Implementing Genetic Algorithms

Learning Objectives

After completing this lecture, you will be able to:-

- Design and implement a complete genetic algorithm
- Select operators and parameters to improve the performance of a genetic algorithm
- Discuss the benefits and limitations of genetic algorithms
- Anticipate issues in practical use of genetic algorithms

Genetic Algorithm Building Blocks

- Problem definition: a **chromosome** which encodes a solution to the problem
- **Initialization** procedure: to create the initial **population**
- Genetic operators: **selection**, **crossover**, and **mutation**
- Objective evaluation: **fitness function** or **fitness score**
- **Termination** condition

Genetic Algorithm

- **Initialization**: Start with a large **population** of randomly generated **chromosomes**
- Repeat:
  - **Evaluate** each solution (with a **fitness function**)
  - Keep fitter solutions, eliminate poorer ones (**selection**)
  - Generate new solutions (**crossover**)
  - Add small random variances (**mutation**)
- Stop when your solution is satisfactory (**convergence**) or you run out of time (**termination condition**)

Initialization

- Chromosomes are **randomly generated** for a **population** (with size N)
- The chromosomes must contain information (**genes**) about a solution for the problem being solved
- The chromosomes are **encoded** in one of several forms (depending on the problem domain)
- There are a few types of encoding methods (covered in the next few slides) which define the **mapping** between **genotype** and **phenotype**

Initialization – Binary Encoding

- Each chromosome is represented using a binary string
- Every gene is represented using the bits 0 or 1
  - Each bit or group of bits represents some aspect of the problem (e.g. ‘rain’ or ‘no rain’)
- Each gene shows some **characteristic** of the solution
- Each chromosome represents a value in the search space

- **Used for (example)**
  - Knapsack problem, given a fixed capacity and a list of items with value/weight/size, select items to maximize value without exceeding capacity
- **Encoding**
  - Each bit represents whether the corresponding item is in the knapsack

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

Initialization – Value Encoding

- Each chromosome is represented as a string of some values
- Each gene represents a variable
- Value can be an integer, a real number, a character, or some object

- **Used for (example)**
  - Finding neural network weights, given a certain architecture, find the best weights to achieve a certain output
- **Encoding**
  - Real values in chromosomes which represent the corresponding neural network weights

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Initialization – Permutation Encoding

- Each chromosome represents a **sequence of items**
- Each gene represents an item
- Useful for **ordering problems** (problems where the solutions have a specific order)

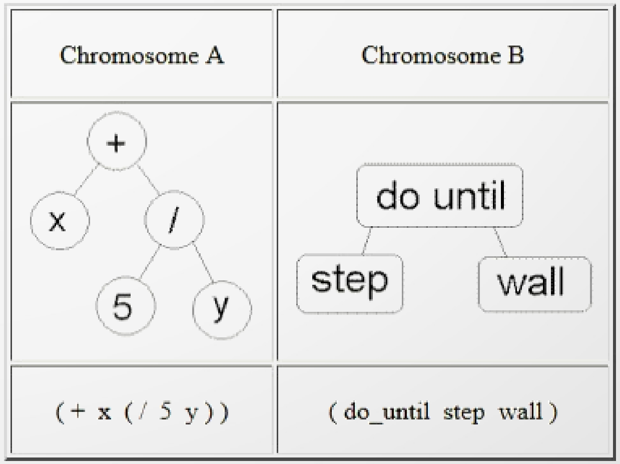
- **Used for (example)**
  - Travelling salesman problem (TSP), given a number of cities and distances between them, find the shortest sequence of trips which visits all the cities
- **Encoding**
  - Chromosome represents order in which cities will be visited

Chromosome A	1	5	3	2	6	4	7	9	8
Chromosome B	8	5	6	7	2	3	1	4	9

Initialization – Tree Encoding

- Each chromosome is a **tree of objects** (such as functions/commands in a programming language)
- Each gene represents an object in the tree
- Mainly used for **evolving programs** or **genetic programming**

- **Used for (example)**
  - Finding a (programming) function which will achieve a certain output for a fixed set of inputs
- **Encoding**
  - Chromosome represents functions in a tree



Initialization

Two methods for initializing the population:-

- **Random Initialization**
  - Populate the initial population with completely random solutions
- **Heuristic Initialization**
  - Populate the initial population using a known heuristic (rules learned via experience) for the problem

Search Space

- The **population** exists within a defined (possibly infinite) search space
- Each **individual** represents a solution within this search space, with one dimension per gene (on average)
- The high dimensionality of the search space normally precludes easy visualization
  - We can still imagine how a search space ‘looks’

Search Space

- A completely random search space would be bad for GA (and any other optimization method)
  - Inheriting 'good' traits has no benefit
- A single-valley space without local minima is more efficiently solved by gradient-descent related methods
- A search spaces with a fairly continuous surface and multiple valleys is suitable for GA (especially if it is prohibitively large)

Iterative Evolution

With an initial population the following is done iteratively:-

- **Selection**
  - Evaluate individual fitness and give preference to ‘fitter’ individuals
- **Crossover (Mating/Recombination/Reproduction)**
  - 2 individuals (from selection step) exchange genes, creating a new (hopefully better) solution
- **Mutation**
  - Random modifications are introduced to individuals

Selection

- Preference should be given to **better individuals** to pass on their genes to the next generation
- '**Better**' is defined by an individual's **fitness**
- **Fitness** is determined by an **objective/fitness function**
- Selection should favour **fitter** chromosomes, but there are no fixed rules as to how much favouritism should be applied
- No selection strategy consistently performs best for all types of problems

Selection

There are two kinds of selection:

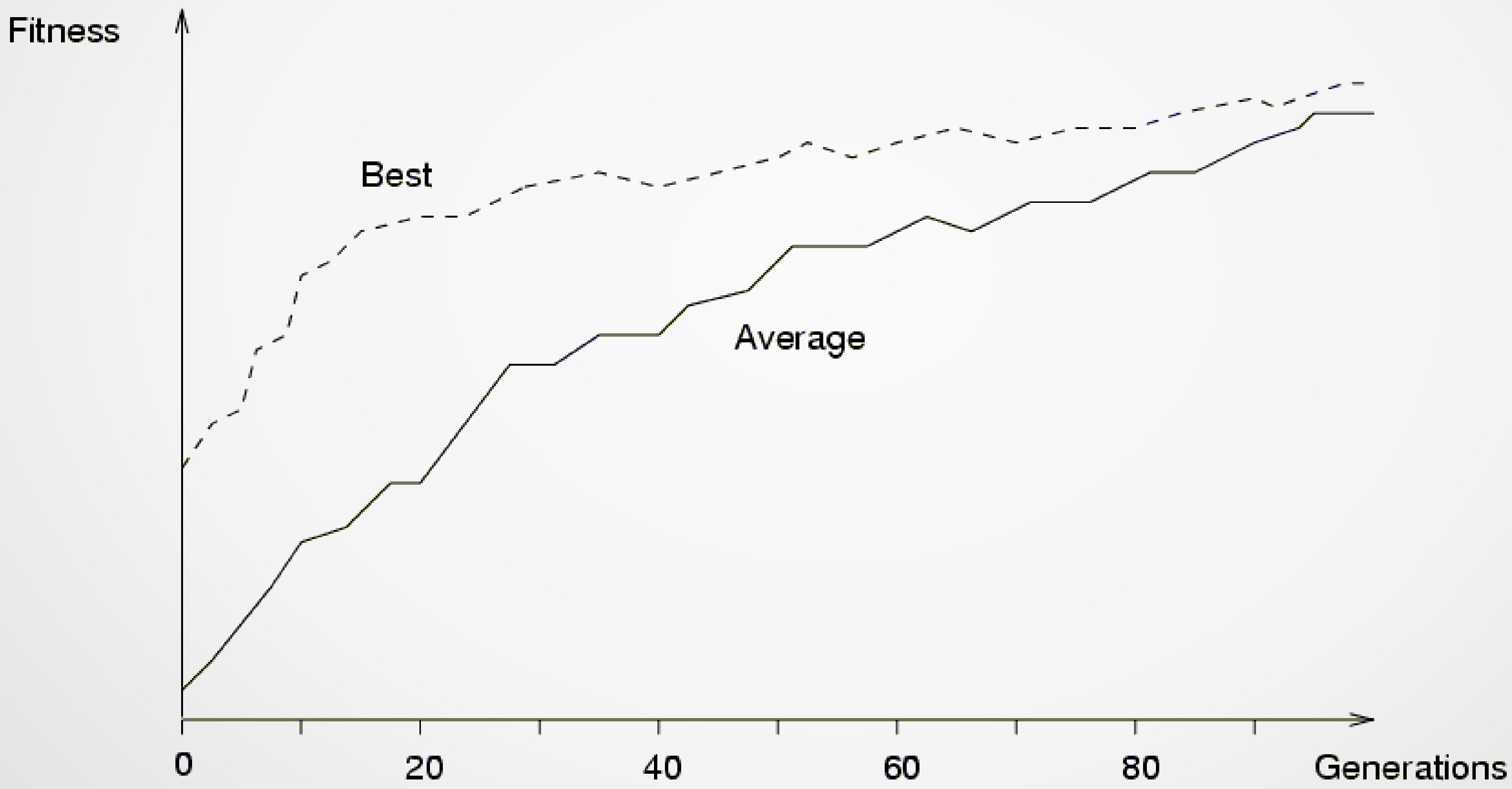
- **Parent selection**
  - Selecting which parents mate and recombine to create offspring for the next generation
- **Survivor selection**
  - Selecting which individuals are to be kicked out and which are to be kept for the next generation

Selection

- The **fitness value/score** of each individual is the value being optimized (minimized or maximized) by the GA
- In general, fitness scores are used:-
  - **Parent selection**: Better fitness scores increases chances of being a parent
  - **Survivor selection**: Better fitness scores increases odds of surviving
- Over the generations, less-fit individuals will die (be removed), leaving each generation better off than previous generations
- **Convergence** happens when successive generations don't improve fitness much

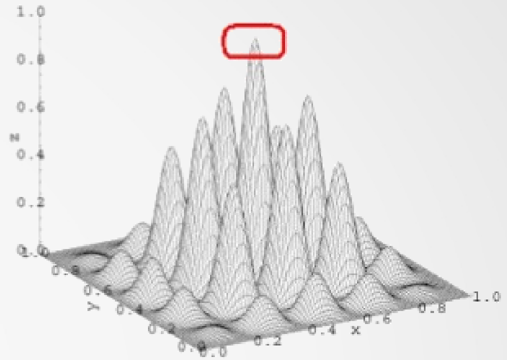
Selection

Example of convergence

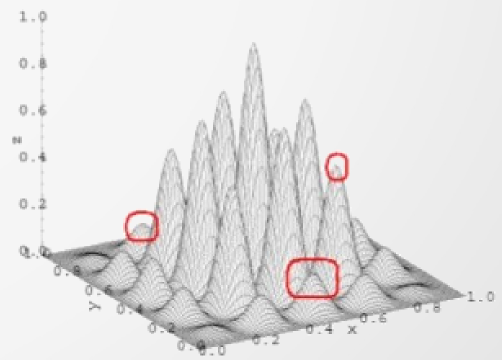


Selection

- Maintaining good population **diversity** is extremely critical for a successful GA
- If the entire population consists of variations of one extremely fit solution (**premature convergence**) the GA is likely to underperform
- Dilemma: We want fit, but not too fit (both fit and diversity important)



Our objective is to attain some maximum value



Premature convergence at certain local niches

Parent Selection

In current practice, the following parent selection strategies are generally used

- Fitness-proportional selection (only for single-sign fitness)
  - Roulette wheel slection
  - Stochastic universal sampling
- Tournament selection (handles negative fitness)
- Rank selection (handles negative and low/high variance fitness)
- Truncation selection
- Steady-state selection (incorporates survivor selection)
- Random selection (pointless)

Parent Selection – Roulette Wheel

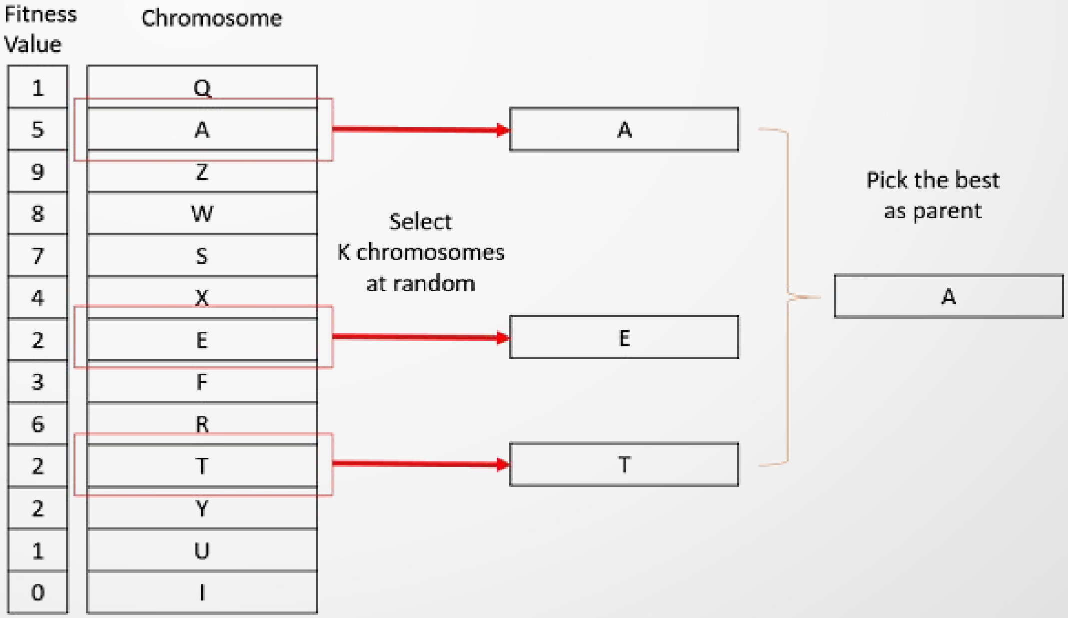
- Every chromosome has a slice of the roulette wheel proportional to its fitness, the wheel is then 'spun' to see which chromosome is chosen as a parent
- In general:-
  - Calculate sum of fitness S
  - Generate random number r between 0 and S
  - Loop through each chromosome, adding its fitness to a sum until the sum is greater than r, choose the matching chromosome

Parent Selection – SUS

- Stochastic Universal Sampling is similar to Roulette wheel selection, but only one random number is used (one spin of the wheel)
- All parents are then chosen at evenly spread intervals around the wheel
- Avoids too much bias if random values aren't properly distributed for Roulette wheel selection

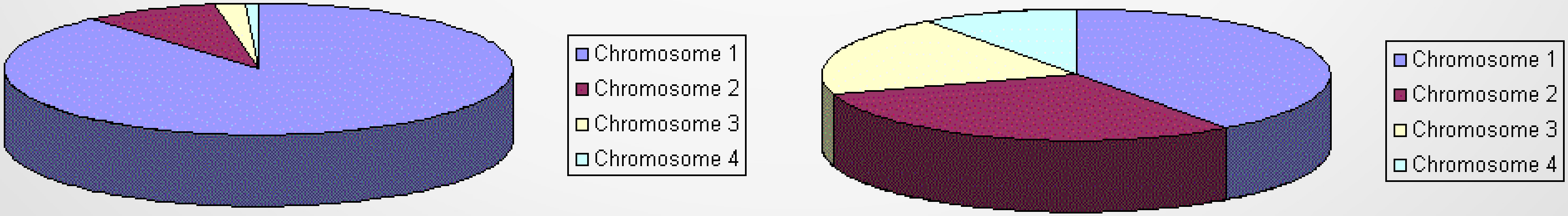
Parent Selection – Tournament

- A few chromosomes are chosen at random, and a **tournament** is then run between them
- The **winner** (best fitness) is selected as a parent
- **Tournament size** is an important parameter which changes selection pressure
  - Large tournament size disadvantages weak individuals, small tournament size increases randomness



Parent Selection – Rank

- When a population has very close fitness values, there is very little **selection pressure**, making GA effectively random
- Alternatively, when a population has very different fitness values, there is too much **selection pressure**, which could lead to premature convergence
- Rank selection assigns probability of selection based on **fitness rank** rather than fitness



Parent Selection – Rank

- After fitness is calculated, all individuals are ranked
- Each individual receives a fixed (decreasing with lower rank) probability of being selected (e.g. 0.5, 0.25, 0.125…)
- Selection is then done using one of the fitness proportionate methods, but the probability is used instead of the fitness
- Higher fitness still gives preference, but this is now bounded
- Negative values also work with rank selection

Parent Selection – Truncation

- A fixed proportion of the fittest individuals are selected for recombination
- Based on animal/plant breeding practices (directed evolution)
- Less sophisticated than the other methods discussed here (except random selection) and not often used in practice

Parent Selection – Steady-State

- Main idea: a big part of current chromosomes should survive
- In every generation, select a few (high fitness) chromosomes for creating new offspring
- Then select a few (low fitness) chromosomes to be replaced
- The rest of the population survives to the next generation
- Convergence is slower due to lower turnover, also may be more vulnerable to local minima (just increase population)

Survivor Selection

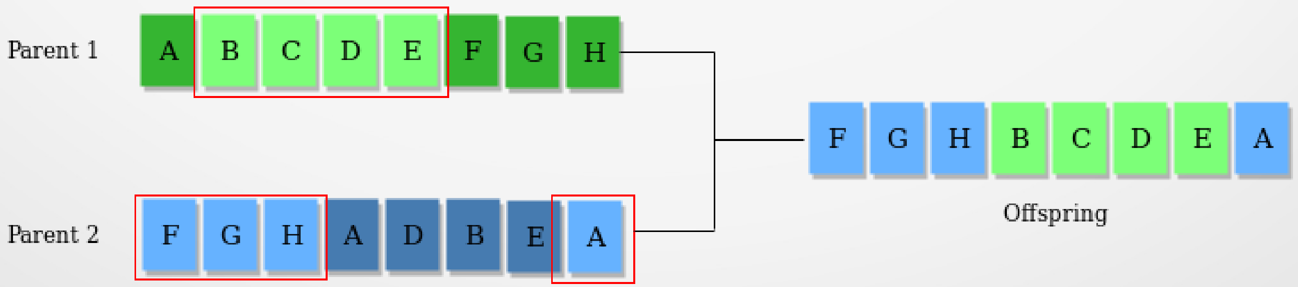
- Good chromosomes (solutions) can be lost due to crossover or mutation resulting in weaker offspring
- These can be **rediscovered**, but there's no guarantee
- In genetic algorithms, **elitism** is the practice of copying a **small proportion** of the fittest chromosomes unchanged (no crossover or mutation)
- This can dramatically impact performance (quality and speed) of the genetic algorithm search process
- **Elites** remain eligible for selection as parents

Survivor Selection

- Survivor selection determines which individuals are kicked out (die) and which are kept (elitism) in the next generation
- Survivor selection strategies:-
  - **Fitness based** – this is traditional elitism
  - **Age based** – each individual is allowed to remain for a finite number of generations before it is kicked out
    - This allows for multiple ‘tries’ at reproduction, increasing the chance of passing on good genes

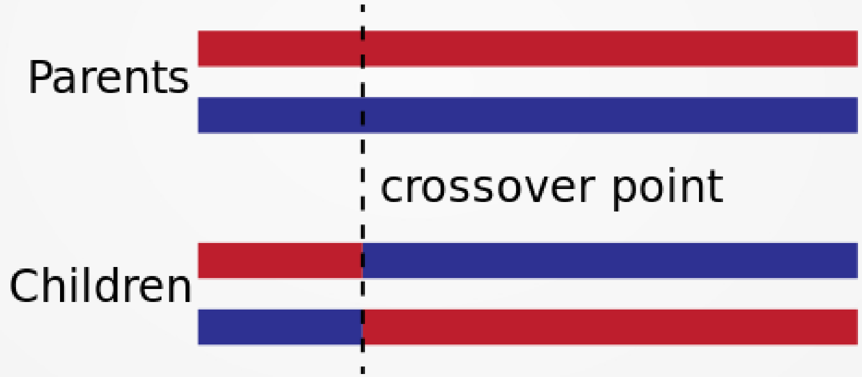
Crossover

- Also known as **mating/recombination/reproduction**
- Randomly mixes genes between parents (output of parent selection)
- The parents each provide part (50% or otherwise) of their genes (unique traits/characteristics)
- The new **offspring** hence inherit both parent's traits in their chromosome
- This can increase **diversity**



Crossover General Algorithm

- Input: two parents
- Randomly choose a **crossover site**
- Exchange genes up to this site



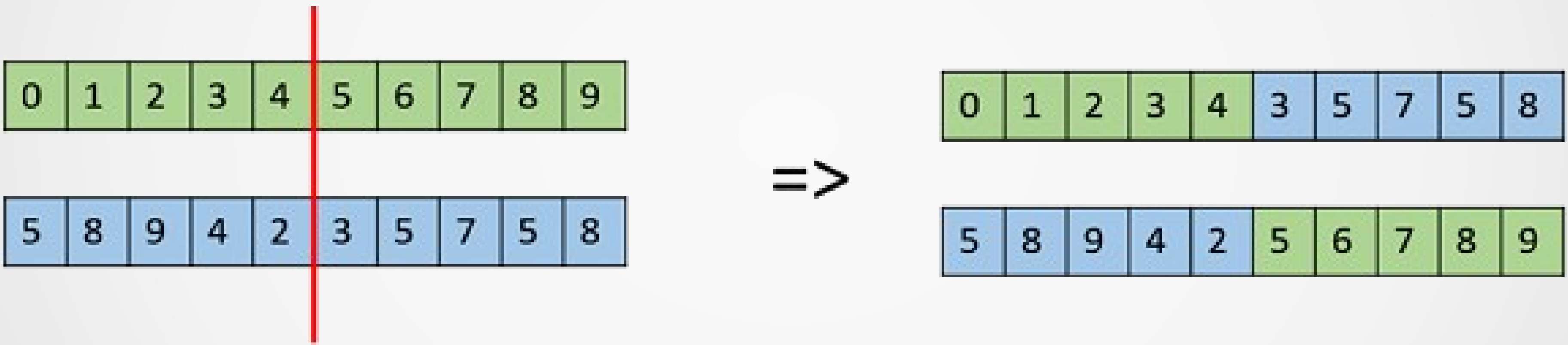
- The two offspring are then put into the next generation of the population

Simple Crossover Methods

- One Point / Single Point Crossover
- Multi Point Crossover
- Uniform Crossover
- Whole Arithmetic Recombination

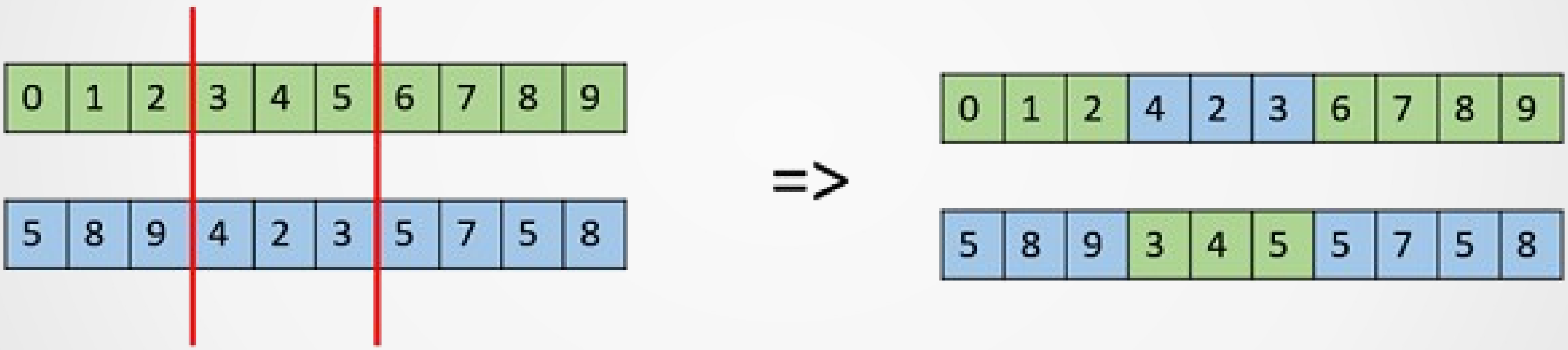
One Point Crossover

A random crossover point is selected and the tails of the parents are swapped to get new offspring



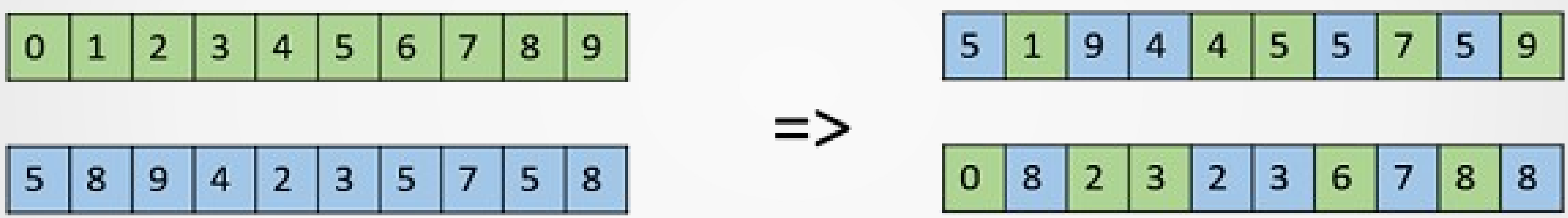
Multi Point Crossover

A generalization of one point crossover, where multiple points are used to swap segments of the parents



Uniform Crossover

Each gene is treated separately. Essentially, flip a coin for each gene to see which child it ends up in (coin can be biased away from 0.5)



Whole Arithmetic Recombination

- Commonly used for **integer representations** and works by taking the **weighted average** of the two parents

$$C_1 = \alpha P_1 + (1-\alpha) P_2$$
$$C_2 = (1-\alpha) P_1 + \alpha P_2$$

- If $\alpha = 0.5$ then both children will be identical

0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5
0.2	0.3	0.2	0.2	0.3	0.2	0.3	0.2	0.3	0.2

=>

0.15	0.2	0.2	0.2	0.3	0.25	0.35	0.3	0.2	0.35
0.15	0.2	0.2	0.2	0.3	0.25	0.35	0.3	0.2	0.35

Other Crossover Methods

The simple crossover methods make assumptions about chromosome design etc. that may not be suitable for a particular problem. Here are some alternatives:-

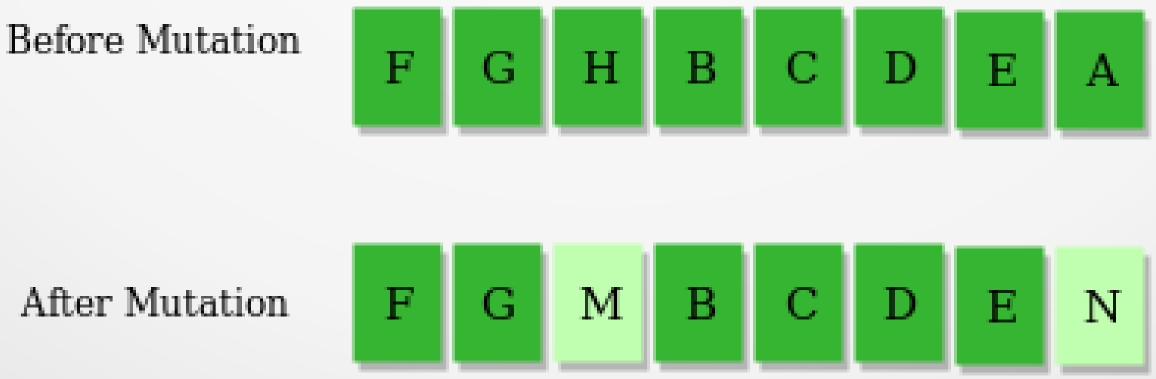
- Davis' Order Crossover (OX1)
- Partially Mapped Crossover (PMX)
- Order based crossover (OX2)
- Shuffle crossover
- Ring Crossover
- … and many more (custom designed methods are common)

Mutation

- Mutation is a **random change** of genes
- In nature, mutation is the result of copying errors in DNA, possibly due to toxins, radiation, or chemical substances
- Most of these changes are negative and may result in illnesses
- However, some may have neutral or positive impact
- Mutations also contribute significantly to **diversity** (which is the primary point of its inclusion in GA)
- Mutation alone is effectively random search

Mutation

- In genetic algorithms, mutation is implemented as a **small random tweak** in a chromosome
- It is used to maintain and introduce **diversity** and try to avoid premature convergence
- It is usually applied with a **low probability** to avoid GA reducing to a random search



Mutation Methods

- Bit Flip Mutation
- Random Resetting
- Swap Mutation
- Scramble Mutation
- Inversion Mutation
- … and many more (custom designs as well)

Bit Flip Mutation

- Select one or more random bits and **flip** them

0	0	1	1	0	1	0	0	1	0

=>

0	0	1	0	0	1	0	0	1	0

Random Resetting

- Extension of the bit flip for non-binary encodings
- A **random value** is assigned to a randomly chosen gene

Swap Mutation

- Select 2 positions on the chromosome at random, and **interchange/swap** their values
- This is common in **permutation (order) based** encodings.

1	2	3	4	5	6	7	8	9	0

=>

1	6	3	4	5	2	7	8	9	0

Scramble / Shuffle Mutation

- A subset of genes is chosen and their values **scrambled** or **shuffled** randomly (for **permutation based** encodings)

0	1	2	3	4	5	6	7	8	9

=>

0	1	3	6	4	2	5	7	8	9

Inversion Mutation

- Select a subset of genes like in scramble/shuffle mutation, but instead of shuffling the subset, we merely **invert/reverse** the entire string in the subset

0	1	2	3	4	5	6	7	8	9

=>

0	1	6	5	4	3	2	7	8	9

Terminating Genetic Algorithms

- The iterative portion of GA should stop when:-
  - Fixed number of generations reached
  - Allocated budget (computer time/money) reached
  - Highest ranking solution's fitness has plateaued (results not improving for best solution)
  - Expert decides its time to stop
  - Some combination of the above

Genetic Algorithm Parameters

- Population and Generation Size (N)
- Crossover Probability (P_c)
- Mutation Probability (P_m)
- Termination Condition

Often these parameters need ‘tuning’ based on results obtained, no general theory to calculate ‘good’ values (therefore heuristic selection)

Genetic Algorithm Parameters

Population and Generation Size (N)

- How many chromosomes in population
- Too few – search space not explored much
- Too many – computation takes too much time
- There is an (unknown) upper bound to N above which problems cannot be solved faster
- Proper choice of N avoids unnecessary computation

Genetic Algorithm Parameters

Crossover Probability (P_c)

- Crossover is done hoping that children inherit good parts of their parents, resulting in a better solution
- If P_c is 100%, all offspring are the result of crossover
- Any value of P_c under 100% is effectively a form of survivor selection
- There is no optimum value for P_c, it normally depends on heuristics (and the problem)

Genetic Algorithm Parameters

Mutation Probability (P_m)

- Mutation is done to avoid premature convergence (increase diversity) by providing an opportunity for solutions to escape local minima
- If P_m is 100%, all genes/chromosomes are changed
- Mutation should not occur too often, because the GA would then become a random search
- There is no optimum value for P_m, it normally depends on heuristics (and the problem)

Benefits of Genetic Algorithms

- Easy to understand
- Optimizes both **continuous** and **discrete** functions and also **single-objective** and **multi-objective** problems.
- Good for **noisy** environment (error-prone data, crossover and mutation increase diversity)
- We always get solution in a reasonable time (though may not be optimal) and solution gets better over time
- **Faster** and **more efficient** as compared to the traditional methods.
- Inherently **parallel** and easily **distributed**
- May provide **a list of "good" solutions** and not just a single solution. Easy to exploit for previous or alternate solutions.
- **Flexible** in forming building blocks for hybrid applications (mix and match different solutions)
- Useful when the **search space is very large** and there are a **large number of parameters** involved.
- Has substantial history and range of use (proven effectiveness)

Limitations of Genetic Algorithms

- GAs are not suited for all problems, especially problems which are simple.
- **Fitness scores** have to be calculated repeatedly (for different chromosomes) which might be **computationally expensive** for some problems (though we may remember the scores for those chromosomes already evaluated)
- Implementation (choosing parameters and operators) is still an art
- Being **stochastic** (probabilistic), there are **no guarantees on the optimality** or the quality of the solution.
- GAs may not converge to the optimal solution. **Premature convergence** may lead the algorithm to converge on the local optimum

Issues for Practitioners

- Basic implementation decisions
  - Representation/encoding
  - Population size (N) and crossover/mutation probabilities (P_c, P_m)
  - Selection policies
- Termination criterion (When to stop? When does it converge?)
- Performance (how fast is a solution needed)
- Scalability (how big is the data set)
- Fitness score must be accurate (wrong fitness function guarantees bad performance)

Genetic Algorithm Conclusion

- GAs are a powerful, robust **optimization search technique**
- GAs will converge over successive generations toward a near global optimum via **selection**, **crossover**, and **mutation** operations
- GAs combine direction (selection and crossover) and chance (mutation) elements into a single **effective** and **efficient** search
- GAs can find **good solutions** in **reasonable time** (good enough and fast enough)

End of Lecture