Biological evolution consists of processes or mechanisms like selection, reproduction, migration and survival of the fittest, which makes the individuals in the population adapt to the changing environment. Evolutionary computation is an optimization method based on these biological evolution processes. The algorithms inspired by evolutionary computation are known as evolutionary algorithms. These algorithms can be used to solve complex problems. The travelling salesman problem is one of the complex problems which is studied intensively in numerous domains. Various methods and techniques are applied to the travelling salesman problem to solve the problem, though none of the methods or techniques prove to get a solution to the problem in reasonable amount of time. Therefore various algorithms compete to get only better solutions than the already known. This also helps in checking the performance of the algorithms.

The aim of this chapter is to give the background and literature survey of the undertaken research area. It begins with a brief discussion of the main components of evolutionary algorithms, explaining their role as optimization technique. Among the evolutionary algorithms, the more popular genetic algorithm are described in detail. History and overview of genetic algorithm is given which provides a base for the current research work. Next the travelling salesman problem is defined. Its history is provided and few applications are listed, which shows the importance of travelling salesman problem. Further the basics of high performance computing is explained. Finally, a few parallel models for genetic algorithm are mentioned.
Chapter 2. Theory and Background

2.1 Evolutionary Algorithms

Evolutionary algorithms are a broad class of stochastic optimization algorithms introduced in the 1960s and achieved great popularity since 1980s. They simulate the behavior of natural evolution and treat the solution candidates of a given problem as individuals, that compete in the virtual environment. Evolutionary algorithms operate on population of potential individuals and apply the principle of *survival of the fittest* to produce better individuals. General scheme of an evolutionary algorithm is shown in Figure 2.1

![Figure 2.1: General Scheme of an Evolutionary Algorithm](image)

Initial population of individuals is generated. Competent individuals are selected. The selected individuals (parents) exchange information among themselves and reproduce offsprings. The offsprings which perform better in the environment become the part of the new generation. This process continues through multiple generations, which leads to the evolution of population of individuals that are better suited to the environment than their ancestors. The following features of evolutionary algorithms distinguishes them from other traditional search and optimization methods [16]:

- Evolutionary algorithms search a population of solutions/individuals in parallel, not just a single solution.
- They do not require derivative information or other auxiliary knowledge; only the objective function and corresponding fitness levels influence the directions of search.
- They use probabilistic transition rules, not deterministic ones.
- They are generally straightforward to apply, because no restrictions for the definition of the objective function exist.
- They can provide more than one potential solutions to a given problem.
- They can handle arbitrary kinds of constraints and objectives with ease.

The family of evolutionary algorithms comprises five major members as illustrated below:
Chapter 2. Theory and Background

2.1.1 Evolutionary Strategies

Ingo Rechenberg [17], of the Technical University of Berlin, invented the evolution strategy optimization technique, which is based on the ideas of adaptation and evolution. They primarily use mutation and selection, as search operators. Few evolutionary strategies are mentioned below:

(a) (1 + 1) Evolutionary Strategy: This is a two-member algorithm where, one parent generates one offspring in each generation by applying mutations. This means smaller steps are more likely than bigger ones, until a child performs better than its ancestor, and takes its place. To self-adapt (1 + 1) evolutionary strategy algorithm, the ratio between successful mutations and all mutations should come to 1/5. Hence the so called 1/5th success rule was discovered [17]. This is the simplest possible working scheme of evolutionary strategies.

(b) (mu + lambda) Evolutionary Strategy: The (1 + 1) evolutionary strategy algorithm, using mutation alone, has been enhanced to (mu + lambda) strategy, which incorporates recombination since, several parents are now available. It generates lambda individuals from a population of mu parents. From the mu parents and the lambda individuals taken together, it keeps the mu best individuals to constitute the new generation. This strategy is also termed as plus strategy, where the parental generation is taken into account during selection.

(c) (mu, lambda) Evolutionary Strategy: The (mu, lambda) evolutionary strategy algorithm generates lambda individuals from a population of mu parents (usually where lambda > mu). From the mu parents and the lambda individuals taken together, it keeps the mu best individuals to constitute the new generation. This strategy is also termed as comma strategy, where only the offsprings undergoes selection, and the parents become extinct.

2.1.2 Evolutionary Programming

Lawrence Fogel and his colleagues [18], of the University of California in San Diego, started their experiments on evolutionary programming, which is a stochastic optimization technique. This technique emphasizes on the behavioral linkage between parents and their offsprings rather than imitating specific genetic operations as observed in nature. Over the decades, it has more or less merged into genetic programming and the other evolutionary algorithms.
2.1.3 Genetic Algorithm

John Holland [19], of the University of Michigan, and his students invented the genetic algorithm, which is also an optimization technique that mimics biological evolution as a problem solving strategy. This heuristic is routinely used to generate useful solutions to optimization and search problems using techniques inspired by natural evolution, such as selection, crossover and mutation.

2.1.4 Learning Classifier Systems

Learning classifier systems [20], first described by John Holland, is a machine learning system, which is adaptive in nature and learns to perform the best action on the given input. It has close links to reinforcement learning and genetic algorithms. In fact internally it makes use of a genetic algorithm. Learning classifier systems consists of a population of binary rules on which a genetic algorithm is applied to alter and select the best rules.

2.1.5 Genetic Programming

John Koza [21], of the Stanford University, had used genetic algorithm to evolve programs to perform certain tasks. He called this method genetic programming. In genetic programming the objects that constitute the population are not fixed length character strings that encode possible solutions to the problem at hand. Instead each individual is a computer program that explores the algorithmic search space and evolves to perform a defined task. These programs are expressed in GP as “parse trees”, rather than as lines of code. LISP programs were used, because programs in this language can be expressed in the form of a “parse tree”. In genetic programming the crossover operation is implemented by taking randomly selected subtrees in the individuals and exchanging them. Genetic programming usually does not use any mutation as a genetic operator.

The above evolutionary algorithm methods have been classified according to their semantics corresponding to their search and problem spaces. They share a common conceptual base of simulating the evolution of individual structures via processes of selection, reproduction and/or mutation. They form the backbone of evolutionary algorithms, out of which genetic algorithms are most prominent and widely used.

Some example areas of application of evolutionary algorithms are: Function optimization [22, 23], Economics and Finance [24, 25, 26], Scheduling [27, 28, 29, 30], Image Processing [31, 32], Engineer-
Chapter 2. Theory and Background

ing, Structural Optimization and Design [33, 34], Robotics [35], Mathematical Problems [36] and many more.

2.2 Genetic Algorithm

This section concentrates on the genetic algorithm. It provides the history of genetic algorithm, which includes a theoretical background justifying why genetic algorithms work so well in practice. An overview of a simple genetic algorithm is given, which mentions the steps followed by the algorithm. Each step of the genetic algorithm is detailed. Finally, a few important parameters of the genetic algorithm are mentioned. As observed from the previous sections, genetic algorithms are an important sub-class of evolutionary algorithms. Genetic algorithm provides an alternative method for solving problems and outperforms other traditional methods in most of the problems. They are important search algorithms that judiciously do both exploration and exploitation of the search space. Genetic algorithms are used for optimization, although they cannot be considered only as optimizers. These computational models are problem solvers and can act as basis for competent machine learning.

2.2.1 Brief History

Genetic algorithms were invented by John Holland when he and his students conducted studies on cellular automata. Their studies resulted into this technique known as genetic algorithm. He published a book *Adaptation in Natural and Artificial Systems* in 1975, which popularized genetic algorithm. This book consists of basics of genetic algorithm as models of machine learning that derive their behavior from concepts of biological theory of evolution. The idea of genetic algorithm is to use the power of evolution to solve difficult optimization problems. One of the earliest attempts to understand genetic algorithm was through the schema theory. The notion of schemata and the schema theorem provides the theoretical background for explanations of why genetic algorithms work so well in practice.

Schemata are simple similarity templates, which describes a subset of strings with similarities at some portions. For example, consider binary strings of length 6. The schema 1*10*1 describes the set of all strings of length 6 with 1’s at positions 1, 3 and 6 and a 0 at position 4. The * is a wildcard symbol, which means that positions 2 and 5 can have a value of either 1 or 0. The order of a schema is defined as the number of fixed positions in the template, while the defining length \( \delta(H) \) is the distance between the first and last specific positions. The order of 1*10*1 is 4 and its defining length is 5. The fitness of a schema is the average fitness of all strings matching the schema. The fitness of a string is a measure of
the value of the encoded problem solution, as computed by a problem-specific evaluation function.

Using the established methods and genetic operators of the genetic algorithms, the schema theorem states that short, low-order schemata with above-average fitness increase exponentially in successive generations. It is expressed in Equation (2.1) [37]:

\[ E(m(H, t + 1)) \geq \frac{m(H, t)f(H)}{a_t}[1 - p] \quad (2.1) \]

where, \( m(H, t) \) is the number of strings belonging to schema ‘H’ at generation ‘t’. \( f(H) \) is the observed fitness of schema ‘H’ and ‘\( a_t \)’ is the observed average fitness at generation ‘t’. The probability of disruption ‘\( p \)’ is the probability that crossover or mutation that will destroy the schema ‘H’. It can be expressed as shown in Equation (2.2):

\[ p = \frac{\delta(H)}{l-1}p_c + o(H)p_m \quad (2.2) \]

where \( o(H) \) is the number of fixed positions, ‘\( l \)’ is the length of the code, ‘\( p_m \)’ is the probability of mutation and ‘\( p_c \)’ is the probability of crossover. So a schema with a shorter defining length \( \delta(H) \) is less likely to be disrupted.

The above mentioned framework was formalized by John Holland [19] and familiarized by David Goldberg [38] when he was able to solve a difficult problem involving gas-pipeline transmission for his dissertation in 1989. It proved to be a milestone for the development of the genetic algorithm. Also Daniel Goldberg, director of the Illinois Genetic Algorithms Laboratory, expressed that genetic algorithms worked because they could find good building blocks. Building blocks are just schemata with a short defining length which consist of bits that work well together [39]. Research in genetic algorithms remained largely theoretical until the mid-1980s, when the First International Conference on Genetic Algorithms was held at the University of Illinois. As academic interest grew, there was a dramatic increase in desktop computational power that allowed for practical application of this new technique.

During 1989, General Electric and Rensselaer Polytechnic Institute started selling the world’s first genetic algorithm product, a mainframe-based toolkit designed for industrial processes. During the same period, Axcelis, Inc. released Evolver, the world’s second genetic algorithm product and the first for desktop computers. The New York Times technology writer John Markoff wrote about Evolver in 1990. Custom computer applications began to emerge in a wide variety of fields, and these algorithms are now used by a majority of Fortune 500 companies to solve difficult scheduling, data fitting, trend spotting and budgeting problems, and virtually any other type of combinatorial optimization problem [40, 41].


2.2.2 Simple Genetic Algorithm

In nature, the fittest individuals are most likely to survive and mate. Since the next generation is bred from such fit individuals they are more likely to be fitter and healthier (since good traits from parents are passed to children). This same idea is incorporated in the genetic algorithm. Individuals/solutions are traditionally represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution starts with an initial population of individuals, which is usually generated randomly. The fitness of each individual is calculated, which is problem specific. The individuals with better fitness are selected for improvement, using reproduction operators like crossover and mutation. Crossover operator is applied on the selected individuals which generates children. Further these children undergo the mutation operation. The mutated individuals form the population for the next generation. One iteration of the algorithm is referred to as a generation and evolution happens in generations. Individuals in the new population are evaluated and this cycle of selection, reproduction and evaluation is repeated until some termination criteria is met.

In a broader range of the term, the genetic algorithm is a population based model that encodes a solution to a problem as a simple chromosome like data structure and then uses the selection procedure and recombination operators to generate new sample points in a search space. Genetic algorithms have become increasingly popular in solving combinatorial optimization and search problems. They are known to be “general purpose” search methods since, the description of their application and modeling extends beyond the field of computer science to include dynamical systems theory, game theory, molecular biology, ecology, evolutionary biology, machine learning, population genetics etc. [42].

2.2.3 Standard Genetic Algorithm terminologies

Following are few terms that are used interchangeably throughout this thesis:

- **Gene**: or bit is a sub unit of a solution.

- **Individual**: or solution or chromosome is a set of genes.

- **Population**: Group of individuals.

- **Population Size**: Total number of individuals in a population.

- **Subpopulation**: Set of individuals which is isolated from the rest of the population.

- **Crossover Rate**: Probability of a crossover execution.
Chapter 2. Theory and Background

- **Mutation Rate**: Probability of a mutation execution.

- **Fitness Function**: This is a particular type of objective function, which determines the environment within which the individuals survive. This is problem specific.

- **Fitness**: This is an real value which is the output of the fitness function.

- **Parents and Children**: The genetic algorithm selects certain individuals from the current population, called parents, and uses them to create individuals called children to form new population for the next generation.

- **Generation**: An iteration step of the algorithm.

### 2.2.4 Genetic Algorithm Overview

Following steps are followed in a simple genetic algorithm:

(a) **Initialization**: The evolution starts with an initial population of individuals, which is usually generated randomly. Some predefined techniques can also be used to generate the initial population. Number of individuals in a population may vary from hundreds to thousands.

(b) **Evaluation**: The quality of the individuals is measured with an evaluation function, which is problem specific. The evaluation function returns a real value known as fitness.

(c) **Selection**: During each generation, the individuals with better fitness values are selected, which will be improved using reproduction operators, like crossover and mutation. Various implementations of selection procedure are mentioned in Section 2.2.5

(d) **Crossover**: Crossover operator is applied on the selected pair of individuals which generates children. The children typically share many of the characteristics of the parents. Various implementations of crossover operation are mentioned in Section 2.2.5

(e) **Mutation**: Further the children undergo the mutation operation. Unlike crossover, mutation operator is applied on single individual. The mutated individuals form the new population for the next generation. Various implementations of mutation operation are mentioned in Section 2.2.5

(f) **Termination**: The cycle of selection, reproduction and evaluation is repeated until some termination criteria is met. Termination criteria are mentioned in Section 2.2.5. In the above mentioned genetic algorithm procedure, few fit individuals can survive and more probably pass over to the next generation(s) without alteration. This is known as **elitism**.
Flowchart of a simple genetic algorithm is shown in Figure 2.2.

![Flowchart of a standard Genetic Algorithm](image)

**Figure 2.2: Flowchart of a standard Genetic Algorithm**

### 2.2.5 Genetic Algorithm Detailed Steps

There are various ways of representing an individual, initializing the population, selecting competent individuals, reproducing children and terminating the genetic algorithm. Few of them are discussed below:

(a) **Representation:** To apply a genetic algorithm to any problem, one must encode solutions to that problem in a structure that can be stored and processed in the computer. Genetic algorithms work well when applied properly. Following are few ways of encoding solutions to represent an individual in a genetic algorithm:

- **Binary encoding:** Binary encoding is the oldest and one of the most common encoding methods used for representing solutions to many problems. Every individual is represented in binary as strings of bits, 0 or 1. Figure 2.3 shows an example of individuals with binary encoding.

- **Value Encoding:** Here each individual is represented as a string of some value of the variables. Values can be anything related to problem like numbers, characters, real numbers, complicated objects etc. Figure 2.4 shows an example of individuals with value encoding.
Chapter 2. Theory and Background

30

Figure 2.3: Binary Encoding

Figure 2.4: Value Encoding

• Tree Encoding: Tree encoding is used mainly for evolving programs or expressions. For example, when genetic algorithms are used for program designing and construction, each individual may represent some objects, mathematical operations or other components of program (like some functions or commands) in a tree like structure. Figure 2.5 shows an example of individuals with tree encoding.

Figure 2.5: Tree Encoding

• Permutation Encoding: Permutation encoding or order encoding is useful when individual fitness depends on positions of genes in the individual. Here every individual is represented with numbers in a sequence. Permutation encoding is mostly useful for ordering problems. For such problems, some crossover and mutation corrections must be made to leave the individual consistent (i.e. have real sequence in it). Figure 2.6 shows an example of individual with permutation encoding.

Representation of the individuals are problem specific. Few representation may suit few problems and not the others. For example, binary encoding can be used for the protein structure prediction
problem, order encoding for the travelling salesman problem, while value encoding for neural networks problems.

(b) Population Initialization: Initialization involves setting the parameters for the algorithm and creating the initial population of individuals. The most standard way to initialize the population is to randomly generate the individuals. Random initialization of population ensures that the entire solution space is considered. Other alterative initialization methods can be applied by incorporating the domain knowledge of the problem at hand. The solutions may also be “seeded” in areas where optimal solutions are likely to be found.

(c) Selection Procedure: The selection procedure determines how individuals are chosen for mating. The idea is to give preference to better individuals, allowing them to pass on their genes to the next generation. There are different techniques which a genetic algorithm can employ to select the competent individuals. Following are a few common methods of selection:

- Roulette Wheel Selection: Roulette Wheel Selection or Fitness Proportionate Selection is one of the oldest selection methods, introduced by John Holland [19]. It has been applied in the conventional genetic algorithm. In the Monte Carlo roulette wheel selection by De Jong [43], the individuals of a population are imagined to be placed on a roulette wheel as shown in Figure 2.7

Selection is done according to the fitness of an individual. The size of the area on the wheel
for an individual is proportional to its fitness. The wheel is spun, and the individual where it stops is selected for mating. This procedure is repeated until necessary number of individuals have been selected. Selection methods have been studied in the past and found that the roulette wheel selection has a bad performance compared to other methods like tournament selection \cite{44, 45} or ranking selection \cite{44, 46}.

This selection method only works well if the fitness value of an individual is proportional to the probability of being selected.

- **Rank Selection:** Rank Selection is introduced by Baker \cite{47} and is more thoroughly discussed by Whitley \cite{48}, Blickle & Thiele \cite{46, 49}, and Goldberg & Deb \cite{44}. It is basically roulette wheel selection, but with a presort. Each individual in the population is assigned a numerical rank based on its fitness. Selection is done based on this numerical rank. For example, individuals are sorted in order of highest to lowest fitness with the highest being awarded a rank of 1 and lowest say 10 (assuming population size to be 10). The higher the rank, the higher is the fitness and hence, higher the probability of an individual being selected in the sorted list of all individuals in the population. Figure 2.7 also depicts an example of rank selection where the percentages are assigned according to the fitness after sorting the individuals.

- **Tournament Selection:** Tournament selection is proposed by Wetzel \cite{50} and studied by Brindle \cite{51}. It is one of the most popular and effective selection methods for genetic algorithm. Here for each single tournament, a random subset of say ‘\( t \)’ individuals is selected which is known as the tournament size. Then the individual(s) with best fitness amongst these is/are selected. Often tournaments are held only between two individuals (binary tournament) but a generalization is possible to an arbitrary group size ‘\( t \)’. If we assume that the mating pool will contain number of individuals as the population size, each individual will, on an average, participate in two tournaments. Although this is a simple selection strategy, it is very powerful and therefore used in many practical applications \cite{52, 53, 54, 55}. This strategy also works on parallel architectures and allows the selection pressure to be easily adjusted. Figure 2.8 depicts an example of tournament selection.

Selection does not introduce new points in the search space. To introduce new individuals into the population, genetic operators like crossover, which is an example of binary search operation, and mutation, which is an example of unary search operation are used. Usage of any other problem specific reproduction operators is also possible.

**Crossover Operation:** The crossover operator is used to create new individuals (children) into the population by combining the features of two existing individuals (parents). The idea behind the
The applicability of this operator is to exchange information between individuals to get better individuals, in a sense to get genetic material from the previous generation to the subsequent generation. This is one of the important operators which has distinguished genetic algorithm from many other algorithms.

In the context of optimization, *exploitation* is the process of improving and combining the traits of the currently known solutions, as done by the crossover operator in genetic algorithms. It is usually done by manipulating the already tested individuals leading to new, very similar solution candidates known as children. Crossover occurs during evolution according to a user-definable crossover probability (see Section 2.2.6 for more details). Following are the few ways of implementing crossover in genetic algorithm:

- **Single Point crossover:** Here a single crossover point is chosen randomly for the selected individuals (parents). All the genes (bits) after the crossover point are exchanged between these two parent individuals. The resulting individuals are the children. Figure 2.9 shows a typical single point crossover (P: Parent, C:Child).

- **Two Point crossover:** Here two crossover points are chosen randomly for the selected individuals (parents). The genes (bits) between these two crossover points are exchanged between the
Chapter 2. Theory and Background

parent individuals thus forming the children. Figure 2.10 shows a typical two point crossover (P: Parent, C:Child).

Figure 2.10: Two-Point Crossover

- Multi Point crossover: Multi point crossover or $n$-point crossover is a generalized form of two point crossover where multiple crossover points are chosen randomly for the selected individuals (parents). The information between two crossover points are exchanged between the parent individuals thus forming the children. Figure 2.11 shows a typical multi point crossover (P: Parent, C:Child).

Figure 2.11: Multi-Point Crossover

- Uniform crossover: Uniform crossover operator decides (with some probability, known as the mixing ratio), which parent will contribute each of the gene values in the children. Unlike the other crossover operators, this allows the parent individuals to be mixed at the gene level rather than the segment level. If the mixing ratio is 0.5, approximately half of the genes in the children will come from parent 1 and the other half will come from parent 2. Figure 2.12 shows a typical uniform crossover (P: Parent, C:Child).

Figure 2.12: Uniform Crossover

- Arithmetic crossover: Simple arithmetic crossover performs some defined arithmetic operation on parents to generate the children. Here single random crossover point is chosen and the
Chapter 2. Theory and Background

35
genes after the crossover point are manipulated. Figure 2.13 shows an example of arithmetic crossover. (P: Parent, C: Child)

![Simple Arithmetic Crossover](image)

Different encoding schemes may need designing different crossover operators.

(e) **Mutation Operation:** In the standard genetic algorithm, after crossover operation is performed on individuals, mutation takes place. Mutation operator corresponds to a small, random variation in an individual. The idea behind the applicability of this operator is to get some extra variability in the population of individuals. It alters one or more genes in a single individual. This results in an entirely new individual and ensures genetic diversity within the population.

In the context of optimization, *exploration* means finding new points in areas of the search space, which have not been investigated before, as done by the mutation operator in the genetic algorithms. Exploration is a metaphor for the procedure, which allows search operations to find novel and maybe better solution candidates.

Similar to crossover, during evolution, mutation occurs according to a user-definable mutation probability (see Section 2.2.6 for more details). This probability should be set low. If it is set too high, the search will turn into a primitive random search [56]. Following are few ways of implementing mutation in the genetic algorithm:

- **Flip-bit Mutation:** This mutation operator randomly chooses a bit in an individual and flips it (i.e. if the bit is 1, it is changed to 0 and vice versa). This way a new individual is formed. This operator is usually used with binary encoding. Figure 2.14 shows a typical flip-bit mutation.

- **Mutation Swapping:** This mutation operator randomly chooses two points in an individual, and swaps the bits (genes) at those positions, thus creating a new individual. Figure 2.15 shows a typical mutation swapping.
Chapter 2. Theory and Background

Mutation Inversion: This mutation operator randomly chooses two points in an individual so as to form a window. The order of genes within this window is reversed. This way a new individual is formed. Figure 2.16 shows a typical mutation inversion.

Uniform Mutation: This mutation operator replaces the value of a randomly chosen gene, with a uniform random value selected between the user-specified upper and lower bounds for that gene. This way a new individual is formed. This operator can only be used for integer and float genes. Figure 2.17 shows a typical uniform mutation.

Different encoding schemes may need designing of different mutation operators.

(f) Termination: There are certain stopping criteria which determine when the genetic algorithm needs to terminate. When the stopping criteria is met, the genetic algorithm terminates and the results are obtained. Few possible stopping criteria for the termination of genetic algorithm [57, 58, 59]
Chapter 2. Theory and Background

could be computation time, total number of iterations or total evaluations. Alternatively, the genetic algorithm could also be terminated when no improvement in the solution quality is detected or if the algorithm has already found a sufficiently good solution. Two or more combinations of the above-mentioned stopping criteria could also be considered for terminating the genetic algorithm.

(g) **Elitism** One may lose the best individuals from the present generation if the next generation is created only by new offsprings. To avoid this, at least one or few best individual(s) are allowed to pass over to the next generation without alteration. This is known as *elitism*. This makes it possible for the best individuals to compete and survive till the end. Elitism can rapidly increase performance of the genetic algorithm, because it prevents dropping out of the best-found individuals. However, the Elitism needs to be controlled appropriately in order to avoid premature convergence of the genetic algorithm.

### 2.2.6 Parameters of Genetic Algorithm

(a) **Probability of crossover**: Probability of crossover $P_c$ or crossover rate is the parameter that effects the rate at which the crossover operator is applied. It says how often the crossover operation will be performed. If the crossover probability is satisfied crossover is performed on two individuals and offspring are created, otherwise there is no crossover and the offspring is exact copy of parents. 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 the individuals from old population (but this does not mean that the new generation is the same, since mutation operator can change the individual before it becomes a part of new generation) [60]. The importance of $P_c$ in controlling the genetic algorithm performance has long been acknowledged, as its value determines whether the algorithm will find an optimal solution efficiently or not [61].

(b) **Probability of mutation**: Probability of mutation $P_m$ or mutation rate is the parameter which says how often the parts of an individual be mutated. If the mutation probability is satisfied, mutation is performed and part of the individual is changed, otherwise there is no mutation and the offspring after crossover is taken to the new generation without any change. If mutation probability is 100%, whole individual is changed, if it is 0%, nothing is changed [60]. Similar to $P_c$, $P_m$ also controls the performance of genetic algorithm [61].

(c) **Number of Generations**: The number of generations, a positive non-zero integer value, is one of the termination criteria used in the genetic algorithm. The genetic algorithm terminates on reaching the total number of generations after which the results of the evolution process are obtained. Dana
Vrajitoru [62] studied various genetic algorithm parameters to search for optimal balance between the two genetic parameters: the population size and the number of generations. They used the genetic algorithm to improve the performance of information retrieval systems. Also, it has been observed that larger populations have better chance of significantly improving the effectiveness of information retrieval.

(d) **Population Size:** Total number of individuals in a population is termed as Population Size. The population size is a critical parameter in the genetic algorithm. If the population size is too small then the genetic algorithm converges to non-optimal solutions. If the population size is too large, the genetic algorithm may take longer time to converge to the optimal solution leading to unnecessary usage of the computational resources. The population size in the genetic algorithm is related to the problem size and difficulty. It is known from nature that large populations are more stable and resist evolutionary changes more than small populations [63], but there is a large cost for evaluation of a large population.

### 2.3 Travelling Salesman Problem

This section concentrates on the travelling salesman problem. Definition and a brief introduction is provided. This is followed by the history of the travelling salesman problem. Finally few applications of the travelling salesman problem are listed. Travelling salesman problem is a well known optimization problem and can (alternatively) be defined as follows [64]: Given a set of cities and the cost of travel (or travel distance) between each possible pairs, the travelling salesman problem, is to find the most feasible way of visiting all the cities exactly once and returning to the starting point that minimizes the travel cost (or travel distance). For ‘N’ number of cities, complexity of travelling salesman problem is \((N-1)! / 2\). The magnitude of time complexity to find the optimal solution of travelling salesman problem is \(O(N!)\). When ‘N’ is large, the computing time for solving the travelling salesman problem is too costly [65].

There are two major variants (or types) of travelling salesman problem viz. The *symmetric travelling salesman problem* and the *asymmetric travelling salesman problem*. The symmetric travelling salesman problem is the one, where the distance between two cities is the same to and fro. It means the cost of travelling from city 1 to city 2 is same as that of travelling from city 2 to city 1. In the asymmetric travelling salesman problem the distance between the different cities is different, depending on the direction of travel. This means the cost of travelling from city 1 to city 2 may NOT be same as travelling from city 2 to city 1. Another variant of travelling salesman problem is the *probabilistic travelling salesman problem*, where the salesman requires to visit a city only with a given probability.
Chapter 2. Theory and Background

The main difference between probabilistic travelling salesman problem and the other variants of travelling salesman problem is that, in the probabilistic travelling salesman problem the probability of visiting each city is between 0.0 and 1.0, while in the other variants the probability of visiting each city is 1.0 [66]. Travelling salesman problem has become a benchmark problem, since a lot of information and the best known solutions for many travelling salesman problem instances are already available.

2.3.1 Brief history

The travelling salesman problem is a relatively old problem. Mathematical problems related to the travelling salesman problem were documented as early as 1756 by Leonard Euler, whose interest was in solving the knights tour problem. The problem was to find a sequence of knights moves that will take the piece from a starting square on a chessboard, through every other square exactly once and returning to the start. A correct solution would have a knight visit each of the 64 squares of a chessboard exactly once on its tour. Eulers solution is depicted in Figure 2.18, where the order of moves is indicated by the numbers on the squares [3]. In the 1800s, Irish mathematician Sir William Rowan Hamilton and the British mathematician Thomas Penyngton Kirkman also treated mathematical problems related to the travelling salesman problem. Discussion of the early work of Hamilton and Kirkman can be found in the book, Graph Theory [67].

![Euler's solution to Knight's Problem](image)

In the 1930s, mathematician Karl Menger studied the general form of the travelling salesman problem in Vienna and Harvard. He called the problem as the messenger problem (considering that in practice, the problem had to be solved by every postman, and also by many travellers). He defined the problem as: “Finding the shortest path by joining all of the finite set of points, whose pairwise distances are known”. He considered the brute-force algorithm, but it takes a long time to receive a solution. He also observed the non-optimality of the nearest neighbour heuristic when applied to the messenger problem. Detailed study can be found in ref. [68]. During the same period, an American mathematician Merrill Meeks
Chapter 2. Theory and Background

Flood was notable for developing, with Melvin Dresher, the basis of the game Prisoner’s dilemma model of cooperation and conflict [69].

In late 1940s, Flood publicized the name *Travelling Salesman Problem* within the mathematical community at mass by presenting it at the RAND Corporation. The Corporation’s reputation helped travelling salesman problem become a well known and popular problem. According to Flood, during that time he was struggling with the problem in connecting with school-bus routing in New Jersey. Since then, for over 70 years the travelling salesman problem has been studied intensively, which has led the way to improve solution methods in many areas.

2.3.2 Applications related to the Travelling Salesman Problem

Travelling salesman problem has received attention from researchers in various fields all over the world. This is because even though travelling salesman problem is known to be a difficult combinatorial optimization problem, there are important practical problems that can be formulated as travelling salesman problems. There are many problems which are generalizations of travelling salesman problem. Also the travelling salesman problem often comes up as a subproblem in more complex combinatorial problems. Few of such problems are described below:

Dantzig and Ramser [70] proposed the *Truck Dispatching Problem*, which is a generalization of the travelling salesman problem. The truck dispatching problem was concerned with the optimum routing of a fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal. Ratliff and Rosenthal [71] developed an efficient algorithm to find shortest order picking routes, which is considered as a solvable case of the travelling salesman problem. The problem is stated as: given that the order picker has to collect a number of products in specified quantities at known locations, in what sequence should the order picker visit these locations in order to minimize the distance travelled?

Clustering problem can also be solved using travelling salesman problem. This was first suggested by Lenstra [72]. The idea is that, from one cluster to the next, larger jumps are necessary and thus the objects in the same cluster are visited in consecutive order. Clusters can later be separated [73, 74]. Further, Climer and Zhang [75] suggested that for clustering with ‘k’ clusters, add ‘k’ dummy cities, whose distance to each of the other cities is equal to a constant ‘c’. After solving this new travelling salesman problem, the dummy cities can separate the clusters.

Jacques Bertin introduced permutation matrices to analyze multivariate data with medium to low sample size [76]. The idea was to make a data matrix more understandable by simultaneously rearranging
Chapter 2. Theory and Background

rows (variables) and columns (cases) and then grouping them. They find (a near) optimal permutation by solving two travelling salesman problems in order to solve the above problem. They used the sum of rank differences as distances. Boland et al. [77] used the travelling salesman problem to solve the aircraft fleeting and routing problem which can be formulated as: given a schedule of flight legs to be flown by an airline, the fleet assignment problem is to determine the minimum cost assignment of flights to aircraft types, called fleets, such that each scheduled flight is assigned to exactly one fleet, and the resulting assignment is feasible to fly given a limited number of aircraft in each fleet.

The job-shop problem is a combinatorial optimization problem, which is also a generalization of the travelling salesman problem. A number of jobs have to be done and every job consists of using a number of machines for a certain amount of time. The problem is to find the best planning to do all the jobs on all the different machines in the shortest period of time [78]. Further Farouk Yalaqui and Chengbin Chu [79] proposed an efficient heuristic approach to solve the parallel machine scheduling problem with job splitting and sequence-dependent setup times. It aims to minimize the timespan. They first reduce the problem into a single machine scheduling problem with sequence-dependent setup times. This reduced problem is transformed into a travelling salesman problem to obtain a feasible initial solution. This initial solution is then improved in a step by step manner, taking into account the setup times and job splitting.

DNA sequencing [10] is the process of reading the nucleotide bases in a DNA molecule. It can be done by any method or technology that is used to determine the order of the four bases. Travelling salesman problem appears to be a subproblem of DNA sequencing. Here the concept of city represents customers, soldering points or DNA fragments, and the concept of distance represents travelling time or cost, or a similarity measure between DNA fragments.

Robert and David [80] performed experiments in X-ray crystallography and presented a preliminary report on computation for the same. Large travelling salesman problems were detected by determining efficient sequences of X-ray diffraction measurements on a four-circle diffractometer. Optimizing the travelling salesman problem heuristics gave substantial improvements in utilization at small computational expense. Jiefeng et al. [81] reduced the Digital Data Service problem to the travelling salesman problem in order solve it efficiently. The network design problems in the telecommunications industry is to interconnect a set of customer locations through a ring of end offices so as to minimize the total tariff cost and provide reliability. Digital data service is a high-quality digital transport service in the telecommunications industry using permanent network connections and dedicated transmission facilities.

These applications related to travelling salesman problem substantiates that an efficient, optimal algorithm for travelling salesman problem would yield an efficient, optimal algorithms for thousands of computational problems of practical importance. Therefore finding algorithms that lower the costs of
travelling salesman problem while providing reasonably good solutions is of value.

2.4 High Performance Computing

Faster and better solutions to large, complex problems are often limited by the physical limits of sequential computing architectures. In modern computing, parallel processing, which is a method of having many small tasks to solve one large problem has emerged as a key enabling technology. For the past many years, several initiatives has been carried out to adopt parallel computing in almost every field of science and engineering as well as in general purpose applications. High-performance computing (HPC) is the use of parallel processing for running advanced applications efficiently, reliably and quickly. In order to gain good performance using parallel computers (like multi-core processors, clusters or grids), one needs to write a parallel code that automatically splits-up a given problem into several tasks that can be executed concurrently on various processors. The parallel code should enable the processors to communicate with each other while executing these tasks.

High performance computing may require more memory bandwidth, better integer computing performance, or a high performance I/O system to achieve high levels of performance [82]. Parallel computer programs are more difficult to write than their corresponding sequential versions [83], because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different tasks of a problem are typically some of the greatest obstacles that may hinder achievement of good parallel programming performance. However, new technologies may be difficult to acclimatize at beginning, but invariably become easier to adapt over time. Access to multi-core processors has become easier and cheaper, and programming such parallel processors has become central to the programming enterprise. Hence, parallelism today has become ubiquitous.

2.4.1 Parallel Computer Architectures

In 1966 Michael J. Flynn [84, 85, 86] proposed a classification of parallel computer architecture known as Flynn’s taxonomy, which has remained the focal point in this field. The classification is based on the number of concurrent instruction(s) and number of data stream(s) executed on parallel machines per clock cycle. There are four categories, which are described below:
(a) **Single Instruction Single Data (SISD):** In SISD machines, a processor executes a single instruction on single data stored in the memory in one clock cycle. It does not support parallelism in either the instruction or data level. Example of an SISD architecture is the traditional uniprocessor machine (e.g., PC).

(b) **Single Instruction Multiple Data (SIMD):** In SIMD machine, a processor exploits multiple data streams against a single instruction stream to perform operations. It has a single control unit which fetches an instruction from memory and after decoding, broadcasts control signals to multiple processing element available. SIMD machines are also called array / vector processor, since they are employed for processing complex operations on large arrays or vectors. Examples of SIMD architecture are the array processors or graphics processing units (GPUs).

(c) **Multiple Instruction Single Data (MISD):** In MISD machine, multiple instructions operate on a single data stream. The same stream of data flows through a linear array of processors executing different instruction streams. This is an uncommon architecture which is generally used for fault tolerance. This concept is useful in computations where the same input is to be subjected to several different computations. Some computational scientists consider the pipeline processing architecture as MISD.

(d) **Multiple Instruction Multiple Data (MIMD):** In MIMD machine, multiple autonomous processors have their own control unit and simultaneously execute different instructions on different data. Distributed systems are generally recognized as MIMD architectures, where they either exploit a single shared memory space or a distributed memory space. They are true multiprocessor systems which exploit asynchronous parallelism. Examples of MIMD architectures are the multi-core superscalar processors, platforms which include the Sun Ultra Servers, multiprocessor PCs, workstations, clusters, etc.

When all the processors in MIMD architecture execute the same program, it is known as Single Program Multiple Data (SPMD) computation. The SPMD model is widely used by many parallel platforms and requires minimal architectural support. When all the processors in MIMD architecture simultaneously operating at least two independent programs, it is known as Multiple Program Multiple Data (MPMD).
2.4.2 Memory Architectures

Main memory in parallel system architectures are classified on the basis of how processors communicate with each other. Broadly there are two categories: Shared memory architecture and Distributed memory architecture. In the shared memory architecture, single address space is shared by all processors and communication between the tasks is done through read and write operations thus supporting implicit communication and explicit synchronization. Shared memory machines can be divided into two main classes based upon memory access times:

- **Uniform Memory Access (UMA)**, where each element of memory is assessed with same latency and bandwidth.

- **Non-Uniform Memory Access (NUMA)**, where elements of memory do not have same access time to all memory locations.

Shared memory architectures are usually connected by a 'bus', hence, they are not scalable with respect to addition of larger number of processors and/or parallel performance. In distributed memory architectures, each processor has its own local memory and address space. Each processor can only access its local memory, and communication among these processes is performed by passing messages through some communication network. This architecture supports explicit data distribution, communication and synchronization. Also, careful address mapping is done by the Operating system, which reduces the need for accessing memory modules over the ‘bus’, and hence, distributed computers are highly scalable. We could also have hybrid memory architecture, which employs both shared and distributed memory formalisms. Here processing elements have their own local memory and also have access to memory on non-local processors.

There are many ways to express parallelism in a program using parallel programming models. Various Applications Programming Interfaces (APIs), libraries, and parallel programming languages have been created for developing programs and algorithms suitable for above mentioned systems and memory architectures. Shared memory programming languages enable communication by manipulating shared memory variables. Distributed memory paradigms enable message passing between different processors. For shared memory programming, POSIX Threads and OpenMP are two programming paradigms. OpenMP, an extension to a compiler, uses threads, but the details are hidden from the programmer.

To use OpenMP, the programmer adds “pragmas” to the program, which get substituted by appropriate code by the compiler to facilitate creation and execution of threads on various cores or processors of the parallel machine. The resulting program actually uses operating system threads to run in parallel
Chapter 2. Theory and Background

[87, 88, 89]. Though both coarse-grained and fine-grained parallelism are possible with OpenMP, it is difficult to debug synchronization calls and race conditions [90, 91]. Also the scalability is limited by memory architecture suited for openMP.

Other way to express parallelism is with parallel virtual machine, which is a distributed environment and message passing system. Parallel virtual machine is a software tool for networking of computers. The parallel virtual machine software must be installed on every machine to facilitate parallel programming [92]. Amongst various message-passing paradigms, the Message Passing Interface (MPI) is the most widely used standard. Most programmers use MPI-based libraries to develop and execute distributed memory programs and also to send messages between multiple processes.

Genetic algorithms are mostly implemented over multi-core architectures using MPI, OpenMP or a combination of MPI with openMP. MPI programs perform better on the distributed memory architectures, whereas OpenMP programs perform well on the shared memory architectures. This research work in parallel genetic algorithm implementation has been carried out on hybrid architectures. Unlike OpenMP, the distributed nature of MPI allows one to work on almost in any parallel environment, hence, we have considered mpiJava for development of parallel genetic algorithms.

2.4.3 MPI

MPI is a specification created and maintained by the Message Passing Interface Forum (MPIF) committee. Various implementations of MPI are available on a wide variety of computing platforms, the more popular amongst these is MPICH. The “CH” in MPICH stands for “Chameleon”, a symbol of adaptability and portability to ones environment [93]. MPI parallelized programs are portable (because MPI has been implemented for almost every distributed memory architecture). MPI’s goals are high performance, scalability, and portability. MPI remains a dominant model for high-performance computing today [94]. Discussions on the design of MPI and its implementations can be found in [95, 96, 97, 98, 99].

2.4.4 mpiJava

MPI is the standard interface for message-passing libraries. There have been several implementations of Java message-passing libraries [100]. The two main proposed APIs are the mpiJava 1.2 API [101], which tries to adhere to the MPI C++ interface defined in the MPI standard version 2.0, and the Java Grande Forum MPI (Message-Passing interface for Java) API [102], which has been proposed by Java Grande Forum [103] to standardize the MPI-like Java API. The main differences between these two APIs are in
Chapter 2. Theory and Background

the naming conventions of variables and methods.

mpiJava is an object-oriented Java interface to the standard MPI. It is developed within HPJava project [104] and maintained by LarKC project [105]. Recently a new version of mpiJava has been released, which is managed and maintained by the High Performance Computing Center Stuttgart in the framework of LarKC [106]. It is a set of Java Native Interface (JNI) wrappers for the native MPI API. It allows Java applications to efficiently run on a distributed, parallel, and high performance architecture. It was suggested by Carpenter et. al. in late 1997. The mpiJava codes can run on distributed memory multi-processor machines, shared memory multiprocessor machines, a cluster of workstations, or on combination of all of the above. A virtual topology describes a mapping or ordering of MPI processes into a geometric “shape”. MPI topologies are virtual - there may be no relation between the physical structure of the parallel machine and the process topology. They must be developed by the programmer. MPI has explicit functions to create cartesian topologies.

2.5 Parallel Genetic Algorithm

Due to increasing demands placed on the genetic algorithms, such as searching large problem spaces with costly evaluation functions and large population sizes, there is a need to achieve high-quality solutions in reasonable amount of time. As the complexity of the applications increases, genetic algorithms exhibit high computational cost and degradation in the quality of the solution(s). Efforts to address these shortcomings have been made in the form of several research initiatives. The parallel genetic algorithms formalism is one of the most significant [107] such effort in this direction.

Parallel Genetic Algorithm Models Parallel genetic algorithm is an mechanism that combines the inherent parallelism of genetic algorithms with the concurrency supported in parallel computers. It enhances the solution speed of the population. Parallel genetic algorithm are broadly divided into four types, namely global parallel genetic algorithm, coarse grained parallel genetic algorithm, fine-grained parallel genetic algorithm and mixed parallel genetic algorithm.

2.5.1 Global Parallel Genetic Algorithm

or Master Slave model consists of a single population of individuals similar to the sequential genetic algorithm. The master processor generates the initial population and does the selection and reproduction (hence known as “global” model), and distributes the individuals among the slave processors for evalu-
ation. The slave processors evaluate the individuals and send the fit individuals back to the master. The communication therefore occurs after every generation. The evaluated individuals when received by the master can be inserted randomly, or replace the worst individuals or compete against other individuals.

Cantu-Paz [108] has presented an analysis of the execution time of global parallel genetic algorithms that includes a simple model of the time used in communications. As more slaves are considered, the communications cost increases, however the computational cost for each slave decreases. He shows that there is an optimal number of processors that minimizes the execution time. To further reduce the execution time he recommends the use of hybrids that combine global and coarse-grained parallel genetic algorithms.

Computation/communication ratio [109] is the ratio of the number of calculations a process does to the total size of the messages it sends. This ratio is used to quantify the relation between communication time and computation time in multiprocessor systems. The ratio may also be formulated as the ratio of the time spent calculating to the time spent in communicating, in which case the ratio’s value depends on the relative speeds of the processor and communications medium, and on the startup cost and latency of communication. If computation/communication ratio is high we call the parallel genetic algorithm a coarse grain parallel genetic algorithm. If the ratio is low we call it a fine-grain parallel genetic algorithm.

2.5.2 Island Model Parallel Genetic Algorithm

Also called as coarse-grain parallel genetic algorithm or distributional model, are based on independent genetic algorithms, which evolve separately on each processor (island), and after every few generations exchange genetic material. Here, the total population is divided into multiple sub-populations known as demes and each processor will have its own subpopulation for evolution. Each processor will execute genetic algorithm on its sub-population and occasionally migrate few individuals to other processors to ensure population diversity. There are different migration strategies such as to select emigrants and replace them randomly, or alternatively, based on fitness values. Typically, the best individual(s) will be selected for migration (and it is also retained), and the worst individual(s) will be replaced by an immigrant.

The migration of individuals from one island to another is controlled by a topology that defines the connectivity between the sub-populations, by a migrate rate, which controls the number of individuals to migrate, and by a migration interval that affects the frequency of the migrations. Selection, reproduction and evaluation occur within the sub-population. Use of Islands is motivated from nature where, isolated
populations may develop specialism depending on the local environment. More details related to island model are described by Zbigniew Skolicki [110].

2.5.3 Fine-Grained Parallel Genetic Algorithm

Also called as cellular model or neighbourhood model [111] has only one or very few individual(s) in the sub-population for each processor. These individual(s) can communicate only with neighbouring individual(s), depending on the spatial structure. Any individual can only mate with individuals located on the neighbouring processing nodes. The selection is a local phenomenon rather than a global, in contrast to simple and distributed genetic algorithms. This fine-grained system consists of a large number of processing nodes that evolve a large number of small, overlapping sub-populations, hence, it is well suited for massively parallel computers.

Manderick and Spiessens [112] implemented a fine-grained parallel genetic algorithm with the population distributed on a 2-D grid. Selection and mating were only possible within a neighborhood, and the authors observed that the performance of the algorithm degraded as the size of the neighborhood increased. Turton et al. [113] applied fine-grained parallel genetic algorithm to an image processing application. Algorithms for real-time image analysis are generally simulated on conventional computers or are designed for expensive hardware systems. The author used a simulation of the hardware architecture to verify its effectiveness. The algorithm was tested on three separate images. Experimental results showed that a perfect fit was found when the transformation matched all 4096 pixels of an image in minimum amount of time.

Robertson [114] parallelized the genetic algorithm of a classifier system on a Connection Machine. Two general-purpose classifier systems have been implemented and validated against each other using a letter sequence prediction task. He parallelized the selection of parents, the selection of classifiers to replace, mating, and crossover. The execution time of his implementation was independent of the number of classifiers. Shapiro and Naveta [115] presented a new method for predicting RNA secondary structure based on a fine-grained parallel genetic algorithm. The genetic algorithm is used to explore a very large search space of RNA secondary structure to find for optimal solutions. Comparison to a dynamic programming algorithm shows that their method performs better.

For people who would want to benefit from parallel implementations of genetic algorithm, but may not have access to parallel computers or supercomputers, Keith Vertanen [116] developed parallel virtual machine based island model software. The parallel island model genetic algorithms proved to be a very good candidate for use with a parallel virtual machine network. He showed that their software attains
sub-optimal solutions but linear speedup when compared to single population serial genetic algorithm.

2.6 Summary

This chapter gave the background and literature review of the overall research area. It begins with a brief discussion of the main components of evolutionary algorithms, explaining their role as an optimization techniques. Among the evolutionary algorithms, the genetic algorithms, which are considered for the present study are described in detail. History and overview of genetic algorithms is given which provides a base for this research work. Various selection procedures and recombination operators of the genetic algorithm are explained. Standard genetic algorithms and various terminologies related to genetic algorithms are addressed. Benefits and usage of genetic algorithms as a parallel model are outlined. The travelling salesman problem has been elaborated and its historical overview is provided. Few applications of this problem are also listed. Further the basics of high performance computing is provided along with the brief information of the system and memory architectures. Finally, few parallel models for genetic algorithms are mentioned.