On Modeling Genetic Algorithms for Flexible Job-shop Scheduling Problems

Khaled Mesghouni , Slim Hammadi and Pierre Borne

Ecole Centrale de Lille

LAIL - URA CNRS D 1440

59651 Villeneuve d'Ascq Cedex

FRANCE

e-mails : mesghouni@ec-lille.fr, hammadi@ec-lille.fr, p.borne@ec-lille.fr

Abstract: Several problems in various industrial environments are combinatorial problems. This is the case for numerous scheduling problems. Generally, there do not exist efficient algorithms to solve these problems in their general form. Generally, the combinations of goals and resources have exponentially growing search space, described in computer science terms as NP-complete. Therefore, exact methods such as branch and bound methods, dynamic programming methods, etc., take considerable computing time and/or require complex mathematical formulation. Recently stochastic search techniques such as genetic algorithms have been successfully applied to job -shop scheduling problems. Genetic algorithms present some advantages. They are robust in that they provide a good solution on a wide range of problems. In addition, they can easily be modified with respect to the objective function and constraints.

Our objective in this article is to improve performance of the genetic algorithms based approach to job -shop scheduling problems by developing effective genetic operators, such as a parallel representation of the chromosome, on the one hand, and genetic operators associated with this original representation, on the other one

In this article we deal with the problem of flexible job-shop scheduling which presents two difficulties : the first one is the assignment of each operation to a machine, and the second one is scheduling this set of operations in order to minimize our criterion (e.g. the makespan).

Keywords: Genetic Algorithms (GAs), Parallel Representation, Initial Population, Priority rules, Constraint Logic Programming

Dr. **Khaled Mesghouni** was born in Constantine, Algeria, in 1968. He received the "Diplôme d'Ingénieur en Electronique" from the University of Constantine, Algeria, and MSc. degree in Electrical Engineering from Ecole Centrale de Lille, France, in 1993 and 1995, respectively. Currently he prepares a Ph.D thesis in Automatic Control at the University of Science and Technology of Lille, France. His research interests include applications of Artificial Intelligence in scheduling problems and manufacturing systems, and regulation of automatic transport systems using Fuzzy Logic. Dr Khaled Mesghouni is a student member of IEEE.

Dr. **Slim Hammadi** is an Associate Professor of production planning and control at the Ecole Centrale de Lille. Born in Gafsa, Tunisia, in 1962, he obtained MSc. degree in Computer Science from the University of Lille, France, in 1988. In 1991 Dr. Hammadi obtained a Ph.D degree in Job-shop Scheduling and Control from the Ecole Centrale de Lille. He is a member of IEEE/SMC and a lecturer for IEEE/SMC Journal. His teaching and research interests focus on the areas of production control, production planning, computer science, discrete and dynamic programming and CIM.

He co-supervised two Ph.D theses: the first in the area of the application of genetic algorithms to solve job-shop scheduling; the second in the area of the application of fuzzy logic to solve job-shop scheduling in uncertain environment.

Professor **Pierre Borne** graduated from the University of Lille in 1967 and obtained the Master degrees in Applied Mathematics, Physics, Mechanics and Electronics in 1968. In 1968 he also obtained the Diploma of Engineer of the IDN (French "Grande Ecole" with the new name "Ecole Centrale de Lille"). He obtained the Ph.D degree in Automatic Control in 1970 and Dsc. of Physics in 1976. He is now Professor of Automatic Control, head of the Automatic Control Department and Scientific Director of the Ecole Centrale de Lille.

He is involved in various international activities with IEEE/SMC Society and with IMACS. In 1996 he became a Fellow IEEE.

His research interests include stability analysis of non-linear systems, and various aspects of robust control of non-linear and complex systems.

1. Introduction

Scheduling and planning are difficult problems with a long varied history in the areas of operations research and artificial intelligence, and they continue to be active areas of research because a good scheduling algorithm can enhance productivity in various fields such as flexible manufacturing systems, production planning, computer design, logistics, communication, etc.

Scheduling can be defined as a problem of finding the optimal sequence for executing a finite set of operations under a certain set of constraints which must be satisfied. A scheduler usually attempts to maximize the utilization of individuals and / or machinery and minimize the time required to complete the entire process being scheduled. Therefore the scheduling problem is very hard to solve. It is then quite difficult to obtain an optimal solution

for satisfying real time constraints using exact methods such as branch and bound, and integer programming techniques[1] require considerable computing time and / or complex mathematical formulations. Recently stochastic search techniques such as simulated annealing, tabu search, and Genetic Algorithms (GAs) have been used to solve job-shop scheduling problems and give good solution close to the optimal one.

In this article we apply the GAs in order to provide a flexible job -shop scheduling which is to minimize the makespan of jobs. In simple genetic algorithms, the chromosome (potential solution) is encoded into a binary string; this representation was used by Nakano and Yamada [2, 3] to solve job-shop scheduling problems, but it is not practical because some correction process must be introduced. For this reason some variations on standard genetic operators must be tried.

- A specific genetic representation (or encoding) depending on problems in order to determine the feasible solutions for job -shop optimization problems.
- The initial population greatly influences the results. That is why, finding a method to help us produce those solutions will probably improve the solving quality. In the classical genetic algorithms, we can create an initial population randomly, but this is not possible with scheduling problems because a set of constraints (e.g. precedence and resources constraints)must be satisfied. For this reason we should find an efficient method able to generate the initial population, in order to accelerate convergence[4].

Section 2 describes the flexible job- shop scheduling. In Section 3, a description of genetic algorithms centred on scheduling problems is made. The implementational operators of the proposed methodology and the experimental results are presented in Section 4. Finally the discussion and conclusion are presented in Section 5.

2. The Flexible Job -shop Problem

The problem of flexible job -shop scheduling presents two difficulties. The first one is the assignment of each operation to a machine, and the second one is the scheduling of this set of operations in order to minimize our criterion.

The data and constraints of our problem are as follows:

- there are N jobs, indexed by j, and these jobs are independent of one another;
- each job j has an operating sequence which is called G_j;
- each operating sequence G_j is an ordered series of x_j operations $O_{i,j}$ for i = 1... x_j;
- the realization of each operation $O_{i,j}$ requires a resource or a machine selected from a set of machines, $M_{i,j}$ = {$M_{i,j,k}$, k = 1,2, ...,M / M is the total number of machines existing in the shop} which implies that an assignment problem exists;
- there is a predefined set of processing times; for a given machine, and a given operation, the processing time is defined and called $P_{i,j,k}$;
- a started operation runs to completion (non preemption condition);
- each machine can perform operations one after another (resource constraints).

Then, the aim of the search is to find a job sequence for each machine, so that the resulting schedule is feasible and the timespan to complete all jobs (the makespan of schedule) is minimized.

3. Genetic Algorithms in Scheduling Problems

3.1 The Concept

Genetic Algorithms (GAs) are general- purpose optimization algorithms with a probabilistic component that provides a means for searching poorly understood, and irregular spaces. The first rigorous description of the genetic algorithms process was given by Holland in 1960 [5]. Primarily the GAs have been used in two major areas : optimization and machine learning. For example in an optimization application, GAs have been used in many diverse fields such as function optimization, image processing, the travelling salesman

problem, scheduling problem, system identification and control.

Genetic algorithms are a group of global optimization methods based on simulated evolution, which simultaneously evaluates many points in the search space, and it is more likely to converge toward the global solution. In GAs, a set of variables for a given problem is encoded into a string (or other coding structure) similar to a chromosome in nature. Each string contains a possible solution to the problem. The GAs apply operators inspired by the mechanics of natural selection to a population of binary strings (or other structures) encoding the parameter space on each generation, explore different areas of the parameter space, and then orientate the search to regions where there is high probability of finding improved performance. A major difference between GAs and traditional optimization methods is that GAs operate on a population of chromosomes, searching the peaks in parallel, while traditional methods will consider one solution at a time. By working with a population of solutions, the algorithm can effectively seek for many local optima and thereby increase the likelihood of finding the global optima.

Simple GAs begin with a population randomly generated (or by other methods) and evolve towards a better solution by applying the principle of survival of the fittest to successively produce better approximations to a solution. On each generation of the GAs, a new set of approximations is created by the process of selecting individuals according to their level of fitness in population and of breeding them together using genetic operators such as crossover and mutation [6].

The structure of simple GAs is described as follows:

Step 1 : Define a genetic representation of the problem

Step 2 : Create a first population of chromosomes

Step 3 : Create a new chromosome by mating current chromosomes; apply mutation and crossover as the parent chromosomes mate

Step 4 : Delete members of the population to make room for new chromosomes

Step 5 : Evaluate the new chromosomes and insert them into the population

Step 6: If time is up, stop and return the best chromosome; else, go to Step 3

Genetic algorithms have some components that must be designed when the use of GAs to solve a problem is contemplated. These components make the syntax of the chromosomes. The evaluation of these chromosomes has been made by applying a set of genetic operators on these chromosomes. Other decisions such as the method for selecting parents must also be made, but these considerations are usually dependent on the particular problem to solve [6, 7].

To solve the NP-complete problem, the traditional representation of the chromosome (binary coding) and the manner how to generate the first population (random generation) are not convenient because we must observe the ordering dependencies. On these grounds, we should find a new chromosome syntax to fit the problem and other methods for creating the initial population.

3.2 Genetic Representation

Selection of a representational scheme of solution is a basic and essential prerequisite for a successful application of GAs. Traditionally, chromosomes are simple vectors of the form $(a_1,a_2,\ldots,a_n)$ where a_i is called a gene or allele. The values taken by a_i are from a set of symbols called the alphabet of the problem. A set of variables for a given problem is generally encoded into a binary string, in the binary code the alphabet consisting of symbols "1" and "0". But this simple representation of the chromosome is not convenient in case of representing complicated problems such as our flexible job-shop scheduling problem, when a certain order such as a precedence and resource constraint [5, 6, 8, 9] should be observed.

For this reason we suggest a new coding of the chromosomes, called parallel representation, which is a directly feasible scheduling and gives all the necessary information to a foreman, and also permits to conjointly treat the assignment and scheduling problems. In the case of our problem, the chromosome is represented by a set of machines put in parallel and each machine is a vector which contains its assignment operations. These operations are represented by three terms. The first one is the order number of the operation in its operating sequence, the second one is the number of the job which this operation belongs to and the third one is the starting time of the operation if its assignment on the machine is definitive.This starting time is calculated by taking into

account all the constraints. Indeed, the parallel coding is presented as follows:

M1	($O_{i,j}$, J, $t_{i,j,1}$)	...
M2	($O_{i,j,1}$, J_1,$t_{i,j,2}$)	...
M3	...	...
M4	...	...

Figure 1. Parallel Coding

where:

$O_{i,j}$ represents the operation i of job J
$t_{i,j,k}$ is the starting time of the operation $O_{i,j}$ performed on machine M_k.

Example 1:

We shall consider three jobs and five machines. The operating sequences of these jobs are presented as follows:

Job 1: O1,1, O2,1, O3,1

Job 2: O1,2, O2,2, O3,2

Job 3: O1,3, O2,3

O1,1 is the first operation of job 1 and O2,1 is the second operation of job 1, etc.

In this example, where any operation can be performed no matter of what machine with different processing times, we have full flexibility. The processing time of these operations is shown in Table 1.

Table 1. Processing Time of the Operations

	M1	M2	M3
O1, 1	1	9	3
O2, 1	3	5	2
O3, 1	6	7	1
O1, 2	1	4	5
O2, 2	2	8	4
O3, 2	9	5	1
O1, 3	1	5	9
O2, 3	5	9	2

One chromosome has a parallel representational scheme indicated as below:

M1	(1, 1, 0)	(1, 2, 1)	(2, 2, 2)
M2	(2, 1, 1)	(1, 3, 6)	(3, 1, 11)
M3	(3, 2, 4)	(2, 3, 11)	

Figure 2. Parallel Representation of the Chromosome

3.3 Initial Population

The choice of the first generation plays an important part in the search for the good solution. Generally, when we deal with an optimization problem using a binary coding, the initial population is usually chosen randomly. But for the job -shop problem we must satisfy a set of constraints (precedence and resources constraints); in this case it is not possible to use a binary code and to generate a first population randomly. For these reasons we have designed the parallel representation of the chromosome, and we should, in order to vary our first population and to permit that our set of solutions evolves in a very large domain, use a combination of some methods. In this article we generate an initial population using a combination of these methods:

1. Use a set of solutions given by Constraints Logic Programming (CLP) as a first population [10].

2. Taking the solution of our problem solved by other methods such as branch and bound or temporal decomposition approach, we will then apply genetic operators, especially the mutation operators, to extend the population.

3. Use a combination of the priority rules[11]:

   a) SPT: high priority for the operation that has the Shortest Processing Time.

   b) LPT: high priority for the operation that has the Latest Processing Time.

   c) LM: high priority for the operation that permits a balance of the load of the machine.

This combination of rules gives a set of populations used as a first population of our genetic algorithms.

3.4 Selection

Selection is one of the most important elements of all GAs. Selection determines which individuals in the population will have all or some of its genetic material passed on to the next generation of individuals. The objective of the selection method practiced in genetic

algorithms is to give polynomial increasing trials to the fittest individuals. Generally we use for the selection the roulette wheel technique.

3.5 Crossover

In nature, crossover occurs when two parents exchange parts of their corresponding chromosomes. In a genetic algorithm, crossover recombines the genetic material in two parents chromosomes to make two children, in order to generate a better solution [5, 6, 9].

Child one is given by the following algorithm:

Step 1: Parent 1, Parent 2 and machine M_k are randomly selected.

Step 2: $\{O_{i,j}\}_k$ of Child 1 $\leftarrow$ $\{O_{i,j}\}_k$ of Parent 1

$I \leftarrow 1$

Step 3: While ($I < M$) do

If ($I \neq k$) then

Copy the non- existing operations of M_I of parent 2 into child 1

$I \leftarrow I + 1$

End if

End while

Step 4: $I \leftarrow k$

If (any operation of child 1 is missing) then

Scan M_k of parent 2 and copy the missing operation

End if

To obtain child 2 go to Step 2 and invert the role of parent 1 and parent 2.

Unfortunately when we have full flexibility (i.e. any operation can be performed by any machine), the sequence of operations defined by a chromosome may be incompatible with the precedence constraints of the operations, we create a cycle in the precedence constraint graph[12, 13]. Therefore some of the generated chromosomes define nonfeasible schedules. This problem is illustrated in the following example:

Let consider two jobs and two machines. The operating sequences of these jobs are like:

Job 1: O1,1, O2,1.

Job 2: O1,2, O2,2.

Suppose the chromosome is

M1	(2, 2, ?)	(1, 1, ?)
M2	(2, 1, ?)	(1, 2, ?)

Machine M1 should first execute operation 2 of job 2, but it cannot do this until operation 1 of job 2 has been completed; likewise M2 should first execute operation 2 of job 1, but it cannot do it until operation 1 of job 1 has been completed. A deadlock situation has arisen, therefore the chromosome does not define any feasible solution (the starting time is represented by symbol "?"). This case of illegal schedule is produced by Step 4 of the crossover algorithm, and we have a violation of the precedence constraints.

There are two possible ways of dealing with this problem:

1. To modify genetic operators so that they can always produce (through suitable manipulations) chromosomes which feasible schedules correspond to.

2. To define a different encoding where all chromosomes produce feasible schedules.

In this article, we use the first possibility. We replace Step 4 of the crossover algorithm by the following procedure:

New Step 4 of the crossover algorithm

We suppose that: $O_{i,j}$ is the missing operation. So we scan machine M_k by applying the following rules.

If ($I = 1$) then

Put $O_{i,j}$ at the beginning of machine M_k

End if

If ($I = x_j$) then

Put $O_{i,j}$ at the end of machine M_k

End if

If ($I \in]1, x_j[$) then

Find the row of $O_{i-1,j}$ and the row of $O_{i+1,j}$ in child 1

Put $O_{i,j}$ between the row of $O_{i-1,j}$ and the row of $O_{i+1,j}$ on machine M_k

End if

Example 2: We will use the same data as in Example 1.

Step 1: Suppose that : parent 1, parent 2 and machine M3 are randomly selected.

Parent 1

M1	(1, 1, 0)	(1, 2, 1)	(2, 2, 2)
M2	(2, 1, 1)	(1, 3, 6)	(3, 1, 11)
M3	(3, 2, 4)	(2, 3, 11)	

Parent 2

M1	(1, 1, 0)	(3, 1, 6)	(1, 3, 12)
M2	(2, 1, 1)	(1, 2, 6)	
M3	(2, 2, 10)	(3, 2, 14)	(2, 3, 15)

Step 2: Copy the operations assigned in M3 of parent 1 (respectively parent 2) in child 1

(respectively child 2) on machine M3

Child 1 in construction

M1			
M2			
M3	(3, 2, ?)	(2, 3, ?)	

Child 2 in construction

M1			
M2			
M3	(2, 2, ?)	(3, 2, ?)	(2, 3, ?)

Step 3: Copy the non- existing operations of M1 and M2 of parent 2 (respectively parent 1)

into child 1 (respectively child 2)

Child 1 in construction

M1	(1, 1, ?)	(3, 1, ?)	(1, 3, ?)
M2	(2, 1, ?)	(1, 2, ?)	
M3	(3, 2, ?)	(2, 3, ?)	

Child 2 in construction

M1	(1, 1, ?)	(1, 2, ?)	
M2	(2, 1, ?)	(1, 3, ?)	(3, 1, ?)
M3	(2, 2, ?)	(3, 2, ?)	(2, 3, ?)

Step 4: If any operation is missing:

If any operation is missing

No for child 2 => New chromosome

Yes for child 1=> operation 2 of job 2 is missing.

This operation is neither the first nor the last one in the operating sequence of job 2, and in this case we have O2,2 $\in]1, 3[$ we find the row of $O_{i-1,j}$ (O1,2) in child 1, this operation is placed in row two on machine M2, and if the row of $O_{i+1,j}$ (O3,2) in child 1 is equal to 1, we should put it before operation O3,2. In such a case we move all the operations to the next row and put O2,2 in the first row on machine M3.

We shall calculate the starting time of each operation through observing the precedence and resources constraints according to the formula indicated in Section 3.7. Finally we obtain:

Child 1

M1	(1, 1, 0)	(3, 1, 6)	(1, 3, 12)
M2	(2, 1, 1)	(1, 2, 4)	
M3	(2, 2, 10)	(3, 2, 14)	(2, 3, 15)

Child 2

M1	(1, 1, 0)	(1, 2, 1)	
M2	(2, 1, 1)	(1, 3, 6)	(3, 1, 11)
M3	(2, 2, 2)	(3, 2, 6)	(2, 3, 11)

3.6 Mutation

Another important genetic operator is mutation. Although mutation is important, it is secondary to crossover. It introduces some extra variability; it provides and maintains diversity in a population, typically works with a single chromosome, always creates another chromosome [6]. We will consider here two operators of mutation; the assigned mutation and the swap mutation.

A/ Assigned mutation:

In this case, based on the flexibility of our problem, the operation can be performed by one or more machines. The algorithm of the assigned mutation is as follows:

Step 1: One chromosome and one operation are randomly selected.

Step 2: Reassign this selected operation to another machine in the same position if possible, sticking to the precedence and resources constraints.

Example 3:

Step 1: Suppose that the following chromosome and operation 3 of job 1 are randomly selected (this operation is assigned on machine M2 in the third position)

M1	(1, 1, 0)	(1, 2, 1)	(2, 2, 2)
M2	(2, 1, 1)	(1, 3, 6)	
M3	(3, 2, 4)	(2, 3, 11)	**(3, 1, 13)**

Step 2: Reassign O3,1 on machine M3, and obtain the following chromosome

M1	(1, 1, 0)	(1, 2, 1)	(2, 2, 2)
M2	(2, 1, 1)	(1, 3, 6)	**(3, 1, 11)**
M3	(3, 2, 4)	(2, 3, 11)	

B/ Swap mutation:

The algorithm of the swap mutation is as follows:

Step 1: We randomly select one chromosome, one position, one direction and two machines.

Step 2:

If (direction = 0) then

do a left swap.

Else If (direction = 1) then

do a right swap.

End if

Example 4:

Step 1: assume that the following chromosome is randomly selected and machine M1, machine M2, position two are randomly selected.

M1	(1, 1, 0)	(1, 2, 1)	
M2	(2, 2, 2)	(1, 3,10)	(3, 1, 15)
M3	(2, 1, 1)	(3, 2, 10)	(2, 3, 15)

Step 2:

First case: direction = 0 → Do left swap, we obtain the following chromosome:

M1	(1, 1, 0)	(1, 2, 1)	
M2	**(2, 1, 1)**	(1, 3, 6)	(3, 1, 11)
M3	**(2, 2, 2)**	(3, 2, 6)	(2, 3, 11)

Second case: direction = 1 → Do right swap, we obtain the following chromosome:

M1	(1, 1, 0)	(1, 2, 1)	
M2	(2, 2, 2)	(1, 3,10)	**(2, 3, 15)**
M3	(2, 1, 1)	(3, 2, 10)	**(3, 1, 11)**

3.7 Evaluation Function

The evaluation function (fitness function) is the link between genetic algorithms and the problem to solve. An evaluation function takes a chromosome as input and returns a number or list of numbers that is a measure of the chromosome's performance on the problem to solve. Evaluation functions play the same role in GAs as the environment in natural evolution.

The evaluation of the chromosomes is a critical task of the system. Furthermore, the evaluation function must indicate what about the schedules that makes them seem good or bad to the system's users [14, 15]. In our case, the fitness function is the minimizing of a makespan. For each chromosome we have M machines, and we calculate the time that it takes for this machine

to execute all the assigned operations and we take the maximum time of M machines. This time is calculated as follows:

Step 1: Take the starting time of the last operation ($O_{i,j}$) performed on Machine k (called T_k).

Step 2: Add T_k to the processing time of this last operation ($P_{i,j,k}$), and obtain ($T_k + P_{i,j,k}$)

These two steps will be repeated for all machines in the chromosome. The makespan of this chromosome is calculated as follows:

MAX $[(T_{k1} + P_{i,j,k1}), (T_{k2} + P_{i,j,k2}), ..., (T_{kM} + +P_{i,j,kM}])$.

4. Computational Results

This Section reports on the experiments made to show the performance of the proposed genetic algorithms. The implementation has been in a C-language program.

Firstly, we propose a minimization of the makespan of a small problem, 5 machines and 3 jobs using different methods, as previously mentioned, to generate the initial population.

Operating sequences of these jobs are:

Job 1: O1,1, O2,1, O3,1.

Job 2: O1,2, O2,2, O3,2.

Job 3: O1,3, O2,3.

Table 2. Processing time of the operations

	M1	M2	M3	M4	M5
O1, 1	1	9	3	7	5
O2, 1	3	5	2	6	4
O3, 1	6	7	1	4	3
O1, 2	1	4	5	3	8
O2, 2	2	8	4	9	3
O3, 2	9	5	1	2	4
O1, 3	1	5	9	3	2
O2, 3	5	9	2	4	3

Two methods have been used in order to generate the first population:

1st case: Temporal decomposition method: obtain 50 chromosomes through extending the solution given by the temporal decomposition method using our two types of mutations (assigned and swap mutations) in the following proportion:

- Swap mutation with probability = 0.75
- Assigned mutation with probability = 0.25

2nd case: Constraint logic programming: this method gives 10 chromosomes, a succession of our mutations will take place for extending the first population and obtaining 50 chromosomes with the same probability of mutation as presented in the first case.

Our genetic algorithm is applied with the following operators

- Size of Population (SP) = 50
- Crossover Probability (CP) = 0.65
- Assigned Mutation Probability = Swap Mutation Probability = 0.05
- Number of Generation (NB GEN) = 20

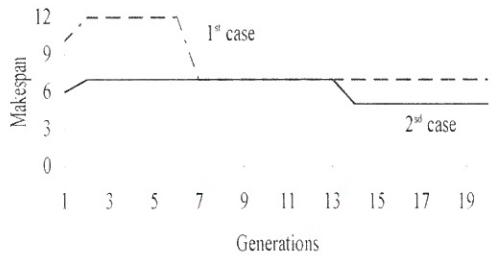

Figure 3. Comparison Between the 1st Case and the 2nd Case

We can notice that: when our first population contains some varied chromosomes, the results can quickly be improved.

Secondly, the use of our method to another problem, that has an important size, is proposed. Here 10 jobs and 10 machines are considered and the problem presents full flexibility (any machine can perform any operation). Each job has 3 operations in its operating sequence.

Processing times of these operations are shown in Table 3.

Table 3. Operating sequences of jobs and their processing times on all machines

	Ops	Order	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
J1	O1,1	1,3,2	1	4	6	9	3	5	2	8	9	5
	O2,1		3	2	5	1	5	6	9	5	10	3
	O3,1		4	1	1	3	4	8	10	4	11	4
J2	O1,2	2,1,3	4	8	7	1	9	6	1	10	7	1
	O2,2		2	10	4	5	9	8	4	15	8	4
	O3,2		6	11	2	7	5	3	5	14	9	2
J3	O1,3	1,2,3	8	5	8	9	4	3	5	3	8	1
	O2,3		9	3	6	1	2	6	4	1	7	2
	O3,3		7	1	8	5	4	9	1	2	3	4
J4	O1,4	1,2,3	5	10	6	4	9	5	1	7	1	6
	O2,4		4	2	3	8	7	4	6	9	8	4
	O3,4		7	3	12	1	6	5	8	3	5	2
J5	O1,5	2,3,1	6	1	4	1	10	4	3	11	13	9
	O2,5		7	10	4	5	6	3	5	15	2	6
	O3,5		5	6	3	9	8	2	8	6	1	7
J6	O1,6	1,2,3	8	9	10	8	4	2	7	8	3	10
	O2,6		7	3	12	5	4	3	6	9	2	15
	O3,6		4	7	3	6	3	4	1	5	1	11
J7	O1,7	3,2,1	5	4	2	1	2	1	8	14	5	7
	O2,7		3	8	1	2	3	6	11	2	13	3
	O3,7		1	7	8	3	4	9	4	13	10	7
J8	O1,8	3,1,2	8	3	10	7	5	13	4	6	8	4
	O2,8		6	2	13	5	4	3	5	7	9	5
	O3,8		5	7	11	3	2	9	8	5	12	8
J9	O1,9	1,2,3	3	9	1	3	8	1	6	7	5	4
	O2,9		4	6	2	5	7	3	1	9	6	7
	O3,9		8	5	4	8	6	1	2	3	10	12
J10	O1,10	3,2,1	9	2	4	2	3	5	2	4	10	23
	O2,10		3	1	8	1	9	4	1	4	17	15
	O3,10		4	3	1	6	7	1	2	6	20	6

This example makes a combination of some methods for generating the initial population and shows the advantage of using this method for improving the problem results.

1st case: we take the solution yielded by the temporal decomposition method (in this example the best makespan given by this method is equal to 16 units of time) as for the first chromosome, and increase this population using our mutations with the following rate:

- Assigned Mutation Probability = 0.5

- Swap Mutation Probability = 0.5

We stop this process when 50 chromosomes are obtained, and apply our genetic algorithms with the following parameters

- Size of Population (SP) = 50

- Crossover Probability (PC) = 0.75

- Assigned Mutation Probability (AMP) = 0.05

- Swap Mutation Probability (SMP) = 0.05

- Number of Generation (NB GEN) = 60

2nd case: we use the constraint logic programming for generating a set of solutions, this set contains 50 valid schedules representing our first population. Then our genetic algorithm

is applied to improve this solution and obtain a valid schedule with a makespan as nearest the optimal one as possible, using the same rate of genetic operators as presented in the first case.

3rd case: we use three methods to generate the first population as follows:

1) Take the solution given by the temporal decomposition method as the first chromosome and apply different mutations to extend it and obtain 10 chromosomes.

2) Generate 20 chromosomes using the SPT rule.

3) Generate 20 chromosomes using the LPT rule.

Then we take this set of 50 chromosomes and introduce it into our GAs as an initial population, and we apply the genetic operators on improving the solution towards an optimal one using a similar rate of genetic operators as previously mentioned.

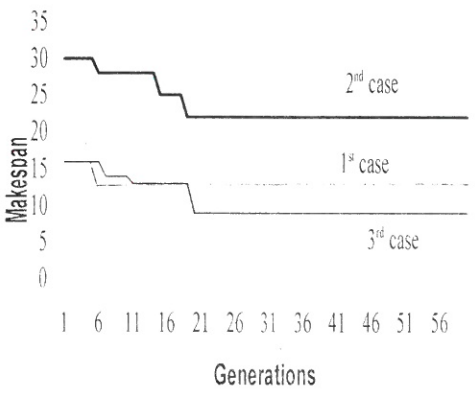

Figure 4. The Best Makespan of Different Methods for Creating the First Population

In the first case, we created the initial population using only a succession of mutations. We obtained 13 units of time as a makespan, and ameliorated the solution by over 18 %.

In the second case, the constraint logic programming was used. The best makespan found was equal to 22 units of time.

In the third case, we generated the first population using two methods such as temporal decomposition method and priority rules. The makespan with this method was equal to 9 units of time.

It is interesting to note that a good performance of the proposed algorithm is also observed in a large size problem. We can notice that the makespan in the second case is the smaller one, effectively in the first case. Between the first population and the last one, the makespan is improved by 18% and in the second case the improvement is of over 26% but for the third case, when we use some different methods to generate the initial population, we have very good results, the makespan given by the last method is the smaller one and the improvement between the first and the last population is higher than 43%. Note further that if we run a GAs based algorithm, with the possible greatest diversified initial population by applying some other different methods, the problem results will be good. Indeed it is one of the reasons why the performance of the algorithm is dependent on the initial seed point.

5. Conclusion

The application of genetic algorithms to a flexible job-shop scheduling problem with real-world constraints has been defined. New genetic operators have been proposed using the original idea of combining the assignment and scheduling problems. In this paper we demonstrated that choosing a suitable representation of chromosomes (parallel representation), genetic operators and the initial population using a combination of some optimization methods such as temporal decomposition method, constraint logic programming and the priority rules, was an important step to getting better results. Simulation results show that the results presented in the second case are better than the solution given by the other case. This confirms the effectiveness of using a combination of varied methods to create an initial population.

Proper selection of genetic parameters for an application of GAs is still an open issue. These parameters (crossover probability, mutation probability, population size,...) are usually selected heuristically. There are no guidelines on the exact strategies to be adopted for different problems. In this work we have looked upon a fixed population size. Crossover probability and mutation probability are kept variable, having a high initial value, then decreasing and finally increasing again. Mutation could also have been used, which had combined the merits of both genetic search and gradient descent search for convergence

acceleration. Investigation is therefore necessary for determining these controlling parameters properly, in order to improve the performance of the proposed method. Again, a modification of the fitness function so as to incorporate the information existing in job - shop scheduling will be part of a further investigation.

REFERENCES

1. CARLIER J. and CHRETIENNE, P., **Problèmes d'ordonnancement: Modélisation / complexité / algorithmes**, ÉDITIONS MASSON, 1988.

2. NAKANO, R. and YAMADA, T., **Conventional Genetic Algorithm for Job Shop Problems,** Proceedings of the fourth International Conference on Genetic Algorithms, MORGAN KAUFMANN; San Mateo, CA, USA, 1991, pp. 477-479.

3. YAMADA, T. and NAKANO, R., **A Genetic Algorithm Applied To Large-scale Job-shop Problems,** in R. Männer et al (Eds.) Parallel Problem Solving From Nature, Amsterdam, 1992, pp. 281-290.

4. MESGHOUNI, K., HAMMADI, S. and BORNE, P., **Production Job-shop Scheduling Using Genetic Algorithms,** Proceedings of IEEE /SMC, Vol. 2, Beijing, China, October 14-17, pp. 1519-1524.

5. GOLDBERG, D.E., **Genetic Algorithms in Search, Optimization, and Machine Learning**, ADDISON- WESLEY, 1989.

6. SYSWERDA, G., **Schedule Optimization Using Genetic Algorithm,** in Handbook of Genetic Algorithm, VAN NOSTRAND REINHOLD, New York, 1990.

7. PARK, I. J. and PARK, C. H., **Application of Genetic Algorithm to Job-shop Scheduling Problems With Active Schedule Constructive Crossover** , Proceedings of IEEE/SMC, Vol. 1, October 22-25, 1995, pp. 530-535.

8. PORTMANN, M.C., **Genetic Algorithms and Scheduling: A State of the Art and Some Propositions,** Proceedings of the Workshop on Production Planning and Control, Mons, Belgium, September 9-12, 1996.

9. TSUJIMURA, Y., CHENG, R. and GEN, M., **Improve Genetic Algorithms for Job-shop Scheduling Problems**, ENGINEERING DESIGN & AUTOMATION, Vol. 3, No. 2, 1997, pp. 133-144.

10. MESGHOUNI, K., PESIN, P., HAMMADI, S., TAHON, C. and BORNE, P., **Genetic Algorithms - Constraint Logic Programming. Hybrid Method for Job Shop Scheduling,** OE/IFIP/IEEE International Conference on Integrated and Sustainable Industrial Production, Lisbon, Portugal, May 14-16, 1997, pp. 151-160.

11. MESGHOUNI, K., HAMMADI, S. and BORNE, P., **Hybrid Representation for Genetic Algorithm To Solve Flexible Job Shop Scheduling,** Proceedings of 15th IMACS, Vol.5, Berlin, Germany, August 24-29, 1997, pp. 433-438.

12. CROCE, F., TADEI, R. and VOLTA, G., **A Genetic Algorithm for the Job Shop Problem**, COMPUTER AND OPERATIONS RESEARCH, Vol. 22, No.1, 1995, pp.15-24.

13. MESGHOUNI, K., HAMMADI, S. and BORNE, P., **Parallel Genetic Operators for Flexible Job-shop Scheduling**, Proceedings of 1st International Conference on Engineering Design & Automation, Bangkok, Thailand, March 18-21, 1997, pp. 626-630.

14. GEN, M., TSUJIMURA, Y. and KUBOTA, E., **Solving Job-shop Scheduling Problems By Genetic Algorithm,** Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vol. 2, October 2-5, 1994, pp. 1577-1582.

15. CARLIER, J. and PINSON, E., **An Algorithm for Solving the Job Shop Problem**, MANAGEMENT SCIENCE, Vol. 35, No. 2, 1989, pp. 164-176.