Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Modelos de programación lineal entera para el problema de programación de la producción de ensamblado en dos etapas

Autor: David Cáceres Aycart
Tutora: Carla Talens Fayos

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

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Modelos de programación lineal entera para el problema de programación de la producción de ensamblado en dos etapas

Autor:
David Cáceres Aycart

Tutora:
Carla Talens Fayos

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

Trabajo Fin de Grado: Modelos de programación lineal entera para el problema de programación de la producción de ensamblado en dos etapas

Autor: David Cáceres Aycart

Tutora: Carla Talens Fayos

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

Agradecimientos

En primer lugar, me gustaría agradecerle a mi tutora, Carla Talens, todo el apoyo que me ha brindado en este proyecto y su dedicación. Sin duda, su ímpetu y sus conocimientos en este área han sido el impulso que necesitaba en los momentos en los que más me costaba avanzar. Gracias, de verdad.

Por otro lado, me gustaría agradecer a toda mi familia su apoyo incondicional. A mis padres, que siempre están ahí apoyando mis decisiones, o debatiéndolas, si ven que me desvío de mi objetivo, que no es otro que ser feliz. A mis hermanos, que cada uno me aporta algo diferente, y a la vez, necesario en mi vida. Marta, calma. Javier, sobriedad. Inés, racionalismo. A mi abuela Angelines que, pese a tener 10 hijos y 35 nietos, se preocupa por cada uno de nosotros y estaba deseando que llegara este momento de mi vida. A mi abuelo Pepe, que en paz descanse, que fue mi principal referente de inteligencia, pasión y cordura, el verdadero impulsor de mi carrera como ingeniero. Muchas gracias a mis abuelos José Luis y Pilarín, por todo el cariño que me dan cada vez que les visito (y cuando me llama la abuela sin querer, también), y los valores que me han enseñado: el trabajo, la austeridad y la honestidad. Gracias a todos mis tíos y a mis 49 primos hermanos, a los que debo mucho mi forma de ser, aunque nombrarlos sería una locura. Y gracias a ti, Ángela, porque hace ya un año y medio desde que comenzamos este viaje juntos, y cada día tengo más claro que quiero acabarlo contigo. Tu amor y tu apoyo son imprescindibles en mi vida.

También, agradecer a todas las personas que están ahí en el día a día conmigo, ya sea practicando cualquier deporte (mis compañeros de fútbol, mi '*team*' de paddle surf…), o apoyando y aconsejándome en la vida: Pablo Kóvacs, Membrive, Herrada… y otros más que, aun no siendo mencionados, pueden darse por aludidos.

En último lugar, pero no menos importante, querría agradecer a todos aquellos profesores que han pasado por mi vida (tanto del colegio como de la universidad) que, gracias a su pasión por la enseñanza y a su gran trato humano, me han inspirado para convertirme en un mejor profesional y persona.

David Cáceres Aycart

Sevilla, 2021

Resumen

En este Trabajo de Fin de Grado se aborda un problema de programación de la producción de ensamblado en dos etapas, con *m* máquinas en la primera etapa y una máquina en la segunda, y cuyo objetivo es minimizar la suma de las tardanzas y adelantos de todos los trabajos.

Para el estudio de este problema, se van a desarrollar cuatro modelos MILP diferentes adaptados de artículos con problemas relacionados. Dichos modelos van a ser programados en lenguaje C# y ejecutados con el programa *Gurobi*.

Se pretende analizar la eficiencia de cada uno de estos modelos a través de la resolución de 630 instancias, tomando como indicadores de eficiencia el número de óptimos, el tiempo de cómputo medio y el ARPD en cada uno de los modelos.

Abstract

This Final Degree Project addresses a two-stage assembly production scheduling problem, with *M* machines in the first stage and one machine in the second stage, and whose objective is to minimize the sum of the tardiness and earliness of all the jobs.

For the study of this problem, four different MILP models adapted from articles with related problems are going to be developed. These models are going to be programmed in C# language and executed with the Gurobi program.

It is intended to analyse the efficiency of each of these models through the resolution of 630 instances, taking as efficiency indicators the number of optimal, the average computational CPU time and the ARPD in each of the models.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Nunca andes por el camino trazado, pues te conducirá a donde otros ya fueron.

- Graham Bell -

Hoy en día, la competencia que hay entre empresas del sector industrial es cada vez mayor, debido a un avance exponencial de la tecnología y al enraizamiento de la filosofía de la mejora continua en el sector empresarial. Esta dura competencia provoca que las empresas se encuentren en una búsqueda constante de la mejora del producto sin que ésta suponga demasiado sobrecoste, la reducción de costes en el sistema productivo o la minimización de fallos en la producción. En definitiva, se busca la optimización de los recursos de la empresa.

La organización industrial es la rama de la ingeniería que se encarga de optimizar dichos recursos a través de la logística, y la planificación y programación de la producción, tomando decisiones estratégicas, tácticas y operacionales que aprovechen al máximo todos los recursos de los que dispone una empresa.

En el ámbito operacional, la programación de la producción es una función muy importante en cualquier industria, ya que es la que organiza la actividad de producción, atendiendo a los objetivos de la empresa y teniendo en cuenta los recursos y capacidad disponibles.

Dicha función conlleva una gran complejidad debido a que la cantidad de posibles soluciones aumenta de forma considerable a medida que aumenta el tamaño del problema. En el caso de un problema de flujo regular, sin ninguna otra restricción, con *n* trabajos y *m* máquinas, el número de posibles soluciones sería $(n!)^m$. Por tanto, para la resolución de este tipo de problemas se requiere de herramientas potentes que ayuden a optimizar la ordenación de trabajos y la ocupación de recursos.

1.1 Objeto del Trabajo

Este TFG tiene como objetivo analizar un problema de programación de la producción tipo taller de flujo de ensamblaje en dos etapas, con *m* máquinas en la primera etapa y una máquina en la segunda, mediante cuatro modelos diferentes de programación lineal entera (MILP). Utilizando el programa *Gurobi* como herramienta para optimizar, y *Visual Studio* como herramienta para programar en lenguaje C#, se va a estudiar la eficiencia de cada uno de ellos y se va a determinar cuál de ellos es el más eficiente para la función objetivo $\min\left(\sum_j (T_j + E_j)\right)$.

El motivo por el que se ha elegido esta función objetivo es porque no ha sido estudiada hasta ahora para este tipo de problema. Por ejemplo, en C.N. Potts et al. (1995) se estudia el tiempo total de finalización de todos los trabajos y, en Al-Anzi and Allahverdi (2006), el sumatorio de tiempos de finalización de cada trabajo. Por tanto, surge la necesidad de analizar dicha función objetivo para el problema en cuestión, mediante su resolución de manera exacta con distintos modelos matemáticos.

Los objetivos del Trabajo son los siguientes:

1. Analizar el problema que se aborda y problemas relacionados en la literatura.
2. Analizar diferentes modelos matemáticos y adaptarlos al problema en cuestión.
3. Programar los distintos modelos en *Visual Studio*, en lenguaje C#, para la resolución del problema.
4. Validación de los modelos resolviendo pequeños problemas.
5. Programar en *Visual Studio* la generación de instancias, incluyendo todos los datos necesarios, para ser resueltas con los modelos.
6. Resolución de las instancias para su posterior análisis.
7. Comparar resultados de cada modelo y calcular su eficiencia.

La contribución principal de este TFG es que se analizan diferentes modelos MILP para el problema de ensamblado en dos etapas, dado que no se ha hecho hasta el momento, y se espera comprobar si modelos creados para problemas relacionados pueden resolver correctamente, y en un tiempo razonable, el problema que se estudia.

1.2 Sumario

Este documento se estructura de la siguiente manera:

En el Capítulo 2, se van a explicar diferentes conceptos relacionados con la programación de la producción, que son de importancia para comprender el problema que se plantea en este trabajo y, posteriormente, se va a describir dicho problema determinando sus características, el entorno de trabajo y su objetivo, y resolviendo un pequeño ejemplo utilizando el diagrama de Gantt.

A continuación, en el Capítulo 3, se va a exponer la notación que va a ser utilizada en los modelos MILP adaptados y se van a desarrollar cada uno de estos modelos explicando la procedencia de estos (si han sido adaptados de artículos ya existentes en la literatura o modificados de modelos ya adaptados en este trabajo), sus respectivas restricciones, y la resolución del ejemplo aportado en el Capítulo 2 a través de dichos modelos.

Más adelante, en el Capítulo 4, se va a detallar la forma de obtención de las instancias que van a ser resueltas por los modelos y los indicadores que van a ser utilizados para determinar la eficiencia de estos. Por último, se van a explicar los resultados obtenidos, comparando los valores obtenidos por los modelos en cada uno de los 3 indicadores de eficiencia.

Finalmente, en el Capítulo 5, se van a desarrollar las conclusiones obtenidas del análisis de resultados realizado en el capítulo anterior, añadiendo un pequeño subapartado donde se van a proponer futuras líneas de investigación relacionadas con este proyecto.

En el Anexo se incluyen los códigos desarrollados para la programación de los modelos adaptados en el Capítulo 3 y la generación de instancias.

2 DESCRIPCIÓN DEL PROBLEMA

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge.

- Stephen Hawking -

En la primera parte de este apartado, se explican los conceptos más importantes sobre la programación de la producción para familiarizar al lector con los términos que se van a utilizar durante este TFG, separados en tres subapartados: los conceptos relacionados con las máquinas, los relacionados con los trabajos, y los asociados a la función objetivo. Posteriormente, se describe el problema estudiado en este trabajo y se plantea un ejemplo para facilitar su entendimiento.

2.1 Notación genérica de la descripción

Un problema de programación de la producción se define por los tres aspectos que determinan el tipo de problema que es: $\alpha|\beta|\gamma$ (Framiñan et al., 2014).

2.1.1 Características de máquinas (α)

En primer lugar, α representa el entorno en el que se desarrolla el problema, el número de máquinas, las características de éstas y el número de estaciones o etapas de las que se compone (en caso de que el entorno sea híbrido). Toda esta información queda recogida en α mediante una notación específica utilizada en los problemas de programación, que varía en función del modelo:

1. **Una sola máquina (*Single machine*):** Se representa como $\alpha = 1$. Se trata del modelo más simple que hay. Todos los trabajos *n* tienen que procesarse en la única máquina que existe.
2. **Máquinas paralelas (*Parallel machines)*:** En este entorno, las máquinas pueden funcionar de manera simultánea. Según la relación entre las máquinas, pueden ser de varios tipos:
    1. **Máquinas paralelas idénticas:** Se representa como $\alpha = Pm$, donde *m* es el número de máquinas del modelo. Todas las máquinas son idénticas, por lo que el tiempo de proceso de los trabajos no depende de éstas (p_j).
    2. **Máquinas paralelas uniformes:** Se representa como $\alpha = Qm$. Cada máquina tiene una velocidad distinta. Si el tiempo de proceso del trabajo j es p_j y la velocidad de la máquina *i* es v_i, ésta tardará en procesar dicho trabajo $p_{ij} = \frac{p_j}{v_i}$.
    3. **Máquinas paralelas no relacionadas:** Se representa como $\alpha = Rm$. Cada trabajo *j* tiene un tiempo de proceso para cada máquina *i* (p_{ij}).

3. **Entornos tipo taller:** Las máquinas se disponen en serie y todos los trabajos pasan por todas las máquinas, donde cada una realiza una operación diferente. Las rutas de los trabajos determinan el tipo de taller del modelo:

    1. **Taller de flujo (*Flow shop*):** Se representa como $\alpha = Fm$. Todos los trabajos tienen la misma ruta, es decir, que todos pasan por todas las máquinas en el mismo orden.

    2. **Taller de trabajos (*Job shop*):** Se representa como $\alpha = Jm$. En este tipo de taller, cada trabajo sigue una ruta diferente, la cual es conocida desde el inicio (R_j).

    3. **Taller abierto (*Open shop*)**: Se representa como $\alpha = Om$. No hay una ruta predeterminada para ninguno de los trabajos.

4. **Entornos híbridos (*Hybrids layout*):** Se trata de un taller formado por etapas en el cual, en vez de máquinas, cada etapa está compuesta de una máquina o de varias máquinas paralelas. La notación depende de los entornos hibridados. Primero, se indica el tipo de taller que se ha hibridado con una "H" delante ($\alpha = HFm, HJm, HOm$) y luego, para cada etapa, se indica qué tipo de entorno implica. Por ejemplo, para un entorno híbrido de taller de flujo de dos etapas, donde el primero está compuesto de máquinas paralelas y el segundo de una sola máquina, la notación sería: $\alpha = HF2, Pm, 1$.

2.1.2 Características de los trabajos (β)

En segundo lugar, se encuentra el campo β, que representa las características de los trabajos o restricciones. En este apartado, se incluyen los datos inicialmente dados que van a condicionar la programación de los trabajos, como pueden ser los tiempos de preparación o los tiempos de llegada. Algunas de las restricciones más comunes son:

1. **Interrupción (*preemption*):** Los trabajos u operaciones pueden interrumpirse durante el procesamiento en una máquina, ya sea por indisponibilidad de esta o porque sea conveniente para mejorar la eficiencia. Las interrupciones pueden ser: *Non-resumable* ($\beta = pmtn - non - resumable$), si la tarea comienza desde cero al ser interrumpida; *Semi-resumable* ($\beta = pmtn - semi - resumable$), si se reinicia la tarea parcialmente; o *Resumable* ($\beta = pmtn - resumable$), si la tarea se reinicia por donde se había quedado.

2. **Fecha de llegada (*release dates*):** Fecha en la que se encuentran disponibles los trabajos ($\beta = r_j$). Si no existe esta restricción, todos los trabajos tienen $r_j = 0$.

3. **Fecha de entrega (*due dates*):** Fecha en la que el trabajo debe estar terminado. Este dato no aparece en β salvo en dos excepciones:

    1. **Plazos de entrega (*Deadlines*):** Cuando los trabajos tienen la obligación de ser terminados antes o en su fecha ($\beta = \bar{d}_j$).

    2. **Fechas de entrega comunes:** Cuando todos los trabajos tienen la misma fecha de entrega ($\beta = d_j = d$).

4. **Tiempos de preparación (*setup times*):** Es el tiempo de preparación de una máquina para empezar a procesar un determinado trabajo. Estos tiempos pueden ser: independientes de la secuencia, es decir, que hay un tiempo de preparación por cada trabajo *j* procesado en máquina *i* ($\beta = s_{ij}$); o dependientes, donde habría un tiempo de preparación por cada trabajo *k* procesado en máquina *i* y precedido por el trabajo *j* ($\beta = s_{ijk}$). A su vez, estos tiempos pueden ser anticipatorios o no anticipatorios en función de si se pueden iniciar antes o después de estar disponible el trabajo a procesar.

5. **Precedencia:** No se puede empezar a procesar un trabajo hasta que no hayan acabado todos los trabajos que lo preceden ($\beta = prec$). Los datos de esta restricción se representan mediante un diagrama de precedencia, como el siguiente:

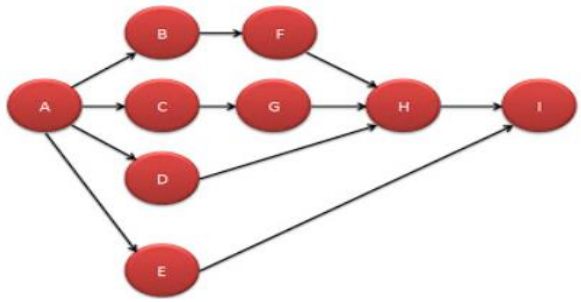

Figura 1. Ejemplo de diagrama de precedencia (Estacio, 2011)

6. **Restricciones de los entornos taller:**

    1. **Permutación:** Todos los trabajos tienen la misma ruta ($\beta = prmu$).
    2. **Sin tiempo ocioso:** Una vez que la máquina empieza a procesar el primer trabajo, no puede parar ($\beta = no - idle$).
    3. **Sin espera:** Una vez que un trabajo ha empezado a procesarse, no puede esperar entre máquinas o etapas ($\beta = no - wait$).
    4. **Almacén (*Buffer*):** El espacio de almacenamiento de trabajos de una máquina está limitado ($\beta = buffer = b_i$). Si $buffer = 0$, entonces sería un $\beta = no - wait$.

Existen otros tipos de restricciones como tiempos de proceso cambiantes debido al deterioro (*deteriorating-effect*) o el aprendizaje *(learning-effect*), o la lotificación (*batching*), mediante la cual varios trabajos se pueden procesar a la vez, ya sea en paralelo o en serie.

2.1.3 Características de la función objetivo (γ)

En último lugar, se encuentra γ, que representa la función objetivo del problema. En este campo, se mide la calidad de las soluciones obtenidas en función del objetivo que se quiera alcanzar. Siempre se procurará minimizar dicha función. Existen diferentes medidas para determinar el rendimiento de una solución (*performance measures*):

1. $\boldsymbol{C_j}$**:** Tiempo de finalización de un trabajo *j*.
2. $\boldsymbol{L_j}$**:** Retraso del trabajo *j* respecto a su fecha de entrega. $L_j = C_j - d_j$

3. $\boldsymbol{T_j}$**:** Tardanza del trabajo j. Si el trabajo termina antes de su fecha de entrega, su tardanza será cero. $T_j = \max(0, L_j) = \max(0, C_j - d_j)$
4. $\boldsymbol{E_j}$**:** Adelanto del trabajo j. Si el trabajo termina después de su fecha de entrega, su adelanto es cero. $E_j = \max(0, d_j - C_j)$
5. $\boldsymbol{U_j}$**:** Es una variable binaria. Determina si el trabajo j ha tenido tardanza o no. $\begin{cases} U_j = 1 \;\; si\; T_j > 0 \; (C_j > d_j) \\ U_j = 0 \;\; en\; caso\; contrario \end{cases}$
6. $\boldsymbol{F_j}$**:** Tiempo de flujo del trabajo j, es decir, tiempo que pasa desde que el trabajo está disponible hasta que termina de procesarse ($F_j = C_j - r_j$).

Hay dos maneras en las que estas medidas pueden aparecer en la función objetivo: en forma de sumatorio (***sum-form***: $f = \sum_{j=1}^{n} g(C_j)$); o hallando el máximo (***max-form***: $f = \max_{1 \le j \le n} g(C_j)$).

En cuanto a la función objetivo en sí, se pueden encontrar de dos tipos:

1. **Relacionadas con la fecha de entrega:** Dependen del parámetro d_j:

Max-form	***Sum-form***
$\max_{1 \le j \le n} L_j$	$\sum_{j=1}^{n} L_j$
$\max_{1 \le j \le n} T_j$	$\sum_{j=1}^{n} T_j$
$\max_{1 \le j \le n} E_j$	$\sum_{j=1}^{n} E_j$
	$\sum_{j=1}^{n} U_j$

2. **No relacionadas con la fecha de entrega:** No dependen del parámetro d_j:

Max-form	***Sum-form***
$C_{max} = \max_{1 \le j \le n} C_j$	$\sum_{j=1}^{n} C_j$
$\max_{1 \le j \le n} F_j$	$\sum_{j=1}^{n} F_j$

Además, se le puede otorgar un peso a cada trabajo y añadir el parámetro $\boldsymbol{w_j}$ a la función objetivo como coeficiente de las medidas anteriormente mencionadas. Por ejemplo:

$\max_{1 \le j \le n} w_j F_j$	$\sum_{j=1}^{n} w_j C_j$
$\max_{1 \le j \le n} w_j E_j$	$\sum_{j=1}^{n} w_j T_j$

2.2 Descripción del problema

El problema que se estudia en este TFG es un problema de ensamblado en dos etapas, con *m* máquinas en la primera etapa y una máquina en la segunda, donde se procesan *n* trabajos con *m+1* operaciones cada uno. El objetivo del problema es ajustar los tiempos de finalización de los trabajos lo máximo posible a sus fechas de entrega, minimizando así la tardanza y el adelanto de cada uno de ellos. Por lo tanto, la función objetivo del problema queda de la siguiente manera:

$$\sum_{j=1}^{n} (T_j + E_j)$$

En cuanto a la complejidad de este problema, se puede afirmar que la complejidad es NP-Hard ya que el problema de *Custormer Order or Open shop Concurrent* ($DPm||\sum_{j=1}^{n}(T_j + E_j)$) es también NP-Hard (Roemer, 2006).

Para entender mejor el problema que se plantea, se plantea un pequeño ejemplo, llamado "ejemplo", para resolver el problema de ensamblado de dos etapas, con 2 máquinas en la primera etapa y una en la segunda, y con 6 trabajos a procesar. Los datos del ejemplo son los siguientes:

p_{ij}	**1**	**2**	**3**	**4**	**5**	**6**
M 1.1	4	2	1	6	8	5
M 1.2	2	5	3	5	1	9
M 2	6	8	5	3	7	4

Tabla 1. Tiempos de proceso de la instancia "ejemplo"

TRABAJOS	**1**	**2**	**3**	**4**	**5**	**6**
d_j	16	25	9	23	30	37

Tabla 2. Fechas de entrega de la instancia "ejemplo"

En primer lugar, para obtener un programa con una solución factible, se utiliza la regla de despacho EDD (*Earliest Due Date*), que consiste en procesar los trabajos en orden creciente en función de sus fechas de entrega (d_j). De esta manera, la secuencia en la que se procesan los trabajos es:

$$S = (3, 1, 4, 2, 5, 6)$$

Siguiendo dicha secuencia, el programa queda representado en el diagrama de Gantt de la siguiente manera:

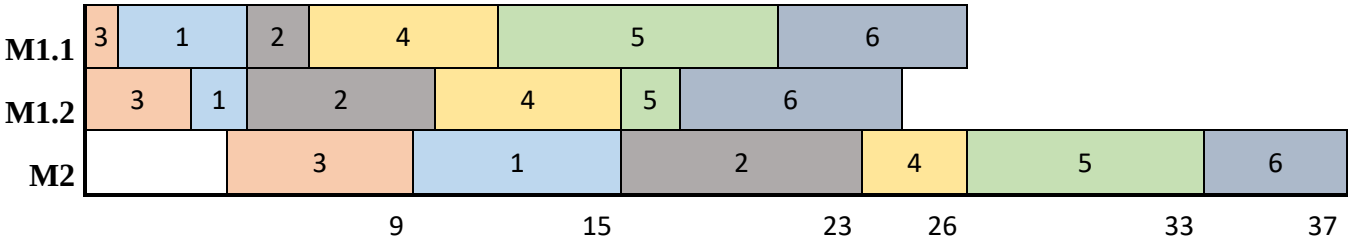

Figura 2. Diagrama de Gantt de la solución propuesta con la instancia "ejemplo"

En último lugar, se realizan los cálculos para hallar el valor de la función objetivo en dicha solución:

$$T_j = \max(0, C_j - d_j) \qquad E_j = \max(0, d_j - C_j)$$

$$T_1 = \max(0, 14 - 16) = 0 \qquad E_1 = \max(0, 16 - 14) = 2$$

$$T_2 = \max(0, 25 - 25) = 0 \qquad E_2 = \max(0, 25 - 25) = 0$$

$$T_3 = \max(0, 8 - 9) = 0 \qquad E_3 = \max(0, 9 - 8) = 1$$

$$T_4 = \max(0, 17 - 23) = 0 \qquad E_4 = \max(0, 23 - 17) = 6$$

$$T_5 = \max(0, 32 - 30) = 2 \qquad E_5 = \max(0, 30 - 32) = 0$$

$$T_6 = \max(0, 36 - 37) = 0 \qquad E_6 = \max(0, 37 - 36) = 1$$

$$\sum_{j=1}^{n} (T_j + E_j) = (0 + 0 + 0 + 0 + 2 + 0) + (2 + 0 + 1 + 6 + 0 + 1) = 12$$

3 MODELIZACIÓN

The mind that opens to a new idea never returns to its original size.

- Albert Einstein -

En este apartado, se van a presentar los cuatro modelos adaptados de problemas relacionados con el problema estudiado en este TFG. En primer lugar, se va a explicar la notación utilizada en los distintos modelos y, posteriormente, se detallará la fuente de la que se han estudiado y adaptado los tres primeros modelos, sus restricciones y la adaptación realizada. En el caso del cuarto modelo, al no ser obtenido directamente de un artículo, detalla que ha sido planteado a partir del modelo 1 para ver qué ocurre si se obliga a que no haya tiempo ocioso en ninguna de las etapas. Además, se va a resolver con cada uno de estos modelos el ejemplo presentado en la sección anterior y se muestran las soluciones obtenidas con cada modelo.

Estos cuatro modelos son MILP (*Mixed-Integer Linear Programming*), es decir, modelos matemáticos de resolución exacta, cuyas variables pueden ser continuas o enteras, y que aparecen de forma lineal, por lo que son separables tanto en las restricciones como en la función objetivo (Caballero, 2011).

3.1 Notación

A continuación, se expone la notación que ha sido utilizada por los modelos: índices y parámetros comunes, y todas las variables de decisión que han sido manejadas en los cuatro modelos, habiéndole asignado el mismo nombre a todas aquellas variables que, aun perteneciendo a diferentes modelos, representaran lo mismo.

Parámetros:

d_j	Fecha de entrega del trabajo *j*
p_{ij}	Tiempo de proceso del trabajo *j* en la máquina *i* de la etapa 1
a_j	Tiempo de proceso del trabajo *j* en la máquina de la etapa 2
N	Número de trabajos a procesar
M	Número de máquinas en la etapa 1

Índices:

i	Índice de máquinas en la primera etapa ($1 < i < M$)
j	Índice de trabajos ($1 < j < N$)
r	Índice de posición de un trabajo en la secuencia ($1 < r < N$)

Variables de decisión:

$T_{[r]}$ Tardanza del trabajo en la posición *r*

$E_{[r]}$ Adelanto del trabajo en la posición *r*

$FT_{[r]i}$ Tiempo de finalización del trabajo en la posición *r* en la máquina *i* de la etapa 1

$C_{[r]}$ Tiempo de finalización del trabajo en la posición *r* en la etapa 1

$CT_{[r]}$ Tiempo de finalización del trabajo en la posición *r* en la etapa 2

$S_{[r]i}$ Tiempo de inicio del trabajo en la posición *r* en la máquina *i* de la etapa 1

$S2_{[r]}$ Tiempo de inicio del trabajo en la posición *r* en la máquina de la etapa 2

$PT_{[r]i}$ Tiempo de proceso del trabajo en la posición *r* en la máquina *i* de la etapa 1

$AT_{[r]}$ Tiempo de proceso del trabajo en la posición *r* en la etapa 2

$ZZ_{[r]}$ Tiempo en el que el trabajo en la posición *r* puede empezar el proceso de la etapa 2

$DD_{[r]}$ Fecha de entrega del trabajo en la posición *r*.

x_{jr} $\begin{cases} 1 \text{ si el trabajo } j \text{ está en la posición } r \\ 0 \text{ en caso contrario} \end{cases}$

$Y_{[r]}$ $\begin{cases} 1 \text{ si el trabajo en la posición } r \text{ se procesa justo después que el trabajo anterior} \\ 0 \quad \text{en caso contrario} \end{cases}$

$h_{[r]}$ $\begin{cases} 1 \text{ si el trabajo en la posición } r \text{ se retrasa } (T_{[r]} > 0) \\ 0 \quad \text{si no} \end{cases}$

$hh_{[r]}$ $\begin{cases} 1 \text{ si el trabajo en la posición } r \text{ se adelanta } (E_{[r]} > 0) \\ 0 \quad \text{si no} \end{cases}$

Estas variables de decisión no se van a utilizar en todos los modelos, solamente en aquellos que así lo requieran para sus restricciones.

3.2 Modelos

3.2.1 Modelo 1

Este modelo ha sido obtenido a partir del artículo Lee and Bang (2016), donde se estudia el mismo problema con dos máquinas en la primera etapa y una en la segunda para minimizar la tardanza total. En primer lugar, se han adaptado las restricciones de la primera etapa, ya que se consideran más de dos máquinas en el problema que se ha estudiado. Por otro lado, se ha añadido la restricción del adelanto de trabajos, esto es, la diferencia entre la fecha de entrega de un trabajo y el tiempo de finalización de este (al contrario que en la restricción de la tardanza), ya que la función objetivo que se estudia incluye esta variable. A continuación, se van a desarrollar las restricciones adaptadas del modelo.

Restricciones Modelo 1:

$$T_{[r]} \geq CT_{[r]} - \sum_{j=1}^{n} d_j * x_{jr} \quad \forall r = 1 \dots n \tag{1}$$

$$E_{[r]} \geq \sum_{j=1}^{n} d_j * x_{jr} - CT_{[r]} \quad \forall r = 1 \dots n \tag{2}$$

$$CT_{[r]} \geq S2_{[r]} + \sum_{j=1}^{n} a_j * x_{jr} \quad \forall r = 1 \dots n \tag{3}$$

$$S2_{[r]} \geq S_{[r]i} + \sum_{j=1}^{n} p_{ij} * x_{jr} \quad \forall r = 1 \dots n, i = 1 \dots m \tag{4}$$

$$S2_{[r+1]} \geq CT_{[r]} \quad \forall r = 1 \dots n - 1 \tag{5}$$

$$S_{[r+1]i} \geq S_{[r]i} + \sum_{j=1}^{n} p_{ij} * x_{jr} \quad \forall r = 1 \dots n - 1, \qquad i = 1 \dots m \tag{6}$$

$$S_{[1]i} \geq 0 \quad \forall i = 1 \dots m \tag{7}$$

$$\sum_{r=1}^{n} x_{jr} = 1 \quad \forall j = 1 \dots n \tag{8}$$

$$\sum_{j=1}^{n} x_{jr} = 1 \quad \forall r = 1 \dots n \tag{9}$$

$$T_{[r]}, E_{[r]}, CT_{[r]}, S_{[r]i}, S2_{[r]} \geq 0 \quad \forall r, i \tag{10}$$

$$x_{jr} \in \{0,1\} \quad \forall j, r \tag{11}$$

Las restricciones (1) y (2) calculan la tardanza y el adelanto de los trabajos, respectivamente. La restricción (3) calcula el tiempo de finalización de los trabajos en la segunda etapa, mientras que las (4) y (5) se aseguran de que los trabajos empiecen la segunda etapa una vez hayan terminado todas las operaciones en la primera. En la restricción (6), se asegura que cada trabajo empiece una operación en la etapa 1 después de que haya acabado el anterior y, en la (7), que el tiempo de inicio del primer trabajo sea igual o superior a cero. La restricción (8) garantiza que cada trabajo esté sólo en una posición, mientras que la (9) asegura que en cada posición sólo haya un trabajo. Finalmente, las restricciones (10) y (11) determinan el dominio de las variables de decisión enteras y binarias, respectivamente.

Ejemplo Modelo 1:

Tomando de ejemplo la instancia utilizada en el apartado 2.2, se va a resolver el problema mediante este modelo con el fin de diferenciar la forma que tienen de resolver un mismo problema cada uno de los modelos adaptados. Aun así, debido a la sencillez de la instancia, algunos de los modelos van a resolver de forma idéntica la instancia. La secuencia resultante es la siguiente:

$$S = (3, 1, 2, 4, 5, 6)$$

Dada esta secuencia, se calculan los valores de las variables del modelo, omitiendo la variable x_{jr}, que determina el orden de la secuencia en la que se encuentra el óptimo (o la solución factible más próxima, en su defecto).

	$S_{[r]1}$	$S_{[r]2}$	$S2_{[r]}$	$CT_{[r]}$	$E_{[r]}$	$T_{[r]}$
$r = 1$	0	0	4	9	0	0
$r = 2$	1	3	9	15	1	0
$r = 3$	5	5	15	23	2	0
$r = 4$	7	10	23	26	0	3
$r = 5$	13	15	26	33	0	3
$r = 6$	28	24	33	37	0	0

Tabla 3.Valores de las variables en el modelo 1 con la instancia "ejemplo"

Obtenidos los valores de $E_{[r]}$ y $T_{[r]}$, se puede calcular el valor de la función objetivo:

$$\sum_{r=0}^{n} E_{[r]} + \sum_{r=0}^{n} T_{[r]} = 9$$

El diagrama de Gantt resultante es el siguiente:

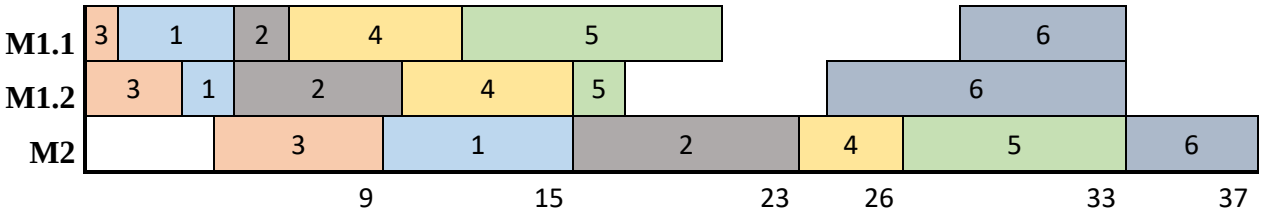

Figura 3. Diagrama de Gantt de la instancia "ejemplo" en el modelo 1

El trabajo en la posición $r = 1$, que es el trabajo 3, empieza a ser procesado en la 2ª etapa en $t = 4$. Es decir, la máquina de la segunda etapa está ociosa, esperando a procesar el trabajo 3, durante una unidad de tiempo, ya que su inicio en la 2ª etapa se retrasa una unidad de tiempo para entregar el trabajo lo más cerca de su fecha de entrega y así minimizar el adelanto. Por otro lado, se observa que el trabajo 6 empieza a ser procesado en la primera etapa de manera que ambas operaciones finalicen cuando la máquina en la segunda etapa esté disponible. Esto es sólo una holgura del problema, ya que el tiempo de inicio de las operaciones del trabajo 6 en la 1ª etapa no influye en la calidad de la solución (siempre que no finalicen después de quedar libre la máquina de la 2ª etapa).

3.2.2 Modelo 2

Este modelo ha sido obtenido a partir del artículo Navaei et al. (2013). En dicho artículo, se presenta un problema de ensamblado de *m* máquinas en la primera etapa y *kk* máquinas en la segunda, y una función objetivo que minimiza el tiempo de finalización del último trabajo procesado (*makespan*). Como se puede observar, este modelo presenta 14 restricciones, tres más que en el modelo 1. Esto se debe a que, en este modelo, se utilizan variables que indican los tiempos de proceso en función de la posición de los trabajos ($PT_{[r]i}$ y $AT_{[r]}$), mientras que en el modelo 1 se utilizan los datos aportados por las instancias, que son los tiempos de proceso en función del orden en que son dados. Por otro lado, en el modelo 2 se tiene en cuenta el tiempo de finalización en la primera etapa, y en el modelo 1, no. Para adaptar al problema el modelo propuesto en Navaei et al. (2013), se han eliminado las restricciones relacionadas con el tiempo de finalización del último trabajo y se ha suprimido el índice de máquinas en la segunda etapa, simplificando así varias de las restricciones. Por último, se han añadido las variables de adelanto y tardanza de los trabajos, así como sus respectivas restricciones, y se ha eliminado la variable binaria que determinaba a qué máquina de la segunda etapa estaba asignado cada trabajo. A continuación, se exponen las restricciones resultantes de la adaptación del modelo.

Restricciones Modelo 2:

$$ZZ_{[r]} \geq C_{[r]} \quad \forall r = 1 \dots n \tag{12}$$

$$ZZ_{[r]} \geq CT_{[r-1]} \quad \forall r = 2 \dots n \tag{13}$$

$$CT_1 \geq C_1 + AT_1 \tag{14}$$

$$CT_{[r]} \geq ZZ_{[r]} + AT_{[r]} \quad \forall r = 2 \dots n \tag{15}$$

$$\sum_{r=1}^{n} x_{j[r]} = 1 \quad \forall j = 1 \dots n \tag{16}$$

$$\sum_{j=1}^{n} x_{j[r]} = 1 \quad \forall r = 1 \dots n \tag{17}$$

$$PT_{[r]i} = \sum_{j=1}^{n} (x_{j[r]} * p_{ij}) \quad \forall r = 1 \dots n, \forall i = 1 \dots m \tag{18}$$

$$AT_{[r]} = \sum_{j=1}^{n} (x_{j[r]} * a_j) \quad \forall r = 1 \dots n \tag{19}$$

$$FT_{[r]i} \geq \sum_{a=1}^{r} PT_{ai} \quad \forall r = 1 \dots n, \forall i = 1 \dots m \tag{20}$$

$$C_{[r]} \geq FT_{[r]i} \quad \forall r = 1 \dots n, \forall i = 1 \dots m \tag{21}$$

$$T_{[r]} \geq CT_{[r]} - \sum_{j=1}^{n} x_{j[r]} * d_j \quad \forall r = 1 \dots n \tag{22}$$

$$E_{[r]} \geq \sum_{j=1}^{n} (x_{j[r]} * d_j) - CT_{[r]} \quad \forall r = 1 \dots n \tag{23}$$

$$ZZ_{[r]}, CT_{[r]}, PT_{[r]i}, AT_{[r]}, FT_{[r]i}, C_{[r]}, T_{[r]}, E_{[r]} \geq 0 \tag{24}$$

$$x_{jr} \in \{0,1\} \quad \forall j, r \tag{25}$$

Las restricciones (12) y (13) aseguran que cada trabajo empiece a procesarse en la etapa 2 una vez haya terminado de procesarse en la etapa 1 y la máquina de ensamblaje esté disponible. La restricción (14) calcula el tiempo de finalización del primer trabajo en la etapa 2, mientras que la (15) hace lo mismo con los demás trabajos de la secuencia. En la (16) y en la (17), se asegura que cada trabajo esté sólo en una posición, y que en cada posición sólo haya un trabajo, respectivamente. La restricción (18) calcula el tiempo de proceso de cada trabajo en la etapa 1 según su posición en la secuencia y, similarmente, se calcula el tiempo de proceso en la etapa 2 en la restricción (19). La restricción (20) calcula el tiempo de finalización de cada trabajo en cada máquina de la primera etapa, mientras que la (21) asegura que el tiempo de finalización de un trabajo en la etapa 1 siempre sea mayor o igual que el tiempo calculado anteriormente. Las restricciones (22) y (23) calculan la tardanza y el adelanto, respectivamente, de cada trabajo. Por último, en las restricciones (24) y (25), se determina el dominio de las variables de decisión enteras y binarias.

Ejemplo Modelo 2:

Al igual que en el modelo 1, se toman los datos del ejemplo del apartado 2.2 para resolver el problema mediante este modelo. A diferencia del modelo 1, en este modelo no hay variables que indiquen el inicio de los procesos en cada etapa. En su lugar, hay variables que determinan la finalización de los trabajos en cada etapa y el tiempo en el que cada trabajo puede empezar a procesarse en la segunda etapa. La secuencia resultante es la siguiente:

$$S = (3, 1, 2, 4, 5, 6)$$

Por tanto, los valores de las variables quedan de la siguiente manera:

	$FT_{[r]1}$	$FT_{[r]2}$	$C_{[r]}$	$ZZ_{[r]}$	$CT_{[r]}$	$E_{[r]}$	$T_{[r]}$
$r = 1$	1	3	3	3	9	0	0
$r = 2$	5	5	5	9	15	1	0
$r = 3$	7	10	10	15	23	2	0
$r = 4$	13	15	15	23	26	0	3
$r = 5$	21	16	21	26	33	0	3
$r = 6$	26	25	26	33	37	0	0

Tabla 4. Valores de las variables en el modelo 2 con la instancia "ejemplo"

En este modelo, se puede apreciar que la variable $\boldsymbol{C_{[r]}}$ es el máximo entre los valores de $\boldsymbol{FT_{[r]i}}$. A su vez, la variable $\boldsymbol{ZZ_{[r]}}$ es el máximo entre la variable $\boldsymbol{C_{[r]}}$ y la $\boldsymbol{CT_{[r-1]}}$. El valor de la función objetivo es:

$$\sum_{r=0}^{n} \boldsymbol{E_{[r]}} + \sum_{r=0}^{n} \boldsymbol{T_{[r]}} = \mathbf{9}$$

El diagrama de Gantt para el modelo 2 es el siguiente:

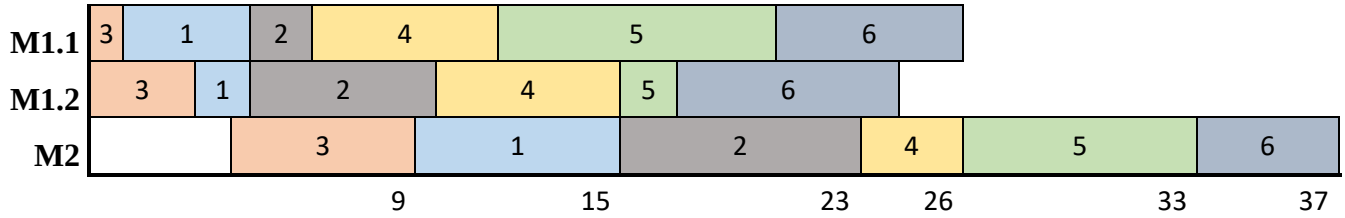

Figura 4. Diagrama de Gantt de la instancia "ejemplo" en el modelo 2

Se puede observar que este modelo resuelve la instancia procesando lo antes posible todos los trabajos en la primera etapa, a diferencia del modelo 1. Aunque en la segunda etapa, retrasa el inicio del proceso del trabajo 3 para ajustarlo a su fecha de entrega exacta. El resto de los trabajos son procesados sin tiempo ocioso en la segunda etapa para minimizar holguras entre su fecha de entrega y su tiempo de finalización ($\boldsymbol{CT_{[r]}}$).

3.2.3 Modelo 3

Este tercer modelo es una adaptación del problema presentado en el artículo Navaei et al. (2014). En él, se estudia un problema de ensamblado en dos etapas con *m* máquinas en la primera etapa y *kk* máquinas en la segunda, por lo que, para su adaptación, se ha eliminado la variable binaria que determinaba el orden en que se procesaba cada trabajo en cada máquina de la segunda etapa, y sus respectivas restricciones. Además, se han eliminado los parámetros y las variables relacionados con los costes, y los tiempos de preparación. Al trabajar en la función objetivo del artículo con variables de adelanto y retraso, no ha sido necesario añadir ninguna restricción que determinara el valor de éstas. Este modelo es el que más restricciones presenta de los cuatro (19). Esto es consecuencia de que las variables $T_{[r]}$ y $E_{[r]}$ están reguladas por las variables binarias $h_{[r]}$ y $hh_{[r]}$, lo que genera cinco restricciones adicionales.

Restricciones Modelo 3:

$$\sum_{r=1}^{n} x_{j[r]} = 1 \quad \forall j = 1 \dots n \tag{26}$$

$$\sum_{j=1}^{n} x_{j[r]} = 1 \quad \forall r = 1 \dots n \tag{27}$$

$$PT_{[r]i} = \sum_{j=1}^{n} (x_{j[r]} * p_{ij}) \quad \forall i = 1 \dots m, \forall r = 1 \dots n \tag{28}$$

$$AT_{[r]} = \sum_{j=1}^{n} (x_{j[r]} * a_j) \quad \forall r = 1 \dots n \tag{29}$$

$$DD_{[r]} = \sum_{j=1}^{n} (x_{j[r]} * d_j) \quad \forall r = 1 \dots n \tag{30}$$

$$FT_{[r]i} \geq \sum_{a=1}^{r} PT_{a\,i} \quad \forall r = 1 \dots n, \forall i = 1 \dots m \tag{31}$$

$$FT_{[r]i} \geq FT_{[r-1]i} + PT_{[r]i} \quad \forall r = 2 \dots n, \forall i = 1 \dots m \tag{32}$$

$$C_{[r]} \geq FT_{[r]i} \quad \forall r = 1 \dots n, \forall i = 1 \dots m \tag{33}$$

$$CT_{[r]} \geq FT_{[r]i} + AT_{[r]} \tag{34}$$

$$CT_{[r]} \geq CT_{[r-1]} + AT_{[r]} \tag{35}$$

$$T_{[r]} \geq CT_{[r]} - DD_{[r]} \quad \forall r = 1 \dots n \tag{36}$$

$$E_{[r]} \geq DD_{[r]} - CT_{[r]} \quad \forall r = 1 \dots n \tag{37}$$

$$h_{[r]} + hh_{[r]} \leq 1 \quad \forall r = 1 \dots n \tag{38}$$

$$T_{[r]} \geq \varepsilon * h_{[r]} \quad \forall r = 1 \dots n \tag{39}$$

$$T_{[r]} \leq M * h_{[r]} \quad \forall r = 1 \dots n \tag{40}$$

$$E_{[r]} \geq \varepsilon * hh_{[r]} \quad \forall r = 1 \dots n \tag{41}$$

$$E_{[r]} \leq M * hh_{[r]} \quad \forall r = 1 \dots n \tag{42}$$

$$T_{[r]}, E_{[r]}, DD_{[r]}, PT_{[r]i}, AT_{[r]}, FT_{[r]i}, C_{[r]}, CT_{[r]} \geq 0 \tag{43}$$

$$x_{j[r]}, h_{[r]}, hh_{[r]} \in \{0,1\} \quad \forall j, r \tag{44}$$

Las restricciones (26) y (27) aseguran que cada trabajo esté solamente en una posición, y que en cada posición haya sólo un trabajo, respectivamente. La restricción (28) calcula los tiempos de proceso de los trabajos en la primera etapa según su posición, y la (29) hace lo mismo para la segunda etapa. Así mismo, la restricción (30) calcula las fechas de entrega de cada trabajo según la posición también. En el caso de las restricciones (31) y (32), se calcula el tiempo de finalización de cada trabajo en cada máquina de la etapa 1 y se asegura que los trabajos empiezan a procesarse en la máquina *i* de la primera etapa una vez haya terminado el trabajo anterior, mientras que en la (33) se asegura que cada trabajo no termina dicha etapa hasta que todas las operaciones de ésta hayan sido realizadas. Las restricciones (34) y (35) calculan el tiempo de finalización de cada trabajo en la etapa 2. Las restricciones (36) y (37) calculan la tardanza y el adelanto, respectivamente, de cada trabajo. En la (38), se garantiza que cada trabajo no pueda llevar tardanza ($h_{[r]}$) y adelanto ($hh_{[r]}$) al mismo tiempo. En las restricciones (39) y (40) se asegura que, si hay tardanza, $h_{[r]} = 1$, mientras que en las (41) y (42) se garantiza que, si hay adelanto, $hh_{[r]} = 1$. Nótese que M y ε en las restricciones de la

(39) a la (42) representan parámetros positivos muy alto y extremadamente pequeño, respectivamente. Finalmente, las restricciones (43) y (44) determinan el dominio de las variables de decisión.

Ejemplo Modelo 3:

En tercer lugar, se va a resolver la instancia "ejemplo" con el modelo que se plantea. Este modelo es parecido al anterior, a excepción de las variables binarias que ayudan a determinar el retraso o adelanto de los trabajos. En este caso, la secuencia resultante es:

$$S = (3, 1, 2, 4, 5, 6)$$

Dentro de las variables de decisión utilizadas para resolver el problema, se van a omitir las que corresponden a los tiempos de proceso, tanto de la primera como de la segunda etapa ($\boldsymbol{PT_{[r]i}}$, $\boldsymbol{AT_{[r]}}$), y a las fechas de entrega ($\boldsymbol{DD_{[r]}}$), ya que su única función es ordenar según la secuencia los datos proporcionados por la instancia. Los valores de las demás variables son los siguientes:

	$\boldsymbol{FT_{[r]1}}$	$\boldsymbol{FT_{[r]2}}$	$\boldsymbol{C_{[r]}}$	$\boldsymbol{CT_{[r]}}$	$\boldsymbol{T_{[r]}}$	$\boldsymbol{E_{[r]}}$	$\boldsymbol{h_{[r]}}$	$\boldsymbol{hh_{[r]}}$
$\boldsymbol{r=1}$	1	3	3	9	0	0	0	0
$\boldsymbol{r=2}$	5	5	5	15	0	1	0	1
$\boldsymbol{r=3}$	7	10	10	23	0	2	0	1
$\boldsymbol{r=4}$	13	15	15	26	3	0	1	0
$\boldsymbol{r=5}$	21	16	21	33	3	0	1	0
$\boldsymbol{r=6}$	33	33	33	37	0	0	0	0

Tabla 5. Valores de las variables en el modelo 3 con la instancia "ejemplo"

Se pueden observar dos variables binarias nuevas, que son $\boldsymbol{h_{[r]}}$ y $\boldsymbol{hh_{[r]}}$, cuyo objetivo es determinar si un trabajo tiene retraso o adelanto, respectivamente. El valor de la función objetivo es:

$$\sum_{r=0}^{n} \boldsymbol{E_{[r]}} + \sum_{r=0}^{n} \boldsymbol{T_{[r]}} = \mathbf{9}$$

Al ser una instancia poco compleja, los resultados se asemejan a los de los otros modelos. En el caso del diagrama de Gantt, el de este modelo es igual al del modelo 1 para la instancia "ejemplo":

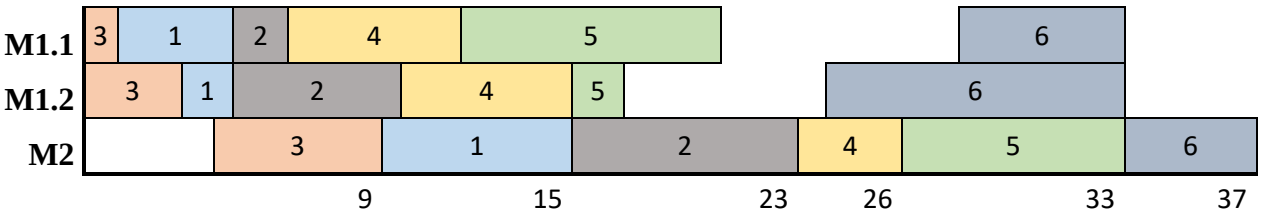

Figura 5. Diagrama de Gantt de la instancia "ejemplo" del modelo 3

3.2.4 Modelo 4

Este último modelo ha sido obtenido a partir del modelo 1, anteriormente propuesto. En éste, se han modificado varias restricciones de forma que el modelo obtenga una solución en la que no haya tiempo ocioso en ninguna de las dos etapas, porque los trabajos se secuencien inmediatamente uno tras otro. En la primera etapa, en cada una de las maquinas, cada trabajo se secuenciará inmediatamente después del anterior. En la segunda, inmediatamente después del trabajo anterior o cuando acabe el trabajo en la primera etapa. De esta manera, se ha añadido una nueva variable binaria que determina si un trabajo empieza a procesarse justo después que el trabajo anterior, o no ($\boldsymbol{Y}_{[r]}$). El objetivo de este modelo es analizar cómo varían las soluciones de un modelo en el que no se permiten tiempos ociosos, respecto a los modelos anteriores en los que sí se permiten. Este modelo presenta 13 restricciones, dos más que el modelo 1, ya que se han añadido dos restricciones que aseguran que no haya tiempo ocioso en ninguna de las dos etapas. Las restricciones del modelo se detallan a continuación.

Restricciones Modelo 4:

$$T_{[r]} \geq CT_{[r]} - \sum_{j=1}^{n} d_j * x_{jr} \quad \forall r = 1 \dots \tag{45}$$

$$E_{[r]} \geq \sum_{j=1}^{n} d_j * x_{jr} - CT_{[r]} \quad \forall r = 1 \dots n \tag{46}$$

$$CT_{[r]} \geq S2_{[r]} + \sum_{j=1}^{n} a_j * x_{jr} \quad \forall r = 1 \dots n \tag{47}$$

$$S2_{[r]} \geq S_{[r]i} + \sum_{j=1}^{n} p_{ij} * x_{jr} \quad \forall r = 1 \dots n, i = 1 \dots m \tag{48}$$

$$S2_{[r]} \leq S_{[r]i} + \sum_{j=1}^{n} p_{ij} * x_{jr} + M * Y_{[r]} \quad \forall r = 1 \dots n, i = 1 \dots m \tag{49}$$

$$S2_{[r+1]} \geq CT_{[r]} \quad \forall r = 1 \dots n - 1 \tag{50}$$

$$S2_{[r+1]} \leq CT_{[r]} + M * (1 - Y_{[r]}) \quad \forall r = 1 \dots n - 1 \tag{51}$$

$$S_{[r+1]i} = S_{[r]i} + \sum_{j=1}^{n} p_{ij} * x_{jr} \quad \forall r = 1 \dots n - 1, \qquad i = 1 \dots m \tag{52}$$

$$S_{[1]i} = 0 \quad \forall i = 1 \dots m \tag{53}$$

$$\sum_{r=1}^{n} x_{jr} = 1 \quad \forall j = 1 \dots n \tag{54}$$

$$\sum_{j=1}^{n} x_{jr} = 1 \quad \forall r = 1 \dots n \tag{55}$$

$$T_{[r]}, E_{[r]}, CT_{[r]}, S_{[r]i}, S2_{[r]} \geq 0 \quad \forall r, i \tag{56}$$

$$\boldsymbol{x_{jr}, Y_{[r]} \in \{0, 1\}} \quad \boldsymbol{\forall j, r} \tag{57}$$

Las restricciones (45) y (46) calculan la tardanza y el adelanto, respectivamente, de cada trabajo, y en la (47), se calcula el tiempo de finalización de los trabajos en la segunda etapa. Las restricciones (48) y (50) se aseguran de que los trabajos empiecen la segunda etapa una vez hayan terminado todas las operaciones en la primera y no haya otro trabajo procesándose en la segunda etapa, mientras que las (49) y (51) se aseguran de que no haya tiempos ociosos en ninguna de las dos etapas. Por otro lado, la restricción (52) asegura que cada trabajo empiece una operación en la etapa 1 una vez haya terminado el trabajo anterior, y la (53), que el primer trabajo en procesarse empiece en cero. La restricción (54) garantiza que cada trabajo esté sólo en una posición, mientras que la (55) asegura que en cada posición sólo haya un trabajo. Por último, las restricciones (56) y (57) determinan el dominio de las variables enteras y binarias, respectivamente.

Ejemplo Modelo 4:

En último lugar, se va a resolver la instancia "ejemplo" con el último modelo que se plantea. Este es muy parecido al primer modelo, solo que esta vez no se van a permitir tiempos ociosos en las máquinas, por lo que el diagrama de Gantt va a ser diferente. Al igual que en los otros modelos, la secuencia es:

$$\boldsymbol{S = (3, 1, 2, 4, 5, 6)}$$

Siguiendo este orden, los valores de las variables son:

	$\boldsymbol{S_{[r]1}}$	$\boldsymbol{S_{[r]2}}$	$\boldsymbol{S2_{[r]}}$	$\boldsymbol{CT_{[r]}}$	$\boldsymbol{E_{[r]}}$	$\boldsymbol{T_{[r]}}$	$\boldsymbol{Y_{[r]}}$
$\boldsymbol{r = 1}$	0	0	4	9	0	0	1
$\boldsymbol{r = 2}$	1	3	9	15	1	0	1
$\boldsymbol{r = 3}$	5	5	15	23	2	0	1
$\boldsymbol{r = 4}$	7	10	23	26	0	3	1
$\boldsymbol{r = 5}$	13	15	26	33	0	3	1
$\boldsymbol{r = 6}$	21	16	33	37	0	0	1

Tabla 6. Valores de las variables en el modelo 4 con la instancia "ejemplo"

Se ha omitido la variable binaria $\boldsymbol{x_{jr}}$ (como en los modelos anteriores), ya que determina solamente que la secuencia sea correcta. Sin embargo, se ha añadido $\boldsymbol{Y_{[r]}}$, ya que asegura que se obtenga una solución factible (en todos los trabajos debe ser igual a 1).

Por otro lado, se observa que el tiempo de inicio de la segunda etapa coincide con el tiempo de finalización del trabajo anterior ($\boldsymbol{S2_{[r]} = CT_{[r-1]}}$). Esto se debe a dos factores: que las fechas de

entrega tienen valores bajos, por lo que los trabajos tienden a procesarse lo antes posible; y que los tiempos de proceso en la primera etapa no son lo suficientemente grandes como para retrasar el inicio del proceso en la segunda etapa de ningún trabajo. El valor de la función objetivo es:

$$\sum_{r=0}^{n} \boldsymbol{E}_{[r]} + \sum_{r=0}^{n} \boldsymbol{T}_{[r]} = \mathbf{9}$$

Como se ha comentado anteriormente, el diagrama de Gantt debe confirmar que no hay tiempos ociosos entre trabajos en una misma etapa.

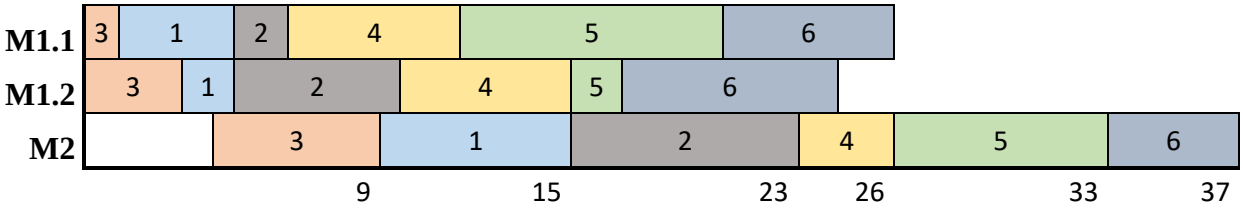

Figura 6. Diagrama de Gantt de la instancia "ejemplo" del modelo 4

Observando la figura 6, se aprecia que todos los trabajos tanto en la primera como en la segunda etapa se procesan de forma consecutiva sin tiempos ociosos en ninguna de las máquinas.

4 RESULTADOS

Vision without execution is just hallucination.

- Henry Ford -

Dentro de este apartado, se explica la forma de obtención de las instancias a resolver por los modelos, los resultados obtenidos al realizar los cálculos en el ordenador y los valores de eficiencia de cada modelo.

El problema estudiado ha sido programado en lenguaje C# en el entorno de desarrollo integrado *Visual Studio*, el cual ha sido complementado con el programa de optimización *Gurobi* para el desarrollo de los modelos y la resolución de las instancias. El ordenador utilizado para ejecutar dichos cálculos ha sido un ordenador portátil *Lenovo Ideapad Y700-15ISK,* con un procesador *Intel® Core™ i7 – 6700HQ CPU @ 2.60 GHz*, con una memoria RAM de 16 GB, y con el sistema operativo Windows 10.

4.1 Instancias

En primer lugar, se genera una batería de 60 instancias, que se obtiene a partir de tres niveles diferentes de máquinas en la primera etapa $m = \{2, 4, 6\}$, cuatro niveles diferentes de trabajos a procesar $n = \{20, 24, 28, 32\}$ y cinco instancias por cada combinación (3 niveles *m* x 4 niveles *n* x 5 instancias). Los tiempos de proceso se generan aleatoriamente siguiendo la distribución uniforme U [1, 100].

Para generar las fechas de entrega, se crea un código que las calcula utilizando la ecuación (58) (Al-Anzi and Allahverdi, 2007), con la que se generan números aleatorios comprendidos en el rango calculado. Esta ecuación tiene dos parámetros *T* y *R*, con los siguientes niveles: $T = \{0.5, 0.8\}$ y $R = \{0.2, 0.4, 0.6\}$, siendo *T* el factor Tardanza y *R* el rango relativo de las fechas de entrega. La ecuación (59) (Al-Anzi and Allahverdi, 2007), calcula la variable LB (*Lower Bound*), que es el tiempo en el que se terminan todos los trabajos, donde se utilizan los parámetros $p_{i,j}$ y a_j, que son los tiempos de proceso de los trabajos en la primera y segunda etapa, respectivamente (obtenidos de las instancias ya generadas).

$$DD = \left[LB\left(1 - T - \frac{R}{2}\right), LB\left(1 - T + \frac{R}{2}\right)\right] \tag{58}$$

$$LB = \max\left(\max_{i=1\ldots m}\left\{\sum_{j=1}^{n} p_{i,j}\right\} + \min_{j=1\ldots n} a_j, \sum_{j=1}^{n} a_j\right) \tag{59}$$

Al considerar los distintos niveles de los parámetros de la ecuación que calcula las fechas de entrega, se ha generado finalmente una batería de 360 instancias, llamada B_1 (60 instancias x 2 niveles de *T* x 3 niveles de *R*).

Tras analizar estos resultados obtenidos con esta batería, B_1 (ver Capítulo 4.3), se determina que, para obtener un mayor grado de relevancia en los resultados, se va a generar otra batería de instancias, llamada B_2, con mayores niveles de *n*. Las características de esta segunda batería de instancias son: 3 niveles de trabajos, $n = \{40, 45, 50\}$, 3 niveles de máquinas en la primera etapa, $m = \{2, 4, 6\}$, 2 niveles del parámetro $T = \{0.5, 0.8\}$, 3 niveles del parámetro $R = \{0.2, 0.4, 0.6\}$ y 5 réplicas. En total, se obtienen 270 instancias (3 niveles *n* x 3 niveles *m* x 2 niveles *T* x 3 niveles *R* x 5 instancias).

4.2 Indicadores

Para comparar los resultados obtenidos con los cuatro modelos matemáticos se van a calcular los siguientes indicadores para analizar tanto la calidad de la solución como el tiempo de cómputo:

- **Número de óptimos**. Se contabiliza, para el total de las 360 instancias (B_1) y 270 instancias (B_2), cuántas veces ha llegado el sistema a la solución óptima y cuántas no ha llegado a la óptima, pero ha llegado a una solución factible. Tanto para B_1 como para B_2 se ha establecido un tiempo límite igual a 300 segundos.

- **ACPU (*Average Computational CPU Time*)**. Con este criterio, se puede analizar el tiempo medio de cómputo empleado por cada modelo y su eficiencia.

- **ARPD (*Average Relative Percentage Difference*)**. Para obtener este valor, en primer lugar, se comparan los valores de la función objetivo de todos los modelos para cada instancia y se obtiene el mínimo. Una vez obtenido, se comparan los modelos uno por uno con ese valor y se calcula su RPD (diferencia porcentual relativa) con la ecuación (60), donde el subíndice *i* indica el número de instancia y *m* el modelo. Finalmente, se calcula el ARPD con la ecuación (61), donde *l* es el número de instancias para las que se calcula la media.

$$RPD_{i,m} = \frac{Valor\ FO_{i,m} - minimo\ valor\ FO_i}{minimo\ valor\ FO_i} * 100 \tag{60}$$

$$ARPD_m = \frac{\sum_{i=1}^{l} RPD_{i,m}}{l} \tag{61}$$

4.3 Resultados computacionales

Una vez generadas dichas instancias con sus respectivas fechas de entrega, se ha generado el código de resolución del problema, añadiendo los cuatro modelos MILP adaptados y programados (ver código en Anexo). El tiempo total de cómputo de los cuatro modelos, para las dos baterías de instancias, ha sido de 50 horas y 15 min, repartidos de la siguiente manera:

BATERÍA	MODELO 1	MODELO 2	MODELO 3	MODELO 4	TOTAL BATERÍA
B_1	3 h 21 min	3 h 21 min	3 h 22 min	3 h 24 min	13 h 28 min
B_2	9 h 11 min	9 h 11 min	9 h 13 min	9 h 12 min	36 h 47 min
TOTAL MODELO	12 h 32 min	12 h 32 min	12 h 35 min	12 h 36 min	**50 h 15 min**

Tabla 7. Tiempo total de cómputo de los modelos

Como se puede apreciar en la tabla 7, los tiempos de cómputo de cada modelo en una misma batería son muy similares, lo cual quiere decir que ninguno de ellos es significativamente mejor que otro atendiendo a este dato. Por otro lado, resulta de interés observar que B_2 requiere de casi tres veces más tiempo de cómputo que B_1, aun siendo esta inferior en cuanto a número de instancias se refiere. Esto se debe a que en los problemas NP-hard, la complejidad no aumenta de forma polinomial conforme aumenta el tamaño del problema, en este caso *n* y *m*, sino que se hace más y más difícil de resolver.

4.3.1 Número de óptimos

Este indicador determina cuántas instancias han llegado al óptimo antes de los 300 segundos de cómputo. Nótese en la siguiente tabla que el máximo de óptimos por cada combinación de *n* y *m* es 30.

ÓPTIMOS EN B_1					
n	m	Modelo 1	Modelo 2	Modelo 3	Modelo 4
20	2	30	30	30	30
20	4	30	30	30	30
20	6	30	30	30	30
24	2	29	29	29	30
24	4	29	29	29	29
24	6	30	30	30	30
28	2	28	28	28	26
28	4	27	27	27	28
28	6	29	29	29	30
32	2	26	26	26	27
32	4	25	25	25	25
32	6	28	28	28	28
Total óptimos		341	341	341	343

Tabla 8. Número de óptimos en cada modelo para B_1

Los tres primeros modelos obtienen resultados idénticos. Sin embargo, en el modelo 4 hay una ligera diferenciación de 2 óptimos más que en los otros (ver Tabla 8). Para una comparativa más precisa, se han calculado los óptimos en la segunda batería de instancias, y ésta ha mostrado datos aún más similares, habiendo el mismo número de óptimos en todos los modelos (ver Tabla 9). Véase también que la proporción de óptimos en B_2 (78,89%) es claramente inferior a la de B_1 (94,72%). Por tanto, se comprueba que conforme aumenta el tamaño del problema, los métodos exactos son menos eficientes en la resolución del problema, pues la complejidad del problema aumenta y el tiempo necesario para su resolución también.

ÓPTIMOS EN B_2					
n	m	Modelo 1	Modelo 2	Modelo 3	Modelo 4
40	2	24	24	24	24
40	4	23	23	23	24
40	6	22	22	22	23
45	2	23	23	23	23
45	4	27	27	27	27
45	6	25	25	25	24
50	2	21	21	21	20
50	4	22	22	22	22
50	6	26	26	26	26
Total óptimos		213	213	213	213

Tabla 9. Número de óptimos en cada modelo para B_2

4.3.2 ACPU

Como se ha mostrado al principio del apartado 4.3, los tiempos totales de cómputo son muy similares entre los modelos, por lo que resulta interesante comparar los tiempos medios de cómputo en conjuntos de 30 instancias y así obtener un estudio más preciso del comportamiento de cada uno de estos conjuntos.

ACPU EN B_1					
n	m	Modelo 1	Modelo 2	Modelo 3	Modelo 4
20	2	4,91	4,86	4,88	4,08
20	4	7,91	7,62	7,74	13,31
20	6	5,02	5,00	5,47	6,29
24	2	23,07	23,15	23,38	24,30
24	4	17,06	17,13	17,23	16,03
24	6	15,57	15,73	15,92	15,52
28	2	50,36	50,52	50,51	54,44
28	4	46,46	46,55	46,62	45,37
28	6	43,72	43,73	43,67	47,89
32	2	59,39	58,89	58,96	52,84
32	4	78,04	77,82	77,88	78,45
32	6	51,02	51,03	51,14	51,23
Promedio ACPU		33,54	33,50	33,62	34,15

Tabla 10. ACPU (en segundos) de cada modelo para B_1

Según los datos recogidos en la Tabla 10, el modelo 2 es el que menor tiempo medio de cómputo presenta, mientras que el modelo 4 es el que más. Los tres primeros modelos tienen tiempos medios casi idénticos en todos los conjuntos, con una diferencia entre ellos siempre < 1 segundos. El modelo 4, por otra parte, presenta variaciones según el conjunto, llegando a haber una diferencia de $+5{,}69$ segundos en $n = 20$ y $m = 4$, y de $-6{,}05$ segundos en $n = 32$ y $m = 2$ con respecto al mínimo.

ACPU EN B_2					
n	m	Modelo 1	Modelo 2	Modelo 3	Modelo 4
40	2	100,11	100,18	100,03	97,36
40	4	94,91	95,00	94,95	88,90
40	6	103,20	103,32	103,49	107,82
45	2	111,77	112,17	112,31	104,24
45	4	111,47	111,59	111,82	116,13
45	6	118,96	119,59	119,96	132,05
50	2	167,44	167,36	168,12	159,46
50	4	140,80	140,29	141,86	139,98
50	6	151,63	150,76	151,69	156,03
Promedio ACPU		122,61	122,60	123,05	122,80

Tabla 11. ACPU (en segundos) de cada modelo para B_2

En la segunda batería de instancias, se aprecian unos resultados algo parecidos a los de la primera. El modelo 2 sigue siendo el que menor tiempo medio de cómputo presenta, esta vez junto al modelo 1 (0,01 segundos de tiempo medio se puede considerar despreciable). En este caso, el que mayor tiempo medio presenta es el modelo 3, aunque con una diferencia menor a 0.5 segundos con respecto al modelo 2. En cuanto al modelo 4, su tiempo medio se encuentra muy próximo al de los modelos 1 y 2, pero sigue teniendo una gran fluctuación según el conjunto de instancias. En $n = 45$ y $m = 6$, la diferencia es de $+13{,}09$ segundos, y en $n = 50$ y $m = 2$, de $-7{,}90$ segundos con respecto al mínimo. Estudiando más en profundidad las diferencias del modelo 4 con respecto al mínimo se aprecia que, generalmente, a mayor nivel de *m*, mayor es esta diferencia.

Se puede decir que, a nivel de tiempo medio de cómputo, los cuatro modelos requieren de un tiempo similar para resolver las dos baterías de instancias.

4.3.3 ARPD

Este indicador muestra la diferencia porcentual del valor de la función objetivo de cada modelo con respecto al mínimo.

ARPD EN B_1					
n	m	Modelo 1	Modelo 2	Modelo 3	Modelo 4
20	2	0	0	0	5,54
20	4	0	0	0	0,49
20	6	0	0	0	7,10
24	2	0	0	0	2,77
24	4	0	0	0	0
24	6	0	0	0	16,73
28	2	0	0	0	0,20
28	4	0	0	0	0
28	6	0	0	0	4,50
32	2	0	0	0	0,44
32	4	0	0	0	2,55
32	6	0	0	0	1,82
Promedio ARPD		0	0	0	3,51

Tabla 12. Valores de ARPD de cada modelo para B_1

En la primera batería, los modelos 1, 2 y 3 obtienen los mismos valores de la función objetivo en las 360 instancias, por lo que cualquiera de los tres modelos se puede utilizar para resolver de manera exacta el problema estudiado. En el caso del modelo 4, se aprecia ya una diferenciación con respecto a los otros modelos, siempre con valores de la FO iguales o superiores a los otros tres (ver Tabla 12). Esto se debe a que, en este modelo, no se permiten tiempos ociosos en las máquinas, lo que provoca que en la solución encontrada haya trabajos que se entreguen antes de su fecha de entrega y se obtengan mayores valores de la función objetivo.

ARPD EN B_2					
n	m	Modelo 1	Modelo 2	Modelo 3	Modelo 4
40	2	0	0	0	7,80
40	4	0	0	0	0,62
40	6	0	0	0	5,59
45	2	0	0	0	0
45	4	0	0	0	1,32
45	6	0	0	0	0
50	2	0	0	0	0,29
50	4	0	0	0	0,41
50	6	0	0	0	3,38
Promedio ARPD		0	0	0	2,16

Tabla 13. Valores de ARPD de cada modelo para B_2

En la segunda batería, los resultados obtenidos son semejantes a los de B_1, aunque con unos valores de FO en el modelo 4 que se aproximan más a los de los otros modelos (ver Tabla 13). Por el mismo motivo que en la batería anterior, al obligar el modelo 4 a que no haya tiempo ociosos en las máquinas, algunos trabajos son forzados a terminar su procesamiento antes de su fecha de entrega, obteniendo unos valores de la variable de adelanto (E_j) (y como consecuencia, de la función objetivo) considerablemente mayores que en los otros modelos.

Por tanto, se puede afirmar que los modelos 1, 2 y 3 muestran valores de ARPD idénticos, como consecuencia de tener los mismos valores de la función objetivo en las 630 instancias resueltas, y que el modelo 4 se encuentra por debajo de estos en cuanto a eficiencia se refiere, según el criterio de ARPD, ya que presenta valores de este indicador siempre iguales o superiores, y nunca inferiores.

En lo que respecta a este indicador, resulta de interés mostrar los resultados de B_2 en función de los parámetros T y R. Tal y como se aprecia en la tabla 14, el ARPD del modelo 4 en la combinación $T = 0.8$ y $R = 0.6$ es igual a cero, es decir, que el modelo que obliga a no tener tiempos ociosos en las máquinas encuentra siempre la misma solución que los otros modelos. Además, en la combinación $T = 0.5$ y $R = 0.6$, las soluciones obtenidas están muy cercanas al óptimo, ya que presenta un ARPD de 0,63. Este comportamiento se debe al valor que las fechas de entrega han adquirido con la ecuación (58), la cual depende de T y R. Cuanto mayor es el nivel de R, mayor es la amplitud del rango del que se obtienen aleatoriamente las fechas de entrega. Cuanto mayor es el nivel de T, menores son los valores de las fechas de entrega.

ARPD EN B_2					
T	R	Modelo 1	Modelo 2	Modelo 3	Modelo 4
0,5	0,2	0	0	0	2,76
0,5	0,4	0	0	0	2,77
0,5	0,6	0	0	0	0,63
0,8	0,2	0	0	0	1,84
0,8	0,4	0	0	0	4,93
0,8	0,6	0	0	0	0
Promedio ARPD		0	0	0	2,16

Tabla 14. Valores de ARPD de cada modelo para B_2 (en función de T y R)

Para determinar con mayor exactitud el factor o factores que condicionan estos resultados, se va a realizar el cálculo de la media y la desviación típica del ARPD mediante las ecuaciones (62) y (63), respectivamente (Romero & Zúnica, 2013).

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{62}$$

$$s = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N-1}} \tag{63}$$

Para la Tabla 14 y la Tabla 15, se debe tener en cuenta que las instancias están agrupadas en conjuntos de 45, a diferencia de las otras tablas que dependen de *n* y *m*.

T	R	Media	Desv. típica
0,5	0,2	2,76	11,55
	0,4	2,77	12,14
	0,6	0,63	3,06
	Total	2,05	9,81
0,8	0,2	1,84	7,73
	0,4	4,93	17,15
	0,6	0	0
	Total	2,26	10,97
Total	0,2	2,30	9,78
	0,4	3,85	14,81
	0,6	0,32	2,17
	Total	2,16	10,39

Tabla 15. Media y desviación típica del ARPD del modelo 4 para B_2

Dados estos resultados, se puede afirmar que, a mayor rango (R), el método exacto sin tiempos ociosos es igual al problema con tiempos ociosos en cuanto a ARPD se refiere. En la combinación $T = 0.5$ y $R = 0.6$, el valor del ARPD es muy bajo, por lo que se considera despreciable, y en $T = 0.8$ y $R = 0.6$, el ARPD es igual a cero. Además, en estas dos combinaciones es donde el modelo presenta menor desviación típica. Asimismo, se puede apreciar que el tamaño de los valores de las fechas de entrega (T) no parece tener influencia en la calidad de las soluciones, pues los valores de ARPD presentan poca diferencia. En ambos niveles, $T = 0.5$ y $T = 0.8$, el ARPD es semejante (2.05 y 2.26, respectivamente).

5 CONCLUSIONES

If you think education is expensive, try ignorance.

- Derek Bok -

Para concluir, en este último capítulo se van a comentar las conclusiones a las que se han llegado en este estudio y las futuras investigaciones que se pueden realizar en la línea de lo investigado en este proyecto.

5.1 Conclusiones

En este Trabajo, se han adaptado y programado cuatro modelos de programación lineal entera. Tres de ellos a partir de artículos ya existentes en la literatura de la programación de la producción sobre problemas de ensamblado en dos etapas, y un cuarto modelo diseñado a partir del modelo 1 donde se ha obligado a las máquinas a no tener tiempo ocioso. El objetivo de este estudio es comparar los resultados obtenidos para una serie de instancias y analizar la eficiencia de cada uno de estos modelos.

Tras la adaptación y programación de los 4 modelos y el posterior análisis de los resultados obtenidos para 630 instancias, tomando como indicadores los mencionados en el apartado 4.2, se ha concluido que los modelos 1, 2 y 3 presentan una eficiencia prácticamente idéntica, con 2 de los 3 indicadores exactamente iguales (número de óptimos y ARPD), y el ACPU con diferencias inferiores al 0,4% entre el máximo (modelo 3) y el mínimo (modelo 2), lo cual no es considerado como una diferencia significativa.

En el caso del modelo 4, se ha determinado que es menos eficiente que los tres modelos anteriores en cuanto a ARPD, ya que presenta valores siempre iguales o superiores a estos, con una media del 2,16% de ARPD. En cuanto al ACPU, este último modelo presenta una diferencia con respecto al mínimo (modelo 2) del 0,53%, lo cual no resulta concluyente pero sí indicativo para determinar una menor eficiencia. En último lugar, atendiendo al número de óptimos, el modelo 4 es en el que más instancias han llegado al óptimo antes de los 300 segundos, con una diferencia de 2 en la primera batería y cero en la segunda, lo que significa una diferencia del 0,32% con respecto a los modelos 1, 2 y 3.

Tras el estudio más preciso de los valores del ARPD para el modelo 4 en la segunda batería, se ha determinado que con el mayor rango de valores de fechas de entrega ($R = 0.6$), dicho modelo es igual de eficiente que los otros tres atendiendo al ARPD, con diferencias despreciables para la determinación de eficiencia. El valor de T, en este caso, no influye en la calidad de las soluciones.

Como conclusión final, se puede afirmar que los modelos 1, 2 y 3 presentan una eficiencia semejante para el problema de ensamblado en dos etapas $DPm \rightarrow 1||\sum(E_j + T_j)$, mientras que el modelo 4 presenta una eficiencia ligeramente inferior con respecto a estos tres modelos. Con una debida amplitud del rango de valores de las fechas de entrega, el modelo 4 podría alcanzar el mismo grado de eficiencia que los otros 3 modelos.

5.2 Futuras líneas de investigación

En cuanto a futuras líneas de investigación relacionadas con este proyecto, se podrían implementar variaciones en los modelos adaptados, como se ha hecho en el modelo 4, que restrinjan más el problema, para reducir el número de soluciones que se estudien y con ellas el tiempo de cómputo.

Por otro lado, se podrían estudiar los modelos expuestos en este trabajo para un problema semejante, pero con varias máquinas en la segunda etapa, y adaptarlos a esta nueva variante del problema. Esta variación aumentaría la complejidad del problema, aunque sería de gran interés estudiarla ya que abarca tanto el problema estudiado, donde sólo hay una máquina en la segunda etapa, como otros problemas estudiados en la literatura en los que hay varias máquinas en la segunda etapa, como en Navaei et al. (2013).

Además, también resultaría de interés realizar un estudio sobre diferentes métodos heurísticos y/o metaheurísticos para el problema $DPm \rightarrow 1||\sum(E_j + T_j)$, y analizar la eficiencia de cada uno de ellos, tal y como se ha hecho en este proyecto con métodos de resolución exacta.

REFERENCIAS

Framinan, J. M., Leisten, R., & García, R. R. (2014). Manufacturing scheduling systems. *An integrated view on Models, Methods and Tools*, 51-63.

Estacio, C. (14 de octubre de 2011). *Explicación diagramas de precedencia*. Métodos de trabajo. http://metodostrabajo.blogspot.com/2011/10/explicacion-diagramas-de-precedencia.html

Al-Anzi, F. S., & Allahverdi, A. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, *182*(1), 80-94.

Lee, J. Y., & Bang, J. Y. (2016). A two-stage assembly-type flowshop scheduling problem for minimizing total tardiness. *Mathematical problems in engineering*, *2016*.

Navaei, J., Ghomi, S. F., Jolai, F., Shiraqai, M. E., & Hidaji, H. (2013). Two-stage flow-shop scheduling problem with non-identical second stage assembly machines. *The International Journal of Advanced Manufacturing Technology*, *69*(9-12), 2215-2226.

Navaei, J., Ghomi, S. M. T. F., Jolai, F., & Mozdgir, A. (2014). Heuristics for an assembly flow-shop with non-identical assembly machines and sequence dependent setup times to minimize sum of holding and delay costs. *Computers & operations research*, *44*, 52-65.

Caballero, J.A. (2011). *Introducción a la programación matemática con variables discretas*. https://rua.ua.es/dspace/bitstream/10045/19734/6/Introduccion_MILP.pdf

Roemer, T. A. (2006). A note on the complexity of the concurrent open shop problem. *Journal of scheduling*, *9*(4), 389-396.

Romero Villafranca, R.; Zúnica Ramajo, LR. (2013). *MÉTODOS ESTADÍSTICOS PARA INGENIEROS*. Editorial Universitat Politècnica de València. http://hdl.handle.net/10251/71972

ANEXO

I. Código generador de instancias para B_1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Generador_Instancias
{
    class Program
    {
        static int N_instancias = 5;
        static void Main(string[] args)
        {
            gen_testbed_ITOR();
        }
        static public void gen_testbed_ITOR()
        {
            string[] name = new string[80];
            int id_problema = 0;
            int[] rango_trabajos = { 20, 24, 28, 32 };
            int[] rango_maquinas_etapa_1 = { 2, 4, 6 };
            int rango_maquinas_etapa_2 = 1;

            //Bucle para el número de trabajos
            for(int i=0;i<4;i++)
            {
                //Bucle para el número de máquinas
                for(int j=0;j<3;j++)
                {
                    //Bucle para el número de instancias
                    for(int k=0;k<N_instancias;k++)
                    {
                        id_problema++;
                        string nombre_archivo = "ITOR_2_" + id_problema.ToString();

                        //Generar la semilla de números para que todas las instancias
                        que se iteran en cada caso sean las mismas
                        int random_seed = 0;
                        for(int s=0;s<Convert.ToInt32(id_problema);++s)
                        {
                            random_seed += (s + 1) * Convert.ToInt32(id_problema);
                        }
                        Random rand = new Random(random_seed);

                        using (StreamWriter archivo_salida = new
StreamWriter(nombre_archivo, true))
                        {
                            archivo_salida.WriteLine(rango_maquinas_etapa_1[j]+"
"+rango_maquinas_etapa_2);
```

```
                            archivo_salida.WriteLine(rango_trabajos[i]);

                            int[,] pt = new int[rango_maquinas_etapa_1[j],
rango_trabajos[i]];
                            int[] at = new int[ rango_trabajos[i]];

                            for (int l=0; l<rango_trabajos[i]; l++)
                            {
                                for(int m=0;m<rango_maquinas_etapa_1[j];m++)
                                {
                                    pt[m, l] = rand.Next(1, 101);
                                    archivo_salida.Write(pt[m, l]+" ");
                                }
                                at[l] = rand.Next(1, 101);
                                archivo_salida.Write(at[l]);
                                archivo_salida.Write("\n");
                            }
                        }
                    }
                }
            }
        }
    }
}
```

II. Código generador de instancias para B_2

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;


namespace Generador_Instancias
{
    class Program
    {
        static int N_instancias = 5;
        static void Main(string[] args)
        {
            gen_testbed_ITOR();
        }
        static public void gen_testbed_ITOR()
        {
            string[] name = new string[80];
            int id_problema = 0;
            int[] rango_trabajos = { 40, 45, 50 };
            int[] rango_maquinas_etapa_1 = { 2, 4, 6 };
            int rango_maquinas_etapa_2 = 1;

            //Bucle para el número de trabajos
            for(int i=0;i<3;i++)
            {
                //Bucle para el número de máquinas
                for(int j=0;j<3;j++)
                {
                    //Bucle para el número de instancias
                    for(int k=0;k<N_instancias;k++)
```

```
                    {
                        id_problema++;
                        string nombre_archivo = "ITOR_2_" + id_problema.ToString();

                        int random_seed = 0;
                        for(int s=0;s<Convert.ToInt32(id_problema);++s)
                        {
                            random_seed += (s + 1) * Convert.ToInt32(id_problema);
                        }
                        Random rand = new Random(random_seed);

                        using (StreamWriter archivo_salida = new
StreamWriter(nombre_archivo, true))
                        {
                            archivo_salida.WriteLine(rango_maquinas_etapa_1[j]+"
"+rango_maquinas_etapa_2);
                            archivo_salida.WriteLine(rango_trabajos[i]);

                            int[,] pt = new int[rango_maquinas_etapa_1[j],
rango_trabajos[i]];
                            int[] at = new int[ rango_trabajos[i]];

                            for (int l=0; l<rango_trabajos[i]; l++)
                            {
                                for(int m=0;m<rango_maquinas_etapa_1[j];m++)
                                {
                                    pt[m, l] = rand.Next(1, 101);
                                    archivo_salida.Write(pt[m, l]+" ");
                                }
                                at[l] = rand.Next(1, 101);
                                archivo_salida.Write(at[l]);
                                archivo_salida.Write("\n");
                            }
                        }
                    }
                }
            }
        }
    }
}
```

III. Código generador de fechas de entrega para B_1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Instancias_con_dd
{
    public class Instancia
    {
        public int orders;
        public int total_mach;
        public int f_stage_mach;
        public int assembly_mach;
        public int[,] pt; //(máquina primera etapa, trabajo)
        public int[,] at; //(máquina ensamblaje, trabajo)
```

```
        public int[] measures;
        public string nombre_instancia;
        public string nombre;

        public Instancia()
        {
            measures = new int[orders];
        }

        public Instancia(string archivo)
        {
            //Cargar datos del archivo
            string[] lines = File.ReadAllLines(archivo);
            nombre_instancia = Path.GetFileName(archivo);

            //Comprueba que carga todo ok (máq y trab)
            if (lines.Length < 2)
            {
                Console.Write("Fallo en formato");
                Environment.Exit(0);
            }

            //parte la primera línea
            string[] framework = lines[0].Split(' ');

            f_stage_mach = int.Parse(framework[0]);
            assembly_mach = int.Parse(framework[1]);
            total_mach = f_stage_mach + assembly_mach;

            //Carga el número de trabajos
            string[] total_orders = lines[1].Split(' ');
            orders = int.Parse(total_orders[0]);

            //Comprueba que carga bien tiempos de proceso
            if (lines.Length < (orders + 2))
            {
                Console.Write("Error en archivo (no hay tiempo para todos los
pedidos)");
                Environment.Exit(0);
            }

            pt = new int[f_stage_mach, orders];
            at = new int[assembly_mach, orders];
            for (int i = 2; i < lines.Length; i++)
            {
                string[] tiempos_pt = lines[i].Split(' ');
                //comprueba
                if (tiempos_pt.Length < f_stage_mach)
                {
                    Console.Write("Fallo en el formato del archivo (no hay tiempos
para todas las máquinas del pedido)" + (i - 2).ToString());
                    Environment.Exit(0);
                }

                for (int j = 0; j < f_stage_mach; j++)
                {
                    pt[j, i - 2] = int.Parse(tiempos_pt[j]);
                }
                for (int j = 0; j < assembly_mach; j++)
                {
```

```
                    at[j, i - 2] = int.Parse(tiempos_pt[f_stage_mach]);
                }
            }

        }//fin constructor

    }

    class Program
    {
        static void Main(string[] args)
        {
            if (args.Length != 2)
            {
                Console.WriteLine("Incorrect number of arguments. Usage: Instancias
[input file] [config_file] [output file]");
                Console.WriteLine(args.Length);
            }
            else
            {
                double[] T = { 0.5,0.8};
                double[] R = { 0.2,0.4,0.6};
                int nueva_inst = 1;

                for (int num_inst = 1; num_inst <= 60; num_inst++)
                {
                    Instancia inst = new Instancia("ITOR_1_" + num_inst);
                    double LB = calculo_LB(inst);
                    for (int t = 0; t < T.Length; t++)
                    {
                        for (int r = 0; r < R.Length; r++)
                        {
                            double[] dd = calculo_fechas_entrega(inst, LB, num_inst,
T[t], R[r]);
                            StreamWriter inst_dd = new StreamWriter("ITOR_dd_1_" +
nueva_inst);

                            inst_dd.WriteLine(inst.f_stage_mach + " " +
inst.assembly_mach);
                            inst_dd.WriteLine(inst.orders);
                            for (int rows_jobs = 2; rows_jobs < inst.orders + 2;
rows_jobs++)
                            {
                                for (int columns = 0; columns < inst.f_stage_mach;
columns++)
                                {
                                    inst_dd.Write(inst.pt[columns, rows_jobs - 2] + "
");
                                }
                                inst_dd.Write(inst.at[0, rows_jobs - 2] + " ");
                                inst_dd.WriteLine(dd[rows_jobs - 2]);
                            }
                            inst_dd.Close();
                            nueva_inst++;
                        }
                    }
                }
            }
        }
```

```
        static public double calculo_LB(Instancia inst)
        {
            double LB = 0;
            int n = inst.orders;
            int m1 = inst.f_stage_mach;
            int m2 = inst.assembly_mach;

            double[] w = new double[n]; //average amount of stage-one work order
            double[] w_suma = new double[n];
            int[] orden_1 = new int[n]; //orden para la primera parte del primer
término --> LPT de w_j
            int[,] claves_2 = new int[n, m1];
            for (int j = 0; j < n; j++)
            {
                for (int k = 0; k < m1; k++)
                {
                    w_suma[j] += inst.pt[k, j];
                }
                w[j] = (double)Math.Round((double)w_suma[j] / m1, 0);
                orden_1[j] = j;
            }
            Array.Sort(w_suma, orden_1); //Secuencia SPT con w como indicador
            int max_tp_1etapa = 0; //Almacena el máximo en  todas las máquinas de la
primera etapa de la suma de los tiempos de proceso de todos los jobs i secuenciados
antes que j
            //CÁLCULO PRIMER TÉRMINO
            double termino_CO = 0;
            int suma_p = 0;
            double suma_w = 0;

            for (int i = 0; i < n; i++)
            {
                suma_w = 0;
                max_tp_1etapa = 0;
                for (int j = 0; j <= i; j++) //Para todo j menor que i, calculo suma
w --> primera parte
                {
                    suma_w += w[orden_1[j]];
                }
                for (int k = 0; k < m1; k++)
                {
                    int[] p = new int[n];
                    int[] orden_2 = new int[n];
                    for (int j = 0; j < n; j++)
                    {
                        p[j] = inst.pt[k, j];
                        claves_2[j, k] = j;
                        orden_2[j] = claves_2[j, k];
                    }
                    Array.Sort(p, orden_2); //Secuencia SPT para cada máquina, 2a
parte, primer término
                    suma_p = 0;
                    for (int j = 0; j < i; j++) //Para todo j menor que i, suma p -->
segunda parte
                    {
                        suma_p += inst.pt[k, orden_2[j]];
                    }
                    if (suma_p > max_tp_1etapa)
                    {
                        max_tp_1etapa = suma_p;
```

```
                    }
                }
                termino_CO += Math.Max(suma_w, max_tp_1etapa);
            }
            //CÁLCULO SEGUNDO TÉRMINO
            int termino_assembly = 0;
            for (int i = 0; i < n; i++)
            {
                termino_assembly += inst.at[0, orden_1[i]]; //como se suman todos los
at, el orden es indiferente
            }
            LB = termino_CO + termino_assembly;

            return LB;
        }

        static public double[] calculo_fechas_entrega(Instancia inst, double LB, int
id_problem, double T, double R)
        {
            double[] dd_1 = new double[inst.orders];
            double[] dd_2 = new double[inst.orders];

            //Genera la semilla para cada instancia
            int random_seed = 0;
            for (int s = 0; s < Convert.ToInt32(id_problem); ++s)
            {
                random_seed += (s + 1) * Convert.ToInt32(id_problem);
            }
            // genera la semilla de números
            Random rand = new Random(random_seed);

            double lim_sup = (1 - T + (R / 2)) * LB;
            double lim_inf = Math.Max((1 - T - (R / 2)) * LB, 0);
            for (int j = 0; j < inst.orders; j++)
            {
                dd_1[j] = rand.NextDouble() * ((lim_sup + 1) - (lim_inf + 1)) +
(lim_inf + 1);
                dd_2[j] = Math.Round(dd_1[j]);
            }

            return dd_2;
        }
    }
}
```

IV. Código generador de fechas de entrega para B_2

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Instancias_con_dd
{
    public class Instancia
    {
```

```
        public int orders;
        public int total_mach;
        public int f_stage_mach;
        public int assembly_mach;
        public int[,] pt; //(máquina primera etapa, trabajo)
        public int[,] at; //(máquina ensamblaje, trabajo)
        public int[] measures;
        public string nombre_instancia;
        public string nombre;

        public Instancia()
        {
            measures = new int[orders];
        }

        public Instancia(string archivo)
        {
            //Cargar datos del archivo
            string[] lines = File.ReadAllLines(archivo);
            nombre_instancia = Path.GetFileName(archivo);

            //Comprueba que carga todo ok (máq y trab)
            if (lines.Length < 2)
            {
                Console.Write("Fallo en formato");
                Environment.Exit(0);
            }

            //parte la primera línea
            string[] framework = lines[0].Split(' ');

            f_stage_mach = int.Parse(framework[0]);
            assembly_mach = int.Parse(framework[1]);
            total_mach = f_stage_mach + assembly_mach;

            //Carga el número de trabajos
            string[] total_orders = lines[1].Split(' ');
            orders = int.Parse(total_orders[0]);

            //Comprueba que carga bien tiempos de proceso
            if (lines.Length < (orders + 2))
            {
                Console.Write("Error en archivo (no hay tiempo para todos los
pedidos)");
                Environment.Exit(0);
            }

            pt = new int[f_stage_mach, orders];
            at = new int[assembly_mach, orders];
            for (int i = 2; i < lines.Length; i++)
            {
                string[] tiempos_pt = lines[i].Split(' ');
                //comprueba
                if (tiempos_pt.Length < f_stage_mach)
                {
                    Console.Write("Fallo en el formato del archivo (no hay tiempos
para todas las máquinas del pedido)" + (i - 2).ToString());
                    Environment.Exit(0);
                }
```

```
                for (int j = 0; j < f_stage_mach; j++)
                {
                    pt[j, i - 2] = int.Parse(tiempos_pt[j]);
                }
                for (int j = 0; j < assembly_mach; j++)
                {
                    at[j, i - 2] = int.Parse(tiempos_pt[f_stage_mach]);
                }
            }

        }//fin constructor

    }

    class Program
    {
        static void Main(string[] args)
        {
            if (args.Length != 2)
            {
                Console.WriteLine("Incorrect number of arguments. Usage: Instancias
[input file] [config_file] [output file]");
                Console.WriteLine(args.Length);
            }
            else
            {
                double[] T = { 0.5,0.8};
                double[] R = { 0.2,0.4,0.6};
                int nueva_inst = 1;

                for (int num_inst = 1; num_inst <= 45; num_inst++)
                {
                    Instancia inst = new Instancia("ITOR_2_" + num_inst);
                    double LB = calculo_LB(inst);
                    for (int t = 0; t < T.Length; t++)
                    {
                        for (int r = 0; r < R.Length; r++)
                        {
                            double[] dd = calculo_fechas_entrega(inst, LB, num_inst,
T[t], R[r]);
                            StreamWriter inst_dd = new StreamWriter("ITOR_dd_2_" +
nueva_inst);

                            inst_dd.WriteLine(inst.f_stage_mach + " " +
inst.assembly_mach);
                            inst_dd.WriteLine(inst.orders);
                            for (int rows_jobs = 2; rows_jobs < inst.orders + 2;
rows_jobs++)
                            {
                                for (int columns = 0; columns < inst.f_stage_mach;
columns++)
                                {
                                    inst_dd.Write(inst.pt[columns, rows_jobs - 2] + "
");
                                }
                                inst_dd.Write(inst.at[0, rows_jobs - 2] + " ");
                                inst_dd.WriteLine(dd[rows_jobs - 2]);
                            }
                            inst_dd.Close();
                            nueva_inst++;
```

```
                    }
                }
            }
        }
    }

    static public double calculo_LB(Instancia inst)
    {
        double LB = 0;
        int n = inst.orders;
        int m1 = inst.f_stage_mach;
        int m2 = inst.assembly_mach;

        double[] w = new double[n]; //average amount of stage-one work order
        double[] w_suma = new double[n];
        int[] orden_1 = new int[n]; //orden para la primera parte del primer
término --> LPT de w_j
        int[,] claves_2 = new int[n, m1];
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < m1; k++)
            {
                w_suma[j] += inst.pt[k, j];
            }
            w[j] = (double)Math.Round((double)w_suma[j] / m1, 0);
            orden_1[j] = j;
        }
        Array.Sort(w_suma, orden_1); //Secuencia SPT con w como indicador
        int max_tp_1etapa = 0; //Almacena el máximo en  todas las máquinas de la
primera etapa de la suma de los tiempos de proceso de todos los jobs i secuenciados
antes que j
        //CÁLCULO PRIMER TÉRMINO
        double termino_CO = 0;
        int suma_p = 0;
        double suma_w = 0;

        for (int i = 0; i < n; i++)
        {
            suma_w = 0;
            max_tp_1etapa = 0;
            for (int j = 0; j <= i; j++) //Para todo j menor que i, calculo suma
w --> primera parte
            {
                suma_w += w[orden_1[j]];
            }
            for (int k = 0; k < m1; k++)
            {
                int[] p = new int[n];
                int[] orden_2 = new int[n];
                for (int j = 0; j < n; j++)
                {
                    p[j] = inst.pt[k, j];
                    claves_2[j, k] = j;
                    orden_2[j] = claves_2[j, k];
                }
                Array.Sort(p, orden_2); //Secuencia SPT para cada máquina, 2a
parte, primer término
                suma_p = 0;
                for (int j = 0; j < i; j++) //Para todo j menor que i, suma p -->
segunda parte
```

```
                    {
                        suma_p += inst.pt[k, orden_2[j]];
                    }
                    if (suma_p > max_tp_1etapa)
                    {
                        max_tp_1etapa = suma_p;
                    }
                }
                termino_CO += Math.Max(suma_w, max_tp_1etapa);
            }
            //CÁLCULO SEGUNDO TÉRMINO
            int termino_assembly = 0;
            for (int i = 0; i < n; i++)
            {
                termino_assembly += inst.at[0, orden_1[i]]; //como se suman todos los
at, el orden es indiferente
            }
            LB = termino_CO + termino_assembly;

            return LB;
        }

        static public double[] calculo_fechas_entrega(Instancia inst, double LB, int
id_problem, double T, double R)
        {
            double[] dd_1 = new double[inst.orders];
            double[] dd_2 = new double[inst.orders];

            //Genera la semilla para cada instancia
            int random_seed = 0;
            for (int s = 0; s < Convert.ToInt32(id_problem); ++s)
            {
                random_seed += (s + 1) * Convert.ToInt32(id_problem);
            }
            // genera la semilla de números
            Random rand = new Random(random_seed);

            double lim_sup = (1 - T + (R / 2)) * LB;
            double lim_inf = Math.Max((1 - T - (R / 2)) * LB, 0);
            for (int j = 0; j < inst.orders; j++)
            {
                dd_1[j] = rand.NextDouble() * ((lim_sup + 1) - (lim_inf + 1)) +
(lim_inf + 1);
                dd_2[j] = Math.Round(dd_1[j]);
            }

            return dd_2;
        }
    }
}
```

V. Código modelos

1. Lectura de datos

```
public class Instancia
    {
```

```
        public int n; //Trabajos
        public int m; //Máquinas en la etapa 1
        public int kk; //Máquinas en la etapa 2
        public int total_machines; //Máquinas totales
        public int[,] pt; //Tiempo de proceso en etapa 1
        public int[] at; //Tiempo de proceso en etapa 2
        public int[] dd; //Fechas de entrega
        public string datos;
        public Instancia(string archivo) //Método
        {
            datos = archivo;
            //Carga datos del archivo
            string[] lines = File.ReadAllLines(archivo);
            //Comprueba que no haya errores al cargar el numero de maquinas
            if (lines.Length < 2)
            {
                Console.WriteLine("Fallo en el formato del archivo (menos de dos
lineas)");
                Environment.Exit(0);
            }
            //Carga el numero de máquinas
            string[] framework = lines[0].Split(' ');
            m = int.Parse(framework[0]);
            kk = int.Parse(framework[1]);
            total_machines = m + kk;

            //Carga el número de trabajos
            string[] total_orders = lines[1].Split(' ');
            n = int.Parse(total_orders[0]);

            //Comprueba que no haya errores al cargar los tiempos de proceso
            if (lines.Length < (n + 2))
            {
                Console.WriteLine("Fallo en el formato del archivo (no hay tiempos
para todos los trabajos");
                Environment.Exit(0);
            }
            //Tiempos de proceso de la primera etapa
            pt = new int[n, m];
            int i, j;
            for (i = 2; i < lines.Length; i++)
            {
                string[] tiempos = lines[i].Split(' ');
                //más comprobaciones
                if (tiempos.Length < m)
                {
                    Console.WriteLine("Fallo en el formato del archivo. No hay
tiempos para todas las máquinas del trabajo " + (i - 2).ToString());
                    Environment.Exit(0);
                }
                for (j = 0; j < m; j++)
                {
                    pt[i - 2, j] = int.Parse(tiempos[j]);
                }
            }
            //Tiempos de proceso de la segunda etapa
            at = new int[n];
            for (i = 2; i < lines.Length; i++)
            {
                string[] tiempos = lines[i].Split(' ');
```

```
                //más comprobaciones
                if (tiempos.Length < kk)
                {
                    Console.WriteLine("Fallo en el formato del archivo (no hay
tiempos para todas las maquinas del trabajo" + (i - 2).ToString());
                }
                for(j=m;j<total_machines;j++)
                {
                    at[i - 2] = int.Parse(tiempos[j]);
                }
            }
            //Fechas de entrega
            dd = new int[n];
            for(i=2;i<lines.Length;i++)
            {
                string[] tiempos = lines[i].Split(' ');
                for(j=total_machines;j<total_machines+1;j++)
                {
                    dd[i - 2] = int.Parse(tiempos[j]);
                }

            }
```

2. Cálculo de los tiempos de finalización y de la FO

```
public int calculoFO(int[] secuencia)
        {
            int[,] ct1 = new int[n, m];
            int[] max_ct1 = new int[n];
            int[] ct2 = new int[n];
            int i, j;


            //Primer trabajo de la primera etapa
            for (i = 0; i < m; i++)
            {
                ct1[0, i] = pt[secuencia[0], i];
            }
            for (i = 1; i < m; i++)
            {
                max_ct1[0] = Math.Max(ct1[0, i - 1], ct1[0, i]);
            }
            //Primer trabajo de la segunda etapa
            ct2[0] = max_ct1[0] + at[secuencia[0]];

            //Resto de trabajos de la primera etapa
            for (j = 1; j < n; j++)
            {
                for (i = 0; i < m; i++)
                {
                    ct1[j, i] = ct1[j - 1, i] + pt[secuencia[j], i];
                }
                for (i = 1; i < m; i++)
                {
                    max_ct1[j] = Math.Max(ct1[j, i - 1], ct1[j, i]);
                }
            }
            //Resto de trabajos de la segunda etapa
            for (j = 1; j < n; j++)
```

```
            {
                ct2[j] = Math.Max(ct2[j - 1], max_ct1[j]) + at[secuencia[j]];
            }
            //Cálculo de la Función Objetivo
            int FO;
            int sumT = 0;
            int sumE = 0;
            int[] Lj = new int[n];
            int[] Tj = new int[n];
            int[] Ej= new int[n];
            for(i=0;i<n;i++)
            {
                Lj[i] = ct2[i] - dd[secuencia[i]];
                if (Lj[i] > 0)
                {
                    Tj[i] = Lj[i];
                    Ej[i] = 0;
                    sumT += Tj[i];
                }
                else
                {
                    Ej[i] = -Lj[i];
                    Tj[i] = 0;
                    sumE += Ej[i];
                }
            }
            FO = sumE + sumT;
            return FO;

        }
}
```

3. Declaración de variables devueltas

```
public class Resultado
    {
        public double tiempo_CPU;
        public int valor_obtenido_FO;
        public double valor_obtenido_modelo;
        public double estado;
        public Resultado()
        {

        }
        public Resultado(double _tiempo_CPU, int _valor_obtenido_FO, double
_valor_obtenido_modelo, double _estado)

        {
            tiempo_CPU = _tiempo_CPU;
            valor_obtenido_FO = _valor_obtenido_FO;
            valor_obtenido_modelo = _valor_obtenido_modelo;
            estado = _estado;
        }

    }
```

4. Ejecución de los modelos

```
class Program
```

```
    {
        static void Main(string[]args)
        {
            //Ejecución modelos
            if (args.Length != 3)
            {
                Console.WriteLine("Incorrect number of arguments");
            }
            else
            {
                Instancia inst = new Instancia(args[0]);

                //datos del entorno de Gurobi
                GRBEnv env = new GRBEnv();
                GRBModel model = new GRBModel(env);

                string[] parametros = File.ReadAllLines(args[1]);
                Resultado res = new Resultado();

                //Ejecución Modelo 1
                res = modelo1(inst, double.Parse(parametros[0]));
                using(StreamWriter writer=new StreamWriter(args[2], true))
                {
                    string optimal = "NO_OPTIMO";
                    if(res.estado==GRB.Status.OPTIMAL)
                    {
                        optimal = "OPTIMO";
                    }
                    writer.WriteLine(args[0] + "    " +
res.valor_obtenido_FO.ToString() + "    " + res.valor_obtenido_modelo.ToString() + "
" + res.tiempo_CPU.ToString() + "   " + optimal);
                }

                //Ejecución Modelo 2
                res = modelo2(inst, double.Parse(parametros[0]));
                using (StreamWriter writer = new StreamWriter(args[2], true))
                {
                    string optimal = "NO_OPTIMO";
                    if (res.estado == GRB.Status.OPTIMAL)
                    {
                        optimal = "OPTIMO";
                    }
                    writer.WriteLine(args[0] + "    " +
res.valor_obtenido_FO.ToString() + "    " + res.valor_obtenido_modelo.ToString() + "
" + res.tiempo_CPU.ToString() + "   " + optimal);
                }

                //Ejecución Modelo 3
                res = modelo3(inst, double.Parse(parametros[0]));
                using (StreamWriter writer = new StreamWriter(args[2], true))
                {
                    string optimal = "NO_OPTIMO";
                    if (res.estado == GRB.Status.OPTIMAL)
                    {
                        optimal = "OPTIMO";
                    }
                    writer.WriteLine(args[0] + "    " +
res.valor_obtenido_FO.ToString() + "    " + res.valor_obtenido_modelo.ToString() + "
" + res.tiempo_CPU.ToString() + "   " + optimal);
                }
```

```
                //Ejecución Modelo 4
                res = modelo4(inst, double.Parse(parametros[0]));
                using (StreamWriter writer = new StreamWriter(args[2], true))
                {
                    string optimal = "NO_OPTIMO";
                    if (res.estado == GRB.Status.OPTIMAL)
                    {
                        optimal = "OPTIMO";
                    }
                    writer.WriteLine(args[0] + "    " +
res.valor_obtenido_FO.ToString() + "    " + res.valor_obtenido_modelo.ToString() + "
" + res.tiempo_CPU.ToString() + "   " + optimal);
                }
            }

        }
```

5. Desarrollo del modelo 1

i. Declaración de variables

```
    public static Resultado modelo1(Instancia inst, double tiempocpu)
            {
                GRBEnv env = new GRBEnv();
                GRBModel model = new GRBModel(env);
                model.GetEnv().Set(GRB.DoubleParam.TimeLimit, tiempocpu);
                double tiempo;
                int status;
                int FO;
                double FO_modelo = 0;
                int i, j, r;

                try
                {
                    //1. VARIABLES

                    //Variable Xij (si el trabajo j esta en la posicion r)
                    GRBVar[,] X = new GRBVar[inst.n, inst.n];
                    for (j = 0; j < inst.n; j++)
                    {
                        for (r = 0; r < inst.n; r++)
                        {
                            string nombre_var = "X_" + j.ToString() + "_" +
    r.ToString();
                            X[j, r] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY,
    nombre_var); //(Limite inferior,Limite superior,Coeficiente en la FO,Tipo de
    variable,Nombre)
                        }
                    }
                   //Tiempo de finalización del trabajo en la posición r en la
    etapa 2
                    GRBVar[] CT = new GRBVar[inst.n];
                    for (r = 0; r < inst.n; r++)
                    {
                        string nombre_var = "CT_" + r.ToString();
                        CT[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
    nombre_var);
                    }
                    //Earliness
```

```
                    GRBVar[] E = new GRBVar[inst.n];
                    for (r = 0; r < inst.n; r++)
                    {
                        string nombre_var = "E_" + r.ToString();
                        E[r] = model.AddVar(0.0, GRB.INFINITY, 1.0, GRB.INTEGER,
    nombre_var);
                    }
                    //Tardiness
                    GRBVar[] T = new GRBVar[inst.n];
                    for (r = 0; r < inst.n; r++)
                    {
                        string nombre_var = "T_" + r.ToString();
                     T[r] = model.AddVar(0.0, GRB.INFINITY, 1.0, GRB.INTEGER,
nombre_var);
                    }
                    //Tiempo de inicio del trabajo en la posición r en la máquina
    i de la etapa 1
                    GRBVar[,] S = new GRBVar[inst.m, inst.n];
                    for (i = 0; i < inst.m; i++)
                    {
                        for (r = 0; r < inst.n; r++)
                        {
                            string nombre_var = "S_" + i.ToString() + "_" +
    r.ToString();
                            S[i, r] = model.AddVar(0.0, GRB.INFINITY, 0.0,
    GRB.INTEGER, nombre_var);
                        }
                    }
                    //Tiempo de inicio del trabajo en la posición r en la máquina
    de la etapa 2
                    GRBVar[] S2 = new GRBVar[inst.n];
                    for (r = 0; r < inst.n; r++)
                    {
                        string nombre_var = "S2_" + r.ToString();
                        S2[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
    nombre_var);
                    }
                    model.Update();
```

ii. Función objetivo

```
GRBLinExpr obj = 0.0;
for (r = 0; r < inst.n; r++)
{
    obj.AddTerm(1.0, E[r]);
    obj.AddTerm(1.0, T[r]);
}
model.SetObjective(obj, GRB.MINIMIZE);
model.Update();
```

iii. Restricciones

```
//3.1 Tardiness
for (r = 0; r < inst.n; r++)
{
    GRBLinExpr parte_izq = 0.0;
    GRBLinExpr parte_dcha = 0.0;
```

```
                    GRBLinExpr suma = 0.0;
                    parte_izq.AddTerm(1.0, T[r]);
                    parte_dcha.AddTerm(1.0, CT[r]);
                    for(j=0;j<inst.n;j++)
                    {
                        suma.AddTerm(inst.dd[j], X[j, r]);
                    }
                    model.AddConstr(parte_izq >= (parte_dcha - suma),
"Restriccion_1_trabajo_posicion_" + r.ToString());
                }

                //3.2 Earliness
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr parte_izq = 0.0;
                    GRBLinExpr parte_dcha = 0.0;
                    GRBLinExpr suma = 0.0;
                    parte_izq.AddTerm(1.0, E[r]);
                    parte_dcha.AddTerm(1.0, CT[r]);
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(inst.dd[j], X[j, r]);
                    }
                    model.AddConstr(parte_izq >= (suma - parte_dcha),
"Restriccion_2_trabajo_posicion_" + r.ToString());
                }

                //3.3 Tiempo de finalización de los trabajos en la segunda etapa
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr parte_izq = 0.0;
                    GRBLinExpr parte_dcha = 0.0;
                    GRBLinExpr suma = 0.0;
                    parte_izq.AddTerm(1.0, CT[r]);
                    parte_dcha.AddTerm(1.0, S2[r]);
                    for(j=0;j<inst.n;j++)
                    {
                        suma.AddTerm(inst.at[j], X[j, r]);
                    }
                    model.AddConstr(parte_izq >= (parte_dcha + suma),
"Restriccion_3_trabajo_posicion_" + r.ToString());
                }

                //3.4 Que los trabajos empiecen en la etapa 2 una vez hayan terminado
todas las operaciones en la etapa 1
                for(i=0;i<inst.m;i++)
                {
                    for(r=0;r<inst.n;r++)
                    {
                        GRBLinExpr parte_izq = 0.0;
                        GRBLinExpr parte_dcha = 0.0;
                        GRBLinExpr suma = 0.0;
                        parte_izq.AddTerm(1.0, S2[r]);
                        parte_dcha.AddTerm(1.0, S[i, r]);
                        for(j=0;j<inst.n;j++)
                        {
                            suma.AddTerm(inst.pt[j, i], X[j, r]);
                        }
                        model.AddConstr(parte_izq >= (parte_dcha + suma),
"Restriccion_4_maquina_" + i.ToString() + "_trabajo_posicion_" + r.ToString());
```

```
                    }
                }

                //3.5 Idem de la restriccion 4
                for(r=0;r<inst.n-1;r++)
                {
                    GRBLinExpr parte_izq = 0.0;
                    GRBLinExpr parte_dcha = 0.0;
                    parte_izq.AddTerm(1.0, S2[r + 1]);
                    parte_dcha.AddTerm(1.0, CT[r]);
                    model.AddConstr(parte_izq >= parte_dcha,
"Restriccion_5_trabajo_posicion_" + (1 + r.ToString()));
                }

                //3.6 Que cada trabajo empiece una operación en la etapa 1 después de
que haya acabado el anterior
                for(i=0;i<inst.m;i++)
                {
                    for(r=0;r<inst.n-1;r++)
                    {
                        GRBLinExpr parte_izq = 0.0;
                        GRBLinExpr parte_dcha = 0.0;
                        GRBLinExpr suma = 0.0;
                        parte_izq.AddTerm(1.0, S[i, r + 1]);
                        parte_dcha.AddTerm(1.0, S[i, r]);
                        for(j=0;j<inst.n;j++)
                        {
                            suma.AddTerm(inst.pt[j, i], X[j, r]);
                        }
                        model.AddConstr(parte_izq >= (parte_dcha + suma),
"Restriccion_6_maquina_" + i.ToString() + "_trabajo_posicion_" + r.ToString());
                    }
                }

                //3.7 Que el tiempo de inicio del primer trabajo sea igual o superior
a cero
                for (i = 0; i < inst.m; i++)
                {
                    GRBLinExpr term = 0.0;
                    term.AddTerm(1.0, S[i, 0]);
                    model.AddConstr(term, GRB.GREATER_EQUAL, 0.0,
"Restriccion_7_maquina_" + i.ToString());
                }

                //3.8 Que cada trabajo esté sólo en una posición
                for (j = 0; j < inst.n; j++)
                {
                    GRBLinExpr suma = 0.0;
                    for (r = 0; r < inst.n; r++)
                    {
                        suma.AddTerm(1.0, X[j, r]);
                    }
                    double term = 1.0;
                    model.AddConstr(suma == term, "Restriccion_8_trabajo_" +
j.ToString());
                }

                //3.9 Que en cada posición sólo haya un trabajo
                for (r = 0; r < inst.n; r++)
                {
```

```
                    GRBLinExpr suma = 0.0;
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(1.0, X[j, r]);
                    }
                    double term = 1.0;
                    model.AddConstr(suma == term, "Restriccion_9_trabajo_posicion_" +
r.ToString());
                }

                //3.10 Que las variables de decisión sean siempre positivas
                for(r=0;r<inst.n;r++)
                {
                    //T[r]
                    GRBLinExpr tr = 0.0;
                    tr.AddTerm(1.0, T[r]);
                    model.AddConstr(tr, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10a_trabajo_posicion_" + r.ToString());
                    //E[r]
                    GRBLinExpr er = 0.0;
                    er.AddTerm(1.0, E[r]);
                    model.AddConstr(er, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10b_trabajo_posicion_" + r.ToString());
                    //CT[r]
                    GRBLinExpr ct = 0.0;
                    ct.AddTerm(1.0, CT[r]);
                    model.AddConstr(ct, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10c_trabajo_posicion_" + r.ToString());
                    //S[r]i
                    for(i=0;i<inst.m;i++)
                    {
                        GRBLinExpr sri = 0.0;
                        sri.AddTerm(1.0, S[i,r]);
                        model.AddConstr(sri, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10d_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                    //S2[r]
                    GRBLinExpr s2 = 0.0;
                    s2.AddTerm(1.0, S2[r]);
                    model.AddConstr(s2, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10e_trabajo_posicion_" + r.ToString());
                }
                //3.11 Que las variables binarias esten entre 0 y 1
                for(r=0;r<inst.n;r++)
                {
                    for(j=0;j<inst.n;j++)
                    {
                        GRBLinExpr xjr = 0.0;
                        xjr.AddTerm(1.0, X[j, r]);
                        model.AddConstr(xjr,GRB.GREATER_EQUAL, 0.0,
"Restriccion_11a_trabajo_" + j.ToString() + "_posicion_" + r.ToString());
                        model.AddConstr(xjr, GRB.LESS_EQUAL, 1.0,
"Restriccion_11b_trabajo_" + j.ToString() + "_posicion_" + r.ToString());
                    }
                }

                model.Update();
                model.Optimize();
                tiempo = model.Get(GRB.DoubleAttr.Runtime);
                status = model.Get(GRB.IntAttr.Status);
```

```
                model.Write("modelo_1.lp");
```

iv. Sacar por pantalla la secuencia y las variables

```
                int[] secuencia = new int[inst.n];
                if (model.Get(GRB.IntAttr.Status) == GRB.Status.INFEASIBLE)
                {
                    using StreamWriter writer = new
StreamWriter("resultados_MILP.txt");
                    writer.WriteLine(inst.datos + " Infeasible model");
                }
                else
                {
                    System.Console.WriteLine("Instancia: " + inst.datos.ToString()+"
Modelo 1");
                    // la secuencia:

                    Console.Write("(");
                    for (r = 0; r < inst.n; r++)
                    {
                        j = -1;
                        do
                        {
                            j++;
                        } while (X[j, r].Get(GRB.DoubleAttr.X) == 0);
                        secuencia[r] = j;
                        if (r < (inst.n - 1))
                        {
                            Console.Write(j.ToString() + ",");
                        }
                        else
                        {
                            Console.Write(j.ToString() + ")");
                            Console.WriteLine(" ");
                        }
                    }

                    //Variable X[j,r]
                    for(j=0;j<inst.n;j++)
                    {
                        for(r=0;r<inst.n;r++)
                        {
                            Console.WriteLine("X_"+j.ToString()+"_"+r.ToString()+" =
"+X[j,r].Get(GRB.DoubleAttr.X));
                        }
                    }
                    //Tardiness
                    for(r=0;r<inst.n;r++)
                    {
                        Console.WriteLine("T_"+r.ToString()+" =
"+T[r].Get(GRB.DoubleAttr.X));
                    }
                    //Earliness
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("E_" + r.ToString() + " = " +
E[r].Get(GRB.DoubleAttr.X));
                    }
                    //Completion time
```

```
                    for(r=0;r<inst.n;r++)
                    {
                        Console.WriteLine("CT_"+r.ToString()+" =
"+CT[r].Get(GRB.DoubleAttr.X));
                    }
                    //S
                    for(i=0;i<inst.m;i++)
                    {
                        for(r=0;r<inst.n;r++)
                        {
                            Console.WriteLine("S_"+i.ToString()+"_"+r.ToString()+" =
"+S[i,r].Get(GRB.DoubleAttr.X));
                        }
                    }
                    //S2
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("S2_" + r.ToString() + " = " +
S2[r].Get(GRB.DoubleAttr.X));
                    }

                }
```

v. Extraer resultados del modelo

```
                for (r=0;r<inst.n;r++)
                {
                    FO_modelo += T[r].Get(GRB.DoubleAttr.X) +
E[r].Get(GRB.DoubleAttr.X);
                }

                FO = inst.calculoFO(secuencia);

            }
            catch (GRBException excep)
            {
                Console.WriteLine("Error code: " + excep.ErrorCode + ". " +
excep.Message);
                return null;
            }
            return new Resultado(tiempo, FO, FO_modelo, status);
        }
```

6. Desarrollo del modelo 2

i. Declaración de variables

```
        public static Resultado modelo2(Instancia inst, double tiempocpu)
        {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);
            model.GetEnv().Set(GRB.DoubleParam.TimeLimit, tiempocpu);
            int i, j, r;
            double tiempo = 0;
            int status;
            int FO;
            double FO_modelo = 0;
```

```
            try
            {
                //1. VARIABLES

                //Variable Xij (si el trabajo j esta en la posicion r)
                GRBVar[,] X = new GRBVar[inst.n, inst.n];
                for (j = 0; j < inst.n; j++)
                {
                    for (r = 0; r < inst.n; r++)
                    {
                        string nombre_var = "X_" + j.ToString() + "_" + r.ToString();
                        X[j, r] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY,
nombre_var);
                    }
                }

                //Tiempo en el que el trabajo en la posición r puede empezar el
proceso de la etapa 2
                GRBVar[] ZZ = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "ZZ_" + r.ToString();
                    ZZ[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }

                //Tiempo de finalización del trabajo en la posición r en la etapa 2
                GRBVar[] CT = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "CT_" + r.ToString();
                    CT[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }

                //Tiempo de proceso del trabajo en la posición r en la máquina i de
la etapa 1
                GRBVar[,] PT = new GRBVar[inst.n, inst.m];
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        string nombre_var = "PT_" + r.ToString() + "_" +
i.ToString();
                        PT[r, i] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                    }
                }

                //Tiempo de proceso del trabajo en la posición r en la etapa 2
                GRBVar[] AT = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "AT_" + r.ToString();
                    AT[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }

                //Tiempo de finalización del trabajo en la posición r en la máquina i
de la etapa 1
```

```
                GRBVar[,] FT = new GRBVar[inst.n, inst.m];
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        string nombre_var = "FT_" + r.ToString() + "_" +
i.ToString();
                        FT[r, i] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                    }
                }

                //Tiempo de finalización del trabajo en la posición r en la etapa 1
                GRBVar[] C = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "C_" + r.ToString();
                    C[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }

                //Earliness
                GRBVar[] E = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "E_" + r.ToString();
                    E[r] = model.AddVar(0.0, GRB.INFINITY, 1.0, GRB.INTEGER,
nombre_var);
                }

                //Tardiness
                GRBVar[] T = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "T_" + r.ToString();
                    T[r] = model.AddVar(0.0, GRB.INFINITY, 1.0, GRB.INTEGER,
nombre_var);
                }
                model.Update();
```

ii. Función objetivo

```
               GRBLinExpr obj = 0.0;
                for (r = 0; r < inst.n; r++)
                {
                    obj.AddTerm(1.0, E[r]);
                    obj.AddTerm(1.0, T[r]);
                }
                model.SetObjective(obj, GRB.MINIMIZE);
                model.Update();
```

iii. Restricciones

```
                //3.1 Que cada trabajo empiece a procesarse en la etapa 2 una vez
    haya terminado de procesarse en la etapa 1
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
```

```
                    izq.AddTerm(1.0, ZZ[r]);
                    dcha.AddTerm(1.0, C[r]);
                    model.AddConstr(izq >= dcha, "Restriccion_1_trabajo_posicion_" +
r.ToString());
                }

                //3.2 Que cada trabajo empiece a procesarse en la etapa 2 una vez la
máquina de ensamblaje esté disponible
                for (r = 1; r < inst.n; r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    izq.AddTerm(1.0, ZZ[r]);
                    dcha.AddTerm(1.0, CT[r - 1]);
                    model.AddConstr(izq >= dcha, "Restriccion_2_trabajo_posicion_" +
r.ToString());
                }

                //3.3 Tiempo de finalización del primer trabajo en la etapa 2
                GRBLinExpr ct1 = 0.0;
                GRBLinExpr parte_derecha = 0.0;
                ct1.AddTerm(1.0, CT[0]);
                parte_derecha.AddTerm(1.0, C[0]);
                parte_derecha.AddTerm(1.0, AT[0]);
                model.AddConstr(ct1 >= parte_derecha, "Restriccion_3_trabajo_0");

                //3.4 Tiempo de finalización del resto de trabajos en la etapa 2
                for (r = 1; r < inst.n; r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    izq.AddTerm(1.0, CT[r]);
                    dcha.AddTerm(1.0, ZZ[r]);
                    dcha.AddTerm(1.0, AT[r]);
                    model.AddConstr(izq >= dcha, "Restriccion_4_trabajo_" +
r.ToString());
                }

                //3.5 Que cada trabajo esté sólo en una posición
                for (j = 0; j < inst.n; j++)
                {
                    GRBLinExpr suma = 0.0;
                    for (r = 0; r < inst.n; r++)
                    {
                        suma.AddTerm(1.0, X[j, r]);
                    }
                    model.AddConstr(suma == 1.0, "Restriccion_5_trabajo_" +
j.ToString());
                }

                //3.6 Que en cada posición haya sólo un trabajo
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr suma = 0.0;
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(1.0, X[j, r]);
                    }
                    model.AddConstr(suma == 1.0, "Restriccion_6_trabajo_posicion_" +
r.ToString());
```

```
                }

                //3.7 Tiempo de proceso de cada trabajo en la etapa 1 según su
posición en la secuencia
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr izq = 0.0;
                        GRBLinExpr suma = 0.0;
                        izq.AddTerm(1.0, PT[r, i]);
                        for (j = 0; j < inst.n; j++)
                        {
                            suma.AddTerm(inst.pt[j, i], X[j, r]);
                        }
                        model.AddConstr(izq == suma,
"Restriccion_7_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                }

                //3.8 Tiempo de proceso de cada trabajo en la etapa 2 según su
posición en la secuencia
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr suma = 0.0;
                    izq.AddTerm(1.0, AT[r]);
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(inst.at[j], X[j, r]);
                    }
                    model.AddConstr(izq == suma, "Restriccion_8_trabajo_posicion_" +
r.ToString());
                }

                //3.9 Tiempo de finalización de cada trabajo en cada máquina de la
etapa 1
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr izq = 0.0;
                        GRBLinExpr suma = 0.0;
                        izq.AddTerm(1.0, FT[r, i]);
                        for (int a = 0; a < r + 1; a++)
                        {
                            suma.AddTerm(1.0, PT[a, i]);
                        }
                        model.AddConstr(izq >= suma,
"Restriccion_9_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                }

                //3.10 Que el tiempo de finalización de un trabajo en la etapa 1
siempre sea >= que el tiempo de finalización del mismo en cada máquina
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr izq = 0.0;
```

```
                        GRBLinExpr dcha = 0.0;
                        izq.AddTerm(1.0, C[r]);
                        dcha.AddTerm(1.0, FT[r, i]);
                        model.AddConstr(izq >= dcha,
"Restriccion_10_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                }

                //3.11 Tardiness
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    GRBLinExpr suma = 0.0;
                    izq.AddTerm(1.0, T[r]);
                    dcha.AddTerm(1.0, CT[r]);
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(inst.dd[j], X[j, r]);
                    }
                    model.AddConstr(izq >= dcha - suma,
"Restriccion_11_trabajo_posicion_" + r.ToString());
                }

                //3.12 Earliness
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    GRBLinExpr suma = 0.0;
                    izq.AddTerm(1.0, E[r]);
                    dcha.AddTerm(1.0, CT[r]);
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(inst.dd[j], X[j, r]);
                    }
                    model.AddConstr(izq >= suma - dcha,
"Restriccion_12_trabajo_posicion_" + r.ToString());
                }

                //3.13 Que las variables de decisión sean siempre positivas
                for(r=0;r<inst.n;r++)
                {
                    //ZZ[r]
                    GRBLinExpr zz = 0.0;
                    zz.AddTerm(1.0, ZZ[r]);
                    model.AddConstr(zz, GRB.GREATER_EQUAL, 0.0,
"Restriccion_13a_trabajo_posicion_" + r.ToString());
                    //CT[r]
                    GRBLinExpr ct = 0.0;
                    ct.AddTerm(1.0, CT[r]);
                    model.AddConstr(ct, GRB.GREATER_EQUAL, 0.0,
"Restriccion_13b_trabajo_posicion_" + r.ToString());
                    //PT[r]i
                    for(i=0;i<inst.m;i++)
                    {
                        GRBLinExpr pt = 0.0;
                        pt.AddTerm(1.0, PT[r,i]);
                        model.AddConstr(pt, GRB.GREATER_EQUAL, 0.0,
"Restriccion_13c_trabajo_posicion_" + r.ToString()+"_maquina_"+i.ToString());
```

```
                    }
                    //AT[r]
                    GRBLinExpr at = 0.0;
                    at.AddTerm(1.0, AT[r]);
                    model.AddConstr(at, GRB.GREATER_EQUAL, 0.0,
"Restriccion_13d_trabajo_posicion_" + r.ToString());
                    //FT[r]i
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr ft = 0.0;
                        ft.AddTerm(1.0, FT[r, i]);
                        model.AddConstr(ft, GRB.GREATER_EQUAL, 0.0,
"Restriccion_13e_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                    //C[r]
                    GRBLinExpr c = 0.0;
                    c.AddTerm(1.0, C[r]);
                    model.AddConstr(c, GRB.GREATER_EQUAL, 0.0,
"Restriccion_13f_trabajo_posicion_" + r.ToString());
                    //T[r]
                    GRBLinExpr t = 0.0;
                    t.AddTerm(1.0, T[r]);
                    model.AddConstr(t, GRB.GREATER_EQUAL, 0.0,
"Restriccion_13g_trabajo_posicion_" + r.ToString());
                    //E[r]
                    GRBLinExpr e = 0.0;
                    e.AddTerm(1.0, E[r]);
                    model.AddConstr(e, GRB.GREATER_EQUAL, 0.0,
"Restriccion_13h_trabajo_posicion_" + r.ToString());
                }

                //3.14 Que las variables binarias estén entre 0 y 1
                for(r=0;r<inst.n;r++)
                {
                    for(j=0;j<inst.n;j++)
                    {
                        GRBLinExpr x = 0.0;
                        x.AddTerm(1.0, X[j, r]);
                        model.AddConstr(x, GRB.GREATER_EQUAL, 0.0,
"Restriccion_14a_trabajo_" + j.ToString() + "_posicion_" + r.ToString());
                        model.AddConstr(x, GRB.LESS_EQUAL, 1.0,
"Restriccion_14b_trabajo_" + j.ToString() + "_posicion_" + r.ToString());
                    }
                }
                model.Update();
                model.Optimize();
                tiempo = model.Get(GRB.DoubleAttr.Runtime);
                status = model.Get(GRB.IntAttr.Status);
                model.Write("modelo_2.lp");
```

iv. Sacar por pantalla la secuencia y las variables

```
                int[] secuencia = new int[inst.n];
                if (model.Get(GRB.IntAttr.Status) == GRB.Status.INFEASIBLE)
                {
                    using (StreamWriter writer = new
StreamWriter("resultados_MILP.txt"))
                    {
                        writer.WriteLine(inst.datos + " Infeasible model");
```

```
                    }
                }
                else
                {
                    System.Console.WriteLine("Instancia: " + inst.datos.ToString() +
"   Modelo 2");
                    // la secuencia:


                    Console.Write("(");
                    for (r = 0; r < inst.n; r++)
                    {
                        j = -1;
                        do
                        {
                            j++;
                        } while (X[j, r].Get(GRB.DoubleAttr.X) == 0);
                        secuencia[r] = j;
                        if (r < (inst.n - 1))
                        {
                            Console.Write(j.ToString() + ",");
                        }
                        else
                        {
                            Console.Write(j.ToString() + ")");
                            Console.WriteLine(" ");
                        }
                    }
                    //Variable X[j,r]
                    for (j = 0; j < inst.n; j++)
                    {
                        for (r = 0; r < inst.n; r++)
                        {
                            Console.WriteLine("X_" + j.ToString() + "_" +
r.ToString() + " = " + X[j, r].Get(GRB.DoubleAttr.X));
                        }
                    }
                    //ZZ[r]
                    for(r=0;r<inst.n;r++)
                    {
                        Console.WriteLine("ZZ_"+r.ToString()+" =
"+ZZ[r].Get(GRB.DoubleAttr.X));
                    }
                    //CT
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("CT_" + r.ToString() + " = " +
CT[r].Get(GRB.DoubleAttr.X));
                    }
                    //AT
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("AT_" + r.ToString() + " = " +
AT[r].Get(GRB.DoubleAttr.X));
                    }
                    //Earliness
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("E_" + r.ToString() + " = " +
E[r].Get(GRB.DoubleAttr.X));
```

```
                    }
                    //Tardiness
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("T_" + r.ToString() + " = " +
T[r].Get(GRB.DoubleAttr.X));
                    }
                    //C[r]
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("C_" + r.ToString() + " = " +
C[r].Get(GRB.DoubleAttr.X));
                    }
                    //PT
                    for(i=0;i<inst.m;i++)
                    {
                        for(r=0;r<inst.n;r++)
                        {
                            Console.WriteLine("PT_"+r.ToString()+"_"+i.ToString()+" =
"+PT[r,i].Get(GRB.DoubleAttr.X));
                        }
                    }
                    //FT
                    for (i = 0; i < inst.m; i++)
                    {
                        for (r = 0; r < inst.n; r++)
                        {
                            Console.WriteLine("FT_" + r.ToString() + "_" +
i.ToString() + " = " + FT[r, i].Get(GRB.DoubleAttr.X));
                        }
                    }
                }
```

v. Extraer resultados del modelo

```
                for (r = 0; r < inst.n; r++)
                {
                    FO_modelo += T[r].Get(GRB.DoubleAttr.X) +
E[r].Get(GRB.DoubleAttr.X);
                }

                FO = inst.calculoFO(secuencia);
            }
            catch (GRBException excep)
            {
                Console.WriteLine("Error code: " + excep.ErrorCode + ". " +
excep.Message);
                return null;
            }
            return new Resultado(tiempo, FO, FO_modelo, status);
        }
```

7. Desarrollo del modelo 3

i. Declaración de variables

```
        public static Resultado modelo3(Instancia inst,double tiempocpu)
        {
```

```
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);
            model.GetEnv().Set(GRB.DoubleParam.TimeLimit, tiempocpu);
            int i, j, r;
            double tiempo = 0;
            int status;
            int FO;
            double FO_modelo = 0;

            try
            {
                //1. VARIABLES

                //Earliness
                GRBVar[] E = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "E_" + r.ToString();
                    E[r] = model.AddVar(0.0, GRB.INFINITY, 1.0, GRB.INTEGER,
nombre_var);
                }

                //Tardiness
                GRBVar[] T = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "T_" + r.ToString();
                    T[r] = model.AddVar(0.0, GRB.INFINITY, 1.0, GRB.INTEGER,
nombre_var);
                }

                //Fechas de entrega segun su posicion en la secuencia
                GRBVar[] DD = new GRBVar[inst.n];
                for(r=0;r<inst.n;r++)
                {
                    string nombre_var = "DD_" + r.ToString();
                    DD[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }

                //Tiempo de proceso del trabajo en la posición r en la máquina i de
la etapa 1
                GRBVar[,] PT = new GRBVar[inst.n, inst.m];
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        string nombre_var = "PT_" + r.ToString() + "_" +
i.ToString();
                        PT[r, i] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                    }
                }

                //Tiempo de proceso del trabajo en la posición r en la etapa 2
                GRBVar[] AT = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "AT_" + r.ToString();
```

```
                    AT[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }

                //Tiempo de finalización del trabajo en la posición r en la máquina i
de la etapa 1
                GRBVar[,] FT = new GRBVar[inst.n, inst.m];
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        string nombre_var = "FT_" + r.ToString() + "_" +
i.ToString();
                        FT[r, i] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                    }
                }

                //Tiempo de finalización del trabajo en la posición r en la etapa 1
                GRBVar[] C = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "C_" + r.ToString();
                    C[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }

                //Tiempo de finalización del trabajo en la posición r en la etapa 2
                GRBVar[] CT = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "CT_" + r.ToString();
                    CT[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }

                //Variable Xij (si el trabajo j esta en la posicion r)
                GRBVar[,] X = new GRBVar[inst.n, inst.n];
                for (j = 0; j < inst.n; j++)
                {
                    for (r = 0; r < inst.n; r++)
                    {
                        string nombre_var = "X_" + j.ToString() + "_" + r.ToString();
                        X[j, r] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY,
nombre_var);
                    }
                }

                //Variable h[r] que indica tardanza de trabajo
                GRBVar[] h = new GRBVar[inst.n];
                for(r=0;r<inst.n;r++)
                {
                    string nombre_var = "h_" + r.ToString();
                    h[r] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, nombre_var);
                }

                //Variable hh[r] que indica adelanto de trabajo
                GRBVar[] hh = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
```

```
                string nombre_var = "hh_" + r.ToString();
                hh[r] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, nombre_var);
            }
            model.Update();
```

ii. Función objetivo

```
            GRBLinExpr obj = 0.0;
            for (r = 0; r < inst.n; r++)
            {
                obj.AddTerm(1.0, E[r]);
                obj.AddTerm(1.0, T[r]);
            }
            model.SetObjective(obj, GRB.MINIMIZE);
            model.Update();
```

iii. Restricciones

```
            //3.1 Que cada trabajo esté sólo en una posición
            for (j = 0; j < inst.n; j++)
            {
                GRBLinExpr suma = 0.0;
                for (r = 0; r < inst.n; r++)
                {
                    suma.AddTerm(1.0, X[j, r]);
                }
                model.AddConstr(suma == 1.0, "Restriccion_1_trabajo_" +
j.ToString());
            }

            //3.2 Que en cada posición haya sólo un trabajo
            for (r = 0; r < inst.n; r++)
            {
                GRBLinExpr suma = 0.0;
                for (j = 0; j < inst.n; j++)
                {
                    suma.AddTerm(1.0, X[j, r]);
                }
                model.AddConstr(suma == 1.0, "Restriccion_2_trabajo_posicion_" +
r.ToString());
            }

            //3.3 Tiempos de proceso de los trabajos en la etapa 1 según su
posición
            for(r=0;r<inst.n;r++)
            {
                for(i=0;i<inst.m;i++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr suma = 0.0;
                    izq.AddTerm(1.0, PT[r, i]);
                    for(j=0;j<inst.n;j++)
                    {
                        suma.AddTerm(inst.pt[j,i], X[j, r]);
                    }
                    model.AddConstr(izq == suma,
"Restriccion_3_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                }
```

```
                }

                //3.4 Tiempos de proceso de los trabajos en la etapa 2 según su
posición
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr suma = 0.0;
                    izq.AddTerm(1.0, AT[r]);
                    for(j=0;j<inst.n;j++)
                    {
                        suma.AddTerm(inst.at[j], X[j, r]);
                    }
                    model.AddConstr(izq == suma, "Restriccion_4_trabajo_posicion_" +
r.ToString());
                }

                //3.5 Fechas de entrega según su posición
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr suma = 0.0;
                    izq.AddTerm(1.0, DD[r]);
                    for(j=0;j<inst.n;j++)
                    {
                        suma.AddTerm(inst.dd[j], X[j, r]);
                    }
                    model.AddConstr(izq == suma, "Restriccion_5_trabajo_posicion_" +
r.ToString());
                }

                //3.6 Tiempo de finalización de cada trabajo en cada máquina de la
etapa 1
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr izq = 0.0;
                        GRBLinExpr suma = 0.0;
                        izq.AddTerm(1.0, FT[r, i]);
                        for (int a = 0; a < r + 1; a++)
                        {
                            suma.AddTerm(1.0, PT[a, i]);
                        }
                        model.AddConstr(izq >= suma,
"Restriccion_9_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                }

                //3.7 Asegurar que los trabajos empiezan a porcesarse en la primera
etapa una vez termina el trabajo anterior
                for(r=1;r<inst.n;r++)
                {
                    for(i=0;i<inst.m;i++)
                    {
                        GRBLinExpr izq = 0.0;
                        GRBLinExpr dcha = 0.0;
                        izq.AddTerm(1.0, FT[r, i]);
                        dcha.AddTerm(1.0, FT[r - 1, i]);
                        dcha.AddTerm(1.0, PT[r, i]);
```

```
                        model.AddConstr(izq >= dcha,
"Restriccion_7_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                }

                //3.8 Que cada trabajo no termine la etapa 1 hasta que todas las
operaciones de ésta hayan sido realizadas
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    izq.AddTerm(1.0, C[r]);
                    for(i=0;i<inst.m;i++)
                    {
                        GRBLinExpr dcha = 0.0;
                        dcha.AddTerm(1.0, FT[r, i]);
                        model.AddConstr(izq >= dcha,
"Restriccion_8_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                }

                //3.9 Tiempo de finalización de los trabajos en la etapa 2
                for (r = 0; r < inst.n; r++)
                {
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr izq = 0.0;
                        GRBLinExpr dcha = 0.0;
                        izq.AddTerm(1.0, CT[r]);
                        dcha.AddTerm(1.0, FT[r, i]);
                        dcha.AddTerm(1.0, AT[r]);
                        model.AddConstr(izq >= dcha, "Resticcion_9_trabajo_posicion_"
+ r.ToString() + "_maquina_" + i.ToString());
                    }
                }

                //3.10 Que el trabajo r empiece una vez haya terminado el trabajo r-1
                for (r = 1; r < inst.n; r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    izq.AddTerm(1.0, CT[r]);
                    dcha.AddTerm(1.0, CT[r - 1]);
                    dcha.AddTerm(1.0, AT[r]);
                    model.AddConstr(izq >= dcha, "Restriccion_10_trabajo_posicion_" +
r.ToString());
                }
                //3.11 Tardiness
                for (r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha1 = 0.0;
                    GRBLinExpr dcha2 = 0.0;
                    izq.AddTerm(1.0, T[r]);
                    dcha1.AddTerm(1.0, CT[r]);
                    dcha2.AddTerm(1.0, DD[r]);
                    model.AddConstr(izq >= dcha1 - dcha2,
"Restriccion_11_trabajo_posicion_" + r.ToString());
                }

                //3.12 Earliness
```

```
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha1 = 0.0;
                    GRBLinExpr dcha2 = 0.0;
                    izq.AddTerm(1.0, E[r]);
                    dcha1.AddTerm(1.0, CT[r]);
                    dcha2.AddTerm(1.0, DD[r]);
                    model.AddConstr(izq >= dcha2 - dcha1,
"Restriccion_12_trabajo_posicion_" + r.ToString());
                }

                //3.13 Que cada trabajo no pueda llevar tardanza y adelanto al mismo
tiempo
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    izq.AddTerm(1.0, h[r]);
                    izq.AddTerm(1.0, hh[r]);
                    model.AddConstr(izq <= 1.0, "Restriccion_13_trabajo_posicion_" +
r.ToString());
                }

                //3.14 Si hay tardanza, h[r]=1
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    izq.AddTerm(1.0, T[r]);
                    double e = 0.0001;
                    dcha.AddTerm(e, h[r]);
                    model.AddConstr(izq >= dcha, "Restriccion_14_trabajo_posicion_" +
r.ToString());
                }

                //3.15 Idem de la restricción 13
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    izq.AddTerm(1.0, T[r]);
                    double M = 1000000;
                    dcha.AddTerm(M, h[r]);
                    model.AddConstr(izq <= dcha, "Restriccion_15_trabajo_posicion_" +
r.ToString());
                }

                //3.16 Si hay adelanto, hh[r]=1
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    izq.AddTerm(1.0, E[r]);
                    double e = 0.0001;
                    dcha.AddTerm(e, hh[r]);
                    model.AddConstr(izq >= dcha, "Restriccion_16_trabajo_posicion_" +
r.ToString());
                }

                //3.17 Idem de la restricción 15
```

```
                for(r=0;r<inst.n;r++)
                {
                    GRBLinExpr izq = 0.0;
                    GRBLinExpr dcha = 0.0;
                    izq.AddTerm(1.0, E[r]);
                    double M = 1000000;
                    dcha.AddTerm(M, hh[r]);
                    model.AddConstr(izq <= dcha, "Restriccion_17_trabajo_posicion_" +
r.ToString());
                }

                //3.18 Que las variables sean positivas
                for (r = 0; r < inst.n; r++)
                {
                    //T[r]
                    GRBLinExpr t = 0.0;
                    t.AddTerm(1.0, T[r]);
                    model.AddConstr(t, GRB.GREATER_EQUAL, 0.0,
"Restriccion_18a_trabajo_posicion_" + r.ToString());
                    //E[r]
                    GRBLinExpr e = 0.0;
                    e.AddTerm(1.0, E[r]);
                    model.AddConstr(e, GRB.GREATER_EQUAL, 0.0,
"Restriccion_18b_trabajo_posicion_" + r.ToString());
                    //DD[r]
                    GRBLinExpr dd = 0.0;
                    dd.AddTerm(1.0, DD[r]);
                    model.AddConstr(dd, GRB.GREATER_EQUAL, 0.0,
"Restriccion_18c_trabajo_posicion_" + r.ToString());
                    //PT[r]i
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr pt = 0.0;
                        pt.AddTerm(1.0, PT[r, i]);
                        model.AddConstr(pt, GRB.GREATER_EQUAL, 0.0,
"Restriccion_18d_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                    //AT[r]
                    GRBLinExpr at = 0.0;
                    at.AddTerm(1.0, AT[r]);
                    model.AddConstr(at, GRB.GREATER_EQUAL, 0.0,
"Restriccion_18e_trabajo_posicion_" + r.ToString());
                    //FT[r]i
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr ft = 0.0;
                        ft.AddTerm(1.0, FT[r, i]);
                        model.AddConstr(ft, GRB.GREATER_EQUAL, 0.0,
"Restriccion_18f_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
                    //C[r]
                    GRBLinExpr c = 0.0;
                    c.AddTerm(1.0, C[r]);
                    model.AddConstr(c, GRB.GREATER_EQUAL, 0.0,
"Restriccion_18g_trabajo_posicion_" + r.ToString());
                    //CT[r]
                    GRBLinExpr ct = 0.0;
                    ct.AddTerm(1.0, CT[r]);
                    model.AddConstr(ct, GRB.GREATER_EQUAL, 0.0,
"Restriccion_18h_trabajo_posicion_" + r.ToString());
```

```
                }

                //3.19 Que las variables binarias esten entre 0 y 1
                for (r = 0; r < inst.n; r++)
                {
                    //X[j,r]
                    for (j = 0; j < inst.n; j++)
                    {
                        GRBLinExpr x = 0.0;
                        x.AddTerm(1.0, X[j, r]);
                        model.AddConstr(x, GRB.GREATER_EQUAL, 0.0,
"Restriccion_19a_trabajo_" + j.ToString() + "_posicion_" + r.ToString());
                        model.AddConstr(x, GRB.LESS_EQUAL, 1.0,
"Restriccion_19b_trabajo_" + j.ToString() + "_posicion_" + r.ToString());
                    }
                    //h[r]
                    GRBLinExpr hr = 0.0;
                    hr.AddTerm(1.0, h[r]);
                    model.AddConstr(hr, GRB.GREATER_EQUAL, 0.0,
"Restriccion_19c_trabajo_posicion_" + r.ToString());
                    model.AddConstr(hr, GRB.LESS_EQUAL, 1.0,
"Restriccion_19d_trabajo_posicion_" + r.ToString());
                    //hh[r]
                    GRBLinExpr hhr = 0.0;
                    hhr.AddTerm(1.0, hh[r]);
                    model.AddConstr(hhr, GRB.GREATER_EQUAL, 0.0,
"Restriccion_19e_trabajo_posicion_" + r.ToString());
                    model.AddConstr(hhr, GRB.LESS_EQUAL, 1.0,
"Restriccion_19f_trabajo_posicion_" + r.ToString());
                }

                model.Update();
                model.Optimize();
                tiempo = model.Get(GRB.DoubleAttr.Runtime);
                status = model.Get(GRB.IntAttr.Status);
                model.Write("modelo_3.lp");
```

iv. Sacar por pantalla la secuencia y las variables

```
                int[] secuencia = new int[inst.n];
                if (model.Get(GRB.IntAttr.Status) == GRB.Status.INFEASIBLE)
                {
                    using (StreamWriter writer = new
StreamWriter("resultados_MILP.txt"))
                    {
                        writer.WriteLine(inst.datos + " Infeasible model");
                    }
                }
                else
                {
                    System.Console.WriteLine("Instancia: " + inst.datos.ToString() +
"   Modelo 3");
                    // la secuencia:


                    Console.Write("(");
                    for (r = 0; r < inst.n; r++)
                    {
                        j = -1;
```

```
                        do
                        {
                            j++;
                        } while (X[j, r].Get(GRB.DoubleAttr.X) == 0);
                        secuencia[r] = j;
                        if (r < (inst.n - 1))
                        {
                            Console.Write(j.ToString() + ",");
                        }
                        else
                        {
                            Console.Write(j.ToString() + ")");
                            Console.WriteLine(" ");
                        }
                    }
                    //X[j,r]
                    for (j = 0; j < inst.n; j++)
                    {
                        for (r = 0; r < inst.n; r++)
                        {
                            Console.WriteLine("X_" + j.ToString() + "_" +
r.ToString() + " = " + X[j, r].Get(GRB.DoubleAttr.X));
                        }
                    }
                    //PT
                    for (i = 0; i < inst.m; i++)
                    {
                        for (r = 0; r < inst.n; r++)
                        {
                            Console.WriteLine("PT_" + r.ToString() + "_" +
i.ToString() + " = " + PT[r, i].Get(GRB.DoubleAttr.X));
                        }
                    }
                    //AT
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("AT_" + r.ToString() + " = " +
AT[r].Get(GRB.DoubleAttr.X));
                    }
                    //DD[r]
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("DD_" + r.ToString() + " = " +
DD[r].Get(GRB.DoubleAttr.X));
                    }
                    //FT
                    for (i = 0; i < inst.m; i++)
                    {
                        for (r = 0; r < inst.n; r++)
                        {
                            Console.WriteLine("FT_" + r.ToString() + "_" +
i.ToString() + " = " + FT[r, i].Get(GRB.DoubleAttr.X));
                        }
                    }
                    //C[r]
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("C_" + r.ToString() + " = " +
C[r].Get(GRB.DoubleAttr.X));
                    }
```

```
                    //CT
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("CT_" + r.ToString() + " = " +
CT[r].Get(GRB.DoubleAttr.X));
                    }
                    //Tardiness
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("T_" + r.ToString() + " = " +
T[r].Get(GRB.DoubleAttr.X));
                    }
                    //Earliness
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("E_" + r.ToString() + " = " +
E[r].Get(GRB.DoubleAttr.X));
                    }
                    //h[r]
                    for(r=0;r<inst.n;r++)
                    {
                        Console.WriteLine("h_" + r.ToString() + " = " +
h[r].Get(GRB.DoubleAttr.X));
                    }
                    //hh[r]
                    for (r = 0; r < inst.n; r++)
                    {
                        Console.WriteLine("hh_" + r.ToString() + " = " +
hh[r].Get(GRB.DoubleAttr.X));
                    }
                }
```

v. Extraer resultados del modelo

```
                for (r = 0; r < inst.n; r++)
                {
                    FO_modelo += T[r].Get(GRB.DoubleAttr.X) +
E[r].Get(GRB.DoubleAttr.X);
                }

                FO = inst.calculoFO(secuencia);
            }
            catch (GRBException excep)
            {
                Console.WriteLine("Error code: " + excep.ErrorCode + ". " +
excep.Message);
                return null;
            }
            return new Resultado(tiempo, FO, FO_modelo, status);
                }
```

8. Desarrollo del modelo 4

i. Declaración de variables

```
public static Resultado modelo4(Instancia inst, double tiempocpu)
{
    GRBEnv env = new GRBEnv();
    GRBModel model = new GRBModel(env);
    model.GetEnv().Set(GRB.DoubleParam.TimeLimit, tiempocpu);
    double tiempo;
    int status;
    int FO;
    double FO_modelo = 0;
    int i, j, r;

    try
    {
        //1. VARIABLES

        //Variable Xij (si el trabajo j esta en la posicion r)
        GRBVar[,] X = new GRBVar[inst.n, inst.n];
        for (j = 0; j < inst.n; j++)
        {
            for (r = 0; r < inst.n; r++)
            {
                string nombre_var = "X_" + j.ToString() + "_" + r.ToString();
                X[j, r] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY,
nombre_var); //(Limite inferior,Limite superior,Coeficiente en la FO,Tipo de
variable,Nombre)
            }
        }
        //Variable Yr (si el trabajo en la posicion r se porcesa justo
después que el anterior)
        GRBVar[] Y = new GRBVar[inst.n];
        for (r = 0; r < inst.n; r++)
        {
            string nombre_var = "Y_" + r.ToString();
            Y[r] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, nombre_var);
        }
        //Tiempo de finalización del trabajo en la posición r en la etapa 2
        GRBVar[] CT = new GRBVar[inst.n];
        for (r = 0; r < inst.n; r++)
        {
            string nombre_var = "CT_" + r.ToString();
            CT[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
        }
        //Earliness
        GRBVar[] E = new GRBVar[inst.n];
        for (r = 0; r < inst.n; r++)
        {
            string nombre_var = "E_" + r.ToString();
            E[r] = model.AddVar(0.0, GRB.INFINITY, 1.0, GRB.INTEGER,
nombre_var);
        }
        //Tardiness
        GRBVar[] T = new GRBVar[inst.n];
        for (r = 0; r < inst.n; r++)
```

```
                {
                    string nombre_var = "T_" + r.ToString();
                    T[r] = model.AddVar(0.0, GRB.INFINITY, 1.0, GRB.INTEGER,
nombre_var);
                }
                //Tiempo de inicio del trabajo en la posición r en la máquina i de la
etapa 1
                GRBVar[,] S = new GRBVar[inst.m, inst.n];
                for (i = 0; i < inst.m; i++)
                {
                    for (r = 0; r < inst.n; r++)
                    {
                        string nombre_var = "S_" + i.ToString() + "_" + r.ToString();
                        S[i, r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                    }
                }
                //Tiempo de inicio del trabajo en la posición r en la máquina de la
etapa 2
                GRBVar[] S2 = new GRBVar[inst.n];
                for (r = 0; r < inst.n; r++)
                {
                    string nombre_var = "S2_" + r.ToString();
                    S2[r] = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER,
nombre_var);
                }
                model.Update();
```

ii. Función objetivo

```
               GRBLinExpr obj = 0.0;
                for (r = 0; r < inst.n; r++)
                {
                    obj.AddTerm(1.0, E[r]);
                    obj.AddTerm(1.0, T[r]);
                }
                model.SetObjective(obj, GRB.MINIMIZE);
                model.Update();
```

iii. Restricciones

```
                //3.1 Tardiness
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr parte_izq = 0.0;
                    GRBLinExpr parte_dcha = 0.0;
                    GRBLinExpr suma = 0.0;
                    parte_izq.AddTerm(1.0, T[r]);
                    parte_dcha.AddTerm(1.0, CT[r]);
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(inst.dd[j], X[j, r]);
                    }
                    model.AddConstr(parte_izq >= (parte_dcha - suma),
"Restriccion_1_trabajo_posicion_" + r.ToString());
                }

                //3.2 Earliness
                for (r = 0; r < inst.n; r++)
```

```
                {
                    GRBLinExpr parte_izq = 0.0;
                    GRBLinExpr parte_dcha = 0.0;
                    GRBLinExpr suma = 0.0;
                    parte_izq.AddTerm(1.0, E[r]);
                    parte_dcha.AddTerm(1.0, CT[r]);
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(inst.dd[j], X[j, r]);
                    }
                    model.AddConstr(parte_izq >= (suma - parte_dcha),
"Restriccion_2_trabajo_posicion_" + r.ToString());
                }

                //3.3 Tiempo de finalización de los trabajos en la segunda etapa
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr parte_izq = 0.0;
                    GRBLinExpr parte_dcha = 0.0;
                    GRBLinExpr suma = 0.0;
                    parte_izq.AddTerm(1.0, CT[r]);
                    parte_dcha.AddTerm(1.0, S2[r]);
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(inst.at[j], X[j, r]);
                    }
                    model.AddConstr(parte_izq >= (parte_dcha + suma),
"Restriccion_3_trabajo_posicion_" + r.ToString());
                }

                //3.4 Que los trabajos empiecen en la etapa 2 una vez hayan terminado
todas las operaciones en la etapa 1
                for (i = 0; i < inst.m; i++)
                {
                    for (r = 0; r < inst.n; r++)
                    {
                        GRBLinExpr parte_izq = 0.0;
                        GRBLinExpr parte_dcha = 0.0;
                        GRBLinExpr suma = 0.0;
                        parte_izq.AddTerm(1.0, S2[r]);
                        parte_dcha.AddTerm(1.0, S[i, r]);
                        for (j = 0; j < inst.n; j++)
                        {
                            suma.AddTerm(inst.pt[j, i], X[j, r]);
                        }
                        model.AddConstr(parte_izq >= (parte_dcha + suma),
"Restriccion_4_maquina_" + i.ToString() + "_trabajo_posicion_" + r.ToString());
                    }
                }

                //3.4b Que el trabajo en la posicion r se procese en la segunda etapa
lo antes posible
                for(i=0;i<inst.m;i++)
                {
                    for(r=0;r<inst.n;r++)
                    {
                        GRBLinExpr izq = 0.0;
                        GRBLinExpr dcha = 0.0;
                        GRBLinExpr suma = 0.0;
                        double M = 100000;
```

```
                        izq.AddTerm(1.0, S2[r]);
                        dcha.AddTerm(1.0, S[i, r]);
                        dcha.AddTerm(M, Y[r]);
                        for(j=0;j<inst.n;j++)
                        {
                            suma.AddTerm(inst.pt[j, i], X[j, r]);
                        }
                        model.AddConstr(izq <= (dcha + suma),
"Restriccion_4b_maquina_" + i.ToString() + "_trabajo_posicion_" + r.ToString());
                    }
                }

                //3.5 Que los trabajos empiecen en la etapa 2 una vez haya terminado
el trabajo anterior
                for (r = 0; r < inst.n - 1; r++)
                {
                    GRBLinExpr parte_izq = 0.0;
                    GRBLinExpr parte_dcha = 0.0;
                    parte_izq.AddTerm(1.0, S2[r + 1]);
                    parte_dcha.AddTerm(1.0, CT[r]);
                    model.AddConstr(parte_izq >= parte_dcha,
"Restriccion_5_trabajo_posicion_" + (1 + r.ToString()));
                }

                //3.5b Que los trabajos empiecen en la etapa 2 justo después de
terminar el trabajo anterior
                for (r = 0; r < inst.n - 1; r++)
                {
                    GRBLinExpr parte_izq = 0.0;
                    double M = 100000;
                    GRBLinExpr parte_dcha = M;
                    parte_izq.AddTerm(1.0, S2[r + 1]);
                    parte_dcha.AddTerm(1.0, CT[r]);
                    parte_dcha.AddTerm(-M, Y[r]);
                    model.AddConstr(parte_izq <= parte_dcha,
"Restriccion_5b_trabajo_posicion_" + (1 + r.ToString()));
                }

                //3.6 Que cada trabajo empiece una operación en la etapa 1 después de
que haya acabado el anterior
                for (i = 0; i < inst.m; i++)
                {
                    for (r = 0; r < inst.n - 1; r++)
                    {
                        GRBLinExpr parte_izq = 0.0;
                        GRBLinExpr parte_dcha = 0.0;
                        GRBLinExpr suma = 0.0;
                        parte_izq.AddTerm(1.0, S[i, r + 1]);
                        parte_dcha.AddTerm(1.0, S[i, r]);
                        for (j = 0; j < inst.n; j++)
                        {
                            suma.AddTerm(inst.pt[j, i], X[j, r]);
                        }
                        model.AddConstr(parte_izq == (parte_dcha + suma),
"Restriccion_6_maquina_" + i.ToString() + "_trabajo_posicion_" + r.ToString());
                    }
                }

                //3.7 Que el tiempo de inicio del primer trabajo sea igual o superior
a cero
```

```
                for (i = 0; i < inst.m; i++)
                {
                    GRBLinExpr term = 0.0;
                    term.AddTerm(1.0, S[i, 0]);
                    model.AddConstr(term, GRB.EQUAL, 0.0, "Restriccion_7_maquina_" +
i.ToString());
                }

                //3.8 Que cada trabajo esté sólo en una posición
                for (j = 0; j < inst.n; j++)
                {
                    GRBLinExpr suma = 0.0;
                    for (r = 0; r < inst.n; r++)
                    {
                        suma.AddTerm(1.0, X[j, r]);
                    }
                    double term = 1.0;
                    model.AddConstr(suma == term, "Restriccion_8_trabajo_" +
j.ToString());
                }

                //3.9 Que en cada posición sólo haya un trabajo
                for (r = 0; r < inst.n; r++)
                {
                    GRBLinExpr suma = 0.0;
                    for (j = 0; j < inst.n; j++)
                    {
                        suma.AddTerm(1.0, X[j, r]);
                    }
                    double term = 1.0;
                    model.AddConstr(suma == term, "Restriccion_9_trabajo_posicion_" +
r.ToString());
                }

                //3.10 Que las variables de decisión sean siempre positivas
                for (r = 0; r < inst.n; r++)
                {
                    //T[r]
                    GRBLinExpr tr = 0.0;
                    tr.AddTerm(1.0, T[r]);
                    model.AddConstr(tr, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10a_trabajo_posicion_" + r.ToString());
                    //E[r]
                    GRBLinExpr er = 0.0;
                    er.AddTerm(1.0, E[r]);
                    model.AddConstr(er, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10b_trabajo_posicion_" + r.ToString());
                    //CT[r]
                    GRBLinExpr ct = 0.0;
                    ct.AddTerm(1.0, CT[r]);
                    model.AddConstr(ct, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10c_trabajo_posicion_" + r.ToString());
                    //S[r]i
                    for (i = 0; i < inst.m; i++)
                    {
                        GRBLinExpr sri = 0.0;
                        sri.AddTerm(1.0, S[i, r]);
                        model.AddConstr(sri, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10d_trabajo_posicion_" + r.ToString() + "_maquina_" + i.ToString());
                    }
```

```
                    //S2[r]
                    GRBLinExpr s2 = 0.0;
                    s2.AddTerm(1.0, S2[r]);
                    model.AddConstr(s2, GRB.GREATER_EQUAL, 0.0,
"Restriccion_10e_trabajo_posicion_" + r.ToString());
                }
                //3.11 Que las variables binarias esten entre 0 y 1
                for (r = 0; r < inst.n; r++)
                {
                    for (j = 0; j < inst.n; j++)
                    {
                        GRBLinExpr xjr = 0.0;
                        xjr.AddTerm(1.0, X[j, r]);
                        model.AddConstr(xjr, GRB.GREATER_EQUAL, 0.0,
"Restriccion_11a_trabajo_" + j.ToString() + "_posicion_" + r.ToString());
                        model.AddConstr(xjr, GRB.LESS_EQUAL, 1.0,
"Restriccion_11b_trabajo_" + j.ToString() + "_posicion_" + r.ToString());
                    }
                }

                model.Update();
                model.Optimize();
                tiempo = model.Get(GRB.DoubleAttr.Runtime);
                status = model.Get(GRB.IntAttr.Status);
                model.Write("modelo_4.lp");
```

iv. Sacar por pantalla la secuencia y las variables

```
               int[] secuencia = new int[inst.n];
                if (model.Get(GRB.IntAttr.Status) == GRB.Status.INFEASIBLE)
                {
                    using StreamWriter writer = new
StreamWriter("resultados_MILP.txt");
                    writer.WriteLine(inst.datos + " Infeasible model");
                }
                else
                {
                      System.Console.WriteLine("Instancia: " +
inst.datos.ToString()+"   Modelo 4");
                    // la secuencia:

                    Console.Write("(");
                    for (r = 0; r < inst.n; r++)
                    {
                        j = -1;
                        do
                        {
                            j++;
                        } while (X[j, r].Get(GRB.DoubleAttr.X) == 0);
                        secuencia[r] = j;
                         if (r < (inst.n - 1))
                         {
                             Console.Write(j.ToString() + ",");
                         }
                         else
                         {
                             Console.Write(j.ToString() + ")");
                             Console.WriteLine(" ");
                         }
```

```
                    }

                    //Variable X[j,r]
                     for(j=0;j<inst.n;j++)
                     {
                         for(r=0;r<inst.n;r++)
                         {
                             Console.WriteLine("X_"+j.ToString()+"_"+r.ToString()+" =
"+X[j,r].Get(GRB.DoubleAttr.X));
                         }
                     }
                    //Variable Y[r]
                    for (r = 0; r < inst.n; r++)
                    {
                       Console.WriteLine("Y_" + r.ToString() + " = " +
Y[r].Get(GRB.DoubleAttr.X));
                    }
                    //Tardiness
                    for (r=0;r<inst.n;r++)
                     {
                         Console.WriteLine("T_"+r.ToString()+" =
"+T[r].Get(GRB.DoubleAttr.X));
                     }
                     //Earliness
                     for (r = 0; r < inst.n; r++)
                     {
                         Console.WriteLine("E_" + r.ToString() + " = " +
E[r].Get(GRB.DoubleAttr.X));
                     }
                     //Completion time
                     for(r=0;r<inst.n;r++)
                     {
                         Console.WriteLine("CT_"+r.ToString()+" =
"+CT[r].Get(GRB.DoubleAttr.X));
                     }
                     //S
                     for(i=0;i<inst.m;i++)
                     {
                         for(r=0;r<inst.n;r++)
                         {
                             Console.WriteLine("S_"+i.ToString()+"_"+r.ToString()+" =
"+S[i,r].Get(GRB.DoubleAttr.X));
                         }
                     }
                     //S2
                     for (r = 0; r < inst.n; r++)
                     {
                         Console.WriteLine("S2_" + r.ToString() + " = " +
S2[r].Get(GRB.DoubleAttr.X));
                     }
                }
```

v. Extraer resultados del modelo

```
                for (r = 0; r < inst.n; r++)
                 {
                     FO_modelo += T[r].Get(GRB.DoubleAttr.X) +
E[r].Get(GRB.DoubleAttr.X);
                 }
```

```
                FO = inst.calculoFO(secuencia);

            }
            catch (GRBException excep)
            {
                Console.WriteLine("Error code: " + excep.ErrorCode + ". " +
excep.Message);
                return null;
            }
            return new Resultado(tiempo, FO, FO_modelo, status);
        }
    }
}
```