Chapter 2

Literature Review

The chapter is organized as following: Section 2.1 revisits basic concepts of distributed database query processing, highlighting the architecture of a general query processor & introduces to use of a database profile for query optimization purposes. Section 2.2 contains the Literature Review and assesses various previous works in the fields of fragmentation and general distributed query processing. Section 2.3 and 2.4 introduce Evolutionary Computing Techniques and Genetic Algorithm fundamentals.

2.1 Distributed Database Query Processing

Query Processing in Distributed Relational Database involves translation of high level language user queries (like SQL) from one of the nodes of network sites, into a sequence of low level (Relational Algebra) operations. This decomposition of a calculus query into an algebraic query can be broadly divided into two components: the first part analyzes the query at the site of query generation and data accessed by the query is localized so as to translate the operations to bear on local data fragments at different sites locally. Finally the algebraic query on fragments must be re organised to minimize the use of computing resources such as I/Os, CPUs, and communication network [Ozsu et al. (2006)], [Kuan Tsae,1982].

There are two popular measure of resource consumption, the Total Cost and Response Time of the query [Saco & Yao, (1982)]. Total cost is the sum of all times incurred in processing the various operations of the query and intersite communication. Response time is the time elapsed from origin to completion of the query [Ozsu et al. (2006)]. In this thesis the approach adopted for query optimization is Total Cost Minimization of OLTP(Online Transaction Processing) queries .Queries are categorized mainly in two groups, OLTP and DSS(Decision Support System) queries. The former are generally repetitive in nature and hence optimizer should seek good throughput by
concentrating on reducing the total cost of the query. DSS queries more often seek response time optimization and use of parallel processing.

### 2.1.1 Architecture of a General Query Processor

![Architecture of a General Query Processor](image)

- **Parser**: The query is parsed and translated into an internal representation (tools like *flex* and *bison* can be used for the construction of SQL parser). [Kossman(2000)]

- **Query Rewrite**: query rewrite transforms a query in order to carry out optimizations that are good regardless of the physical state of the system (Few of transformations are like: elimination of redundant predicates, unnesting of subqueries, and simplification of expressions). Query rewrite is carried out by a rule engine.
• Query Optimizer (QO): this component carries out optimizations that depend on the physical state of the system. (QO) decides which index, which method, and in which order to execute operations of a query.

• Query optimizer: in distributed system Query optimizer (QO) must decide at which site each operation is to be executed. QO enumerates alternative plans and chooses the best plan using a cost estimation model

• Plan: specifies precisely how the query is to be executed. The nodes are operators, and every operator carries out one particular operation. The edges represent consumer-producer relationships of operators.

• Plan Refinement/Code Generation: this component transforms the plan into an executable plan. In some systems this transformation involves the generation of an assembler like code to evaluate expressions and predicates efficiently.

• Query Execution Engine: provides generic implementation for all involved operators like send, scan, nested loop join, etc. Operators are usually implemented as iterators with a common interface.

• Catalog: stores all the information needed in order to parse, rewrite and optimize a query. It maintains all vital stats about state of a database like, schema, view definitions, functions, integrity constraints, location and copies of partitions of tables etc.

2.1.2 Database profile

A cost based query processor needs the database profile to calculate the cost of alternative execution plans and will be used extensively in later part of thesis for implementing GA_FA and GA_SA. A typical Database profile contains following information about database.

• The number of tuples in each relation Ri (card(Ri))

• The size of each attribute A (size(A) )

• The size of Ri is sum of the sizes of its attributes
• For each attribute A in each relation Ri: the number of distinct values appearing in Ri (val(A[Ri])), max and min

Profile of partial results of an algebraic operation - SELECTION:

Let S denote the result of performing a unary relation over a relation R

• **Cardinality** - to each selection we associate a selectivity factor \( \rho \) which is the fraction of tuples satisfying the Selection Operation

Selection attribute = value (A=v),

Selectivity \( \rho \) is defined as follows:

\[
\rho = \frac{1}{\text{val}(A[Ri])}
\]

under an assumption that attribute values are homogeneously distributed.

Thus \( \text{card}(S) = \rho \times \text{card}(R) \)

• **Size**: selection does not affect the size of tuple of relations

\[
\text{Size_of_tuple}(S) = \text{size_of_tuple}(R)
\]

• **Distinct values**: depends on the selection criterion, and may be computed as follows [Selinger et al. 1979][Ozsu et al. 2006]

\[
\text{SFs} (A = \text{value}) = \frac{1}{\text{card}(\pi_A(R))}
\]

\[
\text{SFs} (A > \text{value}) = \frac{\text{max}(A) - \text{value}}{\text{max}(A) - \text{min}(A)}
\]

\[
\text{SFs} (A < \text{value}) = \frac{\text{value} - \text{min}(A)}{\text{max}(A) - \text{min}(A)}
\]

etc.

Profile of partial results of another algebraic operation - PROJECTION

Let S denote the result of performing a unary relation over a relation R

**Cardinality** – projection affects the cardinality of operands if duplicates are eliminated from the result. The following three rules can be applied

– If the projection involves a single attribute A, then set \( \text{card}(S) = \text{val}(A[R]) \)
If the product $\Pi_{A_i \in \text{Attr}(S)} \text{val}(A_i[R])$ is less than $\text{card}(R)$, where Attr(S) are the attributes in the result of the projection, set

$$\text{card}(S) = \Pi_{A_i \in \text{Attr}(S)} \text{val}(A_i[R])$$

- If the projection includes a key of R, set

$$\text{card}(S) = \text{card}(R)$$

- If the system does not eliminate duplicates, the cardinality of the result is the same as the cardinality of the operand relation

- **Size**: the size of the result of a projection is reduced to the sum of the sizes of attributes in its specification

- **Distinct values**: the distinct values of projected attributes are the same as in the operand relation

### 2.2 Literature Review

The Literature Survey briefly examines the different key research works in the field of Distributed Query Optimization since last four decades. Attribute Partitioning is discussed exclusively in first part of literature survey and then in second part, literature on general query optimization, data & operation allocation is reviewed.

Most of the Distributed query optimization literature vouches for procedures aiming at either reducing ‘Total Time Cost’ of a query or ‘Response Time’ of a query. Total Time models aims to reduce overall utilization of system and network resources, and hence reduces commercial cost of a query. Response Time model aims to reduce the response time of query by parallelizing independent operations.

#### 2.2.1 Previous Work on Attribute Partitioning (Fragmentation):

[Kennedy (1973)] was first to give a mathematical model for attribute partition through computation of ‘data accessing cost’ by the transactions involved.[Hoffer & Severance(1975)] defined a measure of affinity between a pair of attributes based on which partitioning is carried out. Later Hoffer developed a non linear, zero one program for the solution of the vertical partitioning problem which minimizes a linear combination of storage, retrieval and update costs, under capacity constraints assigned to each subfile.
[March & Severance(1977)] extended the model to incorporate block factors for both primary and secondary memory. [Hammer and Namir(1979)] demonstrated heuristic nature of general vertical partitioning problem. Hoffer and Sevrance measured the “affinity between pairs of attributes”, and used the Bond Energy Algorithm (BEA) to cluster attributes. This approach had a drawback of leaving the cluster design for database designer. [Hoffer(1976)] introduced another method to reduce cost of a linear combination of storage, retrieval and updation operations. Non Linear zero-one programming was used for optimization. Primary and Secondary record fragmentation and access cost analysis was introduced by [Eisner & Severance (1976)].

[Schkolnic(1977)] addressed the problem of minimizing access time by partitioning for hierarchical databases. [Garey & Johnson(1979)] demonstrated the NP-Hardness of Vertical Partitioning problem. All the earlier approaches suffered from high computation complexity and when number of relations, fragments and sites increased they become unsuitable. Henceforth further approaches to partition problem were heuristic based. Approach followed in this thesis of using Genetic Algorithm is heuristic too. These approaches can withhold the computation complexity of large scale problems.

[Hammer & Namir(1979)] proved that for large size problems optimal methods go computational intractable, so use of heuristics is a better choice. They used a hill climbing heuristic based on ‘File Usage Pattern’ and ‘Attribute Characteristics’ for exploring a Greedy Algorithm, for access cost optimization. [Navathe et al.(1984)] proposed a binary iteration algorithms for reducing disk access costs. They made an extensive use of number, length, cardinality, selectivity of cardinality etc for cost calculation. [Cornell & Yu(1987)] present a two phase approach for determination of fragments. [Cornell & Yu (1990)] extended their earlier works and presented a recursive binary partitioning algorithm to reduce disk access costs.

[Ceri et al. (1989)] proposed a divide and conquer approach, whereas [Navathe et al. (1989)] presented a graphical Algorithm for vertical Partitioning that generates all fragments consecutively at every iteration to reduce costs. Thus far no approach concentrated on the effectiveness or “goodness level” of the partition algorithm. [Chakravarthy et al.1993] came up with an idea of Partition Evaluator for evaluating goodness of a partition scheme.

Algorithms mentioned so far may be grouped into category of scheme of fragmentation based on relation attributes. Second category of partition algorithms that
are based on transactions emerged in [Chu et al. (1993)]. Chu used ‘Branch & Bound’ technique to optimize disk access costs. A composite algorithm for fragmentation and allocation of fragments to various sites, using ‘Simulated Annealing’ method was presented by [Perez et al. (1998)]. They achieved the goal of optimizing disk access costs by concentrating on minimizing storage, access, migration and transfer costs of fragments on various sites.

[Barker et al. (2006)] came up with a novel idea of applying restricted growth to chromosome string: GRGS-GA (Group Oriented Restricted Growth String - GA) to exclude redundant chromosomes from further GA process. They applied them for m-way partitioning of relations. They designed two new crossover operators and four new mutation operators. They further proved the effectiveness of using “Binary Merge Crossover Operator” and “Merge and Jump Mutation Operators” for large scale partitioning problems.

[Amossen (2009)] presented two algorithms for vertical partitioning of relational OLTP databases using integer programming. He presented a model that, given a schema along with vertical partitioning on a workload, it estimates the costs (bytes read/written by storage layer access methods and bytes transferred between sites) of evaluating the workload on given partitioning. It was tested on TPC-C benchmark database transactions and showed 37% reduction in disk access costs as compared to GA of [Barker et al. (2006)]

[Goli & Rankooohi (2011)], Presented an optimization algorithm for vertical partition, based on a heuristic resulting from integration of ant clustering algorithm emerged , very recently in 2011., It addresses computational complexity as well the accuracy of obtained results with that of GRGS-GA (Group Oriented Restricted Growth String - GA) of [Barker et al. (2006)].

### 2.2.2 Previous work on Distributed Query Optimization

[Wong (1977)], [Epstein et al. (1978)]

Wong’s algorithm is one of the earliest works providing a comprehensive solution to the distributed query processing problem. Wong’s hill climbing algorithm translates a distributed Query Q into a sequence of relational algebra operations (selection, projection
and joins). It defines an operation **MOVE** for moving relations from one site to another. An **Initial Feasible Solution** is solution is constructed after deciding the query destination site $S_0$, then **Moves** all relations $R$ referenced by $Q$ to $S_0$. Processes $Q$ at $S_0$ as if it was a local query. A recursive improvement is made in the solution by moving lower cost sequences of **MOVEs** and relational algebra operations until no further lower cost sequence is produced. It can be considered as a greedy heuristic. Wong’s Algorithm was adopted by earlier versions of distributed INGRES.

[Hevner & Yao(1979)]

It was the first successful attempt at using semi joins in a distributed query process. After performing the local processing part of simple queries, each relation contains one common joining attribute. Authors generalized the algorithm for equi-join queries. They introduced the concept of **Selectivity** for join operations, as number of domain values currently appearing in the joining columns divided by the total domain values. They assumed that selectivity of one joining domain does not affect selectivity of other joining domain. A heuristic algorithm with improved exhaustive search was proposed for general queries. This approach also suffered from scaling problems, in the way that if number of relation joins and number of sites involved move into double digit figure, algorithm computing time increases to exponential growths and quickly goes intractable.

[Goodman et al. (1981)], [Bernstein et al.(1980)]

The authors also proposed an algorithm that used a semi join approach as of Wong. They approached the query processing problem in a two phase problem. Phase 1 was called a **Reducer Phase** and Phase 2 was called **Final Processing Phase**. Phase 1 took care of full reducer problem, by finding an optimal sequence of semi join operations that reduced the inter-site data transmission requirements. In final processing phase, a site is selected for finally producing query results. A heuristic algorithm prunes unnecessary semi-joins due to final selection of site and reduced data from other sites are transmitted here. SDD-1 used this approach for implementing its query strategy.[Hevner and Yao (1979)] & [Yu et al. (1982)] proved that combinatory nature of query processing problem is NP Hard.
Authors studied the use of semi-joins for chain queries for a distributed database query. A powerful and efficient dynamic programming algorithm was developed that translated a chain query into sequence of semi joins. It has a computing complexity of the order of $O(n^3)$, where $n$ is the number of relations referenced by the query. Finally they extend their work to optimize a larger class of queries called tree queries.

Chen & Li (1984) & Yu and Chang (1984), Extended the work of Chiu by proposing certain properties of a tree query to check the usefulness of a join sequence or a non optimality of one. Then they extend this work by imposing more restrictions on tree queries to convert them into star queries. Gavish & Segev (1982) proposed a mathematical model for a special set of queries and two way join queries later implemented by Segev (1986). They prove the problem to be NP Complete and propose a heuristic solution by partitioning data horizontally and no use of semi joins.

Apers and others in their classic paper [Apers et al. (1983)] propose three versions of an optimization algorithm for arbitrary complex queries, one for minimizing the response time and two for minimizing total time cost of a distributed query under certain conditions for a distributed database. Perrizo (1984) extended this work to propose greedy approach for a pair of joins instead of single join considered earlier [Perrizo (1984)]. Blankinship, Hevner, Yao present an Iterative heuristic method for distributed database query optimization and data allocation [Blankinship et al. 1997]. The optimization heuristic iterates between finding minimum cost query strategies and minimum cost data allocation until a local minima for the combined problem is found. Apers developed a distributed data allocation algorithm that utilized an actual query processing schedule to implement a virtual network of sites without assigning relations to sites.

Describe how Processing schedules are generated for distributed query optimization. Virtual data sites are merged into actual network sites to find an optimal data allocation plan, by minimizing intermediate relation to relation transmission costs. Finally the query result is sent to originating site. The drawback of the approach was intractability in case of large size problems [Apers, 1982].
Sakti & Vineyard (1988) presented a semijoin reducer cover set based technique for optimizing join queries in distributed databases. The technique converts semijoin program into a partial order graph which allows concurrent processing of semi join programs.

Douglas W, Cornell and Philip S Yu, developed a methodology to assign relations and determine the join sites simultaneously. It decomposes queries into relational algebra operations and then makes site assignments based on a linear integer programming technique to minimize the intersystem communication. It describes procedures for balancing resource utilization across systems. Further it uses a heuristic technique to minimize average response time [Cornell & Yu (1989)].

T.P. Martin, K.H. Lam & Judy Russell demonstrate the cost effectiveness of four algorithms, Branch & Bound, Greedy, Local Search and Simulated Annealing for site selection during the optimization of compiled queries in a large replicated distributed database system. They conclude that enumerative algorithms are best suited for simple queries and recommend a local search algorithm for complex queries [Martin et al. (1990)].

Salvatore T. March, S. Rho in their paper on Allocating Data and Operations to Nodes in Distributed Database Design [March and Rho, (1995)], extend the modelling of Cornell and Yu and integrate the work of Apers in determining file fragments for allocation. It also illustrates a comprehensive mathematical model for data allocation, retrieval and update queries across a set of sites of a network.

Huang, Chen (2001) proposed a simple and comprehensive model for Fragment Allocation in Distributed Database Design that reflects a transaction behaviour. Deshpandem & Hellerstein (2001) studied the problem of query optimization in federated database systems and highlighted the need of decoupling various aspects of query processing. They implemented it on ‘Cohera’ federated database system and demonstrated superiority of 2PO algorithm in case of pre known physical design of database.[Cui & Lin 2004] gave a multi-objective genetic algorithm for distributed database management. They formulated a multiobjective combinatorial optimization problem based on ‘Pereto Dominance’ and ‘Pereto Optimality’. Multiple criteria are developed with a goal to provide trade-off optimal performances for web services.

Barker et al (2006) came up with a novel idea of applying restricted growth to chromosome string :GRGS-GA (Group Oriented Restricted Growth String - GA) to
exclude redundant chromosomes from further GA process. They applied them for m-way partitioning of relations. They designed two new crossover operators and four new mutation operators. They further proved the effectiveness of using “Binary Merge Crossover Operator” and “Merge and Jump Mutation Operators” for large scale partitioning problems.

Zehai Zhou(2007) proposed using heuristics and genetic algorithms for large scale database query optimization. The NP Hard problem is reduced to a join ordering problem similar to a variant of a Travelling Salesman Problem. Several heuristics and a GA were proposed for solving the join order problem. Computation experiments showed the heuristics and GA are viable methods for large scale distributed query optimization.

Hababeh et al.(2007) explored an integrated method for grouping the distributed sites into clusters and customizing the database fragments allocation to the clusters and their sites. They design a high speed clustering and allocating method to determine which fragments will be allocated to which cluster and site so as to maintain data availability and a constant systematic reliability. Performance of the method is evaluated and tested over different network sites and results are highlighted in a tabular and graphical representation.

Li & Luo (2008) present a tree based GA with new coding method of genetic parameters with tree structure based on position and value. Improved Crossover and Mutation operators are devised to claim improved stochastic coding rules of genetic algorithms.[Ghaemi et al. (2008)], in their paper on evolutionary query optimization for heterogeneous distributed database systems, discuss a multiagent based architecture and use of genetic algorithms. They demonstrate the superiority of the GA over Dynamic Programming methods in case of large scale problems.

Rahmani, Torkzaban, & T.Haghighat (2009), proposed an innovative model by clustering sites based on the cost of communication between sites, to allocate data on nodes of a Distributed Database. Authors claim significant reduction in the data redundancy in fragment allocation and network traffic [Rehmani et al.(2009)].

Sevinc & Cosar(2011), proposed a new Genetic Algorithm (NGA) for distributed database query optimization. They experimented on a synthetic database with replicated
relations, without fragmentation. They also investigated the effect of increasing the message size, number of nodes and relations on the performance of a GA.

2.3 Introduction to Evolutionary Computing

Evolutionary Computing Algorithms are designed to mimic the working and robustness of natural biological systems. The main focus of designers of artificial systems is to somehow try to mimic the robustness, efficiency and flexibility exhibited by natural biological systems, through features of self repair, self guidance, instinct to select a mate and reproduce a fit child. These are naturally there for biological systems (By God’s Grace!), whereas they barely exist even in the most sophisticated artificial (Man Made!) systems [Holland, (1975)] [Goldberg (1989)] [Townsend(2003)]. Evolutionary Computing Algorithms are mainly used by two approaches, first: Search and Optimization Approaches, second: Approximate Reasoning Approach.

Figure 2.2: GA’s place in hierarchy of Evolutionary Computing
2.3.1 Evolutionary Computing

In contrast to traditional computing referred as “Hard Computing”, Soft Computing, that includes Evolutionary Computing, comprises of a collection of algorithms based on the evolution of a population towards a solution of a certain problem with features like tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness, low solution cost and better rapport with reality [Townsend(2003)]. It can be divided into following two main categories:

1. Approximate Reasoning
   a) Probabilistic Reasoning
   b) Fuzzy Logic
2. Search/Optimization
   c) Neural Networks
   d) Evolutionary Algorithms

2.3.2 Evolutionary Algorithms

Evolutionary Algorithms are characterized by the general property that a population of possible solutions evolve from one generation to the next, ultimately arriving at the search or optimization goal of the problem. These are inspired by the Charles Darwin’s famous “Survival of the Fittest” theory of biological evolution. There are many types of evolutionary algorithms which differ from each other in a way new population evolves from a present one e.g. Simulated Annealing belongs to Gradient Methods, Random Search Techniques, Tabu Search, Random Search Algorithms and Genetic Algorithms.

2.4 Genetic Algorithms

Genetic Algorithms have their origin in works of John Holland and his colleagues and students at the University of Michigan [Holland (1975)]. Original goal of the research was to explain the adaptation of natural systems and to design artificial systems that try to embrace adaptive and robust properties of natural systems.

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string
structures with a structured yet randomized information exchange to form a search algorithm to produce a new generation. It consists of group of individuals whose average fitness is better than previous generation as in case of natural adaptation [Goldberg (1999)].

Genetic Algorithms work as Approximate Algorithms to solve NP-Hard problems. These algorithms can be useful for finding a suitable solution, but these algorithms do not necessarily provide the best solution. The solutions found by these methods are often considered as good solutions, because it is not often possible for NP Hard problems to prove what the optimum solution is.

Genetic algorithms are typically implemented as a computer simulation in which a population of abstract representations (called chromosomes) of candidate solutions (called individuals) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but different encodings are also possible. The evolution starts from a population of completely random individuals and happens in generations. In each generation, the fitness of the whole population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), modified (mutated or recombined) to form a new population, which becomes current in the next iteration of the algorithm.

Stating simply a genetic algorithm is a computer algorithm that searches for good solutions to a problem from among a large number of possible solutions. All GAs begin with a set of solutions (represented by chromosomes) called population. A new population is created from solutions of an old population in hope of getting a better population. Solutions which are then chosen to form new solutions (offspring) are selected according to their fitness. [Mitchell(1996)], [Reeves(1993)] [Goldberg(1999)], [<Wikipedia>].

2.4.1 A General Genetic Algorithm

In the following section, given is a general GA (Genetic Algorithm), highlighting use of basic operations of Selection, Crossover and Mutation.
GA_General: A _General_Genetic_Algorithm_

```plaintext
{ 
  t ← 0 ;  // Counter for No. Of Generations 
  Start_Poulation P(t);  // Initial population generation 
  Evaluation P(t);  // Evaluates Individual's Fitness 
  While {requisite number of generations Max(t), are not populated Or the solution fitness has
  not improved since last 10 generations} 
  begin 
      { t ← t + 1 ;  // Counter Increase 
        Selection P(t) from P(t-1);  // Select the couples 
        Crossover P(t);  // for crossover 
        Mutation P(t);  // Apply Mutation 
        Evaluation P(t);  // Evaluates Individual's new Fitness 
        Survive P(t)  // Select Survivor Individuals 
        Reort_Genration P(t);  // Print 't' Population Report 
    } 
  end; 
  Output the best solution found; 
}
```

**Figure 2.3: A General Genetic Algorithm (GA_General)**

### 2.4.2 GA Operators

The GA will generally include three fundamental genetic operators: Selection, Crossover & Mutation. These operations are used to select and manipulate a population of solutions and select the most appropriate offspring to pass on to succeeding generations [Townsend(2003)].

- **Selection**

The selection process is to choose, based on the fitness of a chromosome. Chromosomes that are evaluated with higher values (fitter) will most likely be selected to reproduce, whereas, those with low values will get lesser chance. The fittest chromosomes may be selected several times, however, the number of chromosomes selected to reproduce is equal to the population size, therefore, keeping the size constant for every generation.
This phase has an element of randomness just like the survival of organisms in nature. The most commonly used selection methods are following:

- Roulette-Wheel Selection
- Stochastic Universal Sampling
- Ranked selection
- Truncation Selection
- Tournament Selection

To increase the performance of GAs, the selection methods are enhanced by *elitism*. Elitism is a method, that first copies a few of the top scored chromosomes to the new population and then continues generating the rest of the population. Thus, it prevents losing the few best found solutions.

- **Crossover**

Crossover is the process of combining the bits of one chromosome with those of another. This is to create an offspring for the next generation that inherits traits of both parents. Crossover randomly chooses a locus (a bit position in chromosome string) and exchanges the sub sequences before and after that locus between two chromosomes to create two offspring. For example, consider the following parents and a crossover point at position 3:

<table>
<thead>
<tr>
<th></th>
<th>1 0 0</th>
<th>0 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent 1</td>
<td>1 1 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>Parent 2</td>
<td>1 1 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>Offspring 1</td>
<td>1 0 0 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>Offspring 2</td>
<td>1 1 1 0 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

In this example, Offspring 1 inherits bits in position 1, 2, and 3 from the left side of the crossover point from Parent 1 and the rest from the right side of the crossover point from Parent 2. Similarly, Offspring 2 inherits bits in position 1, 2, and 3 from the left side of Parent 2 and the rest from the right side of Parent 1.

Commonly used recombination or Crossover Techniques are following:

- One Point Crossover
- Two Point Crossover
- Uniform Crossover
- Shuffle Crossover
- Partially Matched Crossover (PMX)
- Order Crossover (OC)
- Cycle Crossover (CC)

- Mutation
  Mutation operation changes the new offspring by flipping bits from 1 to 0 or from 0 to 1. Mutation can occur at each bit position in the string with some probability, usually very small (e.g. 0.01). Two child chromosomes from a Crossover Operation are subjected to Mutation with a very small probability, to maintain a sufficient level of Genetic Variety, it acts as an insurance policy to prevent loss of genetic material. Mutation is performed after crossover to prevent falling all solutions in the population into a local optimum of solved problem.

  For example, consider a chromosome with mutation point at position 2:

  Not mutated chromosome: 1 0 0 1 1 1

  Mutated: 1 1 0 0 1 1 1 (The 0 at position 2 flips to 1 after mutation).

2.4.3 Parameters of a Genetic Algorithm

There are two basic parameters of GA - crossover probability and mutation probability.

- Crossover probability

It determines how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. If crossover probability is 100%, then all offspring are made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population. Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survive to next generation. Its value is usually high like 60 to 70% of population is generated thru crossover.
• **Mutation probability**

It determines how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, because then GA will in fact change to random search. [Goldberg(1999)]. It is usually applied to less than 1% of the population of a generation.

• **Population size**

It determines how many chromosomes are to be generated in a population (in one generation). If there are too few chromosomes, GA has few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to use very large populations because it does not solve the problem faster than moderate sized populations.

• **Chromosome’s Encoding & Fitness function**

The chromosomes in GAs represent the space of candidate solutions. Possible chromosomes encodings are binary, decimal, permutation value, and tree encodings. Gas require a fitness function which allocates a score to each chromosome in the current population. Thus, it can calculate how well the solutions are coded and how well they solve the problem . ”,[Mitchell(1996)], [Reeves(1993)][Goldberg(1999)], [<Wikipedia>].

### 2.4.4 Genetic Algorithm Lingo

Since GAs apply operations drawn from nature, the nomenclature used in this field is closely related to the terms we can find in biology. The next table summarizes the meaning of these special terms.

---

46
| **Genotype** | The code, devised to represent the parameters of the problem in the form of a string. (Structure) |
| **Chromosome** | One encoded string of parameters (binary, floating point number, etc...) |
| **Individual** | One of more chromosomes with an associated fitness value. |
| **Gene** | The encoded version of a parameter of the problem being solved. (Feature, Character or Detector) |
| **Allele** | Value which a gene can assume (A Feature Value: binary or integer). |
| **Locus** | The position that the gene occupies in the chromosome. (String Position) |
| **Phenotype** | Parameter Set, Problem’s alternative solution. |
| **Fitness** | Real value indicating the quality of an individual as a solution to the problem. |
| **Population** | A set of individuals with their associated statistics |
| **Selection** | Policy for selecting one individual from the population, usually based on fitness function. |
| **Crossover** | Operation that merges the genotypes of two selected parents to yield two new children. |
| **Mutation** | Operation that randomly changes one or more alleles of the genotype. |

Table 2.1: GA Lingo

2.5 Summary

In this chapter a review of basic concepts of distributed database query process is done, highlighting the architecture of a general query processor. It introduced concept of a database profile to design a cost function for cost evaluation of various alternative plans. Literature Review assessed previous works in the fields of fragmentation and general distributed query processing since last three decades. Evolutionary Computing was introduced and an exposure to Genetic Algorithm’s fundamental concepts was provided. Finally, Basic operators and parameters of an outlined general GA were discussed.