Proyecto Fin de Carrera

Grado en Ingeniería de las Tecnologías Industriales

Algoritmos para la programación de la producción en un entorno de flujo regular distribuido de permutación

Autor: Mario Galera Prieto

Tutor: José Manuel Framiñan Torres

Dep. Organización Industrial y Gestión de Empresas 1
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado Ingeniería
de las Tecnologías Industriales

Algoritmos para la programación de la producción en un entorno de flujo regular distribuido de permutación

Autor:
Mario Galera Prieto

Tutor:
Dr. José Manuel Framiñan Torres
Catedrático de Universidad

Dep. de Organización Industrial y Gestión de Empresas 1
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Proyecto Fin de Carrera: Algoritmos para la programación de la producción en un entorno de flujo regular distribuido de permutación

Autor: Mario Galera Prieto

Tutor: Dr. José Manuel Framiñan Torres

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Agradecimientos

Este trabajo es la culminación a años de esfuerzos y dedicación, con una gran adquisición de conocimientos a través del aprendizaje con los que poder hacer frente a nuevas etapas mediante la aplicación de éstos. Agradecer a mis padres y mi hermano por el apoyo y confianza mostrada en mí en estos duros años y por tenderme la mano cuando lo he necesitado. A todos los que alguna vez me han ayudado y en especial a Esther por ser un pilar fundamental en el éxito conseguido.

Por último agradecer a José Manuel Framiñán Torres por toda la paciencia y tiempo dedicado en mí como guía durante la elaboración de este proyecto.

Mario Galera Prieto

Sevilla, 2017

Resumen

Este proyecto implica la documentación, estudio y posterior implementación de un Algoritmo de Optimización basado en la Biogeografía para un tipo Flow-Shop de permutación distribuido. Estos tipos de problemas consta de dos etapas: producción y montaje (en este documento nos centramos en la etapa de producción), en la que los productos se producen de forma secuencial. Éstos se elaboran por partes en distintas fábricas para que en el caso de que fallase alguna, la producción no se detenga. Para ello, habrá que tener claro en qué se fundamenta la optimización basada en la biogeorafía y como funciona el Algoritmo. Después de hacer la correspondiente implementación se transcurrirá a la comparación de resultados entre los distintos algoritmos que existen para resolver este problema: algoritmo HBBO y la Heurística basada en la inserción (HBI) de Hatami (2013).

Antes de describir el problema es necesario tener unos conocimientos previos, como por ejemplo qué es un Flow-Shop, qué tipos de entornos existen en la programación de operaciones, los tipos de restricciones que se pueden dar y qué tipo de objetivos se plantean en las fábricas. Todas estas explicaciones están elaboradas en el capítulo 2 del documento.

En la descripción de los algoritmos que se ha realizado en el apartado 3 se han incorporado la explicación de las funciones más importantes utilizadas, además de alguna suposición que se ha tomado debido a qué no se han tenido los datos suficientes para abordarlo o se han detectado errores en el documento Jian y Shuai (2016).

En la implementación se ha utilizado un progama llamado Code::Blocks (comentado en el objeto del problema) que se utiiza para codificar en C, junto con la librería <schedule_lib.h>, para hacer más sencilla la imlementación, ya que esta libería posee una serie de funciones que son utilizadas en la elaboración del documento y facilita la inicialización de elementos como vectores o matrices.

Para la comparación de estos dos métodos de programación, se han utilizado una serie de datos que han sido recopilados del documento Jian y Shuai (2016) y que se expresan en el capítulo 4 del proyecto. Los datos a emplear son el número de trabajos (n), el número de máquinas (m), el número de fábricas (F), la cantidad de instancias (smax) y el número de iteraciones máximas a realizar en cada experimento ($Iter_{max}$). Estos datos tienen dos bloques diferenciados: instancias de pequeño tamaño e instancias de gran tamaño. Los experimentos a elaborar consistirán en la combinación del número de trabajo, máquinas y fábricas (n x m x F) teniendo en cuenta que para cada tamaño de instancias se usa una cantidad de hábitats y número de iteraciones diferentes. Para cada experimento se obtendrá las soluciones que serán comparadas con el fin de comprobar la eficacia del algoritmo.

Todos los resultados que se han obtenido se pueden apreciar en el apartado 4 de la memoria, que con la ayuda del Excel para crear tablas y gráficas se han podido realizar la comparación entre ambos métodos y sacar una conclusión. La conclusión final que se puede deducir es que para pequeñas instancias (número de trabajos, máquinas y fábricas pequeños) el algoritmo HBBO es más eficiente que el algoritmo HBI, pero para instancias de gran tamaño (número de trabajos, máquinas y fábricas más grandes) el algoritmo HBBO da peores resultados que el algoritmo HBI. En conclusión, si se trabaja con instancias de pequeño tamaño es más adecuado la utilización del algoritmo HBBO para obtener la mejor solución, pero para instancias de gran tamaño, el uso de algoritmo HBBO resulta ineficiente y el tiempo de obtención de los resultados es mucho mayor, siendo así la utilización del algoritmo HBI una mejor opción.

Índice

Índice de Tablas

Índice de Figuras

Notación

GITI	Grado en Ingeniería de las Tecnologías Industriales
PO	Programación de Operaciones
HBBO	Nombre del algoritmo (Hybrid biogeography-based optimization)
HBI	Heurística Basada en la Inserción
N	Número de hábitats
F	Número de fábricas
n	Número de trabajos
m	Número de máquinas
λ	Ratio de inmigración
μ	Ratio de emigración
m_r	Probabilidad de mutación
m_{max}	Probabilidad máxima de mutación
C_{max}	Objetivo del problema; minimizar el tiempo de finalización de todos los trabajos
Makespan	Diferencia de tiempos entre el inicio y el final de una secuencia de trabajos o tareas
Algoritmo	Conjunto de pasos estructurados y ordenados que permiten realizar una actividad
Heurística	Conjunto de ténicas o métodos para resolver un problema en poco tiempo pero que no siempre es eficiente.
Flow-Shop	Tipo de problema en optimización dónde los trabajos se realizan en serie por cada máquina
Permutación	Hay una sola secuencia para todas las máquinas
α	Característica de la máquina
r_j	Fecha de llegada del trabajo j
d_j	Fecha de entrega de trabajo j
β	Característica de los trabajos
s_{ij}	Tiempo de setup en la máquina i antes de procesar el trabajo j
p_{ij}	Tiempo de proceso del trabajo i en la máquina j
ÍNDICES	
i	Índice para trabajos donde i=1,…,n
j	Índice para máquinas donde j=1,…,m
f	Índice para fábricas donde f=1,…,F

1 INTRODUCCIÓN

1.1. Objeto del trabajo

El objeto del trabajo es el de analizar y comparar los mejores algoritmos disponibles para encontrar la solución de la manera más eficiente posible, para un entorno de flujo regular distribuido de permutación, en el que los trabajos se podrán procesar en distintas fábricas y máquinas.

Es conveniente tener en cuenta el tiempo de ejecución a la hora de seleccionar un algoritmo u otro y que el valor obtenido del makespan sea el menor posible. El makespan consiste en la diferencia de tiempos entre el inicio de la producción (en el momento en el que se empieza a realizar el primer trabajo) hasta el tiempo de terminación del último trabajo.

Para ello, se analizarán los resultados con una serie de datos de entrada como son el número de trabajos, el número de máquinas, el número de fábricas, etc. La heurística empleada es denominada por HBI, siendo una simplificación de la que se encuentra en el artículo de Hatami (2013) y está basada en la heurística de Framiñan y Leisten (2003). Emplea la inserción de trabajos dentro de una misma secuencia en la búsqueda de la mejor solución.

La codificación de los algoritmos se ha llevado a cabo mediante el programa Code::Blocks y el lenguaje de programación C. Este programa se ha elegido por su facilidad con la que se permite crear elementos como matrices o vectores mediante la librería <schedule_lib.h> que ha sido facilitada por el tutor de este trabajo (Framiñan, 2016). Esta biblioteca tiene multitud de funciones declaradas que pueden ser utilizadas, ahorrando así, tiempo en la confección del código. Todas las funciones de las que dispone la librería están descritas en el Anexo 1 al final del documento.

Para comprobar la eficacia del sistema propuesto, se ha llevado a cabo la comparación entre el algoritmo HBBO y la Heurística basada en la inserción (HBI). En esta comparación se utilizan una serie de datos que se muestran en Jian y Shuai (2016). Se ha comparado para distintos números de fábricas, máquinas y trabajos para obtener un abanico de soluciones con las que poder realizar una conclusión más contrastada. Para hacer más simple la visualización de los resultados se utiliza el programa Excel en la elaboración de tablas y gráficas.

Después de realizar la comparación entre las soluciones de los algoritmos se llega a la conclusión de que para instancias pequeñas, el uso del algoritmo HBBO resulta más eficiente, encontrando mejores soluciones, mientras que para instancias de gran tamaño, las soluciones del algoritmo HBBO resultan peores y los tiempos de computación son demasiados elevados, perdiendo así, mucho tiempo en la búsqueda del resultado. Por tanto, para instancias de gran tamaño es conveniente el uso del algoritmo HBI.

1.2. Justificación

"Los problemas tipo flowshop de permutación es un problema de optimización combinatoria ampliamente investigado y juega un papel muy importante en los sistemas de fabricación y procesos industriales (Jian & Shuai, 2016)"

La programación de la producción es una parte fundamental en el crecimiento de los sistemas de fabricación ya que al conocer la mejor solución en cuanto al orden de fabricación de un producto, se ahorra en tiempos de ejecución, lo que a la larga supone un beneficio para la propia empresa.

El estudio de algoritmos en la búsqueda de soluciones para entornos de flujo regular resulta de interés, dado que en la mayoría de las fábricas, los productos que se elaboran siguen un determinado orden de procesamiento y son introducidos en las máquinas de una manera ya predefinida. Por ello, tiene gran importancia la obtención de un buen algoritmo que sea capaz de encontrar un resultado óptimo o al menos, cercano a él.

Los entornos de flujo regular (en los que hay más de una fábrica, y cada una de ellas tiene m máquinas) cuenta con la ventaja de que si se produce un fallo en una máquina de alguna de las fábricas, las demás pueden seguir con su producción y no tener que parar, no provocando así, un retraso en las órdenes de entrega. Sin embargo, el problema aumenta de dificultad considerablemente ya que las soluciones de éstos no se obtienen de manera sencilla, y rara vez se consigue llegar a alcanzar el óptimo del problema en un tiempo razonable. Este problema es denominado *NP-hard* (no polinomial). Este tipo de entorno es cada vez más común en los sistemas de fabricación en los que los productos permiten ser elaborados en distintas fábricas y después ensamblados en uno final.

1.3. Sumario

Este documento consta de 5 capítulos (contando con la introducción) en los cuáles se comenta cada parte del problema a abordar:

- Capítulo 1. Introducción. Se presenta el objeto del trabajo de estudio, su justificación y el sumario.
- Capítulo 2. Descripción del problema. Se comentan el tipo de problema que se tiene que abordar (entorno, restricciones y objetivo) y los conocimientos previos para poder hacerlo.
- Capítulo 3. Descripción de la metodología. Se describen de forma detallada los algoritmos HBBO y HBI, junto con los pseudocódigos de mayor importancia, para su posterior comparación.
- Capítulo 4. Aplicación de la metodología. Se presentan los resultados obtenidos de los dos algoritmos con la ayuda de gráficas y tablas para facilitar su visualización.
- Capítulo 5. Conclusiones. Repaso del trabajo realizado durante todo el documento y explicación de los resultados obtenidos llegando así a una conclusión final.

2 DESCRIPCIÓN DEL PROBLEMA

En este apartado, como se ha comentado anteriormente, se realizará la descripción del problema a abordar en este documento. Se introduce el tipo de entorno de estudio, el tipo de restricción que presenta el problema y el objetivo que se pretende evaluar, además de una serie de conocimientos básicos que se deben tener de programación de la producción. Con esto, se quiere dar a conocer el problema de estudio, para posteriormente detallar los algoritmos utilizados para su resolución.

En la elaboración de este documento han sido necesarios los conocimientos de la asignatura Programación de Operaciones perteneciente a la rama de Organización y Gestión de Empresa impartida en el Grado en Ingeniería de las Tecnologías Industriales de la Escuela Superior de Ingeniería.

Antes de comenzar, es importante conocer una serie de suposiciones generales que servirán durante todo el documento como se indica en Pérez (2017) que son las siguientes:

- Todos los trabajos están disponibles al principio del horizonte de programación.
- Los trabajos no se pueden interrumpir, esto es que desde que un trabajo en una máquina empieza, debe de seguir hasta que acabe.
- Las máquinas están siempre disponibles para su utilización. No se producen averías, tiempos de preparación de las máquinas ni paradas por mantenimiento preventivo.
- Cada máquina puede hacer un trabajo, y un trabajo puede ser realizado sólo en una máquina. Un mismo trabajo no puede ser realizado en dos máquinas a la vez.
- El buffer entre máquinas se supone infinito.
- El tiempo de transporte de un trabajo de una máquina a otra se supone despreciable, no influyendo en el resultado final.
- Los tiempos de preparación (set up) son independientes de la secuencia obtenida y son incluidos en los tiempos de proceso de cada trabajo o bien son ignorados.

2.1. Conocimientos previos

En este apartado se realizará la descripción del entorno del problema, la restricción y la función objetivo que se debe evaluar.

Los diferentes tipos de entornos vienen denominados por la letra griega α como se indica en Pérez (2017). Hay varios tipos como son:

- *Single Machine* (α=1). Indica que es un entorno con una sola máquina.
- *Parallel Machine* ($\alpha=P_m$). Está formado por una serie de máquinas de forma paralela por las cuáles se puede procesar cualquier trabajo.
- *Flow Shop* ($\alpha=F_m$). Es el entorno del problema (se explica posteriormente).
- *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.

- Híbridos ($\alpha=H_m$). Está formado por varios entornos a la vez.

Los tipos de restricciones se denominan por la letra griega β según señala Pérez (2017) y son las siguientes:

- Secuencia de permutación (β=prmu). Es la restricción del problema (se explica a continuación).
- Fechas de entrega y fechas de llegadas ($\beta=r_j$ $\beta=d_j$). 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) ($\beta=s_{ij}$). 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-ilde). 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.

Los tipos de objetivos se designan por la letra griega γ según se detalla en Pérez (2017) y existen los siguientes:

- Tiempo de terminación del trabajo (*completion time)* $\gamma=\min(\max C_j)$. Minimiza la terminación de los trabajos. Es el objetivo a evaluar del problema.
- Tiempo de flujo del trabajo *(flowtime)* $\gamma=\min(\max F_j)$. 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)* $\gamma=\min(\max L_j)$. 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)* $\gamma=\min(\max T_j)$. 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)* $\gamma=\min(\max E_j)$. El adelanto de los trabajos se define como: $E_j=\max\{0, -L_j\}=\max\{0,-(C_j-d_j\}$.
- Trabajo tardío *(tardy job)* $\gamma=\min(\max U_j)$. La definición de trabajo tardío es la siguiente: $U_j=1$ 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.

Todos estos objetivos también se pueden medir de forma global para todos los trabajos mediante el sumatorio de los objetivos.

2.1.1 Flow shop $\alpha=F_m$

Está formado por un conjunto de máquinas por las cuales todos los trabajos u operaciones pasan sucesivamente hasta conseguir el producto completo. Cada trabajo visita todas las máquinas y tienen una ruta predeterminada. A través del estudio del orden de secuenciación de los trabajos es posible que se obtenga una mejor solución. Este tipo de entorno es el más habitual en los sistemas de fabricación.

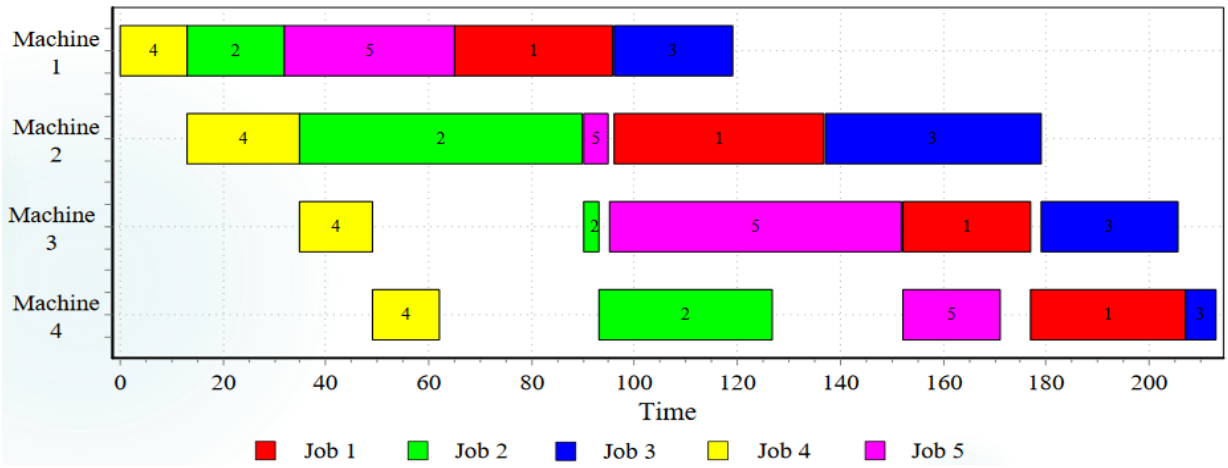

Figura 2-1. Diagrama de Gantt de un entorno Flow-Shop. [Fuente: (Perez, 2017)]

2.1.2. Secuencia con permutación β=prmu

Permutación implica que solamente hay una secuencia para todas las máquinas del problema. Esto es, los trabajos seguirán el mismo orden de procesamiento por cada máquina.

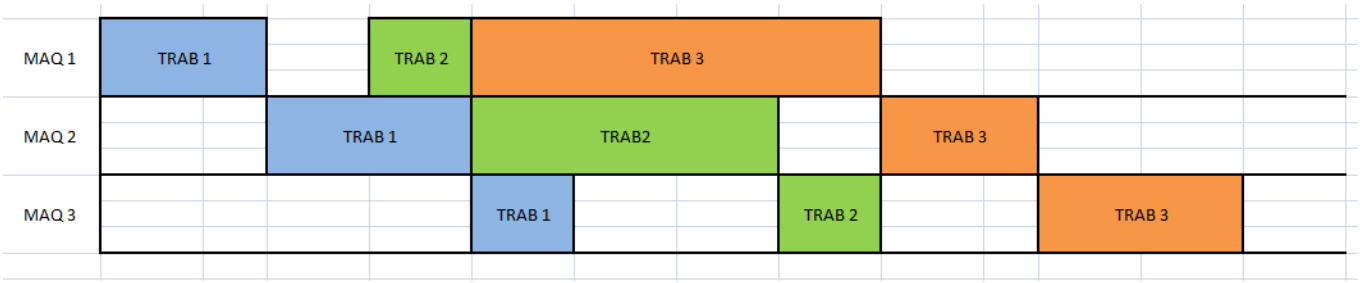

Figura 2-2. Secuencia con permutación. [Fuente: (Elaboración propia)]

2.1.3. Tiempo de terminación del trabajo (*completion time*) γ=min (max C_i)

Para este tipo de objetivos lo que se quiere es minimizar la terminación de los trabajos que se realizan para la elaboración de productos. 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. Este es el objetivo a analizar en este problema. Se le puede denominar también makespan.

2.2. Problema tipo flow-shop distribuido con permutación

Los problemas tipo flow-shop de permutación juegan un papel muy importante en los sistemas de fabricación industriales dado que muchas industrias de fabricación utilizan este tipo de entornos con esta restricción, en los cuáles un producto va pasando por máquinas en serie realizándose un número fijo de operaciones para obtener el producto final. Para un número de máquinas mayor a 3, estos tipos de problemas resultan ser muy complejos de solucionar, convirtiendo el problema en NP-hard. En ellos, la obtención de una solución factible (ya que óptima es difícil de conseguir), no se obtiene en un tiempo razonable y suponen una gran dificultad. Es más común pensar que exista sólo un centro de fabricación en la elaboración de un producto, y todos los trabajos a realizar para ese producto, se confeccionen dentro de la fábrica. Pero se ha comprobado en Chan, Chung y Chan (2005) que en la producción mediante más de una fábrica, los productos elaborados tienen mayor calidad, además de reducir considerablemente el nivel de riesgo, (ya que si se produce un fallo en una fábrica, las demás pueden seguir produciendo) y los costes de producción.

Este tipo de problema consta de dos etapas: producción y montaje, de los cuales, este documento sólo se centra en la parte de producción. Además se puede subdividir en tres subproblemas: programación de tareas, programación y asignación de los trabajos a las fábricas. Todas las fábrica son capaces de confeccionar cualquier trabajo y cada fábrica es un flow-shop de permutación con m máquinas. La segunda etapa (que en este documento no se tiene en cuenta, aunque sí se nombra) es una única máquina de montaje que reúne los trabajos de todas las demás máquinas y los ensamblan, elaborándose así el producto final. Cada producto está formado por un número de trabajos, y éstos deben ser procesados en las fábricas antes de ensamblarse unos con otros. En este problema, el tiempo máximo de terminación (makespan) en las fábricas, es el objetivo a minimizar.

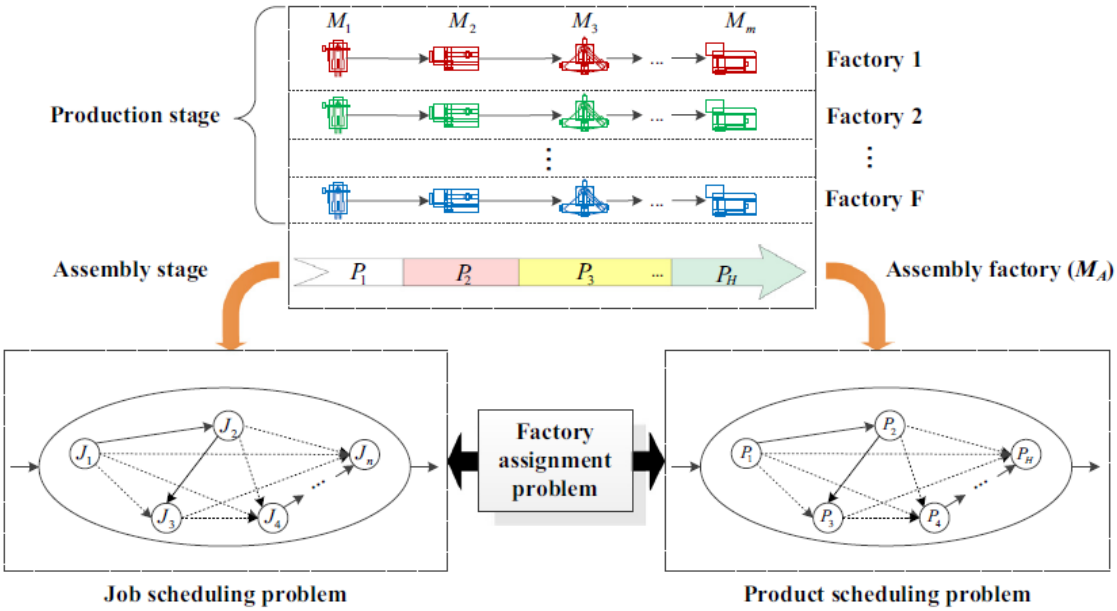

Figura 2-3. Flow-shop distribuido con permutación. [Fuente: (Jian & Shuai, 2017]

Desde el punto de vista de un gerente, la programación de sistemas distribuidos es más compleja que los problemas de programación de una sola fábrica. En los problemas de una fábrica, el único objetivo es encontrar una programación de trabajos para un conjunto de máquinas, mientras que un problema distribuido, la asignación de los trabajos a las fábricas adecuadas, supone una importante decisión adicional. Por tanto, dos decisiones importantes tienen que ser tomadas: asignación de los trabajos a las fábricas y planificación de las tareas en cada fábrica. Diferentes asignaciones de trabajos a distintas fábricas dan lugar a diferentes programas de producción lo que consecuentemente afecta a la cadena de suministros (Chan, Chung & Chan, 2005).

Para simplificar el problema se supone que todas las máquinas tienen la misma permutación de trabajos. En otras palabras, si un trabajo está en la posición j en la máquina 1, entonces, ese mismo trabajo tiene que estar en la posición j para todas las demás máquinas.

En cuanto al problema de ensamblado, Lee, Cheng y Lin (1993) propone tres tipos de ensamblado para tres máquinas en un entorno flowshop considerando la minimización del makespan como función objetivo. En su modelo considerado, cada producto se compone de dos tipos de puestos de trabajos, dónde el tipo a y el tipo b son procesados por las máquinas M_a y M_b, respectivamente, y la máquinas M_2, ensambla los dos trabajos en un producto. Estos autores también presentan un procedimiento de solución aproximada. Poco después, Potts (1995) amplió este modelo considerando m máquinas de producción paralelas en lugar de las dos primeras máquinas de producción. Además, Tozkapan (2003) consideró dos etapas para el problema de ensamblado. También presentaron un procedimiento heurístico para hallar un límite superior inicial. Por otra parte, Al-Azin y Allahverdi (2006) trataron el modelo de Tozkapan (2003) utilizando metaheurísticas para resolver su modelo y propusieron el Recocido Simulado (SA), la Búsqueda Tabú (TS) y el Tabú Híbrido de búsqueda heurística para casos generales.

2.3. Recapitulación del apartado 2

Aspectos fundamentales tratados en este capítulo:

- Se exponen unas suposiciones iniciales que son necesarias para la comparación de los algoritmos.
- Se describen conocimientos básicos adicionales de Programación de Operaciones.
- Se detallan el tipo de entorno del problema de estudio, la restricción y el objetivo a evaluar.
- Se especifica el tipo de problema a resolver (flow-shop distribuido con permutación) y las contribuciones anteriores.

A continuación se realizará la descripción de los algoritmos a analizar paso por paso, destacando los aspectos fundamentales.

3 DESCRIPCIÓN DE LA METODOLOGÍA

Para resolver el tipo de problema detallado en el apartado 2.2, se pueden emplear heurísticas, que encuentran una solución factible al problema de manera rápida.

Distintos algoritmos evolutivos se han propuesto en los últimos años para encontrar la mejor solución a estos tipos de problemas, como por ejemplo, la optimización en enjambre de partículas desarrollado por Tasgetiren, Liang, Sevkli y Gencyilmaz (2007) que utiliza la analogía de los movimientos de las partículas en la búsqueda de la mejor solución mediante reglas matemáticas que tienen en cuenta la posición y la velocidad de las partículas. Este movimiento de las partículas se ve influido por la posición en la que se encuentren y el objetivo es que la nube de partículas converja rápidamente hacia las mejores soluciones. Otro algoritmo que intenta encontrar una solución a este tipo de problema es el llamado algoritmo colonias de abejas, que ha sido realizado por Tasgetiren, Pan, Suganthan y Chen (2011), basado en el comportamiento inteligente de los enjambres de abejas en busca de la miel. Las abejas exploradoras comunican la información de dónde se encuentra la fuente de alimento por medio de una danza, las danzas con mayor duración indican fuentes de alimento más rentables. Una vez esté agotado el alimento, se abandona y se comienza la búsqueda de otra fuente. Cada posible zona de alimentación representa una solución del problema.

Se han desarrollado también algoritmos de evolución diferencial según se indica en Pan, Tasgetiren y Lian (2008), los cuáles mantiene una población de soluciones candidatas y las van mejorando mediante búsquedas locales. Sin embargo muy pocos se han aplicado a problemas distribuidos. Muy recientemente Jian y Shuai (2016) ha elaborado un algoritmo basado en la biogeografía para encontrar la mejor solución.

Toma gran relevancia en la elaboración de algoritmos evolutivos la selección de la población a analizar, el cruce de especies que se pueden producir, la mutación entre especies y la selección de una especie para ser mutada o cruzada con otra.

En el cruce de especies se lleva a cabo la obtención de una descendencia, cortando por dos partes al azar, que al unirse dan lugar a una nueva especie, cuyo valor de la función objetivo puede ser mejor. Hay distintos tipos de cruces que van desde el cruce de especie en un punto hasta el cruce de especies en n puntos.

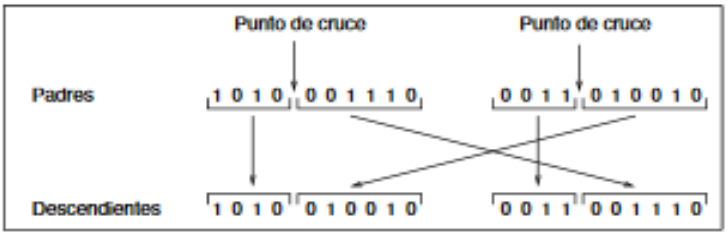

Figura 3-1. Cruce de especies en un punto. [Fuente: (Berzal, s.f)]

En cuanto al tamaño de la población parece intuitivo pensar que para un tamaño pequeño puede provocar que no cubra el espacio de búsqueda y para un tamaño grande puede provocar problemas a la hora de hacer cálculos computacionales. Según Goldberg (1989), el tamaño de la población ideal es l, pero se puede comprobar que con este tamaño de población, los algoritmos evolutivos no resultarían competitivos respecto a otros métodos de optimización. Más tarde, Alander (1992) comprobó que el tamaño óptimo está compendido entre l y 2l, siendo suficiente para obtener soluciones con éxito.

Para escoger la población inicial del método, se utilizan secuencias formuladas al azar. También se podría utilizar una técnica heurística para obtener una población inicial pudiendo acelerar la convergencia del algoritmo. Sin embargo, en muchas ocasiones, sólo se consigue la convergencia hacia óptimos locales.

La selección se realiza mediante la elección de dos especies para obtener la descendencia. Si las especies seleccionadas son buenas, se obtendrá una descendencia mejor. Se suelen utilizar mecanismos de selección estocásticos, teniendo los mejores individuos más probabilidad de ser seleccionados. Este tipo de selección se utiliza para escapar de óptimos locales.

La mutación se considera un operador básico en los algoritmos evolutivos y es la encargada de producir variaciones de modo aleatorio en especies. Mediante la mutación se obtiene mayor diversidad de especies, dando lugar a una búsqueda más exhaustiva del óptimo. Es conveniente ir modificando la probabilidad de mutación a medida que se aumenta el número de iteraciones con el fin de conseguir mejores resultados.

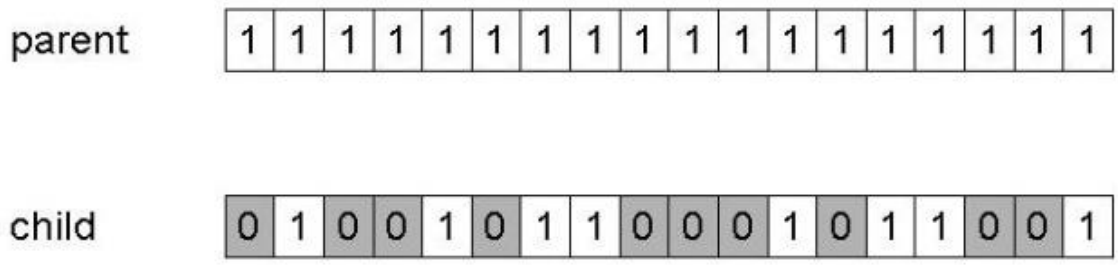

Figura 3-2. Mutación estándar. [Fuente: (Berzal, s.f)]

3.1 Características del Algoritmo Híbrido basado en la Biogeografía (HBBO)

Este algoritmo integra varias heurísticas para resolver problemas de flow-shop de permutación distribuidos, con el objetivo de minimizar el makespan. En primer lugar se utiliza una heurística en la fase de migración como estrategia de búsqueda local de productos para optimizar la secuencia de montaje. En segundo lugar se utiliza una heurística en la fase de mutación para determinar la permutación de trabajo para cada producto. Luego se utiliza un nuevo método de búsqueda local para mejorar todavía más al individuo más prometedor y así tener una mejor solución. En este documento no se tiene en cuenta la etapa de ensamblado de trabajos en productos, centrándose solamente en la elaboración de los trabajos.

3.1.1. Optimización basada en la biogeografía

Este algoritmo fue propuesto por Simon (2008) y ha dado un buen rendimiento en comparación con otros algoritmos evolutivos para resolver el problema comentado en el apartado 2.2. Está basado en la población, cada solución del problema se le denomina hábitat y la calidad del hábitat se mide mediante el índice de idoneidad. El algoritmo utiliza dos operadores principales que son la migración y la mutación:

El operador de migración se utiliza para escoger el mejor hábitat de las existentes. Está formado por dos ratios: el ratio de inmigración λ y el ratio de emigración μ, y cada hábitat tiene el suyo propio.

$$\begin{cases} \lambda_r = I(1 - \dfrac{r}{S_{max}}) \\ \mu_r = \dfrac{E * r}{S_{max}} \end{cases} \quad (1)$$

Donde r es el número de especies y S_{max} es el máximo posible de especies que pueden ser soportadas por el hábitat. Además $I \in [0,1]$ y $E \in [0,1]$ son las tasas máximas de los ratios de inmigración y emigración respectivamente que están comprendidas entre 0 y 1. El ratio de inmigración λ es el encargado de determinar si un hábitat se debe modificar o no, mientras que el ratio de emigración μ determina cuáles de los hábitats migrará una dimensión aleatoria al hábitat seleccionado por λ.

El operador de mutación es el encargado de modificar una propiedad al azar del hábitat seleccionada basándose en la probabilidad de mutación. Mediante este operador se pueden mejorar las características de los hábitats. La probabilidad de mutación está definida de la siguiente manera:

$$m_r = m_{max}(1 - \frac{P_r}{P_{max}}) \quad (2)$$

Donde m_{max} es el ratio de la máxima probabilidad de mutación y dato del problema y P_{max}=arg max P_i, i=1,…, n siendo n el tamaño de la población. El valor de m_{max} es dato e irá variando entre {0.01, 0.1, 0.3, 0.5}. Además P_r es el recuento de especies con probabilidad del hábitat con r especies en el tiempo t que está definido de la siguiente manera:

$$P_r(t) = \begin{cases} -(\lambda_r + \mu_r)P_r(t-1) + \mu_{r+1}P_{r+1}(t-1), & r = 0 \\ -(\lambda_r + \mu_r)P_r(t-1) + \lambda_{r-1}P_{r-1}(t-1) + \mu_{r+1}P_{r+1}(t-1), & 1 \le r \le S_{max} - 1 \\ -(\lambda_r + \mu_r)P_r(t-1) + \lambda_{r-1}P_{r-1}(t-1), & r = S_{max} \end{cases} \quad (3)$$

3.1.2. Descripción del Algoritmo Híbrido basado en la Biogeografía (HBBO)

El algoritmo a analizar en este documento está basado en el algoritmo de Simon (2008) denominado HBBO (*hybrid biogeography-based optimization)* pero empleando una optimización híbrida. Para facilitar su entendimiento, se ilustrará mediante una figura los pasos a seguir en la realización del algoritmo. Además se procederá a la explicación de cada paso para la resolución de cualquier duda existente.

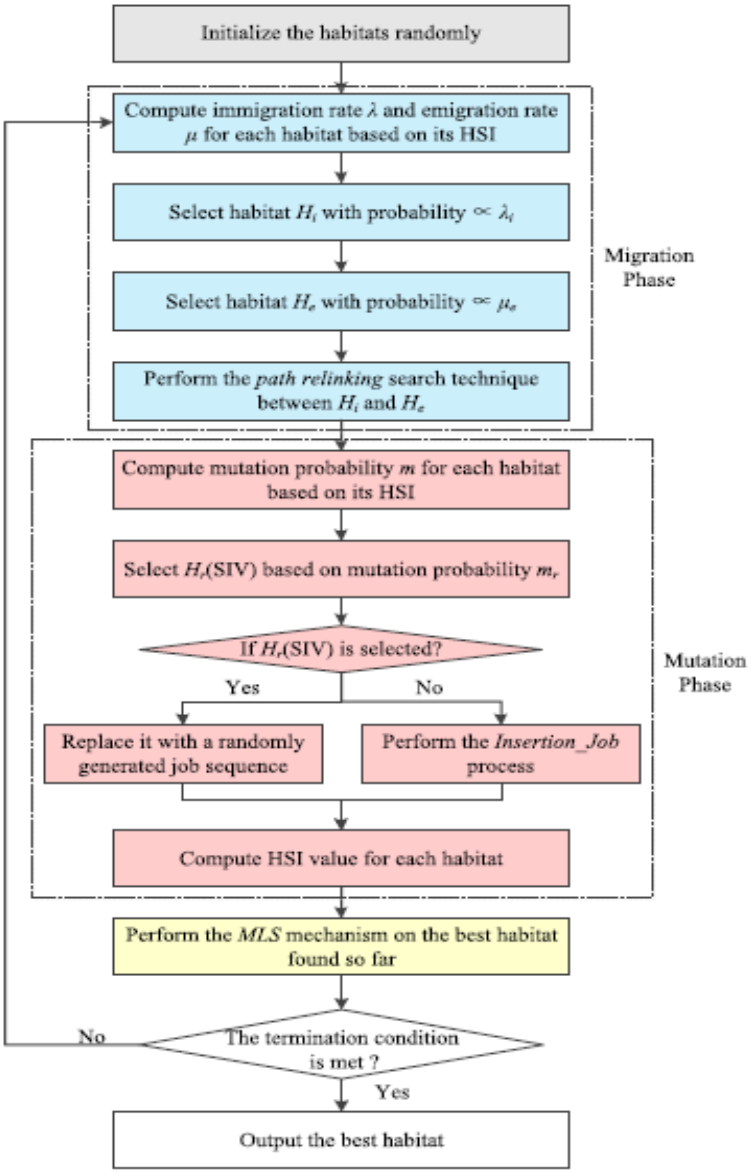

Figura 3-3. Descripción del Algoritmo. [Fuente: (Jian & Shuai, 2016)]

En la figura 3-3 se muestran las etapas a seguir en el algoritmo. Se pueden observar dos partes claramente diferenciadas: la fase de migración, de dónde se obtendrá un hábitat y la fase de mutación, de la cual se obtendrá otro. Con estos dos hábitats, se realizará un método de búsqueda local para adquirir una mejor secuencia. Este algoritmo se ejecutará un número de iteraciones máxima hasta obtener el mejor hábitat de todos.

La fase de migración consta de los siguientes pasos:

- Paso 1. Inicializar los hábitats aleatoriamente. Dado el número de hábitats y el número de trabajos de cada hábitat, podemos calcular todos los hábitats. Se utilizan hábitats aleatorios para que cuando se vuelva a ejecutar, éstos sean totalmente diferentes a los anteriores, teniendo así más diversidad y poder conseguir un óptimo mejor. Además también se inicializa la asignación de cada trabajo del hábitat con la fábrica correspondiente.

- Paso 2. Realizar el cálculo de los ratios de inmigración λ y emigración μ para cada hábitat mediante las ecuaciones descritas en el apartado 3.1.1.

- Paso 3. Seleccionar el hábitat H_i con probabilidad proporcional a λ_i. Con esto, se elige el mejor hábitat según el ratio de inmigración y se desechan los demás.

- Paso 4. Seleccionar el hábitat H_e con probabilidad proporcional a μ_e. De la misma manera que el paso 3, se escoge el hábitat según el ratio de emigración y se desechan las demás.

- Paso 5. Se realiza el método llamado *path relinking* entre los hábitats H_i y H_e. Este método es una búsqueda local en el que se obtiene un hábitat intermedia de las anteriores, la cual, el valor de su función objetivo es el menor de todos los obtenidos.

A partir de este paso comienza la fase de mutación, en la cual se conseguirá otro hábitat que más tarde habrá que utilizar junto con el hábitat de la fase de migración para obtener uno mejor.

- Paso 6. Calcular la probabilidad de mutación m para cada hábitat como está descrito en el apartado 3.1.1.

- Paso 7. Seleccionar el hábitat H_r con probabilidad proporcional a m_r. Se realiza de la misma manera que el paso 2 y 3 pero con la probabilidad de mutación, obteniendo así, un nuevo hábitat.

- Paso 8. Efectuar un recorrido por todos los hábitats iniciales, en el cual se diferencian entre los hábitats no seleccionados y el seleccionado en el Paso 7. Para los no seleccionados, se ejecutará un método llamado *Insertion_Job*, que permitirá sacar un hábitat nuevo mediante el intercambio de posiciones de los trabajos en cada hábitat. Para el hábitat seleccionado, se reemplazará éste por una secuencia aleatoria de trabajos. Con esto se quiere aumentar la diversidad de hábitats.

- Paso 9. Evaluar el valor de la función objetivo (el cálculo del C_{max}) de todos los hábitats conseguidos en el paso 8.

- Paso 10. Realizar el método llamado *MLS (Modified Local Search)* entre la secuencia obtenida en la fase de migración y la obtenida en la fase de mutación. Este método es otra búsqueda local para obtener una nueva secuencia intermedia.

- Paso 11. Todos los pasos anteriores se hacen un número de iteraciones máximas y entre todas esas iteraciones se elige el hábitat con menor valor del C_{max}. Este hábitat será el mejor de todos y con esto se pondrá fin al algoritmo.

Para facilitar la comprensión del algoritmo HBBO se expondrá un ejemplo de la resolución dado el número de máquinas, trabajos, fábricas y la asignación de los trabajos a cada fábrica junto con el diagrama de Gantt adecuado.

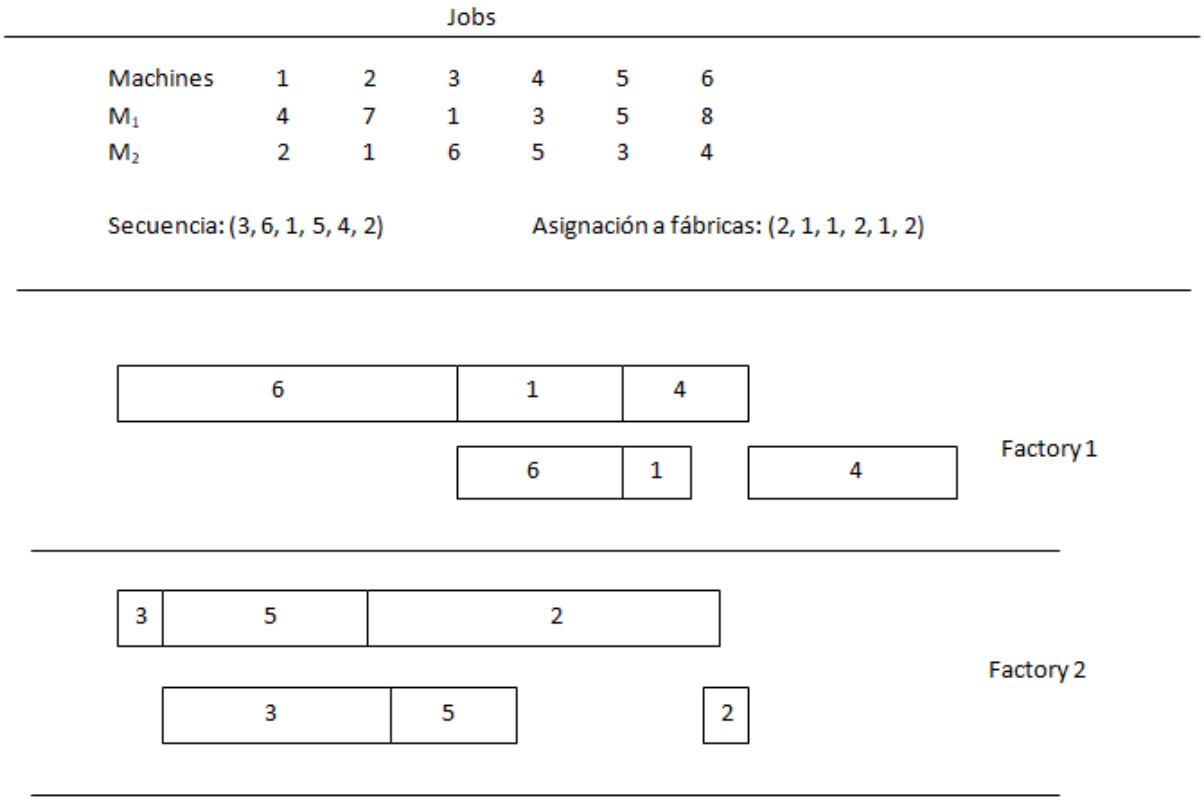

Figura 3-4. Ejemplo del HBBO [Fuente: (Elaboración propia)]

3.2. Pseudocódigos utilizados en el Algoritmo HBBO

3.2.1. Pseudocódigo del Path Relinking

En este apartado se describe el pseudocódigo perteneciente al método Path Relinking utilizado en el paso 5 en la fase de migración. Las variables de entradas a la función son los hábitats H_i y H_e. Esta función es considerada una de las principales en la elaboración del algoritmo HBBO. Se basa en una búsqueda local, mediante el intercambio de trabajos en las posiciones de una secuencia en función de otra secuencia. Se van obteniendo por cada iteración secuencias distintas, que finalmente se comparan las funciones objetivos de éstas con la inicial, escogiendo la mejor de ellas.

1. Declarar variables.
2. Realizar un bucle desde i=1 hasta n que recorra todos los trabajos de la secuencia Hi y He.
   2.1. Encontrar la posición s del trabajo $H_i(i)$ en H_e.
   2.2. Intercambiar la posición de los trabajos $H_i(i)$ y $H_i(s)$ para generar un hábitat intermedio H_i'.
3. Encontrar el mejor hábitat intermedio H_{best}' con el valor del makespan más bajo.
4. Si el C_{max} de H_{best}' es menor al C_{max} de H_i entonces:
   4.1. Hacer que H_i sea ahora H_{best}'.
5. Mostrar resultados.
6. Liberar memoria.

Figura 3-5. Pseudocódigo de Path Relinking. [Fuente: (Jian & Shuai, 2016)]

3.2.2. Pseudocódigo del Insertion_job

En esta sección se representa el pseudocódigo referente al método Insertion_Job que se necesita en el paso 8 en la etapa de mutación. La variable de entrada es la secuencia Π (es la secuencia a la que se le quiere realizar la inserción de trabajos).

1. Declarar variables.
2. Realizar un bucle desde i=1 hasta n que recorra todos los trabajos de la secuencia.
   2.1. Eliminar el trabajo Π (s) de la secuencia Π.
   2.2. Encontrar la mejor secuencia de trabajos Π_b insertando el trabajo Π (s) en todas las posiciones de Π.
   2.3. Si el C_{max} de Π_b es menor al C_{max} de Π entonces:
      2.3.1. Hacer que Π sea ahora Π_b.
3. Mostrar resultados.
4. Liberar memoria.

Figura 3-6. Pseudocódigo del Insertion_Job. [Fuente: (Jian & Shuai, 2016)]

3.2.3. Pseudocódigo del MLS

En este capítulo se describe el pseudocódigo perteneciente al método MLS que se emplea en el paso 10. Es importante comentar que este pseudocódigo es diferente al del artículo Jian y Shuai (2016), dado que se ha cambiado parte de éste porque estaba mal desarrollado. Las variables de entrada a la función son las secuencias obtenidas de la fase de migración y de la fase de mutación que denominaremos como ω y ω*.

1. Declarar variables.
2. Hacer que t sea igual a 1.
3. Hacer que Π sea igual a ω.
4. Mientras t sea menor que n:
   4.1. Ejecutar una variable aleatoria entre 0 y 1 la que denominaremos número.
   4.2. Ralizar un buce desde i=1 hasta n que haga recorrer todos los trabajos:
      4.2.1. Eliminar el trabajo ω*(número+1) de la secuencia Π.
      4.2.2. Encontrar la mejor secuencia Π_b insertando el trabajo ω*(número+1) en todas las posiciones de Π.
      4.2.3. Si el C_{max} de Π_b es menor al C_{max} de Π entonces:
         4.2.3.1. Realizar el método Insertion_Job en la secuencia Π para encontrar la nueva mejor solución $\tilde{\Pi}_b$.
         4.2.3.2. Hacer que Π sea igual a $\tilde{\Pi}_b$.
         4.2.3.3. Igualar t a 1.
      4.2.4. Sino:
         4.2.4.1. Hacer que t sea igual a t+1.
5. Si el C_{max} de la secuencia Π es menor al C_{max} de la secuencia ω entonces:
   5.1. Hacer que ω=Π.
6. Mostrar resultados.
7. Liberar memoria.

Figura 3-7.Pseudocódigo del MLS. [Fuente: (Jian & Shuai, 2016)]

3.3. Descripción de la Heurística basada en la inserción (HBI)

"Este problema es NP-completo (modelos con algoritmos no polinomiales). Por tanto es necesario un enfoque heurístico para poder obtener una solución, al menos factible de ést e (Naderi & Ruiz, 2010)".

La heurística HBI a aplicar es una modificación de la heurística 1 que aparece en el artículo de Hatami (2013), en la cual se ha obviado el ensamblado de los trabajos en productos. Sólo se tiene en cuenta la parte de producción, con el fin de tener una buena comparación con el algoritmo HBBO previamente descrito.

- Paso 1.Se obtiene una secuencia aleatoria de trabajos, para así empezar con ella a trabajar.
- Paso 2. Se aplica la regla SPT a esa secuencia de trabajos, los cuáles se ordenan de menor a mayor tiempo de proceso, para obtener una buena secuencia inicial de trabajos.
- Paso 3. El primer trabajo de esa secuencia se sitúa en la primera posición de una nueva secuencia final y el siguiente trabajo se inserta en las posibles posiciones de esa nueva secuencia.
- Paso 4. Se escoge la secuencia con menor valor del C_{max} y se inserta el siguiente trabajo de la misma manera.
- Paso 5. Una vez insertado el último trabajo en las diferentes posiciones, y escogida la mejor secuencia, se procede al cálculo del C_{max}.

Para la asignación de los puestos de trabajos a cada fábrica, se utiliza la regla de asignación de Naderi y Ruiz (2010), que es necesaria para la evaluación del makespan del problema y es la siguiente:

- (R1) Asignar el trabajo j a la fábrica que tiene el valor más bajo de C_{max}.
- (R2) Asignar el trabajo j a la fábrica que tiene el valor más bajo de C_{max} después de incluir el trabajo j.

5	3	9	8	6	1

3	5	9	8	6	1
5	3	9	8	6	1
5	9	3	8	6	1
5	9	8	3	6	1
5	9	8	6	3	1
5	9	8	6	1	3

Figura 3-8. Inserción del trabajo 3 en las diferentes posiciones. [Fuente: (Elaboración propia)]

Para entender mejor el proceso que se debe realizar en HBI para comparar con el algoritmo HBBO propuesto, se ilustrará un ejemplo de resolución dado los tiempos de proceso y la realización del diagrama de Gantt correspondiente.

	Jobs								
Machines	1	2	3	4	5	6	7	8	9
M_1	1	5	7	9	9	3	8	4	2
M_2	3	8	5	7	3	4	1	3	5

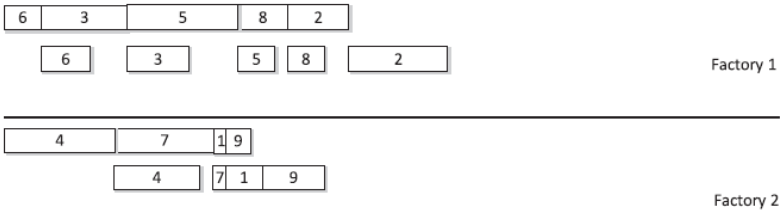

Figura 3-9. Ejemplo del algoritmo HBI. [Fuente: (Hatami, 2013)]

3.4. Recapitulación del apartado 3

En este capítulo se ha detallado la siguiente información:

- Se dan a conocer los aspectos más importantes en un algoritmo evolutivo mediante una breve explicación.
- Se han comentado las características del algoritmo híbrido basado en la biogeografía (HBBO). Posteriormente se ha realizado la descripción detallada del algoritmo, explicando los pasos a seguir para su elaboración. Además se ha realizado un ejemplo mediante un diagrama de Gantt.
- Desarrollo de las funciones principales del algoritmo HBBO mediante pseudocódigo.
- Descripción del algoritmo HBI por pasos y se ha elaborado un diagrama de Gantt para facilitar su entendimiento.

A continuación, una vez elaborados los algoritmos a examinar, se generan las soluciones mediante distintos datos de entradas. Las soluciones obtenidas son comparadas, obteniendo finalmente una conclusión.

4 APLICACIÓN DE LA METODOLOGÍA

En este bloque se aplicará la metodología descrita en el aparatado 4 a los algoritmos HBBO y HBI. Se realizarán una serie de experimentos dónde se obtendrán los valores de la función objetivo y los tiempos de ejecución. El tiempo de ejecución toma gran importancia y es primordial a la hora de escoger un método u otro, ya que en una fábrica siempre es rentable disminuir el tiempo de los cálculos para ahorrar en costes, siempre y cuando se dé una solución admisible y aceptable. Cabe esperar que para instancias mayores con número de trabajos, máquinas y fábricas mayores, los tiempos de ejecución de los algoritmos se incrementarán considerablemente debido a la cantidad de operaciones que se realizan. Nos apoyaremos del Excel para crear las tablas necesarias en la resolución de los métodos. Una vez obtenido todos los resultados, se realizarán las conclusiones. En ella se comentará la validez del algoritmo HBBO frente al HBI, la calidad de los resultados obtenidos y en qué momento se elegiría un método u otro.

Las simulaciones computacionales se ejecutan en dos conjuntos. Cada conjunto se compone de diferentes combinaciones de tres variables (n x m x F). El primer conjunto se compone de 900 instancias y las variables son: n= {8, 12, 16, 20, 24}, m= {2, 3, 4, 5} y F= {2, 3, 4}, siendo 5 el número de iteraciones máximas para este conjunto. El segundo conjunto está compuesto por 810 instancias dónde n= {100, 200, 500}, m= {5, 10, 20} y F= {4, 6, 8} y el número de iteraciones máximas son 10. Se utilizan estos dos conjuntos para poder observar cómo se comporta el algoritmo HBBO en dos situaciones opuestas, además de examinar la evolución de las soluciones.

En el desarrollo de los algoritmos HBBO y HBI es necesario el cálculo de los tiempos de procesos, y toma una gran importancia la manera en cómo se calculan, ya que éstos (además de otros factores como la asignación a las fábricas) son los encargados de dar el valor del C_{max}. Se procederá a crear un archivo aparte, que realizará una escritura en un documento de texto. Este documento se copiará y pegará en las carpetas de ambos métodos que luego se vinculará mediante la función *LoadPTimes_nrows*, que es la encargada de ejecutar el archivo de texto en modo lectura y usar los tiempos de procesos en las operaciones que sean necesarios. El cálculo a proceder en el archivo de los tiempos de procesos se basa según el documento Hatami (2013), en el cuál se realizan los tiempos de procesos para la etapa de producción de manera aleatoria comprendidos entre [1,99].

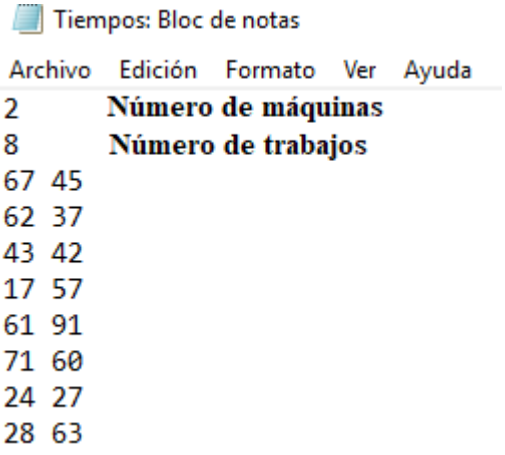

Figura 4-1. Tiempos de procesos aleatorios. [Fuente: (Elaboración propia)]

Para evaluar el algoritmo HBBO se emplea el porcentaje de idoneidad (PI) definido por:

$$PI = \sum_{i=1}^{R} \left(\frac{C_i - C_h}{C_h} \times 100\right) \quad (4)$$

Donde C_h es el valor del makespan obtenido por el algoritmo HBI y C_i es el valor del makespan obtenido del algoritmo HBBO. Una mejor solución del algoritmo HBBO será obtenido si el valor del PI es menor que cero.

4.1. Resultado para instancias de pequeño tamaño.

En este apartado se ejecutarán los algoritmos para un tamaño de 900 instancias, con un máximo de 5 iteraciones para los siguientes datos: n= {8, 12, 16, 20, 24}, m= {2, 3, 4, 5} y F= {2, 3, 4}. El valor de m_{max} irá variando por cada resolución entre los valores {0.01, 0.1, 0.3, 0.5}.

4.1.1. Para dos fábricas

Resultados obtenidos para dos fábricas:

F x m x n	Mmax	HBBO	HBI	PI
2 x 2 x 8	0.01	216	287	**-24,74**
2 x 3 x 8	0.1	308	329	**-6,38**
2 x 4 x 8	0.3	397	430	**-7,67**
2 x 5 x 8	0.5	486	526	**-7,60**
2 x 2 x 12	0.01	389	412	**-5,58**
2 x 3 x 12	0.1	375	410	**-8,54**
2 x 4 x 12	0.3	441	451	**-2,22**
2 x 5 x 12	0.5	621	690	**-10,00**
2 x 2 x 16	0.01	539	558	**-3,41**
2 x 3 x 16	0.1	558	662	**-15,71**
2 x 4 x 16	0.3	643	670	**-4,03**
2 x 5 x 16	0.5	637	651	**-2,15**
2 x 2 x 20	0.01	537	603	**-10,95**
2 x 3 x 20	0.1	702	720	**-2,50**
2 x 4 x 20	0.3	684	733	**-6,68**
2 x 5 x 20	0.5	790	803	**-1,62**
2 x 2 x 24	0.01	623	824	**-24,39**
2 x 3 x 24	0.1	766	827	**-7,38**
2 x 4 x 24	0.3	872	942	**-7,43**
2 x 5 x 24	0.5	931	938	**-0,75**

Tabla 1. Resultados para dos fábricas con instancias de pequeño tamaño.
[Fuente: (Elboración propia)]

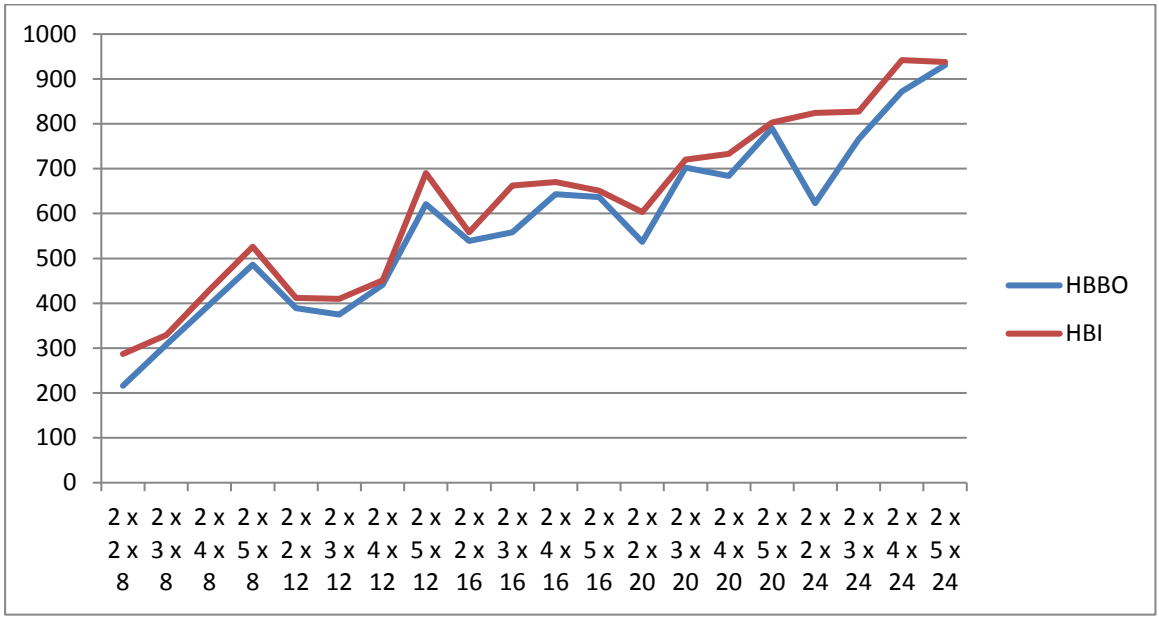

Figura 4-2. Evolución del makespan para dos fábricas. [Fuente: (Elaboración propia)]

Tiempos de ejecución:

F x m x n	Mmax	HBBO (s)	HBI (s)
2 x 2 x 8	0.01	0.313	0.016
2 x 3 x 8	0.1	0.813	0.031
2 x 4 x 8	0.3	0.919	0.031
2 x 5 x 8	0.5	0.797	0.688
2 x 2 x 12	0.01	1.217	0.716
2 x 3 x 12	0.1	0.891	0.703
2 x 4 x 12	0.3	1.281	0.688
2 x 5 x 12	0.5	1.313	0.688
2 x 2 x 16	0.01	1.313	1.172
2 x 3 x 16	0.1	1.344	1.109
2 x 4 x 16	0.3	1.016	1.125
2 x 5 x 16	0.5	1.453	1.141
2 x 2 x 20	0.01	0.938	0.734
2 x 3 x 20	0.1	1.406	1.141
2 x 4 x 20	0.3	3.797	0.703
2 x 5 x 20	0.5	1.781	1.141
2 x 2 x 24	0.01	3.791	1.203
2 x 3 x 24	0.1	1.547	0.719
2 x 4 x 24	0.3	1.703	1.156
2 x 5 x 24	0.5	1.734	1.141

Tabla 2. Tiempos de ejecución para dos fábricas con instancias de pequeño tamaño. [Fuente: (Elab. propia)]

4.1.2. Para tres fábricas

Valores del C_{max}:

F x m x n	Mmax	HBBO	HBI	PI
3 x 2 x 8	0.01	208	220	-5,45
3 x 3 x 8	0.1	266	276	-3,62
3 x 4 x 8	0.3	313	313	0,00
3 x 5 x 8	0.5	344	375	-8,27
3 x 2 x 12	0.01	262	284	-7,75
3 x 3 x 12	0.1	317	345	-8,12
3 x 4 x 12	0.3	479	547	-12,43
3 x 5 x 12	0.5	498	501	-0,60
3 x 2 x 16	0.01	315	355	-11,27
3 x 3 x 16	0.1	422	486	-13,17
3 x 4 x 16	0.3	444	456	-2,63
3 x 5 x 16	0.5	565	620	-8,87
3 x 2 x 20	0.01	420	432	-2,78
3 x 3 x 20	0.1	489	513	-4,68
3 x 4 x 20	0.3	495	502	-1,39
3 x 5 x 20	0.5	557	560	-0,54
3 x 2 x 24	0.01	500	509	-1,77
3 x 3 x 24	0.1	540	542	-0,37
3 x 4 x 24	0.3	614	606	1,32
3 x 5 x 24	0.5	710	715	-0,70

Tabla 3. Resultados para tres fábricas con instancias de pequeño tamaño.
[Fuente: (Elaboración propia)]

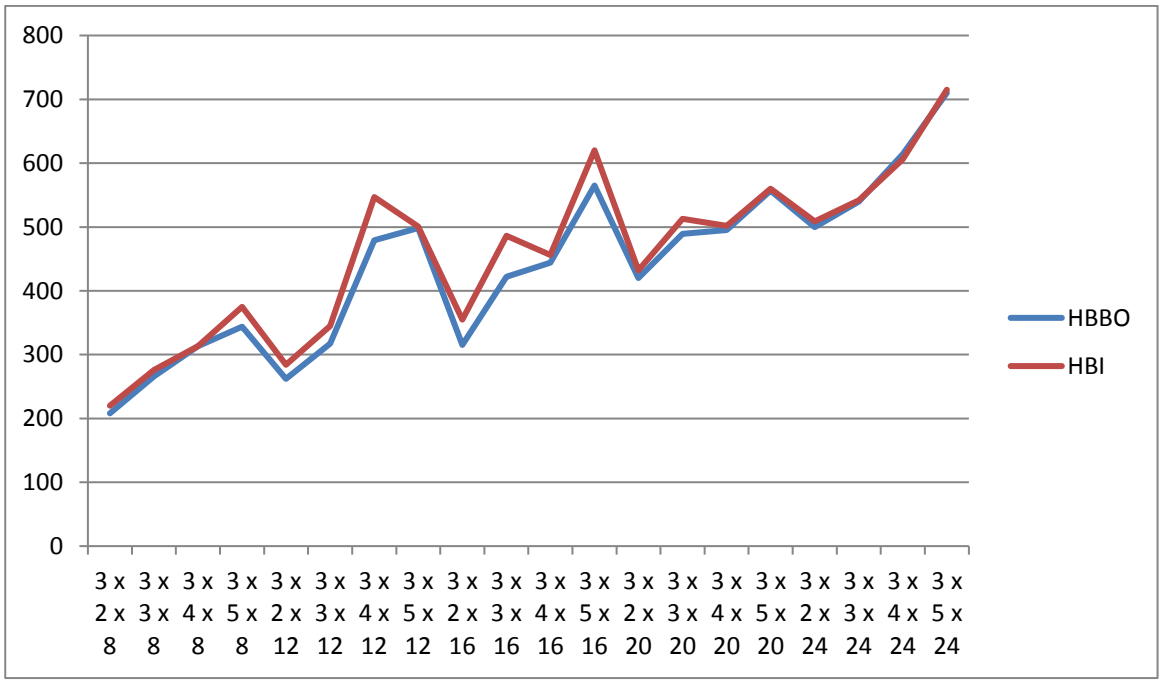

Figura 4-3. Evolución del makespan para tres fábricas. [Fuente: (Elaboración propia)]

Tiempos de ejecución:

F x m x n	Mmax	HBBO (s)	HBI (s)
3 x 2 x 8	0.01	1.060	0.766
3 x 3 x 8	0.1	1.281	1.188
3 x 4 x 8	0.3	1.250	1.281
3 x 5 x 8	0.5	1.250	1.156
3 x 2 x 12	0.01	1.328	1.156
3 x 3 x 12	0.1	0.875	0.766
3 x 4 x 12	0.3	1.578	0.813
3 x 5 x 12	0.5	0.953	1.156
3 x 2 x 16	0.01	1.406	1.203
3 x 3 x 16	0.1	1.422	1.125
3 x 4 x 16	0.3	1.031	0.703
3 x 5 x 16	0.5	1.109	1.031
3 x 2 x 20	0.01	1.101	0.719
3 x 3 x 20	0.1	1.094	1.203
3 x 4 x 20	0.3	0.469	0.047
3 x 5 x 20	0.5	0.625	0.063
3 x 2 x 24	0.01	1.141	0.031
3 x 3 x 24	0.1	1.656	0.047
3 x 4 x 24	0.3	1.734	0.062
3 x 5 x 24	0.5	1.875	0.078

Tabla 4. Tiempos de ejecución para tres fábricas con instancias de pequeño tamaño.
[Fuente: (Elab. propia)]

4.1.3. Para cuatro fábricas

Resultados obtenidos:

F x m x n	Mmax	HBBO	HBI	PI
4 x 2 x 8	0.01	181	256	-29,30
4 x 3 x 8	0.1	212	248	-14,52
4 x 4 x 8	0.3	257	297	-13,47
4 x 5 x 8	0.5	289	322	-10,25
4 x 2 x 12	0.01	223	305	-26,89
4 x 3 x 12	0.1	292	302	-3,31
4 x 4 x 12	0.3	328	363	-9,64
4 x 5 x 12	0.5	378	378	0,00
4 x 2 x 16	0.01	322	370	-12,97
4 x 3 x 16	0.1	356	393	-9,41
4 x 4 x 16	0.3	364	416	-12,50
4 x 5 x 16	0.5	451	500	-9,80
4 x 2 x 20	0.01	308	360	-14,44
4 x 3 x 20	0.1	404	426	-5,16
4 x 4 x 20	0.3	478	530	-9,81
4 x 5 x 20	0.5	606	628	-3,50
4 x 2 x 24	0.01	330	390	-15,38
4 x 3 x 24	0.1	427	464	-7,97
4 x 4 x 24	0.3	561	630	-10,95
4 x 5 x 24	0.5	565	569	-0,70

Tabla 5. Resultado para cuatro fábricas con instancias de pequeño tamaño.
[Fuente: (Elaboración propia)]

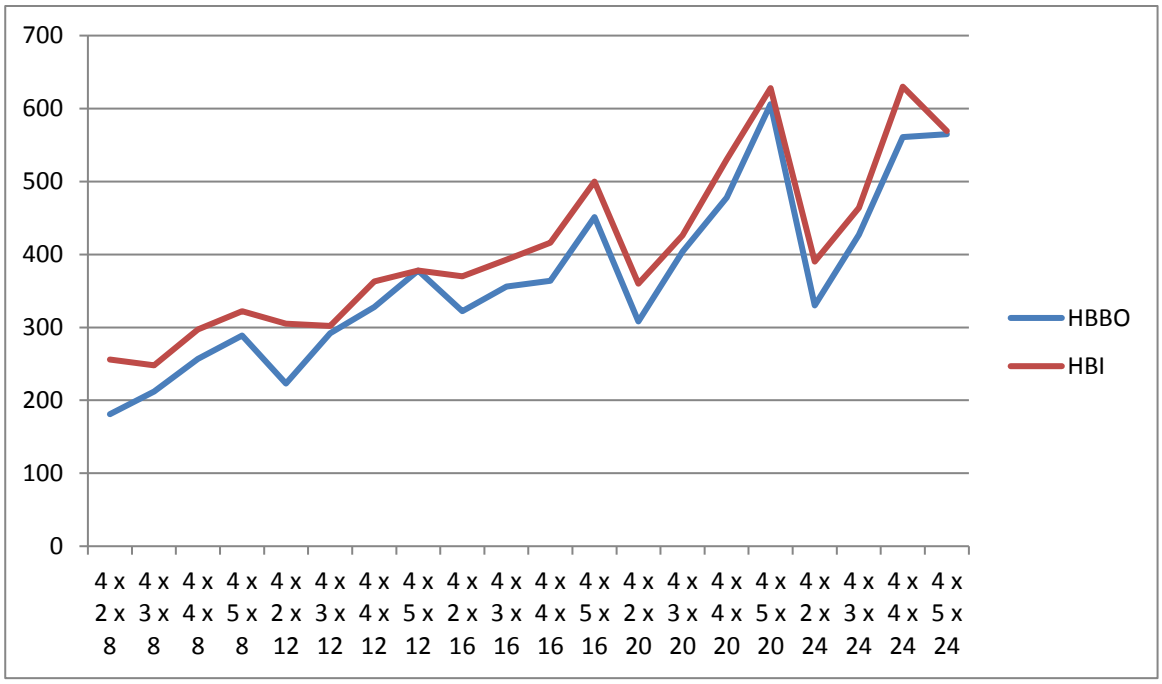

Figura 4-4. Evolución del makespan para cuatro fábricas. [Fuente: (Elaboración propia)]

Tiempos de ejecución:

F x m x n	Mmax	HBBO (s)	HBI (s)
4 x 2 x 8	0.01	1.266	0.038
4 x 3 x 8	0.1	1.297	0.056
4x 4 x 8	0.3	1.252	0.047
4 x 5 x 8	0.5	1.313	0.047
4 x 2 x 12	0.01	1.313	0.031
4 x 3 x 12	0.1	1.328	1.203
4 x 4 x 12	0.3	1.344	1.118
4 x 5 x 12	0.5	1.047	0.750
4 x 2 x 16	0.01	0.953	0.750
4 x 3 x 16	0.1	1.438	1.172
4 x 4 x 16	0.3	1.125	0.719
4 x 5 x 16	0.5	2.422	0.109
4 x 2 x 20	0.01	1.047	1.156
4 x 3 x 20	0.1	1.188	0.791
4 x 4 x 20	0.3	1.641	1.188
4 x 5 x 20	0.5	1.734	1.234
4 x 2 x 24	0.01	1.578	0.703
4 x 3 x 24	0.1	1.281	0.797
4 x 4 x 24	0.3	1.813	0.734
4 x 5 x 24	0.5	2.450	1.204

Tabla 6. Tiempos de ejecución para cuatro fábricas con instancias de pequeño tamaño. [Fuente: (Elab. propia)]

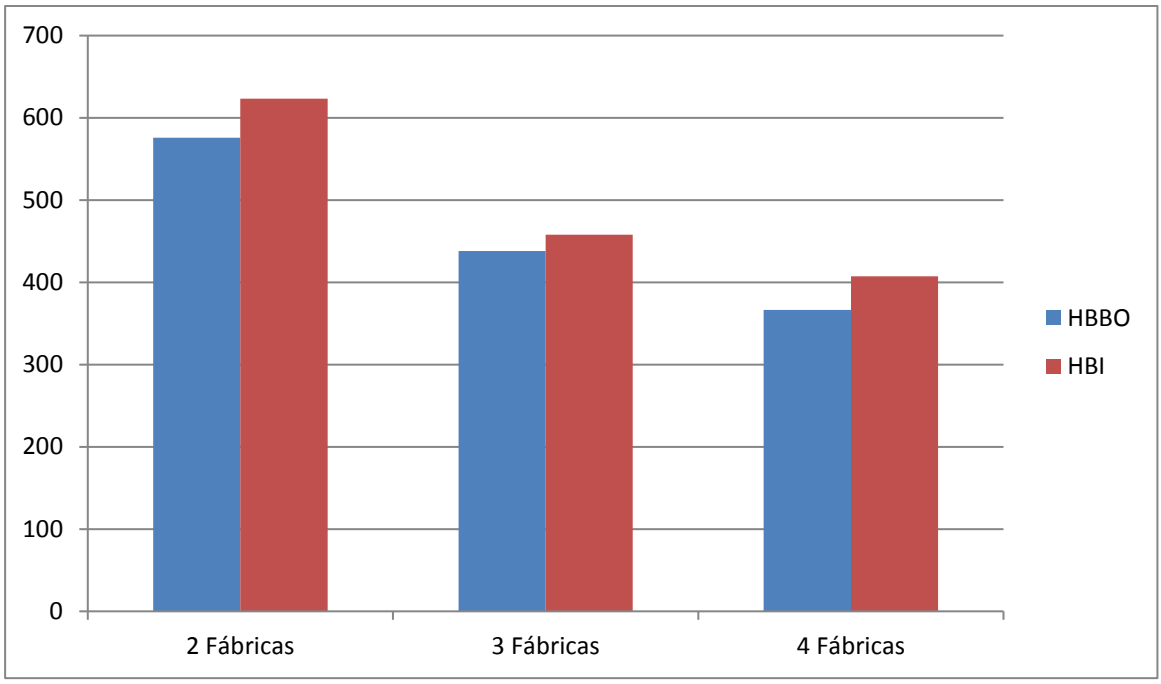

Figura 4-5. Valor promedio del makespan para instancias de pequeño tamaño. [Fuente: (Elab. propia)]

Como se ha podido observar en las Tablas 1, 3 y 5 o en la figura 4-5, los valores de la función objetivo a calcular son menores en el algoritmo HBBO que para el algoritmo HBI, debido a que el algoritmo HBBO realiza una exploración más exhaustiva por medio de varias búsquedas locales y obtiene una mayor variedad de secuencias que comparar para seleccionar la mejor de ellas. Por otra parte, los tiempos de ejecución por parte del algoritmo HBBO son más grandes que los del algoritmo HBI como se indican en las Tablas 2, 4 y 6.

4.2. Resultado para instancias de gran tamaño.

En esta sección, para la computación de los algoritmos se utilizarán los datos siguientes: 810 instancias de gran tamaño dónde el número de trabajos es n= {100, 200, 500}, el número de máquinas m= {5, 10, 20} y el número de fábricas a tener en cuenta son F= {4, 6, 8}. Para esta comprobación se realizará el algoritmo HBBO con un máximo de 10 iteraciones. El valor de m_{max} irá variando entre {0.01, 0.1, 0.3} para cada nivel.

4.2.1. Para cuatro fábricas

Valores del makespan obtenidos:

F x m x n	Mmax	HBBO	HBI	PI
4 x 5 x 100	0.01	1792	1827	-1,92
4 x 10 x 100	0.1	2248	2215	1,49
4 x 20 x 100	0.3	3021	2935	2,93
4 x 5 x 200	0.01	3143	3126	0,54
4 x 10 x 200	0.1	3794	3620	4,81
4 x 20 x 200	0.3	4719	4585	2,92
4 x 5 x 500	0.01	7284	7241	0,59
4 x 10 x 500	0.1	8118	7956	2,04
4 x 20 x 500	0.3	9281	8927	3,97

Tabla 7. Valores del makespan para cuatro fábricas con instancias de gran tamaño. [Fuente: (Elab. propia)]

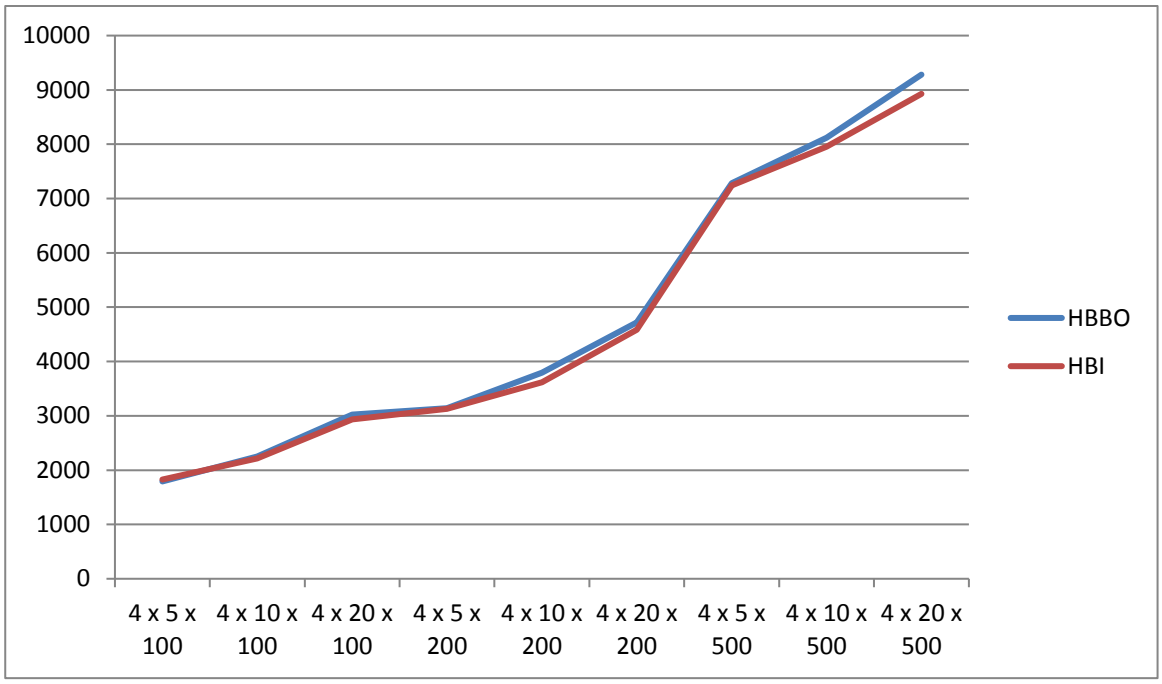

Figura 4-6.Evolución del makespan para cuatro fábricas. [Fuente: (Elaboración propia)]

Tiempos de ejecución:

F x m x n	Mmax	HBBO (s)	HBI (s)
4 x 5 x 100	0.01	22.452	1.234
4 x 10 x 100	0.1	41.302	1.266
4 x 20 x 100	0.3	73.372	2.594
4 x 5 x 200	0.5	81.667	2.141
4 x 10 x 200	0.01	151.129	2.250
4 x 20 x 200	0.1	286.755	4.516
4 x 5 x 500	0.3	2,955.361	8.703
4 x 10 x 500	0.5	1,567.890	16.295
4 x 20 x 500	0.01	1,764.744	34.010

Tabla 8. Tiempos de ejecución para cuatro fábricas con instancias de gran tamaño. [Fuente: (Elab. propia)]

4.2.2. Para seis fábricas

Valores de la función objetivo obtenidos:

F x m x n	Mmax	HBBO	HBI	PI
6 x 5 x 100	0.01	1804	1825	-1,15
6 x 10 x 100	0.1	2032	2049	-0,83
6 x 20 x 100	0.3	3038	2930	3,69
6 x 5 x 200	0.01	3167	3076	2,96
6 x 10 x 200	0.1	3748	3720	0,75
6 x 20 x 200	0.3	4657	4453	4,58
6 x 5 x 500	0.01	7291	7242	0,68
6 x 10 x 500	0.1	8096	7915	2,29
6 x 20 x 500	0.3	6939	6614	4,91

Tabla 9. Valores del makespan para seis fábricas con instancias de gran tamaño. [Fuente: (Elab. propia)]

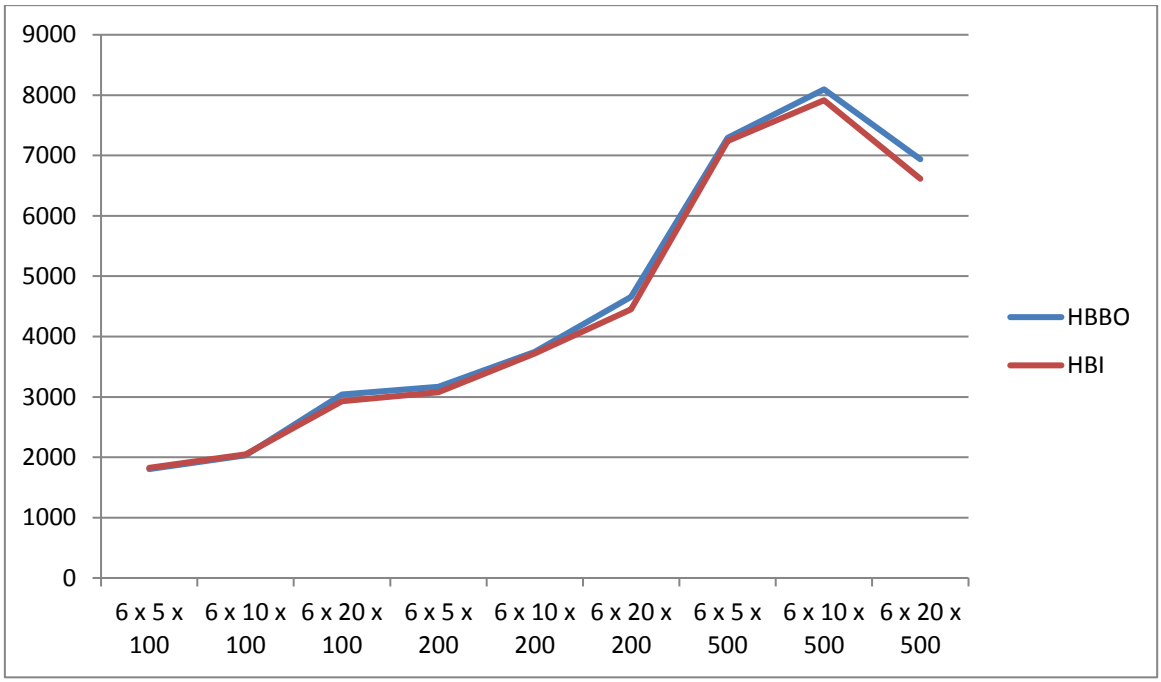

Figura 4-7. Evolución del makespan para seis fábricas. [Fuente: (Elaboración propia)]

Tiempos de ejecución:

F x m x n	Mmax	HBBO (s)	HBI (s)
6 x 5 x 100	0.01	22.029	0.203
6 x 10 x 100	0.1	40.345	0.500
6 x 20 x 100	0.3	72.725	1.078
6 x 5 x 200	0.5	80.848	0.641
6 x 10 x 200	0.01	151.820	1.469
6 x 20 x 200	0.1	287.875	3.486
6 x 5 x 500	0.3	488.170	7.702
6 x 10 x 500	0.5	941.205	16.315
6 x 20 x 500	0.01	1,853.63	34.934

Tabla 10. Tiempos de ejecución para seis fábricas con instancias de gran tamaño.
[Fuente: (Elab. propia)]

4.2.3. Para ocho fábricas

Valores del makespan obtenidos:

F x m x n	Mmax	HBBO	HBI	PI
8 x 5 x 100	0.01	1370	1326	3,32
8 x 10 x 100	0.1	1765	1724	2,38
8 x 20 x 100	0.3	2471	2470	0,04
8 x 5 x 200	0.01	2298	2237	2,73
8 x 10 x 200	0.1	2857	2773	3,03
8 x 20 x 200	0.3	3679	3498	5,17
8 x 5 x 500	0.01	5126	5076	0,99
8 x 10 x 500	0.1	4711	4413	6,75
8 x 20 x 500	0.3	5629	5379	4,65

Tabla 11. Valores del makespan para ocho fábricas con instancias de gran tamaño.
[Fuente: (Elab. propia)]

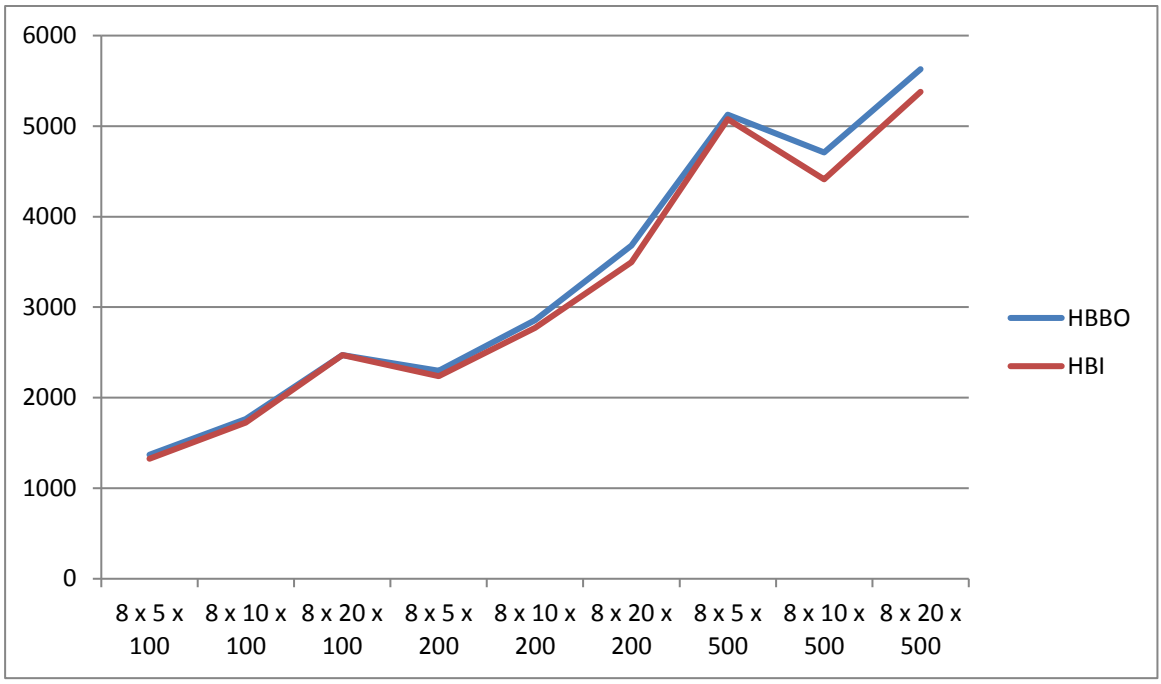

Figura 4-8. Evolución del makespan para ocho fábricas. [Fuente: (Elaboración propia)]

Tiempos de ejecución:

F x m x n	Mmax	HBBO (s)	HBI (s)
8 x 5 x 100	0.01	24.704	0.359
8 x 10 x 100	0.1	45.021	0.844
8 x 20 x 100	0.3	83.808	2.250
8 x 5 x 200	0.5	94.240	0.797
8 x 10 x 200	0.01	176.808	1.849
8 x 20 x 200	0.1	333.453	4.673
8 x 5 x 500	0.3	2,996.481	7.930
8 x 10 x 500	0.5	1,579.362	18.289
8 x 20 x 500	0.01	1,629.308	37.135

Tabla 12. Tiempos de ejecución para ocho fábricas con instancias de gran tamaño. [Fuente: (Elab. propia)]

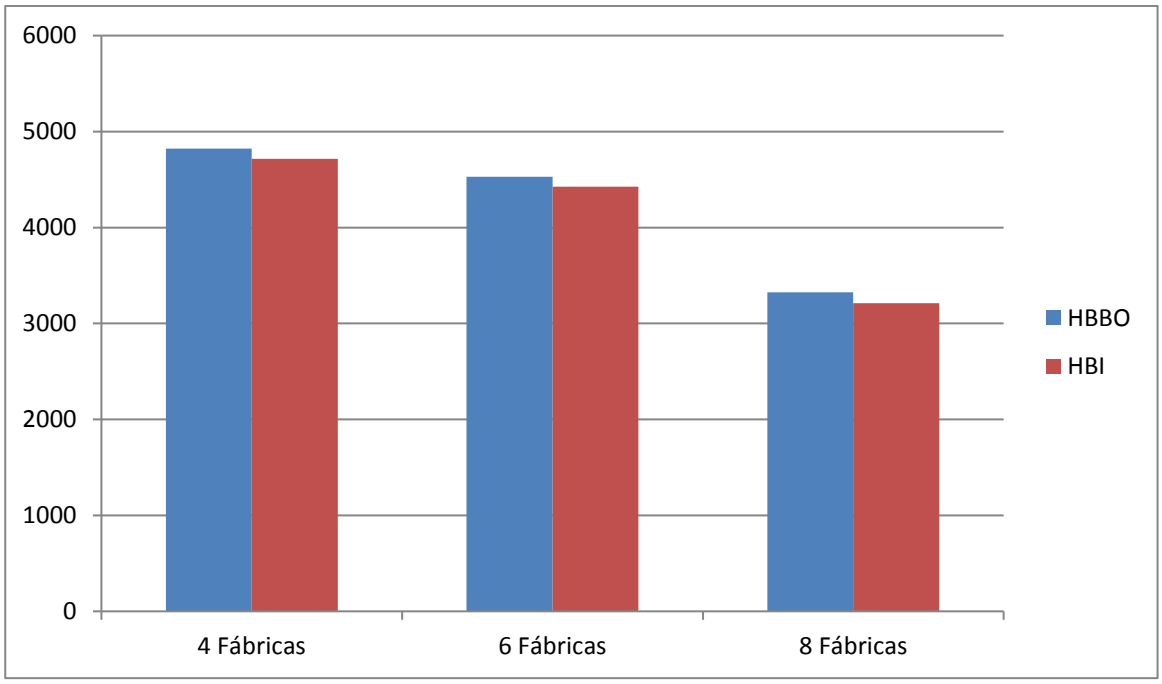

Figura 4-9. Valor promedio del makespan para instancias de gran tamaño. [Fuente: (Elab. propia)]

Como se puede comprobar en las tablas 7, 9 y 11 o en la figura 4-9, el valor del makespan para el algoritmo HBBO es mayor que para el algoritmo HBI, dando un peor resultado, debido a que el algoritmo se basa en mucha aleatoriedad que se utiliza en la selección de trabajos para el intercambio y al ser secuencias de gandes tamaños, estos intercambios pueden no ser los correctos para obtener la mejor solución. Además los tiempos de ejecución para el algoritmo HBBO se incrementan considerablemente.

4.3. Recapitulación del apartado 4

Los aspectos fundamentales tratados en el apartado 4 son los siguientes:

- Se detallan los datos con los que se realizarán el análisis de los resultados. Además se elabora un índice que determine la eficacia del algoritmo HBBO.

- Se realizan los cálculos para instancias de pequeños tamaños. Se elaboran tablas y gráficas con las que facilitar la comparación entre los algoritmos HBBO y HBI. A continuación se realiza una conclusión.

- Se efectúan los cálculos para instancias de grandes tamaños, en las que se desarrollan tablas y gráficas por la misma razón. Posteriormente se redacta una conclusión a esos cálculos.

Por último, obtenido todos los conocimientos y resultados del estudio, se elabora una conclusión final en la cual se indica para qué tipos de problemas es conveniente el uso del algoritmo HBBO.

5 CONCLUSIONES

El objetivo principal de este documento es comprobar la eficacia del algoritmo HBBO mediante una comparación con el algoritmo HBI y ver en qué circunstancias es conveniente la utilización del mismo, además del aprendizaje con programas informáticos que puedan servir en un futuro y la obtención de nuevos conocimientos, como por ejemplo, sobre algoritmos evolutivos. En general, afrontar estos tipos de problemas es lo que le proporciona al alumno una mejora de las capacidades en la búsqueda de información, de razonamiento crítico, de saber redactar un documento técnico, etc.

Una vez realizado el estudio y comprobado la eficacia del algoritmo HBBO, es posible llegar a una conclusión aceptable del problema. El algoritmo HBBO realiza una búsqueda exhaustiva mediante varios métodos de búsquedas locales, ejecutándose un número de iteraciones máximo para obtener una mejor solución. Por cada iteración se obtiene un hábitat, que es comparado con los demás, para escoger el mejor de ellos. En estas búsquedas locales se realizan permutaciones de trabajos en cada hábitat. Estos intercambios vienen influidos por aleatoriedad, ya que un número aleatorio es el que indica que trabajo intercambiar con los demás. En cambio, el algoritmo HBI resulta ser más simple, realizando como búsqueda local, sólo la inserción de los trabajos en todas las posiciones, ejecutándose en un tiempo mucho menor. No siempre será apropiado el uso del algoritmo HBBO antes que el HBI, ya que se puede tener como inconveniente el tiempo de ejecución, siendo en alguna ocasión, tan elevados que puedan suponer una pérdida de tiempo.

Los resultados computacionales para instancias de pequeño tamaño realizados en el apartado 4.1, demuestran que el algoritmo HBBO es más eficiente que el algoritmo HBI, resultando los valores del makespan más bajo. Sin embargo, los tiempos de ejecución son mayores (en torno a 1 segundo), pero no son lo suficientemente elevados como para tener importancia en la selección de un algoritmo u otro. Por tanto, si se tiene una empresa en la cual el número de trabajos y máquinas no son muy elevados, resulta más adecuado el uso del algoritmo HBBO para la obtención de la secuencia óptima.

Por otra parte, para instancias de gran tamaño, como se puede observar en el apartado 4.2, indican que el algoritmo HBI da mejores resultados que el algoritmo HBBO, dando valores del makespan menores. Además, el tiempo de ejecución del algoritmo HBBO puede suponer cada vez un mayor problema, ya que éste va aumentando considerablemente conforme se aumentan el número de trabajos, máquinas y fábricas, llegando a resultar una pérdida de tiempo por parte del encargado, mientras que el algoritmo HBI da mejores resultados de forma más rápida. En conclusión, si la empresa a estudiar tiene un gran tamaño y maneja un alto número de trabajos y máquinas, resulta más eficiente el uso del algoritmo HBI que el uso de algoritmo HBBO en la exploración para encontrar la mejor secuencia.

REFERENCIAS

Al-Anzi, F. S., and A. Allahverdi. (2006). "A Hybrid Tabu Search Heuristic for the Two-stage Assembly Scheduling Problem". *TheInternational Journal of Operational Research , 3* (2), 109–119.

Berzal, F. (S. F) "Algoritmos Genéticos ". Universidad de Granada, Departamento de Ciencias de la Computación, Granada.

Chann, F. T., Chung, S. H., & Chan, P. (2005). "An adaptative genetic algorithm with dominated genes for distributed scheduling problems". *Expert Systems with Applications, , 29* (2), 364-371.

D. E. Golberg (1989). "Genetic Algorithms in Search, Optimization and Machine Learning*". Addison-Wesley*, Reading, MA.

Framinan, J. M., and R. Leisten. (2003). "An Efficient Constructive Heuristic for Flowtime Minimisation in Permutation Flow Shops." *Omega, The International Journal of Management Science* 31 (4): 311–317.

Framiñan. (2016). "Librería del programa Code::Blocks". Universidad de Sevilla, Sevilla.

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2013). "The distributed assembly permutation flowshop scheduling problem". *51* (17), págs. 5292-5308.

Jian Lin, Shuai Zhang (2016). "An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem". *Computers & Industrial Engineering , 97*, 128-136.

J. T. Alander (1992). "On optimal population size of genetic algorithms". *Computer Systems and Software Engineering, 6th Annual European Computer Conference,* 65-70.

Lee, C. Y., T. C. E. Cheng, and B. M. T. Lin. (1993). "Minimizing the Makespan in the 3-machine Assembly-type Flowshop Scheduling". *Management Science , 39* (5), 616–625.

Naderi, B., and R. Ruiz.. (2010). "The Distributed Permutation Flowshop Scheduling Problem". *Computers & Operations Research , 37* (4), 754-768.

Pan, Q, K., Tasgetiren, M. F., & Lian, Y. C. (2008). "A discrete differential evolution algorithm for the permutation flowshop scheduling problem". *Computers & Industrial Engineering , 55* (4), 795-816.

Perez, Paz (2017). "Apuntes de Programacion de Operaciones". Universidad de Sevilla, Sevilla.

Potts, C.N., S.V. Sevast Janov,V.A. Strusevich, L. N.VanWassenhove, and C. M. Zwaneveld (1995). "The Two-stageAssembly Scheduling: Complexity and Approximation". *Operations Research , 43* (2), 346–355.

Simon, D. (2008). "Biogeography-based optimization". *IEEE Transactions on Evolutionary Computation , 12* (6), 702-713.

Tasgetiren, M. F., Pan, Q, K., Suganthan, P. N., & Chen, A. H. (2011). "A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops". *Information Sciences , 181* (16), 3459-3475.

Tasgetiren, M, F., Liang, Y.-C., Sevkli, M., & Gencyilmaz, G. (2007). "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem". *European Journal of Operational Research , 177* (3), 1930-1947.

Tozkapan, A., O. Kirca, and C. S. Chung. (2003). "A Branch and Bound Algorithm to Minimize the Total Weighted Flowtime for theTwo-stage Assembly Scheduling Problem". *Computers & Operations Research , 30* (2), 309–320.

ANEXOS

1. ANEXO 1. FUNCIONES DE LA BIBLIOTECA SCHEDULE

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);

Máximos y mínimos entre dos números.

- int iMax(numero1, numero2)
- long iMax(numero1, numero2)
- int iMin(numero1, numero2)
- long iMin(numero1, numero2)

Redondea un número (convierte un doublé en un long int). Si excess vale 1 redondea al entero superior. En caso de valer cero se redondea al entero inferior.

- long int round(doublé numero, int excess)

Generar una secuencia aleatoria con un número de elementos definidos.

- void randSequence(VECTOR_INT vector, int numero_elementos)

Rellenar un vector o matriz con números iguales.

- void setva_IVector (vector, tamaño, valor)
- void setval_LVector (vector, tamaño, valor)
- void setval_DVector(vector, tamaño, valor)
- void setval_IMatrix(matriz, filas, columnas, valor)
- void setval_LMatrix(matriz, filas, columnas, valor)
- void setval_DMatrix(matriz, filas, columnas, valor)

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)

Copia una fila de una matriz en un vector.

- void copyrowImatrix(matriz, vector_destino, fila, numero_columnas)
- void copyrowLmatrix(matriz, vector_destino, fila, numero_columnas)
- void copyrowFmatrix(matriz, vector_destino, fila, numero_columnas)
- void copyrowDmatrix(atriz, vector_destino, fila, numero_columnas)

Ordena el vector en función de sus valores de manera ascendente o descendente.

- void sortIVector(vector, numero_elementos, orden)

Ordena un vector en función de sus índices de manera ascendente o descendente.

- void sortLVector(vector, vector_orden, numero_elementos, orden)
- void sortFVector(vector, vector_orden, numero_elementos, orden)
- void sortDVector(vector, vector_orden, 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)