# Nature Inspired Search and Optimisation
# Advanced Aspects of Nature Inspired Search and Optimisation

## Lecture 7: Real-valued Coded Evolutionary Algorithms

Shan He

School for Computational Science
University of Birmingham

February 4, 2020

# Outline of Topics

# Arguments for using binary coding

- “The binary alphabet maximises the level of **implicit parallelism**” [1]
- ‘**Schema**’: a template that identifies a subset of strings with similarities at certain string positions

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Chromosome 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| Chromosome 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| Schema 1 | * | 0 | * | * | 0 | 1 | * | 0 |
| Another Schema | * | * | * | * | 0 | 1 | * | * |

- The ‘*’ symbol is a wildcard that represents either a 0 or 1.

[1] Goldberg, D.E. (1991c). Genetic and Evolutionary Algorithms Come of Age. Communication of the Association for Computing Machinery 37(3), 113–119.

# Explaination of implicit parallelism

- If a chromosome is of length $L$ then it contains $3^L$ schemata (as 3 possibilities, i.e., 0, 1 or * at each position)
- For a population of $M$ individuals we are evaluating up to $M \cdot 3^L$ schemata
- Example:
  - The binary representation of the decimal number 4 is 100, which contains the schemata *00, 1*0, 1**, **0, *0*, and 10*.
- Some schemata are fitter and some are weaker, but by selection and reproduction, we will create a population that is full of fitter schemata (See an detailed explanation here)
- **'Implicit parallelism'**: we are not only evolving $M$ individuals but also manipulating $M \cdot 3^L$ schemata
- This essentially means that binary coding requires fewer strings to construct more schemata to sample larger search space

# Drawbacks of binary coding: Hamming cliff problem

- Hamming cliff problem: one-bit mutation can make a large (or a small) jump; a multi-bit mutation can make a small (or large) jump.
- Example:

| Genotype | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| Phenotype | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Solution to Hamming cliff problem

- Solution: Gray encoding, which is an encoding of numbers so that adjacent numbers have a single digit differing by 1.
- For $a \in \{0,1\}^L$ and $b \in \{0,1\}^L$ where $a$ is the standard binary encoded, and $b$ is Gray encoded, then

$$b_i = \begin{cases} a_i & if\ i = 1 \\ a_{i-1} \oplus a_i & if\ i > 1 \end{cases}$$

  where $\oplus$ means "exclusive or", i.e., logical operation that outputs 1 (true) only when inputs differ
- Example:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Binary encoded | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Gray encoded | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |
| Phenotype | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Drawbacks of binary coding

- Problem in discrete search spaces:
  - Redundancy problem: when the variables belongs to a finite discrete set with a cardinal different from a power of two, some binary strings are redundant, which correspond infeasible solutions
  - Example: Suppose we have a combinatorial optimisation problem whose feasible set $\mathcal{A}$ is $\mathcal{A} = 0, 2, 3$, the cardinal of the set is $|\mathcal{A}| = 3$ but we need a binary string of length of 2:

| Genotype | 00 | 01 | 10 | 11 |
| --- | --- | --- | --- | --- |
| Phenotype | 0 | **1** | 2 | 3 |

# Drawbacks of binary coding

- Problem in continuous search spaces: Precision
  - Decoding function:
    - Divide $\vec{a} \in \{0,1\}^L$ into $n$ segments of equal length $\vec{s_i} \in \{0,1\}^{\frac{L}{n}}$, $i = 1, \cdots, n$
    - Decode each segment into an integer $K_i$, $i = 1, \cdots, n$, and $K_i = \sum_{j=0}^{\frac{L}{n}} s_{i_j} \cdot 2^j$
    - Apply decoding function $h(K_i)$, i.e., map the integer linearly into the interval bound $x_i \in [u_i, v_i]$:

$$h(K_i) = u_i + K_i \cdot \frac{v_i - u_i}{2^{\frac{L}{n}} - 1}$$

  - The precision depends on $L$: might produce difficulties if the problem is large dimensional ($n$ is large) and requires great numerical precision

# Real real-valued vector representation

- For continuous optimisation problems, real-valued vector representation is a natural way to represent solutions
  - No differences between genotypes and phenotypes: only a vector of real numbers called chromosome, e.g., $\mathbf{x} = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^n$
  - Each gene in the chromosome represents a variable of the problem
  - The precision is not restricted by the decoding/encoding functions
- Evolution Strategies, Evolutionary Programming and Differential Evolution are all based on real-valued vector representation
- Advantages:
  - Simple, natural and faster: no need to encode and decode
  - Better precision and easy to handle large dimensional problems

# Real valued mutation

- Randomly select a parents with probability $p_m \in [0, 1]$ for mutation, and then randomly select a gene $c_i$ and apply mutation operator
- Real number mutation operators:
  - Uniform mutation
  - Non-uniform mutation
  - Gaussian mutation

# Real valued mutation: Uniform/Gaussian mutation

- **Uniform mutation**: replace $c_i$ with a random (uniform) number $c_i'$ generated from the interval bound of the variable $x_i \in [u_i, v_i]$
- **Gaussian mutation**: replace $c_i$ with $c_i'$ which is calculated from:

$$c_i' = \min(\max(N(c_i, \sigma_i), u_i), v_i),$$

  where $N(c_i, \sigma_i)$ is a Gaussian distribution with mean $c_i$ and standard deviation $\sigma_i$ which may depend on the length $\ell_i = v_i - u_i$ of the interval bound and typically $\sigma_i = \frac{1}{10}\ell_i$.

# Real valued mutation: Non-uniform mutation

- **Non-uniform mutation:** replace $c_i$ with $c_i'$ which is calculated from:

$$c_i' = \begin{cases} c_i + \Delta(t, v_i - c_i) & \text{if } \tau \geq 0.5 \\ c_i - \Delta(t, c_i - u_i) & \text{if } \tau < 0.5 \end{cases}$$

  where $t$ is the number of current generation and $\tau$ is a random number in the range of $[0, 1]$, and

$$\Delta(t, y) = y(1 - r^{1-\frac{t}{g_m}})^b$$

  where $r$ is a random number in the range of $[0, 1]$, $g_m$ is the maximum number of generations and $b$ is a constant
- **Question**: what is the intuition of $\Delta(t, y)$

# Real valued crossover

- Randomly select two parents $\mathbf{x_1} = \{x_1^{[1]}, x_2^{[1]}, \cdots, x_n^{[1]}\}$ and $\mathbf{x_2} = \{x_1^{[2]}, x_2^{[2]}, \cdots, x_n^{[2]}\}$, then apply a crossover operator
- Crossover operators:
  - Flat crossover
  - Simple crossover
  - Whole arithmetical crossover
  - Local arithmetical crossover
  - Single arithmetical crossover
  - BLX-$\alpha$ crossover

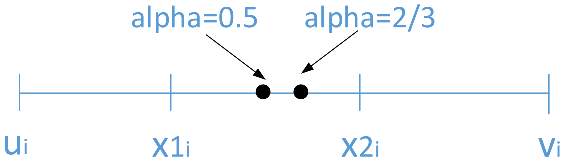
# Real valued crossover operators

- **Flat crossover:** One offspring, $\mathbf{h} = \{h_1, h_2, \cdots, h_n\}$, is generated, where $h_i$ is a randomly (uniformly) chosen value in the interval $[x_i^{[1]}, x_i^{[2]}]$ if $x_i^{[1]} < x_i^{[2]}$ or $[x_i^{[2]}, x_i^{[1]}]$ if $x_i^{[2]} < x_i^{[1]}$
- **Simple crossover:** a crossover point $i \in \{1, \cdots, n\}$ is randomly chosen, and the variables beyond this point are swap to create two new offspring:
  $\mathbf{h_1} = \{x_1^{[1]}, x_1^{[1]}, \cdots, x_i^{[1]}, x_{i+1}^{[2]}, \cdots, x_n^{[2]}\}$
  $\mathbf{h_2} = \{x_1^{[2]}, x_1^{[2]}, \cdots, x_i^{[2]}, x_{i+1}^{[1]}, \cdots, x_n^{[1]}\}$
- **Whole arithmetical crossover:** two offspring $\mathbf{h_k} = \{h_1^k, h_2^k, \cdots, h_n^k\}$, $k = 1, 2$ are generated, where $h_i^{[1]} = \alpha x_i^{[1]} + (1-\alpha) x_i^{[2]}$ and $h_i^{[2]} = \alpha x_i^{[2]} + (1-\alpha) x_i^{[1]}$ and parameter $\alpha$ is a random number in the range of $[0, 1]$

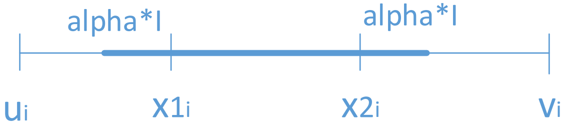
# Real valued crossover operators

- **Local arithmetical crossover:** the same as whole arithmetic crossover, except $\alpha \in \mathbb{R}^n$ is **a vector** of which each element is random number in the range of $[0, 1]$
- **Single arithmetical crossover:** choose a gene and then replace it with the arithmetic average of genes at the position of two parents, other genes are copied from the parents.
- **BLX-$\alpha$ crossover:** an offspring is generated: $\mathbf{h} = \{h_1, h_2, \cdots, h_n\}$, where $h_i$ is a randomly (uniformly) generated number of the interval $[h_{min} - I \cdot \alpha, h_{max} + I \cdot \alpha]$, $h_{max} = \max(x_i^{[1]}, x_i^{[2]})$, $h_{min} = \min(x_i^{[1]}, x_i^{[2]})$ and $I = h_{max} - h_{min}$

# Examples




Arithmetical crossover




BLX-alpha crossover

# Conclusion

- Binary coded GAs, despite its biological plausibility, are not ideal for a lot of problems
- Real valued representation is the most natural way for continuous optimisation problems
- Variation operators for real-valued coded GAs are also mutations and crossover

# Reading list


- Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 1996
- T. Baeck, D. B. Fogel, and Z. Michalewicz, Handbook on Evolutionary Computation, 1997