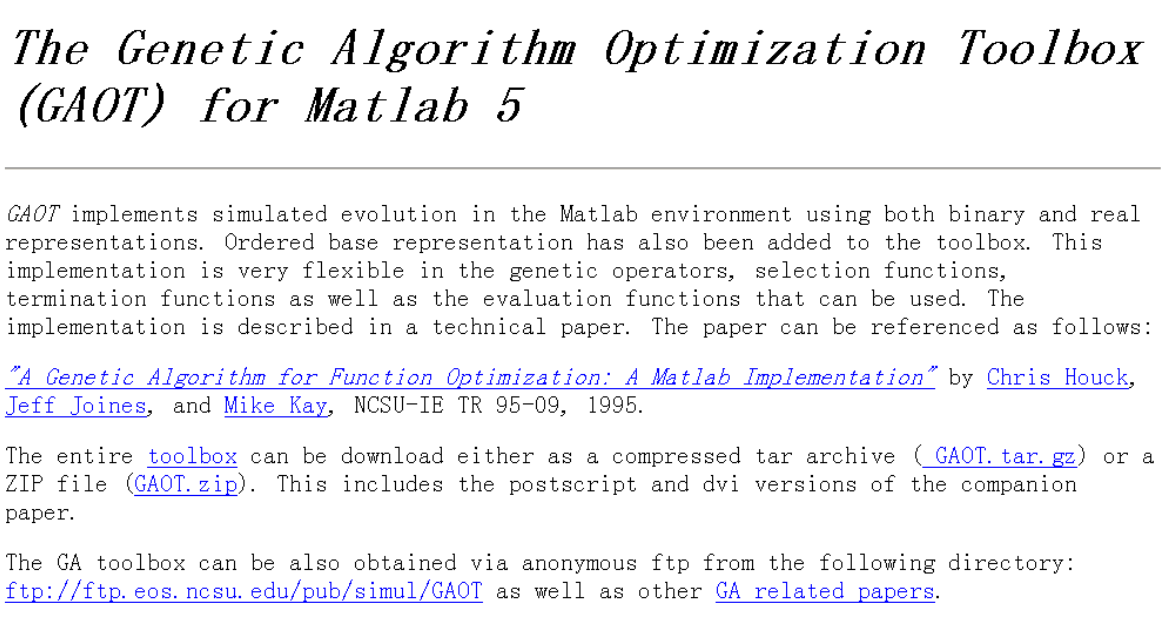
- Support for binary, integer and real-valued representations.
- A wide range of genetic operators.
- High-level entry points to most low-level functions.
- Many variations on the standard GA.
- Support for virtual multiple subpopulations.





Genetic Algorithm Optimization Toolbox (GAOT)[32]

- binary and real representations
- genetic operators: selection functions, termination functions, evaluation functions



The Genetic Algorithm Optimization Toolbox (GAOT) for Matlab 5

GAOT implements simulated evolution in the Matlab environment using both binary and real representations. Ordered base representation has also been added to the toolbox. This implementation is very flexible in the genetic operators, selection functions, termination functions as well as the evaluation functions that can be used. The implementation is described in a technical paper. The paper can be referenced as follows:

"A Genetic Algorithm for Function Optimization: A Matlab Implementation" by Chris Houck, Jeff Joines, and Mike Kay, NCSU-IE TR 95-09, 1995.

The entire toolbox can be download either as a compressed tar archive (GAOT.tar.gz) or a ZIP file (GAOT.zip). This includes the postscript and dvi versions of the companion paper.

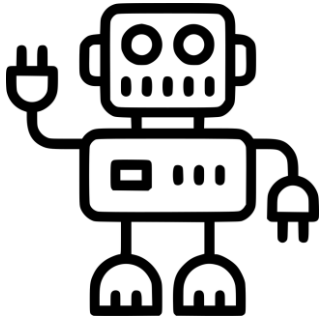
The GA toolbox can be also obtained via anonymous ftp from the following directory: ftp://ftp.eos.ncsu.edu/pub/simul/GAOT as well as other GA related papers.

More GA tools for MATLAB

- [33]

Section Contents

- What is Genetic Algorithms
- History of Genetic Algorithms
- Basic Structure and Workflow
- Terminology
- ‘New’ Genetic Algorithms
- Genetic Algorithms Tools for MATLAB
- Case Studies

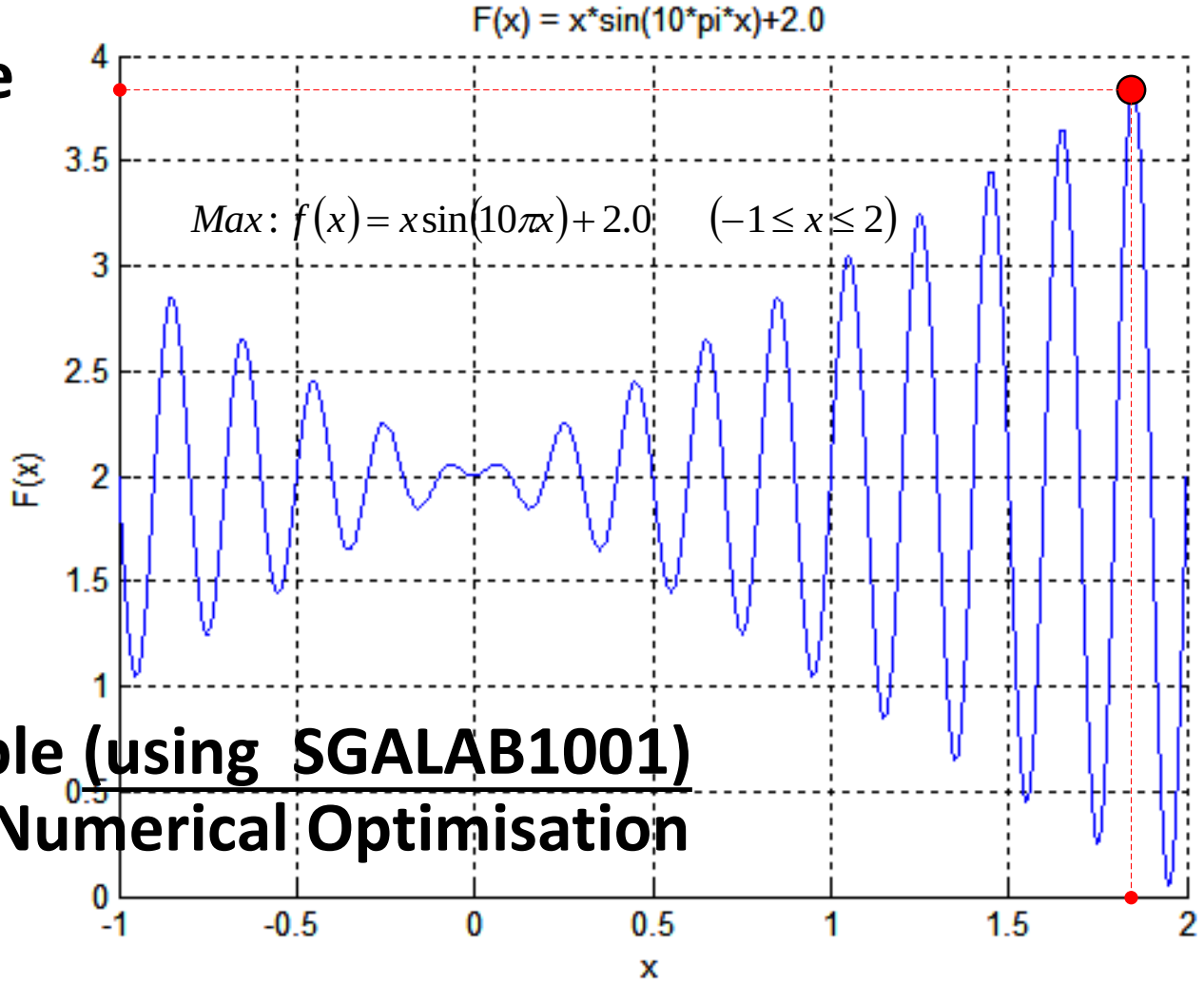


https://github.com/SGALAB/What-is-Genetic-Algorithms

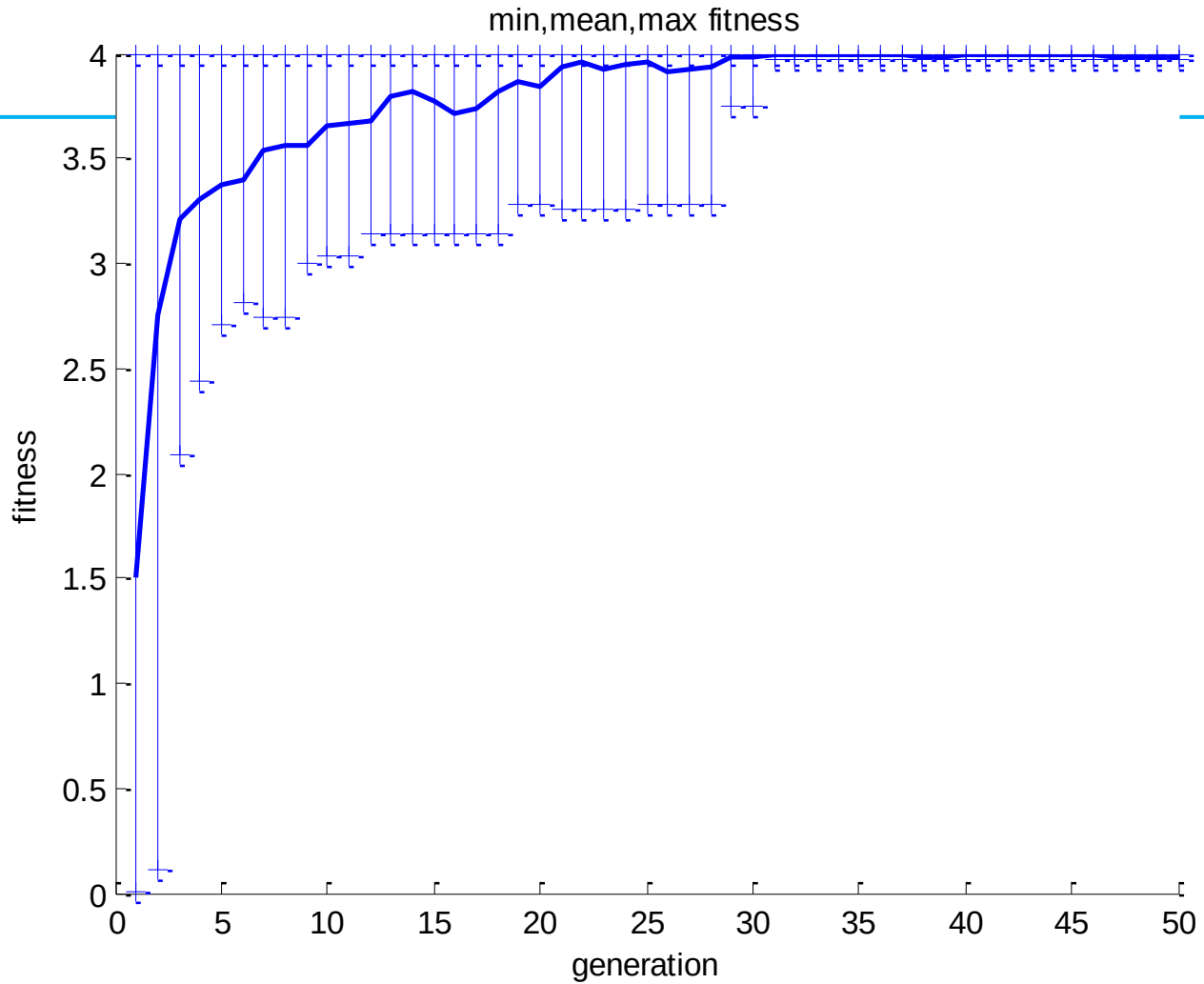
Case Studies

- 01 Simple Case: unconstrained optimisation problem
- 02 TSP 13 Cities
- 03 Multi-objective Problems

01 Simple Case



A Simple Example (using SGALAB1001)
Unconstrained Numerical Optimisation


min,mean,max fitness
fitness
generation
0
0.5
1
1.5
2
2.5
3
3.5
4
0
5
10
15
20
25
30
35
40
45
50

Parameters	Value	
Max Generation	100	300
Crossover Probability	0.8	0.9
Mutation Probability	0.01	0.01
Population	30	100
step	0.01	0.001
$F(x_0)\|_{max}$	3.843702	3.849781
$F(x_0)\|_{min}$	0.060569	0.071226
$F(x_0)\|_{mean}$	3.635146	3.830261
x_0	1.853229	1.851282
Cost time (sec.)	1.2350	15.1090

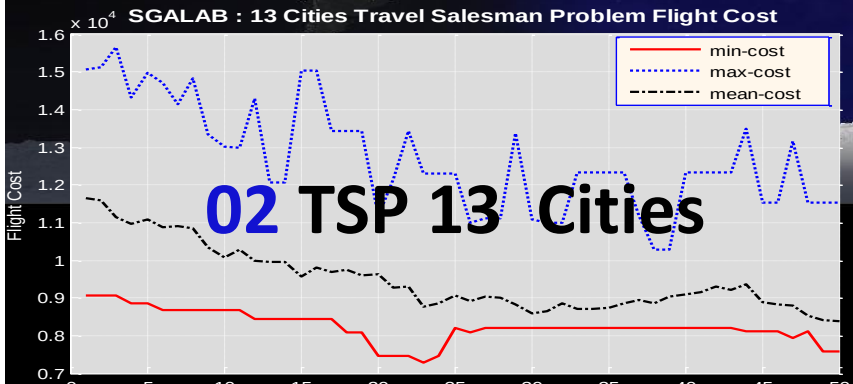
13 Cities Traveling Salesman Problem

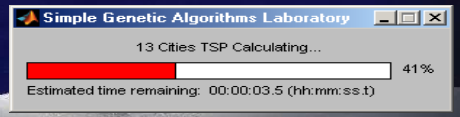
Simple Genetic Algorithms Laboratory (SGALAB)1.0.0.3 for Matlab 7.x

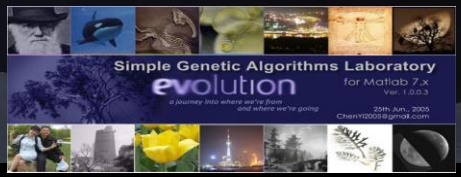
31st Nov., 2005

Chen Yi
chenyi2005@gmail.com

Cost	1SA	2NY	3BR	4MA	5LO	6ST	7JO	8MO	9ND	10CQ	11SH	12TO	13CA
1SA	0	106.05	1345	845	845	845	1295	845	607	656.69	525	1019	735.34
2NY	106.05	0	916	575	297	381	989	685	604	775.69	644	359	933.34
3BR	1109	999	0	784	919	1068	1279	1059	1643	2145	1971	1266	1954.68
4MA	382.95	273.71	882.89	0	44.18	79.78	352.26	239.34	1042.07	942.65	365.76	870.23	2163.71
5LO	564.38	496.88	632.54	47.49	0	18.26	465.82	155.27	405.54	886.89	548.03	652.16	1296.52
6ST	1374.72	260.9	877.05	54.18	18.06	0	702.44	124.43	434.84	936.59	634.87	600.09	2864.24
7JO	948.65	816.11	2054.74	610.59	580.81	647.82	0	610.59	707.39	1496.76	896.53	787.81	2696.92
8MO	509	315	810	291	170	145	433	0	307	774	525	700	1286.8
9ND	597.23	568.52	1519.73	384.98	372.12	385.9	551.29	385.9	0	546.69	470.89	493.86	1804.95
10CQ	703.19	836.1	2510.72	927.92	927.92	927.92	1496.76	1074.72	712.86	0	180.02	918.26	1767.14
11SH	555.79	459.13	1555	490.29	398.71	473.63	146.31	451.88	567.87	180.02	0	459.13	1753.85
12TO	1647.04	2229.67	3952.34	2352.91	2352.91	3323.96	3006.5	2711.45	2117.62	1266.09	427.26	0	2582.71
13CA	1319.61	1758.67	2873.73	911.69	911.69	1102.61	1114.11	911.69	794.37	2343.89	1025.93	2920.63	0







Best TSP Path :

10 →13→9→1→2→6→5→3→4→7→8→12→11→10

Lowest Flight Cost : 7.3505e+003

TSP 13 Cities Demo using SGALAB1003

The ‘SGALAB’ will take a world wide trip, there are 13 cities on the list of the travelling plan. Give 1~13 numbers to each city as the left table.

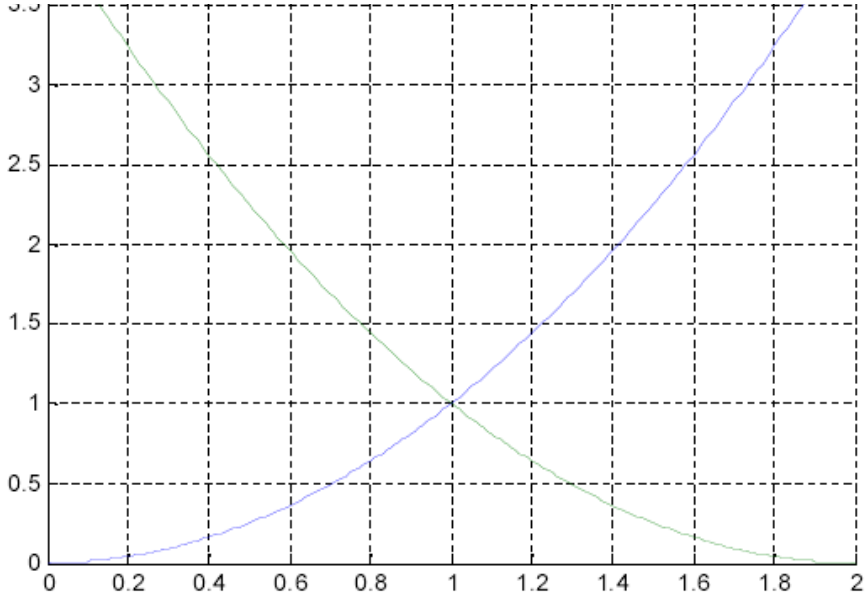
The start point is the city of Chongqing (10), how can SGALAB plan the visiting to all the listed cities with lowest traveling cost?

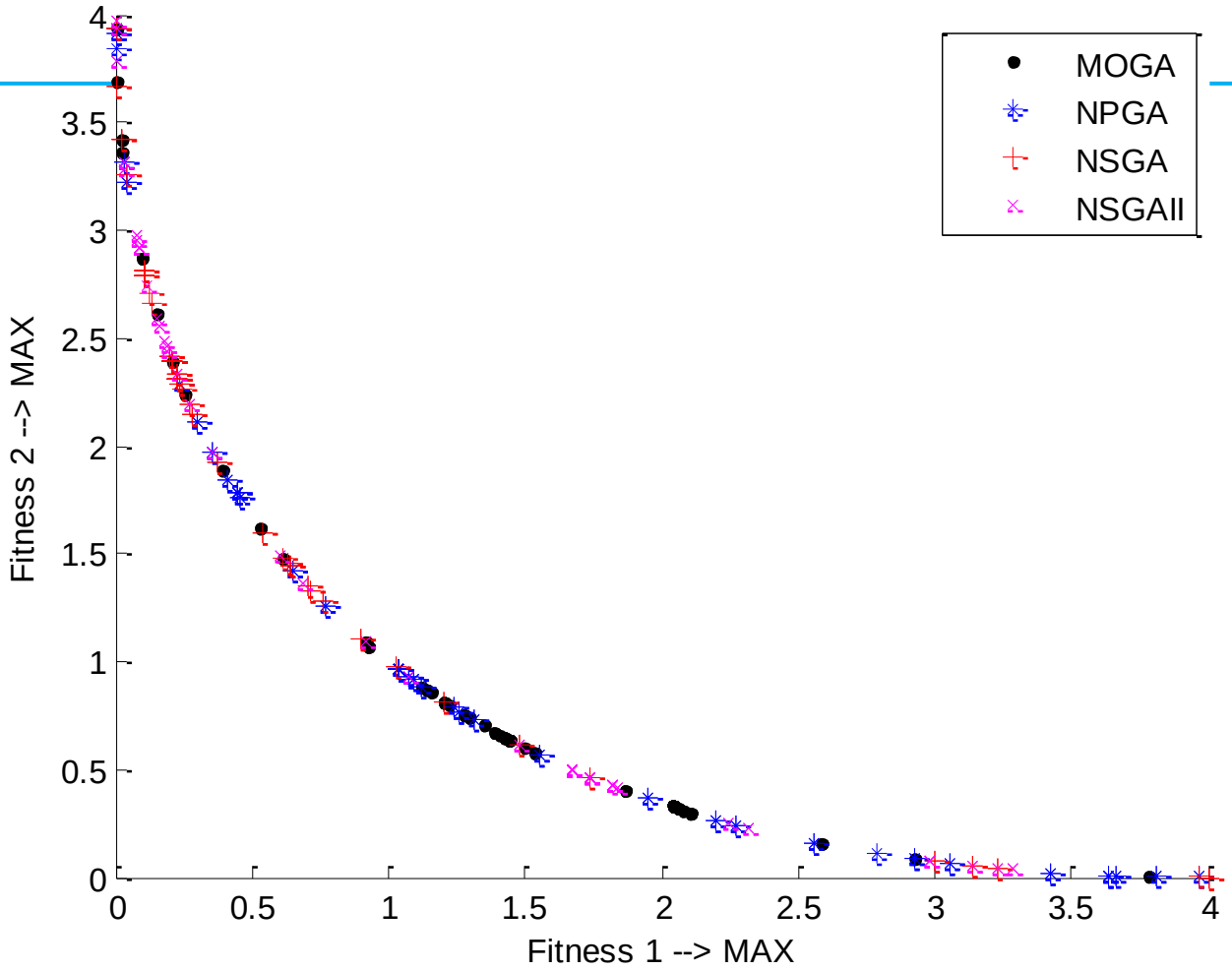
Num.	City
1	*San Francisco*
2	*New York*
3	*Brasilia*
4	*Madrid*
5	*London*
6	*Stockholm*
7	*Johannesburg*
8	*Moscow*
9	*New Delhi*
10	***Chongqing***
11	*Shanghai*
12	*Tokyo*
13	*Canberra*

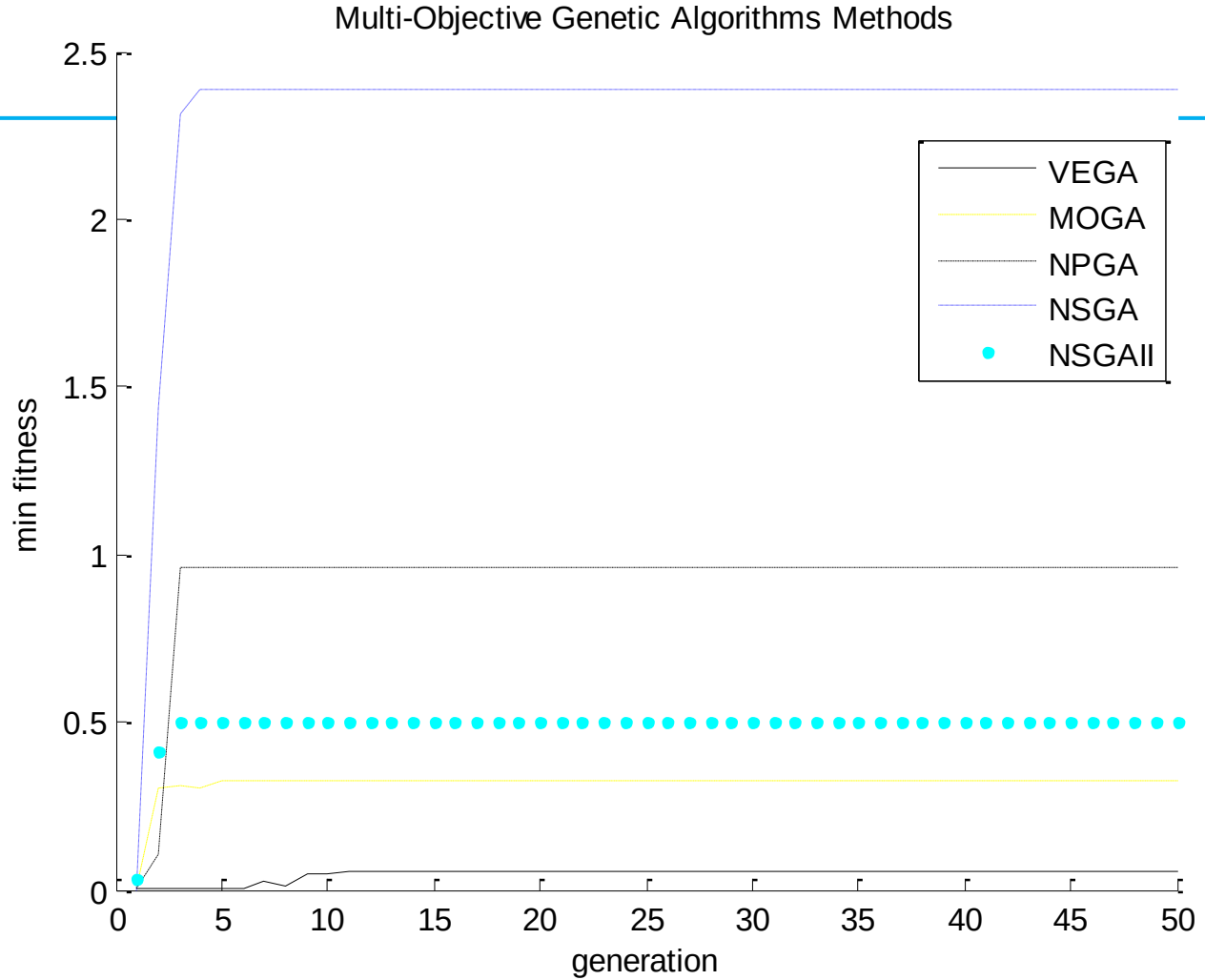
Case 03 Multi-objective Problems

Assume the following 2 fitness ,

$$\begin{cases} f_1(x) = x^2 \\ f_2(x) = (x-2)^2 \end{cases} \quad x \in [0,2]$$, so f1 = f2 = 1 should be the best results




MOGA
NPGA
NSGA
NSGAII
Fitness 2 --> MAX
Fitness 1 --> MAX
0
0.5
1
1.5
2
2.5
3
3.5
4


Multi-Objective Genetic Algorithms Methods
min fitness
generation
VEGA
MOGA
NPGA
NSGA
NSGAII
0
0.5
1
1.5
2
2.5
0
5
10
15
20
25
30
35
40
45
50

Links

- **GAs Videos**

01-MATLAB - What is a Genetic Algorithm (4:40mins)

02-How algorithms evolve (4:41mins)

03-Genetic Algorithms Explained By Example(11:51mins)

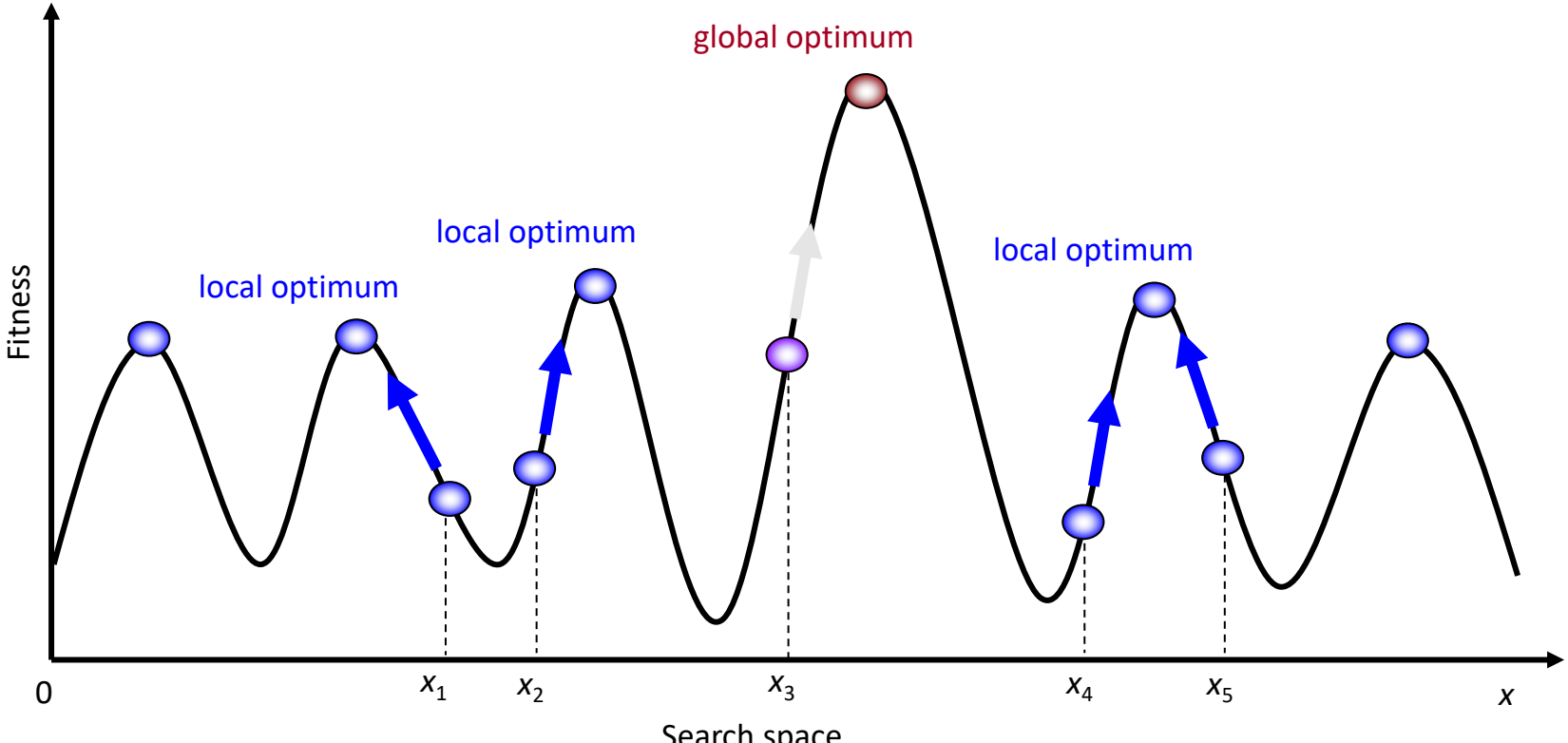
04-Genetic Algorithm from Scratch in Python (tutorial with code) (12:17mins)

FAQ

- **FAQ 01** Terminology (continued)
- **FAQ 02** Schema Theorem
- **FAQ 03** Advantages and Limitations
- **FAQ 04** Premature Convergence

FAQ 01 Terminology (continued)

- 15 Optimisation



15 Optimisation

- Optimisation is the process of **making something better**. In any process, we have a set of inputs and a set of outputs as shown in the following figure.
- Optimisation refers to finding the values of inputs in such a way that we get the "**best**" output values. The definition of "best" varies from problem to problem, but in mathematical terms, it refers to maximising or minimising one or more objective functions, by varying the input parameters.

FAQ 02 Schema Theorem

- A **Schema** is a "template". Formally, it is a string over the alphabet = {0,1,*}, where * is don't care and can take any value.
- Therefore, *10*1 could mean 01001, 01011, 11001, or 11011
- Geometrically, a schema is a hyper-plane in the solution search space.
- **Order** of a schema is the number of specified fixed positions in a gene.

FAQ 02 Schema Theorem

- **Defining length** is the distance between the two furthest fixed symbols in the gene.
- The schema theorem states that this schema with above **average** fitness, **short** defining length and **lower** order is more likely to survive crossover and mutation.
- The schema theorem **states** **that** this schema with above average fitness, short defining length and lower order is more likely to survive crossover and mutation.

FAQ 03 Advantages

- Does **not** require any derivative information (which may not be available for many real-world problems).
- Is **faster** and more **efficient** as compared to the traditional methods.
- Has very good **parallel** capabilities.
- Optimises **both** continuous and discrete functions and also multi-objective problems.
- Provides a list of "**good**" **solutions** and **not** just a single solution.
- Always gets an answer to the problem, which gets **better** over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

FAQ 03 Limitations

- GAs are **not** suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is calculated repeatedly which might be **computationally expensive** for some problems.
- Being stochastic, there are **no** guarantees on the optimality or the quality of the solution.
- If **not** implemented properly, the GA may not converge to the optimal solution.

Conventional Method vs. Genetic Algorithm

Conventional Method (point-to-point approach)

- Generally, algorithm for solving optimization problems is a sequence of computational steps which asymptotically converge to optimal solution.
- Most of classical optimization methods generate a deterministic sequence of computation based on the gradient or higher order derivatives of objective function.
- The methods are applied to a single point in the search space.
- The point is then improved along the deepest descending direction gradually through iterations.
- This point-to-point approach takes the danger of falling in local optima.

Conventional Method vs. Genetic Algorithm

Genetic Algorithm (population-to-population approach)

- Genetic algorithms performs a multiple directional search by maintaining a population of potential solutions.
- The population-to-population approach is hopeful to make the search escape from local optima.
- Population undergoes a simulated evolution: at each generation the relatively good solutions are reproduced, while the relatively bad solutions die.
- Genetic algorithms use probabilistic transition rules to select someone to be reproduced and someone to die so as to guide their search toward regions of the search space with likely improvement.

FAQ 04 Premature Convergence

- **Maintaining good diversity** in the population is extremely crucial for the success of a GA.
- If, this taking up of the entire population by one extremely fit solution is known as **premature convergence** and is an undesirable condition in a GA.
- To prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the **solutions being close to one another** in the solution space thereby leading to a **loss of diversity**.

FAQ 05 Termination Condition

We usually want a termination condition such that our solution is close to the optimal, at the end of the run. Usually, we keep one of the following termination conditions:

- *When there has been **no improvement** in the population for X iterations.*
- *When we reach an absolute **number of generations**.*
- *When the objective function value has **reached a certain pre-defined value**.*

FAQ 05 Termination Condition

- Like other parameters of a GA, the termination condition is also **highly problem specific**
- the GA designer should try out various options to see what suits his particular problem the best.

FAQ 06 GAs' Application Areas

- **Optimization** − Genetic Algorithms are most commonly used in optimization problems wherein we have to maximize or minimize a given objective function value under a given set of constraints. The approach to solve Optimization problems has been highlighted throughout the tutorial.
- **Economics** − GAs are also used to characterize various economic models like the cobweb model, game theory equilibrium resolution, asset pricing, etc.
- **Neural Networks** − GAs are also used to train neural networks, particularly recurrent neural networks.
- **Parallelization** − GAs also have very good parallel capabilities, and prove to be very effective means in solving certain problems, and also provide a good area for research.
- **Image Processing** − GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.

FAQ 06 GAs' Application Areas

- **Vehicle routing problems** − With multiple soft time windows, multiple depots and a heterogeneous fleet.
- **Scheduling applications** − GAs are used to solve various scheduling problems as well, particularly the time tabling problem.
- **Machine Learning** − as already discussed, genetics based machine learning (GBML) is a niche area in machine learning.
- **Robot Trajectory Generation** − GAs have been used to plan the path which a robot arm takes by moving from one point to another.
- **Parametric Design of Aircraft** − GAs have been used to design aircrafts by varying the parameters and evolving better solutions.
- **DNA Analysis** − GAs have been used to determine the structure of DNA using spectrometric data about the sample.

FAQ 06 GAs' Application Areas

- **Multimodal Optimization** – GAs are obviously very good approaches for multimodal optimization in which we have to find multiple optimum solutions.
- **Traveling salesman problem and its applications** – GAs have been used to solve the TSP, which is a well-known combinatorial problem using novel crossover and packing strategies.

FAQ 07 Further Readings

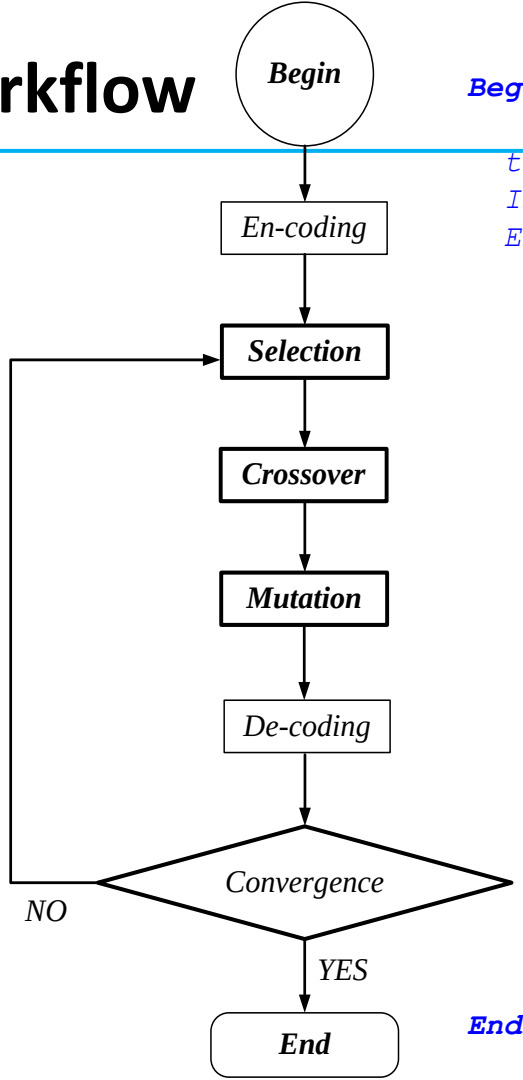
- Genetic Algorithms in Search, Optimization and Machine Learning by **David E. Goldberg**[7].
- Genetic Algorithms + Data Structures = Evolutionary Programs by **Zbigniew Michalewicz**[19].
- Randy L. Haupt, Sue Ellen Haupt, Practical Genetic Algorithms, 2nd Edition, John Wiley & Sons, Inc., 2004..[36]
- Multi Objective Optimization using Evolutionary Algorithms by **Kalyanmoy Deb**[37].

FAQ 08 Fitness Scaling Mechanisms

- Linear scaling $f_k^{'} = a \times f_k + b$
- Power low scaling $f_k^{'} = f_k^{\alpha}$
- Normalising scaling $f_k^{'} = \frac{f_k - f_{\min} + \gamma}{f_{\max} - f_{\min} + \gamma},$

  $0 < \gamma < 1$ (for maximizati on problem
- Boltzmann scaling $f_k^{'} = e^{f_k / T}$

GA Workflow




```
Begin (1)

  t = 0 ;
  Initialise P(t);
  En-coding P(t);

    While ( Not termination-condition) do

      Begin (2)

        {
          t = t+1;

          Select P(t) from P(t-1)
          Crossover P(t);
          Mutation P(t);

          De-coding P(t);
          Evaluate P(t);
                                        }

      End (2)

End (1)
```

Introduction to Artificial Intelligence -02-02 Genetic Algorithms

Thanks and Questions

This is the Last Page

Dr Leo Chen
leo.chen@ieee.org
23/Feb/2023