Autor: Francisco Sánchez Firth

Tutor: Jose Manuel García Sánchez

Dpto. Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Sevilla, 2019

Proyecto Fin de Grado
Ingeniería de Organización Industrial

Análisis y comparativa del desempeño práctico de diversos modelos matemáticos de problemas de scheduling

Autor:
Francisco Sánchez Firth

Tutor:
José Manuel García Sánchez
Profesor titular

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

Proyecto Fin de Carrera: Análisis y comparativa del desempeño práctico de diversos modelos matemáticos de problemas de scheduling

Autor: Francisco Sánchez Firth

Tutor: José Manuel García Sánchez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Tras un periodo que ha supuesto varios meses de trabajo y esfuerzo hoy pongo fin a este trabajo con el que concluyo mi grado. Puedo afirmar con confianza que sido un periodo de desarrollo y aprendizaje tanto a nivel científico-técnico como personal.

En primer lugar, me gustaría agradecer a mis compañeros del grado por los consejos y el apoyo que me han brindado a lo largo de la elaboración de este trabajo. Además, me gustaría dar las gracias a mi tutor, José Manuel, por su valiosa ayuda sin la cual este trabajo no hubiera sido viable. Definitivamente me ha sabido guiar a lo largo de todo el proceso, aportando las herramientas y el conocimiento necesario para que el trabajo fuera interesante, ameno y acorde a lo que buscaba realizar.

También me gustaría agradecer a mis padres y amigos por sus sabios consejos y su comprensión. Siempre habéis estado ahí para mí.

Francisco Sánchez Firth

Sevilla, 2019

Resumen

Vivimos en un mundo altamente industrializado donde la cantidad de datos que manejan las empresas son cada vez mayores y donde, para conseguir mejores índices de competitividad, es fundamental conseguir que el manejo de los recursos disponibles sea lo más eficiente posible. Es en este contexto en que las técnicas de resolución de problemas de *Scheduling* cobran protagonismo como una herramienta fundamental para la asignación de dichos recursos. Dichos problemas han sido ampliamente estudiados en el campo de la optimización debido a la complejidad de los mismos y a las múltiples ventajas que presentan.

A través de este trabajo pretendo estudiar a través de la aplicación de diversas técnicas de *MILP,* la manera más conveniente de abordar una serie de problemas de *Scheduling* desde el punto de vista de eficiencia en términos computacionales. Los problemas abordados serán los de *Single Machine* y *Flow Shop*, y el modelado se realizará atendiendo a perspectivas diferentes: en función del tiempo de finalización de cada trabajo, en función de las posiciones ocupadas por los trabajos o utilizando formulaciones que combinan las dos anteriores.

Las conclusiones que se obtienen en este trabajo tienen una alta aplicabilidad en el sector de industria puesto que permiten determinar qué forma de modelar obtiene una mejor solución en cada problema, permitiendo de esta manera una mejor asignación de los recursos y, con ello, una mejora en los procesos y en la gestión.

Para la ejecución de los modelos lineales utilizaré el Software Lingo y para el manejo de los ficheros de datos un compilador de programación en C.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

C_j	Tiempo de terminación del trabajo j
Pt_{ij}	Tiempo de proceso del trabajo j en máquina i
C_{max}	*Makespan*
S_{ijk}	Tiempo de *Set up* dependiente de la secuencia en máquina i
R_m	Máquinas no relacionadas
F_m	Máquinas en taller de flujo
F_j	Flujo del trabajo j
E_j	Earliness del trabajo j
L_j	Lateness del trabajo j
T_j	Tardiness del trabajo j
O_m	Open shop
J_m	Job shop
U_j	Tardy job
P_m	Máquinas paralelas idénticas
Q_m	Máquinas paralelas relacionadas
R_m	Máquinas paralelas no relacionadas
r_j	Fecha de llegada del trabajo j
d_j	Fecha de entrega del trabajo j

1 INTRODUCCIÓN

"A plan is what, a schedule is when. It takes both a plan and a schedule to get things done."

Peter Turla

Los problemas de *Scheduling* cobran protagonismo en la industria dado que son un elemento clave para el funcionamiento de las empresas. Se tratan de problemas que organizan en tiempo de ejecución de tareas o trabajos a lo largo de un horizonte temporal, teniendo en cuenta restricciones generadas por factores del entorno. Este tipo de problemas siempre se plantean buscando la mejora de una o varias funciones objetivo (optimización mono-criterio o multi-criterio), que normalmente suelen estar asociadas directa o indirectamente relacionados a factores económicos y, como consecuencia, los resultados obtenidos se tienen en cuenta para la toma de decisiones.

Los problemas de *scheduling* pertenecen al campo científico de la optimización combinatoria y surgen en 1958 aplicados a la gestión de proyectos en el cálculo del camino crítico. Hoy en día destaca cómo sus aplicaciones se extienden a otros como la organización de máquinas en talleres de mecanizado, logística, planificación de la producción, gestión de inventarios o la mejora de sistemas operativos informáticos.

Generalmente los problemas de *scheduling* se describen mediante una serie de tareas o trabajos a ejecutar y una serie de máquinas o unidades ejecutoras que procesan los trabajos en un determinado tiempo. Esta visión da lugar, como consecuencia, a la utilización de diagramas de Gantt a modo de horizonte temporal en el que representar el progreso y disposición de los trabajos en cada máquina.

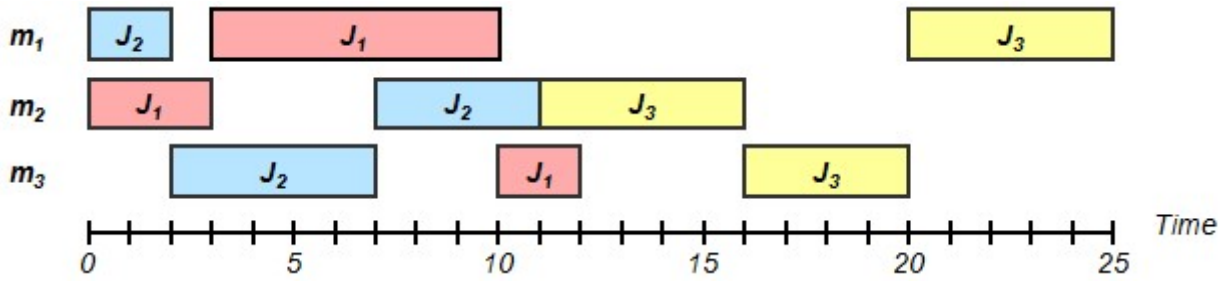


Figura 1. Ejemplo de Gantt aplicado a problema de *scheduling*

Las ventajas de aplicar scheduling en la industria son variadas, pero destacan:

- Reducción de inventarios
- Aumento en las tasas de producción
- Mejora en el cumplimiento de fechas
- Nivelamiento de la carga de trabajo

La complejidad de este tipo de problemas varía enormemente en función de las restricciones impuestas o de la función objetivo estudiada. Desde la perspectiva de la complejidad computacional se distinguen dos tipos de problemas: los polinomiales (P) y no polinomiales (NP-Hard). En el caso de los problemas P es relativamente fácil obtener una solución óptima debido a que el número de soluciones aumenta de forma polinomial al aumentar las dimensiones del problema, y la enumeración es una técnica a considerar. Sin embargo, en el caso de los NP-Hard el número de soluciones aumenta de forma no polinomial haciéndose en ocasiones inviable la obtención de óptimo en un tiempo de ejecución razonable. Es por esto por lo que para abordar problemas NP-Hard se recurre a heurísticas o metaheurísticas que permitan obtener soluciones buenas de acuerdo con los objetivos buscados, sin ser necesariamente optimas.

La programación matemática de este tipo de problemas se basa en la formulación de una o varias funciones objetivo a optimizar, y numerosas restricciones que acotan la región admisible y con ello el espectro de soluciones. Es común utilizar el modelado MILP/MINLP (mixed-integer linear programming/mixed-integer non linear programming), como la manera más cómoda de trabajar con modelos con variables enteras y no enteras, como es en este caso.

1.1. Objetivo de este trabajo

A través de este documento se busca realizar un análisis computacional que compare de forma cuantitativa la eficiencia resolutiva de diversas maneras de modelar a través de MILP una serie de problemas de programación de la producción.

Dichos problemas son:

Single machine con función objetivo de sumatorio de *completion time*: $1|\ |\ \sum_1^n C_j$

Single machine con función objetivo sumatorio de *tardiness*: $1|\ |\sum_1^n T_j$

Flowshop de permutación con función objetivo de sumatorio de *completion time*: $\boldsymbol{F_m}|$ pmru $|\ C_{max}$

1.2. Procedimiento aplicado

Para evaluar las distintas maneras de modelar los problemas partiremos de una batería de datos con valores generados de forma aleatoria. Dichos datos se incluirán en un fichero en formato lingo[1] a través de programas en lenguaje C. Posteriormente se ejecutará cada modelo y se analizarán los resultados.

Como consecuencia para la correcta comprensión de este documento será necesario tener conocimientos de lenguaje C aplicado al manejo de ficheros, modelado matemático lineal y sintaxis Lingo.

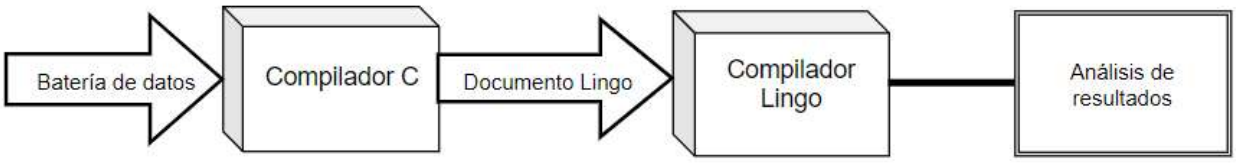


Figura 2. Diagrama secuencial del desarrollo de este documento.

Por lo general ejecutaremos cada modelo para una serie predefinida de trabajos y demás datos de entrada y esperaremos a su ejecución por parte del programa para comparar el tiempo de resolución. Excepcionalmente, si el modelo es bastante ineficiente y, como consecuencia, sus tiempos de resolución son muy elevados, se establecerá una cota temporal máxima.

Todos los modelos se ejecutarán en un ordenador con las siguientes características:

- Procesador AMD con 3.0GHz (2CPUs).
- Memoria RAM de 8192 MB.
- Memoria ROM de 250 GB.
- Fabricante: Lenovo.

[1] Software de resolución de modelos lineales.

2 PROGRAMACIÓN DE LA PRODUCCIÓN

"The success of a production depends on the attention paid to detail."

David O. Selznick

Uno de los ámbitos donde las técnicas de *scheduling* ha cobrado más protagonismo es en la programación orientada a la producción, debido a que permite la asignación de distintos recursos de una empresa a la fabricación de productos.

2.1 Elementos básicos en la programación de la producción

Máquina

Es la unidad productiva con capacidad de realizar operaciones orientadas a la fabricación de un producto. Se puede dar el caso de que varias máquinas realicen la misma operación o que realicen operaciones diferentes.

Trabajo

Se trata del elemento que va a sufrir una transformación como consecuencia de la actuación de una máquina. Dicha transformación tendrá asociada una cierta duración llamada tiempo de proceso.

Programa

Producto de la asignación de una escala temporal concreta de las máquinas de una empresa para la variación de trabajos, permitiendo de esta manera concretar el comienzo y fin de cada operación. Es usual, a modo de representación gráfica, recurrir a diagramas de Gantt para visualizar programa determinado.

Un programa será factible si cumple con el conjunto de restricciones y especificaciones del proceso productivo.

Un programa será semi-activo si, como consecuencia de la disposición de trabajos, no es posible adelantar ninguna operación sin cambiar el orden de los mismos en alguna máquina. Por lo general en este documento consideraremos únicamente programas semi-activos de manera que a través de una secuencia[2] pueda quedar definido un programa.

[2] Secuencia: orden que siguen los trabajos a la entrada de las máquinas

2.2 Clasificación de los modelos de programación de la producción[3]

Los modelos programación de la producción se diferencian de acuerdo a tres elementos:

- La función objetivo C
- Las características de los trabajos B
- Las características de las máquinas A

Generalmente cada modelo se representa con la siguiente notación: A|B|C.

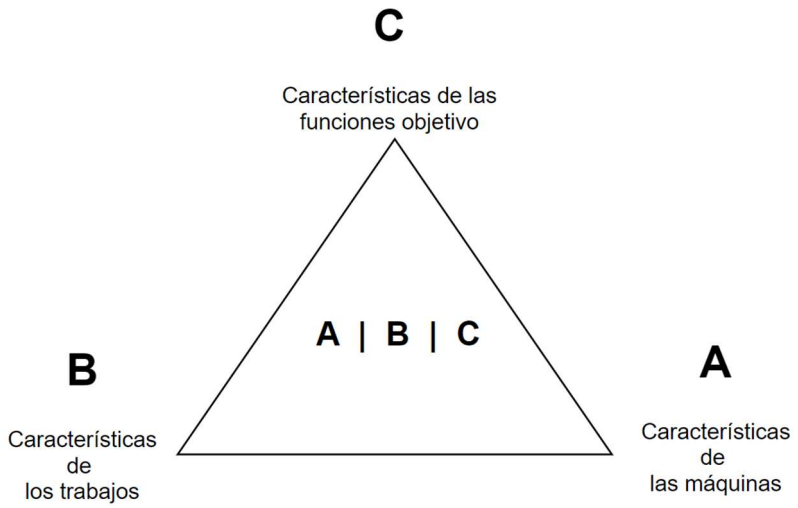


Figura 3. Mapa nomenclatura de modelos de programación de la producción.

2.2.1 Modelos según la función objetivo[4]

En este tipo de modelos siempre se minimiza la función objetivo. Dicha función puede ser de dos tipos:

No relacionadas con fechas de entrega:

- *Makespan*: $C_{max} = máx C_j$. En este caso se minimiza el mayor de los tiempos de terminación de los trabajos
- *Maximum Flowtime*: $máx\ F_j$. Se define el flujo de un trabajo como la diferencia entre el tiempo de terminación C_j y momento en el que entra en el Sistema r_j.
- *Total Completion*: $\sum_{j=1}^{n} C_j$. Suma de todos los tiempos de terminación de los trabajos.
- *Total Flowtime*: $\sum_{j=1}^{n} F_j$. Suma de todos los flujos de los trabajos.

Relacionadas con fechas de entrega:

- *Maximum Lateness*: $máx L_j$. Se define el *Lateness* como la diferencia entre el tiempo de terminación de un trabajo C_j y la fecha de entrega correspondiente d_j.
- *Maximum Tardiness*: $máx\ T_j$. Siendo el *Tardiness* $T_j = máx\{0, L_j\}$
- *Maximum Earliness*: $máx\ E_j$. Siendo el *Earliness* $E_j = máx\{0, -L_j\}$
- *Total Lateness*: $\sum_{j=1}^{n} L_j$

[3]A lo largo de este capítulo usaremos los subíndices i para referirnos a las máquinas y j para referirnos a los trabajos.

[4]Cada una de las funciones objetivo descrita tienen su forma equivalente ponderada, donde para cada trabajo se le asigna un determinado peso w_j.

- *Total Tardiness*: $\sum_{j=1}^{n} T_j$
- *Total Earliness*: $\sum_{j=1}^{n} E_j$
- *Number of tardy jobs*: $\sum_{j=1}^{n} U_j$. $U_j$vale 1 si la diferencia entre el tiempo de terminación C_j y la fecha de entrega d_j, es mayor que 0. En otro caso vale 0.

2.2.2 Modelos según las características de las máquinas

Generalmente se distinguen las siguientes maneras de disponer las máquinas:

- *Single Machine:* 1. Una única máquina se encarga de la producción.
- *Identical Parallel Machines:* P_m. Un conjunto de m máquinas procesan los trabajos. Todas las máquinas tienen la misma velocidad de procesamiento para un determinado trabajo.
- *Uniform Parallel Machines*: Q_m. Un conjunto de m máquinas con distintas velocidades de procesamiento. Se caracterizan porque el tiempo de procesamiento de una máquina determinada se puede expresar como el cociente entre el tiempo de procesamiento genérico y la velocidad de la máquina. $P_{ij} = P_j/V_i$.
- *Unrelated Parallel Machines:* R_m. En este caso cada máquina tiene un tiempo diferente de procesamiento para cada trabajo con respecto a las otras máquinas. Consecuencia de esto es que los tiempos de proceso vendrán expresados de forma matricial en lugar de vectorial (como ocurría en los casos anteriores).
- *Flowshop:* F_m. Este entorno tipo taller se caracteriza porque todos los trabajos han de ser procesados por todas las máquinas siendo necesaria una ruta determinada[5] común a todos ellos. La representación de los tiempos de proceso también es matricial.
- *Jobshop:* J_m. Al igual que en entorno Flowshop, todos los trabajos han de ser procesados por todas las máquinas. En este caso cada trabajo tiene una ruta y cada máquina una secuencia.
- *Openshop:* O_m. Es el más general y complejo de los talleres dado que no hay rutas predefinidas para los trabajos

2.2.3 Modelos según las restricciones

Si los trabajos se interrumpen mientras se procesan distinguimos:

- *Preemption non-resumable:* se pierde en todo el proceso realizado y la tarea ha de reiniciarse.
- *Preemption semi-resumable:* se pierde parte del progreso realizado.
- *Preemption resumable:* una tarea puede interrumpirse y continuarse posteriormente sin penalización

Si tenemos fecha de llegada de los trabajos y de entrega de los mismos:

- Las fechas de llegada r_j indican que los trabajos no están necesariamente disponibles en el instante de inicio.
- *Deadlines* $\bar{d}_j$. El proceso global tiene una fecha prefijada de finalización que ha de cumplirse de forma obligatoria
- *Common due date*$_j$. Existe una fecha ideal en la que los trabajos han de ser procesados. Dicha fecha no determina la admisibilidad del modelo.

[5] Ruta: orden de las máquinas para cada trabajo

Tiempos de setup[6]:

- Independientes de la secuencia s_{ij}. Cada trabajo en cada máquina tiene un cierto tiempo de *setup* que ha de cumplirse previamente a su procesamiento.
- Dependientes de la secuencia s_{ijk}. En este caso se añade un subíndice k para hacer referencia a un trabajo que se procesa posteriormente al trabajo j. En este caso se considera el tiempo de*setup* del primer trabajo como 0.

Lotes:

- Lotes en paralelo *p-batch.* Se pueden procesas varios trabajos de forma simultánea obteniendo un tiempo de proceso global equivalente al mayor de ellos.
- Lotes en serie *s-batch*. Los trabajos se disponen en serie y como consecuencia el tiempo de proceso resultante es la suma de cada tiempo de proceso individual.

Precedencia entre trabajos[7] *prec*. Se trata de un caso particular en el que un trabajo determinado no se puede procesar hasta que se finalicen otros.

Restricciones asociadas a entornos Taller (F_m, J_m, O_m):

- *Flowshop* de permutación *prmu*. Se caracteriza por tener una misma ruta para todos los trabajos.
- Sin tiempo ocioso *no-idle.* No se permite que las máquinas dejen de procesar hasta completar todos los trabajos
- Sin espera *no-wait.* Los trabajos no pueden esperar entren máquina y máquina.
- Almacenes de las máquinas *buffer.* Hay un límite de trabajos que puede tener una máquina.

[6] Los tiempos de setup pueden ser anticipatorios o no anticipatorios en función de si se pueden realizar previamente a tener el trabajo en la máquina o no. En este documento los consideraremos no anticipatorios.

[7] Es típico acompañar este tipo de restricciones con diagramas de precedencia.

3 MODELOS MATEMÁTICOS

"The achievements of an organization are the results of the combined effort of each individual."

Vince Lombardi.

Esta sección del documento tiene por objetivo definir cada modelo matemático, explicando cada grupo de restricciones, la función objetivo a optimizar, y la naturaleza de las variables. También, con el propósito de confirmar el correcto funcionamiento de dichos modelos, se incluirá al final de cada modelo los resultados asociados a la simulación de un problema de pequeña dimensión.

Los modelos matemáticos que se van a plantear son el de los sistemas *Single Machine* en sus variantes de sumatorio de tiempos de terminación y el sistema *Flow Shop* de permutación. Sus principales características son las siguientes:

- Sistema *Single Machine:*

El sistema productivo de Single Machine es aquel en el que se asignan tareas o trabajos a una única máquina. Dichos trabajos son programados de manera que uno o varios objetivos son optimizados.

Un sistema de *Single Machine* se representa de la siguiente manera:

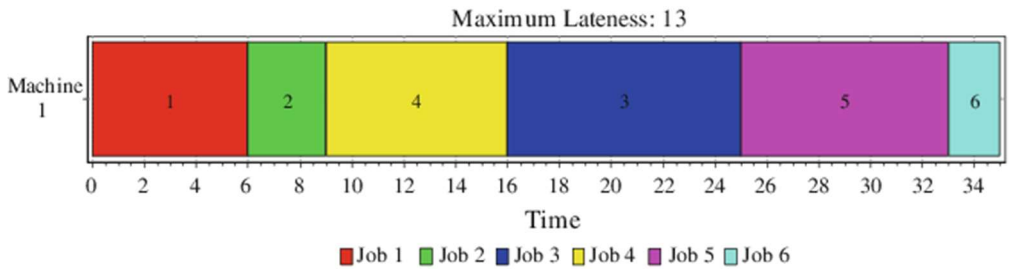


Figura 4. Representación Gantt de un sistema *Single Machine*

En este documento se analiza este problema desde la perspectiva de dos funciones objetivo: la de sumatorio de tiempos te terminación y la de sumatorio de los tardiness. Ambas explicadas en el capítulo anterior.

- Sistema *Flow Shop*:

El sistema productivo más relevante al que se le van a aplicar diversas técnicas y metodologías de modelado es el sistema *Flow Shop*, también conocido como producción en masa. Este sistema de producción, introducido por Henry Ford en 1901 supuso una revolución en la producción industrial a través del desarrollo de la cadena de montaje. Hoy en día, esta manera de fabricación se sigue utilizando dado que es ideal para productos homogéneos, fácil de automatizar y se puede aplicar a grandes volúmenes de producción.

Un sistema *Flow Shop* se puede representar de la siguiente manera:

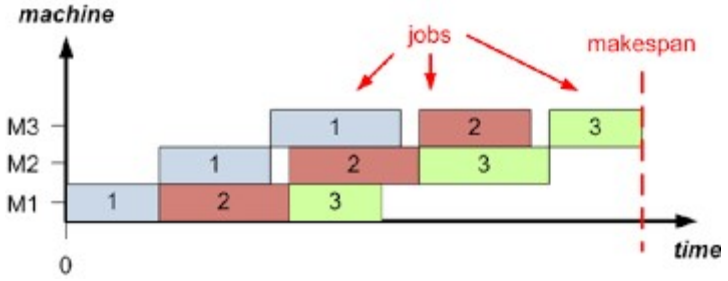


Figura 5. Representación Gantt de un sistema *Flow Shop*

Se puede apreciar que cada producto ha de completar un tiempo de proceso en cada una de las máquinas.

En este documento esta configuración de máquinas siempre se va a enfocar a minimizar el mayor de los tiempos de terminación.

3.1 Modelo A

Este modelo se basa en la posición que ocupa cada trabajo en el horizonte temporal y para ello se define la posición de terminación de un determinado trabajo j, a través del subíndice k.

3.1.1 Aplicación al problema $1|\ |\sum_1^n C_j$:

$$Min \sum_{k=1}^{l} Cp_k \quad (1)$$

$$Cp_k \geq \sum_{j=1}^{n} (p_j * x_{jk}) + Cp_{k-1} \qquad \forall k \quad (2)$$

$$\sum_{j=1}^{n} x_{jk} = 1 \qquad \forall k \quad (3)$$

$$\sum_{k=1}^{l} x_{jk} = 1 \qquad \forall j \quad (4)$$

Siendo x_{jk} la variable binaria que asigna en trabajo j a la posición k.

El resto de las variables toman valores enteros.

Lógica del modelo:

1. Se trata de la función objetivo. Minimiza el sumatorio de las posiciones de terminación de todos los trabajos.
2. Esta restricción impone que la posición de terminación de un trabajo es k, mayor o igual que la de la posición anterior más el sumatorio de los tiempos de proceso de todos los trabajos asignados en la posición k.
3. En cada posición k solo se puede asignar un trabajo j.
4. Cada trabajo j va en una posición k.

Comprobación del modelo:

Para asegurarnos que el modelo es válido y resuelve el problema de forma correcta simulamos un caso con pocos trabajos.

Para 5 trabajos

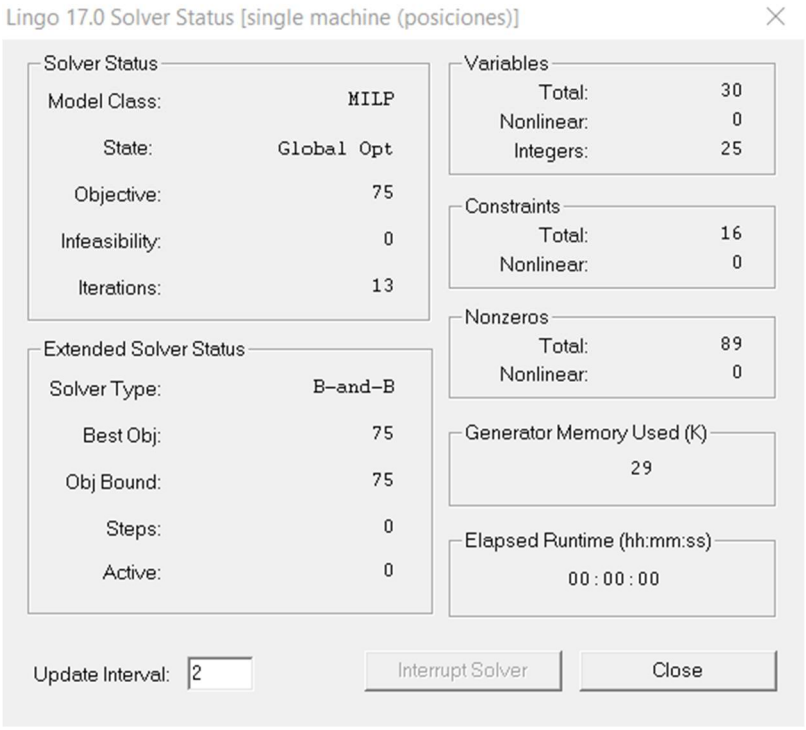


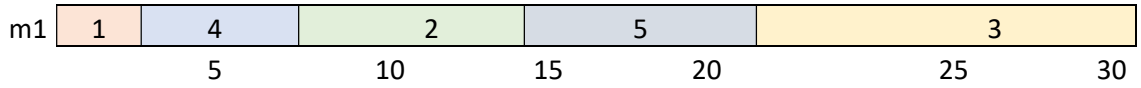
Figura 6. Resultado simulación en 1| $|\sum_1^n C_j$ con 5 trabajos

Infeasibilities:	0.000000
Extended solver steps:	0
Total solver iterations:	13
Elapsed runtime seconds:	0.07

Model Class:	MILP
Total variables:	30
Nonlinear variables:	0
Integer variables:	25
Total constraints:	16
Nonlinear constraints:	0
Total non-zeros:	89
Nonlinear non-zeros:	0

Variable	Value	Reduced Cost
PT(1)	3.000000	0.000000
PT(2)	7.000000	0.000000
PT(3)	9.000000	0.000000
PT(4)	4.000000	0.000000
PT(5)	7.000000	0.000000
CP(1)	3.000000	0.000000
CP(2)	7.000000	0.000000
CP(3)	14.00000	0.000000
CP(4)	21.00000	0.000000
CP(5)	30.00000	0.000000

PT: 3 7 9 4 7



$$C_1 = 3, C_2 = 14, C_3 = 30, C_4 = 7, C_5 = 21$$
$$\sum C_j = 75$$

Se puede apreciar que su resolución es correcta y, como consecuencia, el modelo es válido.

3.1.2 Aplicación al problema $1|\ |\sum_1^n T_j$:

A partir del modelo anterior es relativamente sencillo deducir otros, como el caso de función objetivo sumatorio de los tardiness de las posiciones.

$$Min \sum_{k=1}^{l} T_k \quad (1)$$

$$\sum_{j=1}^{n} x_{jk} = 1 \quad \forall k \quad (2)$$

$$\sum_{k=1}^{l} x_{jk} = 1 \quad \forall j \quad (3)$$

$$Cp_k \geq \sum_{j=1}^{n} (p_j * x_{jk}) + Cp_{k-1} \quad \forall k \quad (4)$$

$$T_k \geq Cp_k - d_k \forall k \quad (5)$$

$$T_k \geq 0 \quad \forall k \quad (6)$$

Siendo x_{jk} la variable binaria que asigna el trabajo j a la posición k.

El resto de las variables toman valores enteros.

Lógica del modelo:

1. Se trata de la función objetivo. Minimiza el sumatorio de los retrasos asociados a las posiciones de los trabajos.
2. En cada posición k solo se puede asignar un trabajo j.
3. Cada trabajo j va en una posición k.
4. Esta restricción impone que la posición de terminación de un trabajo es k es mayor o igual que la de la posición anterior más el sumatorio de los tiempos de proceso de todos los trabajos asignados en la posición k.
5. El retraso de la posición k es mayor o igual al tiempo de terminación de dicha posición menos la fecha de terminación del trabajo k.
6. Todos los retrasos asociados a las posiciones k han de ser mayores o iguales a 0.

Comprobación del modelo:

Para asegurarnos que el modelo es válido y resuelve el problema de forma correcta simulamos un caso con 5 trabajos.

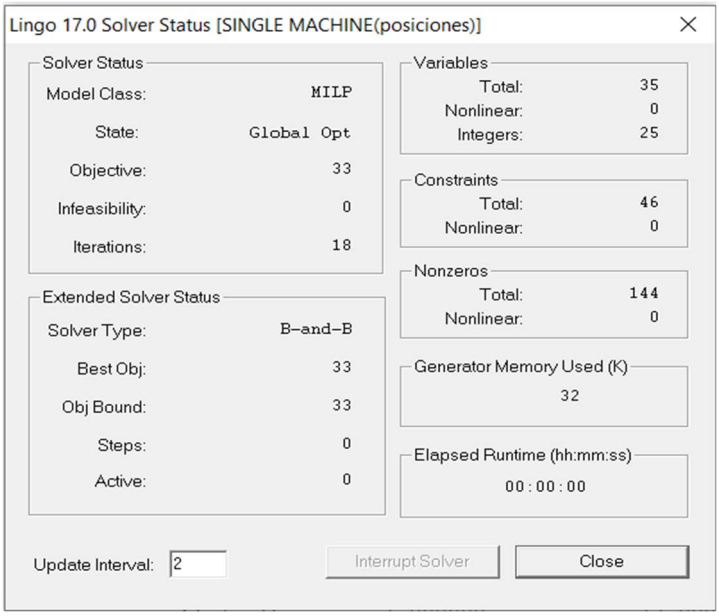


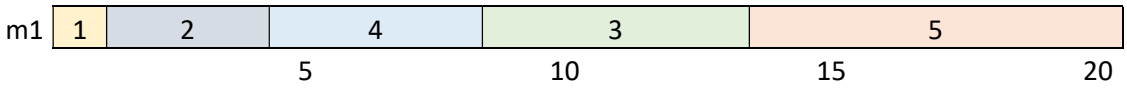
Figura 7. Resultado simulación en 1| $|\sum_1^n T_j$ con 5 trabajos

Global optimal solution found.

Objective value:	33.00000
Objective bound:	33.00000
Infeasibilities:	0.000000
Extended solver steps:	0
Total solver iterations:	18
Elapsed runtime seconds:	0.06
Model Class:	MILP
Total variables:	35
Nonlinear variables:	0
Integer variables:	25
Total constraints:	46
Nonlinear constraints:	0
Total non-zeros:	144

Variable	Value	Reduced Cost
PT(1)	1.000000	0.000000
PT(2)	3.000000	0.000000
PT(3)	5.000000	0.000000
PT(4)	4.000000	0.000000
PT(5)	7.000000	0.000000
DD(1)	1.000000	0.000000
DD(2)	2.000000	0.000000
DD(3)	3.000000	0.000000
DD(4)	5.000000	0.000000
DD(5)	2.000000	0.000000
CP(1)	1.000000	0.000000
CP(2)	4.000000	0.000000
CP(3)	8.000000	0.000000
CP(4)	13.00000	0.000000
CP(5)	20.00000	0.000000
TARD(1)	0.000000	1.000000
TARD(2)	2.000000	0.000000
TARD(3)	5.000000	0.000000
TARD(4)	8.000000	0.000000
TARD(5)	18.00000	0.000000

PT	**1**	**3**	**5**	**4**	**7**
DD	**1**	**2**	**3**	**5**	**2**



$$C_1 = 1, C_2 = 4, C_3 = 13, C_4 = 8, C_5 = 20$$
$$T_1 = 0, T_2 = 2, T_3 = 10, T_4 = 3, T_5 = 18$$
$$\sum T_j = 33$$

Se puede apreciar que su resolución es correcta y, como consecuencia, el modelo es válido.

3.1.3 Aplicación al problema $F_m|$ pmru $|\ C_{max}$:

$$MinC_{max} \quad (1)$$

$$C_{ij} \leq C_{max}\ \forall i,j \quad (2)$$

$$\sum_{k=1}^{l} x_{jk} = 1 \quad \forall j \quad (3)$$

$$\sum_{j=1}^{n} x_{jk} = 1 \quad \forall k \quad (4)$$

$$C_{1,1} \geq \sum_{j=1}^{n} pt_{1,j} * x_{1,j} \ (5)$$

$$C_{i,k} \geq C_{i,k-1} + \sum_{j=1}^{n} pt_{ij} * x_{jk} \quad \forall k > 1, i \quad (6)$$

$$C_{i,k} \geq C_{i-1,k} + \sum_{j=1}^{n} pt_{ij} * x_{jk} \quad \forall i > 1, k \quad (7)$$

La variable x_{jk} es binaria. Toma el valor 1 si el trabajo j se posiciona inmediatamente antes que el k.

El resto de variables toman valores enteros.

Lógica del modelo:

1. La función objetivo consiste en minimizar makespan.
2. El makespan valdrá como mínimo el instante en el que se procesa el último trabajo.
3. Para todo trabajo j ha de haber otro trabajo k posicionado inmediatamente después.
4. Para todo trabajo k ha de haber otro trabajo j posicionado inmediatamente antes.
5. El tiempo de terminación del primer trabajo en posicionarse en la primera máquina ha de valer como mínimo el sumatorio de todos los tiempos de proceso asociados a dicho trabajo[8].
6. El tiempo de terminación de un trabajo k en una máquina i toma un valor mayor o igual al del trabajo anterior más los tiempos de proceso de los trabajos anteriores.
7. El tiempo de terminación de un trabajo k en una máquina i toma un valor mayor o igual al tiempo de terminación del mismo trabajo en la máquina anterior más los tiempos de proceso de los trabajos anteriores.

[8] Será un único tiempo de proceso dado que es el primer trabajo en posicionarse.

Comprobación del modelo:

Para asegurarnos que el modelo es válido y resuelve el problema de forma correcta simulamos un caso con 5 trabajos y 2 máquinas.

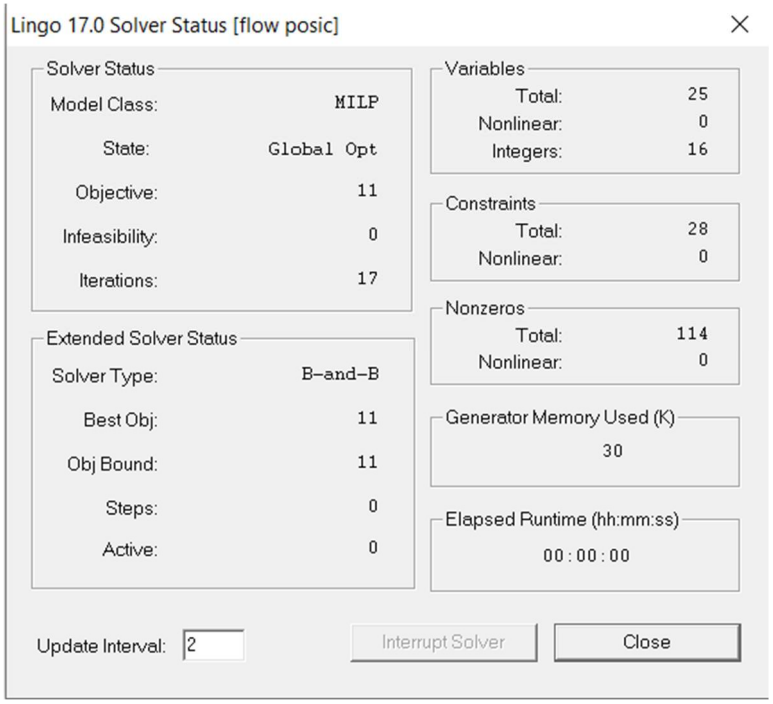


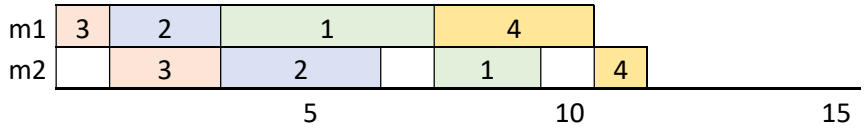
Figura 8. Resultado simulación en $F_m|\,pmru\,|\;\;C_{max}$ con 5 trabajos y 2 máquinas

Global optimal solution found.

Objective value:	11.00000
Objective bound:	11.00000
Infeasibilities:	0.000000
Extended solver steps:	0
Total solver iterations:	17
Elapsed runtime seconds:	0.07
Model Class:	MILP
Total variables:	25
Nonlinear variables:	0
Integer variables:	16
Total constraints:	28
Total non-zeros:	114

Variable	Value	Reduced Cost
N	4.000000	0.000000
M	2.000000	0.000000
CMAX	11.00000	0.000000
P(1, 1)	4.000000	0.000000
P(1, 2	2.000000	0.000000
P(2, 1)	2.000000	0.000000
P(2, 2)	3.000000	0.000000
P(3, 1)	1.000000	0.000000
P(3, 2)	2.000000	0.000000
P(4, 1)	3.000000	0.000000
P(4, 2)	1.000000	0.000000
C(1, 1)	1.000000	0.000000
C(1, 2)	3.000000	0.000000
C(2, 1)	3.000000	0.000000
C(2, 2)	6.000000	0.000000
C(3, 1)	7.000000	0.000000
C(3, 2)	9.000000	0.000000
C(4, 1)	10.00000	0.000000
C(4, 2)	11.00000	0.000000

(PT)
m1: 4 2 1 3
m2: 2 3 2 1



$$C_{max} = 11$$

Se puede apreciar que su resolución es correcta y, como consecuencia, el modelo es válido.

3.2 Modelo B

Este modelo es uno más tradicional y teórico que define un trabajo j y otro k. Cabe destacar que este modelo en sus restricciones incluye un grupo disyuntivo que establece si el trabajo j se programa antes o después del k.

3.2.1 Aplicación al problema $1|\ |\sum_1^n C_j$:

$$Min \sum_{j=1}^{n} C_j \quad (1)$$

$$C_j = t_j + pt_j \quad \forall j \quad (2)$$

$$x_{jk} + x_{kj} \geq 1 \quad \forall j,k \quad (3)$$

$$t_j - t_k \geq pt_k * x_{jk} - V * (1 - x_{jk}) \quad \forall j,k,j<k \quad (4)$$

$$t_k - t_j \geq pt_j * x_{kj} - V * (1 - x_{kj}) \quad \forall j,k,j<k \quad (5)$$

Las variables x_{jk} son binarias y toman el valor 1 si el trabajo j se posiciona antes del k.

La constante V toma un valor muy alto para hacer el modelo admisible.

Las variables t_j, t_k hacen referencia al instante de inicio de los trabajos j y k respectivamente.

Lógica del modelo:

1. Función objetivo de sumatorio de tiempos de terminación de los trabajos.
2. Define el tiempo de terminación de un trabajo j como el instante de inicio del trabajo j más su tiempo de proceso.
3. Impone que un trabajo j tenga que ir antes o después que otro k.

4 y 5. Permiten definir la diferencia de los instantes de inicio de un par de trabajos j y k, como el tiempo de proceso correspondiente multiplicado por su variable binaria de asignación.

Comprobación del modelo:

Para asegurarnos que el modelo es válido y resuelve el problema de forma correcta simulamos un caso con pocos trabajos.

Simulación con 5 trabajos:

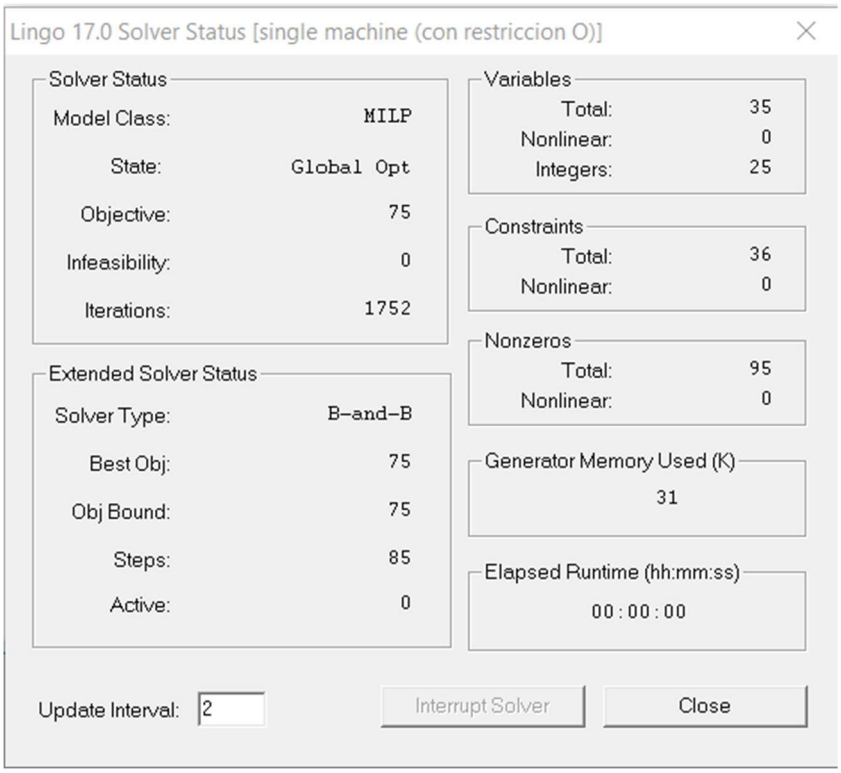


Figura 9. Resultado simulación en 1| $|\sum_1^n C_j$ con 5 trabajos

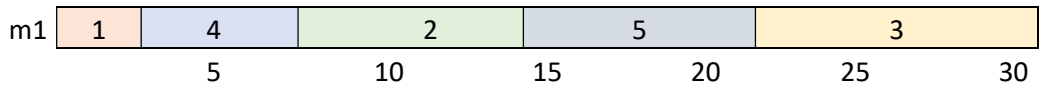
Global optimal solution found.

Objective value:	75.00000
Objective bound:	75.00000
Infeasibilities:	0.000000
Extended solver steps:	85
Total solver iterations:	1752
Elapsed runtime seconds:	0.12
Model Class:	MILP
Total variables:	35
Nonlinear variables:	0
Integer variables:	25
Total constraints:	36

Nonlinear constraints:	0
Total non-zeros:	95

Variable	Value	Reduced Cost
V	10000.00	0.000000
T(1)	0.000000	5.000000
T(2)	7.000000	0.000000
T(3)	21.00000	0.000000
T(4)	3.000000	0.000000
T(5)	14.00000	0.000000
PT(1)	3.000000	0.000000
PT(2)	7.000000	0.000000
PT(3)	9.000000	0.000000
PT(4)	4.000000	0.000000
PT(5)	7.000000	0.000000
C(1)	3.000000	0.000000
C(2)	14.00000	0.000000
C(3)	30.00000	0.000000
C(4)	7.000000	0.000000
C(5)	21.00000	0.000000

PT: 3 7 9 4 7



$$C_1 = 3, C_2 = 14, C_3 = 30, C_4 = 7, C_5 = 21$$
$$\sum C_j = 75$$

Se puede apreciar que su resolución es correcta y, como consecuencia, el modelo es válido.

3.2.2 Aplicación al problema $1|\ |\sum_1^n T_j$:

A partir de este modelo se deduce el análogo con función objetivo sumatorio de tardiness:

$$Min \sum_{j=1}^{n} T_j \quad (1)$$

$$C_j = t_j + pt_j \quad \forall j \quad (2)$$

$$x_{jk} + x_{kj} \geq 1 \quad \forall j,k \quad (3)$$

$$t_j - t_k \geq pt_k * x_{jk} - V * (1 - x_{jk}) \quad \forall j,k,j<k \quad (4)$$

$$t_k - t_j \geq pt_j * x_{kj} - V * (1 - x_{kj}) \quad \forall j,k,j<k \quad (5)$$

$$T_j \geq C_j - d_j \forall j \quad (6)$$

$$T_j \geq 0 \quad \forall k \quad (7)$$

Las variables x_{jk} son binarias toman el valor 1 si el trabajo j se posiciona antes del k.

La constante V toma un valor muy alto para hacer el modelo admisible.

Las variables t_j, t_k hacen referencia al instante de inicio de los trabajos j y k respectivamente.

Lógica del modelo:

1. Función objetivo de sumatorio de los retrasos en la terminación de trabajos.
2. Define el tiempo de terminación de un trabajo j como el instante de inicio del trabajo j más su tiempo de proceso.
3. Impone que un trabajo j tenga que ir antes o después que otro k.

4 y 5. Permiten definir la diferencia de los instantes de inicio de un par de trabajos j y k, como el tiempo de proceso correspondiente multiplicado por su variable binaria de asignación.

6. Se define el retraso en la terminación del trabajo j como la diferencia entre su tiempo de terminación y la fecha de entrega.
7. Impone que todos los retrasos han de ser mayores o iguales a 0.

Comprobación del modelo:

Para asegurarnos que el modelo es válido y resuelve el problema de forma correcta simulamos un caso con pocos trabajos.

Simulación con 5 trabajos:

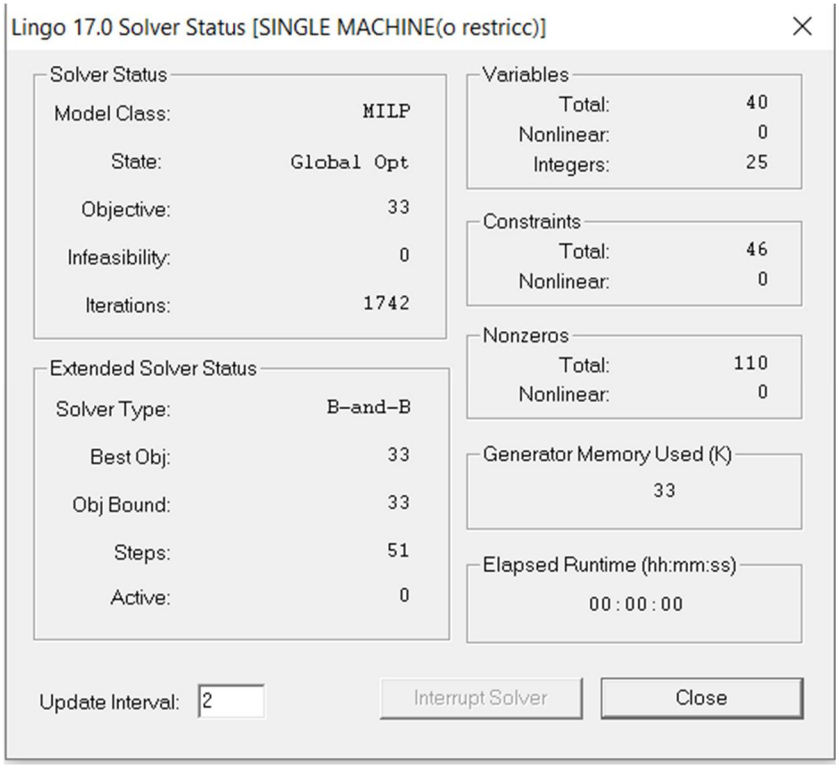


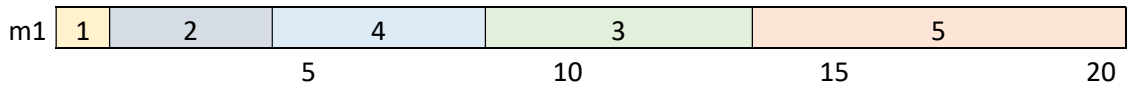
Figura 10. Resultado simulación en 1| $|\sum_1^n T_j$ con 5 trabajos

Global optimal solution found.

Objective value:	33.00000
Objective bound:	33.00000
Infeasibilities:	0.000000
Extended solver steps:	51
Total solver iterations:	1742
Elapsed runtime seconds:	0.16
Model Class:	MILP
Total variables:	40
Nonlinear variables:	0
Integer variables:	25
Total constraints:	46
Nonlinear constraints:	0
Total non-zeros:	110

Variable	Value	Reduced Cost
V	10000.00	0.000000
T(1)	0.000000	5.000000
T(2)	1.000000	0.000000
T(3)	8.000000	0.000000
T(4)	4.000000	0.000000
T(5)	13.00000	0.000000
PT(1)	1.000000	0.000000
PT(2)	3.000000	0.000000
PT(3)	5.000000	0.000000
PT(4)	4.000000	0.000000
PT(5)	7.000000	0.000000
C(1)	1.000000	0.000000
C(2)	4.000000	0.000000
C(3)	13.00000	0.000000
C(4)	8.000000	0.000000
C(5)	20.00000	0.000000
DD(1)	1.000000	0.000000
DD(2)	2.000000	0.000000
DD(3)	3.000000	0.000000
DD(4)	5.000000	0.000000
DD(5)	2.000000	0.000000
TARD(1)	0.000000	0.000000
TARD(2)	2.000000	0.000000
TARD(3)	10.00000	0.000000
TARD(4)	3.000000	0.000000
TARD(5)	18.00000	0.000000

PT	**1**	**3**	**5**	**4**	**7**
DD	**1**	**2**	**3**	**5**	**2**



$$C_1 = 1, C_2 = 4, C_3 = 13, C_4 = 8, C_5 = 20$$
$$T_1 = 0, T_2 = 2, T_3 = 10, T_4 = 3, T_5 = 18$$
$$\sum T_j = 33$$

Se puede apreciar que su resolución es correcta y, como consecuencia, el modelo es válido.

3.2.3 Aplicación al problema F_m| pmru | C_{max}:

$$Min\ C_{max} \qquad (1)$$

$$C_{M,j} \leq C_{max} \qquad \forall j \qquad (2)$$

$$C_{i,j} = t_{i,j} + pt_{i,j} \qquad \forall i,j \qquad (3)$$

$$x_{j,k} + x_{k,j} = 1 \qquad \forall j,k \qquad (4)$$

$$t_{i,j} + pt_{i,j} \leq t_{i+1,j} \qquad \forall\ i < M, i+1 \qquad (5)$$

$$t_{i,j} + pt_{i,j} \leq t_{i,k} + v * (1 - x_{j,k}) \qquad \forall i,j,k \qquad (6)$$

Las variables x_{jk} son binarias y toman el valor 1 si el trabajo j se sitúa antes del k.

La constante V toma un valor muy alto para hacer el modelo admisible.

Las variables t_j, t_k hacen referencia al instante de inicio de los trabajos j y k respectivamente.

La constante M hace referencia al total de máquinas.

Lógica del modelo:

1. Función objetivo: minimizar el makespan.
2. Se impone que el makespan es mayor o igual a todos los tiempos de terminación de la última máquina.
3. Se define el tiempo de terminación de un trabajo j en una máquina i como la suma del instante de inicio más temprano posible y su tiempo de proceso.
4. Un trabajo j puede estar antes o después de otro k, pero no ambos de forma simultánea.
5. El instante de inicio de un trabajo j es mayor o igual al instante de inicio en la máquina anterior más el tiempo de proceso en la máquina anterior.
6. El instante de inicio de un trabajo k en una máquina i es mayor o igual al instante de inicio de un trabajo j en la misma máquina más el tiempo de proceso del trabajo j en dicha máquina, siempre que j se sitúe antes de k.

Comprobación del modelo:

Para asegurarnos que el modelo es válido y resuelve el problema de forma correcta simulamos un caso con pocos trabajos. En este caso simularemos 4 trabajos en 2 máquinas.

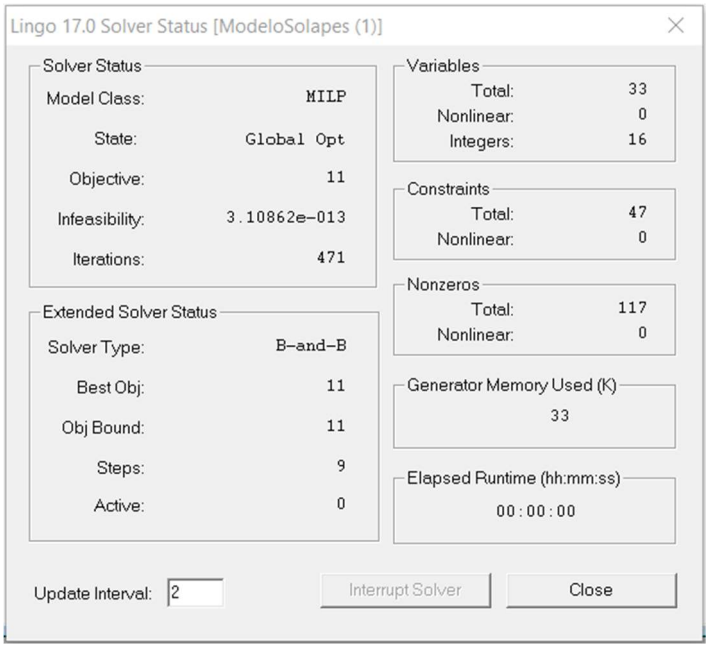


Figura 11. Resultado simulación en $F_m|$ pmru $|\ C_{max}$: con 5 trabajos y 2 máquinas

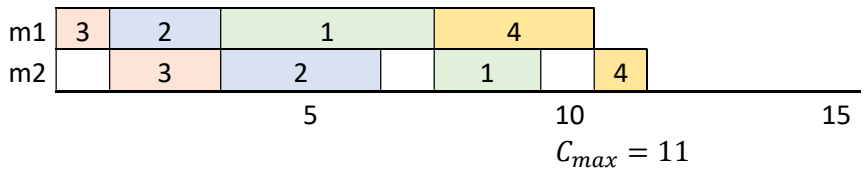
Global optimal solution found.

Objective value:	11.00000
Objective bound:	11.00000
Infeasibilities:	0.000000
Extended solver steps:	9
Total solver iterations:	471
Elapsed runtime seconds:	0.1
Model Class:	MILP
Total variables:	33
Nonlinear variables:	0
Integer variables:	16
Total constraints:	47
Nonlinear constraints:	0
Total non-zeros:	117
Nonlinear non-zeros:	0

Variable	Value	Reduced Cost
N	4.000000	0.000000
M	2.000000	0.000000

V	10000.00	0.000000
CMAX	11.00000	0.000000
C(1, 1)	5.000000	0.000000
C(1, 2)	7.000000	0.000000
C(1, 3)	1.000000	0.000000
C(1, 4)	10.00000	0.000000
C(2, 1)	7.000000	0.000000
C(2, 2)	10.00000	0.000000
C(2, 3)	5.000000	0.000000
C(2, 4)	11.00000	0.000000
PT(1, 1)	4.000000	0.000000
PT(1, 2)	2.000000	0.000000
PT(1, 3)	1.000000	0.000000
PT(1, 4)	3.000000	0.000000
PT(2, 1)	2.000000	0.000000
PT(2, 2)	3.000000	0.000000
PT(2, 3)	2.000000	0.000000
PT(2, 4)	1.000000	0.000000
T(1, 1)	1.000000	0.000000
T(1, 2)	5.000000	0.000000
T(1, 3)	0.000000	1.000000
T(1, 4)	7.000000	0.000000
T(2, 1)	5.000000	0.000000
T(2, 2)	7.000000	0.000000
T(2, 3)	3.000000	0.000000
T(2, 4)	10.00000	0.000000

(PT)
m1: 4 2 1 3
m2: 2 3 2 1



$C_{max} = 11$

Se puede apreciar que su resolución es correcta y, como consecuencia, el modelo es válido.

3.3 Modelo C

Este modelo es una variante del anterior que trabaja con un trabajo inicial con tiempo de proceso de 0 y cuyas restricciones definen que un trabajo determinado solo tiene un antecesor a lo sumo y como mínimo un predecesor.

3.3.1 Aplicación al problema $1|\ |\sum_1^n C_j$:

$$Min: \sum_{j=1}^{n} C_j \quad (1)$$

$$\sum_{j=1}^{n} x_{k,j} \leq 1 \qquad \forall k, j \neq k \quad (2)$$

$$x_{k,1} = 0 \qquad \forall k \quad (3)$$

$$\sum_{k=1}^{l} x_{kj} \geq 1 \qquad \forall j, j \neq k, j > 1 \quad (4)$$

$$C_j \geq t_j + pt_j \forall j \quad (5)$$

$$t_k \geq C_j \ - V * (1 - x_{jk}) \forall j, k, j \neq k \quad (6)$$

La variable binaria $x_{k,j}$ toma valor 1 si el trabajo k se sitúa inmediatamente antes que el trabajo j.En caso contrario toma valor 0.

El resto de las variables son enteras.

Lógica del modelo:

1. Función objetivo. Minimizar sumatorio de tiempos de terminación de cada trabajo.
2. Cada trabajo k puede tener a lo sumo otro posterior j.
3. El primer trabajo no tiene antecesor.
4. Cada trabajo j> 1 (no el primero) tiene como mínimo otro situado anteriormente.
5. Define el tiempo de terminación como un valor igual o superior al instante en que se puede iniciar dicho trabajo más el tiempo de proceso asociado a dicho trabajo.
6. El instante más temprano en el que se puede iniciar un trabajo es mayor o igual al tiempo de terminación del trabajo anterior.

Comprobación del modelo:

Para asegurarnos que el modelo es válido y resuelve el problema de forma correcta simulamos un caso con pocos trabajos.

Simulación con 5 trabajos[9]

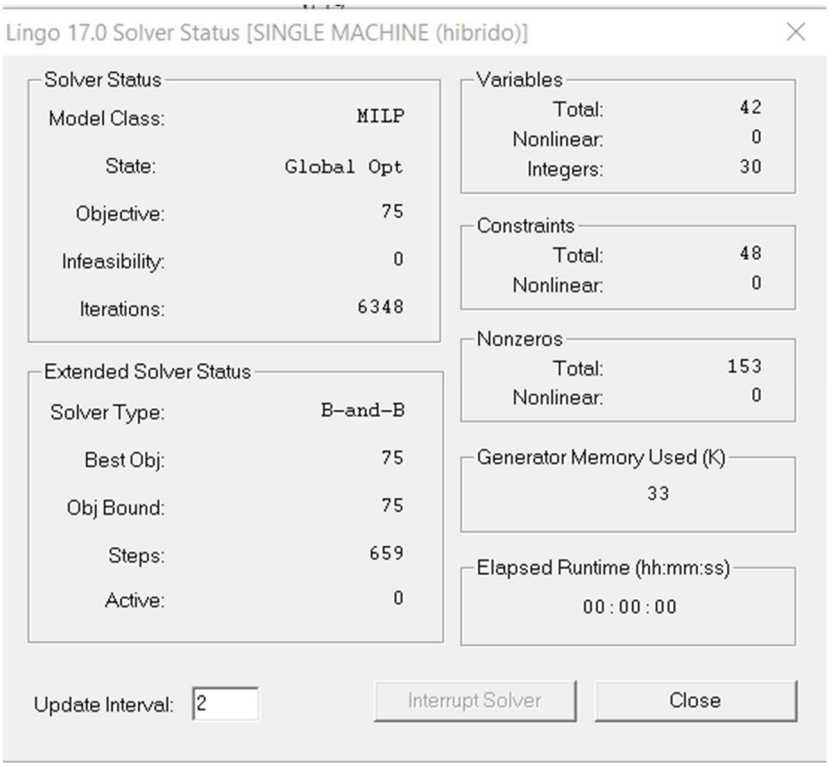


Figura 12. Resultado simulación en 1| $|\sum_1^n C_j$ con 5 trabajos

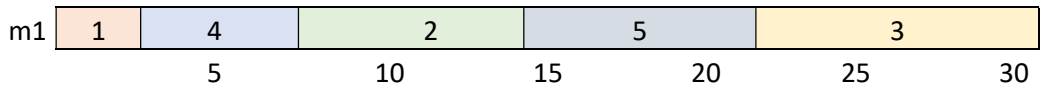
Global optimal solution found.

Objective value:	75.00000
Objective bound:	75.00000
Infeasibilities:	0.000000
Extended solver steps:	659
Total solver iterations:	6348
Elapsed runtime seconds:	0.84
Model Class:	MILP
Total variables:	42
Nonlinear variables:	0
Integer variables:	30
Total constraints:	48
Nonlinear constraints:	0

[9] Aunque la simulación se realice para 5 trabajos hay que añadir uno extra al inicio con tiempo de proceso nulo.

Total non-zeros:		153	
Variable	Value	Reduced Cost	
	V	10000.00	0.000000
	T(1)	0.000000	6.000000
	T(2)	0.000000	0.000000
	T(3)	14.00000	0.000000
	T(4)	21.00000	0.000000
	T(5)	3.000000	0.000000
	T(6)	7.000000	0.000000
	PT(1)	0.000000	0.000000
	PT(2)	3.000000	0.000000
	PT(3)	7.000000	0.000000
	PT(4)	9.000000	0.000000
	PT(5)	4.000000	0.000000
	PT(6)	7.000000	0.000000
	C(1)	0.000000	0.000000
	C(2)	3.000000	0.000000
	C(3)	21.00000	0.000000
	C(4)	30.00000	0.000000
	C(5)	7.000000	0.000000
	C(6)	14.00000	0.000000

PT: 0 3 7 9 4 7



$$C_1 = 0, C_2 = 3, C_3 = 14, C_4 = 30, C_5 = 7, C_6 = 21$$
$$\sum C_j = 75$$

Se puede apreciar que para simular 5 trabajos hay que añadir un sexto al inicio con tiempo de proceso nulo. El modelo resuelve 5 trabajos sin problemas y en un tiempo instantáneo.

3.3.2 Aplicación al problema $1|\ |\sum_1^n T_j$:

$$Min: \sum_{j=1}^{n} T_j \quad (1)$$

$$\sum_{j=1}^{n} x_{k,j} \leq 1 \qquad \forall k, j \neq k \quad (2)$$

$$x_{k,1} = 0 \qquad \forall k \quad (3)$$

$$\sum_{k=1}^{l} x_{kj} \geq 1 \qquad \forall j, j \neq k, j > 1 \quad (4)$$

$$C_j \geq t_j + pt_j \forall j \qquad (5)$$

$$t_k \geq C_j \ - V * (1 - x_{jk}) \forall j, k, j \neq k \quad (6)$$

$$T_j = \ C_j - d_j \quad \forall j \quad (7)$$

$$T_j \geq 0 \quad \forall j \quad (8)$$

La variable binaria $x_{k,j}$ toma valor 1 si el trabajo k se sitúa inmediatamente antes que el trabajo j.En caso contrario toma valor 0.

El resto de las variables son enteras.

Lógica del modelo:

1. Función objetivo. Minimizar sumatorio de tiempos de terminación de cada trabajo.
2. Cada trabajo k puede tener a lo sumo otro posterior j.
3. El primer trabajo no tiene antecesor.
4. Cada trabajo j > 1 (no el primero) tiene como mínimo otro situado anteriormente.
5. Define el tiempo de terminación como un valor igual o superior al instante en que se puede iniciar dicho trabajo más el tiempo de proceso asociado a dicho trabajo.
6. El instante más temprano en el que se puede iniciar un trabajo es mayor o igual al tiempo de terminación del trabajo anterior.
7. La tardanza en la terminación de un trabajo es igual al tiempo de terminación menos la fecha de entrega.
8. Toda tardanza en la terminación de un trabajo ha de ser mayor o igual a 0.

Simulación con 5 trabajos:

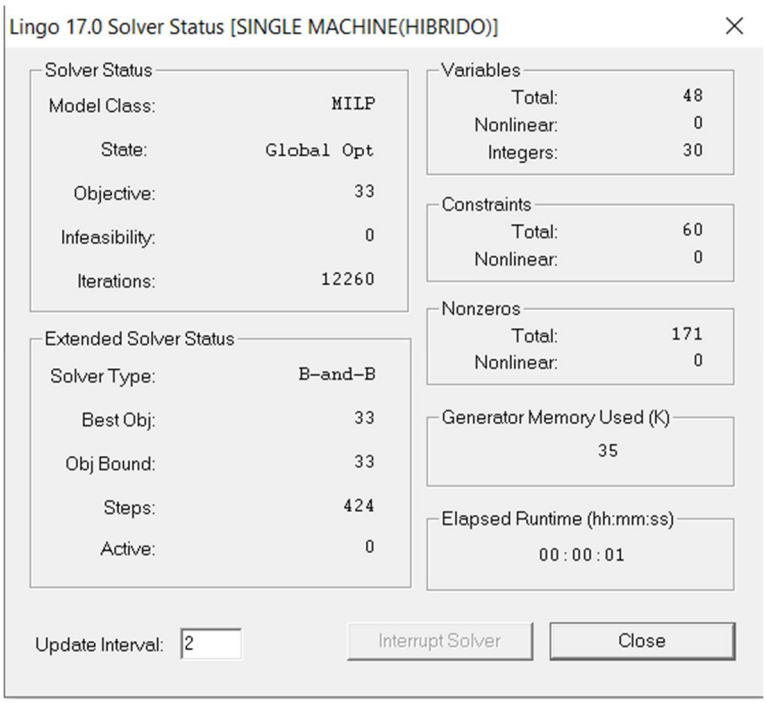


Figura 13. Resultado simulación en 1| $|\sum_1^n T_j$ con 5 trabajos

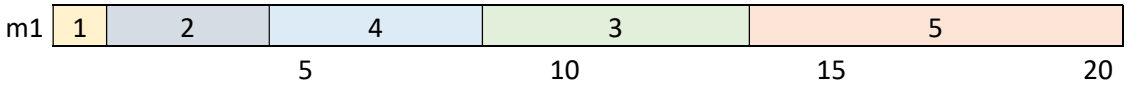
Global optimal solution found.

Objective value:	33.00000
Objective bound:	33.00000
Infeasibilities:	0.000000
Extended solver steps:	424
Total solver iterations:	12260
Elapsed runtime seconds:	1.06
Model Class:	MILP
Total variables:	48
Nonlinear variables:	0
Integer variables:	30
Total constraints:	60
Nonlinear constraints:	0
Total non-zeros:	171
Nonlinear nonzeros:	0

Variable	Value	Reduced Cost
V	10000.00	0.000000
T(1)	0.000000	6.000000
T(2)	0.000000	0.000000
T(3)	1.000000	0.000000
T(4)	8.000000	0.000000

T(5)	4.000000	0.000000
T(6)	13.00000	0.000000
PT(1)	0.000000	0.000000
PT(2)	1.000000	0.000000
PT(3)	3.000000	0.000000
PT(4)	5.000000	0.000000
PT(5)	4.000000	0.000000
PT(6)	7.000000	0.000000
C(1)	0.000000	0.000000
C(2)	1.000000	0.000000
C(3)	4.000000	0.000000
C(4)	13.00000	0.000000
C(5)	8.000000	0.000000
C(6)	20.00000	0.000000
DD(1)	0.000000	0.000000
DD(2)	1.000000	0.000000
DD(3)	2.000000	0.000000
DD(4)	3.000000	0.000000
DD(5)	5.000000	0.000000
DD(6)	2.000000	0.000000
TARD(1)	0.000000	0.000000
TARD(2)	0.000000	0.000000
TARD(3)	2.000000	0.000000
TARD(4)	10.00000	0.000000
TARD(5)	3.000000	0.000000
TARD(6)	18.00000	0.000000

PT	**1**	**3**	**5**	**4**	**7**
DD	**1**	**2**	**3**	**5**	**2**



$$C_1 = 1, C_2 = 4, C_3 = 13, C_4 = 8, C_5 = 20$$

$$T_1 = 0, T_2 = 2, T_3 = 10, T_4 = 3, T_5 = 18$$

$$\sum T_j = 33$$

Se puede apreciar que se resuelve correctamente.

4 IMPLEMENTACIÓN DE LOS MODELOS

"It's important to have a sound idea, but the really important thing is the implementation."
Wilbur Ross

Este capítulo tiene por objetivo detallar el proceso que comprende transformar el modelo lineal en lenguaje matemático a un modelo Lingo con el que se puede experimentar. Para ello, y con el fin de facilitar dicha experimentación, se establece un programa C por cada modelo que, tomando una serie de datos de entrada referido a los trabajos (y máquinas), devuelve el modelo Lingo preparado para su simulación. A lo largo de esta sección incluiré todos los modelos Lingo y un modelo de programa C. El resto de programas C, debido a su extensión, se recogerán en el anexo al final del documento

Cada código C utiliza una serie de datos recogidos en forma de batería, que siempre son leídos de forma lineal de izquierda a derecha. En el caso de los problemas de *Single Machine* únicamente se necesitan los datos de los tiempos de proceso, mientras que en el caso de los *Flow Shop* hará falta un valor previo que indique el número de máquinas. A partir de dicho valor se podrá dividir el número de tiempos de proceso entre el número de máquinas para determinar el número de trabajos.

Ejemplo de estructura de la batería de datos para *Single Machine:*

23 33 12 56 65 77 54 12 56 22 95…. ⟶ Tiempos de proceso.

Ejemplo de estructura de la batería de datos para *Flow Shop:*

5 23 33 12 56 65 77 54 12 56 22 95…. ⟶ Tiempos de proceso

Número de máquinas

Las baterías de datos usados en los modelos Flow Shop pertenecen a la página web del Prof. Éric Taillard[10].

[10] http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

Procedimiento para el manejo de ficheros en C:

De cara al manejo de ficheros, en líneas generales, el procedimiento consiste en, a través de unos datos de entrada, también en un fichero, crear un array que los almacene y permita trabajar con ellos con relativa facilidad. Para ello vamos a utilizar principalmente las siguientes funciones:

- fopen: permite almacenar en una variable el contenido de un fichero y la forma en la que se va a tratar dicho fichero.
- fscanf: esta función lee un carácter de un fichero y lo almacena en una variable. Esto lo hace ideal para de forma secuencial, a través de una estructura en bucle, almacenar cada dato en cada una de las posiciones de un array previamente definido.
- fprintf: su funcionamiento es análogo al de la función fscanf, solo que, en lugar de almacenar variable a variable información de un fichero, la escribe.
- fwrite: permite escribir en un fichero strings predefinidos y almacenados en variables tipo Char. Esta función es ideal para trabajar con la parte de los modelos que no varía en función del número de datos de entrada.
- rewind: permite volver al inicio de un fichero que ya ha sido leído.
- fclose: libera la memoria de la variable donde se haya almacenado el fichero.

Las librerías de funciones utilizadas son: schedule.h, stdio.h, stdlib.h y string.h. De todas ellas, la única que no es de uso común es schedule.h, desarrollada por personal de la ETSI para, entre otros, trabajar cómodamente con vectores y simplificar operaciones de ordenación.

Sintaxis y estructura general en Lingo:

Lingo es un paquete de software para programación lineal, programación de enteros, programación no lineal, programación estocástica y optimización global. La estructura general de Lingo comprende los siguientes apartados:

- Model/Endmodel: Sirve para delimitar el inicio y fin del programa.
- Sets/Endsets: Dentro de este elemento se describen las variables de nuestro modelo y los subconjuntos a los que pertenecen. Una forma intuitiva de comprender cómo se estructura este campo es a partir de los subíndices con los que se modelan las variables.
- Data/Enddata: Aquí se sitúa la información en forma de datos numéricos que cobran determinadas variables de nuestro modelo. Dentro de este campo irán, entre otros, los tiempos de proceso, las fechas de entrega o los tiempos de setup.
- Conjunto de restricciones y función objetivo.

La sintaxis que utiliza Lingo es relativamente sencilla dado que es similar a la utilizada matemáticamente. Sin embargo, cabe destacar:

- Símbolo “para todo” (∀): En lingo se representa por @for.
- Símbolo “sumatorio” (Σ): En lingo se representa por @sum.
- Símbolo “tal que” (:): En lingo se representa por |.

- Operadores lógicos:

Operadores Lógicos[11]	Valor devuelto
#NOT#	Verdadero si el operador a la derecha es falso.
#EQ#	Verdadero si el operador a la izquierda es igual al de la derecha.
#NE#	Verdadero si el operador a la izquierda no es igual al de la derecha.
#GT#	Verdadero si el operador a la izquierda es estrictamente mayor al de la derecha.
#GE#	Verdadero si el operador a la izquierda es mayor o igual al de la derecha.
#LT#	Verdadero si el operador a la derecha es estrictamente mayor al de la izquierda.
#LE#	Verdadero si el operador a la derecha es mayor o igual al de la izquierda.
#AND#	Verdadero si ambos operadores son verdaderos.
#OR#	Verdadero si uno de los operadores es verdadero.

Tabla 1. Explicación operadores lógicos de Lingo.

- Naturaleza de las variables:

Tipo de variable	Sintaxis lingo
Libre	@Free
Entera	@Gin
Binaria	@Bin

Tabla 2. Sintaxis asociada a la definición de variables en Lingo.

[11] Los operadores lógicos recogidos en la tabla irán asociados a los elementos @for y @sum. En el resto de elementos de las restricciones se utilizará la notación matemática estándar en caso de que se quiera expresar un operador lógico (=, >=,<=,…).

4.1 Modelo A

4.1.1 Código en C para1| $|\sum_1^n C_j$:

A continuación, se detalla el código en C que permite, a partir de un conjunto de datos de entrada que hagan referencia a los tiempos de proceso de los trabajos, generar un código ejecutable Lingo.

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
intmain()
{
   FILE *f;  //Declara la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
if(f==NULL)//si no se puede abrir, print no se puede
   {
printf("no se puede abrir el fichero\n\n");
   }
else// se abre correctamente print bien
   {
printf("Fichero abierto correctamente\n\n");
}
   int z,w;
   int i=0;
   int j=0;
int cont=0;
  while (!feof(f))
{
fscanf (f, "%d", &z);
printf ("Extraido:%d --", z);
cont=cont+1;
}
      printf("\n\nNumero de dato leidos: %d\n\n",cont);
rewind(f);
```

Conjunto de librerías utilizadas.

Verificación de que el programa reconoce el fichero de entrada.

Se genera un contador que contabiliza el número de elementos de la batería de datos.

```
cont= cont;
VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
while (!feof(f))
      {
      fscanf (f, "%d", &z);
      a[i]=z;
i=i+1;
      }
print_vector(a,cont);
 rewind(f);
////////////////////////////////////////
char parte1[] = "MODEL: \nSETS: \njob/1..";
char parte2[] = "/:pt;\n";
char parte3[] = "pos/1..";
char parte4[] = "/: cp;\n";
char parte5[] = "orden(job,pos):x; \nENDSETS\n DATA:\n pt=";
char parte6[] = ";\nENDDATA \nmin=@sum(pos(k): cp(k)); \n@for(pos(k):@sum(job(j): x(j,k)) =1);
@for(job(j):@sum(pos(k): x(j,k)) =1); \n@for(pos(k)|k#GT#1:cp(k) >= @sum(job(j): pt(j)*x(j,k)) + cp(k-1));
\ncp(1) >= @sum(job(j): pt(j)*x(j,1)); \n@for(job(j):@for(pos(k):@BIN(x(j,k))));";
FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );
if(fichero== NULL)
  {
  printf("\nERROR fichero lingo no se puede crear");
  }
else
  {
  printf("\nSe puede crear el fichero lingo");
}
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
fprintf (fichero, "%d", cont);
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
fwrite(parte3 , 1 , sizeof(parte3) , fichero);
fprintf (fichero, "%d ",cont);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);
fwrite(parte5 , 1 , sizeof(parte5) , fichero);
for (i = 0; i <cont; i++)
   fprintf (fichero, "%d ", a[i]);
```

Se genera un vector con una dimensión equivalente a los elementos de la batería de datos.

Se almacena en cada posición del vector un elemento de la batería de datos.

Se almacena en variables tipo char las partes del código Lingo que no dependen de los datos de entrada.

Se escribe en el fichero de salida los elementos correspondientes para generar el código deseado.

```
fwrite(parte6 , 1 , sizeof(parte6) , fichero);
fclose(fichero);
return 0;
}
```

4.1.2 Código Lingo para $1||\sum_1^n C_j$:

A partir del código C anterior se puede generar este modelo:

```
MODEL:
SETS:
job/1..3/: pt; !j;
pos/1..3/: cp;  !k;
orden(job,pos):x; !j,k;
ENDSETS

DATA:
pt= 1 2 3;
ENDDATA
min=@sum(pos(k): cp(k)); (1)
@for(pos(k):   @sum(job(j): x(j,k)) =1);  (2)
@for(job(j):   @sum(pos(k): x(j,k)) =1);   (3)
@for(pos(k)|k#GT#1:cp(k)>=@sum(job(j):pt(j)*x(j,k))+cp(k-1)); (4)
cp(1)  >=  @sum(job(j): pt(j)*x(j,1));   (4)
@for(job(j):@for(pos(k): @BIN(x(j,k))));
END
```

En este apartado se definen las variables del modelo. Cada grupo de variables pertenecerá a un conjunto u otro según los subíndices que presenten.

Datos de entrada del modelo Lingo.

Función objetivo, restricciones.

Naturaleza de las variables.

Este modelo de Lingo se obtiene a partir del siguiente modelo matemático:

$$Min \sum_{k=1}^{l} Cp_k \quad (1)$$

$$Cp_k \geq \sum_{j=1}^{n} (p_j * x_{jk}) + Cp_{k-1} \quad \forall k \quad (4)$$

$$\sum_{j=1}^{n} x_{jk} = 1 \quad \forall k \quad (2)$$

$$\sum_{k=1}^{l} x_{jk} = 1 \quad \forall j \quad (3)$$

4.1.3 Código Lingo para $1||\sum_1^n T_j$:

Para la función objetivo sumatorio de *tardiness*:

```
MODEL:

SETS:
job/1..10/: pt, dd; !j;
pos/1..10/: cp,tard;  !k;
orden(job,pos):x; !j,k;
ENDSETS

DATA:
pt=1 3 5 4 7 5 3 6 1 5;
dd=1 2 3 5 2 3 6 8 20 15;
ENDDATA

min=@sum(pos(k): tard(k));  (1)
@for(pos(k):           @sum(job(j): x(j,k)) =1); (2)
@for(job(j):           @sum(pos(k): x(j,k)) =1);  (3)
@for(pos(k)|k#GT#1:    cp(k)  >=  @sum(job(j): pt(j)*x(j,k)) + cp(k-1));  (4)
cp(1)  >=  @sum(job(j): pt(j)*x(j,1));  (4)
@for(pos(k): tard(k)>=cp(k)-dd(k)));  (5)
@for(pos(k): tard(k)>=0); (6)
@for(job(j):@for(pos(k):      @BIN(x(j,k)));

END
```

Este modelo de Lingo se obtiene a partir del siguiente modelo matemático:

$$Min \sum_{k=1}^{l} T_k \quad (1)$$

$$\sum_{j=1}^{n} x_{jk} = 1 \quad \forall k \quad (2)$$

$$\sum_{k=1}^{l} x_{jk} = 1 \quad \forall j \quad (3)$$

$$Cp_k \geq \sum_{j=1}^{n} (p_j * x_{jk}) + Cp_{k-1} \quad \forall k \quad (4)$$

$$T_k \geq Cp_k - d_k \forall k \quad (5)$$

$$T_k \geq 0 \quad \forall k \quad (6)$$

4.1.4 Código Lingo para $F_m|$ pmru $|\ C_{max}$:

```
MODEL:
SETS:
  Jobs/1..4/;
  Machines/1..2/;
  JM(Machines,Jobs): P, C;
  JJ(Jobs,Jobs): Alfa;
ENDSETS
DATA:
N = 4;
M = 2;
P=4 2 1 3 2 3 2 1 ;
ENDDATA
MIN = Cmax;  (1)
@For(Machines(i):@For(Jobs(j):C(i,j)<=Cmax));  (2)
@For(Jobs(j):@Sum(Jobs(k): Alfa(j,k))=1);  (3)
@For(Jobs(k):@Sum(Jobs(j): Alfa(j,k))=1);  (4)
C(1,1)>=@SUM(Jobs(j): p(1,j)*Alfa(j,1));  (5)
@For (Jobs(k)|k#GT#1: @For(Machines(i): C(i,k)>=C(i,k-1)+
@SUM(Jobs(j):p(i,j)*alfa(j,k))));  (6)
@For (Machines(i)|i#GT#1: @For(Jobs(k): C(i,k)>=C(i-1,k)+ @SUM(Jobs(j):
p(i,j)*alfa(j,k))));  (7)
@For(Jobs(j):@For(Jobs(k): @Bin(Alfa(j,k))));
End
```

Este modelo de Lingo se obtiene a partir del siguiente modelo matemático:

$$Min\ C_{max} \qquad (1)$$

$$C_{ij} \leq C_{max}\ \forall i,j \qquad (2)$$

$$\sum_{k=1}^{l} x_{jk} = 1 \quad \forall j \qquad (3)$$

$$\sum_{j=1}^{n} x_{jk} = 1 \quad \forall k \qquad (4)$$

$$C_{1,1} \geq \sum_{j=1}^{n} pt_{1,j} * x_{1,j} \ (5)$$

$$C_{i,k} \geq C_{i,k-1} + \sum_{j=1}^{n} pt_{ij} * x_{jk} \quad \forall k > 1, i \qquad (6)$$

$$C_{i,k} \geq C_{i-1,k} + \sum_{j=1}^{n} pt_{ij} * x_{jk} \quad \forall i > 1, k \qquad (7)$$

4.2 Modelo B

4.2.1 Código Lingo para $1||\sum_1^n C_j$:

El modelo B en Lingo para 8 trabajos es el siguiente:

```
model:
sets:
job/1..8/:t,pt,c;
orden(job,job):x;
endsets

data:
pt=5 5 6 3 2 6 8 10;
v= 10000;
enddata

min=@sum(job(j):                    c(j));    (1)
@for(job(j):                         c(j) = t(j)+pt(j));   (2)
@for(job(j):@for(job(k)|j#LT#k:      x(j,k)+x(k,j) >=1));   (3)
@for(job(j):@for(job(k)|j#LT#k: t(j)-t(k) >= pt(k)*x(j,k) - v*(1-x(j,k))));   (4)
@for(job(j):@for(job(k)|j#LT#k: t(k)-t(j) >= pt(j)*x(k,j) - v*(1-x(k,j))));   (5)
@for(job(j):@for(job(k):           @BIN(x(j,k))));
END
```

Este modelo de Lingo se obtiene a partir del siguiente modelo matemático:

$$Min \sum_{j=1}^{n} C_j \quad (1)$$

$$C_j = t_j + pt_j \quad \forall j \quad (2)$$

$$x_{jk} + x_{kj} \geq 1 \quad \forall j,k \quad (3)$$

$$t_j - t_k \geq pt_k * x_{jk} - V * (1 - x_{jk}) \quad \forall j,k,j<k \quad (4)$$

$$t_k - t_j \geq pt_j * x_{kj} - V * (1 - x_{kj}) \quad \forall j,k,j<k \quad (5)$$

4.2.2 Código Lingo para $1||\sum_1^n T_j$:

Para una función objetivo de sumatorio de *tardiness*:

```
model:

sets:
job/1..10/:t,pt,c,dd,tard;
orden(job,job):x;
endsets

data:
pt=1 3 5 4 7 5 3 6 1 5;
dd=1 2 3 5 2 3 6 8 20 15;
v= 10000;
enddata

min=@sum(job(j):                    Tard(j));   (1)
@for(job(j):                         c(j) = t(j)+pt(j));   (2)
@for(job(j):@for(job(k)|j#LT#k:      x(j,k)+x(k,j) >=1));   (3)
@for(job(j):@for(job(k)|j#LT#k:t(j)-t(k)>= pt(k)*x(j,k)-v*(1-x(j,k))));   (4)
@for(job(j):@for(job(k)|j#LT#k:t(k)-t(j)>= pt(j)*x(k,j)-v*(1-x(k,j))));   (5)
@for(job(j): tard(j)>= c(j)-dd(j));   (6)
@for(job(j): tard(j)>=0);   (7)
@for(job(j):@for(job(k):            @BIN(x(j,k))));

END
```

Este modelo de Lingo se obtiene a partir del siguiente modelo matemático:

$$Min \sum_{j=1}^{n} T_j \quad (1)$$

$$C_j = t_j + pt_j \quad \forall j \quad (2)$$

$$x_{jk} + x_{kj} \geq 1 \quad \forall j,k \quad (3)$$

$$t_j - t_k \geq pt_k * x_{jk} - V * (1 - x_{jk}) \quad \forall j,k,j<k \quad (4)$$

$$t_k - t_j \geq pt_j * x_{kj} - V * (1 - x_{kj}) \quad \forall j,k,j<k \quad (5)$$

$$T_j \geq C_j - d_j \forall j \quad (6)$$

$$T_j \geq 0 \quad \forall k \quad (7)$$

4.2.3 Código Lingo para $F_m|$ pmru $|\ C_{max}$:

```
MODEL:

sets:
job/1..5/:;
machine/1..4/:;
orden(job,job):x
completion_time(machine,job):c,pt,t;
endsets

data:
N=5;
M=4;
pt=17 22 31 42 53
64 71 81 12 140
17 22 31 42 53
   64 71 81 12 140
17 22 31 42 53
   64 71 81 12 140
17 22 31 42 53
   64 71 81 12 140;
v= 10000;
enddata

min=Cmax; (1)
@for(job(j): c(M,j)<=Cmax);  (2)
@for(machine(i):@for(job(j):   c(i,j) = t(i,j)+pt(i,j)));  (3)
@for(job(j):@for(job(k)|j#LT#k:   x(j,k)+x(k,j) =1));  (4)
@for(machine(i)|i#LT#M:@for(machine(ii)|ii#EQ#i+1:@for(job(j):t(i,j)+ pt(i,j)
<= t(ii,j)))); (5)
@for(machine(i):@for(job(j):@for(job(k)|j#NE#k:t(i,j)+pt(i,j)<= t(i,k) +
v*(1-x(j,k))));  (6)

@for(job(j):@for(job(k): @BIN(x(j,k))));

END
```

Este modelo de Lingo se obtiene a partir del siguiente modelo matemático:

$$
\begin{aligned}
&Min\ C_{max} && && (1)\\
&C_{M,j} \le C_{max} && \forall j && (2)\\
&C_{i,j} = t_{i,j} + pt_{i,j} && \forall i,j && (3)\\
&x_{j,k} + x_{k,j} = 1 && \forall j,k && (4)\\
&t_{i,j} + pt_{i,j} \le t_{i+1,j} && \forall\ i < M, i+1 && (5)\\
&t_{i,j} + pt_{i,j} \le t_{i,k} + v * \left(1 - x_{j,k}\right) && \forall i,j,k && (6)
\end{aligned}
$$

4.3 Modelo C

4.3.1 Código Lingo para $1||\sum_1^n C_j$:

El modelo Lingo aplicado a 4 trabajos resultante es:

```
model:

sets:
job/1..5/:t,pt,c;
orden(job,job):x;
endsets

data:
pt= 0 3 3 2 5;
v= 10000;
enddata
MIN = @Sum(job(j):C(j));  (1)
@for(job(k): @sum(job(j)|j#NE#k: x(k,j)) <= 1);  (2)
@for(job(k):x(k,1)=0);  (3)
@for(job(j)|j#GT#1: @sum(job(k)|j#NE#k: x(k,j))>=1);  (4)
@for(job(j): c(j) >= t(j)+pt(j));  (5)
@for(job(j):@for(job(k)|j#NE#k: t(k)>= c(j) - v*(1-x(j,k))));  (6)
@for(job(j):@for(job(k): @BIN(x(j,k))));
END
```

Este modelo de Lingo se obtiene a partir del siguiente modelo matemático:

$$Min: \sum_{j=1}^{n} C_j \quad (1)$$

$$\sum_{j=1}^{n} x_{k,j} \leq 1 \qquad \forall k, j \neq k \quad (2)$$

$$x_{k,1} = 0 \qquad \forall k \quad (3)$$

$$\sum_{k=1}^{l} x_{kj} \geq 1 \qquad \forall j, j \neq k, j > 1 \quad (4)$$

$$C_j \geq t_j + pt_j \forall j \quad (5)$$

$$t_k \geq C_j - V * (1 - x_{jk}) \forall j, k, j \neq k \quad (6)$$

4.3.2 Código Lingo para $1||\sum_1^n T_j$:

Si la función objetivo fuera sumatorio de tardiness:

```
model:

sets:
job/1..5/:t,pt,c,dd,tard;
orden(job,job):
x;
endsets

data:
pt=0 1 3 5 4;
dd=0 1 2 3 5;
v= 10000;
enddata

min=@sum(job(j):Tard(j));
@for(job(k): @sum(job(j)|j#NE#k: x(k,j)) <= 1);
@for(job(k):x(k,1)=0);
@for(job(j)|j#GT#1: @sum(job(k)|j#NE#k: x(k,j))>=1);
@for(job(j):c(j) >= t(j)+pt(j));
@for(job(j):@for(job(k)|j#NE#k: t(k)>= c(j) - v*(1-x(j,k))));
@for(job(j):@for(job(k): @BIN(x(j,k))));
@for(job(j): tard(j)>= c(j)-dd(j));
@for(job(j): tard(j)>=0);

END
```

Este modelo de Lingo se obtiene a partir del siguiente modelo matemático:

$$Min: \sum_{j=1}^{n} T_j \quad (1)$$

$$\sum_{j=1}^{n} x_{k,j} \leq 1 \qquad \forall k, j \neq k \quad (2)$$

$$x_{k,1} = 0 \qquad \forall k \quad (3)$$

$$\sum_{k=1}^{l} x_{kj} \geq 1 \qquad \forall j, j \neq k, j > 1 \quad (4)$$

$$C_j \geq t_j + pt_j \forall j \qquad (5)$$

$$t_k \geq C_j - V * (1 - x_{jk}) \forall j, k, j \neq k \quad (6)$$

$$T_j = C_j - d_j \quad \forall j \quad (7)$$

$$T_j \geq 0 \quad \forall j \quad (8)$$

5 EXPERIMENTACIÓN

"No amount of experimentation can ever prove me right;
a single experiment can prove me wrong."

Albert Einstein.

En este capítulo se recoge toda la información referida a la experimentación de modelos, llevada a cabo con el software de Lingo y el apoyo de C. Se simulan secuencialmente 10, 25 y 50 trabajos para cada modelo de single machine y, en el caso de los modelos Flow Shop, 5, 10, 20 máquinas y 20, 50, 100 trabajos. En el caso de que la simulación en alguno de los modelos supere 1h sin obtener el óptimo, se intentará buscar el número de trabajos más próximo que permita simular en dicho tiempo.

5.1 Modelo A

5.1.1 Experimentación en $1|\ |\sum_1^n C_j$

Para 10 trabajos:

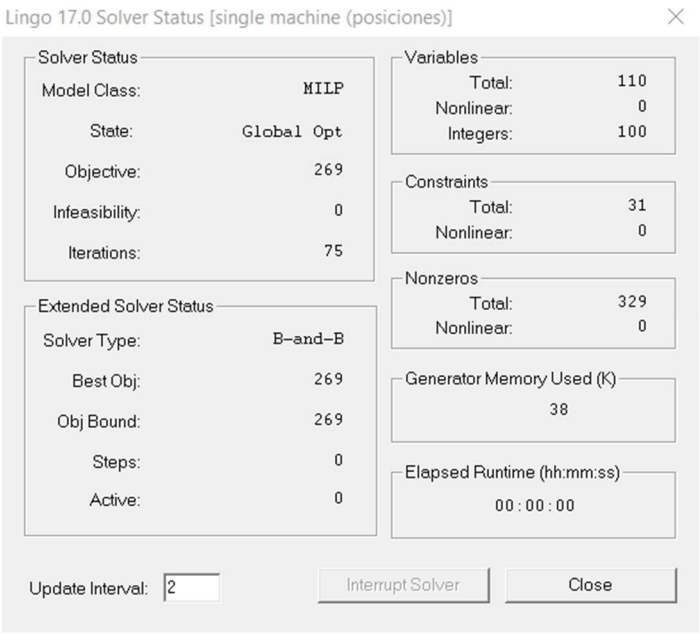


Figura 14. Resultado simulación en modelo A:$1|\ |\sum_1^n C_j$ con 10 trabajos

Se puede apreciar que lo resuelve sin problemas y en un tiempo prácticamente instantáneo.

Para 25 trabajos:

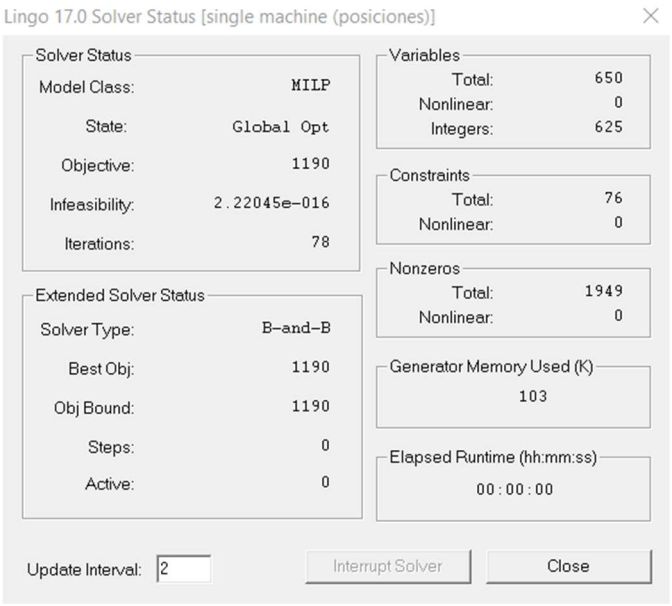


Figura 15. Resultado simulación en modelo A: 1| $|\sum_1^n C_j$ con 25 trabajos

Se puede apreciar que lo resuelve sin problemas y en un tiempo prácticamente instantáneo.

Para 50 trabajos:

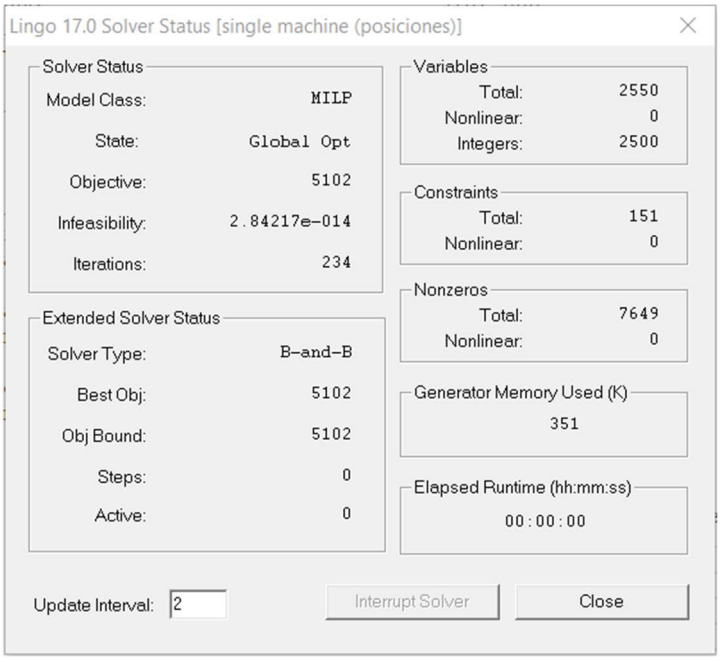


Figura 16. Resultado simulación en modelo A: 1| $|\sum_1^n C_j$ con 50 trabajos

Se puede apreciar que lo resuelve sin problemas y en un tiempo prácticamente instantáneo.

Tabla resumen:

Número de trabajos	Valor función objetivo	Tiempo de resolución (min)	Variables	Restricciones
10	269	0	110	31
25	1190	0	650	76
50	5102	0	2550	151

Tabla 3. Tiempo de resolución simulando trabajos crecientes en modelo A: 1| $|\sum_1^n C_j$

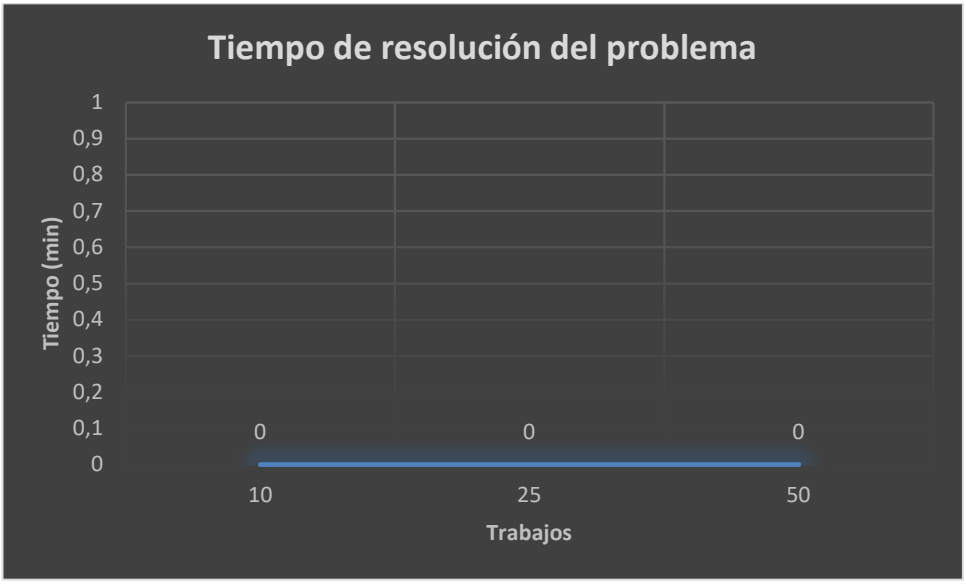


Figura 17. Gráfica resumen de modelo A: 1| $|\sum_1^n C_j$

Conclusión:

Tras las diversas simulaciones el modelo no parece ralentizarse. Como consecuencia, se puede afirmar que este modelo soporta simulaciones con grandes volúmenes de datos de entrada, es eficiente en su resolución y los tiempos de computación son buenos.

5.1.2 Experimentación en $1|\ |\sum_1^n T_j$

Para 10 trabajos:

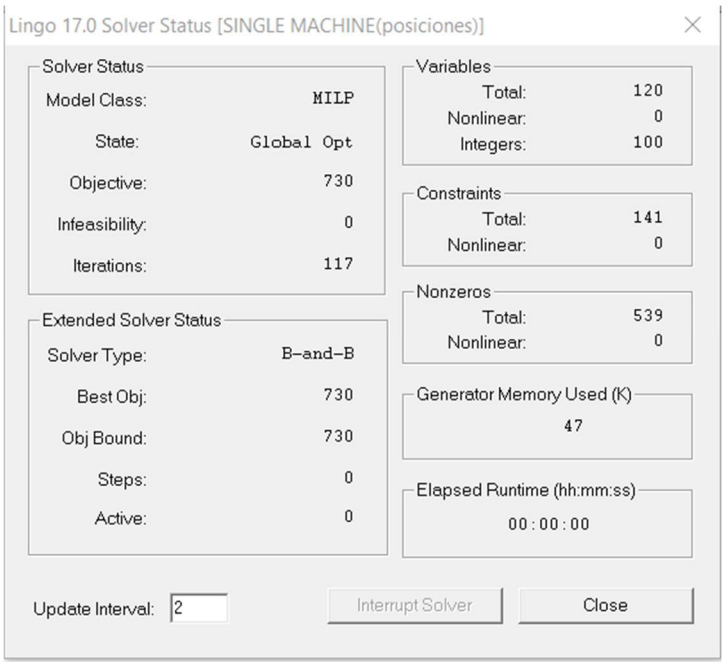


Figura 18. Resultado simulación en modelo A: $1|\ |\sum_1^n T_j$ con 10 trabajos

Se puede apreciar que lo resuelve sin problemas y en un tiempo prácticamente instantáneo.

Para 25 trabajos:

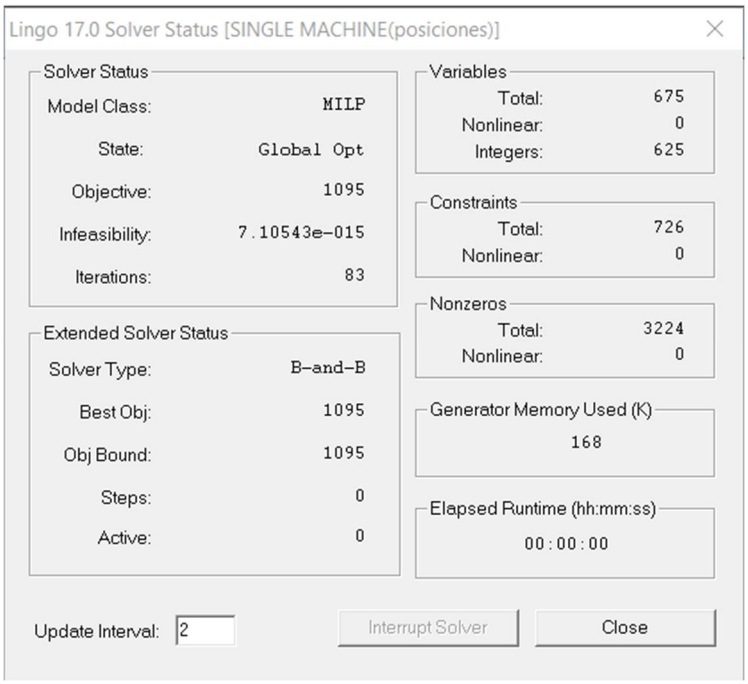


Figura 19. Resultado simulación en modelo A :$1|\ |\sum_1^n T_j$ con 25 trabajos

Se puede apreciar que lo resuelve sin problemas y en un tiempo prácticamente instantáneo.

Para 50 trabajos:

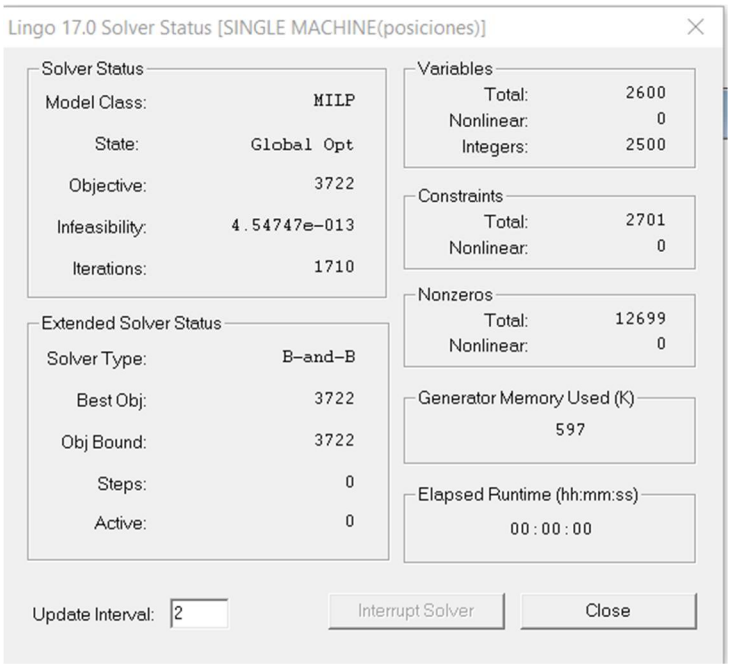


Figura 20. Resultado simulación en modelo A: 1| $|\sum_1^n T_j$ con 50 trabajos

Se puede apreciar que lo resuelve sin problemas y en un tiempo prácticamente instantáneo.

Número de trabajos	Valor función objetivo	Tiempo de resolución (min)	Variables	Restricciones
10	730	0	120	141
25	1095	0	675	726
50	3722	0	2600	2701

Tabla 4. Tiempo de resolución simulando trabajos crecientes en modelo A: 1| $|\sum_1^n T_j$

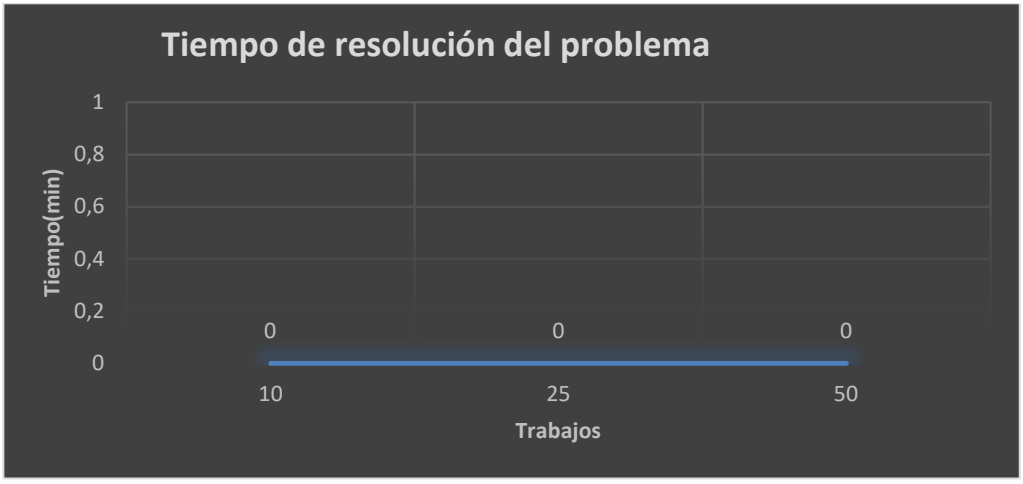


Figura 21. Gráfica resumen de modelo A:1| $|\sum_1^n T_j$

Conclusión:

Al igual que en modelo aplicado a sumatorio de tiempos de terminación, los tiempos de simulación son buenos y como consecuencia se puede afirmar que se trata de un modelo relativamente eficiente.

5.1.2 Experimentación en $F_m|$ pmru $|\ C_{max}$:

Se han realizado simulaciones con 3 grupos de baterías diferentes con 5, 10, 20 máquinas y 20, 50, 100 trabajos. Obtenemos los siguientes resultados[12]:

Grupo de simulaciones 1:

Número Máquinas	Número Trabajos	Función objetivo obtenida (F)	Valor Óptimo [13](F*)	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
	20	1278	1278	0	501	317	1045746
	50	2724	2724	0	2751	795	220407
5	100	5495	5493	2	10501	1597	2396624
	20	1607	1582	25	601	612	27276303
	50	3124	3025	99	3001	1524	12583076
10	100	5861	5770	91	11001	3092	764556
	20	2347	2297	50	801	1202	14372253
	50	4062	3875	187	3501	3032	2577722
20	100	6592	6286	306	12001	6082	251763

Tabla 5. Tabla resumen de los resultados de simulación 1 en modelo A: $F_m|$ pmru $|\ C_{max}$

Grupo de simulaciones 2:

Número Máquinas	Número Trabajos	Función objetivo obtenida (F)	Valor Óptimo (F*)	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
	20	1359	1359	0	501	317	1045746
	50	2834	2834	0	2751	795	220407
5	100	5268	5268	0	10501	1597	2396624
	20	1689	1659	30	601	612	27276303
	50	2958	2892	66	3001	1524	12583076
10	100	5451	5349	102	11001	3092	764556
	20	2161	2100	61	801	1202	14372253
	50	3857	3715	142	3501	3032	2577722
20	100	6533	6241	292	12001	6082	251763

Tabla 6. Tabla resumen de los resultados de simulación 2 en modelo A: $F_m|$ pmru $|\ C_{max}$

[12]Todas las simulaciones están acotadas temporalmente a 1 hora de simulación.

Grupo de simulaciones 3:

Número Máquinas	Número Trabajos	Función objetivo obtenida (F)	Valor Óptimo (F*)	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
	20	1081	1081	0	501	317	1045746
	50	2621	2621	0	2751	795	220407
5	100	5175	5175	0	10501	1597	2396624
	20	1517	1496	21	601	612	27276303
	50	2940	2864	76	3001	1524	12583076
10	100	5792	5677	115	11001	3092	764556
	20	2389	2326	63	801	1202	14372253
	50	3787	3668	119	3501	3032	2577722
20	100	6562	6241	321	12001	6082	251763

Tabla 7. Tabla resumen de los resultados de simulación 3 en modelo A: $F_m|$ pmru $|\ C_{max}$

Si hacemos el promedio entre las simulaciones:

Número Máquinas	Número Trabajos	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
	20	0,0	501	317	1045746
	50	8,3	2751	795	220407
5	100	16,7	10501	1597	2396624
	20	17,0	601	612	27276303
	50	80,3	3001	1524	12583076
10	100	134,7	11001	3092	764556
	20	42,0	801	1202	14372253
	50	117,3	3501	3032	2577722
20	100	306,3	12001	6082	251763

Tabla 8. Tabla resumen de los resultados promedios de simulación en modelo A: $F_m|$ pmru $|\ C_{max}$

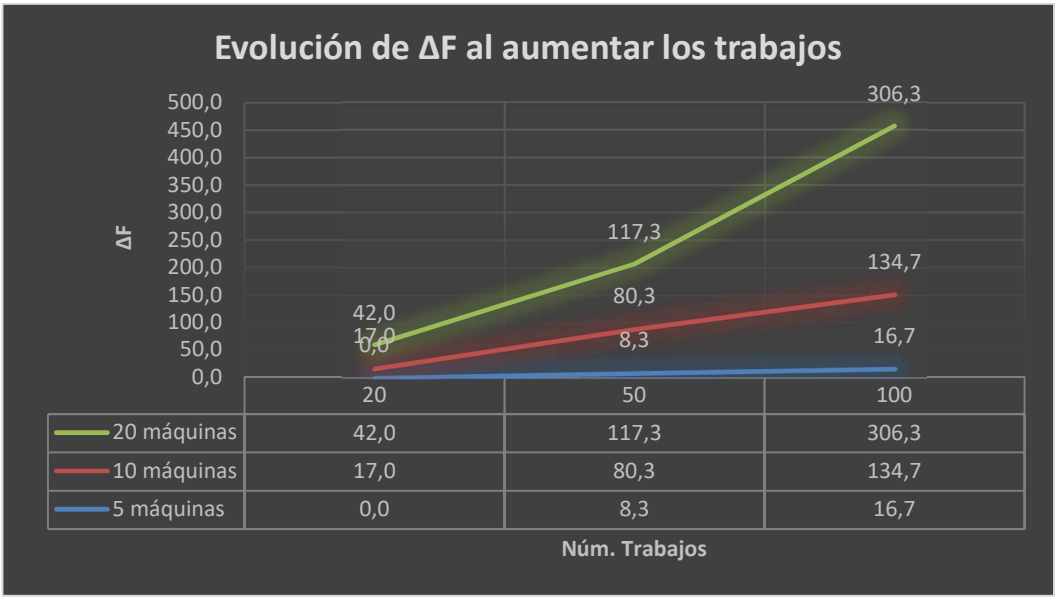


Figura 22. Gráfica resumen con aumento de trabajos de modelo A:F_m| pmru | C_{max}

Se puede apreciar que, como es lógico, al aumentar el número de trabajos manteniendo constante el número de máquinas, el valor de la función objetivo con respecto al óptimo empeora. Ocasionalmente esto no se cumple como consecuencia de que el método de resolución que aplica Lingo para este tipo de problema es Branch and Bound y dicho método puede encontrar buenas soluciones en sus primeras iteraciones.

Esta tabla es resultado de reordenar los datos anteriores:

Número Trabajos	Número Máquinas	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
20	5	0,0	501	317	1045746
	10	17,0	601	612	27276303
	20	42,0	801	1202	14372253
50	5	8,3	2751	795	220407
	10	80,3	3001	1524	12583076
	20	117,3	3501	3032	2577722
100	5	16,7	10501	1597	2396624
	10	134,7	11001	3092	764556
	20	306,3	12001	6082	251763

Tabla 9. Tabla resumen reordenada de los resultados promedios de simulación en modelo A: F_m| pmru | C_{max}

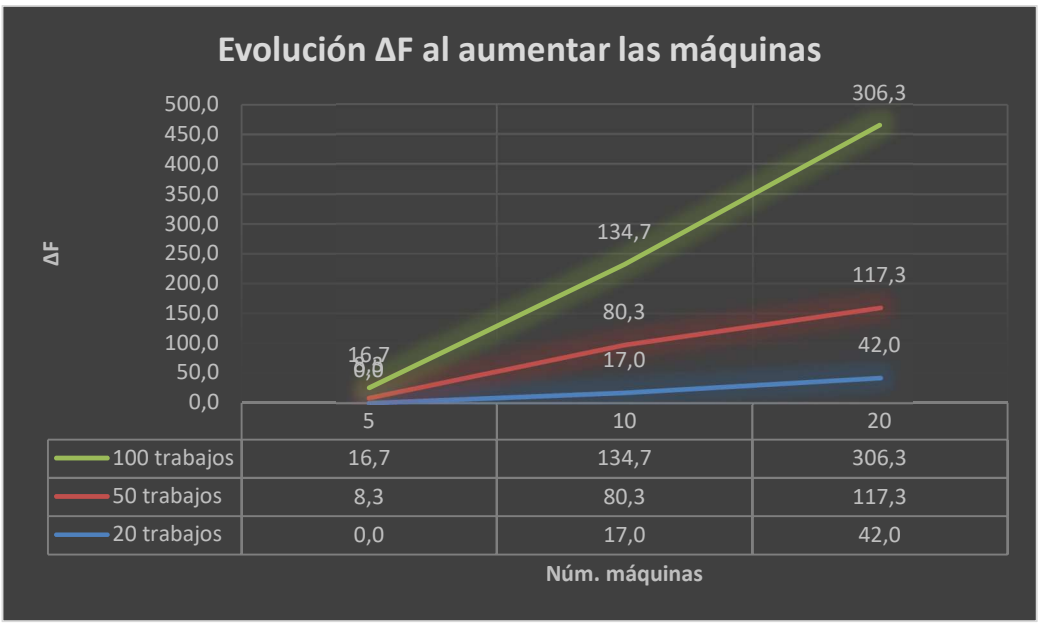


Figura 23. Gráfica resumen con aumento de máquinas de modelo A:F_m| pmru | C_{max}

Esta gráfica es el resultado de aumentar el número de máquinas al mantener constante el número de trabajos. Se observa una tendencia creciente, y más agresiva que en el caso anterior, en el valor de la función objetivo obtenida con respecto al valor óptimo.

Toda la información anterior se puede visualizar también de esta forma:

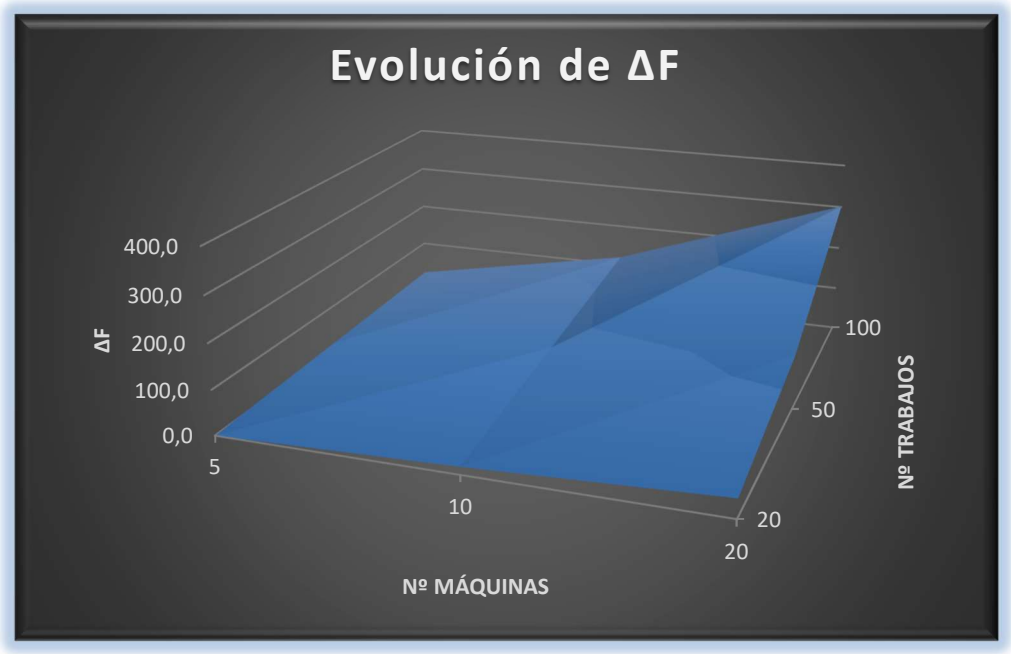


Figura 24. Gráfica resumen con aumento de máquinas y trabajos de modelo A:F_m| pmru | C_{max}

Se concluye que al aumentar el número de máquinas las simulaciones se vuelven peores a mayor velocidad que al aumentar los trabajos.

5.2 Modelo B

5.2.1 Experimentación en $1|\ |\sum_1^n C_j$

A través de sucesivas simulaciones con Lingo obtenemos:

Número de trabajos	Valor función objetivo	Tiempo de resolución (min)	Variables	Restricciones
8	160	0,08	80	93
9	228	0,25	99	118
10	260	2,53	120	146
11	376	15,55	143	177
12	457	60,00	160	211

Tabla 10. Tiempo de resolución simulando trabajos crecientes en modelo B:$1|\ |\sum_1^n C_j$

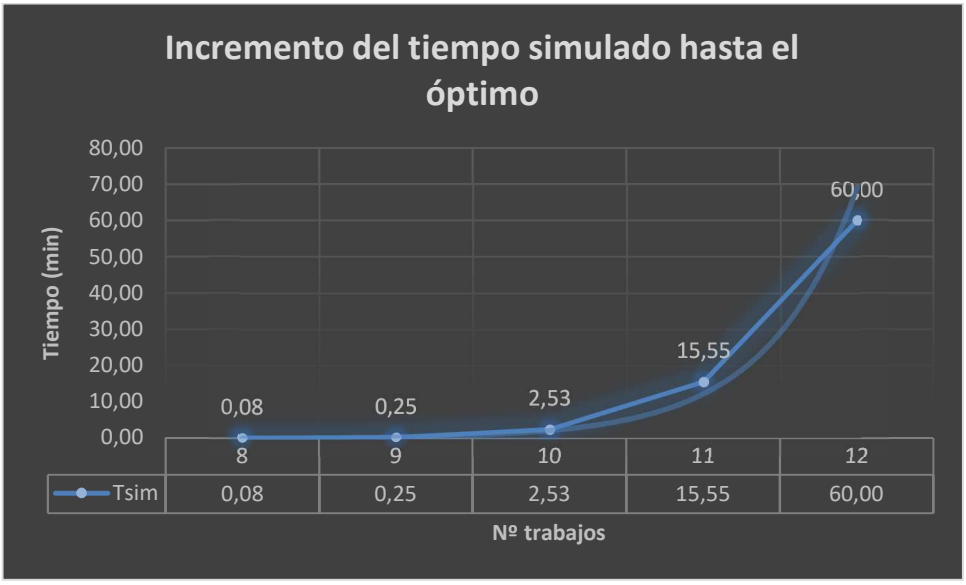


Figura 25. Gráfica resumen de modelo B: $1|\ |\sum_1^n C_j$

Se puede apreciar que, a medida que incrementamos los datos de entrada, se dispara el tiempo de computación. He marcado un tiempo máximo de simulación de 1 hora y, cuando se simulan 12 trabajos o más, se supera dicho margen temporal. El tiempo de simulación evoluciona de manera similar al de una función exponencial.

Conclusión:

Se trata de un modelo poco eficiente que tiene problemas a la hora de trabajar con grandes volúmenes de datos de entrada dado que para alcanzar el valor óptimo requiere un gran número de iteraciones y su consumo de recursos es elevado.

5.2.2 Experimentación en $1|\ |\sum_1^n T_j$

Número de trabajos	Valor función objetivo	Tiempo de resolución (min)	Variables	Restricciones
8	91	0,18	88	109
9	115	0,57	108	136
10	116	2,23	130	166
11	141	10,72	154	199
12	194	60,00	180	235

Tabla 11. Tiempo de resolución simulando trabajos crecientes en modelo B:$1|\ |\sum_1^n T_j$

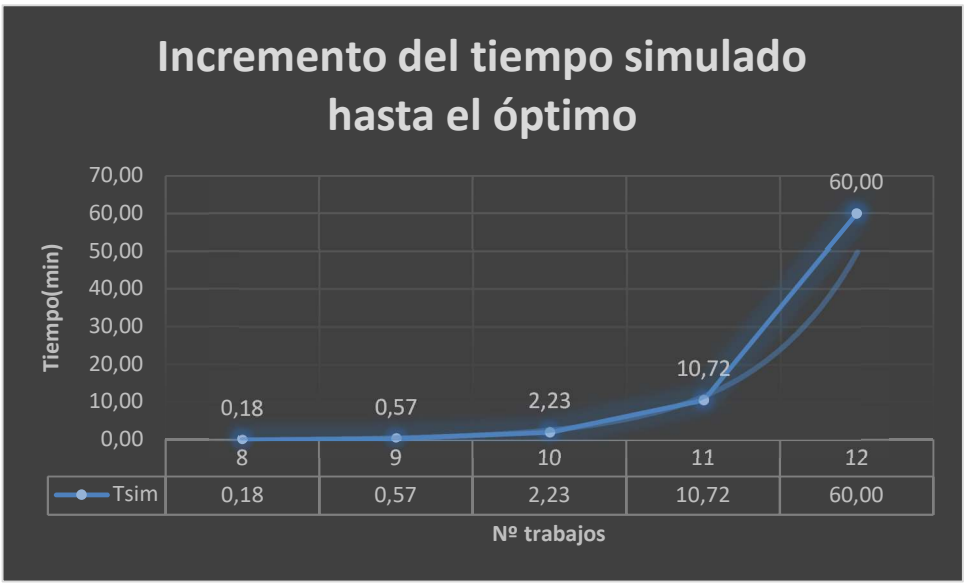


Figura 26. Gráfica resumen de modelo B:$1|\ |\sum_1^n T_j$

Se puede apreciar que, a medida que incrementamos los datos de entrada, se dispara el tiempo de computación. He marcado un tiempo máximo de simulación de 1 hora y, cuando se simulan 12 trabajos o más, se supera dicho margen temporal. De nuevo la gráfica toma una forma similar al de una exponencial.

Conclusión:

Al igual que en el modelo anterior, podemos afirmar que este modelo es relativamente ineficiente puesto que, al incrementar el número de trabajos simulados, se dispara el tiempo de resolución.

5.2.3 Experimentación en $F_m|$ pmru $|\ C_{max}$:

Se han realizado simulaciones con 3 grupos de baterías diferentes con 5, 10, 20 máquinas y 20, 50, 100 trabajos. Se obtienen los siguientes resultados[14]:

Grupo de simulación 1:

Número Máquinas	Número Trabajos	Función objetivo obtenida (F)	Valor Óptimo (F*)	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
5	20	1336	1278	58	601	2291	13365814
	50	3095	2724	371	3001	13976	1501526
	100	6209	5493	716	11001	55451	509006
10	20	1729	1582	147	801	4391	6119281
	50	3742	3025	717	3501	26726	1171154
	100	6842	5770	1072	12001	105951	295005
20	20	2712	2297	415	1201	8591	3835284
	50	4877	3875	1002	4501	52226	725545
	100	7864	6286	1578	14001	206951	431411

Tabla 12. Tabla resumen de los resultados de simulación 1 en modelo B: $F_m|$ pmru $|\ C_{max}$

Grupo de simulación 2:

Número Máquinas	Número Trabajos	Función objetivo obtenida (F)	Valor Óptimo (F*)	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
5	20	1419	1359	60	601	2291	13365814
	50	3222	2834	388	3001	13976	1501526
	100	6025	5268	757	11001	55451	509006
10	20	1819	1659	160	801	4391	6119281
	50	3623	2892	731	3501	26726	1171154
	100	6401	5349	1052	12001	105951	295005
20	20	2545	2110	435	1201	8591	3835284
	50	4711	3715	996	4501	52226	725545
	100	7866	6241	1625	14001	206951	431411

Tabla 13. Tabla resumen de los resultados de simulación 2 en modelo B: $F_m|$ pmru $|\ C_{max}$

[14]Todas las simulaciones están acotadas a 1 hora.

Grupo de simulación 3:

Número Máquinas	Número Trabajos	Función objetivo obtenida (F)	Valor Óptimo (F*)	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
	20	1137	1081	56	601	2291	13365814
	50	2994	2621	373	3001	13976	1501526
5	100	5903	5175	728	11001	55451	509006
	20	1633	1496	137	801	4391	6119281
	50	3567	2864	703	3501	26726	1171154
10	100	6680	5677	1003	12001	105951	295005
	20	2747	2326	421	1201	8591	3835284
	50	4658	3668	990	4501	52226	725545
20	100	7893	6329	1564	14001	206951	431411

Tabla 14. Tabla resumen de los resultados de simulación 3 en modelo B: $F_m|$ pmru $|\ C_{max}$

Los valores promedios de las simulaciones son los siguientes:

Número Máquinas	Número Trabajos	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
	20	58,0	601	2291	13365814
	50	377,3	3001	13976	1501526
5	100	733,7	11001	55451	509006
	20	148,0	801	4391	6119281
	50	717,0	3501	26726	1171154
10	100	1042,3	12001	105951	295005
	20	423,7	1201	8591	3835284
	50	996,0	4501	52226	725545
20	100	1589,0	14001	206951	431411

Tabla 15. Tabla resumen de los resultados promedios de simulación en modelo B: $F_m|$ pmru $|\ C_{max}$

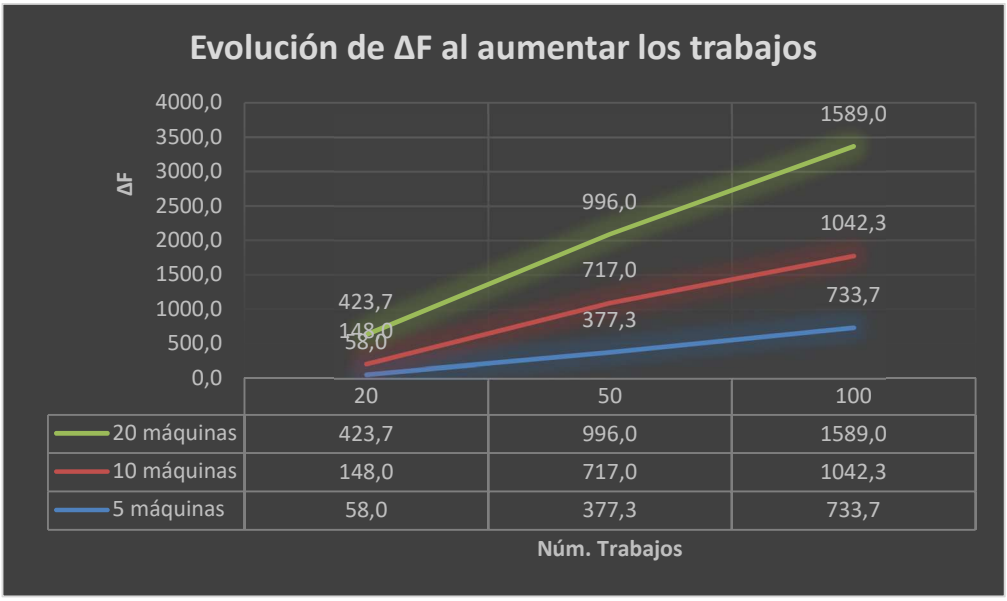


Figura 27. Gráfica resumen con aumento de trabajos de modelo B: $F_m|$ pmru $|\ C_{max}$

Se puede apreciar un crecimiento prácticamente lineal en el empeoramiento de la función objetivo obtenida conforme se aumenta el número de trabajos, manteniendo las máquinas constantes.

Reestructurando los datos anteriores obtenemos:

Número Trabajos	Número Máquinas	ΔF= F-F*	Variables del problema	Restricciones del modelo	Número de iteraciones
	5	58,0	601	2291	13365814
	10	148,0	801	4391	6119281
20	20	423,7	1201	8591	3835284
	5	377,3	3001	13976	1501526
	10	717,0	3501	26726	1171154
50	20	996,0	4501	52226	725545
	5	733,7	11001	55451	509006
	10	1042,3	12001	105951	295005
100	20	1589,0	14001	206951	431411

Tabla 16. Tabla resumen reordenada de los resultados promedios de simulación en modelo B: $F_m|$ pmru $|\ C_{max}$

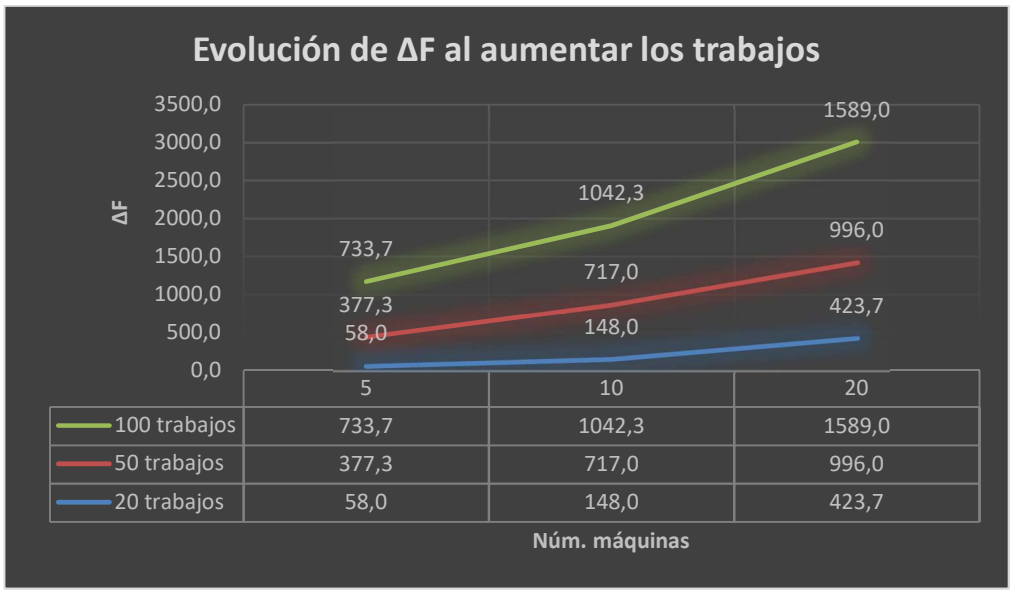


Figura 28. Gráfica resumen con aumento de máquinas de modelo B:F_m| pmru | C_{max}

Cuando mantenemos constantes los trabajos y aumentamos las máquinas se obtiene un crecimiento similar al de cuando se aumentan los trabajos.

A partir de las tablas anteriores se puede representar lo siguiente:

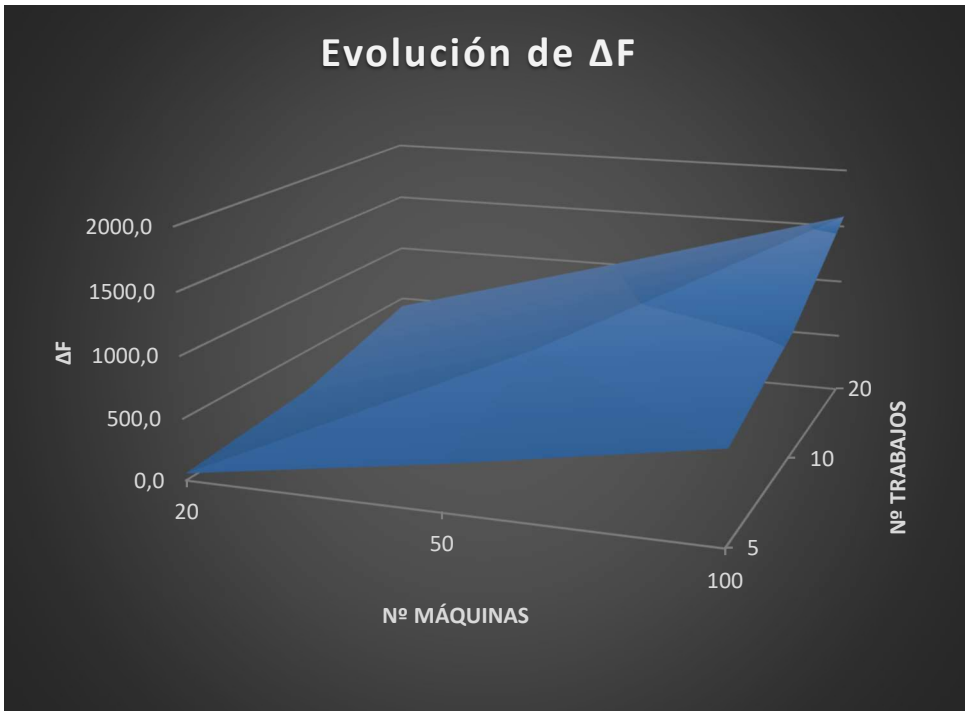


Figura 29. Gráfica resumen con aumento de trabajos y máquinas de modelo B:F_m| pmru | C_{max}

Conclusión:

A medida que aumentamos las máquinas o los trabajos, como es lógico, obtenemos un problema que requiere mayor tiempo de computación para obtener el óptimo. Se puede apreciar que este modelo B es más sensible a un aumento de datos de entrada que el modelo A, puesto que sus soluciones se alejan más del óptimo y, como consecuencia, es peor si queremos resolver casos prácticos.

Partiendo de la premisa de que en número de variables de los modelos A y B son similares; la ineficiencia del modelo B se puede justificar desde la comparativa entre el número de restricciones que presenta con respecto al modelo A. Ese gran número de restricciones del modelo B es causante de que, en un determinado tiempo de simulación, se puedan realizar menos iteraciones y, como consecuencia, la exploración de soluciones realizada por Lingo es menor.

5.3 Modelo C

5.3.1 Experimentación en $1|\ |\sum_1^n C_j$

Número de trabajos	Valor función objetivo	Tiempo de resolución (min)	Variables	Restricciones
6	83	0,03	66	63
7	117	0,28	72	80
8	169	2,43	90	99
9	212	18,52	110	120
10	270	60,00	132	143

Tabla 17. Tiempo de resolución simulando trabajos crecientes en modelo C:$1|\ |\sum_1^n C_j$

Figura 30. Gráfica resumen de modelo C: $1|\ |\sum_1^n C_j$

Se puede apreciar que, debido a la naturaleza del modelo, cuando queremos simular un número n de trabajos, hay que introducir en el programa n+1 trabajos. Esto se debe a que el modelo funciona con un trabajo inicial con tiempo de proceso nulo que asegura que las restricciones funcionen de acuerdo con lo previsto. Se observa un comportamiento similar al del modelo B; al aumentar los trabajos se genera un crecimiento exponencial de los tiempos de simulación hasta el óptimo, haciendo de este modelo uno bastante poco práctico.

Conclusión:

Este modelo manifiesta los mismos defectos que el modelo B, y a la hora de simular 10 trabajos se supera una hora de procesamiento. Es capaz de obtener el óptimo, pero para un número relativamente alto de datos de entrada se vuelve poco práctico al consumir mucho tiempo de computación.

5.3.2 Experimentación en $1|\ |\sum_1^n T_j$

Número de trabajos	Valor función objetivo	Tiempo de resolución (min)	Variables	Restricciones
7	44	0,08	80	96
8	66	1,05	99	117
9	72	3,18	120	140
10	75	17,27	143	165
11	83	60,00	158	182

Tabla 18. Tiempo de resolución simulando trabajos crecientes en modelo C: $1|\ |\sum_1^n T_j$

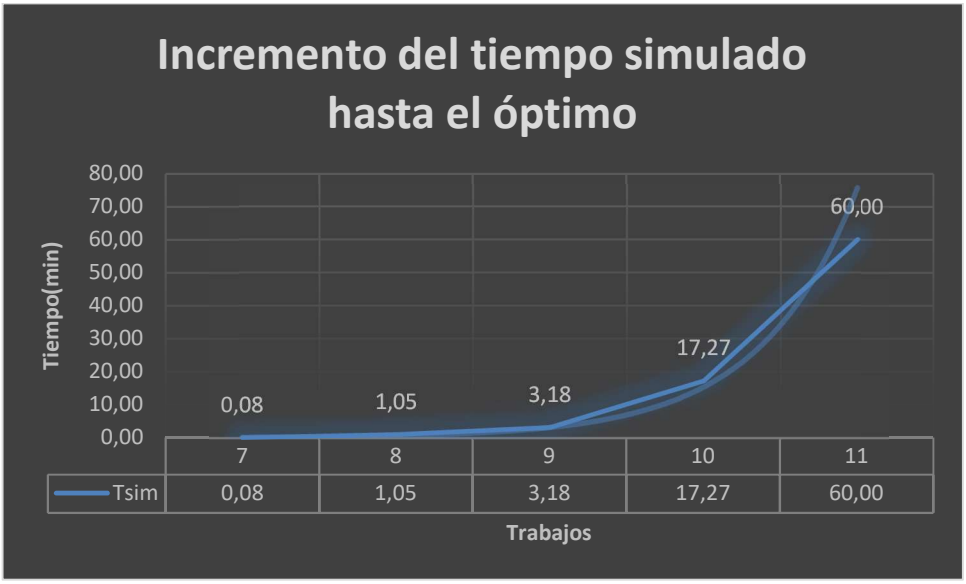


Figura 31. Gráfica resumen de modelo C:$1|\ |\sum_1^n T_j$

Misma tendencia exponencial que en casos anteriores.

Conclusión:

Al igual que en el caso anterior, podemos afirmar que se trata de un modelo poco práctico que te permite obtener el óptimo cuando simulamos pocos trabajos, pero a medida que se incrementan dichos trabajos el modelo se muestra sumamente ineficiente computacionalmente hablando.

6 CONCLUSIONES

> "All great achievements in science start from intuitive knowledge, namely, in axioms, from which deductions are then made. Intuition is the necessary condition for the discovery of such axioms..."
>
> *Albert Einstein.*

Tras la realización del Proyecto de Fin de Grado se puede concluir que los problemas de Scheduling pueden ser sencillos o inmensamente complejos en su resolución. También se puede afirmar que un enfoque acertado a la hora de modelar un problema de este tipo será fundamental para que su resolución, computacionalmente hablando, sea viable desde un enfoque práctico.

Entrando más en detalle, si modelamos un problema de Scheduling desde la perspectiva que hemos llamado de "posiciones" obtenemos resultados cuantiosamente más favorables que si lo hacemos desde la perspectiva de "solapes". Otra observación interesante es qué, cuando generamos un modelo híbrido que presenta características de los modelos de "posiciones" y "solapes", se heredan defectos de ambos, y como consecuencia, el modelo resultante es bastante ineficiente también.

Desde el punto de vista de la complejidad, en este documento hemos podido observar que, para un cierto problema, un modelo determinado y otro diferente pueden dar lugar a que los tiempos de exploración de soluciones crezcan de forma lineal o exponencial conforme incrementamos de los datos de entrada. Aquellos con un crecimiento lineal, como es lógico, serán más favorables para su resolución.

Como última conclusión puedo afirmar con confianza que la obtención de soluciones mejores a la hora de resolver un problema de Scheduling a través de exploraciones de soluciones va a verse favorecida con el avance de los procesadores y el estudio de la computación cuántica.

ANEXO

En este anexo incluyo todos los códigos en C que permiten generar los modelos en código Lingo para posteriormente hacer efectiva la simulación.

Modelo A: Código en C para 1| $|\sum_1^n C_j$:

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   FILE *f;  //Declara el la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
if(f==NULL)//si no se puede abrir, print no se puede
   {
printf("no se puede abrir el fichero\n\n");
   }
else// se se abre correctamente print bien
{ printf("Fichero abierto correctamente\n\n");}
int z,w;
   int i=0;
   int j=0;
int cont=0;
  while (!feof(f))
{
fscanf (f, "%d", &z);
printf ("Extraido:%d --", z);
cont=cont+1;
}
      printf("\n\nNumero de dato leidos: %d\n\n",cont);
rewind(f);
cont= cont;
VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
while (!feof(f))
```

```
        {
        fscanf (f, "%d", &z);
        a[i]=z;
i=i+1;
        }
print_vector(a,cont);
 rewind(f);
//////////////////////////////////////
char parte1[] = "MODEL: \nSETS: \njob/1..";
char parte2[] = "/:pt;\n";
char parte3[] = "pos/1..";
char parte4[] = "/: cp;\n";
char parte5[] = "orden(job,pos):x; \nENDSETS\n DATA:\n pt=";
char parte6[] = ";\nENDDATA \nmin=@sum(pos(k): cp(k)); \n@for(pos(k):@sum(job(j): x(j,k)) =1); @for(job(j):@sum(pos(k): x(j,k)) =1); \n@for(pos(k)|k#GT#1:cp(k) >= @sum(job(j): pt(j)*x(j,k)) + cp(k-1)); \ncp(1) >= @sum(job(j): pt(j)*x(j,1)); \n@for(job(j):@for(pos(k):@BIN(x(j,k))));";
FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );
if(fichero== NULL)
  {
   printf("\nERROR fichero lingo no se puede crear");
  }
else
   {
   printf("\nSe puede crear el fichero lingo");
}
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
fprintf (fichero, "%d", cont);
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
fwrite(parte3 , 1 , sizeof(parte3) , fichero);
fprintf (fichero, "%d ",cont);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);
fwrite(parte5 , 1 , sizeof(parte5) , fichero);
for (i = 0; i <cont; i++)
    fprintf (fichero, "%d ", a[i]);
fwrite(parte6 , 1 , sizeof(parte6) , fichero);
fclose(fichero);
return 0;
}
```

Modelo A: Código en C para 1| | $\sum_1^n T_j$:

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   FILE *f;  //Declara el la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
if(f==NULL)//si no se puede abrir, print no se puede
   {
printf("no se puede abrir el fichero\n\n");
   }
else// se se abre correctamente print bien
   {
printf("Fichero abierto correctamente\n\n");
}
   ////////primer while para determinar el tamaño del macrovectorvector /////////segundo while para escribir el macrovector//////////
int z,w;
   int i=0;
   int j=0;
   int cont=0;
  while (!feof(f))
{
fscanf (f, "%d", &z);
printf ("Extraido:%d --", z);
cont=cont+1;
}
printf("\n\nNumero de dato leidos: %d\n\n",cont);
rewind(f);
cont= cont;
VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
while (!feof(f))
      {
      fscanf (f, "%d", &z);
      a[i]=z;
```

```
i=i+1;
      }
print_vector(a,cont);
 rewind(f);
VECTOR_INT pts= DIM_VECTOR_INT(cont/2);
VECTOR_INT dds= DIM_VECTOR_INT(cont/2);
for(i=0;i<(cont/2);i++)
{
   pts[i]=a[i];
}
trabajos=cont/2;
k=1+(cont/2);
for(i=0;i<(cont/2);i++)
{
   pts[i]=a[k];
   k=k+1;
}
 ////////////////////////////////////////
char parte1[] = "MODEL: \nSETS: \n job/1..";
char parte2[] = "/: pt, dd; \npos/1.. ";
char parte3[] = "/: cp,tard;  \n orden(job,pos):x; \nENDSETS \n DATA: \n pt=";
char parte4[] = ";\n dd=";
char parte5[] = "; \nENDDATA\n";
char parte6[] = "min=@sum(pos(k): tard(k));@for(pos(k):@sum(job(j): x(j,k)) =1);@for(job(j):@sum(pos(k): x(j,k))  =1);@for(pos(k)|k#GT#1:cp(k)>=@sum(job(j): pt(j)*x(j,k)) + cp(k-1));cp(1)>=@sum(job(j): pt(j)*x(j,1));@for(pos(k): tard(k)>=cp(k)-dd(k));@for(pos(k): tard(k)>=0);@for(job(j): @for(pos(k): @BIN(x(j,k))));";
char parte7[]="!@for(job(k):@GIN(cp(j))); \nEND ";


FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );
if(fichero== NULL)
  {
  printf("\nERROR fichero lingo no se puede crear");
  }
else
   {
  printf("\nSe puede crear el fichero lingo");
}
```

```
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
fprintf (fichero, "%d", trabajos);
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
fprintf (fichero, "%d ",trabajos);
fwrite(parte3 , 1 , sizeof(parte3) , fichero);
for (i = 0; i <trabajos; i++)
fprintf (fichero, "%d ", pts[i]);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);
for (i = 0; i <trabajos; i++)
fprintf (fichero, "%d ", dds[i]);

fwrite(parte5 , 1 , sizeof(parte5) , fichero);
fwrite(parte6 , 1 , sizeof(parte6) , fichero);
fwrite(parte7 , 1 , sizeof(parte7) , fichero);
fclose(fichero);
   return 0;
}
```

Modelo A: Código en C para $F_m|$ pmru $| C_{max}$:

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   FILE *f;  //Declara el la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
   if(f==NULL)//si no se puede abrir, print no se puede
   {
      printf("no se puede abrir el fichero\n\n");
   }
   else// se se abre correctamente print bien
   {
      printf("Fichero abierto correctamente\n\n");
   }
   ////////primer while para determinar el tamaño del macrovectorvector /////////segundo while para escribir el macrovector//////////
   int z,w;
   int i=0;
   int j=0;
   int cont=0;
  while (!feof(f))
      {
      fscanf (f, "%d", &z);
      printf ("Extraido:%d --", z);
      cont=cont+1;
      }
      printf("\n\nNumero de dato leidos: %d\n\n",cont);

 rewind(f);

cont= cont;

VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
 while (!feof(f))
```

```
      {
      fscanf (f, "%d", &z);

      a[i]=z;
      i=i+1;
      }
 print_vector(a,cont);
 rewind(f);
int nmaq = a[0];
int ntrab;
ntrab=(cont-1)/nmaq;
VECTOR_INT pt = DIM_VECTOR_INT(cont-1);
i=0;
for(j=1;j<cont;j++)
{
   pt[i]=a[j];
   i=i+1;
}
}
 ////////////////////////////////////////

char parte1[] ="MODEL:\nSETS:\nJobs/1..";
char parte2[] ="/;\nMachines/1..\n" ;
char parte3[] ="/;\nJM(Machines,Jobs): P, C;\nJJ(Jobs,Jobs): Alfa;\nENDSETS\nDATA:\nN = ";
char parte4[] =";\n M =" ;
char parte5[] =";\nP=" ;
char parte6[] =";\nENDDATA\n MIN = Cmax;\n @For(Machines(i):@For(Jobs(j):C(i,j)<=Cmax));\n@For(Jobs(j):@Sum(Jobs(k): Alfa(j,k))=1);\n@For(Jobs(k):@Sum(Jobs(j): Alfa(j,k))=1); " ;
char parte7[]="\nC(1,1)>=@SUM(Jobs(j): p(1,j)*Alfa(j,1));\n@For (Jobs(k)|k#GT#1: @For(Machines(i): C(i,k)>=C(i,k-1)+ @SUM(Jobs(j):p(i,j)*alfa(j,k))));";
char parte8[]="\n@For (Machines(i)|i#GT#1: @For(Jobs(k): C(i,k)>=C(i-1,k)+ @SUM(Jobs(j): p(i,j)*alfa(j,k)))); \n @For(Jobs(j):@For(Jobs(k): @Bin(Alfa(j,k))));\nEnd";
  FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );


if(fichero== NULL)
  {
```

```
   printf("\nERROR fichero lingo no se puede crear");
  }
else
   {
   printf("\nSe puede crear el fichero lingo");
   }
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
fprintf (fichero, "%d", ntrab);
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
fprintf (fichero, "%d ",nmaq);
fwrite(parte3 , 1 , sizeof(parte3) , fichero);
fprintf (fichero, "%d", ntrab);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);
fprintf (fichero, "%d ",nmaq);
fwrite(parte5 , 1 , sizeof(parte5) , fichero);
for (i = 0; i <(cont-1); i++)
    fprintf (fichero, "%d ", pt[i]);
fwrite(parte6 , 1 , sizeof(parte6) , fichero);
fwrite(parte7 , 1 , sizeof(parte7) , fichero);
fwrite(parte8 , 1 , sizeof(parte8) , fichero);

fclose(fichero);
   return 0;
}
```

Modelo B: Código en C para 1| | $\sum_1^n C_j$:

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
intmain()
{
   FILE *f;  //Declara el la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
if(f==NULL)//si no se puede abrir, print no se puede
   {
printf("no se puede abrir el fichero\n\n");
   }
else// se se abre correctamente print bien
   {
printf("Fichero abierto correctamente\n\n");
}
int z,w;
   int i=0;
   int j=0;
   int cont=0;

  while (!feof(f))
{
fscanf (f, "%d", &z);
printf ("Extraido:%d --", z);
cont=cont+1;
}
      printf("\n\nNumero de dato leidos: %d\n\n",cont);
rewind(f);

cont= cont;
VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
while (!feof(f))
      {
      fscanf (f, "%d", &z);
      a[i]=z;
```

```
i=i+1;
      }
print_vector(a,cont);
 rewind(f);
 ////////////////////////////////////////
char parte1[] = "MODEL: \nSETS: \njob/1..";
char parte2[] = "/:t,pt,c; \norden(job,job):x; \nendsets \ndata: \n pt=";
char parte3[] = ";\nv= 10000; \nenddata \nmin=@sum(job(j):c(j)); \n@for(job(j):c(j) = t(j)+pt(j)); \n@for(job(j):@for(job(k)|j#LT#k:x(j,k)+x(k,j) >=1)); \n@for(job(j):@for(job(k)|j#LT#k: t(j)-t(k) >= pt(k)*x(j,k) - v*(1-x(j,k))));";
char parte4[] = "\n@for(job(j):@for(job(k)|j#LT#k: t(k)-t(j) >= pt(j)*x(k,j) - v*(1-x(k,j)))); \n@for(job(j):@for(job(k):@BIN(x(j,k)))); \nEND";
FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );
if(fichero== NULL)
  {
printf("\nERRORficherolingo no se puedecrear");
  }
else
   {
printf("\nSepuedecrear el ficherolingo");
}
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
fprintf (fichero, "%d", cont);
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
for (i = 0; i <cont; i++)
    fprintf (fichero, "%d ", a[i]);

fwrite(parte3 , 1 , sizeof(parte3) , fichero);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);

fclose(fichero);
return 0;
}
```

Modelo B: Código en C para 1| $|\sum_1^n T_j$:

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
intmain()
{
   FILE *f;  //Declara el la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
if(f==NULL)//si no se puede abrir, print no se puede
   {
printf("no se puede abrir el fichero\n\n");
   }
else// se se abre correctamente print bien
   {
printf("Fichero abierto correctamente\n\n");
}
   ////////primer while para determinar el tamaño del macrovectorvector /////////segundo while para escribir el macrovector//////////
int z,w;
   int i=0;
   int j=0;
   int cont=0;
  while (!feof(f))
{
fscanf (f, "%d", &z);
printf ("Extraido:%d --", z);
cont=cont+1;
}
printf("\n\nNumero de dato leidos: %d\n\n",cont);
rewind(f);
cont= cont;
VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
while (!feof(f))
      {
      fscanf (f, "%d", &z);
      a[i]=z;
```

```
i=i+1;
      }
print_vector(a,cont);
 rewind(f);

VECTOR_INT pts= DIM_VECTOR_INT(cont/2);
VECTOR_INT dds= DIM_VECTOR_INT(cont/2);
for(i=0;i<(cont/2);i++)
{
   pts[i]=a[i];
}
trabajos=cont/2;
k=1+(cont/2);
for(i=0;i<(cont/2);i++)
{
   pts[i]=a[k];
   k=k+1;
}
 ////////////////////////////////////////
char parte1[] = "model: \nsets: \njob/1..";
char parte2[] = "/:t,pt,c,dd,tard;\norden(job,job):x;\n";
char parte3[] = "endsets\n data:\ pt=\n";
char parte4[] = ";\ndd=";
char parte5[] = ";\nv= 10000;\n";
char parte6[] = "enddata\nmin=@sum(job(j):Tard(j));\n @for(job(j):c(j) =
t(j)+pt(j));\n@for(job(j):@for(job(k)|j#LT#k:x(j,k)+x(k,j) >=1));\n@for(job(j):@for(job(k)|j#LT#k:t(j)-t(k) >=
pt(k)*x(j,k) - v*(1-x(j,k))));\n";
char parte7[]="@for(job(j):@for(job(k)|j#LT#k:t(k)-t(j) >= pt(j)*x(k,j) - v*(1-x(k,j))));@for(job(j): tard(j)>=
c(j)-dd(j));@for(job(j): tard(j)>=0);@for(job(j):@for(job(k):@BIN(x(j,k))));\nEND";

FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );
if(fichero== NULL)
  {
printf("\nERRORficherolingo no se puedecrear");
  }
else
   {
printf("\nSepuedecrear el ficherolingo");
```

```
}
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
fprintf (fichero, "%d", trabajos);
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
fwrite(parte3 , 1 , sizeof(parte3) , fichero);
for (i = 0; i <trabajos; i++)
fprintf (fichero, "%d ", pts[i]);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);
for (i = 0; i <trabajos; i++)
fprintf (fichero, "%d ", dds[i]);
fwrite(parte5 , 1 , sizeof(parte5) , fichero);
fwrite(parte6 , 1 , sizeof(parte6) , fichero);
fwrite(parte7 , 1 , sizeof(parte7) , fichero);
fclose(fichero);
   return 0;
}
```

Modelo B: Código en C para $F_m|$ pmru $|\ C_{max}$:

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   FILE *f;  //Declara el la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
   if(f==NULL)//si no se puede abrir, print no se puede
   {
      printf("no se puede abrir el fichero\n\n");
   }
   else// se se abre correctamente print bien
   {
      printf("Fichero abierto correctamente\n\n");
   }
   ////////primer while para determinar el tamaño del macrovectorvector /////////segundo while para escribir el macrovector//////////
   int z,w;
   int i=0;
   int j=0;
   int cont=0;
  while (!feof(f))
       {
       fscanf (f, "%d", &z);
       printf ("Extraido:%d --", z);
       cont=cont+1;
       }
       printf("\n\nNumero de dato leidos: %d\n\n",cont);
 rewind(f);

VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
 while (!feof(f))
       {
       fscanf (f, "%d", &z);
```

```
      a[i]=z;
      i=i+1;
      }
 print_vector(a,cont);
 rewind(f);
int nmaq = a[0];
int ntrab;
ntrab=(cont-1)/nmaq;
VECTOR_INT pt = DIM_VECTOR_INT(cont-1);
i=0;
for(j=1;j<cont;j++)
{
   pt[i]=a[j];
   i=i+1;
}
}
 ////////////////////////////////////////
char parte1[] ="MODEL:\n sets:\n job/1..";
char parte2[] ="/:;\n machine/1..";
char parte3[] ="/:;\n orden(job,job):x;\n completion_time(machine,job):c,pt,t;\n endsets\n data:\n N=";
char parte4[] =";\n M=";
char parte5[] =";\n pt=";
char parte6[] =";\n v= 10000;\n enddata\n min=Cmax;\n @for(job(j): c(M,j)<=Cmax);\n @for(machine(i):@for(job(j):  c(i,j) = t(i,j)+pt(i,j)));\n";
char parte7[]="@for(job(j):@for(job(k)|j#LT#k: x(j,k)+x(k,j) =1));\n @for(machine(i)|i#LT#M:@for(machine(ii)|ii#EQ#i+1:@for(job(j):t(i,j)+ pt(i,j) <= t(ii,j)))); \n ";
char parte8[]="@for(machine(i):@for(job(j):@for(job(k)|j#NE#k:t(i,j)+pt(i,j)<= t(i,k) + v*(1-x(j,k)))));\n @for(job(j):@for(job(k): @BIN(x(j,k)))); \n END ";
  FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );
if(fichero== NULL)
  {
   printf("\nERROR fichero lingo no se puede crear");
  }
else
   {
   printf("\nSe puede crear el fichero lingo");
   }
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
```

```
fprintf (fichero, "%d", ntrab);
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
fprintf (fichero, "%d ",nmaq);
fwrite(parte3 , 1 , sizeof(parte3) , fichero);
fprintf (fichero, "%d", ntrab);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);
fprintf (fichero, "%d ",nmaq);
fwrite(parte5 , 1 , sizeof(parte5) , fichero);
for (i = 0; i <(cont-1); i++)
    fprintf (fichero, "%d ", pt[i]);
fwrite(parte6 , 1 , sizeof(parte6) , fichero);
fwrite(parte7 , 1 , sizeof(parte7) , fichero);
fwrite(parte8 , 1 , sizeof(parte8) , fichero);

fclose(fichero);
   return 0;
}
```

Modelo C: Código en C para 1| $|\sum_1^n C_j$:

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   FILE *f;  //Declara el la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
if(f==NULL)//si no se puede abrir, print no se puede
   {
printf("no se puede abrir el fichero\n\n");
   }
else// se se abre correctamente print bien
   {
printf("Fichero abierto correctamente\n\n");
}



 ////////primer while para determinar el tamaño del macrovectorvector /////////segundo while para escribir el macrovector//////////
int z,w;
   int i=0;
   int j=0;
   int cont=0;
  while (!feof(f))
{
fscanf (f, "%d", &z);
printf ("Extraido:%d --", z);
cont=cont+1;
}
printf("\n\nNumero de dato leidos: %d\n\n",cont);
rewind(f);
cont= cont;
VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
while (!feof(f))
      {
```

```
        fscanf (f, "%d", &z);


        a[i]=z;
i=i+1;
        }
print_vector(a,cont);
 rewind(f);
////////////////////////////////////////
char parte1[] = "model:\n sets:\n job/1..";
char parte2[] = "/:t,pt,c;\n orden(job,job):x; \n endsets \n data: \n pt= ";
char parte3[] = "; \n v= 10000; \n enddata \n @for(job(k): @sum(job(j)|j#NE#k: x(k,j)) <= 1); \n
@for(job(k):x(k,1)=0); \n @for(job(j)|j#GT#1: @sum(job(k)|j#NE#k: x(k,j))>=1); \n " ;
char parte4[] = "@for(job(j): c(j) >= t(j)+pt(j)); \n @for(job(j): @for(job(k)|j#NE#k: t(k)>= c(j) - v*(1-
x(j,k)))); \n @for(job(j):@for(job(k): @BIN(x(j,k)))); \n MIN = @Sum(job(j):C(j)); \n  END";
FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );
if(fichero== NULL)
  {
  printf("\nERROR fichero lingo no se puede crear");
  }
else
   {
  printf("\nSe puede crear el fichero lingo");
}
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
fprintf (fichero, "%d", cont);
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
for (i = 0; i <cont; i++)
    fprintf (fichero, "%d ", a[i]);
fwrite(parte3 , 1 , sizeof(parte3) , fichero);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);
fclose(fichero);
return 0;
}
```

Modelo C: Código en C para 1| $|\sum_1^n T_j$:

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
FILE *f;  //Declara el la variable tipo fichero f
   f= fopen("data.txt","r"); // almacena f.txt en la variable f
if(f==NULL)//si no se puede abrir, print no se puede
   {
printf("no se puede abrir el fichero\n\n");
   }
else// se se abre correctamente print bien
   {
printf("Fichero abierto correctamente\n\n");
}
   ////////primer while para determinar el tamaño del macrovectorvector /////////segundo while para escribir el macrovector//////////
int z,w;
   int i=0;
   int j=0;
   int cont=0;
  while (!feof(f))
{
fscanf (f, "%d", &z);
printf ("Extraido:%d --", z);
cont=cont+1;
}
printf("\n\nNumero de dato leidos: %d\n\n",cont);
rewind(f);
cont= cont;
VECTOR_INT a= DIM_VECTOR_INT(cont);
printf("El vector leido es:\n");
while (!feof(f))
      {
      fscanf (f, "%d", &z);
      a[i]=z;
```

```
i=i+1;
      }
print_vector(a,cont);
 rewind(f);
VECTOR_INT pts= DIM_VECTOR_INT(cont/2);
VECTOR_INT dds= DIM_VECTOR_INT(cont/2);
for(i=0;i<(cont/2);i++)
{
   pts[i]=a[i];
}
trabajos=cont/2;
k=1+(cont/2);
for(i=0;i<(cont/2);i++)
{
   pts[i]=a[k];
   k=k+1;
}
 ////////////////////////////////////////
char parte1[] = "model:\n sets: \njob/1..";
char parte2[] = "/:t,pt,c,dd,tard;orden(job,job):x;\n endsets \n data: \n pt=";
char parte3[] = "; \n dd= ";
char parte4[] = ";\n v= 10000;\n enddata\n";
char parte5[] = "min=@sum(job(j):Tard(j));\n @for(job(k): @sum(job(j)|j#NE#k: x(k,j)) <= 1); \n
@for(job(k):x(k,1)=0); \n @for(job(j)|j#GT#1: @sum(job(k)|j#NE#k: x(k,j))>=1);\n";
char parte6[] = "@for(job(j): c(j) >= t(j)+pt(j)); \n @for(job(j): @for(job(k)|j#NE#k: t(k)>= c(j) - v*(1-
x(j,k)))); \n @for(job(j):@for(job(k): @BIN(x(j,k)))); \n ";
char parte7[]="@for(job(j): tard(j)>= c(j)-dd(j)); \n @for(job(j): tard(j)>=0);\n END";
  FILE * fichero;
fichero = fopen( "Lingo.txt" , "w" );
if(fichero== NULL)
  {
printf("\nERRORficherolingo no se puedecrear");
  }
else
   {
printf("\nSepuedecrear el ficherolingo");
}
fwrite(parte1 , 1 , sizeof(parte1) , fichero);
fprintf (fichero, "%d", trabajos);
```

```
fwrite(parte2 , 1 , sizeof(parte2) , fichero);
for (i = 0; i <trabajos; i++)
fprintf (fichero, "%d ", pts[i]);
fwrite(parte3 , 1 , sizeof(parte3) , fichero);
for (i = 0; i <trabajos; i++)
fprintf (fichero, "%d ", dds[i]);
fwrite(parte4 , 1 , sizeof(parte4) , fichero);
fwrite(parte5 , 1 , sizeof(parte5) , fichero);
fwrite(parte6 , 1 , sizeof(parte6) , fichero);
fwrite(parte7 , 1 , sizeof(parte7) , fichero);
fclose(fichero);
return 0;
}
```

Batería de datos para los problemas de Flow Shop:

http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html