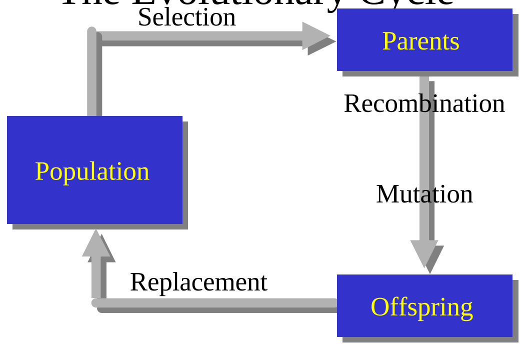
How to Build an Evolutionary Algorithm

The Evolutionary Cycle



The Steps

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

Designing a Representation

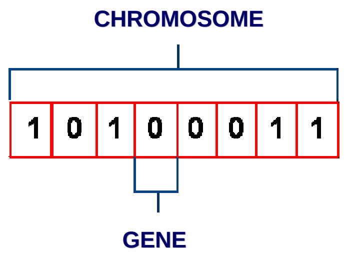
We have to come up with a method of representing an individual as a genotype.

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

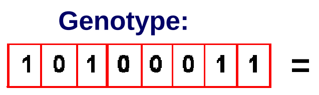
Example: Discrete Representation (Binary alphabet)

- 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)

Phenotype could be a Schedule
e.g. 8 jobs, 2 time steps



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

Phenotype

A 0 in the chromosome means time step 1, a 1 means time step 2

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 might map tuples of real numbers to a single real number:

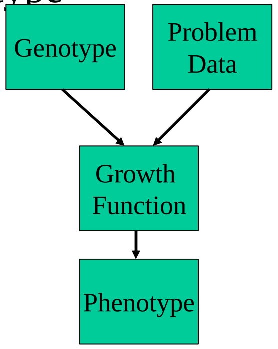
$$f : R^n \rightarrow R$$

Initialization

- Uniformly on the search space
  - 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:
  - Could be very useful
  - Possible loss of genetic diversity
  - Possible unrecoverable local optimum

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



Indirect Representation Example Knapsack Problem

We might have an algorithm which tries to put each items into the knapsack in turn. If it fits without going overweight then it stays in the knapsack. If it doesn't then it is discarded and the next item is considered.

The chromosome could tell us in which order to consider the items:

E.g.

3	**6**	**4**	**8**	**7**	**1**	**2**	**5**	**9**	**0**

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:

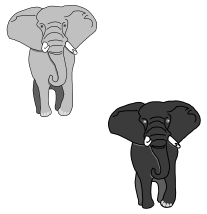
- At least one mutation operator should allow every part of the search space to be reached
- The size of mutation is important and should be controllable
- Mutation should produce valid chromosomes
- Heuristic mutations might be used

Example: Mutation for Discrete Representation

before 1 1 1 1 1 1 1

after 1 1 1 0 1 1 1

mutated gene



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 $N(0,\sigma)$ 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

Mutation Rates

- The rate of mutation is a parameter that it is important to get right.
- Many people allow every gene in the chromosome an equal chance of mutation.
- Often this chance is 1/*l* where *l* is the number of genes in the chromosome
- When writing about mutation rates is important to define *exactly* what you mean

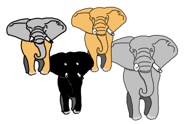
Recombination Operators

We might have one or more recombination operators for our representation.

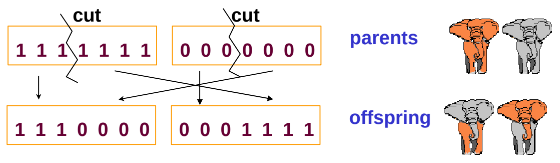
Some important points are:

- 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
- Heuristic recombination might be used

Example: Recombination for Discrete Representation

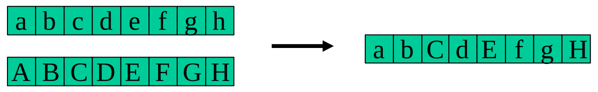
Whole Population:  . . .

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

a	b	c	d	e	f

A	B	C	D	E	F

⎯

(a+A)/2	(b+B)/2	(c+C)/2	(d+D)/2	(e+E)/2	(f+F)/2

This method has problems due to a pressure towards the “middle” of the search space

Selection Strategy

We want:

- better individuals to have a better chance of being parents than less good individuals.

This will give us *selection pressure* which will drive the population forward.

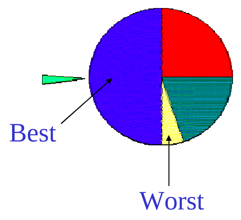
We must be careful:

- less good individuals must have 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



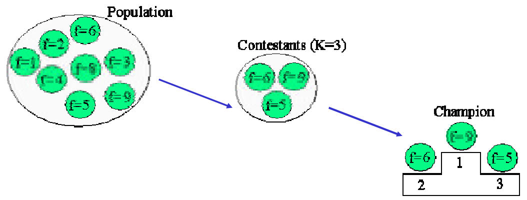
Example: Fitness proportionate selection

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: Tournament selection

- Select ***k*** random individuals from the population
- Take the best (perhaps with some probability)
  - ***k*** is called the size of the tournament



Population Strategy

The selection pressure is also affected by the way in which we manage the population:

How do we decide which individuals to kill to make way for new ones?
Is the population size fixed?
Do we allow duplicates in the population?
Should we always keep the best in the population? (Elitism)
Should we replace the whole population at once? (generational)
Or just one individual at a time? (steady-state),
Or something else?
Do we select between the children?
Do we compare parents and children?

Replacement Strategy

Our population strategy may require us to choose individuals to kill.

We can use the stochastic selection methods in reverse, or there are some deterministic replacement strategies, for example, *replace worst* or *replace oldest.*

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

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 principle:

"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

Exercise

1. An algorithm uses a tournament selection method. If the size of the tournament is increased, will the selection pressure increase or decrease?
2. An algorithm uses a tournament replacement method. If the size of the tournament is increased, will the selection pressure increase or decrease?
3. How will population size affect the selection pressure?
4. The equal grouping problem involves splitting a number of items into a number of groups in such a way that the collective weight of the items in each group is as equal as possible. The groups can contain different numbers of items. For this exercise we will split the items into 10 groups. The measure of how good a candidate solution is will be the difference in weight between the heaviest group and the lightest group - this value should be minimised. Design a suitable representation and operators for an evolutionary algorithm to solve this problem. If you have time, think about how the population might be seeded, or other algorithms that could be hybridised with the EA.