Genetic Algorithm

Unit-7

General Introduction to GA's

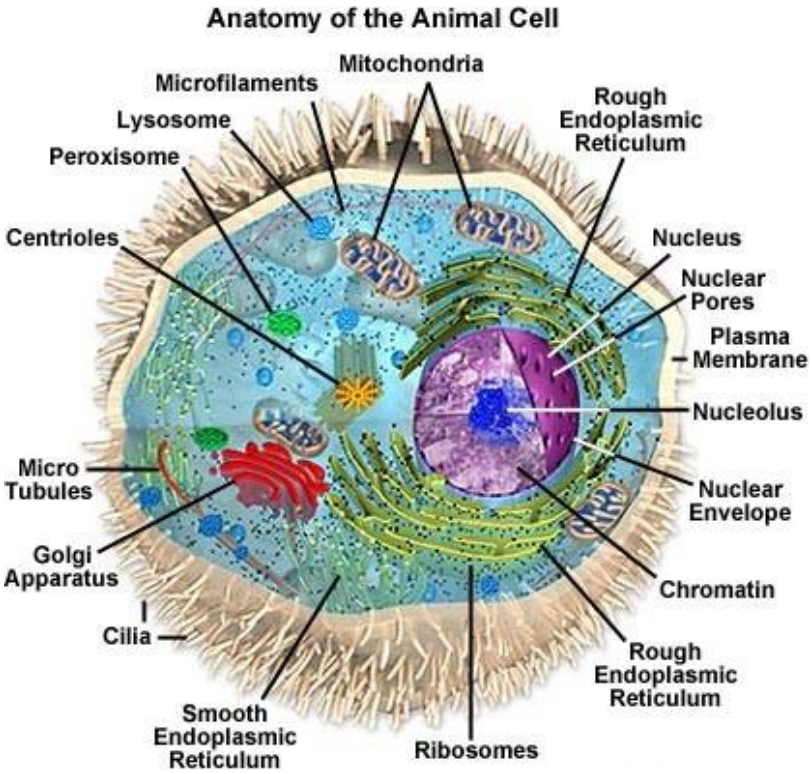
- Genetic algorithms (GA's) are a technique to solve problems which need optimization in search
- GA's are a subclass of Evolutionary Computing

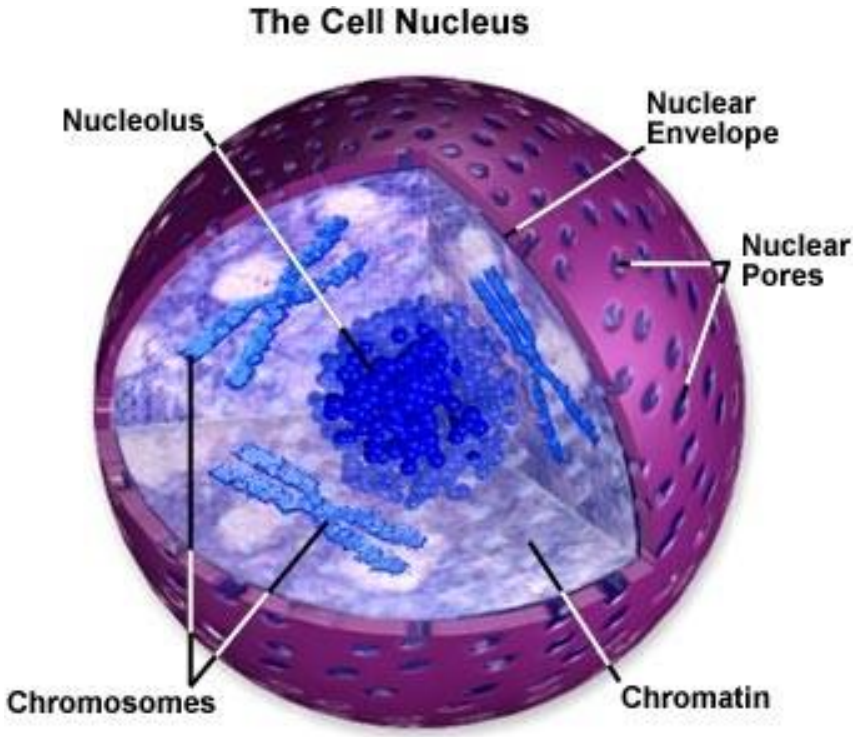
- GA's are based on Darwin's theory of evolution

- History of GA's
- Evolutionary computing evolved in the 1960's.
- GA's were created by John Holland in the mid-70's.

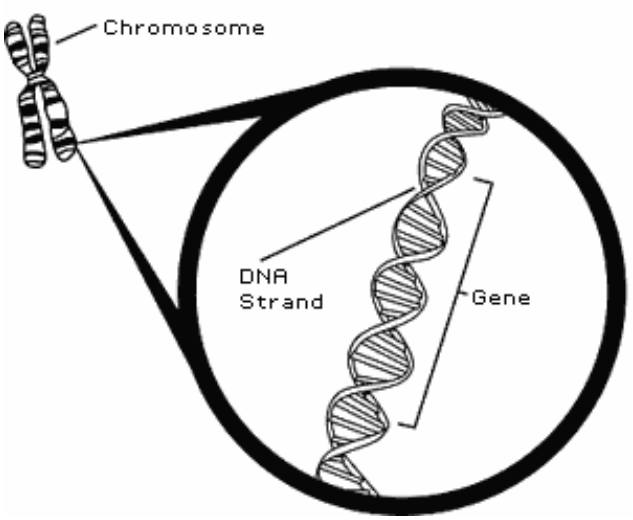
Biological Background (1) – The cell

- Every animal cell is a complex of many small "factories" working together
- The center of this all is the cell nucleus
- The nucleus contains the genetic information





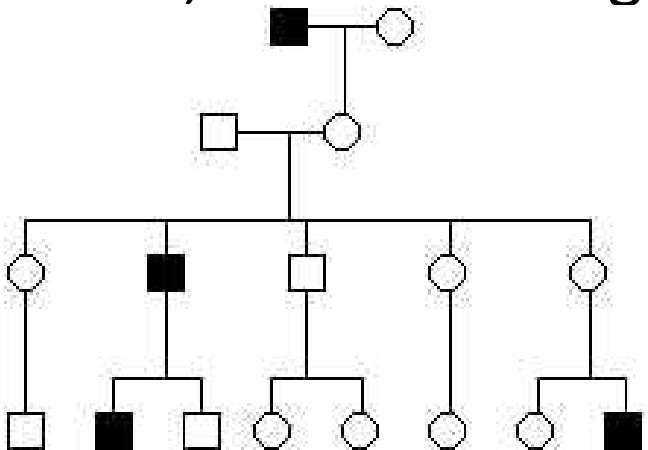
Biological Background (2) – Chromosomes



- Genetic information is stored in the **chromosomes**
- Each chromosome is build of **DNA**
- Chromosomes in humans form pairs
- There are 23 pairs
- The chromosome is divided in parts: **genes**
- Genes code for properties
- The posibilities of the genes for one property is called: **allele**
- Every gene has an unique position on the chromosome: **locus**

Biological Background (3) – Genetics

- The entire combination of genes is called **genotype**
- A genotype develops to a **phenotype**
- **Alleles** can be either dominant or recessive
- Dominant alleles will always express from the genotype to the phenotype
- Recessive alleles can survive in the population for many generations, without being expressed.

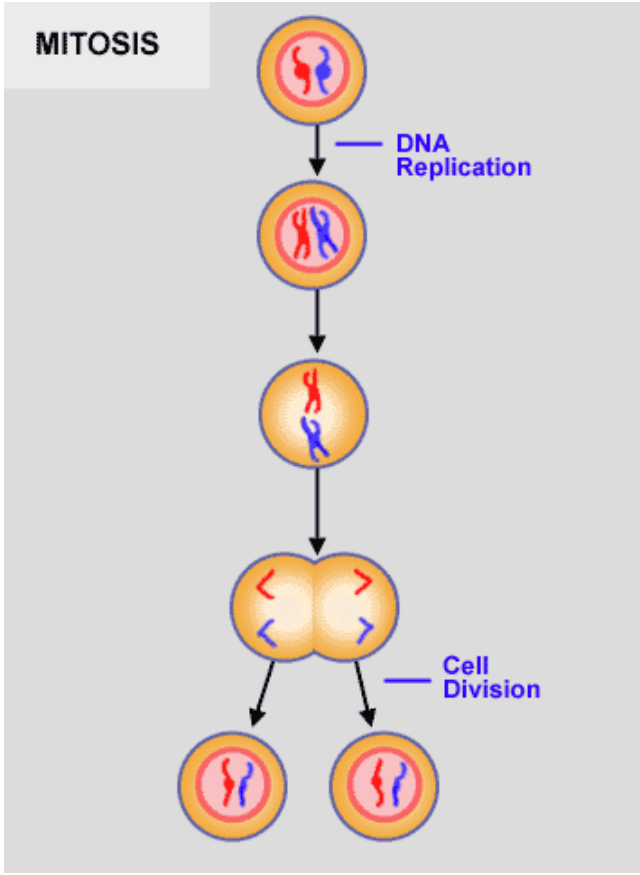


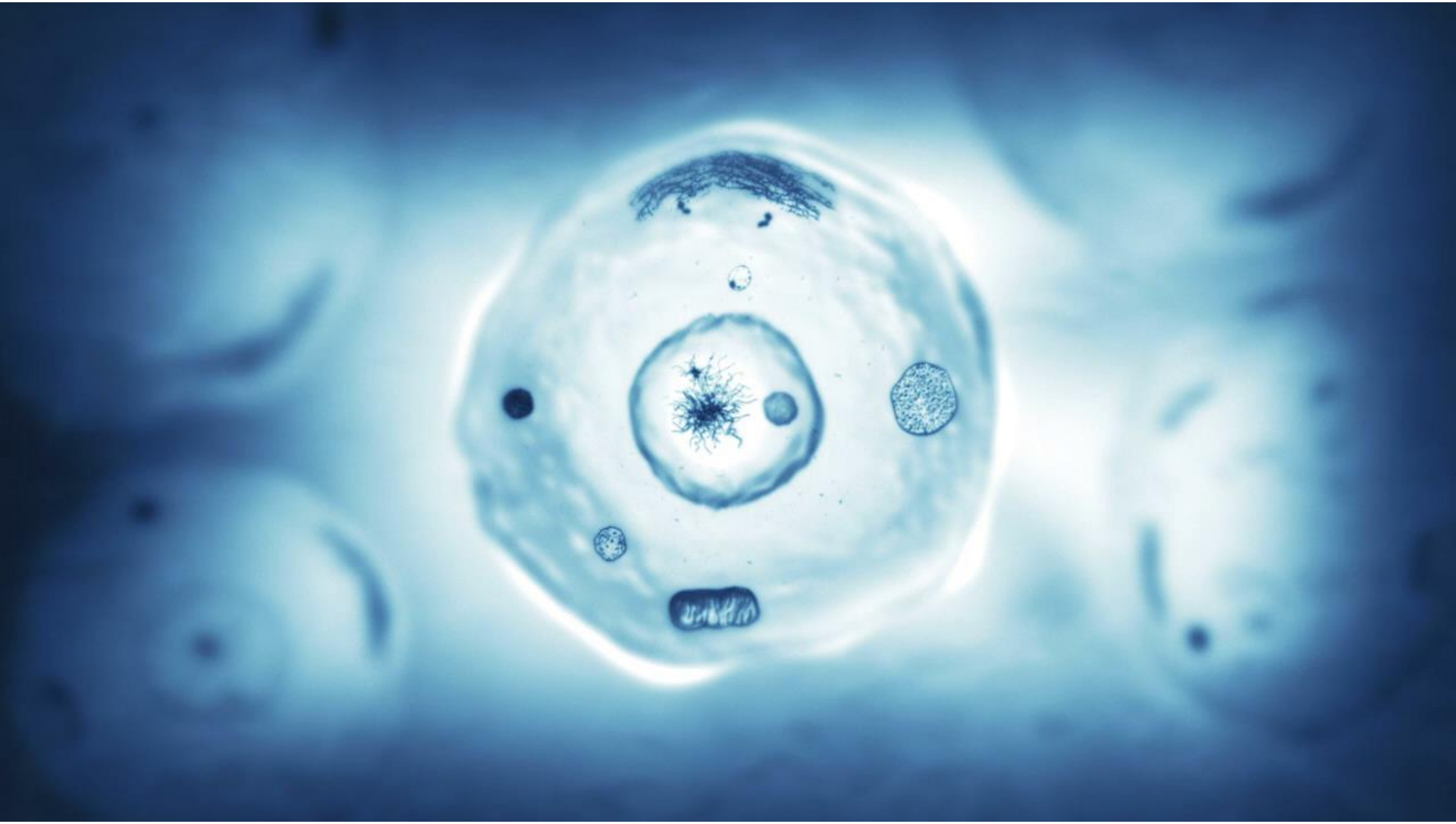
Biological Background (4) – Reproduction

- Reproduction of genetical information
- Mitosis
- Meiosis

- Mitosis is copying the same genetic information to new offspring: there is no exchange of information

- Mitosis is the normal way of growing of multicell structures, like organs.



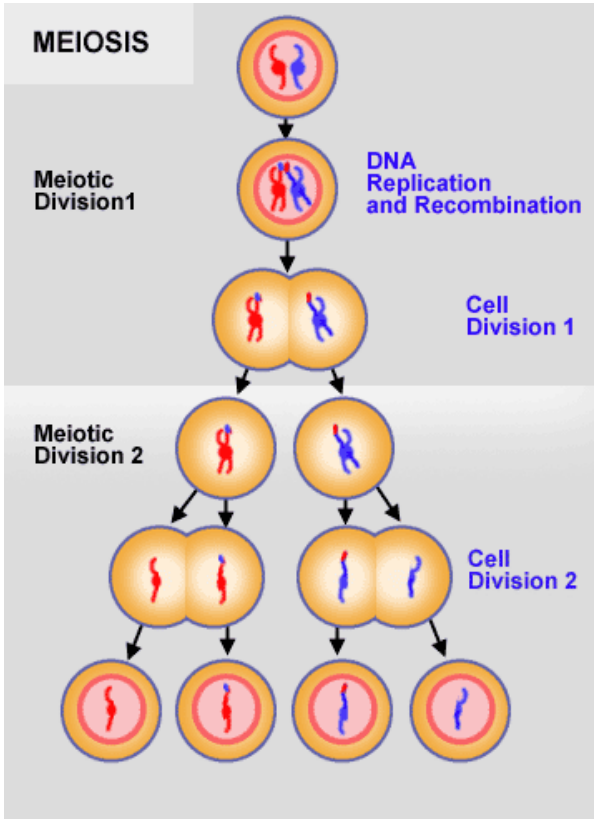



MRC Presents
Real Mitosis Under Microscopes
Medical
Research Community
03:30
Play

Biological Background (5) – Reproduction

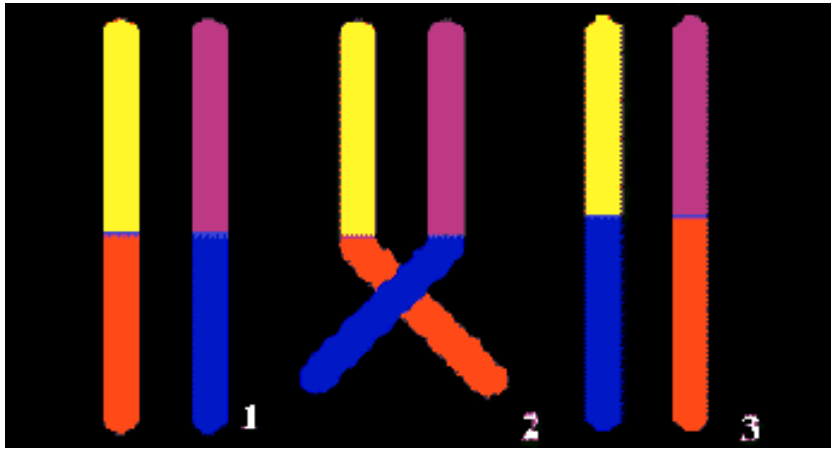
- Meiosis is the basis of sexual reproduction

- After meiotic division 2 gametes appear in the process
- In reproduction two gametes conjugate to a zygote which will become the new individual
- Hence genetic information is shared between the parents in order to create new offspring



Biological Background (6) – Reproduction

- During reproduction “errors” occur
- Due to these “errors” genetic variation exists
- Most important “errors” are:
  - **Recombination (cross-over)**
  - **Mutation**



Biological Background (7) – Natural selection

- The origin of species: “Preservation of favourable variations and rejection of unfavourable variations.”

- There are more individuals born than can survive, so there is a continuous struggle for life.

- Individuals with an advantage have a greater chance for survive: survival of the fittest.

Biological Background (8) – Natural selection

- Important aspects in natural selection are:
- adaptation to the environment
- isolation of populations in different groups which cannot mutually mate

- If small changes in the genotypes of individuals are expressed easily, especially in small populations, we speak of genetic drift

- Mathematical expresses as fitness: success in life

Evolution

- Evolution is the process by which modern organisms have descended from ancient ones
- Microevolution is evolution within a single population; (a population is a group of organisms that share the same gene pool). Often this kind of evolution is looked upon as change in gene frequency within a population

Evolution

- **Heredity**
- Information needs to be passed on from one generation to the next
- **Genetic Variation**
- There has to be differences in the characteristics of individuals in order for change to occur
- **Differential Reproduction**
- Some individuals need to (get to) reproduce more than others thereby increasing the frequency of their genes in the next generation

Evolution

- **Heredity -** Heredity is the transfer of characteristics (or traits) from parent to offspring through genes

Evolution

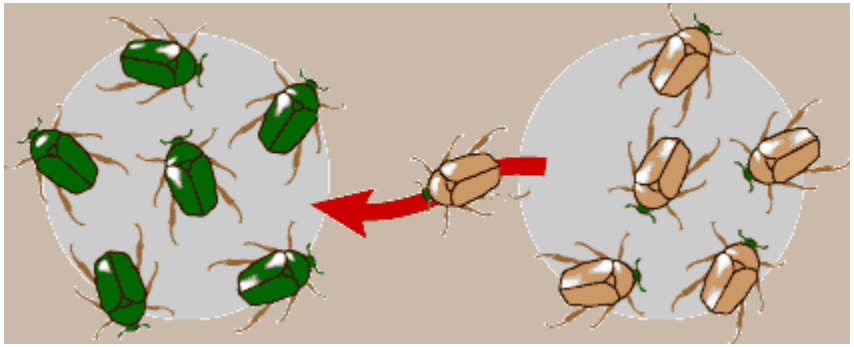
- **Genetic Variation -** Is about variety in the population and hence presence of genetic variation improves chances of coming up with “something new”
- The primary mechanisms of achieving genetic variation are:
- Mutations
- Gene Flow
- Sexual Reproduction

Evolution

- Mutations
- It is a random change in DNA
- It can be beneficial, neutral or harmful to the organism
- Not all mutations matter to evolution

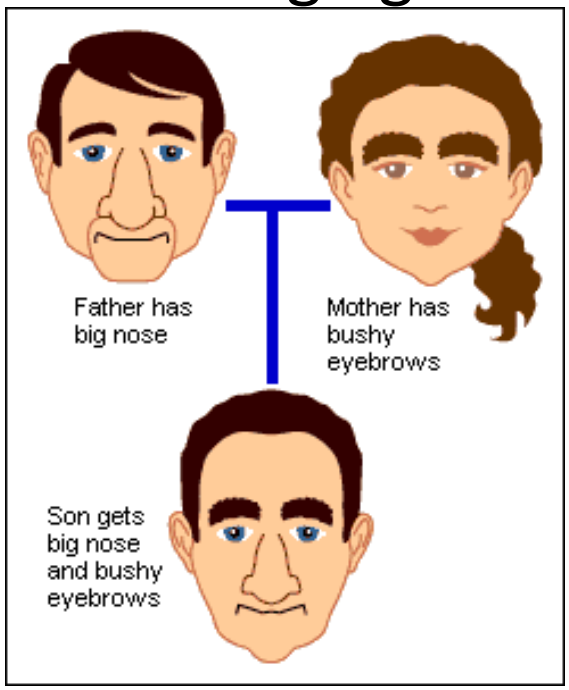
Evolution

- Gene Flow
- Migration of genes from one population to another
- If the migrating genes did not exist previously in the incident population then such a migration adds to the gene pool



Evolution

- Sexual Reproduction
- This type of producing young can introduce new gene combinations through genetic shuffling



Evolution

- Differential Reproduction
- As the genes show up as traits (phenotype) the individuals get affected by what is around; some die young while others live
- Those who live compete for mates; only the winners pass on their gene to the next generation
- In some sense the fitter (with respect to the current environment) gets to leave more of his/her genes in the next population; often the term fitness is used to describe the relative ability of individuals to pass on their genes

What is Evolutionary Computation?

- An abstraction from the theory of biological evolution that is used to create optimization procedures or methodologies, usually implemented on computers, that are used to solve problems.
- Evolution has optimized biological processes; therefore
- Adoption of the evolutionary paradigm to computation and other problems can help us find optimal solutions

What is Evolutionary Computation?

- Nature has always been a great source of inspiration to all mankind.
- Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics.
- GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.

Components of Evolutionary Computing

- Genetic Algorithms
  - invented by John Holland (University of Michigan) in the 1960's
- Evolution Strategies
  - invented by Ingo Rechenberg (Technical University Berlin) in the 1960's
- Started out as individual developments, but have begun to converge in the last few years

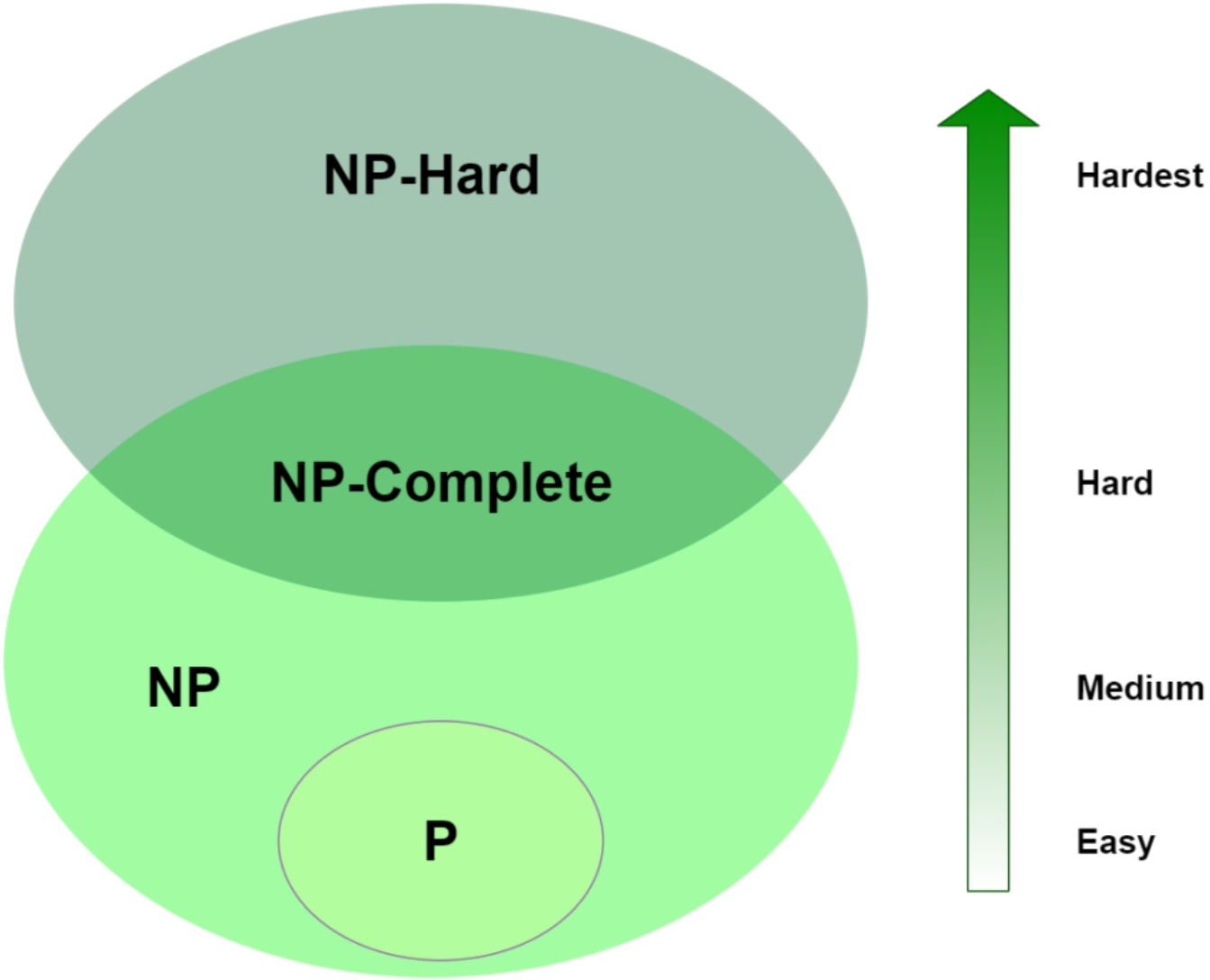
GA – Motivation

- Genetic Algorithms have the ability to deliver a “good-enough” solution “fast-enough”.
- This makes genetic algorithms attractive for use in solving optimization problems.

GA – Motivation

- In computer science, there is a large set of problems, which are **NP-Hard**.
- What this essentially means is that, even the most powerful computing systems take a very long time (even years!) to solve that problem.
- In such a scenario, GAs prove to be an efficient tool to provide **usable near-optimal solutions** in a short amount of time.

GA – Motivation



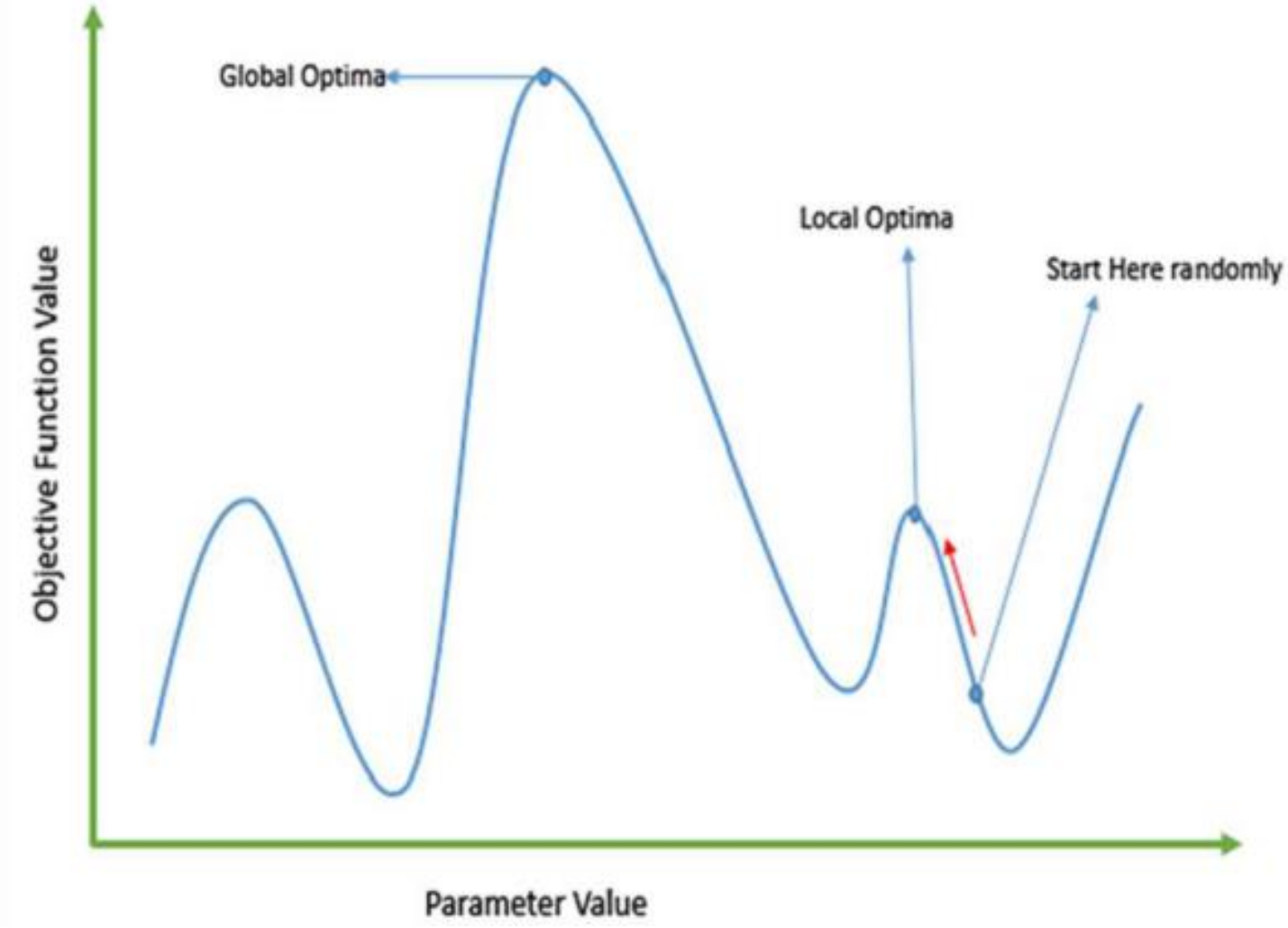
GA – Motivation

- Failure of Gradient Based Methods
- Traditional calculus based methods work by starting at a random point and by moving in the direction of the gradient, till we reach the top of the hill.
- This technique is efficient and works very well for single-peaked objective functions like the cost function in linear regression.

GA – Motivation

- But, in most real-world situations, we have a very complex problem called as landscapes, which are made of many peaks and many valleys, which causes such methods to fail, as they suffer from an inherent tendency of getting stuck at the local optima as shown in the following figure.

GA – Motivation

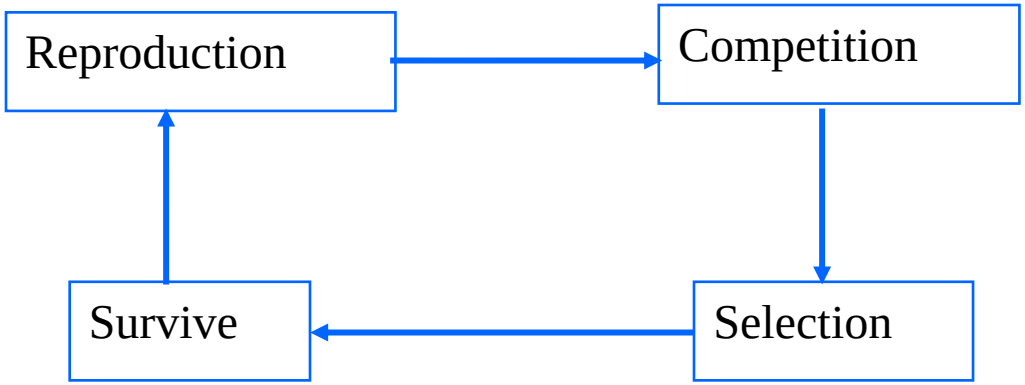


Why Genetic Algorithm?

- Some difficult problems like the Travelling Salesperson Problem (TSP)
- Real-world applications like path finding and VLSI Design.
- Now imagine that you are using your GPS Navigation system, and it takes a few minutes (or even a few hours) to compute the “optimal” path from the source to destination.
- Delay in such real world applications is not acceptable and therefore a “good-enough” solution, which is delivered “fast” is what is required.

Darwinian Paradigm

- Intrinsically a robust search and optimization mechanism

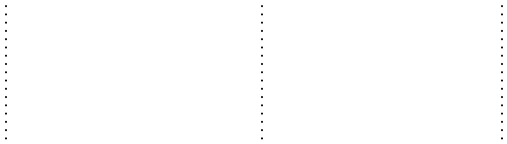


Genetic Algorithm (Holland)

- heuristic method based on ‘ survival of the fittest ’
- useful when search space very large or too complex for analytic treatment
- in each iteration (generation) possible solutions or individuals represented as strings of numbers

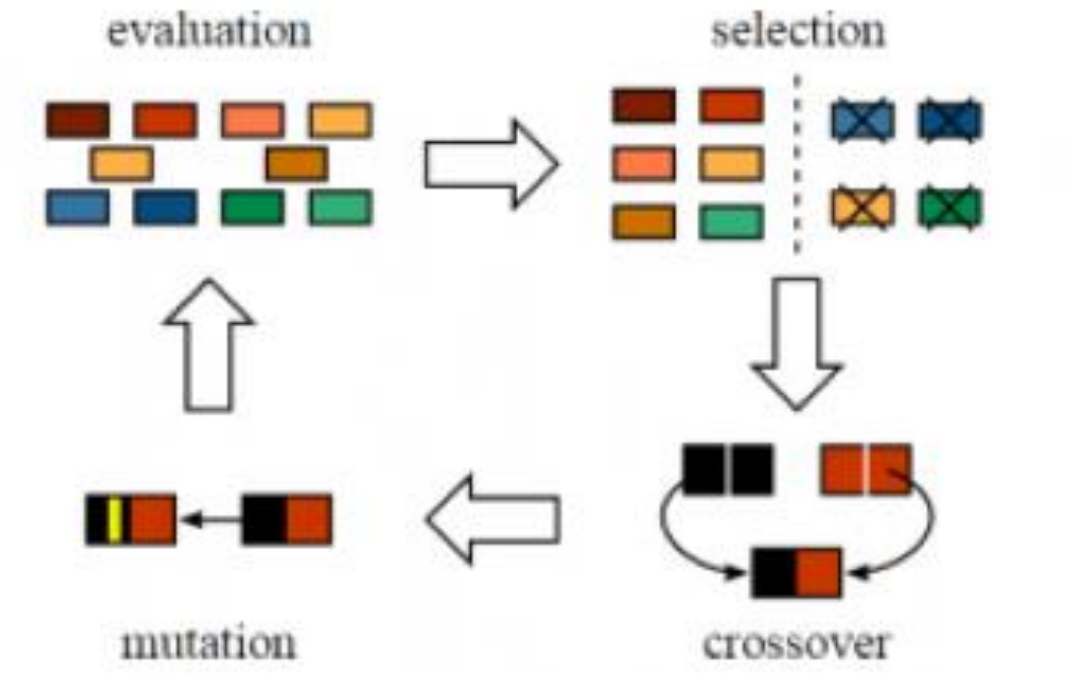
3021 3058 3240

00010101 00111010 11110000
00010001 00111011 10100101
00100100 10111001 01111000



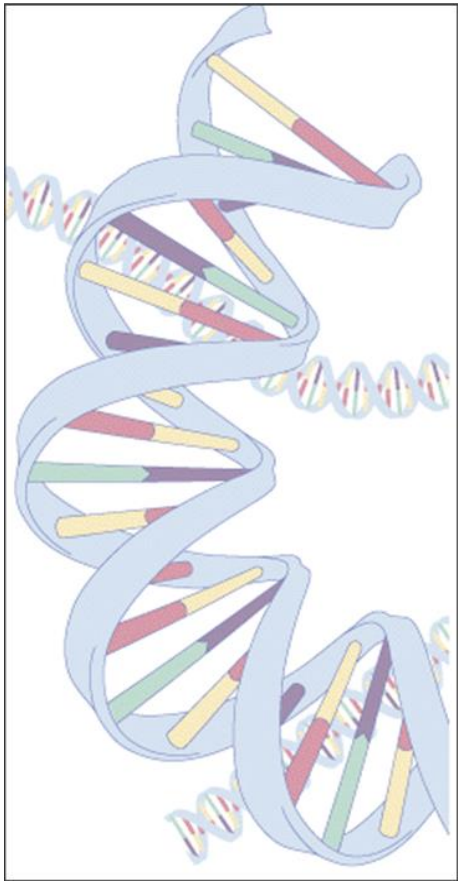
11000101 01011000 01101010

Darwinian Genetic Paradigm



What is Genetic Algorithm?

- Follows steps inspired by the biological processes of evolution.
- Follow the idea of SURVIVAL OF THE FITTEST- Better and better solutions evolve from previous generations until a near optimal solution is obtained.
- Genetic Algorithms are often used to improve the performance of other AI methods.
- The method learns by producing offspring that are better and better as measured by a fitness function.



Genetic Algorithm

- all individuals in population evaluated by fitness function
- individuals allowed to reproduce (selection), crossover, mutate

Applications of GA

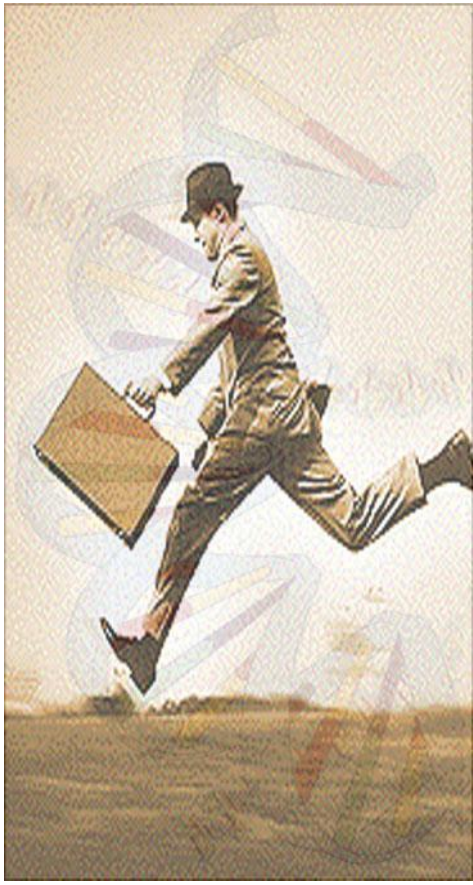
Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

Representations

- Genetic programming can be used to evolve S-expressions, which can be used as LISP programs to solve problems.
- A string of bits is known as a **chromosome.**
- Each bit is known as a **gene.**
- Chromosomes can be combined together to form **creatures**.
- We will see how genetic algorithms can be used to solve mathematical problems.

Knapsack Problem

- You are going on a picnic.
- And have a number of items that you could take along.
- Each item has a weight and a benefit or value.
- You can take one of each item at most.
- There is a capacity limit on the weight you can carry.
- You should carry items with max. values.



Example: (Knapsack Problem)

- **Item: 1 2 3 4 5 6 7**
- **Benefit: 5 8 3 2 7 9 4**
- **Weight: 7 8 4 10 4 6 4**
- **Knapsack holds a maximum of 22 pounds**
- **Fill it to get the maximum benefit**

Outline of Basic Genetic Algorithm

1. **[Start]**
   - ✓ Encoding: represent the individual.
   - ✓ Generate random population of *n* chromosomes (suitable solutions for the problem).
2. **[Fitness]** Evaluate the fitness of each chromosome.
3. **[New population]** repeating following steps until the new population is complete.
4. **[Selection]** Select the best two parents.
5. **[Crossover]** cross over the parents to form a new offspring (children).
6. **[Mutation]** With a mutation probability.
7. **[Accepting]** Place new offspring in a new population.
8. **[Replace]** Use new generated population for a further run of algorithm.
9. **[Test]** If the end condition is satisfied, then **stop**.
10. **[Loop]** Go to step **2** .

Basic Steps

- Encoding: 0 = not exist, 1 = exist in the Knapsack

  Chromosome: 1010110

Item.	1	2	3	4	5	6	7
Chro	1	0	1	0	1	1	0
Exist?	y	n	y	n	y	y	n

  => Items taken: 1, 3 , 5, 6.

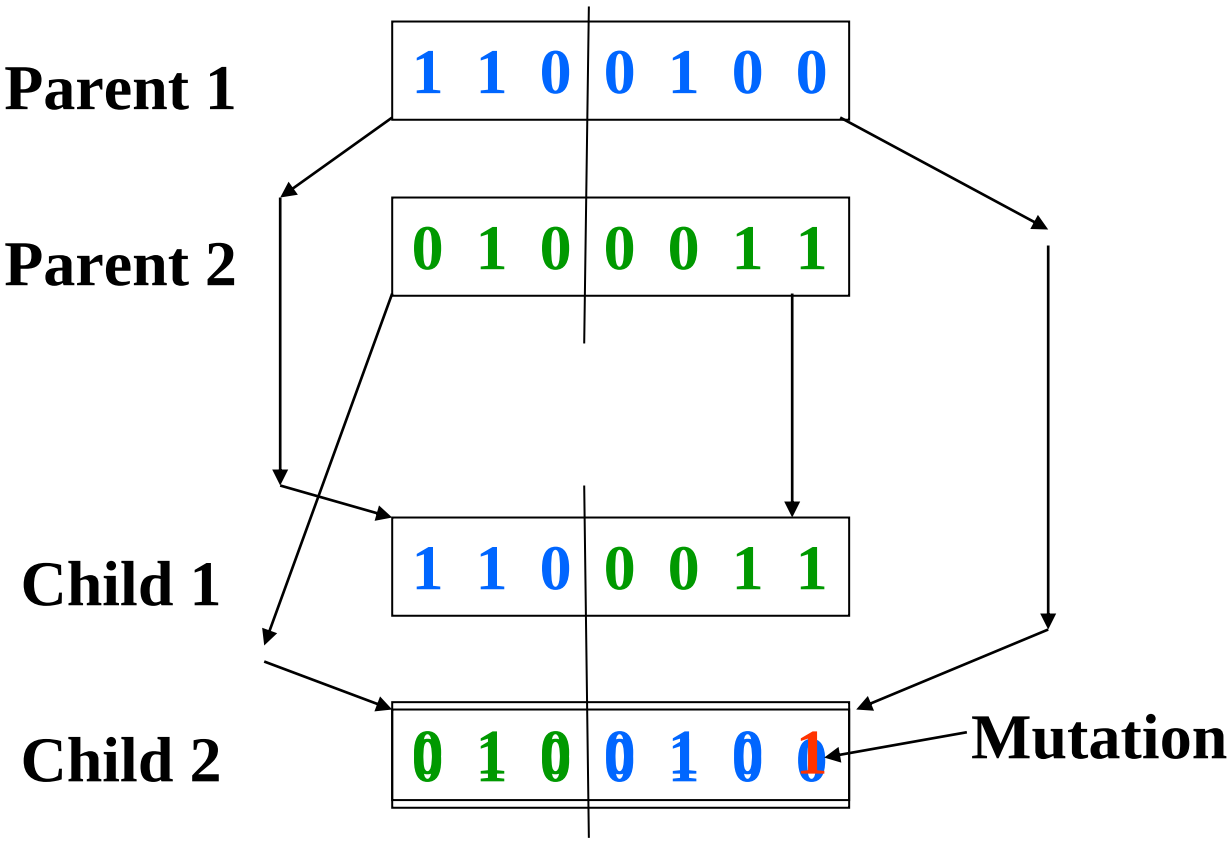
- Generate random population of *n* chromosomes:

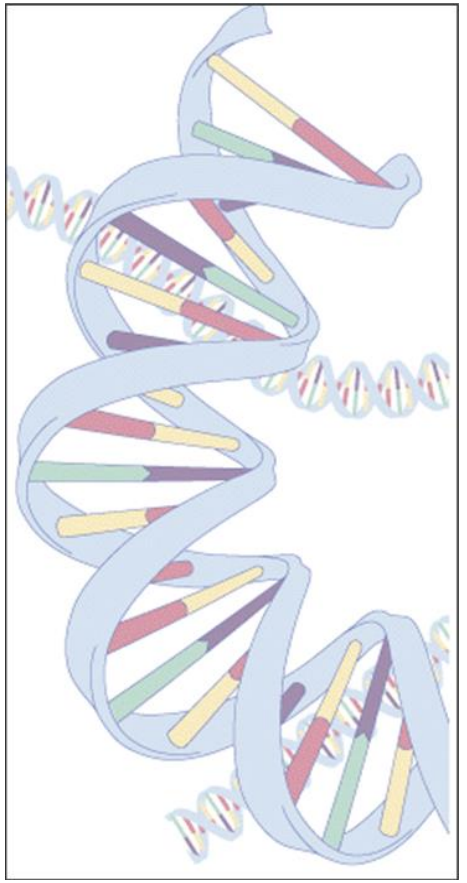
  a) 0101010

  b) 1100100

  c) 0100011

Crossover & Mutation





Accepting, Replacing & Testing

✓Place new offspring in a new population.

✓Use new generated population for a further run of algorithm.

✓If the end condition is satisfied, then **stop**. End conditions:

- Number of populations.
- Improvement of the best solution.

✓Else, return to step 2 **[Fitness].**

The Algorithm

GA(*Fitness, Fitness_threshold, p, r, m*)

- *Initialize*: $P \leftarrow p$ random hypotheses
- *Evaluate*: for each *h* in *P*, compute *Fitness*(h)
- While [$\max_h$ *Fitness*(*h*)] < *Fitness_threshold*
  1. *Select*: Select (1 – *r*) members of *P* to add to P_S based on fitness

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{P} Fitness(h_j)}$$

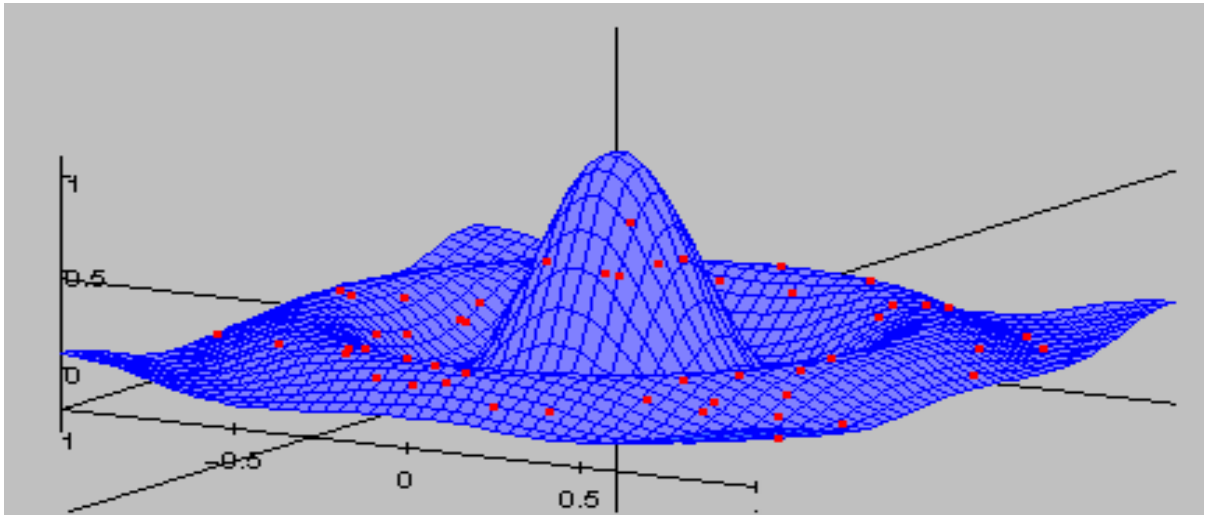
  1. *Crossover*: Probabilistically select pairs of hypotheses from *P*. For each pair, $<h_1, h_2>$, produce two offspring by applying the Crossover operator. Add all offspring to P_S
  2. *Mutate*: Invert a randomly selected bit in $m \cdot p$ random members of P_S
  3. *Update*: $P \leftarrow P_S$
  4. *Evaluate*: for each *h* in *P*, compute *Fitness*(*h*)
- Return the hypothesis from *P* that has the highest fitness

Fitness Function

- Fitness is an important concept in genetic algorithms.
- The fitness of a chromosome determines how likely it is that it will reproduce.
- Fitness is usually measured in terms of how well the chromosome solves some goal problem.
  - E.g., if the genetic algorithm is to be used to sort numbers, then the fitness of a chromosome will be determined by how close to a correct sorting it produces.
- Fitness can also be subjective (aesthetic)
- For each individual in the population, evaluate its relative fitness
- For a problem with *m* parameters, the fitness can be plotted in an *m*+1 dimensional space

Sample Search Space

- A randomly generated population of individuals will be randomly distributed throughout the search space

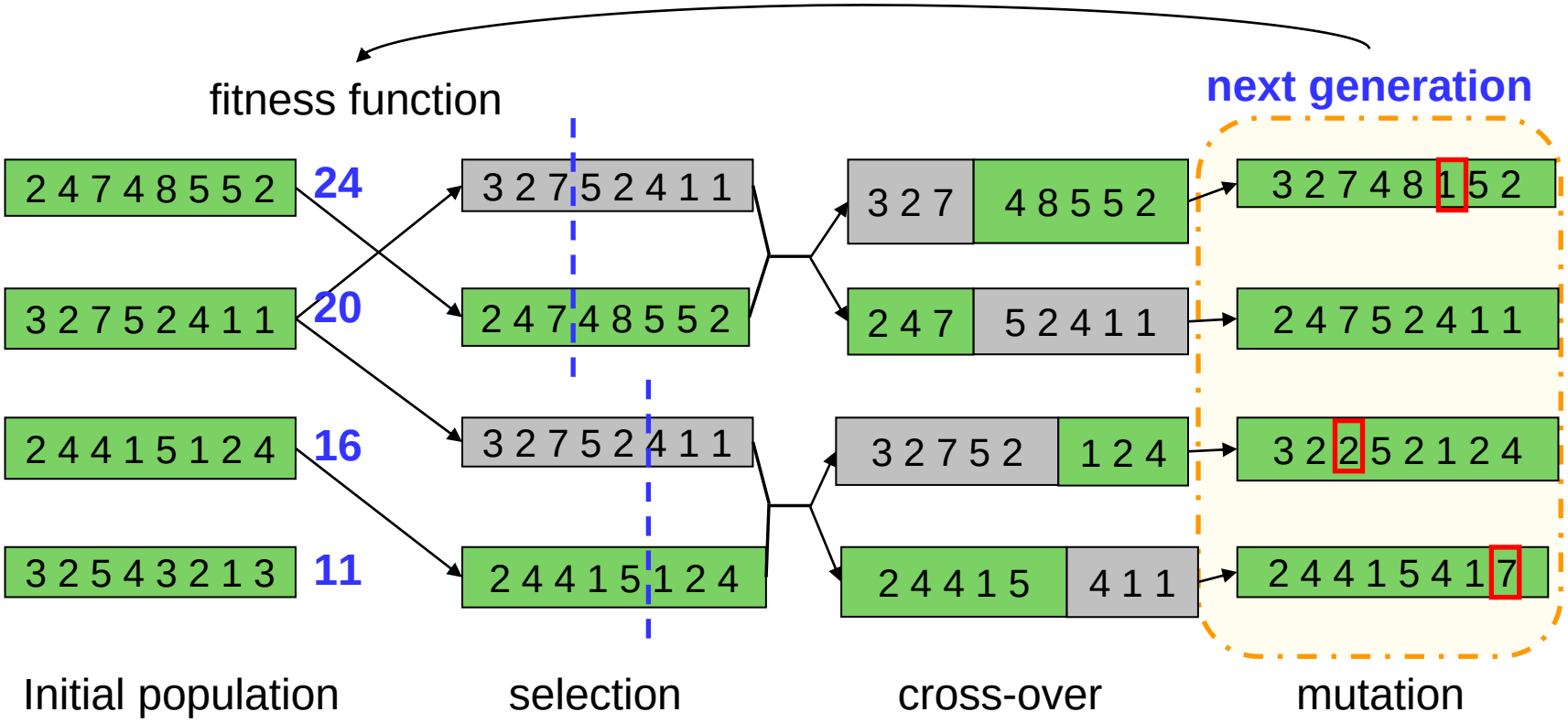


Generations

- As each new generation of *n* individuals is generated, they replace their parent generation

- To achieve the desired results, 500 to 5000 generations are required

Generation


fitness function
next generation
2 4 7 4 8 5 5 2
24
3 2 7 5 2 4 1 1
20
2 4 4 1 5 1 2 4
16
3 2 5 4 3 2 1 3
11
3 2 7 5 2 4 1 1
2 4 7 4 8 5 5 2
3 2 7 5 2 4 1 1
2 4 4 1 5 1 2 4
3 2 7
4 8 5 5 2
2 4 7
5 2 4 1 1
3 2 7 5 2
1 2 4
2 4 4 1 5
4 1 1
3 2 7 4 8 1 5 2
2 4 7 5 2 4 1 1
3 2 2 5 2 1 2 4
2 4 4 1 5 4 1 7
Initial population
selection
cross-over
mutation

Reproduction

- Crossover
  - Two parents produce two offspring
  - There is a chance that the chromosomes of the two parents are copied unmodified as offspring
  - There is a chance that the chromosomes of the two parents are randomly recombined (crossover) to form offspring
  - Generally the chance of crossover is between 0.6 and 1.0
- Mutation
  - There is a chance that a gene of a child is changed randomly
  - Generally the chance of mutation is low (e.g. 0.001)

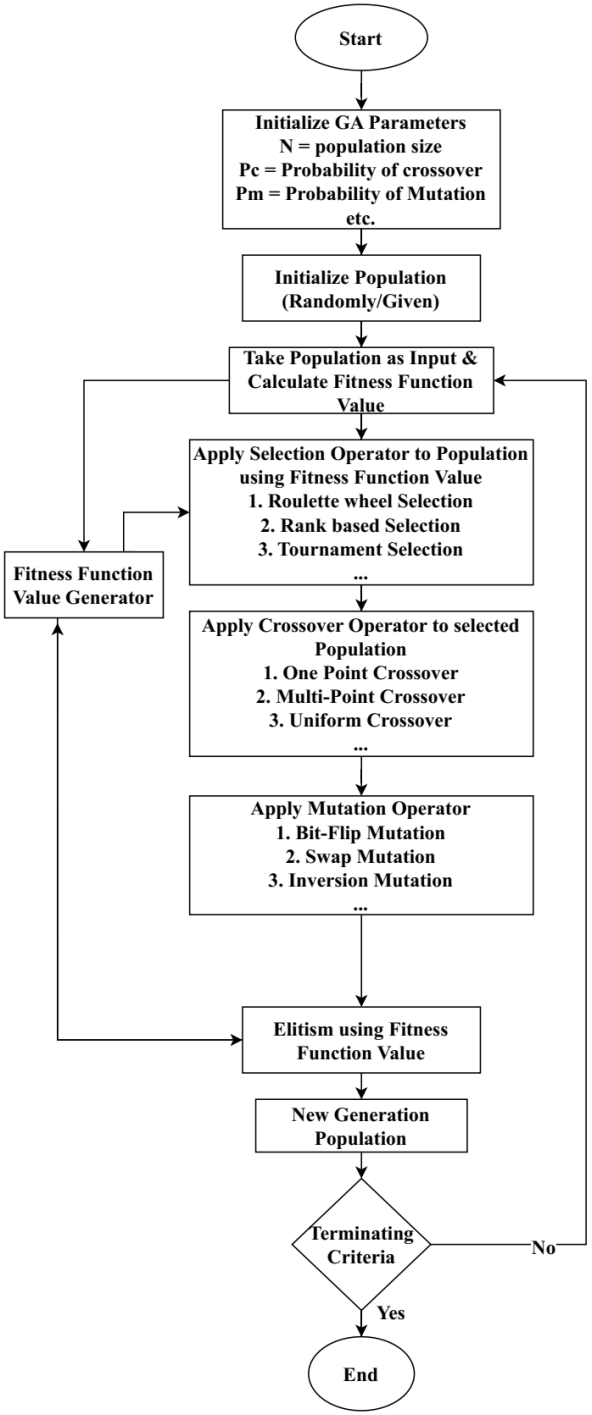
Genetic Algorithm Operators

- Selection Operator
- Crossover Operator
- Mutation Operator
- Elitism Operator

Elitism Operator

- Apply elitism by selecting the best performing individuals from the current generation to carry over unchanged into the next generation.
- This ensures that the best solutions are not lost during the evolutionary process.

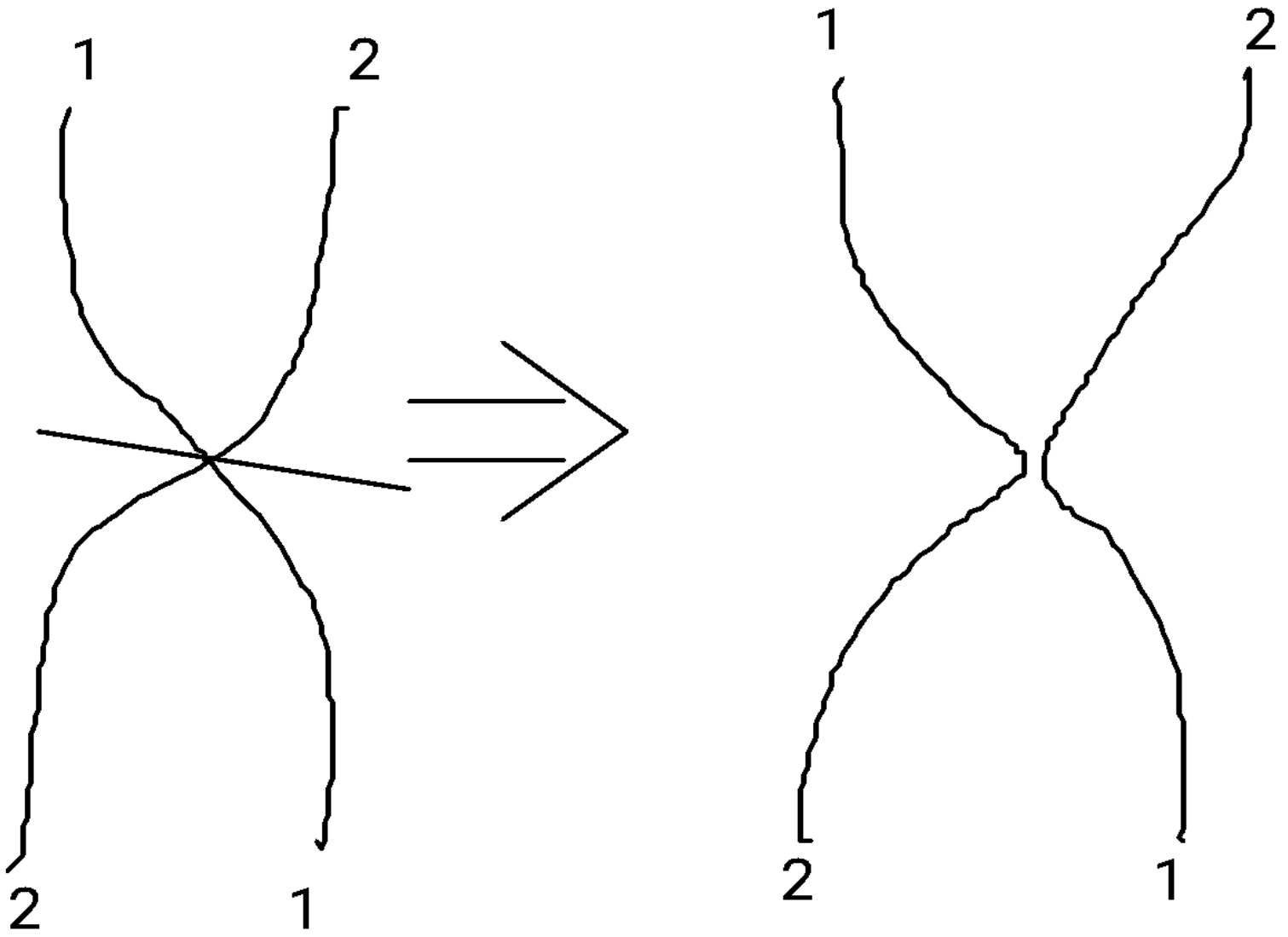
GA Flow C


Start
Initialize GA Parameters
N = population size
Pc = Probability of crossover
Pm = Probability of Mutation
etc.
Initialize Population
(Randomly/Given)
Take Population as Input &
Calculate Fitness Function
Value
Apply Selection Operator to Population
using Fitness Function Value
1. Roulette wheel Selection
2. Rank based Selection
3. Tournament Selection
...
Fitness Function
Value Generator
Apply Crossover Operator to selected
Population
1. One Point Crossover
2. Multi-Point Crossover
3. Uniform Crossover
...
Apply Mutation Operator
1. Bit-Flip Mutation
2. Swap Mutation
3. Inversion Mutation
...
Elitism using Fitness
Function Value
New Generation
Population
Terminating
Criteria
No
Yes
End

Crossover

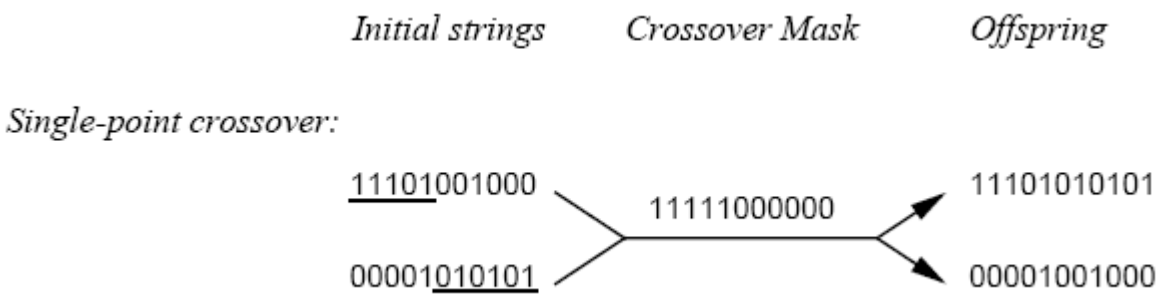
- Crossover
  - Generating offspring from two selected parents
    - Single point crossover
    - Two point crossover (Multi point crossover)
    - Uniform crossover

One-point crossover - Nature



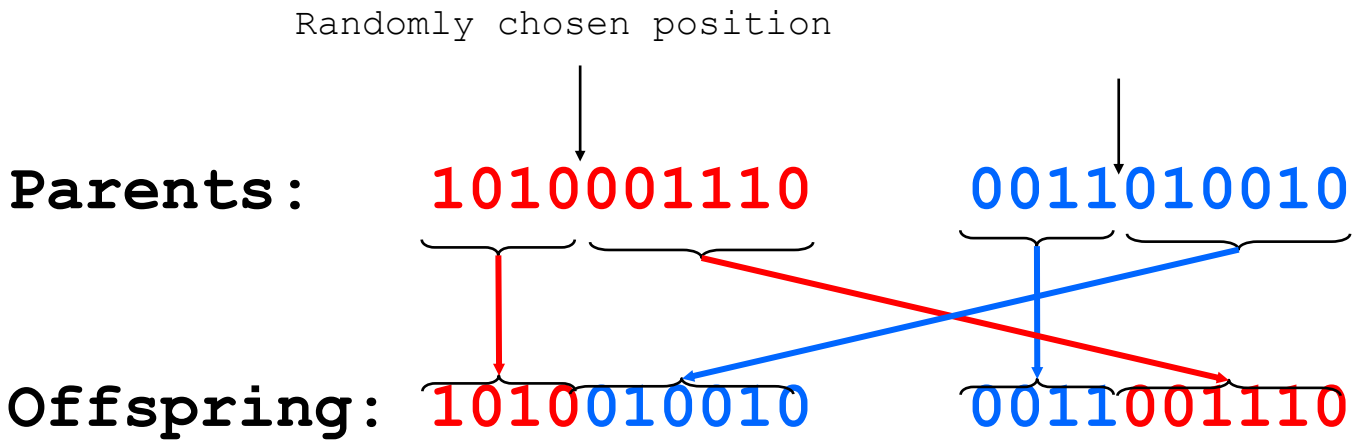
Single-Point Crossover (One-point crossover)

- **Crossover is applied as follows:**
  1) Select a random crossover point.
  2) Break each chromosome into two parts, splitting at the crossover point.
  3) Recombine the broken chromosomes by combining the front of one with the back of the other, and vice versa, to produce two new chromosomes.



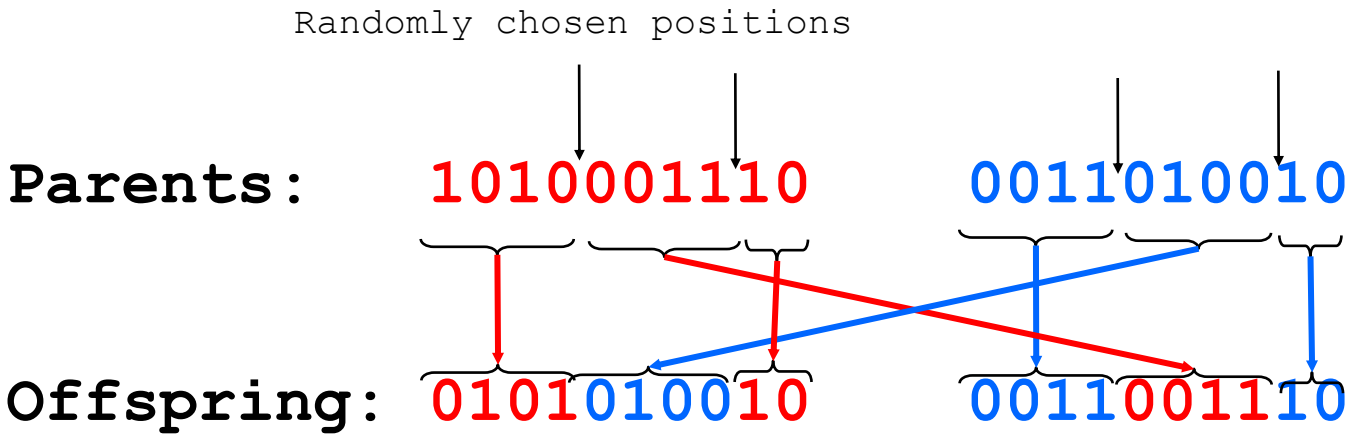
One-point crossover

- Randomly one position in the chromosomes is chosen
- Child 1 is head of chromosome of parent 1 with tail of chromosome of parent 2
- Child 2 is head of 2 with tail of 1



Two-point crossover

- Randomly two positions in the chromosomes are chosen
- Avoids that genes at the head and genes at the tail of a chromosome are always split when recombined

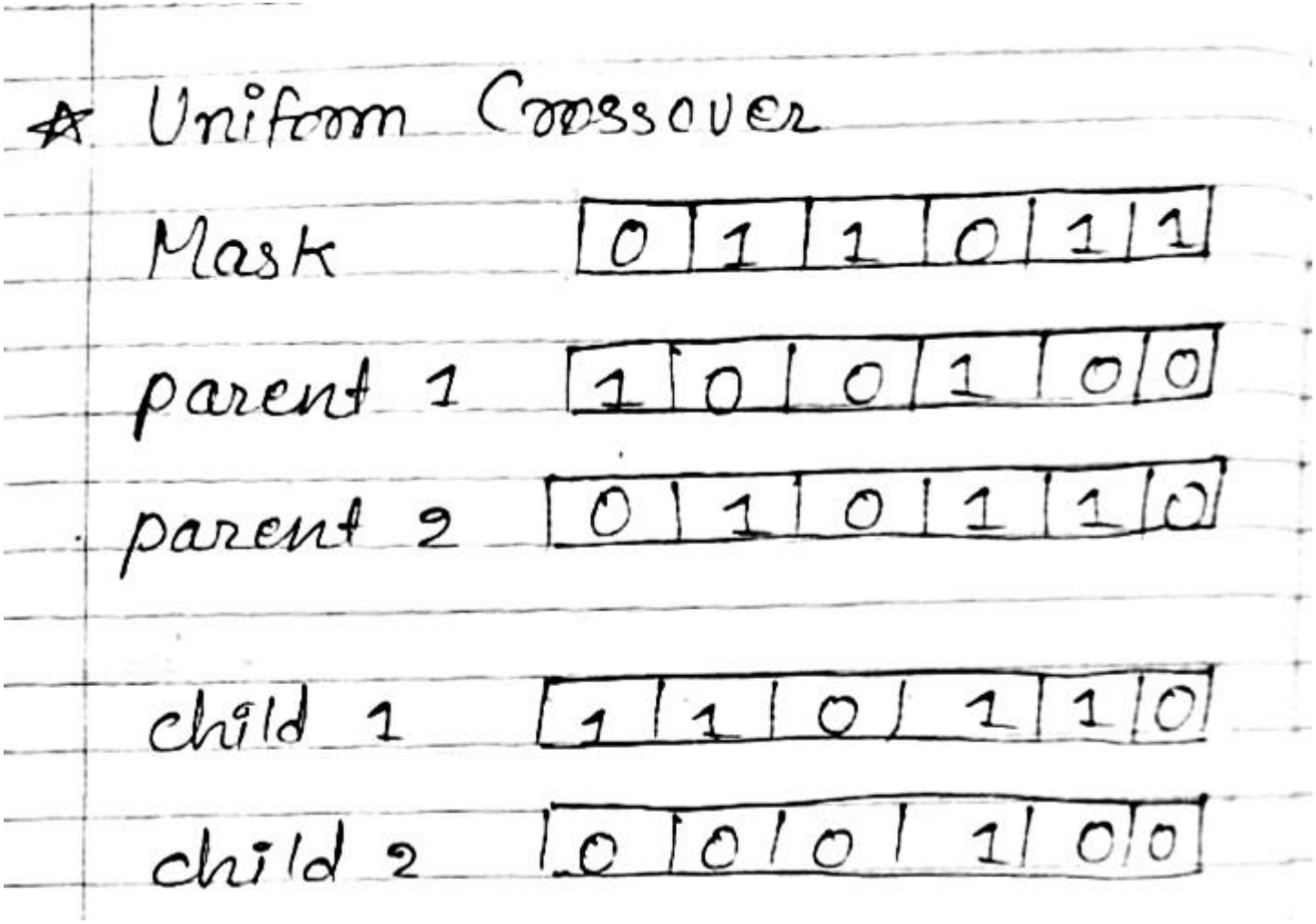


Uniform crossover

- A random mask is generated
- The mask determines which bits are copied from one parent and which from the other parent
- Bit density in mask determines how much material is taken from the other parent (takeover parameter)

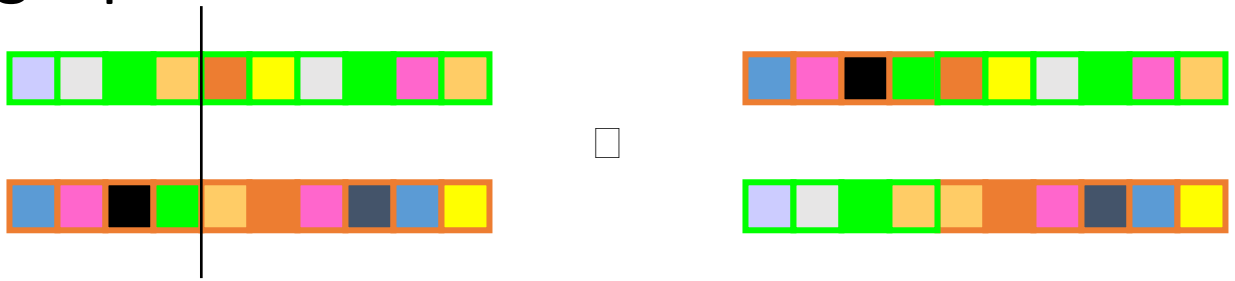
Mask:	**0110011000**	**(Randomly generated)**
Parents:	**1010001110**	**0011010010**
Offspring:	**0011001010**	**1010010110**

☆ Uniform Crossover

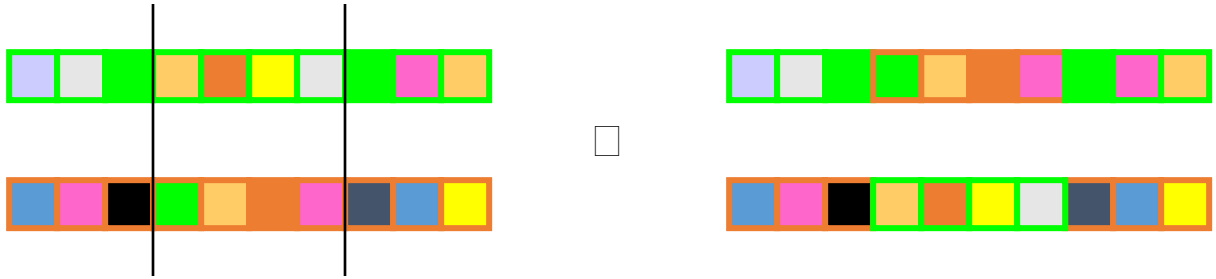


Operators comparison

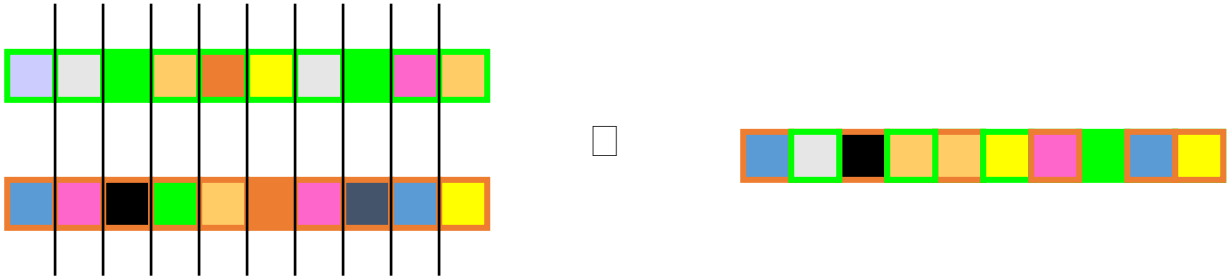
- Single point crossover



- Two point crossover (Multi point crossover)



- Uniform crossover
- Is uniform crossover better than single crossover point?
  - Trade off between
    - Exploration: introduction of new combination of features
    - Exploitation: keep the good features in the existing solution

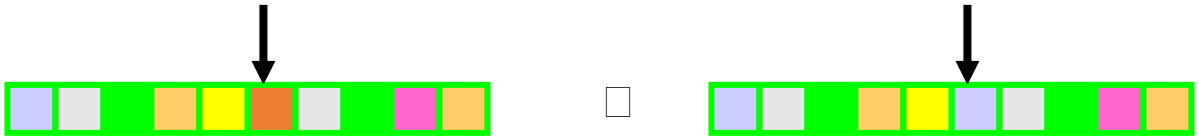


Problems with crossover

- Depending on coding, simple crossovers can have high chance to produce illegal offspring
  - E.g. in TSP with simple binary or path coding, most offspring will be illegal because not all cities will be in the offspring and some cities will be there more than once
- Uniform crossover can often be modified to avoid this problem
  - E.g. in TSP with simple path coding:
    - Where mask is 1, copy cities from one parent
    - Where mask is 0, choose the remaining cities in the order of the other parent

Mutation

- Generating new offspring from single parent

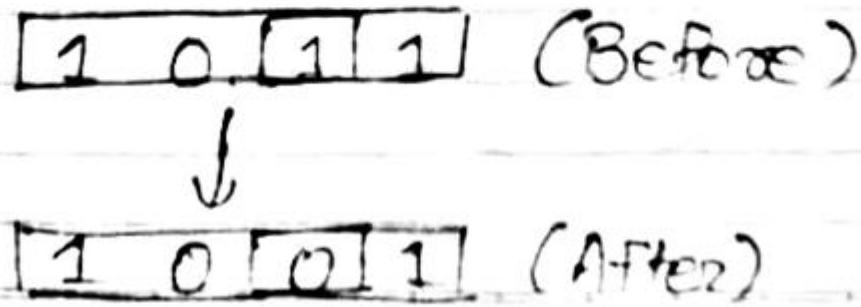


- Maintaining the diversity of the individuals
  - Crossover can only explore the combinations of the current gene pool
  - Mutation can “generate” new genes

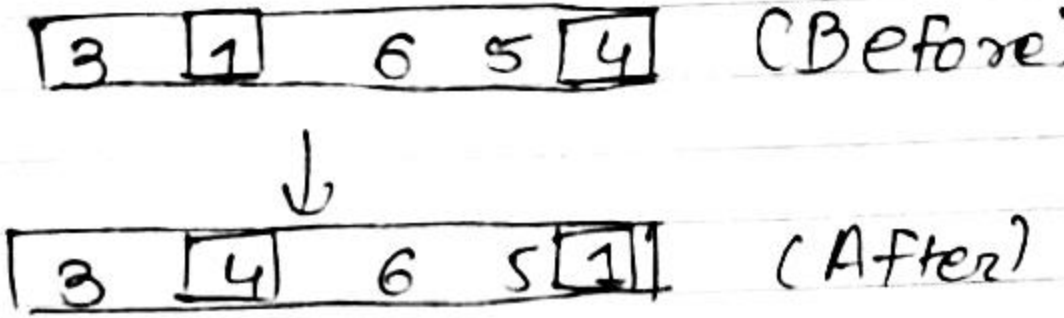
Mutation

Types → Bit Flip Mutation
→ Swap Mutation
→ Inversion Mutation

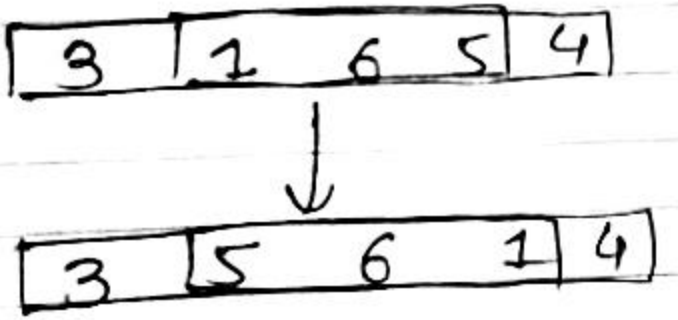
⇒ Bit Flip Mutation



⇒ Swap Mutation



⇒ Inversion Mutation



Reproduction Operators

- Control parameters: **population size, crossover/mutation probability**
  - Problem specific
  - Increase population size
    - Increase diversity and computation time for each generation
  - Increase crossover probability
    - Increase the opportunity for recombination but also disruption of good combination
  - Increase mutation probability
    - Closer to randomly search
    - Help to introduce new gene or reintroduce the lost gene
- Varies the population
  - Usually using crossover operators to recombine the genes to generate the new population, then using mutation operators on the new population

Parent/Survivor Selection

- Strategies
  - Survivor selection
    - Always keep the best one
    - Elitist: deletion of the K worst
    - Probability selection : inverse to their fitness
    - Etc.

Parent/Survivor Selection

- Too strong fitness selection bias can lead to sub-optimal solution
- Too little fitness bias selection results in unfocused and meandering search

Parent selection

Chance to be selected as parent proportional to fitness

- Roulette wheel

To avoid problems with fitness function

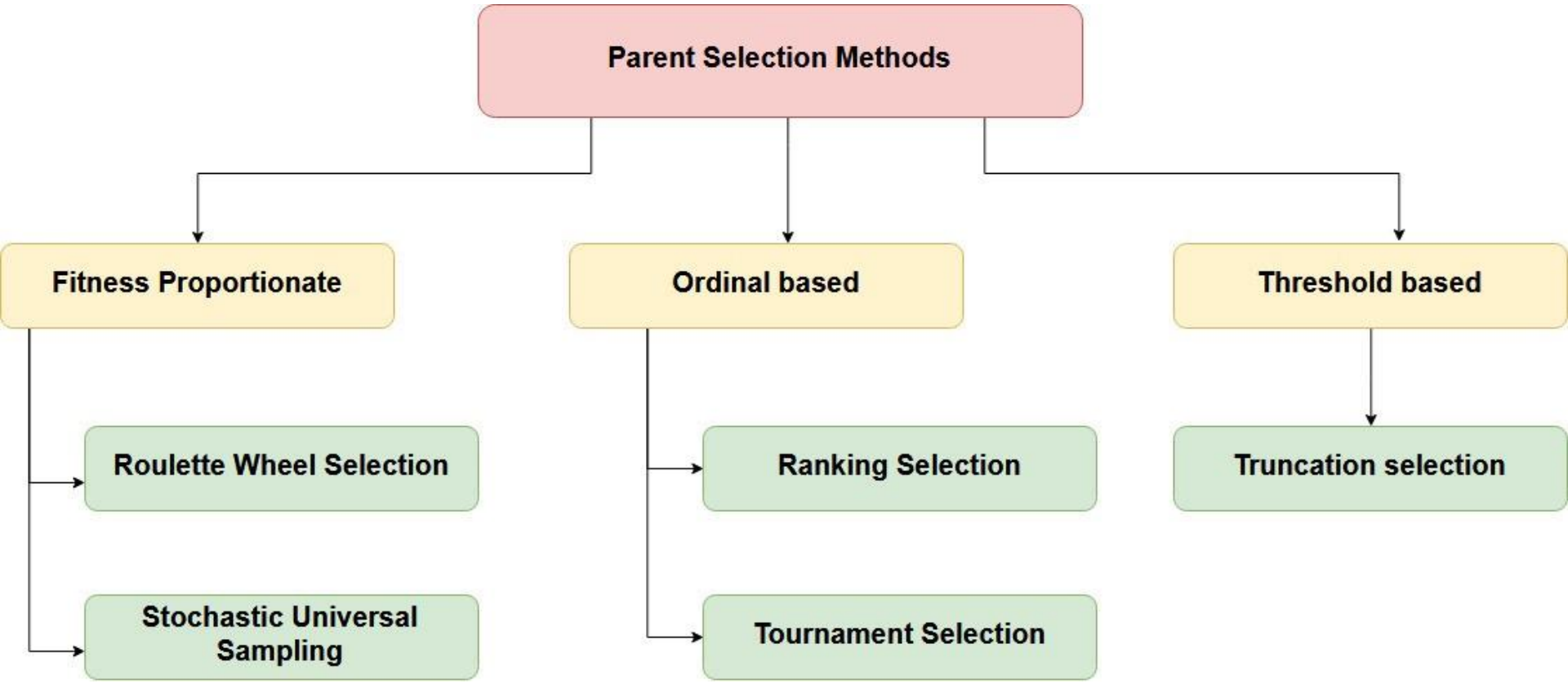
- Tournament

Not a very important parameter

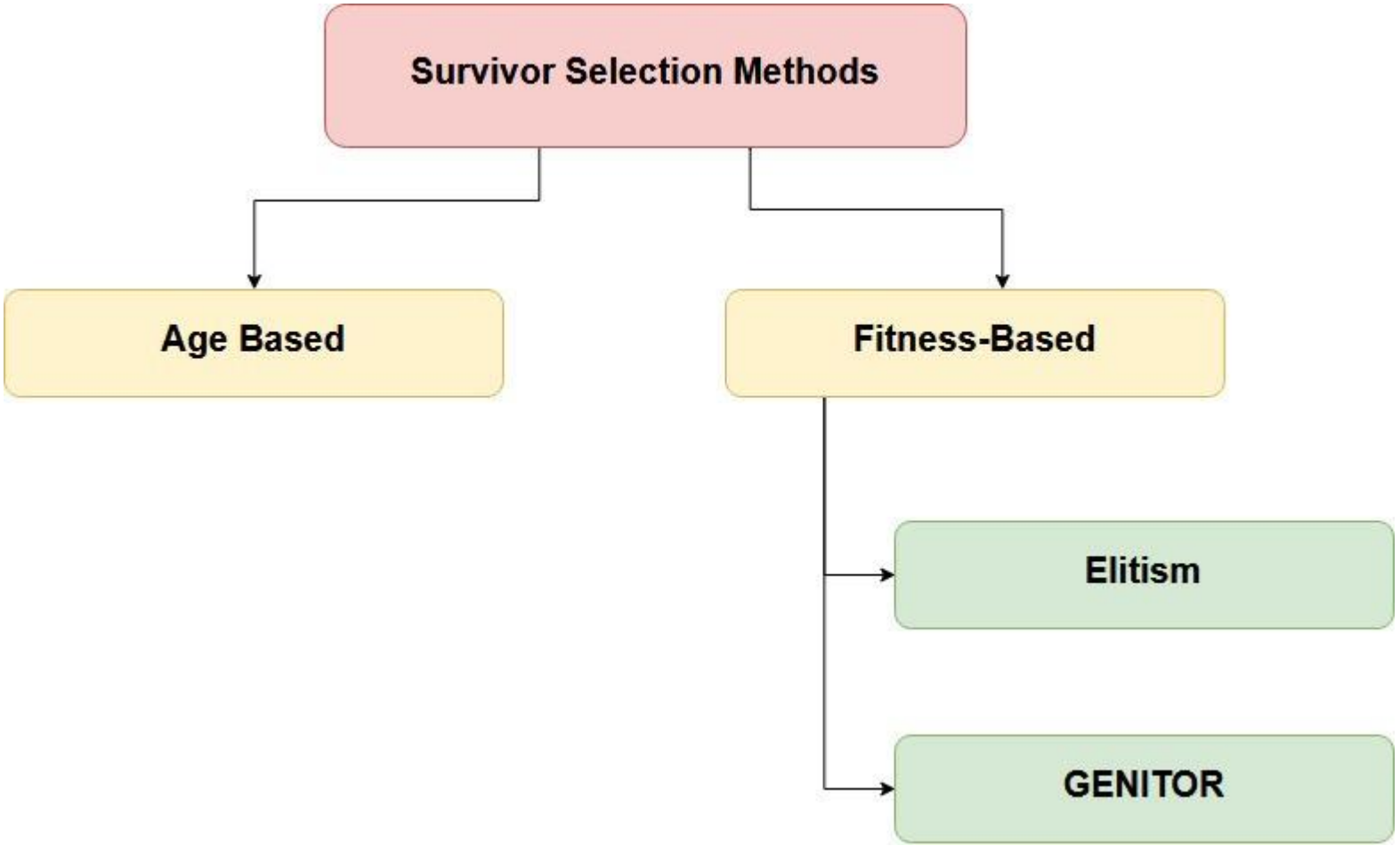
Parent/Survivor Selection

- Strategies
  - Parent selection
    - Uniform randomly selection
    - Probability selection : proportional to their fitness
    - Tournament selection (Multiple Objectives)
      - Build a small comparison set
      - Randomly select a pair with the higher rank one beats the lower one
        - Non-dominated one beat the dominated one
        - **Niche count**: the number of points in the population within certain distance, higher the niche count, lower the rank.
    - Etc.

Selection

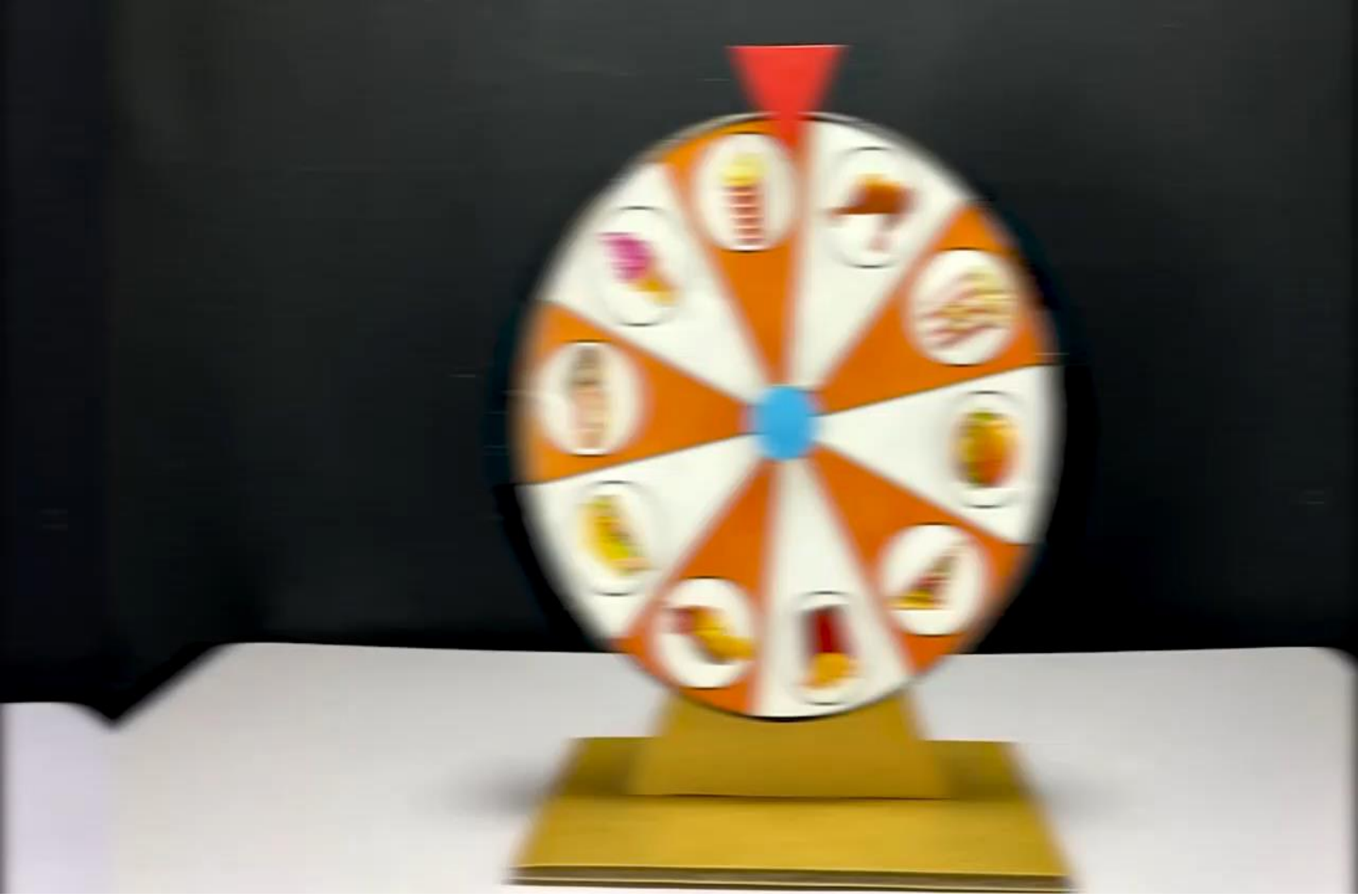

Parent Selection Methods
Fitness Proportionate
Ordinal based
Threshold based
Roulette Wheel Selection
Stochastic Universal Sampling
Ranking Selection
Tournament Selection
Truncation selection

Selection


Survivor Selection Methods
Age Based
Fitness-Based
Elitism
GENITOR

Selection

- Selection.
  - Roulette wheel
  - Rank based
  - Tournament.



Roulette Wheel Selection Method

Grand old method:

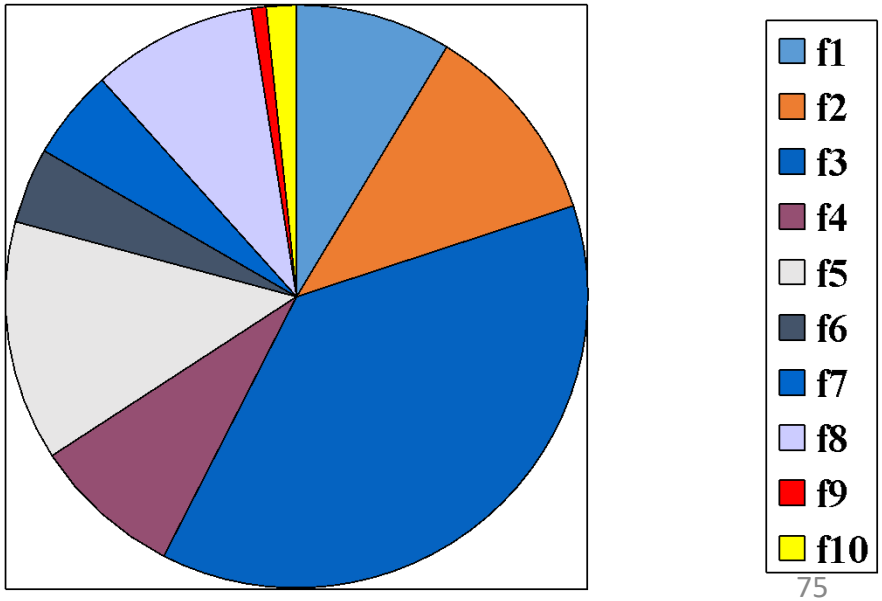
Fitness Proportionate Selection also called *Roulette Wheel selection*

Suppose there are *P* individuals with fitnesses *f*1, *f*2, …, *fP; and* higher values mean better fitness.

The probability of selecting individual *i* is simply:

$$\frac{f_i}{\sum_{k=1}^{P} f_k}$$

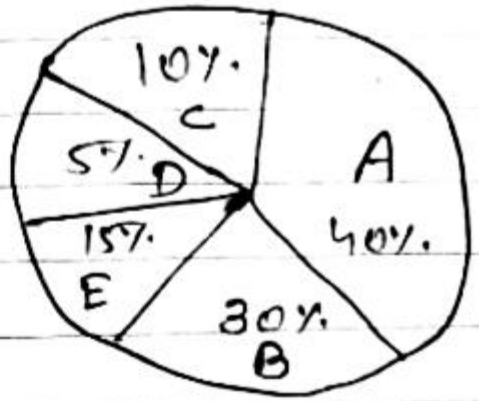
This is equivalent to spinning a roulette wheel with sectors proportional to fitness



Roulette wheel.

A	2.0	40%.
B	1.5	30%.
C	0.5	10%.
D	0.25	5%.
E	0.75	15%.
	$\Sigma f_i = 5$	

$$p_i = \frac{F_i}{\Sigma F_i}$$



Roulette wheel

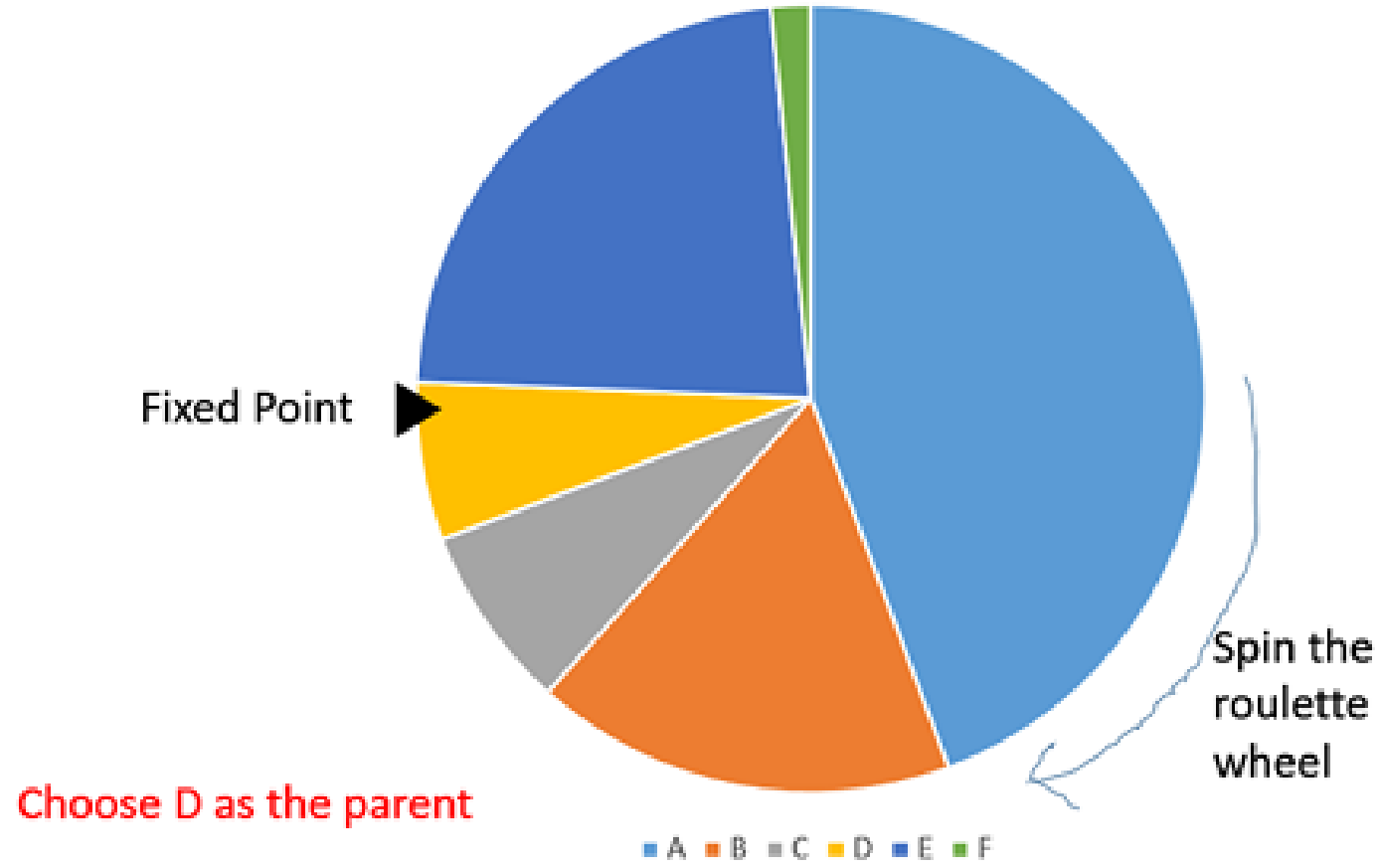
- Sum the fitness of all chromosomes, call it T
- Generate a random number N between 1 and T
- Return chromosome whose fitness added to the running total is equal to or larger than N
- Chance to be selected is exactly proportional to fitness

Chromosome:	1	2	3	4	5	6
Fitness:	8	2	17	7	4	11

Roulette wheel

- Sum the fitness of all chromosomes, call it T
- Generate a random number N between 1 and T
- Return chromosome whose fitness added to the running total is equal to or larger than N
- Chance to be selected is exactly proportional to fitness

Chromosome:	1	2	3	4	5	6
Fitness:	8	2	17	7	4	11
Running total:	8	10	27	34	38	49
N (1 ≤ N ≤ 49):			23			
Selected:			3			



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Demerits of Roulette Wheel Selection

- *Roulette Wheel Selection suffers from problem of premature convergence.*
- *If there exist individuals with prominent values, the diversity of the population is lost. Selective pressure (The degree to which the better individuals are favored) becomes very large.*
- *If dispersion is small(chromosomes' fitness functions are somewhat similar), it becomes almost the same as the random selection . Selective pressure becomes very small.*
- *Have a risk of premature convergence of the genetic algorithm to a local optimum due to the possible presence of a dominant individual that always wins the competition and is selected as a parent.*

Problems with Roulette Wheel Selection

Having probability of selection directly proportional to fitness has a nice ring to it. It is still used a lot, and is convenient for theoretical analyses, but:

What about when we are trying to *minimise* the `fitness' value?

What about when we may have negative fitness values?

We can modify things to sort these problems out easily, but fitprop remains too sensitive to fine detail of the fitness measure. Suppose we are trying to maximise something, and we have a population of 5 fitnesses:

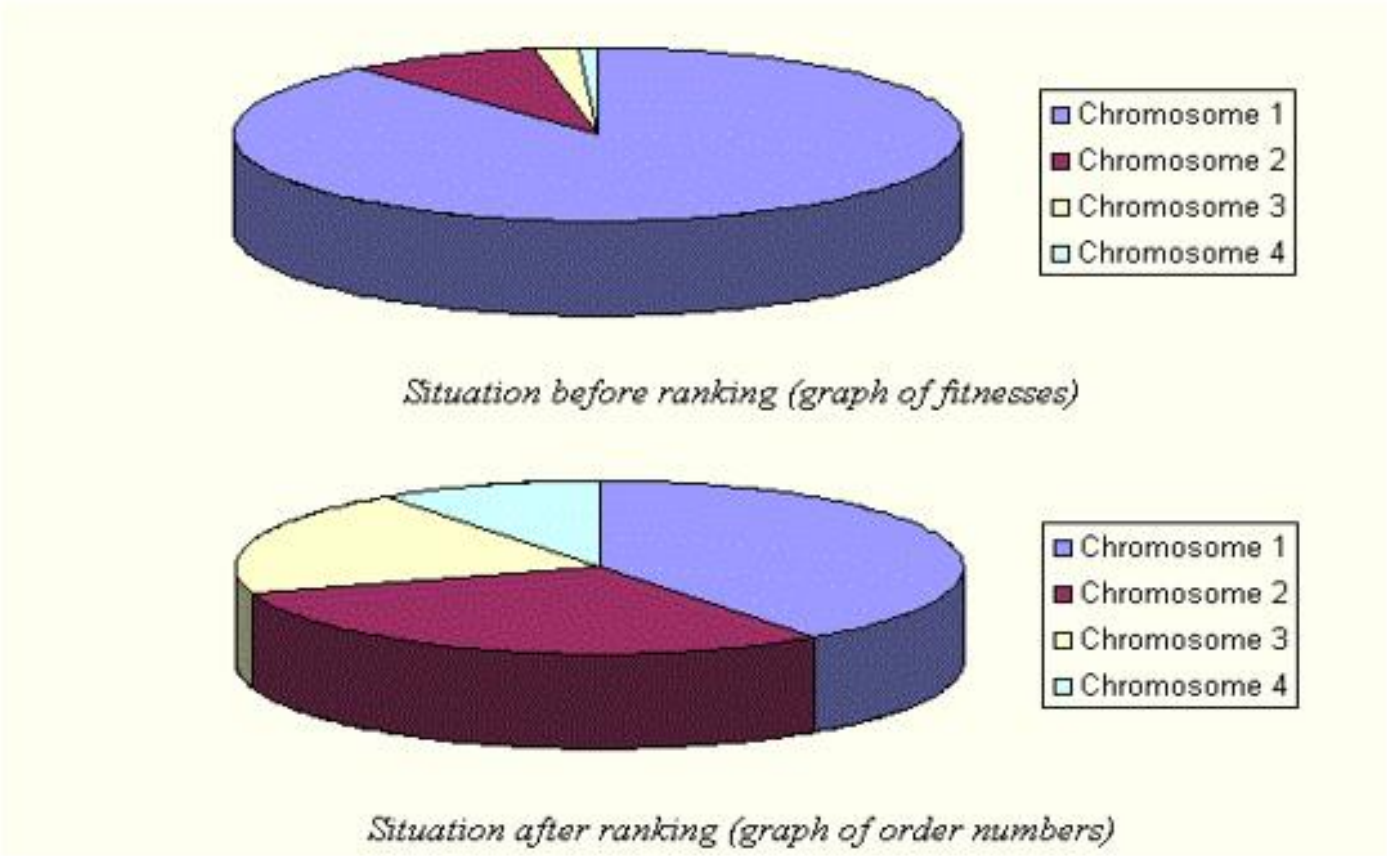
100, 0.4, 0.3, 0.2, 0.1 -- the best is 100 times more likely to be selected than all the rest put together! But a slight modification of the fitness calculation might give us:

200, 100.4, 100.3, 100.2, 100.1 – a much more reasonable situation.

Point is: Fitprop requires us to be very careful how we design the fine detail of fitness assignment.

Other selection methods are better in this respect, and more used now.

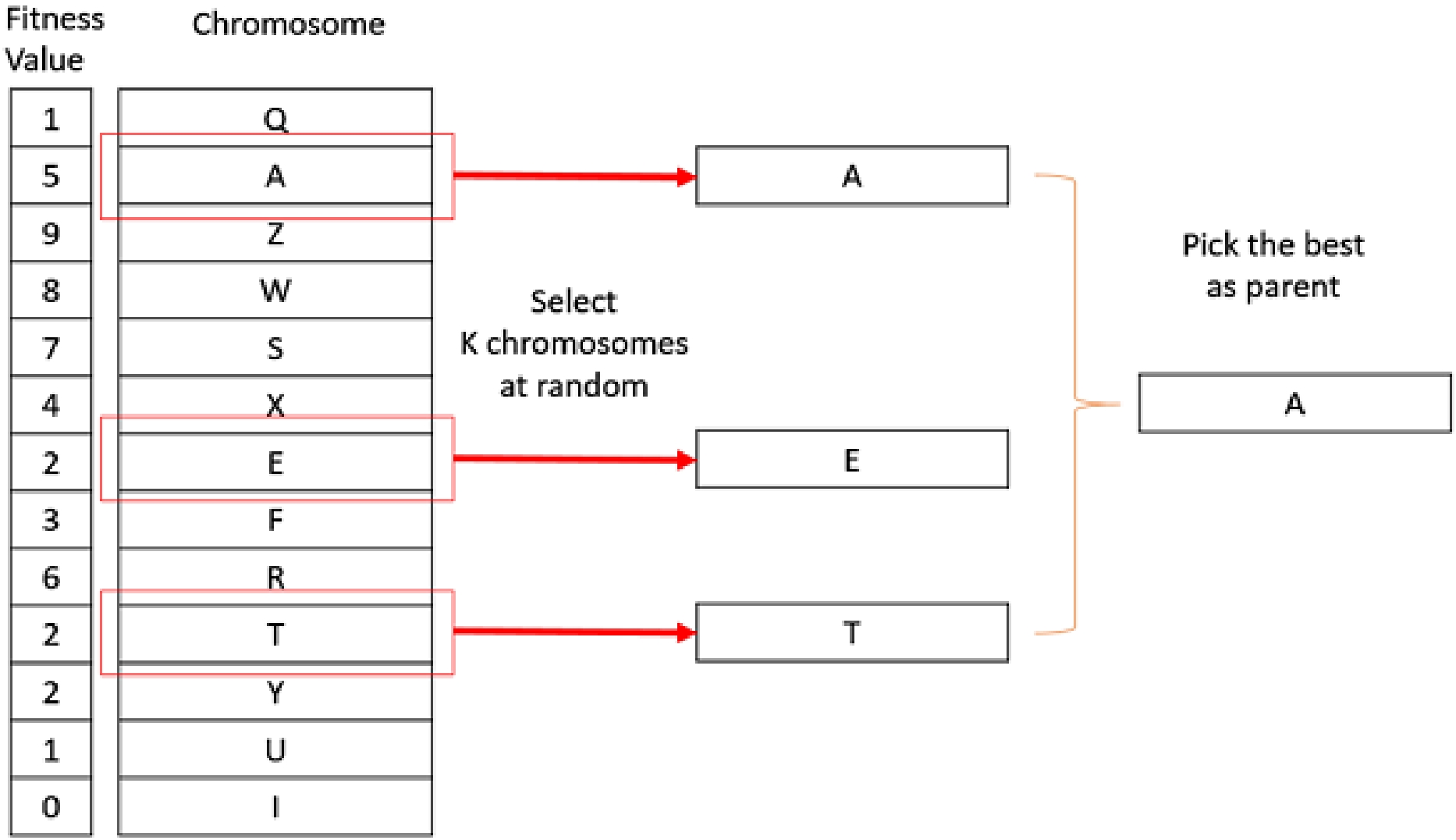
Rank based Selection



Tournament

- Binary tournament
  - Two individuals are randomly chosen; the fitter of the two is selected as a parent
- Probabilistic binary tournament
  - Two individuals are randomly chosen; with a chance p, $0.5<p<1$, the fitter of the two is selected as a parent
- Larger tournaments
  - n individuals are randomly chosen; the fittest one is selected as a parent

- By changing n and/or p, the GA can be adjusted dynamically


Fitness Value
Chromosome
1
Q
5
A
9
Z
8
W
7
S
4
X
2
E
3
F
6
R
2
T
2
Y
1
U
0
I
Select K chromosomes at random
A
E
T
Pick the best as parent
A

Crossover Vs mutation

Exploration: How to discover promising areas in the search space, i.e. gaining information on the problem

Exploitation**:** Optimising within a promising area, i.e. using information

Crossover is explorative: makes a big jump to an area somewhere “in between” two (parent) areas

Mutation is exploitative: creates random small diversions, thus staying near (within the area of) the parent

A balance between Exploration and Exploitation is necessary. Too much exploration results in a pure random search whereas too much exploitation results in a pure local search.

Parameter Control

A GA/EA has many strategy parameters, e.g.

- mutation operator and mutation rate
- crossover operator and crossover rate
- selection mechanism and selective pressure (e.g. tournament size)
- population size

Good parameter values facilitate good performance, but

how do we find good parameter values ?

Termination Criteria

- A genetic algorithm is run over a number of generations until the termination criteria are reached.
- Typical termination criteria are:
  - Stop after a fixed number of generations.
  - Stop when a chromosome reaches a specified fitness level.
  - Stop when a chromosome succeeds in solving the problem, within a specified tolerance.
- Human judgment can also be used in some more subjective cases.

A Simple Example

The traveling salesman problem

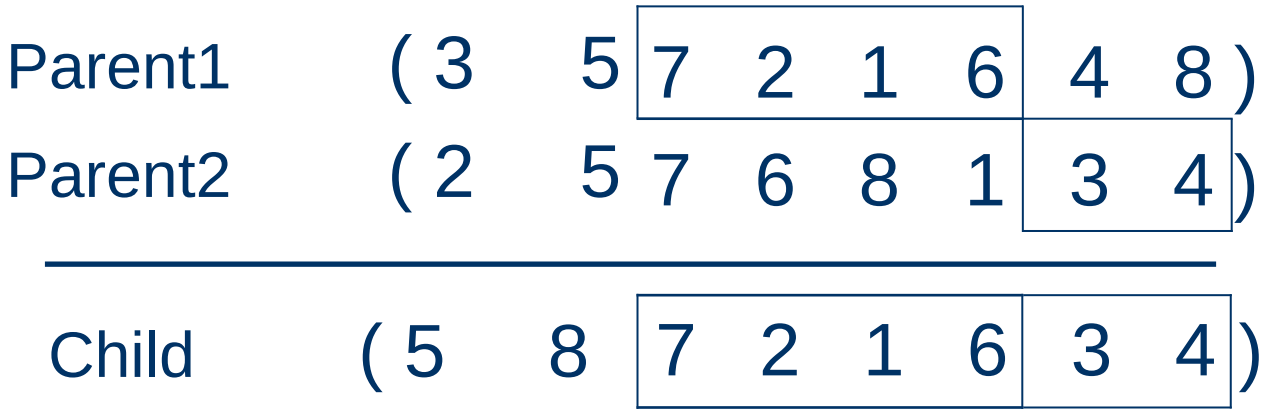
Find a tour of a given set of cities so that:

- Each city is visited only once
- The total distance traveled is minimized

Representation

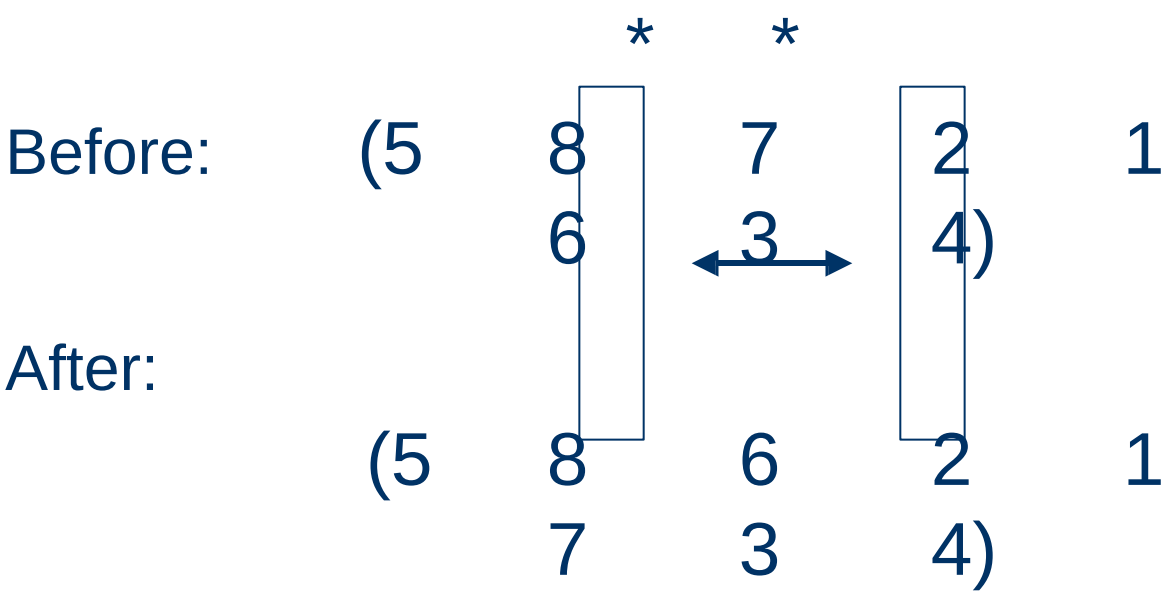
- Representation is an ordered list of city numbers:
  1. Kyiv
  2. Budapest
  3. Teheran
  4. Beijing
  5. Jerusalem
  6. Bucharest
  7. Hamilton
  8. Toronto
- Possible city lists as candidate solutions:
  - (3 CityList1 5 7 2 1 6 4 8)
  - (2 CityList2 5 7 6 8 1 3 4)

Crossover

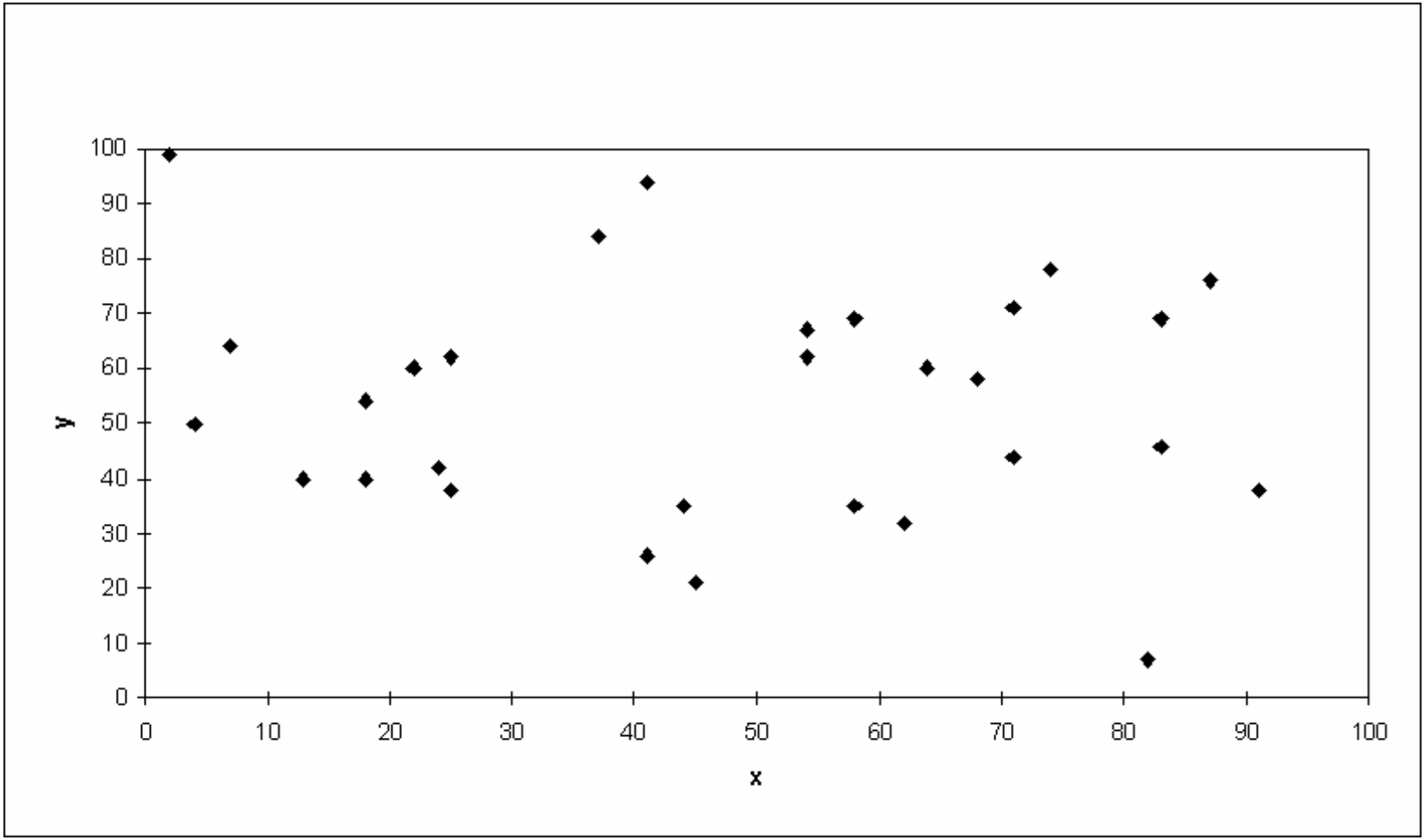


Mutation

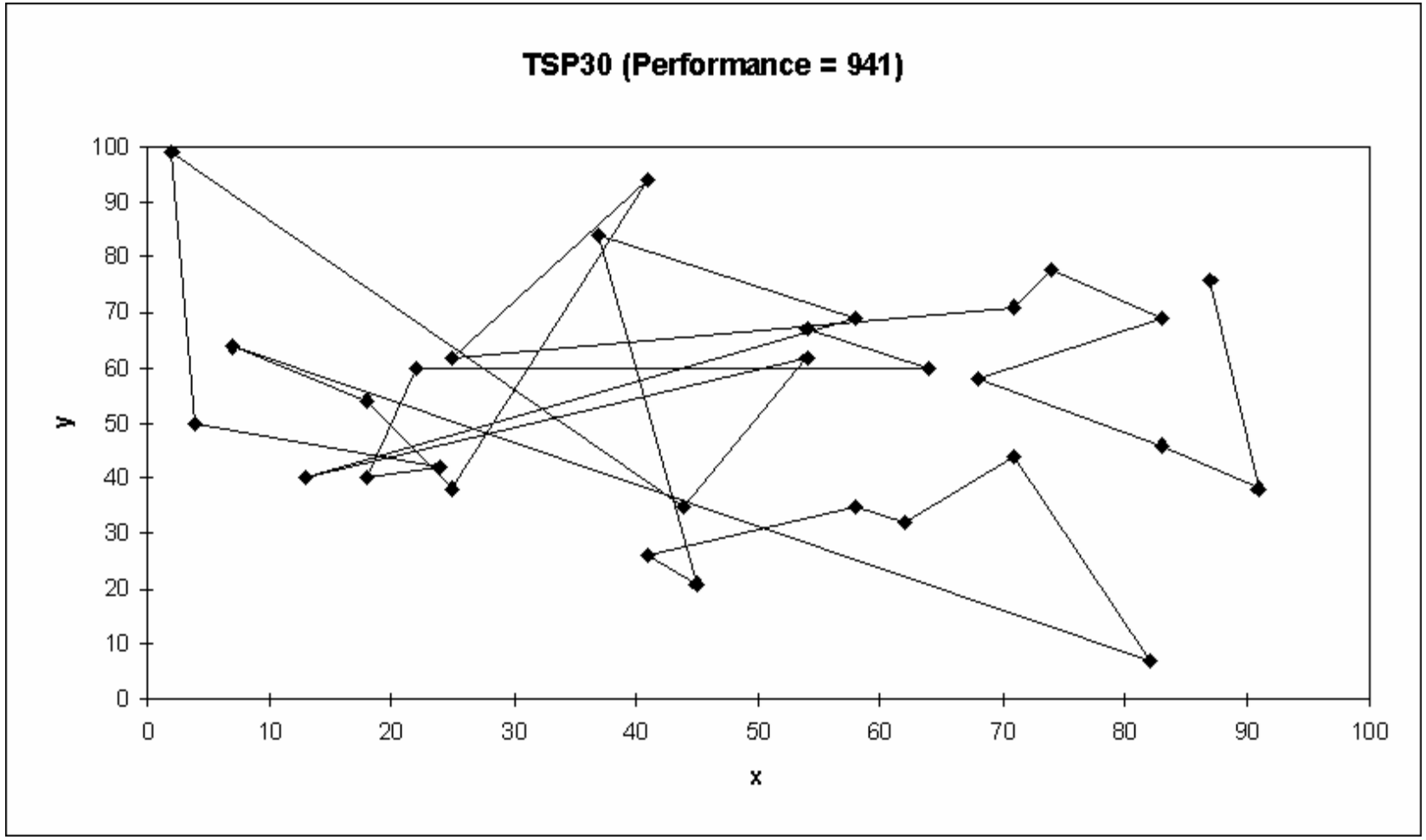
Mutation involves reordering of the list:



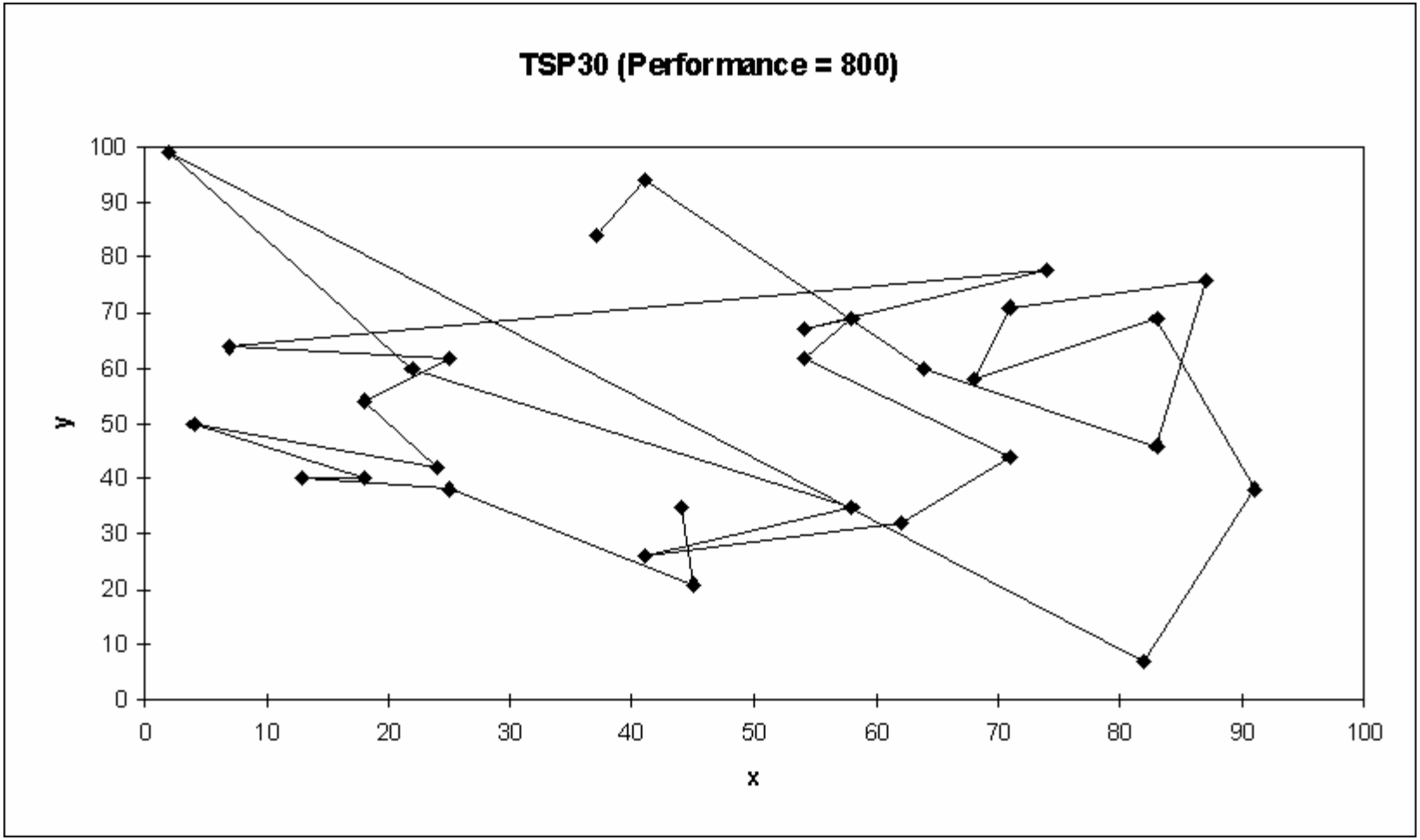
TSP Example: 30 Cities



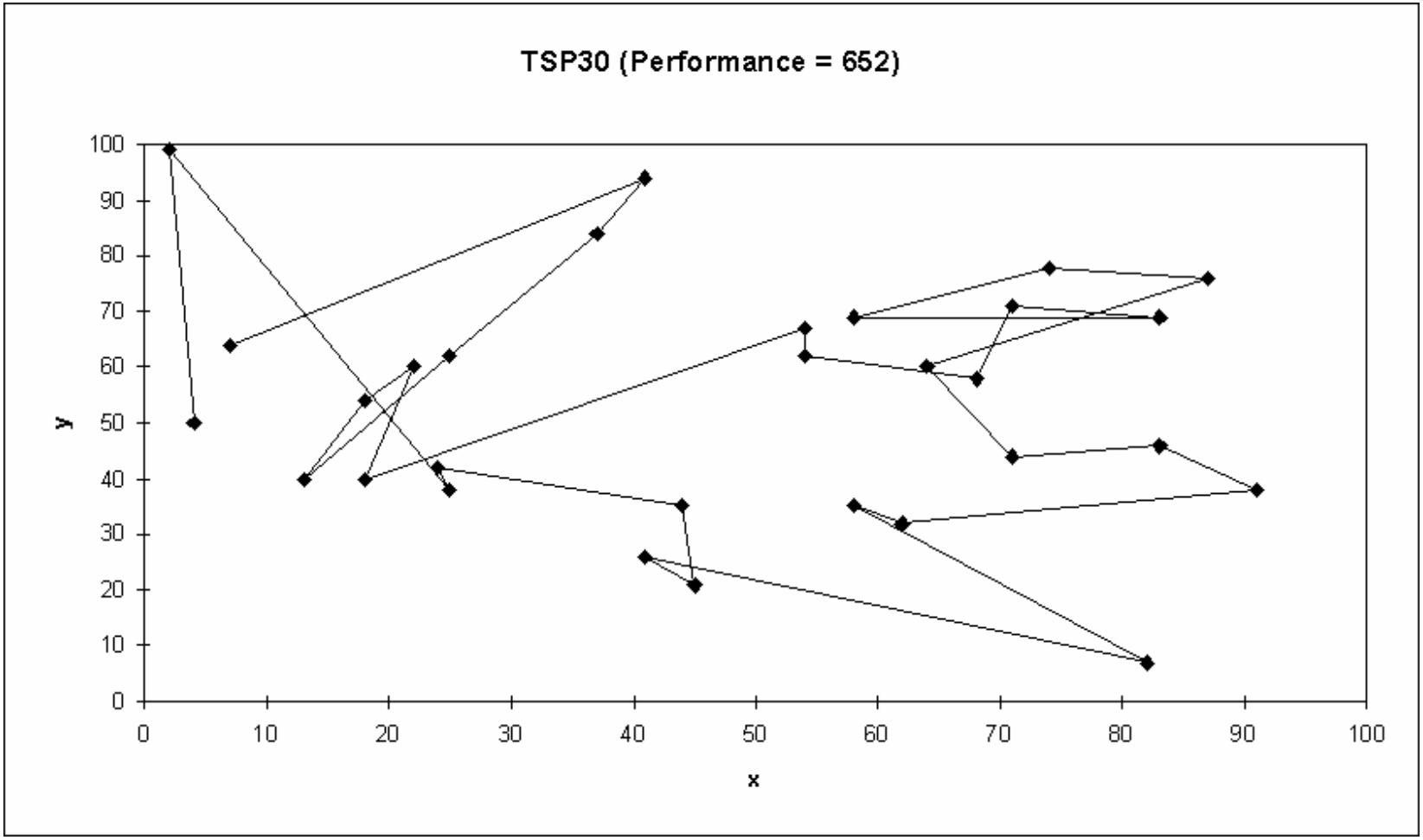
Solution i (Distance = 941)



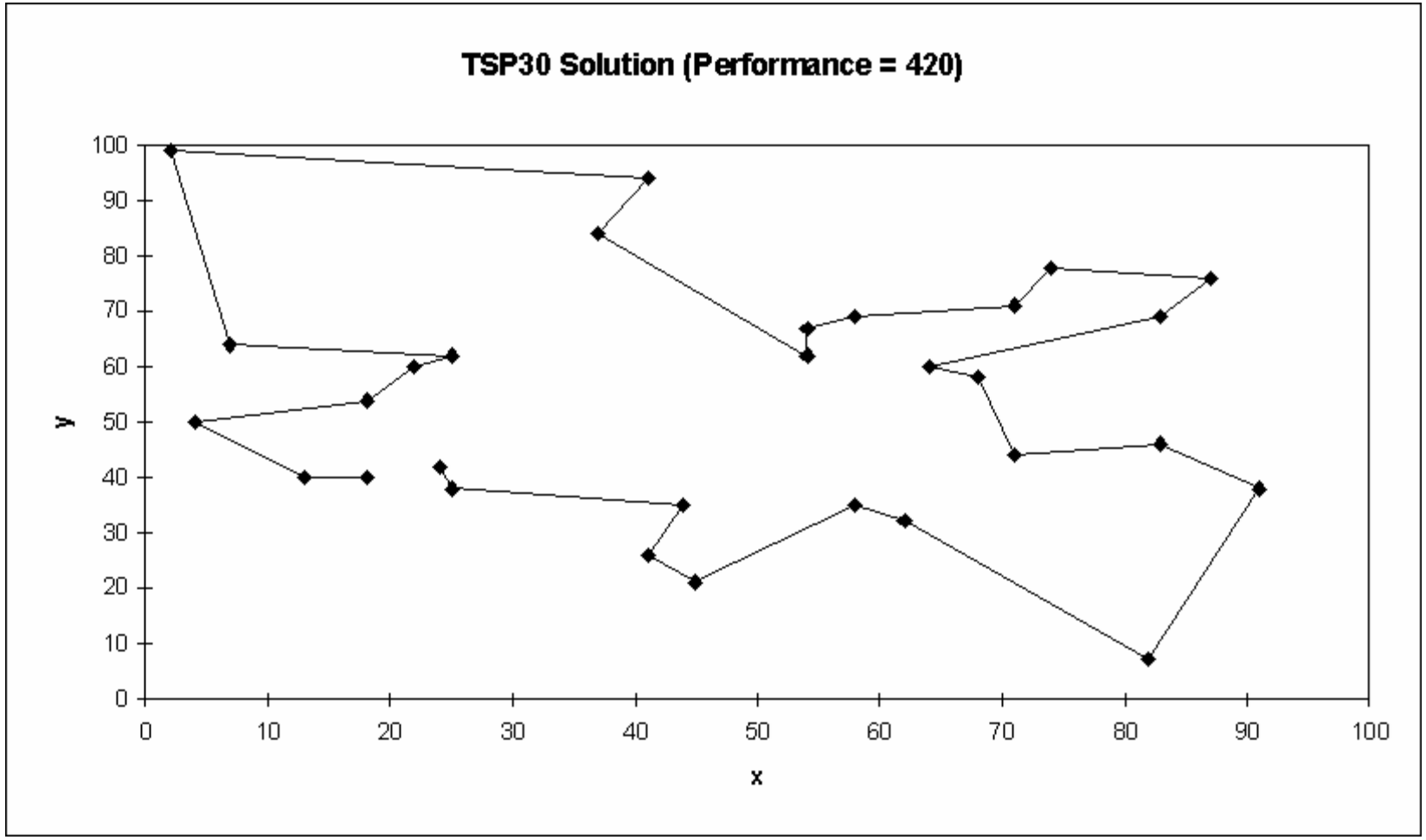
Solution j (Distance = 800)



Solution k (Distance = 652)



Best Solution (Distance = 420)

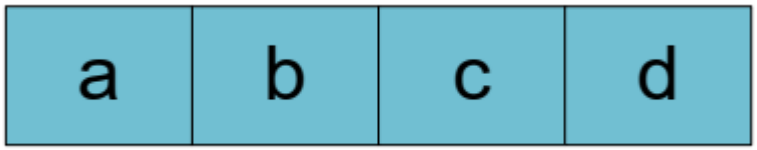


Numerical Example

- Use genetic algorithms to solve the problem of combination.
- Suppose there is equality a + 2b + 3c + 4d = 30, genetic algorithm will be used to find the value of a, b, c, and d that satisfy the above equation.
- First, formulate the objective function, for this problem the objective is minimizing the value of function f(x) where f(x) = ((a + 2b + 3c + 4d) - 30).

Numerical Example

- Since there are four variables in the equation, namely a, b, c, and d, we can compose the chromosome as follow:
- To speed up the computation, we can restrict that the values of variables a, b, c, and d are integers between 0 and 30.



Numerical Example: Step-1

- **Step 1. Initialization**
- For example we define the number of chromosomes in population are 6, then we generate random value of gene a, b, c, d for 6 chromosomes
- Chromosome[1] = [a;b;c;d] = [12;05;23;08]
- Chromosome[2] = [a;b;c;d] = [02;21;18;03]
- Chromosome[3] = [a;b;c;d] = [10;04;13;14]
- Chromosome[4] = [a;b;c;d] = [20;01;10;06]
- Chromosome[5] = [a;b;c;d] = [01;04;13;19]
- Chromosome[6] = [a;b;c;d] = [20;05;17;01]

Step 2. Evaluation

- We compute the objective function value for each chromosome produced in initialization step:
- F_obj[1] = Abs((12 + 2*05 + 3*23 + 4*08) - 30)

  = Abs((12 + 10 + 69 + 32) - 30)

  = Abs(123 - 30)

  = 93
- F_obj[2] = Abs((02 + 2*21 + 3*18 + 4*03) - 30)

  = Abs((02 + 42 + 54 + 12) - 30)

  = Abs(110 - 30)

  = 80

Step 2. Evaluation

- F_obj[3] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)
  = Abs((10 + 08 + 39 + 56) - 30)
  = Abs(113 - 30)
  = 83
- F_obj[4] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)
  = Abs((20 + 02 + 30 + 24) - 30)
  = Abs(76 - 30)
  = 46
- F_obj[5] = Abs((01 + 2*04 + 3*13 + 4*19) - 30)
  = Abs((01 + 08 + 39 + 76) - 30)
  = Abs(124 - 30)
  = 94

Step 2. Evaluation

- F_obj[6] = Abs((20 + 2*05 + 3*17 + 4*01) - 30)
  = Abs((20 + 10 + 51 + 04) - 30)
  = Abs(85 - 30)
  = 55

Step 3. Selection

1. The fittest chromosomes have higher probability to be selected for the next generation. To compute fitness probability we must compute the fitness of each chromosome.

- To avoid divide by zero problem, the value of F_obj is added by 1.
- **Fitness**[1] = 1 / (1+**F_obj**[1])

  = 1 / 94

  = 0.0106

Step 3. Selection

- **Fitness**[2] = 1 / (1+**F_obj**[2])
  = 1 / 81
  = 0.0123
- **Fitness**[3] = 1 / (1+**F_obj**[3])
  = 1 / 84
  = 0.0119
- **Fitness**[4] = 1 / (1+**F_obj**[4])
  = 1 / 47
  = 0.0213
- **Fitness**[5] = 1 / (1+**F_obj**[5])
  = 1 / 95
  = 0.0105
- **Fitness**[6] = 1 / (1+**F_obj**[6])
  = 1 / 56
  = 0.0179
- **Total** = 0.0106 + 0.0123 + 0.0119 + 0.0213 + 0.0105 + 0.0179
  = 0.0845

Step 3. Selection

- The probability for each chromosomes is formulated by: **P[i] = Fitness[i] / Total**
- P[1] = 0.0106 / 0.0845
  = 0.1254
- P[2] = 0.0123 / 0.0845
  = 0.1456
- P[3] = 0.0119 / 0.0845
  = 0.1408
- P[4] = 0.0213 / 0.0845
  = 0.2521
- P[5] = 0.0105 / 0.0845
  = 0.1243
- P[6] = 0.0179 / 0.0845
  = 0.2118

Step 3. Selection

- From the probabilities above we can see that Chromosome 4 that has the highest fitness, this chromosome has highest probability to be selected for next generation chromosomes.
- For the selection process we use roulette wheel, for that we should compute the cumulative probability values:

Step 3. Selection

- C[1] = 0.1254
- C[2] = 0.1254 + 0.1456
  = 0.2710
- C[3] = 0.1254 + 0.1456 + 0.1408
  = 0.4118
- C[4] = 0.1254 + 0.1456 + 0.1408 + 0.2521
  = 0.6639
- C[5] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243
  = 0.7882
- C[6] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 + 0.2118
  = 1.0

Step 3. Selection

- Having calculated the cumulative probability of selection process using roulette-wheel can be done.
- The process is to generate random number R in the range 0-1 as follows.
- R[1] = 0.201
- R[2] = 0.284
- R[3] = 0.099
- R[4] = 0.822
- R[5] = 0.398
- R[6] = 0.501

Step 3. Selection

- If random number R[1] is greater than C[1] and smaller than C[2] then select Chromosome[2] as a chromosome in the new population for next generation:
- NewChromosome[1] = Chromosome[2]
- NewChromosome[2] = Chromosome[3]
- NewChromosome[3] = Chromosome[1]
- NewChromosome[4] = Chromosome[6]
- NewChromosome[5] = Chromosome[3]
- NewChromosome[6] = Chromosome[4]

Step 3. Selection

- Chromosomes in the population thus became:
- Chromosome[1] = [02;21;18;03]
- Chromosome[2] = [10;04;13;14]
- Chromosome[3] = [12;05;23;08]
- Chromosome[4] = [20;05;17;01]
- Chromosome[5] = [10;04;13;14]
- Chromosome[6] = [20;01;10;06]

Step 4. Crossover

- In this example, we use single poin, i.e. randomly select a position in the parent chromosome then exchanging sub-chromosome. Parent chromosome which will mate is randomly selected and the number of mate Chromosomes is controlled using **crossover_rate (Pc)** parameters. Pseudo-code for the crossover pr

```
begin
    k← 0;
    while(k<population) do
    R[k] = random(0-1);
    if(R[k]< ρc) then
      select Chromosome[k] as parent;
    end;
    k = k + 1;
    end;
end;
```

Step 4. Crossover

- Chromosome k will be selected as a parent if **R[k]<Pc**. Suppose we set that the crossover rate is 25%, then Chromosome number k will be selected for crossover if random generated value for Chromosome k below 0.25.
- The process is as follows: First we generate a random number R as the number of population.
- R[1] = 0.191
- R[2] = 0.259
- R[3] = 0.760
- R[4] = 0.006
- R[5] = 0.159
- R[6] = 0.340

Step 4. Crossover

- For random number R above, parents are Chromosome[1], Chromosome[4] and Chromosome[5] will be selected for crossover.
- Chromosome[1] >< Chromosome[4]
- Chromosome[4] >< Chromosome[5]
- Chromosome[5] >< Chromosome[1]

Step 4. Crossover

- After chromosome selection, the next process is determining the position of the crossover point.
- This is done by generating random numbers between 1 to (length of Chromosome – 1).
- In this case, generated random numbers should be between 1 and 3.
- After we get the crossover point, parents Chromosome will be cut at crossover point and its gens will be interchanged.
- For example we generated 3 random number and we get:
- C[1] = 1
- C[2] = 1
- C[3] = 2

Step 4. Crossover

- Then for first crossover, second crossover and third crossover, parent's gens will be cut at gen number 1, gen number 1 and gen number 3 respectively, e.g.
- Chromosome[1] = Chromosome[1] >< Chromosome[4]
  = [02;21;18;03] >< [20;05;17;01]
  = [02;05;17;01]
- Chromosome[4] = Chromosome[4] >< Chromosome[5]
  = [20;05;17;01] >< [10;04;13;14]
  = [20;04;13;14]
- Chromosome[5] = Chromosome[5] >< Chromosome[1]
  = [10;04;13;14] >< [02;21;18;03]
  = [10;04;18;03]

Step 4. Crossover

- Thus Chromosome population after experiencing a crossover process:
- Chromosome[1] = [02;05;17;01]
- Chromosome[2] = [10;04;13;14]
- Chromosome[3] = [12;05;23;08]
- Chromosome[4] = [20;04;13;14]
- Chromosome[5] = [10;04;18;03]
- Chromosome[6] = [20;01;10;06]

Step 5. Mutation

- Number of chromosomes that have mutations in a population is determined by the **mutation_rate** parameter.
- Mutation process is done by replacing the gen at random position with a new value.
- The process is as follows. First we must calculate the total length of gen in the population. In this case the total length of gen is
- **total_gen = number_of_gen_in_Chromosome * number of population**

  = 4 * 6

  = 24

Step 5. Mutation

- Mutation process is done by generating a random integer between 1 and total_gen (1 to 24).
- If generated random number is smaller than mutation_rate(ρm) variable then marked the position of gen in chromosomes.
- Suppose we define **Pm** 10%, it is expected that 10% (0.1) of total_gen in the population that will be mutated:
- number of mutations = 0.1 * 24

  = 2.4

  ≈ 2

Step 5. Mutation

- Suppose generation of random number yield 12 and 18 then the chromosome which have mutation are Chromosome number 3 gen number 4 and Chromosome 5 gen number 2.
- The value of mutated gens at mutation point is replaced by random number between 0-30.
- Suppose generated random number are 2 and 5 then Chromosome composition after mutation are:
- Chromosome[1] = [02;05;17;01]
- Chromosome[2] = [10;04;13;14]
- Chromosome[3] = [12;05;23;**02**]
- Chromosome[4] = [20;04;13;14]
- Chromosome[5] = [10;**05**;18;03]
- Chromosome[6] = [20;01;10;06]

Final

- Finishing mutation process then we have one iteration or one generation of the genetic algorithm. We can now evaluate the objective function after one generation:
- Chromosome[1] = [02;05;17;01]
- F_obj[1] = Abs((02 + 2*05 + 3*17 + 4*01) - 30)
  = Abs((2 + 10 + 51 + 4) - 30)
  = Abs(67 - 30)
  = 37
- Chromosome[2] = [10;04;13;14]
- F_obj[2] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)
  = Abs((10 + 8 + 33 + 56) - 30)
  = Abs(107 - 30)
  = 77
- Chromosome[3] = [12;05;23;02]
- F_obj[3] = Abs((12 + 2*05 + 3*23 + 4*02) - 30)
  = Abs((12 + 10 + 69 + 8) - 30)
  = Abs(87 - 30)
  = 47

Final

- Chromosome[4] = [20;04;13;14]
- F_obj[4] = Abs((20 + 2*04 + 3*13 + 4*14) - 30)
  = Abs((20 + 8 + 39 + 56) - 30)
  = Abs(123 - 30)
  = 93
- Chromosome[5] = [10;05;18;03]
- F_obj[5] = Abs((10 + 2*05 + 3*18 + 4*03) - 30)
  = Abs((10 + 10 + 54 + 12) - 30)
  = Abs(86 - 30)
  = 56
- Chromosome[6] = [20;01;10;06]
- F_obj[6] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)
  = Abs((20 + 2 + 30 + 24) - 30)
  = Abs(76 - 30)
  = 46

Final

- From the evaluation of new Chromosome we can see that the objective function is decreasing, this means that we have better Chromosome or solution compared with previous
- Chromosome generation. New Chromosomes for next iteration are:
- Chromosome[1] = [02;05;17;01]
- Chromosome[2] = [10;04;13;14]
- Chromosome[3] = [12;05;23;02]
- Chromosome[4] = [20;04;13;14]
- Chromosome[5] = [10;05;18;03]
- Chromosome[6] = [20;01;10;06]

Final

- These new Chromosomes will undergo the same process as the previous generation of Chromosomes such as evaluation, selection, crossover and mutation and at the end it produce new generation of Chromosome for the next iteration.
- This process will be repeated until a predetermined number of generations. For this example, after running 50 generations, best chromosome is obtained:
- Chromosome = [07; 05; 03; 01]
- This means that: a = 7, b = 5, c = 3, d = 1
- If we use the number in the problem equation:
- a + 2b + 3c + 4d = 30
- 7 + (2 * 5) + (3 * 3) + (4 * 1) = 30
- We can see that the value of variable a, b, c and d generated by genetic algorithm can satisfy that equality

Question

- Consider the problem of finding the shortest route through several cities, such that each city is visited only once and in the end return to the starting city (the Travelling Salesman problem). Suppose that in order to solve this problem we use a genetic algorithm, in which genes represent links between pairs of cities. For example, a link between London and Paris is represented by a single gene 'LP'. Let also assume that the direction in which we travel is not important, so that LP = PL.
- a) How many genes will be used in a chromosome of each individual if the number of cities is 10?
- b) How many genes will be in the alphabet of the algorithm? (Here, alphabet represents all possible genes)

Answer

- a) Each chromosome will consist of 10 genes. Each gene representing the path between a pair of cities in the tour.
- b) The alphabet will consist of 45 genes. Indeed, each of the 10 cities can be connected with 9 remaining. Thus, $10 \times 9 = 90$ is the number of ways in which 10 cities can be grouped in pairs. However, because the direction is not important (i.e. London–Paris is the same as Paris–London) the number must be divided by 2. So, we shall need $90/2 = 45$ genes in order to encode all pairs. In general, the formula for n cities is: $\frac{n(n-1)}{2}$.

- A Genetic Algorithm uses chromosomes of the form x=abcdefgh with a fixed length of eight genes. Here a, b, c, d, e, f, g, h are genes of chromosomes. Each gene can be any digit between 0 and 9. Let the fitness of individual x be calculated as: f(x) = (a+b)+(c+d)-(e+f)-(g+h)
- and initial population consists of four individuals with the following chromosomes:
    - x1 = 57126541
    - x2 = 23926601
    - x3 = 35321215
    - x4 = 71052904
- **Give the answer of following questions:**
- (i) Write down the Sorting order of Chromosomes according to fitness value in descending order (from high fittest value to least fit value).
- (ii) Perform uniform crossover at positions a, d and f of parents on the following pair:
    - x1 & x4 will generate offspring O1 & O2
    - x2 & x3 will generate offspring O3 & O4
- (iii) Perform inversion mutation on generated offspring from above question at starting position d and ending position h.

- i)
- f(x1) = (5+7)+(1+2)-(6+5)-(4+1)=12+3-11-5=15-16=-1
- f(x2) = (2+3)+(9+2)-(6+6)-(0+1)=5+11-12-1=16-13=3
- f(x3) = (3+5)+(3+2)-(1+2)-(1+5)=8+5-3-6=13-9=4
- f(x4) = (7+1)+(0+5)-(2+9)-(0+4)=8+5-11-4=13-15=-2
- descending order: x3, x2, x1, x4
- ii)
- O1 = 77156941
- O2 = 51022504
- O3 = 33926201
- O4 = 25321615
- iii)
- O1 = 77114965
- O2 = 51040522
- O3 = 33910262
- O4 = 25351612

Application of Genetic Algorithm

- Genetic Algorithms can be applied to virtually any problem that has a large search space.
- The military uses GAs to evolve equations to differentiate between different radar returns.
- Stock companies use GA-powered programs to predict the stock market.

Application of Genetic Algorithm

- Feature Selection
- Engineering Design
  - Engineering design has relied heavily on computer modeling and simulation to make design cycle process fast and economical. Genetic algorithm has been used to optimize and provide a robust solution.
- Traffic and Shipment Routing (Travelling Salesman Problem)
  - This is a famous problem and has been efficiently adopted by many sales-based companies as it is time saving and economical. This is also achieved using genetic algorithm.
- Robotics
  - Genetic algorithm is being used to create learning robots which will behave as a human and will do tasks like cooking our meal, do our laundry etc.

Drawbacks of GA

- Difficult to find an encoding for the problem
- Difficult to define a valid fitness function
- May not return the global maximum
- GA is nondeterministic – two runs may end with different results
- There's no indication whether best individual is optimal

Genetic Algorithms References

1. W. Williams, Genetic Algorithms: A Tutorial, http://web.umr.edu/~ercal/387/slides/GATutorial.ppt
2. A. Eiben, J. Smith, Introduction to Evolutionary Computing, Genetic Algorithms, http://www.cs.vu.nl/~jabekker/ec0607/slides/Lectur e03-Chapter3-GeneticAlgorithms.ppt
3. R. Horst and P.M. Pardalos (eds.), Handbook of Global Optimization, Kluwer, Dordrecht 1995.
4. M. Mitchell, An Introduction To Genetic Algorithms, Cambridge, MA: MIT Press, 1996.

Genetic Algorithm Software

- **GENOCOP III** – Genetic Algorithm for Constrained Problems in C (by Zbigniew Michalewicz)
- **DE** – Differential Evolution Genetic Algorithm in C and Matlab (by Rainer Storn). DE has won the third place at the 1st International Contest on Evolutionary Computation on a real-valued function testsuite
- **PGAPack** – Parallel Genetic Algorithm in Fortran and C (from Argonne National Laboratory)
- **PIKAIA** – Genetic algorithm in Fortran 77/90 (by Charbonneau, Knapp and Miller)
- **GAGA** – Genetic Algorithm for General Application in C (by Ian Poole)
- **GAS** – Genetic Algorithm in C++ (by Jelasity and Dombi)
- **GAlib** – C++ Genetic Algorithm Library (by Matthew Wall)
- **Genetic Algorithm in Matlab** (by Michael B. Gordy)
- **GADS** – Genetic Algorithm and Direct Search Toolbox in Matlab (from MathWorks)
- **GEATbx** – Genetic and Evolutionary Algorithm Toolbox for Matlab (by Hartmut Pohlheim)
- **GAOT** – Genetic Algorithms Optimization Toolbox in Matlab (by Jeffrey Joines)

Software

- GATSS, by Thomas Pederson, a Genetic Algorithm based solver of the Traveling Salesman problem written in GNU C++ linked to CGI-script.
- GAlib, A C++ Genetic Algorithms Library, see in particular the provided examples, the TSP.
- Maugis TSP solver, a program written in ansi-C, for Symmetric Euclidean TSPs, by Lionnel Maugis.
- tsp_solve, a collection of heuristics and optimal algorithms for the TSP, by Chad Hurwitz. He does not have a web page or ftp site to to email him for a copy of the software, it's GNU C so it'll run on most unixes.
- An ATSP code, by Glenn Dicus, Bart Jaworski and Joseph Ou-Yang.
- A TSP program in Parlog, by Steve Gregory.
- The Travelling Spider Problem, by Moshe Sniedovich. We expect this page to include TSP codes based on Dynamic Programming.
- Traveling Salesman Problem solving program (TSPSolver), by Victor V. Miagkikh. TSPSolver is written in Borland C++ and Borlan Some benchmarks are enclosed.
- GRIN, a software package for graphs by Vitali Petchenkine.
- Traveling Salesman Java Applet, by Martin Hagerup. The author comments that the applet has been selected to appear in the book " Dummies" (please, send me one) . Requires Netscape 2.0 in 32 bits or other Java-browser.
- Simple Closed Paths, from the book: *Introduction to Programming with Mathematica, 2nd Edition* , 1995, *TELOS*/Springer-V *Mathematica* notebook.
- tsp1.gms, a page related with GAMS, the General Algebraic Modeling System which is a high-level modeling system for mathematic
- A Java TSP demo program using Kohonen's neural network formulation.
- A Simple TSP-Solver: An ABACUS Tutorial, by Stefan Thienel, 1996.
- A Guided Local Search demo for TSP, by Christos Voudouris, (it requires Microsoft Windows 3.1.).
- Another Guided Local search demo, this one requires SunOS (precompiled for SunOS 5.3) and XView, by Christos Voudouris. A 95/NT is now available.
- A heuristic and a brute force method, Java programs by Aaron Passey.
- BOB: Branch-and-bound Optimization liBrary, a general-purpose parallel Branch-and-Bound algorithm library being developed at P University of Versailles-Saint Quentin en Yvelines. They provide examples for QAP, TSP and VCP. It is freely available via anonymou the file pub/software/BoBL1.0.tar.Z. Click here to get a copy of it.
- Concorde, a powerful code by by David Applegate, Robert Bixby, William Cook, and V. Chvatal. (for download, please use anony ftp.caam.rice.edu, change to directory /pub/people/bico/970827/ and download file cc970827.tgz). A must-see !
- David Neto's Lin-Kernighan ``cluster aware'' heuristic, by David Neto, a literate program written using the CWEB toolset. Another

Genetic programming

- A string of bits could represent a *program*
- If you want a program to do something, you might try to *evolve* one
- As a concrete example, suppose you want a program to help you choose stocks in the stock market
  - There is a huge amount of data, going back many years
  - What data has the most predictive value?
  - What's the best way to combine this data?
- A genetic program is possible in theory, but it might take millions of years to evolve into something useful
  - How can we improve this?