ADVANCED TOPICS OF GENETIC ALGORITHMS

DR H.K. LAM

Department of Informatics

King's College London

Office (N)5.05, Bush House, Strand Campus
Email: hak-keung.lam@kcl.ac.uk

Biologically Inspired Methods (6CCS3BIM/7CCSMBIM)

Outline

Learning Objectives

- To know the limits of GA.
- To cover some advanced topics of GA.
- To understand why binary GA works.

Handling Expensive Cost Functions

- Complicated cost function $\Rightarrow$ time consuming for evaluation
- **Approaches:**
  - Identical chromosomes are not evaluated more than once.
  - Only non-identical chromosomes are allowed in the population.
  - Duplicated offspring is discarded (if searching time is less than evaluation time).
  - Does not allow identical chromosomes to mate.
  - Only evaluate the costs of offspring of mutated chromosomes.

Handling Expensive Cost Functions

Population with non-identical Chromosomes:

Example: 8 Chromosomes; 9 bits, 3 bits for each gene

First 3 bits are different	First bit of each gene is different
000101010	011000010
001101010	011000110
010101010	011100010
011101010	011100110
100101010	111000010
101101010	111000110
110101010	111100010
111101010	111100110

Multiple-Objective Optimisation

Definition: Given a set of cost functions, $f_i(\mathbf{x})$, $i = 1, \cdots, K$, where $\mathbf{x} \in \Re^n$ is a decision variable vector, find a vector $\mathbf{x}^*$ which minimises $\{f_1(\mathbf{x}^*), \cdots, f_K(\mathbf{x}^*)\}$.

- Objectives conflict each other.
- Optimising $\mathbf{x}$ with respect to one cost function will result in degrading others.
- It is not possible to have all cost functions being optimised.
- Find an acceptable solution or a feasible solution (when constraints are considered).

Example: Maximising the yield and minimising the costs (running costs, production costs, labour costs, production time, etc.) are conflicting to each other.

Weighted Sum Approach

- $f(\mathbf{x}) = \sum_{i=1}^{K} w_i f_i(\mathbf{x})$
- $w_i > 0$ and $\sum_{i=1}^{K} w_i = 1$

Advantages:

- Simple and straightforward.
- Single cost function.

Disadvantages:

- w_i has to be provided by the user.
- There is no rule to select w_i.

Note: All $f_i(\mathbf{x})$ must be of the same type of optimisation problem, say, all minimisation.

Gray Codes

Problem: Representation of the variable values using ordinary binary number may lead to slow convergence of a GA.

$$parent_1 = \left[\ \cdots \underbrace{100 \uparrow 00000}_{gene} \cdots\ \right] = [\cdots 128 \cdots]$$

$$parent_2 = \left[\ \cdots \underbrace{011 \uparrow 11111}_{gene} \cdots\ \right] = [\cdots 127 \cdots]$$

Gray Codes

Problem: Representation of the variable values using ordinary binary number may lead to slow convergence of a GA.

$$parent_1 = \left[\cdots \underbrace{100 \uparrow 00000}_{gene} \cdots \right] = [\cdots 128 \cdots]$$

$$parent_2 = \left[\cdots \underbrace{011 \uparrow 11111}_{gene} \cdots \right] = [\cdots 127 \cdots]$$

After crossover:

$$offspring_1 = \left[\cdots \underbrace{10011111}_{gene} \cdots \right] = [\cdots 159 \cdots]$$

$$offspring_2 = \left[\cdots \underbrace{01100000}_{gene} \cdots \right] = [\cdots 96 \cdots]$$

- The offspring have diverging values from the parents.
- In some cases, two adjacent numbers may have many bit difference. For example, 3 is represented by 011 and 4 is represented by 100. All bits are different.
- The binary number encoding is nonlinear in number of bit difference, which introduce nonlinearity in the optimisation problem.

Gray Codes

A solution: Representation of the variable values using Gray code.

Property:

• Binary representations of consecutive numbers have a Hamming distance of one. In words, the number of bit different between two consecutive numbers is one.

• Hamming distance: the number of bits by which two binary numbers differ.

Example: The Hamming distance of 111111 and 010010 is 4.

Gray Codes

Consider the 127 and 128 are represented as 01000000 and 11000000, respectively (Hamming distance is 1).

$$parent_1 = \left[\cdots \underbrace{110 \uparrow 00000}_{gene} \cdots \right] = [\cdots 128 \cdots]$$

$$parent_2 = \left[\cdots \underbrace{010 \uparrow 00000}_{gene} \cdots \right] = [\cdots 127 \cdots]$$

Gray Codes

Consider the 127 and 128 are represented as 01000000 and 11000000, respectively (Hamming distance is 1).

$$parent_1 = \left[\ \cdots \underbrace{110 \uparrow 00000}_{gene} \cdots\ \right] = [\cdots 128 \cdots]$$

$$parent_2 = \left[\ \cdots \underbrace{010 \uparrow 00000}_{gene} \cdots\ \right] = [\cdots 127 \cdots]$$

After crossover:

$$offspring_1 = \left[\ \cdots \underbrace{11000000}_{gene} \cdots\ \right] = [\cdots 128 \cdots]$$

$$offspring_2 = \left[\ \cdots \underbrace{01000000}_{gene} \cdots\ \right] = [\cdots 127 \cdots]$$

- Parents with good solutions are more likely to produce good solutions.

Gray Codes

Benefits of using gray codes instead of binary codes

- The hamming distance between two adjacent numbers is one, which means that the number of bit difference between two adjacent numbers is one, which reduces the nonlinearity introduced by coding.
- In the crossover process, some adjacent numbers are represented by binary codes with large number of bit difference, e.g., 3 represented by 011 to 4 represented by 100 (3 bit difference). After crossover, the offspring are very different from their parents, e.g., the parents 0|11 and 1|00 produce offspring 000 representing 0 and 111 representing 7. The big jump in candidate solutions may affect the search result especially local search is needed (when all candidates are more all less the same quality near the end of the search).
- In the mutation process, usually a small number of bits will be flipped. When binary codes are used, in some cases, a large number of bit flips are required to change a binary number to its adjacent number, e.g., 3 represented by 011 to 4 represented by 100 (3 bit difference). This will affect the search process especially when local search is needed.

Binary Code to Gray Code

Binary code to Gray code conversion: Consider a binary code of n bits, i.e., $b_n b_{n-1} b_{n-2} \cdots b_2 b_1$.

$$\begin{aligned}
g_n &= b_n \\
g_{n-1} &= b_n \oplus b_{n-1} \\
g_{n-2} &= b_{n-1} \oplus b_{n-2} \\
&\vdots \\
g_2 &= b_3 \oplus b_2 \\
g_1 &= b_2 \oplus b_1
\end{aligned}$$

where $\oplus$ denotes the exclusive OR (XOR) operation; $A \oplus B = A\overline{B} + \overline{A}B$

$(0 \oplus 0 = 0; \quad 0 \oplus 1 = 1; \quad 1 \oplus 0 = 1; \quad 1 \oplus 1 = 0)$

Gray code: $g_n g_{n-1} g_{n-2} \cdots g_2 g_1$

Binary Code to Gray Code

Example: Represent the 4-bit binary number 1011 in Gray code.

$$\begin{tikzcd}
 & \text{MSB} & & & & & & \\
\text{Binary:} & 1 \arrow[r] \arrow[d] & \oplus \arrow[r] & 0 \arrow[r] \arrow[d] & \oplus \arrow[r] & 1 \arrow[r] \arrow[d] & \oplus \arrow[r] & 1 \arrow[d] \\
\text{Gray:} & 1 & & 1 & & 1 & & 0
\end{tikzcd}$$

Example: Represent the 5-bit binary number 10000 in Gray code.

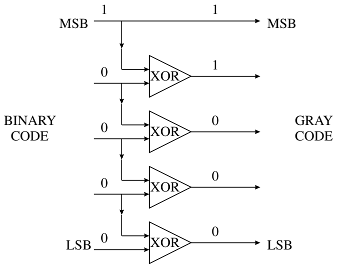


Figure 1: Logic circuit converting binary code to Gray code.

Gray Code to Binary Code

Gray code to binary code conversion: Consider a Gray code of n bits,

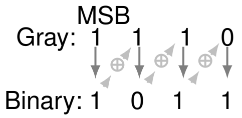
i.e., $g_n g_{n-1} g_{n-2} \cdots g_2 g_1$.

$$\begin{aligned} b_n &= g_n \\ b_{n-1} &= b_n \oplus g_{n-1} \\ b_{n-2} &= b_{n-1} \oplus g_{n-2} \\ &\vdots \\ b_2 &= b_3 \oplus g_2 \\ b_1 &= b_2 \oplus g_1 \end{aligned}$$

Binary code: $b_n b_{n-1} b_{n-2} \cdots b_2 b_1$

Gray Code to Binary Code

Example: Represent the 4-bit Gray code 1110 in binary code.



Example: Represent the 5-bit binary number 10000 in Gray code.

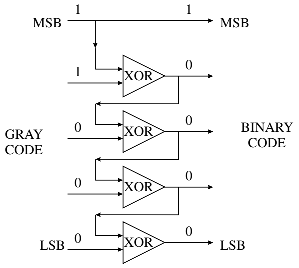


Figure 2: Logic circuit converting Gray code to binary code.

Permutation Problem

- **Permutation problem:** Optimisation problems involve sorting a list or putting things in the right order.
- The standard crossover and mutation operators are not appropriate.

Example: Reorder 6 numbers to minimise a function.

$parent_1 = [3\ 4 \uparrow 6\ 2\ 1\ 5]$

$parent_2 = [4\ 1 \uparrow 5\ 3\ 2\ 6]$

After single-point crossover

$offspring_1 = [3\ 4\ 5\ 3\ 2\ 6]$

$offspring_2 = [4\ 1\ 6\ 2\ 1\ 5]$

Permutation Problem

Four Methods:

1. Partially matched crossover (PMX)
2. Ordered crossover (OX)
3. Cycle crossover (CX)
4. Coding with reference list

Permutation Problem

Partially matched crossover (PMX) - Steps:

1. Select two crossover points.
2. Swap genes between parents according to the crossover points.
3. Swap duplicated genes between parents beyond the crossover points.

Permutation Problem

Partially matched crossover (PMX) Example:

Step 1: Select two crossover points

$parent_1 = [3 \uparrow 4\ 6 \uparrow 2\ 1\ 5]$

$parent_2 = [4 \uparrow 1\ 5 \uparrow 3\ 2\ 6]$

Permutation Problem

Partially matched crossover (PMX) Example:

Step 1: Select two crossover points

$parent_1 = [3 \uparrow 4\ 6 \uparrow 2\ 1\ 5]$

$parent_2 = [4 \uparrow 1\ 5 \uparrow 3\ 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [3\ 1\ 5\ 2\ \underline{1}\ \underline{5}]$

$offspring_{2A} = [\underline{4}\ 4\ 6\ 3\ 2\ \underline{6}]$

Permutation Problem

Partially matched crossover (PMX) Example:

Step 1: Select two crossover points

$parent_1 = [3 \uparrow 4\ 6 \uparrow 2\ 1\ 5]$

$parent_2 = [4 \uparrow 1\ 5 \uparrow 3\ 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [3\ 1\ 5\ 2\ \underline{1}\ \underline{5}]$

$offspring_{2A} = [\underline{4}\ 4\ 6\ 3\ 2\ \underline{6}]$

Step 3: Swap Duplicates

$offspring_{1B} = [3\ 1\ 5\ 2\ \mathbf{4}\ \mathbf{6}]$

$offspring_{2B} = [\mathbf{1}\ 4\ 6\ 3\ 2\ \mathbf{5}]$

Permutation Problem

Ordered crossover (OX) - Steps:

1. Select two crossover points.
2. Swap genes between parents according to the crossover points.
3. Cross out the duplicated genes in the parent.
4. Copy the remaining genes from the parent to the offspring starting from the second crossover point.

Advantage: relative ordering is preserved (although the absolute position within the string is not).

Permutation Problem

Ordered crossover (OX) Example:

Step 1: Select two crossover points

$parent_1 = [3\ 4 \uparrow 6\ 2 \uparrow 1\ 5]$

$parent_2 = [4\ 1 \uparrow 5\ 3 \uparrow 2\ 6]$

Permutation Problem

Ordered crossover (OX) Example:

Step 1: Select two crossover points

$parent_1 = [3\ 4 \uparrow 6\ 2 \uparrow 1\ 5]$

$parent_2 = [4\ 1 \uparrow 5\ 3 \uparrow 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [3\ 4\ 5\ 3\ 1\ 5]$

$offspring_{2A} = [4\ 1\ 6\ 2\ 2\ 6]$

Permutation Problem

Ordered crossover (OX) Example:

Step 1: Select two crossover points

$parent_1 = [3\ 4 \uparrow 6\ 2 \uparrow 1\ 5]$

$parent_2 = [4\ 1 \uparrow 5\ 3 \uparrow 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [3\ 4\ 5\ 3\ 1\ 5]$

$offspring_{2A} = [4\ 1\ 6\ 2\ 2\ 6]$

Step 3: Cross out duplicates

$offspring_{1B} = [?\ ?\ 5\ 3\ ?\ ?]$

$parent_1 = [X\ 4 \uparrow 6\ 2 \uparrow 1\ X]$

$offspring_{2B} = [?\ ?\ 6\ 2\ ?\ ?]$

$parent_2 = [4\ 1 \uparrow 5\ 3 \uparrow X\ X]$

Permutation Problem

Ordered crossover (OX) Example:

Step 1: Select two crossover points

$parent_1 = [3\ 4 \uparrow 6\ 2 \uparrow 1\ 5]$

$parent_2 = [4\ 1 \uparrow 5\ 3 \uparrow 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [3\ 4\ 5\ 3\ 1\ 5]$

$offspring_{2A} = [4\ 1\ 6\ 2\ 2\ 6]$

Step 3: Cross out duplicates

$offspring_{1B} = [?\ ?\ 5\ 3\ ?\ ?]$

$parent_1 = [X\ 4 \uparrow 6\ 2 \uparrow 1\ X]$

$offspring_{2B} = [?\ ?\ 6\ 2\ ?\ ?]$

$parent_2 = [4\ 1 \uparrow 5\ 3 \uparrow X\ X]$

Step 4: Copy the remaining genes from the parent.

$offspring_{1C} = [6\ 2\ 5\ 3\ 1\ 4]$

$offspring_{2C} = [5\ 3\ 6\ 2\ 4\ 1]$

Permutation Problem

Cycle crossover (CX) - Steps:

1. Start from the leftmost genes of the chromosomes.
2. Swap genes between parents.
3. Move to the next duplicated gene and swap genes at that position.
4. Repeat Step 3 until all genes are unique.

Permutation Problem

Cycle crossover (CX) Example:

Step 1: Start from the leftmost genes

$parent_1 = [\mathbf{3}\ 4\ 6\ 2\ 1\ 5]$

$parent_2 = [\mathbf{4}\ 1\ 5\ 3\ 2\ 6]$

Permutation Problem

Cycle crossover (CX) Example:

Step 1: Start from the leftmost genes

$parent_1 = [\mathbf{3}\ 4\ 6\ 2\ 1\ 5]$

$parent_2 = [\mathbf{4}\ 1\ 5\ 3\ 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [4\ \underline{4}\ 6\ 2\ 1\ 5]$

$offspring_{2A} = [3\ 1\ 5\ 3\ 2\ 6]$

Permutation Problem

Cycle crossover (CX) Example:

Step 1: Start from the leftmost genes

$parent_1 = [\mathbf{3}\ 4\ 6\ 2\ 1\ 5]$

$parent_2 = [\mathbf{4}\ 1\ 5\ 3\ 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [4\ \underline{4}\ 6\ 2\ 1\ 5]$

$offspring_{2A} = [3\ 1\ 5\ 3\ 2\ 6]$

Step 3a: Swap Duplicates

$offspring_{1B} = [4\ 1\ 6\ 2\ \underline{1}\ 5]$

$offspring_{2B} = [3\ 4\ 5\ 3\ 2\ 6]$

Permutation Problem

Cycle crossover (CX) Example:

Step 1: Start from the leftmost genes

$parent_1 = [\mathbf{3}\ 4\ 6\ 2\ 1\ 5]$

$parent_2 = [\mathbf{4}\ 1\ 5\ 3\ 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [4\ \underline{4}\ 6\ 2\ 1\ 5]$

$offspring_{2A} = [3\ 1\ 5\ 3\ 2\ 6]$

Step 3a: Swap Duplicates

$offspring_{1B} = [4\ 1\ 6\ 2\ \underline{1}\ 5]$

$offspring_{2B} = [3\ 4\ 5\ 3\ 2\ 6]$

Step 3b: Swap Duplicates

$offspring_{1C} = [4\ 1\ 6\ \underline{2}\ 2\ 5]$

$offspring_{2C} = [3\ 4\ 5\ 3\ 1\ 6]$

Permutation Problem

Cycle crossover (CX) Example:

Step 1: Start from the leftmost genes

$parent_1 = [\mathbf{3}\ 4\ 6\ 2\ 1\ 5]$

$parent_2 = [\mathbf{4}\ 1\ 5\ 3\ 2\ 6]$

Step 2: Swap genes

$offspring_{1A} = [4\ \underline{4}\ 6\ 2\ 1\ 5]$

$offspring_{2A} = [3\ 1\ 5\ 3\ 2\ 6]$

Step 3a: Swap Duplicates

$offspring_{1B} = [4\ 1\ 6\ 2\ \underline{1}\ 5]$

$offspring_{2B} = [3\ 4\ 5\ 3\ 2\ 6]$

Step 3b: Swap Duplicates

$offspring_{1C} = [4\ 1\ 6\ \underline{2}\ 2\ 5]$

$offspring_{2C} = [3\ 4\ 5\ 3\ 1\ 6]$

Step 3c: Swap Duplicates

$offspring_{1D} = [4\ 1\ 6\ 3\ 2\ 5]$

$offspring_{2D} = [3\ 4\ 5\ 2\ 1\ 6]$

Permutation Problem

Coding with reference list - Steps:

1. Start from the leftmost gene.
2. Code the gene with the index of the actual element in the reference list $\{1, \cdots, N\}$.
3. Remove the used element in the list.
4. Move to the next gene and repeat Step 2 based on the updated reference list.

- **Advantage:** Any crossover methods can be applied.
- **Disadvantage:** Coding is required $\Rightarrow$ Computational time increases.

Permutation Problem

Example (Coding):

Step 1: Start from the leftmost

gene

Reference list $\{1, 2, 3, 4, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

Permutation Problem

Example (Coding):

Step 1: Start from the leftmost gene

Reference list $\{1, 2, 3, 4, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

Step 2: Coding

Reference list $\{1, 2, \textcircled{3}, 4, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Permutation Problem

Example (Coding):

Step 1: Start from the leftmost gene

Reference list $\{1, 2, 3, 4, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

Step 2: Coding

Reference list $\{1, 2, \textcircled{3}, 4, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [\mathbf{3} * * * * *]$

Step 3a: Update and code

Reference list $\{1, 2, \textcircled{4}, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ \mathbf{3} * * * *]$

Permutation Problem

Example (Coding):

Step 1: Start from the leftmost gene

Reference list $\{1,2,3,4,5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

Step 2: Coding

Reference list $\{1,2,\textcircled{3},4,5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and code

Reference list $\{1,2,\textcircled{4},5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ \mathbf{3}\ *\ *\ *\ *]$

Step 3b: Update and code

Reference list $\{1,2,5,\textcircled{6}\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ \mathbf{4}\ *\ *\ *]$

Permutation Problem

Example (Coding):

Step 1: Start from the leftmost gene

Reference list $\{1,2,3,4,5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

Step 2: Coding

Reference list $\{1,2,\textcircled{3},4,5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and code

Reference list $\{1,2,\textcircled{4},5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ \mathbf{3}\ *\ *\ *\ *]$

Step 3b: Update and code

Reference list $\{1,2,5,\textcircled{6}\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ \mathbf{4}\ *\ *\ *]$

Step 3c: Update and code

Reference list $\{1,\textcircled{2},5\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ 4\ \mathbf{2}\ *\ *]$

Permutation Problem

Example (Coding):

Step 1: Start from the leftmost gene

Reference list $\{1, 2, 3, 4, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

Step 2: Coding

Reference list $\{1, 2, \textcircled{3}, 4, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and code

Reference list $\{1, 2, \textcircled{4}, 5, 6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ \mathbf{3}\ *\ *\ *\ *]$

Step 3b: Update and code

Reference list $\{1, 2, 5, \textcircled{6}\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ \mathbf{4}\ *\ *\ *]$

Step 3c: Update and code

Reference list $\{1, \textcircled{2}, 5\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ 4\ \mathbf{2}\ *\ *]$

Step 3d: Update and code

Reference list $\{\textcircled{1}, 5\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ \mathbf{1}\ *]$

Permutation Problem

Example (Coding):

Step 1: Start from the leftmost gene

Reference list $\{1,2,3,4,5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

Step 2: Coding

Reference list $\{1,2,\textcircled{3},4,5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and code

Reference list $\{1,2,\textcircled{4},5,6\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ \mathbf{3}\ *\ *\ *\ *]$

Step 3b: Update and code

Reference list $\{1,2,5,\textcircled{6}\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ \mathbf{4}\ *\ *\ *]$

Step 3c: Update and code

Reference list $\{1,\textcircled{2},5\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ 4\ \mathbf{2}\ *\ *]$

Step 3d: Update and code

Reference list $\{\textcircled{1},5\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ \mathbf{1}\ *]$

Step 3d: Update and code

Reference list $\{5\}$

$parent_1 = [3\ 4\ 6\ 2\ 1\ 5]$

$\boxed{coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ \mathbf{1}]}$

Permutation Problem

Example (Decoding):

Step 1: Start from the leftmost gene

Reference list $\{1, 2, 3, 4, 5, 6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

Permutation Problem

Example (Decoding):

Step 1: Start from the leftmost gene

Reference list $\{1, 2, 3, 4, 5, 6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

Step 2: Decode

Reference list $\{1, 2, \textcircled{3}, 4, 5, 6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Permutation Problem

Example (Decoding):

Step 1: Start from the leftmost gene

Reference list $\{1,2,3,4,5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

Step 2: Decode

Reference list $\{1,2,\textcircled{3},4,5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and decode

Reference list $\{1,2,\textcircled{4},5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ \mathbf{4}\ *\ *\ *\ *]$

Permutation Problem

Example (Decoding):

Step 1: Start from the leftmost gene

Reference list $\{1,2,3,4,5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

Step 2: Decode

Reference list $\{1,2,\textcircled{3},4,5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and decode

Reference list $\{1,2,\textcircled{4},5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ \mathbf{4}\ *\ *\ *\ *]$

Step 3b: Update and decode

Reference list $\{1,2,5,\textcircled{6}\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ \mathbf{6}\ *\ *\ *]$

Permutation Problem

Example (Decoding):

Step 1: Start from the leftmost gene

Reference list $\{1,2,3,4,5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

Step 2: Decode

Reference list $\{1,2,\textcircled{3},4,5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and decode

Reference list $\{1,2,\textcircled{4},5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ \mathbf{4}\ *\ *\ *\ *]$

Step 3b: Update and decode

Reference list $\{1,2,5,\textcircled{6}\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ \mathbf{6}\ *\ *\ *]$

Step 3c: Update and decode

Reference list $\{1,\textcircled{2},5\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ 6\ \mathbf{2}\ *\ *]$

Permutation Problem

Example (Decoding):

Step 1: Start from the leftmost gene

Reference list $\{1, 2, 3, 4, 5, 6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

Step 2: Decode

Reference list $\{1, 2, \textcircled{3}, 4, 5, 6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and decode

Reference list $\{1, 2, \textcircled{4}, 5, 6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ \mathbf{4}\ *\ *\ *\ *]$

Step 3b: Update and decode

Reference list $\{1, 2, 5, \textcircled{6}\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ \mathbf{6}\ *\ *\ *]$

Step 3c: Update and decode

Reference list $\{1, \textcircled{2}, 5\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ 6\ \mathbf{2}\ *\ *]$

Step 3d: Update and decode

Reference list $\{\textcircled{1}, 5\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ 6\ 2\ \mathbf{1}\ *]$

Permutation Problem

Example (Decoding):

Step 1: Start from the leftmost gene

Reference list $\{1,2,3,4,5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

Step 2: Decode

Reference list $\{1,2,\textcircled{3},4,5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [\mathbf{3}\ *\ *\ *\ *\ *]$

Step 3a: Update and decode

Reference list $\{1,2,\textcircled{4},5,6\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ \mathbf{4}\ *\ *\ *\ *]$

Step 3b: Update and decode

Reference list $\{1,2,5,\textcircled{6}\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ \mathbf{6}\ *\ *\ *]$

Step 3c: Update and decode

Reference list $\{1,\textcircled{2},5\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ 6\ \mathbf{2}\ *\ *]$

Step 3d: Update and decode

Reference list $\{\textcircled{1},5\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$Decoded\ parent_1 = [3\ 4\ 6\ 2\ \mathbf{1}\ *]$

Step 3d: Update and decode

Reference list $\{5\}$

$Coded\ parent_1 = [3\ 3\ 4\ 2\ 1\ 1]$

$\boxed{Decoded\ parent_1 = [3\ 4\ 6\ 2\ 1\ \mathbf{5}]}$

Permutation Problem

Mutation:

a. **Inversion** – reverse the substring between two points.

Example: $[6 \uparrow 1\ 5\ 3\ 2 \uparrow 4]$, After mutation: $[6 \uparrow 2\ 3\ 5\ 1 \uparrow 4]$.

b. **Insertion and Displacement** – select a substring and insert it in a random place.

Example: $[6\ 1\ 5\ 3\ 2\ 4]$, After mutation: $[6\ 5\ 3\ 2\ 1\ 4]$

c. **Reciprocal exchange** – randomly choose two positions within the chromosome to swap. (Decoding is required when reference-list coding method is applied)

Example: $[6\ \mathbf{1}\ 5\ 3\ \mathbf{2}\ 4]$, After mutation: $[6\ \mathbf{2}\ 5\ 3\ \mathbf{1}\ 4]$

Penalty Function

- A method of handling constraints (equality and/or inequality constraints).

$$f_p(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m} C_i(\mathbf{x})$$

- $f_p(\mathbf{x})$: penalised cost function,
- $f(\mathbf{x})$: unpenalised cost function,
- m: number of constraints,
- $C_i(\mathbf{x})$: a penalty function imposed for violation of constraints i.

Examples

Example 1 (coefficient identification): Identify the coefficients of $(8+1)\sin(5/4)+6\cos(3)+9-7/2=8.1009$, where the coefficients are unique integers in the range of 1 to 9.

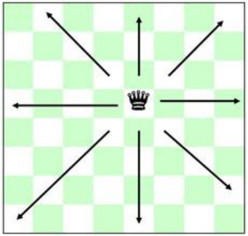
- Chromosome: $\mathbf{c}=[c_1,\cdots,c_9]$
- Cost function:

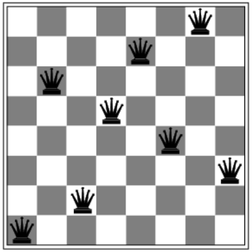
$$f(\mathbf{x})=|(c_1+c_2)\sin(c_3/c_4)+c_5\cos(c_6)+c_7-c_8/c_9-8.1009|.$$

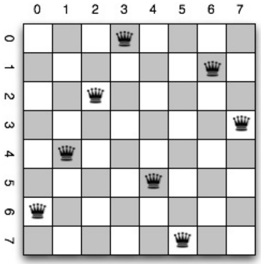
- Minimisation: $\min_{\mathbf{c}} f(\mathbf{c})$.
- $\mathbf{c}^*=[8,1,5,4,6,3,9,7,2]$; $\min_{\mathbf{c}^*} f(\mathbf{c}^*)=0$.

Examples

Example 2 (eight queens problem): Find a solution to the eight queens problem.



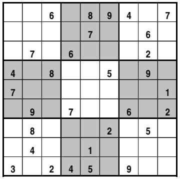




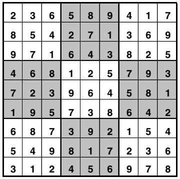
- 8 queens; 28 pairs
- Chromosome: $[x_1, \cdots, x_8]$, e.g., [7 2 6 3 1 4 0 5] (row index)
- Cost function: $f(\mathbf{x}) = 28 - h(\mathbf{x})$, $h(\mathbf{x})$: number of non-attacking pairs.
- Minimisation: $\min_{\mathbf{x}} f(\mathbf{x})$.

Examples

Example 3 (Sudoku puzzle): Solve Sudoku puzzle using GA.



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



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

- $x_{i,j}$: ij^{th} element of the Sudoku Puzzle.
- Chromosome: $[x_{1,1}, \cdots, x_{9,9}]$.
- Each row consists of the missing numbers in the range of 1 to 9.
- Cost function: $f(\mathbf{x}) = \sum_{i=1}^{9} |45 - \sum_{j=1}^{9} x_{i,j}| + \sum_{i=1}^{9} |9! - \prod_{j=1}^{9} x_{i,j}| + \sum_{i=1}^{3}\sum_{j=1}^{3} M_{i,j}$
- $M_{i,j}$: number of missing elements ($\{1, \cdots, 9\}$) in the ij^{th} block.
- Minimisation: $\min_{\mathbf{x}} f(\mathbf{x})$.

Examples

Example 4 (magic square): An $N \times N$ matrix with elements of 1 to N^2.

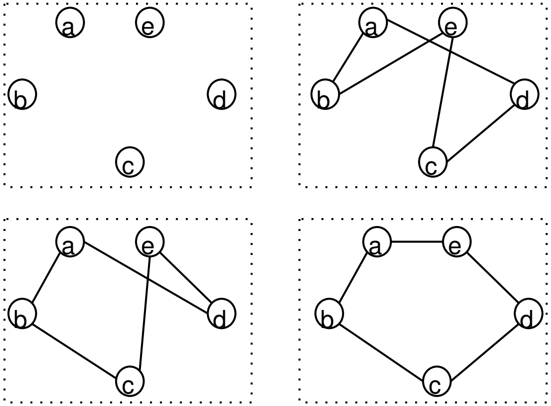
Each element must be of a unique integer. Find the matrix that the sum of the i^{th} row, i^{th} column, the primary and secondary diagonals are all equal.

$$\begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \vdots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix}$$

- Chromosome: $\mathbf{a} = [a_{11} \cdots a_{NN}]$
- $r_i = \displaystyle\sum_{j=1}^{N} a_{ij}$; $c_j = \displaystyle\sum_{i=1}^{N} a_{ij}$
- $d_1 = \sum_{i=1}^{N} a_{ii}$; $d_2 = \sum_{i=1}^{N} a_{i,N-i+1}$
- $s = \dfrac{\sum_{i=1}^{N} r_i + \sum_{j=1}^{N} c_j + d_1 + d_2}{2N+2}$
- $f(\mathbf{a}) = \displaystyle\sum_{i=1}^{N} |(r_i - s)| + \sum_{j=1}^{N} |(c_j - s)| + |d_1 - s| + |d_2 - s|$
- $\min_{\mathbf{a}} f(\mathbf{a})$

Examples

Example 5 (travelling salesman problem (TSP): Given a set of N cities, a salesman requires to find the shortest route visiting all cities once and returning to the starting city.



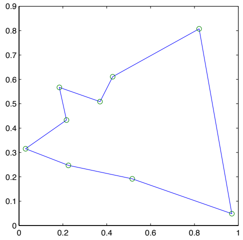
Examples

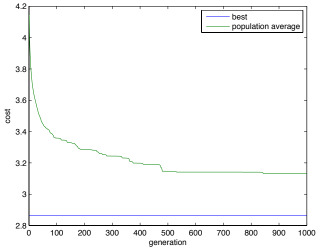
Example 5 (travelling salesman problem (TSP): Given a set of N cities, a salesman requires to find the shortest route visiting all cities once and returning to the starting city.

- City coordinates: (x_i, y_i), i = 1, $\cdots$, N
- Chromosome: $\mathbf{c} = [c_1, \cdots, c_N]$ (order of cities being visited)
- $f(\mathbf{c}) = \sum_{i=1}^{N-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} + \sqrt{(x_N - x_1)^2 + (y_N - y_1)^2}$
- Minimisation: $\min_{\mathbf{x}} f(\mathbf{x})$.

Examples

Example 5 (travelling salesman problem (TSP): Given a set of N cities, a salesman requires to find the shortest route visiting all cities once and returning to the starting city.





- 9 cities
- $N_{pop} = 10000$; $\mu = 0.2$; iterations: 1000; $X_{rate} = 50\%$
- $\mathbf{c}^* = [7\ 6\ 1\ 2\ 9\ 4\ 3\ 8\ 5]; f(\mathbf{c}^*) = 2.8654$

Examples

Example 6 (shipping application): Four items with different weights and values as shown in the following table are to form a shipment with a total weight of not more than 9 tons. Determine the number of each item maximizing the profit.

Item	Weight (Tons)	Net profit (£)
A	2	50
B	4	120
C	5	170
D	3	80

Table 1: Weights and values of Items

a. Formulate as minimisation problem by defining the cost functions and constraints.

b. Determine the penalised cost function for the minimisation problem.

c. Determine the representation of chromosome.

Examples

Let $\mathbf{x} = [x_1, x_2, x_3, x_4]$ be the number of items A, B, C and D, respectively.

a. $\min\limits_{\mathbf{x}} -(50x_1 + 120x_2 + 170x_3 + 80x_4)$ subject to

- $2x_1 + 4x_2 + 5x_3 + 3x_4 \leq 9$

b. $f(\mathbf{x}) = -(50x_1 + 120x_2 + 170x_3 + 80x_4) + \lambda_1 c_1$

- $c_1 = \begin{cases} |2x_1 + 4x_2 + 5x_3 + 3x_4 - 9|, & \text{if } 2x_1 + 4x_2 + 5x_3 + 3x_4 > 9 \\ c_1 = 0, & \text{otherwise} \end{cases}$
- $\lambda_1 > 0$

c. Chromosome: $[x_1, x_2, x_3, x_4]$

Examples

Example 7 (roots): Given $x^3 - 11x^2 - 20x + 300 = 0$, find the three roots. r_1, r_2 and r_3 (within -10 and 10) using GA.

a. Formulate as minimisation problem by defining the cost functions and constraints.

b. Determine the penalised cost function for the minimisation problem.

c. Determine the representation of chromosome.

a. $\min\limits_{x} |x^3 - 11x^2 - 20x + 300|$ subject to $x \neq r_1$ and $x \neq r_2$

b. $f(x) = |x^3 - 11x^2 - 20x + 300| + \lambda_1 \frac{1}{x - r_1} + \lambda_2 \frac{1}{x - r_2}$, $\lambda_1, \lambda_2 > 0$

c. Chromosome: $[x]$

Examples

Example 8 (Production): Two products X and Y are produced using two machines A and B where the processing times for production of each unit of X and Y are shown in the following table. Given 1) 30 units of X and 90 units of Y are currently in stock, 2) the number of units of products X and Y must be at least 75 and 95, respectively, 3) available processing times on machines A and B are 40 hours and 35 hours, respectively, at the start of the current week, maximise the combined sum of units of X and Y in stock at the end of the week.

Product	Machine A	Machine B
X	50	30
Y	24	33

Table 2: Processing times in minutes

a. Formulate as minimisation problem by defining the cost function and constraints.

b. Determine the penalised cost function for the minimisation problem.

c. Determine the representation of chromosome.

Examples

Example 8 (Production):

Denote x and y (integers) as the number of units of X and Y, respectively.

a. $\min_{x,y} -(x+y)$ subject to

- $50x + 24y \leq 40 \times 60$
- $30x + 33y \leq 35 \times 60$
- $x \geq 75 - 30 = 45$
- $y \geq 95 - 90 = 5$

Examples

Example 8 (Production):

b. $f(x,y) = -(x+y) + \lambda_1 c_1 + \lambda_2 c_2 + \lambda_3 c_3 + \lambda_4 c_4$

subject to

- $c_1 = \begin{cases} |50x + 24y - 2400|, & \text{if } 50x + 24y > 2400 \\ c_1 = 0, & \text{otherwise} \end{cases}$
- $c_2 = \begin{cases} |30x + 33y - 2100|, & \text{if } 30x + 33y > 2100 \\ c_2 = 0, & \text{otherwise} \end{cases}$
- $c_3 = |x - 45|$ if $x < 45$, otherwise $c_3 = 0$
- $c_4 = |y - 5|$ if $y < 5$, otherwise $c_4 = 0$
- $\lambda_1, \lambda_2, \lambda_3, \lambda_4 > 0$

c. chromosome: $[x, y]$

Examples

Example 9 (Job shop scheduling): We have 5 jobs denoted as job 1 to job 5. Each job must be processed by machine 1 first and then machine 2. The units of processing time for each job on each machine is given in the following table. Machines 1 and 2 are shared by all jobs. Find the job sequence such that makespan [1] is minimum.

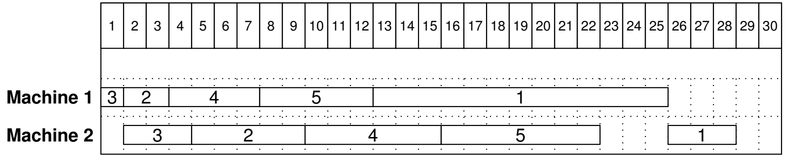
Job	Machine 1	Machine 2
1	13	3
2	2	5
3	1	3
4	4	6
5	5	7

a. Formulate as minimisation problem by defining the cost function and representation of chromosome.

[1] In manufacturing, the time difference between the start and finish of a sequence of jobs or tasks. Source: http://en.wiktionary.org/wiki/makespan

Examples

Example 9 (Job shop scheduling) cont'd:



a. Chromosome: $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]$ where $x_i \in \{1, 2, 3, 4, 5\}$ is a unique job number. This is a permutation problem. $\mathbf{x}$ represents the job sequence, e.g., $\mathbf{x} = \{3, 2, 4, 5, 1\}$ as shown in the above figure.

Cost: $\sum_{i=1}^{5} (T(x_i) + I(x_i))$ where $T(x_i)$ is the time units required to process job x_i by machine 2 and $I(x_i)$ is the idle time unit machine 2 has to wait before processing job x_i.

In this example, cost = $\underbrace{\overbrace{3}^{T} + \overbrace{1}^{I}}_{\text{Job 3}} + \underbrace{\overbrace{5}^{T} + \overbrace{0}^{I}}_{\text{Job 2}} + \underbrace{\overbrace{6}^{T} + \overbrace{0}^{I}}_{\text{Job 4}} + \underbrace{\overbrace{7}^{T} + \overbrace{0}^{I}}_{\text{Job 5}} + \underbrace{\overbrace{3}^{T} + \overbrace{3}^{I}}_{\text{Job 1}} = 28$

Genetic Programming

- The (original) aim of genetic programming (GP) is to evolve executable computer programs (A GP is a computer program to write another computer program).
- The chromosome representation uses a **tree structure** (while GA uses **string**) with terminal set and function set.
- The terminal set specifies all variables and constants.
- The function sets contains all the functions that can be applied to the elements of the terminal set.
- The evolved program is executed to measure its performance within the problem domain to quantify the fitness of that program.

Genetic Programming

Tree-Based Representation:

- Each chromosome represents a program.
- **Adaptive Chromosome**: The chromosomes are of variable length structure in terms of size, shape and complexity.
  - **Size**: tree depth
  - **Shape**: branching factor of nodes in the tree
- **Domain-Specific Grammar:** A grammar needs to be defined to solve a specific problem.
  - **Terminal set** (variables and constants). *Elements of the terminal set form the leaf nodes of the evolved tree*.
  - **Function set** (all the functions applied to the elements of the terminal set, e.g., mathematical, arithmetic and/Boolean functions, even decision structures such as *if-then-else* and loops). *Element of the function set from the non-leaf nodes*.
  - **Semantic rules** to ensure the construction of semantically correct trees.

Genetic Programming

Example 10 (Boolean expression evolution - XOR):

(x_1 AND NOT x_2) OR (NOT x_1 AND x_2)

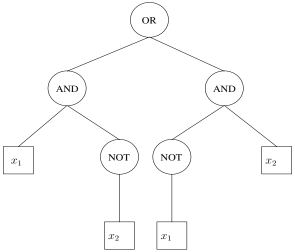


Figure 3: Tree representation of XOR.

x_1	x_2	Target output
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: XOR truth table

- Terminal set: { x_1, x_2 }
- Function set: { AND, OR, NOT }

Genetic Programming

Example 11 (mathematical expression):

$$y = x * \ln(a) + \sin(z) / \exp(-x) - 3.5$$

- Terminal set: { a, x, z, 3.5 }
- Function set: { $-$, +, *, /, sin, exp, ln }

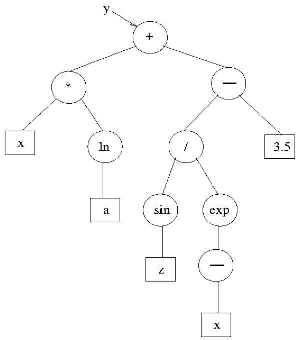


Figure 4: Tree representation of a mathematical expression.

Genetic Programming

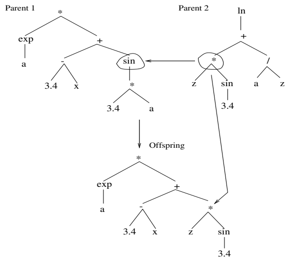
Process

- **Initial population** (randomly generated)
- **Cost (fitness) function**: measure the performance of the chromosomes
- **Crossover**: exchange information between parents
- **Mutation**: explore new information

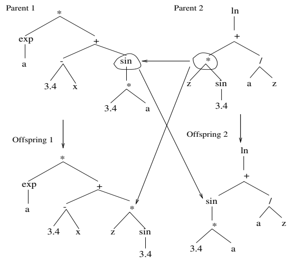
Genetic Programming

Crossover

- **Generating one offspring**
  - Select a random node and replace the corresponding sub-tree of one parent by that of other.
- **Generating two offspring**
  - Select a random node and swap the sub-trees between parents



(a) One offspring



(b) Two offspring

Genetic Programming

Mutation

- Function node mutation
- Terminal node mutation
- Swap mutation
- Grow mutation
- Gaussian mutation
- Trunc mutation

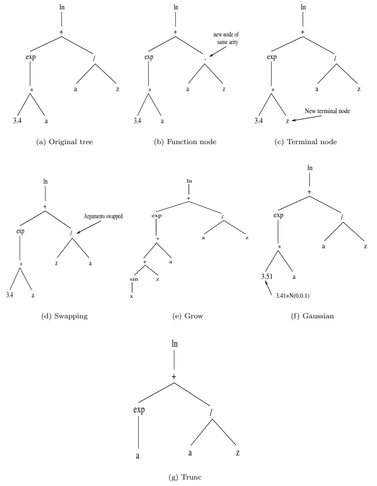


Figure 5: Different mutation methods.

Why do GAs work? Schema Theorem

- **Binary** genetic algorithms are considered.

Schema Theorem (Michalewicz, 1992)

"Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm."

Why do GAs work? Schema Theorem

- **A scheme:** a template representing a subset of binary strings using symbols '0', '1' and $\star$ (don't care symbol).

Example:

Schema $\star$1100 matches 2 strings: {**0**1100, **1**1100}.

Schema $\star$110$\star$ matches 4 strings: {**0**110**0**, **0**110**1**, **1**110**0**, **1**110**1**}.

Schema $\star\star\star\star\star$ matches 2^5 strings: {00000, 00001, $\cdots$, 11110, 11111}.

Schema 11100 matches 1 string {11100}.

String 11100 is matched by 2^5 schemata: {11100, $\star$1100, 1$\star$100, $\cdots$, $\star\star\star\star0$, $\star\star\star\star\star$}.

Why do GAs work? Schema Theorem

Schema properties:

1. **Order** of the schema *S* (denoted by $o(S)$): the number of fixed positions ('0' and '1' positions), i.e., the length of the template minus the number of don't care symbols.
2. **Defining length** of the schema *S* (denoted by $\delta(S)$): the distance between the first and the last fixed positions.

Example:

$$S_1 = \star\star\star 001 \star 110 \Rightarrow o(S_1) = 6, \delta(S_1) = 10 - 4 = 6$$

$$S_2 = \star\star\star\star 00 \star\star 0 \star \Rightarrow o(S_2) = 3, \delta(S_2) = 9 - 5 = 4$$

$$S_3 = 11101 \star\star 001 \Rightarrow o(S_3) = 8, \delta(S_3) = 10 - 1 = 9$$

Why do GAs work? Schema Theorem

Binary GAs:

1. select $P(t)$ based on $P(t-1)$
2. recombine $P(t)$
3. evaluate $P(t)$
4. $t \leftarrow t+1$

Binary GA configurations:

- $N_{keep} = 0$.
- Selection: roulette wheel weighting, **cost weighting**.
- Crossover: single-point crossover (crossover points are allowed any point in between the first and last bits).
- Mutation: uniform mutation (with the probability of mutation p_m for each bit).
- Elitism is not implemented.

Why do GAs work? Schema Theorem

$\xi(S,t)$: the number of strings in the population at generation t matched by a schema S.

Example: Consider the following population (**costs are normalised** and thus of all negative):

$$\begin{array}{rcll} v_1 & = & 100101 & f(v_1) = -15 \\ v_2 & = & 011100 & f(v_2) = -22 \\ v_3 & = & 110111 & f(v_3) = -6 \\ v_4 & = & 000110 & f(v_4) = -1 \\ v_5 & = & 110001 & f(v_5) = -8 \\ v_6 & = & 110011 & f(v_6) = -3 \end{array}$$

$S_1 = \star\star 01 \star\star, \qquad \xi(S_1,t) = 3$

Why do GAs work? Schema Theorem

$f(S,t)$: average cost of all strings in the population matched by the schema S.

Assuming that there are p strings $\{u_1, \cdots, u_p\}$ in the population matched by a schema S at generation t,

$$f(S,t) = \frac{\sum_{i=1}^{p} f(u_i)}{p}$$

Example:

$S_1 = \star\star01\star\star$ is matched by $\{v_1, v_3, v_4\} = \{100101, 110111, 000110\}$

$f(S_1,t) = \frac{f(v_1)+f(v_3)+f(v_4)}{3} = \frac{-15-6-1}{3} = -7.3333$

Why do GAs work? Schema Theorem

Objective: Investigate the probability of survival of all schemata S $(S_1, \cdots, S_{2^{N_{bits}}})$ in the GA process (selection, crossover, mutation).

Selection: The probability of selecting the string v_i:

$$p_i = \frac{f(v_i)}{F(t)}, i = 1, \cdots, N_{pop}$$

where $F(t) = \sum_{j=1}^{N_{pop}} f(v_j)$.

1. The number of strings matched by schema S: $\xi(S,t)$.
2. The average probability of a string matched by schema S to be selected: $\frac{f(S,t)}{F(t)}$.
3. The number of strings to be selected for recombination: N_{pop} $(N_{keep} = 0)$.

Why do GAs work? Schema Theorem

$\xi(S,t+1)$: the number of strings matched by S after the selection process.

$$\begin{aligned}\xi(S,t+1) &= \xi(S,t)\frac{f(S,t)}{F(t)}N_{pop}\\ &= \xi(S,t)\frac{f(S,t)}{\overline{F}(t)}\end{aligned}$$

where $\overline{F}(t) = \frac{F(t)}{N_{pop}}$ is the average cost of the population.

$$\begin{aligned}\xi(S,t+1) &= \xi(S,t)\frac{f(S,t)}{\overline{F}(t)}\\ &= \xi(S,t)\frac{\overline{F}(t)+f(S,t))-\overline{F}(t)}{\overline{F}(t)}\\ &= \xi(S,t)(1+\varepsilon(t))\end{aligned}$$

where $\varepsilon(t) = \frac{f(S,t))-\overline{F}(t)}{\overline{F}(t)}$.

Why do GAs work? Schema Theorem

$$\begin{aligned}
\xi(S,t+1) &= \xi(S,t)(1+\varepsilon(t)) \\
&= \underbrace{\xi(S,t-1)(1+\varepsilon(t-1))}_{\xi(S,t)}(1+\varepsilon(t)) \\
&= \underbrace{\xi(S,t-2)(1+\varepsilon(t-2))}_{\xi(S,t-1)}(1+\varepsilon(t-1))(1+\varepsilon(t)) \\
&= \underbrace{\xi(S,t-3)(1+\varepsilon(t-3))}_{\xi(S,t-2)}(1+\varepsilon(t-2))(1+\varepsilon(t-1))(1+\varepsilon(t)) \\
&\ \vdots \\
&= \xi(S,1)(1+\varepsilon(1))(1+\varepsilon(2))\cdots(1+\varepsilon(t-1))(1+\varepsilon(t))
\end{aligned}$$

- $\varepsilon(t) > 0$ for most of t: $\xi(S,t+1)$ is increasing
- $\varepsilon(t) < 0$ for most of t: $\xi(S,t+1)$ is decreasing

Implication: Above average schemata receive increasing number of strings in the next generation; however, below average schemata will die out as *t* increases.

Why do GAs work? Schema Theorem

Recombination - Crossover

Probability of destruction of a schema S: $p_d(S) = \frac{\delta(S)}{N_{bits}-1}$

Probability of schema S survival: $p_s(S) = 1 - \frac{\delta(S)}{N_{bits}-1}$

Example

A string $v_1 = 1101110010$ is matched by 2^{10} schemata.

$S_1 = 1101110010$ $\quad p_d(S_1) = \frac{9}{9} = 1, p_s(S_1) = 0$

$\vdots$

$S_a = \star\star\star 111 \star\star\star\star$ $\quad p_d(S_a) = \frac{2}{9}, p_s(S_a) = \frac{7}{9}$

$\vdots$

$S_b = 11 \star\star\star\star\star\star 10$ $\quad p_d(S_b) = \frac{9}{9} = 1, p_s(S_b) = 0$

$\vdots$

$S_{2^{N_{bits}}} = \star\star\star\star\star\star\star\star\star\star$ $\quad p_d(S_{2^{N_{bits}}}) = \frac{0}{9} = 0, p_s(S_{2^{N_{bits}}}) = 1$

Why do GAs work? Schema Theorem

$v_1 = \mathbf{1101110010}$

$v_2 = 0101010011$

Example 1: After crossover

$v'_1 = \mathbf{110111}|0011$

$v'_2 = 010101|\mathbf{0010}$

S_a survives but not S_b.

Example 2: After crossover

$v'_1 = \mathbf{110}|1010011$

$v'_2 = 010|\mathbf{1110010}$

S_a survives but not S_b.

Example 3: After crossover

$v'_1 = \mathbf{1101}|010011$

$v'_2 = 0101|\mathbf{110010}$

Both S_a and S_b cannot survive.

Example 4: After crossover

$v'_1 = \mathbf{11011}|10011$

$v'_2 = 01010|\mathbf{10010}$

Both S_a and S_b cannot survive.

Why do GAs work? Schema Theorem

$$p_s(S) = 1 - \frac{\delta(S)}{N_{bits} - 1}$$

Modification of $p_s(S)$ considering that, e.g., v_2 is the same as v_1

$$p_s(S) \geq 1 - \frac{\delta(S)}{N_{bits} - 1}$$

Schema growth equation with the consideration of crossover:

$$\xi(S,t+1) \geq \xi(S,t)\frac{f(S,t)}{\overline{F}(t)}\left(1 - \frac{\delta(S)}{N_{bits} - 1}\right)$$

Why do GAs work? Schema Theorem

Mutation: Uniform mutation - each bit will be mutated if a random number $r < p_m$, where $p_m \in [0,1]$ is the probability of mutation.

Probability of a single bit survival (no mutation takes place):

$$1 - p_m$$

Probability of a schema S survival (no mutation takes place in fixed bits):

$$p_s(S) = (1 - p_m)^{o(S)}$$

Why do GAs work? Schema Theorem

Schema growth equation with the consideration of crossover and mutation:

$$\xi(S,t+1) \geq \xi(S,t)\frac{f(S,t)}{\overline{F}(t)}\left(1-\frac{\delta(S)}{N_{bits}-1}\right)(1-p_m)^{o(S)}$$

Schema Theorem (Michalewicz, 1992)

"Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm."