CHAPTER 3

INTELLIGENCE TECHNIQUES FOR SOLVING MAINTENANCE SCHEDULING PROBLEM

3.1 INTRODUCTION

Three types of main optimization techniques have been used to solve maintenance scheduling problems for power systems; these are Mathematical approaches, Heuristic approaches and Artificial Intelligence approaches.

In this thesis we employ Genetic Algorithms, Particle Swarm Optimization, Evolutionary Programming and Shuffled Frog Leaping algorithm as an optimization technique. Genetic Algorithms are based on natural genetic and evolution mechanisms which can be used to solve complicated optimization problems. In different publications (Dawkins 1976), (Dahal and McDonald 1997), (Dahal et al 1999), (Dahal et al 2000), (Negnevitsky and Kelareva 1999) and (Baskar et al 2003) Genetic Algorithm was considered to solve the maintenance scheduling problem.

3.2 GENETIC ALGORITHM

Genetic Algorithm (GA) works on the theory of Darwin's theory of evolution and the survival-of-the fittest. Genetic algorithms guide the search through the solution space by using natural selection and genetic operators, such as crossover, mutation and the selection.
Genetic Algorithm encodes the decision variables or input parameters of the problem into solution strings of a finite length. While traditional optimization techniques work directly with the decision variables or input parameters, genetic algorithms usually work with the coding. Genetic algorithms start to search from a population of encoded solutions instead of from a single point in the solution space. The initial population of individuals is created at random. Genetic algorithms use genetic operators to create Global optimum solutions based on the solutions in the current population. The most popular genetic operators are (1) selection, (2) crossover and (3) mutation. The newly generated individuals replace the old population and the evolution process proceeds until certain termination criteria are satisfied.

3.2.1 Selection

The selection procedure implements the natural selection or the survival-of-the fittest principle and selects good individuals out of the current population for generating the next population according to the assigned fitness. The existing selection operators can be broadly classified into two classes: (1) proportionate schemes, such as roulette-wheel selection and stochastic universal selection and (2) ordinal schemes, such as tournament selection and truncation selection. Ordinal schemes have grown more and more popular over the recent years and one of the most popular ordinal selection operators is tournament selection. After selection, crossover and mutation recombine and alter parts of the individuals to generate new solutions.

3.2.2 Crossover

Crossover also called the recombination operator, exchanges parts of solutions from two or more individuals, called parents and combines these parts to generate new individuals, called children, with a crossover
probability. There are a lot of ways to implement a recombination operator. The well-known crossover operators include one-point crossover. When using one-point crossover, only one crossover point is chosen at random, for example let there be two parent string $A_1$ and $A_2$ as:

$$A_1 = 1 \ 1 \ 1 \ 1 \ | \ 1 \ 1$$
$$A_2 = 0 \ 0 \ 0 \ 0 \ | \ 0 \ 0$$

Then, one-point crossover recombines $A_1$ and $A_2$ and yields two offsprings $A_{-1}$ and $A_{-2}$ as:

$$A_{-1} = 1 \ 1 \ 1 \ 1 \ | \ 1 \ 1$$
$$A_{-2} = 0 \ 0 \ 0 \ 0 \ | \ 1 \ 1$$

### 3.2.3 Mutation

Mutation usually alters some pieces of individuals to form perturbed solutions. In contrast to crossover, which operates on two or more individuals, mutation operates on a single individual. One of the most popular mutation operators is the bitwise mutation, in which each bit in a binary string is complemented with a mutation probability. For example,

$$A_1 = 1 \ 1 \ 1 \ 1 \ | \ 1 \ 1$$
$$A_{-1} = 0 \ 0 \ 0 \ 0 \ | \ 0 \ 1$$

The step-by-step implementation of Genetic Algorithm is explained as follows:

**Step 1:** Initialize Genetic Algorithm parameters which are necessary for the algorithm. These parameters include population size which indicates the number of
individuals, number of generations necessary for the termination criterion, crossover probability, mutation probability, number of design variables and respective ranges for the design variables. If binary version of Genetic Algorithm is used then string length is also required as the algorithm parameter.

Step 2: Generate random population equal to the population size specified. Each population member contains the value of all the design variables. This value of design variable is randomly generated in between the design variable range specified. In Genetic Algorithm, population means the group of individuals which represents the set of solutions.

Step 3: Obtain the values of the objective function for all the population members. The value of the objective function so obtained indicates the fitness of the individuals. If the problem is a constrained optimization problem then a specific approach such as static penalty, dynamic penalty and adaptive penalty is used to convert the constrained optimization problem into the unconstrained optimization problem.

Step 4: This step is for the selection procedure to form a mating pool which consists of the population made up of best individuals. The commonly used selection schemes are roulette-wheel selection, tournament selection, stochastic selection etc. The simplest and the commonly used selection scheme is the roulette-wheel selection, where an individual is selected for the mating pool with the
probability proportional to its fitness value. The individual (solution) having better fitness value will have more number of copies in the mating pool and so the chances of mating increases for the more fit individuals than the less fit ones. This step justifies the procedure for the survival of the fittest.

Step 5: This step is for the crossover where two individuals, known as parents, are selected randomly from the mating pool to generate two new solutions known as off-springs. The individuals from the population can go for the crossover step depending upon the crossover probability. If the crossover probability is more, then more individuals get chance to go for the crossover procedure. The simplest crossover operator is the single point crossover in which a crossover site is determined randomly from where the exchange of bits takes place. The crossover procedure is explained through Equations (3.1) and (3.2).

Step 6: After crossover, mutation step is performed on the individuals of population depending on the mutation probability. The mutation probability is generally kept low so that it does not make the algorithm unstable. In mutation, a random site is selected from the string of individuals and it is flapped as explained through Equation (3.3).
Step 7: Best obtained results are saved using elitism. All elite members are not modified using crossover and mutation operators but can be replaced if better solutions are obtained in any iteration.

Step 8: Repeat the steps (from step 3) until the specified number of generations or termination criterion is reached.

3.2.4 Flow Chart of Genetic Algorithm

Several modifications can be applied to the basic Genetic Algorithm to improve the performance on practical problems. One of them is Elitism; Probabilistic nature of the selection process gives a chance of reproduction even to the weakest number of the population. Likewise there is a chance that the best-performing member might not be present in the next generation due to structural changes following cross over and mutation. Hence it is derivable to copy the elite structures into the next gene. Best-fit string of each gun is copied to next gun without undergoing cross over and mutation. In order to reduce the memory space requirement for storing the genes, the simple Genetic Algorithm is altered in such a way that chosen genes replace the new child genes in the current generation. The dotted line path indicates this change in the flowchart given. The cross over and mutation operations in an iteration of the basic Genetic Algorithms algorithm is shown in the flowchart of Figure 3.1.
3.2.5 Algorithms

The series of operations carried out when implementing a genetic algorithm is:

1. Construct an initial population (P) of chromosomes by random process.
2. Evaluate fitness of each chromosome.
3. Genetic mating pool based on fitness function values.

4. Select mating pair of chromosomes called parent chromosomes from mating pool.

5. Create two child chromosomes from the parent chromosomes by applying genetic operators.

6. Repeat steps (4-5), till the child population of size P is generated.

7. Store the chromosome having the maximum fitness and also the corresponding objective function.

8. Repeat steps (2-7), until the specified numbers of genetic iterations are completed.

9. Return the chromosome with highest fitness function as the solution.

3.2.6 Applications of the Genetic Algorithms

An overview of applications of genetic algorithms to real-world problems.

- **Genetic Algorithms in Parametric Design of Aircraft**, by Mark F. Bramlette and Eugene E. Bouchard. The authors discuss optimizing aircraft designs when the task is posed as that of optimizing a list of parameters. They have approached the problem with a number of optimization algorithms, including a genetic algorithm using real number representation. They also discuss the performance of each algorithm and describe some innovative techniques used in their quite successful genetic algorithm, including the
technique of generating a large number of initial population members and then working only with the best ones.

- **Dynamic Anticipatory Routing in Circuit-Switched Telecommunications Networks**, by Louis Anthony Cox, Jr., Lawrence Davis and Yuping Qiu. The objective of the study is to optimize the routing of telephone networks in order to minimize costs to US West. It compares the performance of an order-based genetic algorithm with several other optimization techniques on this problem. The authors conclude that the genetic algorithm is a highly successful technique when the problem is complex, but hybridization of these algorithms can lead to better performance than using any of them in isolation.

- **A Genetic Algorithm Applied to Robot Trajectory Generation**, by Yuval Davidor. He shows how to apply genetic algorithm techniques to the task of planning the path which a robot arm is to take in moving from one point to another. Davidor uses variable-length chromosomes in his solution, and devises some novel and interesting crossover operators.

- **Genetic Algorithms, Nonlinear Dynamical Systems, and Models of International Security**, by Stephanie Forrest and Gottfried Mayer-Kress, concerns a problem posed by current research in chaotic models of real processes. Chaotic models of international arms races and economic competition seem to model some features of the real-world processes better than some other more traditional models have done. The authors use a genetic algorithm to find good settings of the
parameters of Mayer-Kress's models in order to enhance their performance on the models.

- **Strategy Acquisition with Genetic Algorithms**, by John J. Grefenstette. He experiments with Samuel, a genetic algorithm that learns techniques for maneuvering a simulated airplane in order to evade a simulated missile. The genetic algorithm he describes employs several techniques of interest, including variable-length chromosomes composed of rules that form a production system. A chromosome is evaluated by using those rules to maneuver the airplane in simulated interactions between airplanes and missiles. Grefenstette has built knowledge of the production rule domain into his operators in clever ways.

- **Genetic Synthesis of Neural Network Architecture**, by Steven A. Harp and Tariq Samad, that describes techniques for encoding neural network architectures on binary chromosomes. The authors use variable-length chromosomes and a variety of other novel techniques. This is a good place to begin in learning how to combine neural networks and genetic algorithms.

- **Air-Injected Hydrocyclone Optimization Via Genetic Algorithm**, by Charles L. Karr, that describes the solution of a design problem by a genetic algorithm using the bit string representation technique. Karr represents the design of an air-injected hydrocyclone as a list of parameters. An interesting feature of his approach is the use of a new operator called "simplex reproduction". Karr shows that a genetic algorithm
using this operator is quite effective as a search technique for finding design parameter combinations.

- **A Genetic Algorithm Approach to Multiple Fault Diagnosis**, by Gunar E. Liepens and W. D. Potter, that discusses the use of a genetic algorithm for finding the most plausible combination of causes for alarms in a microwave communication system. The authors use binary chromosomes to represent solutions to a problem that they show is a type of set covering problem. They show how to incorporate knowledge about set covering optimization into their genetic algorithm in novel ways, yielding a high-performance hybrid solution to the problem.

- **A Genetic Algorithm for Conformational Analysis of DNA**, by C. B. Lucasius, M. J. J. Blommers, L. M. C. Buydens, and G. Kateman. It is a development of a genetic algorithm for determining the structure of a sample of DNA based on spectrometric data about the sample. An interesting "cascaded" evaluation technique that greatly enhances the efficiency of their evaluation function is used. The authors use bit strings to encode molecular structures. Their evaluation function measures the degree to which each decoded structure conforms to the data that have been collected about the sample. The genetic algorithm evolves a description of molecular structure that is in agreement with the data collected. The problem of determining biomolecular structure occupies a central position in the worlds of fundamental and applied chemistry today.

- **Automated Parameter Tuning for Sonar Information Processing**, by David J. Montana. An application of genetic
algorithms to two problems associated with interpreting passive sonar data. The first is a parameterization problem. To solve it, Montana uses a floating-point version of OOGA to find good parameter settings for the algorithms employed in the process of interpreting sonar data. The second problem is a classification problem. For this problem, a genetic algorithm is used to train neural networks classifying sonar signals in various ways. In this second system, Montana and Davis experiment with a number of domain-based operators, including the use of backpropagation -a neural network technique- as a genetic algorithm operator. This application is useful if you are interested in hybrid genetic algorithms, real number representations for parameterization, or neural networks.

- **Interdigitation: A Hybrid Technique for Engineering Design Optimization**, by Gilbert Syswerda. An application of a genetic algorithm to the problem of scheduling activities in a laboratory in which each activity may affect the others in a variety of ways. Syswerda has been implementing this system under contract to the U. S. Navy. The genetic algorithm uses an order-based chromosome to represent its schedule. The chromosome is decoded with a decoder that incorporates a good deal of knowledge about the scheduling domain.

- **The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination**, by Darrell Whitley, Timothy Starkweather, and Daniel Shaner. The authors describe a technique for solving the traveling salesman problem, a well-known combinatorial optimization
problem. Their solution includes novel and ingenious representation, crossover and repair mechanisms. They also show how similar techniques can be applied to the problem of scheduling a Hewlett-Packard production line.

3.3 PARTICLE SWARM OPTIMIZATION

3.3.1 Introduction

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995, inspired by the social behavior of bird flocking or fish schooling.

Particle swarm optimization shares many similarities with evolutionary computation techniques such as Genetic Algorithms. The system is initialized with a population of random solution and searches for optima by updating generations. However, unlike Genetic Algorithms, Particle swarm optimization has no evolutionary operators such as crossover and mutation.

In Particle swarm optimization, the potential solution, called particles, fly through the problem space by following the current optimum particles. Each particle keeps track of the coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far (The fitness value is also stored). This value is called pbest. Another best value that is tracked by the particle swarm optimizer is best value, obtained so far by any particle in the neighborhood of the particles. This location is called gbest.

The Particle swarm optimization concept consists of, at each time step, changing the velocity of (accelerating) each particle towards its pbest and gbest locations. Acceleration is weighted by a random term, with separate
random number being generated for acceleration toward pbest and gbest locations.

In past several years, Particle swarm optimization has been successfully applied in many research and application areas. It is demonstrated that Particle swarm optimization gets better results in a faster and cheaper way compared with other methods.

Another reason for Particle swarm optimization being attractive is that there are few parameters to adjust. One version, with slight variations, works well in a wide variety of applications. Particle swarm optimization has been used for approaches that can be used across a wide range of application, as well as for specific application focused on a specific requirement.

In order to overcome the disadvantage of other technique, a Particle swarm optimization is applied for minimizing of operating and maintenance cost than Genetic Algorithms. It does not depend upon number of units.

3.3.2 Significance

Many advantages of Particle swarm optimization over other traditional optimization techniques can be summarized as follows:

- It is a population based search algorithm. This property ensures Particle swarm optimization to be less susceptible in being trapped on local minima.

- It makes use of the probabilistic transition rules and not deterministic rules. Hence, Particle swarm optimization is a kind of stochastic optimization algorithm that can search a complicated and uncertain area. This makes Particle swarm
optimization more flexible and robust than conventional methods.

- It can easily deal with non-differentiable objective functions because Particle swarm optimization uses payoff (performance index or objective function) information to guide the search in the problem space. Additionally, this property relieves Particle swarm optimization of assumptions and approximations, which are often required by traditional optimization models.

- The solution quality of the proposed approach does not depend on the initial population. Starting anywhere in the search space, the algorithm ensure the convergence to the optimal solution. Therefore, this method is different from traditional techniques.

- It has the flexibility to control the balance between the global and local exploration of a Particle swarm optimization overcomes the premature convergence problem and enhance the search capability which makes it different from Genetic Algorithm and other heuristic algorithms.

Particle swarm optimization is initialized with a group of random particles (solution) and then searches for optima by updating generations. In all iteration, the particle updates its position by following two best values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored). This value is called pbest. Another best value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is the global best and called gbest.

After finding the two best values, the particle updates its velocity and positions with following equation
\[ v^{P+1}_t = \omega v^P_t + C_1 \cdot rand \cdot (p^{\text{KP}} - p^P_t) + C_2 \cdot rand \cdot (p^{\text{gi}}_t - p^P_t) \]  

(3.1)

\[ p^{\text{KP}}_i = p^{\text{KP}}_{i-1} + v^{P+1}_t \]

(3.2)

where \( v^{P+1}_t \) is the updated particle velocity in the next iteration, \( v^P_t \) is the particle velocity in the current iteration, \( \omega \) is the inertial dampener which indicates that the impact of the particle’s own experience on its next movement, \( C_1 \cdot rand \) represents a uniformly distributed number within the interval \([0, C_1]\), which reflects how the neighbors of the particle affects its flight, \( p^{\text{KP}}_{\text{exp}} \) is the neighborhood best position, \( v^P_t \) is the current position of the particle, \( C_2 \cdot rand \) represents a uniformly distributed number within the interval \([0, C_2]\), which indicates how the particle trusts the global best position (usually \( C_1 = C_2 = 2 \)), \( p^{\text{gi}}_t \) is the global best position and \( v^{P+1}_t \) is the updated position of the particle. Under the guidance of these two updating rules, the particles will be attracted to move toward the best position found thus far. That is, the optimal solutions can be sought out due to this driving force.

### 3.3.3 Pseudo Code for PSO

The pseudo code of the procedure is as follows

For each particle

Initialize particle

End

Do

For each particle

Calculate fitness value

If the fitness value is better than the best fitness value (p\text{best}) in history

Set current value as the new p\text{best}

End If

End For

End Do
End

Chose the particle with the best fitness value of all the particles as the gbest.

For each particle

  Calculate particle velocity according to Equation (3.1)

  Update particle position according to Equation (3.2)

End

While maximum iteration or minimum error criteria is not attained, particles velocity on each dimension are clamped to a maximum velocity $V_{\text{max}}$. If the sum of acceleration would cause the velocity on that dimension to exceed $V_{\text{max}}$, which is a parameter specified by the user, then the velocity on the dimension is limited to $V_{\text{max}}$.

3.3.4 Basic Terms used in PSO

The basic terms used in Particle swarm optimization technique are stated and defined as follows:

3.3.4.1 Population

It is a candidate solution represented by a d-dimensional real valued vector, where d is the number of optimized parameters. The population size refers to the number of particles and it depends on the dimension of the problem. The typical range of number of particles is 20 to 40. Actually, for most of the problems ten particles is large enough to get good results. For some difficult or special problems, one can try hundred or two hundred particles as well.
### 3.3.4.2 Swarm

Swarm may be defined as an apparently disorganized population of moving particles that tend to cluster together while each particle seems to be moving in a random direction.

### 3.3.4.3 Particle velocity

Particle velocity is the velocity of the moving particles represented by a $d$ dimensional real-valued vector.

### 3.3.4.4 Individual best

When particles are moving through the search space, it compared its fitness value at the current position to the best fitness value it has ever reached at any iteration up to the current iteration. The best position that is associated with the best fitness encountered so far is called the individual best or pbest. For each particle in the swarm, pbest can be determined and updated during the search.

### 3.3.4.5 Global best

Global best (gbest) is the best position among all the individual best position achieved so far.

### 3.3.4.6 Maximum velocity ($V_{\text{max}}$)

An upper limit is placed on the velocity in all dimensions. This upper limit prevents particles from moving too rapidly from one region in search space to another. If
\[ V_i(t) > V_{\text{max}} \text{ then } V_i(t) = V_{\text{max}} \]

Or

\[ V_i(t) < -V_{\text{max}} \text{ then } V_i(t) = -V_{\text{max}} \]  \hspace{1cm} (3.3)

Where \( V_i(t) \) is the velocity of particle at the time step \( t \). \( V_{\text{max}} \) does not place a limit on the position of a particle, only on the steps made in the hyper dimensional search space. \( V_{\text{max}} \) is usually initialized as a function of the range of the problem.

### 3.3.4.7 Inertial weight factor

Suitable selection of inertia weight \( \omega \) provides a balance between global and local exploration, thus requiring less iteration on average to find a sufficiently optimal solution. As originally developed, \( \omega \) often decrease linearly from about 0.9 to 0.4 during a run. In general, the inertia weight \( \omega \) is set according to the following equation.

\[
\omega = \omega_{\text{max}} - (\omega_{\text{max}} - \omega_{\text{min}}) / \text{iter}_{\text{max}} \times \text{iter} 
\]  \hspace{1cm} (3.4)

Where

\[ \omega_{\text{max}} \] - Initial weight

\[ \omega_{\text{min}} \] - Final Weight

\[ \text{iter}_{\text{max}} \] - Maximum iteration number

\[ \text{iter} \] - Current iteration number
3.3.5 PSO Based Algorithm for Maintenance Scheduling

The step by step procedure to compute the global optimal solution is:

Step 1: Initialize a population of particles with random positions and velocities on dimensions in the problem space.

Figure 3.2 Flow chart for particle swarm optimization algorithm
Step 2: For each particle, evaluate the desired optimization fitness function in the variables.

Step 3: Compare particles fitness evolution with particles $P_{best}$. If current value is better then $P_{best}$, then set $P_{best}$ value equal to the current value and the $P_{best}$ location equal to the current location in the dimensional space.

Step 4: Compare fitness evaluation with the populations overall previous best. If current value is better than $g_{best}$, then reset to the current particles array index and value.

Step 5: Change the velocity and position of the particle according to Equations (3.3) and (3.4) respectively.

Step 6 Loop to step 2 until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations.

3.3.6 Applications of the Particle Swarm Optimization

- Design of a periodic antenna arrays.
- Optimization of profiled corrugated horn antennas.
- Load forecasting.
- STATCOM power system.
- Optimal power dispatch.
- Optimization of internal combustion engines.
- FPGA placement and routing.
3.4 EVOLUTIONARY PROGRAMMING

3.4.1 Introduction

Evolutionary Programming is a mutation-based evolutionary algorithm (Cau and Kaye 2002) and (Nidul Sinha et al 2003) applied to discrete search spaces. (David Fogel 1988) extended the initial work of his father (Larry Fogel 1962) for applications involving real-parameter optimization problems. Real-parameter Evolutionary Programming is similar in principle to evolution strategy (ES), in that normally distributed mutations are performed in both algorithms. Both algorithms encode mutation strength (or variance of the normal distribution) for each decision variable and a self-adapting rule is used to update the mutation strengths. Several variants of Evolutionary Programming have been suggested (Fogel 1992).

3.4.2 Evolutionary Strategies

For the case of Evolutionary strategies (Nidul Sinha et al 2003) Fogel remarks “evolution can be categorized by several levels of hierarchy: the gene, the chromosome, the individual, the species and the ecosystem.” Thus, while Genetic Algorithms stress models of genetic operators, Evolutionary Strategies emphasize mutational transformation that maintains behavioral linkage between each parent and its offspring at the level of the individual. Evolutionary Strategies are a joint development of Bienert, Rechenberg, and Schwefel. The first applications were experimental and addressed some optimization problems in hydrodynamics (Burke and Smith 2000).

Evolutionary programming (Back 1996, Fogel 1995, Fogel, Owens and Walsh 1996) is conducted as a sequence of operations and is given below.
1. The initial population is determined by setting $s_i = S_i \sim U(a_k, b_k)^k$ for $i = 1, \ldots, m$, where $S_i$ is a random vector, $s_i$ is the outcome of the random vector, $U(a_k, b_k)^k$ denotes a uniform distribution ranging over $[a_k, b_k]$ in each of $k$ dimensions, and $m$ is the number of parents.

2. Each $s_i$, $i = 1, \ldots, m$, is assigned a fitness score $S(s_i) = G(F(s_i), v_i)$, where $F$ maps $s_i \rightarrow \mathbb{R}$ and denotes the true fitness of $s_i$, $v_i$ represents random alteration in the instantiation of $s_i$, random variation imposed on the evaluation of $F(s_i)$, or satisfies another relation $s_i$ and $G(F(s_i), v_i)$ describes the fitness score to be assigned. In general, the functions $F$ and $G$ can be as complex as required. For example, $F$ may be a function not only of a particular $s_i$, but also of other members of the population, conditioned on a particular $s_i$.

3. Each $s_i$, $i = 1, \ldots, m$, is altered and assigned to $s_i + m$ such that

$$s_i + m = s_{ij} + N(0, P_j S(s_i) + Z_j), j = 1, \ldots, k$$

$N(0, P_j S(s_i) + Z_j)$ represents a Gaussian random variable with mean $u$ and variance $\sigma^2$. $P_j$ is a constant of proportionality to scale $S(s_i)$, and $Z_j$ represents an offset to guarantee a minimum amount of variance.

4. Each $s_i + m$, $i = 1, \ldots, m$, is assigned a fitness score $S(s_i + m) = G(F(s_i + m), V_i + m)$

5. For each $s_i$, $i = 1, \ldots, 2m$, a value $w_i$ is assigned according to

$$w_i = \sum_{t=1}^{\mathcal{C}} w_t^*$$

$$w_t^* = \begin{cases} 1, & \text{if } S(s_i) \leq S(s_t); \\ 0, & \text{otherwise}; \end{cases}$$
Where \( p = \lceil 2mu + 1 \rceil, p \neq i, \lfloor x \rfloor \) denotes the greatest integer less than or equal to \( x \), \( c \) is the number of competitions, and \( u \sim U(0,1) \).

6. The solutions \( s_i, i = 1...2m \), are ranked in descending order of their corresponding value \( W_i \) [with preference to their actual scores \&(s_i) if there are more than \( m \) solutions attaining a value of \( c \)]. The first \( m \) solutions are transcribed along with their corresponding values \( S(s_i) \) to be the basis of the next generation.

7. The process goes to step 3, unless the available execution time is exhausted or an acceptable solution has been discovered.

The flow chart for the Evolutionary Programming algorithm is shown in Figure 3.3.

![Figure 3.3 Flow chart for EP algorithm](image-url)
3.4.3 Evolutionary Programming Based Algorithm for Maintenance Scheduling

1. Initialize the parent vector \( p = [p_1, p_2, \ldots, p_n], i = 1, 2, \ldots, N_p \) such that each element in the vector is determined by \( p_j \sim \text{random} (p_{j\text{min}}, p_{j\text{max}}), j = 1, 2, \ldots, N, \) with one generator as dependent generator.

2. Calculate the overall objective function is given in equation 3.1 using the trail vector \( p_i \) and find the minimum of \( F_{Ti} \).

3. Create the offspring trail solution \( p_i' \) using the following steps.

   a) Calculate the standard deviation
   \[
   \sigma_j = \beta (F_{Tij} / \min(F_{Ti}))(P_{j\text{max}} - P_{j\text{min}})
   \]

   b) Add a Gaussian random variable to all the state variable of \( p_i \), to get \( p_i' \).

4. Select the first \( N_p \) individuals from the total \( 2N_p \) individuals of both \( p_i \) & \( p_i' \) using the following steps for next iteration.

   i. Evaluate \( r = (2N_p \text{ random } (0,1) + 1) \)

   ii. Evaluate each trail vector by \( W_{pi} = \sum (W_x) \) Where \( x = 1, 2, \ldots, N_p, i = 1, 2, \ldots, 2N_p \) such that
   \( W_x = 1 \) if \( F_{Tij} / (F_{Tij} + F_{Tir}) < \text{random } (0,1) \), otherwise, \( W_x = 0. \)

5. Sort the \( W_{pi} \) in descending order and the first \( N_p \) individuals will survive and are transcribed along with their elements to form the basis of the next generation.

6. The above procedure is repeated from step (2) until a maximum number of generations \( N_m \) is reached.

7. Selection process is done using Evolutionary strategy.
3.4.4 Applications of the Evolutionary Programming

- Automated design of industrial equipment using catalogs of exemplar lever patterns.
- Automated design of sophisticated trading systems in the financial sector.
- Filtering and signal processing (links to particle filters and a tutorial on genetic particle models)
- Finding hardware bugs.
- Game theory equilibrium resolution.
- Genetic Algorithm for Rule Set Production.
- Economics
- Learning robot behavior using Evolutionary Programming.

3.5 SHUFFLED FROG LEAPING ALGORITHM

3.5.1 Introduction

The shuffled frog leaping algorithm is a memetic meta-heuristic that is designed to seek a global optimal solution by performing a heuristic search. It is based on the evolution of memes carried by individuals and a global exchange of information among the population (Eusuff and Lansey 2003). In essence, it combines the benefits of the local search tool of the particle swarm optimization (Kennedy and Eberhart 1995) and the idea of mixing information from parallel local searches to move toward a global solution (Duan et al 1993). The shuffled frog leaping algorithm has been tested on several combinatorial problems and found to be efficient in finding global solutions (Eusuff and Lansey 2003). Shuffled frog leaping algorithm
has been successfully applied to several engineering optimization problems such as water resource distribution (Eslamian et al 2009), bridge deck repairs (Elbehairy et al 2006), job-shop scheduling arrangement (Rahimi-Vahed and Mirzaei 2007), and traveling salesman problem (TSP) (Luo et al 2008). The most distinguished benefit of Shuffled frog leaping algorithm is its fast convergence speed (Elbeltagi et al 2005). The Shuffled frog leaping algorithm combines the benefits of the both the genetic-based memetic algorithm (MA) and the social behavior-based Particle Swarm Optimization algorithm (Kennedy and Eberhart 1995).

The shuffled frog leaping algorithm involves a population of possible solutions defined by a set of frogs (i.e. solutions) that is partitioned into subsets referred to as memeplexes. The different memeplexes are considered as different cultures of frogs, each performing a local search. Within each memeplex, the individual frogs hold ideas, that can be influenced by the ideas of other frogs and evolve through a process of memetic evolution. After a number of memetic evolution steps, ideas are passed among memeplexes in a shuffling process (Liong and Atiquzzaman 2004). The local search and the shuffling processes continue until convergence criteria are satisfied (Eusuff and Lansey 2003).

The shuffled frog leaping algorithm is described by the pseudocode and the corresponding flowchart.

### 3.5.2 Pseudocode for SFL algorithm

Begin;

Generate random population of $P$ solutions (individuals); For each individual $e P$: calculate fitness (i); Sort the whole population $P$ in descending order of their fitness;

Divide the population $P$ into $m$ memeplexes; For each memeplex;
Determine the best and worst individuals;
Improve the worst individual position using Equations 3.5 and 3.6;
Repeat for a specific number of iterations;
End;
Combine the evolved memeplexes;
Sort the population $P$ in descending order of their fitness;
Check if termination = true;
End;

First, an initial population of $P$ frogs is created randomly. For $S$-dimensional problems, each frog $i$ is represented by $S$ variables as $X_i = (x_{i1}, x_{i2}, \ldots, x_{iS})$. The frogs are sorted in a descending order according to their fitness. Then, the entire population is divided into $m$ memeplexes, each containing $n$ frogs (i.e. $P = mn$). In this process, the first frog goes to the first memeplex, the second frog goes to the second memeplex, frog $m$ goes to the $m$th memeplex, and frog $m + 1$ goes to the first memeplex, and so on. Within each memeplex (figure 3.3), the frogs with the best and the worst fitness are identified as $X_b$ and $X_w$, respectively. Also, the frog with the global best fitness is identified as $X_g$. Then, an evolution process is applied to improve only the frog with the worst fitness (i.e. not all frogs) in each cycle. Accordingly, the position of the frog with the worst fitness is adjusted as follows:

$$D_i = \text{rand}() (X_b - X_w)$$  \hspace{1cm} (3.5)

$$X_{neww} = X_{oldw} + D_i (-D_{\max} \leq D_i \leq D_{\max})$$  \hspace{1cm} (3.6)

where $\text{rand}()$ is a random number between 0 and 1; and $D_{\max}$ is the maximum allowed change in a frog's position. If this process produces a better frog (solution), it replaces the worst frog. Otherwise, the calculations in Equations (3.5) and (3.6) are repeated with respect to the global best frog.
(i.e. $X_g$ replaces $X_b$). If no improvement becomes possible in this latter case, then a new solution is randomly generated to replace the worst frog with another frog having any arbitrary fitness. The calculations then continue for a specific number of evolutionary iterations within each memeplex (Eusuff and Lansey 2003). The main parameters of the SFL algorithm are: number of frogs $P$, number of memeplexes, and number of evolutionary iterations for each memeplex before shuffling.

### 3.5.3 A Modified Shuffled Frog Leaping Algorithm

In the shuffled frog leaping algorithm, each memeplex is allowed to evolve independently to locally search at different regions of the solution space. In addition, shuffling all the memeplexes and re-dividing them again into a new set of memeplexes results in a global search through changing the information between memeplexes. As such, the shuffled frog leaping algorithm attempts to balance between a wide search of the solution space and a deep search of promising locations that are close to a local optimum.

As expressed by Equation (3.5), each individual frog (solution) in a memeplex is trying to change its position towards the best frog within the memeplex or the overall best frog. As shown in this equation, when the difference in position between the worst frog $X_w$ (i.e. the frog under evolution) and the best frogs ($X_b$ or $X_g$) becomes small, the change in frog $X_w$'s position will be very small, and thus it might stagnate at a local optimum and lead to premature convergence. To overcome such an occurrence, this study proposes that the right-hand side of equation (1) be multiplied by a factor $C$ called the 'search-acceleration factor', as follows:

$$\text{Change in frog position } D_i = \text{rand()} * C * (X_b - X_w) \quad (3.7)$$
Assigning a large value to the factor \( C \) at the beginning of the evolution process will accelerate the global search by allowing for a bigger change in the frog's position and accordingly will widen the global search area. Then, as the evolution process continues and a promising location is identified, the search - acceleration factor, \( C \), will focus the process on a deeper local search as it will allow the frogs to change its positions. The search-acceleration factor, which can be a positive constant value, linear, or nonlinear function of time, provides the means to balance between global and local search.

### 3.5.4 Flow Chart for Shuffled Frog Leaping Algorithm

The flowchart of Shuffled frog leaping algorithm is illustrated in Figure 3.4. Shuffled frog leaping algorithm is a population based random search algorithm inspired by nature memetics. In the Shuffled frog leaping algorithm, a population of possible solution defined by a group of frogs that is partitioned into several communities referred to as memeplexes. Each frog in the memeplexes is performing a local search. Within each memeplex, the individual frog’s behavior can be influenced by behaviors of other frogs and it will evolve through a process of memetic evolution. After a certain number of memetics evolution steps, the memeplexes are forced to mix together and new memeplexes are formed through a shuffling process. The local search and the shuffling processes continue until convergence criteria are satisfied Equation (3.7).

The varies steps are as follows:

1. The Shuffled frog leaping Algorithm involves a population ‘P’ of possible solution, defined by a group of virtual frogs(n).
2. Frogs are sorted in descending order according to their fitness and then partitioned into subsets called as memplexes (m).

3. Frogs i is expressed as $X_i = (X_{i1}, X_{i2}, \ldots, X_{is})$ where $S$ represents number of variables.

4. Within each memplex, the frog with worst and best fitness are identified as $X_w$ and $X_b$.

5. Frog with globle best fitness is identified as $X_g$.

6. The frog with worst fitness is improved according to the following equation.

$$D_i = \text{rand}() (X_b - X_w) \quad (3.8)$$

$$X_{neww} = X_{oldw} + D_i (-D_{max} \leq D_i \leq D_{max}) \quad (3.9)$$

Where rand is a random number in the range of [0,1];

$D_i$ is the frog leaping step size of the i-th frog and $D_{max}$ is the maximum step allowed change in a frog’s position. If the fitness value of new $X_w$ is better than the current one, $X_w$ will be accepted. If it isn’t improved, then the calculated Equation (3.5) and (3.6) are repeated with $X_b$ replaced by $X_g$. If no improvement becomes possible in the case, a new $X_w$ will be generated randomly. Repeat the update operation for a specific number of iterations. Therefore, shuffled frog leaping algorithm simultaneously performs an independent local search in each memplex using a process similar to the Particle Swarm Optimization algorithm. The flowchart of local search of shuffled frog leaping algorithm is illustrated in Figure 3.5.
Figure 3.4 Flowchart of SFLA
Figure 3.5 Flowchart of local search
3.5.5 Applications of the Shuffle Frog Leaping Algorithms

- Airlines Revenue Management.
- Biology and computational chemistry.
- Building phylogenetic trees
- Clustering. Using genetic algorithms to optimize a wide range of different fit-functions
- Container loading optimization
- Scheduling applications, including job-shop scheduling. The objective being to schedule jobs in a sequence-dependent or non-sequence-dependent setup environment in order to maximize the volume of production while minimizing penalties such as tardiness.
- Stochastic optimization (links to particle methods in regulation, optimization, and optimal control)

3.6 SUMMARY

In this chapter, artificial intelligent technique such as Genetic Algorithms, Particle Swarm Optimization, Evolutionary Programming and Shuffled Frog Leaping Algorithm to solve maintenance scheduling problems with maintenance cost and scheduling models for power systems have been considered. Most of these models considered maintenance costs under a specific maintenance strategy. In following chapters, three hybrid techniques have been proposed for different cost models in order to form a single maintenance cost model. There are many solution methods, in this chapter, a wide range of artificial intelligence approaches was discussed and these techniques were used to solve different maintenance scheduling problems.
Hybrid technique such as Particle Swarm Optimization based Genetic Algorithm (PSO–GA), Particle Swarm Optimization based Evolutionary Programming (PSO–EP) and Particle Swarm Optimization based Shuffled Frog Leaping Algorithm (PSO–SFLA) was found to be a powerful optimization tool for solving maintenance scheduling for power systems. In this research, above said three hybrid technique were used to solve scheduling approach.