Int J Adv Manuf Technol (2004) 23: 846–856
DOI 10.1007/s00170-002-1484-y

ORIGINAL ARTICLE

T. Paul Robert · P. Shahabudeen

Genetic algorithms for cost-effective maintenance of a reactor-regenerator system

Received: 25 April 2002 / Accepted: 15 July 2002 / Published online: 14 February 2004

Abstract Developments in automation and the resulting complexity of the systems involved have made the reliability of machines an important issue. This is especially true in the process industry, which is characterised by expensive specialised equipment and stringent environmental considerations. Nowadays, with profit margins decreasing, the need for good maintenance planning and control is obvious. Determining the best cost-effective maintenance, though, is computationally difficult, when the parameters, viz., the mean time between failures (MTBF) and the mean time to repair (MTTR) of the critical components in the system can be perturbed. In this paper, the use of metaheuristic, genetic algorithms to create cost effective maintenance in a process plant is presented.

Keywords Cost-effective maintenance · Optimization · Genetic algorithm · Orthogonal experiments

1 Introduction

In recent years, with the drive for greater profitability, the process industries have been placing greater emphasis on improving process performance, operating closer to constraints and reducing downtime. Several technologies, such as process modelling, statistical process control, optimisation techniques and advisory systems have helped to make these goals a reality [1]. Difficulties are encountered when process industries begin to deal with optimisation tasks such as minimising total cost and maximising availability [2]. In order to determine optimal maintenance, the effect on plant performance of alternative maintenance policies and the costs of the resources they would require must be compared with alternative uses of these same resources. This requires a full scale detailed analysis if the best policy is to be selected. Dekker [3] presented an overview of the application of maintenance optimisation models, analysed the role of these models in maintenance and discussed the factors that may have hampered applications. Al-Bahi [4] proposed a spare provisioning policy based on the maximisation of availability per cost ratio. Harunuzzaman and Aldemir [5] proposed a methodology based on dynamic programming to find the minimum cost maintenance schedule for nuclear power plant standby safety systems. Bandyopadhyay et al. [6] provided a cost-effective maintenance program for a cross-country petroleum pipeline through risk analysis.

Genetic algorithms (GAs), which are global optimal algorithms based on Darwin's evolutionary theory, have created an immense interest among researchers looking to solve these types of complex optimisation problems. Although there have already been a number of papers on the application of genetic algorithms for solving optimisation problems in the area of scheduling and sequencing [7], group technology, facility layout and location, transportation, cellular manufacturing [8] and kanban systems [9], there is a lack of genetic algorithm applications for performance optimisation in complex industrial systems. Coit and Smith [10] have demonstrated the applications of genetic algorithms for reliability optimisation of series-parallel systems. The problem centres on selecting components and redundancy-levels that optimise design configuration, given system-level constraints on reliability, cost, and/or weight. Coit and Smith [11] also solved the redundancy allocation problem using a combined neural network and GA approach. The genetic algorithm searches for the minimum cost solution by selecting the appropriate components for a series-parallel system, given a minimum-system reliability constraint. A neural network is used to estimate system reliability during the search. Ramachandran et al. [12] have developed models for

T. P. Robert (✉) · P. Shahabudeen
Department of Industrial Engineering,
College of Engineering,
Anna University, 600 025 Chennai, India
E-mail: prpaul@hotmail.com

replacement strategies using genetic algorithms. Two models have been developed: one for implementing the strategy using a discrete function to replace the items whose efficiency deteriorates with time, and the other for implementing the strategy using a continuous function.

A literature review reveals that there is a lack of genetic algorithm applications for solving maintenance optimisation problems. In this study, an attempt has been made to evaluate multiple corrective maintenance policies and to suggest the best policy to the maintenance manager using genetic algorithms. An efficient algorithm for the optimisation problem of determining the cost-effective maintenance of the Reactor-Regenerator system of a fluid catalytic cracking unit (FCCU) is presented in this paper.

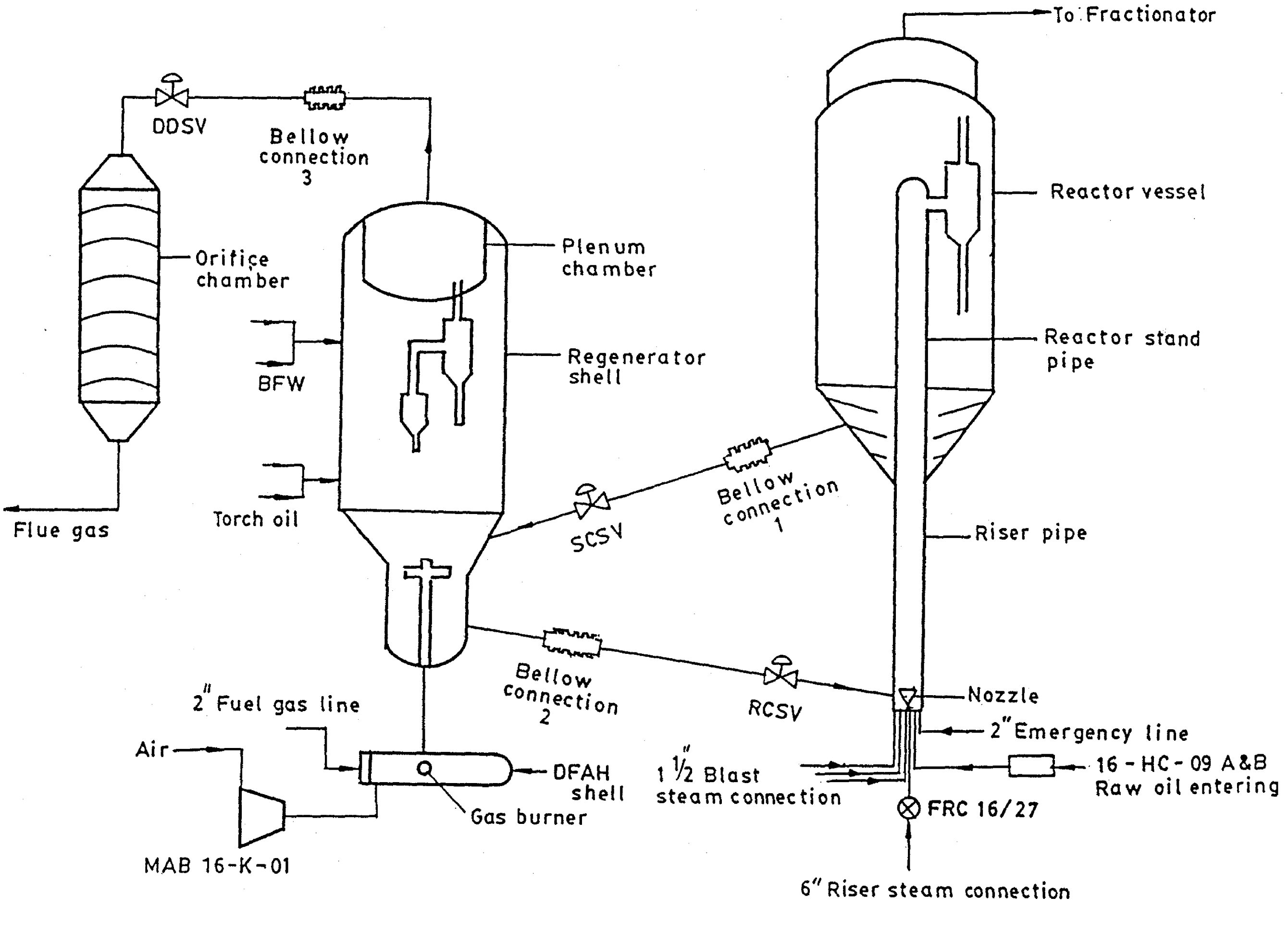
2 The maintenance optimisation problem

The Reactor-Regenerator (R_x–R_g) section of the FCCU was considered for the maintenance study. A fluid catalytic process was utilised to convert heavy gas oils into higher value lighter products by cracking in the presence of a catalyst under appropriate time, temperature and pressure conditions. The use of the catalyst promotes the cracking reaction at a lower temperature and pressure and yields products with more valuable properties than is possible with a thermal cracking process.

A simplified schematic diagram for a reactor-regenerator system for a the FCCU, showing the subsystems and components, was constructed up to a level for which reliability data are available or can be estimated (shown in Fig. 1). Accurate failure and repair data are required for a realistic system performance study. The failure rates per year and the mean time to repair of each of the components in the system, along with the fault trees of the reactor-regenerator system were taken from published data [13,14] and from the in-house plant records maintained for the company's own use. The downtime cost of the R_x–R_g system in the FCCU system is Rs. 1 lakh/hour. The critical components that cause major system shutdowns were identified by simulating a Petri net model of the FCCU [15] and also with the use of a Monte Carlo simulation model [16].

Various corrective maintenance actions on the critical components of the FCCU are proposed in the study to maximize FCCU system availability at minimum cost. In practice, these two corrective maintenance policies, viz. increasing MTBF and/or decreasing MTTR, need not be exclusive. There may be extreme policies for all

Fig. 1 The schematic diagram for the Reactor-regenerator System of the FCCU



possible levels of improvements to both the MTBF and the MTTR or some combination of the two with each policy, all of which can be implemented to a different extent. For example, the MTBF level varies with the frequency of preventive maintenance actions, and similarly, various degrees of reduction in the MTTR can be achieved with more extensive employee training, and/or employing additional maintenance workers/facilities. Maintenance managers can decide on a certain interval between preventive maintenance actions combined with a limited amount of employee training/maintenance-facilities to reduce the adverse effects of component failures. Therefore, the combination of these two distinct maintenance policies creates many possible maintenance alternatives, each associated with different levels of MTBF and MTTR.

Feasible improvements to the MTBF and/or MTTR at the 5%, 10% and 15% levels in one or more critical components are considered in this maintenance study in order to maximise FCCU system availability. In view of the number of feasible alternatives, the corrective maintenance optimisation study is restricted to consideration of +/− 5% improvements in the MTBF and MTTR. The implementation of both MTBF and MTTR policies requires a certain investment. Management can select a corrective maintenance policy based on the economic consequences of investment proposals. The objective of the study is to determine the most cost-effective corrective maintenance policy which maximises system availability.

3 The objective function

The measure of performance considered in the maintenance optimisation study in order to evaluate the various multiple corrective maintenance policies is the ratio of total maintenance cost to system availability per period. The total maintenance cost is a function of downtime cost and operating cost.

The objective function Z to determine the best corrective maintenance level is expressed as

$$Z = \min \frac{\left\{\left[\sum_{k=1}^{n} d_k\right] C_{sd} + \sum_{i=1}^{m} (B_i + R_i)\right\} T_{so}}{\left(T_{so} - \sum_{k=1}^{n} d_k\right)} \tag{1}$$

where

$n=$	number of system shutdowns per two-year period
$m=$	number of critical components under study
$d_k=$	downtime during kth shutdown in hours
$C_{sd}=$	downtime cost of the system per hour
$T_{so}=$	total system operating time in hours
$B_i=$	cost of increasing MTBF of a component i
$R_i=$	cost of decreasing MTTR of a component i

Scheduled maintenance is carried out on the FCCU system once every two years. Therefore, the system is simulated for a period of two years for a sufficient number of replications and the objective function value Z is computed using Eq. 1. For convenient presentation and comparison of the results, the best objective function value Z^* is divided by 10^3.

$$Z^* = Z/10^3 \tag{2}$$

The critical components of the R_x–R_g system of the FCCU and the standby cost details of these components are given in Table 1. The standby cost of each critical component is used to compute the cost of making improvements at different levels within the MTBF and MTTR.

The cost of increasing the MTBF of a component i is computed as,

$$B_i = \sum_{j=1}^{s} k\left(1 + P_j\right)^j \tag{3}$$

where factor

$k=$	$(0.01)S_c$,
$s=$	number of stages of improvement in MTBF
$S_c=$	standby unit cost
$P_j=$	proportion by which MTBF is increased in jth stage of improvement

The cost of decreasing the MTTR of component i follows the expression:

$$R_i = k(S_c) \tag{4}$$

for the first stage of improvement in MTTR (i.e. for 5% reduction in MTTR) where factor $k = 0.004$

$$R_i = C_{j-1} + C_{j-1}\left[1 + Q_j\right] \tag{5}$$

for the remaining stages of improvement to the MTTR. where

$C_{j-1}=$	cost of reducing MTTR in $(j-1)$th stage of improvement
$Q_j=$	proportion by which MTTR is decreased in the jth stage of improvement

3.1 The need for a search heuristic

The maintenance optimisation problem of the FCCU system that was studied involves various corrective maintenance actions on the critical components in the FCCU in order to maximise the system availability at

Table 1 Cost data of critical components

Critical component	Nozzle	RCS valve	Solenoid valve	MAB set	DDS valve	Blast valve	SCS valve
Standby cost (lakh Rs.)	20	65	35	400	50	23	60

minimum cost/effort. The various corrective maintenance alternatives considered in the study are: increase in MTBF and/or decrease in MTTR at the 0%, 5%, 10% and 15% levels, respectively, for one or more of the seven critical components in the R_x–R_g system.

A two-digit binary number is used to represent the operating status of each critical component. For example, a two-digit binary number '00' indicates that there was no increase/reduction in MTBF/MTTR, respectively; '01' indicates that there is 5% increase/reduction in MTBF/MTTR, respectively; '10' represents a 10% increase/reduction in MTBF/MTTR, respectively; '11' represents a 15% increase/reduction in MTBF/MTTR, respectively. Hence the operating status of the components in the R_x–R_g system is represented by a twenty-eight digit binary code. The first fourteen digits of the code are used to represent the level of maintenance employed to increase the MTBF out of the seven critical components considered. The fifteenth to twenty eighth digits of the code are used to represent the degree to which the MTTR of the respective critical component has decreased.

Therefore the number of possible combinations of corrective maintenance alternatives (solutions) for the seven critical components in the FCCU system is $2^{28} = 268{,}435{,}456$ solutions. This makes the search for a globally optimal solution within such a large solution range quite difficult. A large computational time would be needed when searching for quality solutions using traditional, local methods. Therefore, the use of search heuristics such as genetic algorithms—the tool most widely used to solve combinatorial optimisation problems—becomes necessary.

4 Classical search and optimisation methods

Traditional optimisation methods can be classified into two distinct groups: direct methods and gradient-based methods [17]. In direct search methods, only objective function ($f(x)$) and constraint values ($g_j(x)$, $h_k(x)$) are used to guide the search strategy, while gradient-based methods use the first and/or second-order derivatives of the objective function and/or constraints to guide the search process. Since derivative information is not used, the direct search methods are usually slow, requiring many function evaluations for convergence. For the same reason, they can also be applied to many problems without a major change to the algorithm. On the other hand, gradient-based methods quickly converge into an optimal solution but are not efficient in non-differentiable or discontinuous problems. In addition, there are some common difficulties with most of the traditional direct and gradient-based techniques such as:

- Convergence to an optimal solution depends on the chosen initial solution.
- Most algorithms tend to end with a sub-optimal solution.
- An algorithm efficient in solving one optimisation problem may not be efficient in solving a different optimisation problem.
- Algorithms are not efficient in handling problems that have discrete variables.

The above discussion suggests that traditional methods are not as good candidates for engineering design as efficient optimisation algorithms are. In the following section, a genetic algorithm technique that can alleviate some of the above difficulties is described that may constitute an efficient optimisation tool.

5 Genetic algorithms

Over the last decade, genetic algorithms (GA) have been used extensively as search and optimisation tools in various problem domains, including science, commerce, and engineering. The primary reasons for their success are their broad applicability, ease of use, and global perspective [18]. A more comprehensive description of GAs, along with other evolutionary algorithms, can be found in a handbook compiled by Baack et al. [19].

5.1 Principles

Genetic algorithms are stochastic search techniques based on the mechanism of natural selection and natural genetics. Genetic algorithms, in contrast to conventional search techniques, start with an initial set of random solutions called a population. Each individual in the population is called a chromosome, representing a solution to the problem at hand. A chromosome is a string of symbols that is usually—but not always—a binary bit string. The chromosomes evolve through successive iterations called generations. During each generation, the chromosomes are evaluated using some measures of fitness. To create the next generation, new chromosomes, called offspring, are formed by either (a) merging two chromosomes from current generation using a crossover operator or (b) modifying a chromosome using a mutation operator. A new generation is formed by (a) selecting, according to the fitness values, some of the parents and offspring and (b) rejecting others so as to keep the population size constant. Fitter chromosomes have higher probabilities of being selected. After several generations, the algorithms converge into the best chromosome, which hopefully represents an optimal or sub-optimal solution to the problem. Using $P(t)$ and $C(t)$ to represent parents and offspring in the current generation t, the general structure of genetic algorithms is described as follows:

5.1.1 Procedure: Genetic Algorithms

```
Begin
  – $t \leftarrow 0$;
  – initialise $P(t)$;
  – evaluate $P(t)$;
     – *while* (not termination condition) *do*
     – recombine $P$(t) to yield $C(t)$;
     – evaluate $C(t)$;
     – select $P(t+1)$ from $P(t)$ and $C(t)$;
     – $t \leftarrow t+1$;
  – *end*
*end*
```

5.2 Fitness function

GA mimics the survival-of-the-fittest principle of nature to create the search process. Therefore, a GA is naturally suited to solving maximisation problems. Minimisation problems are usually converted into maximisation problems using an appropriate conversion. In general, a fitness function F(x) is first derived from the objective function and then used in successive genetic operations. For maximisation problems, the fitness function can be considered to be the same as the objective function or F(x) = Z. For minimisation problems, the fitness function is an equivalent maximisation problem chosen so that the optimum point remains unchanged. A number of transformations are possible. The following fitness function is often used:

$$F(x) = 1/(1+Z) \tag{6}$$

where Z = objective function value.

This conversion does not alter the location of the minimum, but converts a minimisation problem to an equivalent maximisation problem. The fitness function value of a chromosome is known as its fitness.

5.3 Genetic operators

The three genetic operators are described as follows:

The reproduction operator is applied to emphasise good solutions and eliminate bad solutions in a population, while keeping the population size constant. This is achieved by identifying good (usually above-average) solutions in a population, making multiple copies of good solutions, and eliminating bad solutions from the population so that multiple copies of good solutions can be placed in the population.

Reproduction selects good chromosomes in a population and forms a mating pool. The commonly used reproduction operator is the proportionate reproduction operator where a chromosome is selected for the mating pool with a probability that is proportional to its fitness. One way to implement this selection scheme is to imagine a roulette-wheel with its circumference marked for each chromosome proportionate to the chromosome's fitness. The roulette-wheel is spun *p_siz* times, and one instance of the chromosome is selected by the roulette-wheel pointer for each spin. Since the circumference of the wheel is marked according to a chromosome's fitness, this roulette-wheel mechanism is expected to make $F_i/\bar{F}$ copies of the ith chromosome in the mating pool. The average fitness of the population is calculated as

$$\bar{F} = \sum_{i=1}^{n} Fi/n \tag{7}$$

Crossover is the main genetic operator. It operates on two chromosomes at a time and generates offspring by combining both chromosomes' features. Standard crossover (usually known as one-point crossover) of two parents involves selecting a point on the chromosome and then copying that part of the chromosome before the crossover point from the first parent, and then the part of the chromosome after the crossover from the second parent. If two children are to be produced, the roles of the parents are reversed for the second child. For example, given two parents:

Parent_1	10110|110
Parent_2	00111|001

And a crossover point after bit 5, we get children:

Child_1	10110|001
Child_2	00111|110

This method works well with a bit string representation. The performance of genetic algorithms depends to a great extent on the performance of the crossover operator used. The crossover rate (denoted by p_c) is defined as the ratio of the number of offspring produced in each generation to the population size (*p_siz*). This ratio controls the expected number $p_c \times p\_siz$ of chromosomes that undergoes the crossover operation. A higher crossover rate allows exploration of more of the solution space and reduces the chances of settling on a false optimum; but if this rate is too high, it results in the wastage of a lot of computation time in exploring unpromising regions of the solution space.

Mutation is a background operator that produces spontaneous random changes in various chromosomes. A simple way to achieve mutation would be to alter one or more genes. Mutation usually involves flipping a single bit. For example mutating bit 4 in child_2 will result in:

Child_2(mutated)   00101|110

In genetic algorithms, mutation serves the crucial role of either (a) replacing the genes lost from the population during the selection process so that they can be tried in a new context or (b) providing the genes that were not present in the initial population. The mutation rate (denoted by p_m) is defined as the percentage of the total number of genes subjected to mutation in the population. The mutation rate controls the rate at which new genes are introduced into the population for trial. If it is too low, many genes that would have been useful are never tried out; but if it is too high, there will be a large amount of random perturbation, the offspring will start losing their resemblance to the parents, and the algorithm will lose the ability to learn from the history of the search.

5.4 Differences between GA and conventional optimisation techniques

Genetic algorithms differ from conventional optimisation and search procedures in several fundamental ways. The differences are described in the following paragraphs.

GA works with the coding of a solution set, not with the solutions themselves. Binary GA works with a discrete search space, even though the function may be continuous. On the other hand, since function values at various discrete solutions are required, a discrete or discontinuous function may be tackled using a GA [19]. This allows a GA to be applied to a wide variety of problem domains. The other advantage is that GA operators exploit the similarities in string-structures to create an effective search.

The striking difference between a GA and most of the traditional optimisation methods is that a GA works with a population of solutions instead of a single solution. Most classic optimisation methods generate a deterministic sequence of computation based on the gradient or higher-order derivatives of an objective function. The methods are applied to a single point within the search space. The point is then gradually improved along the deepest descending/ascending direction through iterations. This point-to-point approach has the danger of decreasing in local optima. A GA performs a multiple directional search by maintaining a population of potential solutions. The population-to-population approach attempts to make the search avoid local optima.

The other difference is that a GA uses probabilistic transition rules, as opposed to deterministic rules, to guide the search. The basic problem with most traditional methods is that there are fixed transition rules to move from one solution to another. A GA, on the other hand, uses probabilistic rules and an initial random population. Thus, early on, the search may proceed in any direction and no major decision is made in the beginning. Later on, when the population has converged on some locations, the search direction narrows and a near-optimal solution is found. This characteristic of GAs also allows them to be applied to a wide class of problems. giving them a robustness that is very useful in solving a variety of optimisation problems.

Every good optimisation method needs to balance the extent of the exploration of the information obtained up to the current time with the extent of exploitation of the search space required to obtain new and better solution(s). Most traditional methods have fixed rules and hence have fixed amounts of exploration and exploitation. In contrast, the exploitation and exploration aspects of GA can be controlled almost independently. This provides a lot of flexibility in GA design.

6 Optimal maintenance policy—use of GA

The development of a genetic algorithm to solve a particular problem involves two types of decisions. The first concerns the way in which the problem is to be modelled to fit into the genetic algorithm framework and includes the definition of the range of feasible solutions, the form of the fitness function and the way in which individuals are to be represented as chromosomes. The second concerns the parameters of the genetic algorithm itself and includes the proportions of the population to be produced as a result of reproduction, crossover and mutation, selection procedure, population size, number of generations, and a number of other decisions concerning variants of the basic algorithm.

6.1 Design of chromosomes

In a genetic algorithm it is necessary to code the parameters so as to perform the genetic operation. The length of the chromosome depends upon the number of critical components considered in the study and the number of various levels of maintenance required on these components to achieve maximum system availability. The chromosome design with respect to the reactor-regenerator system of the FCCU with seven critical components and three stages of improvement, both on MTTR and MTBF, is shown in Table 2.

In view of the number of feasible alternatives, the search is restricted to the consideration of +/−5% improvements in the MTBF/MTTR. A twenty eight-digit binary code was developed to represent the operating status of the critical components in the reactor-regenerator system. The first fourteen-digits of the code are used to represent the level of maintenance action employed to increase the MTBF of the component. A set of two-digit binary numbers is used to represent the degree to which the MTBF of the respective component is increased. For example, the two-digit binary number ‘00’ indicates that there is no increase in the MTBF; ‘01’ indicates that there is 5% increase in the MTBF; ‘10’ indicates that there is 10%

Table 2 GA chromosome design for Reactor-Regenerator system

String production	01–14	15–28
Content	Mb_i	Mr_I

Mb_i—percent increase, with respect to the current operational level, in MTBF of component i
Mr_i—percent decrease, with respect to the current operational level, in MTTR of component i

increase and '11' indicates that there is 15% increase in the MTBF of the critical component.

In the same manner, another set of two-digit binary numbers, each from the fifteenth to twenty eighth digit of the code, is used to represent the degree to which the MTTR of the respective component has decreased. For example, the two-digit binary number '00' indicates that there is no reduction in the MTTR; '01' indicates that there is 5% decrease in the MTTR; '10' indicates that there is 10% decrease and '11' indicates that there is a 15% reduction in the MTTR.

6.2 GA parameter setting using orthogonal experiments

Genetic algorithm parameters, namely population size *p_siz*, number of generations *n_gen*, crossover rate c_p and mutation rate m_p, play a vital role in finding the solution to the given problem. Orthogonal experiments are used to determine suitable values for these GA parameters. Initially the parameters, viz. *p_siz*, *n_gen*, c_p and m_p are assigned the values 90, 20, 0.6 and 0.3, respectively. These values are assumed to be the mid-values to be considered in the 3-level orthogonal experiments. A set of values for each parameter with a certain percentage deviation from the mid-values is then considered. Based on the observations made on the quality of solutions obtained with this set of values during the pilot runs conducted on the GA, a lower limit and an upper limit value for each parameter are selected. The GA parameters and the levels considered in the study are given in Table 3.

An L_9 orthogonal array (OA) is used in the design of experiments [20]. The objective function values, Z, obtained during the experiments for a given maintenance effort, are presented in Table 4.

H_0: No significant difference in the objective function value Z due to changes in GA parameter values.

H_1: Significant difference in the objective function value Z due to changes in GA parameter values.

The ANOVA of L_9 OA for GA parameters is given in Table 5.

Based on ANOVA, the following values are set for the GA parameters:

- Population size, $p\_siz = 60$
- Number of generations, $n\_gen = 15$
- Mutation rate, $m_p = 0.1$
- Crossover rate, $c_p = 0.9$

Table 3 Genetic algorithm parameters and level settings

GA parameter	Level		
	1	2	3
Population size, *p_siz* (A)	60	90	120
No. of generations, *n_gen* (B)	15	20	25
Mutation rate, m_p (C)	0.1	0.3	0.5
Crossover rate, c_p (D)	0.3	0.6	0.9

Table 4 GA–Orthogonal experiment settings and objective function values

Sl No.	Parameter settings				Objective function value, Z	
	A	B	C	D	Replication 1	Replication 2
1	1	1	1	1	31115.86	31417.40
2	1	2	2	2	32043.81	34871.16
3	1	3	3	3	32607.17	33580.24
4	2	1	2	3	30017.34	31836.37
5	2	2	3	1	31195.38	30717.85
6	2	3	1	2	33046.04	35793.62
7	3	1	3	2	33103.90	32108.17
8	3	2	1	3	30673.25	31263.98
9	3	3	2	1	32704.92	33041.39

Table 5 ANOVA of L_9 OA for GA parameters

Source	Sum of square (SS)	v	$V = SS/v$	F_{cal}	F_{table} at $\alpha = 0.10$, $v_1 = 2$ & $v_2 = 9$
A	9.31493×10^5	2	4.657465×10^5	0.3886	3.01
B	125.76995×10^5	2	62.884975×10^5	5.2474	
C	1.60979×10^5	2	0.804895×10^5	0.0672	
D	131.59292×10^5	2	65.79646×10^5	5.490	
Error	107.85642×10^5	9	11.98405×10^5		
Total	376.14401×10^5	17			

6.3 The search for the best corrective maintenance policy

The genetic algorithm uses the 'FCCU simulation model'. Initially, for each population with randomly generated chromosomes, the FCCU simulation model is executed to determine the initial population objective function values. Then the simulation is performed and the value of objective function is determined for every generation with the current value of the chromosome.

In this study, the roulette-wheel selection procedure is used for reproduction. A simple crossover that swaps the pair of chromosome bits at the crossover point is used. Mutation is done by inverting the bit at the mutation point.

6.4 Notations and terminology

p_siz	population size
n_gen	total number of generations
c_p	crossover rate
m_p	mutation rate
L_c	length of chromosome
R_{no}	uniform random number between 0 and 1
bin_{jk}	binary code (either 0 or 1) of bit k in population j ($j \in p\_siz$)
ch_{ij}	chromosome vector of generation i and population j ($i \in n\_gen$) ($j \in p\_siz$)
Z_{ij}	objective function value for generation i and population j ($i \in n\_gen$) ($j \in p\_siz$)

ch^*	best maintenance effort obtained so far
Z^*	best objective function value obtained so far.

6.5 Proposed genetic algorithm

```
Input $p\_siz$, $n\_gen$, $c_p$, $m_p$, and $L_c$/Generate initial population.generation $i=0$/create chromosome $ch_{ij}$. For population $j=1$ to $p\_siz$
  – For bit $k=1$ to $L_c$
  – generate $R_{no} \in$ U[0,1]
  – if ($R_{no} > 0.5$)
  – then
    – $bin_{jk}=1$
  – else
    – $bin_{jk}=0$
  – Next $k$
Next $j$ For population $j=1$ to $p\_siz$
  – invoke FCCU simulation model.
  – compute $Z_{ij}$.
Next $j$ Determine $Z^*$ and store the corresponding $ch_{ij}$ in
  $ch^*$. For generation $i=1$ to $n\_gen$
  – reproduce chromosomes using Roulette wheel
    procedure.
  – /To perform crossover operation
  – For population $j=1$ to $p\_siz-1$ step 2
    – consider the chromosome pair $ch_{ij}$ and $ch_{ij+1}$
    – generate $R_{no} \in$ U[0,1]
    – if ($R_{no} \le c_p$)
    – then
    – {
    – determine crossover site at random within $ch_{ij}$
    – perform crossover
    – }
  – Next $j$
  – /To perform mutation operation
  – For population $j=1$ to $p\_siz$
    – For bit $k=1$ to $L_c$
    – generate $R_{no} \in$ U[0,1]
    – if ($R_{no} \le m_p$)
    – then
    – {
    – if ($bin_{jk}=1$)
    – then
    – $bin_{jk}=0$
    – else
    – $bin_{jk}=1$
    – }
    – Next $k$
    – Next $j$
  – For population $j=1$ to $p\_siz$
    – {
    – decode the chromosome $ch_{ij}$.
    – call FCCU simulation model.
    – perform simulation with the value decoded from
      $ch_ij$.
    – compute $Z_{ij}$.
    – update $Z^*$ and the corresponding $ch^*$.
    – }
    – Next $j$
Next $i$ Report $Z^*$ and $ch^*$.
```

7 FCCU simulation model

The objective function value Z is calculated using a simulation of the reactor-regenerator system of the FCCU for a period of two years, since planned shutdown of the system is done every two years. The following assumptions are made in the FCCU simulation model.

- Time between failures follows exponential distribution.
- One repair crew is available.
- Repair times follow exponential distribution.
- Once a repair action begins on a component, it is completed.
- If a failed component causes the system failure, then that component is taken for repair immediately, pre-empting repair on other components.
- Repair can also be by replacement.

7.1 The simulation logic

Initially, the values of the system parameters viz. MTBF and MTTR of the non-critical components and the improved MTBF and MTTR of the critical components considered in the study are determined. When a component fails that does not cause system shutdown, then the failed component joins the queue if the repair crew is busy and undergoes on-line repair if the crew is free. If a failed component causes the system to shut down, then that component is taken for repair immediately, pre-empting repair on other components. The system is restarted after that component is repaired. Since planned shutdown is performed for the FCCU every two years, simulation is terminated after the simulation time reaches two years. After a planned shutdown, the system becomes as good as new.

7.2 Setting the run length of the FCCU simulation experiment

The run length of the FCCU simulation experiment used to compute Z, was varied for a given combination of maintenance actions on the critical components and the results are given in Table 6.

H_0:	No significant difference in objective function value Z due to changes in run length
H_1:	Significant difference in objective function value Z due to changes in run length.

ANOVA was conducted with the Z values and the ANOVA for run length is given in Table 7.

Table 6 Objective function value for various run lengths

Run length	Objective function value, Z			
	Replication 1	Replication 2	Replication 3	Replication 4
40	42609.91	43207.13	42821.66	42117.86
50	42354.54	43219.78	45182.82	42174.36
60	42871.57	42266.41	42719.74	43955.26
70	44354.72	42544.64	42203.15	43628.62
80	43694.34	42613.51	43862.75	43135.67

Table 7 ANOVA for run length

Source of variation	Sum of squares	Degrees of freedom	Mean square	F
Run length	10.542657×10^5	4	2.635664×10^5	0.3363
Error	117.56623×10^5	15	7.837749×10^5	
Total	128.10889×10^5	19		

Since the value obtained for F does not exceed 3.06, the value of $F_{0.05}$, with 4 and 15 degrees of freedom and a null hypothesis H_0 is accepted at the 0.05 level of significance. Therefore the run length of the FCCU simulation experiment is fixed at 40. An initial ANOVA conducted with the Z values for less than 40 runs showed that there was a significant difference in the objective function value due to changes in run length.

8 Results and discussion

In this paper, a genetic algorithm was used to solve an FCCU's corrective maintenance optimisation problem. An orthogonal experiment was conducted to determine GA parameter values. The genetic algorithm was implemented for the maintenance problem with: population size = 60, number of generations = 15, crossover rate = 0.90 and mutation rate = 0.10. The genetic algorithm has been coded in C and executed on a 166 MHz personal computer.

The ratio of the total maintenance cost to system availability per period was been considered as the measure of performance. The results of the genetic algorithms implemented for the maintenance problem, with the objective of obtaining the best corrective maintenance policy to optimise the system performance, are shown in Fig. 2 and Fig. 3. The search pattern of the objective function values for the number of generations is shown in Fig. 2. The convergence graph is shown in Fig. 3. From Fig. 2 and Fig. 3, it is clear that the minimum measure of performance is obtained at the 7th generation (i.e. 7×60 iterations), after which there is no improvement in the quality of the solution. The genetic algorithm results are given in Table 8.

Table 8 shows that the best solution is obtained in 47 CPU time units. The system availability can be maximised to 0.982708 with a minimum total maintenance cost of Rs. 31009.3654×10^3 per period, if the corrective maintenance actions suggested in Table 9 are carried out.

9 Conclusion

Genetic algorithms work with populations of solutions and attempt to guide the search toward improvement, using a survival-of-the-fittest principle. In this paper, the use of genetic algorithms to solve the combinatorial maintenance optimisation problem of an FCCU reactor-regenerator system was described. The application of a GA to obtain the best corrective maintenance strategy to

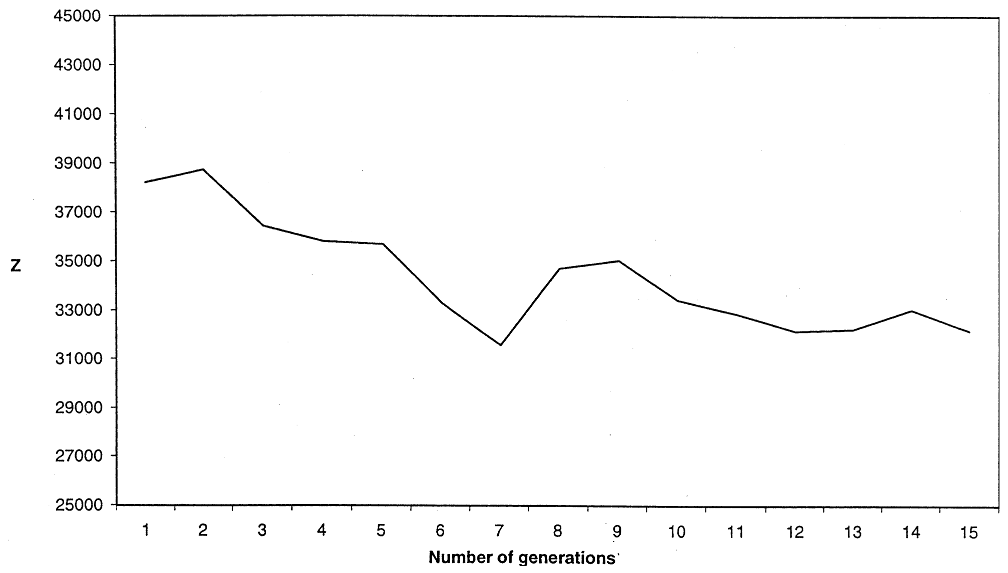


Fig. 2 Global search—Genetic Algorithm

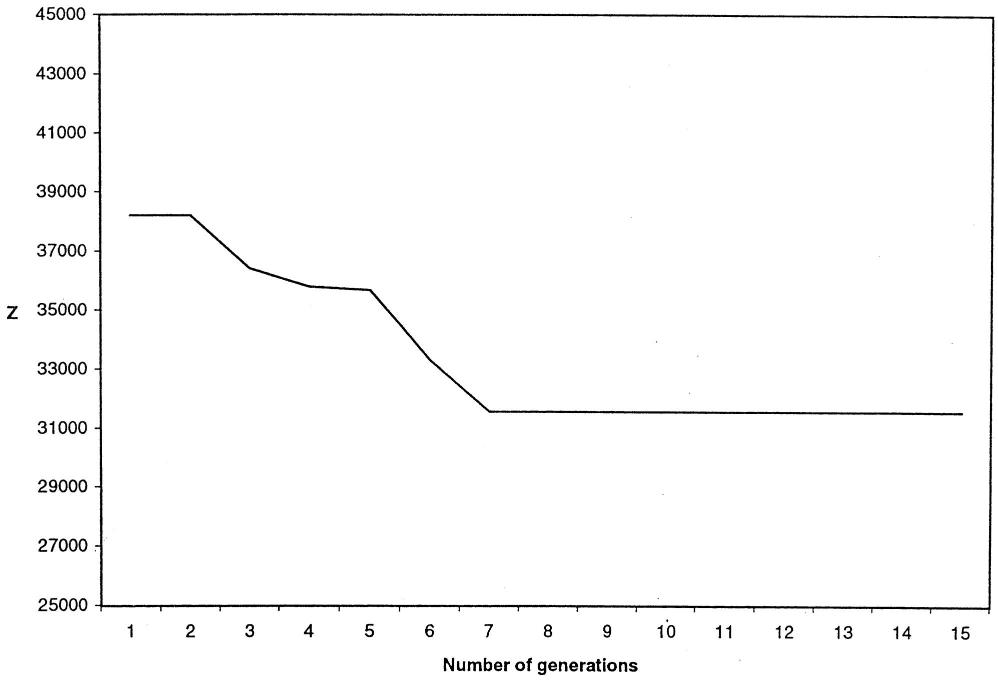


Fig. 3 Convergence graph —Genetic Algorithm

Table 8 Genetic algorithm results

Heuristic	Total cost (Rs.×10^3)	Maximum availability	Z*	Iteration number	CPU (time units)
Genetic algorithm	31009.3654	0.982708	31555.02	(7×60) = 420	47

Table 9 Proposed corrective maintenance strategies

Compo-nent	*Nozzle*		*RCSV*		Solenoid check valve		*MAB set*		*DDSV*		*Blast valve*		*SCSV*	
Heuristic	MTBF	MTTR	MTBF	MTTR	MTBF	MTTR	MTBF	MTTR	MTBF	MTTR	MTBF	MTTR	MTBF	MTTR
GA	5	10	–	15	5	–	–	5	–	15	15	–	10	5

The entries indicate the relative percentage improvement (in MTBF and MTTR) with respect to the current level of operations

optimise the system performance of the FCCU was demonstrated. The best corrective maintenance policy to be recommended to the maintenance manager to achieve maximum system performance was then determined.

References

1. Rowan DA (1989) On-line expert systems in the process industries. AI Expert 30–38
2. Puigjaner P, Espuna A (1991) Computer-oriented process engineering. Elsevier, Amsterdam
3. Dekker R (1996) Applications of maintenance optimisation models: A review and analysis. Reliab Eng 51:229–240
4. Al-Bahi AM (1993) Spare provisioning policy based on maximisation of availability per cost ratio. Comput Ind E 24:81–90
5. Harunuzzaman M, Aldemir T (1996) Optimisation of standby safety system maintenance schedules in nuclear power plants. Nucl Tech 113:354–367
6. Bandyopadhyay T, Dey PK, Gupta SS (1997) Cost-effective maintenance program through risk analysis. AACE International Transactions of the Annual Meeting, AACE Inc., Morgantown, WV, U.S.A. pp 6–9
7. Sridhar J, Rajendran C (1994) A genetic algorithm for family and job scheduling in a flow line-based manufacturing cell. Com Ind Eng 27:472–496
8. Sridhar J, Rajendran C (1996) Scheduling in flowshop and cellular manufacturing system with multiple objectives—A genetic algorithmic approach. Prod Plan C 7:374–382
9. Shahabudeen P, Krishnaiah K (1999) Design of a bi-criteria kanban system using genetic algorithms. Int J M Sys 15:257–274
10. Coit DW, Smith AE (1996 a) Reliability optimisation of series-parallel systems using a genetic algorithm. IEEE Reliab 45:254–263
11. Coit DW, Smith AE (1996 b) Solving the redundancy allocation problem using a combined neural network genetic algorithm approach. Comput Oper 23: 515–526
12. Ramachandran V, Kannan J, Sathiyanarayanan K, Sivakumar V (1997) Optimal replacement strategies—Genetic algorithms approach Microel Rel 37:665-667

13. Davidson J (1988) The reliability of mechanical system. Mechanical Engineering Publication Ltd., The Institution of Mechanical Engineers, London
14. Thangamani G, Narendran TT, Subramanian R (1995) Assessment of availability of a Fluid Catalytic Cracking Unit through simulation. Reliab Eng 47:207–220
15. Robert TP, Jabyabalan V, Vijayalakshmi K (1999) Petri-net based abnormal event identification—A place array approach. Stochastic Processes and their Applications, Narosa, New Delhi, pp 390–398
16. Robert TP, Jayabalan V (1999) Fault identification in a Fluid Catalytic Cracking Unit through simulation. Operations and Quantitative Management in the Global Business Environment. Tata McGraw-Hill, New Delhi, pp 388–392
17. Deb K (1995) Optimisation for engineering design: Algorithms and examples. Prentice-Hall, New Delhi
18. Goldberg DE (1989) Genetic algorithms in search, optimisation, and machine learning. Addison-Wesley, New York
19. Baack T, Fogel D, Michalewicz Z (1997) Handbook of evolutionary computation. Oxford University Press, New York
20. Ross PJ (1989) Taguchi techniques for Quality Engineering. McGraw-Hill, New York