How to Build an Evolutionary Algorithm

Brought to you by *P. Adamidis*

The EvoNet Training Committee

The Steps

In order to build an evolutionary algorithm there are a number of steps that we have to perform:

- Design a representation
- Decide how to initialise a population
- Design a way of mapping a genotype to a phenotype
- Design a way of evaluating an individual

Further Steps

- Design suitable mutation operator(s)
- Design suitable recombination operator(s)
- Decide how to manage our population
- Decide how to select individuals to be parents
- Decide how to select individuals to be replaced
- Decide when to stop the algorithm

Designing a Representation

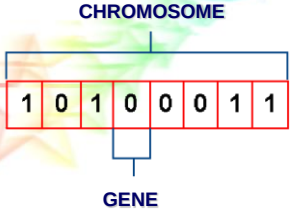
We have to come up with a method of representing an individual as a genotype.

There are many ways to do this and the way we choose must be relevant to the problem that we are solving.

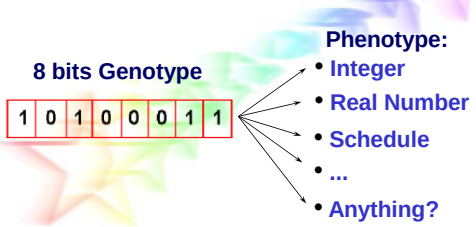
When choosing a representation, we have to bear in mind how the genotypes will be evaluated and what the genetic operators might be

Example: Discrete Representation (Binary alphabet)

- Representation of an individual can be using discrete values (binary, integer, or any other system with a discrete set of values).
- Following is an example of binary representation.



Example: Discrete Representation (Binary alphabet)



Example: Discrete Representation (Binary alphabet)

Phenotype could be integer numbers

Genotype: | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

Phenotype: **= 163**

$1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 =$

$128 + 32 + 2 + 1 = 163$

Example: Discrete Representation (Binary alphabet)

Phenotype could be Real Numbers

e.g. a number between 2.5 and 20.5 using 8 binary digits

Genotype: | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

Phenotype: **= 13.9609**

$$x = 2.5 + \frac{163}{256}(20.5 - 2.5) = 13.9609$$

Example: Discrete Representation (Binary alphabet)

Phenotype could be a Schedule

e.g. 8 jobs, 2 time steps

Genotype: | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | **=**

Job	Time Step
1	2
2	1
3	2
4	1
5	1
6	1
7	2
8	2

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! (i.e., not as a string of 1's and 0's)
- Lots of applications, e.g. parameter optimisation

Example: Real valued representation, Representation of individuals

- Individuals are represented as a tuple of n real-valued numbers:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_i \in R$$

- The fitness function maps tuples of real numbers to a single real number:

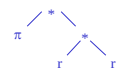
$$f : R^n \rightarrow R$$

Example: Order based representation

- Individuals are represented as permutations
- Used for ordening/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.
- Any S-expression can be drawn as a tree of functions and terminals.
- These functions and terminals can be anything:
  - Functions: sine, cosine, add, sub, and, If-Then-Else, Turn...
  - Terminals: X, Y, 0.456, true, false, π, Sensor0…
- Example: calculating the area of a circle:

$\pi * r^2$ 

Example: Tree-based representation, Closure & Sufficiency

- We need to specify a function set and a terminal set. It is very desirable that these sets both satisfy closure and sufficiency.
- By closure we mean that each of the functions in the function set is able to accept as its arguments any value and data-type that may possible be returned by some other function or terminal.
- By sufficient we mean that there should be a solution in the space of all possible programs constructed from the specified function and terminal sets.

Initialization

- Uniformly on the search space … if possible
  - Binary strings: 0 or 1 with probability 0.5
  - Real-valued representations: Uniformly on a given interval (OK for bounded values only)
- Seed the population with previous results or those from heuristics. With care:
  - Possible loss of genetic diversity
  - Possible unrecoverable bias

Example: Tree-based representation

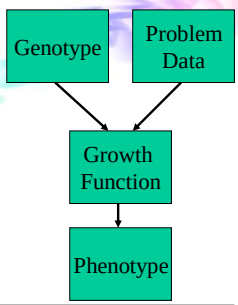
- Pick a function *f* at random from the function set *F*. This becomes the root node of the tree.
- Every function has a fixed number of arguments (unary, binary, ternary, …. , *n*-ary), *z(f)*. For each of these arguments, create a node from either the function set *F* or the terminal set *T*.
- If a terminal is selected then this becomes a leaf
- If a function is selected, then expand this function recursively.
- A maximum depth is used to make sure the process stops.

Example: Tree-based representation, Three Methods

- The Full grow method ensures that every non-back-tracking path in the tree is equal to a certain length by allowing only function nodes to be selected for all depths up to the maximum depth - 1, and selecting only terminal nodes at the lowest level.
- With the Grow method, we create variable length paths by allowing a function or terminal to be placed at any level up to the maximum depth - 1. At the lowest level, we can set all nodes to be terminals.
- Ramp-half-and-half create trees using a variable depth from 2 till the maximum depth. For each depth of tree, half are created using the Full method, and the other half are created using the Grow method.

Getting a Phenotype from our Genotype

- Sometimes producing the phenotype from the genotype is a simple and obvious process.
- Other times the genotype might be a set of parameters to some algorithm, which works on the problem data to produce the phenotype



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)
- You could use approximate fitness - but not for too long

More on Evaluation

- Constraint handling - what if the phenotype breaks some constraint of the problem:
  - penalize the fitness
  - specific evolutionary methods
- Multi-objective evolutionary optimization gives a set of compromise solutions

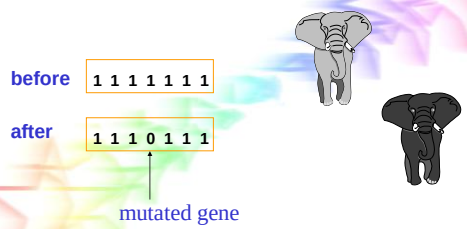
Mutation Operators

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 to be reached
- The size of mutation is important and should be controllable.
- Mutation should produce valid chromosomes

Example: Mutation for Discrete Representation



Mutation usually happens with probability p_m for each gene

Example: Mutation for real valued representation

Perturb values by adding some random noise

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

- 0 is the mean value
- σ is the standard deviation

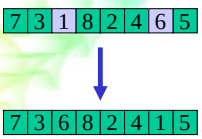
and

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

for each parameter

Example: Mutation for order based representation (Swap)

Randomly select two different genes and swap them.



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.

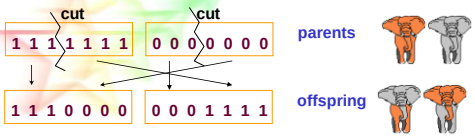
Some important points are:

- The child should inherit something from **each** parent. If this is not the case then the operator is a mutation operator.
- The recombination operator should be designed in conjunction with the representation so that recombination is not always catastrophic
- Recombination should produce valid chromosomes

Example: Recombination for Discrete Representation

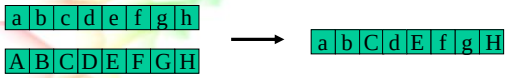
Whole Population:

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



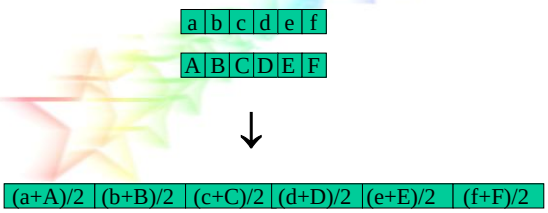
Example: Recombination for real valued representation

Discrete recombination (uniform crossover): given two parents one child is created as follows



Example: Recombination for real valued representation

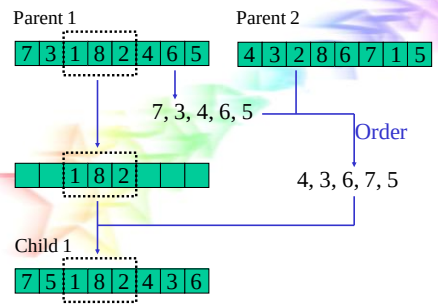
Intermediate recombination (arithmetic crossover): given two parents one child is created as follows



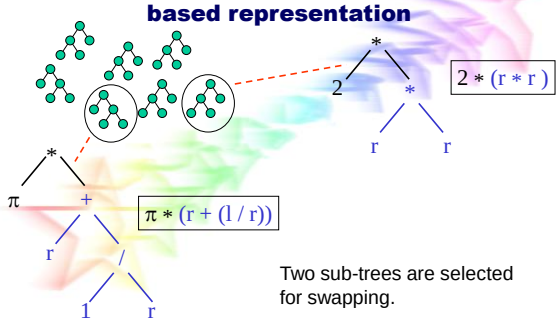
Example: Recombination for order based representation (Order1)

- 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

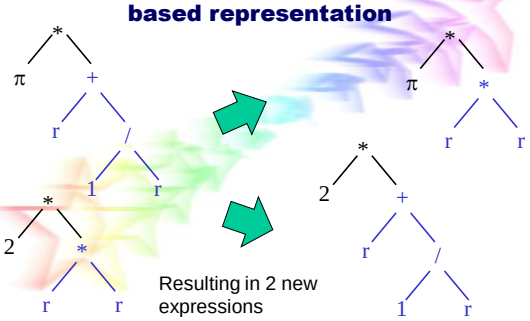
Example: Recombination for order based representation (Order1)



Example: Recombination for tree-based representation



Example: Recombination for tree-based representation



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: Fitness proportionate selection

- Expected number of times f_i is selected for mating is: $f_i / \bar{f}$
- Better (fitter) individuals have:
  - more space
  - more chances to be selected

Example: Fitness proportionate selection

Disadvantages:

- Danger of premature convergence because outstanding individuals take over the entire population very quickly
- Low selection pressure when fitness values are near each other
- Behaves differently on transposed versions of the same function

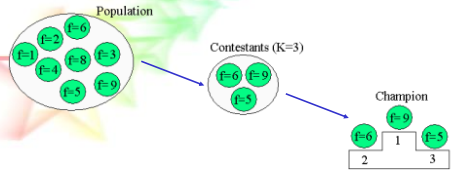
Example: Fitness proportionate selection

Fitness scaling: A cure for FPS

- Start with the raw fitness function ***f***.
- Standardise to ensure:
  - Lower fitness is better fitness.
  - Optimal fitness equals to 0.
- Adjust to ensure:
  - Fitness ranges from 0 to 1.
- Normalise to ensure:
  - The sum of the fitness values equals to 1.

Example: Tournament selection

- Select ***k*** random individuals, without replacement
  - ***k*** is called the size of the tournament
- Take the best



Example: Ranked based selection

- Individuals are sorted on their fitness value from best to worse. The place in this sorted list is called rank.
- Instead of using the fitness value of an individual, the rank is used by a function to select individuals from this sorted list. The function is biased towards individuals with a high rank (= good fitness).

Example: Ranked based selection

- Fitness: *f(A) = 5, f(B) = 2, f(C) = 19*
- Rank: *r(A) = 2, r(B) = 3, r(C) = 1*

$$h(x) = \min + (\max - \min) * \frac{(r(x) - 1)}{n - 1}$$

- Function: *h(A) = 3, h(B) = 5, h(C) = 1*
- Proportion on the roulette wheel:
  p(A) = 11.1%, p(B) = 33.3%, p(C) = 55.6%

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.

We can use the stochastic selection methods in reverse, or there are some deterministic replacement strategies.

We can decide never to replace the best in the population: elitism.

Elitism

- Should fitness constantly improves?
  - Re-introduce in the population previous best-so-far (elitism) or
  - Keep best-so-far in a safe place (preservation)
- Theory:
  - GA: preservation mandatory
  - ES: no elitism sometimes is better
- Application: Avoid user's frustration

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

Recombination vs Mutation (2)

- Historical “irrationale”
  - GA emphasize crossover
  - ES and EP emphasize mutation
- Problem-dependent rationale:
  - fitness partially separable?
  - existence of building blocks?
  - Semantically meaningful recombination operator?

Use recombination if useful!

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

Algorithm performance

- Never draw any conclusion from a single run
  - use statistical measures (averages, medians)
  - from a sufficient number of independent runs
- From the application point of view
  - design perspective:
    find a very good solution at least once
  - production perspective:
    find a good solution at almost every run

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

Genetic diversity

- differences of genetic characteristics in the population
- loss of genetic diversity = all individuals in the population look alike
- snowball effect
- convergence to the nearest local optimum
- in practice, it is irreversible

Key issues (2)

Exploration vs Exploitation

- Exploration =sample unknown regions
- Too much exporation = random search, no convergence

- Exploitation = try to improve the best-so-far individuals
- Too much expoitation = local search only … convergence to a local optimum

Island Model