UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Ingeniería

https://revistas.udistrital.edu.co/index.php/reving/issue/view/1206

DOI: https://doi.org/10.14483/23448393.21162

Research

Transgenic Algorithm Applied to the Job Shop Rescheduling Problem

Algoritmo transgénico aplicado al Job Shop Rescheduling Problem

Néstor Andrés Beltrán-Bernal[1]*, José Ignacio Rodríguez-Molano[2], Diego Ernesto Mendoza-Patiño[3]

[1]Universidad de la Salle (Bogotá, Colombia)

[2]Universidad Distrital Francisco José de Caldas, (Bogotá, Colombia)

[3]Universidad Antonio Nariño (Bogotá, Colombia)

Abstract

Context: Job sequencing has been approached from a static perspective, without considering the occurrence of unexpected events that might require modifying the schedule, thereby affecting its performance measures.
Method: This paper presents the development and application of a genetic algorithm to the Job Shop Rescheduling Problem (JSRP), a reprogramming of the traditional Job Shop Scheduling Problem. This novel approach seeks to repair the schedule in such a way that theoretical models accurately represent real manufacturing environments.
Results: The experiments designed to validate the algorithm aim to apply five classes of disruptions that could impact the schedule, evaluating two performance measures. This experiment was concurrently conducted with a genetic algorithm from the literature in order to facilitate the comparison of results. It was observed that the proposed approach outperforms the genetic algorithm 65 % of the time, and it provides better stability measures 98 % of the time.
Conclusions: The proposed algorithm showed favorable outcomes when tested with well-known benchmark instances of the Job Shop Scheduling Problem, and the possibility of enhancing the tool's performance through simulation studies remains open.

Keywords: disruptions, efficiency, stability, job shop, rescheduling, transgenic algorithm

Article history

Received: 14th/Aug/2023

Modified: 27st/Oct/2023

Accepted: 4th/Nov/2023

Ing, vol. 29, no. 1, 2024. e21162

*Correspondence: nbeltran@unisalle.edu.co

Resumen

Contexto: La secuenciación de trabajos ha sido abordada desde un enfoque estático, sin considerar la aparición de eventos inesperados que requieran modificar el cronograma, lo que incide en sus medidas de desempeño.
Método: Este artículo expone el desarrollo y aplicación de un algoritmo transgénico al *Job Shop Rescheduling Problem* (JSRP), una reprogramación del tradicional *Job Shop Scheduling Problem*. Este enfoque novedoso busca reparar el cronograma de modo que los modelos teóricos representen los entornos de manufactura reales.
Resultados: Los experimentos diseñados para validar el algoritmo pretenden aplicar cinco clases de interrupciones que pueden afectar el cronograma, evaluando dos medidas de desempeño. Este experimento se realizó simultáneamente en un algoritmo genético de la literatura para facilitar la comparación de los resultados. Se observó que el enfoque propuesto tiene un desempeño superior al del algoritmo genético el 65 % de las veces y lo supera en la medida de estabilidad el 98 % de las veces.
Conclusiones: El algoritmo propuesto mostró buenos resultados al ser probado con instancias de comparación reconocidas del *Job Shop Scheduling Problem* (JSSP), y queda abierta la posibilidad de mejorar el desempeño de la herramienta por medio de estudios de simulación.

Palabras clave: interrupciones, eficiencia, estabilidad, *job shop*, *rescheduling*, algoritmo transgénico

Table of contents

1. Introduction

Job sequencing is one of the main challenges faced by the people in charge of planning and scheduling production. This is due mostly to the combinatory nature of the task and the fact that these types of problems can be classified as NP-Hard ones (1), which requires the development of advanced algorithms that may offer nearly optimal solutions, thus allowing for an efficient use of computational resources.

On another note, static models (such as bifurcation and dimensioning) have approached job scheduling without contemplating the appearance of unexpected events in the execution of production processes. This has created a gap between theoretical models and real manufacturing environments (2).

Hence, this work studies the Job Shop Scheduling Problem (JSSP) from a rescheduling-oriented perspective, seeking a better representation of the actual conditions of manufacturing environments. Rescheduling is a process that can react to unexpected events as a part of the dynamics of said environments (3). In this sense, the Job Shop Rescheduling Problem (JSRP) (4–7) is conceived as a problem of higher complexity.

Studies focused on the rescheduling of manufacturing systems are wide and diverse. Therefore, this work is framed within the search for methods to fix production timetables that have been affected by an unexpected event (3). The method developed in the subsequent sections is based on the architecture of a transgenic algorithm (8), which is characterized by metaheuristics applied to diverse problems of high complexity, leading to competitive results compared to more renowned techniques (9–14).

The main objective was to develop a computational algorithm based on the metaphor of transgenic computation (53) which allows production schedulers to generate high-quality timetables in terms of efficiency and stability while also being able to react to unexpected events or interruptions arising during the execution of said timetables, thus mitigating the impact on the performance of the manufacturing system.

2. Proposal

2.1. Job Shop Scheduling Problem

Sequencing problems can be as diverse as manufacturing environments. The JSSP is one of the most studied topics in the research field of operations and computer science. This is mostly due to its relationship with production planning activities within the industry. The JSSP is recognized as an NP-Hard problem (1) and is considered to be among the highly complex and unmanageable optimization problems (15, 55).

JSSP is defined as a manufacturing environment containing a set $J = \{1, \ldots, j, \ldots, n\}$ of n jobs that must be scheduled in a set $M = \{1, \ldots, i, \ldots, m\}$ of m machines. Each job has a technological sequence of machines (stages) for it to be processed. The use of machine i for the processing of job j is denoted as *operation* O_{ij} with a duration equal to t_{ij}, which is known as the *processing time*. Furthermore, the problem revolves around constraints and suppositions such as the following:

- Each machine can only process one job at a time.
- Each job can only be processed by one machine at a time.
- Jobs must only be processed once in each machine.
- The technological sequence of each job has been completely defined.

- The processing times of all operations are known.
- The machines are always available and never interrupted.

The problem consists of determining the sequence of operations to be defined for the machines in order to minimize the makespan (C_{max}), whose value is the time necessary to complete all the operations. The JSSP with makespan as the target can be mathematically described via Eqs. (1), (2), (3), (4), and (5):

$$\text{Min } C_{\text{max}} \tag{1}$$

$$r_{kj} - r_{ij} \geq t_{ij} \quad \forall\, O_{ij} \rightarrow O_{kj} \ \in A \tag{2}$$

$$C_{\text{max}} - r_{ij} \geq t_{ij} \quad \forall\, O_{ij} \ \in N \tag{3}$$

$$r_{ij} - r_{il} \geq t_{il} \quad \forall r_{il} - r_{ij} \geq t_{ij} \quad \forall\, O_{il} \wedge O_{ij}, i = 1, \ldots, m \tag{4}$$

$$r_{ij} \geq 0 \quad \forall\, O_{ij} \in N \tag{5}$$

where r_{ij} is the time at which operation O_{ij} starts, N is the set of all operations O_{ij}, and A is the set of all constraints given by the technological sequence of each job, such as $O_{ij} \rightarrow O_{kj}$, which requires job j to be processed by machine i before being processed by machine k. This formulation of the JSSP is called *disjoint scheduling* given the third set of constraints (16). Furthermore, the JSSP can be defined as an integer scheduling problem (17).

An instance of three jobs for three machines (3 × 3) of JSSP is represented as: As evidenced in this

Table I. Example of a 3 × 3 instance

O_{ij}	j		
i	1	2	3
1	(2)	(1)	(1)
2	(3)	(2)	(3)
3	(1)	(3)	(2)

t_{ij}	j		
i	1	2	3
1	3	6	4
2	2	8	6
3	7	3	3

table, matrix O_{ij} describes the order in which the jobs for each machine are processed. For instance, job 1 ($j = 1$) has the following technological sequence: (1) $O_{31} \rightarrow$ (2) $O_{11} \rightarrow$ (3) O_{21}. Matrix t_{ij} contains the processing times of each operation O_{ij}.

There are different representations of the solution for this problem. The Gantt diagram is one of the most commonly used forms in this regard. Fig. **1** shows an example of the representation obtained for a solution of the instance in Table **I**.

Figure 1. Gantt diagram

The Gantt diagram depicts what is known as the *production timetable*. It shows a feasible solution to the JSSP when all constraints are satisfied. It is worth noting that this definition of the JSSP corresponds to a theoretical problem that constitutes one approximation to a more complex scenario.

On another note, researchers studying the JSSP have opted to define a set of assumptions and constraints, leading to a gap between theoretical models and real manufacturing environments (2).

2.2. Job Shop Rescheduling Problem

The definition of the JSSP described in the previous section is a deterministic and static sequencing model, as its parameters and variables are known beforehand and do not change over time. Thus, this model only represents an approximation of a real manufacturing environment, since the industry experiences some scenarios encompassing variables that are dynamic or have a high degree of uncertainty.

In this sense, when a production timetable is built, both the supervisors and the operators attempt to execute it as closely as possible. The issue lies in unexpected events that alter the execution of the current production timetable, causing it to become inappropriate and unmanageable. Said events are called *interruptions* and need to be treated swiftly in order to minimize any effects that can deteriorate the system performance (55). This is where rescheduling becomes the proper tool to handle interruptions and complete all scheduled jobs.

Rescheduling is defined as the process of updating the existing production timetable in response to the interruptions or changes (3), whose integration with the Job Shop manufacturing environment results in what is referenced in the literature as the *Job Shop Rescheduling Problem* (JSRP) (4–7).

Studies on rescheduling from different perspectives are numerous. The most common approaches are the following: (i) methods to fix a timetable that has been interrupted, (ii) methods to create robust timetables to face interruptions, and (iii) studies on how rescheduling policies affect the performance of dynamic manufacturing systems.

Some of the articles include, for instance, the study of the Job Shop environment under the dynamic arrival of new jobs, variations in processing times, and machine failure (18). Other authors, such as (19), propose an algorithm capable of predicting and absorbing the effects of interruptions while

maintaining system performance. In (20), a genetic algorithm is applied to scheduling and rescheduling in a non-deterministic Job Shop environment.

Given that the studies on rescheduling are as numerous as they are diverse, this work adopts the rescheduling framework defined in (3). This framework allows for the understanding of rescheduling studies, in addition to classifying the technique discussed herein.

In order to properly narrow down the study of rescheduling in manufacturing environments, it is necessary to include an appropriate framework, given that its study can become widely diverse and complex depending on the scope defined by the researcher. According to (3), the rescheduling framework is composed of these elements: rescheduling environments, strategies, policies, and methods.

2.2.1. Rescheduling framework

• Rescheduling environments

Rescheduling environment refers to the set of jobs that must be scheduled. The environment can be static (finite set of jobs) or dynamic (infinite set of jobs). The environment can be deterministic (all the information is given) or stochastic (some information is uncertain). Dynamic environments may also be classified according to the variability in which the arrival takes place: without variability (cyclical production), with variability (flow shop), and with variability in the process flow (job shop).

• Rescheduling strategies

Rescheduling strategies refer to how production is controlled in the aforementioned environments, whether timetables are generated or not. Two common strategies have been identified in this regard: dynamic sequencing and predictive-active sequencing. The former does not generate production timetables, since the jobs are dispatched when necessary, using the information available at the moment of dispatch and with the help of dispatch rules or theoretical control models for manufacturing systems. The second strategy is characterized by two phases: a predictive phase, during which the initial production timetable is generated; and a reactive phase, in which the production timetable is updated as a response to interruptions in order to minimize their impact on system performance. The predictive-reactive strategy includes three policies: periodic, given event, and hybrid. These refer to the moment of rescheduling.

• Rescheduling methods

Rescheduling methods describe how to generate and update production timetables (54). Two approaches have been proposed: the generation and repair of timetables. The generation of production timetables can be coordinated through different techniques, in the form of analytical, heuristic, or more complex algorithms. The timetable repair approach (*i.e.*, update process) includes three main variants: right-shift rescheduling, partial rescheduling, and complete rescheduling.

• **Performance metrics**

In addition to the previously discussed elements, performance metrics are an important guideline in rescheduling studies. They can be classified into three groups: efficiency metrics, stability metrics, and timetable costs. The first group is made up of the most common time-based metrics, such as makespan, average delay, and average flow time. The second group refers to the variables that measure changes within a timetable in terms of launch times or the differences between sequences. Lastly, the cost variables include benefit according to the job, minimization of the total cost, and reduction of the work in process (WIP), among others.

Various techniques have been proposed in literature to solve the rescheduling problem in terms of the aforementioned performance metrics. Methods such as right-shift rescheduling (21), match-up (22), and Affected Operation Rescheduling (AOR) (23) are some of the most well-known group of heuristics and are characterized by a simple implementation process, which is however limited by the range of interruptions that they can face. Furthermore, these heuristics operate under the partial rescheduling strategy, which implies that they try to adhere to the preestablished timetable as much as possible after the interruption (24, 25).

On another note, complete rescheduling techniques seek to reschedule all remaining operations after the point of interruption (26, 27). Under the umbrella of this approach, diverse and sophisticated algorithms have been implemented, as is the case of genetic algorithms (4, 20, 53, 55), which are known to improve the efficiency of timetables while ignoring stability-related metrics.

In this sense, new techniques are constantly being developed to solve combinatorial problems. JSSP is no exception, since it belongs to this class even from the rescheduling (JSRP) viewpoint, implying that timetables require more quality over time with a rational use of computational resources. Hence, the next section exposes an algorithm that has offered many advantages in other problems of the same nature.

2.3. Transgenic algorithm

In the context of evolutionary computation, defined as the study of the foundations and applications of certain heuristic techniques based on natural evolution principles (28), a variety of algorithms have been developed to face highly complex problems such as the one discussed in the present work.

Since the appearance of genetic algorithms in the article titled *Adaptation in natural and artificial systems* (29), techniques based on this metaphor have not been scarce. Some of them have inherited the basic structure of genetic algorithms to be combined with other local search techniques, paving the way for genetic/evolutionary hybrid algorithms (16, 30), Lamarckian genetic algorithms (31), or general memetic algorithms (32, 33).

The term *memetic algorithm* stems from the English term *meme*, coined by R. Dawkins as an analogy to the gene in the context of cultural evolution. (34) defines *memes* as the units of information coded

and transmitted by non-genetic media. These definitions are relevant for the approach described in the subsequent section, which is based on the transgenic algorithm.

• Transgenic computation

Transgenic computation (TC) is a metaphor derived from the use of memetic information (memes) and extra- and intracellular flows to plan and execute genetic manipulations in the context of evolutionary algorithms (8).

To the scope of evolutionary computation, TC brings the use of exogenous and endogenous information to infer the formation and modification processes of the individuals in a given population, to use intracellular flow as an operational mechanism in order to carry out the required manipulations amongst individuals, to explore new populational improvements using transgenic agents and competition between agents and individuals, and to guide the evolutionary process, allowing for the occurrence of evolutionary jumps (9). Further information on the basics of TC can be found in (8,10,11).

This method has been branched into two types of algorithms: the extra-intracellular transgenic algorithm (EITA) (12) and the proto-gen (ProtoG) algorithm (9). The most noticeable difference between them lies in the fact that the ProtoG algorithm does not have reproduction operators (mutation or cross), so it is not based on an extracellular approach (9).

The proposed algorithm is based on the ProtoG algorithm, whose structure and composition are detailed below.

• The ProtoG algorithm

As discussed with TC, transgenic algorithms are metaheuristics that insert information in a planned manner for improving genetic information. Their pillar lies in the intracellular paradigm, through agents that manipulate the genetic context. The information inserted by said agents is created and controlled via the epigenetic paradigm. These paradigms define the ProtoG algorithm *per se*, thus excluding the extracellular paradigm that uses sexual reproduction as the basis of information exchange between individuals (10).

• Intracellular paradigm

In this algorithm, the chromosomes are exposed to a direct attack from the agents in the intracellular flow, which must compete against each other and the defense mechanisms of chromosomes in order to consolidate the transcription of their codes (11). In the computational context, said chromosomes are part of the population C of size q, where $C = \{1, \ldots, c, \ldots, q\}$ is a function f called the *fitness function*. Each chromosome c is a chain of integer length h that represents a solution to the problem, such as $f : c \to \mathbb{R}+$, $c = 1, \ldots, q$, where f returns a real value.

Chromosomic alterations are only carried out by transgenic agents, which makes them essential to the evolutionary process as the only source of intensification and diversification in the search process (9).

Transgenic agents are entities used by the algorithm to insert information into the genetic context. These agents manipulate chromosomes similarly to intracellular vectors used in genetic engineering. As an analogy to the terms used by microbiologists, there are four types of transgenic agents: plasmid, virus, recombined plasmid, and transposon (11).

In the computational context, a transgenic agent λ is defined as a couple $\lambda = (I, \Phi)$, where I is a chain of information (corresponding to the memes) and Φ describes the manipulation method used by the agent. $\Phi = (p_1, \ldots, p_s, \ldots, p_x)$ where $p_s, s = 1, \ldots, x$, are the procedures that determine the behavior of the agents. Table **II** depicts the procedures of the manipulation method (extracted and adapted from (11)).

In this sense, when the chain of information I of λ is a genetic code and its manipulation method uses procedures p_1, p_2, and p_3, λ is called a virus (11). In the case of the ProtoG algorithm, the mobile genetic particle (MGP) is a special case of virus where the virus lifetime (or blocking period) established by procedure p_3 is equal to zero, and the attack procedure p_1 only takes place when chromosome fitness is improved (9).

Table II. Procedures of the manipulation method

Procedure (p_s)	Description
Attack ($p1$)	Defines a criterion that establishes whether a chromosome c is susceptible to the information of a transgenic agent λ. The function $v(c) = 'true'$ if c is vulnerable to λ, $v(c) = 'false'$ otherwise.
Transcription ($p2$)	If $v(c) = 'true'$, the procedure defines the information transferred from λ to c.
Blocking/unblocking ($p3$)	It establishes a time period in which the transferred information cannot be altered in c.
Identification ($p4$)	It identifies the positions in c that will be used to limit the operation of λ.
Recombination ($p5$)	It identifies the origin and length of two or more chains of information in λ.

• Epigenetic paradigm

The difficulty of considering epigenetics in the context of evolutionary computation is discussed by (34). The initial work on this matter simply linked the memes to information obtained outside the extracellular flow [10]. In memetic algorithms, according to (35) and (36), the epigenetics stage is mainly expressed through the improvement of chromosomes using local search procedures. This is where TC adopts a new approach, defining *memes* as memories with information that is stored, coded, and transmitted using non-genetic means, as stated by (33). Hence, memes can be used to adjust a chromosome or block of genes, or said memes could be subjected to competition and selection processes relevant to the cultural media (10).

Memes are units of information that can be spread by a culture in the same way that genes are disseminated through a genetic reservoir (37). The relationship between genes and memes is based on the fact that genes prescribe the epigenetic rules, which are regularities of sensorial perception and mental development that encourage and channel the acquisition of culture. Culture contributes to determining which prescribed genes survive and multiply from one generation to the next one. New properly sequenced genes alter the epigenetic rules of the population. The altered epigenetic rules change the direction and effectiveness of cultural acquisition channels by offering feedback to the coevolution 'gene vs. meme' process (38).

In the computational context, these epigenetic rules are simulated by a transgenic rule-based structure. The evolution control of a population of chromosomes, transgenic agents, and the information of a memetic base is composed of three classes of rules (10,11).

Type 1 rules lead to the construction of information chain I, which is transported by the transgenic agents (11). These types of rules can have knowledge on any aspect of the problem, as well as previous knowledge stored in the memetic base (MB), which constitutes the storage bank of memes. Memes can be comprised of construction blocks or manipulation procedures (11). Type 2 rules define the information contained in I is transcribed into a chromosome, *i.e.*, the operator used by λ. These types of rules can evolve in terms of the resistance shown by chromosomes. Type 3 rules are present throughout the process, defining which agents are used, the number of chromosomes under attack during a given iteration, the number of agents created, and the stopping criterion, among others (11,12).

• Pseudocode

The ProtoG algorithm can be decomposed into two phases. The first one generates the construction blocks (memes) and codes them using transgenic agents. In the second phase, the agents compete to transcribe their information into the chromosomes and thus improve the solutions (9).

The implementation of the ProtoG algorithm improves the fitness of the chromosome population through the manipulation carried out by the transgenic agents. However, not only chromosomes evolve; memes also do. This process makes TC an informed and co-evolutionary search technique (9).

Transgenic algorithms, including ProtoG, have proven to be successful when compared to known techniques in the solution of high-complexity problems. Some of the problems tackled by this technique include the quadratic allocation problem (QAP) (9,11), the traveling purchaser problem (TPP) (12), the graph coloring problem (12), and sequencing in flow-shop with permutation (13), among others (39,40).

The pseudocode of the algorithm is the following (taken and adapted from (9,11)):

Begin
Load *a meme base with a set of solutions and rules to generate memes*
Generate *and assess an initial population*
Repeat
    Generate *an agent from competition between memes*
    For *each chromosome:*
    If *the chromosome is sensitive to manipulation*
    Begin
        Manipulate
        Assess
        If *the chromosome meets the immunity criterion*
            Then *include its memes in the Memetic Base*
End
Until *a stop criterion is met*
End

2.4. Proposed algorithm

As discussed in Section 3, the proper definition of the scope of rescheduling requires an appropriate framework. The approach of this work lies in the methods used to repair a timetable that has been interrupted. Hence, it is important to initially define these interruptions, also known as *rescheduling factors* (41,42), which arise at a given time of the initial timetable known as the *interruption point*. This study contemplates all the rescheduling factors identified in the work of (43), as shown in Table **III**.

Given that rescheduling factors are a significantly wide group, they are classified into five classes. This classification is based on the general actions that would be needed to repair the interrupted timetable and meet the constraints imposed by the arising rescheduling factor. A description of the considered factors can be seen in (44).

2.4.1. Manufacturing environment

The rescheduling framework defined by (3) encompasses the elements of the proposed algorithm framework shown in Table **IV**.

Table III. Classification of rescheduling factors

Id.	Name	Class	Actions
1	Failure		
2	Machine maintenance		
3	Absenteeism	I	Insert inactivity time
4	Tool failure		
5	Delay in material transportation		
6	Depletion of raw materials		
7	Variation in processing time		Insert inactivity time
8	Variation in machine performance	II	+
9	Tool deterioration		Update processing times
10	Variation in preparation times		
11	Arrival of a new job		Update technological sequences
12	Reprocessing	III	+
13	Rejection		Update processing times
14	Urgent job	IV	Schedule
15	Priority shift		priority-marked operations
16	Order cancellation	V	Eliminate scheduled operations
17	Outsourcing		

2.4.2. Transgenic algorithm

As stated in the field of evolutionary computing, one of the most important factors in the success of algorithms is the representation of the problem. In this sense, (16) conducted a study of the representation forms used in the application of genetic algorithms to solve the JSSP. In this case, operation-based representation was selected, which defines n jobs and m machines to create a chain (chromosome) of $n \times m$ positions (genes), where all operations of job j are distinguished by the same symbol (number) that appears exactly m times throughout the chain. Each appearance of the symbol that represents job j corresponds to an operation in the order of the technological sequence of said job. This representation method offers the advantage that all permutations in its elements provide a feasible solution to the JSSP, thus avoiding complex repair processes, as seen in other alternatives. An example of this representation is shown, using the instance data presented in the second section:

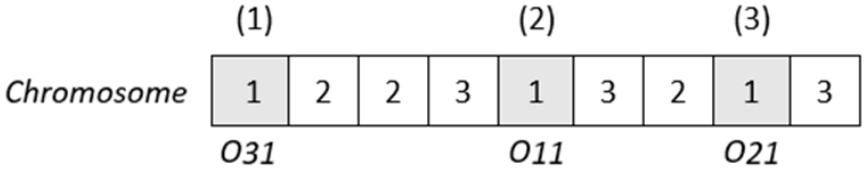

Figure 2. Operation-based representation

As seen in Fig. 2, each appearance of the number 1 ($j = 1$) refers to an operation in the technological sequence. This can be replicated for the remaining jobs. The representation can be useful during the

Table IV. Defined manufacturing environment

Rescheduling environment	
Dynamic	Job shop
There is a continuous arrival of jobs to the manufacturing environment floor (set of finite jobs).	Technological sequences of jobs differ from each other (variability in the process flow).
Rescheduling strategy	
Predictive-reactive	
There are two phases: the first one generates an initial schedule timetable, while the second one is in charge of updating (repairing) the timetable.	
Rescheduling policy	
Hybrid	
Rescheduling can be carried out periodically or when a special event takes place.	
Rescheduling method	
Repair	Complete
The interrupted timetable is updated in response to the arising rescheduling factor.	All remaining operations are rescheduled after the interruption point.
Performance metrics	
Efficiency	Stability
Makespan (E): percentage-form variation of the makespan obtained for the repaired timetable compared to the initial one.	Deviation (D): average difference between the starting time of the repaired timetable operations compared to the initial one.

elaboration of the Gantt diagram. The chromosome is read from left to right, where each position (gene) delivers the information of an operation O_{kj}. The operation is inserted into the diagram by setting the starting time r_{kj} as early as possible, *i.e.*, by checking the conclusion time of the preceding operation O_{ij} and not overlapping times with a previously inserted operation. Lastly, operation O_{kj} is defined from r_{kj} to $r_{kj} + t_{kj}$. An example of the aforementioned procedure can be seen in (16).

After the representation of the solution is defined, the next task involves the design of the rescheduling method based on the ProtoG algorithm architecture for the previously defined manufacturing environment, which is divided into the following two stages:

- **Stage 1: intracellular paradigm**

The transgenic agents of the evolutionary process are defined in this stage. It is worth mentioning

that these agents have been designed in terms of the JSSP features defined in the second section. The MGP is the agent chosen to carry out the genetic modifications. The composition and general structure of the designed agents is shown below.

Let $\lambda = (I_j, \Phi)$ be a transgenic agent composed of an information chain $I_j = (g_{1j}, \ldots, g_{ij}, \ldots, g_{mj})$ or meme with length m, where the element g_{ij} can represent one of the following elements: a position in the processing order, a preceding job, or an adjacent job, as suggested by the evolutionary process, which is taken as a reference to schedule job j in machine i, with $i = 1, \ldots, m$ and $j \int J$. Let $\Phi = (p_1, p_2, p_3)$, be the manipulation method of the agent composed of the procedures p with $s = \{1, 2, 3\}$. Table V defines each procedure.

Table V. Procedures of the manipulation method of the chosen agent

Procedure (p_s)	**Definition**
Attack $(p1)$	Let c be the chromosome before the modification of its structure and c' the manipulated chromosome. If $f(c') > f(c)$, then chromosome c is vulnerable, $v(c) =$ 'true'. Otherwise, $v(c) =$ 'false', where v is the function that defines the susceptibility of the chromosome.
Transcription $(p2)$	If $v(c) =$ 'true', then the operations of job j are scheduled according to the information chain I_j.
Block/unblock $(p3)$	Let y be the function that defines the number of iterations in which the information transcribed by p cannot be altered, where $v(c) = 0$ in all cases.

Three agents were designed for this algorithm. They all inherit the previously defined composition and structure, and the only difference between them is the type of information (meme) that they carry. The following section describes how the epigenetic paradigm creates, stores, and controls this information in the MB to guide the evolutionary process.

• Stage 2: epigenetic paradigm

As previously stated, there are three types of rules that represent this paradigm in the computational context. Recall that type 1 rules lead the construction of the information chain (meme) that is transported by the transgenic agent. This information is stored in the MB. To illustrate this process, Fig. 3 presents a diagram representing the flow of individuals from the population to the transformation of their information into memes. The process starts with the population of chromosomes. This population enters

a selection process, where chromosomes with the best fitness are considered to continue with the process (This procedure is controlled by a factor that will be defined later).

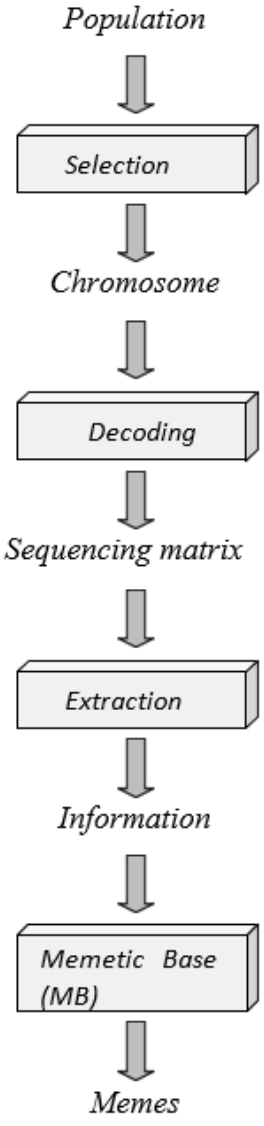

Figure 3. Meme creation process

Once the chromosome has been selected, it undergoes a new procedure called *decoding*. This refers to the conversion of a chromosome into a convenient representation form, which is used to extract the information needed by the MB to create memes. The result of this decoding stage is the sequencing matrix, defined as an array of m rows by n columns, which represents the sequence in which the jobs will be processed by each machine. An example of said procedure is depicted in Fig. 4. The sequencing matrix is derived from the examination of the Gantt diagram linked to the chromosome. In the form of a matrix, this array summarizes the information related to the order in which the jobs of each machine will be processed.

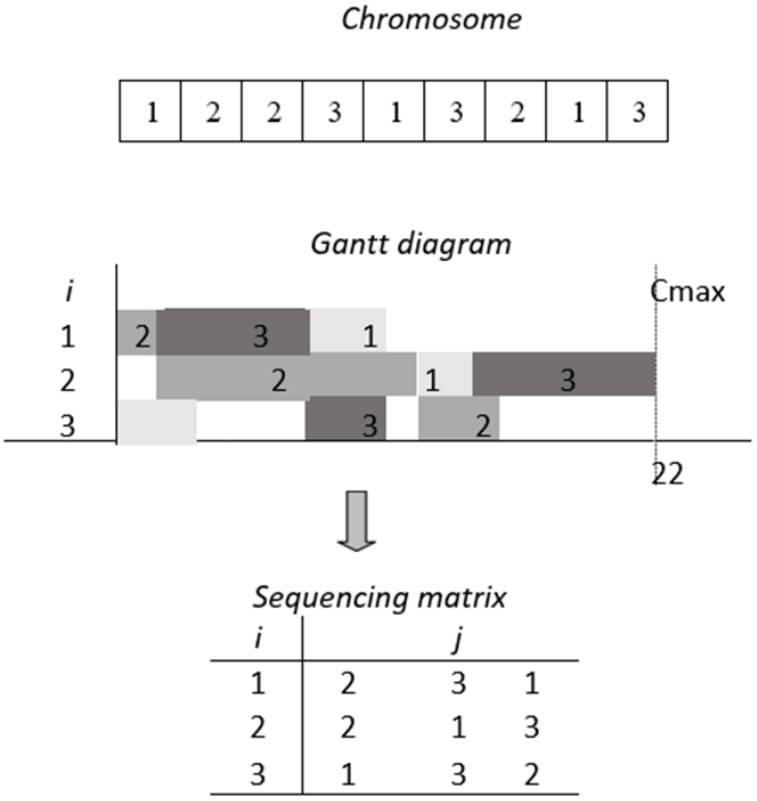

Figure 4. Decoding process of the chromosome into a sequencing matrix

Now that the decoding process has been detailed, the sequencing matrix becomes the main asset for the extraction procedure. It consists of obtaining key information from the structure of the given chromosome.

Once the information has been extracted, it is stored in the MB. The matrices that integrate said base are described below.

Let MB be a set of matrices $MB = \{Mp, Mr, Ma, Mb\}$ that store specific information from the population of chromosomes C through an evolutionary process.

Mp is defined as the *position matrix*, where the element Mp_{ijl} is the average makespan of the chromosomes that have scheduled the job j in the sequence of operations of machine i in position r, with $i = 1, \ldots, m, j = 1, \ldots, n, r = 1, \ldots, n$.

Mr is defined as the precedence matrix, where element M_{rijl} is the average makespan of the chromosomes that have scheduled the job j in machine i before job l with $i = 1, \ldots, m, j = 1, \ldots, n, l = 1, \ldots, n$.

Ma is defined as the adjacency matrix, where the element M_{aijl} is the average makespan of the chromosomes that have scheduled the job j in machine i exactly one position before the job l, with $i = 1, \ldots, m, j = 1, \ldots, n, l = 1, \ldots, n$.

Memes are built based on the minimum values of matrices Mp, Mr, and Ma. The obtained memes are denoted as I_{pj} (meme1), I_{rj} (meme2), and I_{aj} (meme3), respectively, where $j \in J$ is a reference job chosen at random for the construction of the chain.

The last matrix that comprises the memetic base is Mb, defined as the *meme matrix*, where the previously defined memes are stored. Each meme is linked to an accumulative score S_{cw} that is updated every time that the corresponding agent uses the meme to modify the structure of a chromosome. Said score is given by the function $S_{cw}(current) = S_{cw}(previous) + [f(c') - f(c)]$, with $w = \{1, 2, 3\}$. Furthermore, memes have a counter defined by C_{ow}, which adds one unit to its previous value each time that $v(c) = 'false'$, *i.e.*, when the chromosome is not vulnerable to the meme.

After defining the construction process of information chains Ij, type 2 rules are presented, which allow defining how the information contained in said chains is transcribed into the chromosomes. The transcription procedures of type p2 used by the designed agents are slightly different due to the type of information that they carry. The manipulation procedure consists of a positional exchange of genes within the chromosome, which are linked to the meme contained by the agent.

Lastly, type 3 rules are present throughout the evolutionary process, defining the parameters and criteria that dominate the operativity of the algorithm. Table **VI** summarizes the parameters and variables that complement the previously discussed terms. These are also necessary during the construction of the algorithm.

Consequently, the following criteria and rules are defined:

- The creation of the initial population of chromosomes C is performed randomly, where each chromosome represents a solution to the JSSP.
- Chromosomes are assessed based on their fitness using the function f, where $f(c) = Cmax(c)$ for $c \in C$.
- Memes are initially created with the information of the chromosomes present in the initial population. The selection procedure is controlled by the following expression: if $f(c) - min(f) \leq Qi[max(f) - min(f)]$, then the chromosome is selected to continue with the extraction procedure; otherwise, it is rejected.
- The transgenic agent that attacks the given chromosome is chosen in terms of the highest meme score regarding the Mb. In case of a tie, it is chosen at random. It is chosen in this way in the first generation since $S_{cw} = 0$ for $w = \{1, 2, 3\}$.
- Chromosomes attack each other in one generation with the selected agent.
- If chromosome c is vulnerable to the attack, the new chromosome c' takes its place in population C.

Table VI. General parameters and variables of the transgenic algorithm

Parameter/Variable	Description
Size of the population (q)	This refers to the number of chromosomes created in the initial population. This value remains constant throughout the evolutionary process.
Non-improvement counter (C_n)	This variable increases by one unit when the best value of the makespan does not change within a generation.
Non-improvement limit (C_{nmax})	This is the maximum value of generations of the variable C_n.
Immunity limit (C_{omax})	This refers to the maximum value of counter Co_w, for $w = 1, 2, 3$.
Information quality (Q_i)	This factor between 0 and 1 intervenes in the selection procedure. It refers to the information considered in the updating process of the MB.

- The score of memes is updated each time that a chromosome is manipulated.
- The matrices Mp, Mr, and Ma are updated with the information of the manipulated chromosomes. Additionally, counter C_{ow}, is checked for $w = \{1, 2, 3\}$. If it exceeds Co_{max}, a new meme is generated instead, and the counter and score are also reset.
- If counter $Cn = Cn_{max}/3$, the MB is completely reset by subtracting information of the current population and creating new memes from it.

The proposed algorithm has now been defined. It represents the method used to generate and update the production timetables, since it is integrated both in the predictive and reactive stages of the rescheduling process. The reader can recall that this is the chosen strategy to solve the JSRP in the presented manufacturing environment.

The corresponding integration of the algorithm with the manufacturing environment leads to the following diagram, which represents the utility and operation of the algorithm within said environment.

As seen in Fig. 5, the proposed algorithm is integrated into the manufacturing environment both in the predictive and reactive phase. An initial production timetable is obtained for the transgenic algorithm. Then, said timetable is executed, and decisor A defines whether a rescheduling factor is needed. If this is the case, then the timetable is interrupted and enters the reactive phase to be repaired by the transgenic algorithm. Otherwise, decisor B defines whether the timetable has been completely executed. The timetable is then deemed to be executed, and a new rescheduling period begins. Otherwise, the timetable continues its execution.

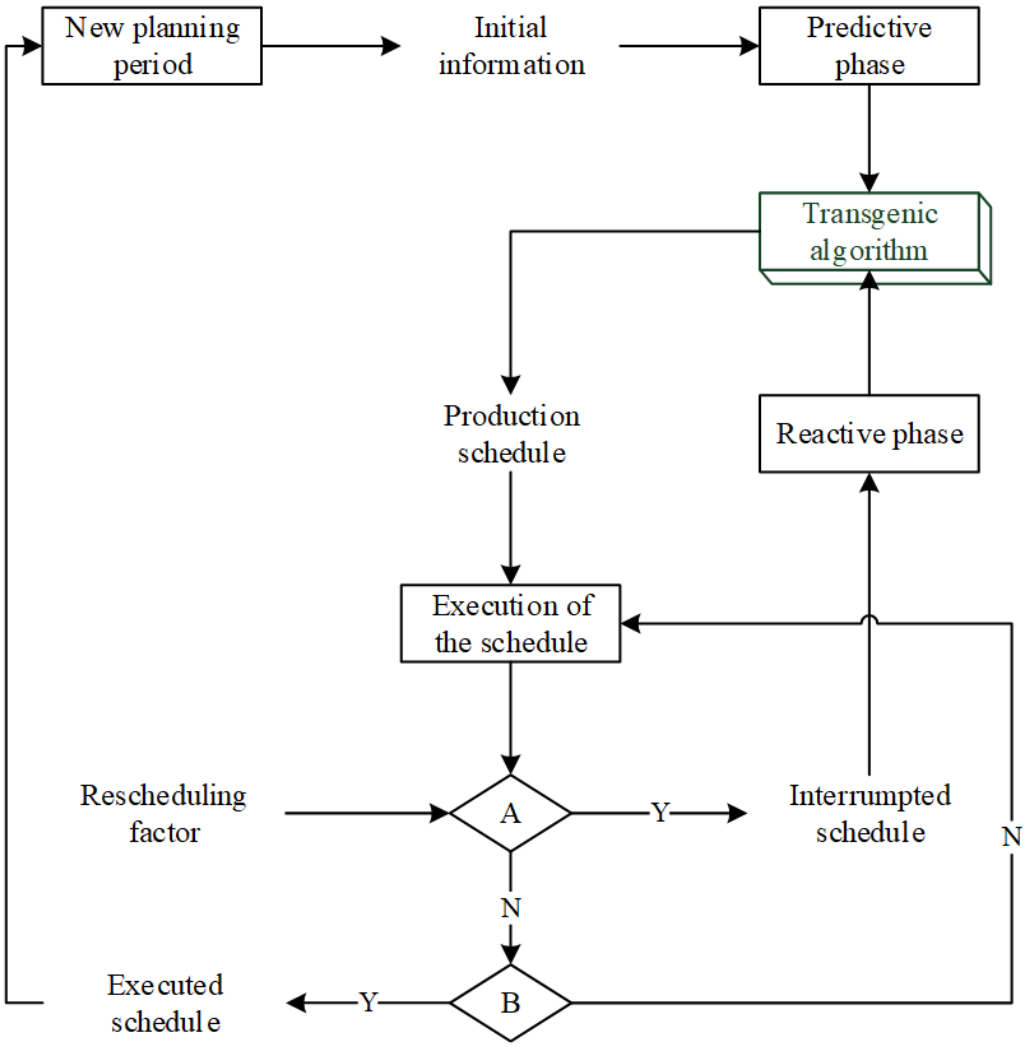

Figure 5. Integration of the transgenic algorithm in the manufacturing environment

3. Results

In order to assess the performance of the proposed algorithm, this study has been divided into two sections: the first one details the experimentation carried out with the transgenic algorithm in the predictive phase, and the second section involves testing the algorithm in the reactive phase.

3.1. Predictive phase

As described in the third section, this stage includes the generation of the initial production timetable. In order to solve the JSSP, many techniques have been developed. A study on said techniques can be found in (44). Furthermore, these techniques have been tested with comparison instances proposed by different authors. In order to validate the performance of the transgenic algorithm, we selected some alternatives, such as the ones proposed by (45–49), and (50).

The results shown below were obtained with the parameterization described in Table VII. Said parameters were set upon the basis of factorial experiments, which are not described in this article in order to avoid content saturation.

Table VII. Parameterization of the transgenic algorithm for the predictive phase

Parameter	Instance size		
	Small	Medium	Large
Population size (q)	50	300	500
Non-improvement limit (C_{nmax})	100	100	100
Immunity limit (C_{omax})	5	30	50
Information quality (Q_i)	0,1	0,1	0,1

The above-presented parameters have been defined for all three instance sizes: a small instance contains between 36 and 75 operations, a medium instance between 100 and 200, and a large instance between 225 and 400 operations.

Table VIII summarizes the results obtained for the selected instances. These instances are assessed in terms of performance compared to makespan. Hence, the BK field yields *the best value according to the literature* for each instance (some are consistent with optimal values). The BF field states the *best value found* by the proposed algorithm. The GAP field represents the relative difference of BF compared to BK in percentage form: $GAP = 100 \times (BF - BK)/BK$. The two following fields correspond to the average makespan and its standard deviation obtained for five replicas of the algorithm with the instance.

The last field corresponds to the average execution time in seconds. The BF values in bold show that the proposed algorithm has found the BK. The algorithm was implemented in MATLAB. These results were obtained using a computer with an Intel Core i5-8300H @4.0 Ghz and 8GB RAM.

Table VIII. Results obtained using the proposed algorithm for the selected instancesv

Name	*n*	*m*	BK	BF	GAP	AVG	SD	RT
Instances proposed by (46)								
Ft06	6	6	55	**55**	0,0	55,0	0,0	1
Ft10	10	10	930	**930**	0,0	941,6	10,8	7
Ft20	20	5	1.165	1.173	0,7	1.178,8	3,1	10
Instances proposed by (47)								
Abz5	10	10	1.234	1.242	0,6	1.243,7	3,1	5
Abz6	10	10	943	947	0,4	947,2	0,4	5

Abz7	20	15	656	685	4,4	692,8	4,8	52
Abz8	20	15	665	698	5,0	707,1	8,5	49
Abz9	20	15	678	697	2,8	713,4	9,4	45
Instances proposed by (48)								
Yn01	20	20	884	917	3,7	930,9	8,4	74
Yn02	20	20	904	933	3,2	942,7	8,0	84
Yn03	20	20	892	928	4,0	937,6	6,8	108
Yn04	20	20	968	1.016	5,0	1.017,8	2,5	170
Instances proposed by (49)								
Orb01	10	10	1.059	**1.059**	0,0	1.073,0	11,0	7
Orb02	10	10	888	894	0,7	896,6	1,0	9
Orb03	10	10	1.005	**1.005**	0,0	1.021,3	9,1	10
Orb04	10	10	1.005	1.011	0,6	1.019,0	7,9	8
Orb05	10	10	887	894	0,8	894,5	1,3	6
Orb06	10	10	1.010	**1.010**	0,0	1.026,2	6,3	8
Orb07	10	10	397	**397**	0,0	400,3	3,7	5
Orb08	10	10	899	912	1,4	924,8	8,1	7
Orb09	10	10	934	**934**	0,0	940,7	4,4	6
Orb10	10	10	944	**944**	0,0	955,5	8,6	9
Instances proposed by (50)								
La01	10	5	666	**666**	0,0	666,0	0,0	1
La02	10	5	655	**655**	0,0	655,0	0,0	1
La03	10	5	597	**597**	0,0	597,0	0,0	1
La04	10	5	590	**590**	0,0	590,0	0,0	1
La05	10	5	593	**593**	0,0	593,0	0,0	1
La06	15	5	926	**926**	0,0	926,0	0,0	1
La07	15	5	890	**890**	0,0	890,0	0,0	1
La08	15	5	863	**863**	0,0	863,0	0,0	1
La09	15	5	951	**951**	0,0	951,0	0,0	1
La10	15	5	958	**958**	0,0	958,0	0,0	1
La11	20	5	1.222	**1.222**	0,0	1.222,0	0,0	3
La12	20	5	1.039	**1.039**	0,0	1.039,0	0,0	3
La13	20	5	1.150	**1.150**	0,0	1.150,0	0,0	2
La14	20	5	1.292	**1.292**	0,0	1.292,0	0,0	2
La15	20	5	1.207	**1.207**	0,0	1.207,0	0,0	3
La16	10	10	945	**945**	0,0	947,6	3,6	2
La17	10	10	784	**784**	0,0	785,6	2,3	3
La18	10	10	848	**848**	0,0	850,8	5,7	4
La19	10	10	842	**842**	0,0	847,2	5,8	3

La20	10	10	902	907	0,6	909,2	2,0	4
La21	15	10	1.046	1.053	0,7	1.067,9	8,9	11
La22	15	10	927	935	0,9	936,0	2,1	13
La23	15	10	1.032	**1.032**	0,0	1.032,0	0,0	11
La24	15	10	935	943	0,9	970,0	12,9	14
La25	15	10	977	**977**	0,0	995,6	10,9	13
La26	20	10	1.218	**1.218**	0,0	1.223,8	4,8	17
La27	20	10	1.235	1.249	1,1	1.271,8	13,5	24
La28	20	10	1.216	1.227	0,9	1.241,6	10,5	26
La29	20	10	1.152	1.201	4,3	1.210,4	7,9	33
La30	20	10	1.355	**1.355**	0,0	1.355,0	0,0	23
La31	30	10	1.784	**1.784**	0,0	1.784,0	0,0	28
La32	30	10	1.850	**1.850**	0,0	1.850,0	0,0	33
La33	30	10	1.719	**1.719**	0,0	1.719,0	0,0	29
La34	30	10	1.721	**1.721**	0,0	1.721,0	0,0	43
La35	30	10	1.888	**1.888**	0,0	1.888,0	0,0	46
La36	15	15	1.268	1.297	2,3	1.301,8	7,6	53
La37	15	15	1.397	1.436	2,8	1.446,2	6,2	65
La38	15	15	1.196	1.225	2,4	1.234,8	9,2	57
La39	15	15	1.233	1.251	1,5	1.252,8	4,0	58
La40	15	15	1.222	1.243	1,7	1.252,0	8,5	59
Instances proposed by (51)								
Ta01	15	15	1.231	**1.231**	0,0	1.233,2	3,9	47
Ta02	15	15	1.244	**1.244**	0,0	1.245,4	3,1	49
Ta03	15	15	1.218	**1.218**	0,0	1.221,6	5,1	56
Ta04	15	15	1.175	1.200	2,1	1.209,4	6,3	93
Ta05	15	15	1.224	1.248	2,0	1.265,6	12,2	112
Ta11	20	15	1.357	1.421	4,7	1.430,6	10,8	150
Ta12	20	15	1.367	1.430	4,6	1.441,2	6,5	184
Ta13	20	15	1.342	1.391	3,7	1.426,4	24,1	157
Ta14	20	15	1.345	1.359	1,0	1.375,8	15,1	93
Ta15	20	15	1.339	1.438	7,4	1.442,0	3,5	196
Ta21	20	20	1.642	**1.642**	0,0	1.646,6	5,5	233
Ta22	20	20	1.600	1.645	2,8	1.656,0	7,9	265
Ta23	20	20	1.557	1.644	5,6	1.670,4	16,4	380
Ta24	20	20	1.644	1.691	2,9	1.708,4	14,8	320
Ta25	20	20	1.595	1.719	7,8	1.741,2	22,9	338

Table IX summarizes the results obtained after applying the transgenic algorithm to the comparison instances. The average values obtained for the $n \times m$ settings are shown. In general, the algorithm

performs well in the predictive phase, delivering the best possible values for 52 % of the selected instances and similar values for the remaining instances. These values, on average, do not surpass 1,3 % in the differences for medium instances and 4,2 % for large ones. The algorithm is fairly consistent since the standard deviation is acceptable. Regarding the average execution time, the obtained values are below the minute, except for the last three settings in the set of large instances.

Table IX. Summary of the results obtained for the proposed algorithm

Small instances				
n	m	$GAP_{prom.}$	$SD_{prom.}$	$RT_{prom.}$
6	6	0,0	0,0	1,0
10	5	0,0	0,0	1,0
15	5	0,0	0,0	1,0
Medium instances				
20	5	0,1	0,5	3,8
10	10	0,3	5,3	6,0
15	10	0,5	7,0	12,4
20	10	1,3	7,4	24,6
Large instances				
30	10	0,0	0,0	35,8
15	15	1,5	6,6	64,9
20	15	4,2	10,3	115,8
20	20	3,9	10,4	219,1

On the other hand, it can be observed that the algorithm's performance slightly deteriorates as the complexity of the instances increases, *i.e.*, when there are more operations to process. Once the algorithm had displayed commendable performance in terms of average gap, standard deviation, and execution time for the three identified sets of instances, its performance in the reactive phase was tested.

3.2. Reactive phase

Once the initial production timetable is derived from the predictive phase, rescheduling factors will occur at different times of the execution. In this sense, a set of experiments was designed to validate the performance of the algorithm in comparison with the genetic algorithm, an essential reference in evolutionary algorithms. This technique has been used in a wide range of combinatory problems, as mentioned in previous sections. An application of genetic algorithms for operation scheduling and rescheduling can be consulted in (19).

First, given that there are many rescheduling factors, one factor is chosen for each class. It is worth mentioning that, although only one is chosen, the entire class can be solved by the algorithm with a similar procedure. The algorithm can process any of the 17 identified factors. In general, the experiments were designed in terms of the following factors and levels:

Table X. Proposed experiment

Levels		
Factors	**1**	**2**
Instance size	Small	Large
Interruption incidence	Early	Tardive
Duration of the interruption	Short	Long

Given the previous factors and levels, a 2^k factorial design was established, with $k = 3$, corresponding to the number of factors. This setting offers a total of eight treatments, which are detailed in Table **XI**:

Table XI. Treatment resulting from the factorial experiment

Treatment	Size	Incidence	Duration
1	Small	Early	Short
2	Small	Early	Long
3	Small	Tardive	Short
4	Small	Tardive	Long
5	Large	Early	Short
6	Large	Early	Long
7	Large	Tardive	Short
8	Large	Tardive	Long

In general, a set of instances were chosen for each class in the predictive phase, in order to design different scenarios. Ten scenarios were built for each treatment, and five replicas were analyzed for each scenario. Regarding the specific values of the levels of each factor, instances with $n \times m$ configurations containing 100 operations were considered as small, and those with 400 operations as large. The early incidence includes a random point between 20 and 40 % of the instance makespan. The tardive instance is located between 60 and 80 % of the makespan. Lastly, a short duration corresponds to 40 % of the maximum processing time among all instance operations, and a long duration corresponds to 160 % of the same parameter.

After clarifying the structure of the experiments, the most relevant sections of the genetic algorithm design are presented, with Eq. (6) serving as a reference point to establish the comparison with the transgenic algorithm.

$$E = 1 - \left(\frac{C^*_{\max} - C_{\max}}{C_{\max}} \right) \times 100\,\% \tag{6}$$

where $C_{\max}$ corresponds to the original timetable makespan, and $C^*_{\max}$ corresponds to the makespan of the repaired timetable. The value of E represents the efficiency level obtained for the new timetable in percentage form. A higher value of E means a higher efficiency of the initial timetable.

The *stability* of the timetable refers to the deviation of the repaired timetable when compared to the initial one (52). It can be measured by calculating the average differences between the starting times of the repaired and initial timetables. Its calculation is obtained via Eq. (7):

$$D = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} |r_{ij}^{*} - r_{ij}|}{\rho(R)} \qquad \text{where} \qquad O_{ij} \in R \tag{7}$$

where r_{ij} is the starting time of operation O_{ij} in the initial timetable, and r_{ij}^{*} is the starting time of the same operation in the repaired timetable. R is the set of remaining operations to be processed from the time instant at which the rescheduling factor appears. Lastly, $\rho(R)$ refers to the cardinality of set R.

On another note, the transgenic algorithm for the reactive phase encompasses what is known as the *stability factor*, which enables the planning staffer to establish a share of operations in R that need to maintain the same sequence as the initial timetable. Hence, the *nervousness* of the system can be mitigated (3). In this study, the stability factor was determined based on an additional experiment design, carried out with the same factors and levels presented in Table **X**.

The efficiency (E) and stability (D) results for both algorithms are presented in this section, for each class of rescheduling factor and for each treatment. They are presented in the form of charts that contain the data regarding the minimum, maximum, and average of the performance metrics obtained for the genetic algorithm (GA) and the transgenic algorithm (TA).

- **Class I: machine failure (1)**

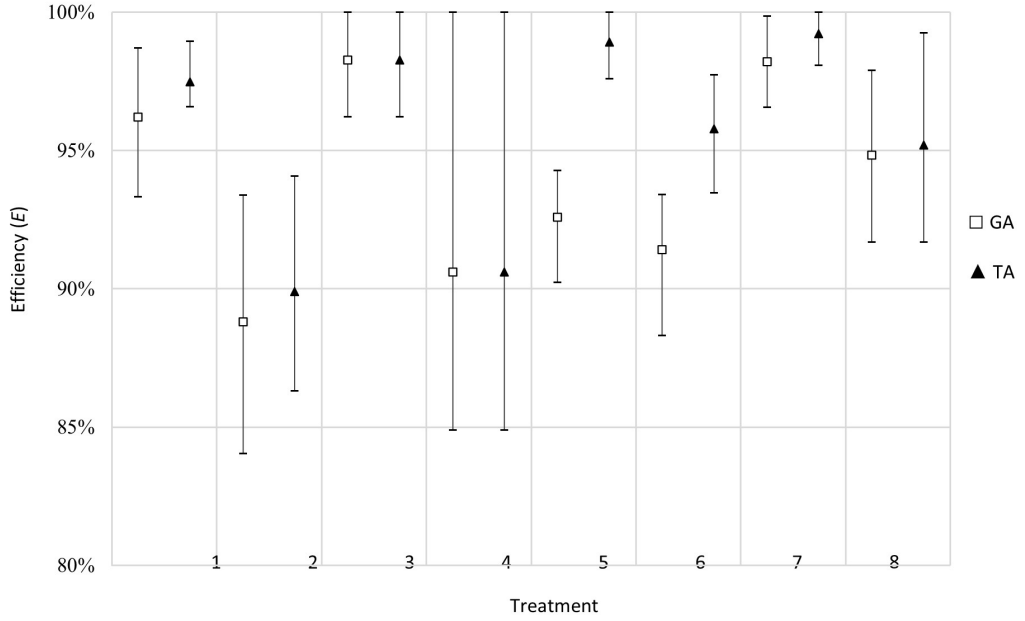

Figure 6. Efficiency results obtained for the *machine breakdown* rescheduling factor

In Fig. **6**, the performance of the TA in terms of efficiency is equal or superior to that of the GA. Furthermore, the proposed algorithm exhibits an efficiency close to 100 % when the interruption is short.

The scenarios with the worst performance are those with small instances and long interruptions, as this could have a more harmful effect on the system floor.

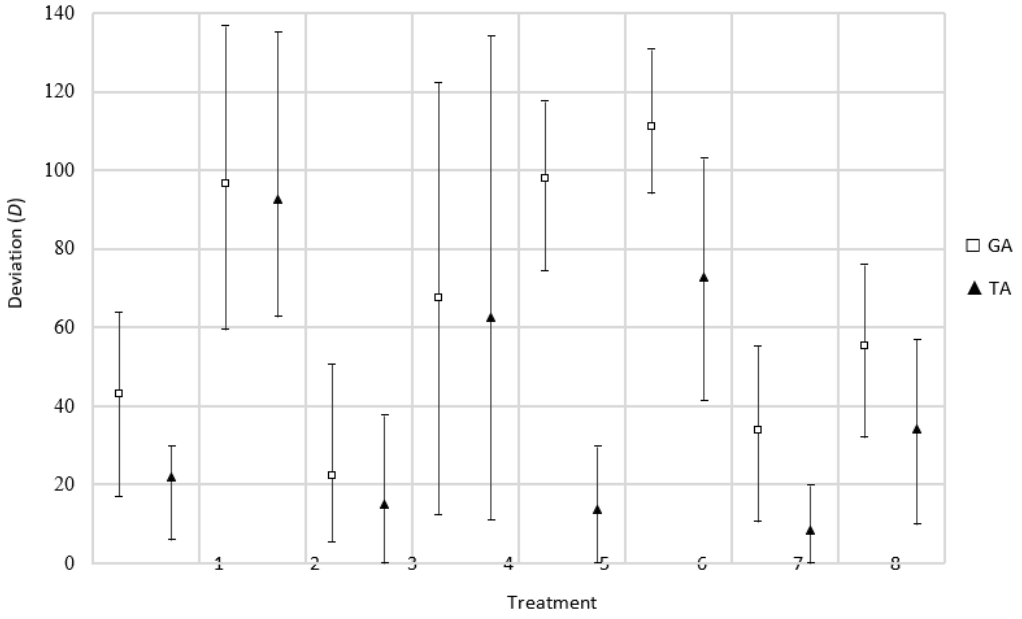

Figure 7. Deviation results obtained for the *machine breakdown* rescheduling factor

As for the deviation, a better performance is obtained, surpassing the GA in all treatments (Fig. 7). Additionally, the TA performs better in the treatments where the instance is large. One particular case regarding the performance differences between the two algorithms corresponds to treatment 5, where the GA has difficulties with delivering a less deviated schedule than the original one.

• Class II: variation in processing time (7)

In this set of treatments, a behavior similar to that of the previous rescheduling factor is observed (Fig. 8). This could indicate that these two classes have the same effects on the interrupted timetable. Another interesting fact is that treatment 4 shows a wide range of results, including the lowest performance. This could be explained by the fact that the worst scenario involves a small instance with the greatest variation in processing times for some operations at a final moment (tardive) of the timetable's execution. The combination of these factors in said levels can lead to a low action range for these algorithms, since there are not many remaining operations that can be rescheduled to compensate for these effects.

In terms of deviation, a better performance is seen once again, surpassing the GA in all treatments. Additionally. The TA shows a better performance in treatments with large instances. A particular case regarding the difference in performance between both algorithms corresponds to treatment 5 where the genetic method seems to have issues with delivering a timetable that is less deviated than the original one.

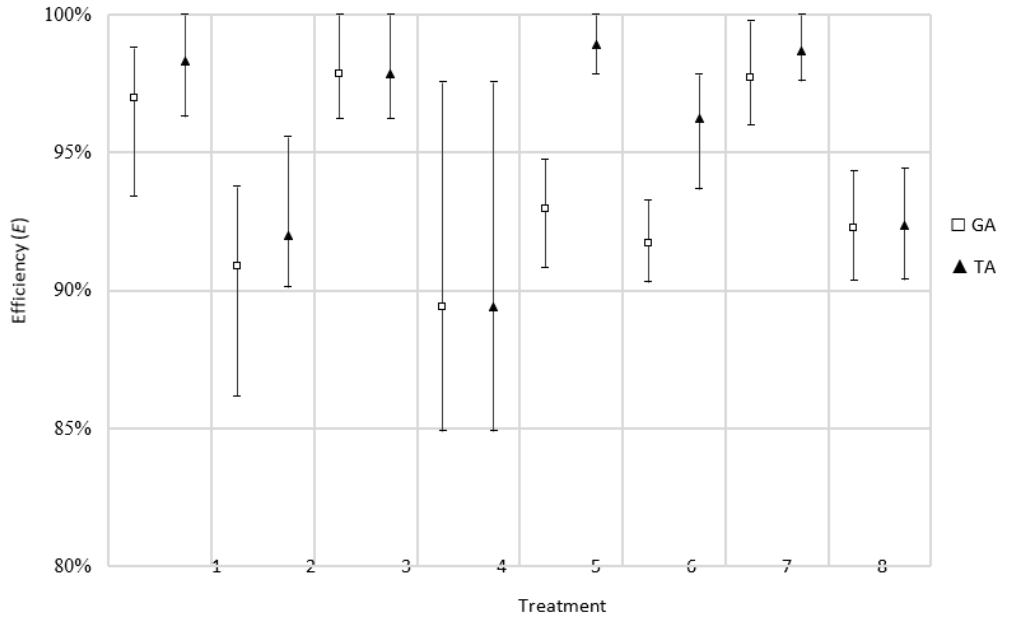

Figure 8. Efficiency results obtained for the *process time variation* rescheduling factor

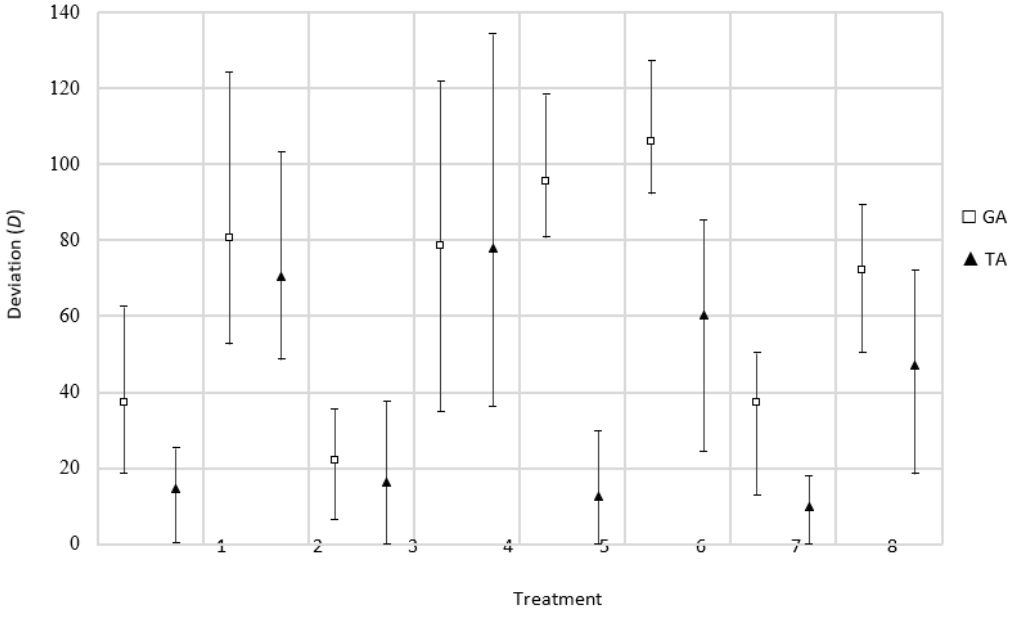

Figure 9. Deviation results obtained for the process time variation rescheduling factor

Regarding the deviation of the timetables, the same behavior of the previous class is evidenced (Fig. 9). Furthermore, it would seem that the interruption incidence factor had no effect in the deviation of the resulting timetables. It is key to acknowledge that, in this class, the deviations of treatments with short interruptions do not exceed 20 units of time.

• Class III: arrival of new job order (11)

In this class, it is important to highlight that the duration of the interruption is conceived analogously through the following considerations: the effect of short and long interruptions is homologated to the effect of including a new job with an average processing time between low and high operations. In this sense, to determine the processing time of the operations in the first level (low), a uniform distribution U (20,60) was considered. For the second level, the distribution was U (80,120). Finally, the number of operations of the incoming job was determined through a uniform distribution U (5,10) for small instances and U (10,20) for large instances.

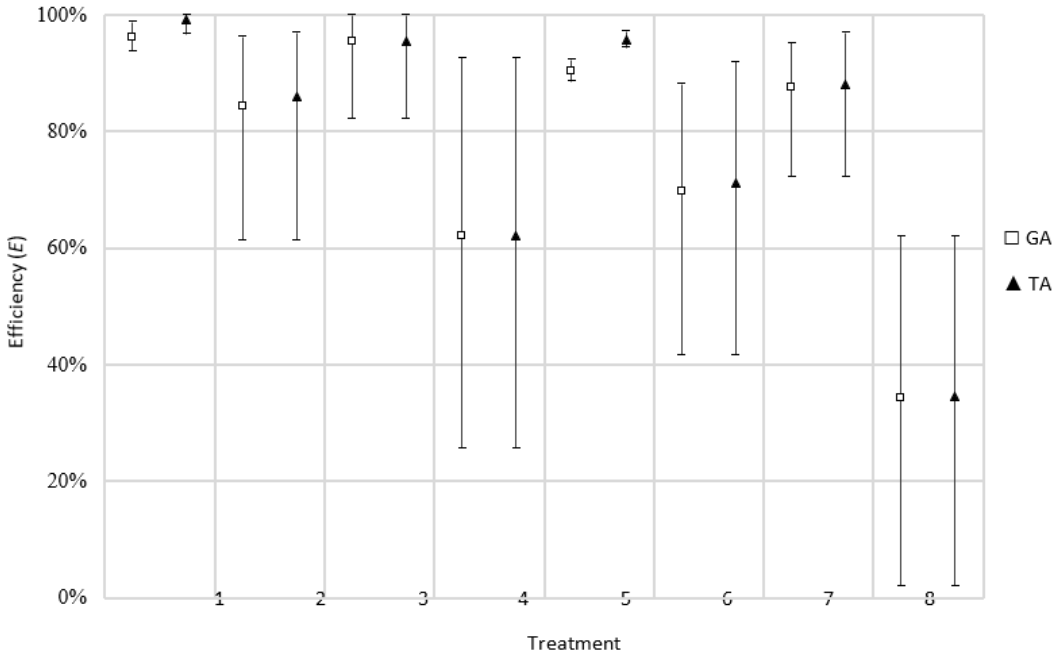

Figure 10. Efficiency results obtained for the *arrival of new job order* rescheduling factor

This leads to the results shown in Fig. 10. The performance of both algorithms is more even than in previous classes. Hence, a generalized behavior can be evidenced, indicating that long duration is the level that most affects the variability results.

In terms of the deviation of the resulting timetables, the TA shows significantly superior results (Fig. 11). These results could be revealing that the initial timetable has not been able to 'absorb' the new job order and has remained almost intact. Thus, the processing of the new order has been delayed for the end. This implies that the efficiency values for this class are not related to the operativity of the algorithms, but with the effect of including a new job after the timetable has begun its execution and, in the worst cases, when it is close to conclusion. Hence, this type of interruption can be handled using the periodic scheduling policy that allows putting the new job order on hold and including it in the next planning period. This ensures an efficient use of productive resources.

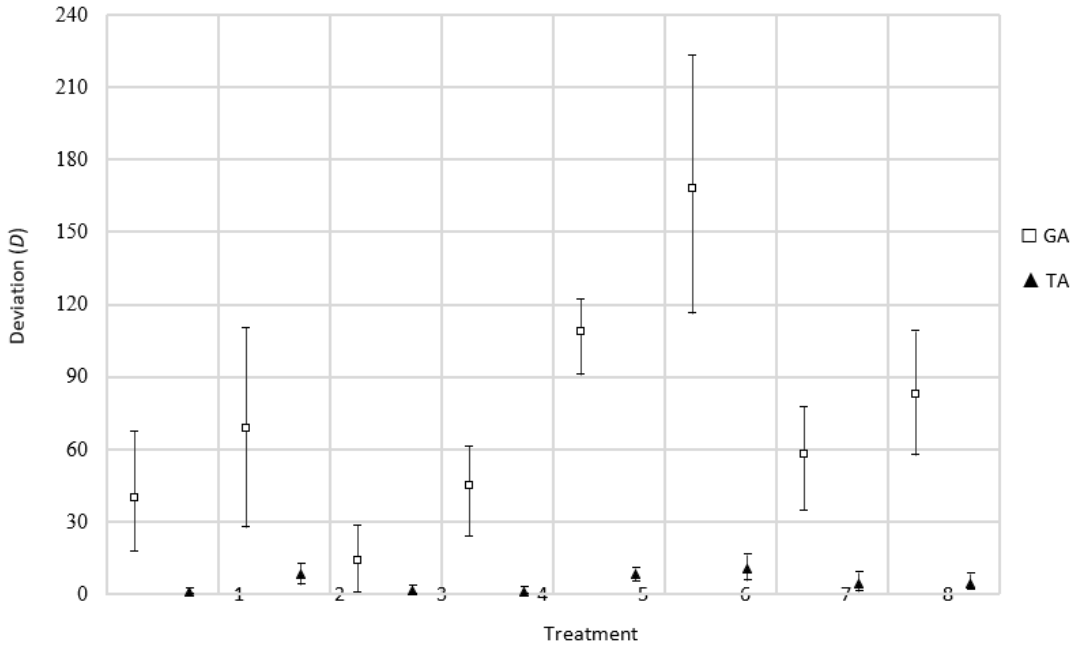

Figure 11. Deviation results obtained for the *arrival of new job order* rescheduling factor

• **Class IV: urgent job (14)**

In this class, given that there is a priority shift in one of the jobs, the interruption duration factor is analog to the number of remaining operations of the urgent job. This means that the duration factor is represented, in the first level, by the job with less remaining operations when the interruption point takes place. The opposite occurs in the second level.

The results obtained for the efficiency metric reveal that the TA has a superior performance (Fig. 12). The average value of this metric in all treatments surpasses 88 %. Furthermore, the most influential factor is the number of operations. When the urgent job has a higher number of remaining operations to be processed, efficiency is affected.

The algorithm once again yields superior results, exhibiting an efficiency equal or over 100 % in all treatments. The most influential factor is the number of remaining operations for the canceled job. Higher values of this parameter translate into a higher efficiency, as the capacity of resources is slightly relieved.

Regarding the deviation of the obtained timetables, this metric is sensitive to the interruption incidence factor (Fig. 13). When the interruption is early, the stability of the timetable decreases. Furthermore, this effect is amplified when the urgent job has more operations yet to be processed.

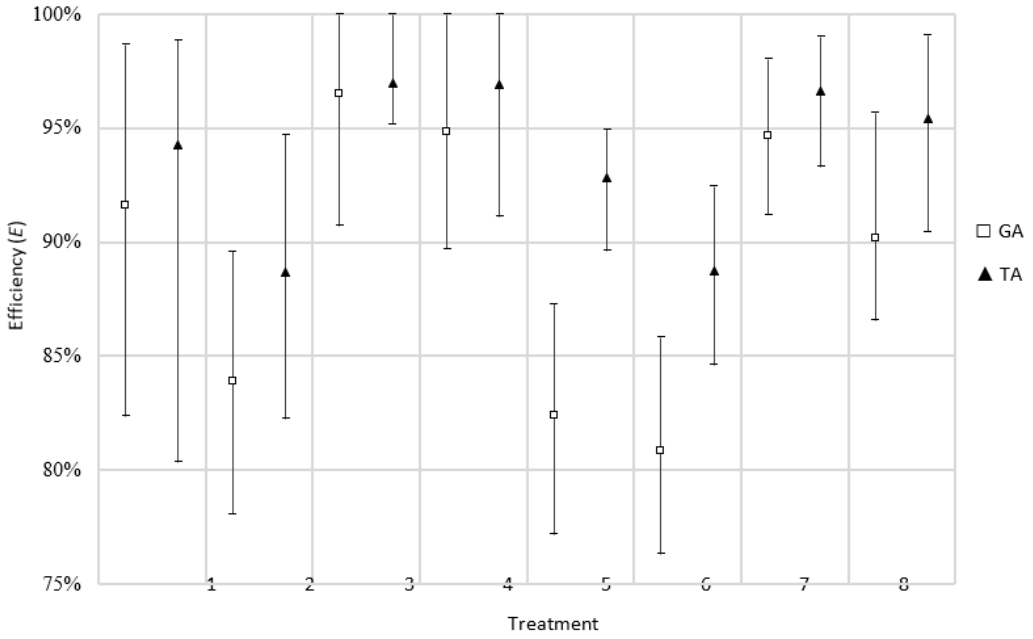

Figure 12. Efficiency results obtained for the *urgent job* rescheduling factor

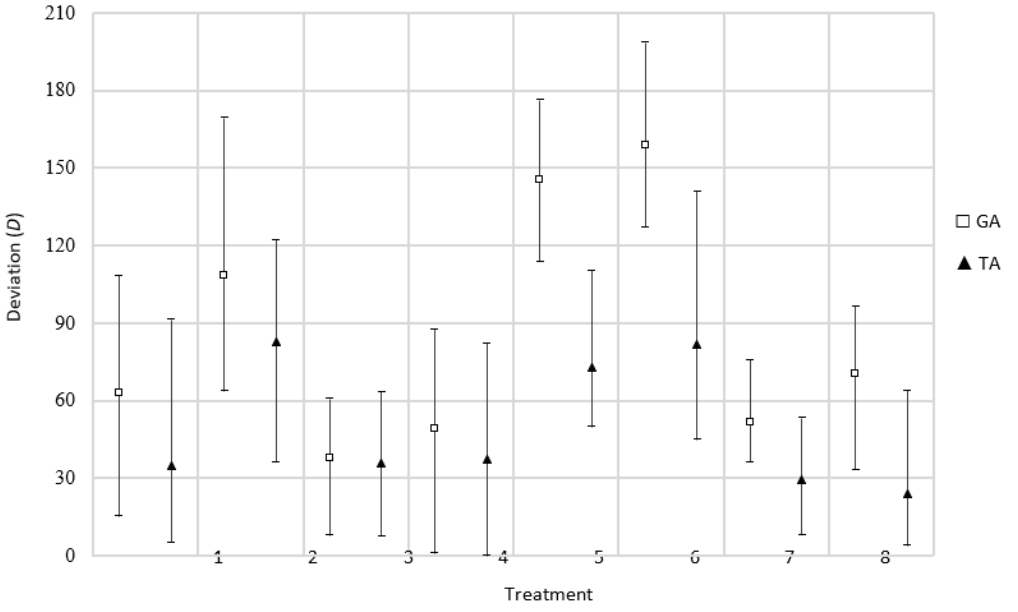

Figure 13. Deviation results obtained for the *urgent job* rescheduling factor

• **Class V: job cancellation (16)**

For this class, the interruption duration factor was approached as in the previous class. In this type of interruption, the algorithms handle the elimination of scheduled jobs in the timetable. Hence, the

makespan after the elimination of these operations can become lower than the initial one. Fig. **14** shows that the values surpass 100 % while maintaining consistency in terms of efficiency.

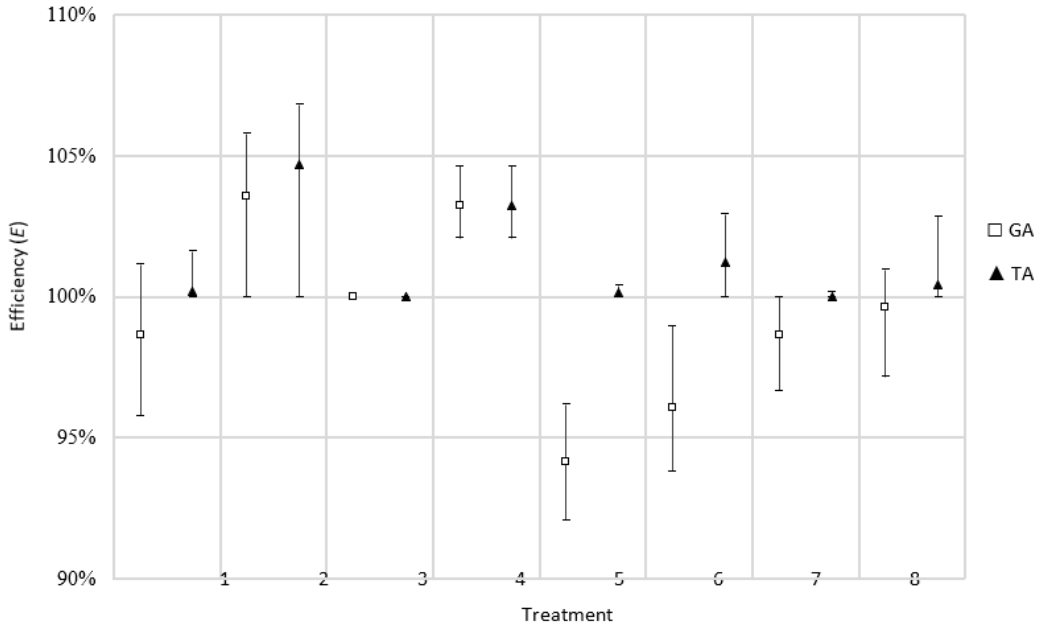

Figure 14. Efficiency results obtained for the *job cancellation* rescheduling factor

Once again, the TA demonstrates superior results, indicating an efficiency of 100 % or higher in all treatments. The most influential factor is the number of remaining operations from the canceled job. A higher number of remaining operations leads to greater efficiency, as resource capacity is slightly relaxed.

The resulting deviation validates the performance of the TA, as shown in Fig. **15**. There is a variability in the results, which is significantly reduced in various treatments. In general, the algorithm outperforms the GA, offering better results in 65 % of the treatments in terms of efficiency and in 98 % of the treatments in terms of stability. The percentage of remaining treatments for both metrics reveal similar performances for both algorithms.

The average efficiency of the TA is 2,3 % higher than that of the GA. The highest difference in performance is observed in class IV, with an efficiency 4,4 % higher on average when compared to the GA. In terms of the stability metric, the TA shows a standard deviation of 28 units, while the GA deviates by 67 units, thus proving that the former is a complete regeneration method that mitigates the nervousness to a greater extent than the latter.

Lastly, the efficiency and stability of a timetable can be conflictive performance metrics, so the person in charge of planning will have to make swift decisions, seeking to obtain a balanced productive system.

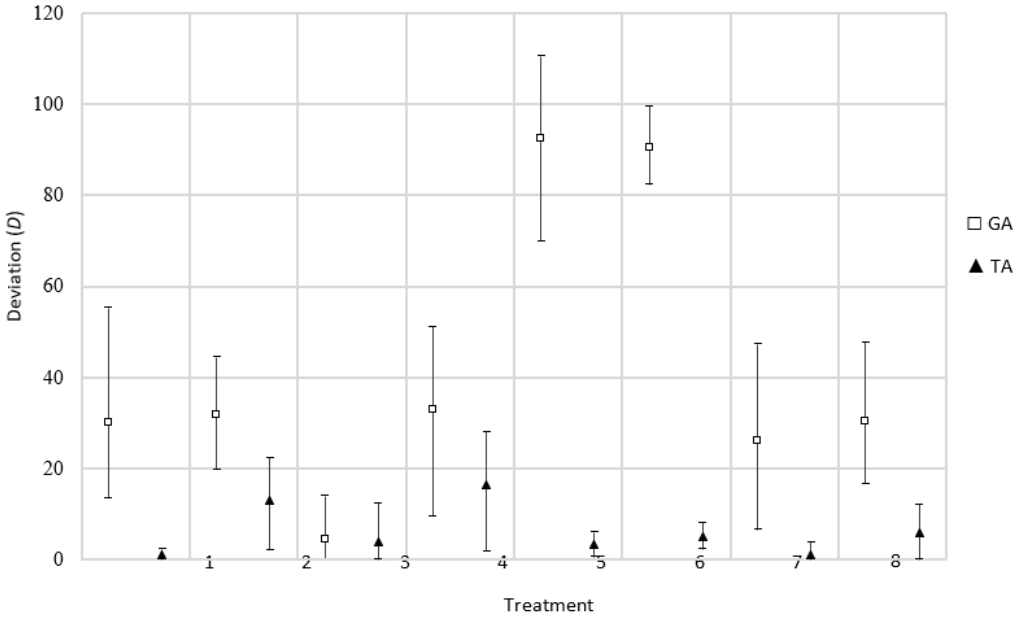

Figure 15. Deviation results obtained for the *job cancellation* rescheduling factor

4. Conclusions

This study focused on developing a computational algorithm to facilitate the job of the people in charge of scheduling production in manufacturing environments. Hence, the theoretical point of view was defined as the starting point in the definition of the Job Shop Scheduling Problem (JSSP).

Given the existing gap between theoretical models and real manufacturing environments, the appearance of unexpected events was included, as they alter the execution of timetables. This sought to provide the proposed algorithm with a more realistic scope. In this sense, a rescheduling framework was selected, which was used to define the Job Shop Rescheduling Problem (JSRP) approach.

Given the increasing complexity entailed by the addition of the aforementioned elements, an algorithm based on transgenic computing was proposed. It was developed while seeking to deliver high-quality production timetables in terms of efficiency and stability.

The performance of the algorithm was validated in both the predictive and the reactive phase. In the predictive phase, a set of well-known instances in JSSP literature were selected. The algorithm performed well, reaching the best-known value in 52 % of the selected instances and similar values for the remaining ones. On average, these values do not exceed 1,3 % of the difference for medium instances and 4,2 % for large instances. Furthermore, as the instance size grows larger, the performance of the algorithm progressively declines. Therefore, the algorithm is not recommended for instances with more than 400 operations, *i.e.*, if similar performance levels are desired.

Regarding the reactive phase, the performance of the transgenic algorithm (TA) was compared to that of the genetic algorithm (GA). The former outperformed the latter, with better results in 65 % of treatments in terms of efficiency and 98 % of treatments in terms of stability. The percentage of remaining treatments for both metrics reveal similar performance levels for both algorithms.

5. Future work

Given the structure of TAs, there is a possibility that more complex agents can be designed, seeking to improve the performance of the algorithm in instances of higher complexity. On the other hand, the performance of the tool can be more comprehensively validated through a simulation-based study, even with the implementation in a real manufacturing environment.

Finally, given the scope of this work, which is limited to the design and validation of the TA applied to the JSRP, said algorithm could be integrated into real manufacturing environments through the design of a graphical user interface.

6. Author Contributions

All authors contributed equally to the research.

References

[1] J. K. Lenstra and A. H. G. R. Kan, "Computational complexity of discrete optimization problems," *A. Discrete Math.*, vol. 4, 2005, pp. 121-140. https://doi.org/10.1016/S0167-5060(08)70821-5 ↑2, 3

[2] J. R. King, "The theory-practice gap in job-shop scheduling," *Prod. Eng.*, vol. 55, no. 3, pp. 137-143, 1976. https://doi.org/10.1049/tpe.1976.0044 ↑3, 5

[3] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing system: A framework of strategies, policies, and methods," *J. Sched.*, vol. 6, no. 1, pp. 39-62, 2003. https://doi.org/10.1023/A:1022235519958 ↑3, 5, 6, 11, 25

[4] H.-L. Fang, P. Ross, and D. Corne, "A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems," in *Fifth Int. Conf. Genet. Algorithms*, no. 623, pp. 375-382, 1993. ↑3, 5, 7

[5] H. Li, Z. Li, L. X. Li, and B. Hu, "A production rescheduling expert simulation system," *Eur. J. Oper. Res.*, vol. 124, no. 2, pp. 283-293, 2000. https://doi.org/10.1016/S0377-2217(99)00381-1 ↑3, 5

[6] Y.-C. E. Li and W. H. Shaw, "Simulation modeling of a dynamic job shop rescheduling with machine availability constraints," *Comput. Ind. Eng.*, vol. 35, no. 1-2, pp. 117-120, 1998. https://doi.org/10.1016/S0360-8352(98)00034-5 ↑3, 5

[7] P. Moratori, S. Petrovic, and A. Vázquez, "Match-up strategies for job shop rescheduling," in *Int. Conf. Ind. Eng. Other App. Applied Intel. Sys.*, 2008, pp. 119-128. https://doi.org/10.1007/978-3-540-69052-8_13 ↑3, 5

[8] R. E. M. Bastos, Ïnvestigação de modelos e algoritmos para o problema do caixeiro viajante com múltiplos passageiros e lotação,"doctoral thesis Universidade Federal do Rio Grande do Norte, Natal, 2023. ↑3, 8

[9] E. Gouvêa and M. Goldbarg, "ProtoG: A computational transgenetic algorithm," in *MIC 2001–4th Metaheuristics*, 2001, pp. 625-630. ↑3, 8, 9, 10, 11

[10] M. Goldbarg, E. Goldbarg, and P. Quadr, "Transgenética computacional: Uma aplicação ao problema quadrático de alocação," *Pesqui. Operacional*, vol. 22, pp. 359-386, 2002. https://doi.org/10.1590/S0101-74382002000300005 ↑3, 8, 10

[11] E. F. G. Goldbarg, M. C. Goldbarg, and L. B. Bagi, "Transgenetic algorithm: A new evolutionary perspective for heuristics design," in *Proc. GECCO 2007 Genet. Evol. Comput. Conf. Companion Mater.*, pp. 2701-2708, 2007. https://doi.org/10.1145/1274000.1274040 ↑3, 8, 9, 10, 11

[12] M. C. Goldbarg and E. Gouvêa, "Extra-intracellular transgenetic algorithm applied to the graph coloring problem," *Design*, pp. 321-326, 2001. ↑3, 8, 10, 11

[13] E. F. G. Goldbarg, M. C. Goldbarg, and W. E. Costa, "A transgenetic algorithm for the permutation flow-shop sequencing problem," *WSEAS Trans. Syst.*, vol. 1, no. 3, pp. 40-45, 2004. ↑3, 11

[14] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Handbooks Oper. Res. Manag. Sci.*, vol. 4, pp. 445-522, 1993. https://doi.org/10.1016/S0927-0507(05)80189-6 ↑3

[15] M. L. Pinedo, *Scheduling: Theory, algorithms, and systems*, 6th ed., New York, NY, USA: Springer, 2022. https://doi.org/10.1007/978-3-031-05921-6 ↑3

[16] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms. I. Representation," *Comput. Ind. Eng.*, vol. 30, no. 4, pp. 983-997, 1996. https://doi.org/10.1016/0360-8352(96)00047-2 ↑4, 7, 12, 13

[17] V. Suresh and D. Chaudhuri, "Dynamic scheduling: A survey of research," *Int. J. Prod. Economics*, vol. 32, no. 1, pp. 53-63, 1993. https://doi.org/10.1016/0925-5273(93)90007-8 ↑4

[18] Y. An, X. Chen, K. Gao, L. Zhang, Y. Li, and Z. Zhao, "A hybrid multi-objective evolutionary algorithm for solving an adaptive flexible job-shop rescheduling problem with real-time order acceptance and condition-based preventive maintenance," *Expert Syst. App.*, vol. 212, art. 118711, Feb. 2023. https://doi.org/10.1016/j.eswa.2022.118711 ↑5

[19] C. Bierwirth and D. C. Mattfeld, "Production scheduling and rescheduling with genetic algorithms," *Evol. Comput.*, vol. 7, no. 1, pp. 1-17, 1999. https://doi.org/10.1162/evco.1999.7.1.1 ↑5, 23

[20] S. F. Smith, "Reactive scheduling systems," in *Intelligent scheduling systems*, New York, NY, USA: Springer, 1995, pp. 155-192. https://doi.org/10.1007/978-1-4615-2263-8_7 ↑6, 7

[21] J. C. Bean, J. R. Birge, J. Mittenthal, and C. E. Noon, "Matchup scheduling with multiple resources, release dates and disruptions," *Oper. Res.*, vol. 39, no. 3, pp. 470-483, 1991. https://doi.org/10.1287/opre.39.3.470 ↑7

[22] Y. Zhang and F. Tao, *Optimization of manufacturing systems using the internet if things*, London, UK: Academic Press, 2016. ↑7

[23] R.-K. Li, Y.-T. Shyu, and S. Adiga, "A heuristic rescheduling algorithm for computer-based production scheduling systems," *Int. J. Prod. Res.*, vol. 31, no. 8, pp. 1815-1826, 1993. https://doi.org/10.1080/00207549308956824 ↑7

[24] H.-H. Wu and R.-K. Li, "A new rescheduling method for computer based scheduling systems," *Int. J. Prod. Res.*, vol. 33, no. 8, pp. 2097-2110, 1995. https://doi.org/10.1080/00207549508904804 ↑7

[25] L. K. Church and R. Uzsoy, "Analysis of periodic and event-driven rescheduling policies in shops," *Int. J. Comput. Integr. Manuf.*, vol. 5, no. 3, pp. 153-163, 1992. https://doi.org/10.1080/09511929208944524 ↑7

[26] G. E. Vieira, J. W. Herrmann, and E. Lin, "Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies," *Int. J. Prod. Res.*, vol. 38, no. 8, pp. 1899-1915, 2000. https://doi.org/10.1080/002075400188654 ↑7

[27] M. Tomassini, "A survey of genetic algorithms," in *Annual reviews of computational physics III*, X., Ed., Singapore: World Scientific, 1995, pp. 87-118. https://doi.org/10.1142/9789812830647_0003 ↑7

[28] J. H. Holland *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, Cambridge, MA, USA: MIT Press, 1992. https://doi.org/10.7551/mitpress/1090.001.0001 ↑7

[29] Z. H. Ahmed, "A hybrid genetic algorithm for the bottleneck traveling salesman problem," *ACM Trans. Embedded Comp. Syst.*, vol. 12, no. 1, pp. 1-10, 2013. https://doi.org/10.1145/2406336.2406345 ↑7

[30] G. M. Morris *et al.*, "Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function," *J. Comput. Chem.*, vol. 19, no. 14, pp. 1639-1662, 1998. https://doi.org/10.1002/(SICI)1096-987X(19981115)19:14<1639::AID-JCC10>3.0.CO;2-B ↑7

[31] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," in *Handbook of Metaheuristics*, Boston, MA, USA: Kluwer Academic Publishers, 2003, pp. 105-144. https://doi.org/10.1007/0-306-48056-5_5 ↑7

[32] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy, and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474-488, Oct. 2005. https://doi.org/10.1109/TEVC.2005.850260 ↑7

[33] H. C. Plotkin, "Non-genetic transmission of information: Candidate cognitive processes and the evolution of culture," *Behav. Processes*, vol. 35, no. 1, pp. 207-213, 1995. https://doi.org/10.1016/0376-6357(95)00056-9 ↑7, 10

[34] E. Chattoe-Brown, "Just how (un)realistic are evolutionary algorithms as representations of social processes?" *J. Artif. Soc. Soc. Simul.*, vol. 1, no. 3, pp. 1-2, 1998. ↑7, 10

[35] P. Merz and B. Freisleben, "A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem," in *1999 Cong. Evol. Comp.-CEC99 (Cat. No. 99TH8406)*, 1999, vol. 3, pp. 2063-2070. ↑10

[36] N. J. Radcliffe and P. D. Surry, "Formal memetic algorithms," Evolutionary Computing: AISB Workshop, 1994. https://doi.org/10.1007/3-540-58483-8_1 ↑10

[37] R. Wright, "Nonzero: The logic of human destiny," *Vintage*, vol. 46, no. 46, art. 11, 2002. ↑10

[38] E. O. Wilson, "Consilience: The unity of knowledge," *Vintage*, vol. 31, 1999. ↑10

[39] A. O. Barboza, "Simulação e técnicas da computação evolucionária aplicadas a problemas de programação linear inteira mista," doctoral thesis, Universidade Tecnológica Federal do Paraná, Curitiba, 2005. ↑11

[40] Goldbarg, E. F. G., Castro, M. P., and Goldbarg M. C., "A transgenetic algorithm for the gas network pipe sizing problem," *Computational Methods*, vol. 1, pp. 893-904, 2006. ↑11

[41] J. S. Dhingra, K. L. Musser, and G. L. Blankenehip, "Reactive operations scheduling for flexible manufacturing systems," in 24^{th} *Conf. Winter Sim.*, 1993. https://doi.org/10.1145/167293.167745 ↑11

[42] A. Dutta, "Reacting to scheduling exceptions in FMS environments," *IIE Trans.*, vol. 22, no. 4, pp. 300-314, 1990. https://doi.org/10.1080/07408179008964185 ↑11

[43] V. Subramaniam and A. S. Raheja, "mAOR: A heuristic-based reactive repair mechanism for job shop schedules," *Int. J. Adv. Manuf. Technol.*, vol. 22, no. 9-10, pp. 669-680, 2003. https://doi.org/10.1007/s00170-003-1601-6 ↑11

[44] A. Arisha, P. Young, and M. El Baradie, "Job shop scheduling problem: An overview," in *Int. Conf. Flex. Autom. Intell. Manuf. (FAIM 01)*, 2001, pp. 682-693. ↑11, 19

[45] H. Fisher and G. L. Thompson, *Probabilistic learning combinations of local job-shop scheduling rules*, Englewood Cliffs, NJ, USA: Prentice Hall, 1963. ↑19

[46] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Manage. Sci.*, vol. 34, no. 3, pp. 391-401, Mar. 1988. https://doi.org/10.1287/mnsc.34.3.391 ↑19, 20

[47] T. Yamada and R. Nakano, "A genetic algorithm applicable to large-scale job-shop problems," *Parallel Prob. Solv. Nature*, vol. 2, pp. 283-292, 1992. ↑19, 20

[48] D. Applegate and W. J. Cook, "A computational study of the job-shop scheduling problem," *INFORMS J. Comput.*, vol. 3, pp. 149-156, 1991. https://doi.org/10.1287/ijoc.3.2.149 ↑19, 21

[49] S. Lawrence, "Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques," graduate thesis, Carnegie-Mellon University, 1984. ↑19, 21

[50] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278-285, 1993. https://doi.org/10.1016/0377-2217(93)90182-M ↑19, 21

[51] M. Zweben and M. Fox, *Intelligent scheduling*, San Francisco, CA, USA: Morgan Kaufman Publishers Inc., 1994. ↑22

[52] G. A. Deac and D. T. Lancu, "Trading strategy hyper-parameter optimization using genetic algorithm" in 24^{th} *Int. Conf. Control Syst. Comp. Sci. (CSCS)*, 2023, pp. 121-127. https://doi.org/10.1109/CSCS59211.2023.00028 ↑25

[53] K. Gao, F. Yang, M. Zhou, Q. Pan, and P. N. Suganthan, "Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm," *IEEE Trans. Cybernetics*, vol. 49, no. 5, pp. 1944-1955, May 2019. https://doi.org/10.1109/TCYB.2018.2817240 ↑3, 7

[54] L. Wang, C. Luo, and J. Cai, "A variable interval rescheduling strategy for dynamic flexible job shop scheduling problem by improved genetic algorithm," *J. Adv. Trans.*, vol. 2017, pp. 1-12, Jan. 2017. https://doi.org/10.1155/2017/1527858 ↑6

[55] N. Kundakci and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," *Comp. Ind. Eng.*, vol. 96, pp. 31-51, Jun. 2016. https://doi.org/10.1016/j.cie.2016.03.011 ↑3, 5, 7

Néstor Andrés Beltrán Bernal

Industrial engineer, Universidad Distrital Francisco José de Caldas, MSc in Industrial Engineering, Universidad Distrital Francisco José de Caldas. Lecturer, Industrial Engineering Program, Department of Engineering, Universidad de la Salle.
Email: nbeltran@unisalle.edu.co

José Ignacio Rodríguez Molano

Industrial engineer, Universidad Distrital Francisco José de Caldas. PhD in Informatics Engineering, Universidad de Oviedo. Full profesor, Industrial Engineering Curricular Project, Department of Engineering, Universidad Distrital Francisco José de Caldas.
Email: jirodriguezm@udistrital.edu.co

Diego Ernesto Mendoza Patiño

Industrial engineer, Universidad Distrital Francisco José de Caldas. PhD in Administration, Universidad Autónoma de Querétaro, Mexico. Associate researcher, Department of Engineering, Universidad Antonio Nariño.
Email: diego.mendoza@uan.edu.co