Trabajo Fin de Grado
Ingeniería de Organización Industrial

Análisis del problema Distributed Flow shop con permutación y objetivo Total Core Idle Time

Autor: Marta Ariza Gamero
Tutor: Paz Pérez González

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

Sevilla, 2021

Trabajo Fin de Grado
Ingeniería de Organización Industrial

Análisis del problema Distributed Flow shop con permutación y objetivo Total Core Idle Time

Autor:
Marta Ariza Gamero

Tutor:
Paz Pérez González
Profesor titular

Dpto. de Organización Industrial y
Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Análisis del problema Distributed Flow shop con permutación y objetivo Total Core Idle Time

Autor: Marta Ariza Gamero

Tutor: Paz Pérez González

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A quién siempre estuvo junto a mí.

Índice

1 OBJETIVO DEL ESTUDIO

El objetivo de este estudio es la búsqueda de un método capaz de dar la mejor solución al problema de Programación de la Producción propuesto. El problema se define por ser un entorno *Distributed Flow shop* con restricción de permutación y con el objetivo de minimizar el *Core Idle* total.

El entorno *Distributed Flow shop* se caracteriza por ser un conjunto de fábricas idénticas siendo cada una de ellas un entorno tipo taller con máquinas en serie, es decir, un entorno Flow shop. El interés de un entorno distribuido se debe a que, en los últimos años, se están descubriendo los grandes beneficios que proporciona la deslocalización en fábricas manufactureras. El problema propuesto se caracteriza, además, por exigir en cada una de las fábricas la restricción de permutación, la secuencia de trabajos en cada una de las máquinas es la misma.

Hasta el momento este entorno no ha sido estudiado con objetivo *Total Core Idle Time*. Minimizar los tiempos ociosos entre trabajos dentro de una misma máquina es un criterio de programación para aquellos entornos donde se busque la mayor eficiencia posible en sistemas productivos, ya que es un indicador directamente relacionado con el flujo de trabajos en el sistema y la utilización de las máquinas.

La estructura del documento es la siguiente:

- En el capítulo 2, Introducción a la Programación de la Producción, se detalla qué es la Programación de la Producción, los entornos que se encuentran, las posibles limitaciones o restricciones del problema y los objetivos más comunes estudiados.
- El problema propuesto se explica en detalle en el siguiente capítulo, Descripción del problema.
- Para encontrar el mejor método se han revisado los modelos existentes en la literatura para problemas similares al propuesto y se han reflejado en Revisión de la literatura, ya que el problema no ha sido llevado al estudio hasta el momento.
- Estos modelos han sido adaptados en Método de resolución y, además, se ha propuesto una nueva heurística en este mismo capítulo.
- Estos métodos, un total de 26 heurísticas constructivas, se han programado en lenguaje de programación C y se han experimentado 72 instancias con el fin de comparar los resultados obtenidos en cada una de las heurísticas.
- Así se concluye en Resultados obtenidos cuál es el mejor método de los estudiados para el problema propuesto.
- En el último capítulo, Conclusiones, se hace una conclusión de todo el proyecto.

2 INTRODUCCIÓN A LA PROGRAMACIÓN DE LA PRODUCCIÓN

La Programación de la Producción es el procedimiento de toma de decisiones que asigna recursos a tareas para el tratamiento de ellas. Este proceso de toma de decisiones juega un rol muy importante tanto en fábricas manufactureras como en aquellas industrias dedicadas al transporte, distribución u otro tipo de servicio (Pinedo, 2012). En primer lugar, una empresa debe llevar a cabo el plan agregado de producción. Aquí se estudia la capacidad, los materiales empleados, el abastecimiento de los proveedores, pero no se detalla el orden de la producción de la planta. Una vez se entiende con claridad y en detalle cuál es la estructura de la planta y cuál es la demanda de trabajos a abastecer se pasa a estudiar la Programación de la Producción. En la Programación de la Producción se toman decisiones con el objetivo de que el producto o servicio resultante sea de la mayor calidad posible, al mínimo coste y con el mínimo plazo de entrega. A pesar de que la programación es una decisión compleja, tiene un ciclo de vida corto. El tiempo medio de vida es tan corto que algunos autores hablan de ella como un proceso de programación continuo. No obstante, a pesar de ser una decisión operativa es una decisión relativamente estructurada y con bastante relevancia para la empresa (Framinan et al., 2014).

En programación es necesario conocer algunos datos como los recursos que se disponen, el número de tareas a procesar, limitaciones de la planta y cuál es el objetivo que se desea optimizar. Siguiendo la notación extraída de (Framinan et al., 2014):

- *N={1,..,n}* denota al conjunto de trabajos o tareas a procesar. Los subíndices *j*, *k* hacen referencia a los trabajos.
- *M={1,…,m}* denota al conjunto de máquinas o recursos donde se procesan los trabajos. El subíndice *i* hace referencia a las máquinas.
- El tiempo de proceso, p_{ij}, es el tiempo que la máquina *i* tarda en procesar el trabajo *j*. También se puede ver como el tiempo que la máquina *i* está ocupada por el trabajo *j*.

Los modelos de programación de la producción comúnmente se definen por la notación propuesta por (Graham et al., 1979) mediante la tripleta $\alpha \mid \beta \mid \gamma$

- Entorno α : indica el tipo de disposición de la planta y el número de máquinas en ella. Los entornos más comunes son una máquina, máquinas paralelas y máquinas en serie, también conocidos como talleres.

  Cuando se tiene una única máquina (α = 1), cada trabajo debe pasar por la máquina con una única operación. El caso se extiende con *m* máquinas paralelas, donde cada trabajo es procesado por tan solo una de las máquinas con la ventaja de que se agiliza la producción en el caso de que incremente. Las máquinas en paralelo se diferencian por los tiempos de procesos:

  - Si son máquinas idénticas donde el tiempo de proceso es independiente de la máquina, $p_j = p_{ij}$ la notación del entorno es $\alpha = P_m$.
  - Si existiera una proporcionalidad entre las máquinas mediante velocidades, es decir, $p_{ij} = p_j/v_i$, la notación del entorno es $\alpha = Q_m$.
  - Si el tiempo de cada trabajo depende además de cada máquina, p_{ij}, el caso más genérico, la notación es $\alpha = R_m$.

No obstante, las máquinas se pueden ordenar en serie cuando cada una de ellas tiene una operación diferente que solo ella puede realizar. En estos casos, los trabajos tienen una ruta para pasar por cada una de las máquinas. Los entornos tipo taller se diferencian por las rutas:

- Si todos los trabajos tienen la misma ruta R_j = (*1, 2, …, m*), es un taller de flujo regular o *Flow shop* ($\alpha = F_m$).
- Si la ruta es un dato del problema, debemos conocer la ruta por trabajo, entonces es un taller de trabajo o *Job shop* ($\alpha = J_m$).
- Si no existe ruta, estamos en un taller abierto u *Open shop* ($\alpha = O_m$). Este es el caso más complejo de todos.

- Restricciones β : indica las limitaciones en el procesamiento de trabajos. En este campo puede entrar más de una restricción o incluso no existir ninguna. Aun no existiendo restricciones, se dan unas suposiciones generales: las máquinas siempre están disponibles, todos los trabajos llegan en el instante 0, los trabajos cuando empiezan a ser procesados no se interrumpen y la capacidad de los almacenes o *buffers* entre máquinas se supone infinito. No obstante, si aparecen restricciones de interrupciones fechas de llegada, las suposiciones correspondientes anteriormente nombradas se omiten.

  En los entornos tipo de taller se puede exigir que no haya tiempos ociosos en las máquinas entre trabajos (β = *no-idle*) o que los trabajos no puedan esperar entre máquinas (β = *no-wait*), por mencionar algunas de las muchas que existen. Además, en el entorno de taller de flujo habitualmente se restringe que sea regular de permutación (β = *prmu*), es decir, que la secuencia de los trabajos en cada una de las máquinas sea la misma.

- Objetivo γ : indica la función a optimizar. En programación de la producción se categorizan según su coste, tiempo, calidad y flexibilidad, tal y como se muestra en *Ilustración 1* (Framinan et al., 2014).

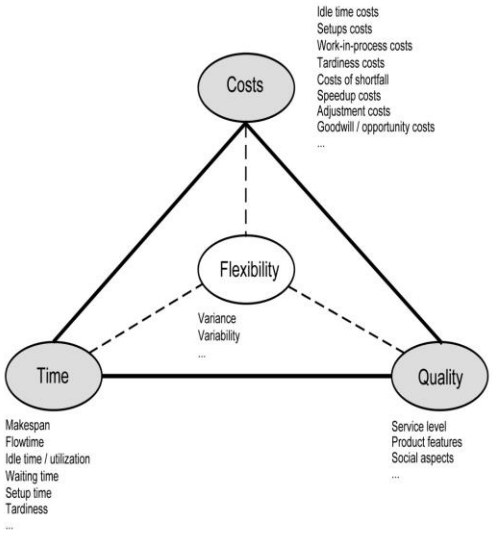

Ilustración 1.Clasificación de los objetivos

Algunos de estos objetivos son: el tiempo de terminación o *makespan* ($\gamma = C_{max}$), momento en el que finaliza el programa o la suma de los tiempos en los que termina cada trabajo ($\gamma = \sum C_j$); el máximo tiempo de flujo o *flowtime* ($\gamma = máx\ F_j$), máximo tiempo en el que un trabajo está en el sistema o la suma de estos *flowtime* ($\gamma = \sum F_j$). Aunque también existen otros relacionados con las fechas de entrega, que miden tanto el valor máximo como la suma para todos los trabajos. Para ello se mide el retraso de cada trabajo o *lateness* (L_j), si termina antes o *tardiness* (T_j), o después *earliness* (E_j) de la fecha de entrega.

Un programa o *schedule* es el resultado del procedimiento de toma de decisiones de la programación. Un programa es admisible (*feasible schedule*) si cumple con las restricciones y características del proceso productivo. Además, un programa es semiactivo (*semi-active schedule*) si además de ser admisible, las tareas no se pueden adelantar más sin cambiar el orden en el que se procesan en las máquinas. En todas las heurísticas constructivas se han supuestos programas semiactivos.

3 DESCRIPCIÓN DEL PROBLEMA

El problema que se va a tratar, según la tripleta $\alpha \mid \beta \mid \gamma$, es $DF_m \mid prmu \mid \sum CIT$ que se va a explicar a continuación.

3.1 Entorno Distributed Flow shop

El entorno *Distributed Flow shop* DF_m es un caso particular del *Flow shop* F_m . Recordando la Introducción, la distribución de una planta *Flow shop* es una disposición tipo taller con máquinas en serie y donde todos los trabajos tienen que pasar por todas las máquinas en el mismo orden. Se asume que primero pasa por la máquina 1, a continuación, la máquina 2 y así hasta la máquina *m.*

En el *Distributed Flow shop* DF_m hay un conjunto $F = \{1,\ldots,f\}$ de fábricas idénticas con *m* máquinas en cada una de las fábrica. Cada una de las máquinas realiza una operación diferente en la fábrica. Cada uno de los *n* trabajos deben ser procesados por una única fábrica *f*. Cualquier trabajo puede asignarse a cualquier fábrica que actúa como un entorno *Flow shop.*

3.2 Restricciones del problema

Es necesario asumir las suposiciones generales explicadas en la Introducción, para la definición del problema. Una operación no puede ser interrumpida. Se presupone también que todos los *n* trabajos están disponibles para ser procesados en el instante 0, así como que todas las *m* máquinas estas disponibles para procesar en el mismo instante 0. Además, un trabajo *n* no puede ser procesado por más de una máquina simultáneamente ni una máquina puede procesar dos trabajos al mismo tiempo. El tiempo de transporte entre máquinas se considera despreciable al igual que los buffers se consideran infinitos.

Además, de seguir el mismo orden o ruta los trabajos que hay en cada fábrica con distribución *Flow shop*, se exige por la restricción *prmu* que la secuencia es la misma para todas las máquinas, es decir, el orden por el que los trabajos pasan por las máquinas para ser procesado es el mismo en cualquiera de las máquinas de una misma fábrica *f*. El tiempo de proceso del trabajo *j* en la máquina *i,* p_{ij}, es el mismo en cualquiera de las fábricas del conjunto *F*.

3.3 Objetivo Total Core Idle Time

El tiempo ocioso de una máquina se puede dividir en tres componenentes (Maassen et al., 2020):

- *Front Idle Time* de la máquina *i,* $\mathrm{FIT_i}$: tiempo que la máquina espera a procesar el primer trabajo programado.
- *Core Idle Time* de la máquina *i,* $\mathrm{CIT_i}$: tiempo que una máquina espera entre dos trabajos consecutivos.
- *Back Idle Time* de la máquina *i,* $\mathrm{BIT_i}$: tiempo que la máquina espera a que todos los trabajos de procesados.

La restricción *no-idle* en un modelo de programación de la producción limita los programas en los que no existen tiempos ociosos de las máquinas entre trabajos. Es posible relajar la restricción si, en lugar de impedir tiempos ociosos, se minimizan para aquellos problemas donde la restricción no sea limitante como sí que sucede en algunos sectores como las industrias alimentarias.

En *Ilustración 2* (Maassen et al., 2020), se muestra mediante un diagrama de Gantt el tiempo ocioso que aparece en una entorno *Flow shop* con valores *j* = 4 trabajos y *m* = 3 máquinas. Si la restricción *no-idle* se exigiera, aparecería FIT_i y BIT_i pero no CIT_i.

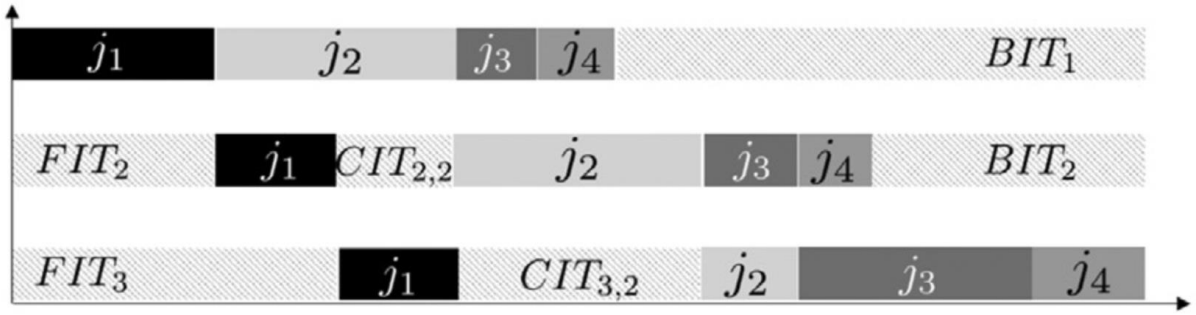

Ilustración 2. Tiempos ociosos en un entorno Flow shop

En la literatura, la restricción *no-idle* se ha estudiado junto al objetivo *makespan* (Ruiz et al., 2009). Para relajar dicho modelo, se sustituye el objetivo *makespan* denominado como C_{max}, por un nuevo objetivo que minimiza el tiempo ocioso entre trabajos dentro de una misma máquina denotado como ΣCIT desapareciendo la restricción *no-idle*. Además, se ha demostrado que los métodos eficientes empleados para la función objetivo clásica es aplicable a la nueva función objetivo, sabiendo que generan soluciones de calidad pero no las óptimas (Maassen et al., 2020).

En este proyecto se plantea considerar como función objetivo la suma o *Total Core Idle Time* generados en cada fábrica, ΣCIT. Se debe a que es tiempo desperdiciado en la programación, no se puede emplear ese tiempo para realizar tareas de mantenimiento, transporte o limpieza, entre otras. Sin embargo, el *Front Idle Time* y el *Back Idle Time* son intervalos de tiempo aprovechables para realizar las tareas anteriormente mencionadas.

Asimismo, la eliminación de cualquier tipo de desperdicio en la producción es uno de los principales objetivos en cualquier industria, a fin de obtener un sistema eficiente. ΣCIT proporciona información sobre la eficiencia de la producción y la utilización de las máquinas.

3.4 Cálculo del objetivo para el problema propuesto

En este apartado, mediante un ejemplo numérico sencillo, se explica cómo se calcula el *Total Core Idle Time* para un problema con *f* = 2 fábricas, *m* = 4 máquinas y *j* = 7 trabajos. Los tiempos de proceso se encuentran en *Tabla 1*.

		trabajo j						
		1	2	3	4	5	6	7
máquina i	1	3	4	2	1	2	3	3
	2	2	1	1	2	4	3	3
	3	1	2	3	4	4	5	1
	4	5	2	2	2	2	1	2

Tabla 1.Tiempos de proceso del ejemplo dado

El programa se muestra en *Ilustración 3*. La fábrica 1 tiene asignada la secuencia (*j1, j2, j3, j4),* mientras que la fábrica 2, la secuencia (*j5, j6, j7*). Comenzando por la fábrica 1, cuando el trabajo *j1* termina de ser procesado en la máquina 1, pasa inmediatamente a la máquina 2 ya que se está trabajando con programas semiactivos. Se puede observar que el primer trabajo secuenciado en una fábrica con entorno *Flow shop* con permutación y con programa semiactivo no va a generar *Core Idle Time.* Además, en la primera máquina de cada fábrica tampoco se puede generar CIT cuando se trabaja con programas semiactivos.

En el momento que el trabajo *j1* termina en la máquina 1 ($p_{1,1}$= 3 u.t.) pasa a la máquina 2 ($p_{2,1}$= 2 u.t.), y en dicho instante, el trabajo *j2* comienza a ser procesado en la máquina 1 ($p_{1,2}$= 4 u.t.). No obstante, cuando el trabajo *j1* termina en la máquina 2, no pasa el trabajo *j2* a la máquina 1 puesto que su tiempo de proceso aún no ha finalizado. La máquina 2 pasa 2 u.t. sin procesar trabajos, el CIT de la máquina 2 hasta el momento es de 2 u.t.

En la máquina 1, cuando el trabajo *j2* termina, el trabajo *j3* comienza a ser procesado en esta máquina ($p_{1,3}$ = 2 u.t.) pero una vez más, el trabajo *j2* termina en la máquina 2 ($p_{2,2}$ = 1 u.t.) antes de que comience el trabajo *j3*. El CIT de la máquina 2 es hasta el momento de 3 u.t.

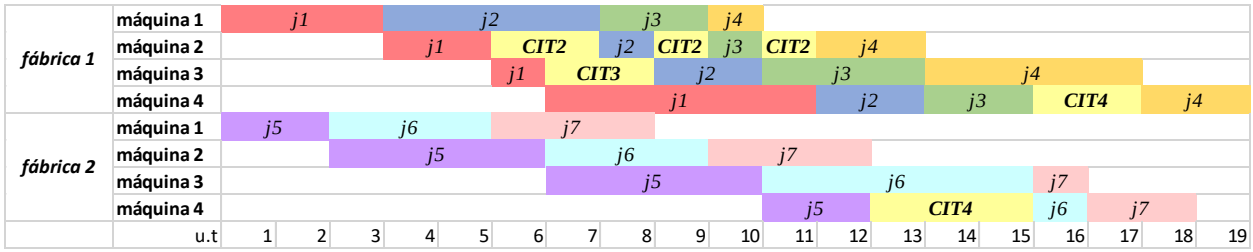

Ilustración 3. Diagrama de Gantt del programa dado

De esta manera, el ΣCIT se calcula como la suma de CIT de cada fábrica y el CIT de cada fábrica se calcula como la suma de CIT de cada máquina. Se representa en *Tabla 2* el cálculo de la función objetivo. En la fábrica 1, el CIT de la máquina 2 son 4 u.t., CIT de la máquina 3 son 2 u.t. y el CIT de la máquina 4 es de 2 u.t. La fábrica 1, en el programa propuesto, desaprovecha 8 u.t. en las que no se están procesando trabajos ni se están realizando tareas de limpieza o mantenimiento. Así, el CIT de la fábrica 2 es de 3 u.t. en la máquina 4. En conclusión, el problema propuesto y siguiendo el programado dado, el *Total Core Idle* es 11 u.t.

	máquina 1	máquina 2	máquina 3	máquina 4	CIT por fábrica
fábrica 1	0	4	2	2	8
fábrica 2	0	0	0	3	3
				CIT total	11

Tabla 2. Cálculo de la función objetivo para el programa dado

4 REVISIÓN DE LA LITERATURA

El problema detallado anteriormente, Descripción del problema, no ha sido estudiado con anterioridad. Es un nuevo foco de estudio a raíz de ciertos problemas relacionados que se encuentran en la literatura. En este capítulo se hace una recopilación de algunos de ellos.

El problema de programación constituido por el entorno *Flow shop*, y más concretamente el entorno *Permutation Flowshop*, es un problema de optimización antiguo y bastante estudiado en la literatura (Nawaz et al., 1983). A este problema se hará referencia como *Permutation Flow shop Scheduling Problem* (PFSP). En él, se debe asignar el orden de los trabajos en una única fábrica con una ruta determinada. Han sido muchos los objetivos estudiados siendo el makespan el más popular de ellos (Fernandez-Viagas & Framinan, 2014).

La deslocalización y la globalización han generado la necesidad de estudiar nuevos entornos, siendo el problema *Distributed Permutation Flow shop* (DPFSP) la evolución de PFSP. A diferencia del PFSP, no solo se asignan trabajos a máquinas con una misma ruta; sino que también esos trabajos deben ser asignados a distintas fábricas que son idénticas.

El DPFSP se ha estado asociando a objetivos de consumo y eficiencia energética como consecuencia de que el sector industrial es uno de los principales consumidores de energía a nivel mundial, y como tal pretenden reducir su gasto. No obstante, (Pan et al., 2019a) y (Fernandez-Viagas et al., 2018) plantean un problema mono objetivo al minimizar el *flowtime* con el interés de estabilizar los recursos al disminuir el tiempo total que los trabajos permanecen en el sistema.

En la literatura se encuentra el denotado problema *Energy-Efficient Distributed No-idle Flow shop Scheduling problem* (EEDNIFSP) (Chen et al., 2019), que además de exigir la misma ruta, no permite tiempos ociosos entre máquinas (*no-idle*). Mediante un algoritmo colaborativo de optimización (COA) con múltiples métodos de inicialización y operadores de búsqueda tanto para la asignación de trabajos en las secuencias de las fábricas como para la determinación de las velocidades de los trabajos por cada máquina, se ha querido minimizar el *makespan* y el consumo energético total (TEC). Por tanto, la solución al problema viene determinada por la asignación trabajos-fábricas y secuencias dentro de las fábricas dada por el algoritmo NEH adaptado y la velocidad dada por el método *Closet Operation Time* (COT).

El estudio de este problema multiobjetivo en concreto se ha extendido y ha sido investigado por otros autores por el interés que supone la relación del *makespan* y el TEC: a medida que se desea minimizar el makespan, se maximiza el TEC porque las velocidades de las máquinas aumentan. No obstante, si se aumenta la velocidad en las operaciones no críticas, es decir, las que no determinan el *makespan*, se reduce el TEC sin afectar al tiempo que la fábrica está procesando trabajos.

En (G. Wang et al., 2020) mediante un nuevo algoritmo multiobjetivo Whale Swarm (MOWSA), introducen nuevos operadores relacionados con la eficiencia energética y nuevas consideraciones en el TEC para buscar la mejor solución al DPFSP. En último lugar, (J. J. Wang & Wang, 2020) gracias a un algoritmo *Knowledge-based Cooperative* (KCA) y comparando sus resultados con los precedentes concluye que se llega a mejores soluciones con KCA.

5 MÉTODO DE RESOLUCIÓN

En este capítulo se van a presentar un total de veintiséis heurísticas aplicadas al problema que han sido programadas para conocer los mejores resultados al problema propuesto. Son heurísticas adaptadas ya que, tal y como se mencionó en Revisión de la literatura, el problema propuesto no se ha estudiado con anterioridad y no se encuentra en la literatura.

5.1 Adaptación de la heurística DLR

La heurística DLR fue propuesta por (Pan et al., 2019b) para el problema *Distributed Permutation Flow shop* con objetivo *flowtime*, $DF_m \mid prmu \mid \sum F_j$. Al tratarse de una heurística constructiva, la secuencia se va creando al insertar los trabajos de uno en uno en ella. El procedimiento se observa en *Ilustración 4*. La heurística, en primer lugar, decide cuál es la mejor fábrica para procesar el siguiente trabajo. Para ello, calcula el *makespan* de cada fábrica y así conoce cuál es la fábrica que antes va a poder procesar el siguiente trabajo, k^*. A continuación, decide cuál es el siguiente trabajo *j* que se va a insertar en dicha fábrica. Este trabajo se inserta en la última posición de la secuencia parcial de la mejor fábrica , n_{k*}.

El mejor trabajo es aquel que genere menor valor del índice $IF_{j,n_{k*}}$. Este valor es una función del tiempo de terminación o *makespan* de la fábrica tras insertar el trabajo que estamos evaluando, $C_{m,j}$ y los tiempos ociosos que genera también tras su inserción, $IT_{j,n_{k*}}$. El cálculo del índice se muestra en *Ecuación 1*, así como el cálculo de los tiempos ociosos *Ecuación 2*. Recordando la notación de Introducción a la Programación de la Producción, *f* hace referencia a la fábrica que se está evaluando, *m* a la máquina y *j* al trabajo.

$$IF_{j,n_{k*}} = ({}^{n}/_{f} - n_{k*} - 2)IT_{j,n_{k*}} + C_{m,j}$$

Ecuación 1. Índice IF (j, nk)*

$$IT_{j,n_{k*}} = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,j} - C_{i,[n_{k*}]}, 0\}}{i + n_{k*} \cdot (m - 1)/({}^{n}/_{f} - 2)}$$

*Ecuación 2. Cálculo Idle Time para nk**

El interés de esta heurística para nuestro problema proviene precisamente de este índice $IF_{j,n_{k*}}$. Al tener en cuenta los tiempos ociosos que se generan y buscar el menor valor de ellos, el valor de la función objetivo también se verá decrementado.

En las primeras *f* iteraciones siempre va a haber, al menos, una fábrica cuyo tiempo de terminación es 0 por no tener asignado ningún trabajo, se calcula el índice $IF_{j,0}$ según *Ecuación 3* con ayuda de *Ecuación 4* y se ordenan los trabajos de manera creciente y se asignan los *f* primeros a las *f* fábricas.

$$IF_{j,0} = ({}^{n}/_{f} - 2)IT_{j,0} + C_{m,j}$$

Ecuación 3. Índice IF(j,0)

$$IT_{j,0} = \sum_{i=2}^{m} \frac{m \cdot C_{i-1,j}}{i}$$

Ecuación 4. Cálculo Idle Time para 0

Procedure DLR

Compute $IF_{j,0}$ for each $j \in N$

$\lambda := (\lambda_1, \lambda_2, \ldots, \lambda_n)$ jobs sorted according to non-descending order to $IF_{j,0}$

$\pi := (\pi_1, \pi_2, \ldots, \pi_n)$ where $\pi_k = (\lambda_k)$, $k=1,2,\ldots,f$

Remove jobs $\lambda_1, \lambda_2, \ldots, \lambda_f$ from λ

while sizeof (λ) > 0 **do**

Find factory k^* that has the lowest makespan

Compute $IF_{j,n_{k*}}$ for each job j included in λ

j^* := job leading to smallest $IF_{j,n_{k*}}$

Append j^* to the end of the π_{k*}

Remove job j from λ

endwhile

return π

Ilustración 4. Pseudocódigo adaptación DLR

5.2 Adaptación de la heurística DNEH

Al igual que la heurística constructiva DLR (Adaptación de la heurística DLR), la DNEH fue propuesta por (Pan et al., 2019b) para el mismo problema. Dado el éxito de la adaptación del NEH clásico para el problema $F_m \mid prmu \mid C_{max}$ (Nawaz et al., 1983) para el problema $DF_m \mid prmu \mid C_{max}$, en ese caso se adaptó para γ = *flowtime*, $DF_m \mid prmu \mid \sum F_j$. Una vez más se emplea el índice $IF_{j,n_{k*}}$para ordenar los trabajos de mayor a menor en lugar de ordenarlo según suma de tiempos de procesos p_{ij}, como ocurre en el NEH clásico.

Los trabajos se van tomando para asignarlos a las fábricas de mayor a menor $IF_{j,n_{k*}}$. Se asignan a aquella fábrica y a aquella posición dentro de la fábrica que genere menor incremento de la función objetivo, es decir, que genere menor incremento de tiempo de flujo. Para el problema que se está estudiando, simplemente hay que buscar el menor incremento de ΣCIT para obtener la programación. El procedimiento se muestra en *Ilustración 5*.

Tal y como ocurría en DLR, los primeros *f* trabajos con mayores valores de $IF_{j,0}$ se asignan a las *f* fábricas.

Procedure DNEH

Compute $IF_{j,0}$ for each $j \in N$

$\lambda := (\lambda_1, \lambda_2, \ldots, \lambda_n)$ jobs sorted according to non-descending order to $IF_{j,0}$

$\pi := (\pi_1, \pi_2, \ldots, \pi_n)$ where $\pi_k = (\lambda_k)$, $k=1,2,\ldots,f$

Remove jobs $\lambda_1, \lambda_2, \ldots, \lambda_f$ from λ

while sizeof (λ) > 0 **do**

Extract the first job *j* from λ

for k:=1 **to** f **do**

Test job *j* at all the possible positions of π_k

Δ_k := mínimum increase of total core idle time

ξ_k := position resulting in Δ_k

endfor

k^* := argmin($\min_{k:=1,2,\ldots,f}(\Delta_k)$)

Insert job *j* at position ξ_k of Δ_k

endwhile

return π

Ilustración 5. Pseudocódigo adaptación DNEH

5.3 Adaptación de las heurísticas NEH (R1, Ai), NEH (R2, Ai)

Al igual que la DNEH (Adaptación de la heurística DNEH), en el artículo (Fernandez-Viagas et al., 2018) se toma la heurística NEH para buscar la mejor solución al problema $DF_m \mid prmu \mid \sum F_j$. Esta vez se tienen en cuenta dos formas de representar las soluciones que determinarán la eficiencia de las heurísticas y seis reglas para asignar los trabajos a las fábricas, dando lugar a doce heurísticas constructivas.

La denominada representación *R1* da una única secuencia de trabajos para el problema. Con ella no se puede conocer la asignación de trabajos a fábricas, es necesario conocer la regla aplicada para determinar la programación. Se supone un problema con *f* = 2 fábricas, *m* = 2 máquinas y *j* = 4 trabajos, si la secuencia fuera (*j3, j4, j1, j2)*, el programa es desconocido hasta que se conozca la regla y varía según el criterio de asignación tal y como se muestra en *Ilustración 6* e *Ilustración 7*. Se ha empleado para dicha secuencia, la representación *R1* y dos criterios de asignación distintos que serán explicados posteriormente, generando programas diferentes.

La denominada representación *R2* da la secuencia por fábricas y por tanto la solución es conocida sin conocer cuál ha sido la regla de asignación empleada. La programación es única siempre que se consideren programas semiactivos, en *Ilustración 8* se indica como es la representación *R2* sin necesitar secuencia alguna.

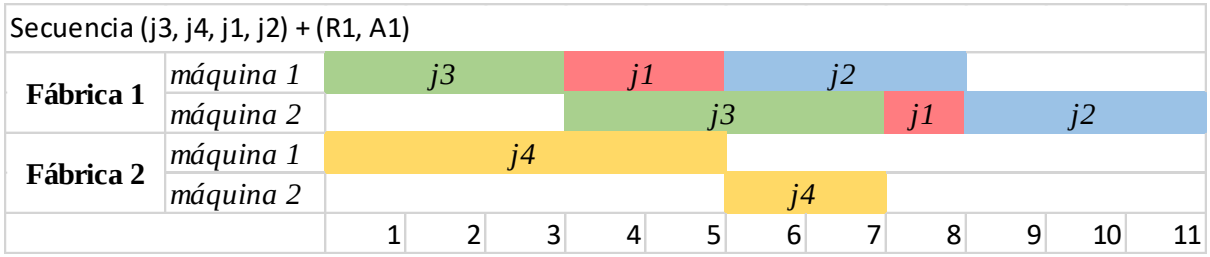

Ilustración 6. NEH con representación R1 y regla de asignación A1

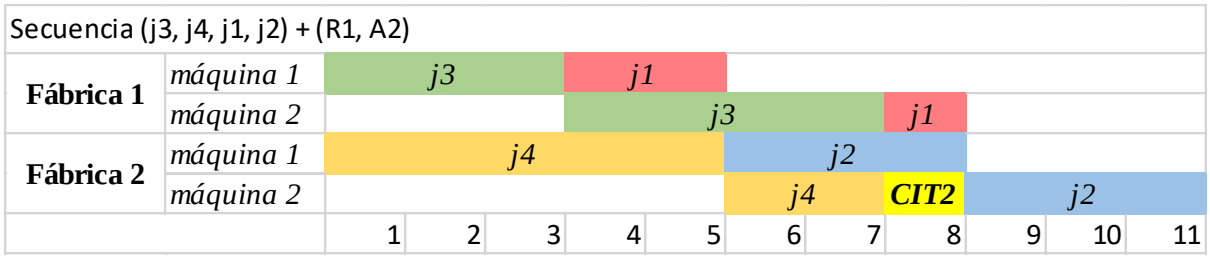

Ilustración 7. NEH con representación R1 y regla de asignación A2

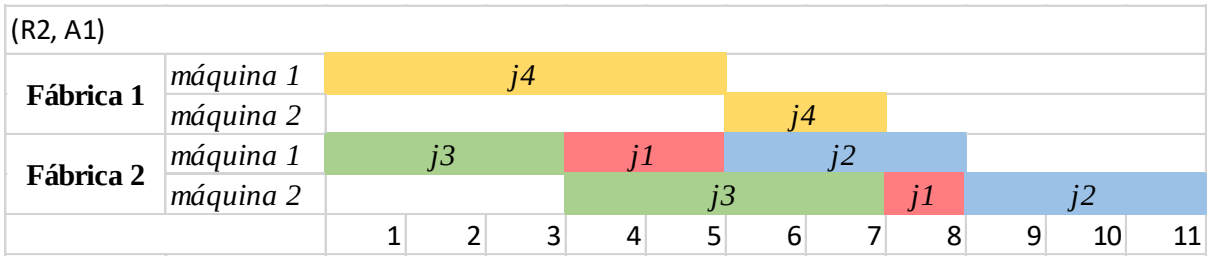

Ilustración 8. NEH con representación R2 y regla de asignación A1

Por otro lado, tenemos las seis reglas de asignación que varían según la representación. Para *R1*, el trabajo se inserta al final de la secuencia parcial de la mejor fábrica en función de la regla elegida. Para *R2*, la regla de asignación ayuda a insertar el trabajo, forma parte de la solución puesto que la programación con las secuencias ya está dada. Las reglas se exponen a continuación:

- *A1:* Asigna el trabajo a insertar al final de la fábrica que tiene menor tiempo de terminación.
- *A2:* Asigna el trabajo a insertar al final de la fábrica que tiene menor tiempo de flujo.
- *A3:* Asigna el trabajo en la mejor posición de la fábrica que genera menor tiempo de flujo en la fábrica tras su inserción.
- *A4:* Asigna el trabajo en la mejor posición de la fábrica que genera menor valor de la función objetivo tras su inserción.
- *A5*: Asigna el trabajo en la mejor posición de la fábrica que genera menor valor de la función objetivo tras su inserción sin evaluar la fábrica que tiene mayor tiempo de flujo.
- *A6*: Asigna el trabajo en la mejor posición de la fábrica que genera menor valor de la función objetivo tras su inserción. Esta regla evalúa el subconjunto *F'* formado por las (*f*/2) fábricas con menores tiempo de flujo total.

Se adaptó esta heurística al problema DF_m | prmu | $\sum CIT$ al adaptar las reglas de asignación. En aquellas reglas donde se busca disminuir el tiempo de flujo, se ha buscado disminuir el CIT. La representación de las soluciones se ha respetado. En *Ilustración 6* e *Ilustración 8*, el número de fábricas, máquinas, trabajos y tiempos de procesos

son los mismos, así como el criterio de asignación empleado. No obstante, la representación es distinta en cada caso generando programas distintos con valores de la función objetivo diferentes.

El procedimiento para la representación *R1* se muestra *Ilustración 9* y para la representación *R2* en *Ilustración 10*. El orden inicial, tal y como ocurre en la NEH, es por orden creciente de suma de procesos.

Procedure NEH (R_1, A_i***)***

$\Omega := (\Omega_1, \Omega_2, \ldots, \Omega_n)$ jobs sorted according to non-increasing order to sums of processing times $(\Omega := \{\omega_1, \omega_2, \ldots, \omega_n\})$

$\Pi := (\omega_1)$

for *k*:=2 **to** *n* **do**

$\Pi^{\omega_k{}^l}$: = Partial sequence formed by inserting job ω_k in position *l* of partial sequence Π where *l*=1, 2, ..., *k*

Compute each partial sequence $\Pi^{\omega_k{}^l}$ by using assigmenr rule A_i to allocate Jobs to factories at the end of that factory. Let *l**be the index *l* whose total completion time is minimal

$\Pi := \Pi^{\omega_k{}^l}$

endfor

return Π

Ilustración 9.Pseudocódigo adaptación NEH (R1, Ai)

Procedure NEH ($\boldsymbol{R_2}$***,*** $\boldsymbol{A_i}$***)***

$\Omega := (\Omega_1, \Omega_2, \ldots, \Omega_n)$ jobs sorted according to non-increasing order to sums of processing times $(\Omega := \{\omega_1, \omega_2, \ldots, \omega_n\})$

$\Pi := (\omega_1)$

$\Pi_f = \emptyset, \forall f \neq 1$

for *k*:=2 **to** *n* **do**

(*l*,*f*) := A_i

Π_f := Sequence formed by inserting job ω_k in position *l* of partial sequences Π_f

endfor

return Π

Ilustración 10. Pseudocódigo adaptación NEH (R2, Ai)

5.4 Nuevas heurísticas

Las heurísticas hasta el momento explicadas son adaptaciones de métodos existentes al problema propuesto. Una nueva heurística constructiva se plantea en este apartado a partir del clásico NEH (Nawaz et al., 1983) y la adaptación al problema $DF_m \mid prmu \mid \sum F_j$ explicado anteriormente (Adaptación de las heurísticas NEH (R1, Ai), NEH (R2, Ai)).

Al observar los buenos resultados que genera el método NEH (*R1*, *Ai*) y NEH (*R2*, *Ai*), se han empleados ambas representaciones y las seis reglas de asignación propuestas para buscar mejores soluciones al problema $DF_m \mid prmu \mid \sum CIT$. En esta nueva heurística los trabajos no se ordenan por suma de tiempo de procesos para ser asignados a las fábricas, sino que se empleada el índice $IF_{j,0}$ para ordenarlos de manera creciente. El interés de esta inteligencia procede a que ordena de forma que genera menores tiempos ociosos, por consiguiente, se obtienen menores valores de la función objetivo. Ambos métodos se encuentran codificados en *Ilustración 11* e *Ilustración 12* haciendo referencia a ellos como DNEH (*R1*, *Ai*) y DNEH (*R2*, *Ai*).

Procedure DNEH (R_1***,*** A_i***)***

Compute $IF_{j,0}$ for each $j \in N$

Generate job permutation $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ according to non-descending order to $IF_{j,0}$

$\Pi := (\lambda_1)$

for *k*:=2 **to** *n* **do**

$\Pi^{\omega_k{}^l} :=$ Partial sequence formed by inserting job λ_k in position *l* of partial sequence Π where *l*=1, 2, ..., *k*

Compute each partial sequence $\Pi^{\omega_k{}^l}$ by using assigmenr rule A_i to allocate Jobs to factories at the end of that factory. Let *l**be the index *l* whose total completion time is minimal

$\Pi := \Pi^{\omega_k{}^l}$

endfor

return Π

Ilustración 11. Pseudocódigo DNEH (R1, Ai)

Procedure DNEH (R_2***,*** A_i***)***

Compute $IF_{j,0}$ for each $j \in N$

Generate job permutation λ= $(\lambda_1, \lambda_2, \ldots, \lambda_n)$ according to non-descending order to $IF_{j,0}$

Π := (λ_1)

for *k*:=2 **to** *n* **do**

(l,f) := A_i

Π_f := Sequence formed by inserting job λ_k in position *l* of partial sequences Π_f

endfor

return Π

Ilustración 12. Pseudocódigo DNEH (R2, Ai)

6 RESULTADOS OBTENIDOS

Los algoritmos presentados en Método de resolución, se han evaluado mediante 72 instancias de (Naderi & Ruiz, 2010) (http://soa.iti.es/). Las instancias presentan problemas con valores de $f \in \{2, 3, 4, 5, 6, 7\}$, $m \in \{2, 3, 4, 5, 10, 20\}$ y $n \in \{4, 6, 8, 10, 12, 14, 16, 20, 50, 100\}$. La experimentación, objeto de este estudio, se ha realizado mediante la codificación de los algoritmos en lenguaje de programación C mediante CodeBlocks y haciendo uso de la librería *schedule* (*Librería SCHEDULE – Grupo de Investigación Organización Industrial*, n.d.). El código de la programación se encuentra en Anexo Código programación en C.

Para comparar los valores obtenidos en la experimentación, se define el índice $RDI^h{}_I$ en *Ecuación 5*, donde *I* es la instancia a evaluar y *h* la heurística con la que se ha evaluado *I*. *H* es el conjunto de los 26 algoritmos (Método de resolución). El $RDI^h{}_I$ muestra la variación de cada instancia evaluada respecto al mejor valor obtenido para dicha instancia. Valores próximos a 1 indican peores valores de la función objetivo para la instancia *I*, mientras que valores próximos a 0 indican que son valores similares a la mejor solución evaluada. Los resultados obtenidos de cada instancia por cada heurística están a disposición del tribunal bajo solicitud en Excel, así como los valores del índice *RDI*.

$$RDI^h{}_I = \frac{FO^h{}_I - \min_H FO_I}{\max_H FO_I - \min_H FO_I} \qquad H := DLR, DNEH, \ldots, DNEH(R_2, A_6)$$

Ecuación 5. Índice RDI

Calculando el promedio del índice $RDI^h{}_I$ para cada *h*, reflejado en *Tabla 3*, se concluye que la mejor heurística programada es la NEH (*R1, A4*) seguida de la nueva heurística propuesta como DNEH (*R1, A4*), puesto que la asignación *A4* destina los trabajos a la mejor posición de la fábrica que genere menor incremento en la función objetivo. Se puede observar en *Ilustración 13* que las heurísticas constructivas NEH con representación *R1* generan mejores resultados que las NEH con representación *R2*. Además, a pesar de esperar mejores resultados con la heurística propuesta, la adaptación de la heurística NEH (*R1, A4*) de (Fernandez-Viagas et al., 2018) sigue funcionando mejor para el problema (*Ilustración 14*).

Promedio RDI			
DNEH	0,057	NEH R2 A6	0,088
DLR	0,929	DNEH R1 A1	0,301
NEH R1 A1	0,183	DNEH R1 A2	0,209
NEH R1 A2	0,115	DNEH R1 A3	0,102
NEH R1 A3	0,136	DNEH R1 A4	0,009
NEH R1 A4	0,003	DNEH R1 A5	0,035
NEH R1 A5	0,023	DNEH R1 A6	0,044
NEH R1 A6	0,035	DNEH R2 A1	0,756
NEH R2 A1	0,335	DNEH R2 A2	0,619
NEH R2 A2	0,334	DNEH R2 A3	0,356
NEH R2 A3	0,396	DNEH R2 A4	0,053
NEH R2 A4	0,026	DNEH R2 A5	0,101
NEH R2 A5	0,084	DNEH R2 A6	0,120

Tabla 3. Promedio RDI por heurística

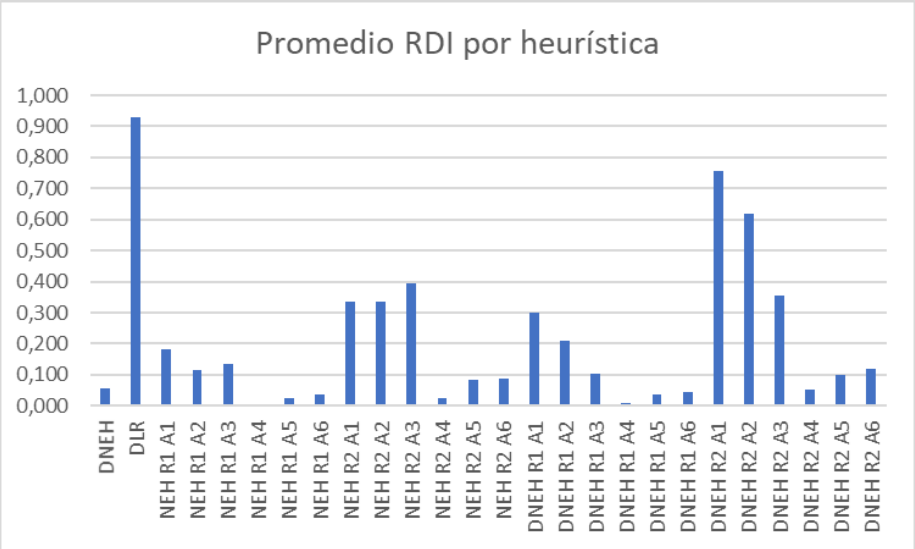

Ilustración 13. Gráfica promedio RDI por heurística

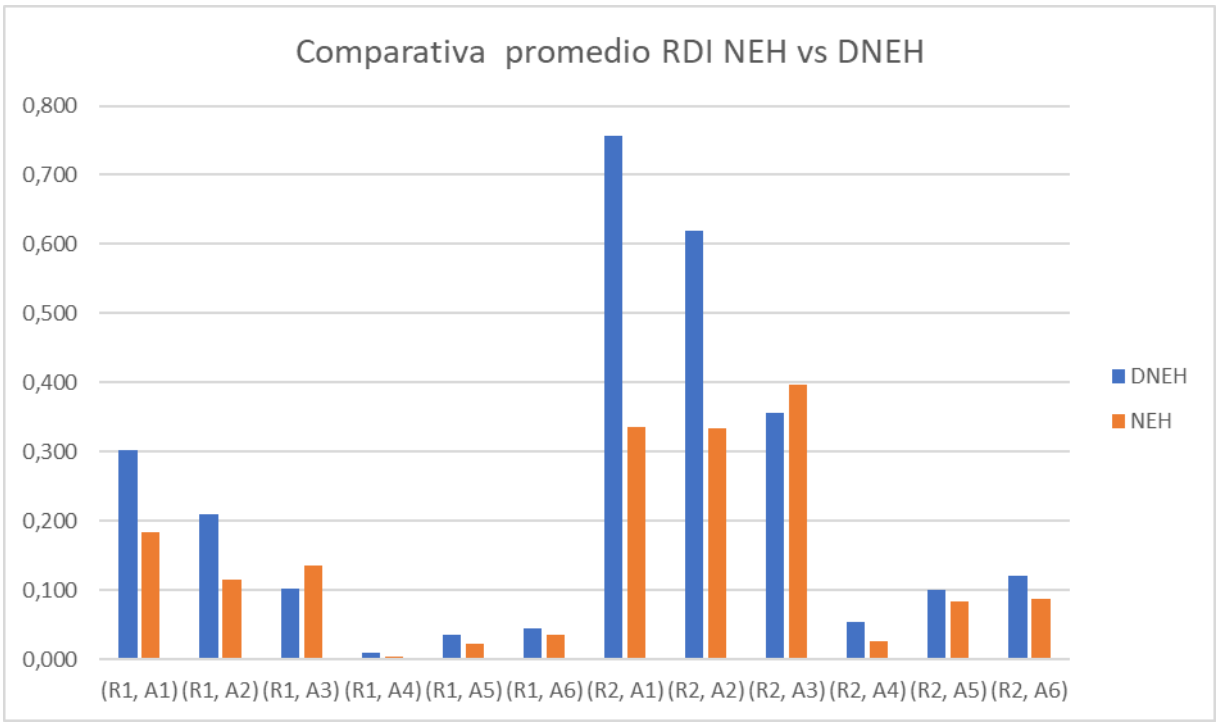

Ilustración 14. Comparativa promedio RDI

Resulta interesante conocer cuál de las heurísticas programadas es mejor para evaluar un problema concreto según los tres parámetros que lo definen: el número de fábricas *f*, de máquinas *m* o de trabajos *n*. Para ellos se hace el promedio del $RDI^h{}_I$ de aquellas instancias que comparten el mismo valor de *f*, *m* o *n*, según el parámetro que nos interese para conocer el menor valor obtenido para un valor del parámetro en concreto.

Tanto la nueva heurística propuesta DNEH (*R1*, *A4*) como la adaptada NEH (*R1*, *A4*), son las dos heurísticas que generan los mejores valores para cualquiera de los parámetros experimentados. Para los valores *f* = 4, *m* = 2, 3 y *n* = 6, 14, 16 la nueva heurística es mejor que la adaptada tal y como se muestra en *Tabla 4*, *Tabla 5* y *Tabla 6*. Cabe destacar que el promedio de $RDI^h{}_I$ para la heurística NEH (*R1*, *A4*) para *m* = 2, 3 y *n* = 4, 10 es 0, es decir, para cada una de las instancias evaluadas cuyos parámetros *m* o *n* era uno de los indicados se ha obtenido la mejor solución siempre en esta heurística. Lo mismo ocurre para y *n* = 6, 16 con la heurística propuesto. En el caso *m* = 3, ambas heurísticas dan los mejores resultados.

f	Promedio de NEH_R1_A4	Promedio de DNEH_R1_A4
2	0,0033	0,0139
3	0,0039	0,0059
4	0,0053	0,0017
5	0,0034	0,0124
6	0,0007	0,0141
7	0,0011	0,0146
Total general	**0,0033**	**0,0094**

Tabla 4. Promedio RDI por parámetro f

m	Promedio de NEH_R1_A4	Promedio de DNEH_R1_A4
2	0,0000	0,0006
3	0,0000	0,0000
4	0,0058	0,0036
5	0,0061	0,0117
10	0,0018	0,0158
20	0,0039	0,0152
Total general	**0,0033**	**0,0094**

Tabla 5. Promedio RDI por parámetro m

n	Promedio de NEH_R1_A4	Promedio de DNEH_R1_A4
4	0,0000	0,0091
6	0,0058	0,0000
8	0,0021	0,0130
10	0,0000	0,0020
12	0,0066	0,0140
14	0,0105	0,0070
16	0,0027	0,0000
20	0,0049	0,0174
50	0,0031	0,0111
100	0,0007	0,0078
Total general	**0,0033**	**0,0094**

Tabla 6. Promedio RDI por parámetro n

Para completer el análisis del índice $RDI^{h}{}_{I}$, se ha realizado un contraste de hipótesis suponiendo igualdad de media entre los dos métodos. Para ello, se ha calculado el intervalo de confianza (IC) para la media de DNEH (*R1*, *A4*) y de NEH (*R1*, *A4*) con un nivel de confianza del 95%. Dado que la muestra está compuesta por los RDI de las 72 instancias, se puede suponer la aproximación a una distribución Normal. El cálculo se halla en *Tabla 7*. No existen evidencias estadísticas para aceptar la hipótesis de que los métodos sean iguales de efiencientes en media, puesto que el IC del método adaptado NEH (*R1*, *A4*) es [0.002, 0.005] y el IC para la nueva heurística DNEH (*R1*, *A4*) es [0.006, 0.013], ambos para el mismo nivel de confianza.

INTERVALO DE CONFIANZA PARA LA MEDIA

DNEH	MEDIA	0,009		
	DESV ESTÁNDAR	0,015		
	n	72		
	NIVEL DE SIGNIFICACIÓN	0,05		
	CANTIDAD PIVOTAL	0,003	Intervalo de confianza	0,013
				0,006
NEH	MEDIA	0,003		
	DESV ESTÁNDAR	0,007		
	n	72		
	NIVEL DE SIGNIFICACIÓN	0,05		
	CANTIDAD PIVOTAL	0,002	Intervalo de confianza	0,005
				0,002

Tabla 7. Cálculo intervalo de confianza para la media con α=0,05

7 CONCLUSIONES

La Programación de la Producción juega un papel bastante importante en un elevado número de empresas, ya no solo manufactureras, sino que también abarca otros sectores como la distribución o el transporte entre otras. Es una decisión operativa con bastante complejidad que precisa conocer la estructura de la planta y la demanda a satisfacer. En este proyecto, la estructura o entorno que se ha llevado al estudio es el entorno *Distributed Flow shop*. En este entorno se tiene un conjunto de fábricas idénticas que se comportan cada una de ellas como un *Flow shop* de permutación, es decir, cada una de las fábricas tiene el mismo número de máquinas, *m*, con una disposición tipo taller de máquinas en serie y los trabajos pasan por cada una de las maquinas en el mismo orden. Además, se exige por la restricción de permutación que la secuencia de trabajos por máquina sea la misma.

En la literatura, hasta el momento este entorno había sido estudiado con distintos objetivos como es el *Total Flowtime* o *Makespan*. Ahora un nuevo problema se plantea al estudiar el *Total Core Idle Time*. Este objetivo busca minimizar los tiempos ociosos de las máquinas y puede ser considerado como un objetivo relacionado con la sostenibilidad, ya que los tiempos ociosos de las máquinas generan un coste energético innecesario al tener las máquinas encendidas sin procesar ningún trabajo. Además, este objetivo también tiene relación con la ineficiencia del proceso, ya que en los tiempos ociosos entre trabajos no se pueden aprovechar de forma tan clara como los tiempos ociosos al inicio o al fin de un programa, para realizar, por ejemplo, tareas de mantenimiento de las máquinas.

Con la adaptación de la heurística DLR y DNEH propuesto en (Pan et al., 2019b), el clásico NEH adaptado al *Distributed Flow shop* con las dos representaciones de las soluciones propuesto en (Fernandez-Viagas et al., 2018) y una nueva heurística constructiva planteada como una nueva adaptación del NEH con trabajos ordenados según el índice *Idle Time,* $IF_{j,0}$, de (Pan et al., 2019b), se ha buscado el mejor método de resolución para el problema propuesto.

Un total de veintiséis heurísticas han sido programadas en lenguaje de programación C mediante CodeBlocks y se han resuelto las instancias extraídas de la literatura de (Naderi & Ruiz, 2010) . Los resultados obtenidos se han comparado mediante el índice *RDI* con el objetivo de conocer el mejor de los métodos. La existente heurísticas NEH (*R1, A4*) y la nueva DNEH (*R1, A4*) generan buenos resultados para las instancias planteadas. Para algunos valores de *f, n* o *m* una heurística funciona mejor que la otra, pero se ha demostrado que la heurística NEH (*R1, A4*) es estadísticamente mejor que la propuesta. No obstante, como se ha ido comentando a lo largo del proyecto, este es un nuevo problema cuyo estudio sigue abierto y los métodos propuestos pueden ser mejorados.

BIBLIOGRAFÍA

Chen, J. fang, Wang, L., & Peng, Z. ping. (2019). A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling. *Swarm and Evolutionary Computation*, *50*(July), 100557. https://doi.org/10.1016/j.swevo.2019.100557

Fernandez-Viagas, V., & Framinan, J. M. (2014). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers and Operations Research*, *45*, 60–67. https://doi.org/10.1016/j.cor.2013.12.012

Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. M. (2018). The distributed permutation flow shop to minimise the total flowtime. *Computers and Industrial Engineering*, *118*(January), 464–477. https://doi.org/10.1016/j.cie.2018.03.014

Framinan, J. M., Leisten, R., & Ruiz García, R. (2014). Manufacturing Scheduling Systems. In *Manufacturing Scheduling Systems*. https://doi.org/10.1007/978-1-4471-6272-8

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and heuristic in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, *5*, 287–326. https://ac.els-cdn.com/S016750600870356X/1-s2.0-S016750600870356X-main.pdf?_tid=cbf345a3-808d-42df-9560-bf8a52069d0e&acdnat=1550949881_e9837bfdf6316caefaf71ae2f3779b10

Librería SCHEDULE – Grupo de Investigación Organización Industrial. (n.d.). Retrieved September 3, 2021, from http://grupo.us.es/oindustrial/investigacion/software-y-librerias/libreria-schedule/

Maassen, K., Perez-Gonzalez, P., & Günther, L. C. (2020). Relationship between common objective functions, idle time and waiting time in permutation flow shop scheduling. *Computers and Operations Research*, *121*, 104965. https://doi.org/10.1016/j.cor.2020.104965

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers and Operations Research*, *37*(4), 754–768. https://doi.org/10.1016/j.cor.2009.06.019

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95. https://doi.org/10.1016/0305-0483(83)90088-9

Pan, Q. K., Gao, L., Wang, L., Liang, J., & Li, X. Y. (2019a). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications*, *124*, 309–324. https://doi.org/10.1016/j.eswa.2019.01.062

Pan, Q. K., Gao, L., Wang, L., Liang, J., & Li, X. Y. (2019b). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications*, *124*, 309–324. https://doi.org/10.1016/j.eswa.2019.01.062

Pinedo, M. L. (2012). *Scheduling. Theory, Algorithms, and Systems* (Fourth). Springer New York Dordrecht Heidelberg London. https://doi.org/10.1007/978-1-4614-2361-4

Ruiz, R., Vallada, E., & Fernández-Martínez, C. (2009). Scheduling in Flowshops with No-Idle Machines. In U. K. Chakraborty (Ed.), *Computational Intelligence in Flow Shop and Job Shop Scheduling* (pp. 21–51). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-02836-6_2

Wang, G., Gao, L., Li, X., Li, P., & Tasgetiren, M. F. (2020). Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm. *Swarm and Evolutionary Computation*, *57*(April), 100716. https://doi.org/10.1016/j.swevo.2020.100716

Wang, J. J., & Wang, L. (2020). A Knowledge-Based Cooperative Algorithm for Energy-Efficient Scheduling of Distributed Flow-Shop. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *50*(5), 1805–1819. https://doi.org/10.1109/TSMC.2017.2788879

ANEXO CÓDIGO PROGRAMACIÓN EN C

```
#include <schedule.h>
#include <stdio.h>
#include<conio.h>
#include<time.h>

MAT_INT loadDPFS (char *data, int *jobs, int *machs, int *facts);
void outputDPFS ( char *inputfile, char *outputfile, int time, int jobs, int machs, int facts, VECTOR_INT seq, MAT_INT seq_fabs, double objetive);
int CIT (VECTOR_INT seq, MAT_INT pt, int jobs, int m, int n);
int sumCIT (MAT_INT seqxfab, VECTOR_INT jobs_facts, MAT_INT pt, int jobs, int machs, int facts);
void DNEH (VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n, int m, int f);
void DLR (VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n, int m, int f);
void NEH_R1 (VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n, int m, int f,int A_i);
void NEH_R2 (VECTOR_INT *seq, VECTOR_INT *jobs_facts,MAT_INT *seq_fabs, MAT_INT pt, int n, int m, int f,int A_i);
int A_1 (MAT_INT seqxfabs, VECTOR_INT jobs_facts,int f, int n, int m, MAT_INT pt);
int A_2 (MAT_INT seqxfabs, VECTOR_INT jobs_facts,int f, int n, int m, MAT_INT pt);
int A_3(VECTOR_INT *fab_insertion, MAT_INT seqxfab, VECTOR_INT jobs_facts, int job_to_insert, int f, int n, int m, MAT_INT pt);
int A_4(VECTOR_INT *fab_insertion, MAT_INT seqxfab, VECTOR_INT jobs_facts, int job_to_insert, int f, int n, int m, MAT_INT pt);
int A_5(VECTOR_INT *fab_insertion, MAT_INT seqxfab, VECTOR_INT jobs_facts, int job_to_insert, int f_max, int f, int n, int m, MAT_INT pt);
int A_6(VECTOR_INT *fab_insertion, MAT_INT seqxfab, VECTOR_INT jobs_facts, int job_to_insert, int f, int n, int m, MAT_INT pt);
void DNEH_R1(VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n, int m, int f,int A_i);
void DNEH_R2(VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n, int m, int f,int A_i);
void insertion (VECTOR_INT seq, VECTOR_INT *nueva_seq, int pos, int job_to_insert, int n);
VECTOR_INT matriz_a_vector(MAT_INT pi, int n, int f);
int makespan (VECTOR_INT seq, int n, int m, MAT_INT pt);
void copy_mat_int( MAT_INT original, MAT_INT destino, int filas, int cols);
```

```
int main(int argc, char *argv[])
{
   int jobs, machs, facts;
   MAT_INT pt;

   pt=loadDPFS(argv[1],&jobs,&machs,&facts);

   VECTOR_INT seq= DIM_VECTOR_INT(jobs);
   setval_vector(seq,jobs,-1);
   MAT_INT seq_fabs=DIM_MAT_INT(facts,jobs);
   setval_matrix(seq_fabs,facts, jobs, -1);

   int objetive, time;
   VECTOR_INT jobs_facts=DIM_VECTOR_INT(facts);
   setval_vector(jobs_facts,facts,0);

   clock_t clock (void);
   DNEH(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts);
   time=((double)clock()/CLK_TCK);
   printf("Secuencia DNEH\n");
   print_vector(seq,jobs);
   objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
   outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

   clock_t clock (void);
   DLR(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts);
   time=((double)clock()/CLK_TCK);
   printf("Secuencia DLR\n");
   print_vector(seq,jobs);
   objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
   outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

   clock_t clock (void);
   NEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,1);
   time=((double)clock()/CLK_TCK);
   printf("Secuencia NEH (R_1,A_1)\n");
   print_vector(seq,jobs);
```

```
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);*/

clock_t clock (void);
NEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,2);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_1,A_2)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
NEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,3);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_1,A_3)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
NEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,4);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_1,A_4)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);


clock_t clock (void);
NEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,5);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_1,A_5)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
NEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,6);
```

```
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_1,A_6)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
NEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,1);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_2,A_1)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
NEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,2);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_2,A_2)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
NEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,3);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_2,A_3)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
NEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,4);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_2,A_4)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);
```

```
clock_t clock (void);
NEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,5);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_2,A_5)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
NEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,6);
time=((double)clock()/CLK_TCK);
printf("Secuencia NEH (R_2,A_6)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,1);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_1,A_1)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,2);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_1,A_2)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,3);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_1,A_3)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
```

```
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,4);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_1,A_4)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,5);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_1,A_5)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R1(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,6);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_1,A_6)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,1);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_2,A_1)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,2);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_2,A_2)\n");
```

```
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,3);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_2,A_3)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,4);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_2,A_4)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,5);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_2,A_5)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

clock_t clock (void);
DNEH_R2(&seq,&jobs_facts,&seq_fabs,pt,jobs,machs,facts,6);
time=((double)clock()/CLK_TCK);
printf("Secuencia DNEH (R_2,A_6)\n");
print_vector(seq,jobs);
objetive=sumCIT(seq_fabs,jobs_facts,pt,jobs,machs,facts);
outputDPFS(argv[1],argv[2],time,jobs,machs,facts,seq,seq_fabs,objetive);

free_vector(seq);
free_vector(jobs_facts);
```

```
   free_matrix(pt,jobs);
   free_matrix(seq_fabs,facts);

   return 0;

}

//LECTURA DE ENTORNO DPFS
MAT_INT loadDPFS (char *data, int *jobs, int *machs, int *facts)
{

   int temp_data;
   register int i,j;
   MAT_INT pt;

   FILE *input;
   input=fopen(data,"rt"); //Abre el archivo con el nombre dado y lee
   if (input==NULL)
   {
      printf("No se ha abierto el fichero\n");
   }

   //Lectura de número de trabajos, máquinas y fábricas
   fscanf(input,"%d\t",&temp_data);
   *jobs=temp_data;

   fscanf(input,"%d\n",&temp_data);
   *machs=temp_data;

   fscanf(input,"%d\n",&temp_data);
   *facts=temp_data;
   //printf("JOBS:%d\t MACHINES: %d\nFACTORIES: %d\n",*jobs,*machs,*facts);

   //Lectuta de tiempos de procesos
   pt=DIM_MAT_INT(*jobs,*machs); //Tamaño de la matriz
   for (j=0;j<*jobs;j++)
    {
```

```
 for (i=0;i<*machs;i++)
{
   fscanf(input,"%d\t",&temp_data);
   pt[j][i]=temp_data;
}
}
//print_matrix(pt,*jobs,*machs);
printf("Fin lectura de datos\n");

fclose(input);
return pt;
}

//RESULTADOS ENTORNO DPFS
void outputDPFS ( char *inputfile, char *outputfile, int CPU, int jobs, int machs, int facts, VECTOR_INT seq,
MAT_INT seq_fabs, double objetive)
{
FILE *output;
register int j,k;

output=fopen(outputfile,"at"); //Crea un archivo con el nombre dado en outputfile y escribe en él

time_t timer;
struct tm *tblock;

timer = time(NULL);
tblock = localtime(&timer);

fprintf(output,"\n%s\t%s\n",inputfile,asctime(tblock));
fprintf(output,"CPU(segundos) = %d\n",CPU);
//Escritura valor FO
fprintf(output,"FO=%f\n", objetive);

//Escritura secuencia
fprintf(output,"%s=(","Secuencia");
for(j=0;j<jobs;j++)
{
   fprintf(output,"\t%d",seq[j]);
```

```
    }

    //Escritura secuencia por fabrica
    fprintf(output,")\n%s","Secuencia por fabrica ");

    for(k=0;k<facts;k++)
    {
       fprintf(output,"\n%s %d","Fabrica",k);
       for(j=0;j<jobs;j++)
       {
          fprintf(output,"\t%d",seq_fabs[k][j]);
       }

    }
    printf("Fin escritura de resultados\n");
    fclose(output);
}

//CIT POR FÁBRICA
int CIT (VECTOR_INT seq, MAT_INT pt, int jobs, int m, int n)
{
    printf("Entra en funcion CIT\n");
    float objetive=0;
    int register i,j; //máquina-i, trabajo-j
    MAT_INT S=DIM_MAT_INT(n,m);
    setval_matrix(S,n,m,-1);
    printf("Mat S\n");
    print_matrix(S,n,m);
    MAT_INT ct=DIM_MAT_INT(n,m); // En una matriz se guardan los completion time de cada trabajo en cada máquina
    setval_matrix(ct,n,m,0);
    printf("Mat ct\n");
    print_matrix(ct,n,m);
    if (seq[0]!=-1)
    {
       S[seq[0]][0]=0;
          for (j=1;j<jobs;j++) //En la primera máquina no hay CIT, un trabajo empieza cuando termina el anterior
```

```
        {
          S[seq[j]][0]+=S[seq[j-1]][0]+pt[seq[j-1]][0];
        printf("S[%d][0]=%d\n",seq[j],S[seq[j]][0]);
        }
        for (i=1;i<m;i++) //El primer trabajo secuenciado pasa a la siguiente máquina cuando termina el anterior
        {
        S[seq[0]][i]=S[seq[0]][i-1]+pt[seq[0]][i-1];
                printf("S[%d][%d]=%d\n",seq[0],i,S[seq[0]][i]);
        }
        for (i=1;i<m;i++) //A partir de la segunda máquina y el segundo trabajo se generan los CIT
        {
                for (j=1;j<jobs;j++)
        {
                printf("Bucle jobs=%d\n",j);
                if(S[seq[j]][i-1]+pt[seq[j]][i-1]<S[seq[j-1]][i]+pt[seq[j-1]][i])
                {
                        S[seq[j]][i]=S[seq[j-1]][i]+pt[seq[j-1]][i];
                        printf("El trabajo anterior en la maquina tarda mas que el mismo trabajo en la maquina anterior\n");
                         }
                 else
                {
                        S[seq[j]][i]=S[seq[j]][i-1]+pt[seq[j]][i-1];
                        printf("El trabajo en la maquina anterior tarda mas que el anterior en la maquina\n");
                }
        }
   }

   printf("Matriz start time\n");
   print_matrix(S,n,m);

   for(j=0;j<jobs;j++) //Completion time por trabajos
   {
      for (i=0;i<m;i++)
      {
         ct[seq[j]][i]=S[seq[j]][i]+pt[seq[j]][i];
      }
   }
```

```
   printf("Matriz completion time trabajos\n");
   print_matrix(ct,n,m);



   for (i=1;i<m;i++) //Calculo CIT
   {
      for(j=1;j<jobs;j++)
      {
         if (S[seq[j]][i]>ct[seq[j-1]][i]) //Si la diferencia es positiva, entonces se va sumando
         {
            objetive+=S[seq[j]][i]-ct[seq[j-1]][i];
         }
      }
   }



   }



   free_matrix(S,n);
   free_matrix(ct,n);
   printf("Calculo de CIT=%f\n",objetive);

   return objetive;
}



//FO SUM (CIT)
int sumCIT (MAT_INT seqxfab, VECTOR_INT jobs_facts, MAT_INT pt, int jobs, int machs, int facts)
{
   printf("Entra en funcion sumCIT\n");
   float objetive=0;
   int register i,j,f; //máquina-i, trabajo-j, fábrica-f
   MAT_INT C=DIM_MAT_INT(facts,machs); // En un matriz se van a guardar los completion time de cada
máquina en cada fábrica
   setval_matrix(C,facts,machs,0);
   MAT_INT S=DIM_MAT_INT(jobs,machs);
   setval_matrix(S,jobs,machs,0);
```

```
MAT_INT ct=DIM_MAT_INT(jobs,machs); // En una matriz se guardan los completion time de cada trabajo en cada máquina
setval_matrix(ct,jobs,machs,0);
for(f=0;f<facts;f++) //Para cada fábrica
{
  if (seqxfab[f][0]!=-1)
  {
    S[seqxfab[f][0]][0]=0;
    for (j=1;j<jobs_facts[f];j++) //En la primera máquina no hay CIT, un trabajo empieza cuando termina el anterior
    {
      S[seqxfab[f][j]][0]+=S[seqxfab[f][j-1]][0]+pt[seqxfab[f][j-1]][0];
    }
      C[f][0]=S[seqxfab[f][j-1]][0]+pt[seqxfab[f][j-1]][0]; //Completion time de la máquina 0

    for (i=1;i<machs;i++) //El primer trabajo secuenciado pasa a la siguiente máquina cuando termina el anterior
    {
      S[seqxfab[f][0]][i]=S[seqxfab[f][0]][i-1] + pt[seqxfab[f][0]][i-1];
    }

    for (i=1;i<machs;i++) //A partir de la segunda máquina y el segundo trabajo se generan los CIT
    {
      for (j=1;j<jobs_facts[f];j++)
      {
        if(S[seqxfab[f][j]][i-1]+pt[seqxfab[f][j]][i-1]<S[seqxfab[f][j-1]][i]+pt[seqxfab[f][j-1]][i])
        {
          S[seqxfab[f][j]][i]=S[seqxfab[f][j-1]][i]+pt[seqxfab[f][j-1]][i]; //El trabajo anterior en la máquina tarda más que el mismo trabajo en la máquina anterior
        }
        else
        {
          S[seqxfab[f][j]][i]=S[seqxfab[f][j]][i-1]+pt[seqxfab[f][j]][i-1]; //El trabajo en la máquina anterior tarda más que el anterior en la máquina
        }
      }
      C[f][i]=S[seqxfab[f][j-1]][i]+pt[seqxfab[f][j-1]][i]; //Completion time de la máquina i
    }

  }
```

```
    printf("Matriz completion time maquinas\n");
    print_matrix(C,facts,machs);
    printf("Matriz start time\n");
    print_matrix(S,jobs,machs);

  }
  for(f=0;f<facts;f++)
  {
    for(j=0;j<jobs_facts[f];j++) //Completion time por trabajos
    {
      if(seqxfab[f][j]!=-1)
      {
        for (i=0;i<machs;i++)
        {
          ct[seqxfab[f][j]][i]=S[seqxfab[f][j]][i]+pt[seqxfab[f][j]][i];
        }

      }

    }

  }

  printf("Matriz completion time trabajos\n");
  print_matrix(ct,jobs,machs);


  for (i=1;i<machs;i++) //Calculo CIT
  {
    for(f=0;f<facts;f++)
    {
      for(j=1;j<jobs;j++)
      {
        if (seqxfab[f][j]!=-1 && seqxfab[f][j-1]!=-1 && S[seqxfab[f][j]][i]>ct[seqxfab[f][j-1]][i]) //Si la diferencia es positiva, entonces se va sumando
        {
```

```
                objetive+=S[seqxfab[f][j]][i]-ct[seqxfab[f][j-1]][i];
            }
        }

    }

}

free_matrix(C, facts);
free_matrix(S,jobs);
free_matrix(ct, jobs);

printf("Calculo de sumCIT=%f\n",objetive);

return objetive;
}

void DNEH (VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n, int
m, int f)
{
printf("\n******Calculo de la secuencia DNEH******\n");
VECTOR_FLOAT IF0= DIM_VECTOR_FLOAT(n);
VECTOR_FLOAT IT0= DIM_VECTOR_FLOAT(n);
VECTOR_FLOAT C= DIM_VECTOR_FLOAT(n);
VECTOR_INT landa= DIM_VECTOR_INT(n);

MAT_INT pi= DIM_MAT_INT(f,n);
register int i,j,k;
for (k=0;k<f;k++)
{
    for(j=0;j<n;j++)
    {

        pi[k][j]=-1;
        IF0[j]=0;
        IT0[j]=0;
        C[j]=0;
        landa[j]=-1;
```

```
    }
}

int dim=n; // para extraer los primeros f trabajos
for (j=0;j<n;j++)
{
    for (i=1;i<m;i++)
    {
        C[j]+=pt[j][i-1];
        IT0[j]+=(m*(C[j]))/(i);
    }
    C[j]+=pt[j][m-1]; // Makespan
    printf("C[%d]=%f",j,C[j]);
    IF0[j]=((n/f)-2)*IT0[j]+ C[j];
    printf("IF0[%d]=%f\n",j,IF0[j]);
}
printf("Vector IF0\t");
print_vector(IF0,n);
sort_vector(IF0,landa,n,'A'); // Vector de trabajos landa ordenado de manera creciente
printf("Vector landa\t");
print_vector(landa,n);
for(j=0;j<f;j++)
{
    pi[j][0]=landa[0]; // Asignar los primeros f trabajos, uno a cada fábrica
    extract_vector(landa,dim,0); //Extraer los trabajos asignados de landa
    dim--;
}

setval_vector(*jobs_facts,f,1);

VECTOR_INT pi_k_vector = DIM_VECTOR_INT(n);
VECTOR_INT best_pi_k_insertion = DIM_VECTOR_INT(n);
VECTOR_INT pi_k_insertion = DIM_VECTOR_INT(n);
VECTOR_INT seq_a_evaluar=DIM_VECTOR_INT(n);
VECTOR_INT best_seq_a_evaluar=DIM_VECTOR_INT(n);
VECTOR_INT jobs_facts_copia =DIM_VECTOR_INT(f);
copy_vector(*jobs_facts,jobs_facts_copia,f);
best_seq_a_evaluar=matriz_a_vector(pi,n,f);
```

```
printf("Mat pi a vector\t");
print_vector(best_seq_a_evaluar,n);
int fo, best_fo;
best_fo = sumCIT(pi,jobs_facts_copia,pt,n,m,f); //Actualmente tengo f trabajos asignados
printf("FO tras asignar los f primeros trabajos=%d\n",best_fo);
int first_job, k_inserted;
register int l;
while (dim>0) //Mientras queden trabajos por insertar
{
   first_job=landa[0];

   //Suponemos que la mejor fo es el trabajo insertado en la primera posicion de la primera fábrica para ir comparando
   for(j=0;j<n;j++) // Copia de la fila de la matriz pi en un vector
   {
      pi_k_vector[j]=pi[0][j];
   }
   insertion(pi_k_vector,&best_pi_k_insertion,0,first_job,n);
   for(l=0;l<n;l++) // Copia del vector insertado en la matriz pi
   {
      pi[0][l]=best_pi_k_insertion[l];
   }
   printf("Matriz pi tras insertion en pos 0\n");
   print_matrix(pi,f,n);
   best_seq_a_evaluar=matriz_a_vector(pi,n,f);
   printf("Mejor secuencia a evaluar\n");
   print_vector(best_seq_a_evaluar,n);
   jobs_facts_copia[0]++;
   printf("Jobs_facts_copia\n");
   print_vector(jobs_facts_copia,f);
   best_fo = sumCIT(pi,jobs_facts_copia,pt,n,m,f);
   k_inserted=0;
   printf("best_fo=%d",best_fo);
   jobs_facts_copia[0]--;

   for(j=0;j<n;j++) // Copia de la fila de la matriz pi en un vector
   {
      pi[0][j]=pi_k_vector[j];
```

```
}
printf("Matriz pi tras conocer best_fo inicial\n");
print_matrix(pi,f,j);

for (k=0;k<f;k++) //Para todas las fábricas
{

   for(j=0;j<n;j++) // Copia de la fila de la matriz pi en un vector
   {
      pi_k_vector[j]=pi[k][j];

   }

   printf("Secuencia parcial pi_%d\n",k);
   print_vector(pi_k_vector,n);
   printf("Jobs_facts_%d\n",k);
   print_vector(jobs_facts_copia,f);
   for(j=0;j<=jobs_facts_copia[k];j++) //Insertar el primer trabajo de landa en todas las posiciones de la fabrica a evaluar
   {

      insertion(pi_k_vector,&pi_k_insertion,j,first_job,n);
      for(l=0;l<n;l++) // Copia del vector insertado en la matriz pi
      {
         pi[k][l]=pi_k_insertion[l];

      }
      printf("Matriz pi tras insertion\n");
      print_matrix(pi,f,n);
      seq_a_evaluar=matriz_a_vector(pi,n,f);
      printf("Secuencia a evaluar\n");
      print_vector(seq_a_evaluar,n);
      jobs_facts_copia[k]++;
      fo = sumCIT(pi,jobs_facts_copia,pt,n,m,f);
      printf("Valor FO (pos=%d en fact=%d)=%d",j,k,fo);
    if (fo<best_fo) //Actualizacion de la mejor fo quedandonos siempre con la primera
    {
```

```
                printf("Entra en if: Se actualiza best_fo\n")
            best_fo=fo;
            k_inserted=k;
            jobs_facts_copia[k]--;

            copy_vector(pi_k_insertion,best_pi_k_insertion,n);
            copy_vector(seq_a_evaluar,best_seq_a_evaluar,n);
          }
          else
          {
            jobs_facts_copia[k]--;
            printf("Entra en else\n");

          }

          }

       for(j=0;j<n;j++) // Copia de la fila de la matriz pi en un vector
          {
            pi[k][j]=pi_k_vector[j];

          }
       }
for(j=0;j<n;j++)
{
   pi[k_inserted][j]=best_pi_k_insertion[j];

}
extract_vector(landa,dim,0); // Extraer el trabajo insertado
jobs_facts_copia[k_inserted]++; //Actualización del vector de trabajos asignados por fábrica
dim--;
}
copy_vector(jobs_facts_copia,*jobs_facts,f);
copy_vector(best_seq_a_evaluar,*seq,n);
copy_mat_int(pi,*seq_fabs,f,n);

free_vector(IF0);
free_vector(IT0);
```

```
   free_vector(C);
   free_vector(landa);
   free_vector(pi_k_vector);
   free_vector(best_pi_k_insertion);
   free_vector(pi_k_insertion);
   free_vector(seq_a_evaluar);
   free_vector(best_seq_a_evaluar);
   free_vector(jobs_facts_copia);
   free_matrix(pi,f);

  printf("Sale de la funcion DNEH\n");
}

void DLR (VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n, int m, int f)
{
   printf("\n******Calculo de la secuencia DLR******\n");
   float n_copia=n;
   float f_copia=f;
   float n_f=n_copia/f_copia;

   VECTOR_FLOAT IF0= DIM_VECTOR_FLOAT(n);
   VECTOR_FLOAT IT0= DIM_VECTOR_FLOAT(n);
   VECTOR_FLOAT C= DIM_VECTOR_FLOAT(n);
   VECTOR_INT landa= DIM_VECTOR_INT(n);
   MAT_INT pi= DIM_MAT_INT(f,n);
   register int i,j,k;
   for (k=0;k<f;k++)
   {
      for(j=0;j<n;j++)
      {

         pi[k][j]=-1;
         IF0[j]=0;
         IT0[j]=0;
         C[j]=0;
         landa[j]=-1;
      }
```

```
}

int aux, dim=n; // para extraer los primeros f trabajos

//CÁLCULO DEL ÍNDICE IF0
for (j=0;j<n;j++)
{
   for (i=1;i<m;i++)
   {
      C[j]+=pt[j][i-1];
      IT0[j]+=(m*(C[j]))/(i);
   }
   C[j]+=pt[j][m-1]; // Makespan de todos los trabajos si se secuenciaran los primeros
   IF0[j]=((n_f)-2)*IT0[j]+ C[j];
}
//TRABAJOS ORDENADOS DE MANERA CRECIENTE SEÚN IF0
printf("Vector IF0\t");
print_vector(IF0,n);
sort_vector(IF0,landa,n,'A'); // Vector de trabajos landa ordenado de manera creciente
printf("Vector landa\t");
print_vector(landa,n);

//ASIGNAR LOS PRIMEROS f TRABAJOS A LAS f FÁBRICAS
for(j=0;j<f;j++)
{
   pi[j][0]=landa[0];
   aux=extract_vector(landa,dim,0); //Extraer los trabajos asignados de landa
   dim--;
}
printf("Matriz pi con los primero f trabajos\n");
print_matrix(pi,f,n);
setval_vector(*jobs_facts,f,1);
*seq=matriz_a_vector(pi,n,f);

int makespan, min_makespan, k_opt;
int max=0, j_opt,pos;
float min_IF_nk;
VECTOR_INT pi_k_vector = DIM_VECTOR_INT(n);
```

```
VECTOR_INT jobs_facts_copia =DIM_VECTOR_INT(f);
MAT_INT S=DIM_MAT_INT(n,m);
MAT_INT CT =DIM_MAT_INT(f,n);
setval_matrix(CT,f,n,0);
setval_matrix(S,n,m,-1);
copy_vector(*jobs_facts,jobs_facts_copia,f);
VECTOR_FLOAT IF_nk=DIM_VECTOR_FLOAT(n);
setval_vector(IF_nk,n,0);
VECTOR_FLOAT IT_nk=DIM_VECTOR_FLOAT(n);
setval_vector(IT_nk,n,0);

while(dim>0) // Mientras queden trabajos por insertar
{
  //CÁLCULO DE COMPLETION TIME
  for(k=0;k<f;k++)
  {
    for(j=0;j<n;j++) //Copia de la fila de la matriz pi en un vector
    {
      pi_k_vector[j]=pi[k][j];

    }
    printf("vector copia fila %d\t",k);
    print_vector(pi_k_vector,n);
    S[pi_k_vector[0]][0]=0; //En el instante cero el primer trabajo secuanciado en la fabrica

    for (j=1;j<jobs_facts_copia[k];j++) //En la primera máquina no hay tiempos ociosos, un trabajo empieza cuando termina el anterior
    {

      S[pi_k_vector[j]][0]+=S[pi_k_vector[j-1]][0]+pt[pi_k_vector[j-1]][0];
      printf("S[%d][0]=%d\n",pi_k_vector[j],S[pi_k_vector[j]][0]);
    }
    for (i=1;i<m;i++) //El primer trabajo secuenciado pasa a la siguiente máquina cuando termina el anterior
    {
      S[pi_k_vector[0]][i]=S[pi_k_vector[0]][i-1] + pt[pi_k_vector[0]][i-1];
    }
    CT[k][pi_k_vector[0]]= S[pi_k_vector[0]][m-1]+pt[pi_k_vector[0]][m-1];  //Completion time del primer trabajo secuenciado en la fábrica k
```

```
printf("CT[%d][%d]=%d\n",k,pi_k_vector[0],S[pi_k_vector[0]][m-1]+pt[pi_k_vector[0]][m-1]);

for (j=1;j<jobs_facts_copia[k];j++) //A partir del segundo trabajo y la segunda máquina se generan los CIT
{
  for (i=1;i<m;i++)
  {
    if(S[pi_k_vector[j]][i-1]+pt[pi_k_vector[j]][i-1]<S[pi_k_vector[j-1]][i]+pt[pi_k_vector[j-1]][i])
    {
      S[pi_k_vector[j]][i]=S[pi_k_vector[j-1]][i]+pt[pi_k_vector[j-1]][i]; //El trabajo anterior en la máquina tarda más que el mismo trabajo en la máquina anterior
    }
    else
    {
      S[pi_k_vector[j]][i]=S[pi_k_vector[j]][i-1]+pt[pi_k_vector[j]][i-1]; //El trabajo en la máquina anterior tarda más que el anterior en la máquina
    }
  }
  CT[k][pi_k_vector[j]]= S[pi_k_vector[j]][m-1]+pt[pi_k_vector[j]][m-1];
}
}

printf("Matriz CT, f filas y n columnas\n");
  print_matrix(CT,f,n);
printf("Matriz S, n filas y m maquinas\n");
print_matrix(S,n,m);

  //MENOR MAKESPAN
min_makespan=CT[0][pi[0][jobs_facts_copia[0]-1]]; //CT del ultimo trabajo secuenciado en la fábrica 0
printf("min_makespan_calculado=%d\n",min_makespan);
k_opt=0;
for (k=1;k<f;k++)
{
  makespan=CT[k][pi[k][jobs_facts_copia[k]-1]];
  if (makespan<min_makespan)
  {
    min_makespan=makespan;
    k_opt=k;
```

```
    }
    printf("makespan=%d \t min_makespan=%d\n",makespan,min_makespan);
    printf("k_opt=%d\n",k_opt);
  }

  //BUSCA j ÓPTIMO PARA LA FÁBRICA k_opt
  for (j=0;j<dim;j++)
  {
    for(i=1;i<m;i++)
    {  if (S[jobs_facts_copia[k_opt]][i]+pt[jobs_facts_copia[k_opt]][i]<S[landa[j]][i-1]+pt[landa[j]][i-1])
        {
          max=S[landa[j]][i-1]+pt[landa[j]][i-1]-S[jobs_facts_copia[k_opt]][i]-
pt[jobs_facts_copia[k_opt]][i];
          printf("max en IT_nk[%d] es %d\n",landa[j],max);

        }
      IT_nk[landa[j]]+=(m*max)/(i+(jobs_facts_copia[k_opt])*(m-i)*((n_f)-2));
      printf("IT_nk[%d]=%f\n",landa[j],IT_nk[landa[j]]);
      max=0;
    }
    IF_nk[landa[j]]=((n_f)-jobs_facts_copia[k_opt]-2)*IT_nk[landa[j]]+min_makespan;

  }
  printf("Vector IF_nk\n");
  print_vector(IF_nk,n);
  min_IF_nk=IF_nk[landa[0]];
  j_opt=landa[0];
  pos=0;
  for (j=1;j<n;j++) // Comparar los IF_nk para obtener el trabajo j
    {
      if (IF_nk[landa[j]]<min_IF_nk && IF_nk[landa[j]]!=0 )
      {
        min_IF_nk=IF_nk[landa[j]];
        j_opt=landa[j];
      }
    }
  printf("min_IF_nk=%f\t corresponde al trabajo %d\n",min_IF_nk, j_opt);
  setval_vector(IF_nk,n,0);
```

```
        //INSERTAR AL FINAL DE LA SECUENCIA DE k_opt EL TRBAJO j_opt
    pi[k_opt][jobs_facts_copia[k_opt]]=j_opt;
    jobs_facts_copia[k_opt]++;
    printf("Matriz tras insertar j_opt en k_opt\n");
    print_matrix(pi, f, n);
    printf("Vector jobs_facts_copia\n");
    print_vector(jobs_facts_copia,f);

        //EXTRAER EL TRABAJO j_opt DE LA SECUENCIA landa
    pos=search_vector(landa,dim,j_opt);
    printf("Vector landa antes de extraer %d en la pos del vector %d\n",j_opt,pos);
    print_vector(landa,aux);
    aux=extract_vector(landa,dim,pos);
    printf("Vector landa tras extraccion\n");
    print_vector(landa,aux);
    dim--;
    printf("aux=%d\tdim=%d\n",aux,dim);


}
copy_vector(jobs_facts_copia,*jobs_facts,f);
copy_mat_int(pi,*seq_fabs, f, n);
*seq=matriz_a_vector(pi,n,f);

free_vector(IF0);
free_vector(IT0);
free_vector(C);
free_vector(landa);
free_vector(pi_k_vector);
free_vector(jobs_facts_copia);
free_vector(IF_nk);
free_vector(IT_nk);
free_matrix(pi,f);
free_matrix(S,n);
free_matrix(CT,f);

printf("Sale de función DLR\n");
```

```
}

void insertion (VECTOR_INT seq, VECTOR_INT *nueva_seq, int pos, int job_to_insert, int n)
{
   copy_vector(seq,*nueva_seq,n);
   extract_vector(*nueva_seq,n,n-1); //Extrae el valor -1 de la ultima posicion de la secuencia
   insert_vector(*nueva_seq,n,job_to_insert, pos);
   printf("Sale de la función insertion\n");

}

VECTOR_INT matriz_a_vector(MAT_INT pi, int n, int f)
{
   VECTOR_INT secuencia=DIM_VECTOR_INT(n);
   int j,k;
   int pos=0;
   for(k=0;k<f;k++)
   {
      for(j=0;j<n;j++)
      {
         if(pi[k][j]!=-1)
         {
            secuencia[pos]=pi[k][j];
            pos++;
         }

      }

   }



   printf("Sale de la funcion matriz-vector\n");

  return secuencia;
}

void NEH_R1 (VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n,
int m, int f,int A_i)
```

```
{
   printf("\n******Calculo de la secuencia NEH,R1******\n");
   VECTOR_INT sum_pt= DIM_VECTOR_INT(n);
   setval_vector(sum_pt,n,0);
   VECTOR_INT omega=DIM_VECTOR_INT(n);
   setval_vector(omega,n,-1);
   VECTOR_INT pi_wkl=DIM_VECTOR_INT(n);
   setval_vector(pi_wkl,n,-1);
   VECTOR_INT jobs_facts_copia=DIM_VECTOR_INT(f);
   setval_vector(jobs_facts_copia,f,0);
   MAT_INT seqxfab= DIM_MAT_INT(f,n);
   setval_matrix(seqxfab,f,n,-1);
   int new_dim, dim_pi;
   register int i,j,l,k;

   //TRABAJOS ORDENADOS POR SUMA DE pt
   for (j=0;j<n;j++)
   {
      for (i=0;i<m;i++)
      {
         sum_pt[j]+=pt[j][i];
      }
   }
   printf("Sum_pt\n");
   print_vector(sum_pt,n);

   sort_vector(sum_pt,omega,n,'D');
   printf("Omega\n");
   print_vector(omega,n);

   ASIGNA TRABAJO CON MAYOR pt A pi
   free_vector(sum_pt);
   *seq[0]=omega[0];
   dim_pi=1;
   seqxfab[0][0]=omega[0]; //Primer trabajo en la primera posicion de la primera fábrica
   copy_mat_int(seqxfab,*seq_fabs,f,n);
   printf("Matriz secuencia por fabricas\n");
   print_matrix(*seq_fabs,f,n);
```

```
new_dim=extract_vector(omega,n,0);  //Extrae el trabajo de omega secuenciado en pi
int aux=new_dim;
printf("Secuencia omega tras insertar el primer trabajo en pi_WKL\n");
print_vector(omega,new_dim);


int CIT_seq, min_CIT, CIT_f,max_CIT, f_max, f_asignada;
VECTOR_INT best_pi_wkl=DIM_VECTOR_INT(n);
setval_vector(best_pi_wkl,n,-1);
VECTOR_INT seq_best_l=DIM_VECTOR_INT(n);
setval_vector(seq_best_l,n,-1);
VECTOR_INT secuencia=DIM_VECTOR_INT(n);
setval_vector(secuencia,n,-1);

while (new_dim>0)
{
   if(A_i==1)
   {
      printf("Entra en A1\n");
      insertion(*seq,&best_pi_wkl,0,omega[0],n);
      printf("Vector best_pi_wkl tras insertion\n");
      print_vector(best_pi_wkl,n);
      dim_pi++;
      printf("dim_pi=%d\n",dim_pi);
      seqxfab[0][0]=best_pi_wkl[0];
      jobs_facts_copia[0]=1;
          for (j=1;j<dim_pi;j++)
      {
         f_asignada=A_1(seqxfab,jobs_facts_copia,f,n,m,pt);
         printf("f_asignada=%d\n",f_asignada);
         seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=best_pi_wkl[j];
         jobs_facts_copia[f_asignada]++;

      }
      copy_mat_int(seqxfab,*seq_fabs,f,n);
      copy_vector(jobs_facts_copia,*jobs_facts,f);
      setval_matrix(seqxfab,f,n,-1);
      setval_vector(jobs_facts_copia,f,0);
```

```
    printf("Matriz seq_fab\n");
    print_matrix(*seq_fabs,f,n);
    min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
    for(l=1;l<dim_pi;l++)
    {
      insertion(*seq,&pi_wkl,l,omega[0],n);
      printf("Vector pi_wkl tras insertion\t");
      print_vector(pi_wkl,n);
      seqxfab[0][0]=pi_wkl[0];
      jobs_facts_copia[0]=1;
      for (j=1;j<dim_pi;j++)
      {
        f_asignada=A_1(seqxfab,jobs_facts_copia,f,n,m,pt);
        printf("f_asignada=%d\n",f_asignada);
        seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=pi_wkl[j];
        jobs_facts_copia[f_asignada]++;

      }
      printf("Matriz seqxfab\n");
      print_matrix(seqxfab,f,n);
      CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
      if (CIT_seq<min_CIT)
      {
        min_CIT=CIT_seq;
        copy_vector(pi_wkl,best_pi_wkl,n);
        copy_vector(jobs_facts_copia,*jobs_facts,f);
        copy_mat_int(seqxfab,*seq_fabs,f,n);
        printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

      }
      setval_matrix(seqxfab,f,n,-1);
      setval_vector(jobs_facts_copia,f,0);
    }
  }
  else if(A_i==2)
  {
    printf("Entra en A2\n");
    insertion(*seq,&best_pi_wkl,0,omega[0],n);
```

```
printf("Vector best_pi_wkl tras insertion\n");
print_vector(best_pi_wkl,n);
dim_pi++;
seqxfab[0][0]=best_pi_wkl[0];
jobs_facts_copia[0]=1;
for (j=1;j<dim_pi;j++)
{
   f_asignada=A_2(seqxfab,jobs_facts_copia,f,n,m,pt);
   printf("f_asignada=%d\n",f_asignada);
   seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=best_pi_wkl[j];
   jobs_facts_copia[f_asignada]++;

}
copy_mat_int(seqxfab,*seq_fabs,f,n);
copy_vector(jobs_facts_copia,*jobs_facts,f);
setval_matrix(seqxfab,f,n,-1);
setval_vector(jobs_facts_copia,f,0);
printf("Matriz seq_fab\n");
print_matrix(*seq_fabs,f,n);
min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
for(l=1;l<dim_pi;l++)
{
   insertion(*seq,&pi_wkl,l,omega[0],n);
   printf("Vector pi_wkl tras insertion\t");
            print_vector(pi_wkl,n);
   seqxfab[0][0]=pi_wkl[0];
   jobs_facts_copia[0]=1;
   for (j=1;j<dim_pi;j++)
   {
      f_asignada=A_2(seqxfab,jobs_facts_copia,f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=pi_wkl[j];
      jobs_facts_copia[f_asignada]++;

   }
   printf("Matriz seqxfab\n");
   print_matrix(seqxfab,f,n);
   CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
```

```
        if (CIT_seq<min_CIT)
        {
           min_CIT=CIT_seq;
           copy_vector(pi_wkl,best_pi_wkl,n);
           copy_vector(jobs_facts_copia,*jobs_facts,f);
           copy_mat_int(seqxfab,*seq_fabs,f,n);
           printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

        }
        setval_matrix(seqxfab,f,n,-1);
        setval_vector(jobs_facts_copia,f,0);
     }

  }
  else if(A_i==3)
  {
     printf("Entra en A3\n");
     insertion(*seq,&best_pi_wkl,0,omega[0],n);
     printf("Vector best_pi_wkl tras insertion\n");
     print_vector(best_pi_wkl,n);
     dim_pi++;   //2
     seqxfab[0][0]=best_pi_wkl[0];
     jobs_facts_copia[0]=1;
     for (j=1;j<dim_pi;j++)
     {
        f_asignada=A_3(&seq_best_l,seqxfab,jobs_facts_copia,best_pi_wkl[j],f,n,m,pt);
        printf("f_asignada=%d\n",f_asignada);
        print_vector(seq_best_l,n);
        pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
        jobs_facts_copia[f_asignada]++;
        printf("Matriz seqxfab\n");
        print_matrix(seqxfab,f,n);
     }
     copy_mat_int(seqxfab,*seq_fabs,f,n);
     copy_vector(jobs_facts_copia,*jobs_facts,f);
     setval_matrix(seqxfab,f,n,-1);
     setval_vector(jobs_facts_copia,f,0);
     printf("Matriz seq_fab\n");
```

```
    print_matrix(*seq_fabs,f,n);
    min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
    for(l=1;l<dim_pi;l++)
    {
       insertion(*seq,&pi_wkl,l,omega[0],n);
       printf("Vector pi_wkl tras insertion\t");
       print_vector(pi_wkl,n);
       seqxfab[0][0]=pi_wkl[0];
       jobs_facts_copia[0]=1;
       for (j=1;j<dim_pi;j++)
       {
          f_asignada=A_3(&seq_best_l,seqxfab,jobs_facts_copia,pi_wkl[j],f,n,m,pt);
          printf("f_asignada=%d\n",f_asignada);
          print_vector(seq_best_l,n);
          pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
          jobs_facts_copia[f_asignada]++;

       }
       printf("Matriz seqxfab\n");
       print_matrix(seqxfab,f,n);
       CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
       if (CIT_seq<min_CIT)
       {
          min_CIT=CIT_seq;
          copy_vector(pi_wkl,best_pi_wkl,n);
          copy_vector(jobs_facts_copia,*jobs_facts,f);
          copy_mat_int(seqxfab,*seq_fabs,f,n);
          printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

       }
       setval_matrix(seqxfab,f,n,-1);
       setval_vector(jobs_facts_copia,f,0);
    }

 }
else if (A_i==4)
{
    printf("Entra en A4\n");
```

```
insertion(*seq,&best_pi_wkl,0,omega[0],n);
printf("Vector best_pi_wkl tras insertion\n");
print_vector(best_pi_wkl,n);
dim_pi++;
seqxfab[0][0]=best_pi_wkl[0];
jobs_facts_copia[0]=1;
for (j=1;j<dim_pi;j++)
{
   f_asignada=A_4(&seq_best_l,seqxfab,jobs_facts_copia,best_pi_wkl[j],f,n,m,pt);
   printf("f_asignada=%d\n",f_asignada);
   print_vector(seq_best_l,n);
   pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
   jobs_facts_copia[f_asignada]++;
   printf("Matriz seqxfab\n");
   print_matrix(seqxfab,f,n);
}
copy_mat_int(seqxfab,*seq_fabs,f,n);
copy_vector(jobs_facts_copia,*jobs_facts,f);
setval_matrix(seqxfab,f,n,-1);
setval_vector(jobs_facts_copia,f,0);
printf("Matriz seq_fab\n");
print_matrix(*seq_fabs,f,n);
min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);

for(l=1;l<dim_pi;l++)
{
   insertion(*seq,&pi_wkl,l,omega[0],n);
   printf("Vector pi_wkl tras insertion\t");
   print_vector(pi_wkl,n);
   seqxfab[0][0]=pi_wkl[0];
   jobs_facts_copia[0]=1;
   for (j=1;j<dim_pi;j++)
   {
      f_asignada=A_4(&seq_best_l,seqxfab,jobs_facts_copia,pi_wkl[j],f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      print_vector(seq_best_l,n);
      pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
      jobs_facts_copia[f_asignada]++;
```

```
        }
        printf("Matriz seqxfab\n");
        print_matrix(seqxfab,f,n);
        CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
        if (CIT_seq<min_CIT)
        {
           min_CIT=CIT_seq;
           copy_vector(pi_wkl,best_pi_wkl,n);
           copy_vector(jobs_facts_copia,*jobs_facts,f);
           copy_mat_int(seqxfab,*seq_fabs,f,n);
           printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

        }
        setval_matrix(seqxfab,f,n,-1);
        setval_vector(jobs_facts_copia,f,0);

    }
}
else if (A_i==5)
{
    printf("Entra en A5\n");
    insertion(*seq,&best_pi_wkl,0,omega[0],n);
    printf("Vector best_pi_wkl tras insertion\n");
    print_vector(best_pi_wkl,n);
    dim_pi++;
    printf("dim_pi=%d\n",dim_pi);
    seqxfab[0][0]=best_pi_wkl[0];
    jobs_facts_copia[0]=1;
    for (j=1;j<dim_pi;j++)
    {
        f_max=0;
        copyrowImatrix(seqxfab,secuencia,0,n);
        max_CIT=CIT(secuencia,pt,jobs_facts_copia[0],m,n);
        for (k=1;k<f;k++)
        {
           copyrowImatrix(seqxfab,secuencia,k,n);
           CIT_f=CIT(secuencia,pt,jobs_facts_copia[k],m,n);
```

```
        if(max_CIT<CIT_f)
        {
            max_CIT=CIT_f;
            f_max=k;

        }
    }
    f_asignada=A_5(&seq_best_l,seqxfab,jobs_facts_copia,best_pi_wkl[j],f_max,f,n,m,pt);
    printf("f_asignada=%d\n",f_asignada);
    print_vector(seq_best_l,n);
    pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
    jobs_facts_copia[f_asignada]++;
    printf("Matriz seqxfab\n");
    print_matrix(seqxfab,f,n);
}
copy_mat_int(seqxfab,*seq_fabs,f,n);
copy_vector(jobs_facts_copia,*jobs_facts,f);
setval_matrix(seqxfab,f,n,-1);
setval_vector(jobs_facts_copia,f,0);
printf("Matriz seq_fab\n");
print_matrix(*seq_fabs,f,n);
min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
for(l=1;l<dim_pi;l++)
{
    insertion(*seq,&pi_wkl,l,omega[0],n);
    printf("Vector pi_wkl tras insertion\t");
    print_vector(pi_wkl,n);
    seqxfab[0][0]=pi_wkl[0];
    jobs_facts_copia[0]=1;
    for (j=1;j<dim_pi;j++)
    {
        f_max=0;
        copyrowImatrix(seqxfab,secuencia,0,n);
        max_CIT=CIT(secuencia,pt,jobs_facts_copia[0],m,n);
        for (k=1;k<f;k++)
        {
            copyrowImatrix(seqxfab,secuencia,k,n);
            CIT_f=CIT(secuencia,pt,jobs_facts_copia[k],m,n);
```

```
            if(max_CIT<CIT_f)
            {
               max_CIT=CIT_f;
               f_max=k;

            }
         }
         f_asignada=A_5(&seq_best_l,seqxfab,jobs_facts_copia,pi_wkl[j],f_max,f,n,m,pt);
         printf("f_asignada=%d\n",f_asignada);
         print_vector(seq_best_l,n);
         pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
         jobs_facts_copia[f_asignada]++;

      }
      printf("Matriz seqxfab\n");
      print_matrix(seqxfab,f,n);
      printf("Antes de entrar en sumCIT\n");
      CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
      if (CIT_seq<min_CIT)
      {
         min_CIT=CIT_seq;
         copy_vector(pi_wkl,best_pi_wkl,n);
         copy_vector(jobs_facts_copia,*jobs_facts,f);
         copy_mat_int(seqxfab,*seq_fabs,f,n);
         printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

      }
      setval_matrix(seqxfab,f,n,-1);
      setval_vector(jobs_facts_copia,f,0);

   }
}
else if(A_i==6)
{
   printf("Entra en A6\n");
   insertion(*seq,&best_pi_wkl,0,omega[0],n);
   printf("Vector best_pi_wkl tras insertion\n");
   print_vector(best_pi_wkl,n);
```

```
dim_pi++;
seqxfab[0][0]=best_pi_wkl[0];
jobs_facts_copia[0]=1;
for (j=1;j<dim_pi;j++)
{
   f_asignada=A_6(&seq_best_l,seqxfab,jobs_facts_copia,best_pi_wkl[j],f,n,m,pt);
   printf("f_asignada=%d\n",f_asignada);
   print_vector(seq_best_l,n);
   pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
   jobs_facts_copia[f_asignada]++;
   printf("Matriz seqxfab\n");
   print_matrix(seqxfab,f,n);
}
copy_mat_int(seqxfab,*seq_fabs,f,n);
copy_vector(jobs_facts_copia,*jobs_facts,f);
setval_matrix(seqxfab,f,n,-1);
setval_vector(jobs_facts_copia,f,0);
printf("Matriz seq_fab\n");
print_matrix(*seq_fabs,f,n);
min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);

for(l=1;l<dim_pi;l++)
{
   insertion(*seq,&pi_wkl,l,omega[0],n);
   printf("Vector pi_wkl tras insertion\t");
   print_vector(pi_wkl,n);
   seqxfab[0][0]=pi_wkl[0];
   jobs_facts_copia[0]=1;
   for (j=1;j<dim_pi;j++)
   {
      f_asignada=A_6(&seq_best_l,seqxfab,jobs_facts_copia,pi_wkl[j],f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      print_vector(seq_best_l,n);
      pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
      jobs_facts_copia[f_asignada]++;

   }
   printf("Matriz seqxfab\n");
```

```
            print_matrix(seqxfab,f,n);
            CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
            if (CIT_seq<min_CIT)
            {
               min_CIT=CIT_seq;
               copy_vector(pi_wkl,best_pi_wkl,n);
               copy_vector(jobs_facts_copia,*jobs_facts,f);
               copy_mat_int(seqxfab,*seq_fabs,f,n);
               printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

            }
            setval_matrix(seqxfab,f,n,-1);
            setval_vector(jobs_facts_copia,f,0);

         }
      }
```

//Tras insertar el trabajo en todas las posiciones de posibles de pi_wkl y saber cual es la fabrica que se asigna, se conoce la que genera menor CIT

```
      new_dim=extract_vector(omega,aux,0);
      aux=new_dim;
      printf("new_dim=aux, %d=%d\n",new_dim,aux);
      printf("Vector omega tras extraer ultimo trabajo secuenciado\t");
      print_vector(omega,new_dim);

      printf("Secuencia pi_wkl\n");
      print_vector(best_pi_wkl,n);
      copy_vector(best_pi_wkl,*seq,n);



   }
   free_vector(omega);
   free_vector(jobs_facts_copia);
   free_vector(pi_wkl);
   free_vector(best_pi_wkl);
   free_vector(seq_best_l);
   free_vector(secuencia);
```

```
   free_matrix(seqxfab,f);
}

void NEH_R2 (VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n,
int m, int f,int A_i)
{
   printf("\n******Calculo de la secuencia NEH,R2******\n");
   VECTOR_INT sum_pt= DIM_VECTOR_INT(n);
   setval_vector(sum_pt,n,0);
   VECTOR_INT omega=DIM_VECTOR_INT(n);
   setval_vector(omega,n,-1);
   VECTOR_INT pi=DIM_VECTOR_INT(n);
   setval_vector(pi,n,-1);
   VECTOR_INT jobs_facts_copia=DIM_VECTOR_INT(f);
   copy_vector(*jobs_facts,jobs_facts_copia,f);
   MAT_INT seqxfab= DIM_MAT_INT(f,n);
   setval_matrix(seqxfab,f,n,-1);
   int new_dim, f_asignada;
   register int i,j,k;

   //TRABAJOS ORDENADOS POR SUMA DE pt
   for (j=0;j<n;j++)
   {
      for (i=0;i<m;i++)
      {
         sum_pt[j]+=pt[j][i];
      }
   }
   printf("Matriz pt, Sum_pt\n");
   print_matrix(pt,n,m);
   print_vector(sum_pt,n);
   sort_vector(sum_pt,omega,n,'D');
   printf("Omega\n");
   print_vector(omega,n);
   free_vector(sum_pt);

   // ASIGNAR TRABAJO CON MAYOR pt A pi
   pi[0]=omega[0];
```

```
copy_vector(pi,*seq,n);
seqxfab[0][0]=omega[0];   //Primer trabajo en la primera posicion de la primera fábrica
copy_mat_int(seqxfab,*seq_fabs,f,n);
printf("Matriz secuencia por fabricas\n");
print_matrix(seqxfab,f,n);
jobs_facts_copia[0]++;
new_dim=extract_vector(omega,n,0);   //Extrae el trabajo de omega secuenciado en pi
int aux=new_dim;
printf("Secuencia omega tras insertar el primer trabajo en pi_WKL\n");
print_vector(omega,new_dim);


int max_CIT, CIT_f, f_max;
VECTOR_INT seq_best_l =DIM_VECTOR_INT(n);
setval_vector(seq_best_l,n,-1);
VECTOR_INT secuencia =DIM_VECTOR_INT(n);
setval_vector(secuencia,n,-1);

while (new_dim>0)
{
   if (A_i==1)
   {
      printf("Entra en A1\n");
      f_asignada=A_1(seqxfab,jobs_facts_copia,f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=omega[0];
      jobs_facts_copia[f_asignada]++;
      pi=matriz_a_vector(seqxfab,n,f);
   }
   else if (A_i==2)
   {
      printf("Entra en A2\n");
      f_asignada=A_2(seqxfab,jobs_facts_copia,f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=omega[0];
      jobs_facts_copia[f_asignada]++;
      pi=matriz_a_vector(seqxfab,n,f);
   }
```

```
else if (A_i==3)
{
   printf("Entra en A3\n");
   f_asignada=A_3(&seq_best_l,seqxfab,jobs_facts_copia,omega[0],f,n,m,pt);
   printf("f_asignada=%d, la seq de esta fab es\t",f_asignada);
   print_vector(seq_best_l,n);
   pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
   jobs_facts_copia[f_asignada]++;
   pi=matriz_a_vector(seqxfab,n,f);
}
else if (A_i==4)
{
   printf("Entra en A4\n");
   f_asignada=A_4(&seq_best_l,seqxfab,jobs_facts_copia,omega[0],f,n,m,pt);
   printf("f_asignada=%d, la seq de esta fab es\t",f_asignada);
   print_vector(seq_best_l,n);
   pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
   jobs_facts_copia[f_asignada]++;
   pi=matriz_a_vector(seqxfab,n,f);

}
else if (A_i==5)
{
   printf("Entra en A5\n");
   f_max=0;
   copyrowImatrix(seqxfab,secuencia,0,n);
   max_CIT=CIT(secuencia,pt,jobs_facts_copia[0],m,n);
   for (k=1;k<f;k++)
   {
      copyrowImatrix(seqxfab,secuencia,k,n);
      CIT_f=CIT(secuencia,pt,jobs_facts_copia[k],m,n);
      if(max_CIT<CIT_f)
      {
         max_CIT=CIT_f;
         f_max=k;

      }
   }
```

```
        printf("fmax=%d\n",f_max);
        f_asignada=A_5(&seq_best_l,seqxfab,jobs_facts_copia,omega[0],f_max,f,n,m,pt);
        printf("f_asignada=%d, la seq de esta fab es\t",f_asignada);
        print_vector(seq_best_l,n);
        pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
        jobs_facts_copia[f_asignada]++;
        pi=matriz_a_vector(seqxfab,n,f);
    }
    else if (A_i==6)
    {
        printf("Entra en A6\n");
        f_asignada=A_6(&seq_best_l,seqxfab,jobs_facts_copia,omega[0],f,n,m,pt);
        printf("f_asignada=%d, la seq de esta fab es\t",f_asignada);
        print_vector(seq_best_l,n);
        pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
        jobs_facts_copia[f_asignada]++;
        pi=matriz_a_vector(seqxfab,n,f);
    }

    //Tras insertar el trabajo en todas las posiciones de posibles de pi_wkl y saber cual es la fabrica que se asigna, se conoce la que genera menor CIT
    new_dim=extract_vector(omega,aux,0);
    aux=new_dim;
        printf("new_dim=aux, %d=%d\n",new_dim,aux);
    printf("Vector omega tras extraer ultimo trabajo secuenciado\t");
    print_vector(omega,new_dim);

    printf("Secuencia pi\n");
    print_vector(pi,n);
    copy_vector(pi,*seq,n);
    copy_vector(jobs_facts_copia,*jobs_facts,f);
    copy_mat_int(seqxfab,*seq_fabs, f, n);

}

free_vector(omega);
free_vector(pi);
free_vector(jobs_facts_copia);
```

```
   free_vector(seq_best_l);
   free_vector(secuencia);
   free_matrix(seqxfab,f);
}

int A_1 (MAT_INT seqxfabs, VECTOR_INT jobs_facts,int f, int n, int m, MAT_INT pt)
{
   int min_C_max, C_max,best_f;
   register int k,j;
   VECTOR_INT copia_fila_seqxfabs=DIM_VECTOR_INT(n);

   for (j=0;j<n;j++)
   {
      copia_fila_seqxfabs[j]=seqxfabs[0][j];
   }
   printf("Copia fila fab 0\n");
   print_vector(copia_fila_seqxfabs,n);
   min_C_max=makespan(copia_fila_seqxfabs,n,m,pt);
   printf("min makespan=%d\n",min_C_max);
   best_f=0;
   for (k=1;k<f;k++)
   {
      for (j=0;j<n;j++)
      {
         copia_fila_seqxfabs[j]=seqxfabs[k][j];
      }
      printf("Copia fila %d \n",k);
      print_vector(copia_fila_seqxfabs,n);
      C_max=makespan(copia_fila_seqxfabs,n,m,pt);
      printf("makespan=%d\n",C_max);
      if(C_max<min_C_max)
      {
         min_C_max=C_max;
         best_f=k;
         printf("Min makespan se ha actualizado(=%d), corresponde a la fabrica %d\n",min_C_max,k);
      }

   }
```

```
   free_vector(copia_fila_seqxfabs);
   printf("Sale de funcion A_1\n");

   return best_f;
}

int A_2 (MAT_INT seqxfabs, VECTOR_INT jobs_facts,int f, int n, int m, MAT_INT pt)
{
   printf("Entra en función A_2\n");
   VECTOR_INT copia_fila_seqxfabs=DIM_VECTOR_INT(n);
   int CIT_k, min_CIT, best_f;
   register int i,k;

   for (i=0;i<n;i++)
   {
      copia_fila_seqxfabs[i]=seqxfabs[0][i];
   }
   printf("Copia fila fab 0\n");
   print_vector(copia_fila_seqxfabs,n);
   min_CIT=CIT(copia_fila_seqxfabs,pt,jobs_facts[0],m,n);
   best_f=0;
   printf("El min_CIT se supone para la fab %d, igual a %d\n",best_f,min_CIT);
   for (k=1;k<f;k++)
   {
      for (i=0;i<n;i++)
      {
         copia_fila_seqxfabs[i]=seqxfabs[k][i];
      }
      printf("Copia fila fab %d\n",k);
      print_vector(copia_fila_seqxfabs,n);

      CIT_k=CIT(copia_fila_seqxfabs,pt,jobs_facts[k],m,n);
      printf("CIT(k=%d)=%d\n",k,CIT_k);
      if(CIT_k<min_CIT)
      {
         min_CIT=CIT_k;
         best_f=k;
```

```
            printf("La mejor fab es %d, cuyo CIT es %d\n", k, min_CIT);
        }

    }

    free_vector(copia_fila_seqxfabs);

    printf("Sale de funcion A_2\n");

    return best_f;
}

int A_3(VECTOR_INT *fab_insertion, MAT_INT seqxfab, VECTOR_INT jobs_facts, int job_to_insert, int f,
int n, int m, MAT_INT pt)
{
    printf("Entra en funcion A_3\n");
    VECTOR_INT copia_fila_fab= DIM_VECTOR_INT(n);
    VECTOR_INT dif_CIT = DIM_VECTOR_INT(f);
    setval_vector(dif_CIT,f,-1);
    VECTOR_INT pos_f = DIM_VECTOR_INT(f);
    setval_vector(pos_f,f,-1);

    register int j,k;
    int best_f=-1, best_pos, min_dif, min_CIT_f, CIT_f,CIT_ini;

    for(k=0;k<f;k++)
    {
        for (j=0;j<n;j++)
        {

            copia_fila_fab[j]=seqxfab[k][j];

        }
        printf("Copia seq fab %d, jobs_facts=%d\n", k, jobs_facts[k]);
        print_vector(copia_fila_fab,n);

        CIT_ini=CIT(copia_fila_fab, pt, jobs_facts[k],m,n);
        printf("min CIT de la fabrica %d se da antes de insertar el trabajo = %d\n", k, min_CIT_f);
```

```
    if(jobs_facts[k]<n)
    {
       pos_f[k]=0;
       insertion(copia_fila_fab,&(*fab_insertion),0,job_to_insert,n);
       printf("Seq fab %d tras insertion en 0\n",k);
       print_vector(*fab_insertion,n);
       jobs_facts[k]++;
       printf("jobs_facts[%d]=%d tras insertion\n",k, jobs_facts[k]);
       min_CIT_f=CIT(*fab_insertion, pt, jobs_facts[k],m,n);
       printf("El CIT de esta seq es =%d\n", CIT_f);
       print_vector(*fab_insertion,n);
       jobs_facts[k]--;
       for(j=1;j<=jobs_facts[k];j++)
       {
          insertion(copia_fila_fab,&(*fab_insertion),j,job_to_insert,n);
          printf("Seq fab %d tras insertion\n",k);
          print_vector(*fab_insertion,n);
          jobs_facts[k]++;
          printf("jobs_facts[%d]=%d tras insertion\n",k, jobs_facts[k]);
          CIT_f=CIT(*fab_insertion, pt, jobs_facts[k],m,n);
          printf("El CIT de esta seq es =%d\n", CIT_f);
          print_vector(*fab_insertion,n);
          jobs_facts[k]--;
          if (CIT_f<=min_CIT_f)
          {
             min_CIT_f=CIT_f;
             pos_f[k]=j;
             printf("Min CIT se ha actualizado, es %d y la mejor pos es %d",min_CIT_f,pos_f[k]);
          }
       }
    dif_CIT[k]=CIT_ini-min_CIT_f;
    }

}
printf("Vector dif_CIT\t");
print_vector(dif_CIT,f);
printf("\nVector pos_f\t");
```

```
print_vector(pos_f,f);

for(k=0;k<f;k++)
{
   if(best_f==-1 && pos_f[k]!=-1 )
   {
      best_f=k;
      best_pos=pos_f[k];
      min_dif=dif_CIT[k];
      k=f; //Para salir del bucle cuando entre por primera vez en if
      printf("Se inicializa con la mejor pos de fab %d, best_pos=%d\n", best_f, best_pos);
   }

}

for(k=best_f;k<f;k++)
{
   if(dif_CIT[k]<min_dif && pos_f[k]!=-1)
   {
      min_dif=dif_CIT[k];
      best_f=k;
      best_pos=pos_f[k];
      printf("Se ha actualizado min_dif=%d y best_f=%d, best_pos=%d\n", min_dif, best_f,best_pos);

   }
}
for (j=0;j<n;j++)
{
   copia_fila_fab[j]=seqxfab[best_f][j];
}
insertion(copia_fila_fab,&(*fab_insertion),best_pos,job_to_insert,n);
printf("Vector fab_insertion en funcion\t");
print_vector(*fab_insertion,n);

free_vector(copia_fila_fab);
free_vector(dif_CIT);
free_vector(pos_f);
```

```
   printf("Sale de funcion A_3\n");

   return best_f;
}

int A_4(VECTOR_INT *fab_insertion, MAT_INT seqxfab, VECTOR_INT jobs_facts, int job_to_insert, int f,
int n, int m, MAT_INT pt)
{
   printf("Entra en funcion A_4\n");
   VECTOR_INT copia_fila_fab= DIM_VECTOR_INT(n);
   VECTOR_INT dif_CIT_sum = DIM_VECTOR_INT(f);
   setval_vector(dif_CIT_sum,f,-1);
   VECTOR_INT pos_f = DIM_VECTOR_INT(f);
   setval_vector(pos_f,f,-1);
   MAT_INT copia_seqxfab=DIM_MAT_INT(f,n);

   register int j,k;
   int best_f=-1, best_pos, max_dif=-1, min_CIT_sum, CIT_sum,CIT_sum_ini;

   *fab_insertion=matriz_a_vector(seqxfab,n,f);
   printf("fab_insertion\n");
   print_vector(*fab_insertion,n);
   CIT_sum_ini=sumCIT(seqxfab,jobs_facts,pt,n,m,f);
   printf("min sumCIT de la fabrica se da antes de insertar el trabajo = %d\n", CIT_sum_ini);

   for(k=0;k<f;k++)
   {
      for (j=0;j<n;j++)
      {

         copia_fila_fab[j]=seqxfab[k][j];

      }
      printf("Copia seq fab %d, jobs_facts=%d\n", k, jobs_facts[k]);
      print_vector(copia_fila_fab,n);

      if(jobs_facts[k]<n)
      {
```

```
        pos_f[k]=0;
        copy_mat_int(seqxfab,copia_seqxfab,f,n);
        insertion(copia_fila_fab,&(*fab_insertion),0,job_to_insert,n);
        printf("Seq fab %d tras insertion\n",k);
        print_vector(*fab_insertion,n);
        jobs_facts[k]++;
        pasterowImatrix(copia_seqxfab,*fab_insertion,k,n);
        min_CIT_sum=sumCIT(copia_seqxfab, jobs_facts, pt,n,m,f);
        printf("sumCIT de la seq anterior es =%d\n", CIT_sum);
        jobs_facts[k]--;

        for(j=1;j<=jobs_facts[k];j++)
        {
           copy_mat_int(seqxfab,copia_seqxfab,f,n);
           insertion(copia_fila_fab,&(*fab_insertion),j,job_to_insert,n);
           printf("Seq fab %d tras insertion\n",k);
           print_vector(*fab_insertion,n);
           jobs_facts[k]++;
           pasterowImatrix(copia_seqxfab,*fab_insertion,k,n);
           CIT_sum=sumCIT(copia_seqxfab, jobs_facts, pt,n,m,f);
           printf("sumCIT de la seq anterior es =%d\n", CIT_sum);
           jobs_facts[k]--;
           printf("jobs_facts[%d]=%d\n",k, jobs_facts[k]);
           if (CIT_sum<=min_CIT_sum)
           {
              min_CIT_sum=CIT_sum;
              pos_f[k]=j;
              dif_CIT_sum[k]=CIT_sum_ini-min_CIT_sum;
              printf("Min CIT se ha actualizado, es %d y la mejor pos es %d para la fab %d\n
dif_CIT_sum[%d]=%d\n", min_CIT_sum, pos_f[k],k,k,dif_CIT_sum[k]);
           }
        }
      }
      dif_CIT_sum[k]=CIT_sum_ini-min_CIT_sum;

   }
   printf("Vector dif_CIT\t");
   print_vector(dif_CIT_sum,f);
```

```
printf("\nVector pos_f\t");
print_vector(pos_f,f);

for(k=0;k<f;k++)
{
   if(best_f==-1 && pos_f[k]!=-1 )
   {
      best_f=k;
      best_pos=pos_f[k];
      max_dif=dif_CIT_sum[k];
      k=f; //Para salir del bucle cuando entre por primera vez en if
      //printf("Se inicializa con la mejor pos de fab %d, best_pos=%d\n", best_f, best_pos);
   }

}

for(k=best_f;k<f;k++)
{
   if(dif_CIT_sum[k]>=max_dif && pos_f[k]!=-1)
   {
      max_dif=dif_CIT_sum[k];
      best_f=k;
      best_pos=pos_f[k];
      printf("Se ha actualizado max_dif=%d y best_f=%d, best_pos=%d\n", max_dif, best_f,best_pos);

   }
}

for (j=0;j<n;j++)
{
   copia_fila_fab[j]=seqxfab[best_f][j];
}
insertion(copia_fila_fab,&(*fab_insertion),best_pos,job_to_insert,n);
printf("Vector fab_insertion en funcion\t");
print_vector(*fab_insertion,n);

free_vector(copia_fila_fab);
free_vector(dif_CIT_sum);
```

```
   free_vector(pos_f);
   free_matrix(copia_seqxfab,f);

   printf("Sale de funcion A_4\n");

   return best_f;

}

int A_5(VECTOR_INT *fab_insertion, MAT_INT seqxfab, VECTOR_INT jobs_facts, int job_to_insert, int f_max, int f, int n, int m, MAT_INT pt)
{
   printf("Entra en funcion A_5\n");
   VECTOR_INT copia_fila_fab= DIM_VECTOR_INT(n);
   VECTOR_INT dif_CIT_sum = DIM_VECTOR_INT(f);
   setval_vector(dif_CIT_sum,f,-1);
   VECTOR_INT pos_f = DIM_VECTOR_INT(f);
   setval_vector(pos_f,f,-1);
   MAT_INT copia_seqxfab=DIM_MAT_INT(f,n);

   register int j,k;
   int best_f=-1, best_pos, max_dif, min_CIT_sum, CIT_sum,CIT_sum_ini;

   CIT_sum_ini=sumCIT(seqxfab, jobs_facts,pt,n,m,f);
   printf("min sumCIT se da antes de insertar el trabajo = %d\n", CIT_sum_ini);

   for(k=0;k<f;k++)
   {
      if (k!=f_max)
      {
         for (j=0;j<n;j++)
         {

         copia_fila_fab[j]=seqxfab[k][j];

         }
      printf("Copia seq fab %d, jobs_facts=%d\n", k, jobs_facts[k]);
      print_vector(copia_fila_fab,n);
```

```
if(jobs_facts[k]<n)
{
   copy_mat_int(seqxfab,copia_seqxfab,f,n);
   pos_f[k]=0;

   insertion(copia_fila_fab,&(*fab_insertion),0,job_to_insert,n);
   printf("Seq fab %d tras insertion\n",k);
   print_vector(*fab_insertion,n);
   jobs_facts[k]++;
   pasterowImatrix(copia_seqxfab,*fab_insertion,k,n);
   min_CIT_sum=sumCIT(copia_seqxfab, jobs_facts, pt,n,m,f);
   printf("min_CIT_sum=%d\n", min_CIT_sum);
   jobs_facts[k]--;

   for(j=1;j<=jobs_facts[k];j++)
   {
      insertion(copia_fila_fab,&(*fab_insertion),j,job_to_insert,n);
      printf("Seq fab %d tras insertion\n",k);
      print_vector(*fab_insertion,n);
      jobs_facts[k]++;
      pasterowImatrix(copia_seqxfab,*fab_insertion,k,n);
      CIT_sum=sumCIT(copia_seqxfab, jobs_facts, pt,n,m,f);
      printf("sumCIT=%d\n", CIT_sum);
      jobs_facts[k]--;
      if (CIT_sum<=min_CIT_sum) //Si tras la nueva insercción, mejora aún más
      {
         min_CIT_sum=CIT_sum;
         pos_f[k]=j;
         dif_CIT_sum[k]=CIT_sum_ini-min_CIT_sum;
         printf("Min CIT se ha actualizado, es %d y la mejor pos es %d para la fab %d\n
dif_CIT_sum[%d]=%d\n", min_CIT_sum, pos_f[k],k,k,dif_CIT_sum[k]);
      }
   }
}


}
```

```
   dif_CIT_sum[k]=CIT_sum_ini-min_CIT_sum;

}
printf("Vector dif_CIT\t");
print_vector(dif_CIT_sum,f);
printf("\nVector pos_f\t");
print_vector(pos_f,f);

for(k=0;k<f;k++)
{
   if(best_f==-1 && pos_f[k]!=-1 && k!=f_max  )
   {
      best_f=k;
      best_pos=pos_f[k];
      max_dif=dif_CIT_sum[k];
      k=f; //Para salir del bucle cuando entre por primera vez en if
      printf("Se inicializa con la mejor pos de fab 0, best_pos=%d\n", best_pos);
   }

}
for(k=best_f;k<f;k++)
{
   if(dif_CIT_sum[k]>=max_dif && pos_f[k]!=-1 && f_max!=k)
   {
      best_f=k;
      max_dif=dif_CIT_sum[k];
      best_pos=pos_f[k];
      printf("Se ha actualizado max_dif=%d y best_f=%d, best_pos=%d\n", max_dif, best_f,best_pos);

   }
}

for (j=0;j<n;j++)
{
   copia_fila_fab[j]=seqxfab[best_f][j];
}
insertion(copia_fila_fab,&(*fab_insertion),best_pos,job_to_insert,n);
printf("Vector fab_insertion en funcion\t");
```

```
    print_vector(*fab_insertion,n);

    free_vector(copia_fila_fab);
    free_vector(dif_CIT_sum);
    free_vector(pos_f);
    free_matrix(copia_seqxfab,f);

    printf("Sale de la funcion A_5\n");

    return best_f;
}

int A_6(VECTOR_INT *fab_insertion, MAT_INT seqxfab, VECTOR_INT jobs_facts, int job_to_insert, int f,
int n, int m, MAT_INT pt)
{
    printf("Entra en funcion A_6\n");
    VECTOR_INT vector_CIT= DIM_VECTOR_INT(f);
    VECTOR_INT copia_fila_fab= DIM_VECTOR_INT (n);
    VECTOR_INT F_2= DIM_VECTOR_INT(f);
    VECTOR_INT pos_f=DIM_VECTOR_INT(f);
    setval_vector(pos_f,f,-1);
    VECTOR_INT dif_CIT_sum=DIM_VECTOR_INT(f);
    setval_vector(dif_CIT_sum,f,-1);
    MAT_INT copia_seqxfab= DIM_MAT_INT(f,n);

    int aux=f,fact, min_CIT_sum, CIT_sum_ini, CIT_sum, best_f=-1, best_pos, max_dif;
    register int j,k;

    for(k=0;k<f;k++)
    {
        for(j=0;j<n;j++)
        {
            copia_fila_fab[j]=seqxfab[k][j];
        }
        printf("Copia fila fab\n");
        print_vector(copia_fila_fab,n);

        vector_CIT[k]=CIT(copia_fila_fab,pt,jobs_facts[k],m,n);
```

```
    printf("CIT de la fab %d =%d\n",k,vector_CIT[k]);
}

sort_vector(vector_CIT,F_2,f,'A');
printf("Vector F_2\t");
print_vector(F_2,f);
printf("min sumCIT se da antes de insertar el trabajo = %d\n", CIT_sum_ini);

for (k=0;k<(f/2);k++)
{
   aux=extract_vector(F_2,aux,(aux-1));
}
printf("vector F_2\t");
print_vector(F_2,aux);
CIT_sum_ini=sumCIT(seqxfab,jobs_facts,pt,n,m,f);

for(k=0;k<aux;k++)
{
   fact=F_2[k];
   printf("fact=%d\t aux=%d\t Vector F_2\t",fact,aux);
   print_vector(F_2,aux);
   for (j=0;j<n;j++)
   {
      copia_fila_fab[j]=seqxfab[fact][j];

   }
   printf("Copia seq fab %d\n", k);
   print_vector(copia_fila_fab,n);

   if(jobs_facts[fact]<n)
   {
      copy_mat_int(seqxfab,copia_seqxfab,f,n);
      pos_f[fact]=0;

      insertion(copia_fila_fab,&(*fab_insertion),0,job_to_insert,n);
      printf("Seq fab %d tras insertion\n",fact);
      print_vector(*fab_insertion,n);
      jobs_facts[fact]++;
```

```
        pasterowImatrix(copia_seqxfab,*fab_insertion,fact,n);
        min_CIT_sum=sumCIT(copia_seqxfab, jobs_facts, pt,n,m,f);
        printf("min_CIT_sum=%d\n", min_CIT_sum);
        jobs_facts[fact]--;
        dif_CIT_sum[fact]=CIT_sum_ini-min_CIT_sum;
        for(j=1;j<=jobs_facts[fact];j++)
        {
           insertion(copia_fila_fab,&(*fab_insertion),j,job_to_insert,n);
           printf("Seq fab %d tras insertion\n",fact);
           print_vector(*fab_insertion,n);
           jobs_facts[fact]++;
           pasterowImatrix(copia_seqxfab,*fab_insertion,fact,n);
           CIT_sum=sumCIT(copia_seqxfab, jobs_facts,pt,n,m,f);
           printf("sumCIT de esta seq es =%d\n", CIT_sum);
           jobs_facts[fact]--;
           if (CIT_sum<=min_CIT_sum)
           {
              min_CIT_sum=CIT_sum;
              pos_f[fact]=j;
              dif_CIT_sum[fact]=CIT_sum_ini-min_CIT_sum;
              printf("Min CIT se ha actualizado, es %d y la mejor pos es %d para la fab %d\n
dif_CIT_sum[%d]=%d\n", min_CIT_sum, pos_f[k],k,k,dif_CIT_sum[k]);
           }
        }
        }
   }
   printf("Vector dif_CIT\t");
   print_vector(dif_CIT_sum,f);
   printf("\nVector pos_f\t");
   print_vector(pos_f,f);

   for(k=0;k<f;k++)
   {
      if(best_f==-1 && pos_f[k]!=-1)
      {
         best_f=k;
         best_pos=pos_f[k];
         max_dif=dif_CIT_sum[k];
```

```
            k=f; //Para salir del bucle cuando entre por primera vez en if
            printf("Se inicializa con best_pos=%d\n", best_pos);
        }

    }

    for(k=best_f;k<f;k++)
    {
        if(dif_CIT_sum[k]>=max_dif && pos_f[k]!=-1 )
        {
            max_dif=dif_CIT_sum[k];
            best_f=k;
            best_pos=pos_f[k];
            printf("Se ha actualizado max_dif=%d y best_f=%d, best_pos=%d\n", max_dif, best_f,best_pos);

        }
    }

    for (j=0;j<n;j++)
    {
        copia_fila_fab[j]=seqxfab[best_f][j];
    }
    insertion(copia_fila_fab,&(*fab_insertion),best_pos,job_to_insert,n);

    free_vector(copia_fila_fab);
    free_vector(dif_CIT_sum);
    free_vector(pos_f);
    free_vector(vector_CIT);
    free_vector(F_2);
    free_matrix(copia_seqxfab,f);

    printf("Sale de la funcion A_6\n");

    return best_f;

}

int makespan (VECTOR_INT seq, int n, int m, MAT_INT pt)
```

```
{
   printf("Entra en funcion makespan\n");
   VECTOR_INT C =DIM_VECTOR_INT(n);
   setval_vector(C,n,0);
   MAT_INT S =DIM_MAT_INT(n,m);
   setval_matrix(S,n,m,-1);
   int makespan=0, ult_trabajo_seq;
   register int i,j,l;
   if (seq[0]!=-1)
   {
      ult_trabajo_seq=seq[0];
      for (l=0;l<n;l++)
      {
      printf("Entra en bucle for (l=%d)\t seq[%d]=%d\n",l,l,seq[l]);

      if(seq[l]!=-1)
      {
         ult_trabajo_seq=seq[l];

      S[seq[0]][0]=0; //El primer trabajo secuenciado no espera
      C[seq[0]]=pt[seq[0]][0];
      for (i=1;i<m;i++) //El primer trabajo pasa de una maquina a otra cuando ha sido finalizada
      {
         S[seq[0]][i]=C[seq[0]];
         C[seq[0]]+=pt[seq[0]][i];
         printf("S[%d][%d]=%d\tC[%d]=%d\n",seq[0],i,S[seq[0]][i],seq[0],C[seq[0]]);

      }

      for (j=1;j<n;j++) //En la primera maquina un trabajo se secuencia detrás de otro sin esperas
      {
         //printf("Bucle for primera maq: j=%d\n",j);
         if(seq[j]!=-1)
         {
            printf("Entra en if, seq[%d]=%d\n",j, seq[j]);
            S[seq[j]][0]=S[seq[j-1]][0]+pt[seq[j-1]][0];
         }
      }
```

```
    for (i=1;i<m;i++)
    {
      for (j=1;j<n;j++)
      {
        if(seq[j]!=-1)
        {
          ult_trabajo_seq=seq[j];
          if(S[seq[j]][i-1]+pt[seq[j]][i-1]<S[seq[j-1]][i]+pt[seq[j-1]][i] )
          {
            printf("\nEntra en if, seq[%d]=%d",j, seq[j]);
            S[seq[j]][i]=S[seq[j-1]][i]+pt[seq[j-1]][i]; //El trabajo anterior en la máquina tarda más que el mismo trabajo en la máquina anterior
          }
          else
          {
            printf("\nEntra en if, seq[%d]=%d",j, seq[j]);
            S[seq[j]][i]=S[seq[j]][i-1]+pt[seq[j]][i-1]; //El trabajo en la máquina anterior tarda más que el anterior en la máquina
          }
        C[seq[j]]=S[seq[j]][i]+pt[seq[j]][i];

        }

      }
    }
  }
  print_vector(C,n);
  }

  printf("Matriz S y Vector C\n");
  print_matrix(S,n,m);
  print_vector(C,n);

  makespan=C[ult_trabajo_seq];//El makespan es el completion time del ultimo trabajo secuencia en la ultima maquina
  printf("Valor del makespan=%d\n",makespan);

  }
```

```
    free_vector(C);
    free_matrix(S,n);

    printf("Sale de funcion makespan\n");



    return makespan;
}



void copy_mat_int( MAT_INT original, MAT_INT destino, int filas, int cols)
{
    register int i,j;

    for (i=0;i<filas;i++)
    {
        for(j=0;j<cols;j++)
        {
            destino[i][j]=original[i][j];
        }
    }
    printf("Sale función copy_may_int\n");
}

void DNEH_R1(VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n,
int m, int f,int A_i)
{
    printf("\n******Calculo de la secuencia DNEH_R1******\n");
    VECTOR_FLOAT IF0= DIM_VECTOR_FLOAT(n);
    VECTOR_FLOAT IT0= DIM_VECTOR_FLOAT(n);
    VECTOR_FLOAT C= DIM_VECTOR_FLOAT(n);
    VECTOR_INT landa= DIM_VECTOR_INT(n);
    VECTOR_INT jobs_facts_copia=DIM_VECTOR_INT(f);
    setval_vector(jobs_facts_copia,f,0);
    VECTOR_INT pi_wkl=DIM_VECTOR_INT(n);
    setval_vector(pi_wkl,n,-1);
    MAT_INT seqxfab=DIM_MAT_INT(f,n);
    setval_matrix(seqxfab,f,n,-1);
```

```
int f_asignada=-1;
register int i,j,k,l;

for (k=0;k<f;k++)
{
   for(j=0;j<n;j++)
   {
      IF0[j]=0;
      IT0[j]=0;
      C[j]=0;
      landa[j]=-1;
   }
}

int new_dim=n, dim_pi; // para extraer los primeros f trabajos
printf("Matriz pi\n");
print_matrix(pi,f,n);
for (j=0;j<n;j++)
{
   for (i=1;i<m;i++)
   {
      C[j]+=pt[j][i-1];
      IT0[j]+=(m*(C[j]))/(i);
   }
   C[j]+=pt[j][m-1]; // Makespan
   IF0[j]=((n/f)-2)*IT0[j]+ C[j];
}
printf("Vector IF0\t");
print_vector(IF0,n);
sort_vector(IF0,landa,n,'A'); // Vector de trabajos landa ordenado de manera creciente
printf("Vector landa\t");
print_vector(landa,n);
*seq[0]=landa[0];
dim_pi=1;
seqxfab[0][0]=landa[0]; //Primer trabajo en la primera posicion de la primera fábrica
copy_mat_int(seqxfab,*seq_fabs,f,n);
printf("Matriz secuencia por fabricas\n");
print_matrix(seqxfab,f,n);
```

```
new_dim=extract_vector(landa,n,0);//Extrae el trabajo de landa secuenciado en pi
int aux=new_dim;
printf("Secuencia landa tras insertar el primer trabajo en pi_wkl\n");
print_vector(landa,new_dim);


int CIT_seq, min_CIT, CIT_f, max_CIT, f_max;
VECTOR_INT best_pi_wkl=DIM_VECTOR_INT(n);
setval_vector(best_pi_wkl,n,-1);
VECTOR_INT seq_best_l=DIM_VECTOR_INT(n);
setval_vector(seq_best_l,n,-1);
VECTOR_INT secuencia=DIM_VECTOR_INT(n);
setval_vector(secuencia,n,-1);

while (new_dim>0)
{
  if(A_i==1)
  {
    printf("Entra en A1\n");
    insertion(*seq,&best_pi_wkl,0,landa[0],n);
    printf("Vector best_pi_wkl tras insertion\n");
    print_vector(best_pi_wkl,n);
    dim_pi++;
    seqxfab[0][0]=best_pi_wkl[0];
    jobs_facts_copia[0]=1;
    for (j=1;j<dim_pi;j++)
    {
      f_asignada=A_1(seqxfab,jobs_facts_copia,f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=best_pi_wkl[j];
      jobs_facts_copia[f_asignada]++;

    }
    copy_mat_int(seqxfab,*seq_fabs,f,n);
    copy_vector(jobs_facts_copia,*jobs_facts,f);
    setval_matrix(seqxfab,f,n,-1);
    setval_vector(jobs_facts_copia,f,0);
    printf("Matriz seq_fab\n");
```

```
    print_matrix(*seq_fabs,f,n);
    min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
    for(l=1;l<dim_pi;l++)
    {
      insertion(*seq,&pi_wkl,l,landa[0],n);
      printf("Vector pi_wkl tras insertion\t");
      print_vector(pi_wkl,n);
      seqxfab[0][0]=pi_wkl[0];
      jobs_facts_copia[0]=1;
      for (j=1;j<dim_pi;j++)
      {
        f_asignada=A_1(seqxfab,jobs_facts_copia,f,n,m,pt);
        printf("f_asignada=%d\n",f_asignada);
        seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=pi_wkl[j];
        jobs_facts_copia[f_asignada]++;

      }
      printf("Matriz seqxfab\n");
      print_matrix(seqxfab,f,n);

      CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
      if (CIT_seq<min_CIT)
      {
        min_CIT=CIT_seq;
        copy_vector(pi_wkl,best_pi_wkl,n);
        copy_vector(jobs_facts_copia,*jobs_facts,f);
        copy_mat_int(seqxfab,*seq_fabs,f,n);
        printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

      }
      setval_matrix(seqxfab,f,n,-1);
      setval_vector(jobs_facts_copia,f,0);

    }
  }
  else if(A_i==2)
  {
    printf("Entra en A2\n");
```

```
insertion(*seq,&best_pi_wkl,0,landa[0],n);
printf("Vector best_pi_wkl tras insertion\n");
print_vector(best_pi_wkl,n);
dim_pi++;
seqxfab[0][0]=best_pi_wkl[0];
jobs_facts_copia[0]=1;
for (j=1;j<dim_pi;j++)
{
   f_asignada=A_2(seqxfab,jobs_facts_copia,f,n,m,pt);
   printf("f_asignada=%d\n",f_asignada);
   seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=best_pi_wkl[j];
   jobs_facts_copia[f_asignada]++;

}
copy_mat_int(seqxfab,*seq_fabs,f,n);
copy_vector(jobs_facts_copia,*jobs_facts,f);
setval_matrix(seqxfab,f,n,-1);
setval_vector(jobs_facts_copia,f,0);
printf("Matriz seq_fab\n");
print_matrix(*seq_fabs,f,n);
min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
for(l=1;l<dim_pi;l++)
{
   insertion(*seq,&pi_wkl,l,landa[0],n);
   printf("Vector pi_wkl tras insertion\t");
   print_vector(pi_wkl,n);
   seqxfab[0][0]=pi_wkl[0];
   jobs_facts_copia[0]=1;
   for (j=1;j<dim_pi;j++)
   {
      f_asignada=A_2(seqxfab,jobs_facts_copia,f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=pi_wkl[j];
      jobs_facts_copia[f_asignada]++;

   }
   printf("Matriz seqxfab\n");
   print_matrix(seqxfab,f,n);
```

```
            CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
            if (CIT_seq<min_CIT)
            {
               min_CIT=CIT_seq;
               copy_vector(pi_wkl,best_pi_wkl,n);
               copy_vector(jobs_facts_copia,*jobs_facts,f);
               copy_mat_int(seqxfab,*seq_fabs,f,n);
               printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

            }
            setval_matrix(seqxfab,f,n,-1);
            setval_vector(jobs_facts_copia,f,0);

         }

      }
      else if(A_i==3)
      {
         printf("Entra en A3\n");
         insertion(*seq,&best_pi_wkl,0,landa[0],n);
         printf("Vector best_pi_wkl tras insertion\n");
         print_vector(best_pi_wkl,n);
         dim_pi++;
         seqxfab[0][0]=best_pi_wkl[0];
         jobs_facts_copia[0]=1;
         for (j=1;j<dim_pi;j++)
         {
            f_asignada=A_3(&seq_best_l,seqxfab,jobs_facts_copia,best_pi_wkl[j],f,n,m,pt);
            printf("f_asignada=%d\n",f_asignada);
            pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
            jobs_facts_copia[f_asignada]++;
            printf("Matriz seqxfab\n");
            print_matrix(seqxfab,f,n);
         }
         copy_mat_int(seqxfab,*seq_fabs,f,n);
         copy_vector(jobs_facts_copia,*jobs_facts,f);
         setval_matrix(seqxfab,f,n,-1);
         setval_vector(jobs_facts_copia,f,0);
```

```
    min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
    for(l=1;l<dim_pi;l++)
    {
       insertion(*seq,&pi_wkl,l,landa[0],n);
       printf("Vector pi_wkl tras insertion\t");
       print_vector(pi_wkl,n);
       seqxfab[0][0]=pi_wkl[0];
       jobs_facts_copia[0]=1;
       for (j=1;j<dim_pi;j++)
       {
          f_asignada=A_3(&seq_best_l,seqxfab,jobs_facts_copia,pi_wkl[j],f,n,m,pt);
          printf("f_asignada=%d\n",f_asignada);
          pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
          jobs_facts_copia[f_asignada]++;
                printf("Matriz seqxfab\n");
          print_matrix(seqxfab,f,n);

       }
       CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
       if (CIT_seq<min_CIT)
       {
          min_CIT=CIT_seq;
          copy_vector(pi_wkl,best_pi_wkl,n);
          copy_vector(jobs_facts_copia,*jobs_facts,f);
          copy_mat_int(seqxfab,*seq_fabs,f,n);
          printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

       }
       setval_matrix(seqxfab,f,n,-1);
       setval_vector(jobs_facts_copia,f,0);
    }
 }
else if (A_i==4)
{
    printf("Entra en A4\n");
    insertion(*seq,&best_pi_wkl,0,landa[0],n);
    printf("Vector best_pi_wkl tras insertion\n");
    print_vector(best_pi_wkl,n);
```

```
dim_pi++;
seqxfab[0][0]=best_pi_wkl[0];
jobs_facts_copia[0]=1;
for (j=1;j<dim_pi;j++)
{
   f_asignada=A_4(&seq_best_l,seqxfab,jobs_facts_copia,best_pi_wkl[j],f,n,m,pt);
   printf("f_asignada=%d\n",f_asignada);
   pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
   jobs_facts_copia[f_asignada]++;
   printf("Matriz seqxfab\n");
   print_matrix(seqxfab,f,n);
}
copy_mat_int(seqxfab,*seq_fabs,f,n);
copy_vector(jobs_facts_copia,*jobs_facts,f);
setval_matrix(seqxfab,f,n,-1);
setval_vector(jobs_facts_copia,f,0);
min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
for(l=1;l<dim_pi;l++)
{
   insertion(*seq,&pi_wkl,l,landa[0],n);
   printf("Vector pi_wkl tras insertion\t");
   print_vector(pi_wkl,n);
   seqxfab[0][0]=pi_wkl[0];
   jobs_facts_copia[0]=1;
   for (j=1;j<dim_pi;j++)
   {
      f_asignada=A_4(&seq_best_l,seqxfab,jobs_facts_copia,pi_wkl[j],f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
      jobs_facts_copia[f_asignada]++;
            printf("Matriz seqxfab\n");
      print_matrix(seqxfab,f,n);

   }

   CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
   if (CIT_seq<min_CIT)
   {
```

```
            min_CIT=CIT_seq;
            copy_vector(pi_wkl,best_pi_wkl,n);
            copy_vector(jobs_facts_copia,*jobs_facts,f);
            copy_mat_int(seqxfab,*seq_fabs,f,n);
            printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

         }
         setval_matrix(seqxfab,f,n,-1);
         setval_vector(jobs_facts_copia,f,0);

      }
   }
   else if (A_i==5)
   {
      printf("Entra en A5\n");
      insertion(*seq,&best_pi_wkl,0,landa[0],n);
      printf("Vector best_pi_wkl tras insertion\n");
      print_vector(best_pi_wkl,n);
      dim_pi++;
      seqxfab[0][0]=best_pi_wkl[0];
      jobs_facts_copia[0]=1;
      for (j=1;j<dim_pi;j++)
      {
         f_max=0;
         copyrowImatrix(seqxfab,secuencia,0,n);
         max_CIT=CIT(secuencia,pt,jobs_facts_copia[0],m,n);
         for (k=1;k<f;k++)
         {
            copyrowImatrix(seqxfab,secuencia,k,n);
            CIT_f=CIT(secuencia,pt,jobs_facts_copia[k],m,n);
            if(max_CIT<CIT_f)
            {
               max_CIT=CIT_f;
               f_max=k;

            }
         }
         f_asignada=A_5(&seq_best_l,seqxfab,jobs_facts_copia,best_pi_wkl[j],f_max,f,n,m,pt);
```

```
    printf("f_asignada=%d\n",f_asignada);
    pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
    jobs_facts_copia[f_asignada]++;
    printf("Matriz seqxfab\n");
    print_matrix(seqxfab,f,n);
  }
  copy_mat_int(seqxfab,*seq_fabs,f,n);
  copy_vector(jobs_facts_copia,*jobs_facts,f);
  setval_matrix(seqxfab,f,n,-1);
  setval_vector(jobs_facts_copia,f,0);
  min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
  for(l=1;l<dim_pi;l++)
  {
    insertion(*seq,&pi_wkl,l,landa[0],n);
    printf("Vector pi_wkl tras insertion\t");
    print_vector(pi_wkl,n);
    seqxfab[0][0]=pi_wkl[0];
    jobs_facts_copia[0]=1;
    for (j=1;j<dim_pi;j++)
    {
      f_max=0;
      copyrowImatrix(seqxfab,secuencia,0,n);
      max_CIT=CIT(secuencia,pt,jobs_facts_copia[0],m,n);
      for (k=1;k<f;k++)
      {
        copyrowImatrix(seqxfab,secuencia,k,n);
        CIT_f=CIT(secuencia,pt,jobs_facts_copia[k],m,n);
        if(max_CIT<CIT_f)
        {
          max_CIT=CIT_f;
          f_max=k;

        }
      }
      f_asignada=A_5(&seq_best_l,seqxfab,jobs_facts_copia,pi_wkl[j],f_max,f,n,m,pt);
      printf("f_asignada=%d\n",f_asignada);
      pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
      jobs_facts_copia[f_asignada]++;
```

```
                printf("Matriz seqxfab\n");
            print_matrix(seqxfab,f,n);

        }
        CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
        if (CIT_seq<min_CIT)
        {
            min_CIT=CIT_seq;
            copy_vector(pi_wkl,best_pi_wkl,n);
            copy_vector(jobs_facts_copia,*jobs_facts,f);
            copy_mat_int(seqxfab,*seq_fabs,f,n);
            printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

        }
        setval_matrix(seqxfab,f,n,-1);
        setval_vector(jobs_facts_copia,f,0);

    }
}
else if(A_i==6)
{
    printf("Entra en A6\n");
    insertion(*seq,&best_pi_wkl,0,landa[0],n);
    printf("Vector best_pi_wkl tras insertion\n");
    print_vector(best_pi_wkl,n);
    dim_pi++;
    seqxfab[0][0]=best_pi_wkl[0];
    jobs_facts_copia[0]=1;
    for (j=1;j<dim_pi;j++)
    {
        f_asignada=A_6(&seq_best_l,seqxfab,jobs_facts_copia,best_pi_wkl[j],f,n,m,pt);
        printf("f_asignada=%d\n",f_asignada);
        pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
        jobs_facts_copia[f_asignada]++;
        //printf("Matriz seqxfab\n");
        //print_matrix(seqxfab,f,n);
    }
    copy_mat_int(seqxfab,*seq_fabs,f,n);
```

```
    copy_vector(jobs_facts_copia,*jobs_facts,f);
    setval_matrix(seqxfab,f,n,-1);
    setval_vector(jobs_facts_copia,f,0);
    min_CIT=sumCIT(*seq_fabs,*jobs_facts,pt,n,m,f);
    for(l=1;l<dim_pi;l++)
    {
       insertion(*seq,&pi_wkl,l,landa[0],n);
       printf("Vector pi_wkl tras insertion\t");
       print_vector(pi_wkl,n);
       seqxfab[0][0]=pi_wkl[0];
       jobs_facts_copia[0]=1;
       for (j=1;j<dim_pi;j++)
       {
          f_asignada=A_6(&seq_best_l,seqxfab,jobs_facts_copia,pi_wkl[j],f,n,m,pt);
          printf("f_asignada=%d\n",f_asignada);
          pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
          jobs_facts_copia[f_asignada]++;
                printf("Matriz seqxfab\n");
          print_matrix(seqxfab,f,n);

       }
       CIT_seq=sumCIT(seqxfab,jobs_facts_copia,pt,n,m,f);
       if (CIT_seq<min_CIT)
       {
          min_CIT=CIT_seq;
          copy_vector(pi_wkl,best_pi_wkl,n);
          copy_vector(jobs_facts_copia,*jobs_facts,f);
          copy_mat_int(seqxfab,*seq_fabs,f,n);
          printf("Se ha actualizado el min CIT cuyo valor es %d\n",min_CIT);

       }
       setval_matrix(seqxfab,f,n,-1);
       setval_vector(jobs_facts_copia,f,0);

    }
}
```

```
    //Tras insertar el trabajo en todas las posiciones de posibles de pi_wkl y saber cual es la fabrica que se asigna, se conoce la que genera menor CIT

    new_dim=extract_vector(landa,aux,0);
    aux=new_dim;
    printf("Vector omega tras extraer ultimo trabajo secuenciado\t");
    print_vector(omega,new_dim);

    printf("Secuencia pi_wkl\n");
    print_vector(best_pi_wkl,n);
    copy_vector(best_pi_wkl,*seq,n);


  }
  free_vector(IF0);
  free_vector(IT0);
  free_vector(landa);
  free_vector(C)
  free_vector(jobs_facts_copia);
  free_vector(pi_wkl);
  free_vector(best_pi_wkl);
  free_vector(seq_best_l);
  free_vector(secuencia);
  free_matrix(seqxfab,f);
}

void DNEH_R2(VECTOR_INT *seq, VECTOR_INT *jobs_facts, MAT_INT *seq_fabs, MAT_INT pt, int n, int m, int f,int A_i)
{
  printf("\n******Calculo de la secuencia DNEH_R2******\n");
  VECTOR_FLOAT IF0= DIM_VECTOR_FLOAT(n);
  VECTOR_FLOAT IT0= DIM_VECTOR_FLOAT(n);
  VECTOR_FLOAT C= DIM_VECTOR_FLOAT(n);
  VECTOR_INT landa= DIM_VECTOR_INT(n);
  VECTOR_INT jobs_facts_copia=DIM_VECTOR_INT(f);
  setval_vector(jobs_facts_copia,f,0);
  MAT_INT seqxfab=DIM_MAT_INT(f,n);
  setval_matrix(seqxfab,f,n,-1);
```

```
VECTOR_INT pi= DIM_VECTOR_INT(n);
setval_vector(pi,n,-1);
int f_asignada=-1;
register int i,j,k;

for (k=0;k<f;k++)
{
   for(j=0;j<n;j++)
   {

      IF0[j]=0;
      IT0[j]=0;
      C[j]=0;
      landa[j]=-1;
   }
}

int new_dim=n; // para extraer los primeros f trabajos
for (j=0;j<n;j++)
{
   for (i=1;i<m;i++)
   {
      C[j]+=pt[j][i-1];
      IT0[j]+=(m*(C[j]))/(i);
   }
   C[j]+=pt[j][m-1]; // Makespan
   IF0[j]=((n/f)-2)*IT0[j]+ C[j];
}
printf("Vector IF0\t");
print_vector(IF0,n);
sort_vector(IF0,landa,n,'A'); // Vector de trabajos landa ordenado de manera creciente
printf("Vector landa\t");
print_vector(landa,n);
*seq[0]=landa[0];
seqxfab[0][0]=landa[0]; //Primer trabajo en la primera posicion de la primera fábrica
copy_mat_int(seqxfab,*seq_fabs,f,n);
printf("Matriz secuencia por fabricas\n");
print_matrix(seqxfab,f,n);
```

```
new_dim=extract_vector(landa,n,0);//Extrae el trabajo de landa secuenciado en pi
int aux=new_dim;
printf("Secuencia landa tras insertar el primer trabajo en pi_wkl\n");
print_vector(landa,new_dim);


int max_CIT, CIT_f, f_max;
VECTOR_INT best_pi_wkl=DIM_VECTOR_INT(n);
setval_vector(best_pi_wkl,n,-1);
VECTOR_INT seq_best_l=DIM_VECTOR_INT(n);
setval_vector(seq_best_l,n,-1);
VECTOR_INT secuencia=DIM_VECTOR_INT(n);
setval_vector(secuencia,n,-1);

while (new_dim>0)
{
  if (A_i==1)
  {
    printf("Entra en A1\n");
    f_asignada=A_1(seqxfab,jobs_facts_copia,f,n,m,pt);
    printf("f_asignada=%d\n",f_asignada);
    seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=landa[0];
    jobs_facts_copia[f_asignada]++;
    pi=matriz_a_vector(seqxfab,n,f);
  }
  else if (A_i==2)
  {
    printf("Entra en A2\n");
    f_asignada=A_2(seqxfab,jobs_facts_copia,f,n,m,pt);
    printf("f_asignada=%d\n",f_asignada);
    seqxfab[f_asignada][jobs_facts_copia[f_asignada]]=landa[0];
    jobs_facts_copia[f_asignada]++;
    pi=matriz_a_vector(seqxfab,n,f);
    printf("Vector pi tras insertion\n");
    print_vector(pi, n);
  }
  else if (A_i==3)
  {
```

```
    printf("Entra en A3\n");
    f_asignada=A_3(&seq_best_l,seqxfab,jobs_facts_copia,landa[0],f,n,m,pt);
    printf("f_asignada=%d, la seq de esta fab es\t",f_asignada);
    pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
    jobs_facts_copia[f_asignada]++;
    pi=matriz_a_vector(seqxfab,n,f);
}
else if (A_i==4)
{
    printf("Entra en A4\n");
    f_asignada=A_4(&seq_best_l,seqxfab,jobs_facts_copia,landa[0],f,n,m,pt);
    printf("f_asignada=%d, la seq de esta fab es\t",f_asignada);
    pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
    jobs_facts_copia[f_asignada]++;
    pi=matriz_a_vector(seqxfab,n,f);

}
else if (A_i==5)
{
    printf("Entra en A5\n");
    f_max=0;
    copyrowImatrix(seqxfab,secuencia,0,n);
    max_CIT=CIT(secuencia,pt,jobs_facts_copia[0],m,n);
    for (k=1;k<f;k++)
    {
        copyrowImatrix(seqxfab,secuencia,k,n);
        CIT_f=CIT(secuencia,pt,jobs_facts_copia[k],m,n);
        if(max_CIT<CIT_f)
        {
            max_CIT=CIT_f;
            f_max=k;

        }
    }
    printf("fmax=%d\n",f_max);
    f_asignada=A_5(&seq_best_l,seqxfab,jobs_facts_copia,landa[0],f_max,f,n,m,pt);
    printf("f_asignada=%d, la seq de esta fab es\t",f_asignada);
    print_vector(seq_best_l,n);
```

```
            pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
            jobs_facts_copia[f_asignada]++;
            pi=matriz_a_vector(seqxfab,n,f);
        }
        else if (A_i==6)
        {
            printf("Entra en A6\n");
            f_asignada=A_6(&seq_best_l,seqxfab,jobs_facts_copia,landa[0],f,n,m,pt);
            printf("f_asignada=%d, la seq de esta fab es\t",f_asignada);
            print_vector(seq_best_l,n);
            pasterowImatrix(seqxfab,seq_best_l,f_asignada,n);
            jobs_facts_copia[f_asignada]++;
            pi=matriz_a_vector(seqxfab,n,f);
        }

        //Tras insertar el trabajo en todas las posiciones de posibles de pi_wkl y saber cual es la fabrica que se asigna, se conoce la que genera menor CIT
        new_dim=extract_vector(landa,aux,0);
        aux=new_dim;
        printf("Vector omega tras extraer ultimo trabajo secuenciado\t");
        print_vector(omega,new_dim);

        printf("Secuencia pi\n");
        print_vector(pi,n);
        copy_vector(pi,*seq,n);
        copy_vector(jobs_facts_copia,*jobs_facts,f);
        copy_mat_int(seqxfab,*seq_fabs, f, n);
    }
    free_vector(IF0);
    free_vector(IT0);
    free_vector(landa);
    free_vector(C)
    free_vector(pi);
    free_vector(jobs_facts_copia);
    free_vector(seq_best_l);
    free_vector(secuencia)
    free_matrix(seqxfab,f);
}
```