Chapter 28
A Novel Way to Schedule Flexible Manufacturing System

Srushti Bhatt, M. B. Kiran, and Jeetendra A. Vadher

Abstract Modeling and scheduling of flexible manufacturing system problems are yet to be addressed simply to compete in the global manufacture market. Optimizing the performance measure of the available resources of manufacturing systems is a key requirement. Many researchers have proposed techniques for solutions like mathematical programming, heuristic dispatching methods, and artificial intelligence and knowledge base system. In the present work, a combined approach of the Timed Petri Net and genetic algorithm is used for the modeling power and optimization capabilities for scheduling flexible manufacturing system. Petrinet plays a vital role in modeling FMS. The power of Timed Petrinet is used for modeling FMS by using the advantage of its ability to model a complex system with efficient net structure and chromosomal representation along with genetic algorithm for optimal solution needed in scheduling a manufacturing problem. The result obtained by this approach concludes that the proposed method performs much better than the existing methods. The algorithm is developed for minimizing makespan and the simulation run for better performance measures.

Keywords FMS · GA · Petri Nets · SPN

S. Bhatt (✉)
Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat 382007, India
e-mail: bhattsrushti12@gmail.com

M. B. Kiran
Department of Industrial Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat 382007, India

J. A. Vadher
Department of Mechanical Engineering, Government Engineering College, Palanpur, Gujarat 385001, India

A. Sachdeva et al. (eds.), *Operations Management and Systems Engineering*,
Lecture Notes on Multidisciplinary Industrial Engineering,
https://doi.org/10.1007/978-981-15-6017-0_28

Nomenclature

FMS Flexible manufacturing system
GA Genetic algorithm
PN Petri nets
TPN Timed Petri nets

28.1 Introduction

In the present market scenario, to stand in a market it is very important for a manufacturing system to accommodate changes quickly as customer demand changes. Many Indian industries have adopted FMS for in time production and quality of the product, but still problems in scheduling still arise. The scheduling problem is nothing but simply a process of resource allocation to accomplish a specified task [1].

Petri nets are a class of modeling tools, which were originated by Petri [2, 3], they have a well-defined mathematical foundation and easy to understand the graphical feature. It is a powerful graphical tool for modeling and analyzing concurrent, parallel, simultaneous, synchronous, distributed, and resource sharing systems with the advantages like an easy visualization of complex systems can model a system hierarchically (a top-down fashion at various levels of abstraction and detail) and can analyze qualitative and quantitative aspects of the system.

The common definition of PNs introduced by Petri is as follows. Petri nets or place/transition net can be defined as a five-tuple: PN $= (P, T, I, O, Mo)$, where P and T are finite non-empty sets of places pictured by circles and transitions pictured by bars, respectively. I: $P \times T \rightarrow \{0, 1\}$ is an input function that defines the set of directed arcs from P to T. O: $P \times T \rightarrow \{0, 1)$ is an output places which are drawn as circles represent possible states or conditions of the system while transition, which is shown in bars or boxes, describe events that may modify the system states. The relationships between places and transitions are represented by a set of arcs that are the only connectors between a place and a transition in either direction. The dynamic behavior of the system can be represented using tokens which graphically appear as black dots in places. A transition can only fire if it is enabled and having one token in that place I to the next transition j. A timed Petri net is associated with places or transitions. Here we consider the time associated with places. The modeling of the problem data takes place using TPN. TPN is a 7 tuple, TPN $= (P, T, I, O, M0, Mr0,$) where $M0$ is initial marking, $Mr0$ is initial vector for remaining processes, is set of time delays associated with it. The problem data consist of job-based data, operation-based data, and machine data including processing time or operation time along with the path of the operation for the given jobs. Different researchers have used different approaches of Petri nets for simulation and GA for fine-tuning the parameters [2]. Different researchers have identified different criteria for solving FMS scheduling problem. One of them is performance measures. The list of performance criteria are

Table 28.1 Criteria for performance parameters (Reproduced from Filho et al. [4])

S. No.	Criteria	Code
1	Idle time	T idle
2	Length of the AGVs route	Route length
3	Number of backtrackings of each AGV	Backtrackings
4	Total flow time	F
5	Mean flow time	F medium
6	Maximum lateness	$L_{\max}$
7	Makespan	$C_{\max}$
8	Tardiness	T
9	Maximum tardiness	$T_{\max}$
10	Due date	D_{d}
11	Cost for tardiness and earliness; production cost, penalty cost	Cost
12	Throughput	T
13	Work in process	WIP
14	Machine utilization	U
15	Maximum utilization of the machines	$U_{\max}$

as stated in Table 28.1. Here, out of all performance parameters attempt is made to address the makespan using a hybrid approach combining Petri nets and GA.

28.2 Problem Definition

The makespan minimization problem of a flexible manufacturing system (FMS) has been recognized as one of the most important planning problems. Job scheduling problems are referred to as NP-hard problems. And to solve such kind of problem there are different four types of representation. They are job-based representation, operation-based representation, priority rule-based, and preference-list-based representation [1].

In this research, a Genetic Algorithm (GA) based heuristic is proposed to solve the makespan of a random type FMS. The objective of the problems is to minimize the system unbalance and maximize the throughput, satisfying the technological constraints such as availability of machining time, and tool slots. The proposed GA-based heuristic determines the part type sequence and the operation-machine allocation that guarantee the optimal solution to the problem, rather than using fixed predetermined part sequencing rules.

The first objective of the research work is to schedule the given FMS, modeling with Petri nets and optimizing using genetic algorithm and obtaining results to conclude with the best option. The makespan minimization problem consists of three types Job shop scheduling (there are n jobs and m identical stations. Each job should

Table 28.2 Problem data

Job	Operation time			Operation path		
	M/C 1	M/C 2	M/C 3	M/C 1	M/C 2	M/C 3
1	4	3	2	1	2	3
2	1	4	4	2	1	3
3	3	2	3	3	2	1
4	3	3	1	2	3	1

be executed on a single machine), Open shop scheduling (there are n jobs and m different stations. Each job should spend some time at each station, in a free order), Flow shop scheduling (there are n jobs and m different stations. Each job should spend some time at each station, in a predetermined order). To optimize makespan the system consists of the following:

The system considered is

n = four jobs and m = three machines with the respective operating time. Each job is processed on every machine at the appropriate time.
The machine can operate only one operation at a time.
All machines are available.
Here one job will leave the machine only after completing the operation and scheduling is done in the way that no classes with the same job will assign to two different machines at the same time or vice versa.
Machine setup time is included in the processing time.
The data of n jobs and m machines are as given in Table 28.2.

28.3 Methodology

For the given data will be evaluated into two phases. The first phase gives the system model and the second gives the optimum result.

Step 1: Creating a system model using Petri net tool using MATLAB Environment
Step 2: Identifying the variables (Table 28.3)
Step 3: Simulating the model in the MATLAB environment. Results of system model simulation.

Figures 28.1 and 28.2 shows the FMS modeling for the given 4 job 3 machine problem. Once the modeling is created next phase is to check for the structural and behavioral properties of the net structure. As shown in Fig. 28.1 the model of each job should spend some time at each station, in a predetermined order. Figure 28.3 shows the property check whether is net structure modeled satisfy all the condition of the manufacturing system. Net is bounded means there will be no overflows in the buffer. Liveness shows a complete absence of deadlock in the manufacturing system. Once the property of the net is checked, next step is to analyze the system. There

Table 28.3 Description of timed Petri net model in Fig. 28.2

P1	Job 1 in queue
P2	Operation 2 of Job 1 on machine 2
P3	Operation 3 of Job 1 on machine 3
P4	Job 1 finish
P5	Operation 1 of Job 2 on machine 2
P6	Operation 2 of Job 2 on machine 1
P7	Operation 3 of Job 2 on machine 3
P8	Job 2 finish
P9	Operation 1 of Job 3 on machine 3
P10	Operation 2 of Job 3 on machine 2
P11	Operation 3 of Job 3 on machine 1
P12	Job 3 finish
P13	Operation 1 of Job 4 on machine 2
P14	Operation 2 of Job 4 on machine 3
P15	Operation 3 of Job 4 on machine 1
P16	Job 4 finish
P17	Loop operation 1 for Job 1
P18	Loop operation 2 for Job 1
P19	Loop operation 3 for Job 1
P20	Loop operation 1 for Job 2
P21	Loop operation 2 for Job 2
P22	Loop operation 3 for Job 2
P23	Loop operation 1 for Job 3
P24	Loop operation 2 for Job 3
P25	Loop operation 3 for Job 3
P26	Loop operation 1 for Job 4
P27	Loop operation 2 for Job 4
P28	Loop operation 3 for Job 4

are different ways of analyzing the net-like coverability tree, incidence matrix, and simple reduction rules. Here the net is analyzed using matrix equation that governs the dynamic behavior of the concurrent system as shown in Figs. 28.4, 28.5, and 28.6.

Step 4: Optimizing using GA

GA starts with a set of solutions called population. Chromosomes from the population are occupied and used to form a new population. This is motivated by the desire, that the new population will be better than the old one in terms of a fitness criterion. Solutions that are selected to form new solutions (offspring) are selected according to their fitness—the more suitable they are the more chances they must reproduce. There

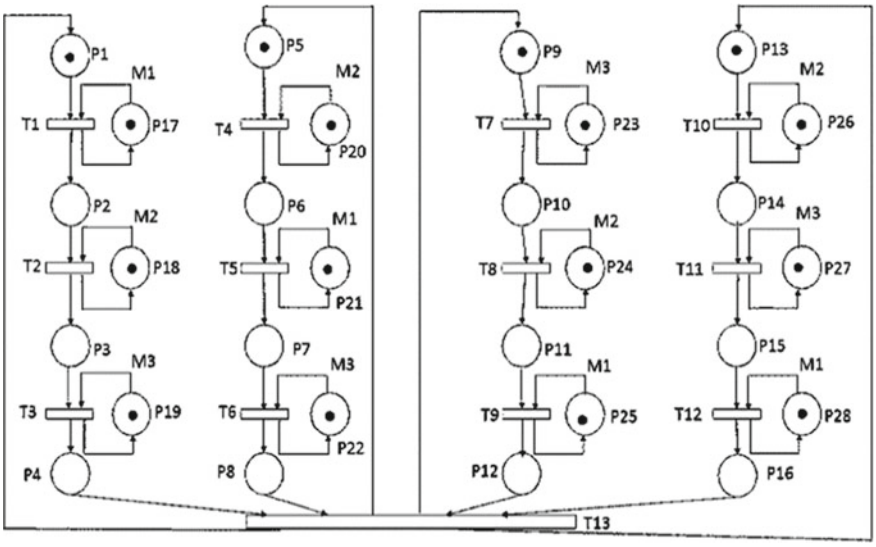

Fig. 28.1 FMS processing of 4 jobs & 3 M/c

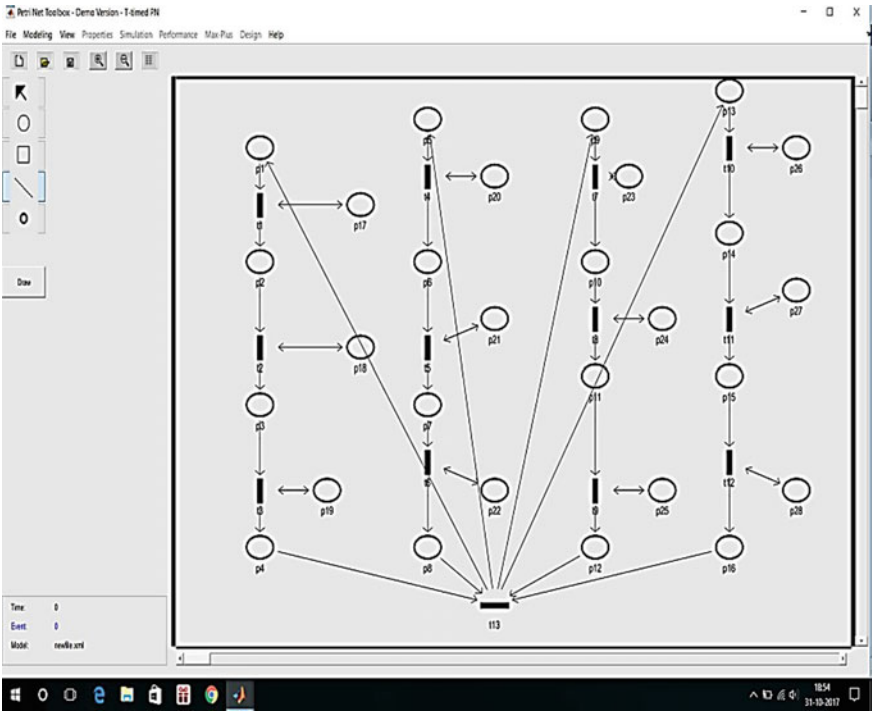

Fig. 28.2 FMS processing of 4 jobs & 3 M/c in MATLAB

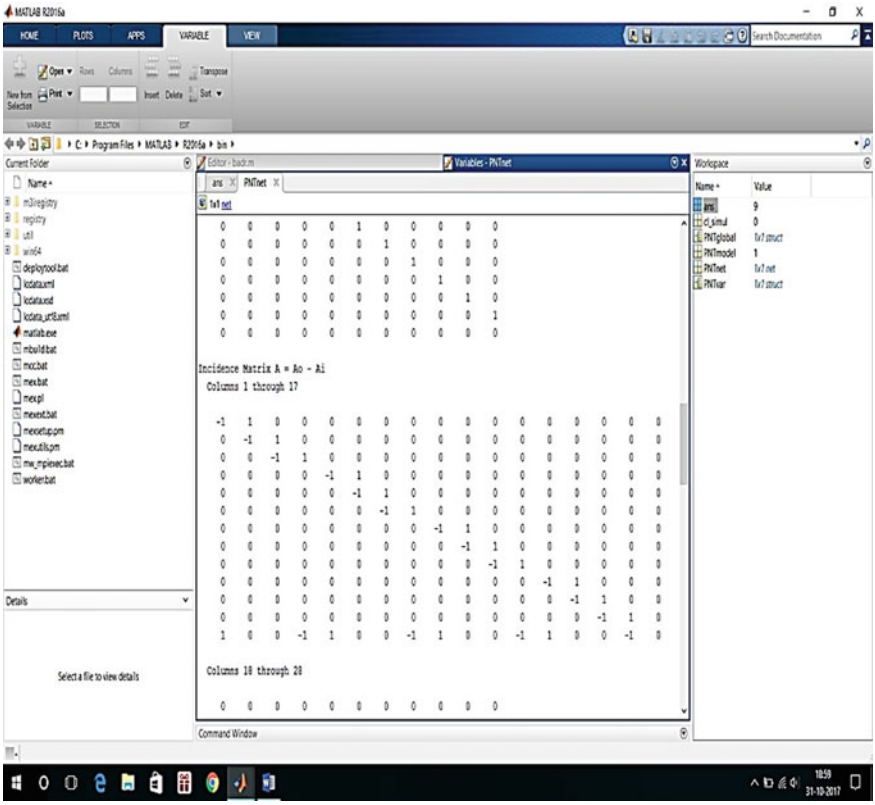

Fig. 28.3 Structural properties

are different representation schemes for evaluation with the makespan objective. They are operation-based representation, job-based representation, preference-list-based representation, priority rule-based representation. In general, GA uses three steps—selection, crossover, and mutation (4). Selection based on the fitness (makespan in our case) is the source of exploitation, and crossover and mutation help us to promote exploration (Fig. 28.7).

A generation of a GA contains a population of individuals, each of which corresponds to a possible solution in the search space. Everyone in the population is evaluated with a fitness function to produce a value which indicates the goodness of a solution. Selection helps in bringing forward certain members from the population to apply crossover and mutation on the given set of problem. Crossover takes pairs of individuals and uses parts of each to produce new individuals. Random mutations swap parts of an individual to prevent the GA from getting caught in a local minimum.

Step 4.1: Creation of First Generation/Initialization:

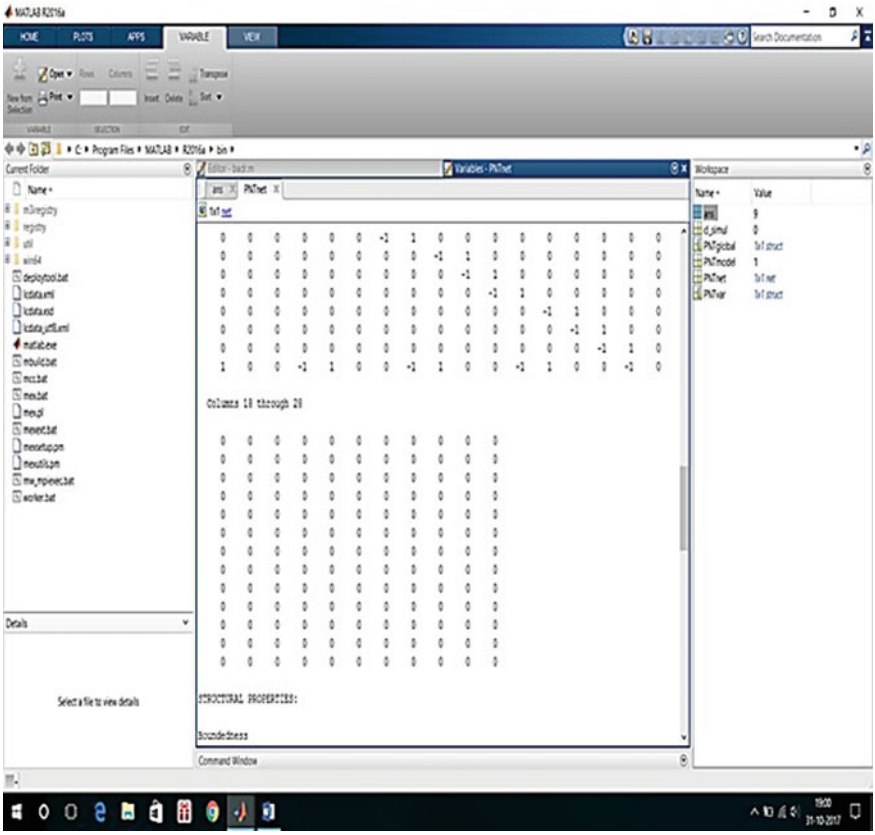

Fig. 28.4 Incidence matrix part 1

In our problem, we are using the datasets that have been taken from Kumar [6]. This data set has initially four jobs and three machines. The term makespan refers to the cumulative time to complete all the operations on all machines [7]. It is the time taken from scheduling the first job submitted until the completion of the last job. The objective of the problem is to find a valid schedule that yields the minimum makespan. The initial population can be generated randomly. Each machine has randomly 3–4 jobs placed on them. The total running time for each machine is then calculated. Here the job-based representation is focused out of four. For the same, the chromosome would be [1 2 0 3]. The first job to be processed is j_1 is m_1, m_2, m_3, and processing time is 4, 3 2 [1]. Similarly, each is processed. The genetic search process starts with a randomly generated set of chromosomes called the initial population. The size of the population (pop_size) depends on the solution space. The chromosome representation of all the representation is as stated in Table 28.4. The outline GA code is as shown in annexure I. The simple GA structure used to evaluate the system

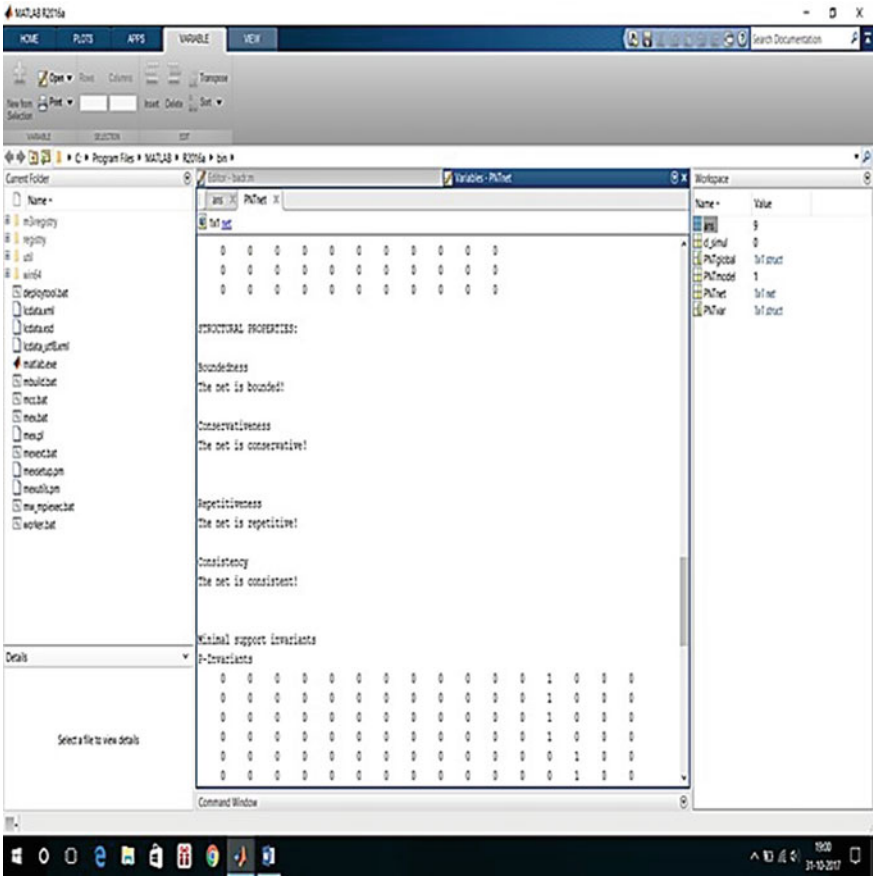

Fig. 28.5 Incidence matrix part 2

is given below. The population at time t is represented by variable s, with the initial population of random estimates as s (0).

Procedure GA
$t = 0$;
initialize $s(t)$;
evaluate $s(t)$;
begin
$t = t + 1$;
select $s(t)$from $s(t - 1)$
reproduce pairs in $s(t)$
evaluate $s(t)$;

Step 4.2: Population evaluation

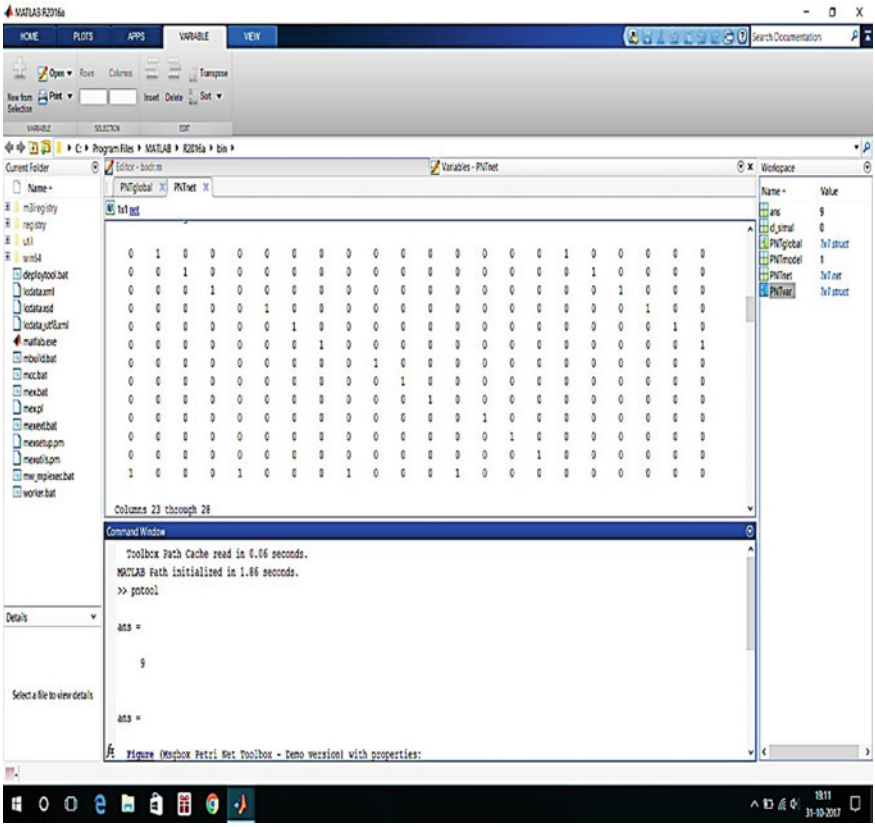

Fig. 28.6 Command window answer

The fitness parameter [fit (c)] considered is the makespan time. The initial population is evaluated for makespan.

Step 4.3: Selection of new population

The process of selecting the chromosomes to represent the next generation has the following steps:

1. Conversion of the fitness parameter values to a new fitness value [new_ fit (c)]. A fitness function considered here, which is suitable for minimization objective is

$$\text{new fit}(c) = 1 - \text{fit}(c)/F \tag{28.1}$$

where fit (c) = makespan time corresponding to chromosome.

F is the sum of the fitness parameter of all chromosomes.

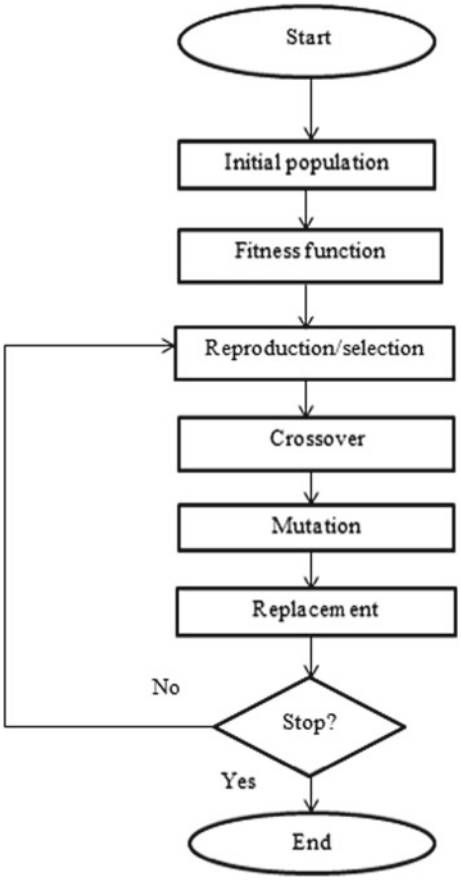

Fig. 28.7 Flow chart for GA. Reproduced from Munibabu et al. [5]

Table 28.4 Chromosome representation for four scheme (Reproduced from [1]

S. No.	Representation	Generation	String length
1	Job-based representation	Job to be processed at a particular instant	Number of jobs
2	Operation-based representation	An operation that is context-dependent	Number of operations
3	Priority rule-based representation	Priority dispatching rule	Number of operations/machine
4	Preference-list-based representation	A string of symbol with length of number of jobs	Number of machines

$$F = \sum_{c=1}^{\text{pop\_size}} \text{fit}(C) \tag{28.2}$$

To have more copies of chromosomes with the smallest objective function value, the ratio fit ©/F is subtracted from 1.

2. Conversion of the new fitness parameter to an expected frequency of selection [$p(c)$].

$$P(c) = \text{new\_fit}(c) / \sum_{c=1}^{\text{pop\_size}} \text{fit}(c) \tag{28.3}$$

3. Calculation of the cumulative probability of survival (cp (c)). A random selection procedure, which is explained below, generates the next population of the same size. A random number between 0 and 1 is obtained and a chromosome c is selected which satisfies the following condition. This selection process is repeated several times equal to the population size. The method used here is more reliable in that it guarantees that the fittest individuals will be selected and that the actual number of times each is selected will be expected frequency ±1. This procedure enables the first chromosome to have multiple copies and the worst will die off [1, 5].

The fitness function calculated here is the output from the Gantt chart. Expected count, new fitness function is given by, 1 − (13/47).

Probability of selection is given by (0.0.7234/2.9999).

The string number here represents the number of job.

Step 4.4: Selection of parents for crossover

Once the makespan is calculated for the different chromosomes, tournament selection is done to filter out those chromosomes which have better makespan values (in this case lesser makespan value) and these chromosomes are then selected to undergo crossover and mutation. In this problem, the tournament size has been taken to be two. Two chromosomes are randomly chosen from the population and their makespan values are compared, whichever chromosome has a lesser makespan value is deemed the winner. After the parents have been chosen, crossover is applied to them.

Chromosomes reproduce among themselves according to a predefined crossover probability. The first child is generated by taking the first portion, from the beginning of the chromosome until the crossover point, of the first parent and the second portion of the second parent, from the crossover point until the end of the chromosome.

Step 4.5: Mutation

Generate the random number, for all chromosomes, if $r \leq$ pm mutate the chromosome, where pm is the probability of mutation. Mutation is a diversification strategy

used mostly to avoid the repetition of chromosomes. The mutation followed in this paper is the order-based mutation (OBM). In this, we pick two loci in the chromosome at random and exchange their genes. For example, 3 0 1 2 is mutated as 3 2 1 0.

Step 4.6: Termination

The above process will be repeated for a fixed number of generations.

The working of the genetic algorithm is explained in Step 4. The GA parameters considered are given in Table 28.5. The parameters used in GA are

Initial population, corresponding fitness values (population evaluation), selection of chromosomes for the next generation, and chromosomes in the mating pool for the considered example are given in Table 28.6. The parents selected for crossover from the mating pool and corresponding offsprings after crossover, chromosomes selected for mutation, new population after mutation, and fitness values for the considered example are given in Table 28.7.

Table 28.5 Parameters used in GA

Parameters	Particulars	Remarks
Population size	A fixed number of individual form the GA population	10
Crossover rate	The probability for an individual to perform crossover	0.8
Mutation rate	The probability for an individual to perform mutation	0.05

Table 28.6 Evaluation and reproduction phase

String No	Initial position	Fitness value	Expected count	Probability of selection	Cumulative probability	Random No	String No.	Mating pool
0	1203	13	0.7234	0.2411	0.2412	0.22	2	3021
1	2130	9	0.8085	0.2695	0.5107	0.02	1	2130
2	3021	14	0.7021	0.234	0.7447	0.36	3	0312
3	0312	11	0.7659	0.2553	1	0.22	2	3021
		47	2.9999					

Table 28.7 Population cross over

Mating pool	Random No.	Selected over crossover	Population after crossover	Random mutation	Population after mutation	Fitness value
3021	0.8900	Yes	3012	0.5000	3210	16
2130	0.7300	No	2130	0.3214	2130	9
0312	0.6200	No	0321	0.2200	0321	11
3021	0.8900	No	3021	0.0300	3021	14

The best makespan and corresponding schedule are given below for the example problem for the job-based representation schemes considered in this example. The best makespan obtained is 9 and Best schedule: 2-1-3-0.

28.4 Conclusions

The scheduling is one of the important aspects of the smooth functioning of flexible manufacturing system. In the present work combined approach of Petrinet and genetic algorithm is used to produce near to optimal result. Here only one performance parameter is considered that is makespan. Due to good modeling power, Petri net is used as this feature is required to model a real manufacturing system. Genetic algorithms are powerful optimization tool and gives better result for combinatorial problems. Out of the four representations scheme the job-based representation is addressed for the optimum makespan. The hybrid approach plays a vital role in scheduling the system and optimizing the makespan.

In the future, the focus must be on a more complex manufacturing system with different representation schemes and with minimum number of constraints. Also, the focus on multi-objective optimization with conflicting objectives/performance measures will be addressed.

Annexure

```
Population population = new Population();
    Individual fittest;
    Individual secondFittest;
    int generationCount = 0;

    public static void main(String[] args) {

        Random rn = new Random();

        GA demo = new GA();

        //Initialize population
        demo.population.initializePopulation(10);

        //Calculate fitness of each individual
        demo.population.calculateFitness();

        System.out.println("Generation: "

+ demo.generationCount + " Fittest: "

+ demo.population.fittest);

        //While population gets an individual with maximum
fitness
```

```
        while (demo.population.fittest < 5) {
      ++demo.generationCount;

            //Do selection
            demo.selection();

            //Do crossover
            demo.crossover();

            //Do mutation under a random probability
            if (rn.nextInt()%7 < 5) {
                demo.mutation();
            }

            //Add fittest offspring to population
            demo.addFittestOffspring();

            //Calculate new fitness value
            demo.population.calculateFitness();

            System.out.println("Generation: "

+       demo.generationCount       +       " Fittest: "       +
demo.population.fittest);
        }

        System.out.println("\nSolution found in generation "  +
demo.generationCount);
        System.out.println("Fitness: "

+ demo.population.getFittest().fitness);
        System.out.print("Genes: ");
        for (int i = 0; i < 5; i ++) {

System.out.print(demo.population.getFittest().genes[i]);
        }

        System.out.println("");

    }

    //Selection
    void selection() {

        //Select the most fittest individual
        fittest = population.getFittest();

        //Select the second most fittest individual
        secondFittest = population.getSecondFittest();
    }

    //Crossover
    void crossover() {
        Random rn = new Random();

        //Select a random crossover point
        int crossOverPoint = rn.nextInt
```

```
(population.individuals[0].geneLength);

        //Swap values among parents
        for (int i = 0; i < crossOverPoint; i++) {
            int temp = fittest.genes[i];
            fittest.genes[i] = secondFittest.genes[i];
            secondFittest.genes[i] = temp;

        }

    }

    //Mutation
    void mutation() {
        Random rn = new Random();

        //Select a random mutation point
        int mutationPoint

= rn.nextInt(population.individuals[0].geneLength);

        //Flip values at the mutation point
        if (fittest.genes[mutationPoint] == 0) {
            fittest.genes[mutationPoint] = 1;
        } else {
            fittest.genes[mutationPoint] = 0;
        }

        mutationPoint

= rn.nextInt(population.individuals[0].geneLength);

        if (secondFittest.genes[mutationPoint] == 0)

{
            secondFittest.genes[mutationPoint] = 1;
        } else {
            secondFittest.genes[mutationPoint] = 0;
        }
    }

    //Get fittest offspring
    Individual getFittestOffspring() {
        if (fittest.fitness > secondFittest.fitness) {
            return fittest;
        }
        return secondFittest;
    }

    //Replace least fittest individual from most fittest

offspring
    void addFittestOffspring() {

        //Update fitness values of offspring
        fittest.calcFitness();
        secondFittest.calcFitness();
```

```
        //Get index of least fit individual
        int leastFittest

Index = population.getLeastFittestIndex();

        //Replace least fittest individual from most fittest

offspring
        population.individuals[leastFittestIndex]            =
getFittestOffspring();
    }

}

//Individual class
class Individual {

    int fitness = 0;
    int[] genes = new int[5];
    int geneLength = 5;

    public Individual() {
        Random rn = new Random();

        //Set genes randomly for each individual
        for (int i = 0; i < genes.length; i ++) {
            genes[i] = Math.abs(rn.nextInt() % 2);
        }

        fitness = 0;
    }

    //Calculate fitness
    public void calcFitness() {

        fitness = 0;
        for (int i = 0; i < 5; i ++) {
            if (genes[i] == 1) {
       ++fitness;
            }
        }
    }

}

//Population class
class Population {

    int popSize = 10;
    Individual[] individuals = new Individual[10];
    int fittest = 0;

    //Initialize population
    public void initializePopulation(int size) {
        for (int i = 0; i < individuals.length; i ++) {
            individuals[i] = new Individual();
        }
```

```
    }

    //Get the fittest individual
    public Individual getFittest() {
        int maxFit = Integer.MIN_VALUE;
        int maxFitIndex = 0;
        for (int i = 0; i < individuals.length; i ++) {
            if (maxFit <= individuals[i].fitness) {
                maxFit = individuals[i].fitness;
                maxFitIndex = i;
            }
        }
        fittest = individuals[maxFitIndex].fitness;
        return individuals[maxFitIndex];
    }

    //Get the second most fittest individual
    public Individual getSecondFittest() {
        int maxFit1 = 0;
        int maxFit2 = 0;
        for (int i = 0; i < individuals.length; i ++)

{
            if (individuals[i].fitness > individuals[maxFit1].

fitness) {
                maxFit2 = maxFit1;
                maxFit1 = i;
            } else if (individuals[i].fitness                >
individuals[maxFit2].fitness) {
                maxFit2 = i;
            }
        }
        return individuals[maxFit2];
    }

    //Get index of least fittest individual
    public int getLeastFittestIndex() {
        int minFitVal = Integer.MAX_VALUE;
        int minFitIndex = 0;
        for (int i = 0; i < individuals.length; i ++) {
            if (minFitVal >= individuals[i].fitness) {
                minFitVal = individuals[i].fitness;
                minFitIndex = i;
            }
        }
        return minFitIndex;
    }

    //Calculate fitness of each individual
    public void calculateFitness() {

        for (int i = 0; i < individuals.length; i ++) {
            individuals[i].calcFitness();
        }
        getFittest();
    }

}
```

References

1. Ponnambalam, S.G, Aravindan, P., Sreenivasa Rao, P.: Comparative Evaluation of Genetic Algorithms for Job-Shop Scheduling, Vol. 12(6), pp. 560–574. Taylor & Francis (2010)
2. Murata, T.: Petri nets: properties, analysis, and application. IEEE **77**(4), 541–580 (1989)
3. Zurawski, R., Zhou, M.: Petri nets and industrial applications: a tutorial. IEEE. Trans. Ind. Electron. **41**(6), 567–583 (1994)
4. Filho, M.G., Barco, C.F., Neto, R.F.T.: Using genetic algorithms to solve scheduling problems on flexible manufacturing systems (FMS): a literature survey, classification and analysis. Flex. Serv. Manuf. J. **26**, 408–431 (2014)
5. Muni Babu, P., Himasekhar Sai, B.V., Sreenivasulu Reddy, A.: Optimization of make-span and total tardiness for flow shop scheduling using genetic algorithm. Int. J. Eng. Res. Gen. Sci. **3**(3) (2015)
6. Kumar, O.C.: Scheduling of flexible manufacturing system using timed Petri nets. In: International Conference on Automation, Indore (1995)
7. Vadher, J., Patel, M.B.: Dynamic scheduling of manufacturing system with stochastic timed petrinet: a genetic algorithm approach. In: Proceedings of ISECON, Vol. 25 (2008)