

# Lecture 01

Xin Yao (x.yao@cs.bham.ac.uk)

CERCIA, School of Computer Science

1. Module overview (topics, lecturers, tutors, assessment, etc.)
2. Brief Introduction to Evolutionary Computation

## Module Overview

**Lecture Plan:** See the handouts, subject to evolution

**Lecturers:** Drs. Per Kristian Lehre, Philipp Rohlfshagen,  
Thorsten Schnier and Xin Yao,

**Tutor:** Victor Landassuri-Moreno.

**Tutorials:** You are required to attend them since they are an  
integral part of the module.

**Exercises:** They are designed to give you feedback and help you  
learn. We will mark them, but not assigning any actual marks.

**Learning to learn:** Absolutely essential.

**Office Hours:** Please do check the web pages and doors.

## Assessment

**Introduction to Evolutionary Computation (22753):** Level 4.

**Normal (sessional):** 1.5 hr examination (80%) and course work (20%).

**Course Work:** Two pieces of work, worth 10% each.

**Resit (supplementary) (where allowed):** 1.5 hr examination (80%). The course work mark will be carried out.

**Evolutionary Computation (02411):** Level 3

**Normal (sessional):** 1.5 hr examination (100%)

**Resit (supplementary) (where allowed):** 1.5 hr examination (100%)

## Why Evolutionary Computation

More generally, why evolutionary computation?

- Rigid and inflexible
- Brittle
- Doesn't learn and generalise
- Need spoon-feeding
- Never grow up
- Not autonomous
- Slow
- ...
- Why should we learn their languages when they are meant to make our lives easier?

## Inspiration, Not Copying

Nature **Inspired** Computation.

## What Is Evolutionary Computation

1. It is the study of computational systems which use ideas and get inspirations from natural evolution.
2. One of the principles borrowed is *survival of the fittest*.
3. Evolutionary computation (EC) techniques can be used in optimisation, learning and design.
4. EC techniques do not require rich domain knowledge to use. However, domain knowledge can be incorporated into EC techniques.

## A Simple Evolutionary Algorithm

1. Generate the initial **population**  $P(0)$  at random, and set  $i \leftarrow 0$ ;
2. REPEAT
  - (a) Evaluate the fitness of each individual in  $P(i)$ ;
  - (b) **Select** parents from  $P(i)$  based on their fitness in  $P(i)$ ;
  - (c) **Generate** offspring from the parents using *crossover* and *mutation* to form  $P(i + 1)$ ;
  - (d)  $i \leftarrow i + 1$ ;
3. UNTIL halting criteria are satisfied

## EA as Population-Based Generate-and-Test

**Generate:** Mutate and/or recombine individuals in a population.

**Test:** Select the next generation from the parents and offsprings.

While drawing analogy to biology may be sexy, it is important to understand the underlying links between “new” AI and “old” AI (GOFAI).

## How Does a Simple EA Work

Let's use the simple EA to maximise the function  $f(x) = x^2$  with  $x$  in the *integer* interval  $[0, 31]$ , i.e.,  $x = 0, 1, \dots, 30, 31$ .

The first step of EA applications is *encoding* (i.e., the representation of chromosomes). We adopt binary representation for integers. Five bits are used to represent integers up to 31.

Assume that the population size is 4.

1. Generate initial population at random, e.g., 01101, 11000, 01000, 10011. These are *chromosomes* or *genotypes*.
2. Calculate fitness value for each individual.
  - (a) Decode the individual into an integer (called *phenotypes*),

$$01101 \rightarrow 13, 11000 \rightarrow 24, 01000 \rightarrow 8, 10011 \rightarrow 19;$$

(b) Evaluate the fitness according to  $f(x) = x^2$ ,

$$13 \rightarrow 169, 24 \rightarrow 576, 8 \rightarrow 64, 19 \rightarrow 361.$$

3. Select two individuals for crossover based on their fitness. If roulette-wheel selection is used, then

$$p_i = \frac{f_i}{\sum_j f_j}.$$

Two offspring are often produced and added to an intermediate population. Repeat this step until the intermediate population is filled. In our example,

$$p_1(13) = 169/1170 = 0.14 \quad p_2(24) = 576/1170 = 0.49$$

$$p_3(8) = 64/1170 = 0.06 \quad p_4(19) = 361/1170 = 0.31$$

Assume we have *crossover*(01101, 11000) and *crossover*(10011, 11000). We may obtain offspring 0110 0 and

1100 1 from  $crossover(01101, 11000)$  by choosing a random crossover point at 4, and obtain 10 000 and 11 011 from  $crossover(10011, 11000)$  by choosing a random crossover point at 2. Now the intermediate population is

01100, 11001, 10000, 11011

4. Apply mutation to individuals in the intermediate population with a *small* probability. A simple mutation is bit-flipping. For example, we may have the following new population  $P(1)$  after random mutation:

01101, 11001, 00000, 11011

5. Goto Step 2 if not stop.

## Different Evolutionary Algorithms

There are several well-known EAs with different

- historical backgrounds,
- representations,
- variation operators, and
- selection schemes.

In fact, EAs refer to a whole family of algorithms, not a single algorithm.

## Genetic Algorithms (GAs)

1. First formulated by Holland for adaptive search and by his students for optimisation from mid 1960s to mid 1970s.
2. Binary strings have been used extensively as individuals (chromosomes).
3. Simulate Darwinian evolution.
4. Search operators are only applied to the *genotypic* representation (chromosome) of individuals.
5. Emphasise the role of recombination (crossover). Mutation is only used as a background operator.
6. Often use roulette-wheel selection.

## Evolutionary Programming (EP)

1. First proposed by Fogel *et al.* in mid 1960s for simulating intelligence.
2. Finite state machines (FSMs) were used to represent individuals, although real-valued vectors have always been used in numerical optimisation.
3. It is closer to Lamarckian evolution.
4. Search operators (mutations only) are applied to the *phenotypic* representation of individuals.
5. It does *not* use any recombination.
6. Usually use tournament selection.

## Evolution Strategies (ES)

1. First proposed by Rechenberg and Schwefel in mid 1960s for numerical optimisation.
2. Real-valued vectors are used to represent individuals.
3. They are closer to Larmackian evolution.
4. They do have recombination.
5. They use self-adaptive mutations.

## Genetic Programming (GP)

1. First used by de Garis to indicate the evolution of artificial neural networks, but used by Koza to indicate the application of GAs to the evolution of computer programs.
2. Trees (especially Lisp expression trees) are often used to represent individuals.
3. Both crossover and mutation are used.

## Preferred Term: Evolutionary Algorithms

- EAs face the same fundamental issues as those classical AI faces, i.e., **representation**, and **search**.
- Although GAs, EP, ES, and GP are different, they are all different variants of population-based generate-and-test algorithms. They share more similarities than differences!
- A better and more general term to use is **evolutionary algorithms** (EAs).

## Variation Operators and Selection Schemes

**Crossover/Recombination:**  $k$ -point crossover, uniform crossover, intermediate crossover, global discrete crossover, etc.

**Mutation:** bit-flipping, Gaussian mutation, Cauchy mutation, etc.

**Selection:** roulette wheel selection (fitness proportional selection), rank-based selection (linear and nonlinear), tournament selection, elitism, etc.

**Replacement Strategy:** generational, steady-state (continuous), etc.

**Specialised Operators:** multi-parent recombination, inversion, order-based crossover, etc.

Key points about operators and selections?

## Major Areas in Evolutionary Computation

1. Optimisation
2. Learning
3. Design
4. Theory

## Evolutionary Optimisation

1. Numerical (global) optimisation.
2. Combinatorial optimisation (of NP-hard problems).
3. Mixed optimisation.
4. Constrained optimisation.
5. Multiobjective optimisation.
6. Optimisation in a dynamic and/or uncertain environment

## Evolutionary Learning

Evolutionary learning can be used in supervised, unsupervised and reinforcement learning.

1. Learning classifier systems (rule-based systems).
2. Evolutionary artificial neural networks.
3. Evolutionary fuzzy logic systems.
4. Co-evolutionary learning.
5. Automatic modularisation of machine learning systems by speciation and niching.

## Evolutionary Design

EC techniques are particularly good at exploring unconventional designs which are very difficult to obtain by hand.

1. Evolutionary design of artificial neural networks.
2. Evolutionary design of electronic circuits.
3. Evolvable hardware.
4. Interactive creative design using evolutionary approaches

## Evolutionary Computation Theory

It explains **how**, **when** and **why** EAs work.

## Summary

1. Evolutionary algorithms can be regarded as population-based generate-and-test algorithms.
2. Evolutionary computation techniques can be used in optimisation, learning and design.
3. Evolutionary computation techniques are flexible and robust.
4. Evolutionary computation techniques are definitely useful tools in your toolbox, but there are problems for which other techniques might be more suitable.

## Esseential Reading for This Lecture

1. D. B. Fogel (ed.), *Evolutionary Computation: The Fossil Record*, IEEE Press, Piscataway, NJ, USA, 1998. (ISBN-10: 0780334817, ISBN-13: 978-0780334816)
  - It is essential that you read the introduction and comments, if you do not have time to read all the classical papers there.
  - It is extremely useful to know the history and origin. Not only do we want to learn brilliant ideas, we also need to learn how they were first conceived and generated.
2. X. Yao, “Evolutionary computation: A gentle introduction,” In *Evolutionary Optimization*, R. Sarker, M. Mohammadian and X. Yao (eds.), Chapter 2, pp.27-53, Kluwer Academic Publishers, Boston, 2002. (ISBN 0-7923-7654-4)