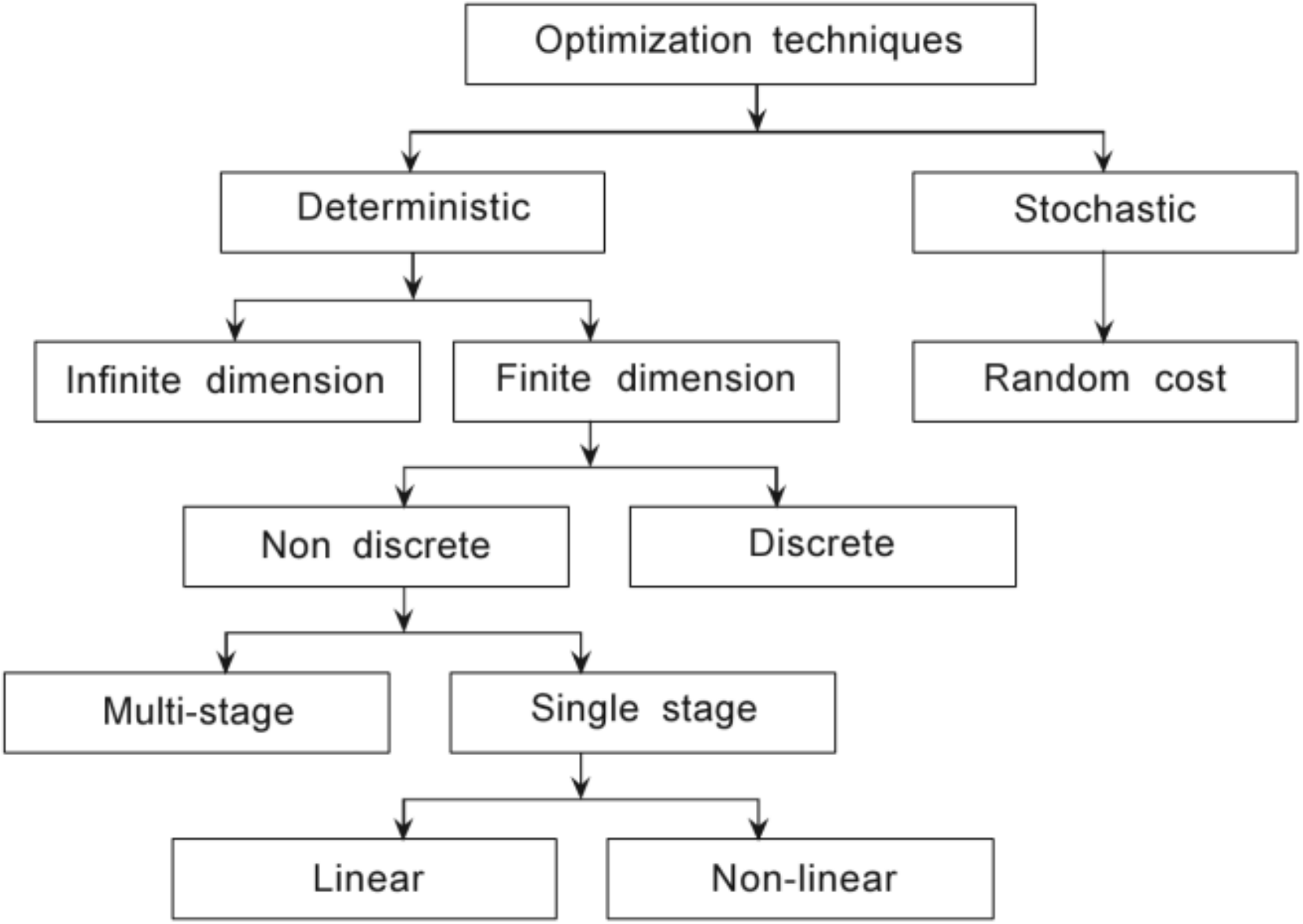
Introduction of Genetic Algorithm

- Computerized search and *optimization algorithms* based on *Darwin's Principle of Natural Selection*
- Part of ***Evolutionary Algorithms***
- *Basic concept* - to simulate processes in natural system necessary for evolution
- Structural Engineering Problems, Biology, Computer Science, Image processing and Pattern Recognition, physical science, social sciences and Neural Networks
- Provide efficient, effective techniques for optimization


Optimization techniques
Deterministic
Stochastic
Infinite dimension
Finite dimension
Random cost
Non discrete
Discrete
Multi-stage
Single stage
Linear
Non-linear

In the case of *deterministic search,* algorithm methods such as steepest gradient methods are employed (using gradient concept), whereas in *stochastic approach*, random variables are introduced. Whether the search is deterministic or stochastic, it is possible to improve the reliability of the results where reliability means getting the result near optimum. A transition rule must be used to improve the reliability. Algorithms vary according to the transition rule used to improve the result.

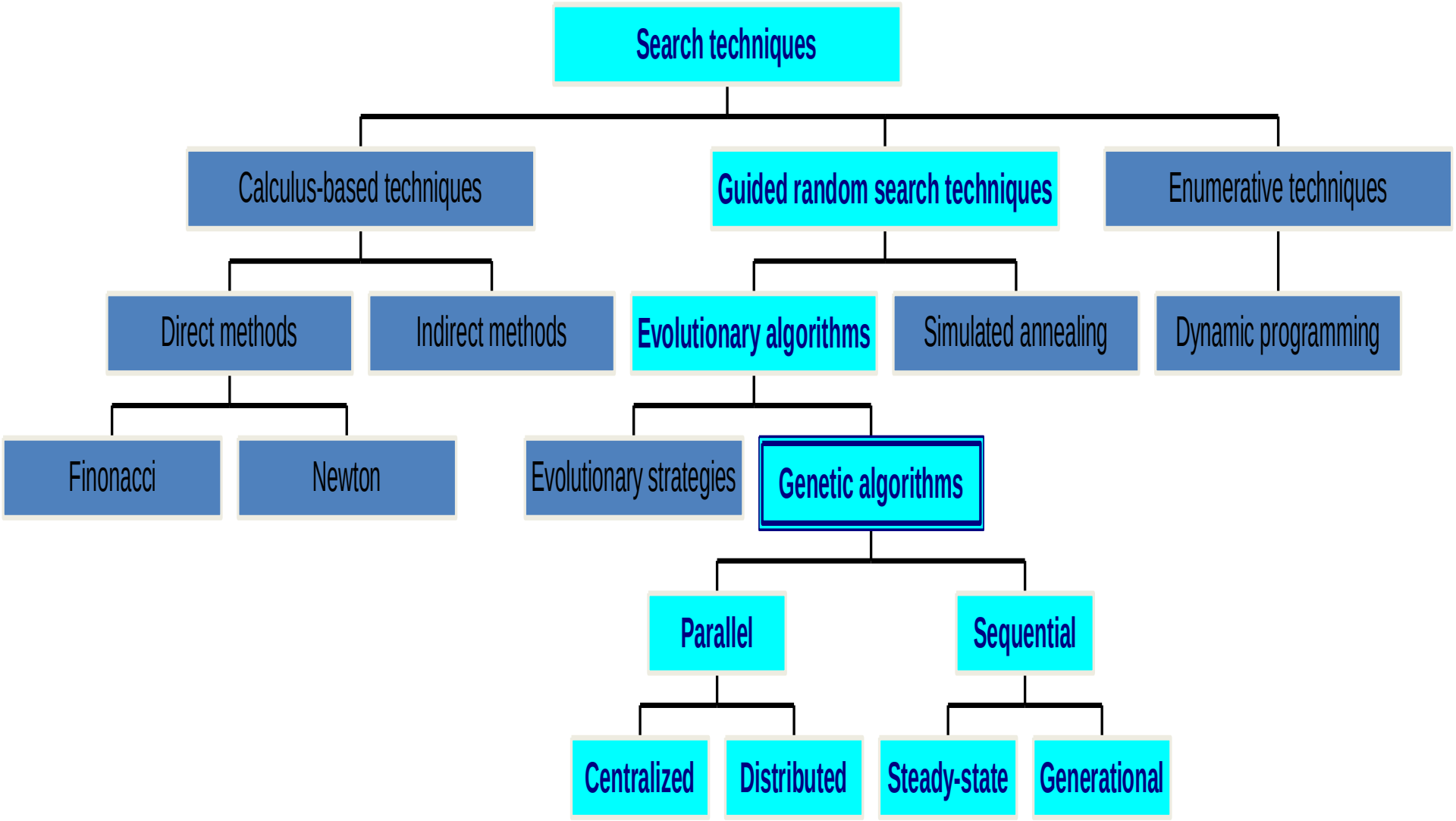
Nontraditional search and optimization methods have become popular in engineering optimization problems in recent past. These algorithms include:

1. Simulated annealing (Kirkpatrik, et al. 1983)
2. Ant colony optimization (Dorigo and Caro, 1999)
3. Random cost (Kost and Baumann, 1999)
4. Evolution strategy (Kost, 1995)
5. Genetic algorithms (Holland, 1975)
6. Cellular automata (Wolfram, 1994)

History

- Evolutionary Computing – 1960 by *Rechenberg*
- Developed by ***John Holland***, University of Michigan - 1970
- Got popular in the late 1980's
- Based on ideas from ***Darwinian*** *Evolution theory* ***"Survival of the fittest"***
- 1986 – Optimization of a Ten Member Plane

Search Techniques



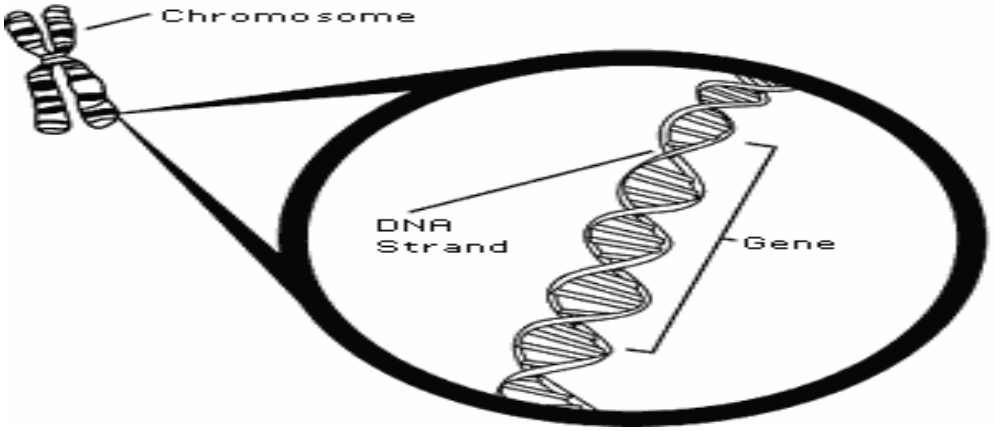
Basic Concepts

- GA converts design space into genetic space
- Works with a coding variables
- Traditional optimization techniques are deterministic in nature, but GA uses randomized operators
- Three important aspects:
  1. Definition of objective function
  2. Definition and implementation of genetic representation
  3. Definition and implementation of genetic operators

Biological Background

Each cell of a living organisms contains ***chromosomes*** - strings of *DNA*

- Each chromosome contains a set of ***genes*** - blocks of DNA
- A collection of genes – ***genotype***
- A collection of aspects (like eye colour) – ***phenotype***
- Reproduction involves recombination of genes from parents
- The ***fitness*** of an organism is how much it can reproduce before it dies
- Genetic operators



Search Space

- The space for all possible feasible solutions
- Each feasible solution can be "marked" by its value of the fitness of the problem
- GA is started with a set of solutions called ***populations***
- Used to form a new population
- More suitable, more chances to select
- This is repeated until best population is obtained

Fitness Function

- Particular solution may be ranked against all
- Measures the quality of the represented solution
- It is always problem dependent
- The fitness of a solution is measured, how best it gives the result
- For instance, Knapsack problem

  Fitness Function = Total value of the things in the knapsack

Simple Genetic Algorithm

Step 1: Encoding of the problem in a binary string

Step 2: Random generation of a population

Step 3: Calculate fitness of each solution

Step 4: Select pairs of parent strings based on fitness

Step 5: Generate new string with crossover and mutation until a new population has been produced

Repeat step 2 to 5 until satisfying solution is obtained

Encoding

- The process of representing the solution in the form of a string

Binary Encoding – Every chromosome is a string of bits, 0 or 1

Chromosome A	101101100011
Chromosome B	010011001100

Ex: Knapsack Problem

(There are things with given values and size. The knapsack has a given capacity. Select things to minimize their value in knapsack not exceeding the capacity of the knapsack)

Encoding: Each bit specifies if the thing is in knapsack or not

Binary encoding gives many possible chromosomes even with small number of alleles. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after genetic operator corrections.

In order to use GA to solve the maximization or minimization problem, unknown variables X_i are first coded in some string structures. It is important to mention that coding of the variable is not absolutely necessary. There exist some studies where GAs are directly used on the variables themselves, but here we shall ignore the exceptions and discuss the encoding for simple genetic algorithm. Binary-coded strings having 1s and 0s are mostly used. The length of the string is usually determined according to the desired solution accuracy. For example, 4-bit binary string can be used to represent 16 numbers

Other encoding ways are:

-Octal encoding

-Hexadecimal encoding

Permutation Encoding – Every chromosome is a string of numbers, which represents the number in the *sequence.* Used in ordering problems.

Ex: Traveling Sales Person Problem (There are cities and given distances between them. Travelling salesman has to visit all of them. Find the sequence of cities to minimize the travelling distance)

Encoding: Chromosome represents the order of cities, in which the salesman will visit them

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

Value Encoding – Every chromosome is a string of some values. Values can be form numbers, real numbers or characters.

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

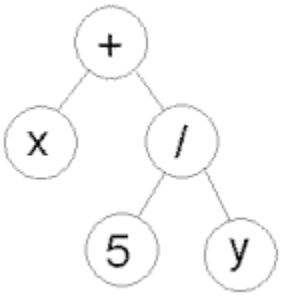
Ex: Finding weights for neural network

***The problem*:** To find the weights of connecting input to hidden layer and hidden layer to output layer

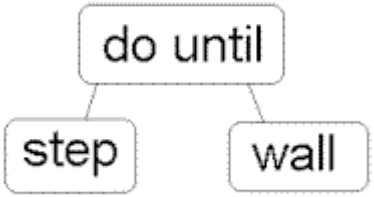
Encoding: Each value chromosome represent the corresponding weights

Tree Encoding – Every chromosome is a tree of some objects, such as functions or commands in a programming language

- Used for evolving programs or expressions in Programming Languages



(+ x (/ 5 y))



(do_until step wall)

- *Ex:* Finding a function from given values

The problem: Some input and output values are given. Task is to find a function, which will give the best relation to satisfy all values.

Encoding: Chromosome are functions represented in a tree

Fitness Function

As pointed out earlier GAs mimic the Darwinian theory of survival of the fittest and principle of nature to make a search process. Therefore, GAs are usually suitable for solving maximization problems. Minimization problems are usually transformed into maximization problems by some suitable transformation. In general, fitness function $F(X)$ is first derived from the objective function and used in successive genetic operations.

Certain genetic operators require that fitness function be non-negative, although certain operators do not have this requirement. Consider the following transformations

$$F(X) = f(X) \text{ for maximization problem}$$

$$F(X) = 1/f(X) \text{ for minimization problem, if } f(X) \neq 0$$

$$F(X) = 1/(1 + f(X)), \quad \text{if } f(X) = 0$$

A number of such transformations are possible. The fitness function value of the string is known as *string's fitness*.

Operators of GA

- Three Basic operators:
  1. Reproduction
  2. Crossover
  3. Mutation
- The new poopulation is further evaluated and tested for termination
- If the termination criteria are not met, the population is iteratively operated
- One cycle of operations and the subsequent evaluation – ***Generation*** in GA

Reproduction

- Reproduction is the first operator applied on population.
- Chromosomes are selected from the population to be parents to crossover and produce offspring. Best ones should survive and create new offspring.
- Also known as *Selection Operator*
- Parents are selected according to their fitness
- There are many methods to select the best chromosomes
  1. *Roulette Wheel Selection*
  2. *Rank Selection*
  3. *Tournament Selection*
  4. *Steady State Selection*
  5. *Elitism*
- The better the chromosomes are, the more chances they have to get selected.

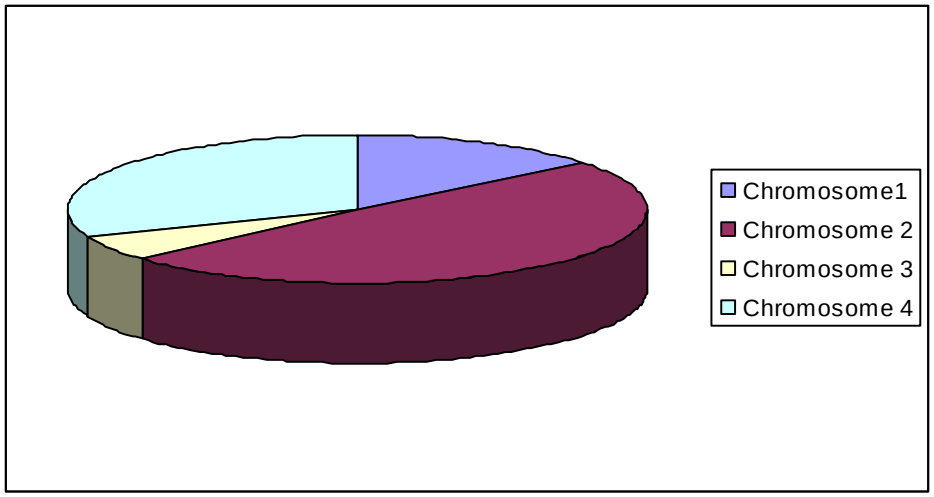
Roulette Wheel Selection:

- Main idea: A string is selected from the mating pool with a probability proportional to the fitness
- Probability of the ith selected string:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

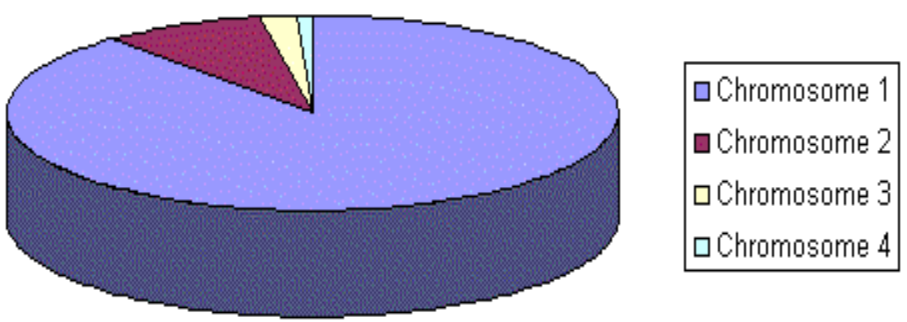
- "f_i " is fitness value for that string and "*N*" is population size.
- Chance to be selected is exactly proportional to fitness
- Chromosome with bigger fitness will be selected more times

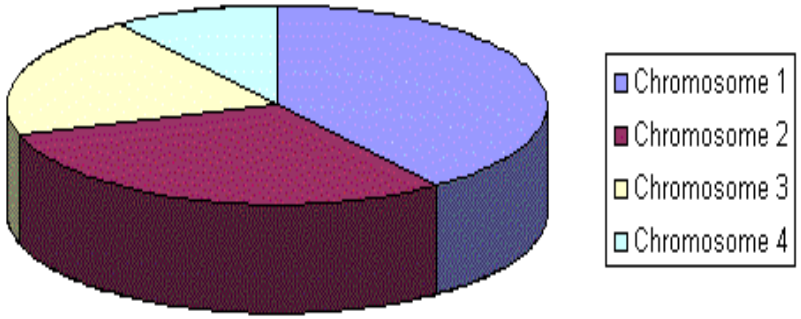
No.	String	Fitness	% Of Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0



Rank Selection:

- Main idea: First ranks the population and then every chromosome receives fitness from this ranking
- Roulette wheel will have problem when the fitness values differ very much.
- The worst will have fitness 1, second worst 2 etc. and the best will have fitness N





- After this all the chromosomes have a chance to be selected. But this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones
- Disadvantage: the population must be sorted on each cycle

Tournament Selection:

- *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

Steady-State Selection:

- Main idea: Big part of chromosomes should survive to next generation
- Working:
  - In every generation is selected a few (good - with high fitness) chromosomes for creating a new offspring
  - Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place
  - The rest of population survives to new generation

Elitism:

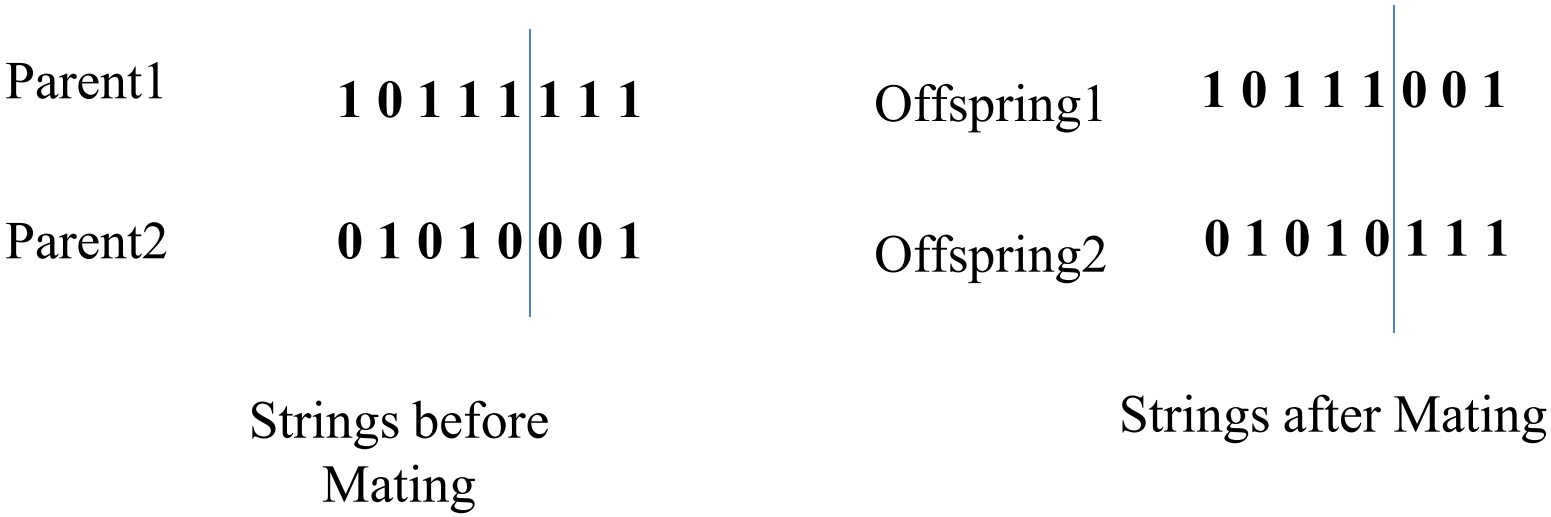
- Main idea: copy the best chromosomes (solutions) to new population before applying crossover and mutation
- When creating a new population by crossover or mutation the best chromosome might be lost
- Forces GA to retain some number of the best individuals at each generation.
- Improves performance

Crossover

- Applied to create a better string
- Proceeds in three steps
  1. The reproduction operator selects at random a pair of two individual strings for mating
  2. A cross-site is selected at random along the string length
  3. The position values are swapped between two strings following the cross-site
- *MAY NOT ALWAYS* yield a better or good solution
- Since parents are good, probability of the child being good is high
- If offspring is not good, it will be removed in the next iteration during “Selection”

Single Point Crossover:

- A cross-site is selected randomly along the length of the mated strings
- Bits next to the cross-site are exchanged
- If good strings are not created by crossover, they will not survive beyond next generation



Two-point Point Crossover:

- Two random sites are chosen
- The contents bracketed by these are exchanged between two mated parents

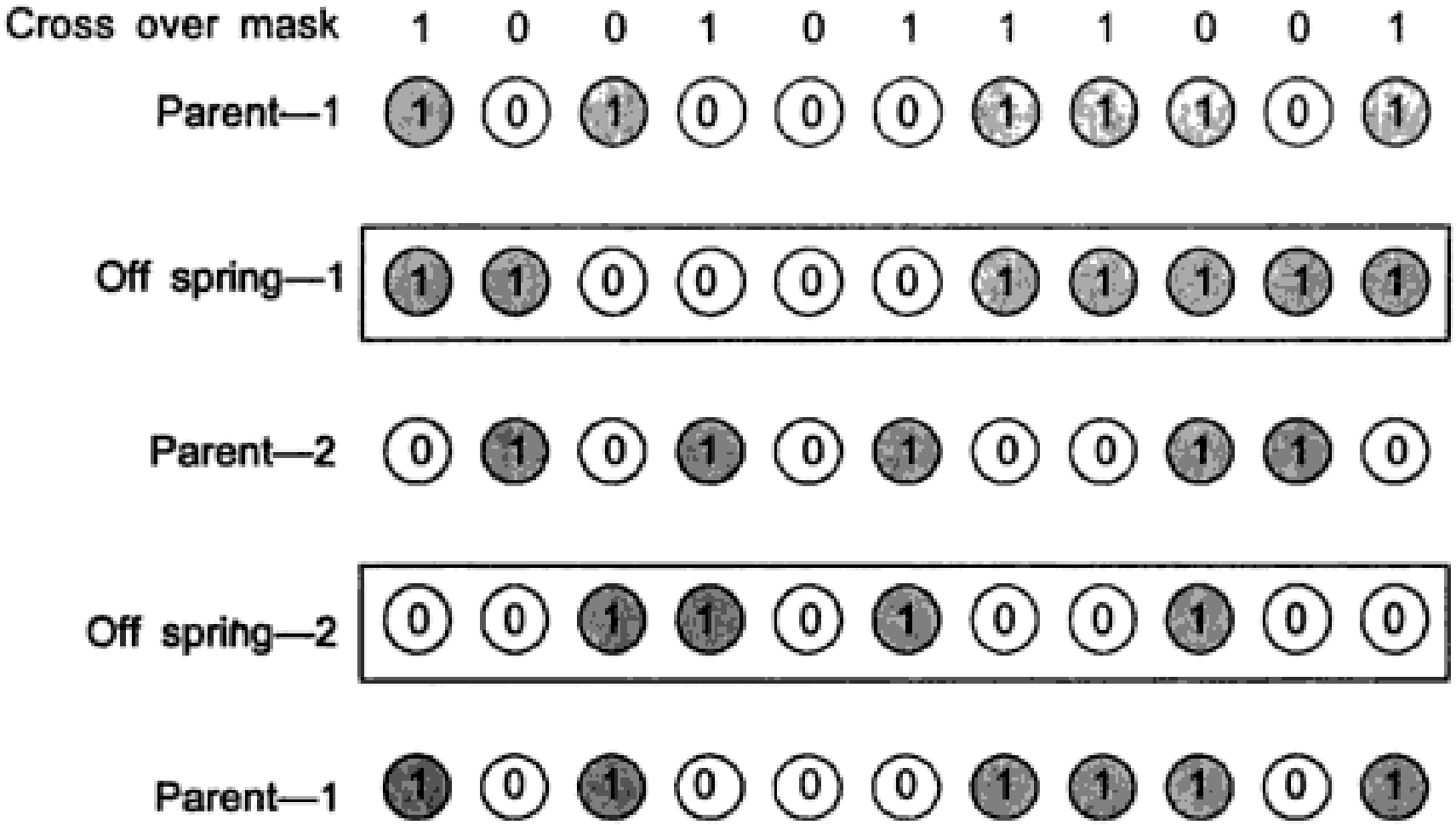
Parent1	**1 0 0 1 0 1 1 1**	Offspring1	**1 0 0 1 0 0 1 1**
Parent2	**0 1 1 1 0 0 0 1**	Offspring2	**0 1 1 1 0 1 0 1**
Strings before Mating		Strings after Mating	

Uniform Crossover:

- Bits are randomly copied from the first or from the second parent
- A random mask is generated
- The mask determines which bits are copied from one parent and which from the other parent

- **Mask: 0110011000 (Randomly generated)**

Parent 1	**1 0 1 0 0 0 1 1 1 0**
Parent 2	**0 0 1 1 0 1 0 0 1 0**
Offspring 1	**0 0 1 1 0 0 1 0 1 0**
Offspring 2	**1 0 1 0 0 1 0 1 1 0**



Uniform cross over using mask.

Mutation

After cross over, the strings are subjected to mutation. Mutation of a bit involves flipping it, changing 0 to 1 and vice versa with a small mutation probability Pm.

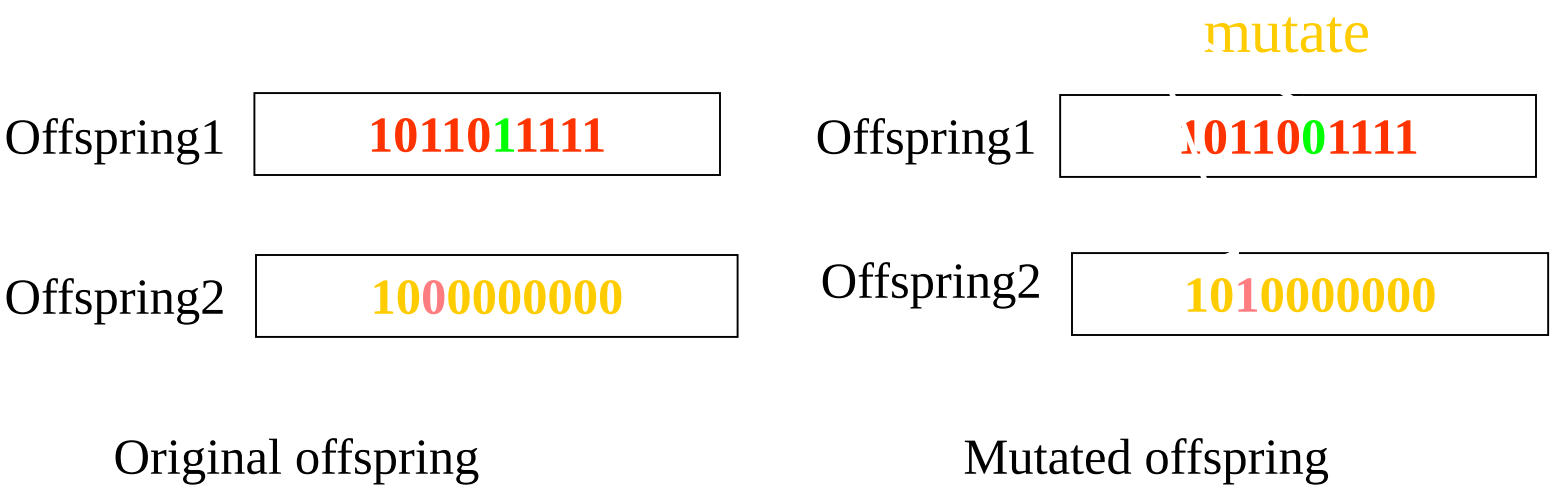
The bit wise mutation is performed bit by bit by flipping a coin with a probability of Pm.

A number between 0 to 1 is chosen at random. If the random number is smaller than Pm then the outcome of coin flipping is true, otherwise the outcome is false. If at any bit, the outcome is true then the bit is altered, otherwise the bit is kept unchanged.

The bits of the strings are independently muted, that is, the mutation of a bit does not affect the probability of mutation of other bits. A simple genetic algorithm treats the mutation only as a secondary operator with the role of restoring lost genetic materials. Suppose, for example, all the strings in a population have conveyed to a zero at a given position and the optimal solution has a one at that position, then cross over cannot regenerate a one at that position while a mutation could. The mutation is simply an insurance policy against the irreversible loss of genetic material.

Mutation

- Restores lost genetic materials
- Mutation of a bit involves flipping it, changing 0 to 1 and vice versa



Parameters of GA

Crossover probability:

- How often will be crossover performed
- If there is no crossover, offspring is exact copy of parents
- If there is a crossover, offspring is made from parts of parents' chromosome
- If crossover probability is 100%, then all offspring is made by crossover
- If it is 0%, whole new generation is made from exact copies of chromosomes from old population

Mutation Probability:

- How often will be parts of chromosome mutated
- If there is no mutation, offspring is taken after crossover (or copy) without any change
- If mutation is performed, part of chromosome is changed
- If mutation probability is 100%, whole chromosome is changed
- If it is 0%, nothing is changed

Population Size:

- Describes how many chromosomes are in population
- If there are too few chromosomes, GA have a few possibilities to perform crossover and only a small part of search space is explored
- If there are too many chromosomes, GA slows down

Example

Traveling Salesman Problem (TSP)

- Assume number of cities N = 8
- **Representation:** it is an ordered list of city

  1) London 3) Dublin 5) Beijing 7) Tokyo

  2) Venice 4) Singapore 6) Phoenix 8) Victoria

  CityList1 (3 5 7 2 1 6 4 8)

  CityList2 (2 5 7 6 8 1 3 4)

- **Crossover:**

  Parent1 (3 5 7 2 1 6 4 8)

  Parent2 (2 5 7 6 8 1 3 4)

  Child (5 8 7 2 1 6 3 4)

- **Mutation:**

  Before: (5 8 7 2 1 6 3 4)

  After: (5 8 6 2 1 7 3 4)

Issues for GA Practitioners

- Basic implementation issues:
  - Representation
  - Population size, mutation rate, ...
  - Selection, deletion policies
  - Crossover, mutation operators
- Termination Criteria
- Performance, scalability
- Solution is only as good as the evaluation function (often hardest part)

Benefits of Genetic Algorithms

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Always an answer; answer gets better with time
- Inherently parallel; easily distributed

Benefits of Genetic Algorithms (cont.)

- Many ways to speed up and improve a GA-based application as knowledge about the problem domain is gained
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications
- Substantial history and range of use

When to Use a GA

- Alternate methods are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using a GA
- Want to hybridize with an existing method
- Benefits of the GA technology meet key problem requirements

Some GA Application Types

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