Desarrollo de un algoritmo transgénico para resolver el Job Shop Rescheduling Problem considerando varios tipos de interrupciones

Mauricio Rojas Ortiz.

Cod. 20132015027

Daniel Andrés Chamorro Cardozo.

Cod. 20132015032

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Universidad Distrital Francisco José de Caldas.

Ingeniería Industrial.

Monografía.

Bogotá, D.C.

Octubre 2019.

Desarrollo de un algoritmo transgénico para resolver el Job Shop Rescheduling Problem considerando varios tipos de interrupciones

Mauricio Rojas Ortiz.

Cod. 20132015027

Daniel Andrés Chamorro Cardozo.

Cod. 20132015032

Trabajo de monografía para optar al título de ingeniero industrial.

Director.

Msc. Néstor Andrés Beltrán

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Universidad Distrital Francisco José de Caldas.

Ingeniería Industrial.

Bogotá, D.C.

Octubre 2019.

Nota de aceptación

Trabajo de monografía

_____

Jurado

_____

Director

Bogotá D.C., 2019

Resumen

El presente trabajo expone el desarrollo y aplicación de un algoritmo transgénico a lo que se ha estudiado como el Job Shop Rescheduling Problem (JSRP). El algoritmo constituye una herramienta útil para los programadores de la producción, con el fin de obtener cronogramas de buena calidad en términos de eficiencia y estabilidad. Adicionalmente, la capacidad de poder reaccionar rápidamente ante los eventos inesperados o interrupciones, que se puedan presentar durante la ejecución del cronograma de producción, logrando así mitigar su impacto sobre el desempeño del sistema de manufactura. El algoritmo propuesto, ha demostrado buenos resultados, al ser probado con instancias de comparación reconocidas del Job Shop Scheduling Problem (JSSP) y al ser comparado con el desempeño de un algoritmo genético.

Tabla de contenido

Lista de tablas

Lista de Figuras

Introducción

Para las personas encargadas de la planeación y programación de la producción, la secuenciación de trabajos en un entorno Job Shop es uno de los problemas más difíciles de tratar, esto se debe a su naturaleza combinatoria y por pertenecer a la clase de problemas NP-Hard, ya que estos exigen el desarrollo de algoritmos avanzados que puedan brindar soluciones cercanas al óptimo haciendo un uso eficiente del recurso computacional. Por otro lado, la mayoría de los modelos clásicos han abordado la programación de trabajos, de una manera estática en la cual no se contemplan la aparición de eventos inesperados en la ejecución de los programas de producción, lo cual ha generado una brecha entre los modelos teóricos y los entornos de manufactura reales.

Acorde a lo anterior, en este trabajo abordamos el Job Shop Scheduling Problem (JSSP) desde el punto de vista de la reprogramación, ya que este enfoque nos permite representar de una mejor forma, las condiciones en las que operan los entornos de manufactura en la realidad. La reprogramación es un proceso que permite reaccionar ante eventos inesperados, los cuales hacen parte de la dinámica de dichos entornos. En este sentido surge lo que se ha estudiado como el Job Shop Rescheduling Problem (JSRP), que aunado con el problema expuesto inicialmente, supone un problema de mayor complejidad.

Los estudios en reprogramación de sistemas de manufactura, son amplios y muy diversos, por lo cual es necesario definir en marco de referencia apropiado para la delimitación del estudio. Dependiendo de cómo se defina, el estudio tendrá una orientación específica. Por lo cual, el presente trabajo se encuentra enmarcado, en los métodos para reparar un cronograma de producción que ha sido afectado por la aparición

de un evento inesperado. El método a desarrollar en secciones posteriores, se encuentra basado en la arquitectura de un algoritmo transgénico. Este se ha destacado por ser una metaheurística aplicada a diversos problemas de gran complejidad, obteniendo resultados muy competitivos con respecto a otras técnicas más reconocidas.

El objetivo principal de este trabajo, consiste entonces, en obtener una herramienta computacional que les permita a los programadores de la producción, generar cronogramas de buena calidad en términos de eficiencia y estabilidad, además de poder reaccionar ante los eventos inesperados o interrupciones, que se puedan presentar durante la ejecución del cronograma de producción, para de esta manera mitigar su impacto sobre el rendimiento del sistema de manufactura. Esta herramienta es constituida principalmente por el algoritmo transgénico propuesto, basado en la metáfora de la computación transgénica.

1. Formulación del problema

1.1. Descripción del problema

El scheduling o la programación de tareas es un tema que ha sido abordado ampliamente por la comunidad investigativa. Se entiende como un proceso de toma de decisiones con el fin de optimizar uno o más objetivos y consiste básicamente en la asignación de recursos escasos durante un intervalo de tiempo. Usualmente a estos recursos se les denota como máquinas, aunque no son el único tipo, esto dependerá de sistema que se esté analizando. Así, el problema en un entorno de manufactura, se basa en la asignación de tareas propias de una orden de producción, en una serie de máquinas. La manera en que cada una de ellas sea asignada, tendrá un impacto directo en la medida

de desempeño que se esté evaluando, como puede ser el Makespan, el tiempo medio de flujo, el ocio total, la tardanza total u otras.

Dependiendo de la naturaleza del sistema, el problema puede ser planteado de diferentes formas. Por ejemplo, pueden llegar un sin número de trabajos, todos con la misma secuencia de producción o diferentes secuencias para cada uno. Este segundo caso es el que se conoce como entorno de manufactura tipo Job Shop, el cual consiste en un conjunto finito de n trabajos $\{J_i\}_{i=1}^{n}$ para ser procesado en un conjunto finito de m máquinas $\{M_k\}_{k=1}^{m}$, donde cada trabajo puede pasar por todas o algunas de las máquinas, basados en cadenas de operaciones que no son necesariamente iguales.

De esta manera, si se desea realizar scheduling en un entorno Job Shop, surge un problema de optimización combinatoria conocido como Job Shop Scheduling Problem (JSSP), que a su vez pertenece a la clase NP-Hard. Por tal razón es común tratar este tipo de problemas con técnicas heurísticas o metaheurístas, que permitan obtener buenas soluciones en términos de la medida de desempeño seleccionada, en un tiempo computacional razonable.

Dada la dificultad inherente a este problema, la mayoría de los modelos clásicos han abordado el scheduling de una manera estática, en la cual no se contemplan la aparición de eventos inesperados en la ejecución de los programas de producción, esto ha generado una brecha entre los modelos teóricos y los problemas reales. Es por ello que en muchas ocasiones, los cronogramas obtenidos por los planeadores resultan inservibles con respecto a la ejecución de las operaciones llevadas a cabo en el piso del sistema. Se necesitan entonces modelos a través de los cuales se tenga mayor acercamiento a la dinámica real de los sistemas de producción, contemplando así los

eventos inesperados y sus efectos dentro del sistema. En la literatura estos eventos se conocen como factores de reprogramación o interrupciones y pueden aparecer por diferentes razones, como por ejemplo el fallo de una máquina, ausentismo de un trabajador, llegadas de nuevos trabajos, fallo de una herramienta, entre otras.

Cada que se presenta una interrupción, se genera la necesidad de hacer un cambio en el cronograma, momento en el cual se hace importante realizar una reprogramación de tareas, o lo que comúnmente se conoce como rescheduling. Así, el rescheduling se entiende como la actualización del cronograma de producción existente, que debe ser realizada de forma rápida y apropiada para poder minimizar los efectos que puedan deteriorar el desempeño del sistema productivo.

En el presente trabajo se pretende resolver el problema de rescheduling en un sistema de manufactura Job Shop (o lo que se conoce como Job Shop Rescheduling Problem - JSRP) teniendo en cuenta diferentes tipos de interrupciones y que pueden aparecer en cualquier instante de tiempo. Para tal efecto se hará uso de un algoritmo de computación evolutiva, llamado algoritmo transgénico, que se medirá y evaluará en términos de estabilidad y eficiencia.

1.2. Planteamiento del problema

Elaborar un algoritmo computacional para la generación y reparación de cronogramas de producción de buena calidad en términos de eficiencia y estabilidad, que permita responder de mejor forma a las condiciones reales presentadas en los entornos de manufactura del tipo Job Shop.

1.3. Variables de investigación

- **Makespan:** se refiere al tiempo requerido para terminar todas las operaciones. Esta variable se usa como medida de desempeño en la generación de cronogramas producción.
- **Tamaño de la instancia:** se refiere al número de operaciones que contiene la instancia particular del problema Job Shop a resolver. Está dado por la multiplicación del número de trabajos y el número de estaciones de trabajo o máquinas.
- **Punto de interrupción:** se refiere al momento en la ejecución del cronograma de producción en que se presenta una interrupción. Su comportamiento se describe a partir de una función de distribución de probabilidad.
- **Duración de la interrupción:** se refiere al tiempo consumido por la interrupción presentada. Su comportamiento se describe a partir de una función de distribución de probabilidad.
- **Porcentaje de eficiencia:** se refiere a la diferencia porcentual entre el makespan del cronograma de producción reparado con respecto al makespan del cronograma de producción inicial. Se usa como medida de desempeño para determinar la eficiencia de los cronogramas de producción obtenidos por el algoritmo de reprogramación propuesto.
- **Desviación:** se refiere a la diferencia promedio de los tiempos de inicio de las operaciones del cronograma reparado con respecto al cronograma inicial. Se usa como medida de desempeño para determinar la estabilidad de los

cronogramas de producción obtenidos por el algoritmo de reprogramación propuesto.

1.4. Delimitación del problema

El algoritmo propuesto en su fase predictiva (generación de cronogramas de producción) está en la capacidad de resolver instancias en ambientes Job Shop con las siguientes características:

- Cada máquina solo puede procesar un trabajo a la vez.
- Cada trabajo solo puede ser procesado por una maquina a la vez.
- Los trabajos deben ser procesados en cada máquina solo una vez.
- La secuencia tecnológica de cada trabajo está completamente definida.
- Los tiempos de procesamiento de todas las operaciones son determinísticos.
- Los tiempos de alistamiento se incluyen en el tiempo de procesamiento de la operación.
- Los tiempos de alistamiento son independientes.
- La medida de desempeño utilizada para evaluar la eficiencia del algoritmo es el makespan.

El algoritmo propuesto en su fase reactiva (reparación de cronogramas de producción) está en la capacidad de enfrentar los siguientes factores de reprogramación o interrupciones:

- Fallo de máquina

- Mantenimiento de máquina
- Ausentismo
- Fallo de herramienta
- Demora en el transporte de material
- Agotamiento de materia prima
- Variación en el tiempo de procesamiento
- Variación en el desempeño de la máquina
- Desgaste de herramienta
- Variación en los tiempos de alistamiento
- Llegada de un nuevo trabajo
- Reproceso
- Rechazo
- Trabajo urgente
- Cambio de prioridad
- Cancelación de orden
- Tercerización

2. Objetivos

2.1. Objetivo general

Obtener un algoritmo evolutivo basado en la computación transgénica para resolver el problema de rescheduling en un entorno Job Shop considerando varios tipos de interrupciones.

2.2. Objetivos específicos

- Caracterizar el problema de la programación de tareas, específicamente en el entorno de manufactura Job Shop.
- Definir un marco de referencia del rescheduling apropiado, para la aplicación del algoritmo propuesto.
- Diseñar y codificar un algoritmo basado en las técnicas de computación transgénica para resolver el problema de rescheduling en un entorno Job Shop.
- Evaluar el desempeño del algoritmo propuesto en la resolución del problema de rescheduling en un entorno Job Shop.
- Comparar el desempeño del algoritmo propuesto tanto en su fase predictiva como reactiva.

3. Justificación

A lo largo de la historia, los investigadores del problema de scheduling, en pro de darle un mejor manejo y reducir la complejidad del mismo, han partido de supuestos que no representan fielmente los entornos de manufactura reales. Un ejemplo de ello es el enfoque de los modelos clásicos, los cuales no contemplan la aparición de eventos inesperados (interrupciones) que afectan los programas de producción realizados en principio. Por tal motivo, es necesario otorgarles a los programadores de tareas un algoritmo que les permita evaluar el impacto de dichas interrupciones y de ser necesario realizar los ajustes para continuar con el programa de producción ya iniciado. Ahora bien, se considera que el rescheduling (reprogramación) es el enfoque más adecuado

para formular dicha herramienta, consiguiendo un mayor acercamiento a los entornos reales en cuanto al proceso de planeación y programación.

Por otro lado, cada día en la comunidad científica van surgiendo nuevas y mejores técnicas para la solución de problemas de alta complejidad por su carácter combinatorio. Como bien se ha abordado, el problema de Job Shop se encuentra en esta categoría y en tal sentido, su complejidad supone un aumento si se visualiza desde la perspectiva del rescheduling, por lo cual es necesario utilizar técnicas con un gran desempeño computacional, mejores a las ya trabajadas. Expuestos los anteriores argumentos, se propone el diseño de un algoritmo basado en la computación transgénica para hacer posible la reprogramación de cronogramas afectados por interrupciones y por ende disminuir los efectos de las mismas en la ejecución del programa de producción.

4. Marco referencial

4.1. Marco Conceptual

4.1.1. Sistemas de manufactura.

Un sistema de manufactura es una red de procesos orientada a objetivos a través de la cual las entidades fluyen. Por lo tanto, el sistema tiene un objetivo, el cual no siempre está relacionado con la parte monetaria. También se dice que contiene procesos que pueden ser físicos (cortado, doblado, fresado, etc.), pero también pueden incluir otros pasos que apoyan el proceso de fabricación directa (ordenes de entrada, envíos, mantenimiento, etc.). Las entidades incluyen no solo las partes que van a ser manufacturadas, sino también la información que es usada para el control del sistema. El flujo de entidades describe cómo estas son procesadas. Finalmente, es importante reconocer que un sistema de manufactura es una red de partes que interactúan, donde

gestionar interacciones puede ser igual o más importante que gestionar los procesos y entidades individuales. (Hopp & Spearman, 2011)

Gershwing (1994) define el sistema de manufactura, haciendo énfasis en la producción, de la siguiente manera: Un sistema de manufactura es un grupo de máquinas, elementos de transporte, computadores, reguladores de almacenamiento y otros elementos que son usados para la manufactura, donde las personas también hacen parte de este sistema. Otros términos alternos son fábrica, sistema de producción e instalación de fabricación.

De esta manera, dentro de los sistemas de manufactura se han encontrado dos procesos que son de suprema importancia dada su influencia en la rentabilidad de fabricación de un producto, su tiempo de entrega y la utilización de los recursos, estos son: los procesos de planeación y la programación de tareas (scheduling). La planeación del proceso es la determinación sistemática de métodos mediante los cuales un producto debe fabricarse de manera económica y competitiva. El objetivo principal de la función de planificación de procesos es generar planes, que especifiquen la materia prima y componentes necesarios para producir un producto, así como los procesos y operaciones necesarios para transformar las materias primas en el producto final; mientras que la programación de tareas asigna tareas específicas a máquinas en particular, en orden de satisfacer una medida de desempeño dada. (Phanden, Jain, & Verma, 2011)

4.1.2. Programación de tareas.

La programación de tareas es un proceso de toma de decisiones con la meta de optimizar uno o más objetivos, que juega un rol importante en la mayoría de entornos de manufactura. Este tiene que ver fundamentalmente con la asignación de recursos escasos

durante el tiempo. Los recursos y tareas en una organización pueden tomar muchas formas. Los primeros pueden ser máquinas en una planta, rutas en un aeropuerto, equipos en una construcción, unidades de procesamiento en un entorno computacional, entre otros. Las tareas pueden ser operaciones en un proceso de producción, despegues y aterrizajes en un aeropuerto, etapas en un proyecto de construcción, ejecución de un programa de computador, entre otros. (Méndez, Álvarez, Caicedo, & Malaver, 2013)

Según Spearman y Hopp (2011) el fin de la programación de la producción es lograr un equilibrio entre realizar entregas a tiempo, minimizar el producto en proceso, acortar los tiempos de entrega al cliente y maximizar la utilización de los recursos, los cuales son objetivos que entran en conflicto, ya que es mucho más fácil finalizar un trabajo a tiempo si la utilización de los recursos es baja, o que los tiempos de entrega puedan esencialmente hacerse cero si se tiene un enorme inventario.

Partiendo de una visión holística, la función de programación de tareas en un sistema de producción u organización de servicio debe interactuar con otras funciones. Estas interacciones son dependientes del sistema y pueden diferir sustancialmente de una situación a otra (Pinedo, 2008). Para ello existen en la actualidad diferentes herramientas informáticas, conocidas como ERP, que nos permiten obtener información del sistema en tiempo real y que permiten mayor rapidez en la toma de decisiones. En ellas interactúan por ejemplo el cargue y descargue de órdenes de producción, actualización de inventarios de insumos y materias primas, restricciones económicas, de capacidades u otras que afecten la gestión en el piso del sistema, entre otras.

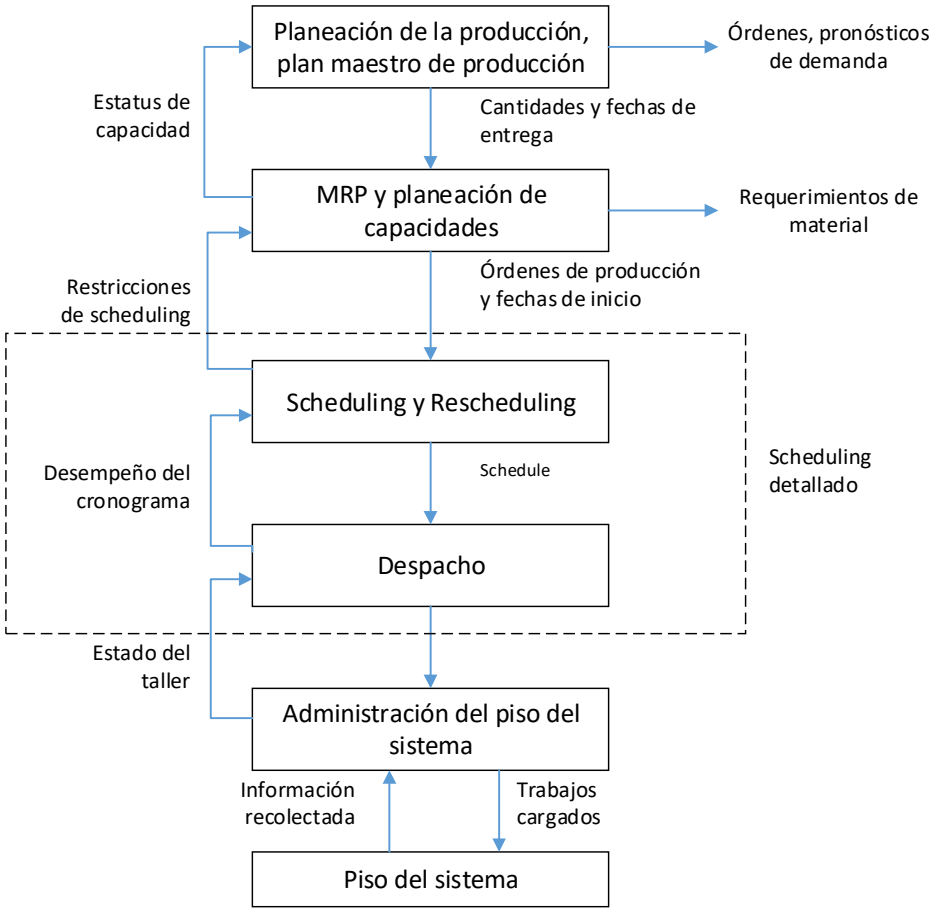


Figura 1. Diagrama de Flujo de información en un sistema de manufactura. Adaptado de (Pinedo, 2008)

Dado lo anterior, el problema de programación de tareas se resume en que un conjunto finito n trabajos $\{J_i\}_{i=1}^{n}$ debe ser procesado en un conjunto finito de m máquinas $\{M_k\}_{k=1}^{m}$. Así, un cronograma para cada trabajo es la ubicación de uno o más intervalos de tiempo para una o más máquinas (Brucker, 2007).

Un trabajo J_i consiste en un número n_i de operaciones $O_{i1},\ldots, O_{i,ni}$. Asociada a cada operación O_{ij} existe un requerimiento de procesamiento p_{ij}. Además, se denota r_i como release date, tiempo en el cual la primera operación de J_i está disponible para comenzar. Asociada a cada operación O_{ij} hay un conjunto de máquinas $\mu_{ij} \subseteq \{M_1, \ldots, M_m\}$. O_{ij} puede ser procesado en cualquiera de las máquinas en μ_{ij}.

Usualmente, todas las μ_{ij} son conjuntos de un elemento o todas las máquinas en el conjunto μ_{ij} son iguales. En el primer caso se habla de máquinas dedicadas (dedicated machines) y en el segundo de máquinas paralelas (parallel machines)

De esta manera, en un problema de planificación siempre existirán tres componentes diferenciadas: las tareas u operaciones (trabajos) que se tienen que realizar, los recursos disponibles para su realización (máquinas) y las finalidades u objetivos (función objetivo) que se desean lograr. De acuerdo con estos componentes, los problemas de planificación se dividen en dos, determinísticos y estocásticos (Espinal, Velásquez, & Restrepo, 2008). Se denomina determinístico cuando todos los datos del problema de planificación son conocidos a priori y estocástico cuando una o más variables son aleatorias.

De acuerdo con Graham (Graham, Lawler, Lenstra, & Kan, 1979) un problema de programación de tareas puede ser descrito como una tripleta $\alpha|\beta|\gamma$. El campo α describe el entorno de la máquina y contiene solo una entrada. El campo β detalla las características y restricciones de procesamiento y puede tener ninguna, una y más de una entrada. Por último el campo γ describe el objetivo a ser minimizado y a menudo contiene solo una entrada.

a. Entornos de Máquinas α.

Los posibles entornos de máquina pueden ser (Brucker, 2007; Pinedo, 2008):

- Máquina única (1): El caso de una sola máquina es el más simple de todos los entornos de máquina posibles y es un caso especial de todos los demás entornos de máquina más complicados.

- Máquinas idénticas en paralelo (Pm): Hay m máquinas idénticas en paralelo. El trabajo j requiere una sola operación y puede procesarse en cualquiera de las m máquinas o en cualquiera que pertenezca a un subconjunto dado. Si la tarea j no se puede procesar en cualquier máquina, solo en alguna que pertenezca a un subconjunto específico M_j, la entrada M_j aparece en el campo β.

- Máquinas en paralelo con diferentes velocidades (Qm): Hay m máquinas en paralelo con diferentes velocidades. La velocidad de la máquina i se denota por v_i. El tiempo que el trabajo j pasa en la máquina i es igual a p_j/v_i (asumiendo que el trabajo j recibe todo su procesamiento de la máquina i). Este entorno se conoce como máquinas uniformes. Si todas las máquinas tienen la misma velocidad, es decir, $v_i = 1$ para todo i y $p_{ij} = p_j$, entonces el entorno es idéntico al anterior (Pm).

- Máquinas no relacionadas en paralelo (Rm): Este entorno es una generalización adicional del anterior. Hay m máquinas diferentes en paralelo. La máquina i puede procesar trabajo j a velocidad v_{ij}. El tiempo que el trabajo j pasa en la máquina i es igual a p_j/v_{ij} (nuevamente, suponiendo que el trabajo j recibe todo su procesamiento de la máquina i). Si las velocidades de las máquinas son independientes de los trabajos, es decir, $v_{ij} = v_i$ para todos i y j, entonces el entorno es idéntico al anterior.

- Flow Shop (Fm): Hay m máquinas en serie. Cada trabajo debe ser procesado en cada una de las m máquinas. Todos los trabajos deben seguir la misma ruta, es decir, deben procesarse primero en la máquina 1, luego en la

máquina 2 y así sucesivamente. Una vez completado en una máquina, un trabajo se une a la cola en la siguiente máquina. Por lo general, se supone que todas las colas operan bajo la disciplina primero en llegar primero en salir (First In First Out - FIFO), es decir, un trabajo no puede "pasar" a otro mientras espera en una cola.

- Flexible Flow Shop (FFc): Un Flexible Flow Shop es una generalización del Flow Shop y los entornos de máquinas paralelas. En lugar de m máquinas en serie, hay c etapas en serie con en cada etapa varias máquinas idénticas en paralelo. Cada trabajo debe procesarse primero en la etapa 1, luego en la etapa 2 y así sucesivamente. Un escenario funciona como un banco de máquinas paralelas; en cada etapa, el trabajo j requiere el procesamiento en una sola máquina y cualquier máquina puede hacerlo. Las colas entre las distintas etapas pueden o no funcionar de acuerdo con la disciplina Primero en llegar primero en ser atendido (First Come First Served - FCFS)

- Job Shop (Jm): En un Job Shop con m máquinas, cada trabajo tiene su propia ruta predeterminada a seguir. Una distinción debe ser hecha entre Job Shops en el cual un trabajo puede visitar cada máquina más de una vez. En este último caso, el campo β = rcrc para recirculación.

- Flexible Job Shop (FJc): Un Flexible Job Shop es una generalización del Job Shop y los entornos de máquinas e paralelo. En lugar de m máquinas en serie hay c centros de trabajo donde en cada centro hay una cantidad de máquinas idénticas en paralelo. Cada trabajo tiene su propia

ruta a seguir a través de la tienda; el trabajo j requiere procesamiento en cada centro de trabajo en una sola máquina y cualquier máquina puede hacerlo.

- Open Shop (Om): Hay m máquinas. Cada trabajo debe ser procesado nuevamente en cada una de las m máquinas. Sin embargo, algunos de estos tiempos de procesamiento pueden ser cero. No hay restricciones con respecto al enrutamiento de cada trabajo a través del entorno de la máquina. El programador puede determinar una ruta para cada trabajo y diferentes trabajos pueden tener diferentes rutas.
- Mixed Shop (Xm): Es una combinación entre Job Shop y Open Shop.

b. Características de los trabajos β.

Algunos de las características más importantes son (Brucker, 2007; Pinedo, 2008):

- Preferencia (prmp): Las preferencias (preeemptions) implican que no es necesario mantener un trabajo en una máquina, una vez iniciado, hasta su finalización. El programador puede interrumpir el procesamiento de un trabajo (anticiparse) en cualquier momento y poner un trabajo diferente en la máquina. No se pierde la cantidad de procesamiento de un trabajo prepago que ya se ha recibido.
- Restricciones de precedencia (prec): Las restricciones de precedencia pueden aparecer en una sola máquina o en un entorno de máquina en paralelo, lo que requiere que uno o más trabajos deban completarse antes de que otro trabajo pueda comenzar su procesamiento. Hay

varias formas especiales de restricciones de precedencia: si cada trabajo tiene a lo sumo un predecesor y a lo sumo un sucesor, las restricciones se conocen como cadenas. Si cada trabajo tiene a lo sumo un sucesor, las restricciones se conocen como un "intree". Si cada trabajo tiene a lo sumo un predecesor, las restricciones se denominan “outtree”.

- Fecha de liberación o Release Dates (r_j): Si este símbolo aparece en el campo β, entonces la tarea j no puede iniciar su procesamiento antes de su fecha de lanzamiento r_j.

- Paradas de máquinas o Breakdowns (brkdwn): Las paradas de máquina implican que una máquina puede no estar continuamente disponible. Si hay varias máquinas idénticas en paralelo, la cantidad de máquinas disponibles en cualquier momento es una función del tiempo, es decir, m(t). A veces, las averías de la máquina también se conocen como restricciones de disponibilidad de la máquina.

- Recirculación (rcrc): La recirculación puede ocurrir en Job Shop o en Flexible Jo Shop cuando un trabajo puede visitar una máquina o centro de trabajo más de una vez.

c. Criterio de optimización γ.

Según Pinedo (2008) objetivo de minimización es siempre una función de los tiempos de finalización de los trabajos, los cuales dependen del cronograma de producción (schedule). El tiempo de finalización de un trabajo se denota como C_{ij} (completion time), d_j es la fecha de entega pactada para el trabajo j y r_j es la fecha

de liberación del trabajo (reléase date). De esta manera Méndez (2013) menciona las siguientes medidas de desempeño como las más importantes:

- Demora del trabajo (Job's Lateness): $L_j = C_j - d_j$
- Tardanza del trabajo (Job's Tardiness): $T_j = \max(0, C_j - d_j)$
- Tiempo de flujo (Flow Time): $F_j = C_j - r_j$
- Tiempo de terminación (Completion Time): C_j. Al máximo valor de tiempo de terminación C_{max} se le conoce como Makespan.
- Número de trabajos tardíos (NT)

A su vez hace énfasis en que las anteriores ecuaciones son en principio indicadores, que se convierten en medidas de desempeño en el momento en que se comparan con los objetivos del sistema.

4.1.3. Problema de programación de tareas en un ambiente Job Shop

El problema de programación de tareas en un ambiente de manufactura Job Shop, también conocido como JSSP por sus siglas en inglés (Job Shop Scheduling Problem), se define como un entorno de máquina el cual posee un conjunto *J={1,…, j,…, n}* de *n* trabajos que deben ser programados en un conjunto *M={1,…, i,…, m}* de *m* máquinas. Cada trabajo tiene una secuencia tecnológica (cadena de operaciones en las máquinas) para ser procesado. El uso de la maquina *i* para el procesamiento del trabajo *j* se nota como la operación O_{ij} con una duración igual a t_{ij}, denominado *tiempo de procesamiento*. Adicionalmente, el problema está sujeto a las siguientes restricciones y supuestos:

- Cada máquina solo puede procesar un trabajo a la vez.

- Cada trabajo solo puede ser procesado por una maquina a la vez.
- Los trabajos deben ser procesados en cada máquina solo una vez.
- La secuencia tecnológica de cada trabajo está completamente definida.
- Los tiempos de procesamiento de todas las operaciones son conocidos.
- Las maquinas siempre están disponibles y nunca se interrumpen.

Dado esto, el problema se encuentra dentro de la clase NP-Hard, lo que significa que no hay un algoritmo polinomial conocido que lo resuelva. Esto implica que el tiempo para encontrar una solución crece exponencialmente (mucho más rápido que una función polinomial) según el tamaño del problema.(Hopp & Spearman, 2011)

En resumen, el problema consiste en determinar la secuencia de operaciones sobre las maquinas con el fin de optimizar una función objetivo, que para este trabajo en particular será minimizar el *makespan* (C_{max}). El JSSP con el makespan como objetivo puede ser formulado matemáticamente como sigue:

min C_{max}

s.a.

$$r_{kj} - r_{ij} \geq t_{ij} \qquad \forall\, O_{ij} \rightarrow O_{kj} \in A \qquad (1)$$

$$C_{max} - r_{ij} \geq t_{ij} \qquad \forall\, O_{ij} \in N \qquad (2)$$

$$r_{ij} - r_{il} \geq t_{il} \vee r_{il} - r_{ij} \geq t_{ij} \qquad \forall\, O_{il} \wedge O_{ij}, i = 1, \dots, m \qquad (3)$$

$$r_{ij} \geq 0 \qquad \forall\, O_{ij} \in N \qquad (4)$$

Donde r_{ij} es el tiempo de inicio de la operación O_{ij}, N es el conjunto de todas las operaciones O_{ij} y A es el conjunto de todas las restricciones dadas por la secuencia tecnológica de cada trabajo, tal como $O_{ij} \rightarrow O_{kj}$, la cual requiere que el trabajo j sea procesado en la maquina i antes de ser procesado por la maquina k.

Esta formulación del JSSP es denominada como *programación disyunta* dado el tercer conjunto de restricciones (Pinedo, 2008). Adicionalmente, una formulación del JSSP como un problema de *programación entera* puede apreciarse en (Cheng, Gen, & Tsujimura, 1996).

Una instancia de tres trabajos por tres máquinas (3 x 3) del JSSP se presenta como sigue:

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

Figura 2. Ejemplo de una instancia de 3 x 3. Elaboración propia.

Como se puede apreciar en la tabla anterior, la primera matriz de izquierda a derecha, presenta el orden en que se deben procesar los trabajos en cada máquina. Por ejemplo, el trabajo 1 (j=1) presenta la siguiente secuencia tecnológica: (1) $O_{31} \rightarrow$ (2) $O_{11} \rightarrow$ (3) O_{21}. Finalmente, la matriz restante contiene los tiempos de procesamiento t_{ij} de cada operación O_{ij}.

Para este problema existen varias formas de representar una solución. Una de las formas más usadas, es el *diagrama de Gantt*. Un ejemplo de esta representación obtenida para una solución de la instancia de la

Figura 2, se puede apreciar en la Figura 3.

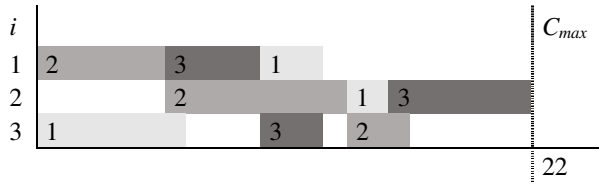


Figura 3. Diagrama de Gantt. Elaboración propia

A través del diagrama de Gantt, podemos representar lo que se conoce como *cronograma de producción*. Este representa una solución factible al problema JSSP cuando todas las restricciones son satisfechas. Ahora bien, la definición del problema del JSSP expuesta, corresponde a un modelo teórico, que, sin duda alguna, representa solo una aproximación a lo que es en la realidad un problema más complejo.

Por otra parte, los investigadores a lo largo del estudio del JSSP, en orden de reducir su complejidad y diseñar técnicas de solución más eficientes, han optado por delimitar este entorno con una gran variedad de supuestos que han terminado por crear una brecha entre los modelos teóricos y los entornos de manufactura reales, un estudio en donde se discute sobre ello puede verse en (King, 1976).

4.1.4. Métodos de Solución.

A lo largo del estudio del Job Shop Scheduling Problem se han propuesto numerosos métodos de solución. Estos pueden ser clasificados en métodos exactos, heurísticos y metaheurísticos. A continuación, se presenta una breve descripción de cada clase junto con algunos de los algoritmos más usados para resolver este problema.

Métodos exactos

Los métodos exactos consideran algoritmos que encuentran soluciones óptimas en un tiempo determinado, sin embargo, su aplicación para la mayoría de problemas de programación de tareas reales requiere un tiempo computacional extenso. Uno de los algoritmos más usados en esta clase es el *Branch and Bound* por ser de los más eficientes. Su principio fundamental es la representación en forma de árbol de todas las soluciones factibles de tal manera que por medio de procedimientos se puedan detectar las que no comparten características con la solución óptima y se puedan ir eliminando del árbol. Aunque este algoritmo al finalizar su ejecución obtiene una solución óptima, suele requerir amplios recursos computacionales además de volverse no práctico en problemas de gran complejidad.

Algoritmos heurísticos

Los algoritmos heurísticos son métodos de aproximación que encuentran soluciones cercanas al óptimo en un tiempo computacional más razonable. Estos métodos son más conocidos por el nombre de *heurísticas* y suelen dividirse en *constructivas* y de *búsqueda local*.

- *Heurísticas constructivas.*

Estos son procedimientos iterativos que construyen una solución de forma gradual, agregando operaciones a la solución inicial hasta obtener una solución completa. La ventaja de estos métodos es que son de fácil implementación y su cálculo es mucho más rápido frente a otros métodos de aproximación. Las heurísticas más usadas en esta clase son las conocidas *reglas de despacho*. En el trabajo de (Panwalkar

& Iskander, 1977) pueden apreciarse más de cien reglas utilizadas en problemas de programación de tareas.

Entre las principales heurísticas de esta clase, se encuentra algoritmo propuesto por *Gliffer* y *Thompson* (Giffler & Thompson, 1960). Este método se concentra en la generación de cronogramas activos a través de la construcción de cronogramas parciales, en los cuales se van identificando las operaciones que compiten por ser programadas en una misma máquina o estación de procesamiento. Estos conflictos se resuelven a través de las ya mencionadas reglas de despacho.

Otro método de gran relevancia para esta clase de heurísticas, es el algoritmo *Shifting Bottleneck* propuesto en (Adams, Balas, & Zawack, 1988). La idea general de esta heurística es dividir una instancia en subproblemas de una máquina. Luego se resuelve cada subproblema de forma individual y combina las soluciones obtenidas para lograr una solución de la instancia original.

- *Heurísticas de búsqueda local.*

Estos algoritmos de búsqueda trabajan con conjuntos de soluciones denominados *vecindades*. Se dice que dos cronogramas son vecinos si uno puede ser obtenido a través de la modificación del otro. Estos métodos parten de una solución inicial y en cada iteración se reemplaza una parte o toda la solución por una mejor de un conjunto de soluciones vecinas evaluadas. Para llevar a cabo estos reemplazos los algoritmos de búsqueda local realizan movimientos que conducen a la formación de nuevas soluciones en la misma vecindad. Algunos de los movimientos más comunes pueden encontrarse en (Tarantilis & Kiranoudis, 2002). La principal desventaja de estos métodos es que quedan

atrapados fácilmente en óptimos locales omitiendo de esta forma varias regiones del espacio de búsqueda.

Algoritmos metaheurísticos.

Los algoritmos metaheurísticos son métodos que exploran el espacio de búsqueda de forma más eficiente evitando quedar atrapados en óptimos locales, haciendo uso de otras heurísticas para proporcionar soluciones más robustas. Estos métodos se caracterizan por hacer uso de estrategias que direccionan la búsqueda además de la diversificación e intensificación de los espacios de solución. Algunas de las metaheurísticas más usadas para la solución del JSSP se describen como sigue:

- *Recocido Simulado.*

Este algoritmo propuesto en (Kirkpatrick, Gelatt, & Vecchi, 1983), es una técnica de búsqueda orientada al azar que se introdujo como una analogía de la simulación por computadora del proceso de recocido de un metal caliente hasta que se alcanza su estado de energía mínima. En esta metaheurística la posición de las moléculas son valores de la secuencia de operaciones por procesar mientras que el papel de la energía lo asume la función objetivo a minimizar.

- *Búsqueda Tabú.*

Esta metaheurística propuesta inicialmente en (Glover & Laguna, 1998), es un método que restringe el espacio de soluciones por medio de la memorización del historial de búsqueda. El algoritmo prohíbe los movimientos a las vecindades que tienen ciertos atributos, con el objetivo de guiar el proceso de búsqueda fuera de las

soluciones que (según la información disponible) parecen duplicar o parecerse a las soluciones alcanzadas anteriormente.

- *Algoritmos evolutivos.*

Los algoritmos evolutivos son metaheurísticas que están fundamentadas en las teorías más aceptadas sobre la evolución de los seres vivos. Aunque el término de evolución ha causado grandes debates por siglos, una de las teorías más aceptadas es la propuesta por Darwin. La cual propone que en entornos en constante cambio surgen competencias en las cuales solo las especies con mejor adaptación logran conservarse, de la misma forma en que los individuos más aptos logran reproducirse y transferir sus características a su descendencia. Estos principios han originado diversas técnicas en el contexto de la optimización, los cuales se han convertido en algoritmos muy eficientes en la búsqueda soluciones casi óptimas en problemas de elevada complejidad. Uno de los principales referentes en esta clase de metaheurísticas son los reconocidos *Algoritmos Genéticos (AG).*

El desarrollo de los algoritmos genéticos se debe en gran medida a John Holland, que desarrolló una técnica que imitaba el proceso de selección natural, la cual fue expuesta por primera vez en su libro *Adaptation in Natural and Artificial Systems* (Holland John, 1975). Los algoritmos genéticos utilizan una representación del problema de optimización, denominado *cromosoma*. Estos cromosomas constituyen una solución al problema y son evaluados por medio de una función *fitness*, la cual permite conocer su ajuste al criterio de optimización. Un conjunto de cromosomas denominado *población* evoluciona en sucesivas iteraciones haciendo uso de los operadores de *cruce* y *mutación*. El operador de cruce imita el proceso de

reproducción entre dos individuos, generando de este modo nuevos cromosomas a partir de los presentes en la población. El operador de mutación modifica esporádicamente los individuos de forma aleatoria, para evitar que en el transcurso del proceso evolutivo la población se concentre alrededor de un óptimo local. Finalmente, el algoritmo se detiene cuando este cumple algún criterio de detención.

Otro de los algoritmos evolutivos referentes en la solución del JSSP, son los denominados *Algoritmos Meméticos (AM)*. Los AM son métodos que combinan conceptos de distintas metaheurísticas como los algoritmos genéticos y otras técnicas de búsqueda local. Aunque los algoritmos meméticos pueden derivar su estructura general de los algoritmos genéticos o evolutivos, estos pueden agregar operadores que los hacen totalmente diferentes e independientes de los últimos. Uno de los principales aportes de los AM es el utilizar el máximo de conocimiento disponible sobre el problema a resolver, algo que tratan de evitar los AG. Un AM está caracterizado por un conjunto de agentes, los cuales están constituidos por una o más soluciones y por un procedimiento de mejora local. La idea central de esta técnica se fundamenta en la mejora de las soluciones en cada uno de los agentes a través de las relaciones de cooperación y competición establecidas entra dichos agentes (Moscato & Cotta, 2003b).

4.2. Marco Teórico

4.2.1. Antecedentes.

A lo largo de la historia, el problema de la programación de tareas en entornos de manufactura siempre ha expuesto múltiples retos a la comunidad científica. No solo por lo que supone gestionar de forma adecuada los recursos en los sistemas productivos sino

por la creciente complejidad que van adquiriendo los escenarios y condiciones de la industria a representar. Los estudios en el JSSP no han sido la excepción, dado que en principio los investigadores en orden de reducir dicha complejidad y diseñar técnicas de solución más eficientes, optaron por delimitar este entorno con una gran variedad de supuestos que terminaron por crear una brecha entre los modelos teóricos y los entornos de manufactura reales, como se discutió en el estudio realizado por (King, 1976).

Por otra parte, a medida que la capacidad de cálculo de los ordenadores de la época fue incrementando, nuevos modelos y técnicas eran propuestas con el objetivo de representar de forma más certera las condiciones presentes en los entornos de manufactura reales. Uno de los enfoques más reconocidos por cumplir este objetivo y sobre todo ampliamente estudiado en la literatura es el *rescheduling* o reprogramación de tareas. El *rescheduling* se ha reconocido de forma general por ser un enfoque más realista, dado que permite reprogramar las tareas en un entorno de manufactura que ha presentado algún tipo de interrupción en la ejecución del cronograma de producción.

Adicionalmente, son numerosos los estudios que se han realizado sobre el *rescheduling* desde diferentes perspectivas. Los tres enfoques más comunes son: los métodos para reparar un cronograma que ha sido interrumpido, los métodos para la creación de cronogramas robustos con respecto a las interrupciones y estudios de cómo las políticas de reprogramación afectan el desempeño de entornos de manufactura dinámicos. En el caso particular del presente trabajo, se ha centrado la atención en la literatura relacionada con los métodos de programación y reprogramación de tareas en entornos de manufactura Job Shop, problema que se ha caracterizado con diferentes nombres como el *Job Shop Rescheduling Problem* (Fang, Ross, & Corne, 1993b; H. Li,

Li, Li, & Hu, 2000; Y.-C. E. Li & Shaw, 1998; Moratori, Petrovic, & Vázquez, 2008), *Dynamic Job Shop Scheduling Problem* (Kundakc\i & Kulak, 2016; Zhang, Gao, & Li, 2013a), entre otros (Gao, Ding, & Zhang, 2009; Salido, Escamilla, Barber, & Giret, 2017; Zhang, Gao, & Li, 2013b).

De igual forma, en la literatura se han propuesto diversas técnicas para atacar el problema expuesto. La mayoría de ellas han caracterizado el entorno Job Shop bajo diferentes condiciones. Las principales variaciones se han presentado en el tipo de técnica utilizada o algoritmo, las medidas de desempeño y los factores de *rescheduling* o interrupciones contempladas. A continuación, se presenta una selección bibliográfica de los antecedentes más relevantes en el estudio que plantea el presente trabajo:

Tabla 1. Antecedentes.

Ref.	Algoritmo	Medida de desempeño	Factores	Aporte
(Subramaniam & Raheja, 2003)	Heurística AOR modificada	Eficiencia y estabilidad	Fallo de máquina, variación en el tiempo de procesamiento, arribo de un nuevo trabajo, trabajo urgente, entre otras.	Proponen una técnica basada en la heurística AOR (Affected Operations Rescheduling) la cual sirve para reparar cronogramas de las interrupciones más comunes en un entorno Job Shop. Proponen acciones básicas de reparación para manejar una amplia gama de interrupciones.
(Rangsaritratsamee, Ferrell Jr, & Kurz, 2004)	Algoritmo de búsqueda local genético	Eficiencia y estabilidad	Arribo de nuevos trabajos	Se propone una metodología de reprogramación que usa una medida de desempeño multiobjetivo la cual contempla eficiencia y estabilidad. La metodología fue probada en un entorno Job Shop simulado, demostrando obtener cronogramas más estables conservando la medida de eficiencia.

Ref.	Algoritmo	Medida de desempeño	Factores	Aporte
(Zou & Li, 2006)	Algoritmo genético	Makespan	- Fallo de máquina - Trabajo urgente	Se propone un modelo de reprogramación integrándolo con un sistema de información empresarial. Adicionalmente, se presenta un sistema de simulación de eventos en el entorno de Job Shop.
(Gao et al., 2009)	Algoritmo genético + Colonia de hormigas	Menor tiempo de terminación, costo mínimo, máxima taza de utilización y mínimo grado de desviación	-Eventos asociados a los trabajos -Eventos asociados a la máquina -Eventos al orden del proceso de producción -Otros eventos	Se propone una estrategia de rescheduling de tres fases. La primera se genera un pre-schedule, la segunda lo monitorea y la tercera realiza rescheduling. En la fase de monitoreo se ven las diferencias existentes entre la fase de pre-scheduling y la ejecución actual.
(Hasan, Sarker, & Essam, 2011)	Algoritmo genético + Shifted Gap Reduction	Makespan	- Maquina no disponible - Fallo de máquina	Se propone una técnica que integra un algoritmo genético y una nueva heurística (SGR). El algoritmo puede ser extendido a otras interrupciones como llegada de un nuevo trabajo, cambios de fechas de vencimiento, inclusión o eliminación de máquinas.
(Moratori, Petrovic, & Vazquez-Rodriguez, 2012)	Heurística Match-Up + Algoritmo genético	Desempeño (grados de satisfacción) y Estabilidad	- Llega de un nuevo trabajo - Trabajo urgente	Proponen un algoritmo que integra el enfoque de Match-up para la programación de trabajos que llegan a una empresa de impresión. La función que se optimiza es la suma de grados de satisfacción del makespan y la estabilidad.
(Sarker, Omar, Hasan, & Essam, 2013)	Algoritmo evolutivo híbrido	Makespan	- Mantenimiento de máquina - Fallo de máquina	Proponen un algoritmo evolutivo hibrido que proporciona mejores soluciones para el JSSP que otros algoritmos existentes. El algoritmo propuesto es puesto a prueba en una serie de escenarios donde las maquinas fallan y en el proceso de reprogramación demuestran ser más eficientes que las practicas existentes.

Ref.	Algoritmo	Medida de desempeño	Factores	Aporte
(Lu & Romanowski, 2013)	Reglas de despacho	Makespan	- Arribo de nuevo trabajo	Presentan un modelo de reprogramación con base de funciones multi-contextuales las cuales describen el tiempo de inactividad de una máquina y el tiempo de espera de un trabajo. Se hace uso de 11 reglas de despacho básicas para generar reglas compuestas para la generación de los cronogramas de producción.
(Zhang et al., 2013b)	Algoritmo genético + Búsqueda Tabú	- Mean flow time - Max flow time - Mean tardiness - Max tardiness - Número de trabajos retrasados	- Arribo de nuevo trabajo - Fallo de máquina	Se proponen un algoritmo evolutivo hibrido junto con simulador de interrupciones para validarlo. El algoritmo de reprogramación propuesta demuestra un desempeño superior bajo cinco medidas de desempeño diferentes.
(Zhang et al., 2013a)	Algoritmo genético + Búsqueda Tabú	Eficiencia y estabilidad	- Arribo de nuevo trabajo - Fallo de máquina	Se presenta un algoritmo hibrido que integra un simulador para resolver el problema del Job Shop dinámico. El algoritmo demostró un mejor desempeño que las reglas de despacho convencionales, aunque este requiera un tiempo de cómputo mayor que las últimas.
(Niehues, Blum, Teschemacher, & Reinhart, 2018)	Algoritmo genético	Función que integra los costos de: alistamiento, demora, transporte, máquina y ocio.	- Desviación entre lo planeado y lo real - Reducción de la capacidad disponible - Incremento de la capacidad de demanda - Reducción de la capacidad de demanda	Presentan un nuevo enfoque de un sistema de control Job Shop basado en una permanente adaptación del plan de producción a la situación actual haciendo uso de un algoritmo genético para llevar a cabo la reprogramación de operaciones.

Ref.	Algoritmo	Medida de desempeño	Factores	Aporte
(Kundakc\i & Kulak, 2016)	Algoritmo genético híbrido	Makespan	- Fallo de máquina - Arribo de nuevo trabajo - Cambio en los tiempos de procesamiento	Proponen un algoritmo genético que integra varias heurísticas para proveer soluciones rápidas y eficientes a problemas de grandes dimensiones en entornos Job Shop dinámicos. Adicionalmente, suministran una serie instancias de prueba que pueden ser útiles para la comparación de trabajos futuros en este campo.
(Salido et al., 2017)	Heurística Match-Up + Algoritmo memético	Makespan y consumo de energía	- Interrupción en máquina	Se proponen dos técnicas para manejar la reprogramación sobre una versión extendida del JSSP que involucra la velocidad en el desempeño de las maquinas lo que infiere en el consumo de energía. Una nueva técnica de Match-Up es usada para determinar una zona de reprogramación y un cronograma factible reprogramado. El algoritmo memético es usado para encontrar un cronograma que minimice el consumo de energía entre la zona de reprogramación además manteniendo el objetivo del makespan.
(Xue, Wang, Cheng, Zeng, & Yu, 2017)	Improved Genetic Algorithm (IGA)	Makespan	Fallo de máquina, variación en el tiempo de procesamiento, arribo de un nuevo trabajo, trabajo urgente, entre otros.	Se propone una metaheurística basada en el proceso evolutivo del algoritmo genético, llamada IGA. Este método hace una reprogramación total de tareas cuando se presenta una interrupción, en un tiempo computacional más bajo que el de un algoritmo genético.

Ref.	Algoritmo	Medida de desempeño	Factores	Aporte
(Ali, Telmoudi, & Gattoufi, 2018)	Algoritmo genético + Búsqueda Tabú	Makespan	Llegadas de nuevos trabajos y cambios en los tiempos de procesamiento	Luego de presentada una interrupción, se realiza el rescheduling mediante el algoritmo genético, la solución que se obtiene se entiende como buena, pero se usa el algoritmo de búsqueda local Tabu Search para mejorar la solución teniendo en cuenta el vecindario de la solución obtenida. De este modo se realiza iterativamente una mejora en el proceso hasta que el criterio de parada es cumplido.

Nota: Elaboración propia

4.2.2. Rescheduling

La definición del JSSP expuesta en el marco conceptual, constituye un modelo de secuenciación determinista y estático, dado que todos sus parámetros y variables son conocidas y no cambian en el tiempo. Es decir, este modelo solo representa una aproximación de lo que podría ser un verdadero entorno de manufactura, ya que como es sabido, la industria y la mayoría de escenarios de la vida real encierran variables que son dinámicas o tienen asociado un grado de incertidumbre.

En este sentido, cuando el cronograma de producción es generado, tanto los supervisores como los operarios tratan de seguirlo lo más cerca posible. El problema ocurre cuando surgen eventos inesperados que alteran la ejecución del cronograma de producción actual, ocasionando que este se vuelva casi obsoleto. Dichos eventos suelen llamarse *Factores de reprogramación* o *interrupciones* y necesitan ser tratadas de forma rápida y apropiada para poder minimizar los efectos que puedan deteriorar el desempeño del sistema productivo. Es aquí donde la *reprogramación* se convierte en una

herramienta indicada para manejar la interrupción presentada y poder terminar los trabajos programados.

El *rescheduling* o *reprogramación* se define entonces como el proceso de actualizar un cronograma de producción existente en respuesta a interrupciones o cambios (Vieira, Herrmann, & Lin, 2003a). Aunque la mayoría de estudios comparten esta definición, la reprogramación de tareas en entornos de manufactura también ha sido abordada con otros nombres como *dynamic scheduling* (Ouelhadj & Petrovic, 2009) y *dynamic rescheduling* (Larsen & Pranzo, 2019)

En cuanto a las *interrupciones*, estas se han caracterizado por ser aquellos eventos inesperados que afectan la ejecución de los cronogramas de producción. En la literatura estos eventos de naturaleza estocástica también se han denominado *eventos dinámicos* (Mohan, Lanka, & Rao, 2019). Subramaniam y Raheja (Subramaniam & Raheja, 2003) por su parte define *interrupción* o *factor de reprogramación*, como una desviación involuntaria de la secuencia original de operaciones planificadas en el cronograma. Esta se origina cuando el operador, el recurso o la pieza, no están disponibles durante el proceso de fabricación. En este mismo trabajo se identifican algunas de las interrupciones más conocidas en el entorno Job Shop, las cuales se indican en la siguiente tabla.

Tabla 2. Interrupciones de los entornos de manufactura

	Interrupción
1	Fallo de máquina
2	Mantenimiento de máquina
3	Ausentismo
4	Fallo de herramienta
5	Variación en el tiempo de procesamiento

6	Demora en el transporte de material
7	Variación en el desempeño de la máquina
8	Desgaste de herramienta
9	Variación en los tiempos de alistamiento
10	Llegada de un nuevo trabajo
11	Reproceso
12	Rechazo
13	Agotamiento de materia prima
14	Trabajo urgente
15	Cambio de prioridad
16	Cancelación de orden
17	Tercerización

Nota: Adaptado de (Subramaniam & Raheja, 2003)

Desde hace varias décadas, el *rescheduling* ha sido un tema de amplio interés para los investigadores y más cuando las oportunidades de lograr una plena integración con la industria siguen en aumento (Uhlmann & Frazzon, 2018). Sin embargo, el sinnúmero de estudios y contribuciones desde diferentes perspectivas, ha ocasionado que no haya como tal una línea de investigación homogénea. Ahora bien, con el fin de tener mayor claridad en cómo se ha estudiado el *rescheduling*, varios autores han propuesto *marcos de referencia* y revisiones, que orientan la compresión de las definiciones y elementos más importantes que integran este campo. Entre las contribuciones más reconocidas se encuentran el *marco de referencia* aportado por (Vieira et al., 2003a), el estudio de (Ouelhadj & Petrovic, 2009) y el *marco de referencia* de (Larsen & Pranzo, 2019). Para el desarrollo del presente trabajo, se ha considerado adoptar el *marco de referencia de reprogramación* aportado por (Vieira et al., 2003a).

Marco de referencia de reprogramación.

Para poder delimitar de forma correcta el estudio de la reprogramación en entornos de manufactura, es necesario contar con un marco de referencia apropiado, dado que su estudio puede ser tan diverso y complejo como se desee.

Según Vieira en (2003), el *marco de referencia de la reprogramación* se encuentra conformado por los siguientes elementos: *entornos, estrategias, políticas y métodos de reprogramación*. Los tres primeros son demarcados con mayor claridad por el mismo autor en el siguiente cuadro:

Tabla 3. Marco de referencia de la reprogramación

<table>
<tr><td colspan="5">Entornos de Rescheduling</td></tr>
<tr><td colspan="2">Estático
(Conjuntos finito de trabajos)</td><td colspan="3">Dinámico
(Conjunto de trabajos infinito)</td></tr>
<tr><td>Determinístico
(toda la información dada)</td><td>Estocástico
(alguna información incierta)</td><td>Sin variabilidad en el arribo
(producción cíclica)</td><td>Variabilidad en los arribos
(Flow shop)</td><td>Variabilidad den el flujo de proceso
(Job Shop)</td></tr>
<tr><td colspan="5">Estrategias de Rescheduling</td></tr>
<tr><td colspan="2">Dinámico
(sin cronograma)</td><td colspan="3">Predictivo reactivo
(generar y actualizar)</td></tr>
<tr><td rowspan="2">Reglas de despacho</td><td rowspan="2">Control teórico</td><td colspan="3">Políticas de rescheduling</td></tr>
<tr><td>Periódico</td><td>Por ocurrencia de un evento</td><td>Híbrido</td></tr>
<tr><td colspan="5">Métodos de Rescheduling</td></tr>
<tr><td colspan="2">Generación de un schedule</td><td colspan="3">Reparación de un schedule</td></tr>
<tr><td>Cronogramas nominales</td><td>Cronogramas robustos</td><td>Rescheduling por cambio a la derecha</td><td>Rescheduling parcial</td><td>Regeneración completa</td></tr>
</table>

Nota: Adaptado de (Vieira et al., 2003b)

De esta manera, cualquier trabajo que involucre la resolución de un problema de rescheduling, independiente del piso del sistema donde se encuentre, se podrá definir mediante el uso de los conceptos expuestos en la anterior tabla. A continuación se definirán cada uno de los elementos principales.

- *Entornos de reprogramación.*

El *entorno de reprogramación* hace referencia al conjunto de trabajos que se deben programar. El entorno puede ser entonces *estático* (conjunto de trabajos finito) o *dinámico* (conjunto de trabajos infinitos). A su vez el entorno estático puede ser *determinístico* (toda la información dada) o *estocástico* (alguna información es incierta). Los entornos dinámicos también se pueden clasificar según la variabilidad con la que se produce el arribo, es decir *sin variabilidad* (producción cíclica), *con variabilidad* (flow shop), además *con variabilidad en el flujo de proceso* (Job shop).

- *Estrategias de reprogramación.*

Las *estrategias de reprogramación* se refieren a como se controla la producción en dichos entornos, es decir si se generan cronogramas o no. Se han identificado dos estrategias comunes a saber: la *secuenciación dinámica* y la *predictiva-reactiva*. La primera no genera cronogramas de producción, dado que los trabajos son lanzados cuando es necesario, usando la información con que se cuenta en el momento del lanzamiento y con la ayuda de *reglas de despacho* o *modelos teóricos de control de sistemas*. La segunda estrategia se caracteriza por tener dos fases: una *predictiva* donde es generado un cronograma de producción inicial y otra *reactiva* donde el cronograma de producción es actualizado en respuesta a las interrupciones presentadas para minimizar su impacto en el desempeño del sistema. La estrategia *predictiva-reactiva* incluye a su vez tres *políticas*: *periódica*, *evento dado* e *hibrida*. Estas hacen referencia al momento en que se debe hacer la reprogramación. La periódica consiste en la realización de un cronograma de producción cada cierto intervalo de tiempo, lo cual puede ser beneficioso para talleres de producción en los cuales no haya una

actualización de la información del sistema en línea (on-line). La política de evento dado se basa en acontecimientos, que comúnmente se conocen como interrupciones o factores de reprogramación, los cuales obligan a que el piso del sistema detenga la ejecución del cronograma de producción establecido en la fase predictiva y se deba generar un nuevo cronograma de producción contemplando los efectos que el evento haya causado. Por último la política híbrida es una combinación de las dos políticas antes mencionadas.

- *Métodos de reprogramación.*

Los *métodos de reprogramación* describen como se generan y se actualizan los cronogramas de producción. Estos se han propuesto bajo dos enfoques: la generación y reparación de cronogramas. La *generación de cronogramas* de producción puede estar a cargo de diversas técnicas ya sean analíticas, heurísticas o algoritmos más complejos. La generación de cronogramas de producción puede clasificarse en dos, la creación de *cronogramas nominales* y *robustos,* siendo los segundos aquellos que son menos sensibles a perturbaciones imprevistas en el taller de producción. Ahora bien, en la *reparación de cronogramas* de producción (actualización), se incluyen tres reconocidos enfoques: *reprogramación right-shift*, *parcial* y *completa,* los cuales se diferencian por el procedimiento que siguen para secuenciar las operaciones que faltan por procesar.

- *Medidas de desempeño.*

Adicionalmente a los elementos anteriormente abordados, las medidas de desempeño constituyen una orientación importante de los estudios de reprogramación. Estas pueden ser clasificadas en tres grupos: *medidas de eficiencia*, *estabilidad* y *costo del cronograma*. El primer grupo está conformado por las medidas basadas en tiempo

más usadas como el *makespan*, *tardanza media*, *tiempo de flujo medio* entre otras. El segundo grupo hace referencia a las que miden los cambios de un cronograma en términos de los tiempos de inicio o de la diferencia en las secuencias. Finalmente, en las de costo pueden incluirse el beneficio según el trabajo, minimización del costo total, reducción del WIP entre otras.

En la literatura han sido propuestas diversas técnicas para resolver el problema de la reprogramación en términos de las medidas de desempeño ya mencionadas. Métodos como la reprogramación *right-shift* (Smith, 1995)*, match-up* (Bean, Birge, Mittenthal, & Noon, 1991) y *Affected Operations Rescheduling (AOR)* (Abumaizar & Svestka, 1997) constituyen el grupo de heurísticas más reconocidas, por su sencilla implementación, pero limitadas por la gama de interrupciones que pueden enfrentar. Adicionalmente, estas heurísticas operan bajo el enfoque de *reprogramación parcial*, lo que implica que tratan de mantener el cronograma prestablecido tanto como sea posible después de la interrupción (R.-K. Li, Shyu, & Adiga, 1993; H.-H. Wu & Li, 1995).

Por otra parte, están las técnicas de *reprogramación completa*, en donde se busca reprogramar todas las operaciones restantes luego del punto de interrupción (Church & Uzsoy, 1992; Vieira, Herrmann, & Lin, 2000). Bajo este enfoque han sido implementados diversos y sofisticados algoritmos, como por ejemplo *algoritmos genéticos* (C. Bierwirth & Mattfeld, 1999; Fang, Ross, & Corne, 1993a). Este enfoque se caracteriza por mejorar la eficiencia de los cronogramas, pero descuidando muchas veces medidas de estabilidad.

En este sentido, cada día se desarrollan nuevas técnicas para la solución de problemas combinatorios. El JSSP por pertenecer a esta clase, no es la excepción,

incluso si es visto desde la perspectiva de la reprogramación (JSRP), por lo cual cada vez se requiere utilizar técnicas que puedan generar cronogramas de mayor calidad, con uso racional del recurso computacional. Acorde a esto, en el siguiente subíndice se expone un algoritmo que ha demostrado dichas prestaciones, al aplicarse en otros problemas de esta naturaleza.

4.2.3. Algoritmo Transgénico.

En el contexto de la *computación evolutiva*, reconocida por ser el estudio de los fundamentos y aplicaciones de ciertas técnicas heurísticas basadas en los principios de la evolución natural (Tomassini, 1995), encontramos que son muchos los algoritmos desarrollados para enfrentar problemas de alta envergadura como el que se discute en el presente trabajo.

Desde la aparición de los *algoritmos genéticos*, con la publicación *"Adaptation in Natural and Artificial Systems"* (Holland & others, 1992), no se han dejado de crear técnicas basadas en esta metáfora. Muchas de estas han heredado la estructura básica de los algoritmos genéticos, para combinarlos con otras técnicas de *búsqueda local* y dar origen a los ya conocidos *algoritmos genéticos/evolutivos híbridos* (He & Mort, 2000; Pinedo, 2008), algoritmos *genéticos lamarckianos* (Morris et al., 1998) o en general *algoritmos meméticos* (Krasnogor & Smith, 2005; Moscato & Cotta, 2003a).

La denominación de los algoritmos meméticos, proviene del término inglés *meme*, acuñado por R. Dawkins como el análogo del *gen* en el contexto de la evolución cultural. Plotkin (Plotkin, 1995) define a los memes como unidades de información que es codificada y transmitida por medios no genéticos. Estas definiciones resultan ser de

gran importancia para el enfoque que se presenta a continuación, del cual se fundamenta el algoritmo transgénico.

Computación transgénica.

La *Computación Transgénica (CT)*, es una metáfora que se basa en la utilización de información memética (memes) y en el empleo de los flujos extra e intracelulares para planificar y ejecutar manipulaciones genéticas en el contexto de los algoritmos evolutivos (Marco César Goldbarg & Gouvêa, 2001). Goldbarg (2002) se refiere a ésta como una metaheurística que utiliza la infiltración planificada de la información como una herramienta para el mejoramiento de la información genética, a través de agentes transgénicos, que a su vez permiten el enriquecimiento del entorno en el cual se desenvuelven al incluir el "ruido" en el ciclo evolutivo. Visto desde un entorno natural intracelular, el ruido proviene de las interferencias ambientales aleatorias, que al ser traducido a un contexto de computación evolutiva, es la diversidad de la información adjunta a procesos de manipulación.

Así, la computación transgénica basa su propuesta en tres ideas principales (M. Goldbarg et al., 2002a):

- Infiltrar, de forma planificada, la información en el proceso evolutivo con el fin de informarla y orientarla.
- Planificar el proceso de infiltración de información y construcción de agentes a través del uso del paradigma epigenético
- Utilizar la recombinación procariota como herramienta complementaria para la preservación de la información genética.

La recombinación procariota es el proceso de generación de nuevas combinaciones genéticas en un individuo, que se puede dar por transformación, conjugación y transducción. La transformación es un modo común de transferencia horizontal de genes en el que el material genético extraño se transfiere a una célula, lo que resulta en una alteración genética. La transducción es un método en el que el transporte de ADN entre organismos implica la mediación de virus. Finalmente, la conjugación es el método donde se produce la transferencia de ADN entre las células bacterianas que están en contacto físico (E.F.G. Goldbarg, Goldbarg, & Bagi, 2007).

Para que la recombinación procariota se dé, es necesario un vehículo de intercambio genético, el cual es el plásmido. Los plásmidos son partículas genéticas móviles, anillos de ADN que pueden intercambiarse entre ciertas células, por lo tanto, haciendo la analogía con la computación evolutiva, el plásmido es la cadena de información que se transmite a los cromosomas de los individuos determinados o en otras palabras los denominados memes.

A continuación se muestra un diagrama en el cual se puede evidenciar el ciclo evolutivo de la computación transgénica:

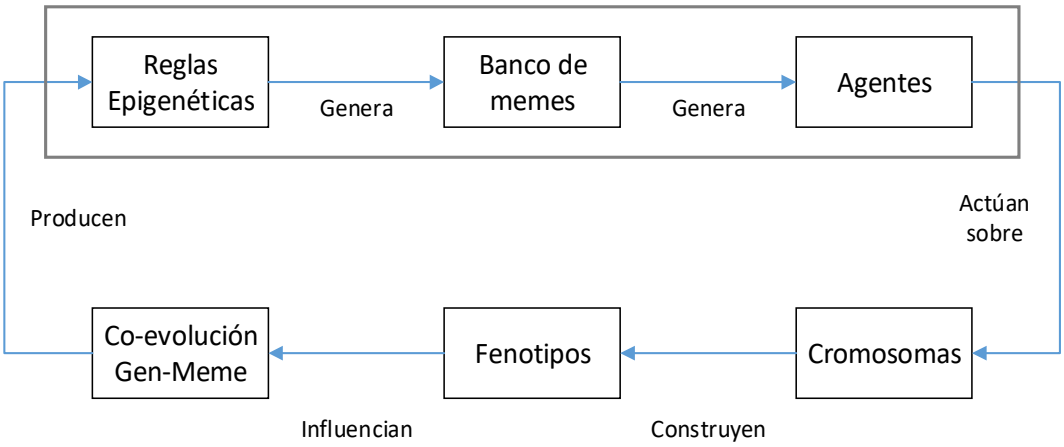


Figura 4. Flujograma del ciclo de evolución de la computación transgénica. Adaptado de (M. Goldbarg et al., 2002a).

En resumen, la CT aporta al contexto de las técnicas de computación evolutiva, los hechos de utilizar información exógena y endógena para inferir en los procesos de formación y modificación de individuos a una población determinada, usar el flujo intracelular como una forma operativa para llevar a cabo las manipulaciones requeridas en los individuos, explorar nuevos procesos de mejora poblacional usando agentes transgénicos, competencia entre agentes e individuos y guiar el proceso evolutivo, permitiendo la ocurrencia de saltos evolutivos (Gouvêa & Goldbarg, 2001).

Bajo este enfoque, la CT ha propuesto dos clases de algoritmos: *Algoritmo Transgénico Extra-Intracelular (ATEI)* (M.C. Goldbarg & Gouvêa, 2001) y *Algoritmo Proto-Gen (ProtoG)* (Gouvêa & Goldbarg, 2001). Los primeros aseguran su proceso evolutivo en los paradigmas extracelulares, intracelulares y epigenéticos, mientras que los segundos solo en paradigmas intracelulares y epigenéticos. Como consecuencia, el algoritmo ProtoG no realiza la llamada reproducción sexual como base para el intercambio de información entre sus individuos (M. Goldbarg et al., 2002a).

El algoritmo propuesto que se presenta en este trabajo se encuentra basado en el algoritmo ProtoG. A continuación, se explica su composición y estructura.

Algoritmo ProtoG.

Como se ha venido abordando a través de la CT, los algoritmos transgénicos son metaheurísticas que insertan información de forma planificada para el mejoramiento de la información genética. Ella fundamenta su acción en el *paradigma intracelular* a través de agentes que manipulan el contexto genético. La información insertada a través

de dichos agentes, se crea y se controla a través del *paradigma epigenético*. Estos paradigmas definen al algoritmo *ProtoG* como tal, excluyendo así, el *paradigma extracelular*, quien utiliza la llamada reproducción sexual como base del intercambio de información entre individuos (M. Goldbarg, Goldbarg, & Quadr, 2002b).

- *Paradigma intracelular.*

En este algoritmo los cromosomas se exponen a un ataque directo de los agentes del flujo intracelular que deben competir entre sí y con las defensas de los cromosomas para concretar la transcripción de sus códigos (M. Goldbarg et al., 2002b). En el contexto computacional, dichos cromosomas hacen parte de una *población* notada como *C* de tamaño *q* donde *C={1,…, c,…, q}* con una función *f* denominada función de ajuste o *fitness*. Donde cada cromosoma *c* es un cadena de longitud entera con valor *h* que representa una solución al problema, tal que $f: c \rightarrow \Re^{+}$, $c=1,\ldots,q$, donde *f* devuelve un valor real.

Las alteraciones cromosómicas son llevadas a cabo únicamente por la intervención de los agentes transgénicos. De este modo, los agentes son la clave del proceso evolutivo, siendo la única fuente de intensificación y diversificación del proceso de búsqueda (Gouvêa & Goldbarg, 2001).

Los *agentes transgénicos* son las entidades utilizadas por el algoritmo para insertar la información en el contexto genético. Estos agentes manipulan los cromosomas de forma similar como lo hacen los *vectores intracelulares* usados en la *Ingeniería Genética*. En analogía con los términos usados por los microbiólogos, existen

cuatro tipos de agentes transgénicos: *plásmido*, *virus*, *plásmido recombinado* y *transposón* (E.F.G. Goldbarg et al., 2007).

En el contexto computacional, un agente transgénico λ, se define como un par $\lambda=(I,\Phi)$, donde I es una cadena de información (correspondiente a los memes) y Φ describe el método de manipulación del agente. $\Phi=(p_1,\ldots, p_s,\ldots, p_x)$, donde p_s, $s=1,\ldots,x$, son procedimientos que determinan el comportamiento de los agentes. En la Tabla 4 se relacionan los procedimientos que integran el método de manipulación c

En este sentido, cuando la cadena de información I de λ es un código genético y su método de manipulación utiliza los procedimientos p_1, p_2 y p_3 se dice que λ es un virus (E.F.G. Goldbarg et al., 2007). En el caso del algoritmo ProtoG, se hace uso de la *Partícula Genética Móvil (MGP)*, la cual en el contexto computacional, es un caso especial de virus donde el periodo de bloqueo (también denominado *virus lifetime*) establecido por el procedimiento p_3 es igual cero y el procedimiento de ataque p_1 solo ocurre si el fitness del cromosoma es mejorado (Gouvêa & Goldbarg, 2001).

Tabla 4. Procedimientos del método de manipulación.

Procedimiento (p_s)	**Descripción**
Ataque (p_1)	Define un criterio que establece si un cromosoma c es susceptible a la información de un agente transgénico λ. Se define la función $v(c)='verdadero'$ si c es vulnerable a λ, $v(c)='falso'$ de lo contrario
Transcripción (p_2)	Si $v(c)='verdadero'$, el procedimiento define como la información es transferida desde λ a c.
Bloqueo/Desbloqueo (p_3)	Establece un periodo de tiempo el cual la información transferida no puede ser alterada en c.
Identificación (p_4)	Identifica las posiciones en c que serán utilizadas para limitar la operación de λ.

Recombinación (p_5)	Identifica el origen y la longitud de dos o más cadenas de información uniéndolas en λ.

Nota: Tomado de (M. Goldbarg et al., 2002a)

- *Paradigma epigenético.*

Basados en la teoría evolutiva, un individuo en el transcurso del tiempo necesita adaptarse a su entorno y evolucionar. Usualmente estos procesos se asocian netamente a contextos físicos, pero no siempre es así, en realidad cualquier ser vivo está sujeto a presiones que surgen de interacciones más sutiles relacionadas con los individuos de la especie, y estos, a su vez, están inmersos en contextos aún más complejos, las sociedades, que tienen sus propias presiones evolutivas. Algunos aspectos de las presiones evolutivas que ocurren fuera del contexto genético se tratan en una disciplina llamada epigenética. (M. Goldbarg et al., 2002a)

La dificultad de considerar la epigenética en el contexto de la computación evolutiva es discutida por Chattoe en (Chattoe-Brown, 1998). Los primeros trabajos sobre este tema asociaron los memes simplemente a información obtenida fuera del flujo extracelular (M. Goldbarg et al., 2002b). En los algoritmos meméticos, según Freisleben (Merz & Freisleben, 1999) y Radcliffe & Surry (Radcliffe & Surry, 1994), la etapa epigenética se expresa principalmente a través de un mejoramiento de los cromosomas mediante procedimientos de búsqueda local. Es aquí donde la CT adopta un nuevo enfoque, definiendo los memes de acuerdo a Plotkin (Plotkin, 1995) como memorias con información almacenada, codificada y transmitida a través de medios no genéticos. De esta forma los memes se pueden utilizar para reajustar un cromosoma o bloque de genes, incluso dichos memes, podrán estar sujetos a procesos de competición y selección pertinentes al medio cultural (M. Goldbarg et al., 2002b).

Los memes son unidades de información que pueden difundirse por una cultura, como los genes se diseminan por un reservorio genético (Wright, 2001). La relación entre los genes y los memes se debe a que los genes prescriben las reglas epigenéticas, que son regularidades de la percepción sensorial y del desarrollo mental que animan y canalizan la adquisición de la cultura. La cultura ayuda a determinar qué genes prescritos sobreviven y se multiplican de una generación a la siguiente. Nuevos genes bien secuenciados alteran las reglas epigenéticas de la población. Las reglas epigenéticas alteradas cambian la dirección y la eficacia de los canales de adquisición cultural realimentando el proceso de coevolución gen vs meme (Wilson, 1999).

En el contexto computacional, estas reglas epigenéticas son simuladas por una estructura de *reglas transgénicas.* El control de la evolución de la población de cromosomas, los agentes transgénicos y la información de la base memética es hecha por tres clases de reglas (E.F.G. Goldbarg et al., 2007; M. Goldbarg et al., 2002b).

Las reglas *tipo 1* dirigen la construcción de la cadena de información *I* que es transportada por los agentes transgénicos (E.F.G. Goldbarg et al., 2007). Este tipo de reglas puede disponer de cualquier tipo de conocimiento del problema o del almacenado en la *Base Memética (MB)*, la cual constituye el banco donde se almacenan todos los memes. Los memes pueden ser conformados por bloques de construcción o procedimientos de manipulación (M. Goldbarg et al., 2002b).

Las reglas *tipo 2* definen como la información consignada en *I* es transcrita en un cromosoma, esto es el operador usado por λ. Este tipo de reglas puede evolucionar en conformidad con la resistencia mostrada por los cromosomas. Las reglas *tipo 3* están presentes en todo el proceso, definiendo que agentes son utilizados, el número de

cromosomas que son atacados en una iteración dada, el número de agentes que es creado, el criterio de parada, entre otros. (E.F.G. Goldbarg et al., 2007; M. Goldbarg et al., 2002b).

Pseudocódigo.

El algoritmo ProtoG, puede ser descompuesto en dos fases: la primera, se encarga de generar los bloques de construcción (memes) y los codifica usando los agentes transgénicos. En la segunda, los agentes compiten por transcribir su información en los cromosomas y como resultado, mejorar sus soluciones (Gouvêa & Goldbarg, 2001).

Con la implementación del algoritmo ProtoG, la población de cromosomas va mejorando su fitness, a través de la manipulación de los agentes transgénicos. Pero no solo los cromosomas evolucionan, los memes también lo hacen. Este proceso convierte a la CT en una técnica de búsqueda informada y co-evolutiva (Gouvêa & Goldbarg, 2001).

Los algoritmos transgénicos, incluyendo el ProtoG, han demostrado ser una técnica exitosa frente a otras técnicas conocidas, en la resolución de problemas de alta complejidad. Entre los problemas abordados se encuentran: *Problema de Asignación Cuadrática (QAP)* (M. Goldbarg et al., 2002b; Gouvêa & Goldbarg, 2001), *Problema del Comprador Viajero* (E.F.G. Goldbarg et al., 2007), *Problema de Coloreado Gráfico* (M.C. Goldbarg & Gouvêa, 2001) y el *Problema de Secuenciación en Flow-shop con Permutación* (ELIZABETH FERREIRA GOUVÊA Goldbarg, Goldbarg, & Costa, 2004), entre otros más (Barboza, 2005; E F G Goldbarg, Castro, & Goldbarg, 2006).

El pseudocódigo del algoritmo es como sigue (extraído y adaptado de (M. Goldbarg et al., 2002b; Gouvêa & Goldbarg, 2001)):

Inicio
Cargar una Base de Memes con un conjunto de soluciones y reglas para generar memes
Generar y evaluar una población inicial
Repetir
  Generar un agente por la competición entre los memes obtenidos de la Base de Memes
  Para cada cromosoma ***hacer***
  Si el cromosoma en sensible a la manipulación
  Inicio
    Manipularlo
    Evaluarlo
    Si el cromosoma satisface el criterio de inmunidad
      Entonces incluir sus memes en la Base de Memes
  Fin
Hasta que un criterio de parada sea satisfecho
Fin

5. Metodología

La metodología usada para el desarrollo de este trabajo constó de las siguientes fases:

Fase 1: Realizar una recopilación y abstracción de la información teórica que permita conocer las principales características y el entorno en el cual se desenvuelven la programación y la reprogramación de tareas enfocadas principalmente en un entorno de manufactura Job Shop.

Fase 2: Elaborar un estudio de los métodos de solución empleados para la resolución de los problemas NP-Hard derivados de la programación y reprogramación de tareas en un entorno Job Shop, enfocándose principalmente en los algoritmos de computación transgénica. De esta manera se podrán conocer sus características principales, funcionamiento y diferencias con los demás métodos examinados.

Fase 3: Basados en la consolidación de la información recopilada, codificar un algoritmo de computación transgénica que permita dar solución al problema de reprogramación de tareas en un ambiente de producción Job shop teniendo en cuenta las delimitaciones definidas en el desarrollo del proyecto.

5.1. Diseño Metodológico

Tabla 5. Diseño Metodológico.

Objetivo	**Actividad**	**Herramienta a emplear**
Caracterizar el problema de la programación de tareas, específicamente en el entorno de manufactura Job Shop	Definir la programación de tareas y los entornos de manufactura en los que está enmarcada	Revisión bibliográfica
	Definir el problema de programación de tareas en un ambiente Job shop	Revisión bibliográfica
Definir un marco de referencia del rescheduling apropiado, para la aplicación del algoritmo propuesto	Conocer cuáles son los entornos, las estrategias, los métodos y las medidas de desempeño usadas en los problemas de rescheduling	Revisión bibliográfica
	Definir cuáles serán las interrupciones se tendrán en cuenta en el trabajo y sus implicaciones en el rescheduling	Revisión analítica de la bibliografía
		Adquisición de conocimiento de expertos
Diseñar y codificar un algoritmo basado en las técnicas de computación transgénica para resolver el problema de rescheduling en un entorno Job Shop	Definir computación transgénica, dar a conocer sus características principales y su método de funcionamiento	Revisión bibliográfica
	Analizar la función y la implicación de cada uno de los procesos usados en algoritmo transgénico en el desarrollo del problema de programación de tareas	Revisión analítica de la bibliografía
	Construir gráficamente el modelo operacional del algoritmo que permita la solución del JSRP	Diagrama de flujo
	Definir datos necesarios para poder tratar una interrupción específica	Revisión bibliográfica
		Adquisición de conocimiento de expertos

Objetivo	**Actividad**	**Herramienta a emplear**
		Análisis e inferencia teórica
	Definir las afectaciones que tienen de cada una de las interrupciones contempladas al scheduling que ha sido generado	Análisis e inferencia teórica
	Codificar el algoritmo transgénico propuesto	Programación algorítmica - Matlab
Evaluar el desempeño del algoritmo propuesto en la resolución del problema de rescheduling en un entorno Job Shop	Descargar instancias conocidas para el JSSP	Repositorio de instancias OR Library
	Ejecutar el algoritmo en su fase predictiva, teniendo en cuenta las instancias seleccionadas	Matlab
	Diseñar escenarios para testeo del algoritmo en su fase reactiva	Experimentación con el algoritmo
	Ejecutar el algoritmo en su fase reactiva, teniendo en cuenta los escenarios definidos	Matlab
	Recopilar resultados obtenidos tanto en la fase predictiva como reactiva	Matlab
Comparar el desempeño del algoritmo propuesto tanto en su fase predictiva como reactiva.	Buscar los mejores valores conocidos de cada una de las instancias escogidas	Repositorio de resultados Optimizer.Net
	Comparar los mejores valores conocidos de las instancias escogidas, versus los resultados recopilados en la fase predictiva	Estadística descriptiva
	Crear un algoritmo genético que se encuentre enmarcado en el mismo entorno de rescheduling que el algoritmo propuesto	Revisión bibliográfica

Objetivo	Actividad	Herramienta a emplear
		Programación algorítmica - Matlab
	Ejecutar el algoritmo genético teniendo en cuenta los escenarios de rescheduling definidos	Matlab
	Comparar los resultados obtenidos del algoritmo transgénico propuesto y del algoritmo genético en la fase reactiva	Experimentación con el algoritmo
		Análisis gráfico

Nota: Elaboración propia

6. Desarrollo del proyecto

6.1. Entorno de manufactura

Como se abordó en el marco referencial, para delimitar de forma correcta el estudio de la reprogramación, es importante definir un marco de referencia apropiado. El enfoque en el cual se enmarca nuestro trabajo, es en *los métodos para reparar un cronograma que ha sido interrumpido*. Por lo cual es importante empezar por definir esas interrupciones, también llamados *factores de reprogramación* (Dhingra, Musser, & Blankenehip, 1993; DUTTA, 1990) los cuales se presentan en un punto de tiempo del cronograma inicial, conocido como *punto de interrupción*. Para este estudio contemplamos todos los factores de reprogramación identificados en el trabajo de Subramaniam en (Subramaniam & Raheja, 2003), como se muestra en la Tabla 2.

Dado que es un grupo amplio de factores de reprogramación, los hemos clasificado en cinco clases. La clasificación se realiza según las acciones (de forma general) que se tendrían que llevar a cabo, para poder reparar el cronograma interrumpido y cumplir con las restricciones que impone el factor de reprogramación presentado. A continuación se expondrá una descripción de cada interrupción, de modo que se pueda entender su funcionamiento y las limitaciones con que han sido desarrolladas para la operatividad de este algoritmo. Hay que notar que cada una de ellas requiere dos parámetros básicos que la persona que ejecuta el algoritmo debe suministrar, el primero es el punto de interrupción del cronograma y el segundo es el factor de estabilidad, este último es un factor propio del algoritmo mediante el cual se gradúa la afectación de una interrupción en el nerviosismo del sistema.

- ***Clase I.***

Cualquiera de las interrupciones contempladas en esta clase requiere que se agregue un tiempo de inactividad, el cual hace referencia a un intervalo de tiempo en el que la estación de trabajo o máquina no estará disponible para operar ningún trabajo.

- ✓ *Fallo de Máquina.*

Una máquina o estación de trabajo se ve afectada por una falla o avería, lo que obliga a que pare y no se pueda procesar ninguna operación mientras el tiempo que dure su reparación. Por lo cual la secuencia de operaciones que se estipuló en la fase predictiva se retrasará tantas unidades de tiempo como dure la reparación.

- ✓ *Mantenimiento de máquina.*

Cualquier tipo de mantenimiento que no haya sido contemplado en el cronograma de producción establecido en la fase predictiva es interpretado como una interrupción, por consiguiente merece insertar el tiempo que este dure en la máquina afectada y luego proseguir con las operaciones necesarias para culminar los trabajos.

- ✓ *Ausentismo.*

Las máquinas o estaciones de trabajo que necesitan de la operación humana para su funcionamiento se ven afectadas por este tipo de interrupción. Consiste en la ausencia del operario encargado de la operación y/o supervisión de una máquina, dentro de las horas laborales. Por lo cual, un tiempo de inactividad es generado hasta el momento en que otro operario se ocupe de la labor o llegue la persona destinada inicialmente a esta, pudiendo así retomar el curso normal del cronograma de producción.

- ✓ *Fallo de herramienta.*

Algunas herramientas son indispensables para el funcionamiento de una máquina, por ejemplo el buril en un torno, que en caso de que fallen no se puede continuar con el correcto procesamiento de los trabajos. Por ello, la franja horaria en que ésta es cambiada o reparada implica muchas veces un tiempo de inactividad que afecta el cronograma de producción, por lo cual debe ser reparado.

- ✓ *Demora en el transporte de material.*

Al tener demoras en el transporte de material se asume que este no está disponible para el procesamiento de cualquier operación en una máquina

específica, por lo cual se genera un tiempo de inactividad que afecta la máquina y retrasa la operación de todos los trabajos que estén pendientes por procesar.

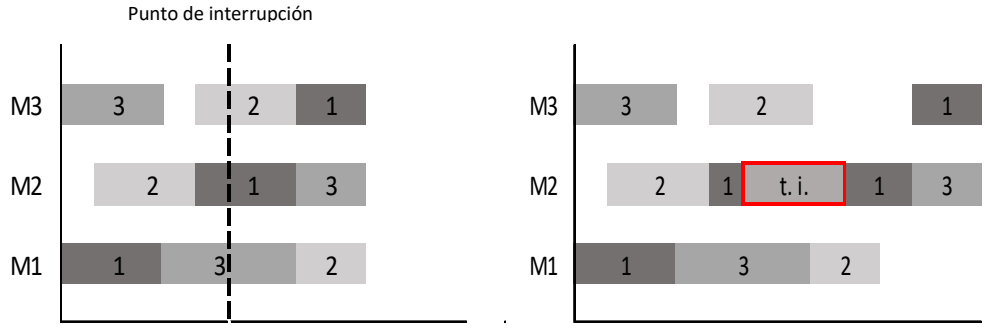
✓ *Agotamiento de materia prima.*

Al no tener una materia prima disponible se asume que toda la máquina o estación de trabajo, no puede continuar con su operación normal, sin importar el tipo de trabajo que sea. Un ejemplo de esta situación ocurre cuando no hay pintura en la estación destinada para tal fin.

Para poder dar trato a cada una de estas interrupciones, el algoritmo requiere que quien lo ejecute le suministre la siguiente información:

- Punto de interrupción
- Número de estación de trabajo afectada
- Tiempo de inactividad
- Factor de estabilidad

A continuación se ejemplifica el efecto de este tipo de interrupción en el cronograma de producción. Para este ejemplo la segunda operación del trabajo uno se ve afectada por un tiempo de inactividad.



(a) Fase Predictiva (b) Fase reactiva

Figura 5. Gantt interrupción clase I. Elaboración propia

- ***Clase II.***

Este tipo de factor de reprogramación obliga a la inserción de un tiempo de exceso y a la modificación de los tiempos de procesamiento de una o más operaciones. El primero es un intervalo de tiempo que se puede insertar dentro de una operación del trabajo afectado, esto con el fin de incrementar las unidades de tiempo invertidas en su procesamiento. Dentro de esta clase se encuentran:

✓ *Variación en el tiempo de procesamiento.*

En el transcurso de la ejecución del cronograma puede que los tiempos se vean afectados por diferentes razones, bien sea que se reduzcan o aumenten según el suceso presentado. Por lo tanto, el algoritmo propuesto permite la modificación de una o varias operaciones en diferentes máquinas simultáneamente, incluyendo aquella que se vea afectada a mitad de su operación.

✓ *Variación en el desempeño de la máquina.*

El hecho de que una nueva persona opere una máquina o que su velocidad de operación cambie voluntaria o involuntariamente, genera una interrupción al cronograma de producción, ya que los tiempos de operación pueden verse afectados notoriamente. Por ello, se permite a quien ejecuta el algoritmo modificar todos los tiempos de operación de una máquina particular.

✓ *Desgaste de una herramienta.*

El desgaste de una herramienta que es fundamental para la operación de una máquina, puede llevar a variación en los tiempos de procesamiento de

los trabajos que pasen por esta, por lo cual se genera una interrupción al cronograma y la posterior modificación de los tiempos de procesamiento de las operaciones que se vean afectadas.

- ✓ *Variación en los tiempos de alistamiento.*

Si se produce un cambio considerable en los tiempos de alistamiento de una máquina para una operación en especial, se debe afectar el tiempo de procesamiento de la operación en cuestión, ya que dentro de las delimitaciones del algoritmo se encuentra que los tiempos de alistamiento están incluidos en los tiempos de procesamiento.

Nota: Puede haber casos en los que hacer uso de un tiempo de exceso no es necesario, por lo cual se permite al usuario asignarle cero como valor.

Para poder dar trato a cada una de estas interrupciones, el algoritmo requiere que quien lo ejecute le suministre la siguiente información:

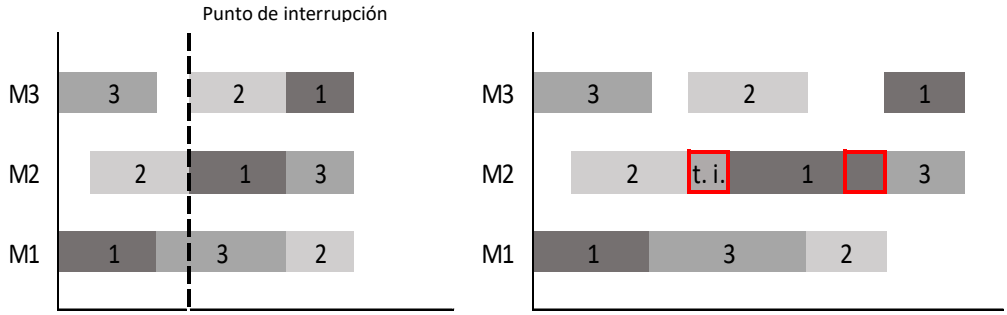
- Punto de interrupción
- Número de estación de trabajo afectada
- Tiempo de exceso
- Factor de estabilidad

Si se desean modificar los tiempos de procesamiento de una o más operaciones:

- o Número de trabajo implicado en la modificación de tiempos
- o Número de máquina implicada en la modificación de tiempos

- Nuevo tiempo de procesamiento

Habiendo expuesto lo anterior, a continuación se ejemplifica la afectación al cronograma después de presentarse una interrupción en la máquina dos, la cual consta de la inserción de un tiempo de exceso y la modificación del tiempo de procesamiento de una operación, que en este caso es la segunda del trabajo uno.



(a) Fase Predictiva (b) Fase reactiva

Figura 6. Gantt interrupción clase II. Elaboración propia.

- ***Clase III.***

En esta clase se encuentran las interrupciones que implican actualización de las secuencias tecnológicas de un trabajo y la correspondiente actualización en los tiempos de procesamiento:

✓ *Llegada de un nuevo trabajo.*

La llegada de un nuevo trabajo implica ingresar una nueva secuencia tecnológica, en la que se especifican la cantidad de operaciones del trabajo (que no puede ser mayor al número de máquinas), para ser secuenciadas junto con las demás operaciones pendientes por tratar.

✓ *Reproceso.*

El reproceso se entiende como el procedimiento en el cual las operaciones de un trabajo pueden ser procesadas de nuevo por una máquina, debido a esto se permite al que ejecuta el algoritmo, ingresar de nuevo una secuencia tecnológica, modificando a la vez los tiempos de procesamiento de cada operación. El reproceso sería entonces sobrescribir la secuencia tecnológica del trabajo dada inicialmente, descartando las operaciones que ya han sido procesadas.

✓ *Rechazo.*

Algunos trabajos pueden tener inconformidades al final o durante el proceso de fabricación, lo que genera rechazos del producto o trabajo y lleve a que este deba volver a ser procesado, bien sea desde el inicio o desde un proceso intermedio. Por ello aquel producto que es rechazado debe volver a ser secuenciado teniendo en cuenta una nueva secuencia tecnológica que puede ser igual o similar a la que tenía inicialmente y que define quien ejecuta el algoritmo.

Para poder dar trato a cada una de estas interrupciones, el algoritmo requiere que quien lo ejecute le suministre la siguiente información:

- Punto de interrupción
- Factor de estabilidad

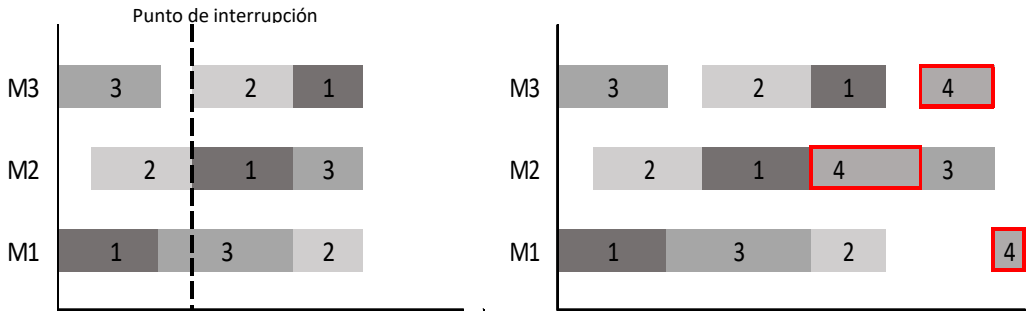
Si el trabajo es nuevo:

  - Secuencia tecnológica del nuevo trabajo
  - Tiempos de procesamiento de cada operación

Si el trabajo no es nuevo:

- Número de trabajo a modificar
- Operaciones a ser modificadas
- Secuencia tecnológica de las operaciones modificadas
- Tiempo de procesamiento de las operaciones modificadas

En los siguientes diagramas se ejemplifica el efecto de agregar un nuevo trabajo.



(a) Fase Predictiva (b) Fase reactiva

Figura 7. Gantt interrupción clase III. Elaboración propia.

- ***Clase IV.***

Esta clase contempla las interrupciones que requieren programar un trabajo con prioridad sobre los demás. Las interrupciones que la conforman son las siguientes:

✓ *Trabajo Urgente.*

La llegada de un nuevo trabajo requiere que quien ejecuta el algoritmo defina su secuencia tecnológica y los tiempos de procesamiento de cada operación. Por ser un trabajo urgente se asume que este trabajo debe

secuenciarse con prioridad sobre los demás, por ello sus operaciones se programan de tal forma que el trabajo se procese en el menor tiempo posible.

- ✓ *Cambio de prioridad.*

En la ejecución del algoritmo propuesto no se contemplan varios órdenes de prioridades, hay una única prioridad, aquel trabajo que tenga la condición de urgente se procesa primero que los demás trabajos. No es posible ordenar trabajos con prioridades secuenciales, de tal forma que uno se procese después del otro.

Para poder dar trato a cada una de estas interrupciones, el algoritmo requiere que quien lo ejecute le suministre la siguiente información:

- Punto de interrupción
- Factor de estabilidad

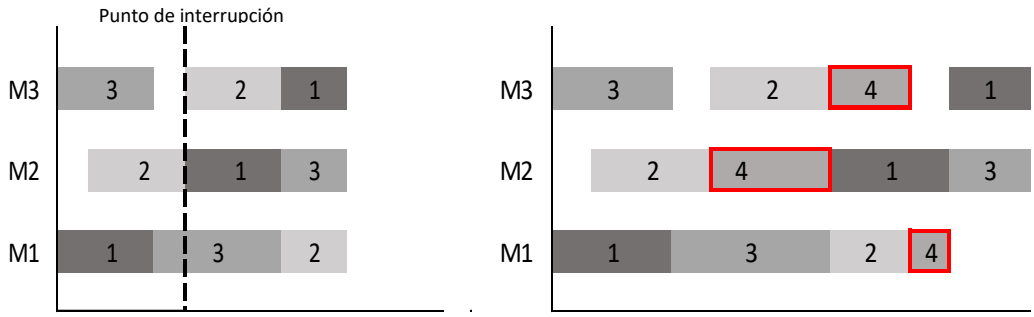
Si hay un trabajo nuevo y urgente:

- Número de operaciones del nuevo trabajo
- Máquina correspondiente a cada operación
- Tiempos de procesamiento de cada operación

Si es un cambio de prioridad:

- Número de trabajo que requiere prioridad

A continuación se puede evidenciar el efecto de este tipo de interrupción en el cronograma de producción.



(a) Fase Predictiva (b) Fase reactiva

Figura 8. Gantt interrupción clase IV. Elaboración propia

- ***Clase V.***

Este último tipo de clase contempla las interrupciones que implican una eliminación de las operaciones que originalmente estaban secuenciadas.

✓ *Cancelación de orden.*

Existen trabajos que por diversas razones no van a seguir siendo procesados, por consiguiente todas las operaciones que no hayan sido tratadas deberán ser canceladas, lo cual merece que se reprograme el cronograma inicial. De esta manera cuando se indique que un trabajo debe ser cancelado, todas sus operaciones son eliminadas del cronograma inicial y se reprograman las operaciones de los trabajos restantes.

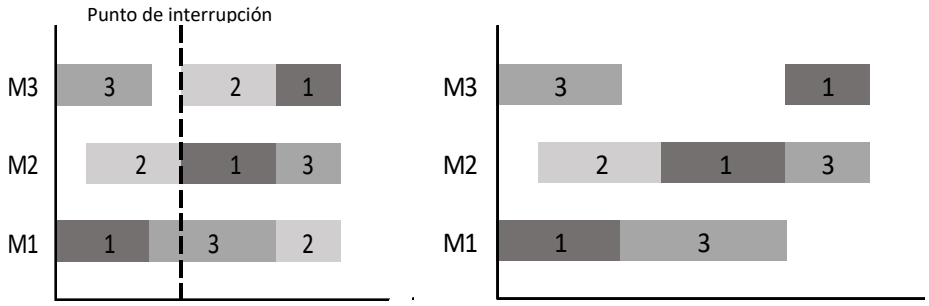
✓ *Tercerización.*

Para efectos de la ejecución del algoritmo, la tercerización de un trabajo se asume completa desde el momento (la operación) en que se indique.

Para poder dar trato a cada una de estas interrupciones, el algoritmo requiere que quien lo ejecute le suministre la siguiente información:

- Punto de interrupción
- Factor de estabilidad
- Número de trabajo implicado en tercerización o cancelación

A continuación, se puede evidenciar el efecto de este tipo de interrupción en el cronograma de producción, al afectar puntualmente al trabajo número dos.



(a) Fase Predictiva (b) Fase reactiva

Figura 9. Gantt interrupción clase V. Elaboración propia.

Teniendo en cuenta los tipos de interrupciones que son contemplados en este algoritmo y la conceptualización y desarrollo de la teoría en el marco referencial, a continuación se expone en un cuadro resumen el entorno de rescheduling en el cual se encuentra enmarcado este trabajo.

Tabla 6. Entorno de rescheduling empleado.

Entorno de reprogramación	
Dinámico	Job Shop
Existe un continuo arribo de trabajos al piso del entorno de manufactura (conjunto de trabajos infinito).	Las secuencias tecnológicas de los trabajos difieren entre ellas (variabilidad en el flujo del proceso).
Estrategia de reprogramación	
Predictiva-Reactiva	
Existe dos fases: la primera genera un cronograma de producción inicial. La segunda, se encarga de actualizar (reparar) el cronograma.	
Política de reprogramación	
Acorde al evento	

La reprogramación se realiza cuando una interrupción, cualquiera de las mencionadas anteriormente, se presenta	
Método de reprogramación	
Reparación	Completa
Se realiza una actualización del cronograma interrumpido, en respuesta al factor de reprogramación presentado.	Se reprograman todas las operaciones restantes después del punto de interrupción.
Medidas de desempeño	
Eficiencia	Estabilidad
Makespan (*E*): variación porcentual del makespan obtenido para el cronograma reparado, con respecto al cronograma inicial.	Desviación (*D*): diferencia promedio del tiempo de inicio de las operaciones del cronograma reparado, con respecto al inicial.

Nota: Elaboración propia

6.2. Algoritmo Transgénico.

Como bien es reconocido en el campo de la computación evolutiva, uno de los factores más importantes del éxito de sus algoritmos, es la representación del problema. En este sentido, el trabajo de Cheng (Cheng et al., 1996) muestra un estudio de las formas de representación utilizadas en la aplicación de algoritmos genéticos para la solución del JSSP. Apoyados en este, hemos escogido la *representación basada en operaciones*, la cual para un problema de *n* trabajos y *m* máquinas se define como una cadena (cromosoma) de *n* x *m* posiciones (genes), en donde todas las operaciones del trabajo *j* se distinguen por un mismo símbolo (número) que aparece exactamente *m* veces a lo largo de la cadena. Cada aparición del símbolo que representa al trabajo *j* corresponde a una operación en el orden de la secuencia tecnológica de dicho trabajo. Esta forma de representación tiene la ventaja de que todas las permutaciones en sus elementos dan como resultado una solución al JSSP factible, evitando así procedimientos complejos de reparación, como es el caso de otras formas de

representación. Un ejemplo de esta forma de representación se muestra a continuación, utilizando los datos de la instancia presentada en la segunda sección:

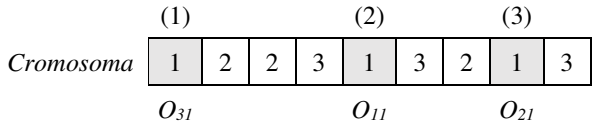


Figura 10. Forma de representación basada de en operaciones. Elaboración propia.

Como se puede visualizar en la Figura 10, cada aparición del número 1 ($j=1$) hace referencia a una operación en su secuencia tecnológica. Lo mismo puede ser replicado para los trabajos restantes. Ahora bien, esta forma de representación también resulta muy útil a la hora de realizar el diagrama de Gantt. Para su elaboración, el cromosoma es leído de izquierda a derecha, donde cada posición (gen) entrega la información de una operación O_{kj}. La operación es insertada al diagrama fijando su tiempo de inicio r_{kj} tan temprano como sea posible, es decir, verificando el tiempo de finalización de la operación precedente O_{ij}, además que no se superponga en una operación previamente insertada en el diagrama. Finalmente, la operación O_{kj} queda definida desde r_{kj} hasta $r_{kj}+t_{kj}$. Un ejemplo del anterior procedimiento se puede visualizar en (Cheng et al., 1996).

Definida la manera en que se representará la solución, continuamos con el diseño del método de reprogramación basado en la arquitectura del algoritmo ProtoG, para el entorno de manufactura previamente definido, el cual se divide en dos etapas como sigue:

- ***Etapa 1: paradigma intracelular.***

En esta etapa diseñamos los principales actores del proceso evolutivo, los agentes transgénicos. Es importante destacar que estos agentes han sido diseñados en función de las características que presenta el JSSP, definido en la segunda sección. El agente seleccionado para llevar a cabo las modificaciones genéticas es una MGP. La composición y estructura general de los agentes diseñados se muestra a continuación:

Sea $\lambda=(I_j,\Phi)$, un agente transgénico compuesto por $I_j=(g_{1j},\ldots, g_{ij},\ldots, g_{mj})$ una cadena de información (meme) con longitud m, donde el elemento g_{ij} puede representar uno de los siguientes tres elementos: una posición en el orden de procesamiento, un trabajo precedente o un trabajo adyacente, sugerido por el proceso evolutivo, el cual se toma como referencia para programar el trabajo j en la maquina i, con $i=1,\ldots,m$ y $j \in J$.

Sea $\Phi=(p_1, p_2, p_3)$, el método de manipulación del agente compuesto por los procedimientos p_s, con $s=1,2,3$. En la Tabla 7 se definen cada uno de los procedimientos.

Tabla 7. Procedimientos del método de manipulación del agente escogido.

Procedimiento (p_s)	**Definición**
Ataque (p_1)	Sea c el cromosoma antes de la modificación de su estructura y c' el cromosoma manipulado. Si $f(c')>f(c)$ entonces el cromosoma c es vulnerable $v(c)='verdadero'$, de lo contrario $v(c)='falso'$, siendo v la función que define la susceptibilidad del cromosoma.
Transcripción (p_2)	Si $v(c)='verdadero'$, entonces las operaciones del trabajo j son programadas acorde a lo establecido por la cadena I_j.
Bloqueo/Desbloqueo (p_3)	Sea y la función que define el número de iteraciones en que la información transcrita por p_2 no

	puede ser alterada, donde $y(c)=0$ para todos los casos.

Nota: Adaptado de (Elizabeth F G Goldbarg, Goldbarg, & Bagi, 2007)

Los agentes diseñados para este algoritmo han sido tres. Todos heredan la composición y estructura previamente definidas, diferenciándose entre sí solo por el tipo de información (meme) que transportan. En la siguiente sección, se define cómo el paradigma epigenético crea, almacena y controla esta información en la base memética (MB) para guiar el proceso evolutivo.

- ***Etapa 2: paradigma epigenético.***

Como se definió previamente, existen tres tipos de reglas que simulan este paradigma en el contexto computacional. Las reglas tipo 1 dirigen la construcción de la cadena de información (meme) que es transportada por el agente transgénico. Esta información es almacenada en un banco denominado Base Memética (MB). Para ilustrar un poco como se lleva a cabo este proceso, en la Figura 11 se expone un diagrama que representa el flujo de los individuos desde la población, hasta la transformación de su información en memes.

El proceso empieza con la población de cromosomas. Esta población ingresa a un procedimiento de *selección*, donde los cromosomas con mejores fitness son tomados en cuenta para continuar con el proceso (este procedimiento es controlado por un factor que se definirá más adelante).

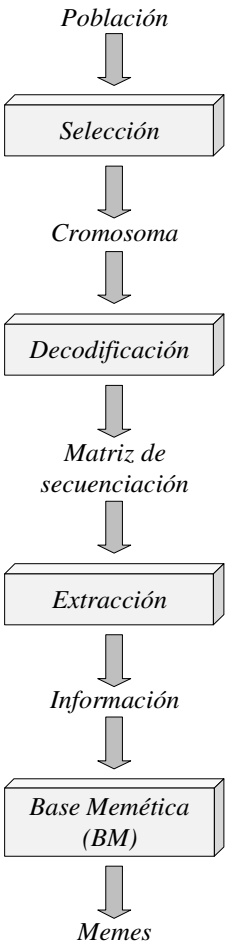


Figura 11. Proceso de creación de memes. Elaboración propia.

Una vez hay un cromosoma seleccionado, el cromosoma ingresa en un nuevo procedimiento denominado *decodificación.* Este se refiere a la conversión del cromosoma en una forma de representación conveniente, la cual es usada para extraer la información que necesita la MB en la creación de los memes. El resultado de esta decodificación es la *matriz de secuenciación*, la cual se define como un arreglo de *m* filas por *n* columnas que representa la secuencia en que se procesaran los trabajos en cada máquina. Un ejemplo de dicho procedimiento se muestra en la Figura 12.

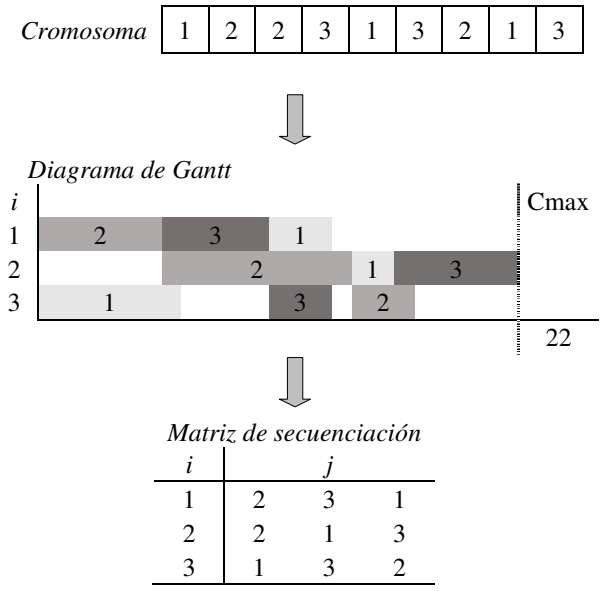


Figura 12. Procedimiento de decodificación de un cromosoma a una matriz de secuenciación. Elaboración propia.

Como se puede apreciar en la Figura 12, la matriz de secuenciación es obtenida a partir de la lectura del diagrama de Gantt asociado al cromosoma. Este arreglo resume de forma matricial, la información asociada al orden en que se van a procesar los trabajos en cada una de las máquinas.

Ahora que ya se ha mostrado la forma en que se decodifica el cromosoma, la matriz de secuenciación se convierte en el insumo base para el procedimiento de *extracción*. Este consiste en obtener *información* clave de la estructura del cromosoma dado, como por ejemplo el orden posicional en que se programan los trabajos en una maquina dada y las operaciones adyacentes o precedentes de un trabajo en un maquina dada.

Una vez se ha extraído la información, esta se almacena en la Base Memética. A continuación, se definen las matrices que la integran:

- Sea *MB* un conjunto de matrices *MB={Mp, Mr, Ma, Mb}*, donde *Mb*, *Mp*, *Mr* y *Ma,* son matrices que almacenan información específica de la población de cromosomas *C* a través del proceso evolutivo.

- *Mp* se define como la matriz de *posición*, donde el elemento Mp_{ijr} es el makespan promedio de los cromosomas que en la secuencia de operaciones de la máquina *i*, tienen programado el trabajo *j* en la posición *r*, con *i=1,…,m, j=1,…,n, r=1,…,n.*

- *Mr* se define como la matriz de *precedencia*, donde el elemento Mr_{ijl} es el makespan promedio de los cromosomas que en la secuencia de operaciones de la máquina *i*, tienen programado el trabajo *j* antes que el trabajo *l*, con *i=1,…,m, j=1,…,n, l=1,…,n.*

- *Ma* se define como la matriz de *adyacencia*, donde el elemento Ma_{ijl} es el makespan promedio de los cromosomas que en la secuencia de operaciones de la máquina *i*, tienen programado el trabajo *j* exactamente una posición antes que el trabajo *l*, con *i=1,…,m, j=1,…,n, l=1,…,n.*

Los memes son construidos a partir de los valores mínimos de las matrices *Mp, Mr* y *Ma*. Los memes obtenidos se notan como $Ip_j\ (meme_1)$, $Ir_j\ (meme_2)$ y $Ia_j\ (meme_3)$ respectivamente, donde $j \in J$ es un trabajo de referencia elegido aleatoriamente para la construcción de la cadena. A continuación, se expone un ejemplo de la construcción de cada uno de los memes:

- ***Construcción de $Ip_j\ (meme_1)$.***

  Paso 1: escoger un trabajo *j* aleatoriamente, *j=2*.

Paso 2: examinar la matriz de posición *Mp* y extraer la posición *r* donde se encuentra el valor $min_{i,j}\{Mp_{ijr}\}$, para *i=1,2,3, r=1,2,3* y *j=2*. Es importante nombrar que las posiciones de la matriz *Mp* que tienen el valor de "-", significan que en el momento de cargar la matriz no se habían registrado cromosomas que hubieran reportado en su estructura la información para calcular el Mp_{ijr} correspondiente.

			r		
i	*j*	1	2	3	*r*
1	1	-	23	22,6	
	2	22,1	25	-	1
	3	25	22	23	
2	1	-	22,4	22,9	
	2	22,5	24	-	1
	3	24	22,7	22,4	
3	1	22,4	24,4	-	
	2	-	-	22,7	3
	3	24,4	22,4	-	

Figura 13. Ejemplo de matriz de posición Mp, para una instancia 3 x 3. Elaboración propia.

Paso 3: construir el meme $Ip_j\ (meme_1)$ con las posiciones obtenidas.

	$Ip_2\ (meme_1)$		
i	1	2	3
r	1	1	3

Figura 14. Ejemplo de un meme Ipj (meme1), para una instancia 3 x 3. Elaboración propia.

- ***Construcción de $Ir_j\ (meme_2)$.***

Paso 1: escoger un trabajo *j* aleatoriamente, *j=3*.

Paso 2: examinar la matriz de precedencia *Mr* y extraer el trabajo *l* donde se encuentra el valor $min_{i,j}\{Mr_{ijl}\}$, para *i=1,2,3, l=1,2,3* y *j=3*. En este caso, el

significado de las posiciones de la matriz *Mr* con el valor de "-", es mucho más simple, dado que el trabajo *j* no puede ser precedente del trabajo *l* si *j=l*, es decir son el mismo trabajo.

		l			
i	*j*	1	2	3	*l*
1	1	-	25,3	22,1	
	2	22,8	-	22,1	
	3	22,8	25,3	-	1
2	1	-	24	22,7	
	2	22,8	-	22,7	
	3	22,8	24	-	1
3	1	-	22,8	22,3	
	2	25,2	-	22,3	
	3	25,2	22,8	-	2

Figura 15. Ejemplo de matriz de precedencia Mr, para una instancia 3 x 3. Elaboración propia.

Paso 3: construir el meme Ir_j *(meme₂)* con los trabajos obtenidos.

	Ir_3 *(meme₂)*		
i	1	2	3
l	1	1	2

Figura 16. Ejemplo de un meme Irj (meme2), para una instancia 3 x 3. Elaboración propia.

- ***Construcción de Ia_j (meme₃).***

Paso 1: escoger un trabajo *j* aleatoriamente, *j=1*.

Paso 2: examinar la matriz de adyacencia *Ma* y extraer el trabajo *l* donde se encuentra el valor $min_{i,j}\{Mr_{ijl}\}$, para *i=1,2,3, l=1,2,3* y *j=1*. Para la matriz *Ma*, las posiciones con el valor de "-" se pueden entender usando el mismo razonamiento usado para la matriz *Mr*.

		l			
i	*j*	1	2	3	*l*
1	1	-	24	22,1	3
	2	23,5	-	22	
	3	22,1	24	-	
2	1	-	24	22,1	3
	2	22,3	-	22	
	3	22,1	24	-	
3	1	-	24	22,1	3
	2	24	-	22,1	
	3	24	22,1	-	

Figura 17. Ejemplo de matriz de adyacencia Ma, para una instancia 3 x 3. Elaboración propia.

Paso 3: construir el meme $Ia_j\,(meme_3)$ con los trabajos obtenidos.

	$Ia_1\,(meme_3)$		
i	1	2	3
l	3	3	3

Figura 18. Ejemplo de un meme Iaj (meme3), para una instancia 3 x 3. Elaboración propia.

La última matriz que conforma la base memética, es *Mb* la cual se define como la matriz de *memes*, donde se encuentran almacenados los tres memes anteriormente definidos. Cada $meme_w$ tiene asociado un puntaje acumulado Sc_w que se actualiza cada vez que el agente respectivo utiliza el meme para modificar la estructura de un cromosoma. Dicho puntaje está dado por la función $Sc_w(actual)=Sc_w(previo)+[f(c)–f(c')]$, con $w=1,2,3$. Adicionalmente, los memes tienen asignado un contador definido por Co_w, el cual suma una unidad a su valor previo cada vez que $v(c)='falso'$, es decir, el cromosoma no es vulnerable al $meme_w$.

Definido el proceso de construcción de las cadenas de información I_j, es hora de presentar las reglas tipo 2, las cuales nos permiten definir la forma, en que la

información contenida en estas cadenas es transcrita en los cromosomas. Los procedimientos del tipo p_2 (transcripción), utilizados por los agentes diseñados, se diferencian levemente por el tipo de información que transportan cada uno de ellos. El procedimiento de manipulación, consiste en un intercambio posicional de genes dentro del cromosoma orientado por el meme que contiene el agente. Para ejemplificar el proceso de transcripción, se presenta un ejemplo para cada uno de los memes previamente construidos:

- ***Transcripción de Ip_j ($meme_1$).***

Paso 1: obtener la matriz de secuenciación del cromosoma al que se le transcribirá el meme.

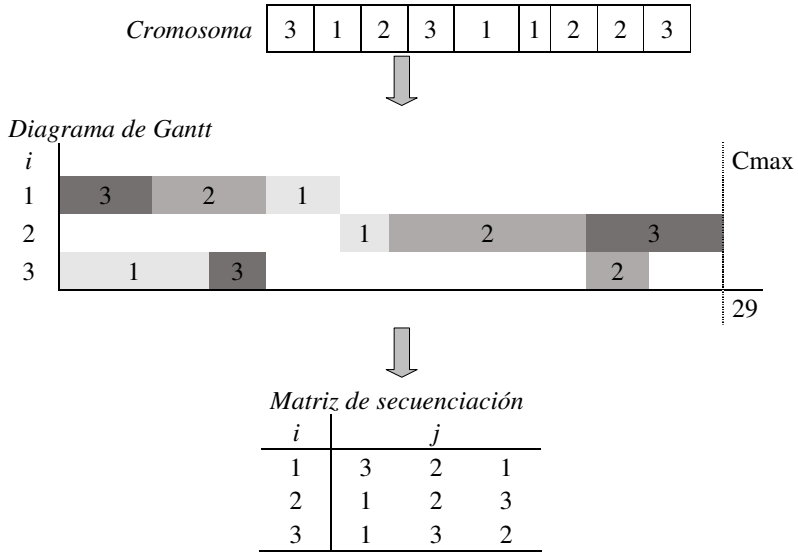


Figura 19. Procedimiento de decodificación del cromosoma al que se le transcribirá el meme. Elaboración propia.

Paso 2: el agente transgénico realiza los intercambios posicionales en la matriz de secuenciación sugeridos por el meme Ip_j ($meme_1$). En este caso el

agente buscará ubicar el trabajo *j=2* en las posiciones sugeridas por el meme Ip_2 $(meme_1)$ en cada una de las máquinas en que es procesado.

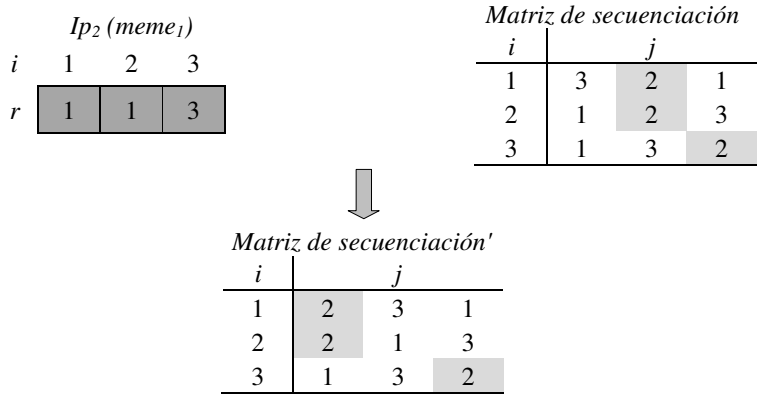


Figura 20. Ejemplo de la transcripción de un meme Ipj, para una instancia 3 x 3. Elaboración propia.

Como se puede apreciar en la figura anterior, el trabajo *j=2* ha sido desplazado a la posición *r=1* en las máquinas 1 y 2 (*i=1,2*). Cabe destacar que la posición *r* del trabajo *j=2* en la máquina *i=3* no ha sufrido cambios ya que el cromosoma contaba con esa estructura desde el principio (*r=3*). También se puede evidenciar como la nueva matriz de secuenciación cumple con lo sugerido por el meme.

Paso 3: luego de que el agente ha modificado la matriz de secuenciación, esta es codificada de nuevo en el formato de cromosoma para obtener su respectivo makespan.

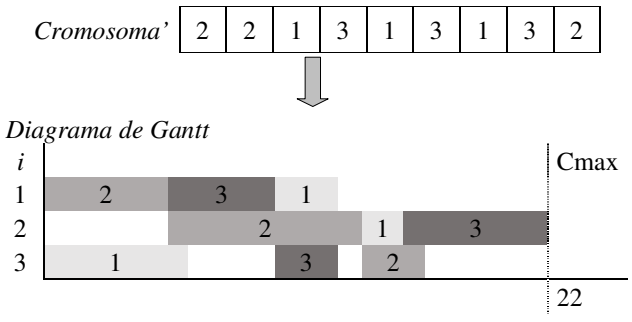


Figura 21. Evaluación del nuevo cromosoma obtenido. Elaboración propia.

Con la anterior figura podemos validar como las modificaciones realizadas por el agente dan a lugar una mejora en el makespan del cromosoma inicial, pasando de 29 a 22 unidades de tiempo. De igual manera se puede constatar que todos los cambios fueron efectuados, dado que en el diagrama de Gantt se preserva la secuenciación obtenida en el paso 2.

- ***Transcripción de Ir_j ($meme_2$).***

Paso 1: obtener la matriz de secuenciación del cromosoma al que se le transcribirá el meme.

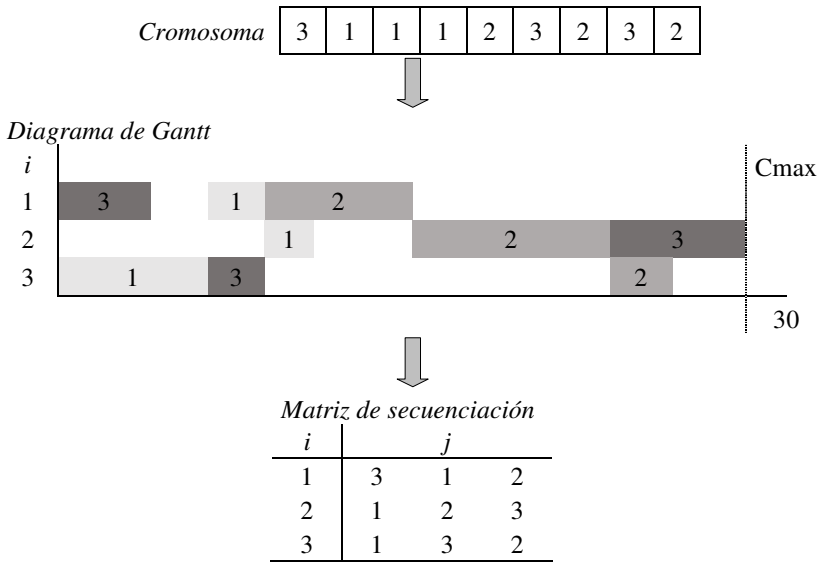


Figura 22. Procedimiento de decodificación del cromosoma al que se le transcribirá el meme. Elaboración propia.

Paso 2: el agente transgénico realiza los intercambios posicionales en la matriz de secuenciación sugeridos por el meme Ir_j ($meme_2$). En este caso el agente buscará ubicar el trabajo $j=3$ en posiciones anteriores a los trabajos sugeridos por el meme Ir_3 ($meme_2$) en cada una de las máquinas.

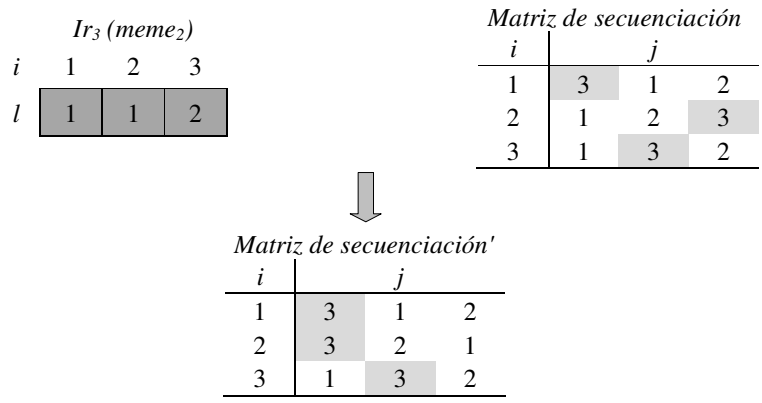


Figura 23. Ejemplo de la transcripción de un meme Irj, para una instancia 3 x 3. Elaboración propia.

Dado que el trabajo *j=3* en la máquina *i=1*, ya se encuentra programado antes que el trabajo *l=1* como lo sugiere el meme, el agente no realizará cambios en esta máquina. Ahora lo mismo sucede con la máquina *i=3*. No obstante, en la máquina *i=2* el trabajo *j=3*, es intercambiado de posición con el trabajo *l=1*, para satisfacer lo sugerido por el meme $Ir_3\ (meme_2)$.

Paso 3: luego de que el agente ha modificado la matriz de secuenciación, esta es codificada de nuevo en el formato de cromosoma para obtener su respectivo makespan.

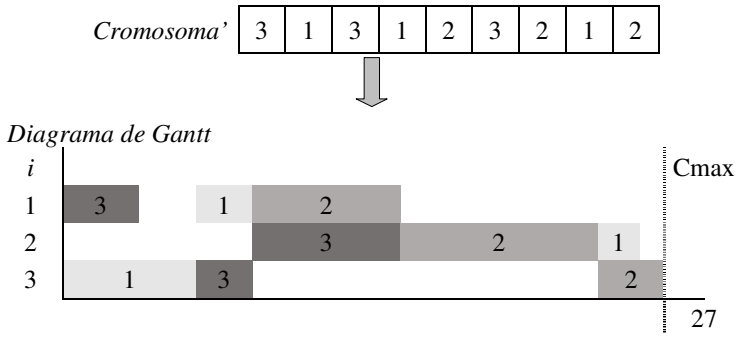


Figura 24. Evaluación del nuevo cromosoma obtenido. Elaboración propia.

Con la anterior figura podemos validar como las modificaciones realizadas por el agente dan a lugar una mejora en el makespan del cromosoma

inicial, pasando de 30 a 27 unidades de tiempo. De igual manera se puede constatar que en el diagrama de Gantt se preserva la secuenciación obtenida en el paso 2.

- ***Transcripción de Ia_j ($meme_3$).***

Paso 1: obtener la matriz de secuenciación del cromosoma al que se le transcribirá el meme.

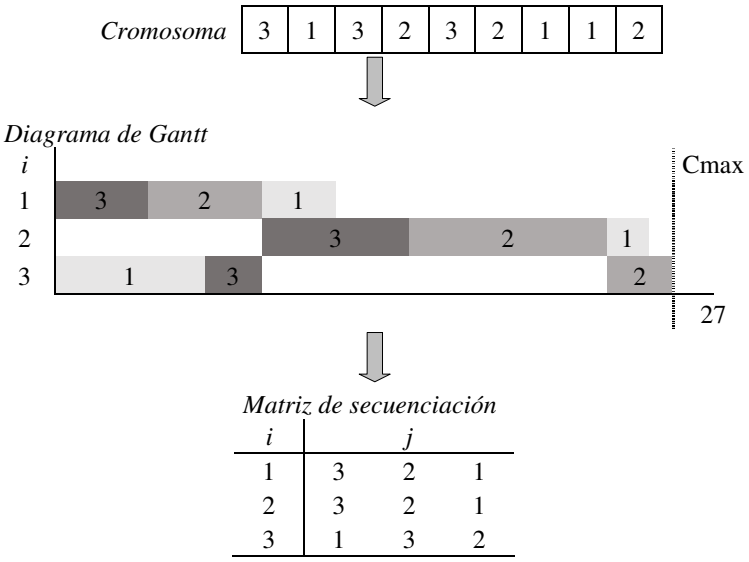


Figura 25. Procedimiento de decodificación del cromosoma al que se le transcribirá el meme. Elaboración propia.

Paso 2: el agente transgénico realiza los intercambios posicionales en la matriz de secuenciación sugeridos por el meme Ia_j ($meme_3$). En este caso el agente buscará ubicar el trabajo $j=1$ en una posición inmediatamente anterior a los trabajos sugeridos por el meme Ia_1 ($meme_3$) en cada una de las máquinas.

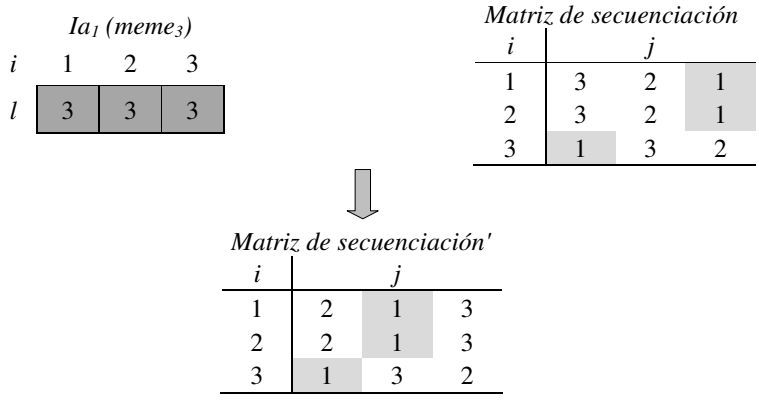


Figura 26. Ejemplo de la transcripción de un meme Iaj, para una instancia 3 x 3. Elaboración propia.

En la figura anterior podemos apreciar claramente, como el meme Ia_1 $(meme_3)$ sugiere que el trabajo $j=1$ sea ubicado una posición inmediatamente anterior que el trabajo $l=3$, tanto en la máquina 1 como en la 2 ($i=1,2$). Dichos cambios son efectuados por el agente evidenciando la nueva configuración en la matriz de secuenciación obtenida.

Paso 3: luego de que el agente ha modificado la matriz de secuenciación, esta es codificada de nuevo en el formato de cromosoma para obtener su respectivo makespan.

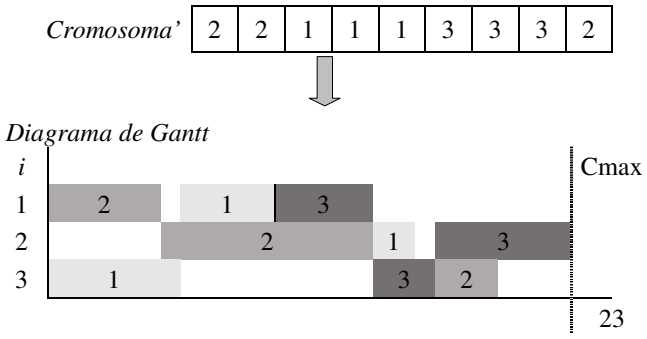


Figura 27. Evaluación del nuevo cromosoma obtenido. Elaboración propia.

Con la anterior figura podemos validar como las modificaciones realizadas por el agente dan a lugar una mejora en el makespan del cromosoma inicial, pasando de 27 a 23 unidades de tiempo. De igual manera se puede constatar que en el diagrama de Gantt se preserva la secuenciación obtenida en el paso 2.

Aunque en los ejemplos mostrados las modificaciones realizadas por los agentes generan mejoras en el makespan de los cromosomas, es importante destacar que no siempre sucede, dado que el cromosoma puede resultar no vulnerable a los cambios sugeridos por el meme que transporta el agente. En este caso el nuevo cromosoma es descartado por el agente. Ahora bien, otro hecho a tener en cuenta es que en ocasiones los cambios realizados por los agentes, pueden originar matrices de secuenciación no factibles dadas las restricciones del JSSP. En estos casos, el algoritmo hace uso de un procedimiento de reparación tratando de conservar los cambios que mantienen la factibilidad del cromosoma original.

Luego, de haber definido tanto la forma como se crean y se transcriben los memes en los cromosomas, solo resta exponer las reglas tipo 3. Dichas reglas se encuentran presentes de forma general en todo el proceso evolutivo, definiendo los parámetros y criterios que dominaran la operatividad del algoritmo. En la Tabla 8 se resumen los parámetros y variables que complementan los que ya se han definido hasta el momento y que son necesarios en la construcción del algoritmo.

Tabla 8. Parámetros y variables generales del algoritmo transgénico.

Parámetro/ Variable	**Descripción**
Tamaño de población (q)	Se refiere al número de cromosomas creados en la población inicial. Este valor permanece constante durante todo el proceso evolutivo.
Contador de no mejoría (Cn)	Es una variable que incrementa su valor en una unidad, cuando en una generación el mejor valor conocido del makespan no cambia.
Límite de no mejoría (Cn_{max})	Se refiere al valor máximo de generaciones que puede tomar la variable Cn.
Límite de inmunidad (Co_{max})	Se refiere al valor máximo que puede tomar el contador Co_w, para $w=1,2,3$.
Calidad de la información (Qi)	Es un factor entre 0 y 1 que interviene en el procedimiento de selección. Se refiere a que información es considerada en el proceso de actualización de la MB.

Nota: Elaboración propia.

Consecuentemente, se declaran los siguientes criterios y reglas:

- La creación de la población inicial de cromosomas C se realiza de forma aleatoria, donde cada cromosoma representa una solución al JSSP.

- Los cromosomas se evalúan según su ajuste o fitness. Este se define a través de la función f, donde $f(c)=C_{max}(c)$, con $c \in C$.

- Los memes son creados en primer lugar con la información de los cromosomas presentes en la población inicial. El procedimiento de selección es controlado por la siguiente expresión: si $f(c)-min(f) \leq Qi[max(f)-min(f)]$ el cromosoma es seleccionado para continuar con el procedimiento de extracción. En caso contrario es rechazado.

- El agente transgénico que ataca el cromosoma dado, se selecciona acorde al meme con mayor puntaje en la *Mb*, si existiera empate, este se rompe seleccionando uno aleatoriamente. En la primera generación es seleccionado de esta forma ya que $Sc_w=0$, para $w=1,2,3$.

- En una generación los cromosomas se van atacando uno por uno, con el agente seleccionado.

- Si el cromosoma *c* es vulnerable al ataque, el nuevo cromosoma *c'* toma su lugar en la población *C*.

- El puntaje de los memes es actualizado cada vez que un cromosoma es manipulado.

- Al finalizar cada generación se actualizan las matrices *Mp*, *Mr* y *Ma* con la información de los cromosomas manipulados. De igual forma, se verifica el contador Co_w, para $w=1,2,3$. Si este excede el Co_{max}, un nuevo meme es generado en su lugar, además de reiniciar su respectivo contador y puntaje.

- Si el contador $Cn=Cn_{max}/3$ la *MB* es reiniciada en su totalidad, sustrayendo información de la población actual y creando nuevos memes con ésta.

Como se ha visto, el algoritmo propuesto ha sido definido. Este algoritmo constituye el método con que se generan y actualizan los cronogramas de producción, ya que este se integra tanto en la etapa predictiva como en la reactiva del proceso de reprogramación. Con el fin de que lector tenga mayor facilidad en el entendimiento del proceso con el cual opera el algoritmo desarrollado, a continuación se presenta su

pseudocódigo, que cobija tanto su parte predictiva como reactiva. Cada uno de los comandos resaltados con negrita está explicado en el cuadro de convenciones que lo sucede.

Pseudocódigo algoritmo fase predictiva.

Inicio
Crear matrices de la instancia (ver Figura 2)
Capturar no. de trabajos de la instancia (*n*)
Capturar no. de máquinas de la instancia (*m*)
Para cada trabajo (*j*) **hacer:**
    Para cada máquina (*i*) **hacer:**
        Capturar el no. de la operación según las secuencias tecnológicas (O_{ij})
        Capturar el tiempo de procesamiento de la operación (t_{ij})
        Insertar valores capturados en las matrices de la instancia
    Fin Para
Fin Para
Llamar Algoritmo transgénico [cronograma inicial]=(matrices de la instancia, *vacío*, *vacío*)
Guardar cronograma de producción obtenido
Fin

Pseudocódigo algoritmo fase reactiva.

Inicio
Capturar el Id. de la interrupción
Capturar el punto de interrupción
Capturar el factor de estabilidad
Cargar matrices de la instancia
Cargar el cronograma de producción inicial
Obtener la clase a la que pertenece la interrupción
Casos (clase de la interrupción**)**
    Caso clase I:
        Capturar el no. de la máquina afectada
        Capturar el tiempo de inactividad
    Caso clase II:
        Capturar el no. de la maquina afectada
        Capturar el tiempo de exceso
        Mientras desee cambiar el tiempo de procesamiento de algún trabajo
    hacer:

Capturar el no. del trabajo
Capturar el no. de la máquina en que se procesa la operación
Capturar el nuevo tiempo de procesamiento
Fin Mientras
Caso clase III:
Si el trabajo es nuevo **entonces:**
Capturar el no. del trabajo
Capturar el no. de la máquina en que se procesa la operación
Capturar el tiempo de procesamiento
Si no:
Capturar el no. del trabajo a modificar
Para cada operación del trabajo **hacer:**
Capturar el no. de la máquina en que se procesa la operación
Capturar el nuevo tiempo de procesamiento
Fin Para
Fin Si
Caso clase IV:
Si se desea agregar un nuevo trabajo **entonces:**
Capturar el no. de operaciones
Para cada operación del trabajo **hacer:**
Capturar el no. de la maquina en donde se procesa la operación
Capturar el tiempo de procesamiento de la operación
Fin Para
Asignar al trabajo con prioridad el nuevo trabajo introducido
Si no:
Capturar el no. del trabajo con prioridad
Fin Si
Caso clase V:
Capturar el no. del trabajo a cancelar/tercerizar
Fin Casos
Obtener las operaciones ejecutadas hasta el punto de interrupción
Obtener las operaciones interrumpidas en el punto de interrupción
Obtener las operaciones a mantener en el mismo orden acorde al factor de estabilidad
Obtener las operaciones restantes por programar
Crear pre-cronograma de producción
Casos (clase de la interrupción**)**
Caso clase I:
Programar el tiempo de inactividad en el pre-cronograma
Caso clase II:
Programar el tiempo de exceso en el pre-cronograma
Caso clase IV:

Programar todas las operaciones del trabajo con prioridad en el pre-cronograma
Fin Casos
Programar operaciones interrumpidas en el punto de interrupción en el pre-cronograma
Programar operaciones a mantener en el pre-cronograma
Actualizar matrices de la instancia
Llamar Algoritmo transgénico [cronograma reparado]=(matrices de la instancia actualizadas, pre-cronograma, operaciones restantes)
Guardar cronograma de producción reparado
Fin

Algoritmo transgénico [*cronograma de producción*]=(*matrices de la instancia*, *pre-cronograma*, *operaciones restantes*)

Inicio
Cargar matrices de la instancia
Crear población inicial de cromosomas (*C*)
Obtener Makespan de cada cromosoma de la población ($C_{max}(c)$)
Obtener el mejor cromosoma de la población ($min(C_{max}(c))$)
Crear Base Memética (*MB={Mp*, *Mr*, *Ma* y *Mb}*)
Seleccionar trabajo de referencia para la creación de memes (*j*)
Obtener memes de posición, precedencia y adyacencia (*Ipj*, *Irj*, *Iaj*)
Mientras detener algoritmo = *falso* **hacer:**
Para cada cromosoma (*c*) en la población (*C*) **hacer:**
Seleccionar el meme con el mayor puntaje en *Mb*
Seleccionar el agente transgénico según el meme seleccionado ($\lambda=(I_j,\Phi)$)
Atacar al cromosoma con el agente seleccionado ($\Phi(p_1)$)
Obtener vulnerabilidad del cromosoma ($v(c)$)
Si el cromosoma es vulnerable **entonces:**
Transcribir meme en el cromosoma ($\Phi(p_2)$)
Obtener el nuevo Makespan ($f(c')$)
Si el nuevo Makespan es menor al inicial ($f(c')<f(c)$) **entonces:**
Reemplazar el cromosoma modificado (*c'*) por el inicial (c) en *C*
Si no:
Sumar una unidad al contador de no mejora del meme (Co_w)
Fin Si
Actualizar puntaje del meme en la *Mb* (Sc_w)
Actualizar matrices posición, precedencia y adyacencia (*Mp*, *Mr*, *Ma*)
Fin Si
Para cada meme en la *Mb* **hacer:**
Si el contador del meme (*Cow*) es mayor al límite (*Comax*) **entonces:**
Seleccionar trabajo de referencia para la creación de memes (*j*)

Obtener un nuevo meme
Insertar el meme en la *Mb*
Reiniciar puntaje (Sc_w) y contador (*Cow*) del meme
Fin Si
Fin Para
Si se ha alcanzado el no. de generaciones máximo **o** el contador de no mejoría es igual al límite ($Cn = Cn_{max}$) **entonces:**
Asignar detener algoritmo = *verdadero*
Fin Si
Si el mejor cromosoma de la generación es el mismo al de la generación anterior **entonces:**
Sumar una unidad al contador de no mejoría (*Cn*)
Si no:
Reiniciar el contador de no mejoría (*Cn*)
Fin Si
Si el contador de no mejora es igual a la tercera parte de su nivel máximo ($Cn=Cn_{max}/3$) **entonces:**
Reiniciar Base Memética (*MB={Mp*, *Mr*, *Ma* y *Mb}*)
Obtener Base Memética (*MB={Mp*, *Mr*, *Ma* y *Mb}*)
Obtener memes de posición, precedencia y adyacencia (*Ipj*, *Irj*, *Iaj*)

Fin Si
Obtener el mejor cromosoma de la población ($min(C_{max}(c))$)
Fin Mientras
Retornar el mejor cromosoma de la población ($min(C_{max}(c))$)
Fin

Tabla 9. Convenciones de lectura del pseudocódigo.

Comando	Descripción
Actualizar	Cambia el valor de la variable por un nuevo valor obtenido
Asignar	Establece un valor especificado a una variable
Atacar	Utiliza el procedimiento de ataque del agente transgénico contra el cromosoma
Capturar	Solicita al usuario que establezca el valor de una variable
Cargar	Importa un conjunto de datos predeterminados
Casos	Estructura de control que ejecuta una serie de sentencias de acuerdo al caso que se presente (valor de la variable)
Crear	Declara una variable dependiendo el tipo en estado *vacío*
Guardar	Escribe el valor de una variable en un archivo local
Insertar	Agrega valores capturados u obtenidos a una variable especificada
Llamar	Invoca una función suministrándole un conjunto de argumentos específicos
Mientras	Estructura de control que ejecuta una serie de sentencias mientras se cumpla una condición dada
Obtener	Calcula el valor de una variable de acuerdo a un conjunto de procedimientos
Para	Estructura de control que ejecuta una serie de sentencias un número determinado de veces
Programar	Inserta una o varias operaciones a un cronograma de producción

Comando	Descripción
Reemplazar	Sustituye el valor actual de una variable por uno nuevo obtenido
Reiniciar	Asigna el estado *vacío* o el valor cero (0) a una variable ya creada
Retornar	Devuelve una variable de interés cuando ha sido llamada una función
Seleccionar	Escoge el valor de una variable acorde a un criterio predeterminado
Si	Estructura de control que ejecuta una serie de sentencias si se cumple una condición dada
Sumar	Adiciona una cantidad al valor actual de una variable
Transcribir	Utiliza el procedimiento de transcripción del meme contra el cromosoma

Nota: Elaboración propia.

Ahora bien, para realizar la respectiva integración del algoritmo propuesto con el entorno de manufactura definido, es presentado el siguiente esquema, el cual representa la utilidad y funcionamiento del algoritmo en el entorno de manufactura.

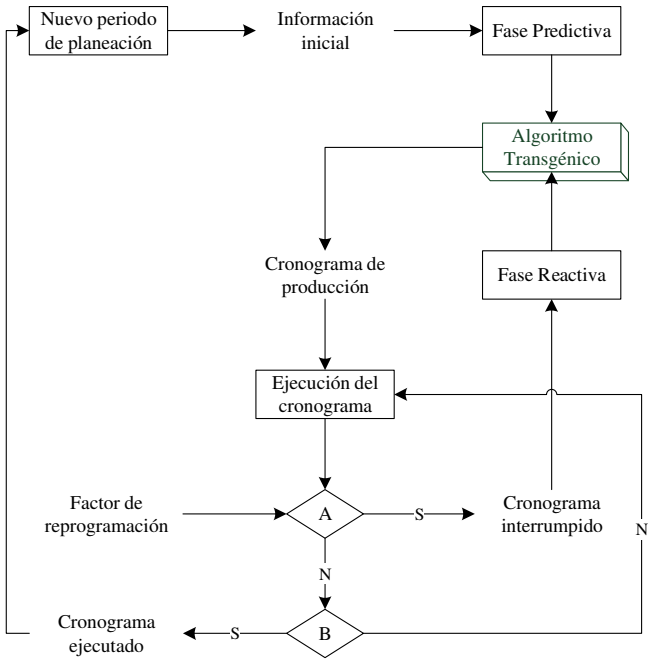


Figura 28. Integración del algoritmo transgénico en el entorno de manufactura. Elaboración propia.

Como se aprecia en la Figura 28, el algoritmo propuesto se integra al entorno de manufactura tanto en la fase predictiva como en la reactiva. Inicialmente un cronograma de producción es obtenido en la fase predictiva por el algoritmo transgénico. Luego que

comienza su ejecución, el decisor “A” define si en algún momento se presenta un factor de reprogramación. De ser así, el cronograma es interrumpido e ingresa a la fase reactiva para ser reparado por el algoritmo transgénico. De lo contrario, el decisor “B” define si el cronograma ha sido ejecutado en su totalidad. De ser así, el cronograma se da como ejecutado y empieza un nuevo periodo de planeación. De lo contario, el cronograma continúa con su ejecución.

7. Resultados

Para validar el desempeño del algoritmo propuesto, se ha dividido el estudio en dos partes: la primera contempla la experimentación del algoritmo transgénico en la fase predictiva. En la segunda parte, el algoritmo es probado en la fase reactiva.

7.1. Fase predictiva

Como se presentó en el marco referencial, esta fase de la estrategia es donde se genera un cronograma de producción inicial. El problema es entonces el ya presentado, JSSP. Para su solución han sido desarrolladas una gran variedad de técnicas (un estudio de estas puede encontrarse en (Arisha, Young, & El Baradie, 2001)). De igual manera, estas técnicas han sido probadas con instancias de comparación propuestas por varios autores. Ahora con el fin de validar el desempeño del algoritmo transgénico, se han seleccionado algunas de las expuestas por Fisher y Thompson (Fisher & Thompson, 1963), Adam, Balas y Zawack (Adams et al., 1988), Nakano y Yamada (Yamada & Nakano, 1992), Applegate y Cook (Applegate & Cook, 1991), Lawrence (Lawrence, 1984) y Taillard (Taillard, 1993).

Los resultados que se mostrarán posteriormente, fueron obtenidos con la parametrización mostrada en la Tabla 10. Dichos parámetros fueron fijados a partir del conocimiento empírico y criterio de los autores, teniendo en cuenta los resultados de varias ejecuciones del algoritmo llevadas a cabo durante su fase de desarrollo.

Tabla 10. Parametrización del algoritmo transgénico para la fase predictiva.

Parámetro	**Tamaño de instancia**		
	Pequeña	**Mediana**	**Grande**
Tamaño de población (q)	50	300	500
Límite de no mejoría (Cn_{max})	100	100	100
Límite de inmunidad (Co_{max})	5	30	50
Calidad de la información (Qi)	0.1	0.1	0.1

Nota: Elaboración propia.

Los anteriores parámetros se han definido para tres tamaños de instancia identificados: una instancia pequeña contiene entre 36 y 75 operaciones, una mediana entre 100 y 200 operaciones y una grande entre 225 y 400 operaciones.

En la Tabla 11 se relacionan los resultados obtenidos para las instancias seleccionadas. Estas instancias son evaluadas por el makespan. De este modo el campo BK en la tabla, relaciona el *mejor valor conocido* en la literatura para cada una de las instancias (la mayoría son valores óptimos). El campo BF relaciona el *mejor valor encontrado* por el algoritmo propuesto. El campo GAP relaciona la diferencia relativa del BF con respecto al BK, en términos porcentuales así: GAP=[(BF-BK)/BK]*100%. Los dos campos siguientes relacionan el promedio y la desviación estándar del makespan obtenido en cinco réplicas del algoritmo con la instancia. Finalmente, el último campo revela el tiempo de ejecución promedio en segundos. Los valores BF en negrita, revelan que el algoritmo propuesto ha encontrado el BK. El algoritmo fue

implementado en MATLAB®, estos resultados fueron obtenidos en un computador con procesador Intel® Core™ i5-8300H @4.0 Ghz y 8Gb de memoria ram.

Tabla 11. Resultados obtenidos por el algoritmo propuesto para las instancias seleccionadas.

Nombre	*n*	*m*	BK	BF	GAP	AVG	SD	RT
Instancias propuestas por Fisher y Thompson (Fisher & Thompson, 1963)								
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	1165	1173	0,7	1178,8	3,1	10
Instancias propuestas por Adam, Balas y Zawack (Adams et al., 1988)								
Abz5	10	10	1234	1242	0,6	1243,7	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
Instancias propuestas por Nakano y Yamada (Yamada & Nakano, 1992)								
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	1016	5,0	1017,8	2,5	170
Instancias propuestas por Applegate y Cook (Applegate & Cook, 1991)								
Orb01	10	10	1059	**1059**	0,0	1073,0	11,0	7
Orb02	10	10	888	894	0,7	896,6	1,0	9
Orb03	10	10	1005	**1005**	0,0	1021,3	9,1	10
Orb04	10	10	1005	1011	0,6	1019,0	7,9	8
Orb05	10	10	887	894	0,8	894,5	1,3	6
Orb06	10	10	1010	**1010**	0,0	1026,2	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
Instancias propuestas por Lawrence (Lawrence, 1984)								
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	1222	**1222**	0,0	1222,0	0,0	3
La12	20	5	1039	**1039**	0,0	1039,0	0,0	3

Nombre	*n*	*m*	BK	BF	GAP	AVG	SD	RT
La13	20	5	1150	**1150**	0,0	1150,0	0,0	2
La14	20	5	1292	**1292**	0,0	1292,0	0,0	2
La15	20	5	1207	**1207**	0,0	1207,0	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	1046	1053	0,7	1067,9	8,9	11
La22	15	10	927	935	0,9	936,0	2,1	13
La23	15	10	1032	**1032**	0,0	1032,0	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	1218	**1218**	0,0	1223,8	4,8	17
La27	20	10	1235	1249	1,1	1271,8	13,5	24
La28	20	10	1216	1227	0,9	1241,6	10,5	26
La29	20	10	1152	1201	4,3	1210,4	7,9	33
La30	20	10	1355	**1355**	0,0	1355,0	0,0	23
La31	30	10	1784	**1784**	0,0	1784,0	0,0	28
La32	30	10	1850	**1850**	0,0	1850,0	0,0	33
La33	30	10	1719	**1719**	0,0	1719,0	0,0	29
La34	30	10	1721	**1721**	0,0	1721,0	0,0	43
La35	30	10	1888	**1888**	0,0	1888,0	0,0	46
La36	15	15	1268	1297	2,3	1301,8	7,6	53
La37	15	15	1397	1436	2,8	1446,2	6,2	65
La38	15	15	1196	1225	2,4	1234,8	9,2	57
La39	15	15	1233	1251	1,5	1252,8	4,0	58
La40	15	15	1222	1243	1,7	1252,0	8,5	59
Instancias propuestas por Taillard (Taillard, 1993)								
Ta01	15	15	1231	**1231**	0,0	1233,2	3,9	47
Ta02	15	15	1244	**1244**	0,0	1245,4	3,1	49
Ta03	15	15	1218	**1218**	0,0	1221,6	5,1	56
Ta04	15	15	1175	1200	2,1	1209,4	6,3	93
Ta05	15	15	1224	1248	2,0	1265,6	12,2	112
Ta11	20	15	1357	1421	4,7	1430,6	10,8	150
Ta12	20	15	1367	1430	4,6	1441,2	6,5	184
Ta13	20	15	1342	1391	3,7	1426,4	24,1	157
Ta14	20	15	1345	1359	1,0	1375,8	15,1	93
Ta15	20	15	1339	1438	7,4	1442,0	3,5	196
Ta21	20	20	1642	**1642**	0,0	1646,6	5,5	233
Ta22	20	20	1600	1645	2,8	1656,0	7,9	265
Ta23	20	20	1557	1644	5,6	1670,4	16,4	380
Ta24	20	20	1644	1691	2,9	1708,4	14,8	320
Ta25	20	20	1595	1719	7,8	1741,2	22,9	338

Nota: Elaboración propia.

Una manera de resumir los resultados obtenidos al aplicar el algoritmo transgénico a las instancias de comparación, se muestra en la Tabla 12, ésta refleja los valores promedio obtenidos para cada configuración de *n* x *m*. En términos generales el algoritmo ha mostrado tener un buen desempeño en la fase predictiva, logrando el mejor valor conocido para el 52% de las instancias seleccionadas y valores muy cercanos para las instancias restantes. Estos valores en promedio no superan el 1,3% de diferencia para las instancias medianas y el 4,2% para las grandes. Se considera que la consistencia del algoritmo es suficiente, ya que la desviación estándar presenta valores aceptables. En cuanto al tiempo de ejecución promedio, los valores obtenidos son menores a un minuto, exceptuando las últimas tres configuraciones en el conjunto de instancias grandes.

Tabla 12. Resumen de los resultados obtenidos por el algoritmo propuesto.

Instancias pequeñas				
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
Instancias medianas				
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
Instancias grandes				
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

Nota: Elaboración propia.

Por otra parte, se puede apreciar que el desempeño del algoritmo se va deteriorando levemente, a medida que incrementa la complejidad de las instancias, es decir, entre más operaciones a procesar contengan.

Una vez que el algoritmo ha demostrado tener un buen desempeño en términos del gap promedio, desviación estándar y tiempo de ejecución para los tres conjuntos de instancias identificadas, se procede a probar su desempeño en la fase reactiva.

7.2. Etapa reactiva

Una vez el cronograma de producción inicial es obtenido a través de la etapa predictiva, en diversos momentos de su ejecución se presentarán lo que ya se ha definido como factores de reprogramación. En este sentido, se ha diseñado un conjunto de experimentos, para validar el desempeño del algoritmo comparándolo con un referente muy importante en los algoritmos evolutivos: el algoritmo genético. No hay duda de que esta técnica ha sido empleada en una amplia gama de problemas combinatorios, por lo cual es tomado como referencia para comparar el desempeño del algoritmo propuesto. Una aplicación de algoritmos genéticos a la programación y reprogramación de operaciones, puede ser visto en (C. Bierwirth & Mattfeld, 1999).

Para empezar, dado que hay una gran variedad de factores de reprogramación, en el estudio se ha seleccionado un factor por cada clase definida previamente. Es importante recordar que, aunque solo se selecciona uno, toda la clase puede ser resuelta por el algoritmo haciendo un tratamiento similar, por lo cual el algoritmo tiene el alcance de procesar cualquiera de los 17 factores identificados.

De forma general, los experimentos han sido diseñados con respecto a los siguientes factores y niveles:

Tabla 13. Diseño de experimentos propuesto.

	Niveles	
Factores	**1**	**2**
Tamaño de la instancia	Pequeña	Grande
Incidencia de la interrupción	Temprana	Tardía
Duración de la interrupción	Corta	Larga

Nota: Elaboración propia.

Con los anteriores factores y niveles definidos se establece un diseño factorial 2^k, con k=3, que corresponde al número de factores. Esta configuración nos daría un total de 8 tratamientos, los cuales se relacionan como sigue:

Tabla 14. Tratamientos resultantes del diseño de experimentos factorial propuesto.

Tratamiento	**Tamaño**	**Incidencia**	**Duración**
1	Pequeña	Temprana	Corta
2	Pequeña	Temprana	Larga
3	Pequeña	Tardía	Corta
4	Pequeña	Tardía	Larga
5	Grande	Temprana	Corta
6	Grande	Temprana	Larga
7	Grande	Tardía	Corta
8	Grande	Tardía	Larga

Nota: Elaboración propia.

De forma general, para cada clase se seleccionaron un conjunto de instancias de comparación ya probadas en la fase predictiva, con el fin diseñar los escenarios. Se construyeron 10 escenarios por cada tratamiento. De cada escenario se recopilaron resultados de 5 réplicas para cada uno de los algoritmos. En cuanto al valor especifico de los niveles de cada factor, se consideraron como instancias pequeñas, las configuraciones *n* x *m* que contienen 100 operaciones y grandes 400 operaciones. La incidencia temprana consta de un punto aleatorio entre el 20 y el 40% del makespan de la instancia. La incidencia tardía se encuentra entre el 60 y 80% del makespan. Por último, la duración corta corresponde al 40% del tiempo de procesamiento máximo entre todas las operaciones de la instancia y la larga al 160%.

Definida la estructura de los experimentos realizados, se presentan los apartados más importantes en el diseño del algoritmo genético, utilizado como referente para comparar el desempeño del algoritmo transgénico.

Tabla 15. Estructura general del algoritmo genético diseñado.

Elemento/ Parámetro	**Descripción**
Forma de representación	Basada en operaciones
Función fitness	El ajuste de los individuos es medido a través del makespan
Población inicial	Generada aleatoriamente
Tamaño de población	100 para instancias pequeñas, 500 para instancias grandes
Operador de selección	Selección de individuos para el cruce mediante torneo. Porcentaje de selección: 0.7
Operador de cruce	Los individuos son cruzados con el operador, *JOX*, *Job Order Crossover* expuesto por Bierwirth en (Christian Bierwirth, 1995)
Operador de mutación	Cada individuo después del cruce, tiene asociada una probabilidad de mutación, si esta es inferior a 0.1 es mutado en 2 posiciones aleatoriamente, en caso contrario el individuo no muta.
Criterio de parada	El algoritmo genético se detendrá cuando alcance las 3000 generaciones.

Nota: Elaboración propia.

En cuanto a la parametrización del algoritmo propuesto para la fase reactiva, se ha tomado la misma parametrización inicial mostrada en la Tabla 10. Cabe resaltar, que el desempeño del algoritmo propuesto será medido en términos de eficiencia y estabilidad de los cronogramas obtenidos.

La eficiencia de un cronograma está asociada al procesamiento de las operaciones programadas, con la mejor utilización de los recursos, en este caso el tiempo de las máquinas o estaciones de trabajo. Dado que, para obtener el cronograma inicial, la

medida de eficiencia considerada es el makespan, en orden de preservar esta medida en el estudio de la fase reactiva, se diseña la siguiente expresión:

$$E = 1 - \left(\frac{C_{max}{}^{*} - C_{max}}{C_{max}}\right) * 100\% \qquad (5)$$

Donde C_{max} corresponde al makespan del cronograma original y $C_{max}{}^{*}$ al makespan del cronograma reparado. El valor de *E* corresponde al nivel de eficiencia obtenido para el nuevo cronograma, medido en términos porcentuales. Es decir, entre mayor sea, más se mantendrá la eficiencia (makespan) del cronograma original.

Ahora bien, la *estabilidad* del cronograma hace referencia a la desviación del cronograma reparado con respecto al inicial (S. D. Wu, Storer, & Pei-Chann, 1993). Esta puede ser medida a través del cálculo promedio de las diferencias en los tiempos de inicio del cronograma reparado con respecto al inicial. Su cálculo es obtenido a través de la siguiente expresión:

$$D = \frac{\sum_{j=1}^{n}\sum_{i=1}^{m}\left|r_{ij}{}^{*} - r_{ij}\right|}{\rho(R)} \qquad donde\ O_{ij} \in R \qquad (6)$$

Donde r_{ij} es el tiempo de inicio de la operación O_{ij} en el cronograma inicial y $r_{ij}{}^{*}$ corresponde al tiempo de inicio de la misma operación, pero en el cronograma reparado. *R* es el conjunto de las operaciones restantes por procesar, a partir del punto en la línea de tiempo en que aparece el factor de reprogramación. Finalmente, *ρ(R)* hace referencia a la cardinalidad del conjunto *R*.

Por otra parte, el algoritmo transgénico para la fase reactiva, incorpora lo que denominamos un *factor de estabilidad*, el cual le permitirá al encargado de la planeación, establecer un porcentaje de las operaciones en *R,* que desea mantener en la

misma secuencia del cronograma inicial, para de esta manera mitigar lo que se conoce como *nerviosismo* del sistema (Vieira et al., 2003b). En el presente estudio, el factor de estabilidad fue determinado de forma empírica, teniendo en cuenta la cantidad de ejecuciones que se llevaron a cabo durante el desarrollo del algoritmo.

Ahora procederemos a presentar los resultados de la eficiencia (E) y estabilidad (D) obtenidos por los dos algoritmos, en cada clase de factores de reprogramación identificada y para cada tratamiento planteado. Estos son presentados a través de gráficas que contienen los datos del mínimo, máximo y promedio de las medidas de desempeño obtenidas para el algoritmo genético (AG) y para el algoritmo transgénico (AT). Adicionalmente, en la sección de **Anexos** se presenta un ejemplo de forma gráfica de cada una de las clases de interrupciones tratadas por el algoritmo.

- ***Clase I: Fallo de máquina (1).***

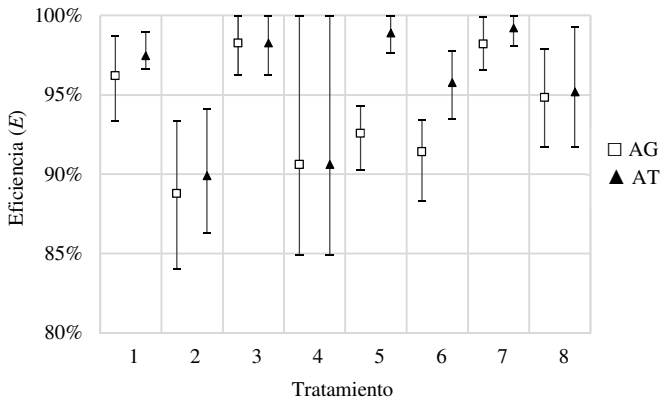


Figura 29. Resultados de la eficiencia obtenidos para el factor de reprogramación: fallo de máquina. Elaboración propia.

En la figura anterior, se puede apreciar que el desempeño del algoritmo transgénico en cuanto a eficiencia es igual o superior al algoritmo genético. Además, el algoritmo propuesto presenta una eficiencia muy cercana al 100%, cuando la duración

de la interrupción es corta. Igualmente es de notar que los escenarios donde se presenta el peor desempeño, es en donde se maneja una instancia pequeña y la duración de la interrupción es larga, pues se puede intuir que representa un efecto más nocivo para el piso del sistema.

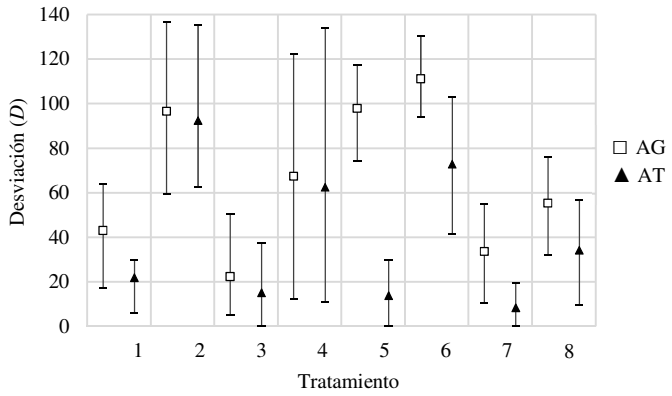


Figura 30. Resultados de la desviación obtenidos para el factor de reprogramación: fallo de máquina. Elaboración propia.

En el caso de la desviación, nuevamente se ve un mejor desempeño, superando al algoritmo genético en todos los tratamientos. Adicionalmente, el algoritmo transgénico muestra un mejor desempeño en los tratamientos donde el tamaño de la instancia es grande. Un caso particular de la diferencia del desempeño entre los dos algoritmos se da en el tratamiento 5, donde el algoritmo genético parece costarle obtener un cronograma menos desviado que el original.

▪ ***Clase II: Variación en el tiempo de procesamiento (7).***

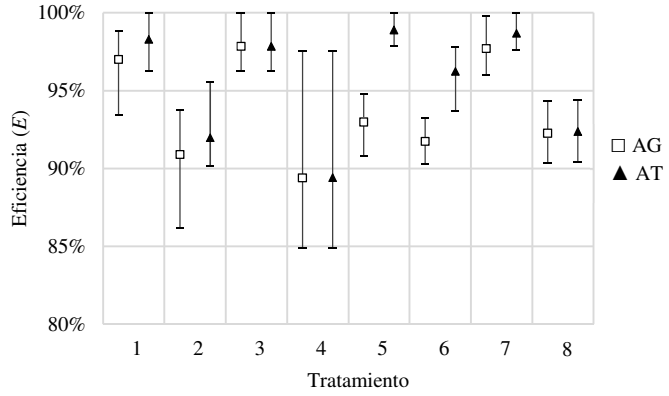


Figura 31. Resultados de la eficiencia obtenidos para el factor de reprogramación: variación en los tiempos de procesamiento. Elaboración propia.

En este conjunto de tratamientos, se muestra un comportamiento muy similar al obtenido en el factor de reprogramación anterior. Esto puede indicar que estas dos clases presentan los mismos efectos en el cronograma interrumpido. Otro hecho interesante es que el tratamiento 4 muestra una gran amplitud en su gama de resultados además del desempeño más bajo, probablemente porque se presenta lo que pareciera ser el peor escenario para una instancia pequeña, y es la variación significativa (larga) de los tiempos de procesamiento de algunas operaciones, en un momento final (tardío) de la ejecución del cronograma. La mezcla de estos factores en dichos niveles, puede ocasionar un bajo rango de acción para los algoritmos, ya que no existen muchas operaciones restantes, que a través de su reprogramación puedan absorber estos efectos.

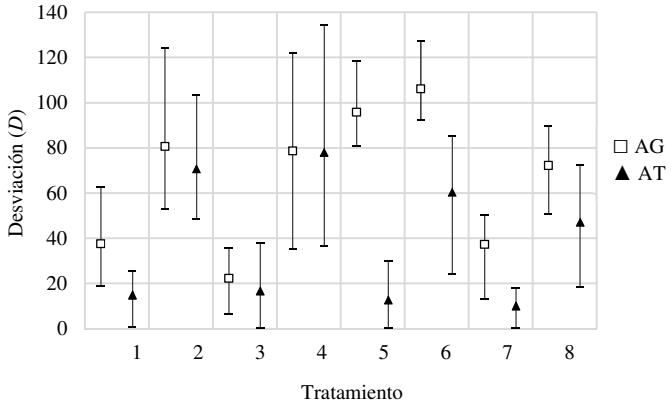


Figura 32. Resultados de la desviación obtenidos para el factor de reprogramación: variación en los tiempos de procesamiento. Elaboración propia.

Con respecto a la desviación de los cronogramas, se evidencia el mismo comportamiento que la clase anterior. Adicionalmente, pareciera que el factor de incidencia de la interrupción, no tuviera un efecto significativo en la desviación de los cronogramas obtenidos. Es clave reconocer que, en esta clase la desviación de los tratamientos con duración de la interrupción corta no sobrepasa las 20 unidades de tiempo.

▪ ***Clase III: Llegada de un nuevo trabajo (11).***

En esta clase es importante destacar que el factor de duración de la interrupción es manejado de forma análoga a través de las siguientes consideraciones: el efecto de la duración corta y larga de la interrupción es homologado con el efecto de incluir un nuevo trabajo con un tiempo medio de procesamiento entre sus operaciones bajo y alto. En este sentido, para determinar el tiempo de procesamiento de las operaciones en el primer nivel (bajo), fue considerada una distribución uniforme U(20,60) y en el segundo (alto) una U(80,120). Por último el número de operaciones del trabajo que llega, se

determinó a través de una distribución de uniforme U(5,10) para las instancias pequeñas y una U(10,20) para las grandes.

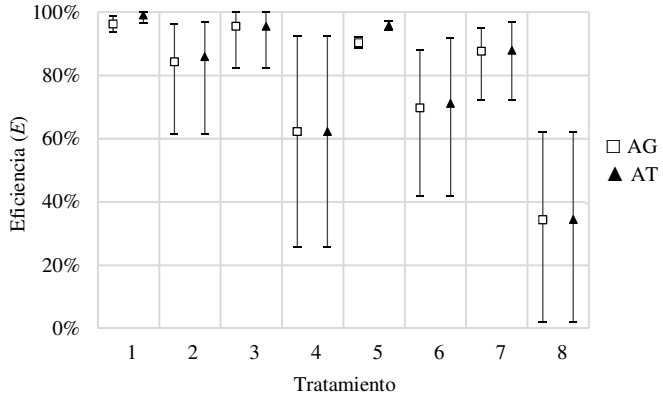


Figura 33. Resultados de la eficiencia obtenidos para el factor de reprogramación: llegada de un nuevo trabajo. Elaboración propia.

De esta manera han sido obtenidos los resultados mostrados en la Figura 33. Se puede visualizar que el desempeño de ambos algoritmos es más parejo que en las anteriores clases. De igual manera, se evidencia un comportamiento generalizado, indicando que el nivel que más afecta la variabilidad de los resultados es la duración larga.

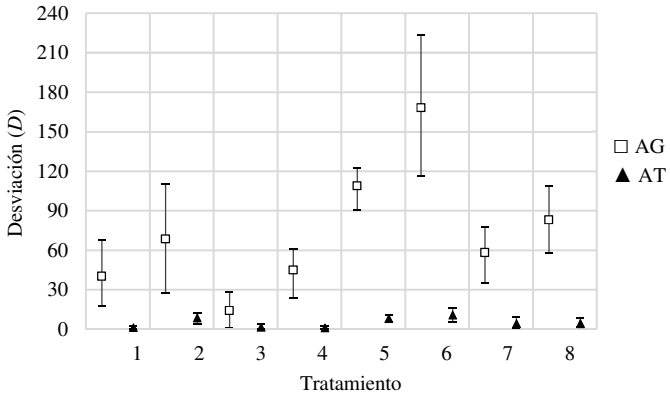


Figura 34. Resultados de la desviación obtenidos para el factor de reprogramación: llegada de un nuevo trabajo. Elaboración propia.

En cuanto a la desviación de los cronogramas obtenidos, se observa que el algoritmo transgénico muestra resultados significativamente superiores. Estos resultados pueden estar evidenciando que el cronograma inicial no ha podido absorber la nueva orden de trabajo y este ha quedado casi intacto, relegando el procesamiento de la nueva orden para el final. Lo anterior permite intuir que los valores eficiencia obtenidos para esta clase no están relacionados con la operatividad de los algoritmos, si no por el efecto que implica incluir un nuevo trabajo luego de haber empezado el cronograma, y en el peor de los casos cuando este ya está finalizando. Por esta razón, esta clase de interrupción puede ser manejada a través de la política de reprogramación periódica, que permite poner en espera la nueva de orden de trabajo para ser incluida en el próximo periodo de planeación, con el fin de mantener un uso eficiente de los recursos productivos.

- ***Clase IV: Trabajo urgente (14).***

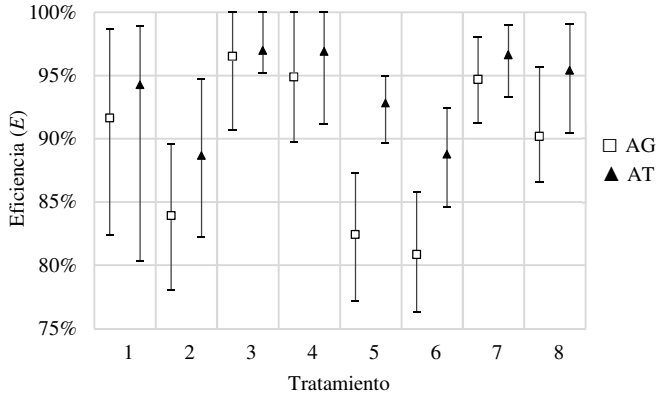


Figura 35. Resultados de la eficiencia obtenidos para el factor de reprogramación: trabajo urgente. Elaboración propia.

En esta clase, dado que hay un cambio de prioridad en uno de los trabajos, el factor de duración de la interrupción, es homologado con el factor de número de

operaciones restantes del trabajo urgente. Es decir, el factor de duración en su primer nivel, es representado por el trabajo con menor número de operaciones restantes en el punto de interrupción y lo contario para el segundo nivel.

Los resultados obtenidos en la medida de eficiencia, siguen demostrando un desempeño superior del algoritmo transgénico. Se puede apreciar que el valor medio de esta medida en todos los tratamientos es superior al 88%. Adicionalmente, el factor más influyente es el número de operaciones. Cuando el trabajo urgente tiene mayor número de operaciones restantes por procesar, la eficiencia se ve afectada.

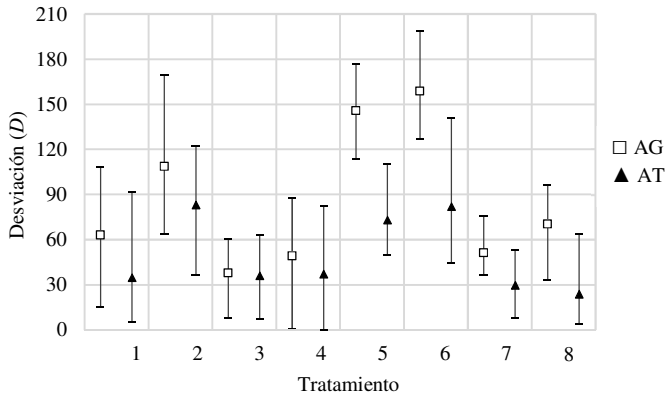


Figura 36. Resultados de la desviación obtenidos para el factor de reprogramación: trabajo urgente. Elaboración propia.

En cuanto a la desviación de los cronogramas obtenidos, esta medida resulta sensible al factor de incidencia de la interrupción, cuando esta es temprana, la estabilidad del cronograma disminuye. Adicionalmente, este efecto es amplificado cuando el trabajo urgente cuenta con más operaciones restantes por procesar.

▪ ***Clase V: Cancelación de trabajo (16).***

Para esta clase, el factor de duración de la interrupción fue abordado de la misma forma que la clase anterior. En esta clase de interrupción, los algoritmos enfrentan la eliminación de operaciones programadas en el cronograma. Por tal razón el makespan después de la eliminación de estas operaciones puede ser inferior al obtenido inicialmente. De esta manera, en la Figura 37 se aprecian valores que superan el 100%, aunque preservando el sentido de la medida de eficiencia, entre mayor sea mejor.

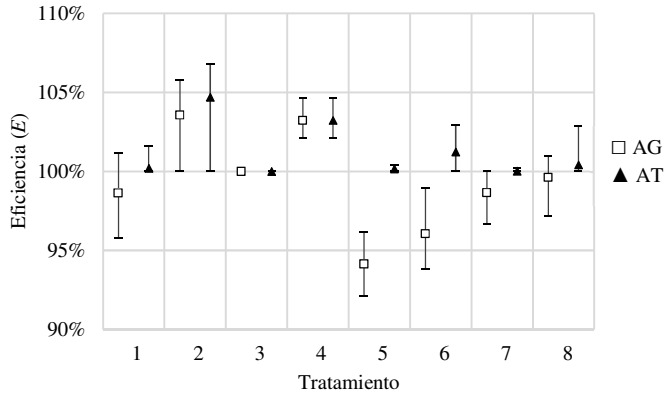


Figura 37. Resultados de la eficiencia obtenidos para el factor de reprogramación: cancelación de trabajo. Elaboración propia.

El algoritmo nuevamente muestra resultados superiores, indicando en todos los tratamientos una eficiencia igual o superior al 100%. El factor más influyente es el número de operaciones restantes del trabajo cancelado, entre mayor sea este, mayor será la eficiencia, dado que se relaja levemente la capacidad de los recursos.

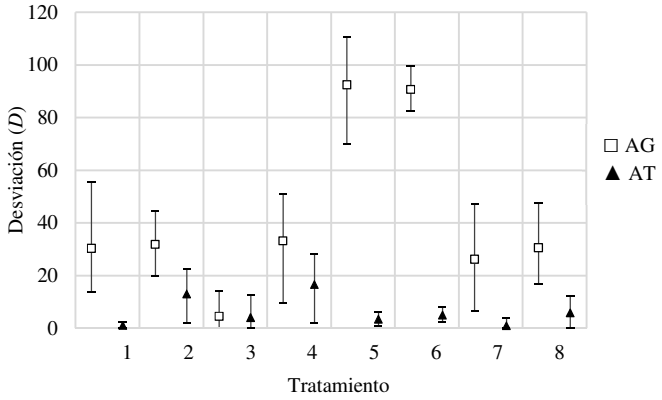


Figura 38. Resultados de la desviación obtenidos para el factor de reprogramación: cancelación de trabajo. Elaboración propia.

La desviación obtenida ratifica el desempeño del algoritmo transgénico. Se puede apreciar que hay una variabilidad en los resultados significativamente reducida en varios de los tratamientos.

De forma general, el algoritmo propuesto ha demostrado tener un desempeño superior al algoritmo genético, obteniendo mejores resultados en el 65% de los tratamientos diseñados con respecto a la medida de eficiencia y en un 98% de los tratamientos con respecto a la medida de estabilidad. El porcentaje de tratamientos restantes para ambas medidas, revelan el mismo desempeño para ambos algoritmos.

La eficiencia promedio de todo el estudio para el algoritmo transgénico es un 2,3% mayor a la obtenida por el algoritmo genético. Donde se percibe una mayor diferencia de su desempeño es en la clase IV, con una eficiencia promedio del 4,4% por encima de la obtenida por el algoritmo genético. Con respecto a la medida de estabilidad, el algoritmo transgénico muestra una desviación promedio de 28 unidades mientras el algoritmo genético de 67 unidades, demostrando de esta manera que el

algoritmo transgénico constituye un método de regeneración completa que mitiga el nerviosismo del sistema en mayor medida que el algoritmo genético.

Finalmente, se ha notado que la eficiencia y la estabilidad de un cronograma, pueden ser medidas de desempeño conflictivas entre sí, por lo que el encargado de la planeación tendrá que tomar decisiones de forma rápida y en función de obtener un sistema productivo balanceado.

8. Conclusiones

El estudio se ha concentrado en desarrollar un algoritmo computacional, para facilitar el trabajo de los encargados de la planeación de la producción en entornos de manufactura Job Shop.

Dada la brecha existente entre los modelos teóricos y los entornos reales de manufactura, se incluyó la aparición de eventos inesperados que alteran la ejecución de los cronogramas de producción, con el fin de darle un alcance más realista al algoritmo propuesto. En este sentido se acogió un marco de referencia de reprogramación, con el cual fue delimitado el enfoque del estudio del JSRP.

Dada la creciente complejidad adquirida con la adición de los anteriores elementos, proponemos un algoritmo basado en la computación transgénica. Este es desarrollado en beneficio de producir cronogramas de producción de calidad, en términos de eficiencia y estabilidad.

El desempeño del algoritmo es validado tanto para la fase predictiva como para la fase reactiva. En la fase predictiva han sido seleccionadas un conjunto de instancias reconocidas en la literatura del JSSP. El algoritmo ha demostrado buenos resultados,

logrando el mejor valor conocido para el 52% de las instancias seleccionadas y valores muy cercanos para las instancias restantes. Estos valores en promedio no superan el 1,3% de diferencia para las instancias medianas y el 4,2% para las grandes. Adicionalmente, se ha percibido cómo a medida que incrementa el tamaño de la instancia, el desempeño del algoritmo se va deteriorando, por lo que si se desea obtener un rendimiento similar no se sugiere que el algoritmo sea aplicado a instancias con más de 400 operaciones. En cuanto a los tiempos medios de solución, el algoritmo demora 1 seg. para resolver instancias pequeñas, 9 seg. para medianas y 116 seg. para grandes. Esto quiere decir que, en promedio, el algoritmo transgénico propuesto resuelve problemas grandes (entre 225 y 400 operaciones) en 2 minutos, con una desviación del makespan, respecto a la mejor solución conocida, de un 5%.

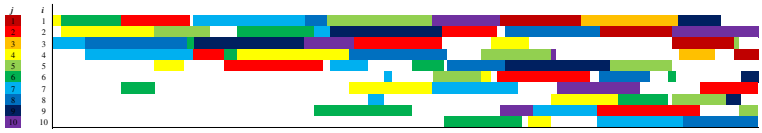
Respecto a la fase reactiva, el desempeño del algoritmo transgénico ha sido comparado con el de un algoritmo genético. El algoritmo propuesto ha demostrado tener un desempeño superior, obteniendo mejores resultados en el 65% de los tratamientos diseñados con respecto a la medida de eficiencia y en un 98% de los tratamientos con respecto a la medida de estabilidad. El porcentaje de tratamientos restantes para ambas medidas, revelan el mismo desempeño para ambos algoritmos.

9. Anexos

Anexo 1

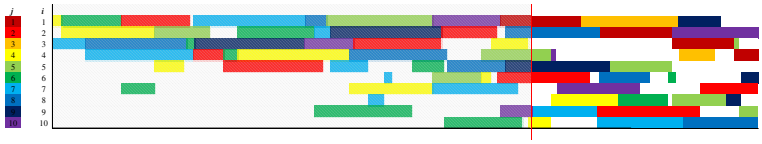
1. Ejemplo de interrupción Clase I

Cronograma inicial



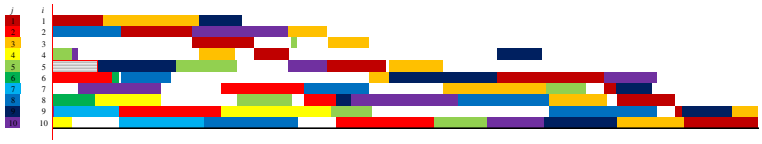
Cronograma inicial	
No. Trabajos (j)	10
No. Máquinas (i)	10
No. Operaciones	100
Makespan	1059 u.t.

Cronograma afectado *



Información de la interrupción	
Clase	I
Interrupción	1. Fallo de máquin
Instancia	100 operaciones
Punto de interrupción	424 u.t
Tiempo de inactividad	40 u.t.
No. Máquina afectada	5
Factor de estabilidad	0,2

Cronograma reparado (desde el punto de interrupción) **



Medidas de desempeño	
Eficiencia	98,96%
Desviación	25,82 u.t.

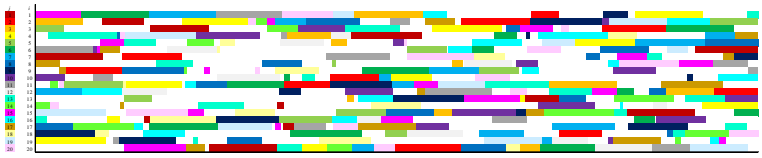
1070

* Antes de la linea roja se presentan las operaciones ejecutadas hasta el punto de interrupción.

** Se muestra el nuevo cronograma obtenido para las actividades pendientes por ejecutar.

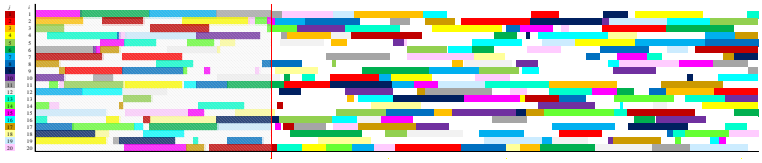
2. Ejemplo de interrupción Clase II

Cronograma inicial



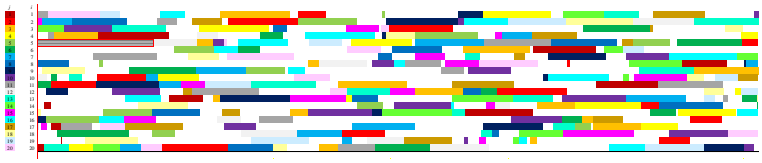
Cronograma inicial	
No. Trabajos (j)	20
No. Máquinas (i)	20
No. Operaciones	400
Makespan	1645 u.t.

Cronograma afectado *



Información de la interrupción	
Clase	II
Interrupción	7. Variación en el t
Instancia	400 operaciones
Punto de interrupción	325 u.t.
Tiempo de inactividad	160 u.t.
No. Máquina afectada	5
Factor de estabilidad	0,4

Cronograma reparado (desde el punto de interrupción) **



Medidas de desempeño	
Eficiencia	94,83%
Desviación	73,08 u.t.

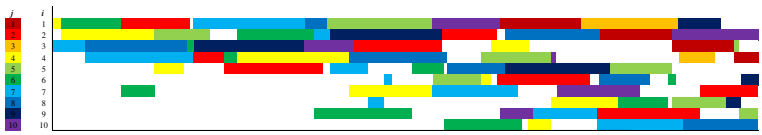
1730

* Antes de la linea roja se presentan las operaciones ejecutadas hasta el punto de interrupción.

** Se muestra el nuevo cronograma obtenido para las actividades pendientes por ejecutar.

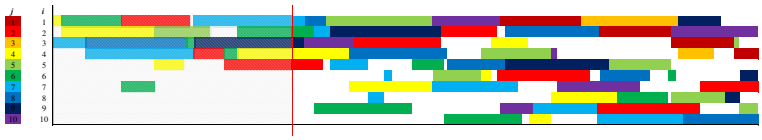
3. Ejemplo de interrupción Clase III

Cronograma inicial



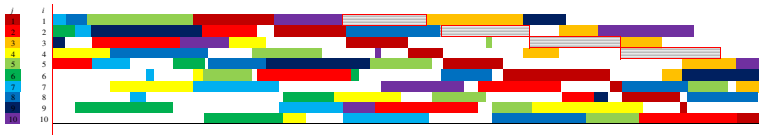
Cronograma inicial	
No. Trabajos (j)	10
No. Máquinas (i)	10
No. Operaciones	100
Makespan	1059 u.t.

Cronograma afectado *



Información de la interrupción	
Clase	III
Interrupción	11. Llegada de un r
Instancia	100 operaciones
Punto de interrupción	212
Tiempos de procesamiento	U(80,120)
No. Operaciones	U(5,10)
Factor de estabilidad	0

Cronograma reparado (desde el punto de interrupción) **



Medidas de desempeño	
Eficiencia	95,00%
Desviación	53,71 u.t.

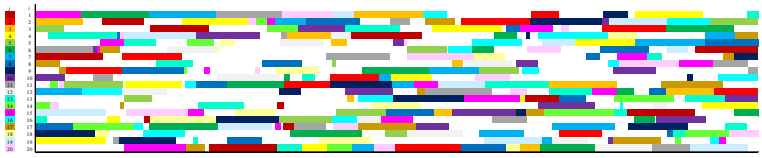
1112

* Antes de la linea roja se presentan las operaciones ejecutadas hasta el punto de interrupción.

** Se muestra el nuevo cronograma obtenido para las actividades pendientes por ejecutar.

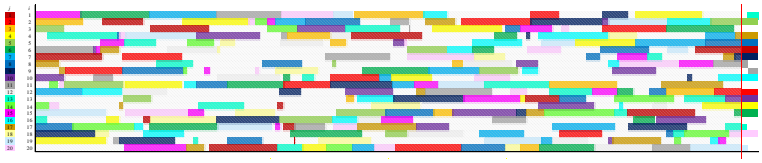
4. Ejemplo de interrupción Clase IV

Cronograma inicial



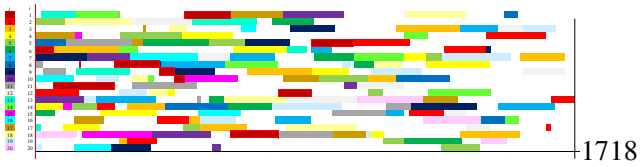
Cronograma inicial	
No. Trabajos (j)	20
No. Máquinas (i)	20
No. Operaciones	400
Makespan	1645 u.t.

Cronograma afectado *



Información de la interrupción	
Clase	IV
Interrupción	14. Trabajo urgente
Instancia	400 operaciones
Punto de interrupción	974 u.t.
No. Trabajo	1
Factor de estabilidad	0,6

Cronograma reparado (desde el punto de interrupción) **

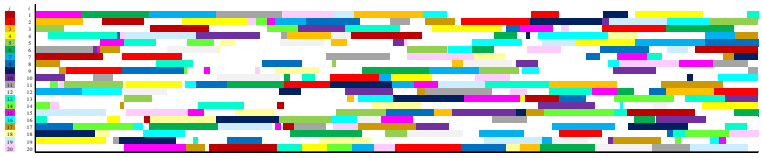


Medidas de desempeño	
Eficiencia	95,56%
Desviación	22,4 u.t.

* Antes de la linea roja se presentan las operaciones ejecutadas hasta el punto de interrupción.
** Se muestra el nuevo cronograma obtenido para las actividades pendientes por ejecutar.

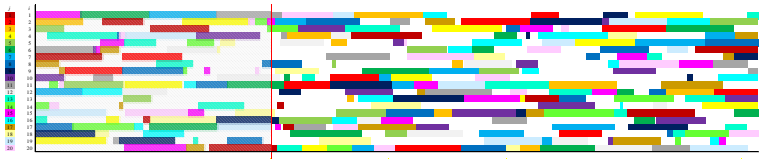
5. Ejemplo de interrupción Clase V

Cronograma inicial



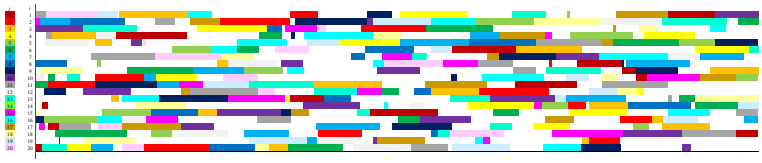
Cronograma inicial	
No. Trabajos (*j*)	20
No. Máquinas (*i*)	20
No. Operaciones	400
Makespan	1645 u.t.

Cronograma afectado *



Información de la interrupción	
Clase	V
Interrupción	16. Cancelación de
Instancia	400 operaciones
Punto de interrupción	325 u.t.
No. Trabajo	14
Factor de estabilidad	0,8

Cronograma reparado (desde el punto de interrupción) **



Medidas de desempeño	
Eficiencia	100,24%
Desviación	4,39 u.t.

1641

* Antes de la linea roja se presentan las operaciones ejecutadas hasta el punto de interrupción.

** Se muestra el nuevo cronograma obtenido para las actividades pendientes por ejecutar.

10. Biografía

Abumaizar, R. J., & Svestka, J. A. (1997). Rescheduling job shops under random disruptions. *International Journal of Production Research*, *35*(7), 2065–2082. https://doi.org/10.1080/002075497195074

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, *34*(3), 391–401.

Ali, K. Ben, Telmoudi, A. J., & Gattoufi, S. (2018). An Improved Genetic Algorithm with Local Search for Solving the DJSSP with New Dynamic Events. *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, *1*, 1137–1144.

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, *3*(2), 149–156.

Arisha, A., Young, P., & El Baradie, M. (2001). *Job shop scheduling problem: an overview*.

Barboza, A. O. (2005). *Simulação e técnicas da computação evolucionária aplicadas a problemas de programação linear inteira mista*.

Bean, J. C., Birge, J. R., Mittenthal, J., & Noon, C. E. (1991). Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, *39*(3), 470–483.

Bierwirth, C., & Mattfeld, D. C. (1999). Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, *7*(1), 1–17.

https://doi.org/10.1162/evco.1999.7.1.1

Bierwirth, Christian. (1995). A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum*, *17*(2–3), 87–92.

Brucker, P. (2007). Scheduling algorithms. In *Scheduling Algorithms*. https://doi.org/10.1007/978-3-540-69516-5

Chattoe-Brown, E. (1998). Just How (Un)realistic Are Evolutionary Algorithms As Representations of Social Processes? *Journal of Artificial Societies and Social Simulation*, *1*(3), 1–2.

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. *Computers & Industrial Engineering*, *30*(4), 983–997.

Church, L. K., & Uzsoy, R. (1992). Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, *5*(3), 153–163.

Dhingra, J. S., Musser, K. L., & Blankenehip, G. L. (1993). *Reactive Operations Scheduling for Flexible Manufacturing Systems*.

DUTTA, A. (1990). Reacting to scheduling exceptions in FMS environments. *IIE Transactions*, *22*(4), 300–314.

Espinal, A. A. C., Velásquez, E. R., & Restrepo, M. I. L. (2008). Secuenciación de operaciones para configuraciones de planta tipo flexible Job Shop: Estado del arte. *Revista Avances En Sistemas e Informática*, *5*(3), 151–161.

Fang, H.-L., Ross, P., & Corne, D. (1993a). *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems*. University of Edinburgh, Department of Artificial Intelligence.

Fang, H.-L., Ross, P., & Corne, D. (1993b). A Promising genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems. *Proceedings of the Fifth International Conference on Genetic Algorithms*, (623), 375–382. https://doi.org/10.1.1.42.6722

Fisher, H., & Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. *Industrial Scheduling*.

Gao, Y., Ding, Y., & Zhang, H. (2009). Job-shop scheduling considering rescheduling in uncertain dynamic environment. *2009 International Conference on Management Science and Engineering*, 380–384.

Gershwin, S. B. (1994). *Manufacturing systems engineering*.

Giffler, B., & Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, *8*(4), 487–503.

Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* (pp. 2093–2229). Springer.

Goldbarg, E.F.G., Goldbarg, M. C., & Bagi, L. B. (2007). Transgenetic algorithm: A new evolutionary perspective for heuristics design. *Proceedings of GECCO 2007: Genetic and Evolutionary Computation Conference, Companion Material*, 2701–2708. https://doi.org/10.1145/1274000.1274040

Goldbarg, E F G, Castro, M. P., & Goldbarg, M. C. (2006). A transgenetic algorithm for the gas network pipe sizing problem. *Computational Methods*, *1*, 893–904.

Goldbarg, Elizabeth F G, Goldbarg, M. C., & Bagi, L. B. (2007). Transgenetic algorithm: A new evolutionary perspective for heuristics design. *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation*, 2701–2708.

Goldbarg, ELIZABETH FERREIRA GOUVÊA, Goldbarg, M. C., & Costa, W. E. (2004). A transgenetic algorithm for the permutation flow-shop sequencing problem. *WSEAS Transactions on Systems*, *1*(3), 40–45.

Goldbarg, M., Goldbarg, E., & Quadr, P. (2002a). Transgenética computacional: Uma aplicação ao problema quadrático de alocação. *Pesquisa Operacional*, 359–386. https://doi.org/10.1590/S0101-74382002000300005

Goldbarg, M.C., & Gouvêa, E. (2001). Extra-Intracellular Transgenetic Algorithm applied to the Graph Coloring Problem. *Design*, 321–326.

Goldbarg, Marco César, & Gouvêa, E. F. (2001). Transgenética computacional. *Relatório Técnico, Programa de Engenharia de Produção Da COPPE/UFRJ*.

Gouvêa, E., & Goldbarg, M. (2001). ProtoG: A computational transgenetic algorithm. *Proceedings of MIC 2001–4th Metaheuristics …*, (January), 625–630.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics* (Vol. 5, pp. 287–326). Elsevier.

Hasan, S. M. K., Sarker, R., & Essam, D. (2011). Genetic algorithm for job-shop scheduling with machine unavailability and breakdowns. *International Journal of Production Research*, *49*(16), 4999–5015.

He, L., & Mort, N. (2000). Hybrid genetic algorithms for telecommunications network back-up routeing. *BT Technology Journal*, *18*(4), 42–50.

Holland, J. H., & others. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Holland John, H. (1975). Adaptation in natural and artificial systems. *Ann Arbor: University of Michigan Press*.

Hopp, W. J., & Spearman, M. L. (2011). *Factory physics*. Waveland Press.

King, J. R. (1976). The theory-practice gap in job-shop scheduling. *Production Engineer*, *55*(3), 137–143.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Krasnogor, N., & Smith, J. (2005). A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues. *IEEE Transactions on Evolutionary Computation*, *9*(5), 474–488. https://doi.org/10.1109/TEVC.2005.850260

Kundakc\i, N., & Kulak, O. (2016). Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. *Computers & Industrial Engineering*, *96*, 31–51.

Larsen, R., & Pranzo, M. (2019). A framework for dynamic rescheduling problems. *International Journal of Production Research*, *57*(1), 16–33.

Lawrence, S. (1984). Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University*.

Li, H., Li, Z., Li, L. X., & Hu, B. (2000). A production rescheduling expert simulation system. *European Journal of Operational Research*, *124*(2), 283–293.

Li, R.-K., Shyu, Y.-T., & Adiga, S. (1993). A heuristic rescheduling algorithm for computer-based production scheduling systems. *The International Journal Of Production Research*, *31*(8), 1815–1826.

Li, Y.-C. E., & Shaw, W. H. (1998). Simulation modeling of a dynamic job shop rescheduling with machine availability constraints. *Computers & Industrial Engineering*, *35*(1–2), 117–120.

Lu, M.-S., & Romanowski, R. (2013). Multicontextual dispatching rules for job shops with dynamic job arrival. *The International Journal of Advanced Manufacturing Technology*, *67*(1–4), 19–33.

Méndez, G., Álvarez, L., Caicedo, C., & Malaver, M. (2013). Sistema experto para la programación de producción-investigación y desarrollo de un prototipo. *Universidad Distrital Francisco José de Caldas, Colombia*.

Merz, P., & Freisleben, B. (1999). A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, *3*, 2063-

2070 Vol. 3. https://doi.org/10.1109/CEC.1999.785529

Mohan, J., Lanka, K., & Rao, A. N. (2019). A Review of Dynamic Job Shop Scheduling Techniques. *Procedia Manufacturing*, *30*, 34–39.

Moratori, P., Petrovic, S., & Vazquez-Rodriguez, J. A. (2012). Match-up approaches to a dynamic rescheduling problem. *International Journal of Production Research*, *50*(1), 261–276.

Moratori, P., Petrovic, S., & Vázquez, A. (2008). Match-up strategies for job shop rescheduling. *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 119–128.

Morris, G. M., Goodsell, D. S., Halliday, R. S., Huey, R., Hart, W. E., Belew, R. K., & Olson, A. J. (1998). Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, *19*(14), 1639–1662.

Moscato, P., & Cotta, C. (2003a). A gentle introduction to memetic algorithms. In *Handbook of metaheuristics* (pp. 105–144). Springer.

Moscato, P., & Cotta, C. (2003b). Una introducción a los algoritmos meméticos. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, *7*(19), 0.

Niehues, M., Blum, M., Teschemacher, U., & Reinhart, G. (2018). Adaptive job shop control based on permanent order sequencing. *Production Engineering*, *12*(1), 65–71.

Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing

systems. *Journal of Scheduling*, *12*(4), 417.

Panwalkar, S. S., & Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, *25*(1), 45–61.

Phanden, R. K., Jain, A., & Verma, R. (2011). Integration of process planning and scheduling: a state-of-the-art review. *International Journal of Computer Integrated Manufacturing*, *24*(6), 517–534.

Pinedo, M. L. (2008). Scheduling: Theory, algorithms, and systems. In *Scheduling: Theory, Algorithms, and Systems*. https://doi.org/10.1007/978-0-387-78935-4

Plotkin, H. C. (1995). Non-genetic transmission of information: Candidate cognitive processes and the evolution of culture. *Behavioural Processes*, *35*(1–3), 207–213.

Radcliffe, N. J., & Surry, P. D. (1994). Formal memetic algorithms. *AISB Workshop on Evolutionary Computing*, 1–16.

Rangsaritratsamee, R., Ferrell Jr, W. G., & Kurz, M. B. (2004). Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers & Industrial Engineering*, *46*(1), 1–15.

Salido, M. A., Escamilla, J., Barber, F., & Giret, A. (2017). Rescheduling in job-shop problems for sustainable manufacturing systems. *Journal of Cleaner Production*, *162*, S121--S132.

Sarker, R., Omar, M., Hasan, S. M. K., & Essam, D. (2013). Hybrid Evolutionary Algorithm for job scheduling under machine maintenance. *Applied Soft Computing*, *13*(3), 1440–1447.

Smith, S. F. (1995). Reactive scheduling systems. In *Intelligent scheduling systems* (pp. 155–192). Springer.

Subramaniam, V., & Raheja, A. S. (2003). mAOR: A heuristic-based reactive repair mechanism for job shop schedules. *The International Journal of Advanced Manufacturing Technology*, *22*(9–10), 669–680.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285. https://doi.org/https://doi.org/10.1016/0377-2217(93)90182-M

Tarantilis, C. D., & Kiranoudis, C. T. (2002). A list-based threshold accepting method for job shop scheduling problems. *International Journal of Production Economics*, *77*(2), 159–171.

Tomassini, M. (1995). A survey of genetic algorithms. In *Annual reviews of computational physics III* (pp. 87–118). World Scientific.

Uhlmann, I. R., & Frazzon, E. M. (2018). Production rescheduling review: Opportunities for industrial integration and practical applications. *Journal of Manufacturing Systems*, *49*, 186–193.

Vieira, G. E., Herrmann, J. W., & Lin, E. (2000). Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies. *International Journal of Production Research*, *38*(8), 1899–1915.

Vieira, G. E., Herrmann, J. W., & Lin, E. (2003a). Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of Scheduling*, *6*(1), 39–

62.

Vieira, G. E., Herrmann, J. W., & Lin, E. (2003b). Resheduling Manufacturing System : A Framework of strategies, Policies , and Methods. *Jounal of Scheduling*, *6*(1), 39–62.

Wilson, E. O. (1999). *Consilience: The unity of knowledge* (Vol. 31). Vintage.

Wright, R. (2001). *Nonzero: The logic of human destiny*. Vintage.

Wu, H.-H., & Li, R.-K. (1995). A new rescheduling method for computer based scheduling systems. *International Journal of Production Research*, *33*(8), 2097–2110.

Wu, S. D., Storer, R. H., & Pei-Chann, C. (1993). One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research*, *20*(1), 1–14.

Xue, L., Wang, P., Cheng, H., Zeng, P., & Yu, H. (2017). Event-driven dynamic job shop scheduling execution based on improved genetic algorithm and ontology. *2017 Chinese Automation Congress (CAC)*, 6430–6435.

Yamada, T., & Nakano, R. (1992). A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems. *PPSN*, *2*, 281–290.

Zhang, L., Gao, L., & Li, X. (2013a). A hybrid genetic algorithm and tabu search for a multi-objective dynamic job shop scheduling problem. *International Journal of Production Research*, *51*(12), 3516–3531.

Zhang, L., Gao, L., & Li, X. (2013b). A hybrid intelligent algorithm and rescheduling

technique for job shop scheduling problems with disruptions. *The International Journal of Advanced Manufacturing Technology*, *65*(5–8), 1141–1156.

Zou, Z., & Li, C. (2006). Integrated and events-oriented job shop scheduling. *The International Journal of Advanced Manufacturing Technology*, *29*(5–6), 551–556.