OUTCOMES

At the end you should be able to discuss:
- Basics of evolutionary computation
- Representation
- At least two selection approaches
- What crossover is
- What is the link between genetic programming and genetic algorithms.

THE STEPS

In order to build an evolutionary algorithm there are a number of steps that we have to perform:
1. Design a representation
2. Decide how to initialize a population
3. Design a way of mapping a genotype to a phenotype
4. Design a way of evaluating an individual
FURTHER STEPS
5. Design suitable mutation operator(s)
6. Design suitable recombination operator(s)
7. Decide how to manage our population
8. Decide how to select individuals to be parents
9. Decide how to select individuals to be replaced
10. Decide when to stop the algorithm

DESIGNING A REPRESENTATION
- We have to come up with a method of representing an individual solution.
- There are many ways to do this and the way we choose must be relevant to the problem that we are solving.
- When choosing a representation, we have to bear in mind how the solutions will be evaluated and what the genetic operators might be.

EXAMPLE: DISCRETE REPRESENTATION (BINARY ALPHABET)
- Representation of an individual can be using discrete values (binary, integer, etc.).
- Following is an example of binary representation.
**Example: Discrete Representation (Binary Alphabet)**

8 bits Genotype: 10100011

**Phenotype:**
- Integer
- Real Number
- Schedule
  - ...
- Anything?

Phenotype: Genotype could be integer numbers

Genotype: 10100011

Phenotype: 163

\[ 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 128 + 32 + 2 + 1 = 163 \]

## Example: Discrete Representation (Binary Alphabet)

Phenotype could be a Schedule

e.g. 8 jobs, 2 time steps

Genotype: 10100011

<table>
<thead>
<tr>
<th>Job</th>
<th>Time Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Phenotype:
**EXAMPLE: REAL-VALUED REPRESENTATION**
- A very natural encoding if the solution we are looking for is a list of real-valued numbers, then encode it as a list of real-valued numbers!
- Lots of applications, e.g. parameter optimisation
- Personal example: Work on evolutionary algorithms and evoked potentials.

**EXAMPLE: ORDER BASED REPRESENTATION**
- Individuals are represented as permutations
- Used for ordering/sequencing problems
- Famous example: Travelling Salesman Problem where every city gets a assigned a unique number from 1 to \( n \). A solution could be (5, 4, 2, 1, 3).
- Needs special operators to make sure the individuals stay valid permutations.

**Example: Tree-based representation**
- Individuals in the population are trees.
- Approach leads to Genetic Programming
- These functions and terminals can be anything:
  - Functions: sine, cosine, add, sub, and, If-Then-Else, Turn...
  - Terminals: X, Y, 0.456, true, false, \( \pi \), Sensor0...
- Example: calculating the area of a circle:

\[
\pi = r^2
\]
EVALUATING AN INDIVIDUAL
- This is by far the most costly step for real applications.
  - do not re-evaluate unmodified individuals
- It might be a subroutine, a black-box simulator, or any external process
  (e.g. robot experiment)

MUTATION OPERATORS
- We might have one or more mutation operators for our representation.
- Some important points are:
  - At least one mutation operator should allow every part of the search space/pattern space to be reached.
  - The size of mutation is important and should be controllable.
  - Mutation should produce valid chromosomes (i.e. valid possible solutions).

EXAMPLE: MUTATION FOR DISCRETE REPRESENTATION

<table>
<thead>
<tr>
<th>before</th>
<th>1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>after</td>
<td>1 1 1 0 1 1 1</td>
</tr>
</tbody>
</table>

Mutation usually happens with a certain probability for each gene.
EXAMPLE: MUTATION FOR REAL VALUED REPRESENTATION

Perturb values by adding some random noise

Often, a Gaussian/normal distribution \( N(0, \sigma) \) is used, where

- \( 0 \) is the mean value
- \( \sigma \) is the standard deviation

and

\[
x'_i = x_i + N(0, \sigma_i)
\]

for each parameter

EXAMPLE: MUTATION FOR ORDER BASED REPRESENTATION (SWAP)

Randomly select two different genes and swap them.

\[
\begin{array}{cccccc}
7 & 3 & 1 & 8 & 2 & 4 \\
\end{array}
\]

\[
\begin{array}{cccccc}
7 & 3 & 6 & 8 & 2 & 4 \\
\end{array}
\]

Example: Mutation for tree based representation

Single point mutation selects one node and replaces it with a similar one.
RECOMBINATION OPERATORS

We might have one or more recombination operators for our representation. Some important points are:
- The child should inherit something from each parent.
- The recombination operator should be designed in conjunction with the representation.
- Recombination should produce valid chromosomes.

EXAMPLE: RECOMBINATION FOR DISCRETE REPRESENTATION

Whole Population:

Each chromosome is cut into n pieces which are recombined. (Example for n=1)

Parents:

```
1 1 1 1 1 1 1
0 0 0 0 0 0 0
```

Offspring:

```
1 1 1 0 0 0 0
0 0 0 1 1 1 1
```

EXAMPLE: RECOMBINATION FOR ORDER BASED REPRESENTATION (ORDER1)

- Choose an arbitrary part from the first parent and copy this to the first child.
- Copy the remaining genes that are not in the copied part to the first child:
  - starting right from the cut point of the copied part
  - using the order of genes from the second parent
  - wrapping around at the end of the chromosome
- Repeat this process with the parent roles reversed.
EXAMPLE: RECOMBINATION FOR ORDER BASED REPRESENTATION (ORDER1)

Parent 1
\[7 \ 8 \ 1 \ 8 \ 2 \ 4 \ 6 \ 5\]
Parent 2
\[4 \ 3 \ 2 \ 8 \ 6 \ 7 \ 1 \ 5\]

| 7, 3, 4, 6, 5 |
| 4, 3, 6, 7, 5 |

Child 1
\[7 \ 5 \ 1 \ 8 \ 2 \ 4 \ 3 \ 6\]

SELECTION STRATEGY

We want to have some way to ensure that better individuals have a better chance of being parents than less good individuals. This will give us selection pressure which will drive the population forward. We have to be careful to give less good individuals at least some chance of being parents - they may include some useful genetic material.

EXAMPLE: ROULETTE WHEEL

- Better (fitter) individuals have:
  - more space
  - more chances to be selected
**EXAMPLE: TOURNAMENT SELECTION**
- Select $k$ random individuals, without replacement
- Take the best
  - $k$ is called the size of the tournament

**REPLACEMENT STRATEGY**
The selection pressure is also affected by the way in which we decide which members of the population to kill in order to make way for our new individuals.

**ELITISM**
- Should fitness constantly improve?
  - Re-introduce in the population previous best-so-far (elitism) or
  - Keep best-so-far in a safe place (preservation)
RECOMBINATION VS MUTATION

- Recombination
  - modifications depend on the whole population
  - decreasing effects with convergence
  - exploitation operator
- Mutation
  - mandatory to escape local optima
  - strong causality principle
  - exploration operator

STOPPING CRITERION

- The optimum is reached!
- Limit on CPU resources:
  - Maximum number of fitness evaluations
- Limit on the user’s patience:
  - After some generations without improvement

PRACTICAL PERFORMANCE

- Never draw any conclusion from a single run
  - use statistical measures (averages, medians)
  - from a sufficient number of independent runs
ALGORITHM PERFORMANCE (2)

Remember the WYTIWYG principal:

“What you test is what you get” - don’t tune algorithm performance on toy data and expect it to work with real data.

KEY ISSUES

What you say are the six key points you should leave this session with?

Please discuss it groups.