A Genetic Algorithm for Two-Machine Flow Shop
with A Batch Processing Machine

Wikrom Jaruphongsa
Sasin Graduate Institute of Business Administration of
Chulalongkorn University
Bangkok, Thailand
e-mail: wikromj@sasin.edu

Yvonne Whye Lin Tan
Department of Industrial and Systems Engineering
National University of Singapore
Singapore
e-mail: tan_whye_lin_yvonne@moe.edu.sg

Abstract — This paper addresses a two-machine flow shop scheduling problem that includes a discrete processing machine followed by a batch processing machine in the line, which is applicable to the burn-in operations in a semiconductor fabrication. The objective is to minimize the makespan. This paper proposes a Genetic algorithm for this NP-hard problem. The experimental results show that the proposed algorithm find near-optimal solutions with much shorter computational times in comparison with those obtained from a mixed integer program (MIP).

Keywords - flow shop scheduling; batch scheduling; Genetic algorithm

I. INTRODUCTION

Machine scheduling has received much attention in the area of operations research. It is widely applied in manufacturing industries such as the semiconductor and the metalworking, computer industries, industries that engage machines etc. Effective scheduling methods are deemed crucial as more automated industrial technologies become available. Scheduling arranges tasks so that one or more performance measures may be optimized and it plays a significant role in determining the efficiency and responsiveness of an industry.

Flow shops are commonly used as applications of machine scheduling and are underlying architectures of manufacturing industries. A flow shop has a fixed line of machines for jobs to be processed. Jobs enter from one point and leave the system at the other endpoint and this structure is maintained by all jobs. Flow shops have advantages of low cost, ease of supervision, etc. [10]. The simplest form of a two-machine flow shop is one that comprises of two discrete processing machines. A classical discrete processing machine processes one job at a time. An optimal schedule to minimize the makespan of this system is the application of the well-known Johnson’s rule. In real world production lines however, there are usually more than two machines in the production line. As the number of machine in a flow shop increases, the system becomes strongly NP-hard.

Batching is an important concept in operations research. In Hall and Potts’ research work [4], batching and delivery is defined as a group of jobs that is delivered to the next downstream stage as a single shipment. Machines in a flow shop can also include batch processing machines. Lee and Uzsoy [7] defined batch processing machine as one where a number of jobs can be processed simultaneously in a batch. The purpose of batching jobs enables the processing of jobs to be more efficient; jobs can be processed faster or more cost effective than being handled individually. Each batch requires one set up time regardless of the number of jobs in the batch, however there is usually an upper limit for batch sizes. Most research works were on single batch processing machine. There is motivation for venturing into a flow shop consisting of a batch processing machine due to its practical applicability to the semiconductor industry.

Semiconductors are widely used in everyday products and the efficacy for its production is crucial for its success. The four main steps of manufacturing very large scaled integrated circuits in the semiconductor manufacturing industry are wafer fabrication, wafer probe, assembly and final testing. In wafer fabrication, the integrated circuits are developed on silicon wafers using lithography, etching, ion implantation and diffusion processes. There are several hundred individual circuits in a single wafer. At the wafer probe, they are cut into individual circuits and tested using probes. Defective circuits are discarded while the rest proceed on to the assembly stage. At the assembly stage, leads are attached and the circuits are packaged and protected from the external environment. The packaged circuits then move on to the final testing stage where the circuits are placed in burn-in ovens. The purpose of this operation is to subject the circuits to thermal stress to extract those that are at risk of premature failure. The scheduling in the final testing stage is critical in productivity and on-time delivery management. Moreover, the processing times of burn-in operations are usually regarded as the bottleneck process because they are generally longer compared to the previous steps [11].

Hence, there is motivation in exploring the penultimate and final steps of manufacturing these integrated circuits. The assembly stage is modeled as a discrete job processing machine, followed by the burn-in operation which is modeled as a batch processing machine. The project is also motivated by the two-machine problem in [2], which only
This paper focused on a scheduling problem for a two-machine flow shop where a discrete processing machine is followed by a batch processing machine, and the processing time of a batch is equal to the longest processing time of the jobs in the batch. The objective is to determine the optimal sequence and batching compositions so as to minimize the makespan of all the jobs. Batch processing machine embodies a source of tension between efficiency and timely completion of jobs. While large batches are more desirable in maximizing machine utilization and minimizing the number of setups, they might delay completion time of jobs in the batch while waiting for more jobs to be added to it, and also delay the start time for following batches. Hence a trade-off between the two is needed.

The contributions of this paper are (i) to model the problem as a mixed-integer program (MIP), and (ii) to develop an efficient genetic algorithm for the problem. The paper is organized as follows. Section 2 presents a literature review of what has been explored in the area of batch scheduling, two machine flow shops and the applications of genetic algorithm. Section 3 provides a mathematical formulation of the problem. A genetic algorithm is proposed in section 4, followed by results analysis and discussion in section 5. Finally, section 6 concludes the paper and discuss future research directions.

II. LITERATURE REVIEW

Much research has been reported on batching and scheduling problems. Potts and Kovalyov [9] provided an extensive review on two different models on batching, namely, family scheduling and batch machine. Family scheduling refers to the batching of similar jobs, such that setup is not required unless the machine switches to process jobs of another family. They explored different algorithms and techniques, with special attention given to dynamic programming algorithms. Batch machines refer to machines which can process a number of jobs simultaneously. They can be classified into two different categories, according to their batch processing time pattern.

In the first category, the batch processing time is constant and independent of the jobs in the batch. This is applicable to the etching tank in wafer fabrication in the semiconductor industry, which was examined by [2]. On the other hand, the second category, which is more extensively reviewed, involves varying batch processing time, which depends on jobs that constitute the batch. The processing time of each batch can be represented by the maximum processing time of the jobs grouped together in the batch. This is applicable to the burn-in ovens in the final stage of the semiconductor manufacturing system. Such a problem is considered by [7]. They dealt with the problem of minimizing makespan on a single batch processing machine with dynamic job arrivals. Lee and Uzsoy [7] provided polynomial and pseudo polynomial time solutions procedures for special cases of the problem. They also suggested fast heuristics and conducted extensive computational experiments, some of which yielded consistently high average performance. Sung and Choung [11] presented a similar problem with dynamic job arrivals in which a branch and bound algorithm can be used to obtain an optimal solution. Several heuristics are also exploited to solve the problem more efficiently. An optimal schedule can be obtained for the case of static problem.

Halls and Potts [4] considered the problem where a supplier makes deliveries to several manufacturers, who in turn make deliveries to their customers. This was done by scheduling jobs and forming them into batches, whereby they would be delivered to the next downstream stage as a single shipment. The problem was broken down into the supplier’s problem, the manufacturer’s problem and finally the combined supplier’s and manufacturer’s problem. Dynamic programming algorithms were derived to minimize the overall scheduling and delivery cost. It was demonstrated that coordinated scheduling and batching decisions between supplier and manufacturer promotes an overall improvement in efficiency of supply chains.

The following literatures considered two-machine flow shop where a discrete processing machine is followed by a batch processing machine and dynamic job arrivals are allowed. The batch processing machine incorporated into the flow shop can belong either to the first or second category as mentioned above.

Chang and Young [2] examined a scheduling problem where the processing time of each batch is fixed regardless of the jobs in the batch. The objective was to minimize the makespan. Where preemption is allowed on the discrete machine, an optimal schedule can be found in \( O(n^2 \log n) \) time. They showed that the problem is NP-complete for the case where no preemption is allowed. For NP-complete problems, it is unlikely to find a polynomial optimal solution. Hence, an efficient heuristic strategy referred to as the DSPT-FOE(\( n, c \)) was proposed which generates good quality schedules. This schedule is optimal for some special cases of the problem and a tight worst-case error bound was derived.

The soldering and testing equipment in the manufacturing of large-scale integrated circuits is an application for batch processing machines with processing time equal to the sum of all the jobs in the batch and is studied by [3] in a two-machine flow shop problem. They presented a strong NP-hardness result for the problem and identified polynomially solvable cases. The result obtained from a special case of the problem is used as a lower bound to evaluate the quality of heuristic solutions.

Oulamara [8] presented a two-machine flow shop consisting of two batch processing machines, one with batch processing time equal to the largest processing job in the batch and the other with batch processing time equal to the sum of all the jobs in the batch. He presented the NP-hard results of the problem and examined the case of constant processing times on the first machine, and then on the second. A heuristic was then generated with guaranteed performance ratio equal to two for the problem.

Genetic algorithms are categorized as global search heuristics and belong to the class of evolutionary algorithms that uses techniques inspired by evolutionary biology. It is used as a method for finding solutions to optimization and
search problems and has been deemed an efficient method as it is able to produce good solutions within short computational time. Kim and Kim [6] used genetic algorithm by incorporating Full Batch Dealing SPT Policy into the initial population to determine the best job sequence for minimizing the total completion time of jobs in a flow shop consisting of a batch and discrete processing machine, where the processing time of a batch is assumed to be constant. They showed that the average solution quality of genetic algorithm approach has a 2.5% - 5% improvement over conventional methods, especially for large-scaled problems.

Kashan et al. [5] applied two different genetic algorithms based on different chromosome representations for the problem of a single batch processing machine with the objective to minimize the makespan. The two algorithms searched the solution space differently and results showed that one outperformed both the other and a simulated annealing approach taken from the literature as a base algorithm, especially for bigger problems. The genetic algorithm is capable of producing near optimal solutions within reasonable CPU times.

In the problem of minimizing maximum lateness on a single batch processing machine with dynamic job arrivals, Wang andUszoy [12] presented a genetic algorithm based on random keys encoding scheme. Instead of the traditional one-point crossover and classic mutation, they used parameterized uniform crossovers and immigration of newly randomly generated chromosomes. They combined genetic algorithm with bisectional search and obtained results which outperformed the best heuristics by an average of 25%.

III. THE PROBLEM

A. Problem Definition

The two-machine flow shop consists of a discrete processing machine followed by a batch processing machine. The discrete processing machine is defined as machine 1 and the batch processing machine is defined as machine 2. There are \( n \) jobs to be processed and the arrival time of each job is denoted by \( a_j, j = 1, 2, ..., n \). Each job \( j \) has a fixed processing time of \( p_j \) on machine 1 and \( q_j \) on machine 2. After each job has finished processing on machine 1 individually, they are grouped into batches to be processed on machine 2. The processing time of each batch of jobs on machine 2 is equal to the longest processing time among jobs in the batch. Therefore, all jobs in a batch will have the same processing time and completion time on machine 2. Machine 2 can only process up to \( B \) jobs simultaneously. The objective is to minimize the makespan. Additional assumptions follow:

1. No batch setup is needed for each batch. We can assume that the setup time has been included into the processing time of the jobs.
2. Preemption is not allowed.
3. Once the processing of a batch has started on machine 2, it cannot be interrupted and other jobs cannot be introduced into the batch.

B. Mathematical Model

The decision variables used in the MIP model are as follows:

\[ C_k = \text{completion time of job in position } k \text{ on machine 1}, \]
\[ S_s = \text{start time of batch } s \text{ on machine 2}, \]
\[ Q_s = \text{processing time of the batch } s \text{ on machine 2}, \]
\[ x_{ik} = 1 \text{ if job } i \text{ is in position } k; 0 \text{ otherwise}, \]
\[ y_{ks} = 1 \text{ if job in position } k \text{ is assigned to batch } s; 0 \text{ otherwise}. \]

The mathematical model for the problem is given by:

\[
\begin{align*}
\min & \quad S_s + Q_s \\
\text{subject to} & \quad \sum_{j=1}^{n} x_{ik} = 1 \quad k = 1, 2, ..., n \quad (1) \\
& \quad \sum_{i=1}^{n} x_{ik} = 1 \quad i = 1, 2, ..., n \quad (2) \\
& \quad \sum_{k=1}^{n} y_{ks} = 1 \quad k = 1, 2, ..., n \quad (3) \\
& \quad \sum_{s=1}^{B} y_{ks} \leq B \quad s = 1, 2, ..., n \quad (4) \\
& \quad C_k \geq \sum_{i=1}^{n} p_i x_{ik} + \sum_{i=1}^{n} q_i x_{ik} \quad k = 1, 2, ..., n \quad (5) \\
& \quad C_k \geq C_{k-1} + \sum_{i=1}^{n} p_i x_{ik} \quad k = 2, 3, ..., n \quad (6) \\
& \quad S_s \geq C_s + (1 - y_{sk})M \quad s, k = 1, 2, ..., n \quad (7) \\
& \quad S_s \geq S_{s-1} + Q_{s-1} \quad s = 2, 3, ..., n \quad (8) \\
& \quad Q_s \geq \sum_{i=1}^{n} q_i x_{ik} - (1 - y_{sk})M \quad s, k = 1, 2, ..., n \quad (9) \\
& \quad C_k, S_s, Q_s \geq 0 \quad s, k = 1, 2, ..., n \quad (10) \\
& \quad x_{ik}, y_{ks} \in \{0, 1\} \quad i, s, k = 1, 2, ..., n \quad (11) \\
\end{align*}
\]

where \( M \) is a very large number (Big \( M \)).

Constraint (1) ensures that each position can be occupied only by one job. Constraint (2) ensures that each job occupies only one position. Constraint (3) imposes the condition that each job in a position can only belong to only one batch and constraint (4) makes sure that the number of jobs that belongs to each batch cannot be more than the batch size limit \( B \). Through constraint (5), jobs can only be completed on machine 1 after they have arrived and are processed by machine 1. Constraint (6) further ensures that jobs can only be completed on machine 1 after the job in the position before them are completed and after they are processed by machine 1. By imposing constraint (7), a batch can only start processing on machine 2 after the last job belonging to the batch has been processed by machine 1. Constraint (8) further limits the start time of the batch on machine 2 to be after the completion time of the previous batch on machine 2. Constraint (9) guarantees that the processing time of a batch on machine 2 is equal to the
largest job in the batch. The non-negativity constraint (10) is imposed on \( \ell_{ij} \), \( \delta_i \), and \( Q_i \), and lastly, constraint (11) ensures that \( x_{ik} \) and \( y_{ik} \) takes only binary values.

C. Model Complexity

Chang and Young [2] considered the problem of a two-machine flow shop with a discrete processing machine followed by a batch process machine. All jobs have dynamic arrivals and a constant processing time on the batch processing machine. Such a problem is proven to be NP-hard. Therefore, the problem considered in this paper is also NP-hard.

It is very inefficient to find the optimal solution using MIP as the problem grows larger in dimension as the number of jobs increases. Hence, we want to develop a heuristic to find a good solution more efficiently. Furthermore, we realize that there is no easy way to batch the jobs on machine 2 even if a predetermined sequence exists on machine 1, i.e. the jobs processed on machine 1 need not to be processed in the same sequence on machine 2. Therefore, in this paper, we propose a meta-heuristic to solve such a problem. Genetic algorithm is chosen in this paper as it deems effective in dealing with machine scheduling problems.

IV. GENETIC ALGORITHM

A. Encoding

In the coding scheme, each job is represented by a gene and each set of \( n \) jobs is represented by a chromosome. To generate the chromosomes, \( 2 \) sets of codes are generated. The first set of codes is generated by using random key representation of Bean, which represents a possible order of jobs on machine 1 with a sequence of numbers drawn from \([0, 1]\). Next, a random integer \( m \) is drawn from \([k, n]\), where \( k = n/B \) is the least number of batches required for the set of jobs and \( n \) is the number of jobs, to determine a feasible number of batches for the set of jobs. The second set of codes is then a sequence of numbers drawn from \([1, m]\), which is used as a solution for the problem of assigning jobs to batches. Hence, each gene contains two codes, one of which represents its position on machine 1 and the other represents the batch to which the job is assigned. Figure 1 shows an example of an encoded chromosome with \( m = 3 \).

<table>
<thead>
<tr>
<th>Code 1</th>
<th>Code 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.493</td>
<td>3</td>
</tr>
<tr>
<td>0.851</td>
<td>1</td>
</tr>
<tr>
<td>0.231</td>
<td>2</td>
</tr>
<tr>
<td>0.734</td>
<td>1</td>
</tr>
<tr>
<td>0.913</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 1. Encoding of a chromosome.

B. Initial Population

The population size of each generation is set to be \( 2n \). The genetic algorithm begins by randomly generating \( 2n \) chromosomes with \( 2 \) sets of numbers encoded. This starting population is called generation 1.

C. Decoding

The first set of codes is random keys for the job sequence on machine 1. To decode the sequence, we sort the keys in descending order and the rank of each key determines its position on machine 1. For example in Figure 1, the largest key 0.913 belongs to job 5 and hence job 5 is the first to be processed on machine 1. Job 1 has the fourth largest key and hence is the fourth job to be processed on machine 1.

The second set of codes is random numbers drawn from 1 to \( m \) and represents the batch to which a job belongs. The number represents only the group and not the position of the batches. For example in Figure 1, there are 3 batches \( \{J2, J4\}, \{J3\} \), and \( \{J1, J5\} \).

Given the sequence of jobs on machine 1 and the batches to which each job belongs, the problem remaining is that of deciding the sequence of batches to be processed on machine 2. This problem can now be simplified into that of \( 1|\tau_j|C_{\text{max}} \) problem where \( \tau_j \) is the arrival time of batches to machine 2. The sequence of the batches on machine 2 is determined by the completion time of the jobs in the batch on machine 1; i.e., the batches which first arrive on machine 2 will be processed first. It is easy to show that this sequencing rule of the batches is optimal. With these information, the makespan of each chromosome can be computed and let \( M_S \) represents the makespan of chromosome \( i \).

D. Reproduction

During the process of reproduction, we have to ensure that the fittest chromosomes are chosen to pass their genes to the next generation. The method of selection employed is that of binary tournament. Two chromosomes are picked randomly from the current generation and the one which is fitter is chosen and placed in the pool of parents. The chromosomes with lower makespan are the fitter ones. The fitness of each chromosome is its makespan; i.e., \( F(i) = M_S \). Once the pool of parent is formed, two chromosomes are then randomly selected from the pool to reproduce two children for the next generation.

E. Crossover

For each randomly selected pair of parents from the pool, two-point crossover is carried out to produce two children for the next generation. This will ensure that the good genetic traits of the parents will be passed on to the next generation. An example of a crossover procedure is illustrated by Figure 2.

For the example in Figure 2, Child 2 would have an infeasible solution after the crossover process if the limit for batch size is 2. Hence, the solutions need to be checked for the batch size feasibility after reproduction. If any solution is found infeasible, it will have to go through a renaming process where a random number from 1 to \( n \) will replace the gene that has the batch number that is over limit. The gene of the child that is being replaced must not belong to its main parent. In the above example, the third gene of Child 2 will be replaced.
J1  J2  J3  J4  J5
Parent 1  0.493  0.851  0.231  0.734  0.913
                   3  1  2  1  3
J1  J2  J3  J4  J5
Parent 2  0.382  0.189  0.982  0.237  0.555
                   3  2  4  1  2
J1  J2  J3  J4  J5
Child 1  0.493  0.851  0.982  0.237  0.913
                   3  1  4  1  3
J1  J2  J3  J4  J5
Child 2  0.382  0.189  0.231  0.734  0.555
                   3  2  2  1  2

Figure 2. Crossover Encoding of a chromosome.

F. Mutation

Mutations are important for diversity in the solutions as well as prevent solutions from being trapped in local optima. Hence there is a low probability that each child will undergo mutation during reproduction. A low probability will prevent destruction of the structure of fit chromosomes. The swapping mutation is used as mutation operator in this GA. It randomly selects two genes from a selected chromosome and swaps them with each other. An example where genes 1 and 4 are swapped is illustrated in Figure 3.

Before
  J1  J2  J3  J4  J5
  0.493  0.851  0.231  0.734  0.913
  3  1  2  1  3

After
  J1  J2  J3  J4  J5
  0.734  0.851  0.231  0.493  0.913
  1  1  2  3  3

Figure 3. Mutation of a chromosome.

G. Stopping Criteria

The algorithm will terminate when it has reached a predetermined number of generations, G. Alternatively, the algorithm will also terminate if there is no more improvement after a predetermined number of consecutive generations in order to avoid useless computations.

H. Parameters and Experimental Design

To form a new generation, 20% of the previous population is selected through binary tournament to form a pool of parents. Hence, the new generation is 80% made up of children reproduced from the pool of parents, with a small chance of being mutated, and 20% made up of the parents themselves, to retain structurally fit chromosomes in the new generation.

Randomly generated test problems are used to test the effectiveness of the algorithm. In this experimental design, we used the parameter values in the below table for our experiments:

<table>
<thead>
<tr>
<th>TABLE I.</th>
<th>EXPERIMENTAL PARAMETERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>Values</td>
</tr>
<tr>
<td>Population size</td>
<td>2n</td>
</tr>
<tr>
<td>Maximum number of generations</td>
<td>200</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>0.8</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>0.1</td>
</tr>
</tbody>
</table>

We took two factors consideration: number of jobs and the batch size limit. The table below shows the total number of problems in the experiment:

<table>
<thead>
<tr>
<th>TABLE II.</th>
<th>EXPERIMENTAL FACTORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factors</td>
<td>Values</td>
</tr>
<tr>
<td>Number of jobs (n)</td>
<td>20, 40, 60, 80</td>
</tr>
<tr>
<td>Batch size (B)</td>
<td>3, 5, 7</td>
</tr>
<tr>
<td>Instances for each setting</td>
<td>10</td>
</tr>
</tbody>
</table>

The set of random values used for arrival time, processing time on machine 1 and machine 2 are from 1 to 30 for any number of jobs.

V. RESULTS ANALYSIS AND DISCUSSIONS

The genetic algorithm (GA) was built into Visual Basic C++ 2008 express edition. To form a basis for comparison, the mathematical optimization model was also built into ILOG OPL CPLEX 6.0. By comparing the results generated by the genetic algorithm with results from OPL, we can know whether the genetic algorithm is effective in searching for near optimal solutions, given that it takes a much shorter computational time than depending on OPL for optimal solutions. The OPL usually requires a long time to search for optimal solutions for larger problems, thus we have limited the maximum running time of the OPL to be 1 hour. We use the following calculation to evaluate the quality of the solution generated by the genetic algorithm.

\[
\text{% Effectiveness} = \frac{\text{Solution of GA} - \text{Solution of MIP}}{\text{Solution of MIP}} \times 100\%
\]

Hence, the negative effectiveness means that the solution obtained from the genetic algorithm is better than that obtained from MIP within one hour. The summary of the computational results are shown in Table 3 below.

From the results, the OPL was able to find optimal solutions within a few minutes for most of the instances of 20 jobs. It was only able to find a few optimal solutions within an hour for the case of 40 jobs. For instances of more
than 60 jobs, the OPL failed to find an optimal solution for any problem instances. For our genetic algorithm, it was able to successfully achieve optimal solutions for the most of the instances of 20 jobs. For instances of 40 and 60 jobs, the algorithm was able to perform almost as well as the results obtained from running OPL for an hour. The average effectiveness of the solution of GA was -7.42% for instances of 80 jobs. This means that the GA could perform better than using OPL within an hour. In fact, based on our experiment, our GA found a better solution that those of OPL for all problem instances within one hour time limit. For the genetic algorithm, the run time ranges from seconds for the small problem instances up to 20 minutes for larger problem instances. Hence, GA is considered to be more practical for use to solve larger problems. It has a diversified solution space and hence avoids being trapped in local optima. However, it can be further improved by experimenting with different values of mutation rate or mutation methods employed, such as pair-wise interchange or random selection of gene for mutation.

For both the MIP and GA, it would be more efficient if an upper bound for the number of batches used in the solutions could be defined, as the number of batches utilized is often less than the total number of jobs. However, in this problem of dynamic job arrivals, this upper bound cannot be defined as jobs may arrive in different timings such that no two jobs can be batched together. Hence, the dimension of the problem is hard to reduce.

VI. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

This paper focused on a two-machine flow shop, with a discrete processing machine and a batch processing machine. The scheduling objective is to minimize the makespan. We develop MIP formulation for the problem which works well for small problems. However, as the problem is classified as NP-hard, it is very difficult to achieve optimal solutions using the MIP model for larger problems. Therefore, we also introduce a genetic algorithm to efficiently find a good solution for the problem.

The problem looked into by this research paper only considered jobs of the same size and type. This paper could be extended by exploring jobs of different family types or job sizes. Furthermore, the orientation of the flow shop could be reverse to a batch processing machine first followed by a discrete processing machine to see the difference in two different types of flow shops.

REFERENCES