Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Análisis y desarrollo de métodos constructivos para talleres abiertos sin restricciones

Autor: Alejandro Hernández Pérez

Tutor: Víctor Fernández-Viagas Escudero

Dpto. de Organización y Gestión de empresas
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Análisis y desarrollo de métodos constructivos para talleres abiertos sin restricciones

Autor:

Alejandro Hernández Pérez

Tutor:

Víctor Fernández-Viagas Escudero

Contratado Doctor

Dpto. de Organización y Gestión de empresas
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Análisis y desarrollo de métodos constructivos para talleres abiertos sin restricciones

Autor: Alejandro Hernández Pérez
Tutor: Víctor Fernández-Viagas Escudero

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mis padres y a mis amigos por apoyarme a lo largo de estos años en todo momento. A María José que llegó en el momento más duro y me motivo para esforzarme al máximo y llegar al final de este viaje. A mi tutor Víctor, por su disposición a ayudarme cuando se enfrentaba a un problema y por su entrega, siempre buscando sacar lo mejor de este proyecto.

Alejandro Hernández Pérez
Sevilla, 2021

Resumen

Este proyecto se centra en la programación de operaciones de un problema de flujo abierto *(Open Shop Scheduling)* sin restricciones de tipo NP-Hard. Para ello se implementan y evalúan diferentes algoritmos aproximados con el objetivo de contrastar su eficiencia frente a este tipo de problemas en dos casos distintos de funciones objetivo de minimización. Se evaluán los resultados y otros factores como el coste computacional *(CPU time)* y en base a estas variables se determina el mejor de los métodos para cada función objetivo propuesta. Así mismo, se reflejan las diferencias entre el comportamiento de los diferentes métodos aplicados según la variable a minimizar.

Abstract

This project focuses on the operation programming of an open shop NP-Hard problem without restrictions. Algorithms and heuristics have been implemented in order to contrast their efficiency with this type of problems in two different cases of minimizing the objective function. The results and other factors, such as CPU time are evaluated, and taking these variables into account, the best method for each suggested objective function is determined. Furthermore, the differences between the behavior of the different applied methods are shown according to the variable that needs to be minimized.

Índice

Índice de Figuras

Índice de Tablas

1 Introducción

El problema que vamos a analizar es recurrente en la industria actual y se aborda desde el ámbito de la organización de la producción, en particular, con la programación de la producción.

Se entiende por "programación de operaciones" el proceso de toma de decisiones que consigue asignar tareas a determinados recursos y determinar el momento en el que comienzan dichas tareas en cada uno de los procesos. El resultado de aplicar la programación de la producción es una secuencia (*production scheduling*) que nos indica qué trabajo se asigna a cada recurso y el correspondiente orden junto con el esquema temporal, es decir los tiempos de comienzo y terminación de cada operación.

Para comprender mejor el significado de los conceptos "trabajo" y "recurso" se ejemplificará a continuación en el contexto de una empresa o industria real. Podríamos considerar las diferentes líneas de productos a desarrollar (cada una de las divisiones de una actividad de fabricación) como trabajos independientes (*Jobs*); por otro lado, definimos los recursos como las operaciones o procesos por las que tiene que pasar cada trabajo para completarse, en términos de *"manufacturing scheduling"*, denominando a los recursos máquinas (*Machines*). Las máquinas pueden representar a una máquina industrial estándar o a un operario que desarrolle manualmente cierto paso del proceso productivo. Existen también diferentes ámbitos de la industria donde la programación de la producción juega un papel importante. Así por ejemplo, un ámbito de aplicación que pudiese no parecer tan evidente partiendo de la descripción dada es el sector sanitario, en particular la gestión de un hospital tratando a los enfermos como los trabajos y a las camas, quirófanos y personal médico como recursos.

Una vez definidos los parámetros principales sobre los problemas de planificación de la producción podemos ir un paso más allá examinando todas las circunstancias que envuelven a un problema de este tipo, es decir, desde el entorno de la fábrica hasta la caracterización de los trabajos (*Jobs*). En la Figura 1.1 se plasman las tres características que definen cualquier problema de programación de la producción:

En primer lugar, trataremos la distribución de las máquinas (*Machine Layout*) que se denomina con la letra griega Alpha (α), pudiendo encontrar diferentes opciones:

- Máquinas de trabajo en paralelo:
  - Identical Parallel machines: Máquinas que realizan el mismo proceso y que tardan el mismo tiempo en realizarlo. El trabajo solo tienen que pasar por una de ellas.
  - Unrelated machines: Al igual que para el caso anterior el trabajo solo tiene que pasar por una de ellas pero la diferencia es que los tiempos de proceso en cada máquina son distintos.
- Máquinas en entornos tipo taller: En un entorno tipo taller los trabajos (jobs) o productos deben de pasar por todas las máquinas que hay, ya que cada una de ellas realizan operaciones distintas que componen las partes para la fabricación del trabajo. Existen distintos tipos de talleres, pero los principales son el *Job Shop* (*Jm*), el *Flow Shop* (*Fm*) y por último el *Open Shop* (*Om*), que será el que abordaremos en nuestro problema y se explicará en el capítulo tres.

A continuación, vemos las restricciones a las que nos podemos enfrentar, principalmente enfocadas a las características de los trabajos (*Job Characteristics*), denominadas con la letra Beta β. Se definen éstas como

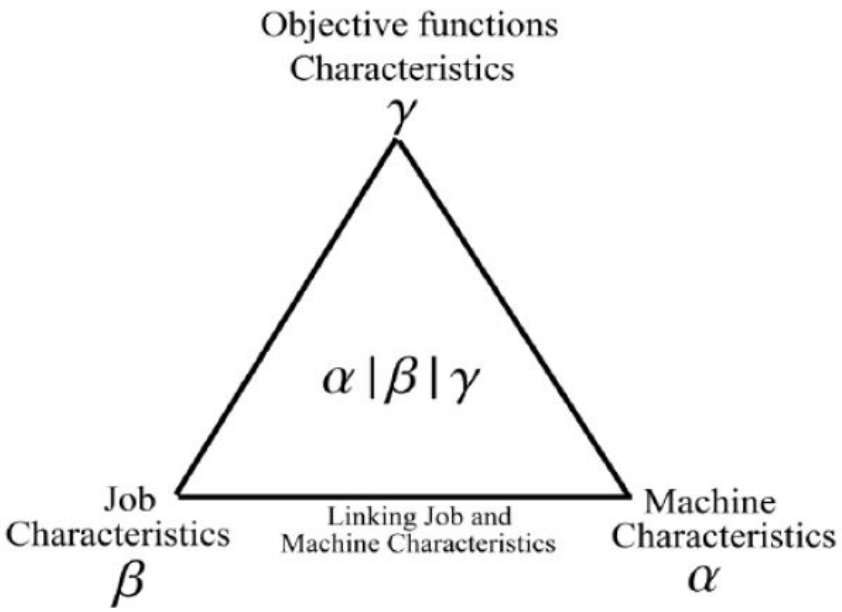

Figura 1.1 Esquema de clasificación de modelos de programación de la producción. [Fuente: [1]].

las condiciones que tienen que cumplir los trabajos para que la secuencia que se obtiene como resultado sea válida, con restricciones que pueden surgir por petición de clientes, cuestiones logísticas y/o técnicas. Por mencionar algunas de ellas:

- d_j: Es un vector que indica las fechas de entrega límite para las que tiene que estar terminado cada trabajo j.
- Prmu: Indica permutación. Todos los trabajos siguen la misma secuencia en todas las máquinas.
- R_j: fecha de llegada del trabajo j. Es decir, es el momento en el que tenemos su disponibilidad para poder introducirlo en cualquier máquina.

Por último, el vértice superior del triángulo de la figura es la caracterización de la función objetivo del problema de programación de la producción (*Objective function Characteristics*), a la que se ha asignado la letra Gamma γ. Algunas de las funciones objetivo que se suelen implementar son:

- La minimización del tiempo de terminación, (Min C_{max}) es decir, del momento en el que finaliza de ejecutarse la última operación asignada.
- La minimización de la suma total de los tiempos de terminación de cada trabajo (Min $\sum C_j$).
- La minimización del número de trabajos tardíos (Min U_j), o número de trabajos que se terminan de procesar más tarde de la fecha de entrega límite.

Siendo los dos primeros casos de funciones objetivo descritas los que trataremos en nuestro problema. Una vez definidos estos conceptos, podemos abordar el problema particular objeto de este proyecto, que se definirá con detalle en los siguientes capítulos.

2 Objeto del proyecto

2.1 Presentación

El objeto de este proyecto es analizar y estudiar siete algoritmos de programación de la producción que se aplican a un problema Open shop sin restricciones, y determinar la mejor opción a implementar en este tipo de entornos. Dependiendo del contexto y las prioridades empresariales se usará una función objetivo distinta, y abordaremos los dos siguientes casos: la minimización del tiempo máximo de terminación, y la minimización de la suma de los tiempos de terminación de todos los trabajos.
Lo que se pretende estudiar es la calidad y rapidez de los resultados comparándose los métodos entre sí, ya que no podemos abordar el estudio de todas las soluciones posibles del problema que planteemos y no tendremos, por tanto, la solución óptima como referencia. Este proceso de estudio se deberá realizar de forma independiente para cada una de las funciones objetivo a examinar, mediante la ayuda de herramientas de software y otras herramientas de manejo de datos que explicaremos a lo largo del capítulo cinco.
Una vez concluido el mencionado estudio, se desarrollará una evaluación de los datos según diversos criterios y para cada función objetivo se determinará que heurística sería la mejor opción a implementar en un entorno empresarial, dependiendo de unas prioridades y/o limitaciones hipotéticas que se proponen.

2.2 Contexto del problema

Actualmente, la correcta secuenciación de trabajos en el conjunto de máquinas de un open-shop es un problema de alto nivel, considerado un problema tipo NP, es decir, no resoluble en un tiempo polinomial. Es importante aclarar que, si bien es posible encontrar una solución válida y factible para la función objetivo que estemos evaluando, la no polinomialidad en los tiempos de computación hace referencia a encontrar la solución óptima del problema examinando todas las posibles o mediante métodos exactos.

Como aclaración adicional y para el entendimiento del alcance de la situación estaríamos hablando de los problemas mencionados en el problema del milenio P vs NP. Sin entrar en detalle, sirva este caso para ejemplificar y entender estos problemas de carácter exponencial: si queremos determinar todas las formas posibles de asignar 30 personas en 30 posiciones diferentes de forma que todas las personas tengan un puesto y ninguna plaza quede vacante, estaríamos analizando 30! soluciones posibles, para lo que un ordenador de computación lenta necesitaría para resolverlo unas seiscientas mil edades del universo [1]. En el caso de un ordenador que computase más rápido serían más de un billón y medio de años para tener la evaluación de todas las opciones.

En relación al proyecto que se aborda, este caso detallado previamente es análogo a un problema de asignación de trabajos en una máquina, los trabajos corresponderían al número de personas y el orden de ellos en la máquina correspondería a las posiciones. Además, un problema de programación de operaciones en entornos de tipo taller abierto con más de una máquina incrementa de manera muy significativa el número de soluciones

[1] Una edad del universo equivale a catorce mil millones de años

posibles dado que hay que determinar el orden de las operaciones en todas las máquinas a diferencia del caso del ejemplo indicado (que sería como ordenar treinta trabajos en una máquina).

2.3 Justificación

Podemos considerar que es uno de los problemas más genérico que existen y, por tanto, puede ser adaptados a diferentes entornos de producción empresarial haciendo modificaciones sobre las heurísticas propuestas, considerando las posibles restricciones que se diesen en cada caso.

Al ser un problema de tipo NP, necesitamos usar heurísticas que acoten todas las soluciones válidas del problema, descartando algunas de ellas que podrían no dar buenos resultados para la función objetivo a analizar (por experiencias previas u otros análisis matemáticos).

Paralelamente examinamos las soluciones del grupo acotado primeramente y tomaríamos la que nos dé el mejor valor como mejor solución. Sin embargo, esto no es garantía de optimalidad ya que el conjunto estudiado no es el total del problema y podría existir una solución mejor.

3 Descripción del problema

En este trabajo estudiaremos el problema *Open Shop Scheduling* (OSS) o secuenciación de un taller abierto, en particular el caso sin restricciones.

Los métodos que trataremos están propuestos en [2] y [4], y fueron originalmente diseñados para solucionar un problema con función objetivo *makespan minimization*, $MinC_{max}$.En primer lugar, estás heurísticas han sido implementadas para el caso de minimización del *makespan* y, posteriormente han sido adaptadas para la minimización del tiempo total de finalización (*total completion time, TCT*). Según la notación explicada en el capítulo 1 los problemas que vamos a abordar en el presente proyecto se denotan como $Om||Min\ C_{max}$ y $Om||Min\ \sum C_j$.

Antes de entrar más en detalle en el concepto de un Open Shop debemos definir un par de términos de vital importancia para comprender los conceptos que se desarrollan más adelante.

- La primera variable que vamos a describir es el tiempo de proceso, y se entiende por el tiempo que tarda una operación en realizarse, por tanto, cada trabajo lleva asociado tantos tiempos de proceso como máquinas en las que se realiza. Los tiempos de proceso se denotan como P_{ij} y se agrupan en una matriz, llamada matriz de tiempos de proceso, de tamaño n filas y m columnas. Los subíndices de la matriz corresponden al número de trabajo y máquina respectivamente.

- La otra variable importante es el tiempo de terminación de un trabajo, denotado por C_{ij}, y se relaciona con el momento en el que un trabajo finaliza su procesado en todas las máquinas por las que debía pasar. De manera más amplia se define la matriz de los tiempos de terminación, del mismo tamaño que la de tiempos de proceso, y siendo el objeto de la resolución del problema ya que nos permitiría obtener cualquier otra información de las operaciones sobre la que estuviésemos interesados.

Para una definición del problema *OSS* nos referimos a [3]: *"In theoretical computer science and operations research, the open-shop scheduling problem (OSSP) is a scheduling problem in which a given set of jobs must each be processed for given amounts of time at each of a given set of workstations, in an arbitrary order, and the goal is to determine the time at which each job is to be processed at each workstation."* .

Adicionalmente a esta definición, que se podría resumir como que el objetivo de la resolución de un problema *OSS* es asignar un tiempo de comienzo a cada trabajo en cada máquina, se establecen las siguientes condiciones que hay que cumplir antes de llevar a cabo esta tarea. La primera de ellas es que un trabajo no puede estar asignado a dos máquinas al mismo tiempo y, por otro lado, el tiempo durante el cual dicho trabajo se encuentra en la máquina es estrictamente igual a su tiempo de proceso.
La solución del problema se ilustra de manera clara en la Figura 3.1, donde cada fila corresponde a una máquina y cada columna a un trabajo. La ruta de cada trabajo se aprecia con las flechas de color azul oscuro y negro que hay en la Figura 3.1 de tal forma que, un trabajo comienza en la máquina a la que le apunte el círculo A que marca el inicio y termina en omega Ω; en la parte intermedia las operaciones siguen el orden que marcan las flechas negras en la columna de cada trabajo. Los números que se encuentran dentro de los círculos corresponden a los tiempos de proceso. Este proceso de asignación lo que construye de manera indirecta finalmente es una secuencia, mediante la cual se puede leer el orden de los trabajos en cada máquina

y conocer todos sus tiempos de comienzo y terminación.

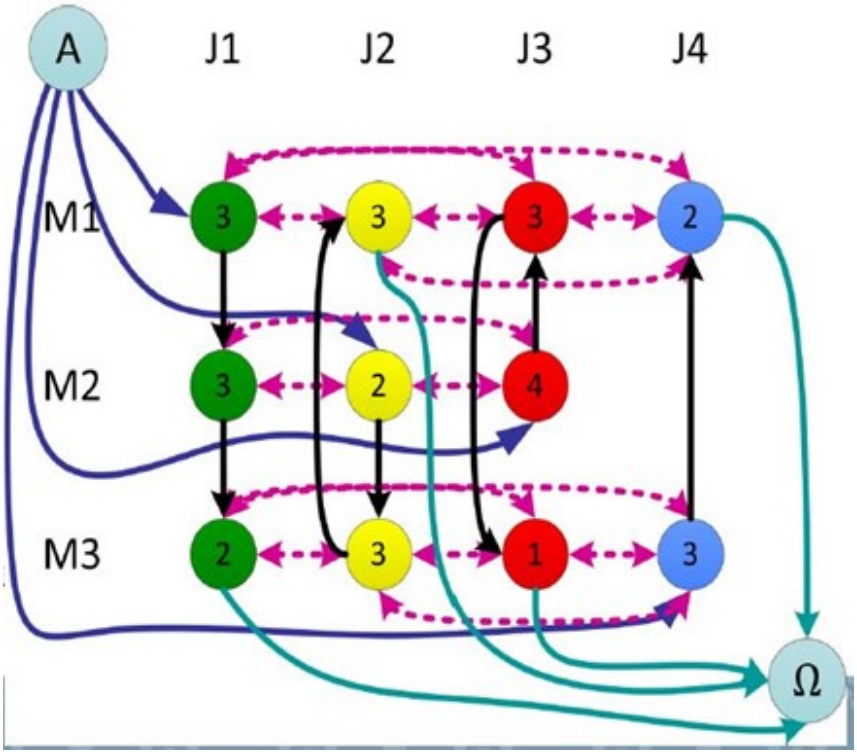

Figura 3.1 Esquema de secuenciación de operaciones en un openshop. [Fuente: [1]].

Una vez definidos los problemas *Openshop*, a modo de ejemplo se plantea un problema de este tipo con tres trabajos a asignar en tres máquinas. Como dato de entrada únicamente tenemos el tamaño del problema, ya mencionado (3x3), y la matriz de tiempos de proceso. Una de las posibles soluciones se muestra en el diagrama de Gant en la Figura 3.2:

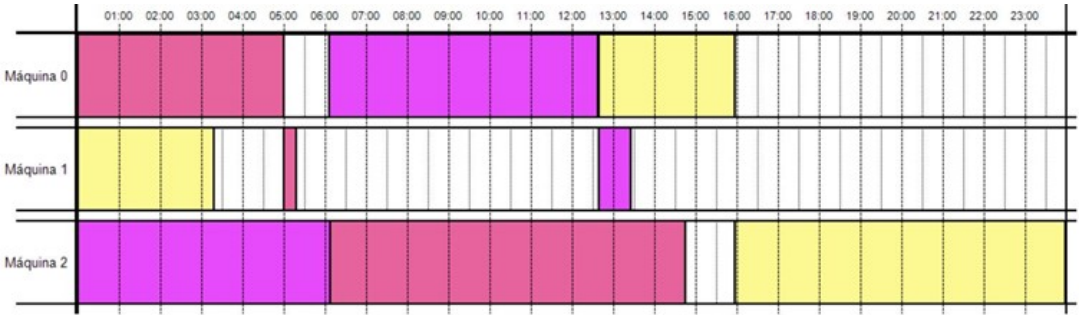

Figura 3.2 Diagrama de Gant. [Fuente: Elaboración propia].

Cada recuadro de un color distinto representa a un trabajo y la ruta de cada trabajo se aprecia visualmente en la Figura 3.2, por ejemplo, el trabajo amarillo se ejecuta en la segunda máquina, luego en la primera y finalmente en la última. Lo que se obtiene como solución al resolver un problema OSS (*Open Shop Scheduling*) es la secuencia de los trabajos en cada máquina (la ruta se extrae de aquí).

4 Descripción de las heurísticas

4.1 Introducción

La idea principal de las siete heurísticas es segmentar o acotar el conjunto de soluciones admisibles para limitar el análisis de las secuencias entre las opciones de este subconjunto dentro de toda la región de soluciones válidas del problema.

El objetivo principal de las cuatro primeras heurísticas es el examen a través de la reestructuración y asignación iterativa de secuencias parciales y el estudio de las vecindades para lograr eventuales mejoras del valor de la función objetivo. Por un lado, esta reestructuración o reordenación de los elementos, se realiza descartando introducir operaciones en ciertas posiciones donde se diese el caso de que la solución obtenida fuese igual a una examinada previamente. Poder distinguir que casos nos van a producir ese tipo de secuencias finalmente y ser capaces de evitar su ejecución permitirá ahorrar un tiempo bastante considerable en el desarrollo de los métodos.

Cuando se da esta situación, es decir, que creamos una secuencia similar a una ya analizada, aparece la condición de redundancia y podemos establecer el criterio que se aplica a la hora de programar para saber si una secuencia es redundante es el siguiente:

- Si al introducir una tarea en la posición *i* de una secuencia, la operación anterior, en la posición *i-1*, tiene el trabajo y la máquina asociada distinto a los de la primera tarea mencionada (posición i) respectivamente. Se tiene que dar la diferencia de ambos factores de una operación respecto a la otra, en cualquier otro caso no se daría la condición de redundancia.

Las heurísticas tienen distintos grados de complejidad, básicamente esto quiere decir que tan grande será el ámbito de soluciones válidas del problema que examina el método. Lo que se pretende lograr introduciendo el concepto de la redundancia en los códigos es descartar (dentro del conjunto de soluciones a examinar) las soluciones que no aporten variaciones en la solución, o en otras palabras, que sean alguna forma de copia de secuencias ya analizadas.

Este criterio se aplica de alguna u otra forma en las cuatro heurísticas implementadas y reduce considerablemente el número de combinaciones que analiza cada uno de estos métodos. La importancia de esto radica en el avance exponencial del tiempo para generar todas las soluciones válidas conforme aumenta linealmente el tamaño de los problemas de programación de la producción.
Por otra parte los tres métodos restantes que completan el análisis que se realiza en este proyecto son casos particulares del mismo algoritmo, *BICH-MIH*. Este algoritmo basa la evaluación de la posición que ocupará una operación en la secuencia en el resultado de una media ponderada de dos criterios. Dicha media se caracteriza por el parámetro de ponderación denotado como alpha α que determina la importancia que se le da a cada uno de los criterios, los cuales se explicarán en la descripción de este algoritmo (Sección 4.2.5). Es por ello por lo que se ramifica este algoritmo en tres métodos ya que se estudian tres casos de valores de α.

El desarrollo de las heurísticas se explicarán según alcancemos cada punto, pero previamente introduciremos otros conceptos en la siguiente sección.

4.1.1 Notación, evaluación e implementación de la secuencia

La decodificación de una secuencia es un tema crucial dentro de la programación de la producción porque no siempre es directa, es decir, que el vector de operaciones almacenados corresponde a la solución del problema. Existen casos en los que la secuencia se somete a un proceso de decodificación que consiste básicamente en una reestructuración del orden siguiendo algún criterio (este concepto lo expondremos más adelante en este mismo capítulo). Dicho proceso de decodificación se realiza de manera previa al proceso de evaluación, el cuál es válido también para casos donde se aplique el algoritmo de decodificación.

Básicamente, el proceso de la evaluación de una secuencia consiste primero en inicializar a cero todas las variables asociadas a los tiempos de finalización de las operaciones y conforme se examina cada componente de la secuencia se completan debidamente. Las operaciones se asignan lo antes posible a las máquinas correspondientes (programa semi-activo) y la notación de las variables y sus interrelaciones que se usan para la evaluación de una secuencia en este documento son:

- *Processingtimes[job,machine]*: Indica el tiempo que tarda en ejecutarse el trabajo (*job*) en la máquina (*machine*).Se usará la notación P_{ij} como abreviatura.
- *Completiontimesjob[job]*: Tiempo de terminación del trabajo (*job*) en la última máquina en la que se ha ejecutado.
- *Trabajospormaquina[machine]*: Es un contador del número de trabajos que se han ejecutado en cada máquina, sirve para llevar correctamente el registro de la disponibilidad de las máquinas.
- *Availabletimes[job, machine]*: Indica la disponibilidad de cada máquina en el momento de insertar el trabajo número *trabajospormaquina[machine]*
- *Startingtimes[job, machine]*: Se define como el tiempo de comienzo (de *job* en *machine*) y es igual al máximo entre el tiempo en el que comienza a estar disponible la máquina (*machine*), equivalente a *Avalaibletimes [machine, trabajospormaquina[machine]*], y el tiempo en el que termina el trabajo (*job*) en la máquina previa donde se hubiese ejecutado, es decir, *Completiontimesjob[job]*.
- *Completiontimes[job, machine]*: Se corresponde con el tiempo de terminación de cada trabajo (*job*) en cada máquina (*machine*) y se calcula como el tiempo de comienzo de ese mismo trabajo en esa maquina más su tiempo de proceso, *Processingtimes[job,machine]*.

Siguiendo las definiciones de las variables y la interrelación entre ellas mismas se pueden determinar al completo todas las características de una secuencia al final de su evaluación. Este proceso de la evaluación de heurística se realiza en la función evaluarsecuencia y se ejecuta hasta llegar al número de operaciones introducidas en el vector secuencia, se indica en "*lengtheval*" como dato de entrada a la función. Veremos más adelante que al ser los métodos iterativos el tamaño de la secuencia varía cada vez que es evaluada y de ahí la necesidad de la la variable que se acaba de indicar. Se puede visualizar lo descrito previamente en la Figura 4.1:

```
for (int k = 0; k < lengtheval; k++)
{
    int completiontimemaxjob;
    int job = Operationsij_aux[secuencia[k], 0];
    int machine = Operationsij_aux[secuencia[k], 1];
    int PO = TP_Oper[secuencia[k]];

    completiontimemaxjob = completiontimesjob[job];
    startingtimes[job, machine] = Math.Max(avalaibletimes[machine, trabajospormaquina[machine]], completiontimemaxjob);
    completiontimes[job, machine] = startingtimes[job, machine] + PO;
    trabajospormaquina[machine]++;
    avalaibletimes[machine, trabajospormaquina[machine]] = startingtimes[job, machine] + PO;
    completiontimesjob[job] = completiontimes[job, machine];
}
```

Figura 4.1 Proceso de evaluación de secuencia. [Fuente: Elaboración propia].

Las heurísticas se desarrollan en funciones independientes y estas reciben tres datos de entrada: el número de trabajos, n; el número de máquinas, m; y por último, la matriz de los tiempos de proceso, donde cada valor de éstos asociados a un trabajo y una máquina siendo las máquinas independientes entre sí. Como se muestra en la Tabla 4.1 (matriz P_{ij}), la matriz de processing times se denota por la letra P, y para hacer referencia a una casilla exacta dentro de dicha matriz se utilizan dos subíndices, el primero para la fila y el segundo para la columna.

Paralelamente a esta matriz de tiempos de proceso tenemos que construir la matriz de operaciones O, para saber que valor le corresponde a cada tarea. La asignación a cada casilla de esta matriz nueva se realiza de izquierda a derecha y de arriba a abajo, se comienza con el número cero y se incrementa en uno el valor que adquirirá cada casilla ij conforme pasamos de una a otra.

Tabla 4.1 Matriz de tiempos de proceso. [Fuente: Elaboración propia].

Pij	1	2	3
1	8	16	6
2	16	4	23
3	25	15	19
4	10	19	20
5	20	16	6

Se adjunta una tabla ilustrando el concepto de la asignación de los índices para cada tarea definiendo así la matriz de operaciones O_{ij}, ver Tabla 4.2.

Tabla 4.2 Matriz de operaciones. [Fuente: Elaboración propia].

Oij	1	2	3
1	0	1	2
2	3	4	5
3	6	7	8
4	9	10	11
5	12	13	14

Como ya hemos explicado en nuestro problema vamos a abordar dos funciones objetivo distintas para todos los métodos que se plantean en este punto, entonces se quiere aclarar la forma de proceder respecto a este tema en las explicaciones siguientes del apartado 4.2 para evitar duplicidades en el desarrollo:

Las heurísticas están desarrolladas como funciones siendo los datos de entrada el tamaño del problema, las variables n y m, y la matriz de tiempos de proceso que se genera aleatoriamente al proponer valores de n y m. El único dato de salida de las heurísticas es un valor de tipo entero y es el resultado de la función objetivo después de aplicar el método completo, es por ello que por cada heurística se hacen dos funciones. La única alteración respecto unas y otras es el valor que devuelve la función, lo cual se indica en el nombre de esta con las extensiones TCT ó Makespan (que hacen referencia al criterio que se busca minimizar en dicha función, correspondiéndose TCT con la minimización del totalcompletiontime y Makespan con la minimización del tiempo de terminación), por otra parte, todas las funciones auxiliares que contribuyen al cálculo de la función objetivo durante el desarrollo de la heurística también tienen una función hermana de manera que los procesos son calcados salvo en la parte que respecta al objetivo que se quiere calcular, lo cual cambia. De manera que según el objetivo que se persiga se hará uso de las funciones principales y auxiliares que corresponden a dicha meta.

Los pseudocódigos de las heurísticas se ven alterados en la medida en la que en vez de usar el criterio "lowest makespan" para elegir la mejor secuencia y usar esa para iterar e introducir las siguientes operaciones, se usaría el criterio "lowest totalcompletiontime" en el caso de que esta fuese nuestra función objetivo. Esta observación se plasma en la codificación de las heurísticas para ambos objetivos de estudio, minimización de tiempo de terminación y minimización de la suma de los tiempos de terminación de los trabajos.

4.2 Heurísticas

4.2.1 Insertion and Reinsertion Heuristic 1 (IRH1)

Esta primera heurística se fundamenta en un proceso iterativo de inserción de las operaciones (una en cada fase del bucle) hasta completar la asignación de todas las operaciones construyendo así una secuencia solución.

El primer paso es ordenar de mayor a menor en un vector los tiempos de proceso, a partir de ahora denotados como P_{ij}, siendo i y j el número del trabajo y de máquina, respectivamente, al que corresponde ese tiempo de proceso. Esta operación se realiza con un algoritmo de complejidad $n * logn$ para ordenar estos tiempos de proceso que vienen dados en la primera fila de la matriz *Elements* junto con su correspondiente número de la operación a la que corresponde cada uno en la segunda fila de dicha matriz. El vector ordenado se denotará como "*array1*" y los valores que tiene son los índices de las operaciones ordenados según el criterio descendente aplicado a sus respectivos tiempos de proceso.

Ahora comenzamos el proceso iterativo en el que consiste el bucle principal del método que engloba el segundo bucle que veremos más tarde (para más detalles ver Figura 4.2):

- Las operaciones se introducen en el orden que indica el vector "*array1*", es decir leyendo de izquierda a derecha, de forma que la primera operación a introducir es la que se encuentre en la posición primera del vector "*array1*" y la última a introducir será la que esté en la posición final $(n * m - 1)$[1].
- Con la operación seleccionada entramos en el segundo bucle cuyo objetivo es encontrar la posición en la que es mejor introducir esta nueva operación, dicha posición únicamente puede ser anterior o posterior a operaciones ya introducidas en el vector secuencia. Este vector secuencia parte inicializado a cero antes de comenzar el proceso iterativo de la heurística y es en el que de forma progresiva se va asignando la totalidad de las operaciones del problema. Respecto a las posiciones en las que se probará la operación es lo que nos va indica la variable que recorre este segundo bucle, dicha variable empieza en cero siempre al entrar por primera vez en el bucle e incrementa en una unidad al terminar el proceso. Por otro lado, la condición de permanencia en el bucle es que el valor de "rr" sea menor que "r+1" de tal forma que la primera operación a introducir del problema se adjudica en la posición inicial del vector secuencia.
- Solo se prueba una operación en la posición "rr" de la secuencia si para el caso de ese valor de *rr* no exista redundancia, habiendo sido explicado este concepto en el apartado anterior. Cada vez que se prueba una operación en alguna posición para un tamaño de operaciones "r" determinado se evalúa la secuencia de forma directa y se comprueba si el valor de la función objetivo es mejor que el que se había almacenado para el resultado que daba evaluar la secuencia en la posición previa (*Best_FO*), y así sucesivamente. Cada vez que el resultado se mejora respecto al anterior tanto el valor de la función objetivo que da la secuencia como el vector secuencia propiamente se almacenan en las variables *Best_FO* y *Best_secuencia*.[2]
- Al haber probado en todas las posiciones que permita el bucle "rr" y volver a empezar el principal extrayendo la siguiente operación del vector de operaciones ordenado según sus tiempos de proceso, la secuencia en la que vamos a introducir la nueva operación escogida es la que almacenamos en el paso previo como *"Best_secuencia"*. Esto se indica en la penúltima línea del pseudocódigo de la Figura 4.2 básicamente expresando que se introduzca la operación en la mejor posición de todas las evaluadas y con esta secuencia empezar el proceso de introducir la siguiente operación (es decir, volver al inicio del bucle principal), lo cual es equivalente a haber almacenado la mejor secuencia de todas las trabajadas y rescatarla al final del bucle "rr" para que esa sea la secuencia en la que se vaya a meter la siguiente operación.

[1] Para hacer referencia a las componentes de los vectores se seguirá la notación normal de programación en C#, la primera componente es la número 0 y la última es igual a la longitud del vector menos uno

[2] Cuando se incrementa "r", el valor que dé la evaluación cuando se introduce la nueva operación por primera vez se almacena directamente en la variable que guarda el mejor valor de la función objetivo y a partir de ahí los resultados de las próximas evaluaciones se compararán respecto a este. La idea es elegir la mejor solución para el número "r" de operaciones en la secuencia, no se pueden comparar distintos resultados de evaluaciones si provienen de tamaños de secuencia distintos.

Este método, *insertion and reinsertion heuristic*, termina cuando hemos introducido en el vector secuencia todas las operaciones del vector de operaciones ordenado inicialmente, o en otras palabras cuando el bucle principal se haya recorrido completo, que correspondería a $r = n * m$.

```
Procedure IRH1
    E = Sort p_ij in non-increasing order
    W = ∅                                   % the permutation of scheduled operations
    for r = 1 to (n · m) do
        i = job the r-th operation of E belong to
        j = machine r-th operation of E is to be processed on
        for rr = 1 to r do
            redundancy = no
            if rr > 1 and job i ≠ (rr-1)-th job in W and machine j ≠ (rr-1)-th machine in W then
                redundancy = yes
            endif
            if redundancy = no then
                Test operation ij at rr-th position in W
            endif
        endfor
        Insert operation ij in W at the position resulting in the lowest makespan
    endfor
```

Figura 4.2 Pseudocódigo heurística irh1. [Fuente: [2]].

4.2.2 Insertion and Reinsertion Heuristic 2 (IRH2)

En esta heurística se propone, en primer lugar, un proceso de decodificación diferente, denominado "decoding scheme", mostrado en la Figura 4.3. Con éste, se pretende lograr una secuencia sin tiempos de espera introduciendo esta restricción implícitamente al usar este método antes evaluar las *permutation list* dadas.

El criterio que se aplicará es, partiendo de la secuencia de permutación dada, asignar en un nuevo vector que corresponderá a nuestra secuencia decodificada, de entre las operaciones que tenemos en la lista la que menor tiempo de comienzo tenga, en otras palabras, la operación que se pueda ejecutar antes sin importar que máquina sea. Si se diese el caso en el que encontramos más de una operación con igual tiempo mínimo de comienzo elegiremos la que esté más adelantada en la *permutation list* inicial para introducirla en la posición k de la secuencia decodificada. La variable mencionada k comienza con valor cero (para almacenar la primera operación en el comienzo del vector de la secuencia decodificada) y se incrementa en una unidad conforme se introduzca una operación nueva en nuestro vector final y por tanto dicha operación se elimine del original propuesto para decodificar.

Una vez definido el *decoding scheme*, la segunda heurística consiste en la introducción de este proceso de decodificación en la primera heurística propuesta, forzando el cambio de una decodificación directa por este esquema de evaluación que se trata en el apartado anterior.

Este proceso se implementa siempre en el momento en el que queramos obtener el valor de la función objetivo de la *permutation list* que llevemos construida hasta el momento, entonces en lugar de leer y evaluar la secuencia tal y como viene construida en dicho vector pues aplicamos el llamado "*decoding scheme*". Lo que produce es una reestructuración del orden de las operaciones en base a la idea de eliminar los tiempos de espera, y esta nueva secuencia dará lugar a un valor de la función objetivo diferente a lo que daría una

Procedure *decoding scheme*
$S = \varnothing$
U =all operations in given permutation θ
while $U \neq \varnothing$
$y = \min \{ s_{ij} \text{ of } O_{ij} \mid O_{ij} \in U\}$
$R = \{ O_{ij} \mid s_{ij} = y, O_{ij} \in U\}$ %R is a set of operations whose starting times are equal to y
Choose O^* from the set of R with the earliest relative position in permutation θ
Extract O^* from U
Put O^* into S
endwhile

Figura 4.3 Decoding scheme. [Fuente: [2]].

decodificación directa. Tanto la forma directa como la que usa el método explicado son funciones que devuelven el valor de la función objetivo que estemos examinando.

Como ya se vio en el apartado 4.2.1 que corresponde a la primera heurística el único segmento del método que requiere una evaluación de la secuencia de permutación es cuando no hubiese redundancia que entraríamos en el condicionante *if (redundancy == "no")* y por tanto tendríamos que evaluar la secuencia con la operación introducida en la posición que sea en dicho momento; pero ahora en esta segunda heurística se llama a la función que realiza el proceso de decodificación para evaluar la secuencia en vez de hacerlo de forma directa.

4.2.3 Insertion and Reinsertion Heuristic 3 (IRH3)

La heurística tercera parte de la asignación dada después de ejecutar la segunda heurística, es decir que tenemos como datos iniciales la secuencia solución y el valor de la función objetivo que resulta de evaluarla. Este valor será el que miraremos para comparar la evaluación de nuestra nueva secuencia que se obtendrá alterando las posiciones de las operaciones según los criterios de la heurística.

Entrando en detalle en el desarrollo del método consiste en reorganizar las operaciones dentro de la secuencia que ya viene totalmente construida de haber aplicado la *IRH2*. Se comienza de derecha a izquierda a reestructurar el vector secuencia dado de forma que la primera componente a la que se le reevaluará su puesto es la *r-1* siendo *r* inicialmente igual a la longitud de la secuencia (n*m).

Ahora, con esa operación se realiza un proceso iterativo que consiste en probar dicha componente en el resto de las posiciones de la secuencia, siempre que el criterio de la redundancia lo permita como ya hemos visto en la introducción de este capítulo. La secuencia inicial antes de modificarse nos había dado el valor de su función objetivo y lo teníamos almacenado (en la variable *Make* que representa el *Makespan*, como ya se ha mencionado desarrollamos todas las heurísticas tanto para el *Makespan* como para el tiempo de finalización total así que nos centraremos solo en los principios de funcionamiento de las heurísticas). Entonces lo que tenemos que observar es si al reubicar la operación que estaba en la posición *r-1* del vector, se ha mejorado el valor de la función objetivo respecto al que teníamos guardado. En el caso de que se haya mejorado se guarda este valor nuevo como el de referencia y la secuencia nueva es con la que se ha logrado esta solución; además el valor de *r* no se altera y se vuelve al comienzo del bucle while a reorganizar la secuencia con la siguiente operación. El concepto de mejora o no mejora es lo que se plasma en la heurística como la variable "improvement", término inglés que significa precisamente "mejora".

En el caso de que no se haya mejorado el valor de referencia de la función objetivo, la secuencia se mantiene intacta respecto como estaba inicialmente en esa iteración, es decir, antes de empezar a probar nuevas posiciones para la operación escogida, pero también hay que disminuir la variable "r" en una unidad.

La idea es que partiendo del final de la secuencia si se observa que la reubicación de una operación no puede dar un mejor valor de la función objetivo que había anteriormente, se pasa directamente a analizar la siguiente operación a su izquierda y así sucesivamente. Como ya se ha mencionado si se encuentra una secuencia mejor que la que había previamente el valor de r no varía y en la siguiente iteración revisaría la compo-

nente que se encuentre en su posición, que será distinta a la operación que había recolocado en el bucle anterior.

El método se considera completo y con todas las variaciones posibles consideradas y evaluadas cuando la variable *r* llega a la primera posición del vector secuencia, lo que quiere decir que su valor sea cero.

La Figura 4.4 corresponde al pseudocódigo de esta heurística donde se ilustra todo lo explicado previamente.

```
Procedure IRH 3
    W = the permutation produced by IRH 2
    Make = makespan of W
    r = n · m
    improvement = yes
    while r > 0 do
        if improvement = no then
            r = r – 1
        endif
        improvement = no
        i = job at rr-th position in W
        j = machine at rr-th position in W
        for r2 = 1 to (n · m) do
            redundancy = no
            if r2 > 1 and job i ≠ (r2-1)-th job in W and machine j ≠ (r2-1)-th machine in W then
                redundancy = yes
            endif
            if redundancy = no then
                WW = produced by inserting job i at r2-th position in W
                If makespan of WW < Make then
                    Improvement = yes
                    Make = makespan of WW
                    r3 = r2
                endif
            endif
        endfor
        if improvement = yes then
            Shift the position of O_ij to r3-th position in W
            r = n · m
        endif
    endwhile
```

Figura 4.4 Pseudocódigo heurística irh3. [Fuente: [2]].

4.2.4 Insertion and Reinsertion Heuristic 4 (IRH4)

Esta heurística va un paso más allá de la primera heurística, por cada operación que se introduce se trata también el tema de si pudiese existir una mejora en el valor de la función objetivo buscando recolocar las operaciones vecinas.

La primera parte de esta heurística es exactamente idéntica a la heurística *IRH1* y, por tanto nos remitimos a la Sección 4.2.1 para su detalle por tanto no se va a explicar, dicha parte corresponde desde el inicio hasta la línea donde comienza la declaración del bucle que recorre la variable *r3*. La parte de la primera heurística termina dándonos una secuencia, pero necesitamos pasar por otro bucle adicional antes de volver a la iteración principal donde comenzaríamos el proceso con otra operación. Ese proceso

```
Procedure IRH 4
    Sort p_ij in non-increasing order
    W = ∅                                   % the permutation of scheduled operations
    for r = 1 to (n · m) do
        i = job(p_ij [r])
        j = machine(p_ij [r])
        for r2 = 1 to r do
            redundancy = no
            if r2 > 1 and job i ≠ (r2-1)-th job in W and machine j ≠ (r2-1)-th machine in W then
                redundancy = yes
            endif
            if redundancy = no then
                Test job i at r2-th position in W
            endif
        endfor
        Insert operation ij in W at the position p resulting in the lowest makespan
        for r3 = max(1, p-k) to min(r, p+k) do
            Extract operation bc at position r3 from W
            for r4 = 1 to r do
                redundancy = no
                if r4 > 1 and job b ≠ (r4-1)-th job in W and machine c ≠ (r4-1)-th machine in W then
                    redundancy = yes
                endif
                if redundancy = no then
                    Test operation bc at r4-th position in W
                endif
            endfor
            Insert operation bc in W at the position p resulting in the lowest makespan
        endfor
    endfor
```

Figura 4.5 Pseudocódigo heurística irh4. [Fuente: [2]].

que nos falta es lo que se va a detallar en adelante. Para este método hace falta definir el parámetro *k*, así pues he escogido este valor igual a cinco como se propone en la descripción de la heurística en la referencia [2].

El último paso de este método es el tercer bucle donde lo que se hace es probar nuevas secuencias a partir del método de la inserción para obtener nuevas vecindades. Las componentes con las que se prueba dicho método son aquellas que se encuentren desde *k* puestos anteriores a la posición *p* (donde se encuentra la operación con la que estamos trabajando desde el bucle principal), hasta las que estén *k* posiciones por delante. Excepcionalmente si no hay suficientes posiciones por delante y/o por detrás para la k escogida se usan otros criterios: examinar desde el comienzo de la secuencia y/o hasta el final de esta.

Cuando ya tenemos escogida la operación con la que vamos a hacer el proceso de creación de una nueva vecindad dicha operación se extrae del vector secuencia y se prueba en las diferentes posiciones si corresponde, es decir, aplicando el mismo criterio de sólo examinar combinaciones de secuencias que no resulten redundantes. Se almacena la mejor secuencia y el valor que da de la función objetivo y se vuelve a realizar este proceso con otro vecino de la operación que asignamos en la primera parte del método. Para la realización de este nuevo proceso hay que tener en cuenta que el parámetro "p" cambia si la vecindad generada ha conseguido mejorar la solución previa . Se redefine "p" como la posición nueva donde se ha reinsertado la componente que habíamos extraído.

Cuando se terminan de insertar las *n*m* operaciones que tiene el problema en la secuencia, pasando cada una de ellas por todo el proceso explicado, se termina el algoritmo. Se ilustra este proceso descrito en forma de pseudocódigo en la Figura 4.5.

4.2.5 BICH-MIH algorithm

Bounded Insertion Constructive Heuristic (BICH)

Propuesta por Fernandez-Viagas y Framiñan (2015) inicialmente para un problema de flujo regular *(FSS, flowshop)* con permutación, con función objetivo Makespan y sujeto a la restricción de *maximun Tardiness*. Lo que persigue esta heurística es puntuar a través de un indicador los trabajos posibles a introducir en la siguiente posición de la secuencia. Primero hay que elegir la máquina en la que se va a introducir la siguiente operación, siendo esta la que menor tiempo de disponibilidad tenga, en otras palabras, donde el trabajo pueda comenzar a ejecutarse antes. Cabe resaltar que en caso de empate se escogerá la máquina con índice menor, es decir, entre la máquina uno y la máquina dos se elegiría la número uno.

Con la máquina escogida ahora tenemos que aplicar el indicador para decidir el trabajo que se insertará en esa máquina. El indicador es mejor cuanto menor sea su valor por lo que entre todas las evaluaciones de los trabajos elegiremos el que nos aporte el mínimo. Este criterio que vamos a seguir para asignar los trabajos es la suma de dos factores:

- El valor de la función objetivo que resulte de evaluar la secuencia una vez introducido el trabajo que se esté probando en ella. En el pseudocódigo de la figura 4.2.5.1 se aprecia que la función objetivo es *TCT*, entre la línea siete y línea ocho, pero en nuestro caso también analizaremos el *Makespan*.
- El *Lower Bound* (*LB*) que consiste en la suma de los tiempos de proceso de las operaciones no asignadas a la secuencia, eliminando de esta suma el trabajo que se está probando en la secuencia.

Una vez probados todos los trabajos que quedasen libres para asignar a la máquina elegida al inicio, simplemente se introduce en la secuencia la operación que menor valor del indicador haya obtenido y volvemos a empezar el método.

Minimal Idleness Heuristic (MIH)

Este algoritmo consiste en el mismo concepto que el anterior salvo la excepción de que el indicador que se usa evalúa el tiempo *idle*, es decir, el tiempo ocioso. El criterio para elegir la máquina sobre la que se elegirá el trabajo a introducir es el mismo que en la heurística anterior pero lo que cambia en este algoritmo es que el criterio de selección del trabajo se basa en el tiempo ocioso de las máquinas.

Cuando el trabajo que se propone introducir en la máquina ha terminado de realizarse en otra máquina en un tiempo más tarde que el momento de disponibilidad de la máquina donde se quiere insertar ahora, aparece idle-time. En cualquier otro caso el idle-time sería cero, es decir, no existiría. Matemáticamente, se representa:

$$\phi = \begin{cases} J_j - M_i & si \quad J_j > M_i \\ 0 & si \quad J_j \leq M_i \end{cases} \tag{4.1}$$

BICH-MIH

La última heurística, denominada *BICH-MIH*, es una combinación de los dos métodos anteriores. Básicamente consiste en la media ponderada de los criterios para la elección del trabajo explicados en los dos algoritmos anteriores. Para ello, llamamos a la función objetivo Phi (Figura 3.1) , que se define como uno menos la constante de ponderación Alpha, α, multiplicando a la función objetivo del algoritmo BICH, el valor de la función objetivo más el valor del *Lower Bound*; sumado esto a la multiplicación de α por el valor de la función objetivo del algoritmo *MIH*, es decir, el valor del *idle-time* si lo hubiera.

Para nuestro proyecto analizaremos cómo se comporta este algoritmo para valores de Alpha cero, cero y medio y por último uno, por tanto, se generan tres problemas distintos cada vez que se modela este problema para ambas funciones objetivo, minimización del *makespan* y minimización del *total completion time*.

Tal y como expresa en la línea 9 del pseudocódigo de la Figura 4.6, el trabajo que se adjudica a la máquina elegida de entre todos los posibles es el que nos dé el valor mínimo cuando lo evaluamos en la expresión algebraica Phi (ϕ) mencionada anteriormente. Y es evidente pero el criterio de selección de la máquina se mantiene igual que en ambos algoritmos que componen el método.

Data: TCT(.), LB(.), p, $setup$, α
Result: A sequence $\Pi := (\pi_{11}, \pi_{12}, ..., \pi_{mn})$

1 $\Pi \leftarrow \emptyset$;
2 $P \leftarrow$ copy(p);
3 $Setup \leftarrow$ copy($setup$);
4 $M \leftarrow$ list with time cumulative in each machine;
5 $J \leftarrow$ list with time cumulative in each job;
6 $\Omega_k \leftarrow$ list with the jobs allocated to machine k, $\forall k \in \{1, ..., m\}$;
7 **while** $\|\Pi\| < n \times m$ **do**
8 machine $k \leftarrow \underset{i \in \{1,...,m\}}{\text{argmin}}\ M_i$; // index from machine finishing first.
9 job $j \leftarrow \underset{j \in \{1,...,n\}, j \notin \Omega_k}{\text{argmin}}\ \Psi_{kj}$; // the best job not allocated in machine k, through the combined approach rule with α.
10 $\Pi \leftarrow \Pi \cup \{\pi_{kj}\}$;
11 $\Omega_k \leftarrow \Omega_k \cup \{j\}$;
12 $P_{kj} \leftarrow 0$;
13 update J and M with time of π_{kj} operation;
14 **end**

Algorithm 5: Pseudo-code of the BICH-MIH heuristic

Figura 4.6 Pseudocódigo algoritmo *BICH-MIH*. [Fuente: [4]].

5 Batería de problemas y tratamiento de resultados

Para programar los pseudocódigos de las heurísticas que están propuestos en [2] y [4] se ha utilizado el software *Visual Studio 2019* [5], dicho software es gratuito y nos ofrece la posibilidad de enfrentarnos a múltiples ámbitos de la programación, existiendo numerosos tipos de proyectos posibles para crear.

En nuestro proyecto el enfoque que le hemos dado a este software ha sido para crear una aplicación de consola (*.NET Core*) y el lenguaje de programación usado es C#, que he aprendido con el soporte de las referencias [6] y [7]. Las librerías usadas son entre otras System.Linq, System.Globalisation y System.IO.

Por último respecto al tratamiento de resultados, el valor que devuelven las funciones de todas las heurísticas es el valor de cada función objetivo. Además de ello, antes de cada heurística hay que declarar un contador de tiempo que empieza a contar antes de comenzar cada método y al terminar de desarrollarse guardar el valor de este contador en un fichero que explicaremos a continuación para conocer el tiempo de computación.

5.1 Generación de instancias

Se ha decidido desarrollar una batería de diez instancias para cada tamaño de problemas desde n=m=3 hasta n=m=15; se ha decidido usar este tipo de casos donde el número de trabajos y máquinas coinciden porque así es como se plantea en los modelos que tomamos de referencia del artículo [2]. Una instancia consiste en generar la matriz de tiempos de proceso de tamaño [n,m] que será junto con el número de trabajos y de máquinas los únicos valores de entrada de los métodos. Para obtener los tiempo de proceso de cada instancia se programa un bucle que rellena con valores aleatorios, entre 1 y 99, dicha matriz (P_{ij}), simplemente teniendo las dimensiones de esta, es decir, el tamaño del problema, como dato. Se crea un bucle que vaya aumentando en una unidad el tamaño del problema cada vez que se hayan ejecutado las diez instancias correspondientes a cada tamaño en todos los algoritmos. En la Figura 5.1 se puede visualizar la explicación anterior:

En el Anexo (capítulo 8), en particular en la función *main* del código C#, se puede ver este segmento de código con líneas de comandos adicionales que corresponden a la obtención de los resultados y tiempos computacionales y posterior inserción en un fichero Excel que se desarrollará más adelante en este capítulo. Para ilustrar el concepto de la generación de instancias se ha decidido omitir dicha parte ya que lo significativo es mostrar el proceso iterativo mediante el que se generan las instancias y se evalúan en los métodos propuestos.

5.2 Ficheros .txt y visualizador diagramas de Gantt

Para la interpretación gráfica de las soluciones se usa el software AVPS , [8], que dibuja un Diagrama de Gantt partiendo de un archivo .txt en el que están escritos el número de trabajos y de máquinas junto a un

```
for (n = 3; n < 16; n++)
{
    m = n;
    for (int k = 0; k < 10; k++)
    {

        int[,] processingtimes = new int[n, m];

        Random aleatorio = new Random();
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                processingtimes[i, j] = aleatorio.Next(1, 99);
            }
        }

        int TCT1 = irh1(n, m, processingtimes);
        double timeActualMilliseconds = timer.ElapsedMilliseconds;

        int Makespan1 = irh1makespan(n, m, processingtimes);

        int TCT2 = irh2(n, m, processingtimes);
        int Makespan2 = irh2makespan(n, m, processingtimes);

        int TCT3 = irh3(n, m, processingtimes);
        int Makespan3 = irh3makespan(n, m, processingtimes);

        int TCT4 = irh4(n, m, processingtimes);
        int Makespan4 = irh4makespan(n, m, processingtimes);


        double alpha = 0;
        for (int k1 = 0; k1 < 3; k1++)
        {
            int TCT5 = BICH_MIH_TCT(n, m, processingtimes, alpha);
            int Makespan5 = BICH_MIH_Makespan(n, m, processingtimes, alpha);
            alpha=alpha+0.5;
        }
    }
}
```

Figura 5.1 Generación de instancias. [Fuente: Elaboración propia].

número uno en la primera fila del archivo mencionado, a continuación, se escriben las matrices de tiempos de proceso y de tiempos de terminación traspuestas, insertando un salto de línea entre ambas.
La declaración de un fichero .txt para escribir en él desde el proyecto de Visual Studio se hace con las líneas de código que se muestran en la Figura 5.2. Para escribir en este fichero desde *Visual Studio* se hace con el comando FicheroTXT.Write(“”) o si se quiere dar un salto de línea se usaría FicheroTXT.WriteLine(“”). Quedaría algo similar a lo que se muestra en la Figura 5.3.

Con esto podemos ir al visualizador de diagramas de Gantt y seleccionar Herramientas → Conversor Producción en las pestañas de la esquina superior izquierda de la ventana de la aplicación. Se abrirá un desplegable en el que se selecciona el archivo .txt que vamos a utilizar. Esta ventana y la configuración utilizada se vería en la Figura 5.4.

Introduciendo el fichero .txt que deseemos visualizar pulsando en Abrir debajo de Seleccionar Fichero y configurando las opciones como se muestra en la Figura 5.4: “Sin Rutas” y “Escalado” con “Último instante programación (minutos): 1440”. Este programa se ha usado para visualizar ejemplos de instancias con tamaños de n y m pequeños y comprobar el resultado gráfico dado con un desarrollo manual de la heurística.

```
String ruta = AppDomain.CurrentDomain.BaseDirectory;
ruta = ruta.Replace("\\", "/").ToString();
DirectoryInfo DIRESC = new DirectoryInfo(ruta + "/temp");
if (!DIRESC.Exists) DIRESC.Create();

String ficBatBorrar = ruta + "temp" + "/resultados.txt";
FileInfo FicheroEliminar = new FileInfo(ficBatBorrar);
if (FicheroEliminar.Exists) File.Delete(ficBatBorrar);
String ficTXT = ruta + "temp" + "/resultados.txt";
StreamWriter FicheroTXT = new StreamWriter(ficTXT, true);
```

Figura 5.2 Creación fichero .txt. [Fuente: Elaboración propia].

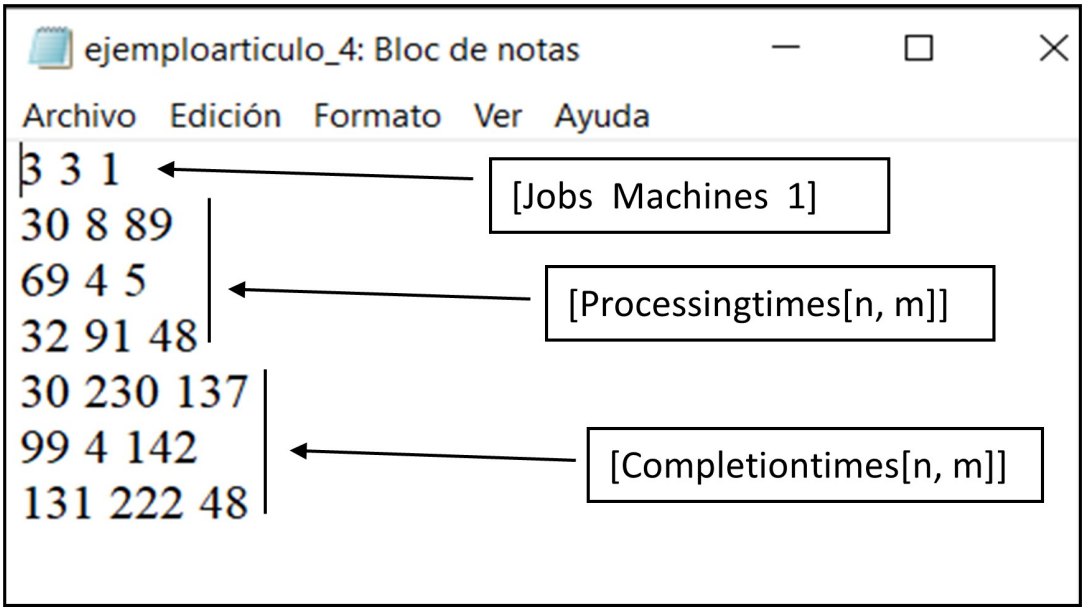

Figura 5.3 Ejemplo de un fichero .txt. [Fuente: Elaboración propia].

A continuación se muestran dos ejemplos de la visualización de la secuencia solución a partir de los archivos .txt mencionados que corresponden a un problema de cuatro trabajos y cuatro máquinas con función objetivo minimización del *makespan* solucionados mediante la heurística *IRH1* e *IRH3* respectivamente, Figura 5.5 y Figura 5.6.

5.3 Ficheros .csv y herramienta Excel

De esta forma se genera un archivo tipo .csv (Figura 5.7) que al abrirlo nos muestra una tabla en la herramienta Excel. Para escribir en este fichero se usa al igual que en el apartado previo la orden FicheroCSV.Write; es importante hacer énfasis en la forma en la que se escribe en la tabla de manera que escribir en el comando FicheroCSV.Write (“;”) hará saltar el cursor de escritura a la siguiente casilla de la fila en la que se encuentre y si escribimos FicheroCSV.Write(“\n”) cambiamos de fila.

Entonces lo que se propone es generar los resultados de tiempo y de valor de función objetivo para la batería de instancia explicada en el apartado 5.1 de manera que resulte una tabla de tamaño con filas igual al número de instancias a generar, ciento treinta (diez casos por cada tamaño de problema multiplicado por trece tamaños distintos), y con número de columnas igual a veintiocho, es decir, el número de métodos que hay

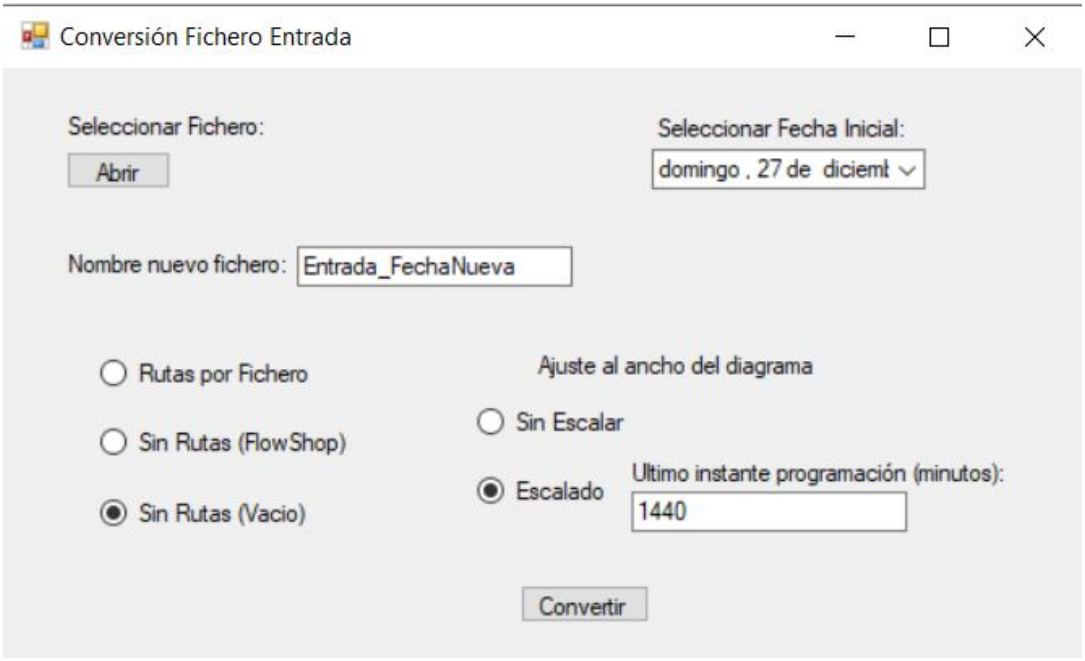

Figura 5.4 Ventana del visualizador de diagramas de Gantt. [Fuente: [8]].

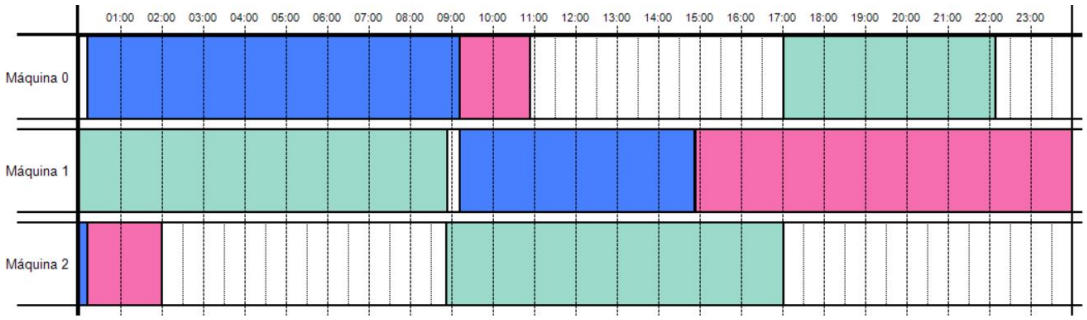

Figura 5.5 Ejemplo diagrama de gant para modelo resulto con IRH1. [Fuente: Elaboración propia].

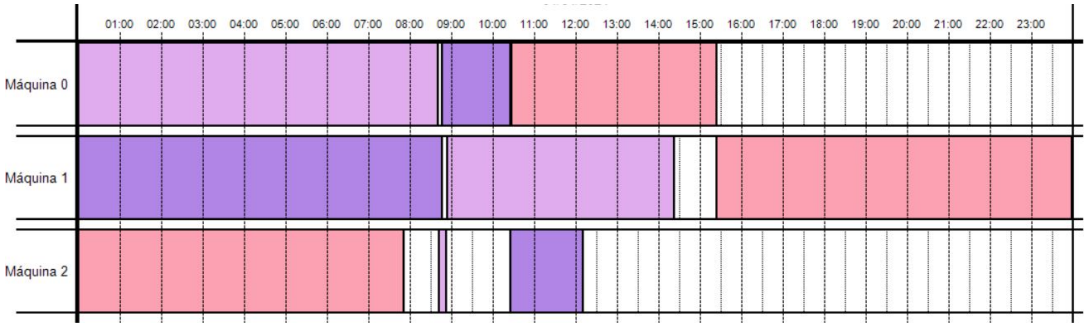

Figura 5.6 Ejemplo diagrama de gant para modelo resulto con IRH3. [Fuente: Elaboración propia].

por cuatro columnas por cada método, ya que evaluamos dos funciones objetivo y por cada función objetivo evaluada se muestra el valor del objetivo y el tiempo de computación del método. Las tablas de resultados mencionadas se expondrán en el capítulo siguiente en cada sección correspondiente.

```
String ruta = AppDomain.CurrentDomain.BaseDirectory;
ruta = ruta.Replace("\\", "/").ToString();
DirectoryInfo DIRESC = new DirectoryInfo(ruta + "/temp");
if (!DIRESC.Exists) DIRESC.Create();

String ficBatBorrar = ruta + "temp" + "/tabla_excel1.csv";
FileInfo FicheroEliminar = new FileInfo(ficBatBorrar);
if (FicheroEliminar.Exists) File.Delete(ficBatBorrar);
String ficCSV = ruta + "temp" + "/tabla_excel1.csv";
StreamWriter FicheroCSV = new StreamWriter(ficCSV, true);
```

Figura 5.7 Creación fichero .csv. [Fuente: Elaboración propia].

6 Evaluación computacional

Todo el proceso para el análisis de datos parte del fichero .csv explicado en el apartado anterior y que muestra todos los datos que necesitamos conocer para poder hacer un estudio comparativo entre métodos y funciones objetivo que es básicamente lo que se propone en este punto.

Se analizarán por una parte los valores de las funciones objetivo, siendo la mejor solución siempre la que ofrezca el menor resultado ya que estamos en casos de minimización; adicionalmente se observa el coste computacional de todos los métodos en términos de tiempo de duración para la compilación de la heurística. El número de mejores resultados que ofrece es la forma en la que estudiaremos la calidad de los métodos en el contexto en el que están siendo aplicados. Cuando un método obtiene el mínimo valor entre el resto de los algoritmos para un determinado tamaño de problema se le suma uno a la cantidad de mejores resultados que lleva dicho método, de forma que el máximo valor que puede adoptar esta variable sería ciento treinta, es decir, la cantidad de tamaños de nuestra muestra. Recalcando el concepto, más de un método puede conseguir tener el mejor resultado para un tamaño de problema si sus resultados coinciden y son el mínimo entre todos.

Para poder comparar la eficacia y la precisión de todos los métodos entre sí se usa el indicador ARPD, *Average Relative Percentage Deviation*, que se define como la desviación promedia de los datos expresada en porcentaje. En el nuestro entorno del problema es útil para concluir la distancia que hay entre las soluciones resultado de los métodos y las mejores soluciones, esto se analiza de manera independiente para cada instancia propuesta y finalmente se calcula el promedio de estas desviaciones. La fórmula queda definida como:

$$[ARPD]_{ij} = ([FO]_{ij} - [FO]_{best}) * 100/[FO]_{best} \tag{6.1}$$

Donde i denota el número de la instancia, la variable j el método al que hacemos referencia y FO_best es el mejor valor obtenido de la función objetivo para ese caso a examinar.
En los siguientes apartados se detallan los resultados de los puntos anteriores dividiendo los datos por funciones objetivo y tamaño del problema.

6.1 Evaluación para FO: *Total completion time*

En la Tabla 6.1 se muestran los valores de la evaluación de los siete métodos propuestos para todas las instancias generadas en el caso de la función objetivo: minimización de la suma de los tiempos de terminación de los trabajos (Min $\sum C_j$).

Tabla 6.1 Resultados de función objetivo para caso *total completion time(TCT)* . [Fuente: Elaboración propia].

Valores Función objetivo TCT							
$n = m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct} = 0$	$ALPHA_{tct} = 0.5$	$ALPHA_{tct} = 1$
3	291	275	275	275	380	288	288
3	312	316	312	316	628	316	316
3	389	410	389	410	493	439	439
3	350	356	337	356	403	356	356
3	520	520	488	520	631	546	546
3	404	388	388	388	638	438	438
3	421	437	421	437	604	446	446
3	494	489	461	445	635	501	501
3	293	337	293	320	531	390	390
3	703	698	666	673	930	666	666
4	849	819	806	833	1083	956	956
4	968	1027	971	991	1635	1004	1004
4	811	743	743	743	1378	904	904
4	847	811	811	840	1025	1130	1130
4	818	820	806	820	1537	926	926
4	947	941	871	875	1511	1014	1014
4	865	932	825	849	1057	876	876
4	753	673	654	673	1206	747	747
4	900	975	913	884	1504	1052	1052
4	1063	1034	1018	1016	1735	1154	1154
5	1304	1235	1235	1235	2090	1544	1544
5	1477	1532	1469	1541	2473	1608	1608
5	1116	1089	1061	1075	2138	1045	1045
5	1508	1378	1378	1440	2696	1686	1686
5	1567	1521	1472	1521	2312	1584	1584
5	1335	1315	1315	1300	2292	1361	1361
5	1278	1217	1201	1200	1956	1244	1244
5	1340	1372	1344	1372	2648	1558	1558
5	1309	1313	1280	1286	2065	1424	1424
5	1417	1341	1318	1368	2213	1496	1496
6	2198	2058	2057	2070	4365	2260	2260
6	1945	1882	1850	1882	3731	2214	2214
6	2088	1957	1912	1886	3467	2057	2057
6	1811	1786	1738	1747	3083	1915	1915
6	2095	1971	1943	1953	3939	2116	2116
6	1955	1767	1739	1736	4132	1956	1956
6	1670	1644	1614	1663	3005	2055	2055
6	1855	1771	1754	1820	3037	1871	1871
6	2390	2230	2211	2187	4215	2355	2355
6	2528	2535	2441	2479	4437	2589	2589
7	2919	2773	2732	2708	5415	3000	3000
7	3022	2937	2894	2852	5258	3130	3130
7	2844	2649	2552	2652	5350	2764	2764
7	2878	2727	2727	2817	5219	2843	2843
7	2756	2596	2518	2633	5711	3062	3062
7	2808	2697	2666	2639	4696	2785	2785
7	2413	2322	2286	2448	5258	2821	2821
7	2759	2811	2745	2855	5345	3111	3111
7	2915	2852	2786	2791	6326	3427	3427
7	2908	2806	2770	2741	5417	3058	3058
8	3366	3251	3233	3299	6857	3457	3457
8	4453	4167	4080	4149	7923	4343	4343

Sigue en la página siguiente.

$n = m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct} = 0$	$ALPHA_{tct} = 0.5$	$ALPHA_{tct} = 1$
8	3755	3520	3436	3513	8636	4134	4134
8	3740	3622	3596	3617	8130	4096	4096
8	3887	3627	3560	3594	8018	3761	3761
8	3864	3654	3571	3659	8260	4183	4183
8	3792	3568	3510	3658	7919	4317	4317
8	3736	3569	3567	3573	6876	4291	4291
8	3882	3699	3627	3731	7165	3701	3701
8	3517	3370	3321	3263	7917	3720	3720
9	4476	4238	4226	4259	10433	5078	5078
9	4842	4553	4464	4671	7947	5119	5119
9	4550	4196	4174	4173	9752	4702	4702
9	4868	4360	4243	4381	9778	4533	4533
9	4837	4441	4311	4267	10218	5285	5285
9	4688	4445	4331	4414	7935	5104	5104
9	4127	3898	3897	3869	9549	4253	4253
9	4409	4465	4416	4333	9479	4743	4743
9	4811	4719	4604	4711	9504	4923	4923
9	4787	4537	4509	4507	10428	5248	5248
10	5783	5592	5534	5575	13316	5967	5967
10	5925	5728	5467	5497	12069	6448	6448
10	6337	6017	5898	5942	14458	6622	6622
10	6210	5793	5719	5606	13659	6406	6406
10	6301	6067	5949	6022	12999	6980	6980
10	5969	5724	5572	5637	14445	6340	6340
10	5841	5617	5575	5709	13883	6021	6021
10	5305	5296	5141	5228	11540	5699	5699
10	5523	5376	5257	5337	12914	5897	5897
10	5685	5670	5536	5578	14148	6806	6806
11	7384	6952	6881	6955	16676	7664	7664
11	6570	6264	6195	6060	14277	7212	7212
11	6759	6213	6091	6244	15791	6709	6709
11	6556	6203	6131	6137	15219	6540	6540
11	7477	6575	6555	6692	16652	7343	7343
11	6817	6363	6299	6332	14789	7097	7097
11	7341	6941	6782	6836	16194	8177	8177
11	7046	6671	6669	6780	17582	7363	7363
11	6779	6409	6276	6406	15640	7000	7000
11	6652	6392	6330	6323	16250	7041	7041
12	8121	7695	7628	7599	23196	8843	8843
12	8666	8165	8161	8067	20595	9131	9131
12	7537	7342	7187	7181	21933	8269	8269
12	8884	8509	8254	8292	18292	9025	9025
12	7838	7374	7279	7282	19356	9046	9046
12	8117	7603	7510	7595	21119	8776	8776
12	8211	7697	7648	7723	21069	8302	8302
12	8116	7725	7629	7634	18069	8594	8594
12	8208	7644	7461	7738	20524	8767	8767
12	8434	7996	7773	7938	21680	8503	8503
13	9203	8637	8629	8766	26650	9925	9925
13	9759	9101	8972	9094	24291	10896	10896
13	9904	9453	9213	9390	24503	10173	10173
13	9367	8677	8620	8718	24880	9429	9429
13	10499	9525	9460	9577	27772	10696	10696
13	9054	8598	8537	8601	20117	9968	9968
13	9831	9189	9135	9150	24954	9859	9859

Sigue en la página siguiente.

$n=m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct}=0$	$ALPHA_{tct}=0.5$	$ALPHA_{tct}=1$
13	10573	9763	9660	9831	24446	11139	11139
13	9880	9057	8943	8859	24001	10225	10225
13	9490	8557	8434	8740	20509	9622	9622
14	12662	11041	10908	11045	31737	12061	12061
14	11744	11194	10879	11006	30292	12269	12269
14	11731	10737	10614	10566	29681	13345	13345
14	10745	10243	10014	10208	31477	12105	12105
14	12165	11117	10934	11026	32562	11796	11796
14	12194	10690	10546	10603	30241	11838	11838
14	11465	10345	10304	10362	32818	12285	12285
14	11053	10096	9984	9813	28426	12139	12139
14	11404	10120	10019	10256	30851	11931	11931
14	11557	10727	10548	10712	28826	12855	12855
15	13280	12165	11983	12110	33617	13092	13092
15	13589	12409	12338	12321	35895	14398	14398
15	13301	12368	12257	12387	33576	13815	13815
15	12844	11894	11726	11830	33596	13238	13238
15	14024	12728	12660	12866	35623	15893	15893
15	13426	12434	12319	12325	32315	14704	14704
15	13668	12350	12280	12301	38685	14091	14091
15	12347	11520	11188	11424	31851	13299	13299
15	13643	12976	12768	12843	37510	14433	14433
15	13424	12608	12302	12522	36379	15078	15078

Se aprecia que los últimos tres métodos, obtienen valores relativamente altos, respecto al resto de métodos, para la función objetivo a estudiar. Los resultados de estos algoritmos empeoran conforme aumenta el tamaño de los problemas (en el eje horizontal se muestra el número de trabajos que coincide con el número de máquinas de la instancia) debido a la baja complejidad de los algoritmos propiamente y conforme se enfrentan a problemas más grandes lo estadísticamente probable es que no encuentren buenas soluciones en el campo tan pequeño que analizan.
Estos métodos corresponden al modelo *BICH-MIH* y el nombre ALPHA hace referencia al parámetro modificable α dentro de este algoritmo. Se usará esta notación en los gráficos en adelante.

6.1.1 Número de mejores soluciones

El número de mejores soluciones que ha obtenido cada método queda representado en la Tabla 6.2.

Tabla 6.2 Mejores soluciones para función objetivo TCT. [Fuente: Elaboración propia].

Mejores soluciones para función objetivo TCT							
$n=m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct}=0$	$ALPHA_{tct}=0.5$	$ALPHA_{tct}=1$
3	0	1	1	1	0	0	0
3	1	0	1	0	0	0	0
3	1	0	1	0	0	0	0
3	0	0	1	0	0	0	0
3	0	0	1	0	0	0	0
3	0	1	1	1	0	0	0
3	1	0	1	0	0	0	0
3	0	0	0	1	0	0	0
3	1	0	1	0	0	0	0
3	0	0	1	0	0	1	1
4	0	0	1	0	0	0	0

Sigue en la página siguiente.

$n = m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct} = 0$	$ALPHA_{tct} = 0.5$	$ALPHA_{tct} = 1$
4	1	0	0	0	0	0	0
4	0	1	1	1	0	0	0
4	0	1	1	0	0	0	0
4	0	0	1	0	0	0	0
4	0	0	1	0	0	0	0
4	0	0	1	0	0	0	0
4	0	0	1	0	0	0	0
4	0	0	0	1	0	0	0
4	0	0	0	1	0	0	0
5	0	1	1	1	0	0	0
5	0	0	1	0	0	0	0
5	0	0	0	0	0	1	1
5	0	1	1	0	0	0	0
5	0	0	1	0	0	0	0
5	0	0	0	1	0	0	0
5	0	0	0	1	0	0	0
5	1	0	0	0	0	0	0
5	0	0	1	0	0	0	0
5	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0
6	0	0	0	1	0	0	0
6	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0
6	0	0	0	1	0	0	0
6	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0
6	0	0	0	1	0	0	0
6	0	0	1	0	0	0	0
7	0	0	0	1	0	0	0
7	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0
7	0	1	1	0	0	0	0
7	0	0	1	0	0	0	0
7	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0
7	0	0	1	0	0	0	0
7	0	0	1	0	0	0	0
7	0	0	0	1	0	0	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	0	0	0	1	0	0	0
9	0	0	1	0	0	0	0
9	0	0	1	0	0	0	0
9	0	0	0	1	0	0	0
9	0	0	1	0	0	0	0
9	0	0	0	1	0	0	0
9	0	0	1	0	0	0	0

Sigue en la página siguiente.

$n=m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct}=0$	$ALPHA_{tct}=0.5$	$ALPHA_{tct}=1$
9	0	0	0	1	0	0	0
9	0	0	0	1	0	0	0
9	0	0	1	0	0	0	0
9	0	0	0	1	0	0	0
10	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0
10	0	0	0	1	0	0	0
10	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	0	0	0	1	0	0	0
11	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	0	0	0	1	0	0	0
12	0	0	0	1	0	0	0
12	0	0	0	1	0	0	0
12	0	0	0	1	0	0	0
12	0	0	1	0	0	0	0
12	0	0	1	0	0	0	0
12	0	0	1	0	0	0	0
12	0	0	1	0	0	0	0
12	0	0	1	0	0	0	0
12	0	0	1	0	0	0	0
12	0	0	1	0	0	0	0
13	0	0	1	0	0	0	0
13	0	0	1	0	0	0	0
13	0	0	1	0	0	0	0
13	0	0	1	0	0	0	0
13	0	0	1	0	0	0	0
13	0	0	1	0	0	0	0
13	0	0	1	0	0	0	0
13	0	0	1	0	0	0	0
13	0	0	0	1	0	0	0
13	0	0	1	0	0	0	0
14	0	0	1	0	0	0	0
14	0	0	1	0	0	0	0
14	0	0	0	1	0	0	0
14	0	0	1	0	0	0	0
14	0	0	1	0	0	0	0
14	0	0	1	0	0	0	0
14	0	0	1	0	0	0	0
14	0	0	0	1	0	0	0
14	0	0	1	0	0	0	0
14	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0

Sigue en la página siguiente.

$n=m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct}=0$	$ALPHA_{tct}=0.5$	$ALPHA_{tct}=1$
15	0	0	0	1	0	0	0
15	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0

Para esta función objetivo vemos que hay muchos algoritmos que ni siquiera han conseguido una mejor solución en ningún tamaño de problema. Es importante ver que el método $\alpha = 1$ solo ha conseguido un mejor resultado y ocurre para el caso de n=m=3, problema de baja complejidad. Por otro lado, los mejores métodos son la *IRH3* e *IRH4* que entre ambos obtienen las mejores soluciones en casi todos los casos.

El número final de mejores soluciones obtenidas para cada método en este caso de función objetivo se representa en la Tabla 6.3

Tabla 6.3 Número de Mejores Soluciones *Total Completion Time*. [Fuente: Elaboración propia].

Número de Mejores Soluciones *Total Completion Time*

$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA=0$	$ALPHA=0{,}5$	$ALPHA=1$
6	7	99	32	0	2	2

6.1.2 CPU times

Los *CPU times* de los métodos para la función objetivo "minimización de la suma de los tiempos de terminación" se muestran en la Tabla 6.4 y reflejan la complejidad del algoritmo de manera indirecta porque representan el tiempo que le toma a cada método en abordar el problema y dar una solución:

Tabla 6.4 Tiempo computacional para el caso *Total Completion Time*. [Fuente: Elaboración propia].

Tiempo computacional para el caso *Total Completion Time*

$n=m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak}=0$	$ALPHA_{mak}=0.5$	$ALPHA_{mak}=1$
3	0,003	0,001	0,002	0,001	0,001	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000
4	0,000	0,000	0,000	0,001	0,000	0,000	0,000
4	0,000	0,000	0,000	0,001	0,000	0,000	0,000
4	0,000	0,000	0,000	0,001	0,000	0,000	0,000
4	0,000	0,000	0,000	0,001	0,000	0,000	0,000
4	0,000	0,000	0,000	0,003	0,000	0,000	0,000
4	0,000	0,000	0,000	0,001	0,000	0,000	0,000
4	0,000	0,000	0,000	0,001	0,000	0,000	0,000
4	0,000	0,000	0,000	0,002	0,000	0,000	0,000
4	0,000	0,000	0,000	0,001	0,000	0,000	0,000
4	0,000	0,000	0,000	0,001	0,000	0,000	0,000

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
5	0,000	0,000	0,001	0,007	0,000	0,000	0,000
5	0,000	0,000	0,001	0,005	0,000	0,000	0,000
5	0,000	0,000	0,001	0,005	0,000	0,000	0,000
5	0,000	0,000	0,001	0,005	0,000	0,000	0,000
5	0,000	0,000	0,001	0,006	0,000	0,000	0,000
5	0,000	0,001	0,001	0,005	0,000	0,000	0,000
5	0,000	0,000	0,001	0,006	0,000	0,000	0,000
5	0,000	0,000	0,001	0,007	0,000	0,000	0,000
5	0,000	0,000	0,001	0,007	0,000	0,000	0,000
5	0,000	0,000	0,001	0,006	0,000	0,000	0,000
6	0,000	0,003	0,003	0,022	0,000	0,000	0,000
6	0,000	0,002	0,005	0,021	0,000	0,000	0,000
6	0,000	0,002	0,006	0,018	0,000	0,000	0,000
6	0,000	0,002	0,005	0,019	0,000	0,000	0,000
6	0,000	0,002	0,003	0,021	0,000	0,000	0,000
6	0,000	0,002	0,003	0,025	0,000	0,000	0,000
6	0,000	0,002	0,003	0,020	0,000	0,000	0,000
6	0,000	0,002	0,006	0,020	0,000	0,000	0,000
6	0,000	0,002	0,006	0,019	0,000	0,000	0,000
6	0,000	0,002	0,003	0,020	0,000	0,000	0,000
7	0,000	0,006	0,015	0,112	0,000	0,000	0,000
7	0,000	0,005	0,007	0,054	0,000	0,000	0,000
7	0,000	0,009	0,017	0,056	0,000	0,000	0,000
7	0,000	0,005	0,008	0,065	0,000	0,000	0,000
7	0,000	0,007	0,013	0,063	0,000	0,000	0,000
7	0,000	0,007	0,009	0,054	0,000	0,000	0,000
7	0,000	0,005	0,013	0,064	0,000	0,000	0,000
7	0,000	0,006	0,009	0,061	0,000	0,000	0,000
7	0,000	0,006	0,009	0,067	0,000	0,000	0,000
7	0,000	0,006	0,010	0,058	0,000	0,000	0,000
8	0,001	0,015	0,024	0,141	0,000	0,000	0,000
8	0,001	0,014	0,023	0,147	0,000	0,000	0,000
8	0,001	0,013	0,031	0,139	0,000	0,000	0,000
8	0,001	0,015	0,020	0,152	0,000	0,000	0,000
8	0,001	0,016	0,020	0,142	0,000	0,000	0,000
8	0,001	0,015	0,020	0,140	0,000	0,000	0,000
8	0,001	0,015	0,023	0,139	0,000	0,000	0,000
8	0,001	0,015	0,023	0,140	0,000	0,000	0,000
8	0,001	0,013	0,021	0,140	0,000	0,000	0,000
8	0,001	0,016	0,021	0,149	0,000	0,000	0,000
9	0,002	0,033	0,049	0,323	0,000	0,000	0,000
9	0,002	0,040	0,069	0,335	0,000	0,001	0,000
9	0,003	0,037	0,054	0,336	0,000	0,000	0,000
9	0,003	0,035	0,079	0,402	0,000	0,000	0,000
9	0,003	0,033	0,070	0,333	0,000	0,000	0,000
9	0,004	0,037	0,066	0,332	0,000	0,000	0,000
9	0,002	0,032	0,045	0,352	0,000	0,000	0,000
9	0,003	0,036	0,058	0,357	0,000	0,000	0,000
9	0,002	0,032	0,042	0,332	0,000	0,000	0,000
9	0,003	0,039	0,075	0,340	0,000	0,000	0,000
10	0,005	0,070	0,091	0,744	0,001	0,001	0,001
10	0,004	0,077	0,143	0,747	0,000	0,000	0,001
10	0,004	0,077	0,137	0,727	0,000	0,000	0,000
10	0,004	0,067	0,112	0,689	0,000	0,000	0,000
10	0,004	0,069	0,109	0,620	0,000	0,000	0,000

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
10	0,003	0,069	0,113	0,641	0,000	0,000	0,000
10	0,005	0,066	0,094	0,633	0,000	0,000	0,000
10	0,004	0,068	0,154	0,641	0,000	0,000	0,000
10	0,004	0,065	0,125	0,648	0,000	0,000	0,000
10	0,005	0,071	0,140	0,680	0,000	0,000	0,000
11	0,006	0,125	0,190	1,293	0,001	0,001	0,000
11	0,008	0,129	0,167	1,339	0,001	0,001	0,001
11	0,007	0,132	0,256	1,193	0,001	0,001	0,001
11	0,006	0,125	0,230	1,370	0,000	0,000	0,001
11	0,007	0,123	0,157	1,270	0,001	0,001	0,001
11	0,007	0,127	0,156	1,265	0,001	0,001	0,000
11	0,006	0,154	0,194	1,240	0,001	0,001	0,001
11	0,009	0,133	0,196	1,350	0,001	0,001	0,001
11	0,013	0,143	0,323	1,472	0,001	0,001	0,001
11	0,009	0,149	0,252	1,321	0,001	0,001	0,001
12	0,011	0,238	0,343	2,429	0,001	0,001	0,001
12	0,009	0,240	0,355	2,490	0,001	0,001	0,001
12	0,011	0,234	0,394	2,288	0,002	0,001	0,001
12	0,010	0,233	0,305	2,414	0,002	0,001	0,001
12	0,010	0,249	0,320	2,296	0,001	0,001	0,001
12	0,010	0,231	0,330	2,284	0,001	0,001	0,001
12	0,011	0,240	0,342	2,264	0,001	0,002	0,001
12	0,011	0,250	0,306	2,127	0,001	0,001	0,001
12	0,011	0,251	0,390	2,743	0,001	0,001	0,003
12	0,014	0,272	0,467	2,315	0,002	0,001	0,002
13	0,017	0,401	0,544	4,044	0,003	0,003	0,002
13	0,016	0,428	0,539	4,096	0,002	0,002	0,002
13	0,015	0,402	0,726	3,962	0,002	0,002	0,002
13	0,014	0,426	0,579	3,830	0,002	0,002	0,002
13	0,014	0,433	0,511	4,254	0,002	0,002	0,002
13	0,015	0,415	0,569	4,606	0,003	0,002	0,002
13	0,019	0,472	0,531	4,306	0,002	0,003	0,002
13	0,016	0,393	0,731	3,892	0,001	0,003	0,003
13	0,014	0,402	0,492	4,054	0,001	0,002	0,002
13	0,014	0,413	0,508	3,995	0,002	0,002	0,002
14	0,025	0,691	0,857	7,183	0,002	0,002	0,003
14	0,021	0,687	0,941	7,019	0,002	0,002	0,002
14	0,022	0,680	0,864	7,016	0,002	0,003	0,003
14	0,023	0,729	1,220	6,800	0,003	0,004	0,003
14	0,021	0,674	0,855	6,749	0,002	0,003	0,002
14	0,022	0,671	0,851	6,726	0,003	0,003	0,003
14	0,022	0,654	0,815	7,183	0,003	0,003	0,003
14	0,025	0,720	0,780	6,956	0,003	0,004	0,005
14	0,026	0,713	0,866	6,539	0,003	0,002	0,003
14	0,021	0,690	0,754	6,642	0,002	0,003	0,003
15	0,033	1,095	1,474	10,307	0,004	0,004	0,004
15	0,032	1,072	1,468	10,577	0,004	0,005	0,004
15	0,032	1,095	1,378	10,480	0,005	0,005	0,004
15	0,038	1,072	1,680	11,178	0,004	0,004	0,005
15	0,030	1,103	1,181	10,771	0,003	0,005	0,004
15	0,030	1,038	1,431	10,713	0,004	0,004	0,004
15	0,031	1,074	1,298	10,812	0,004	0,004	0,004
15	0,031	1,050	1,501	10,544	0,004	0,004	0,004
15	0,031	1,067	1,402	10,812	0,005	0,004	0,004

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
15	0,031	1,058	1,508	10,686	0,004	0,004	0,004

Para mayor facilidad en el tratamiento de estos datos usaremos los promedios de los tiempos de computación teniendo así el marco global de la complejidad y del desarrollo de las heurísticas. Estos valores se adjuntan en la Tabla 6.5 se adjuntan adicionalmente las tablas a partir de las que salen estos datos donde se pueden ver los tiempos de computación seccionados por tamaño de problema.

Tabla 6.5 CPU Promedio *Total Completion Time*. [Fuente: Elaboración propia].

CPU Promedio *Total Completion Time*

$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA = 0$	$ALPHA = 0{,}5$	$ALPHA = 1$
0,0075	0,2068	0,2826	2,0467	0,0008	0,0009	0,0009

6.1.3 ARPD

Como ya se mencionó cuando se vieron los resultados de los valores de los métodos, los métodos finales son los que mayor ARPD tienen, es decir, que las soluciones que aportan distan mucho de la mejor obtenida en los problemas.Se observan los resultados en la Tabla 6.6.

Tabla 6.6 ARPDs para caso *Total Completion Time*. [Fuente: Elaboración propia].

ARPDs para caso *Total Completion Time*

$n = m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct} = 0$	$ALPHA_{tct} = 0.5$	$ALPHA_{tct} = 1$
3	5,82 %	0,00 %	0,00 %	0,00 %	38,18 %	4,73 %	4,73 %
3	0,00 %	1,28 %	0,00 %	1,28 %	101,28 %	1,28 %	1,28 %
3	0,00 %	5,40 %	0,00 %	5,40 %	26,74 %	12,85 %	12,85 %
3	3,86 %	5,64 %	0,00 %	5,64 %	19,58 %	5,64 %	5,64 %
3	6,56 %	6,56 %	0,00 %	6,56 %	29,30 %	11,89 %	11,89 %
3	4,12 %	0,00 %	0,00 %	0,00 %	64,43 %	12,89 %	12,89 %
3	0,00 %	3,80 %	0,00 %	3,80 %	43,47 %	5,94 %	5,94 %
3	11,01 %	9,89 %	3,60 %	0,00 %	42,70 %	12,58 %	12,58 %
3	0,00 %	15,02 %	0,00 %	9,22 %	81,23 %	33,11 %	33,11 %
3	5,56 %	4,80 %	0,00 %	1,05 %	39,64 %	0,00 %	0,00 %
4	5,33 %	1,61 %	0,00 %	3,35 %	34,37 %	18,61 %	18,61 %
4	0,00 %	6,10 %	0,31 %	2,38 %	68,90 %	3,72 %	3,72 %
4	9,15 %	0,00 %	0,00 %	0,00 %	85,46 %	21,67 %	21,67 %
4	4,44 %	0,00 %	0,00 %	3,58 %	26,39 %	39,33 %	39,33 %
4	1,49 %	1,74 %	0,00 %	1,74 %	90,69 %	14,89 %	14,89 %
4	8,73 %	8,04 %	0,00 %	0,46 %	73,48 %	16,42 %	16,42 %
4	4,85 %	12,97 %	0,00 %	2,91 %	28,12 %	6,18 %	6,18 %
4	15,14 %	2,91 %	0,00 %	2,91 %	84,40 %	14,22 %	14,22 %
4	1,81 %	10,29 %	3,28 %	0,00 %	70,14 %	19,00 %	19,00 %
4	4,63 %	1,77 %	0,20 %	0,00 %	70,77 %	13,58 %	13,58 %
5	5,59 %	0,00 %	0,00 %	0,00 %	69,23 %	25,02 %	25,02 %
5	0,54 %	4,29 %	0,00 %	4,90 %	68,35 %	9,46 %	9,46 %
5	6,79 %	4,21 %	1,53 %	2,87 %	104,59 %	0,00 %	0,00 %
5	9,43 %	0,00 %	0,00 %	4,50 %	95,65 %	22,35 %	22,35 %
5	6,45 %	3,33 %	0,00 %	3,33 %	57,07 %	7,61 %	7,61 %
5	2,69 %	1,15 %	1,15 %	0,00 %	76,31 %	4,69 %	4,69 %
5	6,50 %	1,42 %	0,08 %	0,00 %	63,00 %	3,67 %	3,67 %
5	0,00 %	2,39 %	0,30 %	2,39 %	97,61 %	16,27 %	16,27 %
5	2,27 %	2,58 %	0,00 %	0,47 %	61,33 %	11,25 %	11,25 %
5	7,51 %	1,75 %	0,00 %	3,79 %	67,91 %	13,51 %	13,51 %

Sigue en la página siguiente.

$n = m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct} = 0$	$ALPHA_{tct} = 0.5$	$ALPHA_{tct} = 1$
6	6,85 %	0,05 %	0,00 %	0,63 %	112,20 %	9,87 %	9,87 %
6	5,14 %	1,73 %	0,00 %	1,73 %	101,68 %	19,68 %	19,68 %
6	10,71 %	3,76 %	1,38 %	0,00 %	83,83 %	9,07 %	9,07 %
6	4,20 %	2,76 %	0,00 %	0,52 %	77,39 %	10,18 %	10,18 %
6	7,82 %	1,44 %	0,00 %	0,51 %	102,73 %	8,90 %	8,90 %
6	12,62 %	1,79 %	0,17 %	0,00 %	138,02 %	12,67 %	12,67 %
6	3,47 %	1,86 %	0,00 %	3,04 %	86,18 %	27,32 %	27,32 %
6	5,76 %	0,97 %	0,00 %	3,76 %	73,15 %	6,67 %	6,67 %
6	9,28 %	1,97 %	1,10 %	0,00 %	92,73 %	7,68 %	7,68 %
6	3,56 %	3,85 %	0,00 %	1,56 %	81,77 %	6,06 %	6,06 %
7	7,79 %	2,40 %	0,89 %	0,00 %	99,96 %	10,78 %	10,78 %
7	5,96 %	2,98 %	1,47 %	0,00 %	84,36 %	9,75 %	9,75 %
7	11,44 %	3,80 %	0,00 %	3,92 %	109,64 %	8,31 %	8,31 %
7	5,54 %	0,00 %	0,00 %	3,30 %	91,38 %	4,25 %	4,25 %
7	9,45 %	3,10 %	0,00 %	4,57 %	126,81 %	21,60 %	21,60 %
7	6,40 %	2,20 %	1,02 %	0,00 %	77,95 %	5,53 %	5,53 %
7	5,56 %	1,57 %	0,00 %	7,09 %	130,01 %	23,40 %	23,40 %
7	0,51 %	2,40 %	0,00 %	4,01 %	94,72 %	13,33 %	13,33 %
7	4,63 %	2,37 %	0,00 %	0,18 %	127,06 %	23,01 %	23,01 %
7	6,09 %	2,37 %	1,06 %	0,00 %	97,63 %	11,57 %	11,57 %
8	4,11 %	0,56 %	0,00 %	2,04 %	112,09 %	6,93 %	6,93 %
8	9,14 %	2,13 %	0,00 %	1,69 %	94,19 %	6,45 %	6,45 %
8	9,28 %	2,44 %	0,00 %	2,24 %	151,34 %	20,31 %	20,31 %
8	4,00 %	0,72 %	0,00 %	0,58 %	126,08 %	13,90 %	13,90 %
8	9,19 %	1,88 %	0,00 %	0,96 %	125,22 %	5,65 %	5,65 %
8	8,20 %	2,32 %	0,00 %	2,46 %	131,31 %	17,14 %	17,14 %
8	8,03 %	1,65 %	0,00 %	4,22 %	125,61 %	22,99 %	22,99 %
8	4,74 %	0,06 %	0,00 %	0,17 %	92,77 %	20,30 %	20,30 %
8	7,03 %	1,99 %	0,00 %	2,87 %	97,55 %	2,04 %	2,04 %
8	7,78 %	3,28 %	1,78 %	0,00 %	142,63 %	14,01 %	14,01 %
9	5,92 %	0,28 %	0,00 %	0,78 %	146,88 %	20,16 %	20,16 %
9	8,47 %	1,99 %	0,00 %	4,64 %	78,02 %	14,67 %	14,67 %
9	9,03 %	0,55 %	0,02 %	0,00 %	133,69 %	12,68 %	12,68 %
9	14,73 %	2,76 %	0,00 %	3,25 %	130,45 %	6,83 %	6,83 %
9	13,36 %	4,08 %	1,03 %	0,00 %	139,47 %	23,86 %	23,86 %
9	8,24 %	2,63 %	0,00 %	1,92 %	83,21 %	17,85 %	17,85 %
9	6,67 %	0,75 %	0,72 %	0,00 %	146,81 %	9,93 %	9,93 %
9	1,75 %	3,05 %	1,92 %	0,00 %	118,76 %	9,46 %	9,46 %
9	4,50 %	2,50 %	0,00 %	2,32 %	106,43 %	6,93 %	6,93 %
9	6,21 %	0,67 %	0,04 %	0,00 %	131,37 %	16,44 %	16,44 %
10	4,50 %	1,05 %	0,00 %	0,74 %	140,62 %	7,82 %	7,82 %
10	8,38 %	4,77 %	0,00 %	0,55 %	120,76 %	17,94 %	17,94 %
10	7,44 %	2,02 %	0,00 %	0,75 %	145,13 %	12,28 %	12,28 %
10	10,77 %	3,34 %	2,02 %	0,00 %	143,65 %	14,27 %	14,27 %
10	5,92 %	1,98 %	0,00 %	1,23 %	118,51 %	17,33 %	17,33 %
10	7,12 %	2,73 %	0,00 %	1,17 %	159,24 %	13,78 %	13,78 %
10	4,77 %	0,75 %	0,00 %	2,40 %	149,02 %	8,00 %	8,00 %
10	3,19 %	3,01 %	0,00 %	1,69 %	124,47 %	10,85 %	10,85 %
10	5,06 %	2,26 %	0,00 %	1,52 %	145,65 %	12,17 %	12,17 %
10	2,69 %	2,42 %	0,00 %	0,76 %	155,56 %	22,94 %	22,94 %
11	7,31 %	1,03 %	0,00 %	1,08 %	142,35 %	11,38 %	11,38 %
11	8,42 %	3,37 %	2,23 %	0,00 %	135,59 %	19,01 %	19,01 %
11	10,97 %	2,00 %	0,00 %	2,51 %	159,25 %	10,15 %	10,15 %
11	6,93 %	1,17 %	0,00 %	0,10 %	148,23 %	6,67 %	6,67 %
11	14,07 %	0,31 %	0,00 %	2,09 %	154,04 %	12,02 %	12,02 %

Sigue en la página siguiente.

$n = m$	$IRH1_{tct}$	$IRH2_{tct}$	$IRH3_{tct}$	$IRH4_{tct}$	$ALPHA_{tct} = 0$	$ALPHA_{tct} = 0.5$	$ALPHA_{tct} = 1$
11	8,22 %	1,02 %	0,00 %	0,52 %	134,78 %	12,67 %	12,67 %
11	8,24 %	2,34 %	0,00 %	0,80 %	138,78 %	20,57 %	20,57 %
11	5,65 %	0,03 %	0,00 %	1,66 %	163,64 %	10,41 %	10,41 %
11	8,01 %	2,12 %	0,00 %	2,07 %	149,20 %	11,54 %	11,54 %
11	5,20 %	1,09 %	0,11 %	0,00 %	157,00 %	11,36 %	11,36 %
12	6,87 %	1,26 %	0,38 %	0,00 %	205,25 %	16,37 %	16,37 %
12	7,43 %	1,21 %	1,17 %	0,00 %	155,30 %	13,19 %	13,19 %
12	4,96 %	2,24 %	0,08 %	0,00 %	205,43 %	15,15 %	15,15 %
12	7,63 %	3,09 %	0,00 %	0,46 %	121,61 %	9,34 %	9,34 %
12	7,68 %	1,31 %	0,00 %	0,04 %	165,92 %	24,28 %	24,28 %
12	8,08 %	1,24 %	0,00 %	1,13 %	181,21 %	16,86 %	16,86 %
12	7,36 %	0,64 %	0,00 %	0,98 %	175,48 %	8,55 %	8,55 %
12	6,38 %	1,26 %	0,00 %	0,07 %	136,85 %	12,65 %	12,65 %
12	10,01 %	2,45 %	0,00 %	3,71 %	175,08 %	17,50 %	17,50 %
12	8,50 %	2,87 %	0,00 %	2,12 %	178,91 %	9,39 %	9,39 %
13	6,65 %	0,09 %	0,00 %	1,59 %	208,84 %	15,02 %	15,02 %
13	8,77 %	1,44 %	0,00 %	1,36 %	170,74 %	21,44 %	21,44 %
13	7,50 %	2,61 %	0,00 %	1,92 %	165,96 %	10,42 %	10,42 %
13	8,67 %	0,66 %	0,00 %	1,14 %	188,63 %	9,39 %	9,39 %
13	10,98 %	0,69 %	0,00 %	1,24 %	193,57 %	13,07 %	13,07 %
13	6,06 %	0,71 %	0,00 %	0,75 %	135,64 %	16,76 %	16,76 %
13	7,62 %	0,59 %	0,00 %	0,16 %	173,17 %	7,93 %	7,93 %
13	9,45 %	1,07 %	0,00 %	1,77 %	153,06 %	15,31 %	15,31 %
13	11,53 %	2,24 %	0,95 %	0,00 %	170,92 %	15,42 %	15,42 %
13	12,52 %	1,46 %	0,00 %	3,63 %	143,17 %	14,09 %	14,09 %
14	16,08 %	1,22 %	0,00 %	1,26 %	190,95 %	10,57 %	10,57 %
14	7,95 %	2,90 %	0,00 %	1,17 %	178,44 %	12,78 %	12,78 %
14	11,03 %	1,62 %	0,45 %	0,00 %	180,91 %	26,30 %	26,30 %
14	7,30 %	2,29 %	0,00 %	1,94 %	214,33 %	20,88 %	20,88 %
14	11,26 %	1,67 %	0,00 %	0,84 %	197,81 %	7,88 %	7,88 %
14	15,63 %	1,37 %	0,00 %	0,54 %	186,75 %	12,25 %	12,25 %
14	11,27 %	0,40 %	0,00 %	0,56 %	218,50 %	19,23 %	19,23 %
14	12,64 %	2,88 %	1,74 %	0,00 %	189,68 %	23,70 %	23,70 %
14	13,82 %	1,01 %	0,00 %	2,37 %	207,92 %	19,08 %	19,08 %
14	9,57 %	1,70 %	0,00 %	1,55 %	173,28 %	21,87 %	21,87 %
15	10,82 %	1,52 %	0,00 %	1,06 %	180,54 %	9,25 %	9,25 %
15	10,29 %	0,71 %	0,14 %	0,00 %	191,33 %	16,86 %	16,86 %
15	8,52 %	0,91 %	0,00 %	1,06 %	173,93 %	12,71 %	12,71 %
15	9,53 %	1,43 %	0,00 %	0,89 %	186,51 %	12,89 %	12,89 %
15	10,77 %	0,54 %	0,00 %	1,63 %	181,38 %	25,54 %	25,54 %
15	8,99 %	0,93 %	0,00 %	0,05 %	162,32 %	19,36 %	19,36 %
15	11,30 %	0,57 %	0,00 %	0,17 %	215,02 %	14,75 %	14,75 %
15	10,36 %	2,97 %	0,00 %	2,11 %	184,69 %	18,87 %	18,87 %
15	6,85 %	1,63 %	0,00 %	0,59 %	193,78 %	13,04 %	13,04 %
15	9,12 %	2,49 %	0,00 %	1,79 %	195,72 %	22,57 %	22,57 %

Lo realmente destacable de los ARPD para este caso de el tiempo total de finalización es que a pesar de que los primeros dos métodos no dieron ninguna mejor solución la distancia promedio que hay de las soluciones que aportan estos métodos es muy baja siendo el máximo alrededor de un siete por ciento para el primer caso.

En segundo lugar se evidencia lo que vimos en el apartado anterior 6.1.1. de los resultados de Alpha que se alejaban en cierta medida de los que proporcionaban el resto de métodos, si observamos la tabla vemos que el promedio de las desviaciones del método *BICH-MIH* son muy altas en general, excediendo el 100 % en el caso de $\alpha = 0$. Los valores se plasman en la Tabla 6.7 adjunta.

Tabla 6.7 ARPDs Promedio *Total Completion Time*. [Fuente: Elaboración propia].

ARPDs Promedio *Total Completion Time*						
$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA = 0$	$ALPHA = 0{,}5$	$ALPHA = 1$
7,26 %	2,29 %	0,25 %	1,54 %	125,98 %	13,74 %	13,74 %

6.2 Evaluación para FO: *Makespan*

Ahora se presenta la misma gráfica, Tabla 6.8, que para el punto 6.1.1 pero usando en este caso la función objetivo de minimización del tiempo de terminación. Cada barra de distinto color representa los valores que nos aporta cada método como indica la leyenda de dicha figura

Tabla 6.8 Resultados de función objetivo para caso *Makespan*. [Fuente: Elaboración propia].

Valores Función objetivo Makespan							
$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
3	110	114	110	114	117	117	166
3	192	192	192	192	192	192	232
3	155	155	155	155	155	155	216
3	167	167	167	167	167	167	210
3	195	195	195	195	195	195	246
3	227	227	227	227	227	227	291
3	157	157	157	157	187	187	206
3	184	184	184	184	184	184	228
3	143	143	143	143	147	143	183
3	276	276	276	276	290	290	374
4	245	227	227	227	263	263	461
4	301	285	285	275	283	269	403
4	252	252	252	252	271	271	383
4	273	268	268	268	281	281	375
4	256	256	256	256	280	280	450
4	266	276	271	276	285	285	444
4	280	280	280	280	311	303	464
4	219	219	219	219	237	237	311
4	282	277	277	282	294	292	459
4	284	290	284	284	295	295	486
5	325	325	325	325	355	329	572
5	371	359	358	358	407	406	579
5	242	235	225	232	257	242	410
5	331	320	315	320	351	341	485
5	358	349	349	347	347	359	511
5	301	313	313	304	334	325	517
5	279	275	275	281	280	288	417
5	326	305	305	305	330	328	505
5	336	330	330	330	330	409	495
5	337	323	321	321	339	339	417
6	416	413	413	400	424	414	748
6	386	368	364	366	409	409	667
6	351	369	351	351	355	362	558
6	350	350	350	350	364	364	596
6	395	395	395	395	431	431	719
6	357	355	351	355	363	363	660
6	393	378	375	378	413	412	614
6	347	347	347	350	384	373	611

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
6	418	406	406	408	446	428	805
6	484	488	483	483	545	483	786
7	464	453	453	452	483	493	885
7	505	505	505	505	524	530	875
7	474	468	468	475	497	475	719
7	478	478	478	478	485	484	736
7	502	489	489	489	506	503	818
7	466	436	430	450	448	441	778
7	418	408	405	405	424	433	670
7	531	531	531	531	564	531	755
7	570	560	560	560	560	560	829
7	515	489	489	484	521	486	812
8	508	477	474	477	537	512	820
8	632	632	632	632	673	633	1089
8	567	535	534	539	549	540	900
8	541	522	520	518	533	514	940
8	556	556	556	556	594	556	806
8	529	520	505	508	536	528	960
8	525	511	508	515	597	553	1061
8	571	581	571	571	601	576	910
8	596	597	596	596	635	614	897
8	558	558	558	558	593	559	920
9	695	695	695	695	695	696	1091
9	666	666	666	666	680	668	971
9	625	625	625	625	655	629	1023
9	595	576	576	611	626	585	1075
9	596	601	601	596	676	615	1115
9	568	554	551	558	626	598	1114
9	504	502	502	504	524	533	861
9	578	577	568	565	594	566	1082
9	611	593	584	586	605	600	1096
9	606	595	595	600	621	639	951
10	651	643	638	637	688	661	1285
10	709	702	702	702	735	702	1136
10	732	733	712	713	708	721	1212
10	692	694	688	689	696	715	1172
10	714	705	699	712	735	743	1196
10	701	683	681	681	719	687	1248
10	645	662	647	654	719	663	1074
10	593	581	581	581	658	593	1204
10	719	722	722	719	719	745	1226
10	716	689	689	677	756	718	1403
11	775	771	771	771	808	771	1352
11	674	660	653	666	687	696	1415
11	682	648	639	665	696	679	1248
11	672	644	641	647	667	665	1143
11	848	848	848	848	863	848	1336
11	688	692	688	688	743	730	1228
11	786	778	778	776	790	793	1355
11	766	703	687	696	783	773	1426
11	687	668	665	661	675	686	1245
11	715	692	686	687	714	686	1361
12	824	826	819	819	875	819	1678
12	841	798	781	772	836	819	1518
12	717	689	689	691	755	755	1559

Sigue en la página siguiente.

$n=m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak}=0$	$ALPHA_{mak}=0.5$	$ALPHA_{mak}=1$
12	843	800	795	802	822	826	1686
12	770	770	770	770	793	770	1474
12	717	706	699	711	748	740	1356
12	764	754	724	731	790	766	1343
12	809	781	781	776	822	786	1459
12	789	790	772	785	860	827	1441
12	844	844	844	844	869	858	1420
13	823	823	823	824	864	840	1409
13	841	826	825	826	893	823	1650
13	867	867	867	867	884	889	1671
13	804	776	768	770	832	779	1412
13	961	967	961	961	961	968	1833
13	802	766	754	753	810	779	1482
13	870	822	822	824	876	842	1474
13	932	919	898	910	916	904	1779
13	904	904	904	904	940	933	1462
13	821	803	799	787	838	821	1357
14	951	930	929	936	988	958	1917
14	937	926	926	926	991	965	1732
14	946	936	923	923	978	925	1750
14	894	886	881	885	941	936	1680
14	929	932	932	929	1015	973	1748
14	912	900	900	898	962	927	1918
14	887	889	887	892	932	888	1599
14	852	828	821	825	862	844	1650
14	896	869	855	854	940	899	1833
14	904	874	866	868	898	862	1574
15	1015	992	990	986	1045	1012	1938
15	997	965	963	966	1023	1002	1831
15	1013	976	975	979	1035	1017	1846
15	949	905	903	896	999	936	1849
15	1032	1002	1001	1002	1078	1024	1869
15	987	979	979	978	1011	1030	1886
15	991	985	985	991	1014	997	2081
15	955	940	935	935	1001	935	1616
15	1033	983	975	974	1075	1010	2079
15	1012	1020	1004	1005	1068	1032	1880

Al igual que con la primera función objetivo analizada los peores resultados se dan con el caso del algoritmo *BICH-MIH*, en este particular destaca el modelo en el que el parámetro α adopta el valor uno. El resto de los métodos dan valores próximos entre sí sin ningún caso excepcional a simple vista.

6.2.1 Número de mejores soluciones

Tabla 6.9 Mejores soluciones para función objetivo *Makespan*. [Fuente: Elaboración propia].

Mejores soluciones para función objetivo *Makespan*							
$n=m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak}=0$	$ALPHA_{mak}=0.5$	$ALPHA_{mak}=1$
3	1	0	1	0	0	0	0
3	1	1	1	1	1	1	0
3	1	1	1	1	1	1	0

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
3	1	1	1	1	1	1	0
3	1	1	1	1	1	1	0
3	1	1	1	1	1	1	0
3	1	1	1	1	0	0	0
3	1	1	1	1	1	1	0
3	1	1	1	1	0	1	0
3	1	1	1	1	0	0	0
4	0	1	1	1	0	0	0
4	0	0	0	0	0	1	0
4	1	1	1	1	0	0	0
4	0	1	1	1	0	0	0
4	1	1	1	1	0	0	0
4	1	0	0	0	0	0	0
4	1	1	1	1	0	0	0
4	1	1	1	1	0	0	0
4	0	1	1	0	0	0	0
4	1	0	1	1	0	0	0
5	1	1	1	1	0	0	0
5	0	0	1	1	0	0	0
5	0	0	1	0	0	0	0
5	0	0	1	0	0	0	0
5	0	0	0	1	1	0	0
5	1	0	0	0	0	0	0
5	0	1	1	0	0	0	0
5	0	1	1	1	0	0	0
5	0	1	1	1	1	0	0
5	0	0	1	1	0	0	0
6	0	0	0	1	0	0	0
6	0	0	1	0	0	0	0
6	1	0	1	1	0	0	0
6	1	1	1	1	0	0	0
6	1	1	1	1	0	0	0
6	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0
6	1	1	1	0	0	0	0
6	0	1	1	0	0	0	0
6	0	0	1	1	0	1	0
7	0	0	0	1	0	0	0
7	1	1	1	1	0	0	0
7	0	1	1	0	0	0	0
7	1	1	1	1	0	0	0
7	0	1	1	1	0	0	0
7	0	0	1	0	0	0	0
7	0	0	1	1	0	0	0
7	1	1	1	1	0	1	0
7	0	1	1	1	1	1	0
7	0	0	0	1	0	0	0
8	0	0	1	0	0	0	0
8	1	1	1	1	0	0	0
8	0	0	1	0	0	0	0
8	0	0	0	0	0	1	0
8	1	1	1	1	0	1	0
8	0	0	1	0	0	0	0
8	0	0	1	0	0	0	0
8	1	0	1	1	0	0	0

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
8	1	0	1	1	0	0	0
8	1	1	1	1	0	0	0
9	1	1	1	1	1	0	0
9	1	1	1	1	0	0	0
9	1	1	1	1	0	0	0
9	0	1	1	0	0	0	0
9	1	0	0	1	0	0	0
9	0	0	1	0	0	0	0
9	0	1	1	0	0	0	0
9	0	0	0	1	0	0	0
9	0	0	1	0	0	0	0
9	0	1	1	0	0	0	0
10	0	0	0	1	0	0	0
10	0	1	1	1	0	1	0
10	0	0	0	0	1	0	0
10	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0
10	0	0	1	1	0	0	0
10	1	0	0	0	0	0	0
10	0	1	1	1	0	0	0
10	1	0	0	1	1	0	0
10	0	0	0	1	0	0	0
11	0	1	1	1	0	1	0
11	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0
11	1	1	1	1	0	1	0
11	1	0	1	1	0	0	0
11	0	0	0	1	0	0	0
11	0	0	1	0	0	0	0
11	0	0	0	1	0	0	0
11	0	0	1	0	0	1	0
12	0	0	1	1	0	1	0
12	0	0	0	1	0	0	0
12	0	1	1	0	0	0	0
12	0	0	1	0	0	0	0
12	1	1	1	1	0	1	0
12	0	0	1	0	0	0	0
12	0	0	1	0	0	0	0
12	0	0	0	1	0	0	0
12	0	0	1	0	0	0	0
12	1	1	1	1	0	0	0
13	1	1	1	0	0	0	0
13	0	0	0	0	0	1	0
13	1	1	1	1	0	0	0
13	0	0	1	0	0	0	0
13	1	0	1	1	1	0	0
13	0	0	0	1	0	0	0
13	0	1	1	0	0	0	0
13	0	0	1	0	0	0	0
13	1	1	1	1	0	0	0
13	0	0	0	1	0	0	0
14	0	0	1	0	0	0	0
14	0	1	1	1	0	0	0
14	0	0	1	1	0	0	0

Sigue en la página siguiente.

$n=m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak}=0$	$ALPHA_{mak}=0.5$	$ALPHA_{mak}=1$
14	0	0	1	0	0	0	0
14	1	0	0	1	0	0	0
14	0	0	0	1	0	0	0
14	1	0	1	0	0	0	0
14	0	0	1	0	0	0	0
14	0	0	0	1	0	0	0
14	0	0	0	0	0	1	0
15	0	0	0	1	0	0	0
15	0	0	1	0	0	0	0
15	0	0	1	0	0	0	0
15	0	0	0	1	0	0	0
15	0	0	1	0	0	0	0
15	0	0	0	1	0	0	0
15	0	1	1	0	0	0	0
15	0	0	1	1	0	1	0
15	0	0	0	1	0	0	0
15	0	0	1	0	0	0	0

El número de mejores soluciones para este caso particular de función objetivo se ilustra en la Tabla 6.9. Los cuatro métodos correspondientes al primer artículo, *IRH1, IRH2, IRH3 e IRH4* consiguen llegar a la mejor solución en todos los casos de las instancias propuestas, aunque vemos que las dos primeras heurísticas no superan más de tres mejores soluciones se debe a que la complejidad de los algoritmos es bastante más baja que la de sus métodos vecinos *IRH3* e *IRH4*, no obstante, en uno de los próximos apartados veremos sus ARPD para tener una idea más precisa de la actuación de estas heurísticas. Por otro lado, vemos que el algoritmo *BICH-MIH* no consigue darnos buenos resultados, salvo excepcionalmente para el caso $Alpha = 0,5$ y se da para el tamaño de problema más pequeño (n=m=3).

El número final de mejores soluciones obtenidas para cada método en este caso de función objetivo se representa en la Tabla 6.10

Tabla 6.10 Número de Mejores Soluciones *Makespan*. [Fuente: Elaboración propia].

Número de Mejores Soluciones *Makespan*

$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA=0$	$ALPHA=0,5$	$ALPHA=1$
46	52	100	76	13	22	0

6.2.2 CPU times

Tabla 6.11 Tiempo computacional para el caso *Makespan*. [Fuente: Elaboración propia].

Tiempo computacional para el caso *Makespan*

$n=m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak}=0$	$ALPHA_{mak}=0.5$	$ALPHA_{mak}=1$
3	0,003	0,001	0,001	0,001	0	0	0
3	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
4	0	0	0	0,001	0	0	0
4	0	0	0	0,002	0	0	0
4	0	0	0	0,001	0	0	0
4	0	0	0	0,001	0	0	0
4	0	0	0	0,001	0	0	0
4	0	0	0	0,001	0	0	0
4	0	0	0	0,001	0	0	0
4	0	0	0	0,001	0	0	0
4	0	0	0	0,001	0	0	0
4	0	0	0	0,001	0	0	0
5	0	0	0,001	0,005	0	0	0
5	0	0,001	0,001	0,005	0	0	0
5	0	0,001	0,001	0,006	0	0	0
5	0	0	0,001	0,005	0	0	0
5	0	0	0,001	0,005	0	0	0
5	0	0	0,001	0,006	0	0	0
5	0	0	0,001	0,006	0	0	0
5	0	0	0,001	0,005	0	0	0
5	0	0	0,001	0,006	0	0	0
5	0	0	0,001	0,005	0	0	0
6	0	0,002	0,003	0,021	0	0	0
6	0	0,002	0,004	0,026	0	0	0
6	0	0,002	0,004	0,022	0	0	0
6	0	0,002	0,003	0,018	0	0	0
6	0	0,002	0,003	0,02	0	0	0
6	0	0,002	0,003	0,031	0	0	0
6	0	0,002	0,004	0,019	0	0	0
6	0	0,002	0,003	0,019	0	0	0
6	0	0,002	0,002	0,019	0	0	0
6	0	0,002	0,003	0,021	0	0	0
7	0	0,009	0,01	0,053	0	0	0
7	0	0,006	0,008	0,051	0	0	0
7	0,002	0,008	0,008	0,054	0	0	0
7	0	0,007	0,009	0,059	0	0	0
7	0	0,006	0,008	0,056	0	0	0
7	0	0,006	0,01	0,057	0	0	0
7	0	0,006	0,01	0,085	0	0	0
7	0	0,006	0,01	0,053	0	0	0
7	0	0,007	0,01	0,054	0	0	0
7	0	0,006	0,008	0,058	0	0	0
8	0,001	0,015	0,02	0,16	0	0	0
8	0,001	0,014	0,019	0,131	0	0	0
8	0,001	0,013	0,022	0,133	0	0	0
8	0,002	0,016	0,022	0,155	0	0	0
8	0,001	0,014	0,02	0,136	0	0	0
8	0,001	0,015	0,024	0,138	0	0	0
8	0,001	0,015	0,023	0,136	0	0	0
8	0,001	0,016	0,023	0,131	0	0	0
8	0,001	0,014	0,017	0,131	0	0	0
8	0,001	0,015	0,019	0,136	0	0	0
9	0,002	0,035	0,04	0,311	0	0	0
9	0,002	0,038	0,042	0,334	0	0	0
9	0,002	0,034	0,044	0,288	0	0	0
9	0,002	0,038	0,05	0,306	0	0	0
9	0,002	0,033	0,04	0,361	0	0	0

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
9	0,005	0,032	0,05	0,311	0	0	0
9	0,002	0,033	0,042	0,371	0	0	0
9	0,003	0,041	0,057	0,319	0	0	0
9	0,002	0,032	0,04	0,351	0	0	0
9	0,003	0,036	0,04	0,328	0	0	0
10	0,005	0,069	0,088	0,728	0	0,001	0,001
10	0,004	0,077	0,1	0,64	0,001	0,001	0,001
10	0,004	0,071	0,101	0,703	0	0	0
10	0,003	0,068	0,083	0,606	0,001	0	0
10	0,004	0,065	0,107	0,66	0	0	0
10	0,004	0,069	0,088	0,658	0	0	0
10	0,004	0,067	0,089	0,654	0	0	0
10	0,004	0,066	0,079	0,712	0	0	0
10	0,004	0,066	0,08	0,615	0,001	0	0,001
10	0,005	0,072	0,077	0,646	0	0	0
11	0,008	0,126	0,151	1,132	0,001	0,001	0,001
11	0,007	0,124	0,164	1,292	0,001	0,001	0,001
11	0,007	0,128	0,181	1,201	0,001	0,001	0,001
11	0,007	0,128	0,167	1,244	0,001	0,001	0,001
11	0,006	0,124	0,157	1,203	0,001	0,001	0,001
11	0,007	0,162	0,153	1,357	0,001	0,001	0
11	0,007	0,159	0,153	1,285	0,001	0,001	0,001
11	0,009	0,138	0,239	1,335	0,002	0,001	0,002
11	0,008	0,133	0,178	1,366	0,001	0,001	0,001
11	0,008	0,145	0,175	1,204	0,001	0,001	0,001
12	0,011	0,23	0,38	2,2	0,001	0,001	0,001
12	0,011	0,24	0,395	2,257	0,001	0,001	0,001
12	0,011	0,252	0,298	2,317	0,001	0,001	0,001
12	0,01	0,23	0,288	2,206	0,001	0,001	0,001
12	0,009	0,231	0,266	2,201	0,001	0,001	0,002
12	0,013	0,269	0,295	2,312	0,001	0,001	0,001
12	0,01	0,236	0,302	2,506	0,002	0,001	0,001
12	0,011	0,255	0,28	2,376	0,001	0,001	0,002
12	0,01	0,277	0,337	2,486	0,001	0,002	0,002
12	0,012	0,259	0,31	2,291	0,001	0,002	0,001
13	0,015	0,423	0,506	3,968	0,004	0,003	0,002
13	0,021	0,426	0,452	3,906	0,002	0,002	0,002
13	0,015	0,397	0,438	3,538	0,002	0,002	0,002
13	0,015	0,41	0,476	3,95	0,002	0,002	0,002
13	0,015	0,418	0,463	3,732	0,002	0,002	0,002
13	0,015	0,407	0,631	4,491	0,002	0,002	0,002
13	0,018	0,427	0,49	3,973	0,002	0,002	0,003
13	0,015	0,43	0,496	4,19	0,002	0,003	0,003
13	0,015	0,445	0,469	3,637	0,002	0,002	0,003
13	0,015	0,394	0,482	4,05	0,002	0,002	0,002
14	0,027	0,726	0,849	6,65	0,003	0,003	0,003
14	0,022	0,734	0,784	6,9	0,003	0,004	0,003
14	0,022	0,707	0,872	7,035	0,003	0,002	0,002
14	0,023	0,71	0,814	6,493	0,003	0,004	0,003
14	0,021	0,673	0,84	6,969	0,003	0,003	0,003
14	0,02	0,657	0,756	6,424	0,002	0,003	0,003
14	0,021	0,678	0,819	6,766	0,003	0,003	0,003
14	0,027	0,738	0,841	6,991	0,003	0,004	0,004
14	0,026	0,74	0,833	6,447	0,003	0,003	0,003
14	0,022	0,685	0,837	6,412	0,003	0,003	0,003

Sigue en la página siguiente.

$n=m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak}=0$	$ALPHA_{mak}=0.5$	$ALPHA_{mak}=1$
15	0,031	1,071	1,238	10,121	0,004	0,005	0,004
15	0,032	1,073	1,292	10,275	0,004	0,004	0,004
15	0,032	1,08	1,183	10,347	0,005	0,005	0,004
15	0,038	1,081	1,385	11,365	0,004	0,004	0,005
15	0,035	1,083	1,281	10,378	0,004	0,004	0,004
15	0,034	1,073	1,201	10,348	0,005	0,004	0,004
15	0,032	1,095	1,21	10,515	0,004	0,004	0,004
15	0,03	1,131	1,208	10,747	0,004	0,004	0,004
15	0,032	1,077	1,228	10,678	0,004	0,004	0,004
15	0,032	1,09	1,317	10,349	0,005	0,004	0,005

La Tabla 6.11 nos muestra los tiempos de computación de las heurísticas para esta función objetivo (*Makespan*), como se puede ver el método *IRH4* es el que más tarda en resolver todas las instancias, seguido por las heurísticas *IRH2* e *IRH3*. Por último, al resto de métodos, con complejidad menor, les lleva menos de un segundo de media resolver los problemas.

Para mayor facilidad en el tratamiento de estos datos usaremos los promedios de los tiempos de computación teniendo así el marco global de la complejidad y del desarrollo de las heurísticas. Estos valores se adjuntan en la Tabla 6.12 se adjuntan adicionalmente las tablas a partir de las que salen estos datos donde se pueden ver los tiempos de computación seccionados por tamaño de problema.

Tabla 6.12 CPU Promedio *Makespan*. [Fuente: Elaboración propia].

CPU Promedio *Makespan*

$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA=0$	$ALPHA=0,5$	$ALPHA=1$
0,0076	0,2093	0,2480	1,9967	0,0009	0,0009	0,0009

6.2.3 ARPD

Como ya se mencionó cuando se vieron los resultados de los valores de los métodos, los métodos finales son los que mayor ARPD tienen, es decir, que las soluciones que aportan distan mucho de la mejor obtenida en los problemas.Se observan los resultados en la Tabla 6.13.

Tabla 6.13 ARPDs para caso *Makespan*. [Fuente: Elaboración propia].

ARPDs para caso *Makespan*

$n=m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak}=0$	$ALPHA_{mak}=0.5$	$ALPHA_{mak}=1$
3	0,00 %	3,64 %	0,00 %	3,64 %	6,36 %	6,36 %	50,91 %
3	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	20,83 %
3	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	39,35 %
3	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	25,75 %
3	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	26,15 %
3	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	28,19 %
3	0,00 %	0,00 %	0,00 %	0,00 %	19,11 %	19,11 %	31,21 %
3	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	23,91 %
3	0,00 %	0,00 %	0,00 %	0,00 %	2,80 %	0,00 %	27,97 %
3	0,00 %	0,00 %	0,00 %	0,00 %	5,07 %	5,07 %	35,51 %
4	7,93 %	0,00 %	0,00 %	0,00 %	15,86 %	15,86 %	103,08 %
4	11,90 %	5,95 %	5,95 %	2,23 %	5,20 %	0,00 %	49,81 %
4	0,00 %	0,00 %	0,00 %	0,00 %	7,54 %	7,54 %	51,98 %
4	1,87 %	0,00 %	0,00 %	0,00 %	4,85 %	4,85 %	39,93 %
4	0,00 %	0,00 %	0,00 %	0,00 %	9,38 %	9,38 %	75,78 %

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
4	0,00 %	3,76 %	1,88 %	3,76 %	7,14 %	7,14 %	66,92 %
4	0,00 %	0,00 %	0,00 %	0,00 %	11,07 %	8,21 %	65,71 %
4	0,00 %	0,00 %	0,00 %	0,00 %	8,22 %	8,22 %	42,01 %
4	1,81 %	0,00 %	0,00 %	1,81 %	6,14 %	5,42 %	65,70 %
4	0,00 %	2,11 %	0,00 %	0,00 %	3,87 %	3,87 %	71,13 %
5	0,00 %	0,00 %	0,00 %	0,00 %	9,23 %	1,23 %	76,00 %
5	3,63 %	0,28 %	0,00 %	0,00 %	13,69 %	13,41 %	61,73 %
5	7,56 %	4,44 %	0,00 %	3,11 %	14,22 %	7,56 %	82,22 %
5	5,08 %	1,59 %	0,00 %	1,59 %	11,43 %	8,25 %	53,97 %
5	3,17 %	0,58 %	0,58 %	0,00 %	0,00 %	3,46 %	47,26 %
5	0,00 %	3,99 %	3,99 %	1,00 %	10,96 %	7,97 %	71,76 %
5	1,45 %	0,00 %	0,00 %	2,18 %	1,82 %	4,73 %	51,64 %
5	6,89 %	0,00 %	0,00 %	0,00 %	8,20 %	7,54 %	65,57 %
5	1,82 %	0,00 %	0,00 %	0,00 %	0,00 %	23,94 %	50,00 %
5	4,98 %	0,62 %	0,00 %	0,00 %	5,61 %	5,61 %	29,91 %
6	4,00 %	3,25 %	3,25 %	0,00 %	6,00 %	3,50 %	87,00 %
6	6,04 %	1,10 %	0,00 %	0,55 %	12,36 %	12,36 %	83,24 %
6	0,00 %	5,13 %	0,00 %	0,00 %	1,14 %	3,13 %	58,97 %
6	0,00 %	0,00 %	0,00 %	0,00 %	4,00 %	4,00 %	70,29 %
6	0,00 %	0,00 %	0,00 %	0,00 %	9,11 %	9,11 %	82,03 %
6	1,71 %	1,14 %	0,00 %	1,14 %	3,42 %	3,42 %	88,03 %
6	4,80 %	0,80 %	0,00 %	0,80 %	10,13 %	9,87 %	63,73 %
6	0,00 %	0,00 %	0,00 %	0,86 %	10,66 %	7,49 %	76,08 %
6	2,96 %	0,00 %	0,00 %	0,49 %	9,85 %	5,42 %	98,28 %
6	0,21 %	1,04 %	0,00 %	0,00 %	12,84 %	0,00 %	62,73 %
7	2,65 %	0,22 %	0,22 %	0,00 %	6,86 %	9,07 %	95,80 %
7	0,00 %	0,00 %	0,00 %	0,00 %	3,76 %	4,95 %	73,27 %
7	1,28 %	0,00 %	0,00 %	1,50 %	6,20 %	1,50 %	53,63 %
7	0,00 %	0,00 %	0,00 %	0,00 %	1,46 %	1,26 %	53,97 %
7	2,66 %	0,00 %	0,00 %	0,00 %	3,48 %	2,86 %	67,28 %
7	8,37 %	1,40 %	0,00 %	4,65 %	4,19 %	2,56 %	80,93 %
7	3,21 %	0,74 %	0,00 %	0,00 %	4,69 %	6,91 %	65,43 %
7	0,00 %	0,00 %	0,00 %	0,00 %	6,21 %	0,00 %	42,18 %
7	1,79 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	48,04 %
7	6,40 %	1,03 %	1,03 %	0,00 %	7,64 %	0,41 %	67,77 %
8	7,17 %	0,63 %	0,00 %	0,63 %	13,29 %	8,02 %	73,00 %
8	0,00 %	0,00 %	0,00 %	0,00 %	6,49 %	0,16 %	72,31 %
8	6,18 %	0,19 %	0,00 %	0,94 %	2,81 %	1,12 %	68,54 %
8	5,25 %	1,56 %	1,17 %	0,78 %	3,70 %	0,00 %	82,88 %
8	0,00 %	0,00 %	0,00 %	0,00 %	6,83 %	0,00 %	44,96 %
8	4,75 %	2,97 %	0,00 %	0,59 %	6,14 %	4,55 %	90,10 %
8	3,35 %	0,59 %	0,00 %	1,38 %	17,52 %	8,86 %	108,86 %
8	0,00 %	1,75 %	0,00 %	0,00 %	5,25 %	0,88 %	59,37 %
8	0,00 %	0,17 %	0,00 %	0,00 %	6,54 %	3,02 %	50,50 %
8	0,00 %	0,00 %	0,00 %	0,00 %	6,27 %	0,18 %	64,87 %
9	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,14 %	56,98 %
9	0,00 %	0,00 %	0,00 %	0,00 %	2,10 %	0,30 %	45,80 %
9	0,00 %	0,00 %	0,00 %	0,00 %	4,80 %	0,64 %	63,68 %
9	3,30 %	0,00 %	0,00 %	6,08 %	8,68 %	1,56 %	86,63 %
9	0,00 %	0,84 %	0,84 %	0,00 %	13,42 %	3,19 %	87,08 %
9	3,09 %	0,54 %	0,00 %	1,27 %	13,61 %	8,53 %	102,18 %
9	0,40 %	0,00 %	0,00 %	0,40 %	4,38 %	6,18 %	71,51 %
9	2,30 %	2,12 %	0,53 %	0,00 %	5,13 %	0,18 %	91,50 %
9	4,62 %	1,54 %	0,00 %	0,34 %	3,60 %	2,74 %	87,67 %
9	1,85 %	0,00 %	0,00 %	0,84 %	4,37 %	7,39 %	59,83 %

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
10	2,20 %	0,94 %	0,16 %	0,00 %	8,01 %	3,77 %	101,73 %
10	1,00 %	0,00 %	0,00 %	0,00 %	4,70 %	0,00 %	61,82 %
10	3,39 %	3,53 %	0,56 %	0,71 %	0,00 %	1,84 %	71,19 %
10	0,58 %	0,87 %	0,00 %	0,15 %	1,16 %	3,92 %	70,35 %
10	2,15 %	0,86 %	0,00 %	1,86 %	5,15 %	6,29 %	71,10 %
10	2,94 %	0,29 %	0,00 %	0,00 %	5,58 %	0,88 %	83,26 %
10	0,00 %	2,64 %	0,31 %	1,40 %	11,47 %	2,79 %	66,51 %
10	2,07 %	0,00 %	0,00 %	0,00 %	13,25 %	2,07 %	107,23 %
10	0,00 %	0,42 %	0,42 %	0,00 %	0,00 %	3,62 %	70,51 %
10	5,76 %	1,77 %	1,77 %	0,00 %	11,67 %	6,06 %	107,24 %
11	0,52 %	0,00 %	0,00 %	0,00 %	4,80 %	0,00 %	75,36 %
11	3,22 %	1,07 %	0,00 %	1,99 %	5,21 %	6,58 %	116,69 %
11	6,73 %	1,41 %	0,00 %	4,07 %	8,92 %	6,26 %	95,31 %
11	4,84 %	0,47 %	0,00 %	0,94 %	4,06 %	3,74 %	78,32 %
11	0,00 %	0,00 %	0,00 %	0,00 %	1,77 %	0,00 %	57,55 %
11	0,00 %	0,58 %	0,00 %	0,00 %	7,99 %	6,10 %	78,49 %
11	1,29 %	0,26 %	0,26 %	0,00 %	1,80 %	2,19 %	74,61 %
11	11,50 %	2,33 %	0,00 %	1,31 %	13,97 %	12,52 %	107,57 %
11	3,93 %	1,06 %	0,61 %	0,00 %	2,12 %	3,78 %	88,35 %
11	4,23 %	0,87 %	0,00 %	0,15 %	4,08 %	0,00 %	98,40 %
12	0,61 %	0,85 %	0,00 %	0,00 %	6,84 %	0,00 %	104,88 %
12	8,94 %	3,37 %	1,17 %	0,00 %	8,29 %	6,09 %	96,63 %
12	4,06 %	0,00 %	0,00 %	0,29 %	9,58 %	9,58 %	126,27 %
12	6,04 %	0,63 %	0,00 %	0,88 %	3,40 %	3,90 %	112,08 %
12	0,00 %	0,00 %	0,00 %	0,00 %	2,99 %	0,00 %	91,43 %
12	2,58 %	1,00 %	0,00 %	1,72 %	7,01 %	5,87 %	93,99 %
12	5,52 %	4,14 %	0,00 %	0,97 %	9,12 %	5,80 %	85,50 %
12	4,25 %	0,64 %	0,64 %	0,00 %	5,93 %	1,29 %	88,02 %
12	2,20 %	2,33 %	0,00 %	1,68 %	11,40 %	7,12 %	86,66 %
12	0,00 %	0,00 %	0,00 %	0,00 %	2,96 %	1,66 %	68,25 %
13	0,00 %	0,00 %	0,00 %	0,12 %	4,98 %	2,07 %	71,20 %
13	2,19 %	0,36 %	0,24 %	0,36 %	8,51 %	0,00 %	100,49 %
13	0,00 %	0,00 %	0,00 %	0,00 %	1,96 %	2,54 %	92,73 %
13	4,69 %	1,04 %	0,00 %	0,26 %	8,33 %	1,43 %	83,85 %
13	0,00 %	0,62 %	0,00 %	0,00 %	0,00 %	0,73 %	90,74 %
13	6,51 %	1,73 %	0,13 %	0,00 %	7,57 %	3,45 %	96,81 %
13	5,84 %	0,00 %	0,00 %	0,24 %	6,57 %	2,43 %	79,32 %
13	3,79 %	2,34 %	0,00 %	1,34 %	2,00 %	0,67 %	98,11 %
13	0,00 %	0,00 %	0,00 %	0,00 %	3,98 %	3,21 %	61,73 %
13	4,32 %	2,03 %	1,52 %	0,00 %	6,48 %	4,32 %	72,43 %
14	2,37 %	0,11 %	0,00 %	0,75 %	6,35 %	3,12 %	106,35 %
14	1,19 %	0,00 %	0,00 %	0,00 %	7,02 %	4,21 %	87,04 %
14	2,49 %	1,41 %	0,00 %	0,00 %	5,96 %	0,22 %	89,60 %
14	1,48 %	0,57 %	0,00 %	0,45 %	6,81 %	6,24 %	90,69 %
14	0,00 %	0,32 %	0,32 %	0,00 %	9,26 %	4,74 %	88,16 %
14	1,56 %	0,22 %	0,22 %	0,00 %	7,13 %	3,23 %	113,59 %
14	0,00 %	0,23 %	0,00 %	0,56 %	5,07 %	0,11 %	80,27 %
14	3,78 %	0,85 %	0,00 %	0,49 %	4,99 %	2,80 %	100,97 %
14	4,92 %	1,76 %	0,12 %	0,00 %	10,07 %	5,27 %	114,64 %
14	4,87 %	1,39 %	0,46 %	0,70 %	4,18 %	0,00 %	82,60 %
15	2,94 %	0,61 %	0,41 %	0,00 %	5,98 %	2,64 %	96,55 %
15	3,53 %	0,21 %	0,00 %	0,31 %	6,23 %	4,05 %	90,13 %
15	3,90 %	0,10 %	0,00 %	0,41 %	6,15 %	4,31 %	89,33 %
15	5,92 %	1,00 %	0,78 %	0,00 %	11,50 %	4,46 %	106,36 %
15	3,10 %	0,10 %	0,00 %	0,10 %	7,69 %	2,30 %	86,71 %

Sigue en la página siguiente.

$n = m$	$IRH1_{mak}$	$IRH2_{mak}$	$IRH3_{mak}$	$IRH4_{mak}$	$ALPHA_{mak} = 0$	$ALPHA_{mak} = 0.5$	$ALPHA_{mak} = 1$
15	0,92 %	0,10 %	0,10 %	0,00 %	3,37 %	5,32 %	92,84 %
15	0,61 %	0,00 %	0,00 %	0,61 %	2,94 %	1,22 %	111,27 %
15	2,14 %	0,53 %	0,00 %	0,00 %	7,06 %	0,00 %	72,83 %
15	6,06 %	0,92 %	0,10 %	0,00 %	10,37 %	3,70 %	113,45 %
15	0,80 %	1,59 %	0,00 %	0,10 %	6,37 %	2,79 %	87,25 %

Para realizar el análisis de los valores del indicador ARPD de todas las instancias se usarán los promedios para cada método de todos los ciento treinta casos planteados. Estudiando los resultados nos encontramos ante una situación similar a la de la función objetivo del apartado 6.1, uno de los modelos del método *BICH-MIH*, caso $\alpha = 1$ devuelve un valor significativamente alto de desviación promedio llegando al setenta y cinco por ciento, mientras que ninguna otra heurística alcanza a superar el 6,3 %. En relación con los otros métodos, lo último que cabe destacar es que las desviaciones para la segunda, tercera y cuarta heurística los ARPD promedio son inferiores al uno por ciento. Estos datos se muestran en la Tabla 6.14.

Tabla 6.14 ARPDs Promedio *Makespan*. [Fuente: Elaboración propia].

ARPDs Promedio *Makespan*

$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA = 0$	$ALPHA = 0{,}5$	$ALPHA = 1$
2,44 %	0,83 %	0,23 %	0,52 %	6,27 %	4,04 %	75,40 %

6.3 Análisis de resultados

La deducción de las conclusiones de este proyecto se hace a partir de los resultados obtenidos en la Figura 6.1 que compara el número de mejores soluciones obtenidas por método para las ciento treinta instancias creadas en el caso de que la función objetivo sea la minimización del *total completion time*, con el caso en el que evaluamos la minimización del *makespan*. Se aprecia que las heurísticas que peores resultados nos aportan son los distintos modelos del método BICH-MIH , mientras que la *IRH3* alcanza la mejor solución en el 75 % de los casos para ambas funciones objetivo. Los resultados númericos se pueden observar en la Tabla 6.15.

El algoritmo *BICH-MIH* es el menos eficiente entre el resto de métodos independientemente de la función objetivo que se elija, esto se debe a la baja complejidad de este algoritmo comparado con el resto. Podemos concluir que este método independientemente de la ponderación que se le dé con el parámetro α es ineficiente para el problema que estamos abordando.

Pasamos ahora al siguiente factor a analizar que es el tiempo computacional promedio, y se comparan los de ambas funciones objetivo para todos los métodos en la Tabla 6.16 y gráficamente en la Figura 6.2. No se ve nada destacable ya que los tiempos son similares (entre funciones objetivo), más allá del hecho de que el tiempo promedio para la *IRH3* en el caso de la función objetivo *Total Completion Time* es ligeramente superior al del caso *Makespan*.

Por otro lado, tenemos el análisis de los ARPD para ambos casos propuestos en el proyecto: $Om||Min\, C_j$ y $Om||Min\, \sum C_j$. Comparando los valores de la Tabla 6.17 lo que se puede observar de forma más destacable es que el método *BICH-MIH* con $\alpha = 0$ es el que obtiene el máximo valor de desviación para cada objetivo; para la minimización del tiempo total de terminación dicho valor es igual al 125,98 %. En paralelo es también uno de los modelos *BICH-MIH*, con $\alpha = 1$, el que obtiene el peor valor para la función objetivo *Makespan minimization* siendo este igual a 75,40 %. Se deduce que el papel que juega el parámetro α en el método *BICH-MIH* es determinante y los resultados que ofrezca dependerán de la función objetivo evaluada. Los métodos que menores desviaciones promedio tienen y por tanto considerados los mejores según este criterio (*ARPD*) son la *IRH2*, *IRH3* e *IRH4*, con valores inferiores al 2,3 % siendo esto un indicativo de la buena precisión y complejidad de los métodos por lo menos en el rango de tamaño de problemas examinado. Los ARPD promedio es una forma de analizar más en profundidad el primer factor expuesto, el número de mejores soluciones, ya que se puede dar la situación en la que un método consiga pocas mejores soluciones

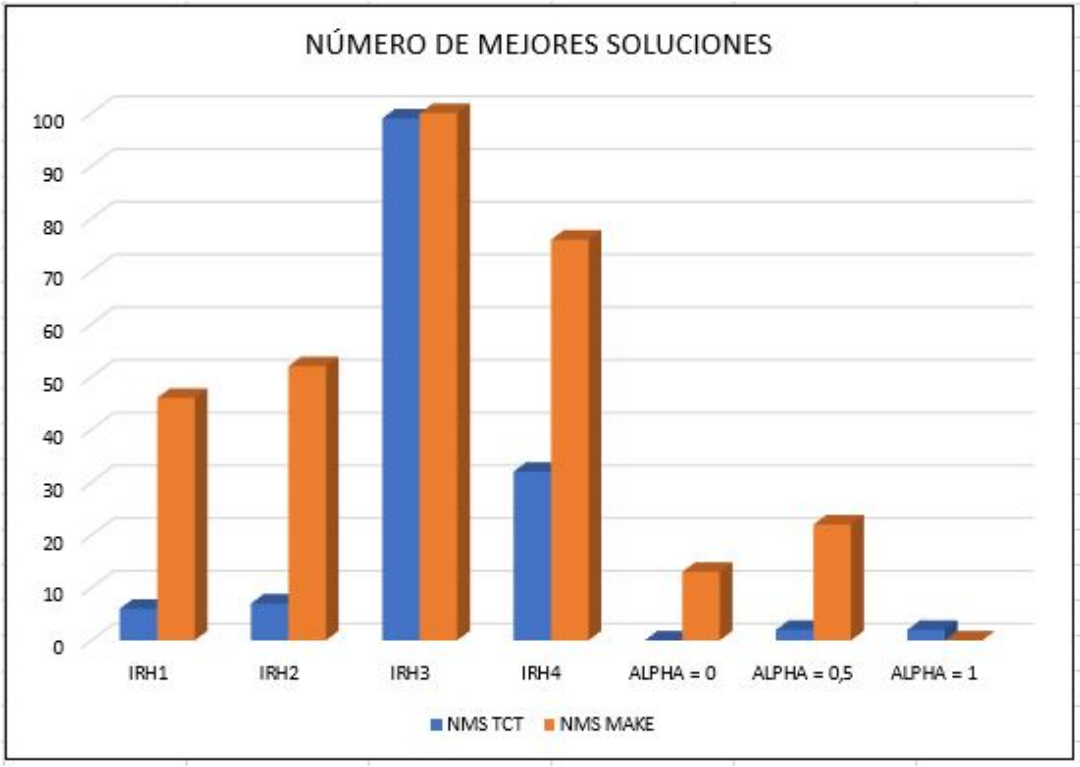

Figura 6.1 Número de mejores soluciones por método para cada función objetivo. [Fuente: Elaboración propia].

Tabla 6.15 Comparación del número de mejores soluciones entre ambas funciones objetivo. [Fuente: Elaboración propia].

Comparación del número de mejores soluciones

Casos	$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA = 0$	$ALPHA = 0{,}5$	$ALPHA = 1$
NMS TCT	6	7	99	32	0	2	2
NMS MAKE	46	52	100	76	13	22	0

Tabla 6.16 Comparación de tiempos computacionales promedio entre ambas funciones objetivo. [Fuente: Elaboración propia].

Comparación de tiempos computacionales promedios

Casos	$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA = 0$	$ALPHA = 0{,}5$	$ALPHA = 1$
CPU PROM TCT	0,0075	0,2068	0,2826	2,0467	0,0008	0,0009	0,0009
CPU PROM MAKE	0,0076	0,2093	0,2480	1,9967	0,0009	0,0009	0,0009

Tabla 6.17 Comparación de ARPD entre ambas funciones objetivo. [Fuente: Elaboración propia].

Comparación de ARPD promedio

Casos	$IRH1$	$IRH2$	$IRH3$	$IRH4$	$ALPHA = 0$	$ALPHA = 0{,}5$	$ALPHA = 1$
ARPD TCT	7,26 %	2,29 %	0,25 %	1,54 %	125,98 %	13,74 %	13,74 %
ARPD MAKE	2,44 %	0,83 %	0,23 %	0,52 %	6,27 %	4,04 %	75,40 %

y sin embargo la desviación del valor obtenido respecto a la mejor solución fuese prácticamente nula. Los valores exactos se muestran en la Tabla 6.17 y se aporta un esquema gráfico comparativo en la Figura 6.3.

Para poder sacar conclusiones de las heurísticas más convenientes para usar en el problema que planteamos entre las opciones analizadas el proceso será el siguiente, consideraremos los factores:

- Número de mejores soluciones obtenida.

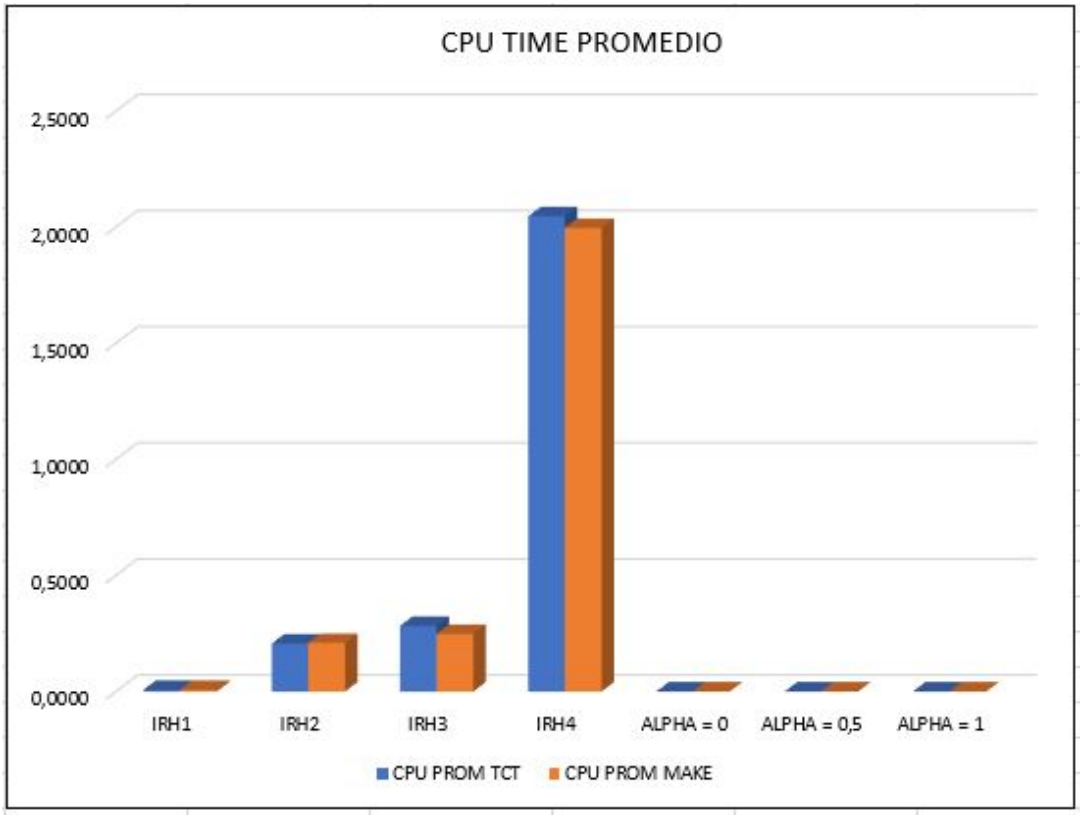

Figura 6.2 CPU times promedios por método para cada función objetivo. [Fuente: Elaboración propia].

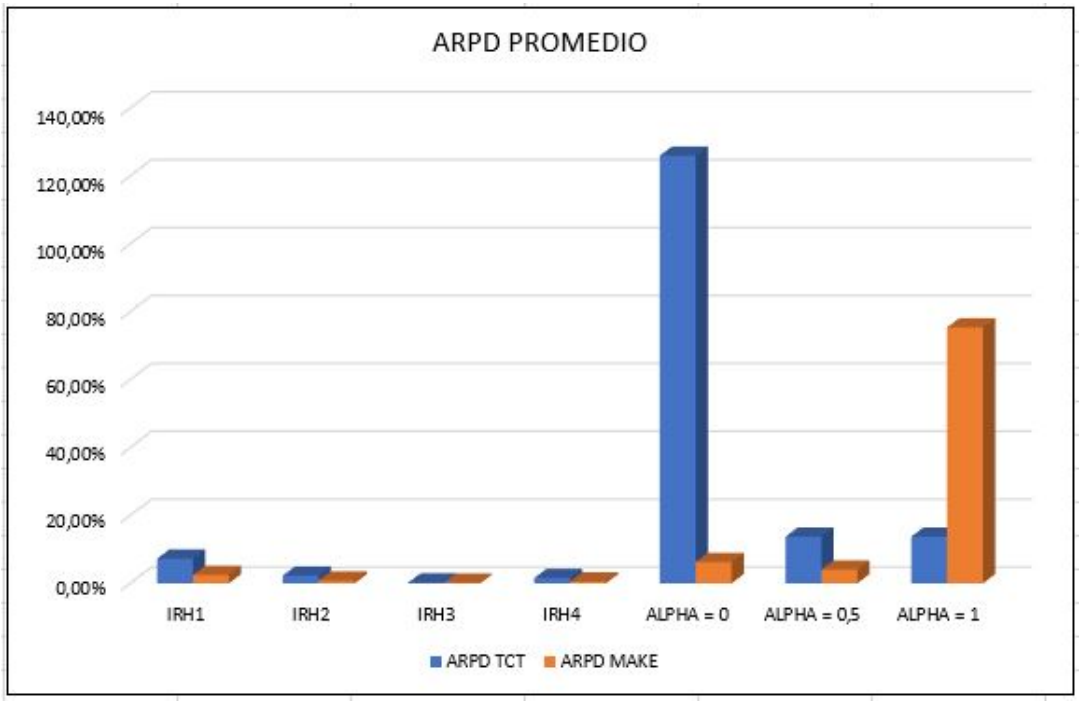

Figura 6.3 ARPD promedios por método para cada función objetivo. [Fuente: Elaboración propia].

- Coste computacional que conlleva usar este método.
- ARPD promedio.

Nos ceñiremos al tamaño de problemas expuesto para determinar esta decisión, es decir, n=m=3 hasta n=m=15. El objetivo es encontrar una heurística equilibrada, que no tarde excesivo tiempo en hallar la solución pero que a su vez aporte un buen resultado no muy desviado de la mejor opción que aportarían algoritmos más lentos.

Para ello se proponen diferentes modelos con pesos para ayudar en la toma de decisiones de que heurística es mejor según la importancia que se le otorgue a cada uno de los factores clave mencionados. Planteamos tres configuraciones distintas para los valores de los pesos, se detallará más adelante.

Para normalizar la ponderación de estos factores se necesita establecer un criterio que será el siguiente. El mejor número de óptimos posible a obtener corresponde a ciento treinta, al igual que un ARPD de 0 % y el tiempo promedio mínimo de entre todas las heurísticas serán los valores que mejor ponderación tengan, por tanto se les asigna a estos casos el valor uno. Y dependiendo de lo alejados que estén el resto de los valores en los métodos para cada factor se calculará su puntuación normalizada (interpolación lineal).

Los factores que son multiplicados por los pesos deberán de ser normalizados para poder comparar las magnitudes distintas:

$$\alpha * NMS_i + \beta * CPUTIME_i + \gamma * ARPD_i = INDICADOR_i \tag{6.2}$$

Siendo "*NMS*" el número de mejores soluciones (por si no quedaba clara la notación). Se evaluará está ecuación para las distintas combinaciones de valores de los parámetros que se propongan y saldrá el método con mejor puntuación. Debemos de realizar este proceso tanto para la función objetivo "*makespan minimization*" como para "*total completion time minimization*". La variable "*INDICADOR*" se asocia con el resultado de la ecuación y ofrecerá un valor para cada método evaluado, es importante determinar que el mejor método, independientemente de configuración de pesos que esté siendo usada, es siempre el que resulte en un mayor valor de la variable "*INDICADOR*".

En la Tabla 6.18 se proponen los diferentes casos de pesos a utilizar y que se van a someter a estudio. Lógicamente la suma de los pesos para cada caso debe ser igual a uno.

Tabla 6.18 Valores de los parámetros para cada caso analizado. [Fuente: Elaboración propia].

Caso	α	β	γ
1	0.4	0.1	0.5
2	0.3	0.2	0.5
3	0.3	0.4	0.5

- *TCT*: La evaluación de los métodos para la función objetivo dividido por factores se hace como se muestra en la Tabla 6.19.

Tabla 6.19 *Total Completion Time*: Ejemplo de valores para cada factor y método. [Fuente: Elaboración propia].

TOTALCOMPLETIONTIME				
	NMS	CPUTIME	ARPD	INDICADOR
IRH1	0,046	0,997	0,942	0,695
IRH2	0,054	0,899	0,982	0,670
IRH3	0,762	0,862	0,998	0,873
IRH4	0,246	0,000	0,988	0,370
ALPHA=0	0,000	1,000	0,000	0,400
ALPHA=0,5	0,015	1,000	0,891	0,672
ALPHA=1	0,015	1,000	0,891	0,672

Cada vez que alteramos los parámetros se modifican los resultados de las casillas de la columna "*INDICADOR*", porque su fórmula es dependiente de ellos. Ejecutando los tres modelos de pesos obtenemos los siguientes resultados, Tabla 6.20:

Tabla 6.20 *Total Completion Time*: Resultados de la variable *Indicador* para cada caso y método. [Fuente: Elaboración propia].

TOTAL COMPLETION TIME			
RESULTADOS			
	1	2	3
IRH1	0,589	0,684	0,695
IRH2	0,602	0,687	0,670
IRH3	0,890	0,900	0,873
IRH4	0,592	0,568	0,370
ALPHA=0	0,100	0,200	0,400
ALPHA=0,5	0,552	0,650	0,672
ALPHA=1	0,552	0,650	0,672

En este caso no hay ninguna duda de que la mejor heurística sería la *IRH3* ya que concluye con el mejor valor del indicador "*INDICADOR*" para todas las situaciones planteadas.

- *Makespan*: La evaluación de los métodos para la función objetivo dividido por factores se hace como se muestra en la Tabla 6.21.

Tabla 6.21 *Makespan*: Ejemplo de valores para cada factor y método. [Fuente: Elaboración propia].

MAKESPAN				
	NMS	CPUTIME	ARPD	INDICADOR
IRH1	0,354	0,997	0,968	0,795
IRH2	0,400	0,896	0,989	0,775
IRH3	0,769	0,876	0,997	0,880
IRH4	0,585	0,000	0,993	0,473
ALPHA=0	0,100	1,000	0,917	0,705
ALPHA=0,5	0,169	1,000	0,946	0,735
ALPHA=1	0,000	1,000	0,000	0,400

Sacando la tabla con los valores de "*INDICADOR*" mediante el mismo proceso que para la función objetivo previa, obtenemos los siguientes resultados, Tabla 6.22:

Tabla 6.22 *Makespan*: Resultados de la variable *Indicador* para cada caso y método. [Fuente: Elaboración propia].

MAKESPAN			
RESULTADOS			
	1	2	3
IRH1	0,725	0,789	0,795
IRH2	0,744	0,794	0,775
IRH3	0,894	0,904	0,880
IRH4	0,730	0,672	0,473
ALPHA=0	0,598	0,688	0,705
ALPHA=0,5	0,641	0,724	0,735
ALPHA=1	0,100	0,200	0,400

Como se aprecia, volvemos a obtener que la heurística con puntuación mayor del criterio que hemos seguido es la *IRH3*, distanciada por amplio margen del resto de métodos.

7 Conclusiones

En este proyecto se ha desarrollado el estudio de siete algoritmos de programación de la producción ya propuestos y adaptados de la literatura, para el caso de un taller de flujo abierto (*Openshop*) sin restricciones. Para el análisis de estos métodos se han examinado diez casos por cada tamaño distinto de problema desde tres trabajos en tres máquinas hasta quince trabajos a asignar en quince máquinas, sumando un total de ciento treinta casos analizados.

Podemos exponer como conclusiones y aportaciones del trabajo:

- Durante el proceso de codificación de las heurísticas se ha optimizado al máximo el coste computacional, consiguiendo tiempos de computación de los algoritmos más bajos que los mostrados en el artículo de referencia de donde extraemos los métodos.
- Con los resultados de cada función objetivo y su tiempo de computación, resultado de procesar los trece tamaños de problema en todos los modelos, se han podido analizar factores críticos para medir cuantitativamente la calidad de cada método, estos son: el número de mejores soluciones obtenidas, el tiempo computacional promedio y el ARPD promedio.
- Las conclusiones que podemos deducir del análisis de resultados de la sección 6.4 es que la heurística que mejores resultados nos ofrece para ambas funciones objetivo y en todos los casos de distintas prioridades que se le da a cada factor crítico es el método *IRH3*.
- Se propone para un posible trabajo futuro, tratar de implementar estas heurísticas en entornos más restrictivos realizando las correspondientes modificaciones y analizar de nuevo los resultados y los factores mencionados en la sección 6.3.

8 Anexo

8.1 Código C

```
using System;
using System.Diagnostics;
using System.Linq;
using System.Globalization;
using System.IO;
using System.Numerics;
using System.Reflection.PortableExecutable;
using System.Runtime.CompilerServices;
using System.Threading.Tasks.Dataflow;
using System.Xml.Schema;

namespace irh1
{
class Program
{
static void Main(string[] args)
{

int n, m;

String ruta = AppDomain.CurrentDomain.BaseDirectory;
ruta = ruta.Replace("\\", "/").ToString();
DirectoryInfo DIRESC = new DirectoryInfo(ruta + "/temp");
if (!DIRESC.Exists) DIRESC.Create();
String ficBatBorrar = ruta + "temp" + "/tabla_excel1.csv";
FileInfo FicheroEliminar = new FileInfo(ficBatBorrar);
if (FicheroEliminar.Exists) File.Delete(ficBatBorrar);
String ficCSV = ruta + "temp" + "/tabla_excel1.csv";
StreamWriter FicheroCSV = new StreamWriter(ficCSV, true);
FicheroCSV.Write("\n");
FicheroCSV.Write("\n");
FicheroCSV.Write("\n");
FicheroCSV.Write("\n");

for (n = 3; n < 16; n++)
{
m = n;
for (int k = 0; k < 10; k++)
{
```

```
int[,] processingtimes = new int[n, m];

Random aleatorio = new Random();
for (int i = 0; i < n; i++)
{
for (int j = 0; j < m; j++)
{
processingtimes[i, j] = aleatorio.Next(1, 99);
}
}
FicheroCSV.Write(";");

var timer = Stopwatch.StartNew();
int TCT1 = irh1(n, m, processingtimes);
double timeActualMilliseconds = timer.ElapsedMilliseconds;
double timeActual = timeActualMilliseconds / 1000;


var timer_1 = Stopwatch.StartNew();
int Makespan1 = irh1makespan(n, m, processingtimes);
double timeActualMilliseconds_1 = timer_1.ElapsedMilliseconds;
double timeActual_1 = timeActualMilliseconds_1 / 1000;


var timer1 = Stopwatch.StartNew();
int TCT2 = irh2(n, m, processingtimes);
double timeActualMilliseconds1 = timer1.ElapsedMilliseconds;
double timeActual1 = timeActualMilliseconds1 / 1000;


var timer1_1 = Stopwatch.StartNew();
int Makespan2 = irh2makespan(n, m, processingtimes);
double timeActualMilliseconds1_1 = timer1_1.ElapsedMilliseconds;
double timeActual1_1 = timeActualMilliseconds1_1 / 1000;


var timer2 = Stopwatch.StartNew();
int TCT3 = irh3(n, m, processingtimes);
double timeActualMilliseconds2 = timer2.ElapsedMilliseconds;
double timeActual2 = timeActualMilliseconds2 / 1000;


var timer2_1 = Stopwatch.StartNew();
int Makespan3 = irh3makespan(n, m, processingtimes);
double timeActualMilliseconds2_1 = timer2_1.ElapsedMilliseconds;
double timeActual2_1 = timeActualMilliseconds2_1 / 1000;


var timer3 = Stopwatch.StartNew();
int TCT4 = irh4(n, m, processingtimes);
double timeActualMilliseconds3 = timer3.ElapsedMilliseconds;
double timeActual3 = timeActualMilliseconds3 / 1000;


var timer3_1 = Stopwatch.StartNew();
int Makespan4 = irh4makespan(n, m, processingtimes);
```

```
double timeActualMilliseconds3_1 = timer3_1.ElapsedMilliseconds;
double timeActual3_1 = timeActualMilliseconds3_1 / 1000;

FicheroCSV.Write($"{TCT1};");
FicheroCSV.Write($"{TCT2};");
FicheroCSV.Write($"{TCT3};");
FicheroCSV.Write($"{TCT4};");

FicheroCSV.Write(";");

FicheroCSV.Write($"{Makespan1};");
FicheroCSV.Write($"{Makespan2};");
FicheroCSV.Write($"{Makespan3};");
FicheroCSV.Write($"{Makespan4};");

FicheroCSV.Write(";");

FicheroCSV.Write($"{timeActual};");
FicheroCSV.Write($"{timeActual1};");
FicheroCSV.Write($"{timeActual2};");
FicheroCSV.Write($"{timeActual3};");

FicheroCSV.Write(";");

FicheroCSV.Write($"{timeActual_1};");
FicheroCSV.Write($"{timeActual1_1};");
FicheroCSV.Write($"{timeActual2_1};");
FicheroCSV.Write($"{timeActual3_1};");

FicheroCSV.Write(";");

double alpha = 0;
for (int k1 = 0; k1 < 3; k1++)
{
var timer4_1 = Stopwatch.StartNew();
int TCT5 = BICH_MIH_TCT(n, m, processingtimes, alpha);
double timeActualMilliseconds4_1 = timer4_1.ElapsedMilliseconds;
double timeActual4_1 = timeActualMilliseconds4_1 / 1000;
FicheroCSV.Write($"{TCT5};");
FicheroCSV.Write($"{timeActual4_1};");

var timer4_2 = Stopwatch.StartNew();
int Makespan5 = BICH_MIH_Makespan(n, m, processingtimes, alpha);
double timeActualMilliseconds4_2 = timer4_2.ElapsedMilliseconds;
double timeActual4_2 = timeActualMilliseconds4_2 / 1000;
FicheroCSV.Write($"{Makespan5};");
FicheroCSV.Write($"{timeActual4_2};");

alpha = alpha + 0.5;
}
// fin

FicheroCSV.Write("\n");

}
Console.WriteLine($"Fin evaluación para {n} maquinas y trabajos en todas las heurísticas");
FicheroCSV.Write("\n");
```

```
}

FicheroCSV.Close();
}

static int irh3(int n, int m, int[,] processingtimes)
{
int contador = 0;
int r;
int k = 0;
int[] W = new int[n * m];

var array1 = new int[n * m];
int[,] Operationsij = new int[n * m, 4];
int[,] Operationsij_aux = new int[n * m, 2];
double[,] Elements = new double[2, n * m];
int[] TP_oper = new int[n * m];

instancias(ref array1, ref Operationsij_aux, ref processingtimes, ref TP_oper, ref Elements, n

int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];

for (int i = 0; i < n * m; i++)
{
W[i] = 0;
copiaW[i] = 0;
}

for (r = 0; r < n * m; r++)
{
int totalcompletiontime;
int bestTCT = 0;
int i, t, j;
int job, machine;
job = 0;
machine = 0;

k = array1[r];
job = Operationsij_aux[array1[r], 0];
machine = Operationsij_aux[array1[r], 1];

for (int rr = 0; rr < r + 1; rr++)
{
string redundancy = "no";
if (rr > 0 && job != Operationsij_aux[W[rr - 1], 0] && machine != Operationsij_aux[W[rr - 1],
{
redundancy = "yes";
}

else
{
if (rr == 0)
contador++;
```

```
copiavector(copiaW, W, n * m);
t = rr;
i = 1;

while (copiaW.Length - i != t)
{
copiaW[copiaW.Length - i] = copiaW[copiaW.Length - i - 1];
copiaW[copiaW.Length - i - 1] = 0;
i++;
}

copiaW[t] = k;

for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

totalcompletiontime = evaluarsecuenciairh2(copiaW, completiontimes, startingtimes, TP_oper, Operation

if (rr == 0)
{
bestTCT = totalcompletiontime;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
else if (totalcompletiontime < bestTCT)
{
bestTCT = totalcompletiontime;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
}
}

for (i = 0; i < n * m; i++)
{
W[i] = mejorsecuencia[i];
}

}

for (int j = 0; j < m; j++)
{
for (int i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

int mejorTCT = evaluarsecuencia(W, completiontimes, startingtimes, TP_oper, Operationsij_aux, n * m,

int[] secuencia = procedimientoirh3(W, n, m, Elements, TP_oper, Operationsij_aux, mejorTCT);
```

```
for (int j = 0; j < m; j++)
{
for (int i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

int TCT = evaluarsecuencia(secuencia, completiontimes, startingtimes, TP_oper, Operationsij_au

return TCT;
}

static int irh3makespan(int n, int m, int[,] processingtimes)
{
int contador = 0;
int r;
int k = 0;
int[] W = new int[n * m];

var array1 = new int[n * m];
int[,] Operationsij = new int[n * m, 4];
int[,] Operationsij_aux = new int[n * m, 2];
double[,] Elements = new double[2, n * m];
int[] TP_oper = new int[n * m];

instancias(ref array1, ref Operationsij_aux, ref processingtimes, ref TP_oper, ref Elements, n

int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];

for (int i = 0; i < n * m; i++)
{
W[i] = 0;
copiaW[i] = 0;
}

for (r = 0; r < n * m; r++)
{
int makespan;
int bestmakespan = 0;
int i, t, j;
int job, machine;
job = 0;
machine = 0;

k = array1[r];
job = Operationsij_aux[array1[r], 0];
machine = Operationsij_aux[array1[r], 1];

for (int rr = 0; rr < r + 1; rr++)
{
string redundancy = "no";
if (rr > 0 && job != Operationsij_aux[W[rr - 1], 0] && machine != Operationsij_aux[W[rr - 1],
```

```
{
redundancy = "yes";
}

else
{
if (rr == 0)
contador++;

copiavector(copiaW, W, n * m);
t = rr;
i = 1;

while (copiaW.Length - i != t)
{
copiaW[copiaW.Length - i] = copiaW[copiaW.Length - i - 1];
copiaW[copiaW.Length - i - 1] = 0;
i++;
}

copiaW[t] = k;

for (i = 0; i < n; i++)
{
for (j = 0; j < m; j++)
{
completiontimes[i, j] = 0;
startingtimes[i, j] = 0;
}
}

makespan = evaluarsecuenciairh2makespan(copiaW, completiontimes, startingtimes, TP_oper, Operationsij

if (rr == 0)
{
bestmakespan = makespan;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
else if (makespan < bestmakespan)
{
bestmakespan = makespan;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
}
}

for (i = 0; i < n * m; i++)
{
W[i] = mejorsecuencia[i];
}

}

for (int j = 0; j < m; j++)
{
for (int i = 0; i < n; i++)
{
```

```
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

int mejorTCT = evaluarsecuenciamakespan(W, completiontimes, startingtimes, TP_oper, Operationsi

int[] secuencia = procedimientoirh3makespan(W, n, m, Elements, TP_oper, Operationsij_aux, mejo

for (int j = 0; j < m; j++)
{
for (int i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

int TCT = evaluarsecuenciamakespan(secuencia, completiontimes, startingtimes, TP_oper, Operatio

return TCT;
}

static void instancias(ref int[] array1, ref int[,] Operationsij_aux, ref int[,] processingtime
{
int s = 0;
int k = 0;

for (int i = 0; i < n; i++)
{
for (int j = 0; j < m; j++)
{
TP_oper[k] = processingtimes[i, j];
array1[k] = processingtimes[i, j];
Operationsij_aux[s, 0] = i;
Operationsij_aux[s, 1] = j;
Elements[0, k] = k;
Elements[1, k] = processingtimes[i, j];
k++;
s++;
}
}

double[,] ElementsSalida = Quicksort(Elements, 0, n * m - 1);
for (int j = 0; j < n * m; j++)
{
array1[j] = (int)ElementsSalida[0, j];
}
}

static int[] procedimientoirh3(int[] secuencia, int n, int m, double[,] Elements, int[] TP_oper
{
int TCT = TCTW;
int r = n * m;
string improvement = "yes";
while (r > 0)
{
```

```
int[] copiaW = new int[n * m];
int[] WW = new int[n * m - 1];
int[] mejorsecuencia = new int[n * m];
improvement = "no";
int jobcolocar = Operationsij_aux[secuencia[r - 1], 0];
int machinecolocar = Operationsij_aux[secuencia[r - 1], 1];
int FILA = secuencia[r - 1];
int r3 = 0;

for (int r2 = 0; r2 < n * m; r2++)
{
string redundancy = "no";
if (r2 > 0)
{

if (r2 > 0 && jobcolocar != Operationsij_aux[secuencia[r2 - 1], 0] && machinecolocar != Operationsij_
{
redundancy = "yes";
}
}

if (redundancy == "no")
{
int z = r2;
Array.Resize(ref WW, copiaW.Length - 1);
copiavector(copiaW, secuencia, n * m);
extractIVector(copiaW, WW, copiaW.Length, r - 1);
Array.Resize(ref WW, WW.Length + 1);
int i = 1;

while (WW.Length - i != z)
{
WW[WW.Length - i] = WW[WW.Length - i - 1];
WW[WW.Length - i - 1] = 0;
i++;
}

WW[z] = secuencia[r - 1];
int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];

for (int j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

int TCTWW = evaluarsecuencia(WW, completiontimes, startingtimes, TP_oper, Operationsij_aux, secuencia

if (TCTWW < TCTW)
{
copiavector(mejorsecuencia, WW, n * m);
improvement = "yes";
TCTW = TCTWW;
```

```
r3 = r2;
}
}

}

if (improvement == "yes")
{
copiavector(secuencia, mejorsecuencia, mejorsecuencia.Length);
r = n * m;
}

if (improvement == "no")
{
r = r - 1;
}
}

return secuencia;
}

static int[] procedimientoirh3makespan(int[] secuencia, int n, int m, double[,] Elements, int[]
{
int r = n * m;
string improvement = "yes";
while (r > 0)
{
int[] copiaW = new int[n * m];
int[] WW = new int[n * m - 1];
int[] mejorsecuencia = new int[n * m];
improvement = "no";
int jobcolocar = Operationsij_aux[secuencia[r - 1], 0];
int machinecolocar = Operationsij_aux[secuencia[r - 1], 1];
int FILA = secuencia[r - 1];
int r3 = 0;

for (int r2 = 0; r2 < n * m; r2++)
{
string redundancy = "no";
if (r2 > 0)
{

if (r2 > 0 && jobcolocar != Operationsij_aux[secuencia[r2 - 1], 0] && machinecolocar != Operati
{
redundancy = "yes";
}
}

if (redundancy == "no")
{
int z = r2;
Array.Resize(ref WW, copiaW.Length - 1);
copiavector(copiaW, secuencia, n * m);
extractIVector(copiaW, WW, copiaW.Length, r - 1);
Array.Resize(ref WW, WW.Length + 1);
int i = 1;
```

```
while (WW.Length - i != z)
{
WW[WW.Length - i] = WW[WW.Length - i - 1];
WW[WW.Length - i - 1] = 0;
i++;
}

WW[z] = secuencia[r - 1];
int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];

for (int j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

int MakespanWW = evaluarsecuenciamakespan(WW, completiontimes, startingtimes, TP_oper, Operationsij_a

if (MakespanWW < MakespanW)
{
copiavector(mejorsecuencia, WW, n * m);
improvement = "yes";
MakespanW = MakespanWW;
r3 = r2;
}
}

}

if (improvement == "yes")
{
copiavector(secuencia, mejorsecuencia, mejorsecuencia.Length);
r = n * m;
}

if (improvement == "no")
{
r = r - 1;
}
}

return secuencia;
}

static void extractIVector(int[] source, int[] destination, int len, int pos)
{
for (int i = 0; i < len - 1; i++)
{
if (i < pos)
{
destination[i] = source[i];
}
```

```
else
{
destination[i] = source[i + 1];
}
}
}

static int irh2(int n, int m, int[,] processingtimes)
{
int contador = 0;
int r;
int k = 0;
int[] W = new int[n * m];
int bestTCT = 0;

var array1 = new int[n * m];
int[,] Operationsij = new int[n * m, 4];
int[,] Operationsij_aux = new int[n * m, 2];
double[,] Elements = new double[2, n * m];
int[] TP_oper = new int[n * m];

instancias(ref array1, ref Operationsij_aux, ref processingtimes, ref TP_oper, ref Elements, n

int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];

for (int i = 0; i < n * m; i++)
{
W[i] = 0;
copiaW[i] = 0;
}

for (r = 0; r < n * m; r++)
{
int totalcompletiontime;

int i, t, j;
int job, machine;
job = 0;
machine = 0;

k = array1[r];
job = Operationsij_aux[array1[r], 0];
machine = Operationsij_aux[array1[r], 1];

for (int rr = 0; rr < r + 1; rr++)
{
string redundancy = "no";

if (rr > 0 && job != Operationsij_aux[W[rr - 1], 0] && machine != Operationsij_aux[W[rr - 1],
{
redundancy = "yes";
}

else
```

```
{
if (rr == 0)
contador++;

copiavector(copiaW, W, n * m);
t = rr;
i = 1;

while (copiaW.Length - i != t)
{
copiaW[copiaW.Length - i] = copiaW[copiaW.Length - i - 1];
copiaW[copiaW.Length - i - 1] = 0;
i++;
}

copiaW[t] = k;

for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

totalcompletiontime = evaluarsecuenciairh2(copiaW, completiontimes, startingtimes, TP_oper, Operation

if (rr == 0)
{
bestTCT = totalcompletiontime;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
else if (totalcompletiontime < bestTCT)
{
bestTCT = totalcompletiontime;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
}

}

for (i = 0; i < n * m; i++)
{
W[i] = mejorsecuencia[i];
}
}

return bestTCT;
}

static int irh2makespan(int n, int m, int[,] processingtimes)
{
int contador = 0;
int r;
int k = 0;
int[] W = new int[n * m];
```

```
int bestmakespan = 0;

var array1 = new int[n * m];
int[,] Operationsij = new int[n * m, 4];
int[,] Operationsij_aux = new int[n * m, 2];
double[,] Elements = new double[2, n * m];
int[] TP_oper = new int[n * m];

instancias(ref array1, ref Operationsij_aux, ref processingtimes, ref TP_oper, ref Elements, n

int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];

for (int i = 0; i < n * m; i++)
{
W[i] = 0;
copiaW[i] = 0;
}

for (r = 0; r < n * m; r++)
{
int makespan;

int i, t, j;
int job, machine;
job = 0;
machine = 0;

k = array1[r];
job = Operationsij_aux[array1[r], 0];
machine = Operationsij_aux[array1[r], 1];

for (int rr = 0; rr < r + 1; rr++)
{
string redundancy = "no";

if (rr > 0 && job != Operationsij_aux[W[rr - 1], 0] && machine != Operationsij_aux[W[rr - 1],
{
redundancy = "yes";
}

else
{
if (rr == 0)
contador++;

copiavector(copiaW, W, n * m);
t = rr;
i = 1;

while (copiaW.Length - i != t)
{
copiaW[copiaW.Length - i] = copiaW[copiaW.Length - i - 1];
copiaW[copiaW.Length - i - 1] = 0;
i++;
```

```
}

copiaW[t] = k;

for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

makespan = evaluarsecuenciairh2makespan(copiaW, completiontimes, startingtimes, TP_oper, Operationsij

if (rr == 0)
{
bestmakespan = makespan;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
else if (makespan < bestmakespan)
{
bestmakespan = makespan;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
}

}

for (i = 0; i < n * m; i++)
{
W[i] = mejorsecuencia[i];
}
}

return bestmakespan;
}

static int irh1(int n, int m, int[,] processingtimes)
{
int contador = 0;
int r;
int k = 0;
int[] W = new int[n * m];

var array1 = new int[n * m];
int[,] Operationsij = new int[n * m, 4];
int[,] Operationsij_aux = new int[n * m, 2];
double[,] Elements = new double[2, n * m];
int[] TP_oper = new int[n * m];

instancias(ref array1, ref Operationsij_aux, ref processingtimes, ref TP_oper, ref Elements, n, m);

int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];
```

```
for (int i = 0; i < n * m; i++)
{
W[i] = 0;
copiaW[i] = 0;
}

for (r = 0; r < n * m; r++)
{
int totalcompletiontime;
int bestTCT = 0;
int i, t, j;
int job, machine;
job = 0;
machine = 0;

k = array1[r];
job = Operationsij_aux[array1[r], 0];
machine = Operationsij_aux[array1[r], 1];

for (int rr = 0; rr < r + 1; rr++)
{
string redundancy = "no";
if (rr > 0 && job != Operationsij_aux[W[rr - 1], 0] && machine != Operationsij_aux[W[rr - 1],
{
redundancy = "yes";
}

else
{
copiavector(copiaW, W, n * m);
t = rr;
i = 1;

while (copiaW.Length - i != t)
{
copiaW[copiaW.Length - i] = copiaW[copiaW.Length - i - 1];
copiaW[copiaW.Length - i - 1] = 0;
i++;
}
copiaW[t] = k;
contador = r + 1;
for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

totalcompletiontime = evaluarsecuencia(copiaW, completiontimes, startingtimes, TP_oper, Operati

if (rr == 0)
{
bestTCT = totalcompletiontime;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
```

```
}

else if (totalcompletiontime < bestTCT)
{
bestTCT = totalcompletiontime;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
}
}

for (i = 0; i < n * m; i++)
{
W[i] = mejorsecuencia[i];
}
}

for (int j = 0; j < m; j++)
{
for (int i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

int mejorTCT = evaluarsecuencia(W, completiontimes, startingtimes, TP_oper, Operationsij_aux, n * m,

return mejorTCT;
}

static int irh1makespan(int n, int m, int[,] processingtimes)
{
int contador = 0;
int r;
int k = 0;
int[] W = new int[n * m];

var array1 = new int[n * m];
int[,] Operationsij = new int[n * m, 4];
int[,] Operationsij_aux = new int[n * m, 2];
double[,] Elements = new double[2, n * m];
int[] TP_oper = new int[n * m];

instancias(ref array1, ref Operationsij_aux, ref processingtimes, ref TP_oper, ref Elements, n, m);

int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];

for (int i = 0; i < n * m; i++)
{
W[i] = 0;
copiaW[i] = 0;
}

for (r = 0; r < n * m; r++)
```

```
{
int makespan;
int bestmakespan = 0;
int i, t, j;
int job, machine;
job = 0;
machine = 0;

k = array1[r];
job = Operationsij_aux[array1[r], 0];
machine = Operationsij_aux[array1[r], 1];

for (int rr = 0; rr < r + 1; rr++)
{
string redundancy = "no";
if (rr > 0 && job != Operationsij_aux[W[rr - 1], 0] && machine != Operationsij_aux[W[rr - 1], 1
{
redundancy = "yes";
}

else
{
copiavector(copiaW, W, n * m);
t = rr;
i = 1;

while (copiaW.Length - i != t)
{
copiaW[copiaW.Length - i] = copiaW[copiaW.Length - i - 1];
copiaW[copiaW.Length - i - 1] = 0;
i++;
}
copiaW[t] = k;
contador = r + 1;
for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

makespan = evaluarsecuenciamakespan(copiaW, completiontimes, startingtimes, TP_oper, Operations

if (rr == 0)
{
bestmakespan = makespan;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}

else if (makespan < bestmakespan)
{
bestmakespan = makespan;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
}
}
```

```
}

for (i = 0; i < n * m; i++)
{
W[i] = mejorsecuencia[i];
}
}

for (int j = 0; j < m; j++)
{
for (int i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}

int mejormakespan = evaluarsecuenciamakespan(W, completiontimes, startingtimes, TP_oper, Operationsij

return mejormakespan;
}

static int irh4(int n, int m, int[,] processingtimes)
{
int TCTprevio = 0;
int contador = 0;
int r;
int k = 0;
int[] W = new int[n * m];

var array1 = new int[n * m];
int[,] Operationsij = new int[n * m, 4];
int[,] Operationsij_aux = new int[n * m, 2];
double[,] Elements = new double[2, n * m];
int[] TP_oper = new int[n * m];

instancias(ref array1, ref Operationsij_aux, ref processingtimes, ref TP_oper, ref Elements, n, m);

int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];

for (int i = 0; i < n * m; i++)
{
W[i] = 0;
copiaW[i] = 0;
}

for (r = 0; r < n * m; r++)
{
int totalcompletiontime = 0;
int p = 0;
int i, t, j;
int job, machine;
int Indice;
```

```
Indice = array1[r];
job = Operationsij_aux[array1[r], 0];
machine = Operationsij_aux[array1[r], 1];

for (int r2 = 0; r2 < r + 1; r2++)
{
string redundancy = "no";
if (r2 > 0 && job != Operationsij_aux[W[r2 - 1], 0] && machine != Operationsij_aux[W[r2 - 1], 1
{
redundancy = "yes";
}

else
{

for (i = 0; i < copiaW.Length; i++)
{
copiaW[i] = W[i];
}

t = r2;
i = 1;

while (copiaW.Length - i != t)
{
copiaW[copiaW.Length - i] = copiaW[copiaW.Length - i - 1];
copiaW[copiaW.Length - i - 1] = 0;
i++;
}

copiaW[t] = Indice;
contador = r + 1;

for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}


int TCT1 = evaluarsecuenciairh2(copiaW, completiontimes, startingtimes, TP_oper, Operationsij_a

if (r2 == 0)
{
totalcompletiontime = TCT1;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
p = r2;
}

else if (TCT1 < totalcompletiontime)
{
totalcompletiontime = TCT1;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
```

```
p = r2;
}
}
}

TCTprevio = totalcompletiontime;

for (i = 0; i < n * m; i++)
{
W[i] = mejorsecuencia[i];
}

int[] WW = new int[n * m];
k = 5;

for (int r3 = Math.Max(0, p - k); r3 < Math.Min(r + 1, p + k); r3++)
{

int Aux = W[r3];
int job2 = Operationsij_aux[Aux, 0];
int machine2 = Operationsij_aux[Aux, 1];

Array.Resize(ref WW, copiaW.Length - 1);
copiavector(copiaW, W, n * m);
Indice = copiaW[r3];
extractIVector(copiaW, WW, copiaW.Length, r3);
Array.Resize(ref WW, WW.Length + 1);

for (int r4 = 0; r4 < r + 1; r4++)
{
if (r4 > 0 && job != Operationsij_aux[W[r4 - 1], 0] && machine != Operationsij_aux[W[r4 - 1], 1])
{
string redundancy = "yes";
}

else
{
i = 1;
while (WW.Length - i != r4)
{
WW[WW.Length - i] = WW[WW.Length - i - 1];
WW[WW.Length - i - 1] = 0;
i++;
}

WW[r4] = Indice;
contador = r + 1;

for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}
```

```
int Totalcompletiontime = evaluarsecuenciairh2(WW, completiontimes, startingtimes, TP_oper, Ope

if (Totalcompletiontime < TCTprevio)
{
TCTprevio = Totalcompletiontime;
copiavector(mejorsecuencia, WW, n * m);
}
}

copiavector(copiaW, W, copiaW.Length);
Array.Resize(ref WW, WW.Length - 1);
extractIVector(copiaW, WW, copiaW.Length, r3);
Array.Resize(ref WW, WW.Length + 1);
}

copiavector(W, mejorsecuencia, n * m);
}
}

return TCTprevio;
}

static int irh4makespan(int n, int m, int[,] processingtimes)
{
int makespanprevio = 0;
int contador = 0;
int r;
int k = 0;
int[] W = new int[n * m];

var array1 = new int[n * m];
int[,] Operationsij = new int[n * m, 4];
int[,] Operationsij_aux = new int[n * m, 2];
double[,] Elements = new double[2, n * m];
int[] TP_oper = new int[n * m];

instancias(ref array1, ref Operationsij_aux, ref processingtimes, ref TP_oper, ref Elements, n

int[,] completiontimes = new int[n, m];
int[,] startingtimes = new int[n, m];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];

for (int i = 0; i < n * m; i++)
{
W[i] = 0;
copiaW[i] = 0;
}

for (r = 0; r < n * m; r++)
{
int mejormakespan = 0;
int p = 0;
int i, t, j;
int job, machine;
int Indice;
```

```
Indice = array1[r];
job = Operationsij_aux[array1[r], 0];
machine = Operationsij_aux[array1[r], 1];

for (int r2 = 0; r2 < r + 1; r2++)
{
string redundancy = "no";
if (r2 > 0 && job != Operationsij_aux[W[r2 - 1], 0] && machine != Operationsij_aux[W[r2 - 1], 1])
{
redundancy = "yes";
}

else
{

for (i = 0; i < copiaW.Length; i++)
{
copiaW[i] = W[i];
}

t = r2;
i = 1;

while (copiaW.Length - i != t)
{
copiaW[copiaW.Length - i] = copiaW[copiaW.Length - i - 1];
copiaW[copiaW.Length - i - 1] = 0;
i++;
}

copiaW[t] = Indice;
contador = r + 1;

for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}


int makespan1 = evaluarsecuenciairh2makespan(copiaW, completiontimes, startingtimes, TP_oper, Operati

if (r2 == 0)
{
mejormakespan = makespan1;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
p = r2;
}

else if (makespan1 < mejormakespan)
{
mejormakespan = makespan1;
copiavector(mejorsecuencia, copiaW, copiaW.Length);
```

```
p = r2;
}
}
}

makespanprevio = mejormakespan;

for (i = 0; i < n * m; i++)
{
W[i] = mejorsecuencia[i];
}

int[] WW = new int[n * m];
k = 5;

for (int r3 = Math.Max(0, p - k); r3 < Math.Min(r + 1, p + k); r3++)
{

int Aux = W[r3];
int job2 = Operationsij_aux[Aux, 0];
int machine2 = Operationsij_aux[Aux, 1];

Array.Resize(ref WW, copiaW.Length - 1);
copiavector(copiaW, W, n * m);
Indice = copiaW[r3];
extractIVector(copiaW, WW, copiaW.Length, r3);
Array.Resize(ref WW, WW.Length + 1);

for (int r4 = 0; r4 < r + 1; r4++)
{
if (r4 > 0 && job != Operationsij_aux[W[r4 - 1], 0] && machine != Operationsij_aux[W[r4 - 1],
{
string redundancy = "yes";
}

else
{
i = 1;
while (WW.Length - i != r4)
{
WW[WW.Length - i] = WW[WW.Length - i - 1];
WW[WW.Length - i - 1] = 0;
i++;
}

WW[r4] = Indice;
contador = r + 1;

for (j = 0; j < m; j++)
{
for (i = 0; i < n; i++)
{
startingtimes[i, j] = 0;
completiontimes[i, j] = 0;
}
}
```

```
int Makespan = evaluarsecuenciairh2makespan(WW, completiontimes, startingtimes, TP_oper, Operationsij

if (Makespan < makespanprevio)
{
makespanprevio = Makespan;
copiavector(mejorsecuencia, WW, n * m);
}
}

copiavector(copiaW, W, copiaW.Length);
Array.Resize(ref WW, WW.Length - 1);
extractIVector(copiaW, WW, copiaW.Length, r3);
Array.Resize(ref WW, WW.Length + 1);
}

copiavector(W, mejorsecuencia, n * m);
}
}

return makespanprevio;
}

static void copiavector(int[] vector1, int[] vector2, int length)
{
for (int i = 0; i < length; i++)
{
vector1[i] = vector2[i];
}
}

static void copiamatriz(int[,] matriz1, int[,] matriz2, int n, int m)
{
for (int i = 0; i < n; i++)
{
for (int j = 0; j < m; j++)
{
matriz1[i, j] = matriz2[i, j];
}
}
}

static int evaluarsecuencia(int[] secuencia, int[,] completiontimes, int[,] startingtimes, int[] TP_O
{
int[,] avalaibletimes = new int[m, n + 1];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];
int[] trabajospormaquina = new int[m];
int[] completiontimesjob = new int[n];

for (int j = 0; j < m; j++)
{
trabajospormaquina[j] = 0;
for (int i = 0; i < n; i++)
{
avalaibletimes[j, i] = 0;
}
}
```

```
for (int j = 0; j < m; j++)
{
avalaibletimes[j, n] = 0;
}

for (int i = 0; i < n; i++)
{
completiontimesjob[i] = 0;
}

for (int k = 0; k < lengtheval; k++)
{
int completiontimemaxjob;
int job = Operationsij_aux[secuencia[k], 0];
int machine = Operationsij_aux[secuencia[k], 1];
int PO = TP_Oper[secuencia[k]];

completiontimemaxjob = completiontimesjob[job];
startingtimes[job, machine] = Math.Max(avalaibletimes[machine, trabajospormaquina[machine]], co
completiontimes[job, machine] = startingtimes[job, machine] + PO;
trabajospormaquina[machine]++;
avalaibletimes[machine, trabajospormaquina[machine]] = startingtimes[job, machine] + PO;
completiontimesjob[job] = completiontimes[job, machine];
}

int TCT = 0;
for (int i = 0; i < n; i++)
{
TCT = TCT + completiontimesjob[i];
}
return TCT;
}

static int evaluarsecuenciamakespan(int[] secuencia, int[,] completiontimes, int[,] startingti
{
int[,] avalaibletimes = new int[m, n + 1];
int[] copiaW = new int[n * m];
int[] mejorsecuencia = new int[n * m];
int[] trabajospormaquina = new int[m];
int[] completiontimesjob = new int[n];

for (int j = 0; j < m; j++)
{
trabajospormaquina[j] = 0;
for (int i = 0; i < n; i++)
{
avalaibletimes[j, i] = 0;
}
}

for (int j = 0; j < m; j++)
{
avalaibletimes[j, n] = 0;
}

for (int i = 0; i < n; i++)
```

```
{
completiontimesjob[i] = 0;
}

for (int k = 0; k < lengtheval; k++)
{
int completiontimemaxjob;
int job = Operationsij_aux[secuencia[k], 0];
int machine = Operationsij_aux[secuencia[k], 1];
int PO = TP_Oper[secuencia[k]];

completiontimemaxjob = completiontimesjob[job];
startingtimes[job, machine] = Math.Max(avalaibletimes[machine, trabajospormaquina[machine]], completi
completiontimes[job, machine] = startingtimes[job, machine] + PO;
trabajospormaquina[machine]++;
avalaibletimes[machine, trabajospormaquina[machine]] = startingtimes[job, machine] + PO;
completiontimesjob[job] = completiontimes[job, machine];
}

int makespan = completiontimesjob.Max();

return makespan;

}

static int evaluarsecuenciairh2(int[] secuencia, int[,] completiontimes, int[,] startingtimes, int[]
{

int[,] avalaibletimes = new int[m, n + 1];
int[] copiaW = new int[lengtheval];
int[] mejorsecuencia = new int[lengtheval];
int[] trabajospormaquina = new int[m];
int[] completiontimesjob = new int[n];
int[] operacioneslibres = new int[lengtheval];

copiavector(copiaW, secuencia, lengtheval);
copiavector(operacioneslibres, copiaW, copiaW.Length);

for (int j = 0; j < m; j++)
{
trabajospormaquina[j] = 0;
for (int i = 0; i < n; i++)
{
avalaibletimes[j, i] = 0;
}
}

for (int j = 0; j < m; j++)
{
avalaibletimes[j, n] = 0;
}

for (int i = 0; i < n; i++)
{
completiontimesjob[i] = 0;
}
```

```
for (int k = 0; k < lengtheval; k++)
{
int Operelegida = 0;
int st_aux;
int completiontimemaxjob;
int job;
int machine;
int mejorst = 0;
int pos = 0;

for (int kk = 0; kk < operacioneslibres.Length; kk++)
{
job = Operationsij_aux[operacioneslibres[kk], 0];
completiontimemaxjob = completiontimesjob[job];
machine = Operationsij_aux[operacioneslibres[kk], 1];
st_aux = Math.Max(avalaibletimes[machine, trabajospormaquina[machine]], completiontimemaxjob);

if (kk == 0)
{
mejorst = st_aux;
Operelegida = operacioneslibres[kk];
pos = kk;
mejorsecuencia[k] = Operelegida;
}

if (st_aux < mejorst)
{
mejorst = st_aux;
Operelegida = operacioneslibres[kk];
pos = kk;
mejorsecuencia[k] = Operelegida;
}
}

operacioneslibres = extractIVector2(operacioneslibres, operacioneslibres.Length, pos);

job = Operationsij_aux[Operelegida, 0];
machine = Operationsij_aux[Operelegida, 1];
int PO = TP_Oper[Operelegida];

startingtimes[job, machine] = mejorst;
completiontimes[job, machine] = startingtimes[job, machine] + PO;
trabajospormaquina[machine]++;
avalaibletimes[machine, trabajospormaquina[machine]] = startingtimes[job, machine] + PO;
completiontimesjob[job] = completiontimes[job, machine];

}

copiavector(secuencia, mejorsecuencia, mejorsecuencia.Length);

int TCT = 0;
for (int i = 0; i < n; i++)
{
TCT = TCT + completiontimesjob[i];
}

return TCT;
```

```
}

static int evaluarsecuenciairh2makespan(int[] secuencia, int[,] completiontimes, int[,] startingtimes
{

int[,] avalaibletimes = new int[m, n + 1];
int[] copiaW = new int[lengtheval];
int[] mejorsecuencia = new int[lengtheval];
int[] trabajospormaquina = new int[m];
int[] completiontimesjob = new int[n];
int[] operacioneslibres = new int[lengtheval];

copiavector(copiaW, secuencia, lengtheval);
copiavector(operacioneslibres, copiaW, copiaW.Length);

for (int j = 0; j < m; j++)
{
trabajospormaquina[j] = 0;
for (int i = 0; i < n; i++)
{
avalaibletimes[j, i] = 0;
}
}

for (int j = 0; j < m; j++)
{
avalaibletimes[j, n] = 0;
}

for (int i = 0; i < n; i++)
{
completiontimesjob[i] = 0;
}

for (int k = 0; k < lengtheval; k++)
{
int Operelegida = 0;
int st_aux;
int completiontimemaxjob;
int job;
int machine;
int mejorst = 0;
int pos = 0;

for (int kk = 0; kk < operacioneslibres.Length; kk++)
{
job = Operationsij_aux[operacioneslibres[kk], 0];
completiontimemaxjob = completiontimesjob[job];
machine = Operationsij_aux[operacioneslibres[kk], 1];
st_aux = Math.Max(avalaibletimes[machine, trabajospormaquina[machine]], completiontimemaxjob);

if (kk == 0)
{
mejorst = st_aux;
Operelegida = operacioneslibres[kk];
pos = kk;
mejorsecuencia[k] = Operelegida;
```

```
}

if (st_aux < mejorst)
{
mejorst = st_aux;
Operelegida = operacioneslibres[kk];
pos = kk;
mejorsecuencia[k] = Operelegida;
}
}

operacioneslibres = extractIVector2(operacioneslibres, operacioneslibres.Length, pos);

job = Operationsij_aux[Operelegida, 0];
machine = Operationsij_aux[Operelegida, 1];
int PO = TP_Oper[Operelegida];

startingtimes[job, machine] = mejorst;
completiontimes[job, machine] = startingtimes[job, machine] + PO;
trabajospormaquina[machine]++;
avalaibletimes[machine, trabajospormaquina[machine]] = startingtimes[job, machine] + PO;
completiontimesjob[job] = completiontimes[job, machine];

}

copiavector(secuencia, mejorsecuencia, mejorsecuencia.Length);

int makespan = completiontimesjob.Max();

return makespan;
}

static int[] extractIVector2(int[] source, int len, int pos)
{
int[] destination = new int[len - 1];
for (int i = 0; i < len - 1; i++)
{
if (i < pos)
{
destination[i] = source[i];
}

else
{
destination[i] = source[i + 1];
}
}
return destination;
}

static int BICH_MIH_TCT(int n, int m, int[,] processingtimes, double alpha)
{
int J, M;
int fo1, fo2, idletime;
int componente;
double FO, FOprevia;
int[] array1 = new int[n * m];
```

```
int[] secuenciafinal = new int[n * m];
int[] mejorsecuencia = new int[n * m];
int[] operaciones = new int[n * m];
int[,] freeoperations = new int[m, n];
int[] completiontimesmaxmachine = new int[m];
int[] completiontimesmaxmachine_aux = new int[m];
int[,] Operationsij = new int[n * m, 2];
int[,] avalaibletimes = new int[m, n + 1];
int[] trabajospormaquina = new int[m];
int[] completiontimesjob = new int[n];
int[,] startingtimes = new int[n, m];
int[,] completiontimes = new int[n, m];
int prevalueCTJ, prevalueCT, prevalueST, prevalueAT, prevalueCTM, prevalueTxM;
int contador = 0;
int mejoroperacion = 0;
int aux = 1;
int fila = 1;
componente = 0;
prevalueCTJ = prevalueCT = prevalueST = prevalueAT = prevalueCTM = prevalueTxM = 0;

for (int i = 0; i < n; i++)
{
completiontimesjob[i] = 0;
for (int j = 0; j < m; j++)
{
completiontimes[i, j] = 0;
startingtimes[i, j] = 0;
}
}
for (int j = 0; j < m; j++)
{
completiontimesmaxmachine[j] = 0;
trabajospormaquina[j] = 0;
avalaibletimes[j, n] = 0;
for (int i = 0; i < n; i++)
{
avalaibletimes[j, i] = 0;
}
}

Random aleatorio = new Random();
int s = 0;

for (int i = 0; i < n; i++)
{
for (int j = 0; j < m; j++)
{
array1[s] = processingtimes[i, j];
Operationsij[s, 0] = i;
Operationsij[s, 1] = j;
operaciones[s] = s + 1;
s++;
}
}
for (int j = 0; j < m; j++)
{
for (int i = 0; i < n; i++)
```

```
{
freeoperations[j, i] = aux;
aux = aux + m;
}
fila = fila + 1;
aux = fila;
}
for (int i = 0; i < m; i++)
{
completiontimesmaxmachine[i] = 0;
completiontimesmaxmachine_aux[i] = 0;
}

while (contador < (n * m))
{
if (contador > 0)
{
evaluarsecuenciaultimacomp(secuenciafinal, completiontimesjob, trabajospormaquina, avalaibletir
}

int maquina = choosemin(completiontimesmaxmachine, freeoperations, completiontimesmaxmachine.Le
int TCTref = 0;
for (int i = 0; i < n; i++)
{
TCTref = TCTref + completiontimesjob[i];
}
FOprevia = 0;

for (int k = 0; k < n; k++)
{
bool flag = false;

if (freeoperations[maquina, k] != 0)
{
flag = true;
componente = freeoperations[maquina, k];

M = completiontimesmaxmachine[maquina];
J = completiontimesjob[k];

prevalueCTJ = completiontimesjob[k];
prevalueCT = completiontimes[k, maquina];
prevalueST = startingtimes[k, maquina];
prevalueAT = avalaibletimes[maquina, trabajospormaquina[maquina]];
prevalueTxM = trabajospormaquina[maquina];
prevalueCTM = completiontimesmaxmachine[maquina];

int job = Operationsij[componente - 1, 0];
maquina = Operationsij[componente - 1, 1];
int PO = array1[componente - 1];

int completiontimemaxjob = completiontimesjob[job];
startingtimes[job, maquina] = Math.Max(avalaibletimes[maquina, trabajospormaquina[maquina]], co
completiontimes[job, maquina] = startingtimes[job, maquina] + PO;
completiontimesmaxmachine[maquina] = completiontimes[job, maquina];
trabajospormaquina[maquina]++;
avalaibletimes[maquina, trabajospormaquina[maquina]] = startingtimes[job, maquina] + PO;
```

```
completiontimesjob[job] = completiontimes[job, maquina];

int TCT = TCTref - prevalueCTJ + completiontimesjob[k];

int valor = LB(operaciones, componente, array1, array1.Length);

fo1 = valor + TCT;

if (J - M > 0)
idletime = J - M;
else
idletime = 0;

fo2 = idletime;

FO = (1 - alpha) * fo1 + alpha * fo2;

if (FOprevia == 0)
{
FOprevia = FO;
mejoroperacion = componente;
}

if (FO < FOprevia)
{
FOprevia = FO;
mejoroperacion = componente;
}
}

if (flag == true)
{
completiontimesjob[k] = prevalueCTJ;
completiontimes[k, maquina] = prevalueCT;
startingtimes[k, maquina] = prevalueST;
avalaibletimes[maquina, trabajospormaquina[maquina]] = prevalueAT;
trabajospormaquina[maquina] = prevalueTxM;
completiontimesmaxmachine[maquina] = prevalueCTM;
}
}

secuenciafinal[contador] = mejoroperacion;
borrarcomponente(mejoroperacion, maquina, freeoperations, operaciones, n, m);
contador = contador + 1;
}

evaluarsecuenciaultimacomp(secuenciafinal, completiontimesjob, trabajospormaquina, avalaibletimes, st
int TCTdef = 0;
for (int i = 0; i < n; i++)
{
TCTdef = TCTdef + completiontimesjob[i];
}
return TCTdef;
}

static int BICH_MIH_Makespan(int n, int m, int[,] processingtimes, double alpha)
{
```

```
int J, M;
int fo1, fo2, idletime;
int componente;
double FO, FOprevia;
int[] array1 = new int[n * m];
int[] secuenciafinal = new int[n * m];
int[] mejorsecuencia = new int[n * m];
int[] operaciones = new int[n * m];
int[,] freeoperations = new int[m, n];
int[] completiontimesmaxmachine = new int[m];
int[] completiontimesmaxmachine_aux = new int[m];
int[,] Operationsij = new int[n * m, 2];
int[,] avalaibletimes = new int[m, n + 1];
int[] trabajospormaquina = new int[m];
int[] completiontimesjob = new int[n];
int[,] startingtimes = new int[n, m];
int[,] completiontimes = new int[n, m];
int prevalueCTJ, prevalueCT, prevalueST, prevalueAT, prevalueCTM, prevalueTxM;
int contador = 0;
int mejoroperacion = 0;
int aux = 1;
int fila = 1;
componente = 0;
prevalueCTJ = prevalueCT = prevalueST = prevalueAT = prevalueCTM = prevalueTxM = 0;

for (int i = 0; i < n; i++)
{
completiontimesjob[i] = 0;
for (int j = 0; j < m; j++)
{
completiontimes[i, j] = 0;
startingtimes[i, j] = 0;
}
}
for (int j = 0; j < m; j++)
{
completiontimesmaxmachine[j] = 0;
trabajospormaquina[j] = 0;
avalaibletimes[j, n] = 0;
for (int i = 0; i < n; i++)
{
avalaibletimes[j, i] = 0;
}
}

Random aleatorio = new Random();
int s = 0;

for (int i = 0; i < n; i++)
{
for (int j = 0; j < m; j++)
{
array1[s] = processingtimes[i, j];
Operationsij[s, 0] = i;
Operationsij[s, 1] = j;
operaciones[s] = s + 1;
s++;
```

```
}
}
for (int j = 0; j < m; j++)
{
for (int i = 0; i < n; i++)
{
freeoperations[j, i] = aux;
aux = aux + m;
}
fila = fila + 1;
aux = fila;
}
for (int i = 0; i < m; i++)
{
completiontimesmaxmachine[i] = 0;
completiontimesmaxmachine_aux[i] = 0;
}

while (contador < (n * m))
{
if (contador > 0)
{
evaluarsecuenciaultimacomp(secuenciafinal, completiontimesjob, trabajospormaquina, avalaibletimes, st
}

int maquina = choosemin(completiontimesmaxmachine, freeoperations, completiontimesmaxmachine.Length,
FOprevia = 0;

for (int k = 0; k < n; k++)
{
bool flag = false;

if (freeoperations[maquina, k] != 0)
{
flag = true;
componente = freeoperations[maquina, k];

M = completiontimesmaxmachine[maquina];
J = completiontimesjob[k];

prevalueCTJ = completiontimesjob[k];
prevalueCT = completiontimes[k, maquina];
prevalueST = startingtimes[k, maquina];
prevalueAT = avalaibletimes[maquina, trabajospormaquina[maquina]];
prevalueTxM = trabajospormaquina[maquina];
prevalueCTM = completiontimesmaxmachine[maquina];

int job = Operationsij[componente - 1, 0];
maquina = Operationsij[componente - 1, 1];
int PO = array1[componente - 1];

int completiontimemaxjob = completiontimesjob[job];
startingtimes[job, maquina] = Math.Max(avalaibletimes[maquina, trabajospormaquina[maquina]], completi
completiontimes[job, maquina] = startingtimes[job, maquina] + PO;
completiontimesmaxmachine[maquina] = completiontimes[job, maquina];
trabajospormaquina[maquina]++;
avalaibletimes[maquina, trabajospormaquina[maquina]] = startingtimes[job, maquina] + PO;
```

```
completiontimesjob[job] = completiontimes[job, maquina];

int Makespan = completiontimesjob.Max();

int valor = LB(operaciones, componente, array1, array1.Length);

fo1 = valor + Makespan;

if (J - M > 0)
idletime = J - M;
else
idletime = 0;

fo2 = idletime;

FO = (1 - alpha) * fo1 + alpha * fo2;

if (FOprevia == 0)
{
FOprevia = FO;
mejoroperacion = componente;
}

if (FO < FOprevia)
{
FOprevia = FO;
mejoroperacion = componente;
}
}

if (flag == true)
{
completiontimesjob[k] = prevalueCTJ;
completiontimes[k, maquina] = prevalueCT;
startingtimes[k, maquina] = prevalueST;
avalaibletimes[maquina, trabajospormaquina[maquina]] = prevalueAT;
trabajospormaquina[maquina] = prevalueTxM;
completiontimesmaxmachine[maquina] = prevalueCTM;
}
}

secuenciafinal[contador] = mejoroperacion;
borrarcomponente(mejoroperacion, maquina, freeoperations, operaciones, n, m);
contador = contador + 1;
}

evaluarsecuenciaultimacomp(secuenciafinal, completiontimesjob, trabajospormaquina, avalaibletir
int Makespandef = completiontimesjob.Max();
return Makespandef;
}

static int LB(int[] operaciones, int componente, int[] array1, int length)
{
int[] operacionesaux = new int[operaciones.Length];
int valor = 0;
copiavector(operacionesaux, operaciones, operaciones.Length);
```

```
for (int i = 0; i < length; i++)
{
if (componente == operacionesaux[i])
operacionesaux[i] = 0;
}
for (int i = 0; i < length; i++)
{
int k = operacionesaux[i];
if (k != 0)
{
valor = valor + array1[k - 1];
}
}
return valor;
}

static void borrarcomponente(int componente, int maquina, int[,] freeoperations, int[] operaciones, i
{
for (int i = 0; i < n; i++)
{
if (freeoperations[maquina, i] == componente)
freeoperations[maquina, i] = 0;
}
for (int j = 0; j < n * m; j++)
{
if (operaciones[j] == componente)
operaciones[j] = 0;
}
}

static int choosemin(int[] vector, int[,] freeoperations, int length, int n)
{

int[] vectoraux = new int[vector.Length];
int s = 0;

for (int i = 0; i < length; i++)
{
int flag = 1;
for (int j = 0; j < n; j++)
{
if (freeoperations[i, j] != 0)
flag = 0;
}

if (flag == 1)
{
vectoraux[s] = i;
s++;
}
}

int maquina = 0;
int r = 0;

for (int i = 0; i < length; i++)
{
```

```
int flag2 = 0;

for (int j = 0; j < s; j++)
{
if (i == vectoraux[j])
{
flag2 = 1;
}
}

if (flag2 == 0)
{
if (r == 0)
{
maquina = i;
r++;
}

if (r > 0)
{
if (vector[i] < vector[maquina])
maquina = i;
}
}
}

return maquina;

}

static int[] evaluarsecuenciaultimacomp(int[] secuencia, int[] completiontimesjob, int[] traba
{

for (int k = lengtheval - 1; k < lengtheval; k++)
{
int completiontimemaxjob;
int job = Operationsij_aux[secuencia[k] - 1, 0];
int machine = Operationsij_aux[secuencia[k] - 1, 1];
int PO = array1[secuencia[k] - 1];

completiontimemaxjob = completiontimesjob[job];
startingtimes[job, machine] = Math.Max(avalaibletimes[machine, trabajospormaquina[machine]], co
completiontimes[job, machine] = startingtimes[job, machine] + PO;
completiontimesmaxmachine[machine] = completiontimes[job, machine];
trabajospormaquina[machine]++;
avalaibletimes[machine, trabajospormaquina[machine]] = startingtimes[job, machine] + PO;
completiontimesjob[job] = completiontimes[job, machine];
}

return completiontimesjob;
}

public static double[,] Quicksort(double[,] elements, int left, int right)
{
int i = left, j = right;
double pivot = elements[1, (left + right) / 2];
```

```
while (i <= j)
{
while (elements[1, i].CompareTo(pivot) > 0)
{
i++;
}

while (elements[1, j].CompareTo(pivot) < 0)
{
j--;
}

if (i <= j)
{

double tmp0 = elements[0, i];
double tmp1 = elements[1, i];
elements[0, i] = elements[0, j];
elements[1, i] = elements[1, j];
elements[0, j] = tmp0;
elements[1, j] = tmp1;

i++;
j--;
}
}

if (left < j)
{
Quicksort(elements, left, j);
}

if (i < right)
{
Quicksort(elements, i, right);
}
return elements;
}
}
}
```

Bibliografía

[1] Paz Pérez González, José Manuel Framiñán Torres, Víctor Fernández-Viagas Escudero, *Programación de operaciones: 4º GITI*, Departamento de Organización Industrial y Gestión de Empresas, Universidad de Sevilla, 2019.

[2] B. Naderia, S.M.T. Fatemi Ghomia, M. Aminnayeria, M. Zandiehb, *A contribution and new heuristics for open shop scheduling*. Computers & Operations Research, vol. 37, pp 213-221, Amirkabir University of Technology, Tehran, Iran, 2010

[3] Gonzalez Teofilo, Sartaj Sahni, *Open shop scheduling to minimize finish time*, Journal of the ACM, University of Minnesota, Minneapolis, Minnesota, 1976.

[4] Levi R. Abreu, Jesus O. Cunha, Bruno A. Prata, Jose M. Framinan, *A genetic algorithm for scheduling open shops with sequence-dependent setup times*, Department of Industrial Engineering, Federal University of Ceará, Campus do Pici, Brazil ,2019.

[5] Descarga software Visual Studio *https://visualstudio.microsoft.com/es/*.

[6] Guía de programación en C# *https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/*

[7] Curso de programación en C# *https://www.youtube.com/watch?v=6EBNIgkrU74&ab_channel=pildorasinformaticas*.

[8] Sergio D Martin, Sandra Leal, Jose M. Molina-Pariente, Victor Fernandez-Viagas, *AVPS: aplicación adaptativa para visualización de la programación de la producción en docencia e investigación*. Congreso Internacional Nodos del Conocimiento: Universidad, innovación e investigación ante el horizonte 2030, 2020.