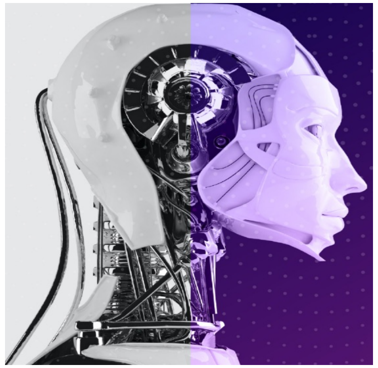
Evolutionary Algorithms I



Lecture 5

Ender Özcan

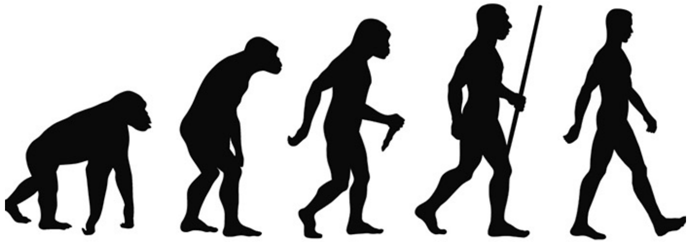
Computational Optimisation & Learning Lab

The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Evolutionary is Revolutionary

- Nature as a Problem Solver: 4.55 Billion years of evolution can’t be wrong.



Evolution: Gradual change in the inherited characteristics of a population of animals or plants over successive generations

- Beauty-of-nature argument: Complexity achieved in *short* time in nature.
- Can we solve complex problems as quickly and reliably on a computer?

Evolution at Work

- Heritable characteristics or heritable traits, e.g., the colour of your eyes are passed from one generation to the next via DNA (a molecule that encodes genetic information)
- Change or genetic variation comes from:
  - Mutations: changes in the DNA sequence,
  - Crossover: reshuffling of genes through sexual reproduction and migration between populations
- Evolution is driven by natural selection – survival of the fittest
- Genetic variations that enhance survival and reproduction become and remain more common in successive generations of a population.

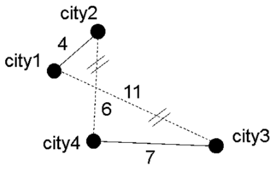
A famous example – peppered moth evolution

white/black-bodied peppered moth

- Before the Industrial Revolution, the black peppered moth was rare.
  - The frequency of the dark allele was about 0.01%
- The rapid industrialization and rampant coal use coated the British trees in a layer of soot
- By the mid-19th century, the number of dark-coloured moths had risen noticeably, and by 1895, the frequency was 98% in Manchester

Evolutionary Algorithms (EAs)

- EAs simulate natural evolution (Darwinian Evolution) of **individual** structures at the genetic level using the idea of ***survival of the fittest*** via processes of **selection**, **mutation**, and **reproduction (recombination)**
- An ***individual*** (***chromosome***) represents a candidate solution for the problem at hand. (e.g., <2 1 3 4>)

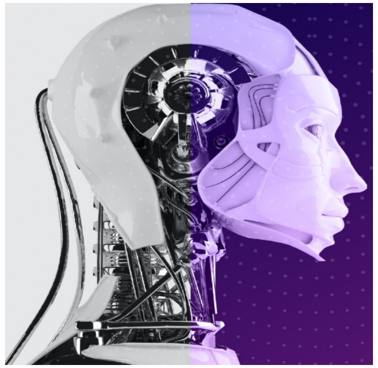


- A collection of individuals currently "alive", called **population** (set of individuals/chromosomes) is evolved from one **generation** (**iteration**) to another depending on the **fitness** of individuals in a given *environment*, indicating how fit an individual is, (how close it is to the *optimal* solution) – <u>objective value</u>. (e.g., *f*(<2 1 3 4>)= 28)
- **Hope**: Last generation will contain the best solution

Evolutionary Algorithms (EAs) II

- **Genetic Algorithms** (evolves (bit) strings)⇒Nils Aall Barricelli 1954, Holland 1975
  - Memetic Algorithms ⇒ Moscato 1989
- **Evolutionary Programming** (evolves parameters of a program with a fixed structure) ⇒ Fogel, Owens, Walsh 1966
- **Evolution Strategies** (vectors of real numbers) ⇒ Rechenberg 1973
- **Genetic Programming** (evolves computer programs in tree form) ⇒ Koza 1992
  - **Gene Expression Programming** (computer programs of different sizes are encoded in linear chromosomes of fixed length)
  - **Grammatical Evolution** (evolves solutions wrt a specified grammar) ⇒ Ryan, Collins and O'Neill 1998

Genetic Algorithms (GAs)



Pseudocode of a Generic Genetic Algorithm

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

Basic Components of GAs

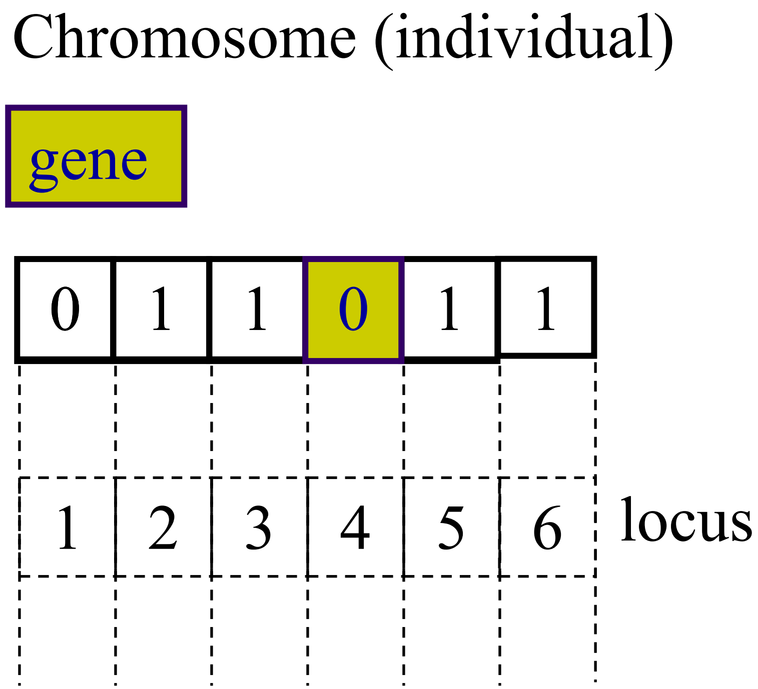
- A genetic **representation (encoding)** for candidate solutions (individuals) to the problem at hand
- An **initialisation** scheme to generate the first population (set) of candidate solutions (individuals)
- A **fitness (evaluation) function** that plays the role of the environment, rating the solutions in terms of their fitness
- A scheme for **selecting mates (parents)** for recombination
- **Crossover (recombination)** exchanges genetic material between mates producing **offspring** (**children**)
- **Mutation** perturbs an individual creating a new one
- **Replacement strategy** to select the surviving individuals for the next generation
- **Termination Criteria**
- Values for various parameters that GA uses (population size, probabilities of applying genetic operators, etc)

GA Components: Representation

- **Haploid structure** is used: Each individual contains one chromosome
- Each individual is evaluated and has an associated **fitness** value
- Chromosomes contain a fixed number of genes: **chromosome length**
- Traditionally **binary encoding** is used for each gene: **Allele** value $\in \{0,1\}$
- A population contains a fixed number of individuals: **population size**
- Each iteration is referred as **generation**

GA Components: Initialisation

- <u>Random Initialisation</u>
- **Population size** number of individuals are created randomly
- Each gene at a locus of an individual is assigned an **allele** value 0 or 1 randomly, decided by flipping a coin (E.g., if the random value is <0.5, then allele is assigned to 0, otherwise to 1).



Example – MAX-SAT: Initialisation

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

$$\overset{0}{(a \vee b)} \wedge \overset{1}{(\neg d \vee f)} \wedge \overset{2}{(\neg a \vee c)} \wedge \overset{3}{(b \vee \neg f)} \wedge \overset{4}{(\neg b \vee c)} \wedge \overset{5}{(c \vee e)}$$

1: <0.98, 0.85, 0.13, 0.04, 0.47, 0.44>

2: <0.09, 0.63, 0.22, 0.54, 0.07, 0.37>

3: <0.21, 0.03, 0.72, 0.84, 0.19, 0.49>

4: <0.61, 0.43, 0.87, 0.53, 0.97, 0.14>

101110

- Assume *population size* is 4
- The individual size/*chromosome length* is 6 (since we have 6 literals: *abcdef*)
- So, create 4 individuals with 6 genes within their chromosomes, where each allele at a locus is determined **randomly** (by throwing a random number in [0,1)).

Example – MAX-SAT: Initialisation

i	Chromosome *abcdef*
1:	110000
2:	010100
3:	001100
4:	101110

$$\overset{0}{(a \vee b)} \wedge \overset{1}{(\neg d \vee f)} \wedge \overset{2}{(\neg a \vee c)} \wedge \overset{3}{(b \vee \neg f)} \wedge \overset{4}{(\neg b \vee c)} \wedge \overset{5}{(c \vee e)}$$

i	Chromosome *abcdef*
1:	110000
2:	010100
3:	001100
4:	101110

- Assume *population size* is 4
- The individual size/*chromosome length* is 6 (since we have 6 literals: *abcdef*)
- So, create 4 individuals with 6 genes within their chromosomes, where each allele at a locus is determined **randomly** (by throwing a random number in [0,1)).

GA Components – Fitness Calculation

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

- **Fitness value** indicates
  - how fit the individual is to survive and reproduce under the current conditions
  - how much the current solution meets the requirements of the objective function
- **Fitness value** is obtained by applying the fitness function to the individual's chromosome (candidate solution) – *genotype* (e.g., 101110) to *phenotype* (e.g., 1) mapping

Example – MAX-SAT: Fitness Calculation

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with pc;// apply crossover
  apply mutation with pm; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

$$\overset{0}{(a \vee b)} \wedge \overset{1}{(\neg d \vee f)} \wedge \overset{2}{(\neg a \vee c)} \wedge \overset{3}{(b \vee \neg f)} \wedge \overset{4}{(\neg b \vee c)} \wedge \overset{5}{(c \vee e)}$$

i	Chromosome *abcdef*	Unsatisfied clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

GA Components – Reproduction

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

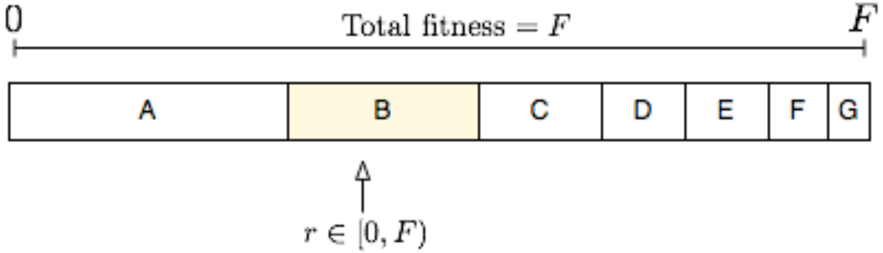
- Reproduction (Mate Selection) consists of
  - selecting individuals: apply selection pressure considering the fitness of individuals in the population ⇒ e.g., *roulette wheel selection, tournament selection,* rank selection, truncation selection, Boltzmann selection, etc.
    - Selection pressure means the individuals with better fitness have higher chance for being selected
  - usually **2** *parents* (individuals/candidate solutions) are selected using the same method, which will go under the crossover operation

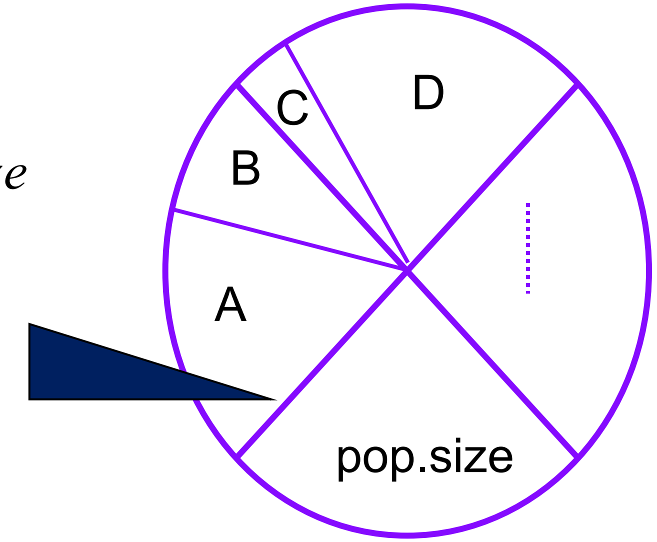
Fitness Proportionate Selection – Roulette Wheel Selection

- Fitness level is used to associate a probability ($prob_i$) of selection with each individual chromosome (i)
- While candidate solutions with a higher fitness will be less likely to be eliminated, there is still a chance that they may be (maximisation problem)
- Expected number of representatives of each individual in the pool is proportional to its fitness (maximisation problem)

maximisation problem

$$prob_i = \frac{fitness_i}{\sum_j fitness_j}\quad , j = 1 \ldots pop.size$$

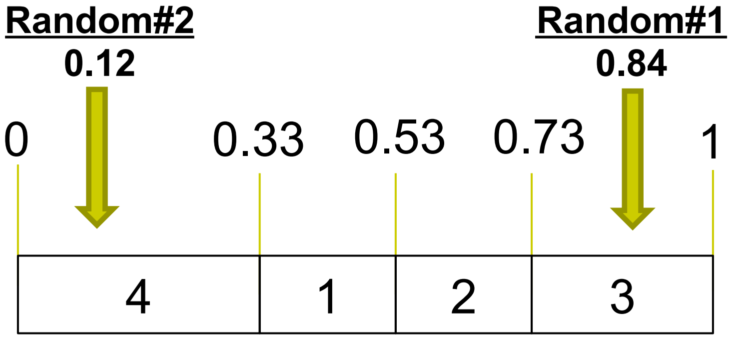




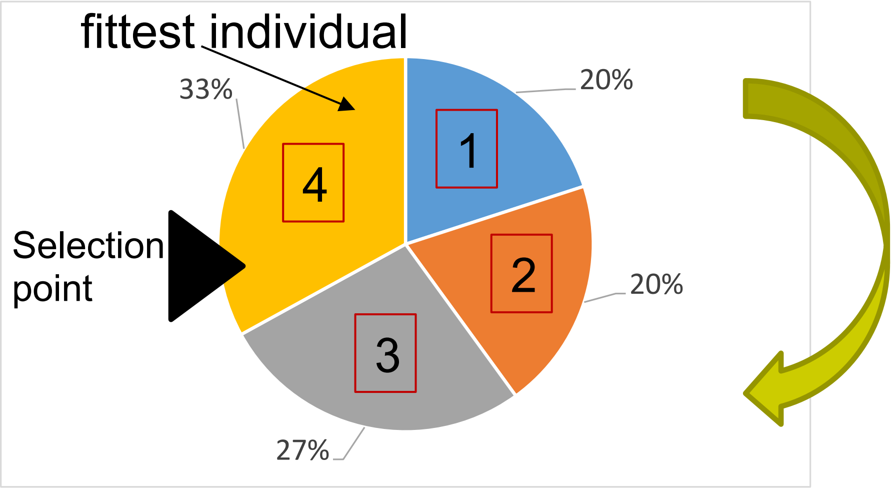
Example: MAX-SAT – Roulette Wheel Selection

$$\overset{0}{(a \vee b)} \wedge \overset{1}{(\neg d \vee f)} \wedge \overset{2}{(\neg a \vee c)} \wedge \overset{3}{(b \vee \neg f)} \wedge \overset{4}{(\neg b \vee c)} \wedge \overset{5}{(c \vee e)}$$

i	Chromosome *abcdef*	Unsat. clauses	Fitness (*fmax-f*)	$prob_i$
1	110000	012345	3 (3)	20%
2	010100	012345	3 (3)	20%
3	001100	012345	2 (4)	27%
4	101110	012345	1 (5)	33%
			total:(15)	



- Rotate the wheel



Random#1: 0.84 → Parent#1: 3

Random#2: 0.12 → Parent#2: 4

Tournament Selection

- This method involves running a number of "tournaments" among randomly chosen individuals (of tour size) selecting the one with best fitness at the end
  - This process is repeated for selecting each parent to be recombined

Example – MAX-SAT: Tournament Selection

tour size = **3**, first parent

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

i	Chromosome *abcdef*	Violated clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

- Throw a random number between 1 and 4 (population size) for **3** times:
  - [3, 1, 2]
- Tournament selection chooses 3 individuals: #1, #2 and #3 at random, then individual#3 with the fitness of 2 is returned as the first parent

Example – MAX-SAT: Tournament Selection

tour size = **3**, second parent

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
 perform reproduction; // select parents
 recombine pairs with p_c;// apply crossover
 apply mutation with p_m; // mutate offspring/children
 calculate fitness values; // eval. population
 replace current population;
} while termination criteria not satisfied;
end
```

i	Chromosome *abcdef*	Violated clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

- Throw a random number between 1 and 4 (population size) for **3** times:
  - [4, 1, 3]
- Tournament selection chooses 3 individuals: #1, #3 and #4 at random, then individual#4 with the fitness of 1 is returned as the second parent

Recombination – Crossover

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with pc;// apply crossover
  apply mutation with pm; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

- Selected pairs/mates (*parents*) are recombined to form new individuals (candidate solutions/children/offspring) – exchange of genetic material
- Crossover is applied with a **crossover probability** p_c which in general is chosen close to 1.0

One Point Crossover (1PTX)

- Generate a random number in [0,1), if it is smaller than a **crossover probability** p_c Then
  - Select a random crossover site in [1,choromosome length]
  - Split individuals at the selected site
  - Exchange segments between pairs to form two new individuals
- Else
  - Copy the individuals as new individuals

Example – MAX-SAT: 1PTX

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

$$\overset{0}{(a \lor b)} \land \overset{1}{(\neg d \lor f)} \land \overset{2}{(\neg a \lor c)} \land \overset{3}{(b \lor \neg f)} \land \overset{4}{(\neg b \lor c)} \land \overset{5}{(c \lor e)}$$

Random number: 2

Randomly determined crossover point

001100 → 101100

101110 → 001110

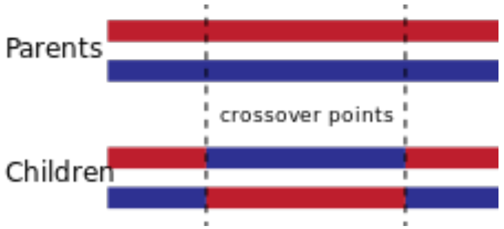
2. Exchange the genetic material

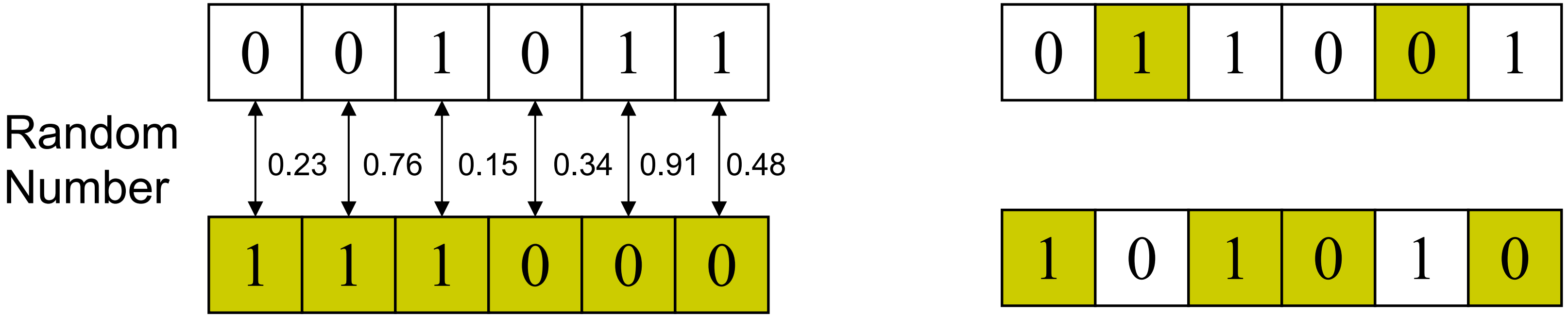
1. Throw a random number in [1..6]

new solutions/children/offspring

Other Crossover Operators

- 2 Point Crossover (2PTX)
- K-point Crossover
- Uniform Crossover (UX)
  - The uniform crossover considers each bit in the parent strings for exchange with a probability of 0.5.





Mutation

- Any offspring might be exposed to mutation
- Loop through all the alleles of all the individuals one by one, and if that allele is selected for mutation with a given probability p_m, you can either change it by a small amount or replace it with a new value
  - For binary representation mutation corresponds to flipping a selected gene value ($0 \rightarrow 1$, $1 \rightarrow 0$)
- Mutation provides diversity and allows GA to explore different regions of the search space (escaping)
- Mutation rate is typically chosen to be very small (0.001, 0.001). Choosing p_m as (*1/chromosome length)* implies on average a single gene will be mutated for an individual.

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

Example – Mutation

1	0	1	1	0	0

0	0	1	1	1	0

- A loop is performed on each individual
  - If a random value in [0,1) is < p_m =0.17 (1/6), then the allele value is flipped, otherwise kept same.

Example – Mutation

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

Random numbers:
<0.23 0.76 0.41 0.14 0.91 0.68 0.47 0.16 0.28 0.94 0.78 0.03 …>

0.23	0.76	0.41	0.14	0.91	0.68
1	0	1	0	0	0

0.47	0.16	0.28	0.94	0.78	0.03
0	1	1	1	1	1

- A loop is performed on each individual
  - If a random value in [0,1) is < p_m =0.17 (1/6), then the allele value is flipped, otherwise kept same.

Replacement Strategy

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

- There are variety of strategies for replacing the old population (generation) by the new (offspring) population to form the next generation
- **Generation gap (α)** controls the fraction of the population to be replaced in each generation, where $\alpha \in [1/N, 1.0]$
  - Number of offspring produced at each generation is $\boldsymbol{g}=\alpha^* N$
- ***(Trans-)Generational GA*** (g>2, that is α>2/N)

  N individuals produce αN offspring, so $(N + \alpha N) \rightarrow N$
  - αN replaces worst αN of N
    - largest *generation gap* where α=1.0 yields $g=N$.
    - GA relies on improvement of average objective values from one population to another
      - It is always a good idea not to loose the best solution found so far.
  - sort $(N + \alpha N)$ and choose the N best (elitism)

Replacement Strategy

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with pc;// apply crossover
  apply mutation with pm; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

- ***Steady-State GA*** **(*g*=2, that is α=2/N)**

  Two *o*ffspring replace two individuals from the old generation.

  - Method#1: two *o*ffspring replace two parents
  - Method#2: two *o*ffspring replace worst two of the population
  - Method#3: best two of (parents and offspring) replace two parents (elitism)
  - Method#4: best two of (parents and offspring) replace worst two of the population (strong elitism)

Example – Transgenerational GA Replacement (no elitism)

$$\overset{0}{(a \vee b)} \wedge \overset{1}{(\neg d \vee f)} \wedge \overset{2}{(\neg a \vee c)} \wedge \overset{3}{(b \vee \neg f)} \wedge \overset{4}{(\neg b \vee c)} \wedge \overset{5}{(c \vee e)}$$

i	Chromosome abcdef	Unsat. clauses	Fitness
1	100100	012345	3
2	010100	012345	3
3	101000	012345	0
4	011111	012345	0

Offspring

i	Chromosome abcdef	Unsat. clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

Old Population

Form the new generation by copying offspring onto the old population, forming the new population

(g=N=4)

i	Chromosome abcdef	Unsat. clauses	Fitness
1	100100	012345	3
2	010100	012345	3
3	101000	012345	0
4	011111	012345	0

New Population/ Generation

Example – TGA Replacement (with elitism)

$$\begin{array}{ccccccccccc} 0 & & 1 & & 2 & & 3 & & 4 & & 5 \\ (a \vee b) & \wedge & (\neg d \vee f) & \wedge & (\neg a \vee c) & \wedge & (b \vee \neg f) & \wedge & (\neg b \vee c) & \wedge & (c \vee e) \end{array}$$

i	Chromosome ***abcdef***	Unsat. clauses	Fitness
1	100100	012345	3
2	010100	012345	3
3	101000	012345	0
4	011111	012345	0

Offspring

i	Chromosome ***abcdef***	Unsat. clauses	Fitness
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

Old Population

Form the new generation by

Selecting the best 4 individuals among both old population and offspring

(g=N=4)

i	Chromosome ***abcdef***	Unsat. clauses	Fitness
1	001100	012345	2
2	101110	012345	1
3	101000	012345	0
4	011111	012345	0

New Population/ Generation

Example – A Steady State GA (with elitism)

$$\overset{0}{(a \vee b)} \wedge \overset{1}{(\neg d \vee f)} \wedge \overset{2}{(\neg a \vee c)} \wedge \overset{3}{(b \vee \neg f)} \wedge \overset{4}{(\neg b \vee c)} \wedge \overset{5}{(c \vee e)}$$

i	**Chromosome *abcdef***	**Unsat. clauses**	**Fitness**
1	010100	012345	3
2	101000	012345	0

Offspring

i	**Chromosome *abcdef***	**Unsat. clauses**	**Fitness**
1	110000	012345	3
2	010100	012345	3
3	001100	012345	2
4	101110	012345	1

Old Population

Next generation ←

Using Method#2: Replace the worst 2 of the population with 2 offspring (g=2)

i	**Chromosome *abcdef***	**Unsat. clauses**	**Fitness**
1	010100	012345	3
2	101000	012345	0
3	001100	012345	2
4	101110	012345	1

New Population/ Generation

Termination Criteria

```
begin
generate initial population; // initialise
calculate fitness values;// evaluate population
do
{
  perform reproduction; // select parents
  recombine pairs with p_c;// apply crossover
  apply mutation with p_m; // mutate
  offspring/children
  calculate fitness values; // eval. population
  replace current population;
} while termination criteria not satisfied;
end
```

- The evolution (main loop) continues until a termination criteria is met, possibly until:
  - A predefined maximum number of generations is exceeded
  - A goal is reached, for example:
    - Expected fitness is achieved
    - Population converges
  - Best fitness does not change for a while
  - A condition is satisfied depending on a combination of above

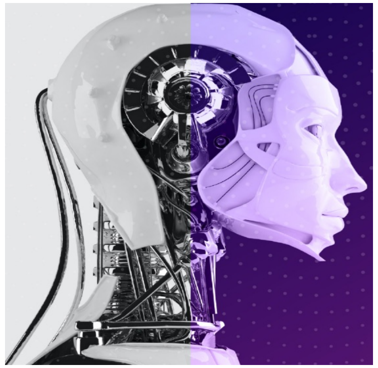
Convergence

- Defined as the progression towards uniformity (individuals become alike)
  - *Gene convergence*: a location on a chromosome is converged when 95% of the individuals have the same gene value for that location
  - *Population (Genotypic) convergence*: a population is converged when all the genes have converged (all individuals are alike – they might have different fitness)
  - *Phenotypic Convergence*: average fitness of the population approaches to the best individual in the population (all individuals have the same fitness)

Key Features of EAs

- Population based search approaches
  - Be independent of initial starting point(s)
    - Start search from many points in the search space
  - Conduct search in parallel over the search space
    - implicit parallelism
- Avoid converging to local optima
- Balances exploration and exploitation?
- May be used together with other approaches (hybrids)

Memetic Algorithms



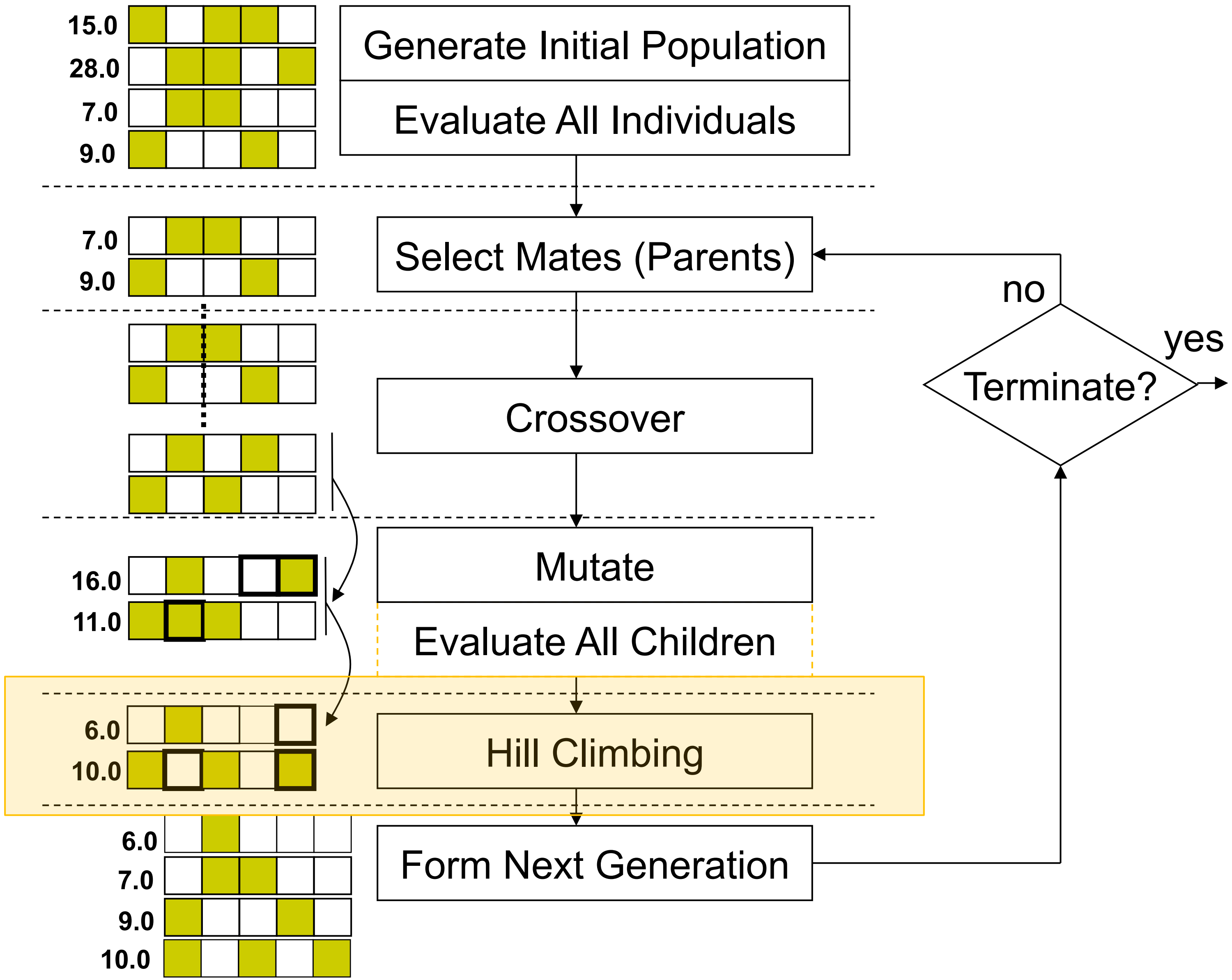
Memetic Algorithms (MAs)

- Meme (Richard Dawkins): contagious piece of information
- Memes are similar to local refinement/local search
- Gene vs meme: Memes can change, evolve using rules and time scales other than the traditional genetic ones
- MAs aim to improve GAs by embedding local search

Memetic Algorithms (MAs) II

- MAs make use of exploration capabilities of GAs and exploitation capabilities of local search
  - MAs have an explicit mechanism to balance exploitation and exploration
- Memetic Algorithms shown to be much faster and more accurate than GAs on some problems, and are the “state of the art” on many problems

A Generic Memetic Algorithm (MA)



Memetic Algorithms (MAs)

Pseudocode of memetic algorithm

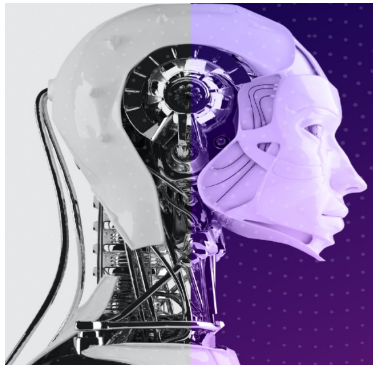
```
CreateInitialSolutions(); // create initial population of solutions

repeat

    SelectParents(); // select solutions from the population to breed
    Crossover(); // apply crossover operator with a given probability
    Mutate(); // apply mutation operator with a given probability
    LocalSearch();
    Evaluate();
    ReplaceSolutions(); // generate new population of solutions
until TerminationCriteriaSatisfied();
```

Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, Caltech Concurrent Computation Program Report 826, California Institute of Technology (1989)

Example – Designing a Genetic/Memetic Algorithm for MAX-SAT

- *Representation*: Bit string of length *N* (truth assignment for each variable)
- *Initialisation*: Randomly generate initial population, population size=*N*
- *Fitness function*: (# of clauses – *C*); # of unsatisfied clauses
- *Mate selection*: Tournament selection with a tour size of 2
- *Crossover*: 1PTX, crossover probability = 0.99
- *Mutation*: random bit-flip, mutation probability = 1/*N*
- ***Hill Climbing*: Davis's Bit Hill Climbing**
- *Replacement*: Steady State GA, best two of (parents and offspring) replaces the worst two individuals in the population (strong elitism)

Early Module Feedback



Q&A

Thank you.

Ender Özcan ender.ozcan@nottingham.ac.uk

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
http://www.nottingham.ac.uk/~pszeo/