*Int. J. Appl. Math. Comput. Sci., 2004, Vol. 14, No. 1, 91–103*

# EVOLUTIONARY ALGORITHMS FOR JOB-SHOP SCHEDULING

KHALED MESGHOUNI*, SLIM HAMMADI*
PIERRE BORNE*

* Ecole Centrale de Lille, LAIL - UMR 8021 BP 48
59651 Villeneuve d'Ascq Cedex, France
e-mail: {khaled.mesghouni, slim.hammadi, p.borne}@ec-lille.fr

This paper explains how to use Evolutionary Algorithms (EA) to deal with a flexible job shop scheduling problem, especially minimizing the makespan. The Job-shop Scheduling Problem (JSP) is one of the most difficult problems, as it is classified as an NP-complete one (Carlier and Chretienne, 1988; Garey and Johnson, 1979). In many cases, the combination of goals and resources exponentially increases the search space, and thus the generation of consistently good scheduling is particularly difficult because we have a very large combinatorial search space and precedence constraints between operations. Exact methods such as the branch and bound method and dynamic programming take considerable computing time if an optimum solution exists. In order to overcome this difficulty, it is more sensible to obtain a good solution near the optimal one. Stochastic search techniques such as evolutionary algorithms can be used to find a good solution. They have been successfully used in combinatorial optimization, e.g. in wire routing, transportation problems, scheduling problems, etc. (Banzhaf *et al.*, 1998; Dasgupta and Michalewicz, 1997). Our objective is to establish a practical relationship between the development in the EA area and the reality of a production JSP by developing, on the one hand, two effective genetic encodings, such as parallel job and parallel machine representations of the chromosome, and on the other, genetic operators associated with these representations. In this article we deal with the problem of flexible job-shop scheduling which presents two difficulties: the first is the assignment of each operation to a machine, and the other is the scheduling of this set of operations in order to minimize our criterion (e.g. the makespan).

**Keywords:** job-shop scheduling, evolutionary algorithms, parallel representation

## 1. Introduction

Several problems in various industrial environments are combinatorial. This is the case for numerous scheduling and planning problems. Generally, it is extremely difficult to solve this type of problems in their general form. Scheduling can be defined as a problem of finding an optimal sequence to execute a finite set of operations satisfying most of the constraints. The problem so formulated is extremely difficult to solve, as it comprises several concurrent goals and several resources which must be allocated to lead to our goals, which are to maximize the utilization of individuals and/or machines and to minimize the time required to complete the entire process being scheduled. Therefore, the exact methods such as the branch and bound method, dynamic programming and constraint logic programming need a lot of time to find an optimal solution. So, we expect to find not necessarily an optimal solution, but a good one to solve the problem. Realistically, we are satisfied by obtaining a good solution near the optimal one. New search techniques such as genetic algorithms, simulated annealing (Kirkpatrick *et al.*, 1983) or tabu search (Golver *et al.*, 1993) are able to lead to our objective i.e. to find near-optimal solutions for a wide range of combinatorial optimization problems.

In this article, we propose improved evolutionary algorithms (EAs) for solving a JSP. EAs are search and optimization algorithms inspired by the process of natural evolution. They employ a probabilistic search for locating a globally optimal solution. They have many advantages. They are robust in the sense that they provide a set of solutions near the optimal one on a wide range of problems. They can be easily modified with respect to the objective function and constraints. In this article we describe the incorporation of the scheduling specific knowledge in operators and in the chromosome representation. In this context, parallel representations of the chromosome and some genetic operators have been created.

This article is organized as follows: Section 2 contains a detailed description and formulation of our flexible job-shop scheduling. In Section 3, a description of evolutionary algorithms adapted to scheduling problems is presented. Implementation details of the proposed methodology and experimental results are presented in Section 4. Finally, the discussion and conclusion are presented in Section 5.

## 2. Description of Job-Shop Scheduling

The task of production scheduling consists in the temporal planning of the processing of a given set of orders. The processing of an order corresponds to the production of a particular product. It is accomplished by the execution of a set of operations in a predefined sequence on certain resources, subject to several constraints. The result of scheduling is a schedule showing the temporal assignment of operations of orders to the resources to be used. In this study, we consider a flexible job shop problem. Each operation can be preformed by some machines with different processing times, so that the problem is known to be NP hard. The difficulty is to find a good assignment of an operation to a machine in order to obtain a schedule which minimizes the total elapsed time (makespan).

The structure of the scheduling problem can be described as follows:

- consider a set of $N$ jobs $\{J_j\}_{1\leq j\leq N}$; these jobs are independent of one another;
- each job $J_j$ has an operating sequence, called $G_j$;
- each operating sequence $G_j$ is an ordered series of $x_j$ operations, $O_{i,j}$ indicating the position of the operation in the technological sequence of the job;
- the realization of each operation $O_{i,j}$ requires a resource or a machine selected from a set of machines, $\{M_k\}_{1\leq k\leq M}$; $M$ is the total number of machines existing in the shop, this implying the existence of an assignment problem;
- there is a predefined set of processing times; for a given machine, and a given operation, the processing time is denoted by $P_{i,j,M_k}$;
- an operation which has started runs to completion (non-preemption condition);
- each machine can perform operations one after another (resource constraints);
- the time required to complete the whole job constitutes the makespan $C_{\max}$.

Our objective is to determine the set of completion times for each operation $\{C_{i,j,k}\}_{1\leq i\leq X_j,\ 1\leq j\leq N,\ 1\leq k\leq M}$ which minimizes $C_{\max}$.

## 3. Evolutionary Algorithms

Evolutionary algorithms are general purpose search procedures based on the mechanisms of natural selection and population genetics. These algorithms have been applied by many users in different areas of engineering, computer science, and operations research. Current evolutionary approaches include evolutionary programming, evolutionary strategies, genetic algorithms, and genetic programming (Banzhaf *et al.*, 1998; Dasgupta and Michalewicz, 1997; Fonseca and Fleming, 1998; Goldberg, 1989; Quagliarella *et al.*, 1998).

### 3.1. How Do EAs Work?

Evolutionary algorithms are inspired by genetic algorithms. There are a relatively new contribution to the field of artificial intelligence. They use various computational models of evolutionary processes to solve problems on a computer. They are based on the mechanism of natural selection in biological systems. Evolutionary algorithms use a structured but randomized way to utilize genetic information in finding a new search direction. They work by defining a goal in the form of a quality criterion and then use this goal to measure and compare solution candidates in a stepwise refinement of set data structures. If successful, an EA will return an optimal solution or a near-optimal one after a number of iterations. The improvement process is accomplished using genetic operators such as crossover and mutation. There are several variants of evolutionary algorithms, and also many hybrid systems which incorporate various features of these paradigms. However, the structure of any evolutionary method is very much the same. After defining a genetic representation, the simple structure of evolutionary algorithms is shown as follows:

1. **Selection of the chromosome structure**
   The problem has to be translated into a chromosome representation. Each gene of the chromosome corresponds to a decision variable of the problem. According to problem complexity, the chromosome structure can be either conventional (a binary string) or not (reals, a series or a sequence of orders, a parallel form, etc.).
2. **Initialization of the population of chromosomes**
   The initial population can be generated at random if the problem structure allows it.
3. **Perform genetic operations on chromosomes**
   Some operators are introduced in genetic algorithms to produce a solution. Among them there are two categories of operators: crossover and mutation.
4. **Evaluation chromosomes**
   During evolutionary generation, an evaluating system is set up to assess the chromosomes and to select those chromosomes that are fit enough for the next generation.

### 3.2. Encoding Requirements

Problems of encoding have been observed in the GA literature (Dasgupta and Michalewicz, 1997), where slightly different problems require completely different genetic encodings for a good solution to be found. Choosing a good representation is a vital component of solving any search problem. However, choosing a good representation for a problem is as difficult as choosing a good search algorithm for a problem. Care must be taken to adopt both representational schemes and the associated genetic operators for an effective genetic search. Traditionally, chromosomes are simple binary vectors. This simple representation is an excellent choice for the problems in which a point naturally maps into a string of zeros and ones. Unfortunately, this approach cannot usually be used for real-word engineering problems such as combinatorial ones (Portman, 1996). A modification should be suggested, such as the permutation of a basic string like that used for a Travelling Salesman Problem (TSP) (Della Croce *et al.*, 1995). For instance, consider a TSP with $N$ cities. A permutation of the numbers from 1 to $N$ is a chromosome, which codes a possible solution. This means that every symbol has to appear only once so that the chromosome makes sense. An illegal solution can obviously be obtained by applying traditional genetic operators (crossover and mutation). Some different encodings are proposed in the literature (Baghi *et al.*, 1991; Bruns, 1993; Uckun *et al.*, 1993). These encodings are split into two categories. The first one is the direct chromosome representation. We can represent a scheduling problem by using the schedule itself as a chromosome. This method generally requires developing specific genetic operators. The second is the indirect chromosome representation: the chromosome does not directly represent a schedule, and transition from the chromosome representation to a legal schedule builder (decoder) is needed prior to evaluation.

In this article, we chose a direct representation to give viability and legibility to a chromosome and a simplicity of utilization for a user. We suggest two new direct representational chromosomes with their genetic operators.

#### 3.2.1. First Approach

*Encoding problem*

The first approach is based on the parallel machine encoding (PME) (Mesghouni, 1999; Mesghouni *et al.*, 1998) which represents directly feasible scheduling, gives a user all the necessary information, and also permits to treat jointly 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 is the order number of the operation in its operating sequence,

| | | |
|---|---|---|
| $M_1$ | $(i, j, t_{i,j,M_1})$ | $\cdots$ |
| $M_2$ | $(i', j', t_{i',j',M_2})$ | $\cdots$ |
| $\cdots$ | $\cdots$ | $\cdots$ |
| $M_m$ | $\cdots$ | $\cdots$ |

Fig. 1. Parallel machine encoding.

the second is the number of the job to which this operation belongs and the third is the starting time of the operation if its assignment on this machine is definitive. This starting time is calculated taking into account all the constraints. Indeed, the parallel machine encoding is presented in Fig. 1. In a general manner, the line of the chromosome is represented as follows:

$$M_k : (i, j, t_{i,j,M_k}), (i', j', t_{i',j',M_k}), \ldots,$$

where $i$ represents the operation to be executed by machine $M_k$, $j$ is the job to which this operation (i) belongs and $t_{i,j,M_k}$ is the starting time of the operation $i$ of job $j$ on machine $M_k$. This time is calculated taking into account the precedence and resources constraints.

**Example 1.** Three jobs and five machines are considered. The operating sequences of these jobs are as follows (the data of this example are used in all of the examples presented in this article):

$$\begin{aligned} \mathrm{Job}_1\text{:} &\quad O_{1,1} - O_{2,1} - O_{3,1}, \\ \mathrm{Job}_2\text{:} &\quad O_{1,2} - O_{2,2} - O_{3,2}, \\ \mathrm{Job}_3\text{:} &\quad O_{1,3} - O_{2,3}, \end{aligned}$$

with $O_{1,1}$ representing the first operation of $\mathrm{Job}_1$ and $O_{2,1}$ the second operation of the same job $(\mathrm{Job}_1)$, etc. We have a total flexibility, as any operation can be performed equally well by any machine with different processing times. According to the machine used, the processing time of operations is different as described Table 1. One possible chromosome has the following parallel machine representation:

| | | | |
|---|---|---|---|
| $M_1$ | $(1, 2, 0)$ | $(2, 2, 1)$ | |
| $M_2$ | $(2, 3, 2)$ | | |
| $M_3$ | $(1, 1, 0)$ | $(2, 1, 3)$ | $(3, 2, 5)$ |
| $M_4$ | $(3, 1, 5)$ | | |
| $M_5$ | $(1, 3, 0)$ | | |

This encoding possesses some advantage, as it gives directly a feasible schedule. The obtained solution contains all information which a user needs, e.g. which machines will execute every operations and at what time, which are loaded machines and which machines are underused. This allows the user to best manage park machines, and thereby the production. ♦

Table 1. Processing time of the operations on different machines.

| | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
|---|---|---|---|---|---|
| $O_{1,1}$ | 1 | 8 | 3 | 7 | 5 |
| $O_{2,1}$ | 3 | 5 | 2 | 6 | 4 |
| $O_{3,1}$ | 6 | 7 | 1 | 4 | 3 |
| $O_{1,2}$ | 1 | 4 | 5 | 3 | 8 |
| $O_{2,2}$ | 2 | 8 | 4 | 9 | 3 |
| $O_{3,2}$ | 9 | 5 | 1 | 2 | 4 |
| $O_{1,3}$ | 1 | 8 | 9 | 3 | 2 |
| $O_{2,3}$ | 5 | 9 | 2 | 5 | 3 |

*Initial population*

The choice of the first population is an important part in the search for a good solution. Generally, when we deal with an optimization problem using a binary coding, the initial population is usually chosen randomly. Such an encoding is efficient when our problem does not impose any order. In this case, the initial population is usually chosen at random. But in a combinatorial problem such as job-shop scheduling, some constraints such as precedence and resources constraints must be satisfied. In this case, the binary representation is not convenient and another chromosome syntax must be found to fit the problem. For these reasons, we have designed a parallel representation of the chromosome, and in order to create and to permit our set of solutions to evolve in a very large domain, we shall use a combination of some methods. In this article we generate an initial population using a combination between the following methods:

1. We use a set of solutions given by Constraint Logic Programming (CLP) as a first population (Mesghouni *et al.*, 1999).
2. Given a solution to our problem found by other methods, such as the branch and bound method or the temporal decomposition approach, we then apply genetic operators, especially the mutation ones, to extend the population.
3. We use a combination of the following priority rules:
   - SPT: a high priority for the operation that has the Shortest Processing Time,
   - LPT: a high priority for the operation that has the Longest Processing Time,
   - LM: a high priority for the operation that permits to balance the load of the machine.

*Crossover operator*

The predominant operator used is crossover. It involves combining elements from two parent chromosomes into one or more child chromosomes. The role of the crossover is to generate a better solution by exchanging information contained in the current good ones (Michalewicz, 1992). The idea is that useful building blocks for the solution of a problem are accumulated in the population and that crossover allows for the aggregation of good building blocks into ever better solutions to the problem. For scheduling problems, different crossover operators have been designed and presented in the literature (Portman, 1996; Syswerda, 1990). Inspired by them, we propose two new crossover operators adapted to our encoding. These operators always generate new legal offspring (a child).

Child 1 is given by the following algorithm (Mesghouni, 1999):

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

**Step 2.** $\{O_{i,j}\} \in M_k$ of Child 1 $\leftarrow \{O_{i,j}\} \in M_k$ of Parent 1; $I \leftarrow 1$.

**Step 3.** ($M$ is the total number of machines)

```
While (I < M) do
    If (I ≠ k) then
        Copy the non-existing operations of M_I
        of Parent 2 into Child 1
        I ← I + 1
    End If
End while
```

**Step 4.** $I \leftarrow k$

We suppose that $O_{i,j}$ is the missing operation. So we scan machine $M_k$ 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 ∈ ]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 that
    of O_{i+1,j} on machine M_k
End If
```

To obtain Child 2, go to Step 2 and invert the roles of Parents 1 and 2.

However, it is necessary to be very careful as for the problem which admits a total flexibility (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 (Croce *et al.*, 1995) and some of the generated chromosomes define infeasible schedules. This problem is illustrated in the following example: Consider two jobs and two machines. The operating sequences of these jobs are presented as follows:

$$\begin{aligned} \text{Job}_1\text{:} &\quad O_{1,1} - O_{2,1}, \\ \text{Job}_2\text{:} &\quad O_{1,2} - O_{2,2}. \end{aligned}$$

Suppose that the chromosome is

| | | |
|---|---|---|
| $M_1$ | $(2,2,?)$ | $(1,1,?)$ |
| $M_2$ | $(2,1,?)$ | $(1,2,?)$ |

Machine $M_1$ should first execute Operation 2 of Job 2, but it cannot do this until Operation 1 of Job 2 has been completed. Likewise, $M_2$ should first execute Operation 2 of Job 1, but it cannot do so until Operation 1 of Job 1 has been completed. A deadlock situation arises and therefore the chromosome does not define any feasible solution (the starting time is represented by the symbol '?'). This case of an illegal schedule is produced by a violation of the precedence constraints.

There are two possible ways of solving this problem:

1. By modifying genetic operators so that they can always produce (through suitable manipulations) chromosomes to which feasible schedules correspond (such as our crossover operators).
2. By defining a different encoding where all chromosomes produce feasible schedules (this method will be presented in the second approach).

We shall illustrate the crossover with the following example.

**Example 2.**

**Step 1.** Suppose that Parent 1, Parent 2 and machine $M_4$ are randomly selected.

Parent 1

| | | | |
|---|---|---|---|
| $M_1$ | $(1,2,0)$ | $(2,2,1)$ | |
| $M_2$ | $(2,3,2)$ | | |
| $M_3$ | $(1,1,0)$ | $(2,1,3)$ | $(3,2,5)$ |
| $M_4$ | $(3,1,5)$ | | |
| $M_5$ | $(1,3,0)$ | | |

Parent 2

| | | | |
|---|---|---|---|
| $M_1$ | $(1,1,0)$ | | |
| $M_2$ | $(3,2,6)$ | | |
| $M_3$ | $(2,3,2)$ | | |
| $M_4$ | $(1,2,0)$ | $(2,1,3)$ | $(3,1,9)$ |
| $M_5$ | $(1,3,0)$ | $(2,2,3)$ | |

**Step 2.** Copy the operations assigned to $M_4$ of Parent 1 (respectively Parent 2) in Child 1 (resp. Child 2) on the same machine $(M_4)$.

Child 1 in construction

| | | | |
|---|---|---|---|
| $M_1$ | | | |
| $M_2$ | | | |
| $M_3$ | | | |
| $M_4$ | $(3,1,?)$ | | |
| $M_5$ | | | |

Child 2 in construction

| | | | |
|---|---|---|---|
| $M_1$ | | | |
| $M_2$ | | | |
| $M_3$ | | | |
| $M_4$ | $(1,2,?)$ | $(2,1,?)$ | $(3,1,?)$ |
| $M_5$ | | | |

**Step 3.** Copy the non-existing operations of $M_1$, $M_2$, $M_3$ and $M_5$ of Parent 2 (resp. Parent 1) into Child 1 (resp. Child 2).

Child 1 in construction

| | | | |
|---|---|---|---|
| $M_1$ | $(1,1,?)$ | | |
| $M_2$ | $(3,2,?)$ | | |
| $M_3$ | $(2,3,?)$ | | |
| $M_4$ | $(3,1,?)$ | | |
| $M_5$ | $(1,3,?)$ | $(2,2,?)$ | |

Child 2 in construction

| | | | |
|---|---|---|---|
| $M_1$ | $(2,2,?)$ | | |
| $M_2$ | $(2,3,?)$ | | |
| $M_3$ | $(1,1,?)$ | $(3,2,?)$ | |
| $M_4$ | $(1,2,?)$ | $(2,1,?)$ | $(3,1,?)$ |
| $M_5$ | $(1,3,?)$ | | |

**Step 4.** Is any operation missing?

The answer is 'no' for Child 2. Then we have a new chromosome. But for Child 1, two operations are missing: Operation 1 of Job 2 and Operation 2 of Job 1. We put these operations following Step 4 of the crossover algorithm and we calculate the starting time of each operation respecting the precedence and resource constraints according to the formula indicated in Section 3.4. Finally, we obtain:

Child 1

| | | | |
|---|---|---|---|
| $M_1$ | $(1,1,0)$ | | |
| $M_2$ | $(3,2,6)$ | | |
| $M_3$ | $(2,3,2)$ | | |
| $M_4$ | $(1,2,0)$ | $(2,1,3)$ | $(3,1,9)$ |
| $M_5$ | $(1,3,0)$ | $(2,2,3)$ | |

Child 2

| | | | |
|---|---|---|---|
| $M_1$ | $(2,2,3)$ | | |
| $M_2$ | $(2,3,2)$ | | |
| $M_3$ | $(1,1,0)$ | $(3,2,5)$ | |
| $M_4$ | $(1,2,0)$ | $(2,1,3)$ | $(3,1,9)$ |
| $M_5$ | $(1,3,0)$ | | |

♦

*Mutation operators*

Mutation is the other of the two main transformation operators in an evolutionary algorithm. It is the principal source of variability in evolution and it provides and maintains diversity in a population. Normally, after crossover, each child produced by the crossover undergoes mutation with a low probability. We consider here two operators of mutation: the assigned mutation and the swap mutation (Mesghouni, 1999).

*A. Assigned mutation:* In this case, we use the flexibility of our problem, as 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.** Re-assign this selected operation to another machine in the same position if possible, respecting the precedence and resource constraints.

**Example 3.**

**Step 1.** Suppose that the following chromosome and Operation 2 of Job 1 are randomly selected (this operation is assigned to machine $M_3$ in the second position):

| | | | |
|---|---|---|---|
| $M_1$ | $(1,2,0)$ | $(2,2,1)$ | |
| $M_2$ | $(2,3,2)$ | | |
| $M_3$ | $(1,1,0)$ | $\mathbf{(2,1,3)}$ | $(3,2,5)$ |
| $M_4$ | $(3,1,5)$ | | |
| $M_5$ | $(1,3,0)$ | | |

**Step 2.** We re-assign $O_{3,1}$ to machine $M_5$, and obtain the following chromosome:

| | | | |
|---|---|---|---|
| $M_1$ | $(1,2,0)$ | $(2,2,1)$ | |
| $M_2$ | $(2,3,2)$ | | |
| $M_3$ | $(1,1,0)$ | $(3,2,3)$ | |
| $M_4$ | $(3,1,7)$ | | |
| $M_5$ | $(1,3,0)$ | $\mathbf{(2,1,3)}$ | |

♦

*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 = false ) then
make a left swap.
Else If (direction = true ) then
make a right swap.
End if

**Example 4.**

**Step 1.** Assume that the following chromosome is randomly selected and machines $M_1$ and $M_3$, and the second position are randomly selected.

| Position | 1 | 2 | 3 |
|---|---|---|---|
| $M_1$ | $\mathbf{(1,2,0)}$ | $(2,2,1)$ | |
| $M_2$ | $(2,3,2)$ | | |
| $M_3$ | $\mathbf{(1,1,0)}$ | $(2,1,3)$ | $(3,2,5)$ |
| $M_4$ | $(3,1,5)$ | | |
| $M_5$ | $(1,3,0)$ | | |

**Step 2.** The first case: direction = false $\rightarrow$ Make a left swap, we obtain the following chromosome:

| | | | |
|---|---|---|---|
| $M_1$ | $\mathbf{(1,1,0)}$ | $(2,2,1)$ | |
| $M_2$ | $(2,3,2)$ | | |
| $M_3$ | $\mathbf{(1,2,0)}$ | $(2,1,5)$ | $(3,2,7)$ |
| $M_4$ | $(3,1,7)$ | | |
| $M_5$ | $(1,3,0)$ | | |

The second case: direction = true $\rightarrow$ Make a right swap, we obtain the following chromosome:

| | | | |
|---|---|---|---|
| $M_1$ | $(1,2,0)$ | $(2,2,1)$ | $\mathbf{(3,2,3)}$ |
| $M_2$ | $(2,3,2)$ | | |
| $M_3$ | $(1,1,0)$ | $(2,1,3)$ | |
| $M_4$ | $(3,1,5)$ | | |
| $M_5$ | $(1,3,0)$ | | |

♦

### 3.2.2. Second Approach

*Encoding problem*

The second approach is based on the second chromosome representation. It is a direct encoding which permits to solve some of the problems met in the first encoding such as illegal solutions (schedule) after a crossover operation and the creation of the first population. Indeed, this encoding integrates the precedence constraints. Consequently, we can create randomly the first population, and the genetic operators are very simple and produce a feasible schedule. The second encoding is called the Parallel Jobs Encoding (PJE). It also enables us to treat together assignment and scheduling problems. The parallel job encoding is given as follows: The chromosome is represented by a matrix where each row is an ordered series of the operating sequences of this job, each element of this row containing two terms. The first is the machine which performs the operation considered, the second is the starting time of this operation if its assignment to this machine is definitive. This starting time is calculated taking into account the resource constraints. The general configuration of this encoding is shown in Fig. 2. Each row of this matrix (chromosome) is presented as follows:

| $\mathrm{Job}_j$ | Operation 1 | Operation 2 | Operation $x_j$ |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_1, t_{M_1})$ | $(M_2, t_{M_2})$ | $\cdots$ |
| $\mathrm{Job}_2$ | $(M_3, t_{M_3})$ | $(M_1, t_{M_1})$ | |
| $\cdots$ | $\cdots$ | | |
| $\mathrm{Job}_n$ | $(M_2, t_{M_2})$ | $(M_5, t_{M_5})$ | $\cdots$ |

Fig. 2. Parallel job encoding.

$$\mathrm{Job}_j : (M_a, t_{M_a}), (M_b, t_{M_b}), \ldots,$$

where each column (operation) of this job contains the machine which performs this operation and the starting time of this operation performed on this machine. For $\mathrm{Job}_j$, the first operation is performed on machine $M_a$ at time $t_{M_a}$, and Operation 2 is performed on machine $M_b$ at time $t_{M_b}$.

For the example presented in Section 3.2.1, one possible chromosome has the following parallel job encoding:

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_3, 0)$ | $(M_3, 3)$ | $(M_4, 5)$ |
| $\mathrm{Job}_2$ | $(M_1, 0)$ | $(M_1, 1)$ | $(M_3, 5)$ |
| $\mathrm{Job}_3$ | $(M_5, 0)$ | $(M_2, 2)$ | |

*Crossover operators*

In this section we present two new crossover operators adapted to our parallel job encoding. These operators always generate a new legal offspring.

*A. Row crossover:* the algorithm of this crossover is presented as follows:

**Step 1.** Choose randomly two parents (chromosomes) and one job (a row of the matrix). We suppose that Parents 1 and 2 and $\mathrm{Job}\, J$ are randomly selected.

**Step 2.** The operations of $\mathrm{Job}\, J$ in Child 1 received the same machines as assigned to the same $\mathrm{Job}\, J$ of Parent 1.

**Step 3.** Browse all of the jobs (the row) $R \leftarrow 1$
While $((R < N)$ and $(R \neq J))$ do
Copy the remainder of the machines
assigned to the operations of $\mathrm{Job}\, R$
of Parent 2 in the same job $(R)$ of Child 1
$R \leftarrow R + 1$
End

To obtain Child 2, go to Step 2 and interchange the roles of Parent 1 and Parent 2.

**Example 5.**

**Step 1.** Assume that two chromosomes, Parent 1 and Parent 2, and $\mathrm{Job}_2$ (the second row in the chromosome) are randomly selected.

Parent 1

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_3, 0)$ | $(M_3, 3)$ | $(M_4, 5)$ |
| $\mathrm{Job}_2$ | $(M_1, 0)$ | $(M_1, 1)$ | $(M_3, 5)$ |
| $\mathrm{Job}_3$ | $(M_5, 0)$ | $(M_2, 2)$ | |

Parent 2

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_1, 0)$ | $(M_4, 3)$ | $(M_4, 9)$ |
| $\mathrm{Job}_2$ | $(M_4, 0)$ | $(M_5, 3)$ | $(M_2, 6)$ |
| $\mathrm{Job}_3$ | $(M_5, 0)$ | $(M_3, 2)$ | |

**Step 2.** The operation of $\mathrm{Job}_2$ in Child 1 (resp. Child 2) received the same machines as those assigned to $\mathrm{Job}_2$ of Parent 1 (resp. Parent 2).

Child 1 in construction

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | | | |
| $\mathrm{Job}_2$ | $(M_1, ?)$ | $(M_1, ?)$ | $(M_3, ?)$ |
| $\mathrm{Job}_3$ | | | |

Child 2 in construction

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | | | |
| $\mathrm{Job}_2$ | $(M_4, ?)$ | $(M_5, ?)$ | $(M_2, ?)$ |
| $\mathrm{Job}_3$ | | | |

**Step 3.** Copy the remainder of the machines assigned to the operation of the other jobs (in this example it was about $\mathrm{Job}_1$ and $\mathrm{Job}_3$) of Parent 1 (resp. Parent 2) in the same jobs of Child 2 (resp. Child 1).

Child 1 in construction

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_1,?)$ | $(M_4,?)$ | $(M_4,?)$ |
| $\mathrm{Job}_2$ | $(M_1,?)$ | $(M_1,?)$ | $(M_3,?)$ |
| $\mathrm{Job}_3$ | $(M_5,?)$ | $(M_3,?)$ | |

Child 2 in construction

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_3,?)$ | $(M_3,?)$ | $(M_4,?)$ |
| $\mathrm{Job}_2$ | $(M_4,?)$ | $(M_5,?)$ | $(M_2,?)$ |
| $\mathrm{Job}_3$ | $(M_5,?)$ | $(M_2,?)$ | |

We can remark that the two offspring (Child 1 and Child 2) are legal, all of the precedence constraints are respected, and the starting time of each operation can be calculated while satisfying resource constraints. The following solutions, representing feasible schedules, are obtained:

Child 1

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_1,0)$ | $(M_4,1)$ | $(M_4,7)$ |
| $\mathrm{Job}_2$ | $(M_1,1)$ | $(M_1,2)$ | $(M_3,4)$ |
| $\mathrm{Job}_3$ | $(M_5,0)$ | $(M_3,2)$ | |

Child 2

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_3,0)$ | $(M_3,3)$ | $(M_4,5)$ |
| $\mathrm{Job}_2$ | $(M_4,0)$ | $(M_5,3)$ | $(M_2,11)$ |
| $\mathrm{Job}_3$ | $(M_5,0)$ | $(M_2,2)$ | |

♦

*B. Column crossover:* we illustrate this operator by the following algorithm.

**Step 1.** Choose randomly two parents (chromosomes) and one operation (a column of the matrix). We suppose that Parent 1, Parent 2 and Operation $i$ are randomly selected.

**Step 2.** Operation $i$ of all jobs in Child 1 received the same machines as those assigned to Operation $i$ of all jobs of Parent 1.

**Step 3.** Browse all the other operations (columns) $C \leftarrow 1$
While $((C < I)$ and $(C \neq i))$ do ($I$ is the total number of operations in the shop)
Copy the remainder of the machines assigned to the other operations of all the jobs Parent 2 in the same operations $(i)$ of Child 1.
$C \leftarrow C + 1$
End while.

To obtain Child 2, go to Step 2 and inverse the roles of Parent 1 and Parent 2.

The following example explains the use of this operator.

**Example 6.**

**Step 1.** It is assumed that the parents of the previous example and Operation 2 are randomly selected.

**Step 2.** Operation 2 of all the jobs ($\mathrm{Job}_1$, $\mathrm{Job}_2$ and $\mathrm{Job}_3$) in Child 1 (resp. Child 2) received the same machines assigned to Operation 2 of all the jobs of Parent 1 (resp. Parent 2) indicated in boldface in Fig. 2.

**Step 3.** Copy the remainder of the machines assigned to the other operations (Operations 1 and 3 of $\mathrm{Job}_1$, $\mathrm{Job}_2$ and Operation 1 of $\mathrm{Job}_3$) of Parent 2 (resp. Parent 1) in the same operations of Child 1 (resp. Child 2) indicated in italic in Fig. 2.

Child 1

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_1,0)$ | $\boldsymbol{(M_3,1)}$ | $(M_4,3)$ |
| $\mathrm{Job}_2$ | $(M_4,0)$ | $\boldsymbol{(M_1,3)}$ | $(M_2,11)$ |
| $\mathrm{Job}_3$ | $(M_5,0)$ | $\boldsymbol{(M_2,2)}$ | |

Child 2

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_1,0)$ | $\boldsymbol{(M_4,1)}$ | $(M_4,7)$ |
| $\mathrm{Job}_2$ | $(M_1,0)$ | $\boldsymbol{(M_5,2)}$ | $(M_3,5)$ |
| $\mathrm{Job}_3$ | $(M_5,0)$ | $\boldsymbol{(M_3,2)}$ | |

Fig. 3. Two new correct children after a column crossover.

♦

*Operator of mutation*

In this part, we present a new mutation operator, called the controlled mutation operator, designed especially for our parallel jobs encoding, as it can balance the machine loads. The algorithm of this operator is presented as follows:

**Step 1.** Choose randomly one chromosome and one operation from the set of operations assigned to a machine with a high load.

**Step 2.** Assign this operation to another machine with a small load, if possible.

**Example 7.**

**Step 1.** It is assumed that the chromosome of Fig. 4 is randomly selected. The working times of the machines are calculated, representing their hourly loads expressed in time units, cf. Fig. 5.

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_1, 0)$ | $\boldsymbol{(M_4, 3)}$ | $(M_4, 9)$ |
| $\mathrm{Job}_2$ | $(M_4, 0)$ | $(M_5, 3)$ | $(M_2, 6)$ |
| $\mathrm{Job}_3$ | $(M_5, 0)$ | $(M_3, 2)$ | |

Fig. 4. Selected chromosome for the mutation.

| Machine | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
|---|---|---|---|---|---|
| Hourly load | 1 | 5 | 2 | 13 | 5 |

Fig. 5. Hourly load of the machine before mutation.

It can be observed that machine $M_4$ has a high load and then that machine $M_1$ has a small load. Operations 2 and 3 of $\mathrm{Job}_1$ and Operation 1 of $\mathrm{Job}_2$ are performed by machine $M_4$. One operation is randomly chosen from this set. In this case Operation 2 of $\mathrm{Job}_1$ is assumed to be randomly selected (in boldface in Fig. 4).

**Step 2.** Re-assign the operation previously selected to machine $M_1$.

After calculating the new starting time of all operations satisfying the resource constraints, we obtain the chromosome of Fig. 6.

| | | | |
|---|---|---|---|
| $\mathrm{Job}_1$ | $(M_1, 0)$ | $\boldsymbol{(M_1, 1)}$ | $(M_4, 4)$ |
| $\mathrm{Job}_2$ | $(M_4, 0)$ | $(M_5, 3)$ | $(M_2, 6)$ |
| $\mathrm{Job}_3$ | $(M_5, 0)$ | $(M_3, 2)$ | |

Fig. 6. Our chromosome after mutation.

The hourly load for this new solution is calculated, giving the results shown in Fig. 7.

| Machine | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
|---|---|---|---|---|---|
| Hourly load | 4 | 5 | 2 | 7 | 5 |

Fig. 7. Hourly load of the machine after mutation.

♦

### 3.3. Fitness Function

Fitness is a measure of how well an algorithm has learnt to predict the outputs from the inputs. The goal of having a fitness evaluation is to give feedback to the learning algorithm regarding which individuals should have a higher probability of being allowed to multiply and reproduce and which of them should have a higher probability of being removed from the population.

Evaluation functions play the same role in EAs as the environment in natural evolution. It must indicate the aspects of schedules which make them seem right or wrong to the users. In this article, the objective is the minimization of a makespan. The fitness function can be expressed in two different manners:

1) corresponding to the time moment at which the belated machine finish executing its last operation, and
2) corresponding to the time moment at which the belated job ends.

The makespan $(C_{\max})$ is calculated, according to the following flow chart:

**Step 1.** Begin $l \leftarrow 1$

**Step 2.** If (first approach (PME) is used) then
While $(l < M)$ do
Calculate $C_{\mathrm{maxl}} \leftarrow C_{i,j,k}$ (time of completion of the last scheduling operation on $M_l$)
$l \leftarrow l + 1$
End While
$C_{\mathrm{maxch}} = \mathrm{Max}(C_{\mathrm{max1}}, C_{\mathrm{max2}}, \ldots, C_{\mathrm{maxM}})$
End If

If (second approach (PJE) is used) then
While $(l < N)$ do
Calculate $C_{\mathrm{maxl}} \leftarrow C_{xj,j,k}$ (time of completion of the last scheduling operation of job $J$)
$l \leftarrow l + 1$
End While
$C_{\mathrm{maxch}} = \mathrm{Max}(C_{\mathrm{max1}}, C_{\mathrm{max2}}, \ldots, C_{\mathrm{maxN}})$
End If

For each chromosome the fitness function aims to find the minimum Cmax, and is represented as follows:

$$\mathrm{Fitness} = \mathrm{Min}\,(C_{\mathrm{maxch1}}, C_{\mathrm{maxch2}}, \ldots, C_{\mathrm{maxchpopsize}}),$$

where the subscripts represent the chromosomes.

### 3.4. Computation of the Starting Time

For efficient use of evolutionary algorithms in such a combinatorial problem, we should choose efficiently a representation of the solution, the encoding should be simple, robust, and give the user the necessary information. Thus when designing our two-chromosome representation, to know the starting time of each operation on each chromosome is a piece of information. In order to calculate the starting time of each operation, we must define the following vectors:

$T_F$: Contains the deadline of the last operation scheduled on $\mathrm{Job}_j$ (size $(T_F) = N$).

$D_{M_k}$: Contains the deadline of the last operation scheduled on machine $M_k$ (size $(D_{M_k}) = M$).

Table 2. Operating sequences of the jobs and their processing times on all machines.

| | Ops | Order | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ | $M_6$ | $M_7$ | $M_8$ | $M_9$ | $M_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $O_{1,1}$ | | 1 | 4 | 6 | 9 | 3 | 5 | 2 | 8 | 9 | 5 |
| Job1 | $O_{2,1}$ | 1,3,2 | 3 | 2 | 5 | 1 | 5 | 6 | 9 | 5 | 10 | 3 |
| | $O_{3,1}$ | | 4 | 1 | 1 | 3 | 4 | 8 | 10 | 4 | 11 | 4 |
| | $O_{1,2}$ | | 4 | 8 | 7 | 1 | 9 | 6 | 1 | 10 | 7 | 1 |
| Job2 | $O_{2,2}$ | 2,1,3 | 2 | 10 | 4 | 5 | 9 | 8 | 4 | 15 | 8 | 4 |
| | $O_{3,2}$ | | 6 | 11 | 2 | 7 | 5 | 3 | 5 | 14 | 9 | 2 |
| | $O_{1,3}$ | | 8 | 5 | 8 | 9 | 4 | 3 | 5 | 3 | 8 | 1 |
| Job3 | $O_{2,3}$ | 1,2,3 | 9 | 3 | 6 | 1 | 2 | 6 | 4 | 1 | 7 | 2 |
| | $O_{3,3}$ | | 7 | 1 | 8 | 5 | 4 | 9 | 1 | 2 | 3 | 4 |
| | $O_{1,4}$ | | 5 | 10 | 6 | 4 | 9 | 5 | 1 | 7 | 1 | 6 |
| Job4 | $O_{2,4}$ | 1,2,3 | 4 | 2 | 3 | 8 | 7 | 4 | 6 | 9 | 8 | 4 |
| | $O_{3,4}$ | | 7 | 3 | 12 | 1 | 6 | 5 | 8 | 3 | 5 | 2 |
| | $O_{1,5}$ | | 6 | 1 | 4 | 1 | 10 | 4 | 3 | 11 | 13 | 9 |
| Job5 | $O_{2,5}$ | 2,3,1 | 7 | 10 | 4 | 5 | 6 | 3 | 5 | 15 | 2 | 6 |
| | $O_{3,5}$ | | 5 | 6 | 3 | 9 | 8 | 2 | 8 | 6 | 1 | 7 |
| | $O_{1,6}$ | | 8 | 9 | 10 | 8 | 4 | 2 | 7 | 8 | 3 | 10 |
| Job6 | $O_{2,6}$ | 1,2,3 | 7 | 3 | 12 | 5 | 4 | 3 | 6 | 9 | 2 | 15 |
| | $O_{3,6}$ | | 4 | 7 | 3 | 6 | 3 | 4 | 1 | 5 | 1 | 11 |
| | $O_{1,7}$ | | 5 | 4 | 2 | 1 | 2 | 1 | 8 | 14 | 5 | 7 |
| Job7 | $O_{2,7}$ | 3,2,1 | 3 | 8 | 1 | 2 | 3 | 6 | 11 | 2 | 13 | 3 |
| | $O_{3,7}$ | | 1 | 7 | 8 | 3 | 4 | 9 | 4 | 13 | 10 | 7 |
| | $O_{1,8}$ | | 8 | 3 | 10 | 7 | 5 | 13 | 4 | 6 | 8 | 4 |
| Job8 | $O_{2,8}$ | 3,1,2 | 6 | 2 | 13 | 5 | 4 | 3 | 5 | 7 | 9 | 5 |
| | $O_{3,8}$ | | 5 | 7 | 11 | 3 | 2 | 9 | 8 | 5 | 12 | 8 |
| | $O_{1,9}$ | | 3 | 9 | 1 | 3 | 8 | 1 | 6 | 7 | 5 | 4 |
| Job9 | $O_{2,9}$ | 1,2,3 | 4 | 6 | 2 | 5 | 7 | 3 | 1 | 9 | 6 | 7 |
| | $O_{3,9}$ | | 8 | 5 | 4 | 8 | 6 | 1 | 2 | 3 | 10 | 12 |
| | $O_{1,10}$ | | 9 | 2 | 4 | 2 | 3 | 5 | 2 | 4 | 10 | 23 |
| Job10 | $O_{2,10}$ | 3,2,1 | 3 | 1 | 8 | 1 | 9 | 4 | 1 | 4 | 17 | 15 |
| | $O_{3,10}$ | | 4 | 3 | 1 | 6 | 7 | 1 | 2 | 6 | 20 | 6 |

Begin
  $i \leftarrow 1$
  While $(i < I)$ do
    $J \leftarrow 1$
    While $(Job_j < N)$ do
      Calculate $(t_{i,j,M_k})$
      Update $(T_F[j], D_{M_k}[i])$
    End while
  End while
End.

Procedure Calculate $(t_{i,j,M_k})$

  Begin
    If $(T_F[j] < D_{M_k}[i])$ then
    $t_{i,j,M_k} \leftarrow D_{M_k}[i]$
    Else
    $t_{i,j,M_k} \leftarrow T_F[j]$
  End

Procedure Update $(T_F[j], D_{M_k}[i])$

  Begin
    $T_F[j] \leftarrow t_{i,j,M_k} + P_{i,j,M_k}$
    $D_{M_k}[i] \leftarrow t_{i,j,M_k} + P_{i,j,M_k}$
  End

## 4. Simulation Results

Computer simulations were performed to evaluate the effectiveness of the parallel machine and parallel job encoding. For this purpose, we analysed the following example: We consider 10 jobs and 10 machines. This problem presents a total flexibility (any machine can perform any operation). The makespan is known and equal to 16 units of time. Each job has 3 operations in its operating sequence. The processing times of these operations are presented in Table 2.

**A) Using the first approach (parallel machine encoding)**

The initial population has been created by taking the known solution as the first chromosome. We applied various mutations to extend it and to obtain 50 chromosomes. We used then genetic operators to improve this set of solutions towards an optimal one. Ten test runs were performed with the following rates of genetic operators:

Crossover rate = 75%,
Mutation rate = 5%,
Number of generations = 5000.

Table 3 shows the generation number giving the best makespan. The best result is presented in Fig. 8.

Table 3. Generation number giving the best makespan.

| Run number | Generation number | Makespan |
|---|---|---|
| 1 | 2265 | 7 |
| 2 | 2136 | 7 |
| 3 | 1965 | 7 |
| 4 | 1936 | 7 |
| 5 | 2035 | 7 |
| **6** | **1853** | **7** |
| 7 | 1896 | 7 |
| 8 | 1906 | 7 |
| 9 | 1885 | 7 |
| 10 | 1869 | 7 |

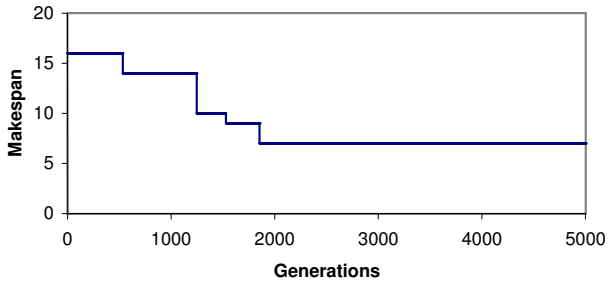



Fig. 8. Decrease in the schedule cost.

We can remark that an improvement in $C_{\max}$ was over 56% of time. The best makespan is equal to 7 time units and was obtained in the best case compared with our ten test runs after 1853 generations.

**B) Using the second approach (parallel job encoding)**

In this case, the initial population was created randomly and is shown in Fig. 9. We use ten simulations with our evolutionary algorithms to improve the solution towards an optimal one, with different crossover and mutation rates. Figure 10 represents the solutions obtained

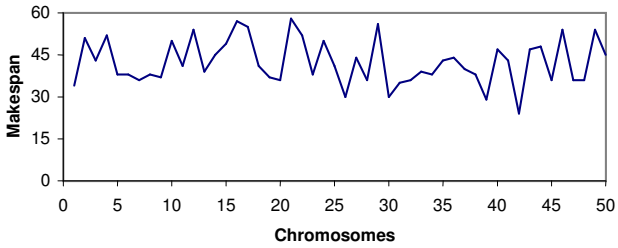



Fig. 9. Makespan of the first randomly generated population.

after ten executions with different genetic parameters. The crossover and mutation rates ($P_c$ and $P_m$) were varying and the population size was fixed to 50 chromosomes. The makespan obtained in all the cases is equal to 7 time units. We run it for 5000 generations.

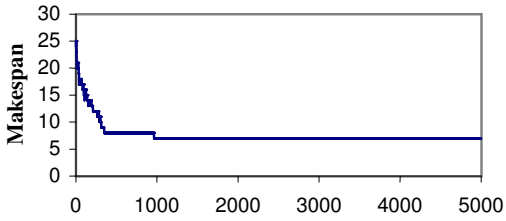
The best solution is obtained after 970 generations with $P_c$ equal to 0.75, $P_m$ equal to 0.1 and it presented in Fig. 10(a). We can remark that we obtained rapid convergence when the mutation rate varied between 0.1 and 0.2. The parallel machines encoding and the parallel jobs encoding give very good results. Therefore, the PJE takes lower time regarding the PME.

## 5. Conclusion

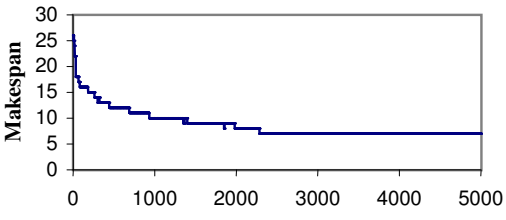
The application of evolutionary algorithms to a flexible job-shop scheduling problem with real-world constraints has been defined. We demonstrated that choosing a suitable representation of chromosomes (parallel encoding) is an important step to get better results.

We have developed genetic operators adapted for each representation (swap and assigned mutation for the PME, and row and column crossover and controlled mutation for the PJE), and an efficient method to create an initial population (a combination of some methods for the PME, and a random approach for the PJE because all of the constraints are integrated on the chromosome syntax).

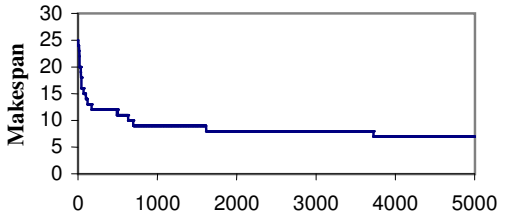
A proper selection of genetic parameters for an application of EAs is still an open issue. These parameters (crossover rate, mutation rate, population size, etc.) are usually selected heuristically. There are no guidelines as to the exact strategies to be adopted for different problems. In this work, we applied a fixed population size with different values of crossover and mutation rates. The controlled mutation reduces the blind aspect of genetic algorithms. Investigations are therefore necessary to determine these controlling parameters properly, in order to improve the performance of the proposed method. Simulation results show that the proposed parallel representations are suitable to the job-shop scheduling problem, confirming the effectiveness of the proposed approach.



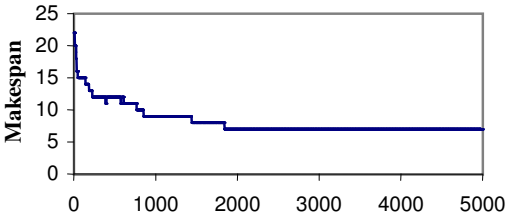

(a) $P_c = 0.75$ and $P_m = 0.1$.



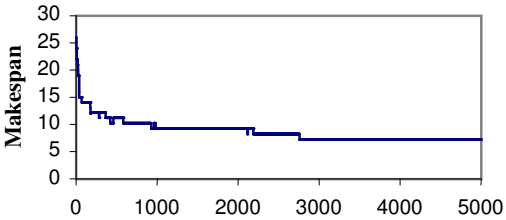

(b) $P_c = 0.75$ and $P_m = 0.2$.



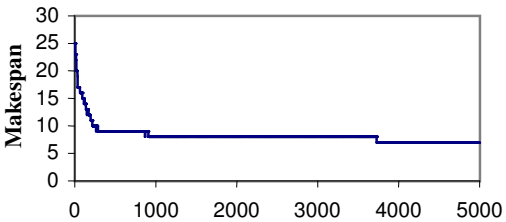

(c) $P_c = 0.75$ and $P_m = 0.3$.




(d) $P_c = 0.95$ and $P_m = 0.1$.




(e) $P_c = 0.95$ and $P_m = 0.2$.




(f) $P_c = 0.95$ and $P_m = 0.3$.

Fig. 10. Solutions obtained after ten exucutions.

## References


Baghi S., Uckun S., Miyab Y. and Kawamura K. (1991): *Exploring problem-specific recombination operators for job shop scheduling*. — Proc. 4-th Int. Conf. *Genetic Algorithms*, University of California, San Diego, pp. 10–17, July 13–16.

Banzhaf W., Nordin P., Keller R.E. and Francone F.D. (1998): *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Application*. — San Francisco: Morgan Kaufmann.

Bruns R. (1993): *Direct chromosome representation and advanced genetic operators for production scheduling*. — Proc. 5-th Int. Conf. *Genetic Algorithms*, University of Illinois at Urbana-Champaign, pp. 352–359.

Carlier J. and Chretienne P. (1988): *Problèmes d'ordonnancement: Modélisation / complexité / algorithmes*. — Paris: Masson.

Croce F., Tadei R. and Volta G. (1995): *A genetic algorithm for the job shop problem*. — Comp. Opers. Res., Vol. 22, No. 1, pp. 15–24.

Dasgupta D. and Michalewicz Z. (1997): *Evolutionary Algorithms in Engineering Applications*. — Berlin: Springer-Verlag.

Della Croce F., Tadei R. and Volta G. (1995): *A Genetic Algorithm for Job Shop Problem*. — Comput. Ops. Res., Vol. 22, No. 1, pp. 15–24.

Fonseca C.M. and Fleming P.J. (1998): *Multiobjective optimization and multiple constraint handling with evolutionary algorithms, Part I: Unified formulation*. — IEEE Trans/SMC, Part A: Syst. Hum., Vol. 28, No. 1, pp. 26–37.

Garey M.R. and Johnson D.S. (1979): *Computers and Intractability: A Guide to Theory of NP-Completeness*. — New York: W.H. Freeman and Co.

Goldberg D.E. (1989): *Genetic Algorithms in Search, Optimization, and Machine Learning*. — Massachusetts: Addison Wesley.

Golver F., Taillard E., De Werra D. (1993): *A user's guide to tabu search*. — Ann. Opers. Res., Vol. 41, No. 1, pp. 3–28.

Kirkpatrick S., Gelatt C.D. and Vecchi M.P. (1983): *Optimization by simulated annealing*. — Science, Vol. 220, No. 4598, pp. 671–680.

Mesghouni K., Hammadi S. and Borne P. (1998): *On modeling genetic algorithm for flexible job-shop scheduling problem*. — Stud. Inform. Contr. J., Vol. 7, No. 1, pp. 37–47.

Mesghouni K. (1999): *Application des algorithmes évolutionnistes dans les problemes d'optimisation en ordonnancement de la production*. — Ph.D. Thesis, Lille 1 University, France.

Mesghouni K., Pesin P., Trentesaux D., Hammadi S., Tahon C. and Borne P. (1999): *Hybrid approach to decision making for job-shop scheduling*. — Prod. Plann. Contr. J., Vol. 10, No. 7, pp. 690–706.

Michalewicz Z. (1992): *Genetic Algorithms + Data Structures = Evolution Programs*. — Berlin: Springer.

Portman C.M. (1996): *Genetic algorithms and scheduling: A state of the art and some proposition*. — Proc. Workshop *Production Planning and Control*, Mons, Belgium, pp. i-xxiv.

Quagliarella D., Périaux J., Poloni C. and Winter G. (1998): *Genetic Algorithms and Evolution Strategies in Engineering and Computer Sciences*. — England: John Whiley.

Syswerda G. (1990): *Schedule optimization using genetic algorithm*, In: Handbook of Genetic Algorithm. — pp. 323–349, New York: Van Nostrand Reinhold.

Uckun S., Baghi S. and Kawamura K. (1993): *Managing genetic search in job-shop scheduling*. — IEEE Expert, Vol. 8, No. 5, pp. 15–24.

Received: 12 February 2001
Revised: 23 July 2001
Re-revised: 2 July 2003