Proyecto Fin de Carrera
Ingeniería de Organización Industrial

Algoritmo ABC para la resolución de problema en un entorno de flow shop con restricción no-idle

Autor: Carlos Javier Viloca Pérez
Tutor: Víctor Fernández-Viagas Escudero

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Carrera
Grado de Ingeniería de Organización Industrial

Algoritmo ABC para la resolución de problema en un entorno de flow shop con restricción no-idle

Autor:
Carlos Javier Viloca Pérez

Tutor:
Víctor Fernández-Viagas Escudero
Profesor Ayudante Doctor

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Algoritmo ABC para la resolución de problema en un entorno de flow shop con restricción no-idle

Autor: Carlos Javier Viloca Pérez

Tutor: Víctor Fernández-Viagas Escudero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

La realización del Trabajo de Fin de Grado es la culminación de los estudios universitarios. Aprovecho este espacio para agradecer a todos los profesores que me han enseñado durante estos años, y especialmente a mi tutor, Víctor Fernández-Viagas Escudero, por la atención e interés que me ha dispensado a lo largo de la realización de este proyecto.

También quiero agradecer a mi familia su apoyo y ayuda en esta etapa de mi vida.

Carlos Javier Viloca Pérez

Sevilla, 2019

Resumen

En este trabajo se utiliza el algoritmo de optimización Artificial Bee Colony (ABC) inspirado en la inteligencia de enjambre, perteneciente a una rama de la Inteligencia artificial que se basa en el comportamiento colectivo de sistemas descentralizados y auto organizados. Este algoritmo reproduce el comportamiento inteligente de las colonias de abejas a la hora de buscar fuentes de alimentos.

En nuestro caso lo aplicamos a la resolución de un problema de programación de la producción. Entre sus muchas variantes, nos hemos centrado en una de las restricciones menos atendidas por la literatura. Concretamente, un problema en un entorno del flow shop, con el fin de minimizar el tiempo de finalización del último trabajo, o makespan, bajo la restricción no idle, esto es, la inactividad en máquinas no está permitida una vez comienza su actividad.

Para la resolución de este problema se proponen una serie de variantes del algoritmo, que son presentadas a lo largo de este documento. Posteriormente, se realiza una evaluación computacional de los resultados obtenidos en las simulaciones con el algoritmo inicial y las variantes aportadas.

El algoritmo inicial y las nuevas variantes que hemos introducido se han codificado en lenguaje C, mediante el programa informático Code::Blocks, con la ayuda de la librería <schedule.h>, que posee una serie de funciones que facilita la implementación.

Abstract

In this work we use the Artificial Bee Colony (ABC) optimization algorithm, an algorithm inspired by swarm intelligence, a branch of artificial intelligence which is based on the collective behavior of decentralized and self-organized systems. This algorithm is motivated by the intelligent behavior of honey bees when looking for food sources.

We apply it to the resolution of a scheduling problem. Among its many variants, we have focused on one problem that has not attracted much attention in the literature. Specifically, a problem in a flow shop environment in order to minimize the makespan, considering the constraint of no idle times allowed on machines.

In order to solve this problem, several variants of the algorithm are proposed, which are presented throughout this document. Subsequently, computational results obtained in the simulations with the initial algorithm and the variants proposed are carried out.

The initial algorithms and the variants are coded in C with the computer program Code::Blocks, with the aid of the functions included on the <schedule.h> library.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICAS

ÍNDICE DE FIGURAS

Notación

ABC	Artificial Bee Colony
m	Número de máquinas
n	Número de trabajos
Algoritmo	Conjunto de pasos estructurados que permiten realizar una actividad
Makespan	Tiempo de terminación del último trabajo
Flow Shop	Tipo de entorno. Taller de flujo regular
no-idle	Espera en máquinas no permitida.
α	Características de las máquinas
β	Características de los trabajos
γ	Función objetivo
pij	Tiempos de proceso
RPI	Incremento porcentual relativo

1 INTRODUCCIÓN

1.1 Justificación del tema elegido

La programación de la producción es un proceso de toma de decisiones que existe en la mayoría de los sistemas de fabricación y producción, así como en la mayoría de los entornos de proceso de información. De igual modo, también se aplica en los entornos de transporte y distribución y en otros tipos de industrias de servicios. Con la programación se busca la asignación de recursos limitados a las tareas en el tiempo. Es un proceso de toma de decisiones que tiene como objetivo la optimización de uno o más objetivos.

Los recursos y tareas pueden tomar muchas formas, por ejemplo, los recursos pueden estar en máquinas en un taller, pistas en un aeropuerto, cuadrillas en un lugar de construcción, unidades de procesamiento en un entorno informático, etc. y las tareas pueden ser operaciones en un proceso de producción, despegues y aterrizajes en un aeropuerto, etapas en un proyecto de construcción, ejecución de programas informáticos, etc. Cada tarea puede tener un nivel de prioridad diferente, según la hora de inicio más temprana posible o la fecha de vencimiento, entre otros.

Los objetivos también pueden tomar muchas formas. Un posible objetivo es la minimización del tiempo de finalización de la última tarea, y otro posible objetivo es la minimización del número de tareas completadas después de las fechas de vencimiento comprometidas.

Del mismo modo, se pueden considerar diversas restricciones, como es el no permitir la inactividad en las máquinas al procesar los distintos trabajos. Esta restricción modela una importante situación práctica cuando se usa maquinaria muy costosa o limitaciones técnicas impiden que esta situación deba producirse. Además, a pesar de su gran relevancia, es un problema que no ha sido tratado especialmente en la literatura, como señala Tasgetiren (2013). Es por ello por lo que abordamos este aspecto en este trabajo.

1.2 Objetivo

El objetivo concreto de este trabajo es la resolución de un problema de programación en un entorno del flow shop, con el fin de minimizar el tiempo de finalización del último trabajo, o makespan, bajo la restricción no idle, esto es, la inactividad en máquinas no está permitida una vez inicia a procesar el primer trabajo.

1.3 Estructura del trabajo

El trabajo está estructurado en 7 capítulos que son los siguientes:

- Un primer capítulo, el presente, que intenta dar una primera imagen del trabajo desarrollado, y donde la importancia del tema y la definición del objetivo quedan concretados.
- El segundo capítulo entra ya en la descripción del problema a resolver, situando y definiendo las características del problema considerado.
- El tercer capítulo explica cómo funciona el algoritmo Artificial Bee Colony (ABC) de manera genérica.
- En el cuarto capítulo se detallan cada una de las fases del algortimo aplicado a la resolución de nuestro

problema.

- Un quinto capítulo, donde se explican las variantes realizadas al algoritmo inicial, que constituyen la aportación original al trabajo.
- El siguiente capítulo analiza la eficacia y resultados obtenidos entre las diferentes variantes del algoritmo, compárandolas entre sí.
- Por último, se presenta un capítulo resumiendo las conclusiones extraídas durante la realización del trabajo y su repercusión atendiendo a los resultados obtenidos.

2 DESCRIPCIÓN DEL PROBLEMA

En este capítulo se da a conocer el problema, describiendo las características principales que lo definen, como son su entorno, las restricciones consideradas y el objetivo a evaluar.

En primer lugar, tenemos que tener en cuenta algunas suposiciones que tendrán efecto a lo largo de todo el documento, y que afectan a la hora de resolver el problema y los resultados del mismo. Son las siguientes, como se detalla en Perez (2017):

- Los trabajos están disponibles al principio del horizonte de programación.
- Los trabajos no se pueden interrumpir.
- Las máquinas están siempre disponibles.
- Cada máquina puede hacer un trabajo, y un trabajo puede ser realizado sólo en una máquina.
- El buffer entre máquinas se supone infinito.
- El tiempo de transporte es despreciable.

2.1 Contexto del problema

El objetivo de este apartado es situar el problema a resolver. Se procederá a ir descendiendo de lo más general a lo particular. Como podemos ver en la Figura 2.1 este proceso comienza con la Planificación, prosigue con su vertiente operativa, la programación, y continúa con la definición del tipo de problema que se va a abordar, considerando entorno, restricciones y objetivo.

Figura 2.1 - Contexto del problema [Fuente: elaboración propia]

En los siguientes epígrafes desarrollaremos brevemente cada apartado. Dentro de los entornos, restricciones y objetivos posibles, iremos concretando las elecciones de este trabajo.

2.2 La Planificación y Control de la Producción

Según Vollmann (2005), la tarea fundamental de los sistemas de Planificación y Control de la Producción consiste en "administrar eficientemente el flujo de materiales, la utilización del personal y del equipo y responder a los requerimientos de los clientes utilizando la capacidad de los proveedores, de las instalaciones internas y (en algunos casos) de los propios clientes para cumplir la demanda del cliente". Dan soporte a la organización en tres niveles:

- **A nivel estratégico** (largo plazo): ofrecen la información para determinar la capacidad que va a ser requerida para satisfacer la demanda de los clientes en el futuro, las características de esa capacidad y los proveedores a los que se va a acudir. Es una etapa que requiere fuertes inversiones y se desarrolla en varios años.
- **A nivel táctico** (medio plazo): hacer un plan agregado de producción, en donde se utiliza como información la venta estimada de una familia o grupo de productos (en lugar de pormenorizar por cada producto individualmente). De esta forma se puede planificar la capacidad agregada necesaria para producir esta cantidad.
- **A nivel operativo** (corto plazo): la planificación incorpora un nivel de detalle mucho mayor para cumplir con los requerimientos de producción. Esta planificación involucra tiempo, personal, material, equipo e instalaciones. En este nivel es donde la programación de actividades o scheduling tiene lugar.

Este Trabajo de Fin de Grado se enmarca en este último nivel, donde se suelen utilizar métodos cuantitativos para estos plazos, dada la alta repetitividad de escenarios y decisiones, como también por la gran cantidad de información que se requiere procesar y analizar (tiempos de procesamiento, fechas de entrega, etc).

2.3 Programación (Scheduling)

La programación se encarga de la asignación de recursos a las tareas que deben realizarse durante un horizonte de tiempo para concretar un plan de producción, y los horizontes temporales suelen ser menores a una semana.

Así, el scheduling es la asignación de recursos a las tareas que definen el plan de producción, y su complejidad e importancia crecerá en la medida en que el uso de los recursos entre en conflicto. Framiñan y otros (2014) señalan las siguientes condiciones comunes del scheduling para todos los sistemas de producción:

- Son decisiones complejas. Y su complejidad seguirá aumentando debido a la tendencia actual de incrementar la flexibilidad en los pedidos de los clientes, lo que deriva en aumentar la complejidad del proceso productivo.
- Tiene horizonte temporal corto, el ciclo de vida de un ordenamiento o schedule suele ser muy corto.
- A pesar de ser decisiones de horizontes cortos, suelen ser muy importantes para la línea de producción de una organización ya que impactan directamente sobre los costos de producción y los tiempos de entrega, lo que afecta a la competitividad de la empresa (tiempo de entrega y costos).
- Por estar situados en el centro de las operaciones de producción, las restricciones y objetivos del scheduling son muy dependientes del tipo de empresa.
- Suelen ser decisiones estructuradas, lo que permite que la información, los criterios y las limitaciones sean fáciles de formalizar.

2.4 Entorno

Los problemas de programación se denominan genéricamente como α | β | γ, siendo α el entorno a considerar. Existen distintos tipos de entornos:

- Single Machine (α=1). Indica que es un entorno con una sola máquina.

- Parallel Machine. Está formado por una serie de máquinas de forma paralela por las cuáles se puede procesar cualquier trabajo. A su vez se pueden dividir en:
  - Identical Parallel machines (α=Pm). Los tiempos de proceso no dependen de la máquina.
  - Uniform Parallel machines (α=Qm). Cada máquina posee diferentes velocidades.
  - Unrelated Parallel machines (α=Rm). El caso más general, con máquinas diferentes.

- Los entornos tipo taller, con máquinas dispuestas en serie, y que se diferencian en función de la ruta de sus trabajos:
  - Flow Shop (α=Fm). Todos los trabajos tienen la misma ruta.
  - Job Shop (α=Jm). Los trabajos siguen una ruta predeterminada por las máquinas.
  - Open Shop (α=Om). La ruta de los trabajos no está definida. Es el más complejo de todos.

- Tambien existen entornor híbridos (α=Hm), formados por la combinación de varios entornos.

De ellas, las configuraciones más comúnmente utilizadas son job shop: cada trabajo a ser procesado posee una ruta particular de operaciones a realizarse en un conjunto de máquinas, y flow shop y sus distintas variantes: todos los trabajos siguen la misma secuencia de máquinas.

Nosotros nos centraremos en esta última.

2.4.1 Flow Shop

En muchas instalaciones de fabricación y ensamblaje se deben realizar varias operaciones en cada trabajo. A menudo, estas operaciones deben realizarse en el mismo orden, lo que implica que los trabajos deben seguir la misma ruta. Se asume que las máquinas se configuran en serie y el entorno se conoce como Flow shop, que es el caso concreto de este proyecto.

La configuración productiva del tipo flow shop consiste en un grupo de máquinas que realizan sus tareas de forma secuencial, como podemos ver en la Figura 2.2. Primero debe realizarse la primera tarea en la primera máquina, luego en la segunda máquina, y así sucesivamente.

Figura 2.2 - Representación de máquinas en un entorno de flow shop.
[Fuente: Imagen obtenida de la web]

Esta figura muestra de forma esquemática como es una configuración flow shop. Cuatro centros de trabajo representados por cuatro máquinas. Las flechas indican el sentido que recorren los trabajos que son procesados en dicho sistema. Todos los trabajos pasan por todas las máquinas y en la misma secuencia.

En un sentido más formal, un sistema flow shop consiste en un conjunto $J = \{J_1, \ldots, J_n\}$ de n trabajos y un conjunto $M = \{M_i, \ldots, M_m\}$ de m máquinas. Cada trabajo consiste de un conjunto $O_j = \{O_{j,1}, \ldots, O_{j,m}\}$ de m operaciones, donde la i - ésima operación se ejecuta en la máquina M_i. Además, la operación $O_{j,i+1}$ puede comenzar si y solo si la operación $O_{j,i}$ fue completada. Cada operación $O_{j,i}$ tiene un tiempo de procesamiento $p_{i,j} \in \mathbb{N}$. Cada máquina M_i puede procesar sólo una operación a la vez.

Un ordenamiento de los trabajos σ se define como la secuencia en que cada máquina procesará los trabajos, y en consecuencia las operaciones que deberá realizar. Se pueden distinguir dos tipos:

- Si este ordenamiento de los trabajos en cada máquina es el mismo, el problema de scheduling se denomina problema permutativo de flow shop (PFS).
- En otro caso se le llama problema no-permutativo de flow shop (NPFS).

En ambos casos el objetivo es encontrar un ordenamiento σ de los n trabajos para las m máquinas para minimizar alguna función objetivo. Los sistemas flow shop suelen ser utilizados para producciones del tipo estandarizada, en donde los productos y sus variaciones están especificados y definidos de ante mano. Esto permite generar sistemas de producción de propósitos específicos que permiten tener alto rendimiento. Algunos casos de productos producidos en sistemas flow shop son el ensamblaje de los electrodomésticos, el envasado de vinos y bebidas, y la fabricación de muebles, entre otros.

El grado de especialización que suelen tener los sistemas flow shop no implica que resolver su programación de actividades sea sencilla. Estos problemas suelen ser para un gran espectro de variantes del tipo de los problemas NP-hard. Sólo en algunos casos particulares existen algoritmos capaces de resolver los problemas en tiempos polinomiales, como es el caso de la regla de Johnson, para los casos de 2 y 3 máquinas con función objetivo de tiempo mínimo de finalización. Esto ha llevado a que se concentren muchos esfuerzos de la comunidad científica en el estudio de los métodos de resolución.

Un aspecto para considerar en los sistemas flow shop es cómo se organizan las secuencias entre las sucesivas etapas o máquinas del sistema productivo. Tradicionalmente, se ha abordado esta organización a través de una simplificación: la misma secuencia para todas las máquinas. Es decir, hay una única secuencia que se mantiene en cada máquina. Sin embargo, esto no es estrictamente necesario en la gran mayoría de las aplicaciones reales de flow shop, y, por consiguiente, tampoco por la definición formal del flow shop. Pero cuando se flexibiliza esta simplificación se da lugar a que el ordenamiento en las etapas puede ser modificado lo que implica que obtener la solución óptima del problema sea considerablemente más difícil.

2.5 Restricciones

Los tipos de restricciones se denominan por la letra griega β y las más comunes son las siguientes:

- Secuencia de permutación (β=prmu). Es la restricción del problema.
- Fechas de entrega y fechas de llegadas (β=r_j β=d_j). Si un problema tiene restricción de fechas de entregas (r_j), los trabajos deben de terminar antes de su r_j a ser posible para cumplir este objetivo. En cambio, si su restricción son las fechas de llegadas (d_j), las operaciones o trabajos deberán ser procesados por las máquinas a partir de su fecha de llegada.
- Precedencia de trabajos (β=prec). Los trabajos seguirán un orden predeterminado al ser procesados por las máquinas.

- Tiempos de cambio (setup times) (β=s_{ij}). Son tiempos que deben hacerse antes del procesamiento de un trabajo. Hay dos tipos de tiempos de setup: anticipatorios y no anticipatorios.

- Máquina no ociosa (β=no-idle). No están permitidos los tiempos ociosos de las máquinas entre trabajos. Una vez que la máquina empieza a procesar el primer trabajo de la secuencia, esta máquina no para hasta terminar el último.

- Lotes (β=batch). Se utiliza para producir ciertos tipos de productos en lotes.

- Interrupciones (β=pmtn). Se emplea cuando en el procesamiento de los trabajos se producen interrupciones.

- Espera no permitida (no-wait) (β=nwt). Los trabajos no pueden esperar a ser procesados entre máquinas. Una vez que se comienza a procesar un trabajo, éste debe continuar por las máquinas hasta ser procesado por completo.

- Almacenaje (β=buffer). Se utiliza para talleres de trabajo dónde es conveniente el uso de buffer.

La restricción a considerar en nuestro problema es no-idle, que pasamos a desarrollar a continuación.

2.5.1 Restricción no-idle

Dentro del entorno de Flow shop, se considera la restricción no-idle. Surge una situación interesante cuando no se permite un tiempo de inactividad en las máquinas entre los distintos trabajos. Esta restricción modela una situación práctica importante que surge cuando se emplea maquinaria costosa. El ralentí en equipos costosos a menudo no se desea o incluso no es posible debido a limitaciones tecnológicas. Algunos ejemplos son los steppers utilizados en la producción de circuitos integrados mediante fotolitografía. Otros ejemplos provienen de sectores en los que se utiliza maquinaria menos costosa, pero donde las máquinas no se pueden detener y reiniciar fácilmente o es completamente antieconómico hacerlo. Los hornos de rodillos cerámicos, por ejemplo, consumen una gran cantidad de gas natural cuando están en funcionamiento. El ralentí no es una opción porque lleva varios días detenerse y reiniciar el horno debido a una inercia térmica muy grande. En todos estos casos, debe evitarse el ralentí.

En términos generales, la secuencia de procesamiento de los trabajos en los que hay una restricción de procesamiento sin inactividad en las máquinas debe garantizar que una vez que la máquina comienza a procesar no se detiene hasta que finaliza el último trabajo de la secuencia. Por ello, el inicio de actividad de cada máquina debe retrasarse para cumplir este requerimiento, como se muestra en la Figura 2.3.

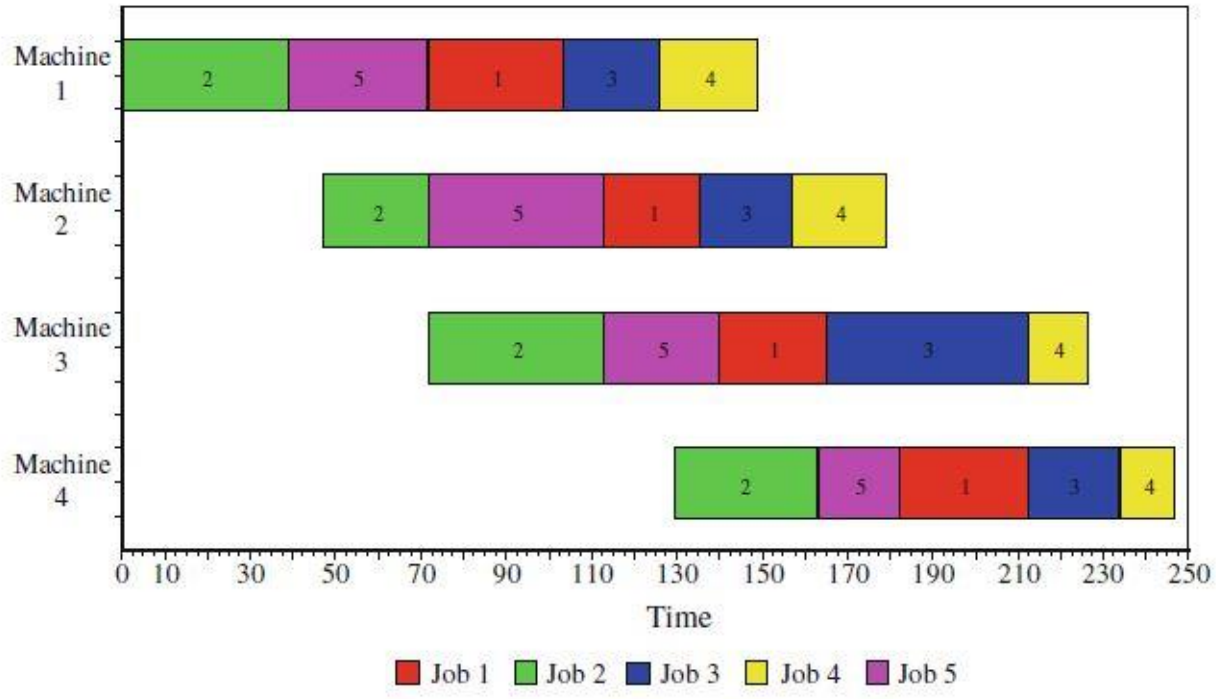


Figura 2.3 - Gráfico Gantt de Flow shop con restricción no-idle.
[Fuente: Manufacturing Scheduling Systems, 2014]

2.6 Objetivos

Los tipos de objetivos se designan por la letra griega γ y existen diversos tipos como son:

- Tiempo de terminación del trabajo (*completion time)* γ=min (max C_j). Minimiza la terminación de los trabajos.

- Tiempo de flujo del trabajo *(flowtime)* γ=min (max F_j). Indica el tiempo en el que el trabajo está en el entorno. F_j puede definirse como: F_j=C_j-r_j siendo r_j las fechas de llegadas de los productos.

- Retraso del trabajo (*lateness)* γ=min (max L_j). L_j puede definirse como: L_j=C_j-d_j siendo d_j las fechas de entrega de los productos, y lo que se quiere en éste es que el producto tenga el mínimo retraso posible.

- Tardanza del trabajo (*tardiness)* γ= min (max T_j). Con este objetivo se intenta cumplir que la tardanza del trabajo en ser entregado sea el mínimo posible. T_j se define como: T_j=max {0, L_j} =max {0, C_j-d_j}.

- Adelanto del trabajo (*earliness)* γ=min (max E_j). El adelanto de los trabajos se define como: E_j= max {0, -L_j} =max {0,- (C_j- d_j}.d_j

- Trabajo tardío (tardy job) γ=min (max U_j). La definición de trabajo tardío es la siguiente: U_j si T_j>0, es decir, si C_j>d_j(si la terminación del trabajo es mayor a su fecha de entrega), U_j =0 en otro caso.

En nuestro caso el objetivo es la minimización del makespan (γ=$C_{\max}$), que es equivalente al tiempo de finalización del último trabajo en abandonar el sistema. Éste se suele utilizar cuando se tienen fechas de

entregas que se deben de cumplir, o cuando es necesario acabar lo antes posible debido a que se van a producir otra serie de productos después o se quiere ahorrar en costes.

2.7 Concreción del problema en este trabajo

En definitiva, consideraremos un problema en un entorno de flowshop, con el objetivo de minimizar el makespan y considerando la restricción de no permitir inactividad en las máquinas entre los trabajos a realizar por las mismas.

Para su resolución, utilizaremos el algoritmo ABC (Artificial Bee Colony) con algunas variantes. La metodología del algoritmo se explica detalladamente en el siguiente capítulo.

3 ALGORITMO ARTIFICIAL BEE COLONY (ABC)

Para resolver el problema descrito se ha elegido el algoritmo "Artificial Bee Colony" (ABC), que ha demostrado tener un buen rendimiento en comparación con otros métodos de resolución para este tipo de problemas en un entorno de Flow shop. Estos resultados se muestran en el capítulo de análisis computacional.

El Artficial Bee Colony es un algoritmo de optimización basado en la inteligencia de enjambre, propuesto por Karaboga (2005) para resolver problemas de optimización modal evolutiva. Una de las ventajas que tiene este algoritmo respecto a otros es el empleo de pocos parámetros de control.

Está inspirado en el comportamiento inteligente que siguen las abejas a la hora de buscar alimento. Las abejas son divididas en 3 grupos:

- Abeja empleada o *employed:* Es aquella que está llevando a cabo la explotación de una fuente de alimento.
- Abeja espectadora o *onlooker:* Son las abejas que esperan en la colmena la información dada por las abejas empleadas, para así decidir cuál de las actuales fuentes de alimento deben ser explotadas.
- Abeja exploradora o *scout:* Son aquellas que realizan una búsqueda aleatoria con el fin de encontrar nuevas fuentes de alimento.

En el algoritmo cada una de las soluciones posibles al problema considerado son llamadas fuentes de alimento o *food sources* y son representadas por un vector de dimensión n, mientras que la calidad de cada solución se corresponde con la cantidad de néctar presente en dicha fuente de alimento.

Al igual que otras técnicas basadas en la inteligencia de enjambre, es un proceso iterativo. El algoritmo inicia con una población de fuentes de alimento generada aleatoriamente, y a continuación se siguen los siguientes pasos hasta que se alcanza un criterio de terminación:

1. Se envían las abejas empleadas a las fuentes de alimento y se miden las cantidades de néctar presentes en las mismas.
2. Las abejas espectadoras seleccionan las fuentes de alimento tras compartir la información con las abejas empleadas.
3. Determinar las abejas exploradoras y enviarlas hacia posibles nuevas fuentes de alimento.

Las principales fases del algoritmo original se explican a continuación.

3.1 Inicialización de los parámetros

Los parámetros del algoritmo ABC original son el número de fuentes de alimento (*SN*), que es igual al número de abejas empleadas o espectadoras, el número de intentos después del cual una fuente de alimento debe ser abandonada (*limit*), y el criterio de terminación.

3.2 Inicialización de la población

La población incial de soluciones se forma con un número SN de vectores de números reales de dimensión n generados aleatoriamente (las fuentes de alimento). Siendo $X_i = \{x_{i1,} x_{i2}, \dots, x_{in}\}$ cada una de las fuentes de

alimento, y cada uno de los valores generados con la siguiente fórmula:

$$x_{ij} = LBj + (UBj - LBj) \cdot r \text{ para j = 1, 2, … , n e i = 1, 2, … , SN} \quad (1)$$

Donde r es un número entre 0 y 1 generado aleatoriamente, y LBj y UBj son los límites inferior y superior (*lower bound* y *upper bound*) para la dimensión j respectivamente.

3.3 Inicialización de la fase de abeja empleada

En esta fase, cada una de las abejas empleadas x_i genera una nueva fuente de alimento x_{new} en la vecindad de su posición actual de la siguiente manera:

$$x_{new(j)} = x_{ij} + (x_{ij} - x_{kj}) \cdot r \quad (2)$$

Donde k ϵ {1,2,…,SN) ∧ k ≠ i y j ϵ {1,2, …, n} son índices elegidos aleatoriamente. r es un número real distribuido aleatoriamente en el intervalo [-1,1].

Una vez que se ha obtenido x_{new}, se evalúa y compara con x_i. Si el valor de x_{new} es igual o mejor que el de x_i, x_{new} remplazará a x_i y se convertirá en un nuevo miembro de la población, en otro caso se mantiene x_i.

3.4 Fase de abeja espectadora

Las abejas espectadoras evaúan la cantidad de nectar tomada por las abejas empleadas y seleccionan una fuente de alimento x_i, dependiendo de su valor de probabilidad p_i calculado con la siguiente formula:

$$p_i = \frac{f_i}{\sum_{i=1}^{SN} f_i} \quad (3)$$

Siendo f_i la cantidad de néctar encontrada en la fuente de alimento x_i. Se puede observar que a mayor valor de f_i, mayor es la probabilidad de que dicha fuente sea seleccionada.

Una vez que la abeja espectadora ha seleccionado una fuente de alimento x_i, produce una modificación en x_i usando la Ecuacion (2). Como en el caso de la abeja empleada, si la nueva fuente de alimento tiene una cantidad de néctar mayor o igual que x_i, la fuente de alimento modificada reemplazará a x_i y pasará a ser un nuevo miembro de la población.

3.5 Fase de abeja exploradora

En caso de que una fuente de alimento x_i se considere que no puede ser mejorada tras un número de intentos *limit*, dicha fuente de alimento es abandonada. La abeja empleada correspondiente a dicha fuente pasa a ser entonces una exploradora. Las abejas exploradoras producen aleatoriamente una nueva fuente de alimento con la siguiente fórmula:

$$x_{ij} = LBj + (UBj - LBj) \cdot r \text{ para j = 1, 2, … ,n.} \quad (5)$$

Donde *r* es un numero distribuido uniformemente en el intervalo [0,1].

En el algoritmo ABC original debe existir al menos una abeja exploradora que genere una nueva fuente de alimento en cada ciclo del proceso.

3.6 Principales pasos del Algoritmo ABC

Siguiendo la explicación dada de cada una de las fases, los principales pasos a seguir del algoritmo pueden ser resumidos como sigue:

- Paso 1: Inicialización.
- Paso 2: Se asignan las abejas empleadas a las fuentes de alimento.
- Paso 3: Asignar las abejas espectadoras en las fuentes de alimentos dependiendo de sus cantidades de néctar.
- Paso 4: Enviar a las exploradoras al área de búsqueda para descubrir nuevas fuentes de alimento.
- Paso 5: Memorizar la mejor fuente de alimento encontrada hasta ese momento.
- Paso 6: Si no se satisface el criterio de terminación, ir al paso 2; de lo contrario, se detiene el proceso y se selecciona la mejor fuente de alimento encontrada.

En el siguiente apartado se explica cómo adaptar y aplicar este algoritmo de optimización en la resolución de un problema de programación de la producción.

4 ALGORITMO ABC APLICADO A NUESTRO PROBLEMA

En este apartado se pretende adaptar y utilizar el algoritmo ABC en la resolución del problema planteado en este TFG, un problema en un entorno de tipo flow shop, considerando la restricción no-idle.

Este algoritmo ha demostrado tener buenos resultados en la resolución de problemas de optimización, en comparación con otros algoritmos similares, con la ventaja de emplear menos parámetros de control.

El algoritmo persigue, siguiendo la misma metodología antes descrita, encontrar una solución óptima al problema. En este apartado se explica el algoritmo ABC propuesto por Pan y otros (2009) para resolver el problema de estudio; así, se explican detalladamente los pasos a seguir por el algoritmo (ver Figura 4.1), así como los cambios necesarios para adaptarlo a nuestro problema, ya que fue originalmente diseñado para la optimización de funciones continuas.

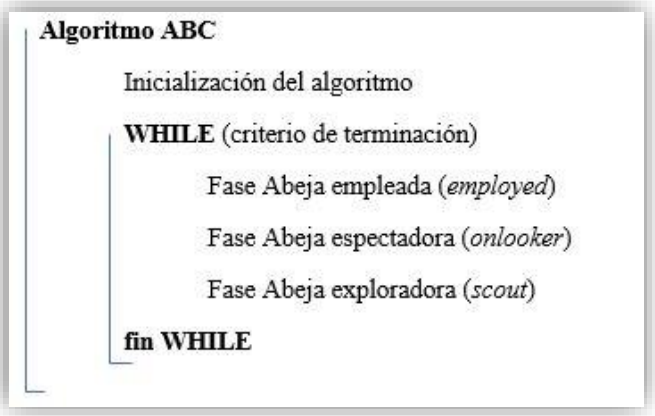


Figura 4.1 - Pseudocódigo del Algoritmo ABC.
[Fuente: Elaboración propia]

4.1 Representación de las soluciones

Para adaptar el algoritmo a nuestro caso como se ha dicho anteriormente, las fuentes de alimento serán representadas mediante secuencias de trabajos, una única secuencia por cada solución ya que se trata de un problema permutativo de flow shop y esto implica que la secuencia a seguir es la misma para todas las máquinas.

Al igual que en el algoritmo original, se consideran un número SN de secuencias iniciales, siendo este número igual al de abejas empleadas y espectadoras.

4.2 Inicialización de la población

Para garantizar una cierta calidad y diversidad de la población, las secuencias iniciales se generan como se explica a continuación:

- Una de las secuencias se ha generado asignando la regla SPT (Shortest Processing Time) a la primera

máquina, es decir, ordenando los tiempos de proceso de los trabajos de la secuencia en orden ascendente.

- Otra de las secuencias se ha generado asignando la regla LPT (Longest Processing Time) a la últma máquina, esto es, los trabajos de la secuencia en orden descendente de tiempo de proceso.
- El resto de las secuencias se generado aleatoriamente, con el fin de asegurar la diversidad de la población como se ha comentado anteriormente.

4.3 Fase de abeja empleada (employed)

Al igual que en el algoritmo ABC original, las abejas empledas generan nuevas soluciones en la vecindad de sus posiciones actuales. En nuestro trabajo las variaciones realizadas sobre las secuencias para obtener nuevas fuentes de alimento son los cambios tipo insert y swap. El cambio tipo insert consiste en extraer un trabajo de la secuencia e insertarlo en otra posición, mientras que el cambio tipo swap consiste en intercambiar la posición de dos trabajos de la secuencia, como podemos ver en la Figura 4.3.

Para nuestro caso se han considerado 4 cambios diferentes:

- Un cambio insert
- Dos cambios insert
- Un cambio swap
- Dos cambios swap

A partir de éstos se generará aleatoriamente una lista *NL*, de donde se irán tomando los cambios necesarios para generar nuevas secuencias. Cada cambio puede tener unos resultados diferentes a lo largo del proceso de evolución del algoritmo, por lo que se propone un mecanismo que se va adaptando en el tiempo y aprendiendo de sí mismo. Este mecanismo se denomina estrategia autoadaptativa y se explica en el epígrafe 4.6.

El pseudocódigo del procedimiento de esta primera fase del algoritmo se muestra en la Figura 4.2.

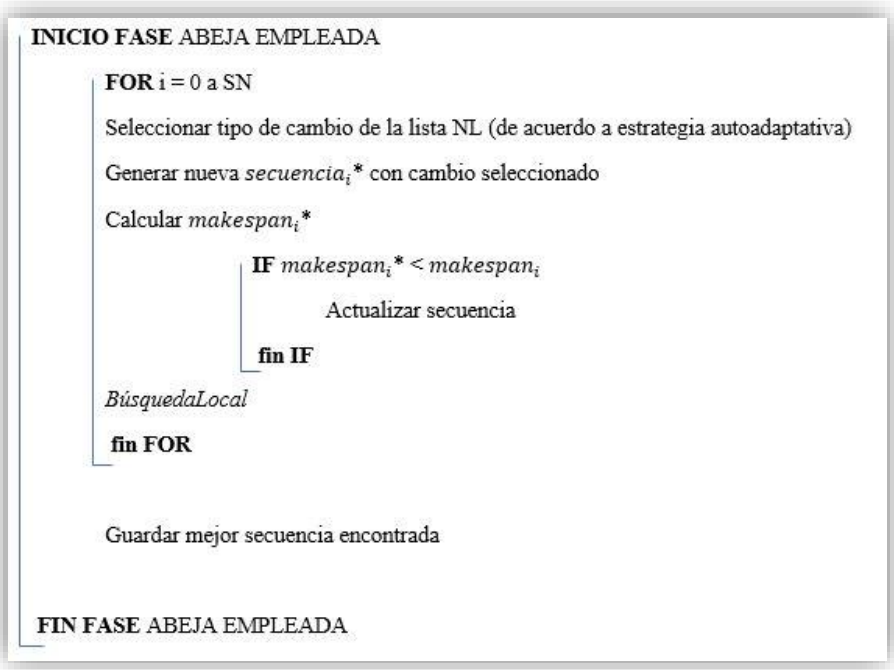


Figura 4.2 - Pseudocódigo de la fase abeja empleada.
[Fuente: Elaboración propia]

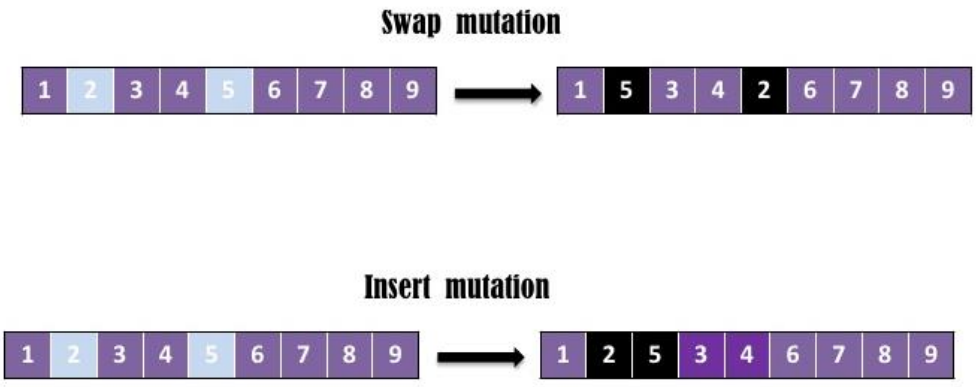


Figura 4.3 - Cambios *insert* y *swap* aplicados a las secuencias.
[Fuente: Imagen obtenida de la web]

4.3.1 Búsqueda local

En esta fase también se realiza, con una probabilidad p_L una búsqueda local para cada secuencia. Esta búsqueda local consiste en realizar pequeñas variaciones sobre una secuencia para encontrar el resultado óptimo en su vecindad. El número de repeticiones de la búsqueda local depende de número de trabajos de la secuencia, definiéndose como $l_{max} = n^2$.

El tipo de cambio utilizado en la búsqueda será el mismo a lo largo de todo el proceso, y para evitar caer en una búsqueda cíclica o un óptimo local, se realizará un cambio insert si esa secuencia se ha obtenido mediante un cambio tipo swap y viceversa, o aleatoriamente en otro caso. El pseudocódigo del proceso de la búsqueda local se muestra en la Figura 4.4.

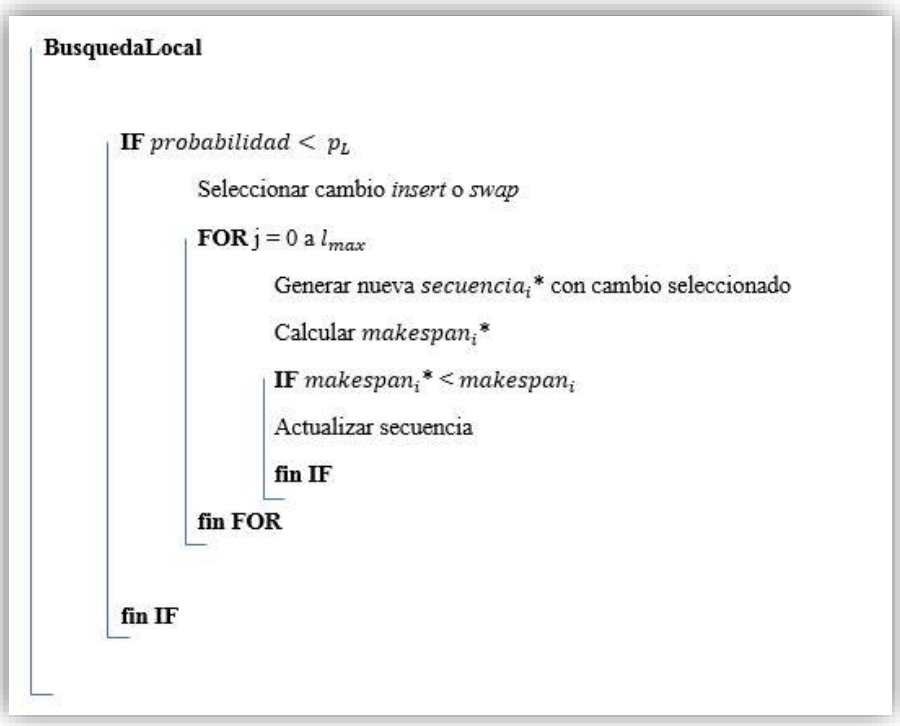


Figura 4.4 – Pseudocódigo de búsqueda local.
[Fuente: Elaboración propia]

4.4 Fase de abeja espectadora (onlooker)

La fase de la abeja espectadora consiste en evaluar las soluciones encontradas por las abejas empleadas, con objeto de identificar aquellas más prometedoras. Para ello se evalúan algunos miembros de la población y se intentan mejorar dichas secuencias. Los cambios utilizados son seleccionados de la lista *NL* como se explica en el apartado 4.6.

El algoritmo considera el mismo número de abejas espectadoras que empleadas, es decir, esta fase también se realiza SN veces.

Las variantes del algoritmo propuestas en este trabajo consisten en pequeñas variaciones en el procedimiento a seguir en esta fase, en concreto en los distintos criterios utilizados para seleccionar estos miembros candidatos a ser explotados. Estos criterios de selección son explicados más adelante en el capítulo 5, dedicado íntegramente a las variantes del algoritmo.

El criterio de selección propuesto por Pan y otros (2009) es un torneo de tamaño 2. Este torneo que consiste en escoger aleatoriamente 2 secuencias de la población y la que obtenga un mejor resultado de la función objetivo es seleccionada.

Podemos ver el procedimiento de esta fase en la Figura 4.5.

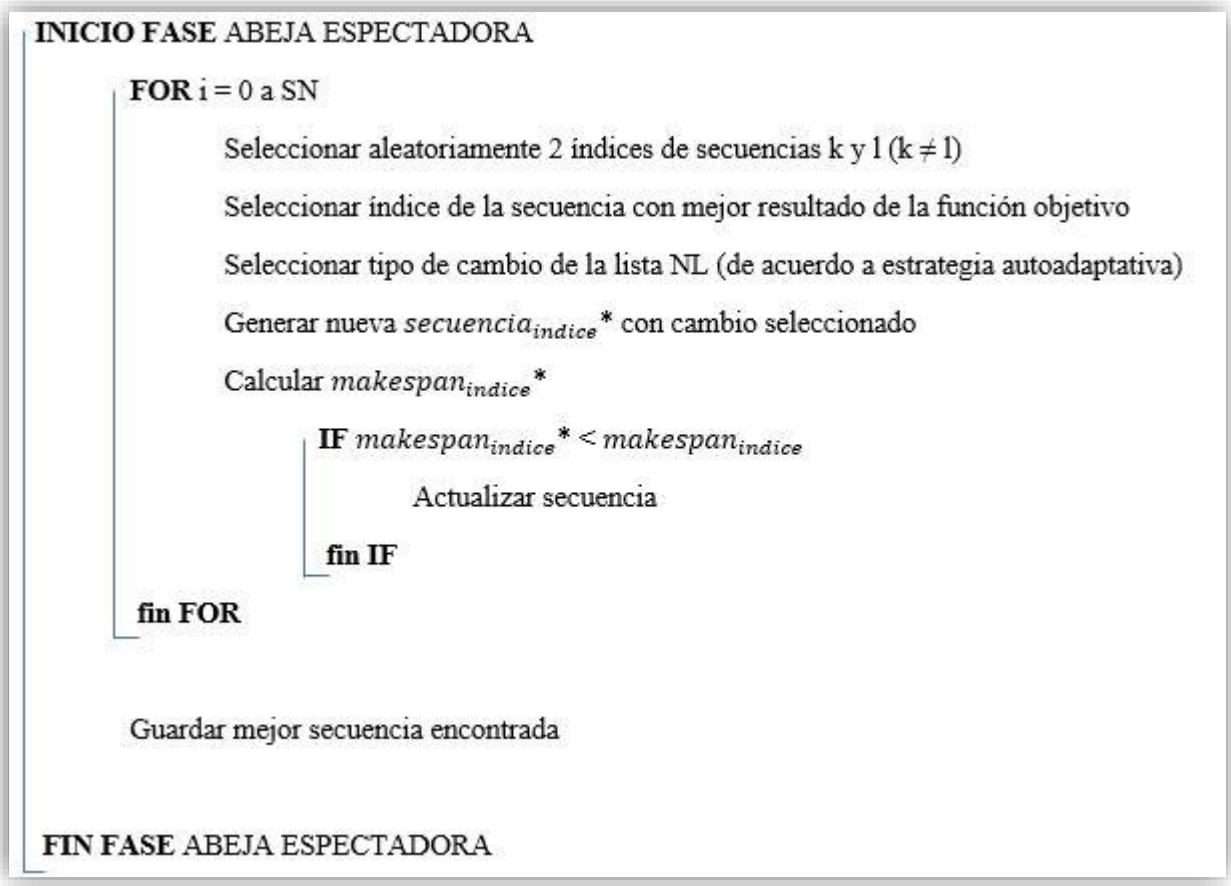


Figura 4.5 - Pseudocódigo de fase abeja espectadora.
[Fuente: Elaboración propia]

4.5 Fase de abeja exploradora (scout)

Por último, la fase de la abeja exploradora consiste en que cuando se considera que una fuente ha sido ya explotada, ésta es abandonada y sustituida por otra en el área de búsqueda.

Análogamente en nuestro problema, si se alcanza un número de intentos de mejora fallidos igual o superior a *limit* para la misma secuencia, se genera otra secuencia de forma aleatoria.

Esta secuencia se genera a partir de la mejor secuencia encontrada hasta el momento, pero en lugar de los cambios de las fases anteriores, se aplican varios cambios insert sobre dicha secuencia, consiguiendo así evitar quedar atrapado en un óptimo local (ver Figura 4.6). Una vez se ha generado esta nueva secuencia, reseteamos el contador de intentos fallidos a 0.

El procedimiento de esta última fase se muestra en la Figura 4.7.

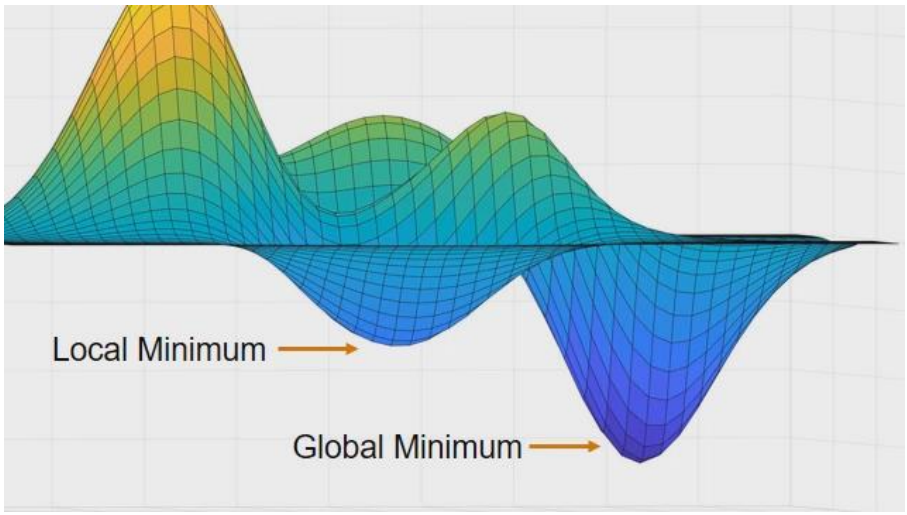


Figura 4.6 - Representacion 3D de optimos local y global.
[Fuente: Extraída de la web de MathWorks]

INICIO FASE ABEJA EXPLORADORA
  FOR i = 0 a SN
    IF $intentos\_fallidos_i > limit$
      $secuencia_i = mejor\_secuencia$
      Aplicar cambios *insert* a $secuencia_i$
    fin IF
    $intentos\ fallidos_i = 0$
  fin FOR
FIN FASE ABEJA EXPLORADORA

Figura 4.7 – Pseudocódigo de la fase abeja exploradora.
[Fuente: Elaboración propia]

4.6 Estrategia autoadaptativa para producir soluciones

Tanto las abejas empleadas como las espectadoras utilizan un mecanismo que intenta adaptarse a cada fase concreta del proceso evolutivo del algoritmo. Este mecanismo funciona como se explica a continuación:

Inicialmente, se genera una lista *NL*, rellenándola aleatoriamente con los 4 tipos de cambios mencionados en el apartado 4.3. A lo largo del proceso del algoritmo, los cambios para producir nuevas soluciones por las abejas empleadas o espectadoras serán seleccionados de esta lista.

En caso de usar un cambio de esta lista y tener éxito, es decir, produzca una secuencia con un mejor valor de la función objetivo, dicho cambio entrará en otra lista de cambios favoritos (WNL).

Una vez que se vacíe la lista NL, se volverá a rellenar teniendo cada elemento una probabilidad del 75% de ser tomado de la lista de cambios favoritos o WNL. El resto de los elementos serán seleccionados aleatoriamente de los 4 cambios mencionados en el apartado 4.3 como se hizo inicialmente.

La nueva lista será usada nuevamente para realizar cambios sobre las secuencias, y este proceso descrito se repetirá hasta que se alcance un criterio de finalización. La lista de cambios favoritos o WNL es vaciada tras crear una nueva lista NL, evitando así efectos acumulativos a largo plazo.

Como resultado, el tipo de cambio apropiado en cada momento del proceso de búsqueda puede ser gradualmente aprendido por el propio algoritmo.

5 VARIANTES DEL ALGORITMO

Las variaciones del algoritmo se han realizado en la fase de abeja espectadora o onlooker, concretamente a la hora de elegir los miembros de la población susceptibles de cambio. Esta fase juega un papel muy importante en el algoritmo, ya que decide donde centrar los esfuerzos de búsqueda.

Haciendo demasiado hincapié solo en las mejores secuencias, nos limitamos a unas áreas de búsqueda determinadas, pudiendo quedar atrapados en un óptimo local, mientras que dando oportunidad a secuencias a priori poco prometedoras se dispersa demasiado la búsqueda dificultando alcanzar una solución óptima al problema. Es por ello por lo que se debe intentar evitar estos dos extremos, conciliando la exploración global con la explotación local.

Hemos realizado 7 variaciones del algoritmo en este aspecto, en todos los casos realizando SN intentos de mejora sobre las secuencias.

En primer lugar, hemos realizado 3 variantes que consisten en un torneo, en el cúal compiten las secuencias entre ellas y solo la mejor es seleccionada, teniendo así:

- $\boldsymbol{ABC}_{[1]}$: Se realiza un torneo de tamaño 2 que se corresponde con la variante utilizada por Pan y otros (2009). En este torneo se selecciona la mejor secuencia entre 2 escogidas aleatoriamente, ver Sección 4.4 para más detalle.

- $\boldsymbol{ABC}_{[2]}$: Igual que en el caso anterior pero con un torneo de tamaño 3 secuencias, es decir, se escogen aleatoriamente 3 secuencias, se comparan entre ellas y la mejor es seleccionada.

- $\boldsymbol{ABC}_{[3]}$: Se asigna a cada secuencia una *probabilidad* ϵ (0,1), entrando en el torneo aquellas que tengan una probabilidad menor a 0.4.

Para el resto de los casos, de entre las SN=10 (ver valores de los parámetros en el Anexo 1) veces que se realiza la fase, en lugar de un torneo directamente repartimos el número de intentos entre las mejores secuencias de la población (Pauzi, y otros, 2015). Por ejemplo, (4,3,3) sería repartir los 10 intentos asignando 4 intentos a la mejor secuencia, 3 intentos a la segunda mejor secuencia, y los otros 3 a la tercera mejor secuencia. Siguiento esta notación, las restantes variantes quedarían como sigue:

- $\boldsymbol{ABC}_{[4]}$: 5 intentos para la mejor secuencia y 5 para la segunda. (5 ,5).

- $\boldsymbol{ABC}_{[5]}$: 2 Intentos de mejora para cada una de las 5 mejores secuencias. (2, 2, 2, 2, 2).

- $\boldsymbol{ABC}_{[6]}$: (3, 2, 2, 1, 1, 1) Intentos de mejora para las mejores secuencias respectivamente.

- $\boldsymbol{ABC}_{[7]}$: (3, 3, 2, 2) Intentos de mejora para las mejores secuencias respectivamente.

Con estas variantes se pretende abordar la problemática antes comentado, de balancelar la búsqueda global con la explotación local de una manera variada. Algunas variantes se centran solo en las mejores secuencias, como por ejemplo la variante $\boldsymbol{ABC}_{[4]}$, mientras que otras involucran y dan oportunidad de mejora a un mayor número de ellas.

Las variantes no tienen por qué ser mejor en términos absolutos y para todos los casos, sino que pueden ser más o menos adecuadas, teniendo en cuenta las características del problema concreto considerado, como es la dimensión de los mismos. Los resultados obtenidos con las variantes para cada una de las instancias propuestas se muestran más adelante en el capítulo de evaluación computacional.

6 EVALUACIÓN COMPUTACIONAL

En este bloque analizaremos la eficiencia y resultados obtenidos al resolver el problema con el algoritmo propuesto y sus variaciones. Como comentamos anteriormente se trata de un problema en un entorno de Flow shop bajo la restricción no-idle o inactividad en máquinas no permitida.

Tanto los parámetros del algoritmo como las instancias consideradas se han establecido de acuerdo a Pan y otros (2009). Tenemos un total de 20 instancias diferentes de acuerdo a los siguientes tamaños de trabajos y máquinas respectivamente, (n = 10, 20, 30, 40; m = 3, 5, 7, 10, 15). Los tiempos de trabajos se generan aleatoriamente siendo $p_{i,j} \in (0,99)$.

Los parámetros del algoritmo son los siguientes: *SN* = 10 (miembros de la población), *limit* = 20 (número de intentos fallidos antes de abandonar fuente de alimento), p_L = 0.1 (probabilidad de realizar búsqueda local), y $l_{max} = n^2$ (número de iteraciones en la búsqueda local). En cuanto al tiempo de proceso, se establece *CT* = $5n^2m$ ms como criterio de terminación.

6.1 Resultados obtenidos

Para cada instancia se han realizado 10 simulaciones con cada una de las variantes del algoritmo. El valor medio del makespan de todas las simulaciones en cada instancia se resume en la Tabla 6.1.

m x n	$ABC_{[1]}$	$ABC_{[2]}$	$ABC_{[3]}$	$ABC_{[4]}$	$ABC_{[5]}$	$ABC_{[6]}$	$ABC_{[7]}$
10 x 3	667,6	575,3	680,8	585,9	630,2	636,3	622,6
10 x 5	811,9	822,2	841	855,3	870,4	790,4	807,5
10 x 7	981,9	1001,8	986,5	1044,1	997,1	1029	1027,7
10 x 10	1355,3	1319,2	1304,3	1421,6	1356,6	1469,6	1360,1
10 x 15	1899,7	1876,7	1964,6	1877,1	1854,5	2022,3	1980,9
20 x 3	1248,7	1175,4	1156,8	1139,5	1174,1	1112,2	1164,5
20 x 5	1425,4	1340,6	1322,2	1333,7	1346,5	1361,7	1378,5
20 x 7	1577,9	1590,4	1695,9	1673,7	1554,2	1556,9	1626,1
20 x 10	2030,1	2120,3	1856,5	2064,1	2015,8	2077,4	2038,6
20 x 15	2666,4	2686	2614	2700,8	2822,5	2778,6	2774,9
30 x 3	1795	1673,5	1658,3	1669,6	1676,1	1677	1683,1
30 x 5	1972,8	1951	2030	1932,3	1897,5	1998,5	1995,4
30 x 7	2160,4	2202	2183,4	2305	2313,8	2293,5	2277,8
30 x 10	2618,1	2553,3	2651,9	2628,2	2629,7	2716	2753,6
30 x 15	3450,7	3690,7	3303,1	3438,2	3498,4	3526,1	3542,3
40 x 3	2210,2	2267,8	2143,5	2184,2	2284,7	2260,1	2193,1
40 x 5	2373,9	2412,9	2555,5	2524,9	2510,3	2563,4	2477,9
40 x 7	2871,9	2770,6	2807	2963,7	2793,4	2720	2689,8
40 x 10	3151,1	3263,2	3227,5	3365	3335,5	3372,3	3284,7
40 x 15	3818,2	3890,3	3859	3915	4044	3993,4	4055

Tabla 6.1 – Makespan obtenido con las variantes del algoritmo. [Elaboración propia]

Calculamos los valores del porcentaje de incremento relativo (RPI) para cada instancia y variante, con la siguiente fórmula:

$$RPI = \frac{O - C^*}{C^*}$$

Siendo O el valor de la función objetivo y C^* el mejor valor encontrado en dicha instancia.

Este valor refleja de manera más clara los resultados obtenidos, ya que son relativos al mejor resultado obtenido en cada instancia. Tambien permite sacar promedios de los resultados de todas o parte de las instancias, al ser porcentajes y no números absolutos como los valores del makespan de la tabla anterior.

A continuación, se muestran los valores obtenidos para todas las instancias por cada variante del algoritmo y el promedio total de cada una de ellas (ver Tabla 6.2). El mejor resultado global es de la segunda variante, aunque no se observan resultados demasiado dispares entre las distintas variantes, a diferencia de si solo consideramos casos más concretos del problema, como veremos mas adelante.

m x n	$ABC_{[1]}$	$ABC_{[2]}$	$ABC_{[3]}$	$ABC_{[4]}$	$ABC_{[5]}$	$ABC_{[6]}$	$ABC_{[7]}$
10 x 3	16,04	0,00	18,34	1,84	9,54	10,60	8,22
10 x 5	2,72	4,02	6,40	8,21	10,12	0,00	2,16
10 x 7	0,00	2,03	0,47	6,33	1,55	4,80	4,66
10 x 10	3,91	1,14	0,00	8,99	4,01	12,67	4,28
10 x 15	2,44	1,20	5,94	1,22	0,00	9,05	6,82
20 x 3	12,27	5,68	4,01	2,45	5,57	0,00	4,70
20 x 5	7,81	1,39	0,00	0,87	1,84	2,99	4,26
20 x 7	1,52	2,33	9,12	7,69	0,00	0,17	4,63
20 x 10	9,35	14,21	0,00	11,18	8,58	11,90	9,81
20 x 15	2,00	2,75	0,00	3,32	7,98	6,30	6,16
30 x 3	8,24	0,92	0,00	0,68	1,07	1,13	1,50
30 x 5	3,97	2,82	6,98	1,83	0,00	5,32	5,16
30 x 7	0,00	1,93	1,06	6,69	7,10	6,16	5,43
30 x 10	2,54	0,00	3,86	2,93	2,99	6,37	7,84
30 x 15	4,47	11,73	0,00	4,09	5,91	6,75	7,24
40 x 3	3,11	5,80	0,00	1,90	6,59	5,44	2,31
40 x 5	0,00	1,64	7,65	6,36	5,75	7,98	4,38
40 x 7	6,77	3,00	4,36	10,18	3,85	1,12	0,00
40 x 10	0,00	3,56	2,42	6,79	5,85	7,02	4,24
40 x 15	0,00	1,89	1,07	2,54	5,91	4,59	6,20
Media	**4,36**	**3,40**	**3,58**	**4,81**	**4,71**	**5,52**	**5,00**

Tabla 6.2 -Valores de RPI obtenidos. [Elaboración propia]

En los apartados que siguen, se muestran y analizan los valores del RPI divididos según el número de trabajos, quedando así 4 grupos de instancias de 10, 20, 30 y 40 trabajos, respectivamente. De esta manera podemos analizar el desempeño de cada unas las variantes para casos de problema concretos.

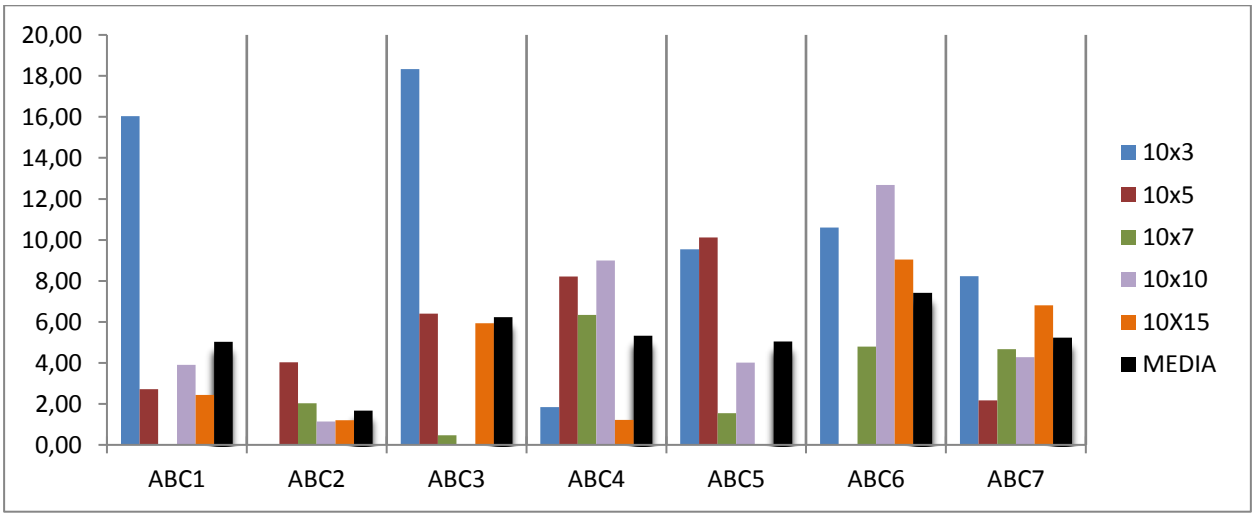
6.1.1 Instancias con 10 trabajos

Para el primer grupo, con 10 trabajos, la segunda variante obtiene el mejor resultado promedio, como podemos ver en los resultados de la media de la Tabla 6.3.

En la Gráfica 6.1 tambien podemos observar estos resultados, siendo la media representada mediane la barra de color negro. Además, también podemos observar con mayor claridad los valores del RPI para cada instancia de trabajo en cada variante del algoritmo.

m x n	$ABC_{[1]}$	$ABC_{[2]}$	$ABC_{[3]}$	$ABC_{[4]}$	$ABC_{[5]}$	$ABC_{[6]}$	$ABC_{[7]}$
10 x 3	16,04	0,00	18,34	1,84	9,54	10,60	8,22
10 x 5	2,72	4,02	6,40	8,21	10,12	0,00	2,16
10 x 7	0,00	2,03	0,47	6,33	1,55	4,80	4,66
10 x 10	3,91	1,14	0,00	8,99	4,01	12,67	4,28
10 x 15	2,44	1,20	5,94	1,22	0,00	9,05	6,82
Media	**5,02**	**1,68**	**6,23**	**5,32**	**5,04**	**7,42**	**5,23**

Tabla 6.3 - RPI en instancias de 10 trabajos. [Elaboración propia]



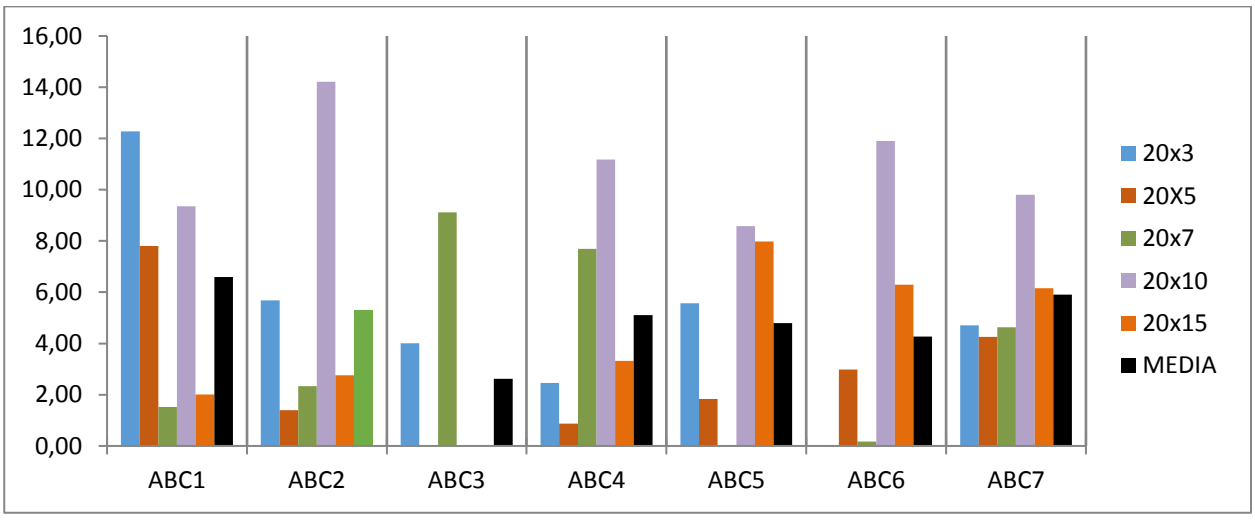
Gráfica 6.1 – RPI de instancias de 10 trabajos. [Elaboración propia]

6.1.2 Instancias con 20 trabajos

Para el caso de instancias con 20 trabajos, podemos observar que la tercera variante obtiene los mejores resultados (ver Tabla 6.4), teniendo el mejor valor promedio además de obtener el mejor valor en 3 de las 5 instancias consideradas, como podemos observar en la Gráfica 6.2.

m x n	$ABC_{[1]}$	$ABC_{[2]}$	$ABC_{[3]}$	$ABC_{[4]}$	$ABC_{[5]}$	$ABC_{[6]}$	$ABC_{[7]}$
20 x 3	12,27	5,68	4,01	2,45	5,57	0,00	4,70
20 x 5	7,81	1,39	0,00	0,87	1,84	2,99	4,26
20 x 7	1,52	2,33	9,12	7,69	0,00	0,17	4,63
20 x 10	9,35	14,21	0,00	11,18	8,58	11,90	9,81
20 x 15	2,00	2,75	0,00	3,32	7,98	6,30	6,16
Media	**6,59**	**5,27**	**2,63**	**5,10**	**4,79**	**4,27**	**5,91**

Tabla 6.4 - RPI en instancias de 20 trabajos. [Elaboración propia]



Gráfica 6.2 - RPI de instancias de 20 trabajos. [Elaboración propia]

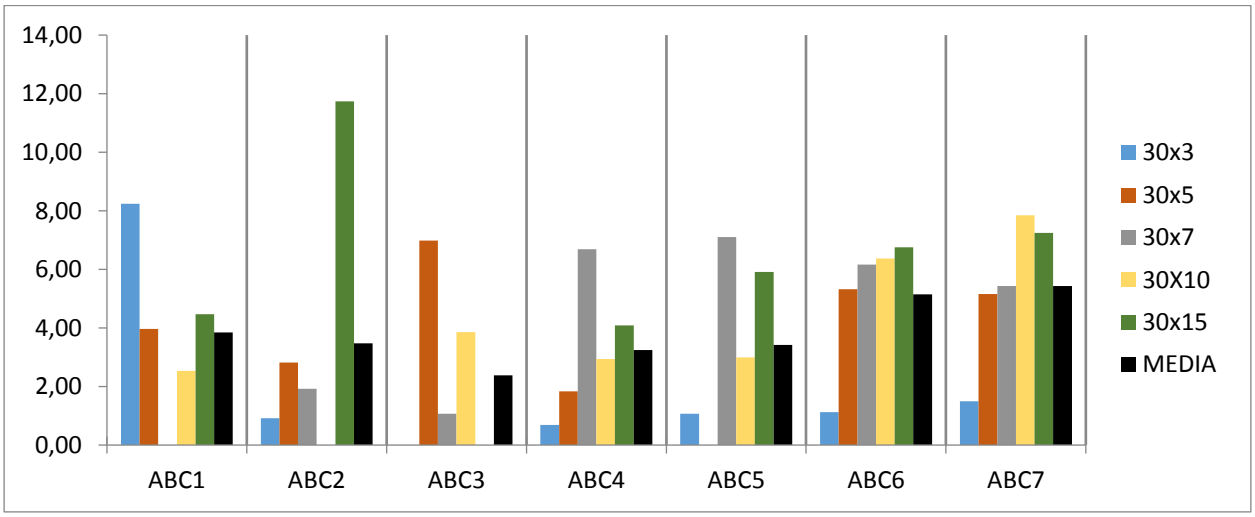
6.1.3 Instancias con 30 trabajos

Para instancias de 30 trabajos, el tercer algoritmo obtiene el mejor resultado promedio, aunque no con una diferencia muy signifivatica, siendo este muy similar al obtenido con otras variantes (ver Tabla 6.5).

Los variantes sexta y séptima obtienen los peores resultados.

m x n	$ABC_{[1]}$	$ABC_{[2]}$	$ABC_{[3]}$	$ABC_{[4]}$	$ABC_{[5]}$	$ABC_{[6]}$	$ABC_{[7]}$
30 x 3	8,24	0,92	0,00	0,68	1,07	1,13	1,50
30 x 5	3,97	2,82	6,98	1,83	0,00	5,32	5,16
30 x 7	0,00	1,93	1,06	6,69	7,10	6,16	5,43
30 x 10	2,54	0,00	3,86	2,93	2,99	6,37	7,84
30 x 15	4,47	11,73	0,00	4,09	5,91	6,75	7,24
Media	**3,84**	**3,48**	**2,38**	**3,25**	**3,42**	**5,15**	**5,44**

Tabla 6.5 - RPI en instancias de 30 trabajos. [Elaboración propia]



Gráfica 6.3 - RPI de instancias de 30 trabajos. [Elaboración propia]

6.1.4 Instancias con 40 trabajos

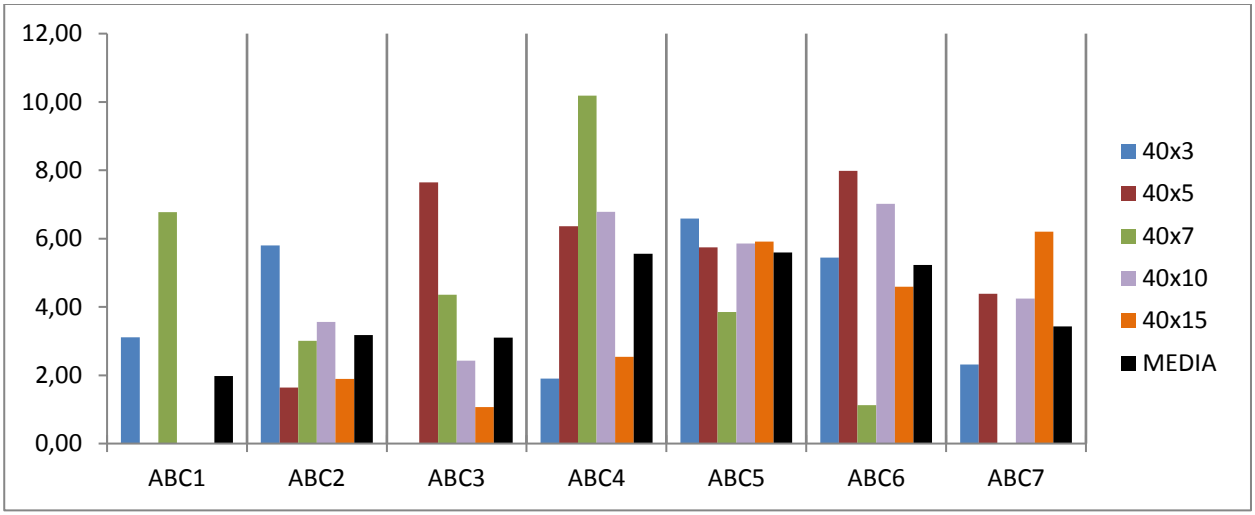
Por úlimo consideramos el caso de instancias de 40 trabajos, donde es el primer algoritmo el que obtiene el mejor resultado como se observa en la Tabla 6.6, existiendo mayor diferencia respecto a las otras variantes que en el caso anterior.

Dicho algoritmo obtiene el mejor resulado en 3 de las instancias consideradas. No obstante, no arroja un buen resultado en el caso concreto de 40 trabajos y 7 máquinas.

Por otro lado, las variantes cuarta, quinta y sexta no obtienen un buen resultado promedio, además de no encontrar la mejor solución en ninguna de las instancias.

m x n	$ABC_{[1]}$	$ABC_{[2]}$	$ABC_{[3]}$	$ABC_{[4]}$	$ABC_{[5]}$	$ABC_{[6]}$	$ABC_{[7]}$
40 x 3	3,11	5,80	0,00	1,90	6,59	5,44	2,31
40 x 5	0,00	1,64	7,65	6,36	5,75	7,98	4,38
40 x 7	6,77	3,00	4,36	10,18	3,85	1,12	0,00
40 x 10	0,00	3,56	2,42	6,79	5,85	7,02	4,24
40 x 15	0,00	1,89	1,07	2,54	5,91	4,59	6,20
Media	**1,98**	**3,18**	**3,10**	**5,55**	**5,59**	**5,23**	**3,43**

Tabla 6.6 - RPI en instancias de 40 trabajos. [Elaboración propia]



Gráfica 6.4 - RPI de instancias de 40 trabajos. [Elaboración propia]

7 CONCLUSIONES

Este trabajo trata de abordar un problema de programación de la producción, en concreto un problema en un entorno de flow shop con objetivo de la minimización del makespan. Consideramos la restricción no-idle, que no permite la inactividad en máquinas una vez que éstas han iniciado a procesar el primer trabajo.

Para la resolución de este problema hemos utilizado un algoritmo de optimización, llamado *Artificial Bee Colony* (ABC), que utiliza la inteligencia de enjambre para encontrar un resultado óptimo al problema. Este algoritmo simula el comportamiento real de las abejas al buscar fuentes de alimento.

Hemos realizado diversas variantes del algoritmo, concretamente en la fase de abeja espectadora o *onlooker*, explicadas anteriormente en este documento. Atendiendo a los resultados obtenidos, las conclusiones alcanzadas son las siguientes:

- La segunda variante ha obtenido los mejores resultados promedios. Esta variante propone la realización de un torneo de tamaño 3 a la hora de elegir las secuencias que son consideradas más prometedoras y que merecen mayor esfuerzo de ser explotadas por parte del algoritmo.
- No obstante, otras variantes han demostrado una mejor eficiencia para instancias concretas entre las propuestas, como por ejemplo la primera variante para las instancias con 40 trabajos, o la tercera para instancias de 20 y 30 trabajos.

Como líneas de investigación futura, se podrían explorar variaciones del algoritmo en otras fases del mismo, además de aplicarlas a otros problemas con características diferentes, que tampoco han sido estudiados suficientemente en la literatura.

REFERENCIAS

Baraz, Daniel y Mosheiov, Gur A note on a greedy heuristic for the flow-shop makespan minimization with no machine idle-time [Publicación periódica]. - [s.l.] : European Journal of Operational Research, 2008.

Galera Prieto, Mario Algoritmos para la programación de la producción en un entorno de flujo regular distribuido de permutación [Libro]. - 2017.

Karaboga, Dervis An Idea Based on Honey Bee Swarm for Numerical Optimization [Libro]. - 2005.

Pan, Quan-Ke [y otros] A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem [Publicación periódica]. - [s.l.] : Elsevier, 2009.

Pauzi Nur Fazlinda Binti Mohd y Bareduan Salleh Ahmad Scheduling analysis for flowshop using artificial bee colony (ABC) algorithm with varying onlooker [Libro]. - 2015.

Pérez, Paz Apuntes de Programacion de Operaciones [Libro]. - 2017.

Pinedo, Michael Sheduling: Theory, Algorithms, and Systems [Libro]. - [s.l.] : Springer, 2014.

Tasgetiren, M. Fatih [y otros] A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion [Publicación periódica]. - [s.l.] : Elsevier, 2013.

Vollmann, Thomas Manufacturing Planning and control for supply chain management [Libro]. - 2005.

ANEXOS

ANEXO 1. Párametros del algoritmo

- SN = 10 — Número de secuencias
- pL = 0.1 — Probabilidad de realizar búsqueda local
- limite = 20 — Número de intentos fallidos antes de abandonar solución
- lmax = n^2 — Número de iteraciones de la búsqueda local
- valorNL=(0,1,2,3) — Tipo de cambio a realizar
- probabilidadWNL = 0.75 — Probabilidad de seleccionar cambio de lista de cambios favoritos
- CT = $5n^2m$ — Tiempo de terminación del algoritmo

ANEXO 2. Funciones de la biblioteca schedule utilizadas

Variables: Definición.
- VECTOR_INT MAT_INT
- VECTO_LONG MAT_LONG
- VECTOR FLOAT MAT_FLOAT
- VECTOR DOUBLE MAT DOUBLE

Variables: Dimensionamiento.
- DIM_VECTOR_INT(n_elementos); DIM_MAT_INT(n_filas, n_cols);
- DIM_VECTOR_LONG(n_elementos); DIM_MAT_LONG(n_filas, n_cols);
- DIM_VECTOR_FLOAT(n_elementos); DIM_MAT_FLOAT(n_filas, n_cols);
- DIM_VECTOR_DOUBLE(n_elementos); DIM_MAT_DOUBLE(n_filas, n_cols);

Variables: Liberación.
- free(VECTOR);
- free_mat_int(MAT_INT, rows);
- free_mat_long(MAT_LONG, rows);
- free_mat_float(MAT_FLOAT,rows);
- free_mat_double(MAT_DOUBLE,rows);

Generar una secuencia aleatoria con un número de elementos definidos.
- void randSequence(VECTOR_INT vector, int numero_elementos)

Insertar un valor en la posición de un vector.
- void insertIVector(vector, tamaño, valor, posición)

Eliminar un elemento de la posición de un vector. Devuelve el tamaño del vector.
- int extractIVector(vector, tamaño, posición)

Busca un valor concreto en un vector (selecciona la primera apaeción).
- int searchInIVector(vector, rows, numeo_buscado)
- int searchInLVector(vector, rows, numero_buscado)
- int searchInFVector(vector, rows, numero_buscado)
- int searchOInIVector(vector, ows, numero_buscado, pos_desde)

Crea un nuevo vector y copiar uno en éste.
- void copyIVector(original, destino, tamaño)
- void copyLVector(original, destino, tamaño)
- void copyFVector(original, destino, tamaño)
- void copyDVector(origina, destino, tamaño)

Ordena el vector en función de sus valores de manera ascendente o descendente.
- void sortIVector(vector, numero_elementos, orden)

Crea una matriz con tiempos de procesos, numero de máquinas y trabajos.
- VECTOR_INT sortPT(num_trab, num_maq, tiempos_proc, criterio)