**Chapter 4**

**Methodology**

Optimization Literature contains a large number of algorithms, each suitable to solve a particular type of problem. Optimization Algorithms are classified into a number of groups single variable v/s multi variable optimization, unconstrained v/s constrained optimization, multi objective optimization, Multi modal optimization, nested optimization, stochastic/ dynamic optimization etc. Accordingly several methods have been proposed depending upon the type of optimization. These methods can be broadly classified as calculus based and search based methods. While Lagrange Multiplier and KKT Conditions are calculus based methods, Search based methods include Linear Programming, Genetic Algorithm, Simulated Annealing etc.

# General Algebraic Modeling System (GAMS)

GAMS is a high-level model specially designed for modeling linear, nonlinear and mixed integer optimization problems. GAMS can easily handle large and complex problems. It is especially useful for handling large complex problems, which may require much revision to establish an accurate model. Models can be developed, solved and documented simultaneously, maintaining the same GAMS model file. The basic structure of a mathematical model coded in GAMS has the components:

sets, data, variable, equation, model and output and the solution procedures are shown below.

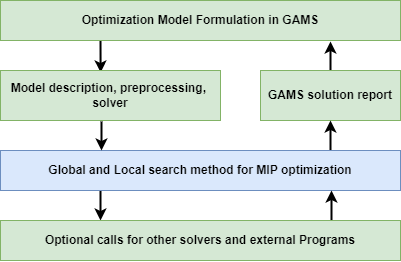


Fig. 2: GAMS modeling and solution procedure.

GAMS formulation follows the basic format as given below:

**Sets**: Declaration, Assignment of members;

**Data** (parameters, tables, scalars), Declaration, Assignment of values;

**Variables**: Declaration, Assignment of type, Assignment of bounds and/or initial values (optional);

**Equations**: Declaration, Definition;

**Model and solve statements**;

**Display statements** (optional);

In order to implement a mathematical model using GAMS optimization package the following steps are needed:

1. Define the decision variables using some of these definitions: variables, integer variables, positive variables or binary variables. The selection of the type of variables depends on the nature of the optimization problem i.e. binary programming model or mixed-integer nonlinear programming model, among others.

2. Define the set of necessary equations using the reserved word equations. First, the names of the equations are defined and then their mathematical expressions are written.

3. Select the name of the model using the reserved word: MODEL name.

4. Solve the mathematical model using the next commands: SOLVE name USING model type MAXIMIZING or MINIMIZING objective function variable.

5. Use the reserve word DISPLAY to see the solution of variables of interest.

**Genetic Algorithm**

According to Azmathulla et al. (2008) Genetic Algorithms are search algorithms based on the mechanism of natural selection and natural genetics. The basic objective of natural genetics is the retention of fit genes and discarding of poor ones. In nature, weak and unfit species within their environment are faced with extinction by natural selection. The strong ones have greater opportunity to pass their genes to future generation. In the long run, species carrying the correct combination in their genes become dominant in their population. Sometimes, during the slow process of evolution, random changes may occur in genes. If these changes provide additional advantage in the challenge for survival, new species evolve from the old ones. Unsuccessful changes are eliminated by natural selection. In GA terminology, a solution vector X is called an individual or a chromosome. Konak (2016) stated that Chromosomes are made of discrete units called genes. Each gene controls one or more features of the chromosome. GA operate with a collection of chromosomes, called population which is generated randomly. As the search proceeds, GA's Operator called selection or sometimes reproduction returns the population which includes only fitter chromosomes generation after generation and eventually converges. GA's two other operators to generate new solutions from existing ones are crossover and mutation. In crossover, generally two chromosomes called parents are combined together to form new chromosomes, called offspring. The parents are selected among existing chromosomes in the population with preference towards fitness so that offspring is expected to inherit good genes which make the parent fitter. Datta et al. (2012) stated that there are different crossover operators which are selected on the way chromosomes are encoded. Single-point, two-point, Multipoint, uniform, arithmetic, ordered crossover are some of the examples which have been reported in the literature By iteratively applying the crossover operator, genes of good chromosomes are expected to appear more frequently in the population, eventually leading to convergence to an overall good solution. The mutation operator introduces random changes into characteristics of chromosomes. Mutation is generally applied at the gene level. In typical GA implementations, the mutation rate (probability of changing the properties of gene) is very small and depends on the length of the chromosome. Therefore the new chromosome produced by mutation will not be very different from the original one. Mutation plays a critical role in GA. Crossover leads the population to converge by making the population alike. Mutation reintroduces genetic diversity back into the population and assists the search escape from local optima. There are many different forms of mutation for different kinds of representation. Flipping, Interchanging, Reversing Gaussian, Boundary, uniform and non-uniform are some of the mutation operators which have been used by researchers. Selection of chromosomes for the next generation is based on the fitness of an individual. There are different selection procedures in GA depending on how the fitness values are used. The most commonly used methods of selecting chromosomes to crossover are Roulette wheel selection, Boltzmann selection, Proportional selection, ranking and tournament selection upon application of said operators on the population, there is a chance that best chromosomes may be lost when new population is created by crossover and mutation. In such cases researchers have suggested the use of Elitism; which recommends the best chromosomes be copied to new population.

A typical genetic algorithm (GA) consists of the following steps (Holland, 1989):

**Step 1:** Generate an initial population of *N* solutions.

**Step 2:** Evaluate each solution of the initial population using a fitness function/objective function.

**Step 3:** Select solutions as parents for the new generation based on probability or randomness. The best solutions (in terms of fitness or objective) have a higher probability of being selected than poor solutions.

**Step 4:** Use the parent solutions from Step 3 to produce the next generation (called offspring). This process is called as crossover. The offspring are placed in the initial set of solutions replacing the weaker solutions.

**Step 5:** Randomly alter the new generation by mutation. Usually this is done using a mutation probability.

**Step 6:** Repeat Steps 2 through 5 until a stopping criteria is met.

A flowchart of a simple GA is shown in Figure 1.1 below:

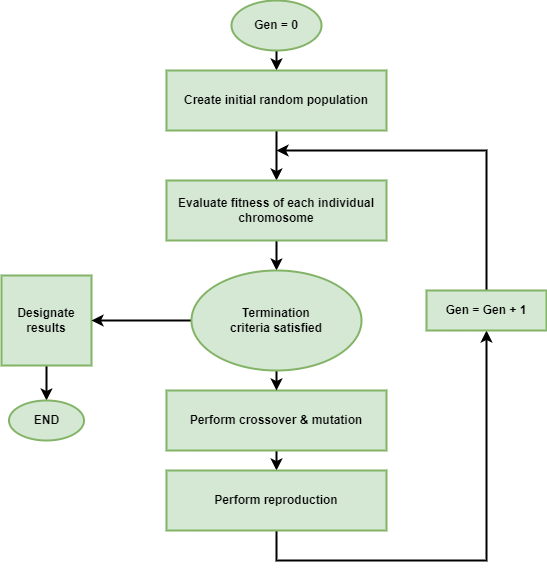


Figure 1.1: Genetic Algorithm Flowchart

The procedure of GA is as follows:

Step 1. Set t=1. Randomly generate N solutions to form the first population, Pt. Evaluate the fitness of solutions in Pt.

Step 2. Crossover: Generate an offspring population Qt as follows:

2.1. Choose two solutions x and y from Pt based on the fitness values.

2.2. Using a crossover operator, generate offspring and add them to Qt.

Step 3. Mutation: Mutate each solution x ϵ Qt with a predefined mutation rate.

Step 4. Fitness assignment: Evaluate and assign a fitness value to each solution x ϵ Qt based on its objective function value and infeasibility.

Step 5. Selection: Select N solutions from Qt based on their fitness and copy them to Pt+1

Step 6. If the stopping criterion is satisfied, terminate the search and return to the current population, else, set t= t+1 go to step 2.

A general pseudo code algorithm for finding solutions is outlined below.

Begin

Iteration= 0

Initiate population P(0)

Evaluate Population P(0)

While (not termination criterion) do

Begin

Iteration=Iteration+1

Selection P(Iteration) from P(Iteration-1)

Alter P(iteration)

Evaluate P(Iteration)

End

End

i = 1;

Pop(1) = random\_pop fitness\_eval(Pop(1)) i = 2;

WHILE terminating\_condition == false

p\_best = findbest(Pop(i-1))

Pop(i) = select(Pop(i-1),sel\_ops);

Pop(i) = crossover(Pop(i));

Pop(i) = mutation(Pop(i));

Fitness\_eval(Pop(i))

Pop(i)=Pop(i) ∪ p\_best

i=i+1;

END WHILE

random\_pop: this function creates a random initial population of 100 individuals in our experiments.

fitness\_eval(Pop): the fitness of each new individual (i.e. resulting from a random initialization, or from a crossover or mutation operation that has produced a solution differing from its parent(s)) in Pop is computed. As mentioned, the chromosome of an individual only specifies a part of the decision variables of the scheduling process (Yid), while the remaining variables (Y0i, Xikm) and the scheduled 15 times for each operation are determined by a sequence of constructive heuristic algorithms that will be described in the next subsection. Once the overall schedule of the whole supply chain has been defined by the constructive algorithms, the value of the cost function associated with each individual is computed and assigned as the fitness of the chromosome.

select(Pop,sel\_ops): this function returns a new population of solutions selected from those in Pop with a strategy that assigns higher probability of selection to individuals with higher fitness. We use tournament selection (Michalewicz, 1996) with two individuals for each tournament.

crossover(Pop): this function randomly selects couples of solutions in Pop to perform a crossover that returns two new individuals, which partially inherit some characteristics of both parents. After the crossover, the resulting offspring replaces the two parents. Finding effective operators and their optimal rate of application for a given problem is a key-issue for any successful application of a GA. The literature offers an extensive amount of comparative analyses of recombination operators for various assignment, sequencing or scheduling problems. While few conclusions are sufficiently general to be extended to our case study, an exhaustive comparison of all the possible combinations of operators is certainly prohibitive in our context. However, there are recent works (e.g. Nearchou (2004), Ishibuchi et al. (2003)), which suggest efficient operators for either part of the chromosome. Although finding more efficient operators for our coding strategy may be an interesting direction for further research, here we focus only on these known operators. Our crossover operator randomly selects a chromosome cut point. If this point falls on the first half of the chromosome, it performs a standard single-cut crossover (Michalewicz, 1996) to the first part of the chromosome (request to depot assignment), otherwise it performs an order-based crossover (Ishibuchi et al., 2003) on the remaining part. Based on a preliminary set of runs for algorithm configuration, the probability of crossover has been chosen equal to 0.5 (i.e. about 50% of the individuals in Pop are subject to crossover).

mutation(Pop): this function randomly alters a solution to obtain a new one. Similarly to crossover, we selectively apply two different mutation operators. A gene in the chromosome is selected randomly. If it belongs to the first part, the gene is replaced by a randomly drawn integer between 1 and D. Otherwise, the inversion mutation (two randomly selected genes are swapped in the sequence) is applied to the order-based part. The effects of this operator are illustrated in Figure 5. Based on preliminary experiments, the probability of mutation was chosen equal to 0.02. This value (as well as the rate of crossover) agrees with the values usually suggested in technical literature on GAs.

terminating\_condition: for all our experiments, we stopped the algorithm after 200 generations.