IMPERIAL

GLOBAL OPTIMISATION METHODS

- ☐ **Gerardium Rush** – Monday Briefing
- ☐ Thomas Davison thomas.davison@imperial.ac.uk

- ☐ Slides by Pablo Brito-Parada p.brito-parada@imperial.ac.uk

Optimisation Problems

A generic maximisation (or minimisation) problem involves having an **objective function** that is dependent on a number of **input variables** for which a maximum (or minimum) value is required

Optimisation Algorithms

Optimisation algorithms can be classified in different ways

i. Gradient-based or Gradient-free

ii. Deterministic or Stochastic

iii. Trajectory-based or Population-based

iv. Local or Global

The classification is not always straightforward as hybrid algorithms are not uncommon

Optimisation Algorithms

Optimisation algorithms can be classified in different ways

i. Gradient-based or Gradient-free

ii. Deterministic or Stochastic

iii. Trajectory-based or Population-based

iv. Local or Global

The classification is not always straightforward as hybrid algorithms are not uncommon

Optimisation Algorithms

Optimisation algorithms can be classified in different ways

i. Gradient-based or **Gradient-free**

ii. Deterministic or **Stochastic**

iii. Trajectory-based or **Population-based**

iv. Local or **Global**

The classification is not always straightforward as hybrid algorithms are not uncommon

Gradient-based methods

- For objective functions that are continuous functions of the input variables, **gradient search methods** are often employed
  - *Seen these before in Inversion & Optimisation*
  - *E.g. conjugate gradient methods*

- For problems where the input variables are either discrete and/or the objective function is discontinuous, gradient search methods are not appropriate and **gradient-free methods** (suitable for global optimisation) can be used instead
  - *Very flexible*
  - *Tolerant to noise in the function*
  - *Can be much slower*

Global methods

- A global optimisation algorithm is intended to locate a global optima. It explores the entire input search space and aims at getting close to (or finding exactly) the extrema of the function.
- The difficulty associated with global optimisation:
  - Easy to locate local optima
  - More difficult to find global optima
- Expensive to use brute force to search the entire space for global optima
  - *Simple*
  - *Most inefficient method*
- Iterative generation of a candidate solution or a population of candidate solutions
  - *Generally looking for increase of objective function, but often allow a decrease too*

    → **High computational cost**

Classification of global methods

- These methods can be classified into ***exact methods*** and ***heuristics*****.**
- An **exact** (or deterministic) method guarantees to find and verify global solutions.
- Otherwise, it is called a **heuristic** (or meta-heuristic).
- Heuristics try to find global solutions without verifying global optimality – they involve a set of rules
- They can be used as stand-alone solvers or as an acceleration tool in deterministic methods.

Meta-heuristics

- Recent literature tends to refer to algorithms with stochastic components, as well as to all modern nature-inspired algorithms, as meta-heuristic
  - *Heuristic* → finding/discovering by trial and error
  - *meta-* → beyond or higher level
    - i.e. meta-heuristics perform better than simple heuristics
- A trade-off of randomization and local search is used

Meta-heuristics guide and modify

- A top-level strategy that guides and modifies an underlying or subordinate heuristic
  - *Produce solutions beyond those normally generated when looking locally*

- Iterative generation process
  - *Combines **exploration** and **exploitation** of the search spaces*

- Uses learning strategies, a guiding process and an application process, to structure information so that near-optimal solutions can be found efficiently
  - *Guiding process decides on possible moves; application process executes those moves*

Meta-heuristics: two key components

- Exploration / Diversification
  - *Generate diverse solutions via randomisation to better explore the search space*

- Exploitation / Intensification
  - *If a current good solution is found in a local region, the region becomes the focus of the search*

A balance of the two is essential and determines convergence rates

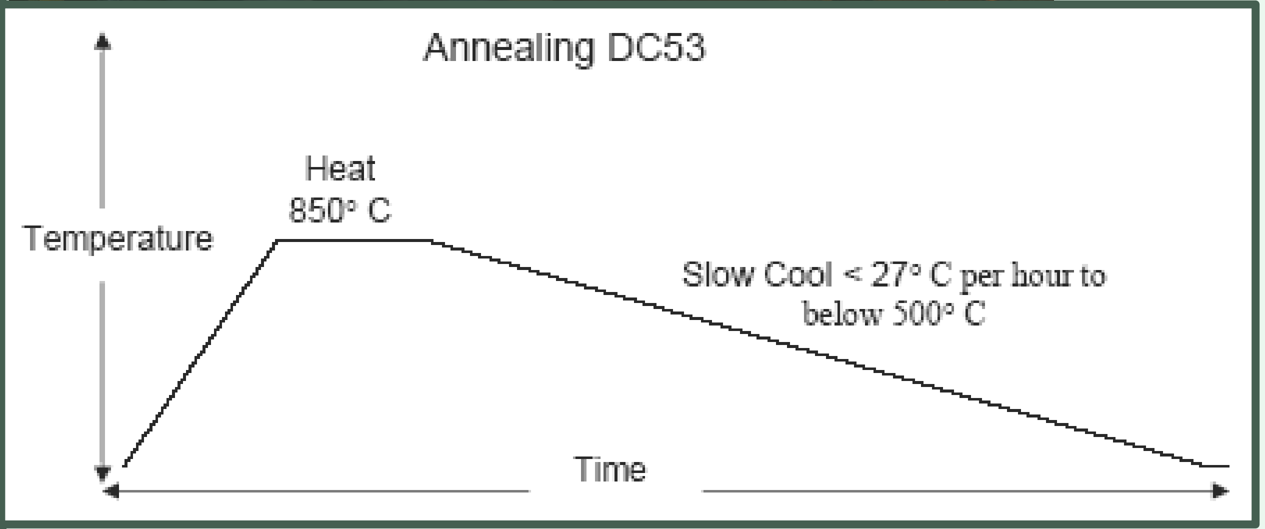
Examples of meta-heuristic search strategies:

- Important applications in engineering, design optimisation, economics, and many more…
- Example of meta-heuristic search strategies include:
  - *Simulated annealing*
  - Memetic algorithms
  - Particle swarm optimisation
  - *Genetic algorithms*

+ An ever-increasing number of “nature inspired” algorithms
  - *ant colony*
  - *artificial bee colony*
  - *firefly*
  - *cuckoo search*

SIMULATED ANNEALING

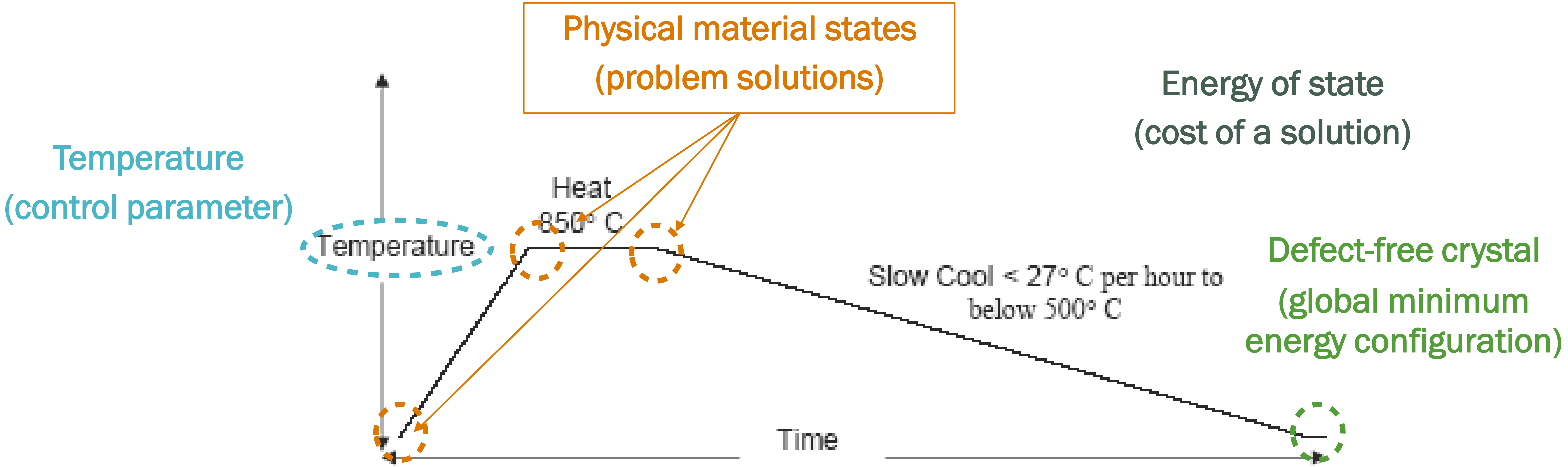
Annealing



- Annealing is a metallurgical process in which a material is heated above its recrystallization temperature for a certain amount of time, and then is slowly cooled.
- The slow cooling brings the metal to a crystalline state, altering the physical/chemical properties to increase its ductility and reduce its hardness.
- Settles to lower energy state (the minimum energy state)
  - *Increases size of crystals*
  - *Reduces defects*

Simulated annealing analogy

- In *simulated annealing*, points of a sample set are modified by applying a descent step with a random search direction and an initially large step-size that is gradually decreased during the optimisation process.
- Typically used in discrete, but very large, configuration spaces.



Simulated annealing process

- Take an initial guess. Then randomly generate new solution near initial solution.
- At each iteration, a new point is randomly generated. The extent of the search is based on a probability distribution (with a scale proportional to the "temperature").
- Importantly, the algorithm always accepts new points that minimise the objective function but also, with a certain probability, points that do not, in order to avoid being trapped in local minima.
- An *annealing schedule* is selected to systematically decrease the temperature as the algorithm proceeds. As the temperature decreases, the algorithm reduces the extent of its search to converge to a minimum.

Simulated annealing analogy comparison

	THERMODYNAMICS	OPTIMISATION ANALOGY
■ At high *T*	The system ignores small changes in the energy and approaches thermal equilibrium rapidly	It performs a coarse search of the space of global states and finds a good minimum
■ As *T* is lowered	The system responds to small changes in the energy	It performs a fine search in the neighbourhood of the already determined minimum and finds a better minimum
■ At *T* = 0	The system must reach equilibrium	The best minimum is accepted as the global minimum

Simulated annealing

- Theoretically, a high probability to reach a global minimum
- *T* is used as a control parameter, the *computational temperature*. It controls the magnitude of the perturbations of the energy function *E*(***x***).
- The probability of a state change is determined by the Boltzmann distribution of the energy difference of the two states:

$$\boxed{P = e^{-\frac{\Delta E}{T}}}$$

ΔE	– change in objection function
T	– computational temperature
P	– acceptance probability

- Negative ΔE means a lower objective function value, so $P > 1$, and the solution is always accepted.
- Positive ΔE implies a worse solution. In this case, the probability of accepting this solution is large at high T , and is low at low T.

Simulated annealing algorithm

```
1. Initialize the system configuration.
   Randomize $\boldsymbol{x}(0)$.

2. Initialize $T$ with a large value.

3. **Repeat**:
   a. **Repeat**:
      i. Apply random perturbations to the state $\boldsymbol{x} = \boldsymbol{x} + \Delta\boldsymbol{x}$.
      ii. Evaluate $\Delta E(\mathbf{x}) = E(\mathbf{x} + \Delta\mathbf{x}) - E(\mathbf{x})$:
         **if** $\Delta E(\mathbf{x}) < 0$, keep the new state;
         **otherwise**, accept the new state with probability $P = e^{-\Delta E/T}$.
      **until** the number of accepted transitions is below a threshold level.
   b. Set $T = T - \Delta T$ .
**until** $T$ is small enough.
```

Note that a simulated annealing algorithm can result in different local minima

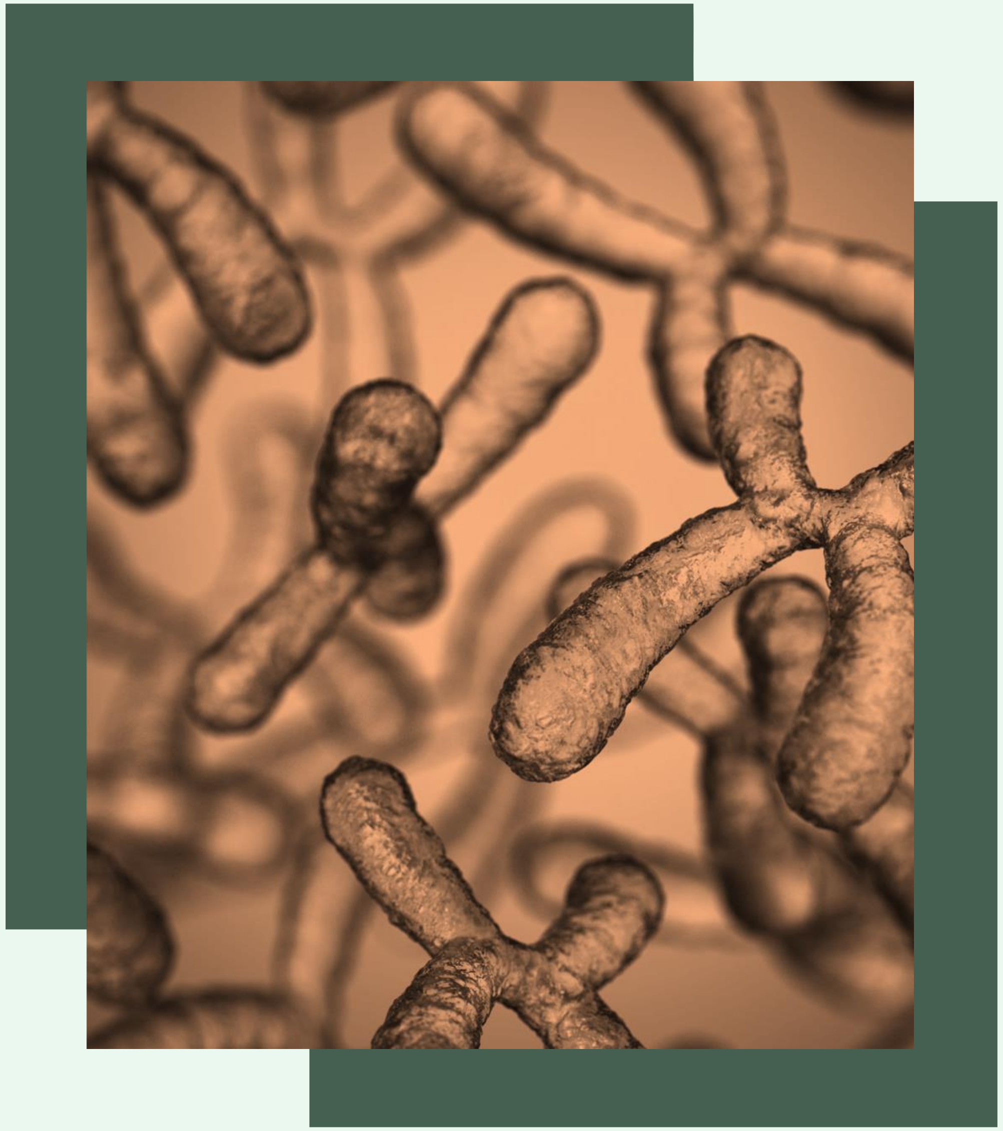
→ a problem needs to be run several times before the solution can be accepted as the global optimum.

Simulated annealing – cooling schedule

- The cooling schedule is critical in terms of efficiency
  - *If T is decreased too rapidly – potential of converging to local minimum*
  - *If T is decreased too slowly – very slow convergence*
- Two common ways to control the cooling rate are linear and geometric annealing schedules
- Some studies have shown that a logarithmic decrease almost ensure convergence to the global optimum:

$$\boxed{T(t) \geq \frac{T_0}{\ln(1+t)}, \qquad t = 1, 2, \ldots}$$

but in practice this is usually too slow, and faster schedules are used instead



GENETIC ALGORITMS

Genetic Algorithms

- Based on the principles derived from the dynamics of natural population genetics.
- Most popular evolutionary algorithms.
- Population-based and the basis for many modern evolutionary algorithms.
- Applications range from graph colouring to pattern recognition, from discrete systems (such as the travelling salesman problem) to continuous systems (e.g., the efficient design of airfoil in aerospace engineering), and from financial markets to multiobjective engineering optimization.

Genetic Algorithms – Basics

- Our problem is a discrete one, in that the variables are the connections between the units
  - Units are either connected to one another or not, thus discrete

- A popular approach to problems with discrete variables
  - *Note that the method can also be used with continuous variables (either some or all the variables are continuous)*
  - *But if all input variables and the objective function are continuous then gradient search methods are likely to perform better*

- Encoding of solutions as arrays of bits or character strings (≈ chromosomes)
- Manipulation of strings by genetic operators and a selection based on their fitness to find a solution to a given problem.

Genetic Algorithms – Population of solutions

- Instead of proposing a single solution to an optimisation problem, a genetic algorithm generates many possible solutions that form a population.

- Then solutions are scored using a fitness function (objective function) to decide which solutions are better than others.

- These candidate solutions are recombined so that the best solutions reproduce to create a new generation of solutions with the best traits of the previous solutions.

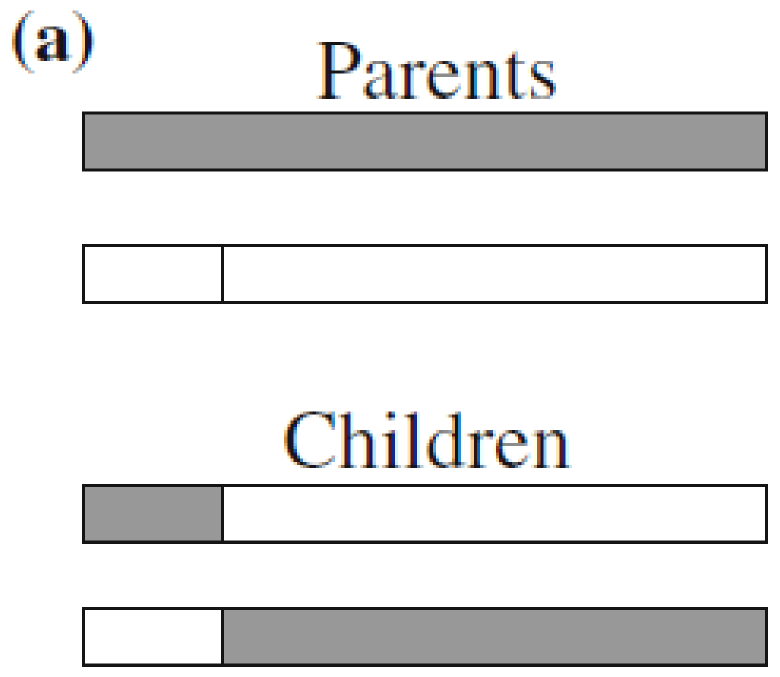
- This continues until improvement stops or the max number of generations is reached.

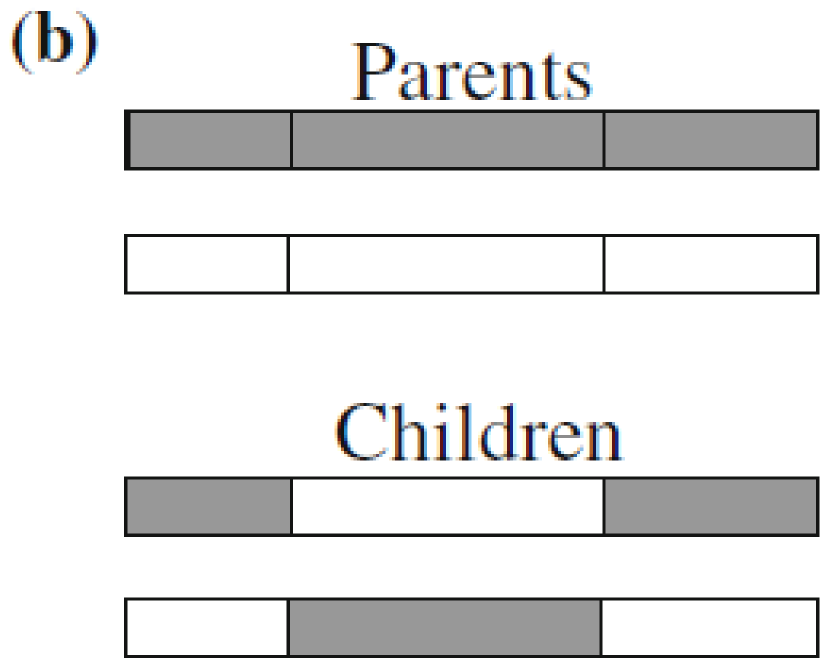
Genetic Algorithms – Producing Children

- A Genetic Algorithm produces “child” vectors from a list of “parent” vectors

- Done using two operations:
  - *Crossover*
    - Roughly equivalent to sexual reproduction.
    - A portion of one parent vector is swapped with a portion of another parent vector to produce two child vectors
    - Over successive generations, values that work well together will end up next to one another in the vector (roughly equivalent to genes)
  - *Mutations*
    - Random changes in the numbers in the vector.

Genetic Algorithms – Crossover





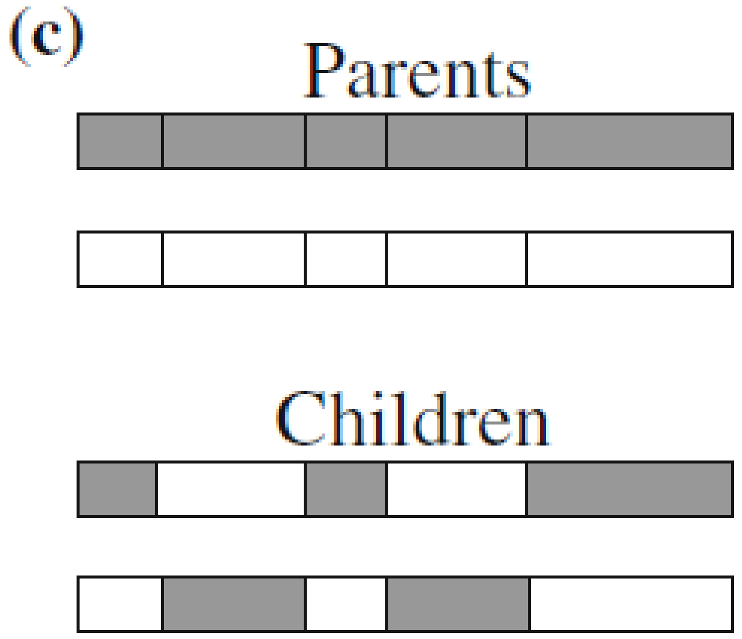
One-point crossover

- Easy to model
- Bias at ends

Two-point crossover

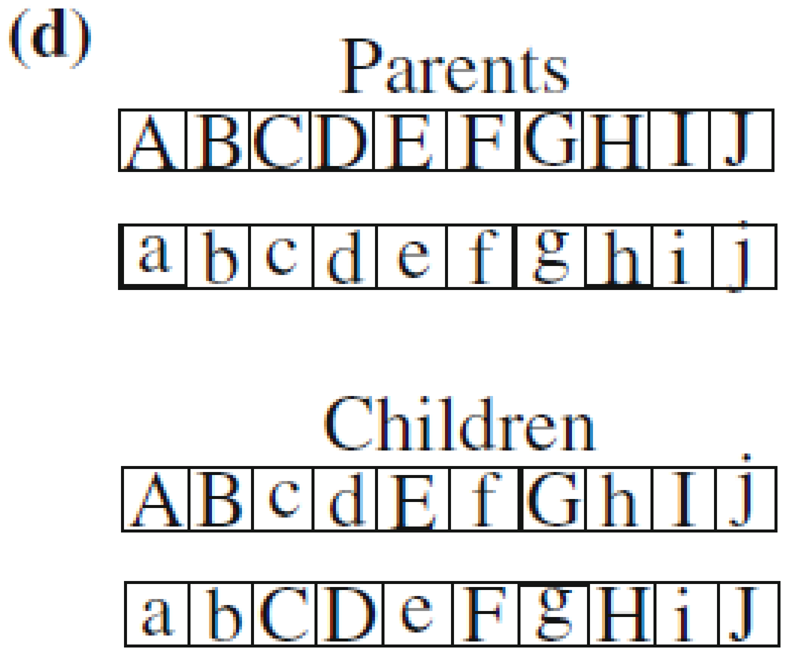
- Better than one-point
- Ends rarely sampled
  - *Unless wrap around at ends*

Genetic Algorithms – Crossover (cont.)



Multipoint crossover

- Each string is a ring with m crossover points
- Less bias than one/two point, but breaks up regions more



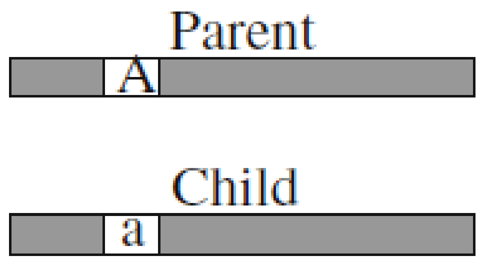
Uniform crossover

- Unbiased
- More exploratory than one/two-point
- Highly disruptive nature

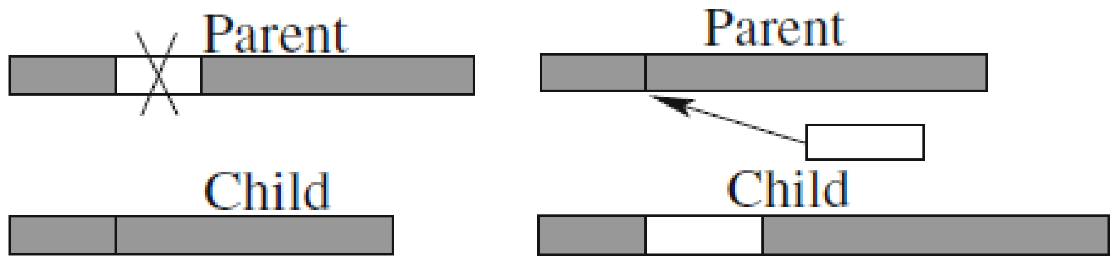
Genetic Algorithms – Mutation

- Using crossover alone can lead to premature convergence
- Introducing a mutation can lead to more diversity in the population.

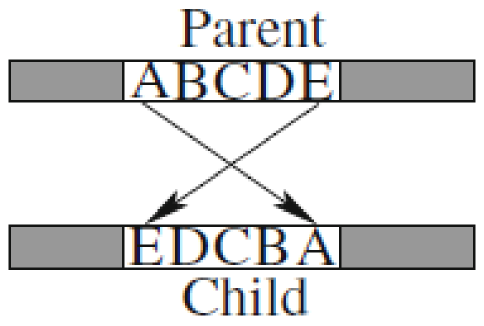
(a) Substitution



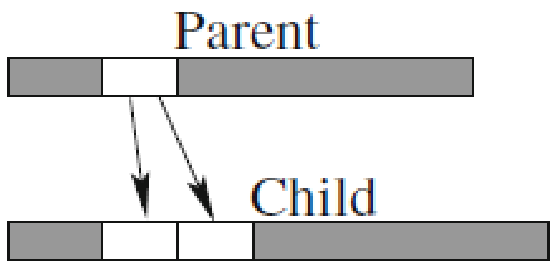
(b) Deletion + Insertion



(c) Inversion / Rearrangement



(d) Duplication (not used in this project)



Genetic Algorithms – Selection of fittest

- Selection of the fittest is carried out according to the fitness of the chromosomes.
- To ensure the best chromosomes remain in the population, they are transferred to the next generation without much change (called elitism).
- A proper criterion for selection is very important. It determines how chromosomes with higher fitness are preserved and transferred to the next generation (with a degree of elitism).

- The basic form is to select the best chromosome (in each generation) which will be carried over to the new generation without being modified by the genetic operators. This ensures that a good solution is attained more quickly.

Genetic Algorithms – General procedure

1) Definition of an encoding scheme [Circuit vector]
2) Definition of a fitness function [Evaluate circuit performance]
3) Creation of a population of chromosomes [Parent vectors]
4) Evaluation of the fitness of every chromosome in the population
5) Creation of a new population (children) by performing fitness-proportionate crossover and mutation (also possibly selection and recombination)
6) Replacement of the old population by the new one.

Steps 4), 5) and 6) are then repeated for a number of generations.

At the end, the best chromosome [vector] is decoded to obtain a solution to the problem.

Genetic Algorithms – Specifics for Gerardium Rush

Starting the calculations –

- We need to start with a list of random parent vectors
  - *How many is best is a tuning parameter – I recommend trying it with around 100 parents, but you should test for what is best*
    - It will depend on both the type and size of problem being optimised

- These vectors should all be for valid circuits
  - *Many vectors will result in circuits that are invalid for various reasons*
    - James talked about this earlier

Genetic Algorithms – Calculation Steps

Main calculation loop is as follows:

1) Start with the vectors representing the initial random collection of valid circuits

2) Calculate the fitness value for each of these vectors

You now wish to create *n* child vectors

3) Take the best vector (the one with the highest fitness value) into the child list unchanged (you want to keep the best solution)

Genetic Algorithms – Calculation Steps (cont.)

4) Select a pair of the parent vectors
   - *Probability of selection depends on the fitness value.*
   - *You might want to start by using a probability that varies linearly between the minimum and maximum finesses of the current population.*
   - *Done "with replacement", which means parents can be selected more than once.*

5) Randomly decide if the pair of parents should crossover.
   - *If they don't crossover, they both go to the next step unchanged.*
   - *If they are to cross, a random point in the vector is chosen; all the values before that point are swapped with the corresponding points in the other vector.*

Genetic Algorithms – Calculation Steps (cont.)

6) Go over each of the numbers in both the vectors and decide whether to mutate them (this should be quite a small probability). If the value is to be mutated, you should move the value by a random amount
   - *As there is no reason why a connection to a unit with a similar number should be favoured you should choose a completely random new number in this step for the actual simulation*

Genetic Algorithms – Calculation Steps (cont.)

7) Check the validity of each of these potential new vectors and, if they are valid, add them to the list of child vectors

8) Repeat this process from step 4 until there are *n* child vectors

9) Replace the parent vectors with these child vectors and repeat the process from step 2 until either a set number of iterations have been completed or a threshold has been met (e.g. the best vector has not changed for a sufficiently large number of iterations).

Tuneable Parameters

There are a few parameters you can tune in a genetic algorithm (note the recommended parameters below are for the main problem):

- The number of offspring *n* that are evaluated in each generation
  - *Try values of order 100*
- The probability of crossing selected parents
  - *Try values between 0.8 and 1.0*
- The rate at which mutations are introduced
  - *Values should be less than 1% per item in the vector*

Final remarks:

Global optimisation / gradient-free

- Easy to implement, in that they don't require derivatives!
  - *Can be applied to problems that are discrete, discontinuous or non-differentiable*
- Slow for large problems
- No guarantee of optimal solution since they are stochastic
- Several versions available
- A large amount of research

GLOBAL OPTIMISATION METHODS

With thanks to Pablo Brito-Parada
p.brito-parada@imperial.ac.uk