Flexible Flowshop problem minimizing total flow time and makespan using Evolutionary Algorithm

Gina M. Galindo Pacheco
Universidad del Norte, Barranquilla, Colombia, ggalindo@uninorte.edu.co

Luis E. Ramirez Polo
Fcimec, Barranquilla, Colombia, lramirez@fcimec.org

Vanessa P. Manotas Niño
Universidad de La Sabana, Chía, Colombia, vanessamn@unisabana.edu.co

ABSTRACT

Scheduling on many stations, with a variable number of machines on each station (Flexible Flowshop), looking to minimize multiple objectives as makespan and total flow time is not a new problem, but it is a low investigated area which has non-optimal solutions for all the cases. Our objective in the problem we've approached was to develop a meta-heuristic based on Strength Pareto Evolutionary Algorithm (SPEA) that give us solutions to the problem described before and be applied through an algorithm to see the sequence, the values of the objectives [Pareto optimal solutions (Cmax, Cprom)] for each station and verify the effectiveness. We are analyzing a making decision problem on the $FF_C\left|\right|C_{Max},\overline{C}$ scenario, this mean that we have a flexible flowshop with k stations that has (M_k) identical machines. The algorithm was executed 120 times with a maximum number of generations equal to 4. It was obtained a group of elitist members of each running, and it was applied dominance criterion to get the final group of non dominated solutions.

Keywords: Multi-objective, Pareto optimal solutions, flexible flowshop, meta-heuristic, SPEA, makespan, total flow time, algorithm

RESUMEN

La programación de operaciones en varias estaciones, con diversos números de maquinas en cada estación (Flowshop flexible), buscando minimizar múltiples objetivos como makespan y tiempo total de flujo no es un problema nuevo, pero si uno con poco trabajo desarrollado y el cual no presenta una solución óptima en todos los casos.

Nuestro objetivo fue plantear una meta-heurística basada en Strength Pareto Evolutionary Algorithm (SPEA) que diera una solución al problema anteriormente descrito y aplicarla mediante un algoritmo que arrojara una secuencia óptima y sus respectivos valores de respuesta [frente pareto (Cmax, Cprom)] para cada estación teniendo en cuenta las respectivas máquinas y a partir de ahí verificar la efectividad.

Estamos analizando un problema de toma de decisiones en el escenario $FF_C\left|\right|C_{Max},\overline{C}$, esto significa que tenemos un Flowshop Flexible con k estaciones que tiene (M_k) maquinas idénticas. El algoritmo fue ejecutado 120 veces con un número máximo de generaciones igual a 4. Se obtuvo obtuvo un grupo de miembros elitistas de cada corrida, y se aplicó criterio de dominancia para obtener el último grupo de soluciones no dominadas.

Palabras claves: Multiobjetivo, frente pareto, flowshop flexible, meta-heurística, SPEA, makespan, tiempo total de flujo, algoritmos.

1. INTRODUCTION

The flexible systems are defined as an automatic controlled process that could produce many items between determined ranges. It is a technology that helps to optimize the manufacturing with better times, lower costs and a better quality, through better control systems.

The flexible systems are conformed by a group of *k* stations and each station has m_k parallel machines that could process more than one job, but we need an optimal sequence of the job in each station to guaranty the minimum time of end of each job or an effective flexible system.

In 1973 began the first studies of scheduling with flexible flowshops, at that time the objective was to minimize the makespan. Hall, Schuurman and Woeginger argued the existence of a polynomial approximation to this type of problems which are NP hard. [1]

Most studies on scheduling problems assume that the machines are available at all times. In real industry settings, however, a machine may not always be available in the scheduling period due to, for example, a breakdown (stochastic) or preventive maintenance (deterministic). This paper considers that machines are always available. [1]

Our objective in the problem we've approached was to develop a meta-heuristic based on Strength Pareto Evolutionary Algorithm (SPEA) that give us solutions to the problem described before and be applied through an algorithm to see the sequence, the values of the objectives [Pareto optimal solutions (Cmax, Cprom)] for each station and verify the effectiveness.

2. PROBLEM STATEMENT

In this paper we are analyzing a making decision problem on the $FF_C \left| \, \right| C_{Max}, \overline{C}$ scenario, this mean that we have a flexible flowshop with *k* stations that has (M_k) identical machines.

For the solution of this problems trough traditional methods were taken as bases the mathematical formulation called **ERMA1** designed by Riane on [1997]. This mathematical formulation proposes to run the model using mixed integer programming (MIP).

The configuration for the problem above is presented below:

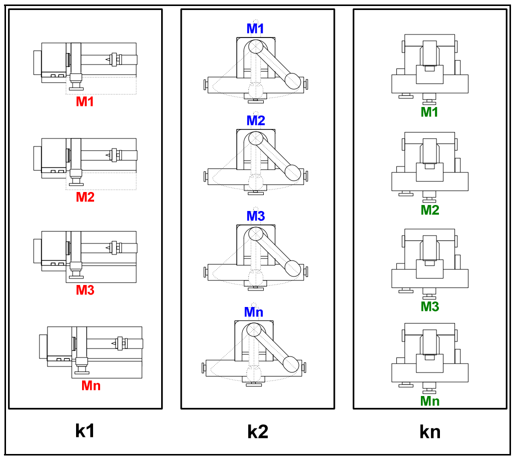


Figure 1. Facility layout

This specific problem has some constraints that should have being in mind, for example:

- A job *i* should pass at leats one time for each station.
- At least a job *i* should be processed in *l* position of each machine.

- A job *i* is processed in *l+1* position only if exists a job processed in *l* position on each machine every station.
- Each job *i* has a termination time C_i^k that depends for the first station on the processing time of each job on each machine of each station and the previous one, for other stations is maximum between the termination time in first station C_i^{k-1} and the previous job on the machine C_{i-1}^k plus the processing time.

3. FORMULATING FLOW SHOP FLEXIBLE SCHEDULING

Step 1: Chromosome

Each solution is represented by a chromosome conformed by two different matrix and it was designed as follows:

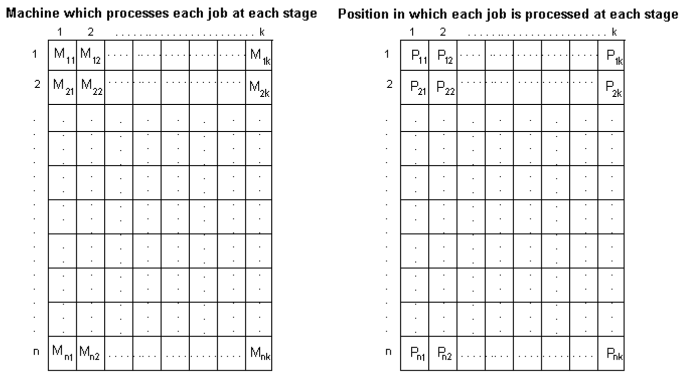

Machine which processes each job at each stage
Position in which each job is processed at each stage
M11 M12 M1k
M21 M22 M2k
Mn1 Mn2 Mnk
P11 P12 P1k
P21 P22 P2k
Pn1 Pn2 Pnk

Figure 2. Chromosome matrix

The matrix on the left contains variables that indicate the machine in which each one of the job is processed at a specific stage. For example, if M_{22} is equal to 3, it means that job 2 is processed in the third machine at the stage 2.

The second matrix contains the values pertaining to the position in which each job is processed at each stage, taking on count the machine specified for the job previously in the first matrix.

For the chromosome designed, it is important to guarantee the following issues:

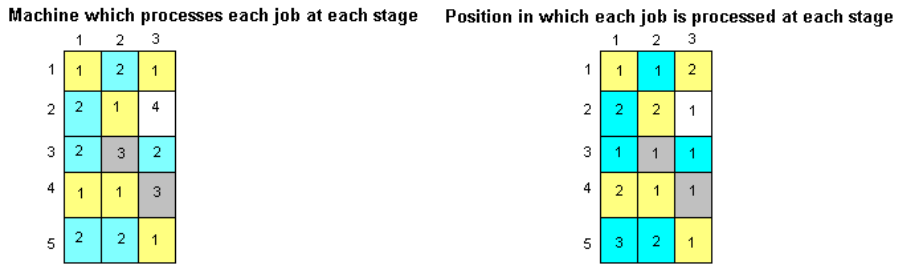
1. In the column pertaining to stage v, containing M(v) machines, any M_{iv} must be less or equal than $M(v)$.
2. There cannot be any case in which more than one job is processed at the same position in the same machine at the same stage.

In order to clarify the design of the chromosome, let's consider the case in which there are 5 jobs, 3 stages and the quantity of machines at each stage is represented by the following values:

Table 1. Quantity of machines at each stage

Stage (v)	Number of machines M(v)
1	2
2	3
3	4

A possible solution for the problem above is presented below:



Machine which processes each job at each stage

	1	2	3
1	1	2	1
2	2	1	4
3	2	3	2
4	1	1	3
5	2	2	1

Position in which each job is processed at each stage

	1	2	3
1	1	1	2
2	2	2	1
3	1	1	1
4	2	1	1
5	3	2	1

Figure 3. Possible solution

The code of colors used in the chromosome above is the following:

Machine Number	
1	🟨
2	🟦
3	▨
4	⬜

Figure 4. Code of colors

Notice that job number 1, at stage 1, is processed in the machine number 1 in the first position, and followed by job number 4 which is processed in the same machine but in second position at the same stage. Notice also that there is not any case in which more than one job is processed at the same position in the same machine at the same stage. Notice also that there is at most M(v) machines used at stage *v*.

In the algorithm, each chromosome is represented by a variable named "cromosoma" with 4 dimensions: the first dimension indicates the number of the solution analyzed; the second one, the job, and the third one, the stage. The last dimension represents the corresponding matrix: if it's equal to 1, it means it corresponds to the machines matrix, and if it's equal to 2, it is referred to the positions matrix.

It might be useful to say that initially it was thought to use a unique matrix as a chromosome with four dimensions, corresponding to the job analyzed, position, machine and stage. This matrix would have had only boolean values where, if X_{1211} had been equal to 1, it would have meant that job number 1 was processed second in the machine number 1 of the stage 1. However, using this type of chromosome would have implicated having too much values equals to 0, which had implicated a misuse of the computational capacity.

The first step when implementing SPEA is to create an initial population. In this case this population was randomly generated taking care of respecting the two important issues explained before.

The methodology used in the algorithm for creation initial population causes that the first job is always at the first position of its pertaining machine. For this reason, the randomness of the initial population is not guaranteed. In order to avoid this situation, every stage of each chromosome is randomly modified guaranteeing the randomness of the initial population.

They were also created other matrix for every stage which helps to determine the order of jobs in each machine at each stage. This type of matrix is called "orden (w, i, b, x)" and indicates which job is processed at position *b* at stage *i*, for the chromosome *w*.

Step 2: Objective values and dominance

In order to calculate the objective values of each chromosome, it was created a matrix containing the completion times for each job at each stage. With the information kept in this matrix they were later calculated makespan and sum of completion times for each chromosome. Then, with this information it can be determined which chromosomes are none dominated and which ones are not. Those chromosomes for which another solution presents smaller values for makespan *and* sum of completion times, is defined as *dominated.*

If the total number of non dominated chromosomes is larger than the maximum size of the elitist population, then it is necessary to apply clustering. Those elements that are taken out from the elitist population through clustering are included once again into the normal population.

Step 3: Fitness

The fitness is calculated in different ways for non dominated solutions and for those ones which belong to normal population.

- External population: in this case the fitness is calculated using the following expression:

$$F_i = 1 + \frac{n_i}{N+1}$$

where n_i is equal to the number of normal population (with size N) members that an external solution *i* dominates. The above equation assigns a larger fitness to a solution that dominates more elements in the current population.

- Current population: For current population members fitness is obtained by the following equation:

$$F'_j = 1 - \frac{\sum_{i \in \bar{P} \wedge i \prec j} F_i}{\sum_{i \in \bar{P}} F_i}$$

The expression above indicates that the fitness of a normal population member *j* is assigned as 1 minus the relation between the sum of fitness values of external population members which dominate j and the total sum of fitness values of external population members.

The expressions above guarantee that the fitness of every external population members will be always larger than the fitness of normal population members. In fact, in the best case in which a normal population member is no dominated by any elitist member, its fitness will be equal to 1, while the fitness of external population members is always larger than 1. On the other side, in the worst case in which a normal population member is dominated by all of the elitist solutions, its fitness will be very small, but never equal to zero in order to keep a little probability of not being eliminated.

Step 4: Cross Over

Once roulette method is applied and those chromosomes which might be crossed are chosen, it is evaluated if these chosen chromosomes will be crossed or not. For this purpose it is generated a random number between 0 and 1. If this random number is smaller than the probability of cross over, the chromosome is crossed with the following chromosome chosen by the roulette method. When both chromosomes to be crossed have been already selected, the cross over operation is applied. This operation consists on interchanging columns of machine matrix between the two solutions selected. The corresponding columns in the position matrix are also interchanged. This operation is illustrated in the figure below:

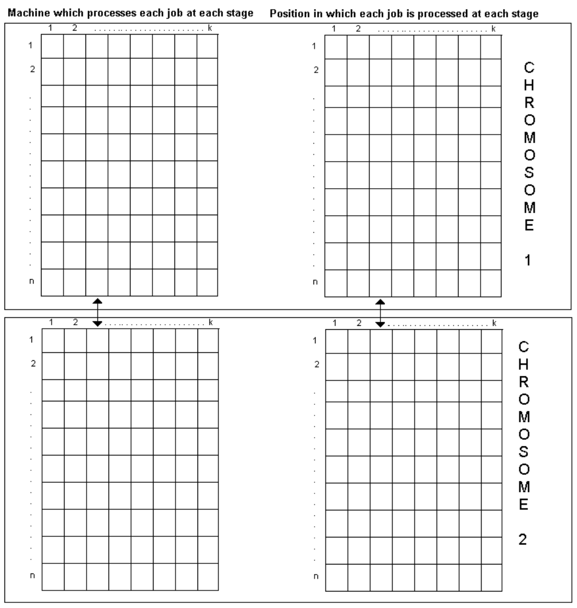

Machine which processes each job at each stage
Position in which each job is processed at each stage
CHROMOSOME 1
CHROMOSOME 2

Figure 5. Cross over matrix

In the algorithm, it is guaranteed that it will never cross two identical chromosomes. One of the advantages of this methodology is that it avoids infeasible solutions, due to the fact that it interchanges the whole set of machines and their corresponding positions of a specified stage randomly selected.

Step 5: Mutation

For those chromosomes which were selected by roulette method and which were not crossed, it is evaluated the probability of mutation through the generation of a random number. If the value of this number is less than the probability of mutation, the chromosome is mutated. The mutation consists on choosing a stage *v* from the chromosome selected and the values of each cell in the column pertaining to the stage *v*, are moved upward *r* positions, where *r* is a random number between 1 and the total number of jobs minus 1. The values of the cells at the top of the column are placed at the end of it. This procedure is done in both, machine and position matrixes, as it are shown on the next figure, where stage 2 is mutated:

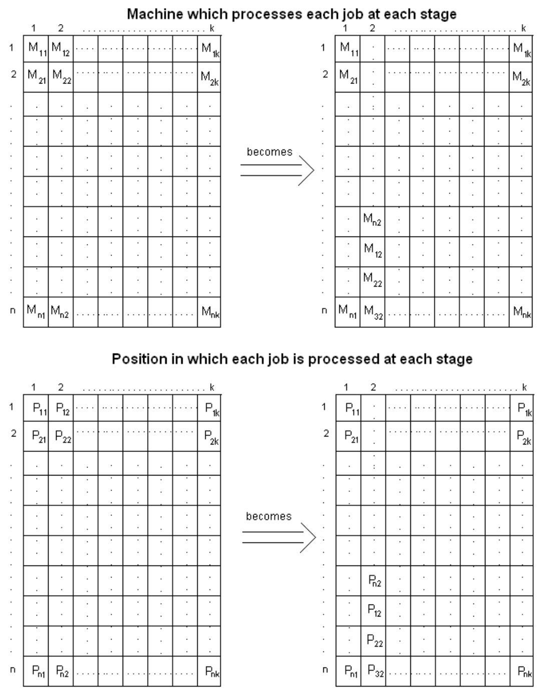

Machine which processes each job at each stage
becomes
Position in which each job is processed at each stage
becomes

Figure 6. Mutation matrix

Step 6: New Population

Finally the new normal population is conformed by those elements that were chosen by the roulette method; in despite of if they were mutated or crossed. Also, in the new generation are included the whole external population and some of the normal population members of the former generation.

4. ANALYSIS OF RESULTS

It was analyzed a case in which 5 jobs must be processed in three stages. The process times for each job and the number of machines at each stage, is given in the table below:

Table 2. Process time for job

Job	Stages		
	A	B	C
1	15	35	30
2	12	45	15
3	30	19	21
4	20	25	42
5	17	55	13
Machines	2	3	2

The algorithm was executed 120 times with a maximum number of generations equal to 4. After these runs, it was obtained a group of elitist members of each running, and it was applied dominance criterion to get the final group of non dominated solutions. The results obtained are shown on the following figure.

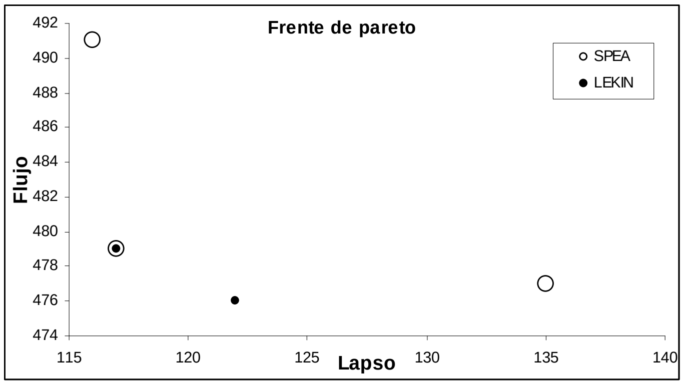


Figure 7. Pareto optimal solution

In the figure above the results pertaining to Lekin are represented by black points, while SPEA results are those white points. As it can be seen, there is a point that was found by both, SPEA and Lekin. It is important to mention that SPEA found a better solution for makespan, which it is due to the fact that Lekin is a heuristic technique so it might produce non optimal solutions in some cases.

5. CONCLUSIONS

Using SPEA for finding optimal solutions produced good results mainly on makespan. In fact, it produced a better solution comparing it to Lekin.

A possible disadvantage of SPEA is that each running takes too much time and the final solutions are not too far from Lekin solutions. However, this computational cost is compensated by the several solutions produced by SPEA on each running.

REFERENCES

Xie, J, and Wang, X. (2005). “Complexity and Algorithms for Two-Stage Flexible Flowshop Scheduling with Availability Constraints”. An international journal computers & mathematics with applications, Vol. 50, No. 10-12, pp 1629-1638.

Salvador, M. S. (1973). A solution to a special case of flow shop scheduling problems, in: S. E. Elmaghraby, (Ed.), Symposium on the Theory of Scheduling and Its Applications, Springer-Verlag, Berlin, 83C91.

Hall, L.A. (1998). “Approximability of flow shop scheduling”. Mathematical Programming Society, Vol 82, pp 175-190.

Schuurman, P. and Woeginger, G. J. (2000). “A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem”. Theoretical Computer Science, Vol 237, pp 105-122.

Pinedo, M. and Chao, X. (1999). “Operations Scheduling: whit applications in manufacturing and services”. New York: McGraw Hill.

Authorization and Disclaimer