Solución del problema del "*Flow Shop*" híbrido (HFS) con máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia y "*buffers"* de tamaño limitado implementando dos algoritmos híbridos.

Diego Fernando Gómez Acevedo

María Alejandra Mantilla Ruíz

Trabajo de Grado para optar el título de Ingeniero Industrial

Director: Eliana Marcela Peña Tibaduiza

M.s.c. Ingeniería industrial.

Codirector: Edwin Alberto Garavito Hernández

Esp. Producción – mejoramiento continuo.

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Estudios Industriales y Empresariales

Bucaramanga

2017

Agradecimientos

A nuestras familias por el apoyo recibido durante el desarrollo del proyecto y toda nuestra formación universitaria, a los docentes M.s.c. Eliana Marcela Peña Tibaduiza y Esp. Edwin Alberto Garavito Hernández por su orientación y acompañamiento en el presente proyecto, al grupo de investigación Ópalo por brindarnos el espacio para el desarrollo del trabajo, a la Escuela de Estudios Industriales y Empresariales, y a la Universidad Industrial de Santander por permitirnos ser parte del alma mater y contribuir a nuestro desarrollo profesional y personal.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de apéndices

Los siguientes apéndices se encuentran en un CD adjunto, en la base de datos de la biblioteca central de la Universidad Industrial de Santander.

Apéndice A. Tablas con los tiempos de procesamiento y alistamiento de las tareas para cada una de las instancias.

Apéndice B. Pseudocódigo de programación en lenguaje Matlab del algoritmo genético híbrido.

Apéndice C. Resultados tenidos en cuenta para el diseño experimental del algoritmo genético híbrido para el problema del HFS con buffers de tamaño limitado.

Apéndice D. Diseño experimental para la determinación de factores incidentes del algoritmo genético híbrido.

Apéndice E. Pseudocódigo de programación en lenguaje Matlab del algoritmo colonia de abejas artificial mejorada.

Apéndice F. Resultados tenidos en cuenta para el diseño experimental de la colonia de abejas artificial mejorada para el problema del HFS con buffers de tamaño limitado.

Apéndice G. Diseño experimental para la determinación de factores incidentes del algoritmo colonia de abejas artificial mejorada.

Apéndice H. Resultados tenidos en cuenta para la validación del algoritmo genético hibrido para el problema del HFS con buffers de tamaño limitado.

Apéndice I. Resultados tenidos en cuenta para la validación del algoritmo basado en una colonia de abejas artificial mejorada para el problema del HFS con buffers de tamaño limitado.

Apéndice J. Resultados tenidos en cuenta para la validación de la metaheurística basada en el ICA para el problema del HFS con buffers de tamaño limitado.

Apéndice K. Artículo “Solución al problema del "*flow shop*” híbrido con máquinas paralelas no relacionadas y “buffers” de tamaño limitado mediante dos algoritmos híbridos”.

Resumen

Título: Solución del problema del "*Flow Shop*" híbrido (HFS) con máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia y "*buffers"* de tamaño limitado implementando dos algoritmos híbridos.*

Autores: Diego Fernando Gómez Acevedo **

María Alejandra Mantilla Ruíz. **

Palabras Claves: *Flow shop* hibrido, máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia, buffers de tamaño limitado, metaheurísticas, algoritmo genético, colonia artificial de abejas.

Descripción:

En la presente investigación se estudia el problema del *flow shop* hibrido con máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia y buffers de tamaño limitado, con el objetivo de minimizar el *makespan*; problema que ha sido clasificado como NP-hard debido a la complejidad computacional que representa. Con base en lo anterior, diversos autores han diseñado heurísticas y metaheurísticas, con el fin de encontrar buenas soluciones en tiempos computacionales razonables.

Entre los métodos comúnmente utilizados se encuentran el algoritmo genético, recocido simulado, la búsqueda Tabú, metaheurísticas basadas en inteligencia artificial de enjambre, entre otras. Por ende, en este proyecto se diseñó un algoritmo genético híbrido, en el cual se genera la población inicial con base en los tiempos de alistamiento y procesamiento, y combina una búsqueda variable de la vecindad. Así mismo, se plantea un algoritmo híbrido basado en una colonia artificial de abejas, que involucra operadores de cruce y mutación del algoritmo genético, una búsqueda variable de la vecindad modificada y procedimientos de construcción y destrucción del algoritmo iterado codicioso.

Los algoritmos se validaron mediante 20 instancias en lenguaje de programación Matlab® y los resultados se compararon con los hallados mediante la metaheurística basada en el algoritmo competitivo imperialista (ICA) y la solución obtenida por un método exacto, utilizando el software GAMS®. Los resultados muestran que, los algoritmos propuestos logran tener soluciones de buena calidad en tiempos computacionales razonables.

---

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Estudios Industriales y Empresariales. Director: Eliana Marcela Peña Tibaduiza, M.s.c. Ingeniería Industrial. Codirector: Edwin Alberto Garavito Hernández. Esp. Producción – mejoramiento continuo.

Abstract

Título: Solution to the hybrid flow shop problem with unrelated parallel machines and limited size buffers using two hybrid algorithms.*

Autores: Diego Fernando Gómez Acevedo **

María Alejandra Mantilla Ruíz. **

Keywords: Hybrid flow shop, unrelated parallel machines, sequence dependent setup times, limited size buffers, genetic algorithm, artificial bee colony, heuristics, metaheuristics.

Description:

This research adresses a hybrid flow shop problem with unrelated parallel machines, sequence dependent setup times and limited size buffers, in order to minimize the makespan; This problem has been classified as NP-hard due to the computational complexity it represents. Based on the above, several authors have designed heuristics and metaheuristics, in order to find good solutions in reasonable computational times.

Methods such as genetic algorithms, simulated annealing, tabu search, metaheuristics based on artificial swarm intelligence, among others, are commonly used to solve hybrid flow shop problems. Therefore, in this project a hybrid genetic algorithm was designed, in which the initial population is generated based on setup and processing times, and new solutions are generated from a variable search of the neighborhood. Also, an improved artificial bee colony (IABC) is proposed, involving crossover and mutation operators of the genetic algorithm, a modified variable neighborhood search, and destruction and construction procedures of the iterated greedy algorithm.

The algorithms were validated using 20 instances in Matlab® programming language and the results were compared with those found using the metaheuristic based on the imperialist competitive algorithm (ICA) and the solution obtained by an exact method using GAMS® software. The results show that the proposed algorithms generate good quality solutions in reasonable computational times.

* Bachelor Thesis

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Estudios Industriales y Empresariales. Director: Eliana Marcela Peña Tibaduiza, M.s.c. Ingeniería Industrial. Codirector: Edwin Alberto Garavito Hernández. Esp. Producción – mejoramiento continuo.

Introducción

Fátima, Perassoli, Santos, Peterson y Paraíso (2014) afirman que "la programación de la producción es una de las actividades más complejas en la gestión de los sistemas de producción" (p.1005), el problema de programación se centra en cómo asignar los recursos disponibles para realizar una serie de actividades en un determinado periodo de tiempo, con el fin de optimizar uno o más objetivos (Pinedo, 2008), y así poder lograr una eficiente programación de los trabajos y sobrevivir en entornos cada vez más competitivos y de constante cambio en el mercado (Salazar y Sarzuri, 2015).

En este orden de ideas, para determinar la programación de los trabajos es importante conocer el sistema de producción con el que cuente la industria, ya sea *Flow Shop*, *Job Shop, JIT* (Just In Time) o líneas de flujo flexible. En un *Flow Shop*, un conjunto de tareas deben ser procesadas por una serie de etapas en un mismo orden, donde cada estación tiene una sola máquina, que puede manejar como máximo una operación a la vez (Low, 2005). Sin embargo, con el fin de aumentar la productividad, atender demandas crecientes o introducir nuevas tecnologías, diferentes industrias se ven en la necesidad de aumentar la capacidad de las estaciones mediante la compra de máquinas, lo que conduce a la coexistencia de varias máquinas paralelas no relacionadas en ciertas estaciones de trabajo, lo que se conoce como *Flow Shop* Híbrido (Miranda, Pérez y Florence, 2013).

El *Flow Shop* híbrido (HFS, por sus siglas en inglés) es una extensión del sistema de producción *Flow Shop* tradicional, el cual consta de dos o más etapas de procesamiento en serie, en el que al menos una de ellas cuenta con dos o más máquinas paralelas (Figielska, 2014), esta configuración permite representar muchos sistemas productivos existentes como lo es la fabricación de semiconductores, producción de motores de aviones, fabricación de vidrio, así como en la producción de la industria farmacéutica, química, alimentaria, textil y automotriz (Salazar y Sarzuri, 2015) (Rahman, Santosa, y Wiratno, 2014).

El problema del *Flow Shop* híbrido tiene gran importancia tanto en el campo teórico como en la ingeniería (Cui & Gu, 2015), es por esta razón que ha sido estudiado por diversos investigadores, desde que fue propuesto décadas atrás por Arthanari y Ramamurthy (Marichelvam, Prabaharam y Yang, 2014), buscando incluir restricciones con el fin que el problema analice la mayor cantidad de características reales, a las que las diversas industrias se ven enfrentadas hoy en día.

Gupta y Tunc probaron que la programación de un *Flow Shop* híbrido con dos estaciones, donde una de ellas tiene solo una máquina y la otra estación cuenta con máquinas paralelas idénticas, es decir, que el tiempo de procesamiento de un trabajo es igual para cada máquina, es NP-Hard (Yaurima, Burtseva y Tchernykh, 2009). Por lo tanto nace la necesidad de desarrollar enfoques eficaces y eficientes que puedan generar soluciones aproximadas cercanas al óptimo pero con un tiempo computacional considerablemente menor (Jun & Park, 2015).

Por tanto, en este trabajo se desarrolló un algoritmo genético híbrido y un algoritmo híbrido basado en una colonia artificial de abejas, para dar solución al problema del HFS con máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia y *buffers* de tamaño limitado, validándolos mediante 20 instancias en lenguaje de programación Matlab. Los

resultados se compararon con los hallados mediante la metaheurística basada en el algoritmo competitivo imperialista (ICA) y la solución óptima obtenida por un método exacto, utilizando el software GAMS. Los aportes de este proyecto son de relevancia para el grupo de investigación OPALO y la Escuela de Estudios Industriales y Empresariales en la consolidación y el fortalecimiento de la línea de investigación de programación de operaciones.

Tabla 1.

Tabla de cumplimiento de objetivos.

Objetivo	**Numerales relacionados**
Realizar una revisión de la literatura sobre el problema del *Flow Shop* híbrido y métodos de solución que han sido implementados para resolver el problema.	5
Diseñar la estructura del algoritmo genético híbrido de acuerdo a las necesidades del problema del *Flow Shop* híbrido con máquinas paralelas no relacionadas y *buffers* de tamaño limitado.	8
Diseñar la estructura del algoritmo híbrido basado en la Colonia Artificial de Abejas (ABC) de acuerdo a las necesidades del problema del *Flow Shop* híbrido con máquinas paralelas no relacionadas y *buffers* de tamaño limitado.	9
Implementar los algoritmos en el software MATLAB y validarlos a través de diferentes instancias.	11 Apéndice B y E
Comparar las soluciones encontradas mediante los algoritmos propuestos con la solución exacta y los resultados obtenidos mediante el Algoritmo Competitivo Imperialista (ICA).	12
Elaborar un artículo de carácter publicable con los resultados obtenidos en la investigación.	Apéndice K

1. Planteamiento del problema

El problema del *Flow Shop* híbrido, se puede describir como un conjunto de tareas que deben ser procesadas en múltiples estaciones, en donde al menos una de las etapas de procesamiento tiene un número de máquinas paralelas mayor a 1 (Fátima, Perassoli, Santos, Peterson, & Paraíso, 2014; Miranda Lugo, Pérez Martínez, & Florence Teixera Jr, 2013). En el HFS existen dos decisiones claves a tomar, las cuales son: determinar la secuencia en el que los trabajos se procesarán en las diversas estaciones y asignar cada una de las tareas a las máquinas que se encuentran en cada etapa, con el objetivo de minimizar el tiempo de culminación del conjunto de trabajos o *makespan*, ya que la optimización de este criterio también lleva a la reducción del inventario en proceso (WIP), minimizar los desórdenes de la planta por los trabajos no completados (Lopez, Giraldo, & Arango, 2015) y maximizar la utilización de las máquinas (Miranda Lugo, Pérez Martínez, & Florence Teixera Jr, 2013).

Teniendo en cuenta que el *Flow Shop* híbrido es un tipo de sistema productivo utilizado en la producción de motores de aviones, la fabricación de semiconductores, fabricación de vidrio, así como en la producción de la industria alimentaria, textil, farmacéutica, química, y automotriz (Salazar y Sarzuri, 2015; Rahman, Santosa, y Wiratno, 2014), el problema presenta diversas variantes para su planteamiento, ya sea el número de etapas del sistema o la cantidad de máquinas por estación, así como múltiples restricciones con el fin de hacer el problema más semejante a la realidad.

En la figura 6 se presentan diversas variantes y restricciones que se pueden tener en cuenta en un problema HFS. Teniendo en cuenta que P_{hjk} es un parámetro que representa el tiempo de procesamiento de un trabajo h, en una máquina j que se encuentra en la estación k, las máquinas paralelas que existen en las estaciones pueden ser de tres tipos: M.P. idénticas cuando el tiempo de procesamiento de un trabajo es igual en todas las máquinas de una misma estación; M.P. uniformes si existe una relación paramétrica del tiempo de procesamiento de cierto trabajo en las diferentes máquinas de cada estación; y M.P. no relacionadas cuando el tiempo de procesamiento de una tarea es diferente para cada máquina en una estación y además no existe una relación paramétrica entre estos (Lopez, Giraldo, & Arango, 2015). El problema a tratar en el siguiente proyecto, considerará k estaciones con máquinas paralelas no relacionadas en las etapas que exista más de un procesador.

Considerando un parámetro binario A_{jkh}, el cual tomará un valor igual 1 indicando que la máquina j de la estación k puede procesar el trabajo h y 0 en caso contrario, el problema del HFS puede considerar que todas las máquinas de cada estación son capaces de procesar todas las tareas o que no todos los procesadores de cada etapa pueden procesar todos los trabajos; lo cual se conoce en la literatura como elegibilidad de las máquinas. El problema que se desarrollará en el proyecto, considerará que los trabajos pueden ser procesados en todas las máquinas que se presenten en las diversas estaciones.

De igual forma, se tendrán en cuenta tiempos de alistamiento dependientes de la secuencia, es decir, que estos lapsos de tiempo no solo dependerán del trabajo a procesar en la máquina, sino que también tendrá en cuenta la tarea que se está procesando. Así mismo, la preparación de las máquinas solo se llevará a cabo cuando la tarea a procesar y la máquina a utilizar estén disponibles, es decir, será una preparación no anticipatoria en lugar de considerar una preparación anticipada,

en la que la máquina se puede alistar cuando esté disponible, sin necesidad de esperar a que la tarea esté lista para ser procesada.

Otra restricción destacada son los *buffers* o almacenamientos intermedios, los cuales pueden tener una capacidad limitada o ilimitada, pero teniendo en cuenta que en la mayoría de las industrias existe un espacio destinado para el inventario en proceso, el presente trabajo considerará almacenamientos intermedios de tamaño limitado. En dado caso que la máquina de la siguiente estación en la que un trabajo debe ser procesado esté ocupada y el buffer no tiene espacio disponible, la tarea deberá esperar en el procesador de la etapa actual, evitando que esta máquina procese otro trabajo.

En la literatura, con el fin de considerar la mayor cantidad de restricciones que se presentan en un ambiente de programación de la producción, ciertas investigaciones tienen en cuenta el tiempo de transporte de los trabajos entre estaciones, el cual es proporcional al espacio en recorrer entre una estación y otra. Así mismo, con el propósito de que un trabajo sea completado primero que otros, en ciertas investigaciones se considera preferencia de las tareas. Estas dos últimas restricciones han sido poco tratadas en la literatura y no serán consideradas en el problema a tratar en el presente trabajo.

De esta manera, el presente proyecto estudiará el problema del *Flow Shop* híbrido, con k estaciones, máquinas paralelas no relacionadas, *buffers* de tamaño limitado, preparación no anticipatoria de las máquinas y tiempos de alistamiento dependientes de la secuencia, buscando darle solución mediante algoritmos híbridos que involucran heurísticas y metaheurísticas.

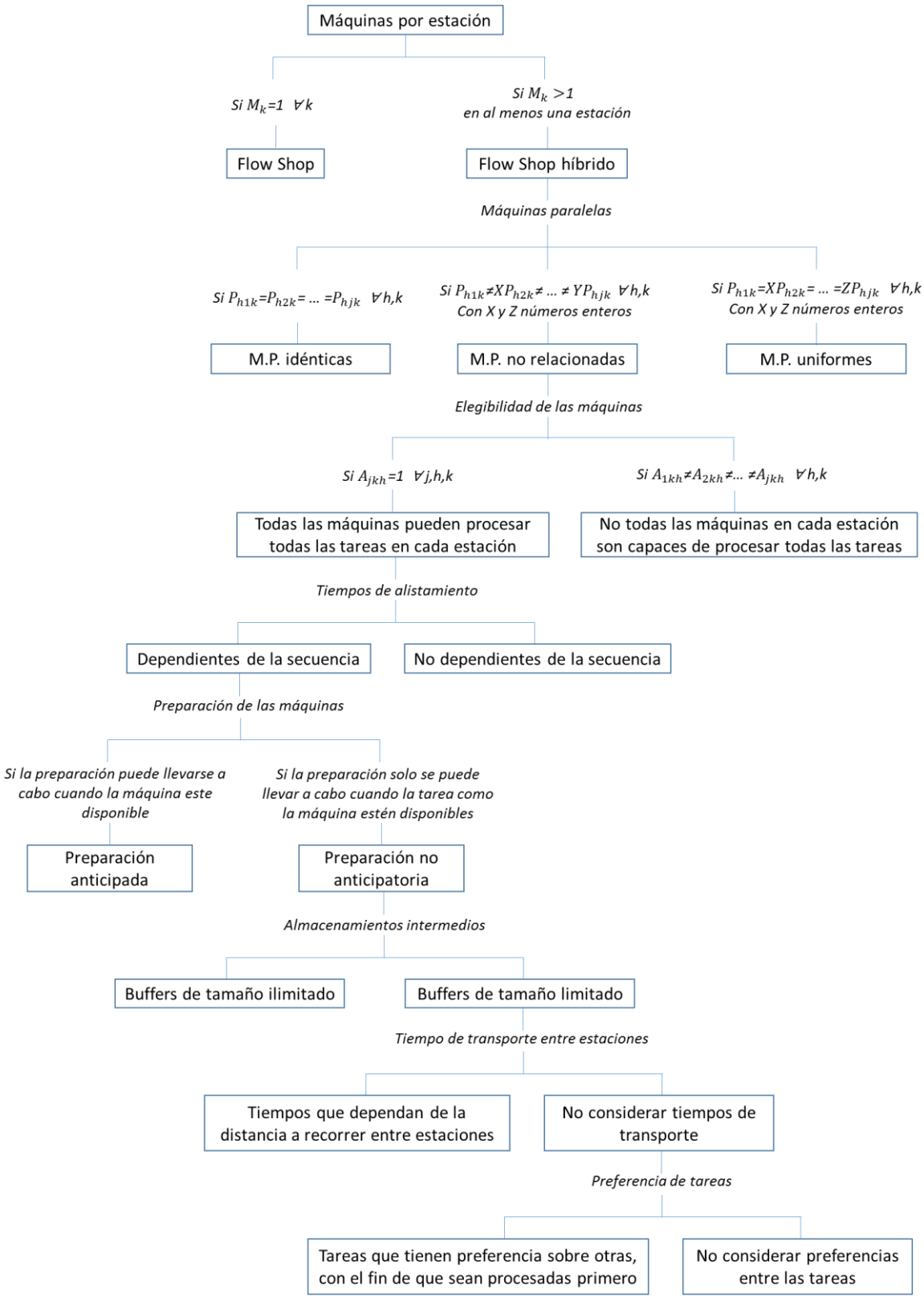


Figura 1. Restricciones del problema del *Flow Shop* híbrido.

2. Justificación del proyecto

La programación de la producción es la encargada de decidir la asignación de recursos a lo largo del tiempo, para realizar un conjunto de trabajos, siendo una actividad de planificación, programación y control de la producción (Fátima, Perassoli, Santos, Peterson, & Paraíso, 2014). Por lo tanto, esta toma importancia en las diversas industrias, buscando obtener una programación óptima o semi-óptima de los trabajos en el proceso de producción, con el objetivo de maximizar la productividad de la planta y así aumentar las ganancias de la compañía (Rabiee , Sadeghi Rad, Mazinani, & Shafaei, 2014).

Debido a que la globalización ha dado lugar a mercados más competitivos, que buscan satisfacer las necesidades de clientes cada vez más exigentes y con constantes cambios culturales, las organizaciones buscan mejorar la flexibilidad de sus procesos, con el fin de adaptarse a los cambios repentinos de la demanda. Dicha flexibilidad, se logra mediante una eficiente programación de las operaciones, que cada día toma mayor relevancia en la gestión de los sistemas de producción y se vuelve más compleja, al tener entornos cada vez más competitivos y de constante cambio en el mercado.

Además, considerando el crecimiento de la tecnología, diversas industrias se ven en la necesidad de adquirir maquinaria que esté a la vanguardia, lo que conlleva a la coexistencia de varias máquinas que realizan una misma operación. De ahí surge el *Flow Shop* híbrido, el cual es un sistema de producción en el que un conjunto de tareas deben ser procesadas en un mismo orden

por múltiples estaciones, en donde al menos una de las etapas cuenta con más de una máquina (Miranda Lugo, Pérez Martínez, & Florence Teixera Jr, 2013).

Teniendo en cuenta que las especificaciones de la nueva maquinaria que se adquiere en las compañías son diferentes a los procesadores actuales, los tiempos de procesamiento de cierta tarea varían entre máquinas de una misma estación, lo que se conoce como máquinas paralelas no relacionadas por estación. Esta restricción en el problema del HFS, hace que no existan algoritmos que puedan encontrar una solución óptima en un tiempo polinómico, es decir, es un problema NP-Hard (Bozorgirad & Logendran, 2012).

Considerando que el problema a tratar en el presente proyecto tendrá en cuenta máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia y buffers de tamaño limitado, se hace necesario plantear métodos heurísticos y metaheurísticos que le den solución al problema, para obtener soluciones de buena calidad.

Por ende, el proyecto diseñará un algoritmo genético híbrido (AG, por sus siglas en ingles), teniendo en cuenta que en la literatura el AG es uno de los métodos más utilizados para este problema y que diversos trabajos similares ha permitido obtener excelentes respuestas. En la literatura investigada, se encontró que Jun y Park (2015) proponen un algoritmo genético híbrido que combina un algoritmo de búsqueda local, con el objetivo de minimizar la tardanza total en un HFS, el algoritmo a diseñar utilizara una búsqueda variable de la vecindad (VNS) para mejorar la solución final que se obtenga de las operaciones de cruce, mutación y selección del algoritmo genético básico.

Por otro parte, el presente proyecto desarrollará un algoritmo híbrido basado en una colonia artificial de abejas, el cual fue utilizado por Cui y Gu (2015) para darle solución a un HFS con

máquinas paralelas idénticas, *buffers* de tamaño ilimitado y sin considerar tiempos de alistamiento de los trabajos, obteniendo mejores soluciones que las encontradas con un AG básico.

Así mismo, teniendo en cuenta que Rabiee, Sadeghi, Mazinani y Shafaei (2014) comprueban que un ICA permite obtener mejores resultados que un AG básico para un problema HFS de dos etapas con máquinas paralelas no relacionadas y tiempos de alistamiento dependientes de la secuencia, y teniendo los resultados obtenidos por el proyecto de pregrado titulado: "Solución al problema del *Flow Shop* híbrido (HFS) con máquinas paralelas no relacionadas y buffers de tamaño limitado mediante una metaheurística basada en el algoritmo competitivo imperialista (ICA)", este proyecto buscará comparar las soluciones que se obtengan mediante un algoritmo genético híbrido, un ICA y un algoritmo híbrido basado en una colonia artificial de abejas, con el objetivo de determinar el método que permita obtener las mejores soluciones a un mismo grupo de instancias.

3. Objetivos

3.1. Objetivo general

Desarrollar dos algoritmos híbridos con el fin de dar solución al problema del "*Flow Shop*" híbrido con máquinas paralelas no relacionadas y "*Buffers*" de tamaño limitado.

3.2. Objetivos específicos

- Realizar una revisión de la literatura sobre el problema del *Flow Shop* híbrido y métodos de solución que han sido implementados para resolver el problema.
- Diseñar la estructura del algoritmo genético híbrido de acuerdo a las necesidades del problema del *Flow Shop* híbrido con máquinas paralelas no relacionadas y *buffers* de tamaño limitado.
- Diseñar la estructura del algoritmo híbrido basado en la Colonia Artificial de Abejas (ABC) de acuerdo a las necesidades del problema del *Flow Shop* híbrido con máquinas paralelas no relacionadas y *buffers* de tamaño limitado.
- Implementar los algoritmos en el software MATLAB y validarlos a través de diferentes instancias.

- Comparar las soluciones encontradas mediante los algoritmos propuestos con la solución exacta y los resultados obtenidos mediante el Algoritmo Competitivo Imperialista (ICA).
- Elaborar un artículo de carácter publicable con los resultados obtenidos en la investigación.

4. Marco teórico

4.1. Optimización

Duarte, Pantrigo y Gallego (2007) definen la optimización como "el proceso de intentar encontrar la mejor solución posible a un problema de optimización, generalmente en un tiempo limitado" (p.1). Se puede encontrar una gran cantidad de problemas de optimización tanto en la ciencia como en la industria, problemas como el diseño de redes de telecomunicación, organización de la producción y re-ingeniería de software (Martí, 2003). Un problema de optimización es aquel en que hay varias posibles soluciones y alguna forma clara de comparación entre ellas, de maniera que éste existe si y solo si se dispone un conjunto de soluciones candidatas diferentes que pueden ser comparadas. Desde el punto de vista matemático un problema de optimización P se puede definir como:

$$P = \begin{cases} opt\ f(x), & Función\ Objetivo \\ & s.a., \\ x \in F \subset SS & Restricciones \end{cases}$$

Donde *f* es la función a optimizar (maximizar o minimizar), *F* es el conjunto de soluciones factibles y *SS* es el espacio de soluciones.

Los problemas de optimización se pueden dividir en dos categorías: aquellos en los que la solución está codificada mediante valores reales y aquellos cuyas soluciones están codificadas por valores enteros. En la segunda categoría se encuentra un tipo particular de problemas los cuales se denominan problemas de optimización combinatoria.

4.1.1. Optimización combinatoria. Un problema de optimización combinatoria consiste en encontrar el máximo (o el mínimo) de una determinada función sobre un conjunto finito de soluciones S, donde las variables han de ser discretas (Martí, 2003). Esta se deriva de la optimización matemática discreta por lo cual es usual encontrar los conceptos de enumeración, combinación y permutación (Yu & Mitsuo, 2010).

Una combinación es una colección desordenada de elementos distintos, un problema de optimización relacionado con la combinación óptima se denomina problema de agrupación. Por otra parte, una permutación es una secuencia que contiene elementos distintos, por lo tanto, a la optimización relacionada con la permutación óptima se le denomina problema de programación. En ese orden de ideas el problema de programación del *Flow Shop* se clasifica como un problema de programación ya que busca la minimización del tiempo de procesamiento total mediante el ajuste de la secuencia de tareas en los puestos de trabajo.

Los problemas de optimización combinatoria presentan como particularidad que siempre existe un algoritmo exacto que permite obtener la solución óptima, estos algoritmos suelen ser ineficientes debido a que el tiempo que emplearía en encontrar una solución crece de forma

exponencial con el tamaño del problema (Duarte Muñoz, Pantrigo Fernández, & Gallego Carrillo, 2007).

4.2. Complejidad computacional

Cruz, Bernal y Peralta (2014) definen la Complejidad como "la cantidad de recursos necesarios para efectuar un cálculo" (p.2), por ende, la teoría de la complejidad computacional, analiza los recursos computacionales requeridos para la resolución de un problema. Estos recursos hacen referencia al tiempo, es decir la cantidad de pasos de ejecución de un algoritmo para resolver un problema, y al espacio, que hace referencia a la cantidad de memoria computacional utilizada para la solución de un problema (Rosenfeld & Irazábal, 2013). Por lo tanto, cuando la solución de un problema requiere más espacio que otro se dice que es más complejo y se llama complejidad espacial; por otra parte, si requiere más tiempo se llama complejidad temporal (Cruz Chávez, Moreno Bernal, & Peralta Abarca, 2014).

Para conocer el grado de complejidad de un problema, se hace uso del modelo computacional de la máquina de Turing, las cuales se presentaron por el matemático ingles Alan Turing en 1936 y hace referencia a un artefacto artesanal que se utiliza para clasificar los problemas de acuerdo al tipo de máquina de Turing que pueda resolverlo (Rosenfeld & Irazábal, 2013). Por tanto, mediante el modelo computacional de Turing se han detectado problemas de los siguientes tipos (Cruz Chávez, Moreno Bernal, & Peralta Abarca, 2014):

- **Problemas NP** (*nondeterministic polynomial time*): problemas que no se pueden resolver en un tiempo razonable cuando el número de variables del problema es una cantidad extremadamente grande. Dentro de este grupo se encuentran dos subconjuntos de problemas:
    - **Problemas P:** Existe una máquina de Turing determinista para resolver este tipo de problemas en un tiempo polinómico, lo que indica que existe un algoritmo determinista que puede darle solución al problema.
    - **Problemas NP- Completos:** No existe una máquina de Turing determinista para resolver el problema, pero se puede encontrar un valor próximo mediante una máquina de Turing no determinista acotando el tiempo polinomial. Por lo general, para resolver este tipo de problemas se utilizan algoritmos heurísticos. Además, un problema es considerado NP-Completo si se puede convertir en otro problema, de tal forma que la solución del segundo se pueda utilizar para resolver el primero (Sanchis de Miguel , Ledezma Espino , Iglesias Martinez , García Jiménez, & Alonso Weber).

Es importante aclarar que los problemas P están incluidos dentro de los problemas NP, debido a que los algoritmos no determinísticos que pueden resolver los problemas NP y NP-Completos, también se pueden utilizar para los problemas P, pero no en caso contrario.

- **Problemas NP-Duros:** Son problemas en donde no existe un algoritmo polinomio que permita verificar la solución, teniendo un nivel de dificultad en su resolución igual o superior que los problemas NP-Completos.

4.3. Métodos de solución.

Diversos métodos de solución se han desarrollado para resolver problemas de programación de trabajos en diferentes ambientes de programación. Estos métodos en tres grandes categorías: métodos exactos, heurísticos y metaheurísticos (Fátima Morais, Godinho Filho, & Perassoli Boiko, 2013).

4.3.1. Métodos exactos. Estos métodos permiten obtener una solución óptima del problema, de acuerdo a la función objetivo establecida. Según MacCarthy y Liu (1993), una solución óptima es: "la solución que entre todas las soluciones factibles, produce el mejor valor de función objetivo (máximo o mínimo)" (p.65). Dentro de los métodos exactos es importante mencionar:

- **Algoritmo de ramificación y acotamiento:** Desarrollado en 1960 por G. Doig. Este método consiste en ir acotando el valor de la función objetivo superior e inferiormente, alcanzando el valor óptimo cuando ambas cotas sean iguales. El método enumera las posibles soluciones del problema, dividiendo el problema original en subproblemas más sencillos. (Solis García, Ríos Mercado, & Alvarez Socarrás).
- **Método Simplex:** Desarrollado en los años cuarenta por el matemático George Dantzing. Este método constituye una forma sistemática y de búsqueda a través de todas las posibles soluciones para obtener la óptima. Aunque se asocia al método gráfico, el método simplex difiere en que tiene como punto de partida el origen siendo esta la solución inicial del problema (Collazo Pedraja).

4.3.2. Métodos heurísticos. El término *heurístico* deriva de la palabra griega *heuriskein* que significa encontrar o descubrir y es usada en el ámbito de la optimización para describir una clase

de algoritmos de resolución de problemas (Martí Cunquero, 2003).Un método heurístico se puede describir como: "Un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución". (Díaz, y otros, 1996).

Martí Cunquero (2003) destaca algunas de las razones por la cuales se utilizan los métodos heurísticos, entre ellas se encuentran:

- El problema es de una naturaleza tal que no se conoce ningún método exacto para su resolución.
- Aunque existe un método exacto para resolver el problema, su uso es computacionalmente muy costoso.
- El método heurístico es más flexible que un método exacto, permitiendo, por ejemplo, la incorporación de condiciones de difícil modelización.
- El método heurístico se utiliza como parte de un procedimiento global que garantiza el óptimo de problema

Se pueden encontrar muchos métodos heurísticos de diferente naturaleza, razón por la cual es complicado dar una clasificación completa, se pueden agrupar en dos categorías (Duarte Muñoz, Pantrigo Fernández, & Gallego Carrillo, 2007): métodos constructivos y métodos de búsqueda.

4.3.2.1. Métodos constructivos. Son procedimientos iterativos que, en cada paso, añaden un elemento hasta completar una solución. Usualmente son métodos deterministas y están basados en seleccionar, en cada iteración, el elemento con mejor evaluación; algunos algoritmos constructivos son:

- Algoritmos de reducción, identifican propiedades o características específicas que contienen en común las soluciones para convertirlas en restricciones del problema con el propósito de limitar el espacio de soluciones y de esta forma reducir el tamaño del problema.

- Algoritmo de descomposición, este algoritmo con el fin de simplificar el proceso de solución, toma el problema principal y lo divide en una serie de sub-problemas más sencillos de resolver, dicha división se realiza de manera sucesiva hasta tener un sub-problema con solución trivial, finalmente, las soluciones obtenidas de los sub-problemas son combinadas hasta obtener una solución al problema original.

- Algoritmo de manipulación del modelo, busca relajar el problema original y usan la solución del modelo relajado para ayudar a encontrar una buena solución al problema original, en este algoritmo se puede linealizar, agrupar variables, eliminar o adicionar restricciones a conveniencia.

- Algoritmo voraz, también conocido como algoritmo ávido, algoritmo miope o estrategia voraz, este algoritmo construye paso a paso una solución factible a partir de una semilla y finaliza cuando no existe una solución vecina de mejor calidad.

4.3.3. Métodos metaheurísticos. Con el propósito de obtener mejores resultados que aquellos alcanzados por los métodos heurísticos, en los últimos años han aparecido una serie de métodos bajo el nombre de metaheurísticas. El término metaheurística apareció por primera vez en el artículo sobre búsqueda tabú de Fred Glover en 1986, y a partir de ahí se han presentado muchas

propuestas de pautas para diseñar los mejores procedimientos de solución de problemas combinatorios (Riojas Cañari, 2005).

Osman y Kelly (1995) definen las metaheurísticas como: "Métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son efectivos". Debido a que no hay características en común entre todas las metaheurísticas, no hay una clasificación formal de ellas, teniendo en cuenta qué tipo de heurística comprende la metaheurística, se presenta la siguiente clasificación:

4.3.3.1. Metaheurísticas de relajación. Una relajación de un problema es un modelo simplificado obtenido al eliminar, debilitar o modificar restricciones (u objetivos) del problema real. Las metaheurísticas de relajación se refieren al diseño, tanto de procedimientos que utilizan formulaciones relajadas del problema para proponer sus soluciones, como soluciones del problema, como de procedimientos que usan dichas relajaciones para guiar las operaciones realizadas para su resolución. Entre las metaheurísticas de relajación se encuentran los métodos de relajación langrangiana o de restricciones subordinadas (Brito Santana, y otros, 2004).

4.3.3.2. Metaheurísticas constructivas. Las metaheurísticas constructivas se orientan a los procedimientos que tratan de la obtención de una solución a partir del análisis y selección paulatina de los componentes que la forman, estas metaheurísticas establecen estrategias para seleccionar los componentes con los cuales se construirá una buena solución. A continuación, se describen algunas de las metaheurísticas que destacan en esta categoría.

4.3.3.2.1. GRASP (Greedy Randomized Adaptative Search Procedures). Desarrollado por Feo y Rezende en 1995, es un método multi arranque en el que cada arranque corresponde a una iteración, en donde cada iteración tiene dos fases: la fase de construcción y la fase de mejora. En

la fase de construcción para poder obtener una solución inicial, se aplica una heurística constructiva, para posteriormente se mejorada en la segunda fase mediante un algoritmo de búsqueda local (Riojas Cañari, 2005).

4.3.3.2.2. Algoritmo Iterated Greedy (IG). Es un método de búsqueda local que genera una secuencia de soluciones por iteración sobre heurísticas constructivas codiciosas usando dos fases principales: destrucción y construcción. Durante la fase de destrucción *d* componentes de una solución candidata son eliminados, y en la fase de construcción estos elementos se incorporan en nuevas posiciones aleatoriamente, con el fin de reconstruir una nueva solución completa, la cual puede reemplazar a la solución candidata dependiendo de los criterios de aceptación (Abdollahpour & Rezaeian , 2015).

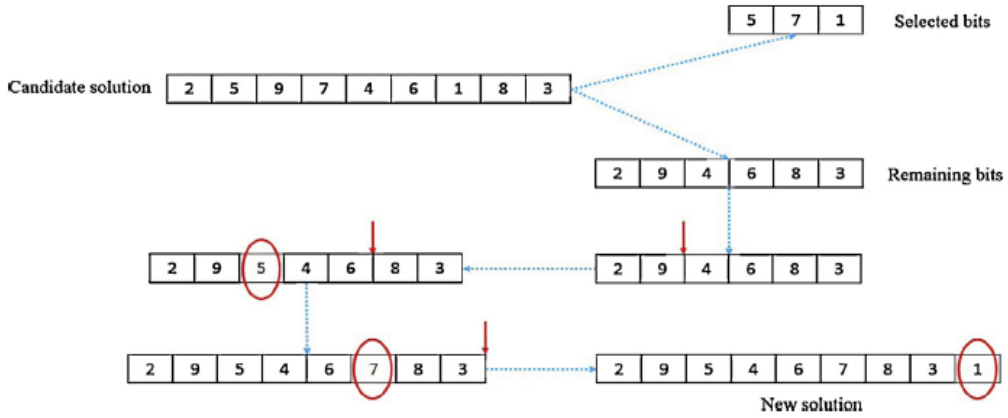


Figura 2. Ejemplo de las fases de destrucción y construcción de un Algoritmo Iterated Greedy. Adaptado de Sana Abdollahpour & Javad Rezaeian (2015). Minimizing makespan for flow shop scheduling problem with intermediate buffers by using hybrid approach of artificial immune system.

Considérese la solución candidata que se muestra en la figura 10, la cual corresponde a una secuencia de 9 tareas a realizar. Con un parámetro de destrucción igual a 3 (*d*), se seleccionan las

tareas 5,7 y 1 de manera aleatoria para destruir la solución candidata. Posteriormente se escogen diversas posiciones de manera aleatoria para incluir las tareas seleccionadas en la fase de destrucción y así obtener una nueva solución, como se muestra en la figura 10.

4.3.3.3. Metaheurísticas de búsqueda. Las metaheurísticas de búsqueda son métodos que presupone que existe una solución y realizan procedimientos de búsqueda, para ello establecen estrategias para recorrer el espacio de soluciones del problema transformando de forma iterativa la solución de partida, uno de los riesgos al usar un algoritmo de búsqueda es el de alcanzar un óptimo local del que ya no sea posible salir. Las metaheurísticas de búsqueda global incorporan tres pautas básicas para de escapar de los óptimos locales: volver a iniciar la búsqueda desde otra solución de arranque, modificar la estructura de entornos que se está aplicando y permitir movimientos o transformaciones de la solución de búsqueda que no sean de mejora (Brito Santana, y otros, 2004). Algunos ejemplos de metaheurísticas de búsqueda se detallan a continuación.

4.3.3.3.1. Metaheurísticas de arranque múltiple. Este método con el fin de evitar quedar atrapado en óptimos locales, establece pautas para reiniciar de forma inteligente las búsquedas descendentes, permitiendo de esta manera la diversificación de la estrategia de búsqueda. Este es un algoritmo iterativo, en el que cada iteración tiene dos fases: la primera en la que se genera una solución y la segunda en la que la solución es típicamente, pero no necesariamente, mejorada. Cada iteración global produce una solución, usualmente un óptimo global, y la mejor de todas es la salida del algoritmo (Martí, 2003).

4.3.3.3.2. Búsqueda Tabú (Tabú Search). Desarrollada por Glover en 1986, guía un procedimiento de búsqueda local para explorar el espacio de soluciones más allá del óptimo local (Riojas Cañari, 2005). Su característica distintiva es el uso de memoria adaptativa y de estrategias

especiales de resolución de problemas. El marco de memoria adaptativa no sólo explota la historia del proceso de resolución del problema, sino que también exige la creación de estructuras para hacer posible tal explotación (Glover & Melián Batista, 2003).

La búsqueda tabú considera el salto a la mejor solución de una vecindad, incluso cuando ésta tenga un costo algo peor que la solución actual, lo que permite la opción de salir de óptimos locales en favor de otros mejores. La nueva vecindad denominada "vecindad legal", está definida por una relación de vecindad de una solución y por una lista de movimientos prohibidos que recuerda los últimos movimientos realizados para evitar que se repitan soluciones anteriores y permitir intensificar o diversificar la búsqueda. El algoritmo se detiene cuando la solución no puede ser mejorada o cuando se cumple el criterio de parada

4.3.3.3.3. Recocido Simulado (Simulated Annealing). Introducido por Kirpatrick, Gellat y Vecchi en 1982, el recocido simulado es un algoritmo de aproximación a la solución óptima, fundado en una analogía del comportamiento de sistemas termodinámicos simples (Vázquez Espí, 1994). Este es un algoritmo aleatorizado de búsqueda local por vecindades, el método acepta de forma inmediata los movimientos que mejoren la función objetivo, mientras que aquellos que desmejoren la función, tienen una probabilidad de ser aceptados, la cual depende de un parámetro controlable llamado temperatura, que disminuye a medida que avanzan las iteraciones.

4.3.3.3.4. Búsqueda de entorno variable (Variable Neighborhood Search, VNS). Metaheurística reciente propuesta por P. Hansen y N. Mladenovic, para resolver problemas de optimización, basada en cambios sistemáticos de estructuras de vecindad, esta idea ayuda a escapar de óptimos locales y explorar otras zonas del espacio de soluciones (Duarte Muñoz, Pantrigo Fernández, & Gallego Carrillo, 2007). Este algoritmo se basa en tres puntos (Mladenovic & Hansen, 1997):

1. Un óptimo local con respecto a una vecindad $N_i(x)$ no tiene por qué serlo con respecto a otra vecindad $N_j(x)$.

2. Un óptimo local con respecto a todas las posibles estructuras de vecindad.

3. Para muchos problemas, los óptimos locales con respecto a una o varias estructuras de vecindad, están relativamente próximos.

Existen varias categorías del algoritmo VNS, que dependen de la forma en la que se combinan los diferentes factores, como lo son: búsqueda del entorno variable descendente, búsqueda del entorno variable reducida (*Reduced Variable Neighborhood Search, RVNS*), búsqueda del entorno variable básica (*Basic Variable Neighborhood Search, BVNS*) y búsqueda de entorno variable general (*General Variable Neighborhood Search, GVNS*).

Además, Se han propuesto en la literatura diversas formas de extender la VNS con el fin de dotarlo de algunas características adicionales, algunas de ellas son: Búsqueda de entorno variable con descomposición (*Variable Neighborhood Decomposition Search, VNDS*) que extiende la VNS en un esquema de entorno variable en dos niveles basado en la descomposición del problema; y la búsqueda de entorno variable sesgada (*Skewed Variable Neighborhood Search, SVNS*), afronta la exploración de valles alejados de la solución actual (Hansen, Mladenovic, & Moreno Pérez, 2003).

A continuación se describe la búsqueda del entorno variable descendente (Hansen, Mladenovic, & Moreno Pérez, 2003):

Búsqueda del entorno variable descendente (*Variable Neighborhood Descent, VND*), Este algoritmo se basa en el punto número 1 mencionado anteriormente y heurísticas de descenso (*steepest descent heuristic*), también conocida como búsqueda local de mejoramiento, esta

heurística consiste en reemplazar iterativamente la solución actual por el resultado de la búsqueda local, mientras se produzca mejora. En la búsqueda del entorno variable descendente el cambio de vecindad se realiza de forma determinista, tal que, si el proceso de búsqueda queda atrapado en un óptimo local, se realiza un cambio en la vecindad a fin de escapar dicho óptimo. A continuación se enuncian los pasos básicos que se deben seguir (Hansen, Mladenovic, & Moreno Pérez, 2003):

Inicialización

Seleccionar el conjunto de estructuras de entornos $N_k, k = 1, \dots, k_{max}$, que se usarán en el descenso; encontrar una solución inicial x.

Iteraciones

Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

1. Hacer $k \leftarrow 1$;

2. Repetir, hasta que $k = k_{max}$, los pasos:

    a. Exploración del entorno Encontrar la mejor solución x' del k-ésimo entorno de x $(x' \in N_k(x))$;
    b. Moverse o no Si la solución obtenida x' es mejor que x, hacer $x \leftarrow x'$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$

4.3.3.4. Metaheurísticas evolutivas. Las metaheurísticas evolutivas establecen estrategias para conducir la evolución en el espacio de búsqueda de conjunto de soluciones (usualmente llamados poblaciones) con la intención de acercarse a la solución óptima con sus elementos. Las diferentes metaheurísticas evolutivas se distinguen por la forma en que combinan la información

proporcionada por los elementos de la población para hacerla evolucionar mediante la obtención de nuevas soluciones (Brito Santana, y otros, 2004).

4.3.3.4.1. Búsqueda dispersa (Scatter Search). Método introducido en 1977 en el cual se realiza una exploración sistemática sobre una serie de buenas soluciones llamadas conjunto de referencia teniendo en cuenta las características de diversos elementos de solución. Este método se basa en el principio de que la información sobre la calidad o el atractivo de un conjunto de reglas, restricciones o soluciones puede ser utilizado mediante la combinación de estas. En concreto, dadas dos soluciones, se puede obtener una nueva mediante su combinación de modo que mejore a las que la originaron (Casado & Martí, 2007).

4.3.3.4.2. Algoritmos genéticos. Son métodos adaptativos que pueden utilizarse para resolver problemas de búsqueda y optimización. Sus principios fueron establecidos por Holland en 1975, y se basan en los postulados de Darwin sobre el proceso genético de los organismos, esta técnica emula la evolución natural para explorar con eficiencia el espacio de búsqueda con el supuesto de que unos individuos con ciertas características son aptos para sobrevivir y transmiten estas características a su descendencia (Sait & Youssef, 1999). Según Goldberg (1989) los algoritmos genéticos son:

> "Algoritmos de búsqueda basados en los mecanismos de selección natural y genética natural. Combinan la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya aleatorizada, intercambiada para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana" (p.50).

Mitchell (1998) describe la forma en que trabaja un algoritmo genético simple en 5 pasos:

1. Comenzar con una población P generada aleatoriamente de n cromosomas de l bits.

2. Calcular la capacidad $f(x)$ para cada cromosoma x de P.

3. Repetir los siguientes pasos hasta que se hayan creado n descendientes:

a. Seleccionar un par de cromosomas padre de P, siendo la probabilidad de selección una función creciente de la capacidad. La selección se realiza "con reemplazo", es decir, que el mismo cromosoma puede ser seleccionado en más de una ocasión para ser padre.

b. Con probabilidad p_c (probabilidad de cruce, o tasa de cruce), cruzar el par en un punto elegido aleatoriamente (con probabilidad uniforme) para formar dos descendientes. Si no tiene lugar ningún cruce, formar dos descendientes que sean copias exactas de sus respectivos padres.

c. Mutar los dos descendientes en cada lugar con probabilidad p_m (probabilidad de mutación, o tasa de mutación), y colocar los cromosomas resultantes en la nueva población P'. Si n es impar, se puede rechazar aleatoriamente a un miembro de la nueva población.

```
BEGIN /* Algoritmo Genetico Simple */
      Generar una poblacion inicial.
      Computar la funcion de evaluacion de cada individuo.
      WHILE NOT Terminado DO
      BEGIN /* Producir nueva generacion */
            FOR Tamaño poblacion/2 DO
            BEGIN /*Ciclo Reproductivo */
                  Seleccionar dos individuos de la anterior generacion,
                  para el cruce (probabilidad de seleccion proporcional
                  a la funcion de evaluacion del individuo).
                  Cruzar con cierta probabilidad los dos
                  individuos obteniendo dos descendientes.
                  Mutar los dos descendientes con cierta probabilidad.
                  Computar la funcion de evaluacion de los dos
                  descendientes mutados.
                  Insertar los dos descendientes mutados en la nueva generacion.
            END
            IF    la poblacion ha convergido THEN
                  Terminado := TRUE
      END
END
```

Figura 3. Pseudocódigo del algoritmo genético simple. Adaptado de Moujahid, Inza y Larrañaga, Algoritmos genéticos.

4. Reemplazar la población actual P con la nueva P'.

5. Volver al paso 2.

La estructura básica de un algoritmo genético está compuesta por los siguientes elementos (Mitchell, 1999): Codificación, población inicial, función de aptitud o *fitness,* operadores genéticos, y criterio de parada.

a. Codificación

La forma de implementar la codificación depende de la naturaleza de las variables de decisión del problema o de la representación de una configuración (Gallego Rendón, Escobar Zuluaga, & Toro Ocampo, 2015). La codificación se puede realizar de varias formas, la más utilizada es mediante una cadena de números binarios, pero también se pueden utilizar números enteros, incluso cadenas de palabras (Arranz de la Peña & Parra Truyol, 2007) .

- Codificación binaria: Cada cromosoma es una cadena de bits (0 o 1).
- Codificación numérica: Se utilizan cadenas de números que representan un número en una secuencia, por lo general se utiliza en problemas en los que hay que ordenar algo.
- Codificación por valor directo: Cada cromosoma es una cadena de valores relacionados con el problema a estudiar, pudiendo ser desde números decimales, cadenas de caracteres o incluso una combinación de varios de ellos. Normalmente es necesario desarrollar nuevas técnicas de reproducción y mutación específicas hacia la resolución del problema.

- Codificación en árbol: Cada cromosoma es un árbol con ciertos objetos. Este tipo de codificación se utiliza principalmente en el desarrollo de programas o expresiones para programación genéticas.

b. Población inicial

Se conoce como población a un conjunto finito de cromosomas, los cuales representan un individuo o solución, donde cada elemento en la cadena se conoce como gen, el cual codifica un elemento particular de la solución (César Vélez & Montoya, 2007).

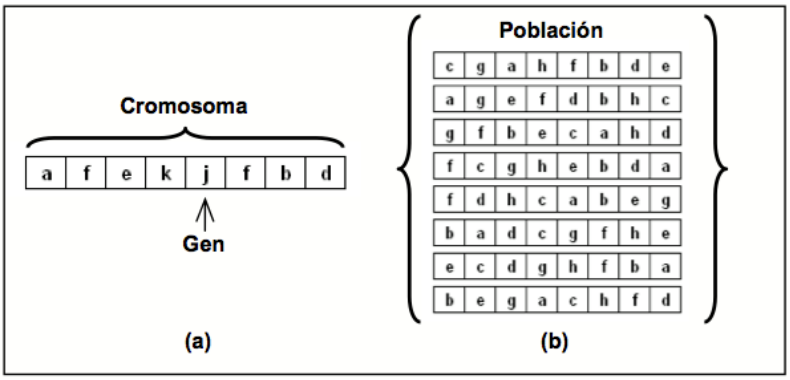


Figura 4. Representación de la población de un algoritmo genético. Adaptado de Vélez y Montoya, Metaheurísticos: una alternativa para la solución de problemas combinatorios en administración de operaciones.

El algoritmo genético puede generar la población inicial ya sea a través de métodos determinísticos o de manera aleatoria. En cuanto a su tamaño, se debe tener en cuenta que si este es muy pequeño, el algoritmo podría converger rápidamente, ya que tendrá pocas posibilidades de realizar reproducciones, realizando de esta manera una búsqueda escasa y poco óptima, de lo contrario, si la población es muy grande, el algoritmo será excesivamente lento, desperdiciándose recursos computacionales (Arranz de la Peña & Parra Truyol, 2007).

c. Función de aptitud o *fitness*

La función *fitness* determina la capacidad de un individuo de sobrevivencia y de reproducir descendencia (Sanhueza H., Hernisch V., Díaz R., & Guirrima C, 1999). A cada individuo se le es asignado un valor de aptitud o *fitness*, este valor se basa en cuán lejos está el individuo de la solución, es decir, de que tan bien el cromosoma resuelve el problema planteado, a mayor valor, mejor será la solución que contiene (Mitchell, 1999).

d. Operadores genéticos

La forma más simple del algoritmo genético implica tres tipos de operadores: selección, cruce y mutación (Mitchell, 1999) :

1. Selección o reproducción

La selección es una parte fundamental en el funcionamiento del algoritmo genético, este operador escoge cromosomas entre la población para efectuar la reproducción. La selección se realiza de forma probabilística, de modo que cuanto más capaz sea el cromosoma, más veces será seleccionado para reproducirse, sin embargo, los individuos menos adaptados también deben tener probabilidad de reproducirse, puesto que su material genético puede ser útil para encontrar buenas soluciones (Cervantes Posada, 2010). La selección se puede realizar de varias formas (Arranz de la Peña & Parra Truyol, 2007):

- Selección por ruleta: En este método los padres se seleccionan de acuerdo con su aptitud, los mejores individuos son los que tienen mayores posibilidades de ser elegidos, siendo la probabilidad de ser seleccionados proporcional a su aptitud.

- Selección por rango: Los individuos son ordenados en un ranking de acuerdo a su *fitness*, su selección se basa en el puesto ocupado en dicho ranking.
- Selección elitista: Copia el mejor cromosoma o alguno de los mejores en la nueva población evitando que estos se pierdan tras la aplicación de los operadores de cruce y mutación.
- Selección por estado estacionario: La descendencia de los individuos seleccionados en cada generación vuelve a la población genética preexistente remplazando a algunos de los miembros menos aptos de la anterior generación.
- Selección por torneo: En este métodos se comparan p individuos escogidos al azar y se selecciona el que tenga mejor aptitud. El que tenga mayor puntuación se reproduce y su descendencia sustituye al que tiene menor puntuación.
- Selección jerárquica: En esta selección, los individuos atraviesan múltiples rondas de selección en cada generación; las evaluaciones en los primeros niveles son más rápidas y menos discriminatorias, mientras que los que sobreviven hasta los niveles más altos son evaluados más rigorosamente.

Existen otras técnicas de selección, entre ellas la selección por prueba de aptitud en la que los cromosomas con más aptitud tienen más posibilidad de ser seleccionados pero no la certeza, también se encuentra la selección generacional, en la que ningún miembro de la población anterior se encuentra en la nueva.

2. Cruce (*Crossover*)

El cruce consiste en el intercambio de material genético entre dos cromosomas para así crear una nueva descendencia. Este intercambio genético se puede llevar a cabo de muchas formas, algunos operadores de cruce son (Cervantes Posada, 2010):

- Cruce basado en 1 Punto: En este cruce los dos individuos seleccionados para ser padres, son recombinados por medio de la selección de un punto de corte, para posteriormente intercambiar las secciones que se encuentran a la derecha de dicho punto.

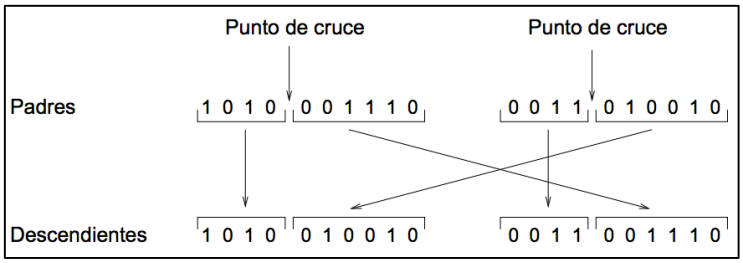


Figura 5. Operador de cruce basado en un punto. Adaptado de Moujahid, Inza y Larrañaga, Algoritmos genéticos.

- Cruce basado en 2 puntos: En este caso los padres se cortan en dos puntos, generando de esta manera tres secciones en cada uno de los cromosomas. El modo más común de realizar este cruce consiste en que el primer hijo hereda directamente los genes del primer y último intervalo, los genes faltantes en el intervalo dos se heredan del segundo padre, finalmente, el segundo hijo se genera de manera análoga.

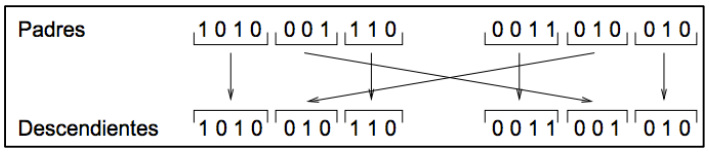


Figura 6. Operador de cruce basado en dos puntos. Adaptado de Moujahid, Inza y Larrañaga, Algoritmos genéticos.

- Cruce uniforme: cada gen en la descendencia se crea copiando el correspondiente gen de uno de los padres, escogido de acuerdo a una "máscara de cruce" generada aleatoriamente. Cuando existe un 1 en la "máscara de cruce", el gen del primer padre es copiado, de lo contrario el gen se copiará del segundo padre.

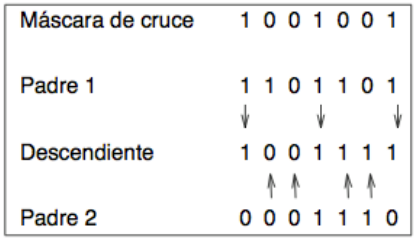


Figura 7. Operador de cruce uniforme. Adaptado de Moujahid, Inza y Larrañaga, Algoritmos genéticos.

3. Mutación

Este operador produce variaciones de modo aleatorio en un cromosoma, mediante la modificación de ciertos genes atendiendo a la probabilidad de mutación establecida anteriormente (Arranz de la Peña & Parra Truyol, 2007). Las mutaciones son beneficiosas pues contribuyen a la diversidad genética, además previenen a las soluciones de la población verse limitadas por un óptimo local los principales operadores de mutación son (Cervantes Posada, 2010):

- *Flip Bit*: Este operador es utilizado únicamente cuando cada gen puede tomar solo dos valores, con codificación binaria, este operador cambia el valor de 0 a 1 y viceversa.
- Inserción: Escoge aleatoriamente una actividad, a continuación la extrae e inserta en un lugar seleccionado al azar.
- Intercambio entre actividades consecutivas: Este operador intercambia una actividad con la siguiente si no tienen ninguna relación de precedencia.

- Movimiento a la derecha: Se intercambia una actividad con una que esté a su derecha.
- Intercambio: Consiste en intercambiar dos actividades cualquiera dentro de la lista de actividades.
- Mutación aleatoria: La lista de actividades se modifica según una probabilidad de P_{mut} o de manera aleatoria.

e. Criterio de parada

Los algoritmos de genéticos son métodos de búsqueda estocásticos, que en principio podrían funcionar para siempre, por lo que en la práctica se necesita un criterio de parada (Reeves, 2003). Existen diferentes criterios de parada, estos pueden ser estáticos, aquellos que han sido especificados previamente, o dinámicos, es decir, que el criterio de parada depende de la evolución del proceso, entre ellos se destacan (Gallego Rendón, Escobar Zuluaga, & Toro Ocampo, 2015):

- Un número específico de generaciones.
- El valor de aptitud de una solución alcanza un valor previamente establecido.
- La población es demasiado homogénea, es decir, las configuraciones son similares y no existe más evolución.
- Determinado tiempo computacional.

4.3.3.5. Metaheurísticas basadas en inteligencia de enjambre. Estas metaheurísticas "están basadas en el estudio del comportamiento colectivo en sistemas autoorganizados y descentralizados" (Muñoz, López, & Caicedo, 2008, p.130). Dentro de esta rama se encuentran las siguientes técnicas: Optimización por enjambre de partículas, Optimización por colonia de hormigas, la búsqueda por difución estocástica, Optimización por emjambre de bacterias, Algoritmo de colonia de abejas artificiales, entro otros.

4.3.3.5.1. Algoritmo de colonia de abejas artificiales. Fue propuesto por Dervis Karaboga en 2005 y esta basado en el comportamiento de los emjambre de abejas cuando buscan comida (Delgado-Osuna, Lozano, & García Martínez, 2015). Esta metaheurística basada en el modelo bioogico de las abejas cosnta de los siguientes elementos (Aguilar Justo, 2014):

- Fuentes de alimento: representan soluciones al problema tratado y tienen un valor númerico que indica su potencial.
- Abejas empleadas: Son las encargadas de explotar las fuentes de alimento y compartir información sobre la calidad de las fuentes a las abejas observadoras.
- Abejas desempleadas: Se encuentran buscando fuentes de alimento para explotar. Se dividen en dos tipos:
    - Abejas exploradoras: Buscan nuevas fuentes de alimento.
    - Abejas obervadoras: Esperan en la colmena para elegir alguna fuente de alimento que se encuentra en explotación por las abejas empleadas.

Los tres elementos se relacionan de la siguiente manera: Las abejas empleadas comunican la información de la fuente de alimento que están explotando a las observadoras por medio de una danza, cuya duración indica la concentración de néctar de la fuente de alimento por lo que son , el ángulo con respecto al sol la dirección de la fuente y el número de movimientos zig-zag durante la danza representa la distancia; las fuentes mas rentables y mas probables a ser elegidas por las observadoras, son aquellas donde la duración de la danza de las empleadas sea mayor. Cuando las fuentes de alimento se han agotado, tanto por las abejas empleadas y observadoras, son abandonadas y reemplazadas por nuevas fuentes encontradas por las abejas exploradoras.

Una de las ventajas del algoritmo es que requiere de pocos parámetros, los cuales son:

- **Número de soluciones:** Cantidad de soluciones iniciales (fuentes de alimento) para el problema.
- **Número de ciclos:** Número de iteraciones del algoritmo o tiempo de ejecución.
- **Limite:** Número de ciclos que se explota una fuente de alimento antes de ser abandonada.

Con los parámetros establecidos el paso a paso del algoritmo es el siguiente (Herrero Giner, 2014):

- Se comienza obteniendo y evaluando un número de soluciones iniciales
- Luego se explotan las fuentes de alimento, lo que implica modificar las soluciones iniciales obteniendo nuevas soluciones, buscando que estas tengan un mayor potencial.
- Despues de cierto número de ciclos sin mejorar la solución, se considera que la fuente de alimento se ha agotado, por lo que se guarda la solución encontrada y se abandona la fuente de alimento.
- Por último, las abejas exploradoras buscan nuevas fuentes de alimento, las cuales serán explotadas, repitiendo los pasos anteriores durante el número de iteraciones definidas.

4.4. Programación de operaciones.

La programación de operaciones o *scheduling* es un proceso de toma de decisiones que puede definirse como la asignación de recursos disponibles y actividades de producción para llevar a cabo determinadas tareas de manera eficiente (Jungwattanakit, Reodecha, Chaovalitwongse, & Werner, 2007). Según Naderi, Tavakkoli-Moghaddam y Khalili (2010) los objetivos considerados

en la programación de las operaciones pueden clasificarse en tres grupos: utilización eficiente de los recursos, respuesta rápida a las demandas y conformidad con los plazos establecidos.

Así mismo, Heizer y Render plantean que existen tres razones por las que el *scheduling* tiene importancia de tipo estratégico en las industrias: permite el movimiento más rápido de bienes y servicios a través de las instalaciones; ayuda en el cumplimiento de compromisos y a realizar entregas fiables; y favorece la capacidad adicional, facturación más rápida y relativa flexibilidad, lo que implica un mejor servicio al cliente.

4.4.1. Tipos de problemas de Scheduling. Teniendo en cuenta que la programación tiene una amplia área de aplicación, pues casi todos los proveedores de servicios y bienes experimentan un tipo de problema de programación. Por ejemplo, en un colegio se debe decidir sobre el horario de clases, en un aeropuerto se deben programar los aterrizajes y despegues de los aviones o en una fábrica de muebles se deben programar las operaciones para producir un sofá (Yagmahan & Yenisey, 2009). Por ende, existen varios tipos de problemas de *Scheduling*, la cual depende de la manera como los centros de trabajo, máquinas y operarios se organicen para procesar las tareas, es decir, del sistema productivo de la compañía. Los tipos son:

- ***Project Scheduling:*** Se encarga de la secuenciación de actividades sujetas a restricciones de precedencia y asignación de recursos a las actividades de un proyecto.
- ***Single Machine Scheduling:*** Se refiere a la secuencia de varios trabajos en una sola máquina.
- ***Parallel Machines Scheduling:*** Es una generalización del problema de programación en una sola máquina. En este tipo de *Scheduling* existen múltiples máquinas por estación, en las que se pueden programar los diversos trabajos. Las máquinas paralelas en cada etapa

pueden ser: idénticas cuando el tiempo de procesamiento de un trabajo es igual en todas las máquinas de una misma estación, uniformes si existe una relación paramétrica del tiempo de procesamiento de cierto trabajo en las diferentes máquinas de cada estación y no relacionadas cuando el tiempo de procesamiento de una tarea es diferente para cada máquina en una estación y además no existe una relación paramétrica entre estos (Lopez, Giraldo, & Arango, 2015).

- ***Shop Scheduling:*** Se puede describir como un conjunto de n trabajos y m máquinas, en el que cada trabajo consiste en un conjunto de operaciones, y cada una de estas debe realizar en una máquina. Este se divide en las siguientes categorías:
    - ***Flow Shop Scheduling:*** La principal característica es que el flujo de trabajos es unidireccional, es decir, todos los trabajos tienen el mismo orden de procesamiento a través de las máquinas que están dispuestas de forma lineal. Por lo tanto, las máquinas pueden numerarse 1, 2, … , m y las operaciones de un trabajo tienen números correspondientes (i,1), (i,2), ... , (i, m).
    - ***Job Shop Scheduling:*** Es una generalización *Flow Shop Scheduling* y la principal diferencia con es que el flujo de los trabajos no es unidireccional. Además, las restricciones de precedencia se definen entre las operaciones de cada trabajo.
    - ***Open Shop Scheduling:*** Es una generalización *Flow Shop Scheduling* en el que no hay ninguna relación de precedencia entre las operaciones de los trabajos. Además, el orden de procesamiento no tiene importancia.

4.5. Flow Shop híbrido.

El *Flow Shop* híbrido, también conocido como *Flow Shop* flexible, *Flow Shop* con múltiples procesadores o *Flow Shop* compuesto (Chen & Chen, 2009), es una extensión del sistema de producción *Flow Shop* tradicional que incorpora todas las dificultades de su predecesor (*Flow Shop*) junto con la necesidad adicional de determinar la asignación de operaciones a las máquinas en cada etapa, puesto que se consideran estaciones con varias máquinas en paralelo (Rashidi, Jahandar, & Zandich, 2010).

En un *Flow Shop* híbrido un conjunto de tareas debe ser procesado en una serie de *k* etapas, cumpliendo las siguientes condiciones:

- Todas las tareas deben pasar por las estaciones de procesamiento bajo un mismo orden.
- Existe la posibilidad de que un trabajo pueda omitir cualquier número de estaciones de trabajo, siempre y cuando sea procesado en por lo menos una de las etapas globales.
- El número de etapas de procesamiento *k* es al menos 2.
- Cada una de las etapas de procesamiento, tiene *k(j)* procesadores o máquinas en paralelo y por lo menos una de las etapas debe tener $k(j)>1$ máquinas.

4.5.1. Buffers limitados. Los *buffers* o almacenamientos intermedios, es una restricción de los problemas de *Flow Shop*, son zonas donde los trabajos son transferidos en caso que la siguiente máquina en la secuencia esté ocupada, según el problema que se quiera trabajar, estas zonas pueden ser consideradas de capacidad limitada o ilimitada. (Rashidi, Jahandar, & Zandich, 2010).

Los *buffers* son zonas de almacenamiento transitorias entre dos etapas de procesamiento consecutivas en una línea de producción, su tarea es albergar los trabajos en proceso durante un

tiempo limitado para luego ser transportados a la siguiente estación. Con el fin de incluir esta restricción en el problema de programación, muchos autores asumen los *buffers* como una etapa de procesamiento más, con máquinas especiales cuyo tiempo de alistamiento y de procesamiento es igual a cero para cualquier trabajo en cualquier secuencia (Miranda Lugo, Pérez Martínez, & Florence Teixera Jr, 2013).

Una vez un trabajo salga de una etapa y la siguiente se encuentre ocupada, este se debe almacenar en un *buffer*, de tal manera que cuando se libere una máquina en dicha etapa, el trabajo saldrá inmediatamente hacia ese procesador. Generalmente, se asume FIFO (por sus iniciales en inglés) como regla de salida de los trabajos, es decir, los primeros trabajos en ser almacenados son los primeros en salir a la siguiente estación (Almeder & Hartl, 2013). En dado caso que el *buffer* se encuentre a su máxima capacidad, el trabajo tendrá que quedarse en la máquina causando su bloqueo hasta que el *buffer* o la siguiente máquina estén disponibles. (Rashidi, Jahandar, & Zandich, 2010).

5. Revisión de la literatura

El *Flow Shop* híbrido (HFS por sus siglas en inglés) toma importancia en la literatura y empieza a ser estudiado en los años setenta (Ruiz & Vázquez-Rodríguez, 2010), debido a que en diversas industrias surge la necesidad de adquirir nuevos procesadores con el fin de aumentar la productividad, conduciendo a la coexistencia de varias máquinas en algunas estaciones de trabajo (Miranda Lugo, Pérez Martínez, & Florence Teixera Jr, 2013). Los diversos trabajos que investigan la programación de la producción de un HFS, se pueden clasificar en tres temáticas:

complejidad de procesamiento, criterios de modelado y métodos de solución (Linn & Zhang, 1999).

En cuanto a la complejidad de procesamiento, Linn & Zhang (1999) dividen el problema en tres categorías: HFS con dos estaciones, HFS con tres etapas y HFS con múltiples estaciones, teniendo en cuenta que la complejidad del problema aumenta al considerar mayor cantidad de etapas de procesamiento y el tipo de máquinas paralelas con las que cuenta cada estación, ya que estas pueden ser idénticas, es decir, los tiempos de procesamiento de un trabajo no varían de un procesador a otro (Lopez, Giraldo, & Arango, 2015), o máquinas paralelas no relacionadas, en las que el tiempo de procesamiento de una tarea dependerá de cada uno de los procesadores (Ribas, Leisten, & Framiñan, 2010).

Los trabajos que investigan la programación de la producción de un HFS con dos estaciones son los más encontrados en la literatura y se clasifican en seis subcategorías, teniendo en cuenta la cantidad y el tipo de máquinas por estación (Linn & Zhang, 1999). Algunos de los autores que han trabajado el problema con dos etapas de procesamiento son: Uetake, Tsubone y Ohba (1995) plantean el problema, en el cual los trabajos pasan por un procesador en la primera estación, y dependiendo de las especificaciones del producto final, el trabajo entra a una de las tres máquinas que se encuentran en la segunda etapa, cuyo objetivo era minimizar el tiempo de completamiento de la última tarea (*makespan*); Los, Hsu y Su (2008), al igual que Figielska (2008) abordaron un problema con m máquinas paralelas no relacionadas en la primera estación y un solo procesador en la segunda etapa, con el objetivo de minimizar el *makespan*.

Parra (2006) realizó un trabajo que busca determinar la programación de la producción de un HFS con tres etapas, desarrollando un algoritmo genético para determinar la secuenciación del

proceso productivo de una ladrillera que consta de una máquina en la primera etapa (extracción), dos procesadores en la segunda estación (secado) y dos máquinas en la última etapa (transformación).

Por otra parte, los trabajos que involucran múltiples estaciones en el problema HFS se encuentran: Seido y João (2002) desarrollaron un trabajo con k estaciones, donde en cualquiera de ellas existen dos máquinas no relacionadas, proponiendo dos heurísticas con el objetivo de minimizar el *makespan*; Jungwattanakit, Reodecha, Chaovalitwongse y Werner (2009;2007), consideraron un problema con k estaciones, donde al menos una estación tiene máquinas paralelas no relacionadas, formulando un programa entero mixto bi-objetivo, buscando minimizar el *makespan* y reducir el número de trabajos tardíos.

Hasta mediados de los años noventa, las investigaciones relacionadas con el HFS no consideraban los tiempos de alistamiento, pues estos eran incluidos en el tiempo de procesamiento de las tareas (Fátima Morais, Godinho Filho, & Perassoli Boiko, 2013); pero con el fin de hacer el problema más semejante a la realidad se tiene en cuenta los tiempos de alistamiento como restricción, autores como Allahverdi (2000), Low (2005), y Marichelvama, Prabaharanb y Yangc (2014) exponen un HFS con máquinas paralelas y tiempos de alistamiento independientes de la secuencia. Agregándole complejidad al problema algunos autores empezaron a considerar tiempos de alistamiento dependientes de la secuencia, es decir, la preparación de una máquina para procesar cierto trabajo depende de la tarea procesada anteriormente en dicha máquina. Scholz-Reiter, Rekersbrink, Görges (2010), abordaron el problema del *Flow Shop* flexible teniendo en cuenta múltiples etapas, máquinas paralelas no relacionadas y tiempos dependientes de la secuencia con el objetivo de minimizar el *makespan*; Por otro lado Ruiz y Maroto (2006), y Miranda y Teixeira

(2012) presentan un HFS considerando la elegibilidad de las máquinas junto con tiempos de alistamiento dependientes de la secuencia.

Otra restricción destacada son los *buffers* o almacenamientos intermedios, donde los trabajos son transferidos en caso que la siguiente máquina en la secuencia esté ocupada, según el problema que se quiera trabajar, estas zonas pueden ser consideradas de capacidad limitada o ilimitada. (Rashidi, Jahandar, & Zandich, 2010). Cui y Gu (2015) consideraron *buffers* con capacidad ilimitada como restricción en un HFS con máquinas idénticas, donde el objetivo era minimizar el *makespan*, para dar solución al problema, los autores proponen un algoritmo basado en una colonia artificial de abejas.

Yaurima, Burtseva y Tchernykh (2009) plantean el problema del *Flow Shop* híbrido con máquinas paralelas no relacionadas, restricciones de disponibilidad y *buffers* de tamaño limitado, los autores utilizan un algoritmo genético para resolverlo y lo aplican a la producción de circuitos impresos de televisión. Miranda, Pérez y Teixeira (2013) consideran el mismo problema e incluyen como restricciones los tiempos de liberación de máquinas, de transporte y la elegibilidad de máquinas, y le dan solución a través de un modelo de programación entera mixta.

Por otra parte, de Fátima Morais, Godinho Filho, & Perassoli Boiko (2013) plantean que los métodos de solución para problemas de programación de la producción se dividen en tres grandes categorias: métodos de solución óptima, métodos heurísticos y métodos metaheurísticos. Los primeros generan una solución óptima de acuerdo al criterio de desempeño adoptado, es decir, entre todas las soluciones factibles escoge la que produzca el mejor valor de función objetivo (Maccarthy & Liu, 1993). Uno de los primeros investigadores en trabajar el problema del *Flow Shop* híbrido fue Gupta, quien demostró que un HFS de dos estaciones, donde una de ellas tiene

solo una máquina es NP-Hard (Costa, Cappadonna, & Fichera, 2014), por lo que existe una necesidad significativa de métodos heurísticos o metaheurísticos eficaces, con el fin de reducir el tiempo que conlleva la solución matemática clásica y el consumo de recursos informáticos (Jun & Park, 2015).

Camargo, de Souza y Freitas (2013) diseñaron un algoritmo de búsqueda iterado codicioso, para dar solución a una generalización del problema HFS denominado línea de flujo hibrida y flexible (HFFL por sus siglas en inglés), en el que los trabajos no son procesados en todas las etapas del sistema, sino al menos en una de ellas; además consideran máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia, elegibilidad de los procesadores y tiempos de liberación de las máquinas. El algoritmo utiliza una heurística constructiva NEH (Heurística de Nawaz-Encore-Ham) para encontrar la solución inicial, la cual se basa en los tiempos de procesamiento promedio de los trabajos, buscando la mejor secuencia posible de las tareas para minimizar el *makespan*. A la solución inicial le aplican un método de búsqueda denominado Mejoramiento Iterativo de Inserción (III por sus siglas en inglés), considerando que los trabajos que se deben mover de su posición inicial deben ser aquellos que hacen parte de la ruta crítica, pues estas tareas son las que determinan el tiempo de completamiento de todos los trabajos.

Otro método de solución empleado para resolver el problema del HFS es una búsqueda de la vecindad variable (VNS por sus siglas en inglés), el cual es adaptado por Almeder y Hartl (2013) en un *Flow Shop* híbrido de dos estaciones en un ambiente de producción estocástico, en el que la primera tiene un solo procesador y la segunda etapa dos máquinas paralelas no relacionadas, además de considerar un *buffer* de tamaño limitado entre las estaciones. Otro algoritmo metaheurístico denominado *Cuckoo Search* es utilizado por Marichelvam, Prabaharan y Yang

(2014), para dar solución a un problema HFS con múltiples estaciones, considerando máquinas paralelas idénticas en las k etapas y tiempos de alistamiento independientes de la secuencia, en el que utilizan la heurística constructiva NEH para obtener una solución que hará parte de la población inicial. El algoritmo *Cuckoo Search* se basa en el comportamiento de las especias de ave cuco en combinación con el tipo de vuelo de Levy de algunas aves de la naturaleza.

El recocido simulado (SA por sus siglas en inglés) es otra heurística utilizada para la solución del HFS, el cual se basa el proceso de enfriamiento de los metales e involucra una probabilidad de aceptación de las diversas soluciones, permitiendo salir de óptimos locales y poder llegar a óptimos globales (CSI/ITESM, 2008). Jungwattanakit, Reodecha y Chaovalitwongse (2007) utilizan el recocido simulado para darle solución a un problema HFS, con máquinas paralelas no relacionadas y tiempos de alistamiento dependientes de la secuencia, cuya función objetivo busca minimizar una suma ponderada del *makespan* y el número de trabajos tardíos. Así mismo, Low (2005) con el objetivo de minimizar el tiempo de flujo total de un HFS de k estaciones y máquinas paralelas no relacionadas, propuso una heurística basada en un recocido simulado, que involucra un procedimiento para determinar la solución inicial y unas estructuras específicas de generación de vecindarios de dicha solución, con el fin de aumentar la calidad de esta.

Uno de los métodos más utilizados para solucionar el problema HFS es el algoritmo genético, y en tal sentido Yaurima, Burtseva y Tchernykh (2009) plantearon uno modificado, donde proponen tres nuevos operadores de cruce, introducen un nuevo criterio de parada y establecen políticas de reinicio con el fin de mejorar la calidad de las soluciones. Finalmente los autores comparan el algoritmo propuesto, con una adaptación del algoritmo genético desarrollado por Ruiz y Maroto (2006), obteniendo mejores resultados el método presentado por Yaurima, Burtseva y Tchernykh (2009).

Jungwattanakit, Reodecha, Chaovalitwongse y Werner (2008) compararon diferentes heurísticas constructivas y estudiaron el algoritmo genético con el fin de determinar sus parámetros más favorables. Para el estudio, los autores consideraron el problema del *Flow Shop* flexible con máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia, y fechas de entrega, donde buscaban minimizar la suma convexa del *makespan* y del número de trabajos entregados tarde. Como resultado de la investigación, se pudo observar que la heurística NEH es mucho mejor que los otros métodos estudiados, a pesar de que su tiempo computacional aumente rápidamente a medida que el tamaño del problema aumenta. Respecto al algoritmo genético, se observó que el operador de cruce *Combined Order and Position-based crossover* (OPX) es mejor que el *Partially mapped crossover* (PMX).

Así mismo, Zandieh, Mozaffari y Gholami (2010) examinan tres métodos heurísticos (SPT, LPT y NEH) y un algoritmo genético, con el fin de programar la producción en un *Flow Shop* híbrido flexible, de modo que se minimice el *makespan*. El algoritmo genético superó significativamente a las heurísticas en todos los problemas de prueba (grandes, medianos y pequeños), además, los resultados revelaron que el algoritmo NEH, tiene un mejor rendimiento que los algoritmos SPT y LPT, y que no se evidencia una diferencia significativa entre estos últimos dos.

Miranda Lugo y Florence Teixera (2012) proponen un algoritmo genético con un operador de cruce *Similar Block 2-Point Order Crossover* (SB2OX) propuesto por Ruiz, Maroto y Alcaraz (2005), sustituyen la fase de mutación con una búsqueda local e implementan un método de reinicio para evitar la convergencia prematura de la población, para dar solución al problema de HFS con máquinas paralelas no relacionadas, tiempos de alistamiento dependientes de la secuencia y elegibilidad de máquinas, con el fin de minimizar el *makespan*. El algoritmo planteado fue

comparado con un algoritmo genético simple propuesto en la literatura para la resolución del mismo problema, los resultados indicaron una clara superioridad del algoritmo propuesto en todas las instancias probadas.

Jun y Park (2015) también proponen un algoritmo genético híbrido que combina la heurística NEH, con un algoritmo de búsqueda local y una regla de asignación de máquinas con el objetivo de minimizar la tardanza total en un HFS. Los autores presentan el caso de estudio de una fábrica de producción de transformadores para evaluar el rendimiento del algoritmo propuesto. Los resultados de la simulación demuestran que este algoritmo supera al algoritmo NEH y a un algoritmo genético simple.

Un algoritmo híbrido inteligente (HA pos sus siglas en inglés) basado en el algoritmo competitivo imperialista (ICA por sus siglas en inglés) para solucionar un *Flow Shop* flexible de dos etapas en un sistema de fabricación sin esperas, con máquinas paralelas no relacionadas y tiempos de alistamiento dependientes de la secuencia fue propuesto por Rabiee, Sadeghi, Mazinani y Shafaei (2014). El algoritmo propuesto basado en el ICA, es un método de solución híbrido que combina el recocido simulado (SA), búsqueda de la vecindad variable (VNS) y algoritmo genético (GA por sus siglas en inglés). Con el propósito de validar el algoritmo propuesto y teniendo en cuenta los algoritmos metaheurísticos más relevantes y usados en ambientes de programación de la producción, el HA es comparado con el ICA, VNS, SA, GA y un algoritmo de las colonias de hormigas (ACO por sus siglas en inglés), superando a los demás algoritmos en términos de las medidas de desempeño. Es importante resaltar que en esta investigación, un ICA básico obtiene mejores resultados que un GA estándar.

Otros autores han planteado algoritmos basados en el comportamiento de las abejas melíferas y los roles que estas desempeñan para explotar o conseguir fuentes de alimentos (Yaurima, Burtseva, & Tchernykh, 2009). Li y Pan (2014), trabajaron el problema HFS con *buffers* de tamaño limitado, con el objetivo de minimizar el *makespan* y para darle solución plantearon un algoritmo híbrido (TABC, por sus siglas en inglés) que combina la colonia artificial de abejas con una búsqueda tabú. Este nuevo algoritmo difiere del tradicional en que en las fases de las abejas empleadas y las abejas espectadoras, las abejas primero seleccionan las fuentes de comida de manera aleatoria, luego implementan una búsqueda tabú, y finalizan su fase evaluando el *fitness* de la nueva fuente de comida generada. Para verificar la eficiencia y efectividad del algoritmo propuesto, hacen una comparación de este con 5 algoritmos más (GAH, GAS, DABC, TS y TSSCS), el análisis demostró que el TABC genera mejores resultados que el GAH, GAS y TS, en instancias pequeñas presenta un desempeño similar al DABC, mientras que el TSSCS presenta mejores resultados, por otro lado, en cuanto a los problemas de gran escala el TABC es el más indicado.

Así mismo, Cui y Gu (2015) buscaron darle solución mediante un algoritmo mejorado basado en una colonia de abejas, a un problema HFS de k etapas, con máquinas paralelas idénticas, *buffers* de tamaño ilimitado y sin considerar tiempos de alistamiento de los trabajos, con el objetivo de minimizar el *makespan*. El algoritmo utiliza una heurística constructiva NEH para obtener una solución inicial, proponen un esquema de evolución diferencial basado en un AG en la fase de las abejas empleadas, incorporan una búsqueda variable de la vecindad para el rol de las abejas observadoras y mediante procedimiento de construcción y destrucción del algoritmo iterado codicioso las abejas exploradoras generen nuevas soluciones. El método es comparado con problemas encontrados en la literatura, que utilizan algoritmos tales como GA, ACO, optimización

por enjambre de partículas (PSO por sus siglas en inglés) y un sistema inmune artificial (AIS por sus siglas en inglés), concluyendo que el método propuesto es más eficiente y efectivo que los otros, debido a que puede resolver problemas con instancias más complejas y permite obtener mejores valores de la función objetivo.

6. Sistema productivo bajo estudio

El sistema productivo a estudiar es el *Flow Shop* híbrido, el cual es una extensión del sistema de producción *Flow Shop* tradicional y consta de dos o más etapas de procesamiento en serie, en el que al menos una de ellas cuenta con dos o más máquinas paralelas (Figielska, 2014). En este tipo de sistema productivo los trabajos a procesar siguen el mismo flujo por las diferentes etapas de procesamiento, las cuales tienen la misma cantidad de máquinas y cada trabajo será procesado en al menos una de las m máquinas de cada estación.

Las máquinas paralelas en cada una de las estaciones del *Flow Shop* híbrido a estudiar, se consideran no relacionadas, es decir, los tiempos de procesamiento para cada uno de los trabajos difiere según la máquina en la que se procese. Además, los procesadores presentan un tiempo de alistamiento antes de empezar a realizar un trabajo, el cual es dependiente del trabajo procesado anteriormente en esa máquina y del procesador como tal.

Otra característica son los *buffers* o zonas de almacenamiento temporal de los trabajos en proceso entre estaciones, con una capacidad limitada b, que representa la cantidad de trabajos que se pueden almacenar en la zona. El uso de los buffers se hace necesario cuando un trabajo termina

de ser procesado en una etapa y todas las máquinas de la siguiente estación se encuentran ocupadas. En la figura 8 se ilustra el sistema de producción descrito.

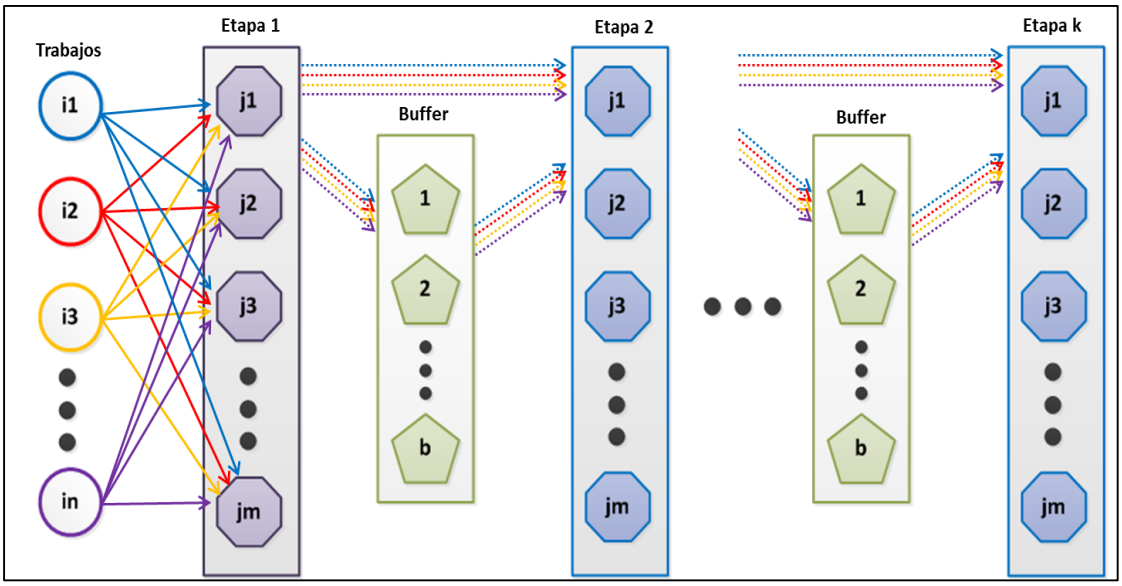


Figura 8. Sistema de producción a estudiar.

Por último, los tiempos de procesamiento de los trabajos son conocidos y siguen una distribución uniforme entre 75 y 125, al igual que los tiempos de alistamiento de las tareas, que siguen una distribución uniforme entre 25 y 75 (Miranda Lugo, Pérez Martínez, & Florence Teixera Jr, 2013).

7. Modelo matemático

Los siguientes supuestos son considerados en el modelo matemático presentado:

- Las máquinas en cada estación no son relacionadas y al menos existen dos por etapa.

- Los tiempos de alistamientos son dependientes de la máquina y la secuencia de los trabajos.
- Todas las máquinas en cada una de las estaciones pueden procesar todas las tareas.
- Todos los trabajos deben ser procesados en cada una de las estaciones.
- Las máquinas de todas las estaciones están disponibles en el instante t=0.
- Cada trabajo debe ser asignado a solo una máquina en cada una de las estaciones de trabajo.
- Cada máquina debe procesar solamente un trabajo a la vez.
- No existen límites de asignación de trabajos para cada máquina.
- Se considera un trabajo imaginario cero en cada una de las máquinas, el cual se toma como un calentamiento de la máquina al inicio de la secuencia y tendrá efecto en la asignación de tareas y los tiempos de alistamiento junto al primer trabajo asignado.
- Los buffers son considerados como máquinas "especiales" dentro de una estación de almacenamiento intermedio entre etapas, los cuales cuentan con tiempo de alistamiento y tiempo de procesamiento igual a cero sea cual sea la secuencia.
- Cuando un trabajo termine de ser procesado en una estación, debe ser asignado a alguna de las máquinas disponibles en la siguiente estación; en caso que no existan procesadores disponibles en la próxima etapa debe pasar a un *buffer* cuyo tamaño es limitado, si tanto el *buffer* como las máquinas de la siguiente estación están ocupados el trabajo se quedará en el procesador de la estación actual bloqueándolo y evitando que pueda procesar otras tareas.

7.1. Modelo de programación lineal entera mixta.

A continuación, se presenta un modelo matemático del problema a tratar, el cual fue adaptado del modelo presentado por Miranda, Pérez y Florence (2013) por el estudiante Camilo Pabón (2017) para su proyecto de grado titulado: "Solución al problema del *flow shop* híbrido (HFS) con máquinas paralelas no relacionadas y buffers de tamaño limitado mediante una metaheurística basada en el algoritmo competitivo imperialista (ICA)":

Índices y conjuntos:

- $h, i, g =$ Tareas.
- $k =$ Estaciones de trabajo.
- $j =$ Máquinas.

Parámetros:

- $B =$ Número escalar muy grande, mayor que la suma de los tiempos de procesamiento de todos los trabajos en todas las máquinas.
- $S_{hijk} =$ Tiempo de alistamiento entre las tareas h e i, en la máquina j, de la estación k.
- $P_{ijk} =$ Tiempo de procesamiento de la tarea i en la máquina j, de la estación k.

Variables de decisión:

- $Y_{hijk} =$ Representa una variable binaria cuya existencia permite la asignación de las tareas en cada estación de trabajo. La variable tomará el valor de 1 en caso de que la tarea h preceda a la tarea i en la máquina j de la estación k, y en caso contrario tomará el valor de 0.
- $C_{ik} =$ Instante de finalización del procesamiento de la tarea i en la estación k.

- $E_{ik} =$ Instante de salida de la tarea i de la estación k.
- $C_{máx} =$ Representa el *makespan*, la cual es la variable a optimizar y se define como el momento en que termina el procesamiento del último trabajo en la última estación.

Función objetivo:

$$\text{Mín } C_{máx}$$

Restricciones:

$$\sum_j \sum_h Y_{hijk} = 1 \qquad \forall i,k;\ \ h \neq i;\ \ i \neq 0; \qquad (1)$$

$$\sum_j \sum_h Y_{ihjk} \leq 1 \qquad \forall i,k;\ \ h \neq i;\ \ h \neq 0;\ \ i \neq 0; \qquad (2)$$

$$\sum_g Y_{ghjk} \geq Y_{hijk} \qquad \forall h,i,j,k;\ \ h \neq i;\ \ g \neq h;\ \ g \neq i;\ \ h \neq 0;\ \ i \neq 0; \qquad (3)$$

$$\sum_j (Y_{hijk} + Y_{ihjk}) \leq 1 \qquad \forall h,i,k;\ \ h \neq i;\ \ i \neq 0; \qquad (4)$$

$$\sum_i Y_{hijk} \leq 1 \qquad \forall h,j,k;\ \ h \neq i;\ \ i \neq 0;\ \ h = 0; \qquad (5)$$

$$C_{ik} = 0 \qquad \forall k;\ \ i = 0; \qquad (6)$$

$$C_{ik} + B\left(1 - Y_{hijk}\right) \geq E_{hk} + S_{hijk} + P_{ijk} \qquad \forall h,i,j,k;\ \ h \neq i;\ \ i \neq 0 \qquad (7)$$

$$C_{ik} + B\left(1 - Y_{hijk}\right) \geq C_{i,k-1} + S_{hijk} + P_{ijk} \qquad \forall h,i,j,k;\ \ h \neq i;\ \ i \neq 0\ \ k \neq 1 \qquad (8)$$

$$E_{i,k-1} = C_{ik} - \sum_j \sum_h Y_{hijk} \cdot S_{hijk} - \sum_j \sum_h Y_{hijk} \cdot P_{ijk} \qquad \forall i,k;\ \ h \neq i;\ \ i \neq 0\ \ k \neq 1 \qquad (9)$$

$$E_{ik} \geq C_{ik} \qquad \forall i,k; \quad k \neq \acute{u}ltima\ estación \qquad (10)$$

$$E_{ik} = C_{ik} \qquad \forall i,k; \quad k = \acute{u}ltima\ estación \qquad (11)$$

$$C_{máx} \geq C_{ik} \qquad \forall i,k; \quad k = \acute{u}ltima\ estación \qquad (12)$$

$$Y_{hijk} \in \{0,1\} \qquad \forall h,i,j,k; \quad h \neq i; \quad i \neq 0$$

$$C_{ik}, E_{ik} \geq 0 \qquad \forall i,k$$

Cada una de las ecuaciones representa una restricción del problema. La ecuación (1) asegura que cada tarea debe ser precedida por únicamente otra tarea, en solamente una máquina de cada estación y garantiza la no división de trabajos en lotes de procesamiento al cerciorarse que las tareas sean asignadas a una sola máquina en cualquier estación de trabajo. Por su parte, la ecuación (2), indica que cada trabajo debe tener a lo sumo un sucesor, en caso de ser asignado a cualquier máquina de cualquier estación. En cuanto a la ecuación (3), su función es cerciorarse que una tarea procesada en una máquina asignada debe tener como máximo un predecesor dentro de la misma máquina. La ecuación (4) evita la existencia de precedencias cruzadas, es decir, garantiza que si al trabajo i, lo precede el trabajo h, entonces el trabajo h no preceda al i. La ecuación (5) asegura que el trabajo ficticio inicial cero tenga como máximo un solo sucesor en cualquier máquina de cualquier estación. La ecuación (6) asegura que el tiempo de completamiento del trabajo ficticio inicial sea cero.

Por su parte, las ecuaciones (7) y (8) son las encargadas de calcular el instante de finalización de cada trabajo, mientras que la ecuación (9) se encarga de realizar el cálculo del instante de salida de cada tarea en cada etapa. La ecuación (10) garantiza que ningún trabajo deje una estación antes

que su procesamiento en la misma esté finalizado. La ecuación (11) indica que cada trabajo sale del sistema cuando su procesamiento en la última estación haya finalizado, mientras que el *makespan* es definido mediante la ecuación (12). Por último, se definen las variables de decisión del problema.

8. Diseño del algoritmo genético híbrido

8.1. Población Inicial

8.1.1. Generación de la Subpoblación inicial. Se genera una matriz pxn, donde p representa el tamaño de la población y n el número de trabajos, en la que cada fila representa una permutación de números enteros aleatorios entre 1 y n, garantizando que ninguna tarea se repita, como se muestra en la figura 9. Cada permutación representa la secuencia a seguir por los trabajos en la primera etapa de procesamiento.

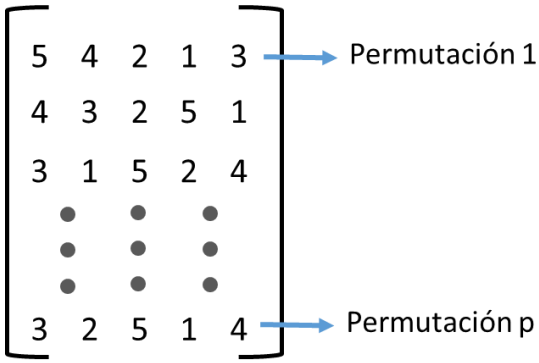


Figura 9. Representación de la matriz pxn (Subpoblación inicial).

8.1.2. Asignación de trabajos en las máquinas. La asignación de la máquina de la primera etapa en la que se procesará cada uno de los trabajos de las secuencias definidas en la matriz pxn, se realiza teniendo en cuenta el menor tiempo posible de operación. Considerando que el tiempo de alistamiento depende del procesador y del trabajo procesado anteriormente en esa máquina, se plantea la ecuación (13), la cual se obtuvo de una modificación hecha a la formulación propuesta por Ruiz y Maroto (2006).

Teniendo en cuenta que el alistamiento de las máquinas es no anticipatorio, es decir, se lleva a cabo cuando el trabajo y la máquina a utilizar se encuentran disponibles, el cálculo del tiempo de completamiento de los trabajos en cada etapa se realiza escogiendo el máximo entre el instante de finalización del trabajo a procesar en la etapa anterior y el tiempo de completamiento del último trabajado procesado en la máquina a utilizar, sumándole el tiempo de alistamiento y procesamiento del trabajo i.

En este orden de ideas, la asignación de los trabajos a las máquinas se hará de modo que se minimice el tiempo de completamiento de las tareas en cada una de las estaciones, mediante la siguiente fórmula:

$$C_{\pi(i)k} = min\left\{max\left\{C_{L_{kj},k}\ ;\ C_{\pi(i),k-1}\right\} + S_{h\pi(i)jk} + P_{\pi(i)jk}\right\} \quad (13)$$

Donde,

$C_{\pi(i)k}$ Es el tiempo de completamiento de la tarea de la i-ésima posición en la secuencia π en la estación k.

L_{kj} Es el último trabajo asignado a la máquina j de la estación k.

$C_{L_{kj},k}$ Es el tiempo de completamiento del último trabajo asignado a la máquina j de la estación k, en la estación k.

$S_{h\pi(i)jk}$ Tiempo de alistamiento de la tarea de la i-ésima posición en la secuencia π en la estación k, en la máquina j, después de haber procesado la tarea h.

$C_{\pi(i),k-1}$ Es el tiempo de completamiento de la tarea de la i-ésima posición en la secuencia π en la estación inmediatamente anterior (k-1).

$P_{\pi(i)jk}$ Tiempo de procesamiento de la tarea de la i-ésima posición en la secuencia π en la estación k, en la máquina j.

Considerando que se tienen 5 trabajos a procesar en 3 estaciones, donde cada una de las etapas tiene 2 máquinas, y la capacidad del buffer es de 1. Los tiempos de procesamiento y alistamiento de la instancia i5j2k3-1 se representan en la figura 10 y 11, respectivamente.

	k1		k2		k3	
	j1	j2	j1	j2	j1	j2
i1	119	92	75	119	101	118
i2	88	123	78	85	79	117
i3	112	96	83	78	100	104
i4	121	90	121	115	98	117
i5	121	122	87	113	106	77

Figura 10. Tiempos (u.t.) de procesamiento para i5j2k3-1.

	k1												k2											
	j1						j2						j1						j2					
	j0	j1	j2	j3	j4	j5	j0	j1	j2	j3	j4	j5	j0	j1	j2	j3	j4	j5	j0	j1	j2	j3	j4	j5
j1	48		27	64	34	38	26		74	44	73	49	66		32	28	25	73	74		58	68	28	72
j2	58	63		37	65	34	28	43		42	55	34	67	28		26	48	59	55	52		37	66	53
j3	69	34	26		37	30	43	32	72		66	51	26	73	52		39	66	31	51	29		26	45
j4	41	68	30	71		49	69	66	31	57		57	29	40	67	32		61	61	50	60	37		52
j5	54	61	63	44	35		70	39	58	66	54		37	49	56	62	40		73	42	40	54	57	

	k1											
	j1						j2					
	j0	j1	j2	j3	j4	j5	j0	j1	j2	j3	j4	j5
j1	73		63	36	30	52	51		60	62	53	52
j2	29	41		63	62	69	54	72		49	65	62
j3	40	55	59		26	72	48	49	49		57	28
j4	73	43	47	69		28	53	44	46	62		36
j5	41	31	47	64	54		51	29	68	61	34	

Figura 11. Tiempos (u.t.) de alistamiento para i5j2k3-1.

Con la primera permutación π [5 4 2 1 3] de la subpoblación inicial y conociendo los tiempos de alistamiento y procesamiento, se aplica (13) para cada uno de los trabajos en cada una de las estaciones, como sigue:

Para i=5, k=1, j=1: $\{max\{0\ ;\ 0\} + \ 121 + 54\}$ = 175

Para i=5, k=1, j=2: $\{max\{0\ ;\ 0\} + \ 122 + 70\}$ = 192

$$min\{175\ ;\ 192\} = 175$$

Para i=4, k=1, j=1: $\{max\{175\ ;\ 0\} + \ 49 + 121\}$ = 345

Para i=4, k=1, j=2: $\{max\{0\ ;\ 0\} + \ 69 + 90\}$ = 159

$$min\{345\ ;\ 159\} = 159$$

Entonces el trabajo de la primera posición de la secuencia, es asignado a la máquina 1 de la primera etapa, debido a que el tiempo de completamiento en ese procesador es menor, mientras que el trabajo 4 es asignado a la máquina 2 en la primera etapa. Lo mismo se realiza para los demás trabajos de la secuencia y se determina que junto a la tarea 5, la 2 y la 3 serán procesadas en la primera máquina de la estación y junto con la 4, se procesa la 1 en la segunda máquina de la etapa, respetando el orden estipulado por π.

La ecuación (13) se utiliza para la asignación de los trabajos a las máquinas en cada una de las estaciones del sistema productivo, teniendo en cuenta que la secuencia de los trabajos en las etapas siguientes se determinará mediante una disciplina FIFO (*First In, First Out)* basada en la secuencia de la estación anterior.

8.1.3. Representación del cromosoma. Un individuo es un vector de ***1 x [(i + j)k – 1],*** el cual representa la secuencia de procesamiento de los trabajos en las diferentes máquinas de cada una de las estaciones. En el individuo se muestra el cambio de las estaciones de procesamiento, calculando estas posiciones mediante la expresión **(i + j)(k – 1)**, para cada una de las etapas, y el cambio de máquina en cada estación se representa mediante un * y depende de la cantidad de trabajos que se asignen a cada una de las máquinas.

Siguiendo con el ejemplo del apartado 8.1.2., en la figura 12 se muestra la representación de la primera etapa del individuo, teniendo en cuenta la asignación de las máquinas mediante (13).

Una vez se tenga la secuencia y la asignación de las máquinas para procesarlos en la primera etapa, la secuenciación en la segunda etapa se regirá mediante una disciplina FIFO (First In First Out), es decir, que el primer trabajo en culminar su procesamiento en la primera etapa, será el primero en entrar en la segunda estación y análogamente con las demás tareas.

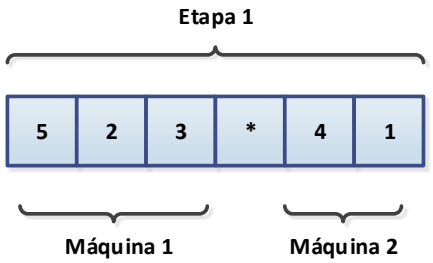


Figura 12. Representación de la primera etapa de un individuo de la instancia i5j2k3-1.

Los trabajos representados en la figura 12, finalizan su procesamiento en el siguiente orden: 4, 5, 2, 1 y 3; por tanto, se procede asignar de acuerdo a la ecuación (13) el trabajo 4 a alguna de las máquinas de la etapa 2, luego se asigna procesador a la tarea 5, después a la 2, posteriormente a la 1 y por último se hace la asignación de la máquina a la tarea 3. Esto se repite tantas veces como número de etapas tenga el sistema productivo.

Con base en esto, el individuo resultante de la primera permutación de la subpoblación representada en la figura 12 se muestra en la figura 13.

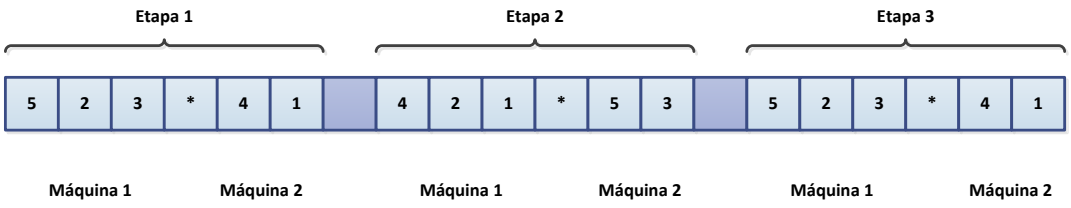


Figura 13. Representación de un individuo para la instancia i5j2k3-1.

8.1.4. Generación de la población inicial. La población inicial será un matriz de tamaño ***p x [(i + j)k – 1]***, donde p es la cantidad de individuos de la población, i la cantidad de trabajos a procesar, j el número de máquinas por estación y k la cantidad de etapas del sistema productivo. Cada una de las filas de la matriz, representará una solución al problema, la cual resulta del paso a paso expuesto anteriormente.

Adicionalmente se utiliza un factor de diversidad de la población inicial (fdpi), el cual permite generar ***fdpi*p*** individuos, con el objetivo de escoger los p individuos con el menor *makespan* e iniciar la evolución con una buena población inicial.

8.2. Cálculo del *fitness* o *makespan*

El *fitness* del algoritmo genético híbrido será igual al *makespan* o tiempo de terminación de las tareas a procesar en el sistema, siendo la variable a optimizar en el algoritmo propuesto. La ecuación (13) permite conocer el tiempo de completamiento de cada una de las tareas en cada etapa de procesamiento, por ende, el mayor de estos valores será igual al *makespan* de la solución en caso que no se presenten bloqueos en las máquinas del sistema; en el ejemplo representado en la figura 13, el *makespan* estará sujeto al tiempo de completamiento de la tarea 3 o 1 en la tercera etapa.

Teniendo en cuenta los tiempos de alistamiento y procesamiento de la instancia i5j2k3-1, la figura 14 muestra el cálculo de los tiempos de completamiento de las 5 tareas en cada estación de procesamiento, siendo el *makespan* el mayor de estos valores.

Como se puede observar en la figura 15, el trabajo 2 una vez finalizado su procesamiento en la primera etapa entra al primer buffer del sistema, con el fin de no bloquear la máquina 1 de esta estación y se puede empezar con el alistamiento para procesar el trabajo 3.

Es importante recalcar que un bloqueo se puede presentar cuando la máquina de la siguiente estación en la que un trabajo debe ser procesado está ocupada y el buffer no tiene espacio disponible, por lo que la tarea deberá esperar en el procesador de la etapa actual, evitando que esta

máquina procese otro trabajo. Teniendo en cuenta lo anterior, el tiempo de completamiento de una tarea en cierta estación estará determinado por el instante de salida del trabajo en esa etapa.

8.3. Selección

El operador de selección utilizado es torneo, en el cual se seleccionan de manera aleatoria dos individuos de la población, entre los cuales se compara el valor de *makespan* correspondiente a cada uno de ellos, y se selecciona aquel que posea el mejor *fitness* (menor *makespan*). Este proceso se realiza dos veces para dar origen a los dos padres

8.4. Proceso de cruce

En este proceso se utiliza el operador de cruce basado en dos puntos, este inicia con la selección de los dos puntos de manera aleatoria para cada etapa, cada uno de estos corresponde a un número entero entre 1 y (n+(m-2)), donde n es el número de trabajos y m la cantidad de máquinas por estación. Una vez generados estos dos puntos los descendientes se crean como se explica a continuación.

	Estación 1						Buffer	Estación 2						Buffer	Estación 3					
	Máquina 1				Máquina 2			Máquina 1				Máquina 2			Máquina 1				Máquina 2	
Individuo	5	2	3		4	1		4	2	1		5	3		5	2	3		4	1
T. Completamiento en la etapa anterior	0	0	0		0	0		159	297	324		175	435		361	435	558		309	542
T. Completamiento trabajo anterior	0	175	297	*	0	159		0	309	435	*	0	361		0	508	656	*	0	479
T. Alistamiento	54	34	26		69	73		29	48	32		73	45		41	69	59		53	30
T. Procesamiento	121	88	112		90	92		121	78	75		113	78		106	79	100		117	118
T. Completamiento	175	297	435		159	324		309	435	542		361	558		508	656	**815**		479	690

Makespan

Figura 14. Cálculo del *makespan* para un individuo de la instancia i5j2k3-1.

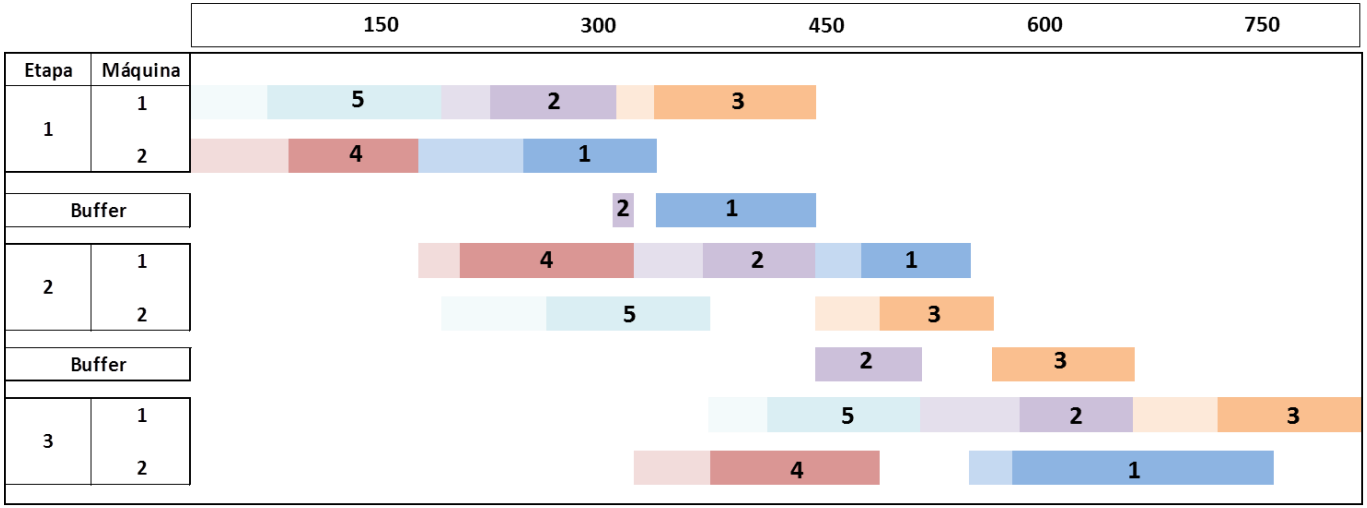


Figura 15. Diagrama de Gantt para un individuo de la instancia i5j2k3-1.

El hijo 1 hereda directamente del padre 1 los genes del primer y último intervalo; los genes faltantes en el segundo intervalo son heredados del padre dos en el mismo orden en que se encuentran en dicho cromosoma, finalmente el segundo hijo se genera de manera análoga, heredando directamente los genes del padre 2 y los faltantes del padre número 1. Tanto la generación de los puntos como el proceso de cruce se realizan en cada una de las etapas.

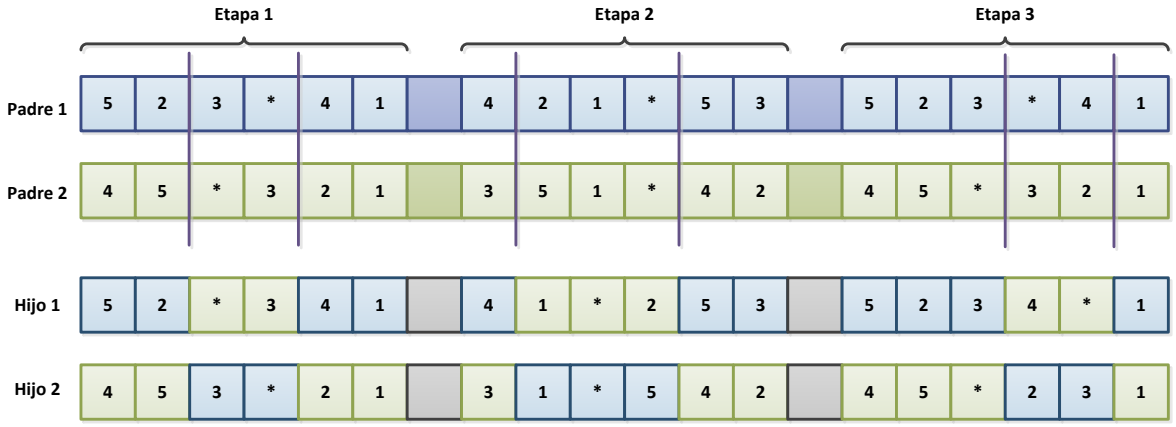


Figura 16. Ejemplo de cruce de dos padres.

8.5. Mutación

Una vez obtenidos los hijos, se genera un número aleatorio entre 0 y 1 asociado a cada uno de los hijos, si este número es menor a la tasa de mutación establecida previamente se ejecuta el proceso de mutación, de lo contrario, los hijos pasan este proceso sin modificaciones. La mutación se realiza utilizando el operador de intercambio, es decir, se seleccionan dos posiciones del cromosoma aleatoriamente y se intercambia la información genética en esos dos puntos, este proceso se debe realizar en cada una de las etapas, así como se ilustra en la figura 17.

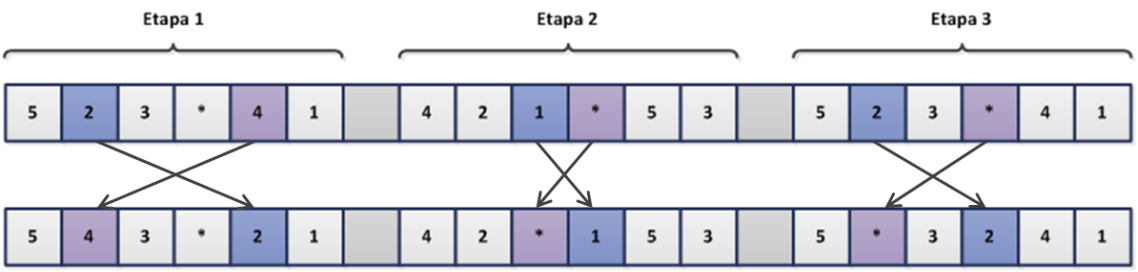


Figura 17. Ejemplo de mutación.

8.6. Selección del descendiente

Una vez obtenidos los hijos, se procede a seleccionar los individuos que entrarán a ser parte de la población, para ello, se determina el *makespan* tanto de los hijos como de los padres, estos valores se comparan y se escogen las dos mejores soluciones, es decir, aquellas que presenten un menor *makespan*.

8.7. Proceso de Aceptación

Para determinar si los descendientes son aptos para ingresar a la población se aplica el siguiente criterio:

Criterio de unicidad: Criterio mediante el cual se verifica que no exista una solución idéntica al descendiente dentro de la población, de ser así el descendiente se descarta y no puede ingresar a la población.

8.8. Criterio de parada

El algoritmo detendrá su ejecución con el cumplimiento de alguno de los dos criterios de parada siguientes:

1. Si la cantidad de iteraciones alcanza el número máximo de iteraciones definido en los parámetros iniciales.

2. Si después de un cierto número de iteraciones consecutivas no existe mejora.

8.9. Búsqueda variable de la vecindad

Una vez obtenida una solución parcial a partir del algoritmo genético, se lleva a cabo una búsqueda variable de la vecindad modificada propuesta por Cui y Gu (2015), la cual utiliza dos operadores de mutación (Insertar e Intercambio).

Antes de dar inicio a la búsqueda de la vecindad se generan dos números aleatorios diferentes u y v, entre 1 y **(n+(m-1))**, para cada una de las etapas, los cuales representan posiciones en la secuencia.

A continuación, se explican los operadores utilizados:

- Insertar: Este operador remueve el trabajo ubicado en la posición u y lo inserta en la posición v, por ejemplo, considérese u=2 y v=5 para la primera etapa, u=4 y v=2 para la segunda etapa, y u=1 y v=3 para la tercera etapa, la nueva solución generada será como se ilustra en la figura 18.

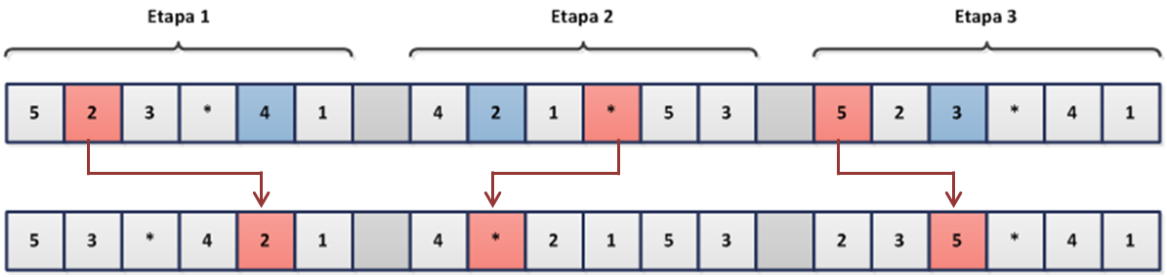


Figura 18. Ejemplo operador insertar.

- Intercambio: Este operador intercambia el trabajo ubicado en la posición u con el trabajo en la posición v, por ejemplo, u=2 y v=5 para la primera etapa, u=4 y v=2 para la segunda etapa, y u=1 y v=3 para la tercera etapa, la nueva solución generada será como se ilustra en la figura 19.

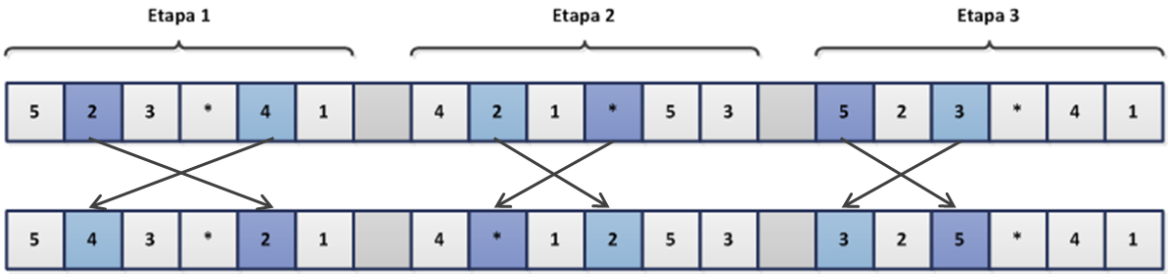


Figura 19. Ejemplo operador intercambio.

Luego de ser generados los números u y v, el algoritmo sigue los siguientes pasos:

1. Se aplica el operador de mutación Insertar, si el *makespan* del cromosoma modificado es menor al de la solución parcial se actualiza el resultado y se continúa con el paso 2, de lo contrario se termina el procedimiento y se aumenta en una unidad el contador de parada.

2. Se aplica el operador de mutación Intercambio, si el *makespan* del nuevo individuo es menor al de la solución parcial se actualiza el resultado y se continúa con el paso 1, de lo contrario se termina el procedimiento y se aumenta en una unidad el contador de parada.

Este algoritmo se detiene cuando el contador es igual a **(n + m – 1) x (n + m – 2)**, teniendo en cuenta que este número es igual a la cantidad de posibles combinaciones que pueden formar u y v.

Las figuras 20 y 21 muestran un diagrama de flujo que describe el paso a paso del algoritmo genético híbrido.

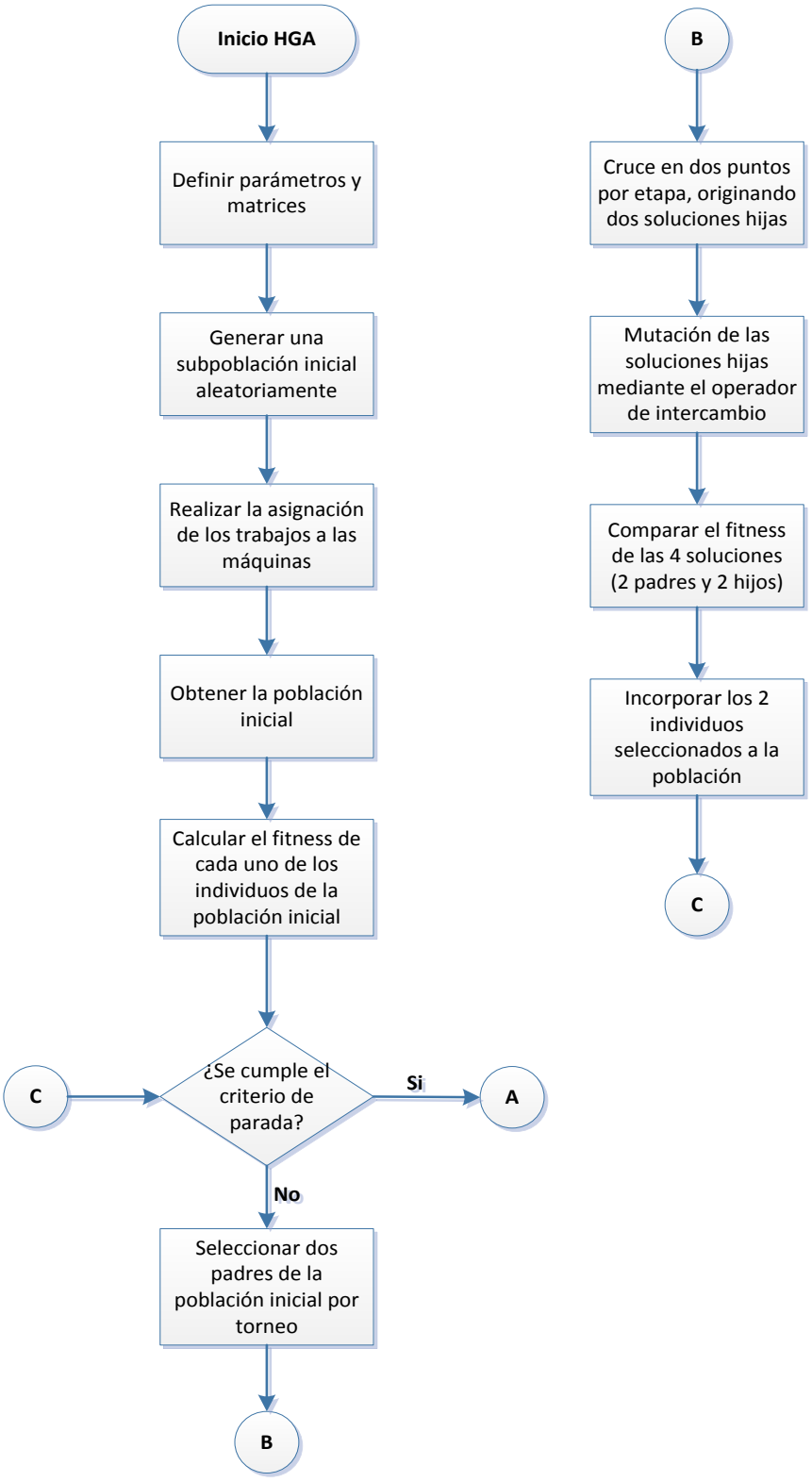


Figura 20. Diagrama de flujo del algoritmo genético hibrido, parte 1.

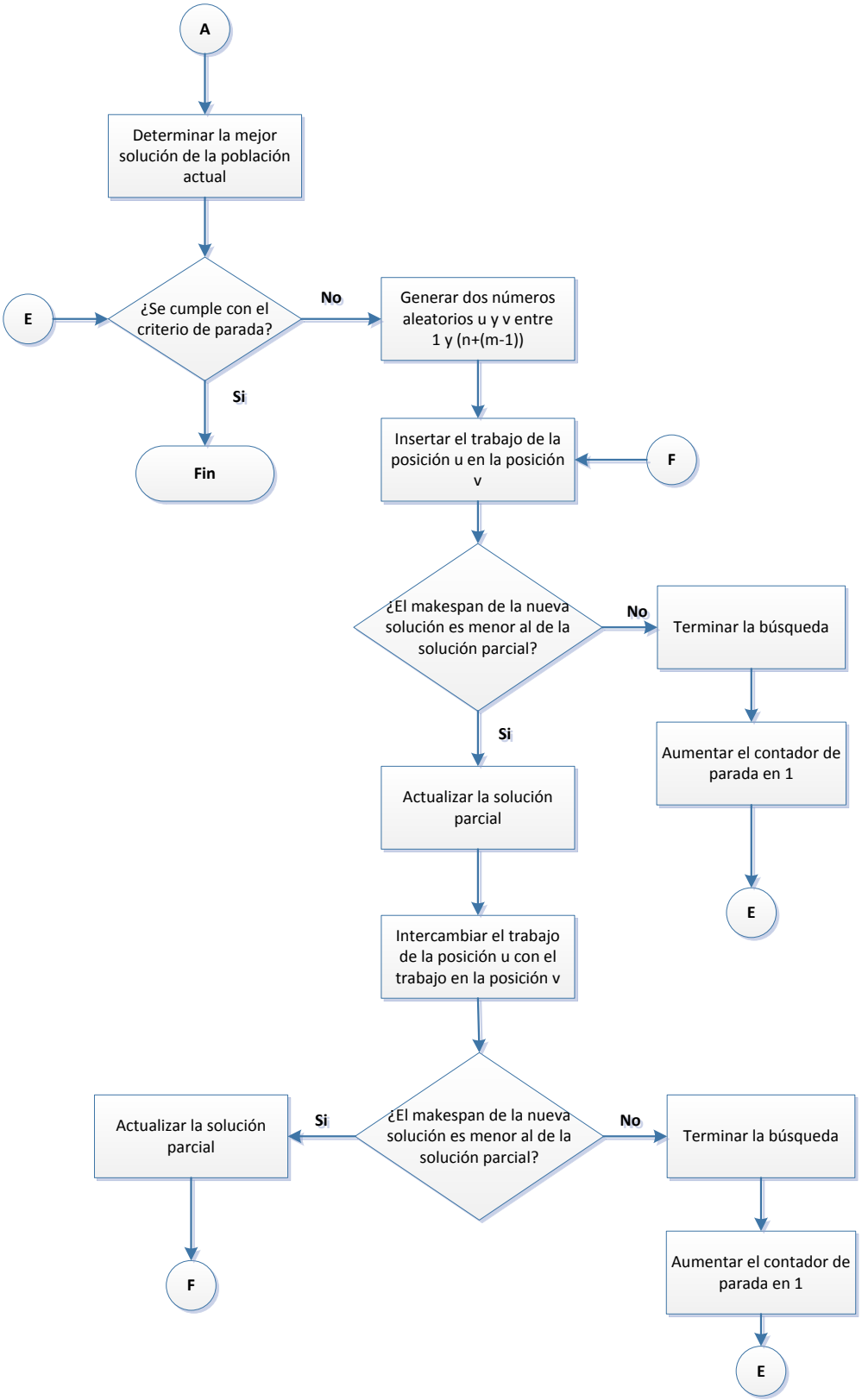


Figura 21. Diagrama de flujo del algoritmo genético hibrido, parte 2.

9. Diseño del algoritmo híbrido basado en una colonia artificial de abejas

9.1. Generación de las fuentes de alimento

Al igual que en el algoritmo genético hibrido, en primera instancia, se obtiene un conjunto de soluciones parciales, mediante una matriz sxn, donde s indica la cantidad de fuentes de alimento, que es igual al número de abejas empleadas y observadores, mientras n representa el número de trabajos a procesar. Cada fila de la matriz representa una permutación de números enteros aleatorios entre 1 y n, los cuales indican la secuencia a seguir por los trabajos en la primera etapa de procesamiento.

Posteriormente se realiza la asignación de la máquina de la primera etapa en la que se procesará cada uno de los trabajos de las secuencias definidas en la matriz sxn, mediante la ecuación (13) detallada en el apartado 8.1.2, la cual se utilizará para la asignación de máquinas a los trabajos en cada una de las estaciones del sistema productivo, teniendo en cuenta que la secuencia de los trabajos en las etapas siguientes se determinará mediante una disciplina FIFO (*First In, First Out)* basada en la secuencia de la estación anterior.

Una fuente de alimento será un vector de ***1 x [(i + j)k – 1],*** el cual representa la secuencia de procesamiento de los trabajos en las diferentes máquinas de cada una de las estaciones, como se muestra en la figura 13 del apartado 8.1.3.

De esta manera, el conjunto de soluciones iniciales o fuentes de alimento de la colonia artificial de abejas, será una matriz de tamaño ***s x [(i + j)k – 1]]***, donde s es la cantidad de fuentes de alimento, i la cantidad de trabajos a procesar, j el número de máquinas por estación y k la cantidad de etapas del sistema productivo; así, cada fila de la matriz representará una solución al problema.

Al igual que en el algoritmo genético hibrido, se utiliza un factor de diversidad del conjunto de soluciones iniciales (fdpi), el cual permite generar ***fdpi*s*** fuentes de alimento, con el objetivo de escoger los s fuentes con el mayor néctar.

9.2. Cálculo del néctar

Teniendo en cuenta que el objetivo del problema es minimizar el *makespan* o tiempo de terminación de las tareas a procesar en el sistema, las fuentes de alimento con menor *makespan* tendrán una mayor cantidad de néctar y por ende serán más atractivas de explotar por las abejas observadoras. La ecuación (14) representa el cálculo del néctar para cada una de las fuentes de alimento de la colonia artificial de abejas, el cual será igual a 1 dividido en el *makespan* de la solución.

$$Néctar = \frac{1}{C_{max}} \quad (14)$$

Por tanto, en primer lugar, se calcula el *makespan* de la fuente de alimento de la misma forma como se explicó en el apartado 8.2, y con ese valor se aplica la formula (14). Así, para el ejemplo de la instancia i5j2k3-1 representado en la figura 13, el néctar será igual a $\frac{1}{815}$.

9.3. Fase de las abejas empleadas

Esta fase se repite tantas veces como fuentes de alimento existan en el conjunto de soluciones iniciales, es decir, s veces, debido a que a cada solución se le es asociada una abeja empleada, con el fin de que explote la fuente de alimento asignada.

9.3.1. Mutación. Uno de los operadores de mutación a utilizar en este algoritmo es el de insertar, el cual remueve un trabajo ubicada en cierta posición y lo inserta en otra posición del vector, para cada una de las etapas de la fuente de alimento, como se muestra en la figura 23 del apartado 8.9. Así mismo, se utilizará el operador de intercambio, el cual como lo indica su nombre, intercambia un trabajo ubicado en cierta posición con una tarea en otra posición, como se muestra en la figura 19 del apartado 8.9

Con el fin de determinar el operador de mutación a utilizar, se genera un número aleatorio entre 0 y 1, con el objetivo de compararlo con la tasa de mutación definida (MR) y aplicar el siguiente procedimiento para cada fuente de alimento (Tasgetiren, Pan, Suganthan , & Chen, 2011):

- Si rand (0,1) < MR, aplicar el operador de mutación insertar.
- Si rand (0,1) ≥ MR, aplicar el operador de mutación intercambio.

9.3.2. Cruce. El operador de cruce a utilizar en este algoritmo se basa en dos puntos, el cual inicia con la selección de los dos puntos de manera aleatoria para cada etapa, cada uno de estos corresponde a un número entero entre 1 y (n+(m-2)), donde n es el número de trabajos y m la cantidad de máquinas por estación. Con los números definidos, el procedimiento de cruce entre la solución titular y la mutada para generar dos nuevas fuentes de solución, es igual que el expuesto en el apartado 8.4 y representado en la figura 16.

El proceso de cruce se ejecutará si un número entre 0 y 1 generado aleatoriamente es mayor a la probabilidad de cruce CR definida.

9.3.3. Selección. Como resultado del proceso de mutación y el de cruce, se obtienen tres nuevas fuentes de alimento, las cuales se deben comparar entre sí y con la titular en términos del néctar, con el fin de seleccionar aquella con la mayor cantidad de néctar y reemplazarla en el conjunto de soluciones iniciales, en dado caso que no sea la titular.

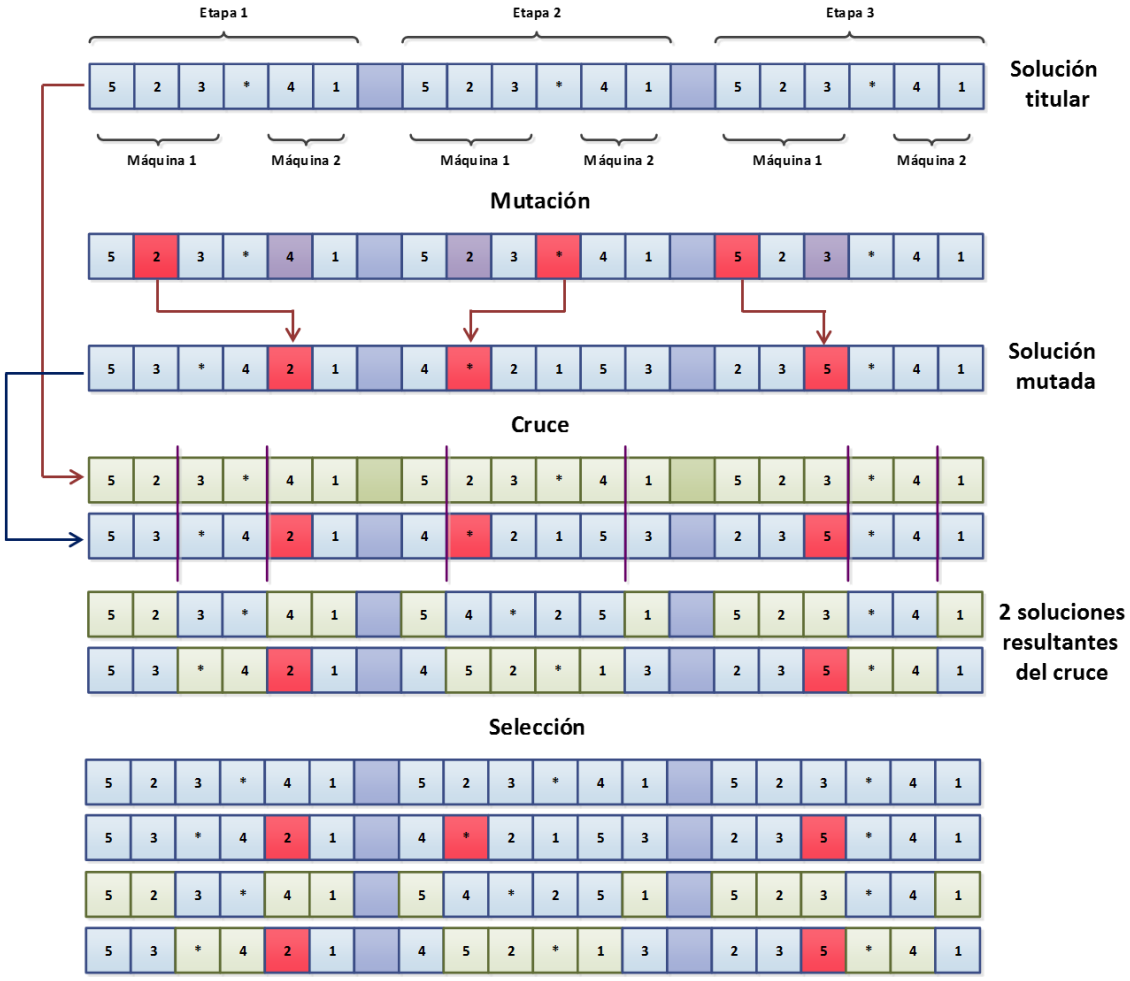


Figura 22. Fase de las abejas empleadas.

La figura 22 representa la fase de las abejas empleadas para una solución de la instancia i5j2k3-1. Como se observa, el operador de mutación utilizado es el de insertar y el operador de cruce se aplica sobre la solución titular y la fuente de alimento resultante del proceso de mutación, con el fin de obtener dos nuevas soluciones al problema. Por último, se escoge la mejor fuente de alimento, es decir, aquella con mayor néctar entre la solución titular, la mutada y las dos resultantes del proceso de cruce. En caso que la mejor solución sea una diferente a la titular, esta se reemplazará en el conjunto de soluciones S, o si la solución titular es la de mejor néctar entre las cuatro, se sigue manteniendo en el conjunto de fuentes de alimento.

9.4. Fase de las abejas observadoras

Esta fase se repite tantas veces como fuentes de alimento existan en el conjunto de soluciones iniciales, es decir, s veces, debido a que el número de abejas observadoras es igual a la cantidad de fuentes de alimento.

En primer lugar, se realiza una selección por ruleta de una fuente de alimento del conjunto de soluciones, la cual se escoge de acuerdo al néctar de cada una, teniendo mayores posibilidades de ser escogidas aquellas con mayor cantidad de néctar, pues estas fuentes de alimento son más atractivas de explotar para las abejas observadoras.

Con la solución seleccionada, se utilizan los operadores de mutación insertar e intercambio, explicados en el apartado 8.9, teniendo en cuenta el siguiente procedimiento, basado en una modificación de la búsqueda variable de la vecindad (VNS) (Cui & Gu, 2015):

1. Aplicar el operador de mutación insertar a la solución seleccionada, si el néctar de la fuente de alimento resultante mejora se actualiza la solución y se realiza el paso 2, de lo contrario termina la búsqueda y el criterio de parada de la fase de las abejas observadoras aumenta en uno.
2. Aplicar el operador de mutación intercambio a la solución actualizada, si el néctar de la fuente de alimento resultante mejora se actualiza la solución y se realiza el paso 1, de lo contrario termina la búsqueda y el criterio de parada de la fase de las abejas observadoras aumenta en uno.

Es importante tener en cuenta que la primera vez que se realice el primer paso, el operador de mutación se aplica sobre la solución seleccionada, pero al momento de aplicar el segundo y nuevamente el primero, si así resulta, el operador de mutación se aplica sobre la solución resultante del proceso de mutación inmediatamente anterior, como se especifica en el diagrama de flujo presentado para el algoritmo hibrido propuesto en esta sección.

9.5. Fase de las abejas exploradoras

Esta fase se repite tantas veces como abejas exploradas tenga la colonia, el cual es un porcentaje de la cantidad de fuentes de alimento o abejas empleadas hayan. El principal objetivo de esta fase es escapar de óptimos locales e incluir nuevas soluciones del problema mediante una búsqueda del entorno predefinido.

En primer lugar, se seleccionan por torneo dos fuentes de alimento del conjunto de soluciones y de ese par se escoge aquella solución con mayor cantidad de néctar, con el fin de aplicar en esa solución procedimientos de construcción y destrucción del algoritmo iterado codicioso (IG).

9.5.1. Destrucción y construcción. Durante la etapa de destrucción d componentes de cada una de las etapas de la solución escogida son eliminados, generando aleatoriamente d números enteros entre 1 y (n+m-1), los cuales indican la posición de cada uno de los trabajos que serán eliminados en cada estación de la solución, obteniendo una serie de trabajos seleccionados y una solución parcial a construir, como lo indica la figura 28.

En la etapa de construcción, se generan aleatoriamente d números enteros entre 1 y (n+m-1), con el fin de ubicar los d trabajos eliminados en la etapa de destrucción en las posiciones que indiquen esos números aleatorios, para cada una de las estaciones del sistema productivo, como se indica en la figura 23.

Considérese que a una solución de la instancia i5j2k3-1 se le va aplicar procedimientos de construcción y destrucción, con un parámetro d=2, siendo 2 y 4 los números aleatorios generados, los cuales indican las posiciones de los trabajos a eliminar, obteniendo así una serie de trabajos eliminados y una solución parcial a construir, como se muestra en la figura 23.

Posteriormente, para la etapa de construcción los números generados aleatoriamente son 1 y 4, por ende, el primer trabajo de cada etapa en la serie de trabajos eliminados se ubica en la primera posición para cada una de las estaciones de la solución parcial, como lo indica la figura 23, obteniendo una serie de trabajos a ubicar y una solución semiconstruida. Posteriormente se ubica en la cuarta posición de cada una de las etapas, la tarea que sigue en la serie de trabajos a ubicar, con el objetivo de obtener una solución construida.

La solución obtenida de los procedimientos de destrucción y construcción reemplazará la peor solución resultante del torneo, sin importar si es mejor o peor que esta.

9.6 Criterio de parada

El algoritmo detendrá su ejecución cuando se cumpla la cantidad de ciclos definidos en los parámetros iniciales. Una vez finalizado se escoge la mejor fuente de alimento del conjunto s en términos de la cantidad del néctar.

Las figuras 24 y 25 muestran un diagrama de flujo que describe el paso a paso del algoritmo híbrido basado en una colonia artificial de abejas.

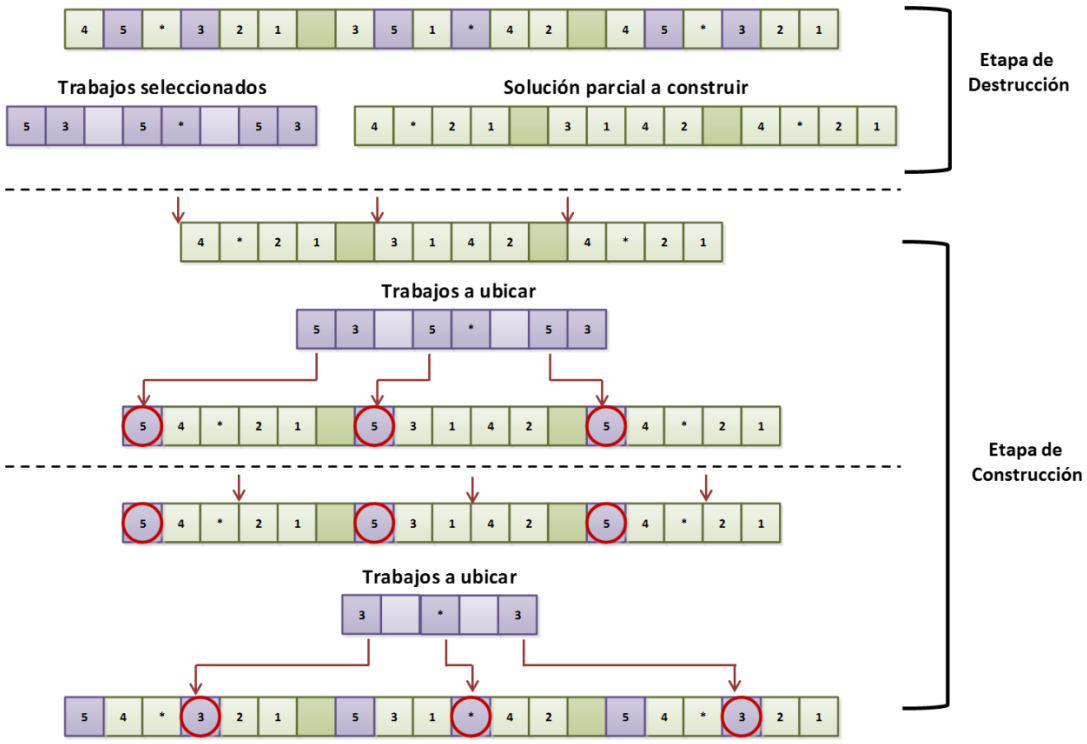


Figura 23. Etapas de construcción y destrucción de la fase de las abejas exploradoras con factor de destrucción 2.

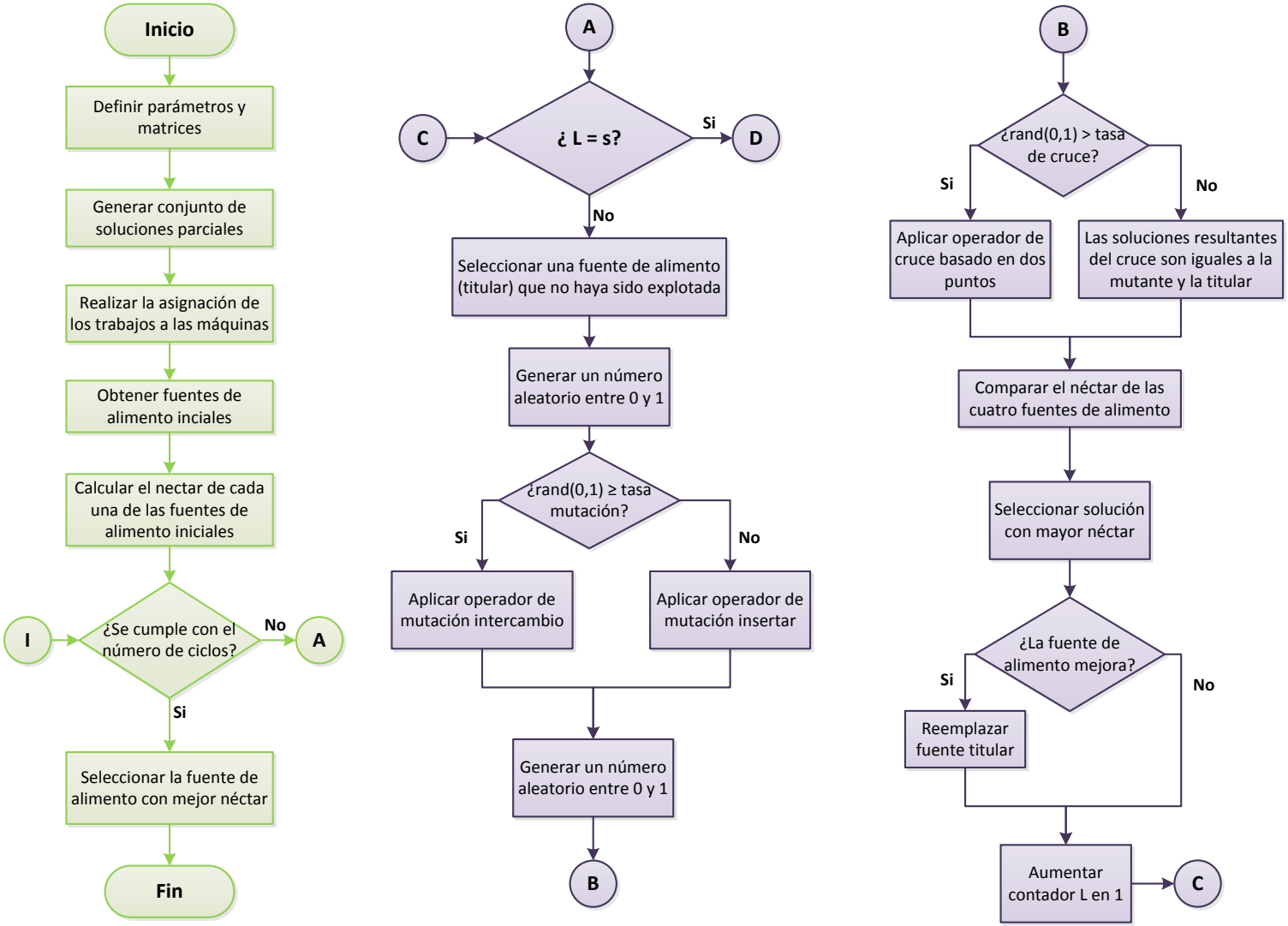


Figura 24. Diagrama de flujo del algoritmo híbrido basado en una colonia artificial de abejas, parte 1.

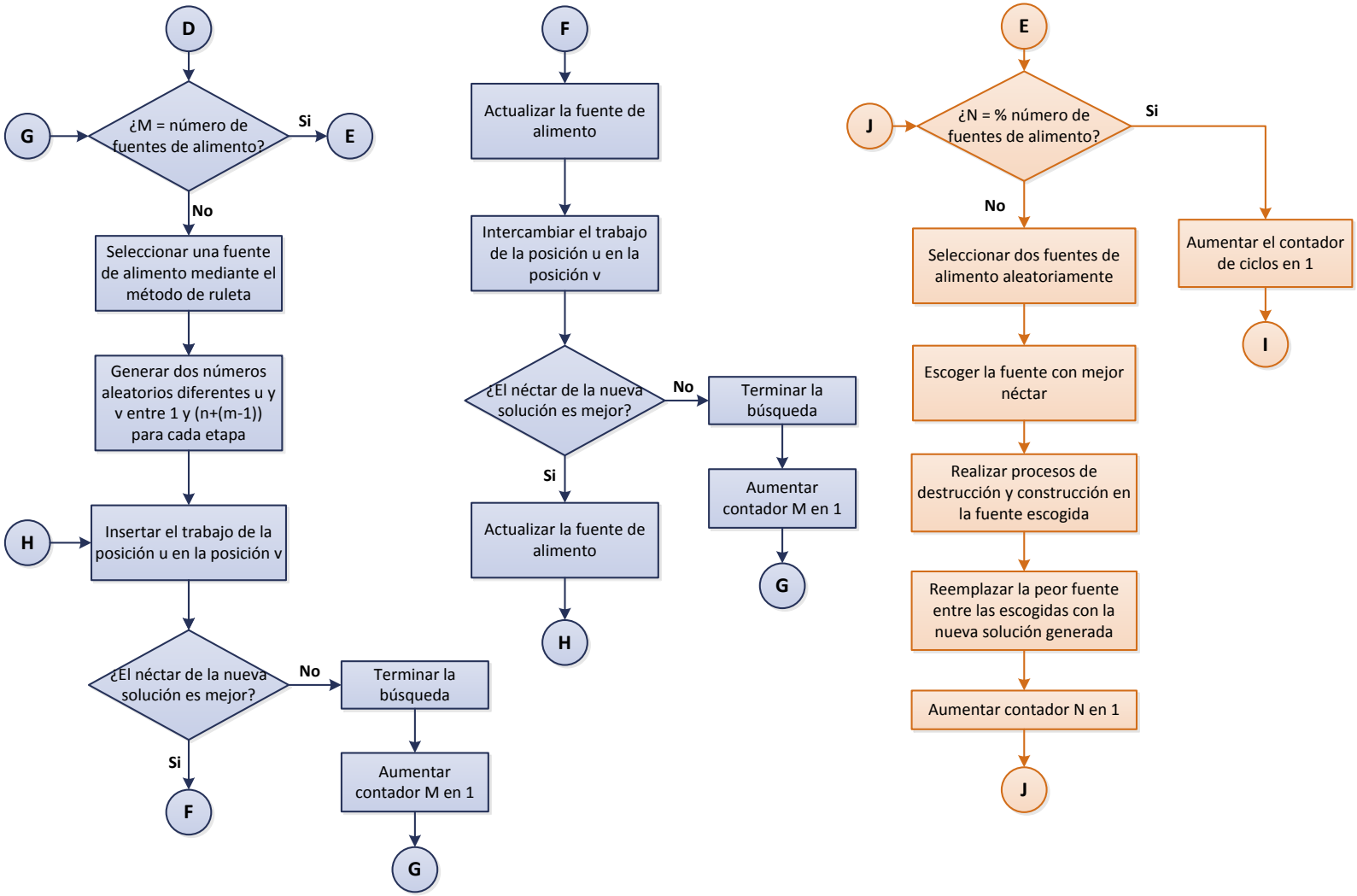


Figura 25. Diagrama de flujo del algoritmo híbrido basado en una colonia artificial de abejas, parte 2.

10. Diseño experimental para la determinación de factores incidentes

10.1 Algoritmo genético híbrido

Para identificar y analizar los factores influyentes sobre la variable respuesta, *makespan*, se realiza un diseño factorial 2^k. En este diseño se consideraron 4 factores k y dos niveles para cada factor.

Los experimentos numéricos se desarrollaron en Matlab versión R2017a en un computador MacBook Pro con procesador 2,7 GHz Intel Core i7, memoria de 4 GB 1333 MHz DDR3 y un sistema operativo de OS X versión 10.11.4, y en un computador con procesador Intel Core i3 2 GHz, memoria de 4 GB y un sistema operativo de 64 bits. Por otra parte, el diseño experimental fue analizado en Minitab 17 en estos mismos ordenadores.

Mediante la revisión de la literatura, se obtuvieron valores de parámetros que diversos autores utilizaron en algoritmos para darle solución a problemas similares al tratado en la presente investigación. Con base en esta revisión de la literatura y teniendo en cuenta el desempeño del algoritmo, en cuanto a la calidad de la respuesta y tiempo de ejecución, los niveles considerados para cada factor se describen en la tabla 2.

Tabla 2.

Niveles para cada factor del algoritmo genético híbrido.

Factor	Bajo	Alto
Tamaño de la población (P)	20	50
Probabilidad de mutación (PM)	0,2	0,5
Número de iteraciones (Int)	100	200
Iteraciones sin mejorar (Ism)	5	10

En la tabla 3 se muestran las 16 posibles combinaciones de los tratamientos del diseño factorial 2^4, cada una de las cuales se ejecutó 3 veces.

Los datos numéricos tenidos en cuenta para el diseño experimental se encuentran en el apéndice C. Así mismo, los resultados del diseño experimental junto con su respectivo análisis para cada una de las 20 instancias, se presentan en el apéndice D.

En la figura 26 se resumen los factores e interacciones con efectos significativos para cada instancia. Como se observa en esa figura, para la mayoría de las instancias el factor tamaño de la población incide significativamente en la variable respuesta y que un nivel alto de este, permite obtener resultados de mayor calidad. De lo anterior, se puede concluir que, como la población inicial no se obtiene aleatoriamente, debido a que la asignación de los trabajos a las máquinas se realiza teniendo en cuenta el tiempo de alistamiento y procesamiento, mediante la ecuación (13) expuesta en el apartado 8.1.2., el proceso de evolución tiene un mejor desempeño, pues empieza a ejecutarse con organismos de buena calidad.

Tabla 3.

Combinaciones de los tratamientos del algoritmo genético híbrido.

Número	**P**	**PM**	**Int**	**Ism**
1	20	0,2	100	5
2	50	0,2	100	5
3	20	0,5	100	5
4	50	0,5	100	5
5	20	0,2	200	5
6	50	0,2	200	5
7	20	0,5	200	5
8	50	0,5	200	5
9	20	0,2	100	10
10	50	0,2	100	10
11	20	0,5	100	10
12	50	0,5	100	10
13	20	0,2	200	10
14	50	0,2	200	10
15	20	0,5	200	10
16	50	0,5	200	10

Instancia	Factores con efecto significativo	Instancia	Factores con efecto significativo
i7j2k3-1	P	i9j2k3-1	P
i7j2k3-2	P		P*PM
i7j3k3-1	P*PM*Int*Ism	i9j2k3-2	P
i7j3k3-2	P	i9j3k3-1	P
	PM*Ism	i9j3k3-2	P
i7j3k3-3	P*PM*Ism	i9j3k3-3	P
i7j2k5-1	P	i9j2k5-1	P*Int
	Ism	i9j2k5-2	P
i7j2k5-2	P	i9j3k5-1	P
i7j3k5-1	----------		PM*Ism
i7j3k5-2	P		P*PM*Int
	P*PM		PM*Int*Ism
	P*PM*Int*Ism	i9j3k5-2	----------
i7j3k5-3	P	i9j3k5-3	P

Figura 26. Efectos significativos para cada instancia del algoritmo genético híbrido.

Por otra parte, con el objetivo de calibrar el algoritmo para realizar las validaciones de este, se establecen parámetros fijos, teniendo en cuenta los valores que minimizan la variable respuesta. Las instancias se agruparon según el número de trabajos a procesar y la cantidad de etapas, estableciendo los parámetros que se encuentran en la figura 27.

Teniendo en cuenta que para unas instancias de cada grupo, la incidencia de un factor o sus interacciones son insignificantes, el nivel que tome esos factores (alto o bajo) no va a incidir en la variable respuesta, como ocurre en la instancia 7i2j3k-1, que solo tiene como factor relevante el tamaño de la población y, por ende, independientemente del valor de los otros factores, la media del *makespan* será aproximadamente igual, y por ende, el valor de esos factores se adoptaran de los niveles que minimicen la variable respuesta de otras instancias del mismo grupo.

Instancia	P	PM	Int	Ism	Instancia	P	PM	Int	Ism
7 trabajos - 3 etapas					**9 trabajos - 3 etapas**				
i7j2k3-1	50	0,50	200	10	i9j2k3-1	50	0,20	200	10
i7j2k3-2					i9j2k3-2				
i7j3k3-1					i9j3k3-1				
i7j3k3-2					i9j3k3-2				
i7j3k3-3					i9j3k3-3				
7 trabajos - 5 etapas					**9 trabajos - 5 etapas**				
i7j2k5-1	50	0,50	100	10	i9j2k5-1	50	0,20	200	5
i7j2k5-2					i9j2k5-2				
i7j3k5-1					i9j3k5-1				
i7j3k5-2					i9j3k5-2				
i7j3k5-3					i9j3k5-3				

Figura 27. Parámetros a utilizar en la validación del algoritmo genético hibrido.

Así mismo, en instancias que tengan como efecto más de una interacción que involucren un mismo factor y el nivel que minimice la variable respuesta sea diferente en cada interacción, se tendrá en cuenta el valor de aquel efecto que sea más relevante en la media del *makespan*, según lo concluido en el apéndice D para cada instancia.

Por último, en caso que un factor minimice la variable respuesta en un nivel alto para una instancia y en un nivel alto para otra del mismo grupo, se analizará la incidencia del parámetro en el diseño del algoritmo, para calibrarlo.

10.2 Colonia artificial de abejas mejorada

Para identificar y analizar los factores influyentes sobre la variable respuesta, *makespan*, se realiza un diseño factorial 2^k. En este diseño se consideraron 5 factores k y dos niveles para cada factor. Los experimentos numéricos se desarrollaron en Matlab versión R2017a en un computador

MacBook Pro con procesador 2,7 GHz Intel Core i7, memoria de 4 GB 1333 MHz DDR3 y un sistema operativo de OS X versión 10.11.4, y en un computador con procesador Intel Core i3 2 GHz, memoria de 4 GB y un sistema operativo de 64 bits. Por otra parte, el diseño experimental fue analizado en Minitab 17 en estos mismos ordenadores.

Al igual que en el algoritmo genético híbrido, los niveles para cada factor que se muestran en la tabla 4, se obtuvieron mediante una revisión de la literatura, de investigaciones de programación de la producción con algoritmos basados en colonias artificiales de abejas.

Tabla 4.

Niveles para cada factor para la colonia artificial de abejas mejorada.

Factor	**Bajo**	**Alto**
Número de fuentes de alimento (F)	10	15
Probabilidad de mutación (PM)	0,3	0,7
Probabilidad de cruce (PC)	0,4	0,8
Número de ciclos (C)	5	10
Parámetro de destrucción (D)	2	4

Tabla 5.

Combinaciones de los tratamientos para la colonia artificial de abejas mejorada.

Número	F	D	PM	PC	C	Número	F	D	PM	PC	C
1	10	2	0,3	0,4	5	**17**	10	2	0,3	0,4	10
2	15	2	0,3	0,4	5	**18**	15	2	0,3	0,4	10
3	10	4	0,3	0,4	5	**19**	10	4	0,3	0,4	10
4	15	4	0,3	0,4	5	**20**	15	4	0,3	0,4	10
5	10	2	0,7	0,4	5	**21**	10	2	0,7	0,4	10
6	15	2	0,7	0,4	5	**22**	15	2	0,7	0,4	10
7	10	4	0,7	0,4	5	**23**	10	4	0,7	0,4	10
8	15	4	0,7	0,4	5	**24**	15	4	0,7	0,4	10
9	10	2	0,3	0,8	5	**25**	10	2	0,3	0,8	10
10	15	2	0,3	0,8	5	**26**	15	2	0,3	0,8	10
11	10	4	0,3	0,8	5	**27**	10	4	0,3	0,8	10
12	15	4	0,3	0,8	5	**28**	15	4	0,3	0,8	10
13	10	2	0,7	0,8	5	**29**	10	2	0,7	0,8	10
14	15	2	0,7	0,8	5	**30**	15	2	0,7	0,8	10
15	10	4	0,7	0,8	5	**31**	10	4	0,7	0,8	10
16	15	4	0,7	0,8	5	**32**	15	4	0,7	0,8	10

En la tabla 5 se muestran las 32 posibles combinaciones de los tratamientos del diseño factorial 2^5, cada una de las cuales se ejecutó 3 veces.

Los datos numéricos tenidos en cuenta para el diseño experimental se encuentran en el apéndice F. Así mismo, los resultados del diseño experimental junto con su respectivo análisis para cada una de las 20 instancias, se presentan en el apéndice G. En la figura 28 se resumen los factores o interacciones con efectos significativos para cada instancia, ordenándolos de manera descendente, con el fin de establecer cual factor o interacción tiene más incidencia en la variable respuesta. Por ejemplo, para la instancia i7j3k5-1, la interacción D*PC*C tiene mayor efecto en el *makespan* que el que genera el factor F.

Instancia	Factores con efecto significativo	Instancia	Factores con efecto significativo
i7j2k3-1	F	i9j2k3-1	----------
i7j2k3-2	F	i9j2k3-2	(F)*(PM)*(C)
	(D)*(PM)*(C)		
	(F)*(PM)*(PC)*(C)		(F)*(PC)*(C)
	(D)*(PC)*(C)		
i7j3k3-1	(F)*(D)*(PM)	i9j3k3-1	F
	(F)*(C)		(D)*(PM)*(PC)*(C)
	(F)*(PM)*(C)		
	F		(F)*(D)*(PM)
i7j3k3-2	PC	i9j3k3-2	(F)*(PM)
	(F)*(PM)*(C)		
	(F)*(D)*(PC)		
	F		
i7j3k3-3	(D)*(PM)*(PC)*(C)	i9j3k3-3	F
			(D)*(C)
i7j2k5-1	F	i9j2k5-1	(F)*(D)*(PM)*(PC)
			(D)*(PM)
i7j2k5-2	(F)*(PM)	i9j2k5-2	F
			(F)*(D)*(PC)*(C)
i7j3k5-1	(D)*(PC)*(C)	i9j3k5-1	F
	F		(F)*(D)*(C)
i7j3k5-2	F	i9j3k5-2	(D)*(PM)*(PC)*(C)
			(F)*(C)
	(PM)*(PC)		F
			(F)*(PM)*(C)
			(PC)*(C)
i7j3k5-3	F	i9j3k5-3	F
	(PM)		(F)*(D)*(PM)*(PC)
	(F)*(D)*(PC)		

Figura 28. Efectos significativos para cada instancia de la colonia artificial de abejas mejorada.

Por otra parte, con el objetivo de calibrar el algoritmo para realizar la validación, se establecen parámetros fijos, teniendo en cuenta los valores que minimizan la variable respuesta, *makespan*. Las instancias se agruparon según el número de trabajos a procesar y la cantidad de etapas, estableciendo los parámetros que se encuentran en la figura 29.

Instancia	F	PC	PM	C	D	Instancia	F	PC	PM	C	D
7 trabajos - 3 etapas						**9 trabajos - 3 etapas**					
i7j2k3-1	15	0,80	0,30	10	2	i9j2k3-1	15	0,80	0,70	10	4
i7j2k3-2						i9j2k3-2					
i7j3k3-1						i9j3k3-1					
i7j3k3-2						i9j3k3-2					
i7j3k3-3						i9j3k3-3					
7 trabajos - 5 etapas						**9 trabajos - 5 etapas**					
i7j2k5-1	15	0,80	0,30	10	2	i9j2k5-1	15	0,80	0,70	10	2
i7j2k5-2						i9j2k5-2					
i7j3k5-1						i9j3k5-1					
i7j3k5-2						i9j3k5-2					
i7j3k5-3						i9j3k5-3					

Figura 29. Parámetros a utilizar en la validación del algoritmo colonia artificial de abejas mejorada.

Al igual que en la calibración del algoritmo genético hibrido, para aquellos factores que no tienen incidencia en la variable respuesta en una instancia, es decir, que independientemente del nivel que se fije para ese factor, el *makespan* no se verá significativamente afectado, el valor se adopta de los niveles que minimicen la variable respuesta de otras instancias del mismo grupo. Así mismo, en instancias que tengan como efecto más de una interacción que involucren un mismo factor y el nivel que minimice la variable respuesta sea diferente en cada interacción, se tendrá en cuenta el valor de aquel efecto que sea más relevante en la media del *makespan*.

Por último, en caso que un factor minimice la variable respuesta en un nivel alto para una instancia y en un nivel alto para otra del mismo grupo, se analizará la incidencia del parámetro en el diseño del algoritmo, para calibrarlo.

11. Validación de los algoritmos

El algoritmo genético híbrido y el algoritmo basado en una colonia artificial de abejas mejorada, fueron programados en Matlab versión R2017a y se ejecutaron en un computador con procesador Intel(R) Core (TM) i5-4570 CPU 3.20GHz, memoria RAM de 8,00 GB y sistema operativo de 64 bits. El algoritmo genético híbrido en lenguaje Matlab se encuentra en el apéndice B y la codificación del algoritmo basado en una colonia artificial de abejas mejorada se detalla en el apéndice E, los cuales se validaron a través de 20 instancias que se detallan en el apéndice A, con el objetivo de comparar las respuestas con las obtenidas mediante el método exacto, desarrollado por Pabón (2017) en su proyecto de grado titulado: "Solución al problema del *flow shop* híbrido (HFS) con máquinas paralelas no relacionadas y *buffers* de tamaño limitado mediante una metaheuristica basada en el algoritmo competitivo imperialista (ICA)".

El modelo de programación lineal entera mixta desarrollado por Pabón (2017), es escrito en lenguaje GAMS y se prueba para las mismas 20 instancias utilizando el solver CPLEX 12.4.0.1 del programa GAMS 23.9.5 en una computadora con procesador Intel Core i5-4570, 3.2 GHz con una memoria RAM de 8 Gigabytes y un sistema operativo de 64 bits. En estas ejecuciones, se establece un tiempo límite de 10800 segundos, por lo que el software no logra hallar la solución

exacta y además da a conocer el % de Gap relativo, que se entiende como la diferencia en porcentaje entre la solución arrojada por el solver y la mejor solución posible del problema (Pabón Serrano, 2017).

Mediante el diseño experimental presentado en el capítulo 10, se logra determinar la mejor combinación de parámetros para cada algoritmo según el grupo de instancias, las cuales se agrupan teniendo en cuenta la cantidad de trabajos y número de etapas de procesamiento. Para el proceso de comparación con los resultados del método exacto, se obtiene el promedio de las 10 réplicas para cada una de las instancias, datos que se encuentran en el apéndice H para el algoritmo genético híbrido y en el apéndice I los del algoritmo basado en una colonia artificial de abejas mejorada.

Para cada uno de los grupos establecidos se presentan los resultados por instancia, indicando los valores de los siguientes parámetros:

- **SC:** Solución conocida obtenida a través del método exacto realizado por Pabón (2017) (en unidades de tiempo).
- **Gap:** Diferencia en porcentaje entre la solución arrojada por el solver y la mejor solución posible del problema.
- **Med:** Media del *makespan* de las diez réplicas (en unidades de tiempo). Este valor se aproxima al siguiente entero si la primera cifra decimal es mayor o igual a 5, o al entero inferior en caso contrario.
- **SD:** Desviación estándar de las cinco réplicas.
- **%Dif:** Diferencia en porcentaje entre la mejor solución conocida mediante el método exacto y la media de las 10 réplicas del algoritmo genético hibrido.

- **MO:** Mejor solución obtenida mediante el algoritmo genético híbrido (en unidades de tiempo).
- **Med-t:** Media de los tiempos de obtención de las respuestas para las 10 réplicas (en segundos).
- **SD-t:** Desviación estándar de los tiempos de obtención de las respuestas para las 10 réplicas (en segundos).

11.1 Algoritmo genético híbrido

A continuación, se presenta una figura para cada grupo de instancias, donde se detallan los resultados del algoritmo genético hibrido y los valores encontrados mediante el método exacto desarrollado por Pabón (2017).

En la figura 30 se observa que para las instancias i7j2k3-1, i7j2k3-2 e i7j3k3-1 el algoritmo genético híbrido encuentra mejores soluciones que la hallada por el método exacto en un tiempo de ejecución de 10800s, aunque para las otras dos instancias la diferencia de la solución del método exacto y la media de las 10 réplicas del algoritmo diseñado no es menor al 6%. Por otra parte, se concluye que la media de los tiempos de completamiento para la instancia i7j2k3-1 es mayor que para la instancia i7j2k3-2, lo cual se puede atribuir al tamaño de los buffers, ya que al tener mayor capacidad se van a presentar menos bloqueos en las máquinas y estas pueden seguir procesando tareas sin atrasar el completamiento de los trabajos.

P = 50 - PM = 0,5 - Int = 200 - Ism = 10								
Instancia	**SC**	**Gap**	**Med**	**SD**	**%Dif**	**MO**	**Med-t**	**SD-t**
i7j2k3-1	873	37.34%	856	10.08	1.92%	841	139.54	4.64
i7j2k3-2	842	9.99%	837	12.45	0.59%	807	147.61	4.99
i7j3k3-1	657	33.31%	652	6.87	0.75%	645	107.33	2.23
i7j3k3-2	628	12.41%	661	10.45	-5.18%	647	105.86	1.88
i7j3k3-3	633	10.59%	658	6.79	-3.93%	645	108.13	1.83

Figura 30. Resultados para instancias con 7 trabajos y 3 etapas de procesamiento.

Por último, para las instancias con 3 máquinas por etapa, la media del *makespan* encontrado por el algoritmo genético híbrido no presenta mucha diferencia cuando aumenta el tamaño del buffer, ya que el existir mayor cantidad de procesadores, los trabajos se distribuyen entre estos y se presentan menos bloqueos o un menor uso de las zonas de almacenamiento.

P = 50 - PM = 0,5 - Int = 100 - Ism = 10								
Instancia	**SC**	**Gap**	**Med**	**SD**	**%Dif**	**MO**	**Med-t**	**SD-t**
i7j2k5-1	1268	43.98%	1123	14.61	11.42%	1096	61.84	2.23
i7j2k5-2	1086	30.17%	1092	9.31	-0.56%	1082	67.31	1.33
i7j3k5-1	937	26.04%	936	13.94	0.13%	917	54.11	3.65
i7j3k5-2	925	23.41%	928	10.19	-0.32%	914	54.65	2.19
i7j3k5-3	973	21.37%	951	7.79	2.26%	940	88.75	1.20

Figura 31. Resultados para instancias con 7 trabajos y 5 etapas de procesamiento.

Con respecto al grupo de instancias con 7 trabajos y 5 etapas de procesamiento, se observa en la figura 31 que para las instancias i7j2k5-1, i7j3k5-1 e i7j3k5-3 el algoritmo genético híbrido encuentra mejores soluciones que la hallada por el método exacto en un tiempo de ejecución de 10800s. En cuanto a la instancia i7j2k5-1, la diferencia entre la media de las 10 réplicas del algoritmo diseñado y la solución encontrada en GAMS supera el 10%, siendo la instancia que tiene

un mayor Gap de las 5 presentadas en la figura. Por otra parte, aunque las medias del *makespan* de las instancias i7j2k5-2 e i7j3k5-2 es mayor que la solución encontrada por el método exacto en un tiempo de ejecución de 10800s, el algoritmo genético híbrido en una de las 10 réplicas logra encontrar una mejor solución que la hallada en GAMS para las dos instancias.

De igual manera, se aprecia que, para las instancias con 2 máquinas de procesamiento por estación, el tamaño del buffer incide en el valor del *makespan*, ya que la media disminuye al aumentar la capacidad de las zonas de almacenamiento.

P = 50 - PM = 0,2 - Int = 200 - Ism = 10								
Instancia	**SC**	**Gap**	**Med**	**SD**	**%Dif**	**MO**	**Med-t**	**SD-t**
i9j2k3-1	1014	55.22%	991	6.65	2.30%	978	138.29	4.89
i9j2k3-2	942	46.32%	942	17.17	0,00%	913	68.10	1.20
i9j3k3-1	831	50.41%	715	10.27	14.02%	701	50.60	1.16
i9j3k3-2	709	36.52%	721	10.87	-1.65%	706	54.64	1.24
i9j3k3-3	700	48.48%	718	12.19	-2.56%	701	54.70	1.29

Figura 32. Resultados para instancias con 9 trabajos y 3 etapas de procesamiento.

Por otra parte, para las instancias con 9 trabajos y 3 etapas de procesamiento, se observa en la figura 32 que para las instancias i9j3k3-2 e i9j3k3-3, el algoritmo genético híbrido no logra llegar a la solución encontrada por el método exacto, aunque la diferencia es menor al 3%. Para la instancia 19j2k3-2 la media de las 10 réplicas ejecutadas para el algoritmo diseñado es igual a la solución encontrada por el método exacto, mientras que para las instancias i9j2k3-1 e i9j3k3-1 el algoritmo genético híbrido logra tener mejores soluciones que la hallada en GAMS durante una ejecución de 10800s, siendo los Gaps de estas instancias mayores a 50%.

P = 50 - PM = 0,2 - Int = 200 - Ism = 5								
Instancia	**SC**	**Gap**	**Med**	**SD**	**%Dif**	**MO**	**Med-t**	**SD-t**
i9j2k5-1			1286	16.34		1257	84.99	3.27
i9j2k5-2	1260	46.62%	1281	5.98	-1.68%	1272	146.39	3.18
i9j3k5-1			1023	11.72		997	120.58	2.47
i9j3k5-2	1016	34.44%	1024	12.26	-0.75%	999	234.28	54.97
i9j3k5-3	1005	37.01%	1007	19.75	-0.20%	980	250.84	9.45

Figura 33. Resultados para instancias con 9 trabajos y 5 etapas de procesamiento.

Para el último grupo de instancias (9 trabajos y 5 etapas de procesamiento), el solver no logró converger en una solución acorde al contexto del problema dentro de los límites de tiempo establecidos para las instancias i9j2k5-1 e i9j3k5-1, lo cual se puede presentar por una restricción del modelo de programación lineal entera mixta que exige mucha capacidad computacional (Pabón Serrano, 2017), por tanto no se puede realizar la comparación de los resultados encontrados con el algoritmo genético hibrido y el método exacto para esas dos instancias. En cuanto a las otras tres instancias del grupo, la media del algoritmo diseñado no logra tomar el valor hallado por el método exacto, aunque para las instancias i9j3k5-2 e i9j3k5-3, se logra llegar a una mejor respuesta en las 10 réplicas realizadas.

11.2 Colonia artificial de abejas mejorada

A continuación, se presenta una figura para cada grupo de instancias, donde se detallan los resultados del algoritmo basado en una colonia artificial de abejas mejorada y los valores encontrados mediante el método exacto desarrollado por Pabón (2017).

F = 15 - PC = 0,8 - PM = 0,3 - C = 10 - D = 2								
Instancia	SC	Gap	Med	SD	%Dif	MO	Med-t	SD-t
i7j2k3-1	873	37,34%	872	20,76	0,09%	836	37,27	1,19
i7j2k3-2	842	9,99%	863	28,82	-2,48%	811	42,81	1,19
i7j3k3-1	657	33,31%	664	13,34	-1,00%	645	27,00	0,53
i7j3k3-2	628	12,41%	672	16,89	-6,94%	647	28,70	0,58
i7j3k3-3	633	10,59%	673	12,00	-6,32%	656	29,29	0,92

Figura 34. Resultados para instancias con 7 trabajos y 3 etapas de procesamiento.

En la figura 34 se observa que solo para la instancia i7j2k3-1 el algoritmo basado en una colonia artificial de abejas modificada, logra un menor valor medio de *makespan* que el método exacto ejecutado en GAMS, aunque la diferencia entre los valores es solo por 1 unidad de tiempo. Es importante resaltar que para las instancias i7j2k3-2 e i7j3k3-1, en una de las 10 corridas, el algoritmo diseñado logra tener un mejor resultado que el hallado por el solver.

F = 15 - PC = 0,8 - PM = 0,3 - C = 10 - D = 2								
Instancia	SC	Gap	Med	SD	%Dif	MO	Med-t	SD-t
i7j2k5-1	1268	43,98%	1144	18,04	9,76%	1120	45,56	0,93
i7j2k5-2	1086	30,17%	1114	11,05	-2,54%	1094	54,59	1,72
i7j3k5-1	937	26,04%	952	16,32	-1,62%	928	34,98	0,92
i7j3k5-2	925	23,41%	944	12,16	-2,10%	929	35,97	1,23
i7j3k5-3	973	21,37%	956	11,75	1,76%	942	39,34	1,42

Figura 35. Resultados para instancias con 7 trabajos y 5 etapas de procesamiento.

Como se puede observar en la figura 35 para las instancias i7j2k5-1 e i7j3k5-3, el algoritmo diseñado arroja en promedio mejores soluciones que las halladas por el método exacto en un tiempo de ejecución de 10800s. Respecto a las instancias restantes, aunque el método exacto arroja una mejor solución, la diferencia que hay entre la media y la solución encontrada es muy pequeña,

además que para las instancias i7j3k5-1 e i7j3k5-3 el algoritmo en una de las 10 réplicas logra encontrar una mejor solución que la hallada en GAMS.

F = 15 - PC = 0,8 - PM = 0,7 - C = 10 - D = 2								
Instancia	SC	Gap	Med	SD	%Dif	MO	Med-t	SD-t
i9j2k3-1	1014	55,22%	1007	14,91	0,71%	982	52,93	1,46
i9j2k3-2	942	46,32%	970	20,84	-2,92%	940	85,93	1,13
i9j3k3-1	831	50,41%	727	20,46	12,53%	693	93,08	4,57
i9j3k3-2	709	36,52%	723	20,48	-1,97%	687	66,37	2,31
i9j3k3-3	700	48,48%	729	15,89	-4,17%	711	108,41	3,39

Figura 36. Resultados para instancias con 9 trabajos y 3 etapas de procesamiento.

En la figura 36 se observa que en las instancias i9j2k3-1 e i9j3k3-1 el algoritmo basado en una colonia artificial de abejas mejorada encuentra mejores soluciones que las halladas por el método exacto en un tiempo de ejecución de 10800s. En cuanto a las instancias i9j3k3-2, i9j3k3-2 e i9j3k3-3, la diferencia entre la media de las 10 réplicas del algoritmo diseñado y la solución encontrada en GAMS no supera el 5%, además que, en dos de estas instancias, en una de las 10 réplicas el algoritmo logra encontrar una mejor solución que la hallada en GAMS.

F = 15 - PC = 0,8 - PM = 0,7 - C = 10 - D = 2								
Instancia	SC	Gap	Med	SD	%Dif	MO	Med-t	SD-t
i9j2k5-1			1306	10,02		1291	139,14	11,43
i9j2k5-2	1260	46,62%	1308	19,38	-3,82%	1282	192,58	5,98
i9j3k5-1			1041	15,46		1019	95,31	2,52
i9j3k5-2	1016	34,44%	1041	23,59	-2,42%	1011	103,75	3,39
i9j3k5-3	1005	37,01%	1049	19,12	-4,41%	1012	102,64	3,42

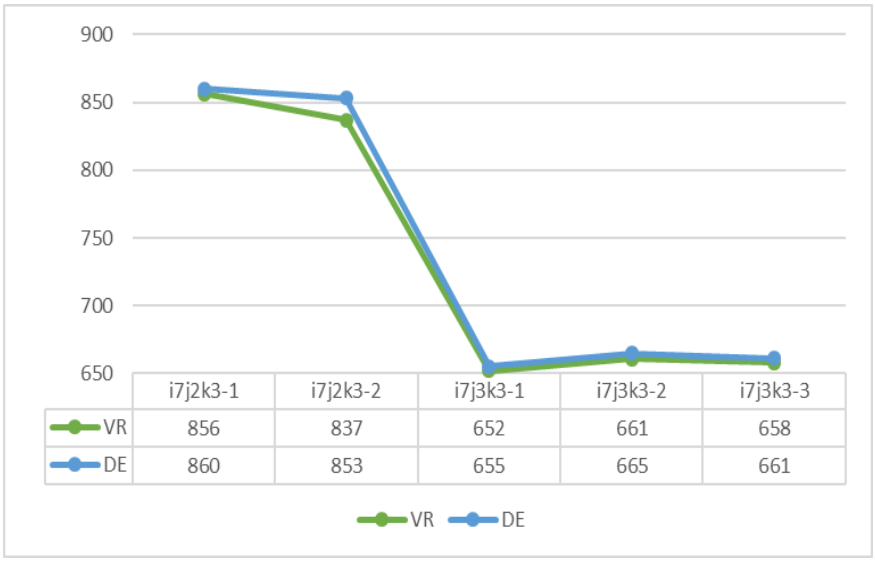
Figura 37. Resultados para instancia con 9 trabajos y 5 etapas de procesamiento.

Como se mencionó en el apartado 11.1 el solver no logró converger en una solución acorde al contexto del problema dentro de los límites de tiempo establecidos para las instancias i9j2k5-1 e i9j3k5-1, por tanto, no se puede realizar la comparación de los resultados encontrados con el algoritmo basado en una colonia artificial de abejas mejorada y el método exacto para esas dos instancias. En cuanto a las otras tres instancias, la media del algoritmo diseñado no logra igualar o mejorar valor hallado por el método exacto en el tiempo computacional predeterminado, aunque para la instancia i9j3k5-2, se logra llegar a una mejor respuesta en las 10 réplicas realizadas.

Es importante tener en cuenta, que el método exacto ejecutado en GAMS no logra llegar a la solución exacta para ninguna de las 20 instancias y por esto, el algoritmo genético hibrido o el algoritmo basado en una colonia artificial de abejas modificada, superan la solución encontrada en el solver para algunas instancias, pues sería incorrecto afirmar que una metaheuristica supera los resultados de un método que permite obtener una solución óptima, es decir, aquella que produce el mejor valor de la función objetivo (Maccarthy & Liu, 1993). En los Gaps encontrados por el solver, es fácil evidenciar que la solución encontrada es lejana a la exacta, puesto que estos valores oscilan entre el 9,99% y 55,22%.

11.3 Comparación entre los resultados tenidos en cuenta para el diseño experimental y los de la validación.

Para verificar que la calibración de los algoritmos fue la correcta, se realizó una comparación entre los resultados tenidos en cuenta para cada uno de los diseños experimentales (apéndice C y apéndice F) y los resultados que se obtuvieron en la validación de estos (apéndice H y apéndice I).



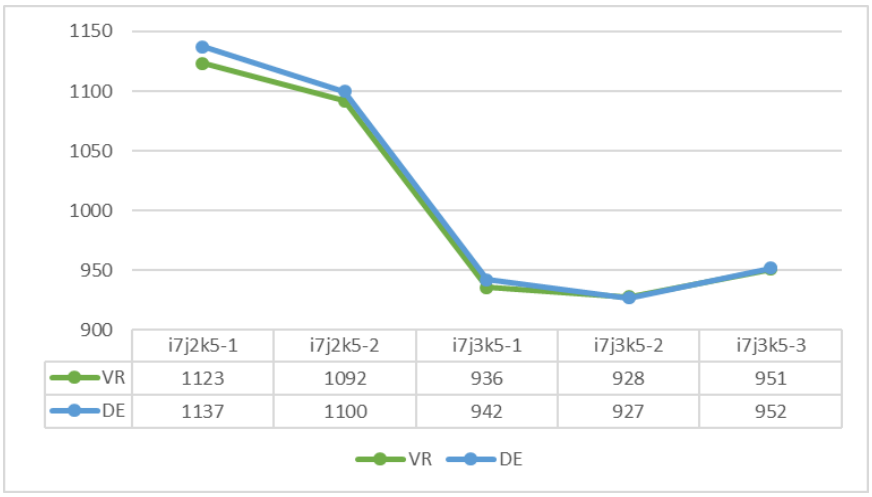


Figura 38. Comparación de las medias por instancia con 7 trabajos entre los resultados tenidos en cuenta para el diseño experimental del algoritmo genético hibrido y los obtenidos en la validación del algoritmo.

La comparación se realizó entre la media del tiempo de completamiento obtenido en la validación de los algoritmos y el promedio del *makespan* de los trabajos de todas las combinaciones, exceptuando aquellos valores resultantes de la combinación que se utilizó en la validación, para cada instancia.

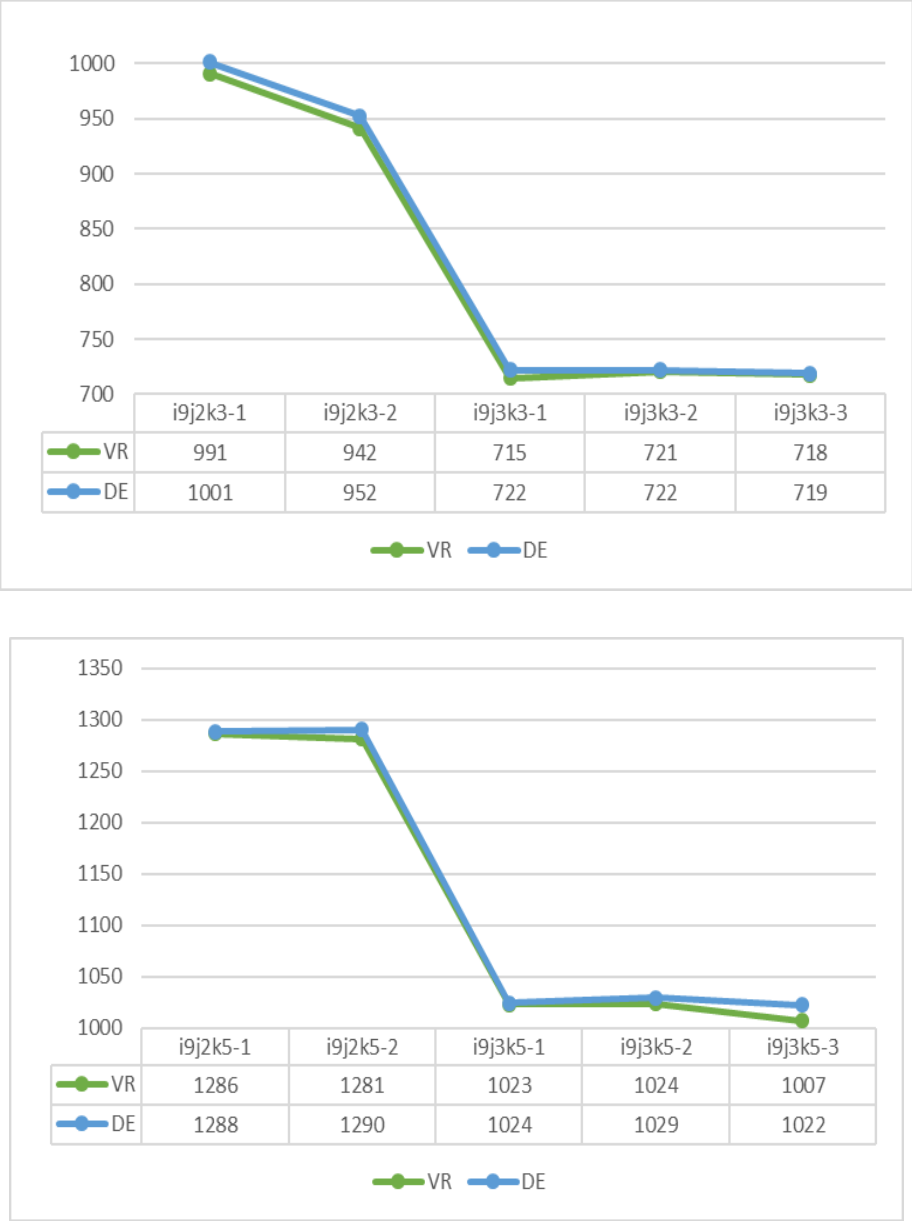


Figura 39. Comparación de las medias por instancia con 9 trabajos entre los resultados tenidos en cuenta para el diseño experimental del algoritmo genético hibrido y los obtenidos en la validación del algoritmo.

En la figura 38 y 39, se puede observar que solo para la instancia i7j3k5-2 se obtiene mejores resultados en los datos utilizados en el diseño experimental que en los obtenidos en la validación

del algoritmo genético hibrido. Por ende, se concluye que la calibración del algoritmo se realizó de la manera adecuada.

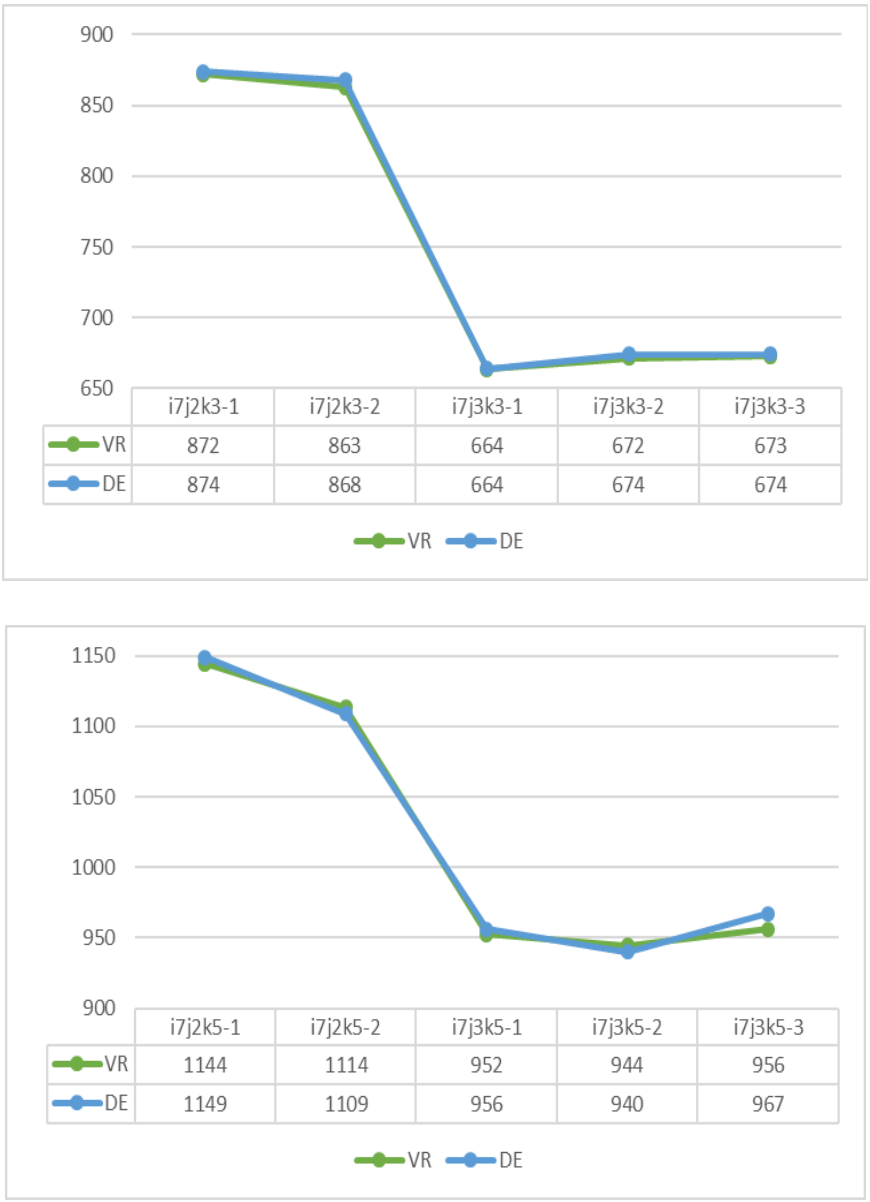


Figura 40. Comparación de las medias por instancia con 7 trabajos entre los resultados tenidos en cuenta para el diseño experimental del algoritmo basado en una colonia artificial de abejas mejorada y los obtenidos en la validación del algoritmo.

Por otra parte, se evidencia en la figura 40 y 41 que, en el algoritmo basado en una colonia artificial de abejas mejorada, para 14 de las 20 instancias se obtienen menores valores del *makespan* con la calibración del algoritmo. Aunque es importante señalar que, para las 6 instancias restantes, la diferencia en la media del tiempo de completamiento de los trabajos es pequeña.

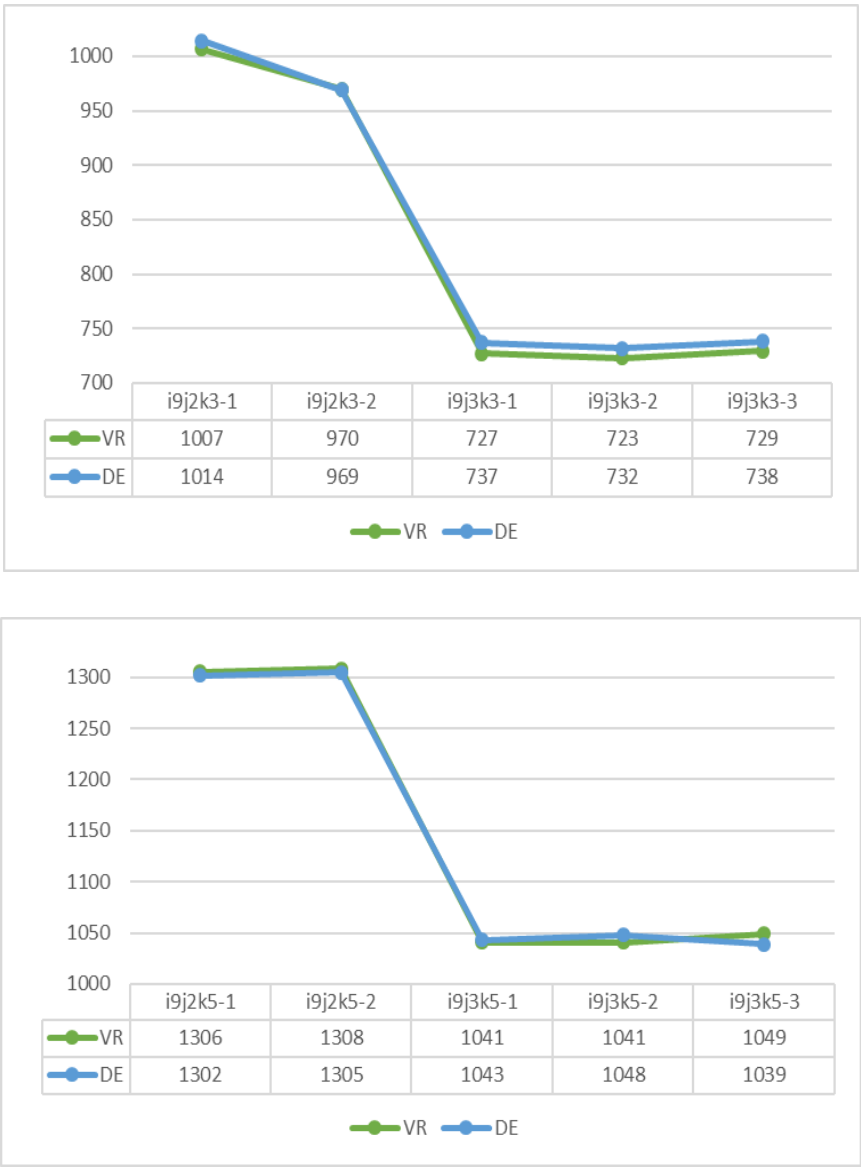


Figura 41. Comparación de las medias por instancia con 9 trabajos entre los resultados tenidos en cuenta para el diseño experimental del algoritmo basado en una colonia artificial de abejas mejorada y los obtenidos en la validación del algoritmo.

12. Comparación entre los resultados del AGH, IABC e ICA

Los resultados obtenidos en la validación de cada uno de los algoritmos, se comparan con los hallados por Pabón (2017) en su proyecto de grado titulado: "Solución al problema del *flow shop* híbrido (HFS) con máquinas paralelas no relacionadas y *buffers* de tamaño limitado mediante una metaheurística basada en el algoritmo competitivo imperialista (ICA)", resultados que se encuentran en el apéndice J. La comparación se realiza entre la media del total de corridas de cada algoritmo, la mejor solución encontrada y el promedio del tiempo de ejecución, para cada una de las 20 instancias.

A continuación, se presenta una tabla por cada grupo de instancias, en donde se detallan los resultados del algoritmo genético híbrido (AGH), la metaheurística basada en el algoritmo competitivo imperialista (ICA) y el algoritmo basado en una colonia artificial de abejas mejorada (IABC). Además, se presenta una gráfica por grupo de instancias, para cada uno de los parámetros a comparar.

	ICA			AGH			IABC		
Instancia	Med	MO	Med-t	Med	MO	Med-t	Med	MO	Med-t
i7j2k3-1	906	882	14,31	856	841	139,54	872	836	37,27
i7j2k3-2	896	853	13,48	837	807	147,61	863	811	42,81
i7j3k3-1	720	697	12,78	652	645	107,33	664	645	27,00
i7j3k3-2	719	692	12,75	661	647	105,86	672	647	28,70
i7j3k3-3	713	675	13,43	658	645	108,13	673	656	29,29

Figura 42. Resultados del ICA, AGH e IABC para instancias con 7 trabajos y 3 etapas de procesamiento.

En cuanto a las instancias con 7 trabajos y 3 etapas de procesamiento, el algoritmo genético híbrido genera en promedio mejores soluciones que el ICA y el IABC, aunque los dos algoritmos diseñados en este proyecto tienen promedios similares para este grupo de instancias, lo que se evidencia en la figura 43. Por otra parte, en la figura 44 se observa que el AGH e IABC en las instancias i7j3k3-1 e i7j3k3-2 obtiene el mismo valor para la mejor solución, mientras que para la instancia i7j2k3-1 el IABC obtiene una mejor solución que el AGH y en las dos instancias restantes el algoritmo genético híbrido logra un menor valor de *makespan*. Cabe resaltar que, tanto para la media como para la mejor solución, los resultados del ICA son mayores a los algoritmos propuestos.

Respecto al tiempo de ejecución, en la figura 43 se observa que la metaheuristica basada en el algoritmo competitivo imperialista, obtiene resultados en tiempos inferiores a 15 segundos, mientras que el tiempo de ejecución del algoritmo genético hibrido oscila entre 100 y 150 segundos, siendo menores para instancias con mayor número de máquinas en las etapas de procesamiento. Así mismo, el tiempo de ejecución del IABC presenta el mismo comportamiento que el AGH, aunque estos oscilan entre 25 y 50 segundos.

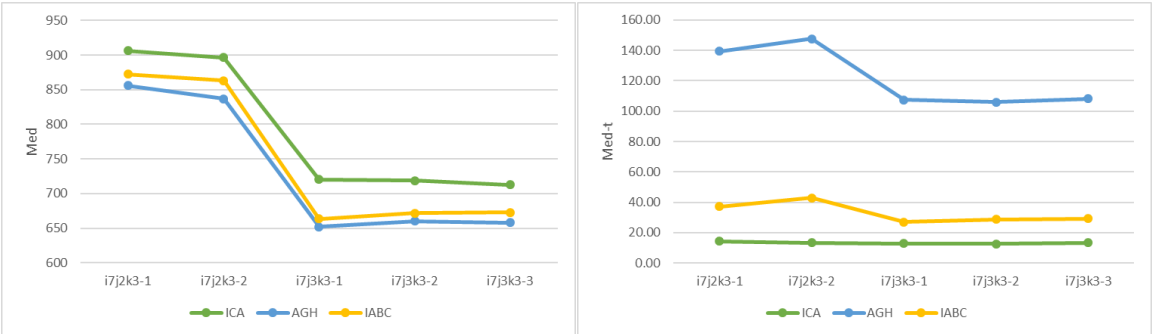


Figura 43. Gráfica de la media de *makespan* y tiempo de ejecución para instancias de 7 trabajos y 3 etapas de procesamiento de cada uno de los algoritmos.

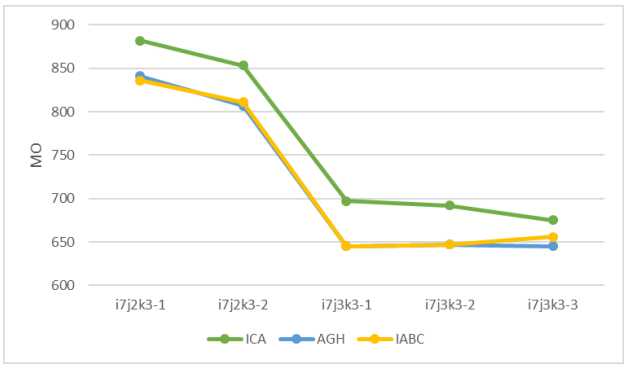


Figura 44. Gráfica de la mejor solución obtenida para instancias de 7 trabajos y 3 etapas de procesamiento de cada uno de los algoritmos.

	ICA			AGH			IABC		
Instancia	Med	MO	Med-t	Med	MO	Med-t	Med	MO	Med-t
i7j2k5-1	1230	1190	24,22	1123	1096	61,84	1144	1120	45,56
i7j2k5-2	1206	1168	24,18	1092	1082	67,31	1114	1094	54,59
i7j3k5-1	1069	1024	21,11	936	917	54,11	952	928	34,98
i7j3k5-2	1048	1012	21,33	928	914	54,65	944	929	35,97
i7j3k5-3	1064	1035	21,60	951	940	88,75	956	942	39,34

Figura 45. Resultados del ICA, AGH e IABC para instancias con 7 trabajos y 5 etapas de procesamiento.

Para las instancias con 7 trabajos y 5 etapas de procesamiento, en la figura 46 se evidencia que el algoritmo genético híbrido genera en promedio mejores soluciones que el ICA y el IABC, aunque presenta un comportamiento en los valores del *makespan* similar al algoritmo basado en una colonia artificial de abejas mejorada. En cuanto a la mejor solución encontrada, los dos algoritmos propuestos obtienen valores de *makespan* menores al ICA, siendo el AGH el algoritmo que obtuvo mejores soluciones, como se muestra en la figura 47.

Respecto a los tiempos de ejecución, en la figura 46 se observa que, el ICA logra obtener los resultados en tiempos computacionales entre 20 y 25 segundos, mientras que los dos algoritmos

propuestos tienen tiempo de ejecución mayores, oscilando el AGH entre 50 y 90 segundos y el IABC entre 30 y 60 segundos.

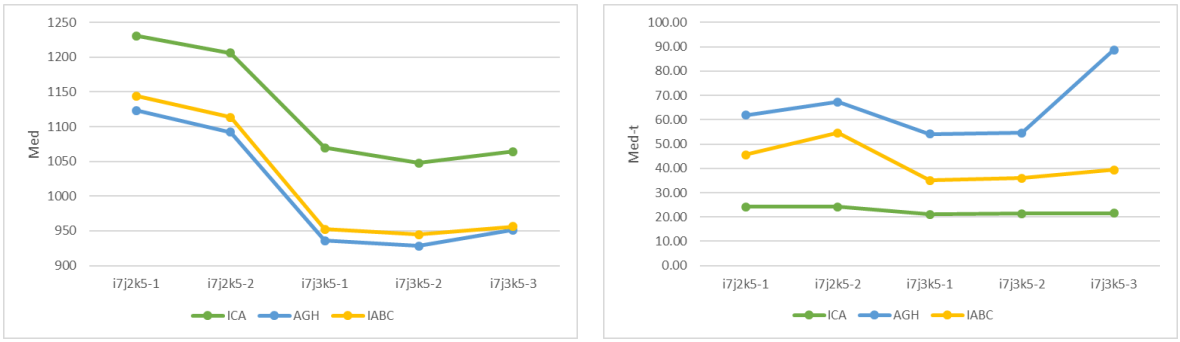


Figura 46. Gráfica de la media de *makespan* y tiempo de ejecución para instancias de 7 trabajos y 5 etapas de procesamiento de cada uno de los algoritmos.

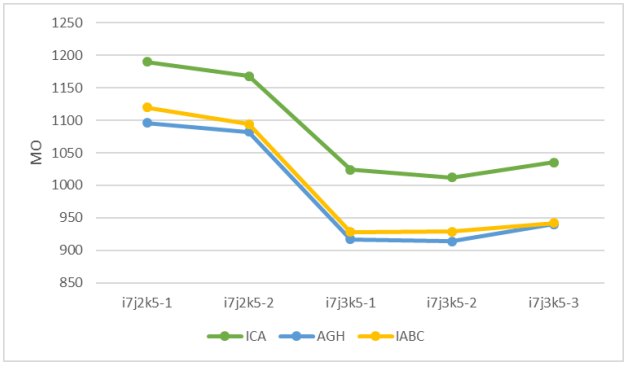


Figura 47. Gráfica de la mejor solución obtenida para instancias de 7 trabajos y 5 etapas de procesamiento de cada uno de los algoritmos.

	ICA			AGH			IABC		
Instancia	Med	MO	Med-t	Med	MO	Med-t	Med	MO	Med-t
i9j2k3-1	1055	1020	20,05	991	978	138,29	1007	982	52,93
i9j2k3-2	1007	973	20,81	942	913	68,10	970	940	85,93
i9j3k3-1	860	801	19,14	715	701	50,60	727	693	93,08
i9j3k3-2	842	773	19,01	721	706	54,64	723	687	66,37
i9j3k3-3	836	796	18,72	718	701	54,70	729	711	108,41

Figura 48. Resultados del ICA, AGH e IABC para instancias con 9 trabajos y 3 etapas de procesamiento.

En la figura 49 se observa que, para las instancias con 9 trabajos y 3 etapas de procesamiento, el algoritmo genético híbrido genera en promedio mejores soluciones que el ICA y el IABC, sin embargo, los promedios entre los algoritmos propuestos son muy cercanos. En cuanto a la mejor solución el IABC supera al AGH en las instancias i9j3k3-1 e i9j3k3-2, mientras que para las instancias restantes el AGH logra mejores soluciones.

Figura 49. Gráfica de la media de *makespan* y tiempo de ejecución para instancias de 9 trabajos y 3 etapas de procesamiento de cada uno de los algoritmos.

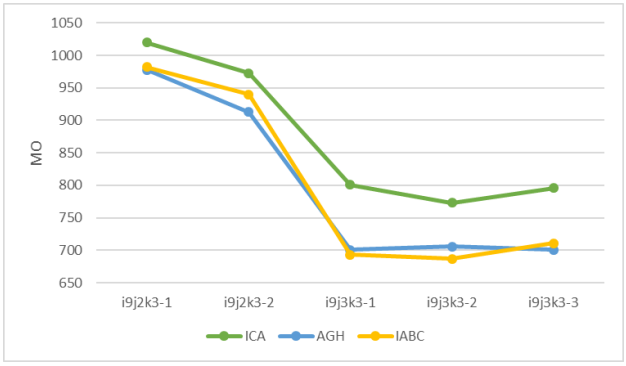


Figura 50. Gráfica de la mejor solución obtenida para instancias de 9 trabajos y 3 etapas de procesamiento de cada uno de los algoritmos.

	ICA			AGH			IABC		
Instancia	Med	MO	Med-t	Med	MO	Med-t	Med	MO	Med-t
i9j2k5-1	1470	1399	31,60	1286	1257	84,99	1306	1291	139,14
i9j2k5-2	1460	1402	33,16	1281	1272	146,39	1308	1282	192,58
i9j3k5-1	1277	1215	29,70	1023	997	120,58	1041	1019	95,31
i9j3k5-2	1256	1168	29,58	1024	999	234,28	1041	1011	103,75
i9j3k5-3	1239	1187	28,82	1007	980	250,84	1049	1012	102,64

Figura 51. Resultados del ICA, AGH e IABC para instancias con 9 trabajos y 5 etapas de procesamiento.

En la figura 49 se observa que, el tiempo de ejecución del ICA es menor que el requerido por los dos algoritmos propuestos para arrojar una respuesta. Así mismo, el IABC tiene un tiempo de ejecución menor que el AGH para la instancia i9j2k3-1, mientras que, para las cuatro instancias restantes, el AGH logra obtener respuestas en tiempos computacionales menores.

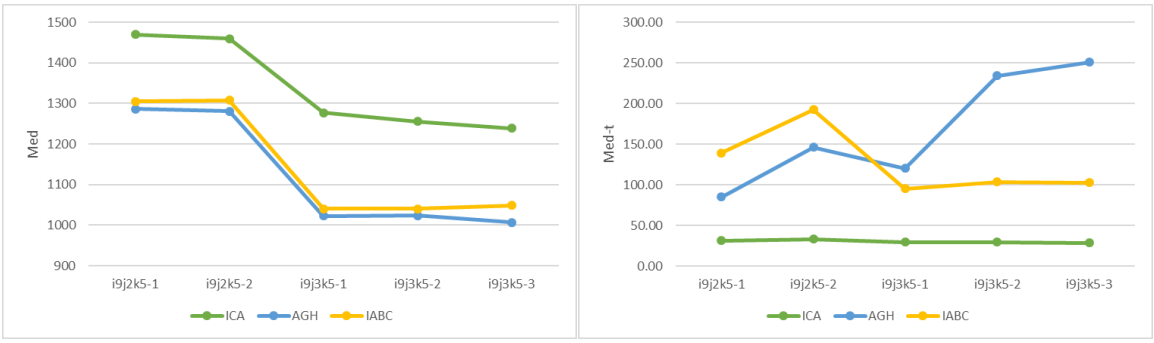


Figura 52. Gráfica de la media de *makespan* y tiempo de ejecución para instancias de 9 trabajos y 5 etapas de procesamiento de cada uno de los algoritmos.

En cuanto a las instancias con 9 trabajos y 5 etapas de procesamiento, el algoritmo genético híbrido genera en promedio mejores soluciones que el ICA y el IABC, aunque los dos algoritmos diseñados en este proyecto tienen promedios similares para este grupo de instancias, lo que se evidencia en la figura 52. Por otra parte, en la figura 53 se observa que el AGH obtiene una mejor

solución en todas las instancias que el IABC. Cabe resaltar que, tanto para la media como para la mejor solución, los resultados del ICA son mayores a los algoritmos propuestos.

Respecto al tiempo de ejecución, en la figura 52 se observa que la metaheurística basada en el algoritmo competitivo imperialista, obtiene resultados en tiempos inferiores a 35 segundos, siendo menores a los requeridos por los algoritmos propuestos para obtener una respuesta. Así mismo, para las instancias con 2 máquinas en cada etapa de procesamiento, el AGH requiere un menor tiempo computacional que el IABC, mientras que, para aquellas instancias con 3 procesadores por estación el IABC requiere menor tiempo de ejecución que el AGH.

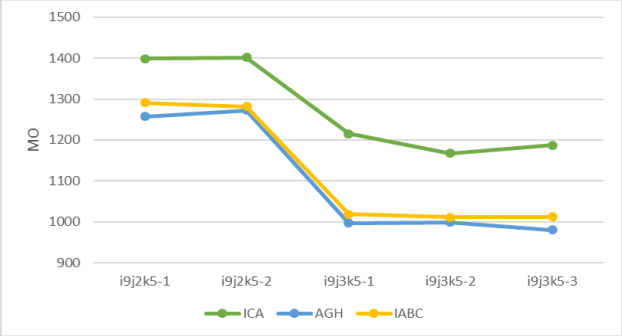


Figura 53. Gráfica de la mejor solución obtenida para instancias de 9 trabajos y 5 etapas de procesamiento de cada uno de los algoritmos.

13. Conclusiones

Mediante la revisión de la literatura, se encontró que el problema del *flow shop* hibrido con *buffers* de tamaño limitado ha sido poco abordado, a diferencia de considerar el sistema productivo con zonas de almacenamiento intermedias de tamaño ilimitado. De la misma manera, se encuentra que, en problemas de gran complejidad computacional, como el estudiado en el presente proyecto, el uso de metaheurísticas permiten obtener resultados de buena calidad en tiempos computacionales pequeños, dado que, para la mayoría de las instancias, los resultados de los algoritmos propuestos se aproximaron o superaron la solución encontrada por el método exacto en un tiempo computacional de 10800s.

Así mismo, en los diseños experimentales se encontró que el tamaño de la población incidía significativamente en la calidad de la variable respuesta en el algoritmo genético híbrido, al igual que la cantidad de fuentes de alimento en el algoritmo basado en una colonia artificial de abejas mejorada. Esto se puede atribuir a que las soluciones iniciales de cada algoritmo, se generaron teniendo en cuenta el tiempo de alistamiento y procesamiento de los trabajos, asignando las tareas a las máquinas que permitieran minimizar el tiempo de salida de la respectiva estación.

El algoritmo genético híbrido permite encontrar mejores resultados que los hallados por el método exacto para 9 de las 20 instancias, mientras que el IABC logra mejores resultados que los encontrados en GAMS para 5 instancias. Además, los dos algoritmos encuentran solución para las instancias i9j2k5-1 e i9j3k5-1, que el modelo de programación entera mixta ejecutado en GAMS no logra converger en una solución acorde al contexto del problema, dentro de los límites de

tiempo establecidos. Es importante tener en cuenta, que el método exacto ejecutado en GAMS no logra llegar a la solución exacta para ninguna de las 20 instancias y por esto, los algoritmos propuestos superan la solución encontrada en el solver para algunas instancias, pues sería incorrecto afirmar que una metaheuristica supera los resultados de un método que permite obtener una solución óptima, es decir, aquella que produce el mejor valor de la función objetivo (Maccarthy & Liu, 1993). En los Gaps encontrados por el solver, es fácil evidenciar que la solución encontrada es lejana a la exacta, puesto que estos valores oscilan entre el 9,99% y 55,22%.

Mediante la comparación de los dos algoritmos propuestos, se evidencia que en promedio el algoritmo genético híbrido permite obtener mejores resultados que el algoritmo basado en una colonia artificial de abejas mejorada, sin embargo, la media de las soluciones para cada instancia del IABC no difieren significativamente que las encontradas por el AGH. Esta diferencia se puede atribuir a que en el AGH empieza la evolución con una mayor cantidad de soluciones iniciales que el IABC y además se realiza un mayor número de iteraciones en el algoritmo con los mejores resultados.

Por otra parte, la comparación de los resultados con la metaheurística basada en el algoritmo competitivo imperialista propuesto por Pabón (2017), permite concluir que los dos algoritmos desarrollados para todas las instancias superan la media de las soluciones encontradas por el ICA.

Finalmente, el tiempo de ejecución de la metaheurística basada en el algoritmo competitivo imperialista para encontrar una respuesta, es menor que el de los algoritmos propuestos. Sin embargo, cabe recalcar que el tiempo computacional de un algoritmo, está sujeto a las características del ordenador en el que se realice la validación y la codificación del algoritmo en un lenguaje de programación.

14. Recomendaciones

Modificar la generación de imperios iniciales de la metaheurística basada en el algoritmo competitivo imperialista propuesto por Pabón (2017), aplicando el mismo método utilizado para la generación de las soluciones iniciales de los algoritmos propuestos en el presente proyecto, pues se podrían obtener soluciones de mejor calidad o iguales al AGH e IABC en tiempos computacionales menores.

Aplicar operadores de cruce y mutación que tengan en cuenta los tiempos de alistamiento y/o procesamiento de los trabajos, con el objetivo de que la solución generada después de aplicarlos, no tengan un cambio significativo, que puedan generar bloqueos que no permitan terminar el procesamiento de las tareas. Así mismo, disminuir la aleatoriedad de ciertos parámetros en los algoritmos, utilizando distribuciones estadísticas en la generación de los números.

Analizar la posibilidad de reestructurar la codificación del algoritmo genético híbrido para el problema del HFS con buffers de tamaño limitado, con el objetivo de disminuir el tiempo de ejecución sin afectar la calidad de las respuestas.

Considerar nuevas restricciones en el problema, como tiempos de alistamiento anticipatorios, capacidad y recursos limitados, tiempos de transferencia entre etapas de procesamiento, preferencia de tareas, procesamiento de trabajos por lotes o elegibilidad de las máquinas, con el fin de que el problema sea más cercano a la realidad.

Ejecutar los algoritmos con los mismos tiempos de alistamiento y procesamiento, según la cantidad de trabajos, máquinas y etapas, con el objetivo de ver el efecto de la capacidad del buffer en la variable respuesta, *makespan*.

Aplicar los algoritmos propuestos a una empresa con un sistema productivo similar al estudiado, ajustando parámetros o restricciones a las que haya lugar.

Referencias Bibliográficas

Abdollahpour, S., & Rezaeian , J. (2015). Minimizing makespan for flow shop scheduling problem with intermediate buffers by using hybrid approach of artificial immune system. *Applied Soft Computing*, *28*, 44-56.

Aguilar Justo, A. (2014). Un algoritmo basado en la colonia artificial de abejas con búsqueda local para resolver problemas de optimización con restricciones (Tesis de Maestría). *Universidad Veracruzana*.

Allahverdi, A. (2000). Minimizing mean flow time in a two-machine flowshop with sequence-independent setup times. *Computers and Operations Research, 27*, 111–127.

Almeder, C., & Hartl, R. (2013). A metaheuristic optimization approach for a real-world stochastic flexible flow shop problem with limited buffer. *Int. J. Production Economics, 145*, 88-95.

Arranz de la Peña, J., & Parra Truyol, A. (2007). *Algoritmos genéticos.* Madrid: Universidad Carlos III.

Błażewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Weglarz, J. (2007). *Handbook on scheduling: from theory to applications.* Springer Science & Business Media.

Bozorgirad, M., & Logendran, R. (2012). Sequence-dependent group scheduling problem on unrelated-parallel machines. *Expert Systems with Applications. 39*, 9021-9030.

Brito Santana, J., Campos Rodríguez, C., García López, F., García Torres, M., Melián Batista, B., Moreno Pérez, J., & Moreno Vega, J. (2004). *Metaheurísticas: una revisión actualizada.* San Cristobal de La Laguna, España: Universidad de La Laguna, Departamento de Estadística.

Camargo de Siqueira, E., de Souza, S., & Freitas Souza, M. (2013). Um algoritmo iterated greedy search aplicado à minimização do makespan no problema flowline híbrido e flexível. *Simpósio Brasileiro de Pesquisa Operacional*, (págs. 44-55).

Casado, S., & Martí, R. (2007). Principios de la búsqueda dispersa. *Recta*, 97-116.

Cervantes Posada, M. (2010). *Nuevos métodos meta heurísticos para la asignación eficiente, optimizada y robusta de recursos limitados.* Valencia: Universidad Politécnica de Valencia.

César Vélez, M., & Montoya, J. (2007). Metaheurísticos: Una alternativa para la solución de problemas combinatorios en administración de operaciones. *Revista EIA, 8*, 99-115.

Chen, C.-L., & Chen, C.-L. (2009). A bottleneck-based heuristic for minimizing makespan in a flexible flow line with unrelated parallel machines. *Computers & Operations Research, 36*, 3073-3081.

Collazo Pedraja , A. (s.f.). *Centro de Investigaciones Comerciales e Inicitivas Académicas.* Obtenido de Universidad de Puerto Rico: http://cicia.uprrp.edu/publicaciones/docentes/metodosimplexdePL.pdf

Companys Pascual, R., & Ribas Vila, I. (2012). El curioso comportamiento del método de inserción de la heurística NEH en el problema Fm|block|Cmax. *XVI Congreso de Ingeniería deOrganización*, (págs. 855-862). Vigo, España.

Costa, A., Cappadonna, F. A., & Fichera, S. (2014). A novel genetic algorithm for the hybrid flow shop scheduling with parallel batching and eligibility constraints. *Int J Adv Manuf Technol, 75*, 833 - 847.

Cruz Chávez, M., Moreno Bernal, P., & Peralta Abarca, J. (2014). Aplicación de la teoría de la complejidad en optimización combinatoria. *La génesis de la cultura universitaria en Morelos*, 35 - 42.

CSI/ITESM, D. (18 de Noviembre de 2008). *Tecnológico de Monterrey*. Obtenido de http://www.mty.itesm.mx/etie/deptos/m/ma00-130/lecturas/m130-19.pdf

Cui, Z., & Gu, X. (2015). An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems. *Neurocomputing, 148*, 248 - 259.

Delgado-Osuna, J., Lozano, M., & García Martínez, C. (2015). Algoritmo de Colonias de Abejas Artificiales para la composición de equipos médicos. *Actas de la XVI Conferencia CAEPIA*, (págs. 199-208). Albacete.

Díaz, A., Glover, F., Ghaziri, H., González, J., Laguna, M., Moscazo, P., & Tseng, F. (1996). *Optimización heurística y redes neuronales.* Madrid, España: Editorial Paraninfo.

Duarte Muñoz, A., Pantrigo Fernández, J., & Gallego Carrillo, M. (2007). *Metaheurísticas.* Madrid: Dykinson.

Fátima Morais, M., Godinho Filho, M., & Perassoli Boiko, T. J. (2013). Hybrid flow shop scheduling problems involving setup considerations: A literature review and analysis. *International Journal of Industrial Engineering, 20*, 614 - 630.

Fátima, M., Perassoli, T., Santos, L., Peterson, R., & Paraíso, P. (2014). Multicriteria hybrid flow shop scheduling problem: literature review, analysis, and future research. *Independent journal of management & production, 5(4)*, 1004 – 1031.

Figielska, E. (2008). A new heuristic for scheduling the two-stage flowshop with additional resources. *Computers & Industrial Engineering, 54*, 750-763.

Figielska, E. (2014). A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages. *European Journal of Operational Research*, *236*, 433-444.

Gallego Rendón, R., Escobar Zuluaga, A., & Toro Ocampo, E. (2015). *Técnicas heurísticas y metaheurísticas de optimización.* Pereira: Universidad Tecnológica de Pereira.

Glover, F., & Melián Batista, B. (2003). Búsqueda tabú. *Revista Iberoamericana de Inteligencia Artificial,19*, 29-48.

Goldberg, D. (1989). *Genetics algorithms in search, optimization and machine learning.* Boston: Addison Wesley Logman Publishing Co.

Hansen, P., & Mladenovic, N. (2003). Chapter 6: Variable neighborhood search. En F. Glover, & G. Kochenberger, *Handbook of metaheuristics* (págs. 145-184). New York, United States: Kluwer Academic Publishers.

Hansen, P., Mladenovic, N., & Moreno Pérez, J. (2003). Búsqueda de entorno variable. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, 7*, 77-92.

Herrero Giner, D. (8 de Mayo de 2014). *Decide Soluciones*. Obtenido de http://www.decidesoluciones.es/adaptacion-del-algoritmo-de-colonia-de-abejas-al-calculo-de-rutas/

Jun, S., & Park, J. (2015). A hybrid genetic algorithm for the hybrid flow shop scheduling problem with nighttime work and simultaneous work constraints: A case study from the transformer industry. *Expert Systems with Applications, 42*, 6196 - 6204.

Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., & Werner, F. (2007). Constructive and Simulated Annealing Algorithms for Hybrid Flow Shop Problems with Unrelated Parallel Machines. *Thammasat Int. J. Sc. Tech, 12(1)*, 31-41.

Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., & Werner, F. (2008). Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Int J Adv Manuf Technol 37*, 354-370.

Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., & Werner, F. (2009). A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Computers & Operations Research, 36*, 358-378.

Li, J.-q., & Pan, Q.-k. (2014). Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. *Information Sciences*.

Linn, R., & Zhang, W. (1999). Hybrid flow shop scheduling: A survey. *Computers & Industrial Engineering, 37*, 57 - 61.

López Vargas, J. (2013). *Metodología de programación de producción en un flow shop híbrido flexible con el uso de algoritmos genéticos para reducir el makespan. Aplicación en la industria textil (Tesis de maestría)* . Manizales: Universidad Nacional de Colombia.

Lopez, J., Giraldo, J., & Arango, J. (2015). Reducción del Tiempo de Terminación en la Programación de la Producción de una Línea de Flujo Híbrida Flexible (HFS). *Información Tecnológica, 26(3),* 157-172.

Low, C. (2005). Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Computers & Operations Research, 32,* 2013-2025.

Low, C., Hsu, C.-J., & Su, C.-T. (2008). A two-stage hybrid flowshop scheduling problem with a function constraint and unrelated alternative machines. *Computers & Operations Research, 35,* 845-853.

Maccarthy, B., & Liu, J. (1993). Adressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research, 31(1),* 59-79.

Maccarthy, B., & Liu, J. (1993). Adressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research, 31, 1,* 59-79.

Marichelvama, M., Prabaharanb, T., & Yang, X. (2014). Improved cuckoo search algorithm for hybrid flow shop scheduling. *Applied Soft Computing, 19,* 93-101.

Martí Cunquero, R. (2003). Algoritmos heurísticos en optimización combinatoria. *Revista Iberoamericana de Inteligencia Artificial,* 123-130.

Martí, R. (2003). Procedimientos metaheurísticos en optimización combinatoria. *Matemàtiques, 1(1),* 3-62.

Melián, B., & Pérez, J. (2003). Metaheurísticas: una visión global. *Revista Iberoamericana de Inteligencia Artificial, 19,* 7-28.

Miranda Lugo, P. L., & Florence Teixera Jr, R. (2012). Algoritmo genético com busca local para a programação da produção em sistemas flow-shop híbridos. *Simpósio Brasileiro de Pesquisa Operacional,* (págs. 19-29). Rio de Janeiro.

Miranda Lugo, P., Pérez Martínez, K., & Florence Teixera Jr, R. (2013). Um modelo de programação inteira mista para a programação da produção em flowshop híbrido com buffers limitados. *Simpósio Brasileiro de Pesquisa Operacional,* (págs. 154-165).

Mitchell, M. (1999). *An introduction to genetic algorithms* . Cambridge, Massachusetts : A Bradford Book The MIT Press.

Mladenovic, N., & Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research, 24 (11),* 1097-1100.

Moratto Chimenty, E., & Pérez Figueredo, L. (2016). Metaheurística basada en el algoritmo competitivo imperialista (ICA) aplicada a la solución del problema de flow shop híbrido (HFS) con máquinas paralelas no relacionadas (Tesis de pregrado). *Universidad Industrial de Santander*.

Moujahid, A., Inza, I., & Larrañaga, P. (s.f.). *Algoritmos genéticos.* Lejona, España: Universidad del País Vasco.

Muñoz, M., López, J., & Caicedo, E. (2008). Inteligencia de enjambres: sociedades para la solución de problemas (una revisión). *Revista Ingeniería e Investigación, 28(2)*, 119-130.

Osman, I., & Kelly, J. (1996). *Meta-Heuristics: Theory and Applications.* Boston, USA: Kluwer Academic.

Pabón Serrano, C. (2017). Solución al problema del flow shop híbrido (HFS) con máquinas paralelas no relacionadas y buffers de tamaño limitado mediante una metaheurística basada en el algoritmo competitivo imperialista (ICA) (Tesis de Pregrado). *Universidad Industrial de Santander*.

Parra Peña, J. (2006). Desarrollo de un modelo para la secuenciación de trabajos en la mediana industria ladrillera de la localidad XIX. Bogotá, Colombia: Universidad de los Andes.

Pinedo, M. (2008). Scheduling: theory, algorithms, and systems. *Springer Science + Business Media*.

Rabiee , M., Sadeghi Rad, R., Mazinani, M., & Shafaei, R. (2014). An intelligent hybrid meta-heuristic for solving a case of no-wait two-stage flexible flow shop scheduling problem with unrelated parallel machines. *Int J Adv Manuf Technol, 71*, 1229–1245.

Rahman, R., Santosa, B., & Wiratno, S. (2014). Hybrid differential evolution and bottleneck heuristic algorithm to solve bi-objective hybrid flow shop scheduling unrelated parallel machines problem. *Proceedings of the 2014 International Conference on Industrial Engineering and Operations Management*, 1339-1347.

Rashidi, E., Jahandar, M., & Zandich, M. (2010). An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *Int J Adv Manuf Technol, 49*, 1129-1139.

Reeves, C. (2003). Genetic algorithms. En F. Glover, & G. Kochenberger, *Handbook of metaheuristics* (págs. 55-83). New York: Kluwer Academic Publishers.

Ribas, I., Leisten, R., & Framiñan, J. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & operations research, 37*, 1439-1454.

Riojas Cañari, A. (2005). *Búsqueda Tabú: conceptos, algoritmo y aplicación al problema.* Lima, Perú: Universidad Nacional Mayor de San Marcos.

Rodríguez Quiñonez, T. (2014). Solución de problemas tipo Flow-Shop mediante algoritmos evolutivos (Tesis de maestria). *Universidad Nacional de Colombia, Bogotá D.C.*

Rosenfeld, R., & Irazábal, J. (2013). *Computabilidad, complejidad computacional y verificación de programas.* Exactas: La Plata, Buenos Aires, Argentina.

Ruiz, R., & Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research, 169* , 781–800.

Ruiz, R., & Vázquez-Rodríguez, J. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research, 205*, 1-18.

Sait, S., & Youssef, H. (1999). Iterative computer algorithms with applications in engineering: Solving combinatorial optimization problems. *Piscataway: IEEE*, 1-248.

Sanchis de Miguel , A., Ledezma Espino , A., Iglesias Martinez , J., García Jiménez, B., & Alonso Weber , J. (s.f.). *Open Course Web.* Obtenido de Universidad Carlos III de Madrid: http://ocw.uc3m.es/ingenieria-informatica/teoria-de-automatas-y-lenguajes-formales/material-de-clase-1/tema-8-complejidad-computacional/view

Sanhueza H., R., Hernisch V., I., Díaz R., H., & Guirrima C, R. (1999). Aplicación de algoritmos genéticos al problema de planificación de sistemas eléctricos de distribución. *Revista Facultad de Ingeniería, 6*, 55-63.

Scholz-Reiter, B., Rekersbrink, H., & Görges, M. (2010). Dynamic flexible flow shop problems—Scheduling heuristics vs. autonomous. *CIRP Annals - Manufacturing Technology, 59*, 465-468.

Seido Nagano , M., & João Vitor , M. (2002). Flowshops paralelos com processadores não-relacionados. *XXII Encontro Nacional de Engenharia de Produção*, (págs. 1-8). Curitiba.

Solis García, N., Ríos Mercado, R., & Alvarez Socarrás, A. (s.f.). Evaluación de un método de ramificación y acotamiento para un modelo de diseño territorial. 1 -14.

Tasgetiren, M., Pan, Q.-K., Suganthan , P., & Chen, A.-L. (2011). A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences, 181*, 3459 - 3475.

Uetake, T., Tsubone, H., & Ohba, M. (1995). A production scheduling system in a hybrid flow shop. *Int. J. Production Economic, 41*, 395-398.

Vázquez Espí, M. (1994). *Recocido simulado: un nuevo algoritmo para la optimación de estructuras.* Madrid, España: Escuela Técnica Superior de Arquitectura de Madrid.

Yagmahan, B., & Yenisey, M. (2009). Scheduling Practice and Recent Developments in Flow Shop and Job Shop Scheduling. *Computational intelligence in flow shop and job shop scheduling, 230*, 261-300.

Yaurima, V., Burtseva, L., & Tchernykh, A. (2009). Hybrid flowshop with unrelated machines, sequence-dependet setup time, availability constraints and limited buffers. *Computers & Industrial Engineering, 56*, 1452-1463.

Yu, X., & Mitsuo, G. (2010). *Introduction to evolutionary algorithms.* Springer Science & Business Media.

Zandieh, M., Mozaffari, E., & Gholami, M. (2010). A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems. *J Intell Manuf, 21*, 731-743.