CHAPTER 3: MATHEMATICAL MODEL

3.1 Introduction

Genetic algorithms (GA) are numerical optimisation processes encouraged by both natural selection and natural genetics. The technique is capable of being functional to an extremely varied range of problems. Several classes of optimisation models including linear programming, non-linear programming, dynamic programming, enumeration technique are discussed in literature review. Although every method has benefits and weaknesses, an effective technique for finding global optimal solution for large networks is still required. GA based mathematical model can be selected by considering its ability to find the global optimal solution in the search space also referred to as the feasible region, satisfying the set of (equality or inequality) constraints.

In this thesis, Genetic Algorithm (GA) has been proposed as a practical means of finding global optimisation of pipe network design for carrying water and slurry. GA has been extensively studied, tested and applied in various fields in engineering world. Not only genetic algorithm deliver an alternative technique to answering problem, it consistently beats other traditional approaches in most of the problems. Many of the real world problems which involve finding optimal parameters might prove hard for traditional techniques but are perfect for genetic algorithms.

The genetic algorithm (GA) is an optimisation and search technique based on the principles of genetics and natural selection. A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the "fitness" (i.e., minimizes the cost function). The method was proposed by John Holland (1975) and finally popularized by one of his students, David Goldberg, who was competent to solve a difficult problem connecting the control of gas-pipeline transmission for his dissertation, Goldberg (1989). He was the first to advance a theoretical basis for GA through his schema theorem. The work of De Jong (1975) presented the practicality of the GA for function optimisation and prepared the first intensive effort to find optimized GA parameters. Some of the advantages of a GA includes

- **a.** It is capable of solving problem with a large number of variables.
- **b.** Variables may be continuous or discrete.
- **c.** GA work with a coding of the parameter set, not with the parameters themselves and doesn't require derivative information.
- **d.** Doesn't require derivative information.
- **e.** GA search from a population of points, not from a single point.
- **f.** Delivers a list of optimum variables, not just a single solution.
- **g.** It can solve analytical functions, numerically generated data or experimental data.

These advantages are fascinating and produce fabulous results when traditional optimisation approaches fail.

3.2 Key Elements

Genetic Algorithms are based on philosophies of natural selection and natural genetics. There are more individuals (solutions) born than can stay alive, so there is a continuous struggle for life. Individuals with an advantage have a greater chance for survive i.e., the survival of the fittest. GA works on survival on fittest theorem.

Like any other optimisation algorithm, the GA begins, by describing the optimisation variables and the objective function. It ends by testing for convergence. In genetic algorithms, the word chromosome typically refers to an arbitrarily selected solution to a problem, frequently encoded as a bit string. An allele in a bit string is either 0 or 1. The term "search space" mentions to all possible solutions to the problem. The idea of searching among a collection of candidate solutions for an anticipated solution from a "search space." A population of individual is sustained within search space for a GA, each signifying a probable solution to a particular problem. All individuals are coded as a limited length vector of components, or variables, in terms of the binary alphabet {0, 1}. To continue the genetic analogy these individuals are associated to chromosomes and the variables are similar to genes. Thus a chromosome (solution) is composed of several variables. Each chromosome has one binary string. Each bit in this string can represent some characteristic of the solution. The various parameters used in genetic algorithms are shown in following table 3.1.

Table 3.1 Various Parameters Used in Genetic Algorithm (GA)

	Parameters Used in GA
Genetic Algorithm	Crossover point
	Mutation Rate
	Population size
	Number of iteration
	Percentage of crossover

3.2.1 Search Space

The set of solutions among which the desired solution resides is called search space. All points in the search space represents one possible solution. Therefore each possible solution can be “marked” by its fitness value, depending on the problem definition. GA are used for the best solution among a number of possible solutions represented by one point in the search space

3.2.2 Individuals or Chromosome or String

An individual is a single solution of the problem in search space. The individual solutions are termed chromosomes. These chromosomes are fixed length (figure-3.1).

3.2.3 Genes

A chromosome is an arrangement of genes. Genes may describe a probable solution to a problem, without actually being the solution. A gene is a bit string of arbitrary lengths.

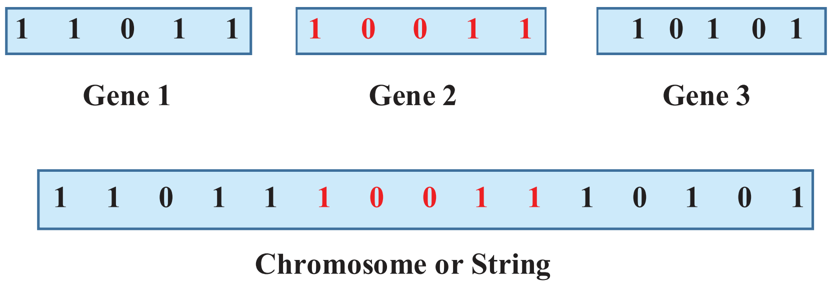


Figure-3.1 Chromosomes in GA

3.2.4 Fitness

The fitness of an individual in a genetic algorithm is the assessment of an objective function. The chromosome has to be first decoded for calculating fitness, and then objective function has to be evaluated. The fitness not only specifies how good the solution is, but also relates to how close the chromosome is to the optimal one.

3.2.5 Populations

A population is an assembly of individuals. For all problem, the population size will depend on the complication of the problem. The size of the population also increases few problems. The larger the population is, the easier it is to explore the search space but it has established that the time required by a GA to converge is high. Goldberg has also shown that GA competence to reach global optimum instead of local ones is largely determined by the size of the population. A higher population is quite beneficial but it needs much more computational cost, memory and time. Basically, a population size of around 100 individuals is quite frequent, but this size can be altered according to the time and the memory available on the machine compared to the excellence of the result to be reached.

Population	Chromosome 1	1 0 1 1 0 1 0 0 0 1 0
	Chromosome 2	1 0 1 1 1 0 1 0 0 0 1
	Chromosome 3	0 0 1 1 1 1 0 1 0 1 0
	Chromosome 4	1 0 0 1 1 0 1 0 0 0 1

Figure-3.2 Population in GA

3.2.6 Fitness Function

The fitness function should be a measure of how closely the model prediction matches the observed or expected data for a given set of model parameters. It must be devised for each problem to be solved. For numerous problems, mainly function optimisation, the fitness function should simply measure the value of the function. For a particular chromosome, the fitness function provides a single numerical fitness which is supposed to be proportional to the utility of the individual which that chromosome represents. GA is an optimisation tool, so generally fitness function is a max/min value function consisting of all the variables.

3.2.7 Coding

A suitable coding for the problem must be developed before running GA programme. It is anticipated that a potential solution to a problem may be signified as a set of parameters. These parameters (known as genes) are combined together to form a string (often referred to as a chromosome). For example, if our problem is to maximize a function of three variables, F (a; b; c), we might signify each variable by a 6-bit binary number. Our chromosome would therefore contain three genes, and consist of 18 binary digits.

3.2.8 GA Operators

The simplest form of genetic algorithm includes three types of operators: selection, crossover, and mutation.

3.2.8.1 Selection

This operator chooses chromosomes in the population for reproduction. The fitter the chromosome, the more times it is possible to be selected to reproduce.

3.2.8.1.1 Roulette Wheel Selection

Roulette wheel selection is a very easy form of proportional selection where an individual candidate result's reproductive probability is proportional to its own fitness. In Figure- 3.1, four solutions form the group of possible parents are shown with a related probability of being selected, these being say 0.6, 0.18, 0.1, and 0.12.

The bigger the percentage of the roulette wheel, the higher area it occupies in roulette wheel and hence the greater the chance of selection. Theoretically, whenever a parent is required, the roulette wheel is rotated. A parent is then selected when the pointer pause inside the area denoted by that parent.

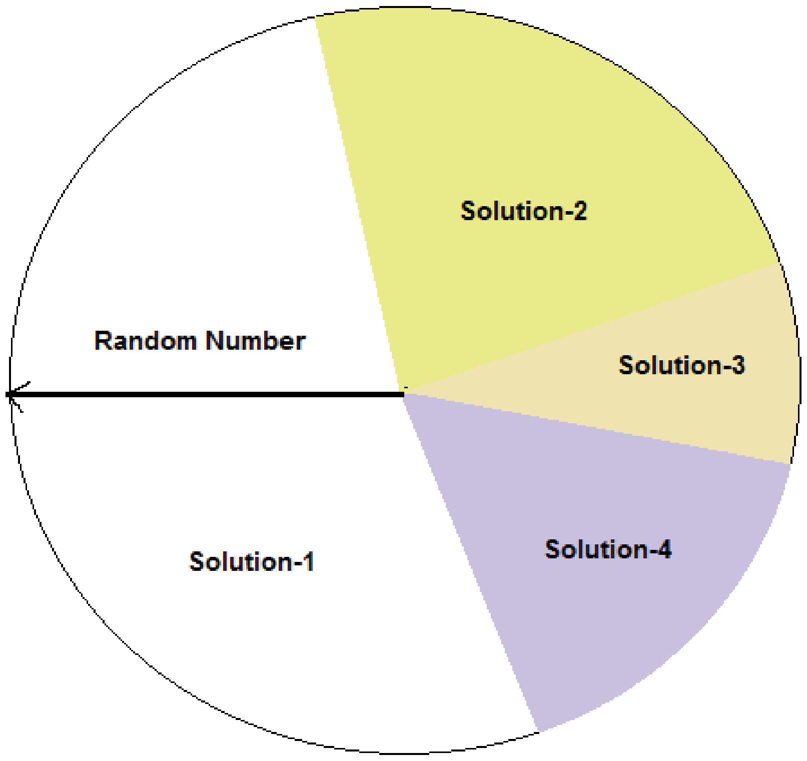


Figure-3.3 Roulette Wheel Selection

There are one possible situation arises within this form of selection. If an applicant parent's fitness is extraordinarily better than any of the rest of the parents in the pool of solutions then this single solution will reside in a very large area in the roulette wheel. Thus this one parent quickly dictates the offspring of the next generation. It is a high chance that the population will converge on only this region of the search space.

3.2.8.1.2 Tournament Selection

Tournament selection Miller et al. (1995), is motivated by the competitive mating activities found in nature, where a competition show the way the probability of mating for a number of potential mates.

Tournament selection works by arranging a tournament among n competitors, n being the tournament size. The individual with the maximum fitness of the n tournament competitors is termed the winner and this winner is then injected into a “mating pool”. The mating pool has a greater average fitness than the average population fitness. Tournament selection carry on until this mating pool become identical in size to the population. The individuals in the mating pool undergo further genetic operators such as reproduction operators (crossover and mutation) to complete the new generation.

3.2.8.2 Crossover

This operator arbitrarily chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to produce two offspring. For example, the strings 10000101 and 11111111 could be crossed over after the third locus in each to produce the two offspring 10011111 and 11100101. The crossover operator roughly imitates biological recombination between two single chromosome organisms. There are different means to make crossover, for example one can select more crossover points. Crossover can be rather complex and rest on encoding of the chromosome. Specific crossover prepared for a specific problem can increase credit of the genetic algorithm (Figure-3.2).

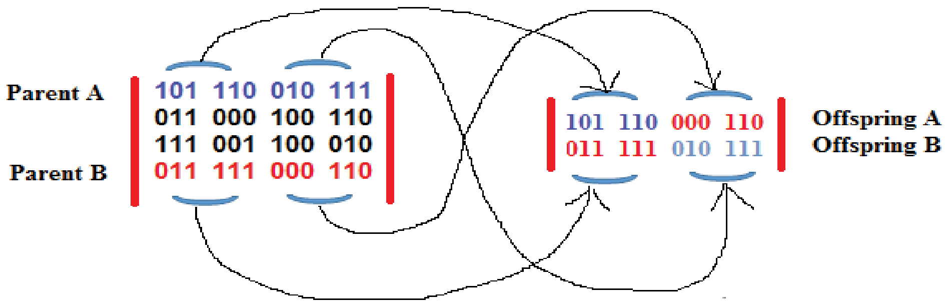
3.2.8.2.1 Single Point Crossover

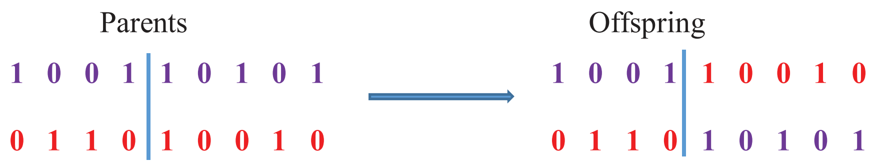
In single point crossover, the two mating chromosomes are cut once at corresponding points and the sections after the cuts exchanged. Here, a cross-site or crossover point is selected arbitrarily along the length of the mated strings and bits next to the cross-sites are exchanged (Figure-3.4(a)).

3.2.8.2.2 Two Point Crossover

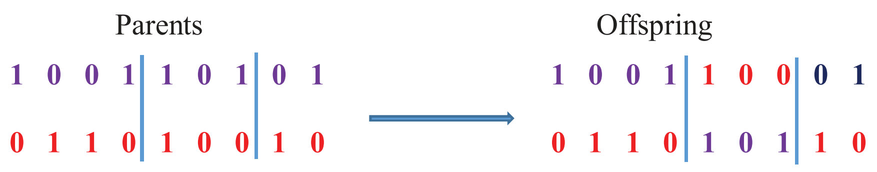
In two-point crossover, two crossover points are selected randomly and the substances (bits) between these points are exchanged between two mated parents.

Apart from single point crossover, several different crossover point algorithms have been formulated, it should be noted that adding more crossover points diminishes the performance of the GA. However, the benefit of having more crossover points is that the problem space may be investigated more comprehensively.





(a) One-Point Crossover



(b) Two-Point Crossover

Figure 3.4. An Example of Different Crossover Schemes

3.2.8.3 Mutation

Mutation involves randomly flips some of the bits in a chromosome in a chromosome. For example, the string 10000100 might be mutated in its second position to yield 11000100. Mutation can happen at each bit position in a string with some probability, usually very small (e.g., 0.001).

3.2.9 Penalty Functions

The penalty functions is the most common method of handling constrained optimisation problems in which constrained optimisation problem is transformed into an unconstrained one by subtracting (or adding) a certain value from the objective function depend upon the amount of constraint violation present in a particular solution. Two main kinds of penalty functions have been considered in classical optimisation: exterior and interior, though hybrid methods have also been suggested by Haftka and Starnes (1976).

For interior penalty functions, the penalty term is selected such that its value is small at points far away from the constraint boundaries and tends to infinity as the boundaries are come close to. The main drawback of interior penalties is the necessity for initial feasible solutions. Finding a feasible solution is very hard in many applications.

For exterior penalty functions, an initial feasible solution is not essential, as the penalty term is zero for feasible solutions and increases as the constraint violations increase. Exterior penalty approaches can be further divided into static and dynamic penalty functions. Static penalty function provides the same value of the penalty term for a given solution over the whole selection process, whereas the dynamic penalty function provides an increase in the penalty term, during the evolution progress. Some researchers have also considered adaptive penalty factors in which the magnitude of the penalty term is dynamically modified (Coello 2002a). The meaning of a good penalty function is very difficult and is problem reliant. Preferably, the penalty should be kept as low as possible, just above the limit, below which infeasible solutions are optimal, according to the minimum penalty rule by Davis (1987). A high penalty may drive the GA inside the feasible region very fast, and this could be a serious disadvantage in the case where the optimum lies at the boundary of the feasible solution. Conversely, if the penalty is too low, then lot of search time is spent exploring the infeasible region because the penalty becomes too small compared to the objective function by Smith and Coit (1997). Therefore, finding an appropriate value for the penalty is a difficult task, which is very often problem dependent. Numerous penalty function methods require the users to define suitable penalty parameter values.

3.3 Outline of the Basic Genetic Algorithm

1. **[Start]** Produce random population of N chromosomes (suitable solutions for the problem)
2. **[Fitness]** Assess the fitness f(x) of each chromosome x in the population
3. **[New population]** Produce a new population by reiterating following steps until the new population is complete

3.1 [Selection] Chose two parent chromosomes from a population on the basis of their fitness (the better fitness, the bigger chance to be selected).

3.2 [Crossover] with a crossover possibility cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.

3.3 [Mutation] with a mutation probability mutate new offspring at each locus (position in chromosome).

3.4 [Accepting] Place new offspring in a new population.

4. [Replace] Practice new generated population for a further run of algorithm.

5. [Test] if the termination condition (for example number of populations or improvement of the best solution) is fulfilled, stop, and return the best solution in current population.

6. [Loop] Go to step 2

The flow chart of the Genetic algorithm is shown in figure-3.5

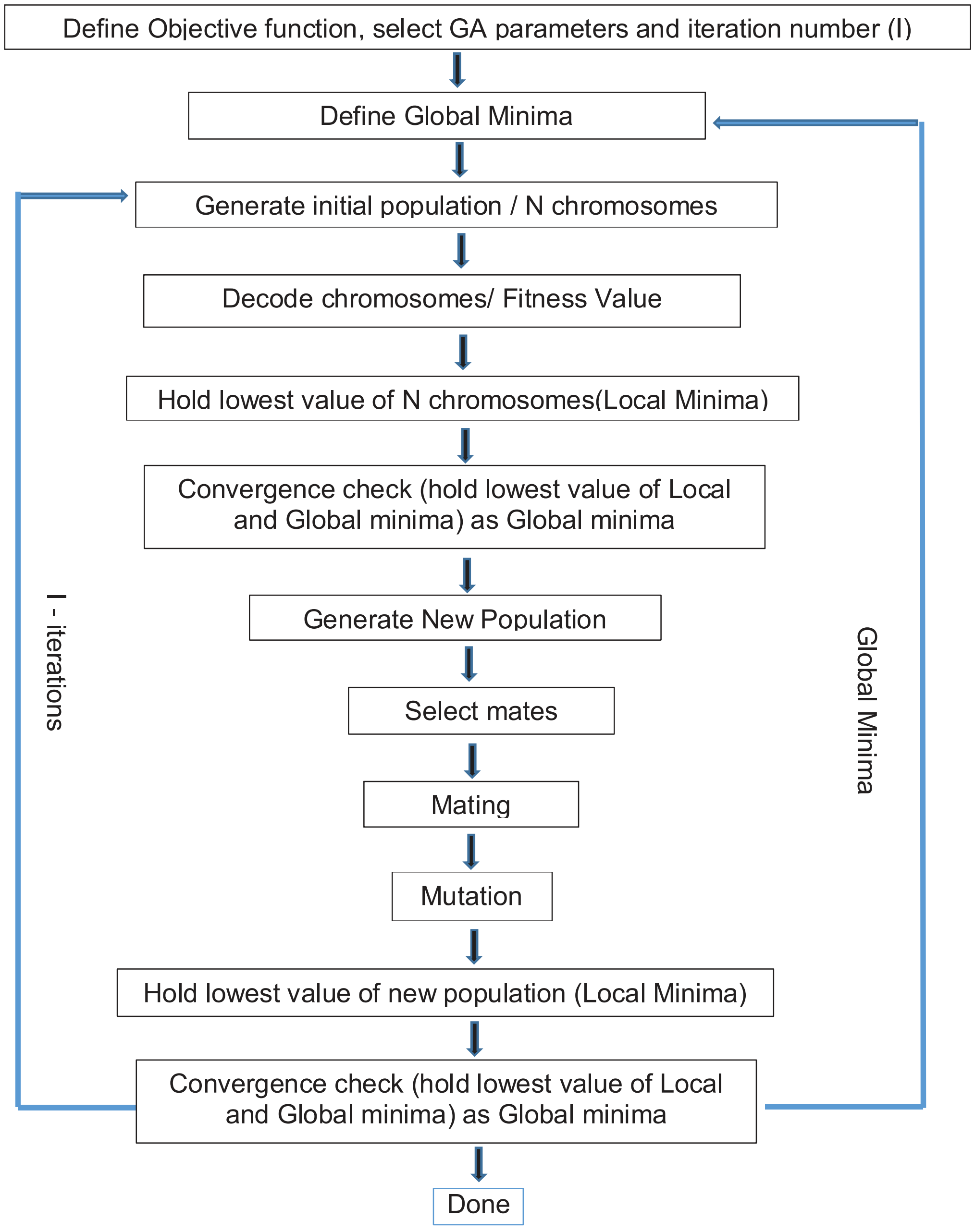

Define Objective function, select GA parameters and iteration number (I)
Define Global Minima
Generate initial population / N chromosomes
Decode chromosomes/ Fitness Value
Hold lowest value of N chromosomes(Local Minima)
Convergence check (hold lowest value of Local and Global minima) as Global minima
Generate New Population
Select mates
Mating
Mutation
Hold lowest value of new population (Local Minima)
Convergence check (hold lowest value of Local and Global minima) as Global minima
Done
I - iterations
Global Minima

Figure-3.5 Flowchart of a Binary GA

3.4 Implementation of GA on Pipe Network

The following steps briefly described the execution of a genetic algorithm to optimize a pipe network of Simpson and Goldberg (1994), Simpson et al. (1993), Simpson et al. (1994).

1. Generate the initial population arbitrarily. The initial population of solutions are created using a random number generator. Each bit location in the string takes on a value of either 1 or 0. Each string represents a different arrangement of a pipe network.

2. Decode every string to the corresponding decision variables.

3. The capital cost of each network component in the generation is calculated. The GA calculates the total cost, including construction, maintenance and operation costs. This step governs the costs of network in the initial population.

4. The network in the population is analysed for heads and discharges under the specified demand(s). The actual heads are compared with the minimum (or maximum) permissible pressure heads.

5. Penalty cost of network is computed. The optimum solution often lies on the boundary between feasible and infeasible solutions by Richardson et al. (1989). Customarily, the penalty multiplier is specified an absolute penalty to each infeasible network connection equal to the maximum value allowed.

6. The total cost of pipe network is computed. The total cost of network is the sum of the network cost (3) and the penalty cost (5).

7. Compute the fitness. For each network in the population, the fitness is taken to be minimum value of the total cost in (6), for example,

8. A new population of network, for the next generation is generated using genetic algorithm operators including: - selection, crossover, mutation, as stated in section 3.5.

9. The steps (2) to (8) is repeated to produce successive generations.

3.5 Suitability

Genetic Algorithms are easy to apply to a wide range of problems. All the operators of GA are important in getting solution of the problem. GA have been shown to be capable to outperform conventional optimisation techniques on complex, discontinuous, multimodal functions. These features are typical of market data, so this technique appears well suited for our objective of market modelling and asset allocation. Its effectiveness of solving problems has made it more favourite choice among the traditional methods, namely gradient search, random search and others. GA are very supportive when the developer does not have precise domain proficiency, because GA possess the ability to explore and learn from their domain.