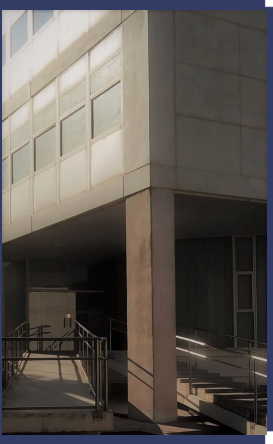
Proyecto Fin de Máster
Máster Universitario en Ingeniería de Análisis de Datos, Mejora de Procesos y Toma de Decisiones

# Modelos exactos para la secuenciación de máquinas paralelas no relacionadas con tiempos de cambio

Autor: Víctor Mirasierra Calleja

Tutor: Rubén Ruiz García



**Departamento de Estadística e Investigación Operativa Aplicadas y Calidad**
**Universitat Politècnica de València**

Valencia, 2020

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Proyecto Fin de Máster
Máster Universitario en Ingeniería de Análisis de Datos,
Mejora de Procesos y Toma de Decisiones

# Modelos exactos para la secuenciación de máquinas paralelas no relacionadas con tiempos de cambio


Autor:
Víctor Mirasierra Calleja

Tutor:
Rubén Ruiz García

Departamento de Estadística e Investigación Operativa
Aplicadas y Calidad
Universitat Politècnica de València



Valencia, 2020

# Agradecimientos

Quería agradecer este trabajo a todos mis compañeros del máster, en especial a Llorenç, Isabel, Jordi y Andrés. Mi estancia en Valencia ha sido genial gracias a ellos y han conseguido que la época de covid sea menos mala.

También quería agradecer a mi tutor Rubén y a Celia toda la ayuda que me han ofrecido. Sin ellos, este trabajo no sería lo mismo.

# Abstract

This work studies the unrelated parallel machines scheduling problem (UPMS) with setup times for every job, which are machine and sequence dependent. It will cover some of the solutions proposed over the time for the problem, which includes models based on mixed integer linear programming, heuristics, matheuristics and constraint programming. After this, a new exact model is proposed. It will be compared with some of the most relevant state of the art exact models. To this end, a computational cluster will be used to test the performance of the different models by means of data instances extracted from previous works from the literature.

Due to its growing interest, a constraint programming model has been included in this work. To this end, I have worked with the CPLEX integrated development environment (IDE) and the CP library, which is focused on constraint programming. The results from the constraint programming model have been compared with the ones of mixed integer linear programming and reaffirm the goodnesses of the constraint programming approach.

# Resumen

Este trabajo estudia el problema de secuenciación en máquinas paralelas no relacionadas con tiempos de cambio dependientes de la máquina y de la secuencia de proceso de los trabajos. Se estudiarán diversas soluciones al problema propuestas a lo largo del tiempo que incluyen modelos de programación lineal entera mixta, modelos heurísticos, matheurísticos y de programación por restricciones. Tras esto, se propondrá un modelo exacto que se comparará con los métodos exactos existentes más relevantes de la literatura. Para esto, se hará uso de un cluster de cómputo en el que se estudiará el desempeño de los diferentes modelos mediante conjuntos de datos extraídos de trabajos anteriores de la literatura.

Se ha decidido incluir el modelo de programación por restricciones en la comparación debido a su creciente interés. Para ello, se ha trabajado con el entorno de desarrollo integrado (IDE) de CPLEX y con la librería CP enfocada a la programación por restricciones. Los resultados obtenidos se han comparado con los resultados de las técnicas basadas en modelos de programación entera mixta y reafirman las bondades del enfoque basado en programación por restricciones.

# Índice

# Índice de Figuras

# Índice de Tablas

# 1 Introducción

## 1.1 Introducción

El problema de asignación y secuenciación de trabajos en diferentes máquinas es un problema que ha sido ampliamente estudiado por su interés industrial y por la variedad de posibilidades que ofrece. Aunque típicamente se trata desde una perspectiva teórica y se prueban los modelos con datos ficticios, el estudio de este tipo de problemas es fundamental, ya que los problemas estudiados en la literatura se pueden considerar simplificaciones de los problemas de asignación y secuenciación reales.

Los problemas de secuenciación se han estudiado mucho en la literatura, en [35] y [36] se recoge gran parte de la investigación en este campo y de la aplicación práctica de los problemas de este tipo en la industria. En esencia, los problemas de asignación y secuenciación (o simplemente de secuenciación) se reducen a, dadas una serie de tareas y de máquinas que procesan dichas tareas: establecer una asignación de las diferentes tareas a las diferentes máquinas y definir la secuencia de tareas en cada máquina tal que se minimice una función objetivo, como puede ser el tiempo en el que se realizan todas las tareas (makespan).

Así, se puede entender un problema de secuenciación como la suma de cuatro elementos:

- Tareas $\rightarrow$ Los trabajos que se desean realizar. Un problema de secuenciación puede tener un número cualquiera de tareas y estas pueden tener diferente naturaleza que requieran un procesado personalizado. Las tareas engloban operaciones muy variadas, desde una operación de pintura a una pieza hasta la salida de un barco de un puerto. A mayor número de tareas, mayor es la complejidad del problema y más costoso será encontrar una solución óptima.
- Recursos $\rightarrow$ Necesarios para realizar las tareas. El procesado de las tareas requiere una cantidad de recursos, que pueden incluir máquinas, unidades de materia prima, etc. Estos recursos pueden ser estacionarios o variar a lo largo del tiempo. También hay ciertos problemas en los que las tareas consume recursos antes o después de ser procesadas. Este es el caso cuando existen tiempos de preprocesado (conocidos tradicionalmente como tiempos de setup) o de posprocesado, que pueden estar justificados por ejemplo por la puesta a punto y limpieza de máquinas antes y después de la operación. Al igual que ocurre con las tareas, hay algunos recursos que pueden afectar significativamente a la complejidad del problema como puede ser el número de máquinas que procesan trabajos.

- Objetivo → Aquello que se pretende optimizar. En muchos casos son unidades de tiempo como el makespan, que se define como el tiempo necesario en procesar todas las tareas o el objetivo de earliness-tardiness, que penaliza la finalización de los trabajos en un instante diferente a una fecha de entrega objetivo establecida. Sin embargo, el objetivo de un problema de secuenciación no se limita a variables temporales, sino que también puede considerar los recursos del problema para por ejemplo minimizar la materia prima consumida o el número de máquinas empleado. Un enfoque más general sería definir la función objetivo como una mezcla de objetivos temporales y objetivos de recursos.

- Relaciones entre tareas → En muchos problemas, existen relaciones entre las tareas. Algunas tareas pueden requerir el procesado de otras antes de empezar a procesarse o pueden requerir empezar después de un cierto instante de tiempo. Las relaciones entre tareas pueden traducirse en forma de restricciones duras, es decir, de obligado cumplimiento, o pueden ser relaciones blandas, cuyo incumplimiento se traduzca en un incremento de la función objetivo.

La solución obtenida en un problema de secuenciación debe definir perfectamente el comienzo y la terminación de las diferentes tareas consideradas, así como la asignación de recursos a estas tareas. Una representación muy empleada para visualizar las soluciones son los diagramas de Gantt. Estos son diagramas bidimensionales de barras en los que se representa el tiempo en el eje de coordanadas horizontal y las máquinas encargadas de procesar las tareas en el eje vertical. Cada máquina está representada por una barra horizontal a la altura de la máquina que la procesa. El principio y el final de la barra representan los instantes de tiempo entre los que dicha tarea se procesa en dicha máquina. En ocasiones se asignan diferentes colores a las barras de las diferentes tareas para facilitar la visualización.

La figura 1.1 es un diagrama de Gantt que representa gráficamente una posible solución a un problema de secuenciación en el que se han asignado 15 tareas a 3 máquinas.

Aunque la premisa parece simple, la realidad es que el abanico de soluciones a los problemas de asignación y secuenciación es en general muy amplio y crece de forma exponencial, por lo que la mayoría de estos problemas, incluyendo el tratado en este trabajo, tienen una complejidad NP-hard incluso cuando tan sólo existen dos máquinas para procesar los trabajos. Esto hace que la investigación de diferentes modelos y algoritmos ([9], [26]) sea imprescindible para reducir el tiempo de resolución hasta hacerlo manejable. Hay que tener en cuenta también que los problemas de secuenciación incluyen multitud de problemas diferentes en función de si hay una o más máquinas, si hay tiempos de cambio entre tareas, si existen relaciones entre los trabajos o de la disponibilidad temporal de las máquinas, entre otras variantes. Estas variantes suelen añadir complejidad a los problemas, aunque en algunos casos añaden restricciones que permiten restringir el espacio de búsqueda de la

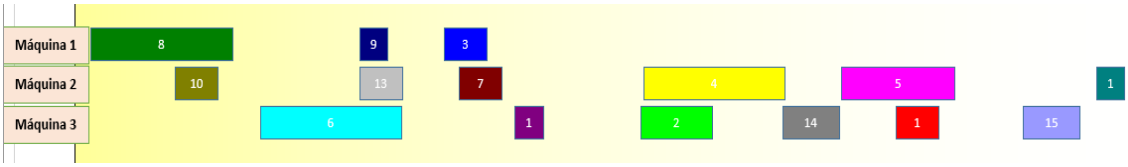



**Figura 1.1** Diagrama de Gantt de una solución a un problema de secuenciación..

solución, lo que conlleva una reducción del tiempo necesario para encontrar una buena solución o una solución óptima.

Debido a su complejidad, hay muchos trabajos que han optado por técnicas heurísticas para resolver este tipo de problemas de una forma aproximada pero computacionalmente manejable. Muchas de las heurísticas empleadas en este campo se han aplicado también a problemas muy diferentes de otros campos de la ciencia y la ingeniería y esa es una de las ventajas que ofrecen: versatilidad. Esta versatilidad permite resolver problemas de secuenciación muy diferentes con ligeros cambios en el algoritmo heurístico. Además, muchas técnicas heurísticas no requieren conocimientos avanzados del problema y son implementables por usuarios con perfiles menos especializados mediante herramientas de programación de alto nivel como Microsoft Excel.

Algunos de los trabajos que incorporan algoritmos heurísticos para resolver problemas de asignación y secuenciación en la literatura incluyen algoritmos genéticos [10][44][43], algoritmos basados en GRASP [6] o búsqueda tabú [12] entre otros.

Con el tiempo se han ido perfeccionando modelos exactos basados en programación matemática, hasta el punto de ser capaces de resolver problemas complejos de forma eficiente [33]. Estos modelos tienen la ventaja de ser capaces de converger a una solución óptima y poder comprobar su optimalidad. Sin embargo, a diferencia de los modelos heurísticos, son poco versátiles y es normalmente necesario cambiar el modelo completo cuando el problema cambia. También son generalmente más lentos para obtener una "buena" solución.

Así, también se han desarrollado métodos que mezclan la rapidez de los métodos heurísticos con la precisión de los métodos de programación matemática. Estos métodos, conocidos como matheurísticos, han conseguido buen desempeño en muchos problemas de asignación y secuenciación. Algunos ejemplos de trabajos que recogen modelos de este tipo son [15], [34] y [29].

El problema tratado en este trabajo se trata de la secuenciación de varios trabajos que deben ser procesados en una de las varias máquinas independientes disponibles. Esto significa que cada trabajo se procesa en un tiempo diferente dependiendo de la máquina en la que lo hace. Así, también existen tiempos de cambio (también conocidos como tiempos de setup) que suponen un tiempo de preparación de la máquina antes de realizar un trabajo. Estos tiempos dependerán tanto de las máquinas en las que se ejecutan los trabajos como de la secuencia de estos.

Nótese que esta variante del problema de secuenciación es bastante general y problemas como el mismo pero con máquinas idénticas o sin tiempos de setup pueden considerarse como casos particulares de éste.

### 1.1.1 Objetivos

El objetivo de este trabajo es el de resolver el problema de secuenciación de máquinas paralelas no relacionadas con tiempos de cambio dependientes de la máquina y la secuencia. Para ello, se hace un repaso de la literatura de los problemas de asignación y secuenciación de la literatura, así como los diferentes métodos que permiten la resolución de estos problemas. El trabajo se centra en la comparación de un tipo particular de modelos, los basados en programación lineal entera mixta. A estos modelos se le suman un modelo basado en programación por restricciones y un modelo propio de programación lineal entera mixta. Durante el trabajo se analizarán las características de los diferentes modelos,

así como la implementación seguida para resolver los sets de instancias presentados. Los resultados de los modelos obtenidos en un cluster de cómputo permitirán probar la eficiencia de los diferentes modelos ante diferentes tamaños de instancias de datos. El objetivo último del trabajo es el de comprobar si existe un método exacto más eficiente que el resto para resolver el tipo de problema estudiado.

### 1.1.2 Organización del trabajo

La estructura de este trabajo es la siguiente: En la sección 2 se detallarán las herramientas usadas para el estudio, así como la metodología de trabajo. Entre las herramientas se incluyen los programas y lenguajes empleados para la generación de archivos que recojan toda la información de los modelos, así como la conexión con el cluster de cómputo y el envío de estos archivos al cluster para que los evalúe. Tras esto, se detallará también el procesado de los resultados del cluster de cómputo. En esta sección también se incluirá una revisión del estado del arte, que incluye los modelos de programación lineal entera mixta y el modelo basado en programación por restricciones y se propondrá un nuevo modelo, unificando la notación de los modelos estudiados para facilitar la lectura. En la sección 3 se estudiarán y compararán los resultados obtenidos por los distintos métodos, haciendo un primer análisis de éstos, para en la sección 4 extraer las conclusiones finales en base a estos resultados. Finalmente se presentará la bibliografía empleada y los anexos que complementan la información presentada. Entre los anexos se incluirá todo el código desarrollado a lo largo del trabajo para conseguir implementar y evaluar los diferentes modelos.

# 2 Materiales y métodos

En este capítulo se detallarán los modelos estudiados, proporcionando primero una notación común que facilite la lectura de éstos, y explicando todas las variables y restricciones presentes en cada modelo. Estos modelos incluyen tanto los modelos de programación lineal entera mixta (MILP) de la literatura, como el modelo propuesto en este trabajo y el modelo basado en programación por restricciones. Tras la explicación de los modelos MILP, se mostrará una tabla que resume las variables presentes en cada uno, así como el número de restricciones. El modelo basado en programación por restricciones se basa en un paradigma de programación diferente, por lo que se ha excluido de la tabla comparativa. Una vez expuestos todos los modelos que se tratarán en el trabajo, se dará información sobre la implementación, haciendo especial énfasis en el lenguaje de programación utilizado, así como el software empleado para generar los ficheros. Se comentarán también las instancias utilizadas para ensayar los modelos y se comentará su estructura. Para terminar, se comentarán los detalles del cluster de cómputo, mencionando las características técnicas de los equipos que ejecutan los modelos, así como de la conexión remota con este cluster y los ficheros necesarios para que el cluster ejecute los modelos y extraiga los resultados procedentes.

## 2.1 Definición del problema

El problema tratado en este trabajo es el problema de secuenciación de máquinas paralelas no relacionadas con tiempos de cambio. Este problema está incluido en la categoría de problemas de secuenciación de máquinas paralelas, que es uno de los problemas más estudiados de secuenciación debido a su importancia y a la variedad de situaciones que ofrece.

Los problemas de secuenciación de máquinas paralelas proponen un escenario en el que se disponen $m$ máquinas que se utilizan para procesar $n$ trabajos (Figura 2.1). El tiempo que tardan estos trabajos puede ser el mismo en todas las máquinas (en este caso se denomina problema de máquinas idénticas) o puede variar en función de la máquina en la que se procese el trabajo (máquinas no relacionadas). Así, el tiempo de procesado de una tarea $i$ en una máquina $j$ es siempre un dato conocido y definitorio de la instancia y se suele representar como $p_{i,j}$. La formulación de un problema de máquinas paralelas tendrá la siguiente forma:

**Minimizar:** función objetivo (2.1a)

**Sujeto a:** (2.1b)

- Cada máquina sólo puede procesar una tarea en cada instante de tiempo. (2.1c)

- El tiempo de procesado de la tarea $i$ en la máquina $j$ viene dado por $p_{i,j}$. (2.1d)

Esta forma de formular el problema con lenguaje natural es muy similar a la empleada en la formulación del modelo basado en programación por restricciones y es una de las virtudes de este paradigma de la programación.

Como se ha mencionado, aunque todos los problemas de máquinas paralelas compartan la estructura de 2.1, existen multitud de problemas diferentes según la función objetivo considerada o las restricciones adicionales que se consideren. Además existen multitud de enfoques a cada uno de los problemas. Algunos trabajos buscan metaheurísticas que resuelvan el problema rápidamente, otros buscan acotar el problema de forma más exacta y otros buscan una formulación matemática eficiente.

Así, existen multitud de trabajos en la literatura que han estudiado problemas de asignación y secuenciación de máquinas paralelas. En [28] se presenta un algoritmo con complejidad polinómica que consigue acotar la solución de un problema de máquinas paralelas con $m$ máquinas y $n$ trabajos, cuyo objetivo de minimizar el makespan. Se considera que el procesado de cada tarea depende únicamente de la máquina a la que esté asignada, siendo el tiempo de procesado de la tarea $j$ en la máquina $i$ igual a $p_{i,j}$. En [27] se trata también el problema de secuenciación de máquinas paralelas, pero, a diferencia del artículo anterior, considera que puede haber máquinas no disponible desde el principio, lo que supone una generalización. Al igual que el anterior trabajo, se estudia acotar el problema bajo estudio con el algoritmo propuesto. En [42] se resuelve también el problema de máquinas paralelas, pero desde un enfoque difuso y considerando una función objetivo multiobjetivo. Este acercamiento tiene en cuenta la incertidumbre innata al tiempo de procesado de las tareas y consigue formular un algoritmo que aproxime la frontera de Pareto del objetivo establecido. Otro acercamiento posible es introduciendo heurísticas como es el caso de [16]. Este trabajo busca minimizar el tiempo máximo de completación de las tareas (makespan) y con ese fin se proponen varias metaheurísticas que consiguen reducir de forma rápida el tamaño del problema original y consiguen soluciones estadísticamente mejores que las

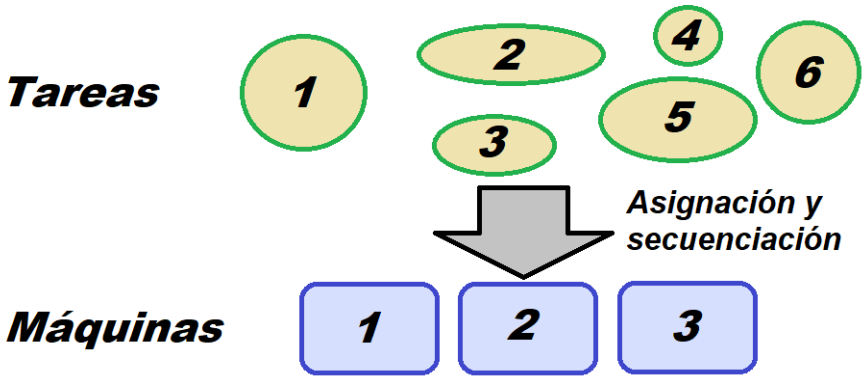



**Figura 2.1** Esquema que representa un problema de asignación y secuenciación de máquinas paralelas.

propuestas hasta entonces en la literatura. En [39] se opta por una metaheurística conocida como *iterated greedy* para resolver el problema de máquinas paralelas con una función objetivo que minimiza los tiempos de completación ponderados de las tareas. También puede considerarse el deterioro de las máquinas a lo largo del tiempo como es el caso de [31]. En este trabajo se considera el tiempo de procesado de las tareas como una función creciente en el tiempo y minimiza una función bi-criterio que depende tanto del tiempo como del coste del deterioro.

Dentro de los problemas de máquinas paralelas, se puede incluir la presencia de tiempos de cambio para generar un problema más cercano a la realidad. Estos tiempos de cambio consideran que es necesario dejar un plazo de tiempo antes del inicio de las tareas. En aplicaciones prácticas, este tiempo puede modelar la limpieza y puesta a punto de la máquina antes de realizar la tarea en cuestión. Los tiempos de cambio pueden depender de las tareas y de las máquinas, así como de la secuencia de tareas. Esta última posibilidad es muy útil para modelar por ejemplo la necesidad de tiempos de cambio pequeños cuando la máquina procesa dos tareas similares seguidas, frente a la necesidad de tiempos de cambio elevados cuando las tareas procesadas sucesivamente son muy distintas, ya que es este caso haría falta una puesta en marcha de la máquina mayor.

En cuanto a la formulación del problema, seguiría una estructura similar a la del problema 2.1, pero incorporando los recién definidos tiempos de cambio:

**Minimizar:** función objetivo (2.2a)

**Sujeto a:** (2.2b)

- Cada máquina sólo puede procesar una tarea en cada instante de tiempo. (2.2c)
- El tiempo de procesado de la tarea $i$ en la máquina $j$ viene dado por $p_{i,j}$. (2.2d)
- El tiempo de cambio entre las tareas $j$ y $k$ en la máquina $i$ viene dado por $s_{i,j,k}$. En este tiempo, no se podrán procesar tareas en dicha máquina. (2.2e)

Los problemas de secuenciación con tiempos de cambio también se ha estudiado en profundidad en la literatura. En [11] se estudia un algoritmo basado en la metaheurística VNS que permite resolver el problema de secuenciación de máquinas paralelas con tiempos de cambio dependientes de la secuencia. En [41] se resuelve un problema similar con una función bi-ojetivo mediante un algoritmo genético. Otro acercamiento heurístico al problema se encuentra en [21], donde se opta por la metaheurística búsqueda tabú para resolver el problema. Un acercamiento mediante la heurística de simulated annealing se recoge en [23], que considera tiempos de cambio dependientes de la secuencia de tareas pero no de la máquina en la que se procesan. En [24] se opta por un enfoque de lotes que aprovecha el hecho de que los trabajos similares se procesan típicamente seguidos para reducir el tiempo de cambio y presenta heurísticas que consideran este hecho en su búsqueda de soluciones. En [43] se opta por un acercamiento de algoritmo genético para resolver el problema de máquinas paralelas no relacionadas con tiempos de cambio.

El problema tratado en este trabajo es el de asignación y secuenciación de máquinas paralelas no relacionadas con tiempo de cambio dependientes de la máquina y la secuencia de tareas (Problema 2.2). El objetivo es el de minimizar el tiempo de completación de las tareas, es decir, el makespan. Para resolver el problema, se han analizado diferentes modelos de programación matemática exactos (no basados en heurísticas).

## 2.2 Notación

En esta sección se detallarán todos los índices y variables usados en los diferentes modelos, que se presentarán a lo largo de los siguientes apartados. Aunque algunos modelos difieran mucho entre sí, se ha unificado la notación lo máximo posible con el fin de facilitar la lectura. Así, esta sección sirve como glosario para los diferentes modelos.

Para empezar, los índices empleados para las máquinas, trabajos y orden son:

- $i$ designa las máquinas que procesan los trabajos.
- $j$,$k$ designan las tareas/trabajos a procesar.
- $t$ indica el orden en el que se procesa una tarea en una máquina.

Así, se definen también los siguientes parámetros característicos de las diferentes instancias del problema:

- Número de máquinas $m$.
- Número de trabajos $n$.
- Tiempo de procesado de cada trabajo ($j$) en cada máquina ($j$) $p_{i,j}$.
- Tiempo de setup de los diferentes trabajos ($k$) en función del trabajo que le precede ($j$) y la máquina que lo procesa ($i$) $s_{i,j,k}$.

Respecto a los tiempos de setup es necesario comentar la existencia de la desigualdad triangular. En algunas implementaciones prácticas, se considera que dadas tres tareas $j$,$k$,$\ell$ asignadas a una máquina $i$, entonces se debe cumplir la siguiente desigualdad triangular:

$$s_{i,j,k} \leq s_{i,j,\ell} + p_{i,\ell} + s_{i,\ell,k},\ i \in \{1,2,\ldots,m\}$$

Esto es una restricción sobre los datos del problema y establece que para cada máquina $i$, el tiempo de setup entre las tareas $j$ y $k$ debe ser menor o igual al tiempo empleado entre las mismas tareas si se coloca una tarea entre ellas $\ell$. Así, el tiempo de setup entre las tareas $j$ y $k$ es menor o igual al tiempo de setup entre las tareas $j$ y $\ell$, más el tiempo de procesado de la tarea $\ell$ más el tiempo de setup entre las tareas $\ell$ y $k$. Normalmente los datos cumplen esta desigualdad triangular, pero en caso de no cumplirse, no se podrán implementar aquellos modelos que requieran su cumplimiento.

En este trabajo no se han recogido modelos que requieran la satisfacción de esta desigualdad, pero en modelos como el propuesto en [4] sí es necesario garantizar su cumplimiento.

Tras esto, se definen diferentes sets que agrupan trabajos y máquinas. Estos sets simplificarán la notación de los diferentes modelos:

- $M = 1,\ldots,m$ es el set que recoge todas las $m$ máquinas consideradas.
- $N = 1,\ldots,n$ es el set que recoge todos los $n$ trabajos.
- $N_0 = 0 \cup N$ es el set que recoge todos los trabajos reales y los trabajos dummy 0.

Los trabajos dummy contemplados en la notación $N_0$ son trabajos ficticios con tiempo de procesado nulo empleados para marcar la primera tarea de cada máquina. Así, una tarea será la primera en ser procesada en una máquina cuando sea la sucesora del trabajo dummy de dicha máquina.

Es necesario también identificar las variables que se utilizarán en los diferentes modelos. Estas variables cambian dependiendo de los modelos, pero la mayoría de los modelos usan una notación similar que se recoge a continuación:

- $X_{i,j,k} = 1$ si $k$ sucede a $j$ en la máquina $i$, cero en otro caso.
- $Y_{i,j} = 1$ si $j$ es procesada en la máquina $i$, cero en otro caso.
- $C_j \geq 0$ es el tiempo de completación de la tarea $j$.
- $C_{i,j} \geq 0$ es el tiempo de completación de la tarea $j$ en la máquina $i$.
- $U_j \in \mathbb{Z}$ es un límite inferior al número de trabajos procesados antes que $j$ en la misma máquina en la que se procesa $j$.
- $C_{\text{máx}}$ es el tiempo de completación máximo (makespan).

La constante $V$ aparece en algunos modelos para establecer la ruptura de subrutas y se identifica con un valor numérico suficientemente grande.

Por otra parte, el modelo propuesto en la sección 2.3.4 enfoca el problema desde otra perspectiva, lo que se traduce en unas variables diferentes que consideran la posición de las tareas dentro de la misma máquina en lugar de solamente las relaciones de precedencia. Las variables empleadas en el modelo propuesto son las siguientes:

- $X_{i,j,t} = 1$ si el trabajo $j$ se procesa en la máquina $i$ en el paso $t$, cero en otro caso.
- $\tilde{S}_{i,j,k} \geq 0$ es el valor del tiempo de setup ($s_{i,j,k}$) entre las tareas $j$ y $k$ en la máquina $i$ si la tarea $j$ precede a la tarea $k$ en la máquina $i$, cero en caso contrario.

Por último, queda concretar la notación necesaria para definir el modelo CP. El principal cambio de este modelo es la inclusión de variables intervalares y secuencia, así como la incorporación de funciones de librería que sirven como restricciones. La variables intervalares son variables que incorporan diferentes atributos de un trabajo, como el inicio, final y el tiempo de procesado. Cada uno de estos atributos requeriría una variable diferente en la programación matemática tradicional, sin embargo, en el paradigma de la programación de la restricciones se pueden englobar bajo el marco de las variables intervalares. A continuación se detallará la notación de las nuevas variables:

- $Z_j$ variable intervalar asociada al trabajo $j$. Contiene información del inicio, fin y duración de este trabajo.
- $W_{i,j}$ variable intervalar opcional que está presente cuando el trabajo $j$ se asigna a la máquina $i$.
- $Q_j = \{Y_{1,j}, Y_{2,j}, \ldots, Y_{M,j}\}$ set de variables intervalares que representa las posibles máquinas a las que se le puede asignar el trabajo $j$.
- $V_d = \{Y_{i,1}, Y_{i,2}, \ldots, Y_{i,N}\}$ set de variables intervalares que representa los posibles trabajos que se pueden asignar a la máquina $i$.

## 2.3 Modelos MIP

Una vez detallada la notación, se utilizará para definir los diferentes modelos exactos estudiados en este trabajo.

### 2.3.1 Modelo estándar

El modelo estándar [20] [43] fue el primer modelo MIP propuesto para resolver el problema de UPMS.

El modelo estándar se define de la siguiente forma:

$$\text{mín} \quad C_{\text{máx}} \tag{2.3a}$$

$$\text{s.a.} \quad \sum_{i\in M}\sum_{\substack{j\in N_0\\ j\neq k}} X_{i,j,k} = 1, \quad k \in N \tag{2.3b}$$

$$\sum_{i\in M}\sum_{\substack{k\in N\\ j\neq k}} X_{i,j,k} \leq 1, \quad j \in N \tag{2.3c}$$

$$\sum_{k=1}^{N} X_{i,0,k} \leq 1, \quad i \in M \tag{2.3d}$$

$$\sum_{\substack{\ell=0\\ \ell\neq k,\ell\neq j}}^{N} X_{i,\ell,j} \geq X_{i,j,k}, \quad j \in N_0, k \in N, j \neq k, i \in M \tag{2.3e}$$

$$C_{i,k} + V(1 - X_{i,j,k}) \geq C_{i,j} + s_{i,j,k} + p_{i,k}, \quad j \in N_0, k \in N, j \neq k, i \in M \tag{2.3f}$$

$$C_{i,0} = 0, \quad i \in M \tag{2.3g}$$

$$C_{\text{máx}} \geq C_{i,j}, \quad j \in N, i \in M \tag{2.3h}$$

$$X_{i,j,k} \in \{0,1\}, C_{i,k} \geq 0., \tag{2.3i}$$

La restricción 2.3b establece que cada trabajo $k$ se procesa en una máquina de $M$ y está precedida por otro trabajo de $N_0$. En 2.3c se fija que cada trabajo $j$ tiene como máximo un sucesor en $N$ y que se procesa como mucho en una máquina de $M$. 2.3d establece que en cada máquina de $M$ existe como mucho un trabajo de $N$ que es el primero en ser procesado. En 2.3e se impone que si un trabajo $j$ perteneciente a $N$ precede a otro trabajo $k$ en cualquier máquina, entonces debe haber un tercer trabajo $\ell$ en esa misma máquina perteneciente a $N_0$ que preceda a $j$. 2.3f establece que si el trabajo $k$ sucede al $j$, entonces el tiempo de completación de $k$ es al menos el tiempo de completación de $j$ más el tiempo de setup entre $j$ y $k$ más el tiempo de procesamiento del trabajo $k$. En la restricción 2.3g se fija que el trabajo dummy de cada máquina tiene un tiempo de completación de 0 y en 2.3h se impone que el makespan no es menor que ninguno de los tiempos de completación de los trabajos.

Así, se observa que se tienen un total de $M(N+1)^2+1$ variables, de las cuales $MN(N+1)$ son variables binarias y un total de $2[M+N+MN^2]+MN$ restricciones, de las cuales $MN^2$ son restricciones que incluyen la constante $V$.

Al ser la primera formulación matemática del problema, el desempeño de este modelo es mejorable. Las restricciones que incluyen la constante $V$ se conocen como restricciones disyuntivas y son difíciles de tratar, puesto que para ello es necesario generar relajaciones

lineales en el modelo de programación lineal entera mixta. Estas relajaciones no son muy eficientes y reducen notablemente el desempeño de los modelos que las incorporan, incrementando considerablemente el tiempo de resolución de las diferentes instancias. Es por ello, que en modelos más recientes se haya optado por unas restricciones diferentes que no incluyen esta constante.

Aún con esto, el modelo estándar tiene la ventaja de no depender de la satisfacción de la desigualdad triangular ($s_{i,j,k} \leq s_{i,j,\ell} + p_{i,\ell} + s_{i,\ell,k},\ i \in \{1,2,\ldots,m\}$), lo que permite la resolución de un abanico de problemas mayor al de modelos como el de [4].

### 2.3.2 Modelo AAA

El modelo AAA propuesto en [3] es más eficiente que el modelo estándar e incorpora las variables y algunas de las restricciones que usarán algunos modelos con mejor desempeño.

El modelo AAA se define de la siguiente forma:

$$\text{mín} \quad C_{\text{máx}} \tag{2.4a}$$

$$\text{s.a.} \quad \sum_{j \in N_0} \sum_{\substack{k \in N \\ k \neq j}} s_{i,j,k} X_{i,j,k} + \sum_{j \in N} p_{i,j} Y_{i,j} \leq C_{\text{máx}}, \quad i \in M \tag{2.4b}$$

$$\sum_{k \in N} X_{i,0,k} \leq 1, \quad i \in M \tag{2.4c}$$

$$\sum_{i \in M} Y_{i,j} = 1, \quad j \in N \tag{2.4d}$$

$$Y_{i,j} = \sum_{\substack{k \in N_0 \\ j \neq k}} X_{i,j,k}, \quad i \in M, j \in N \tag{2.4e}$$

$$Y_{i,k} = \sum_{\substack{j \in N_0 \\ j \neq k}} X_{i,j,k}, \quad i \in M, k \in N \tag{2.4f}$$

$$C_k - C_j + V(1 - X_{i,j,k}) \geq s_{i,j,k} + p_{i,k}, \quad j \in N_0, k \in N, j \neq k, i \in M \tag{2.4g}$$

$$C_0 = 0 \tag{2.4h}$$

$$C_{\text{máx}} \geq C_j, \quad j \in N \tag{2.4i}$$

$$X_{i,j,k} \in \{0,1\}, Y_{i,j} \geq 0, C_j \geq 0. \tag{2.4j}$$

En 2.4b se definen las cotas inferiores del makespan, en 2.4c se establece que como mucho se asigna un trabajo a la primera posición de cada máquina (después del trabajo dummy 0). 2.4d establece que cada trabajo se procesa únicamente en una máquina. Las restricciones 2.4e fijan que todos los trabajos tienen un sucesor (este modelo incluye trabajos dummy al principio y final de cada máquina) y las restricciones 2.4f establecen que tienen un predecesor. En 2.4g se establece el orden de procesado y la ruptura de subtours. En 2.4h se fija el tiempo de completación de los trabajos dummy en 0. En las restricciones 2.4i se establece el makespan como el tiempo de terminación del trabajo más tardío y en 2.4j se define la naturaleza de las variables. Es interesante destacar que, gracias a la forma de definir el modelo con las restricciones 2.4d-2.4f, no es necesario imponer que las variables $Y_{i,j}$ sean binarias, lo que contribuye en gran medida al mejor desempeño de este modelo.

Así, se observa que se tienen un total de $M(N+1)^2+2+N-M$ variables, de las cuales $MN(N+1)$ son variables binarias y un total de $2[M+N+MN]+MN^2+1$ restricciones, de

las cuales $MN^2$ son restricciones que incluyen la constante $V$. Observamos que el número total de variables es algo mayor que en el modelo estándar, aunque el número de variables binarias es el mismo. El número de restricciones totales del modelo AAA es menor que en el modelo estándar, aunque el el número de restricciones que incluye la constante $V$ es el mismo.

El problema está definido por las expresiones 2.4b-2.4g y 2.4j, mientras que el resto de restricciones son cortes al subespacio original con el fin de acotar la búsqueda de la solución. En las siguientes secciones se estudiarán los modelos propuestos en [17], los cuales comparten con el modelo anterior las restricciones 2.4b-2.4f y añaden otras para mejorar la eficacia del la resolución.

### 2.3.3 Modelos recientes

En esta sección se recogen los modelos propuestos en [17]. La idea es partir de las expresiones 2.4b-2.4f del modelo AAA y añadir a estas restricciones para la ruptura de subrutas que terminen de definir el problema y restricciones para acotar el espacio de búsqueda que permitan acelerar la búsqueda de la solución. La definición de subrutas es característica del famoso problema del vendedor viajero (TSP) y aparece en multitud de problemas de complejidad NP por analogía a éste. En lo que respecta a este trabajo, las restricciones de ruptura de subrutas consiguen eliminar ciertas soluciones infactibles del problema, por lo que no incluirlas podría dar una solución infactible.

Todos los modelos considerados en esta sección partirán del siguiente modelo base, al que añadirán nuevas restricciones:

$$\text{mín} \quad C_{\text{máx}} \tag{2.5a}$$

$$\text{s.a.} \quad \sum_{j\in N_0}\sum_{\substack{k\in N\\ k\neq j}} s_{i,j,k}X_{i,j,k} + \sum_{j\in N} p_{i,j}Y_{i,j} \leq C_{\text{máx}}, \quad i \in M \tag{2.5b}$$

$$\text{s.a.} \quad \sum_{k\in N} X_{i,0,k} \leq 1, \quad i \in M \tag{2.5c}$$

$$\sum_{i\in M} Y_{i,j} = 1, \quad j \in N \tag{2.5d}$$

$$Y_{i,j} = \sum_{\substack{k\in N_0\\ j\neq k}} X_{i,j,k}, \quad i \in M, j \in N \tag{2.5e}$$

$$Y_{i,k} = \sum_{\substack{j\in N_0\\ j\neq k}} X_{i,j,k}, \quad i \in M, k \in N \tag{2.5f}$$

$$X_{i,j,k} \in \{0,1\}, Y_{i,j} \geq 0, U_j \in \{0,1,\ldots,N\}. \tag{2.5g}$$

Nótese que, al no incluir las restricciones 2.4g-2.4j, las variables $C_j$ no son necesarias y se pueden sustituir por otras, que en este caso serán las variables $U_j \in \mathbb{Z}$. Estas variables suponen un límite inferior al número de trabajos procesados antes que $j$ en la misma máquina en la que se procesa $j$. Las variables $U_j$ aparecen por analogía al problema del vendedor viajero, donde es común definir las variables $U_j$ para romper las subrutas.

El modelo (incompleto) 2.5a-2.5g tiene el mismo número de variables totales que el modelo AAA ($M(N+1)^2+2+N-M$ variables), aunque el número de variables binarias/enteras aumenta debido a la incorporación de las nuevas variables $U_j$ a un total de

$MN(N+1)+1$. En cuanto a restricciones se tienen $2M+N+2MN$ restricciones, de las cuales ninguna incluye la constante $V$.

#### Restricciones de ruptura de subrutas

Las restricciones MTZ y DL inspiradas en los trabajos [32] y [13] respectivamente, son dos alternativas diferentes a la expresión 2.4f para la eliminación de subrutas. Cualquiera de estas alternativas terminará de definir el modelo 2.5 para que su solución converja a la solución óptima del problema UPMS.

#### MTZ

Las restricciones MTZ aseguran que si una tarea $k$ sucede a otra $j$ en cualquier máquina, entonces $U_k \geq U_j + 1$. Estas restricciones se han usado tradicionalmente en el problema del viajero (TSP) para romper subrutas. Matemáticamente, la restricción MTZ se expresa de la siguiente forma:

$$U_j - U_k + n \sum_{i \in M} X_{i,j,k} \leq n-1, \quad j,k \in N, j \neq k. \tag{2.6}$$

Las restricciones MTZ añaden $N^2$ restricciones al problema base.

#### DL

La segunda de las restricciones que sustituyen a 2.4g es la propuesta en [13] y abreviada como DL. Estas restricciones también han sido ampliamente estudiadas en el problema del viajero y establecen una relación entre las variables X y U más fuerte que las restricciones MTZ. Las restricciones DL vienen dadas por la siguiente expresión:

$$U_j - U_k + n \sum_{i \in M} X_{i,j,k} + (n-2) \sum_{i \in M} X_{i,k,j} \leq n-1, \quad j,k \in N, j \neq k. \tag{2.7}$$

Al igual que las restricciones MTZ, las restricciones DL añaden $N^2$ restricciones al problema base.

#### Restricciones de acotación de búsqueda

El problema 2.5 junto con una de las restricciones de ruptura de subrutas (2.6 o 2.7) es suficiente para definir el problema de UPMS, pero existen unas desigualdades que permiten reducir el conjunto de búsqueda de soluciones sin perder soluciones óptimas potenciales. Estas desigualdades se conocen como KB y AM y, al igual que las restricciones de ruptura de subrutas, se añadirán al problema 2.5. Las restricciones KB y AM imponen que cuando un trabajo $j$ sea el primero en una máquina, entonces $U_j = 0$, independientemente de que dicho trabajo sea también el último de su máquina o no.

#### KB

Las restricciones KB fueron propuestas en [22] y adaptadas al problema de UPMS en [17]. Las restricciones KB son las siguientes:

$$U_j + (n-2) \sum_{i \in M} X_{i,0,j} - \sum_{i \in M} X_{i,j,0} \leq n-2, \quad j \in N \tag{2.8}$$

$$U_j + \sum_{i \in M} X_{i,0,j} \geq 1, \quad j \in N. \tag{2.9}$$

Las restricciones KB añaden $2N$ restricciones al problema base.

### AM

Las restricciones AM fueron propuestas en [17] como una alternativa a las restricciones KB. Su formulación matemática es la siguiente:

$$U_j + (n-1) \sum_{i \in M} X_{i,0,j} \geq n-1, \quad j \in N. \tag{2.10}$$

$$U_j + \sum_{i \in M} X_{i,0,j} \geq 1, \quad j \in N. \tag{2.11}$$

Al igual que las restricciones KB, las restricciones AM añaden $2N$ restricciones al problema base.

### 2.3.4 Modelo Propuesto

En este trabajo proponemos un nuevo modelo con unas variables diferentes a los modelos anteriores. Hasta ahora, todos los modelos estudiados cuentan con una notación similar, si no idéntica, en la que se consideran variables binarias $X_{i,j,k}$ que toman un valor de 1 si existe la sucesión de tareas $j$-$k$ en la máquina $i$ y toman un valor de 0 si no existe tal sucesión. Esta notación ha comprobado ser muy práctica porque enlaza muy bien con los tiempos de cambio característicos del problema, ya que éstos dependen de la secuencia de tareas que se muestra explícita en la variable $X_{i,j,k}$.

Sin embargo, esta notación puede parecer algo difícil de entender, ya que la notación no especifica el orden global de las tareas, sino que expresa relaciones de precedencia entre las tareas dos a dos. Esto puede dificultar en gran medida la implementación de algunas restricciones que exijan el procesado de determinadas tareas en un orden determinado. Además, puesto que toda la investigación encontrada de modelos exactos se ha desarrollado en torno a las mismas variables, es posible que unas variables diferentes abran una línea de investigación con mejores resultados. Con este fin, se ha decidido desarrollar un modelo con variables basadas en el orden global de las tareas dentro de las máquinas en lugar de relaciones de precedencia dos a dos.

La forma de plantear el nuevo modelo es mediante la inclusión del índice temporal $t$, que define el orden en el que se procesa una tarea en una máquina. Así, la primera tarea de cada máquina siempre tendrá asociado $t = 1$, mientras que la última tendrá un valor de $t$ igual al número de tareas que haya asignadas a esa máquina. Este índice temporal hace que las nuevas variables de decisión tomen la forma $X_{i,j,t}$, que toman un valor 1 si el trabajo $j$ se procesa en la máquina $i$ en la posición $t$-ésima y un valor 0 en caso contrario.

Nótese que el número de posiciones $t$ está limitado al número de tareas existentes ($n$) ya que, puesto que el objetivo es minimizar el tiempo de completación de las tareas, no óptimo que una posición esté vacía (sin tareas asignadas). Así, siguiendo la misma lógica, una tarea podrá ser procesada en la posición $t$ de una máquina sólo si hay otra tarea procesada en la posición $t-1$ en la misma máquina o si es la primera tarea en procesarse en la máquina ($t = 1$). Estas propiedades hacen que la estructura de las soluciones sea particular. Las variables con un valor de $t$ bajo serán más proclives a tener un valor de 1 que variables con un valor de $t$ alto. Visto desde un punto de vista práctico, si el número de tareas $n$ es mayor que número de máquinas $m$ y en la solución óptima existe una tarea que se procesa en la última posición de una máquina, $X_{i,j,N} = 1$, entonces dicha máquina debe tener un desempeño considerablemente mayor al del resto de máquinas para que colocar todas las tareas en dicha máquina requiera menos tiempo que distribuirlas a entre las $M$ máquinas

disponibles. Un argumento similar se puede emplear para posiciones altas, ya que expresan un desequilibrio fuerte entre las máquinas.

El caso de $n > m$ es habitual en este tipo de problema. Aunque no es necesario que se cumpla esta condición, es la que más se encuentra en este tipo de problemas en la práctica y la considerada en las instancias de este trabajo.

Las consideraciones sobre la estructura de la solución óptima no se han tenido en cuenta en este trabajo, pero se consideran muy interesantes para futuras líneas de investigación sobre el tema, especialmente para técnica matheurísticas que estudien primero las configuraciones más probables de las soluciones.

Una vez expuesta la idea del modelo, se procede a su definición matemática, que toma la siguiente forma:

$$\text{mín} \quad C_{\text{máx}} \tag{2.12a}$$

$$\text{s.a.} \quad \Big(\sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n}\tilde{S}_{i,j,k}\Big)+\Big(\sum_{j=1}^{n}\sum_{t=1}^{n}p_{i,j}X_{i,j,t}\Big)\leq C_{\text{máx}}, \quad \forall i \tag{2.12b}$$

$$\sum_{i=1}^{m}\sum_{t=1}^{n}X_{i,j,t}=1, \quad \forall j \tag{2.12c}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}X_{i,j,t}\leq m, \quad \forall t \tag{2.12d}$$

$$\sum_{j=1}^{n}X_{i,j,t}\leq 1, \quad \forall i,t \tag{2.12e}$$

$$s_{i,j,k}\Big(X_{i,j,t}+X_{i,k,t+1}-1\Big)\leq\tilde{S}_{i,j,k},\ \forall i,j,k,\ j\neq k, t\in 1,\ldots,n-1 \tag{2.12f}$$

$$\sum_{j}X_{i,j,t}\geq\sum_{j}X_{i,j,t+1}, \quad \forall i,t\in 1,\ldots,n-1 \tag{2.12g}$$

$$X_{i,j,t}\in\{0,1\} \tag{2.12h}$$

$$C_{\text{máx}},\tilde{S}_{i,j,k}\geq 0 \tag{2.12i}$$

En 2.12a se minimiza el makespan. En 2.12b se definen las cotas inferiores del makespan. La ecuación 2.12c establece que cada tarea se debe ejecutar en una sola máquina y en una sola posición temporal $t$. En 2.12d se establece que el número de tareas en la posición $t$ debe ser como mucho igual al número de máquinas, ya que cada una tiene como mucho una tarea en la posición $t$. La restricción 2.12e fuerza que dos tareas no se realicen en la misma máquina en el mismo tiempo. La restricción 2.12f establece los costes de setup reales mediante la variable continua $\tilde{S}$, esto es, si una tarea $j$ precede a una $k$ en la máquina $i$, entonces $\tilde{S}_{i,j,k}=s_{i,j,k}$, mientras que si no existe esa precedencia, entonces no se aplica ningún coste setup para la configuración dada y $\tilde{S}_{i,j,k}=0$. Para no perder la linealidad del modelo, estas relaciones se establecen mediante cotas inferiores de la variable $\tilde{S}$. Por último, 2.12g indica que si hay un trabajo que se procesa en el puesto $t$-ésimo de una máquina, debe haber un trabajo en el puesto $t-1$-ésimo de esa máquina.

La variable continua positiva $\tilde{S}$ se ha diseñado para incorporar los tiempos de setup al coste, ya que la formulación actual no explicita las relaciones de sucesión necesarias para definir los tiempos de setup de la solución. La expresión $X_{i,j,t}+X_{i,k,t+1}-1$ solo es positiva

cuando $k$ es sucesora de $j$, y en ese caso tiene valor 1. Para acotar la variable $\tilde{S}_{i,j,k}$ se ha empleado este término multiplicado por $s_{i,j,k}$. Así, si $k$ no sucede a $j$, entonces $\tilde{S}_{i,j,k}$ tendría la cota inferior en 0, ya que $\tilde{S}_{i,j,k}$ se define como variable positiva en 2.12i. Para minimizar el makespan, $\tilde{S}_{i,j,k}$ acabará tomando siempre el valor de su cota inferior.

Con lo anterior, tenemos que si la tarea $k$ sucede a la tarea $j$, $\tilde{S}_{i,j,k}$ tomaría valor de $s_{i,j,k}$ y se sumaría al makespan como indica la formulación del problema. En otro caso, tomaría un valor nulo y no repercutiría en el tiempo de completación.

Así, el número total de variables del problema vendría dado por $X_{i,j,t}$, $\tilde{S}_{i,j,k}$ y $C_{\text{máx}}$ y tendría un total de $2mn^2+1$ variables, de las cuales $mn^2$ serían enteras. Aunque el número de variables enteras sea menor que el de otros modelos, el número de variables total es mayor cuando el número de trabajos es elevado ($n$ es alto). Aún así, el inconveniente de este modelo parece encontrarse en el número total de restricciones, que depende cúbicamente del número de tareas. Esto es algo que en cierta medida se esperaba, ya que el acercamiento tradicional expresaba las variables $X_{i,j,k}$ de forma muy similar a los tiempos de cambio con el fin de reducir el número de restricciones.

### 2.3.5 Cuadro resumen

Una forma de resumir los diferentes modelos es representar el número de variables y restricciones. Esta medida es meramente orientativa y no concluyente, ya que muchas de las restricciones pueden estar orientadas a reducir el espacio de búsqueda, por lo que no incluirlas empeoraría el desempeño del modelo. Aún así, da información previa a la implementación de los modelos que puede servir para interpretar mejor los resultados obtenidos.

La tabla a continuación recoge el número de variables y restricciones de los modelos MILP recogidos en este trabajo:

**Tabla 2.1** Información sobre los modelos MILP estudiados.

| Modelo | Variables | Variables enteras | Restricciones | Restricciones-V |
|---|---|---|---|---|
| ST | $m(n+1)^2+1$ | $mn(n+1)$ | $2[m+n+mn^2]+mn$ | $mn^2$ |
| AAA | $m(n+1)^2+2+n-m$ | $mn(n+1)$ | $2[m+n+mn]+mn^2+1$ | $mn^2$ |
| DL | $m(n+1)^2+2+n-m$ | $mn(n+1)+N$ | $2[m+mn]+n^2+n$ | |
| DL-KB | $m(n+1)^2+2+n-m$ | $mn(n+1)+N$ | $2[m+n+mn]+n^2+n$ | |
| DL-AM | $m(n+1)^2+2+n-m$ | $mn(n+1)+N$ | $2[m+n+mn]+n^2+n$ | |
| MTZ | $m(n+1)^2+2+n-m$ | $mn(n+1)+N$ | $2[m+mn]+n^2+n$ | |
| MTZ-KB | $m(n+1)^2+2+n-m$ | $mn(n+1)+N$ | $2[m+n+mn]+n^2+n$ | |
| MTZ-AM | $m(n+1)^2+2+n-m$ | $mn(n+1)+n$ | $2[m+n+mn]+n^2+n$ | |
| Propuesto | $2mn^2+1$ | $mn^2$ | $mn^3-mn^2+2mn+m+n$ | |

Se observa que todos los modelos a excepción del propuesto tienen un número de variables totales y enteras aproximadamente igual, esto es debido a que, como se ha comentado previamente, emplean unas variables muy parecidas enfocadas en la secuencia de las tareas. Sin embargo sí se encuentra una diferencia significativa en las restricciones. Los modelos ST y AAA poseen $mn^2$ restricciones que contienen la constante $V$. Como ya se ha mencionado, estas restricciones son difíciles de tratar, por lo que podemos esperar un peor desempeño de estos modelos. En cuanto al número de restricciones totales parece

que el modelo estándar ST tiene mayor número de ellas debido al término $2mn^2$, aunque la diferencia tampoco parece demasiado grande.

Respecto al modelo propuesto, presenta un número inferior de variables, tanto enteras como totales y no presenta restricciones que contengan la constante $V$. Sin embargo, el número de restricciones total depende cúbicamente del número total de trabajos en la forma $mn^3$. Aunque se le sustrae el término $mn^2$, el número de restricciones cuando el número de trabajos sea elevado va a ser superior al resto de modelos. Esto no implica necesariamente un desempeño peor, pero sí un consumo de memoria mayor.

## 2.4 Modelo CP

Los modelos presentados hasta ahora están formulados mediante programación matemática estándar. Sin embargo, en los últimos años se ha desarrollado un tipo de programación conocido como constraint programming (CP) [40] [2]. La programación mediante CP se basa en la descripción de las relaciones entre las diferentes variables y se ha usado con éxito en problemas grandes, especialmente en problemas combinatorios en las áreas de planificación y secuenciación [5][14].

Los modelos incluidos en el paradigma de CP están definidos por un set de variables, otro set con los valores admitidos para las variables (dominio) y un conjunto de relaciones entre las variables (restricciones). Los problemas de optimización se definen normalmente como un conjunto de problemas de satisfacción de las restricciones (CSP). Estos problemas buscan una combinación de valores de las variables que satisface todas las restricciones del problema. La solución óptima del problema de optimización sería aquella solución que además optimiza una función objetivo dada.

Respecto a la programación matemática tradicional, en CP se usan las restricciones de forma activa. De esta forma, no se usan sólo para comprobar si una solución es válida, sino que también se usan para acotar el espacio de búsqueda de soluciones. Típicamente las restricciones utilizadas sirven para deducir nuevas restricciones que acoten aún más el espacio de trabajo, lo que se conoce como propagación de restricciones.

Además de la propagación de restricciones, es común incorporar heurísticas que añadan más restricciones que permitan acotar aún más el espacio de búsqueda. La búsqueda de soluciones se hace típicamente mediante árboles de decisión, que pueden utilizar varios métodos de propagación hacia atrás para trabajar con las diferentes restricciones.

Al definir un modelo CP mediante una librería especializada como la librería CPOptimizer del solver IBM ILOG CPLEX utilizada en este trabajo, ésta utiliza funciones de restricciones que tienen asociados algoritmos de propagación. Esto es posible gracias al principio de localidad, que expone que cada restricción se propaga de forma tan local como sea posible, siendo esta propagación independiente de la existencia de otras restricciones.

Debido a la posibilidad de resolver problemas combinatorios de gran escala y su interés creciente a lo largo de los años, se ha decidido incluir en este trabajo el modelo CP propuesto en [19] y compararlo con los modelos MILP recogidos en la anterior sección. Debido al cambio de paradigma en la programación, la notación de este modelo incluye variables intervalares tal y como se indicó en la sección 2.2.

Además de estas variables, el modelo CP también incluye las siguientes funciones de librería:

- endOf($\cdot$) indica el fin de la variable intervalar que recibe como argumento.

- Alternative(·,·) establece que la variable intervalar que recibe como primer argumento es una (y solo una) de las variables intervalares contenidas en el set que recibe como segundo argumento.
- NoOverlap(·,·) se asegura que las variables intervalares contenidas en el set del primer argumento no se solapen y dejen un espacio entre ellos dado por la distancia de transición del segundo argumento.
- Cumulative(·,·,·) sirve para llevar la cuenta del nivel de cierto recurso a lo largo del tiempo. El tercer argumento es el límite impuesto al recurso que se mide.
- Pulse(·) se emplea dentro de la función Cumulative para incrementar o disminuir el nivel de recursos en una unidad.

### 2.4.1 Formulación modelo CP

La formulación de un modelo con el paradigma de programación por restricciones es muy cercana a la forma de definir el problema con lenguaje natural, lo que hace que el modelo sea fácilmente interpretable, frente a los modelos de programación matemática más tradicionales.

El modelo planteado en esta sección es el propuesto en [19] y tiene la siguiente forma:

$$\begin{aligned}
\text{mín} \quad & C_{\text{máx}} && \text{(2.13a)}\\
\text{sujeto a} \quad & C_{\text{máx}} \geq \text{endOf}(Z_j), \quad j \in J && \text{(2.13b)}\\
& \text{Alternative}(Z_j, Q_j), \quad j \in N && \text{(2.13c)}\\
& \text{NoOverlap}(V_i, ST^i_{j,k}), \quad i \in M && \text{(2.13d)}\\
& \text{Cumulative}(Z_j, \text{Pulse}(Z_j), m) && \text{(2.13e)}\\
& C_{\text{máx}} \geq \text{LB} && \text{(2.13f)}\\
& C_{\text{máx}} \geq 0 && \text{(2.13g)}\\
& \text{LB1} = \frac{1}{m} \sum_{j \in N} \min_{i \in M, k \in N} [p_{i,j} + s_{i,k,j}] && \text{(2.13h)}\\
& \text{LB2} = \max_{j \in N} \{ \min_{i \in M, k \in N} [p_{i,j} + s_{i,k,j}] \} && \text{(2.13i)}\\
& \text{LB} = \max\{\text{LB1}, \text{LB2}\}. && \text{(2.13j)}
\end{aligned}$$

Vemos como en 2.13b se establece que el makespan está acotado por la finalización de cada tarea. La restricción 2.13c asegura que el trabajo $j$ se asigna únicamente a una de las máquinas a las que se le puede asignar. En 2.13d se consigue que los trabajos asignados a cada máquina no se solapen y dejen entre ellos un tiempo definido por la matriz de distancia de transición $ST^i_{j,k}$, que contiene la información de los tiempos de setup $s_{i,j,k}$. La restricción 2.13e limita el número de trabajos activos en cualquier instante de tiempo al número de máquinas existentes. Por último, las restricciones 2.13f-2.13j establecen una cota inferior al makespan.

La búsqueda de las soluciones a este modelo se realizará mediante un árbol de búsqueda y las herramientas comentadas al principio de la sección, por lo que la comparación del número de restricciones entre los modelos MILP y el modelo CP no tienen interés. La

comparación del número de variables tampoco aporta información, ya que el modelo CP incorpora tipos de variable que no son posibles en la programación matemática tradicional. Así, para comparar este modelo con los anteriores habrá que estudiar su desempeño con las instancias propuestas.

## 2.5 Materiales y experimentación

Para entender el material empleado, es necesario comprender los diferentes problemas que se han realizado. En la sección anterior se han explicado diferentes modelos que resuelven el problema de asignación y secuenciación de máquinas paralelas con tiempos de cambio, pero estos modelos necesitan instancias para ejecutarse. Las instancias incluyen la información de los parámetros característicos del problema particular bajo estudio y son por tanto necesarias para terminar de definir el problema.

Como se indicó en la sección 2.2, estos parámetros son el número de máquinas $m$, el número de trabajos $n$, los tiempos de ejecución de los trabajos en las diferentes máquinas $p_{i,j}$ y los tiempos de cambio de los diferentes trabajos en función de la máquina y la secuencia $s_{i,j,k}$. Los conjuntos de estos datos que definen los diferentes problemas reciben el nombre de instancias. Aunque se resuelvan con el mismo modelo, el tamaño de las instancias repercute directamente en la resolución de un problema. En el caso que nos compete, al aumentar el número de máquinas y trabajos, la complejidad del problema aumenta de tal forma que los modelos pasan de obtener la solución óptima en décimas de segundo a no obtenerla en una hora de ejecución del modelo.

Para probar los modelos, se han utilizado dos conjuntos de instancias. Las primeras son instancias pequeñas para comprobar el correcto funcionamiento de los modelos cuando el problema es relativamente pequeño. Estas instancias son un total de 640 y contemplan un número de trabajos entre 6 y 12 y un número de máquinas entre 2 y 5. Estas instancias se han extraído de la web del grupo de investigación SOA [1] y se representan en la Tabla 2.2.

**Tabla 2.2** Distribución del número de instancias pequeñas en función del número de máquinas disponibles y el número de tareas a procesar.

| | | Número de trabajos | | | |
|---|---|---|---|---|---|
| | | 6 | 8 | 10 | 12 |
| Número de máquinas | 2 | 40 | 40 | 40 | 40 |
| | 3 | 40 | 40 | 40 | 40 |
| | 4 | 40 | 40 | 40 | 40 |
| | 5 | 40 | 40 | 40 | 40 |

El segundo grupo de instancias consideradas como medianas fueron utilizadas en [38] y está formado por 180 instancias con un número de trabajos comprendido entre los 20 y los 120 trabajos y un número de máquinas comprendido entre las 2 y 12 máquinas. En la tabla 2.3 se representa la distribución de estas instancias.

**Tabla 2.3** Distribución del número de instancias medianas en función del número de máquinas disponibles y el número de tareas a procesar.

| | | Número de trabajos | | | | |
|---|---|---|---|---|---|---|
| | | 40 | 60 | 80 | 100 | 120 |
| Número de máquinas | 2 | 15 | 15 | 15 | 15 | 15 |
| | 4 | 0 | 15 | 15 | 15 | 15 |
| | 6 | 0 | 0 | 0 | 15 | 15 |
| | 8 | 0 | 0 | 0 | 0 | 15 |

Existen infinitas combinaciones posibles de trabajos y máquinas a considerar. En este trabajo nos hemos centrado en conjuntos pequeños y medianos en los que el número de trabajos es mayor al número de máquinas. Los ensayos con instancias grandes y con instancias con mayor número de máquinas que de trabajos (más asociados a problemas de asignación) también son de interés, pero su estudio queda fuera de este trabajo.

Las instancias utilizadas se presentan en forma de fichero de texto (*".txt"*) y comparten la misma estructura, detallada en la figura 2.7. En ella se puede ver una primera línea que indica el número de trabajos y de máquinas disponibles. El tercer término especifica el número de etapas, que no aplica al tipo de problema de este estudio. Tras esto, la siguiente línea contiene un número que indica la etapa (que tampoco aplica a este tipo de problemas), seguida de una tabla que muestra los tiempos de procesado de cada una de las tareas en cada máquina. Tras esto, se presentan nuevas tablas que representan los tiempos de setup para cada una de las máquinas y cada una de las secuencias de dos trabajos posibles.

```
6  2  1
2
        0       1       1       4
        0       87      1       21
        0       28      1       68
        0       32      1       17
        0       38      1       43
        0       9       1       48
SSD
M0
0       1       8       1       3       9
4       0       7       3       7       8
7       3       0       2       3       5
3       8       3       0       5       2
8       3       7       9       0       5
8       8       1       2       2       0
M1
0       5       1       6       1       7
6       0       7       7       6       2
7       6       0       9       6       9
3       7       3       0       1       7
5       8       5       6       0       9
7       4       1       7       9       0
```

**Figura 2.2** Ejemplo de una instancia con 6 trabajos y 2 máquinas disponibles.

Una vez definido completamente el problema a resolver y expuestos los modelos y las instancias que se desean estudiar, es necesario hablar de la implementación de todo esto en el marco de los experimentos. Esto engloba la definición y creación de los ficheros necesarios para las ejecutar pruebas que resuelvan los problemas de secuenciación, así como aquellos necesarios para la extracción de resultados.

### 2.5.1 Definición de los experimentos

La definición de experimento en el contexto de este trabajo difiere mucho de la definición del campo de diseño de experimentos. En nuestro caso, un experimento se puede definir como el ensayo mediante el cual se resuelve una instancia del problema. En nuestro caso los experimentos se realizarán en un cluster de cómputo, por lo que es imprescindible que el experimento contenga los ficheros necesarios para definir y resolver la instancia del problema. Los ficheros que definen los experimentos realizados, así como la extensión de éstos se muestran en la figura 2.3 y son:

- Ficheros de instancia → compuestos por el modelo y la instancia. Estos pueden estar incluidos en un único archivo con extensión *".lp"* o, como será el caso de los experimentos con el modelo CP, venir en archivos separados. Al implementar el modelo CP, el modelo tendrá extensión *".mod"* y los datos de la instancia tendrá extensión *".dat"*. En ambos casos, los ficheros de instancia contienen toda la información que define el problema.
- Fichero de configuración → contiene información relevante para el solver que resolverá la instancia, como puede ser el tiempo máximo permitido para resolver la instancia, el número de hilos empleados para resolver el problema o la tolerancia de éste. Se puede definir un fichero de configuración para cada experimento o definir ficheros de configuración que engloben a varios experimentos. Los ficheros de configuración empleados en este trabajo son ficheros de texto plano (*".txt"*) como el mostrado en la figura 2.4.
- Ejecutable → es el archivo que pone en marcha el experimento. Para este trabajo se ha utilizado un archivo ejecutable con extensión *".bat"* compuesto por comandos de Windows que permiten dar comienzo al experimento. Se han utilizado dos ficheros ejecutables distintos, uno para los modelos MILP (Figura 2.5) y otro para el modelo CP (Figura 2.6).

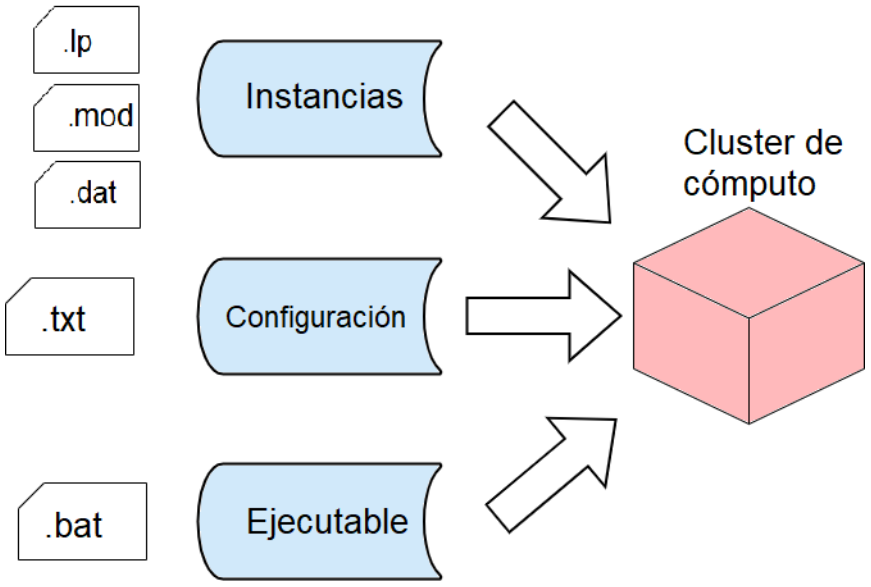



**Figura 2.3** Esquema con los archivos que componen un experimento.

```
set timelimit 3600
set mip display 0
set threads 4
set MIP TOLERANCES ABSMIPGAP 0.999999

READ .\instances\I_12_4_S_1-49_6_ST.LP
set logfile .\Results\I_12_4_S_1-49_6_STCPLEX.txt
mipopt
display solution variables 1-
set logfile *

READ .\instances\I_10_4_S_1-9_6_ST.LP
set logfile .\Results\I_10_4_S_1-9_6_STCPLEX.txt
mipopt
display solution variables 1-
set logfile *
```

**Figura 2.4** Ejemplo de un fichero de configuración empleado para un experimento de una instancia mediana del modelo CP.

```
@ECHO OFF
setlocal enabledelayedexpansion
@echo Cplex needs to find the 'Results' folder so we will to create it
md Results
@echo Searching for txt files in folder
for /f "tokens=*" %%a in ('dir ^"*.txt^" /b /s') do @echo config file: %%a & Start /w Cplex -f "%%a"
```

**Figura 2.5** Ejecutable empleado para los experimentos de los modelos MILP.

```
echo Cplex needs to find the 'Results' folder so we will to create it
md Results
echo Searching for txt files in folder

for %%F in (./Config*.txt) do (
     set file=%%~nxF
  GOTO endstep
)
:endstep

for /f "delims=" %%x in (%file%) do set params=%%x

echo Executing ...
Start /w oplrun %params%
```

**Figura 2.6** Ejecutable empleado para los experimentos del modelo CP.

Para la generación de los ficheros de instancias, se ha realizado una separación en función del tipo de programación empleada. Entre los ficheros MILP, tenemos por una parte los modelos del estado del arte recogidos en [17]. En este trabajo, se empleó una aplicación que permite generar los ficheros *".lp"* que contienen los modelos y las instancias. Esta aplicación se muestra en la figura 2.7. En ella, podemos generar los ficheros *".lp"* a partir de los datos de las instancias sin procesar. Para ello, sólo habría que indicar el modelo deseado junto con las desigualdades oportunas (sólo en el caso de los modelo MTZ y DL) de la lista ofrecida, seleccionar la opción "Only .lp" para que devuelva el fichero demandado

y pulsar “Launch”. La aplicación permite también la resolución de las instancias, para lo que ofrece opciones de configuración. Esta opción es útil cuando hay pocas instancias que evaluar, pero para este trabajo no se ha utilizado la opción de resolver los modelos desde esta aplicación.

Los autores de [17] también utilizaron otra aplicación para crear los ficheros de configuración *“.txt”*. Ambas aplicaciones, así como el fichero del ejecutable de los modelos me han sido proporcionadas por los autores del trabajo, gesto que agradezco mucho.

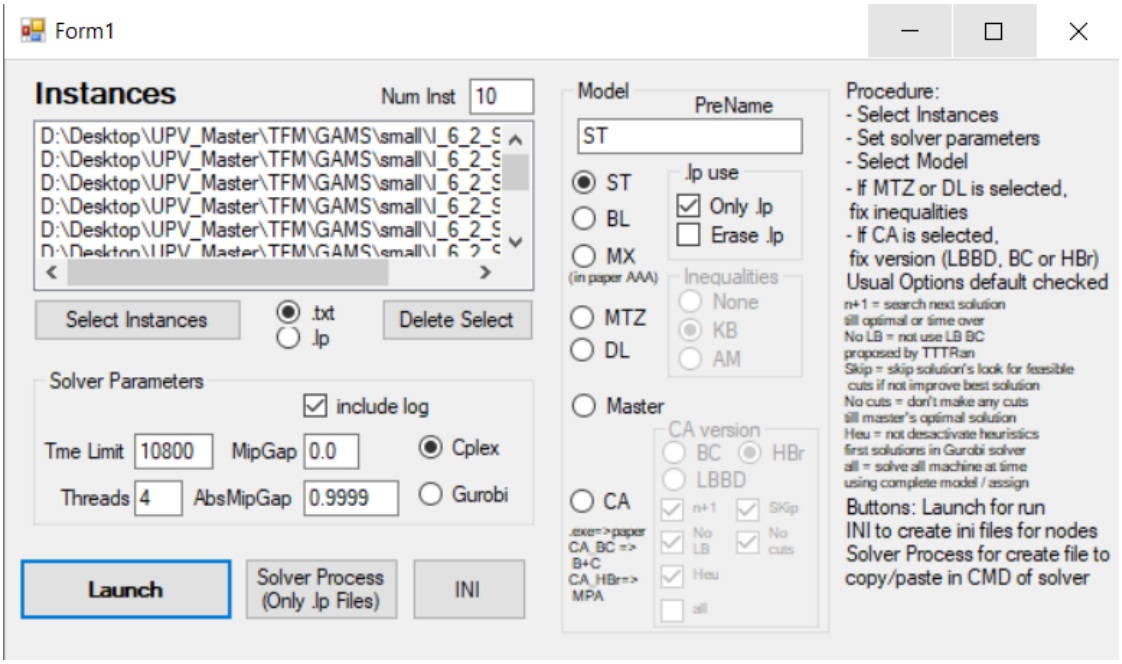



**Figura 2.7** Captura del ejecutable que genera los ficheros “.lp” para algunos de los modelos de optimización matemática tratados en [17].

Para la generación del resto de ficheros, en este trabajo se ha desarrollado código que permite generar el resto de ficheros necesarios para lanzar los experimentos. Esto incluye la generación de los ficheros de instancias *“.lp”* y de configuración *“.txt”* para el modelo propuesto, así como los ficheros de instancias *“.lp”* y *“.dat”* y de configuración *“.txt”* del modelo CP. El código empleado se detallará más adelante en esta sección.

Para entender la generación de los ficheros *“.lp”* es necesario conocer su origen. Estos ficheros son el resultado de la programación con lenguaje matemático GNU MathProg (también conocido como GMPL) [30] de los modelos. Este lenguaje se emplea para describir modelos de programación matemática lineal y contiene el conjunto de expresiones matemáticas y conjuntos de datos especificado por el usuario. El lenguaje GMPL consigue un desempeño mejor que otros lenguajes como R o Python ya que está mejor optimizado para los problemas de optimización matemática lineal.

### Lenguaje GNU MathProg

El lenguaje GMPL admite los problemas de programación lineal (LP), así como problemas de programación lineal entera mixta, siempre que estos problemas estén definidos de la siguiente forma:

Minimizar o maximizar una función objetivo $f(\boldsymbol{x})$ de la forma:

$$f(\boldsymbol{x}) = c_0 + c_1 x_1 + c_2 x_2 + \ldots + c_n x_n.$$

Sujeto a una serie de restricciones lineales expresadas como:

$$\begin{aligned} \mathrm{LB}_1 &\leq a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n \leq \mathrm{UB}_1 \\ \mathrm{LB}_2 &\leq a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n \leq \mathrm{UB}_2 \\ &\ldots\ldots\ldots \\ \mathrm{LB}_m &\leq a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n}x_n \leq \mathrm{UB}_m. \end{aligned} \tag{2.14}$$

Y cotas para las variables:

$$\begin{aligned} \mathrm{lb}_1 &\leq x_1 \leq \mathrm{ub}_1 \\ \mathrm{lb}_2 &\leq x_2 \leq \mathrm{ub}_2 \\ &\ldots\ldots \\ \mathrm{lb}_n &\leq x_1 \leq \mathrm{ub}_n. \end{aligned} \tag{2.15}$$

Donde:

- $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$ son las variables del problema.
- $f(\boldsymbol{x})$ es la función objetivo, dependiente de las variables.
- $\boldsymbol{c} = [c_0, c_1, c_2, \ldots, c_n]^T$ son los coeficientes de la función objetivo.
- $A = \begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{bmatrix}$ son los coeficientes de las restricciones lineales.
- $\mathrm{LB}_1, \mathrm{LB}_2, \ldots, \mathrm{LB}_m$ son las cotas inferiores de las restricciones.
- $\mathrm{UB}_1, \mathrm{UB}_2, \ldots, \mathrm{UB}_m$ son las cotas superiores de las restricciones.
- $\mathrm{lb}_1, \mathrm{lb}_2, \ldots, \mathrm{lb}_m$ son las cotas inferiores de las variables.
- $\mathrm{ub}_1, \mathrm{ub}_2, \ldots, \mathrm{ub}_m$ son las cotas superiores de las variables.

Una vez definido el esquema de los problemas que GMPL es capaz de modelar, se definen los objetos que el lenguaje es capaz de procesar. Estos son:

- Set → Set de datos.
- Parameter → Parámetros del problema.
- Variable → Variables del problema.
- Constraint → Restricciones del modelo.
- Objective → Función de coste.

Para especificar un elemento del objeto, se utilizan subíndices. Esto es, para referirnos a la variable i-ésima, usaremos la notación x[i]. Los objetos escalares debido a su naturaleza no requieren subíndices.

### 2.5.2 Software empleado

El código desarrollado para la implementación del modelo propuesto mediante lenguaje algebraico se ha realizado con el software libre GAMS [7] [8]. Debido a la dificultad de este lenguaje para importar los ficheros originales de las instancias, se ha realizado el proceso de escritura del código GAMS a través de un programa de R, para lo cual se ha empleado el software RStudio [37]. Este programa consigue generar los ficheros *".gms"* que contienen tanto los modelos como las instancias. Puesto que el formato *".gms"* es propio de GAMS, no es válido para ejecutarse en el cluster de cómputo, ya que no tiene instalado este software. Se emplea entonces un fichero batch *".bat"* (Figura 2.8) creado también mediante código de R, que ejecuta los modelos *".gms"* creados para generar los ficheros *".lp"*.

```
gams I_10_2_S_1-124_1_gms_executable.gms
gams I_10_2_S_1-124_10_gms_executable.gms
gams I_10_2_S_1-124_2_gms_executable.gms
gams I_10_2_S_1-124_3_gms_executable.gms
gams I_10_2_S_1-124_4_gms_executable.gms
gams I_10_2_S_1-124_5_gms_executable.gms
gams I_10_2_S_1-124_6_gms_executable.gms
gams I_10_2_S_1-124_7_gms_executable.gms
gams I_10_2_S_1-124_8_gms_executable.gms
```

**Figura 2.8** Recorte del fichero empleado para ejecutar en lote los ficheros .gms.

El último de los modelos estudiados en este trabajo es el modelo de constraint programming (CP) propuesto en [19]. Para este modelo se ha usado el software CPLEX Studio IDE 12.10.0 [18] con la libería CP [25]. En este caso, la información del modelo no se recoge en un único fichero *".lp"*, sino en dos ficheros: un *.mod* que recoge el modelo y un *".dat"* que recoge los datos de las instancias. Para generar estos dos ficheros se ha utilizado nuevamente código propio de R incluido en la sección 5, que ha permitido traducir las instancias iniciales al formato requerido por el fichero *".dat"* y crear los modelos necesarios para cada instancia.

También queda escribir los ficheros de configuración de los experimentos. Para ello, se ha empleado también código R que escriba el fichero de texto plano (Sección 5.2.3) y fije los parámetros de configuración para establecer un límite temporal del experimento a 3600 segundos (1 hora), el número de hilos a 4 y la tolerancia en términos de GAP a 0.999999. Así, el fichero de configuración también debe leer los diferentes archivos y establecer un fichero log que recoja información del experimento.

Todo el código generado en este trabajo se encuentra en la sección de anexos al final del documento.

### 2.5.3 Hardware empleado

Los experimentos han sido ejecutados en un cluster de máquinas virtuales similar al empleado en [17], aunque con diferentes especificaciones. En este trabajo, se han usado máquinas virtuales con 2 núcleos de procesamiento virtuales y 8GBytes de memoria cada uno. Las máquinas virtuales ejecutan el sistema operativo Windows 10 Enterprise 64 bits. Las máquinas virtuales se manejan mediante una plataforma de virtualización OpenStack de 12 blades, cada uno con cuatro procesadores 12-core AMD Opteron Abu Dhabi 6344 corriendo a 2.6GHz. y 256 Gbytes de RAM cada una. La asignación de experimentos a

las diferentes máquinas se ha realizado mediante la distribución aleatoria de experimentos con el fin de agilizar la completación de la experimentación sin recurrir a la computación paralelizada.

La comunicación con el cluster se hace mediante la aplicación Cliente (Figura 2.9). Esta aplicación muestra todos los nodos del cluster con un color que muestra el estado de cada nodo, siendo el verde el color asignado a los nodos libres. Cada nodo representa a cada una de las máquinas virtuales especificadas arriba. Tras elegir los nodos en los que se realizarán los trabajos, la aplicación pide definir el experimento mediante un ejecutable, un archivo de configuración y un archivo de instancias.

Como ejecutable se ha utilizado el fichero *".bat"* de la figura 2.5 o de la figura 2.6, dependiendo del modelo empleado. El fichero de configuración es un archivo de texto que se ha generado mediante código de R (Sección 5.2.3). Para terminar de definir el experimento se incluyen los ficheros de instancias. Estos ficheros cambian dependiendo del método y como se ha comentado pueden tener extensión *".lp"* o extensión *".mod"* y extensión *".dat"*.

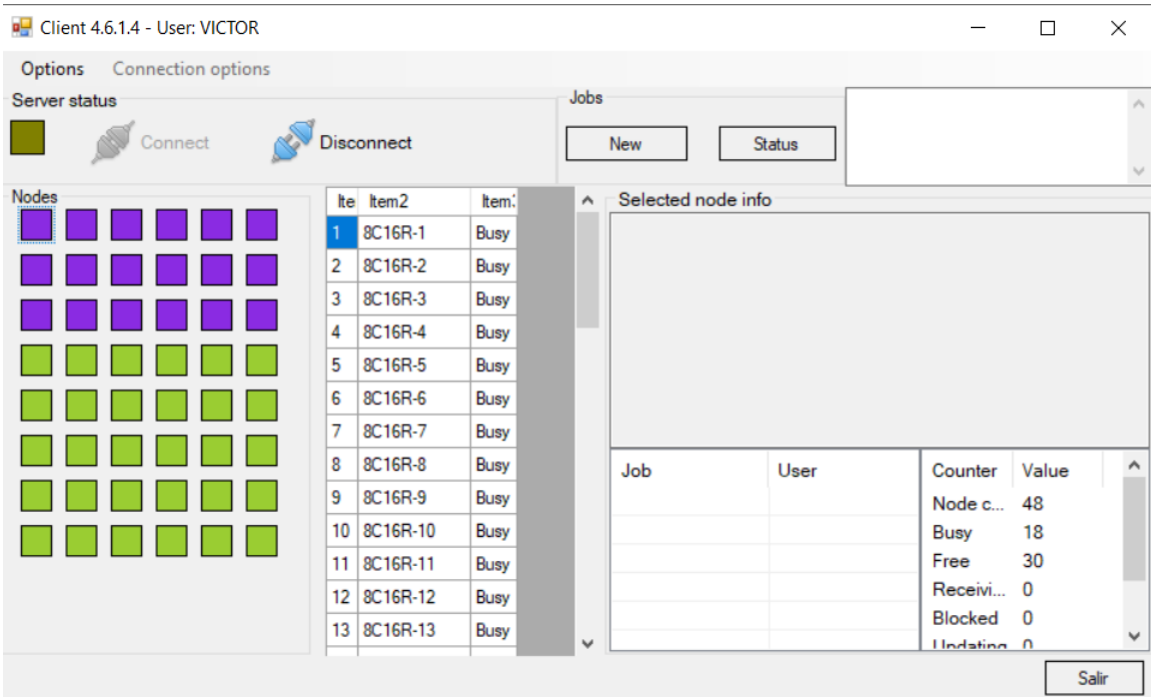



**Figura 2.9** Captura de pantalla de la aplicación Cliente empleada para comunicarse con el cluster de cómputo.

Una vez entregados todos los ficheros necesarios, se empiezan a ejecutar los experimentos. La aplicación muestra información sobre el estado de las diferentes máquinas virtuales, indicando cuando están activas, recibiendo un trabajo, en espera o ha habido un error en ellas.

Para poder analizar los resultados, se deben interpretar primero los ficheros que ofrece el cluster. Estos ficheros son un documento log en formato *".txt"* con la información relacionada con el envío, recepción y correcta ejecución de las tareas por el cluster y un comprimido *".7z"*, que al descomprimirlo revela un documento de texto *".txt* por cada instancia. El formato y contenido de estos documentos de texto viene definido por el archivo que contiene el modelo, pero en todos los casos incluye la solución obtenida, el valor final

de la función objetivo, la mejor cota encontrada de la función objetivo y el tiempo de CPU del experimento. En los modelos MILP estudiados, se ha empleado una aplicación para resumir todos los ficheros de soluciones en uno sólo denominado "Solution_Report" que contiene en cada línea el nombre de una instancia, el valor obtenido de la función objetivo, el estado de finalización del experimento, el gap y el tiempo requerido para encontrar la solución. En cuanto al modelo CP, se ha generado código R mostrado en la sección de anexos que genera un "Solution_Report" similar, pero únicamente con la información del valor de la función objetivo alcanzado y el tiempo de CPU consumido.

Si todo va bien, al terminar los experimentos se podrá descargar una carpeta comprimida con los resultados y ficheros logs de los experimentos. Estos ficheros tendrán toda la información sobre la solución, que se recogerá en un archivo Excel que permita comparar los resultados de los diferentes modelos estudiados.

# 3 Resultados

En esta sección se comentarán los resultados obtenidos de los experimentos detallados en la sección anterior con el fin de comparar todos los modelos estudiados.

Una vez generados los ficheros de "Solution_Report", se ha creado un fichero excel en el que se utilizan los resultados obtenidos por los diferentes modelos y se comparan para calcular la mejor solución de cada instancia y con ella, la métrica Relative Percentage Deviation o RPD. La métrica RPD se define como:

$$\text{RPD} = 100\frac{C_{\text{máx}}(\text{Modelo}) - C_{\text{máx}}(\text{Mejor solución})}{C_{\text{máx}}(\text{Mejor solución})},$$

es decir, es una comparación entre el valor de la función objetivo obtenido por el modelo con el mejor valor conocido. Así, será un número real positivo que tomará el valor 0 cuando la solución del modelo converja al óptimo o sea la mejor solución conocida para esa instancia, y tomará un valor alto si la solución del modelo difiere mucho de la mejor solución conocida.

Además de la métrica RPD, se representa también el tiempo medio que tardan los modelos en llegar a la solución obtenida. Este tiempo junto con el RPD consiguen resumir el desempeño de los modelos en dos valores escalares: el primero ofrece información sobre "lo buena que es la solución obtenida por el modelo", mientras que el segundo se centra en lo rápido que llega el modelo a esa solución. En la práctica se dan ocasiones donde se tiene mucho tiempo y se busca la mejor solución posible, mientras que en otras situaciones puede ser más beneficioso tener una solución rápida, aunque sea mejorable.

Existe no obstante una tercera métrica que se ha representado en los resultados y es el número de fallos de memoria que experimentan los modelos al buscar una solución. Se especifica fallo de memoria ya que ha sido el único fallo encontrado en los experimentos realizados. Estos fallos implican que el modelo no ha terminado su búsqueda de soluciones debido a que la memoria de la máquina virtual se ha visto desbordada. Como consecuencia de esto, se puede obtener una solución infactible o no obtener solución. Así, se ha optado por eliminar del cálculo de RPD y tiempo medio todas las soluciones en las que el solver detecta un fallo de memoria. Estos casos se contabilizan en su lugar en una nueva métrica denominada "Fallos de memoria", que se corresponde con un número natural que indica el número de experimentos eliminados del análisis por estos motivos.

Para visualizar los resultados, separamos los experimentos según el tamaño de las instancias. En primer lugar, se mostrarán los resultados obtenidos para las 640 instancias pequeñas, que tienen entre 6 y 12 trabajos a repartir entre un número de máquinas que

oscila entre 2 y 5. Estos resultados se dividen en un resumen del desempeño promedio entre las instancias pequeñas, seguido de un análisis de cómo afecta el número de máquinas de la instancia al desempeño (en términos de tiempo) del modelo.

Tras esto, se realizarán los experimentos con las 180 instancias medianas, que pasan a tener un total de 20 a 120 trabajos y un número de máquinas comprendido entre las 2 y las 12 máquinas. Al igual que con las instancias pequeñas, se presentará primero una tabla resumen de los resultados obtenidos, seguido de un estudio del comportamiento del desempeño de los modelos frente al número de máquinas de la instancia.

## 3.1 Instancias pequeñas

Para empezar, se ha realizado el estudio de las instancias pequeñas. El análisis de estos problemas de menor complejidad puede servir para hacerse una idea inicial del desempeño de los modelos.

La tabla 3.1 resume los resultados obtenidos para el conjunto de todas las instancias pequeñas. Se observa cómo todos los modelos a excepción del modelo estándar son siempre capaces de converger a la solución óptima. El modelo estándar no consigue converger al óptimo en algunas de las instancias y consume un tiempo muy elevado en comparación con el resto de métodos. Además, en una de las instancias se produce un fallo de falta de memoria y no se obtiene ninguna solución.

**Tabla 3.1** Resultados de los diferentes modelos ante las instancias pequeñas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 0,015 | 252,892 | 1 |
| DL | 0 | 0,203 | 0 |
| DL-AM | 0 | 0,203 | 0 |
| DL-KB | 0 | 0,200 | 0 |
| MTZ | 0 | 0,198 | 0 |
| MTZ-AM | 0 | 0,203 | 0 |
| MTZ-KB | 0 | 0,202 | 0 |
| Propuesto | 0 | 18,352 | 0 |
| CP | 0 | 3,128 | 0 |

En cuanto al resto de métodos, se observa que todos son capaces de llegar a la solución óptima en un intervalo corto de tiempo sin experimentar fallos de memoria. Los modelos DL, MTZ y los derivados de éstos son capaces de converger a la solución óptima en un tiempo cercano a las 2 décimas de segundo, mientras que el modelo CP tarda una media de 3.13 segundos y el método propuesto tarda una media de 18.35 segundos. Aunque notablemente más elevados, se puede considerar que cualquiera de estos es válido para resolver un problema con una instancia pequeña como las tratadas en esta sección.

Para analizar cómo evoluciona el desempeño respecto al número de máquinas del problema, se propone también un desglose de la tabla anterior en cuatro tablas, una por cada

número de máquinas posible en el conjunto de instancias pequeñas. Esto permitirá analizar también dónde se produce el fallo de memoria del modelo ST, así como las soluciones del mismo que no llegan al óptimo.

**Tabla 3.2** Resultados de los diferentes modelos ante las instancias pequeñas con 2 máquinas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 0,060 | 863,599 | 1 |
| DL | 0 | 0,124 | 0 |
| DL-AM | 0 | 0,123 | 0 |
| DL-KB | 0 | 0,124 | 0 |
| MTZ | 0 | 0,119 | 0 |
| MTZ-AM | 0 | 0,124 | 0 |
| MTZ-KB | 0 | 0,124 | 0 |
| Propuesto | 0 | 28,470 | 0 |
| CP | 0 | 6,595 | 0 |

**Tabla 3.3** Resultados de los diferentes modelos ante las instancias pequeñas con 3 máquinas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 0 | 134,218 | 0 |
| DL | 0 | 0,206 | 0 |
| DL-AM | 0 | 0,183 | 0 |
| DL-KB | 0 | 0,190 | 0 |
| MTZ | 0 | 0,206 | 0 |
| MTZ-AM | 0 | 0,191 | 0 |
| MTZ-KB | 0 | 0,188 | 0 |
| Propuesto | 0 | 19,734 | 0 |
| CP | 0 | 2,791 | 0 |

**Tabla 3.4** Resultados de los diferentes modelos ante las instancias pequeñas con 4 máquinas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 0 | 14,516 | 0 |
| DL | 0 | 0,235 | 0 |
| DL-AM | 0 | 0,250 | 0 |
| DL-KB | 0 | 0,254 | 0 |
| MTZ | 0 | 0,229 | 0 |
| MTZ-AM | 0 | 0,247 | 0 |
| MTZ-KB | 0 | 0,255 | 0 |
| Propuesto | 0 | 18,973 | 0 |
| CP | 0 | 1,851 | 0 |

**Tabla 3.5** Resultados de los diferentes modelos ante las instancias pequeñas con 5 máquinas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 0 | 2,928 | 0 |
| DL | 0 | 0,249 | 0 |
| DL-AM | 0 | 0,259 | 0 |
| DL-KB | 0 | 0,237 | 0 |
| MTZ | 0 | 0,239 | 0 |
| MTZ-AM | 0 | 0,252 | 0 |
| MTZ-KB | 0 | 0,244 | 0 |
| Propuesto | 0 | 6,672 | 0 |
| CP | 0 | 1,243 | 0 |

En las tablas anteriores podemos observar cómo el fallo de memoria del modelo ST se produce en el caso de dos máquinas. Así, es también en estos problemas donde el mismo modelo obtiene soluciones que no alcanzan el óptimo. Observamos también cómo los desempeños de los modelos ST y CP, así como el modelo propuesto mejoran a medida que aumenta el número de máquinas disponibles del problema.

Para visualizar mejor los tiempos de las tablas anteriores, se han representado también en la figura 3.1, en la cual podemos comprobar cómo los modelos ST, CP y el propuesto tienen unos tiempos muy superiores a los modelos basados en DL y MTZ. Además se observa nuevamente que el tiempo que necesitan estos modelos para converger a la solución disminuye a medida que aumenta el número de máquinas del problema.

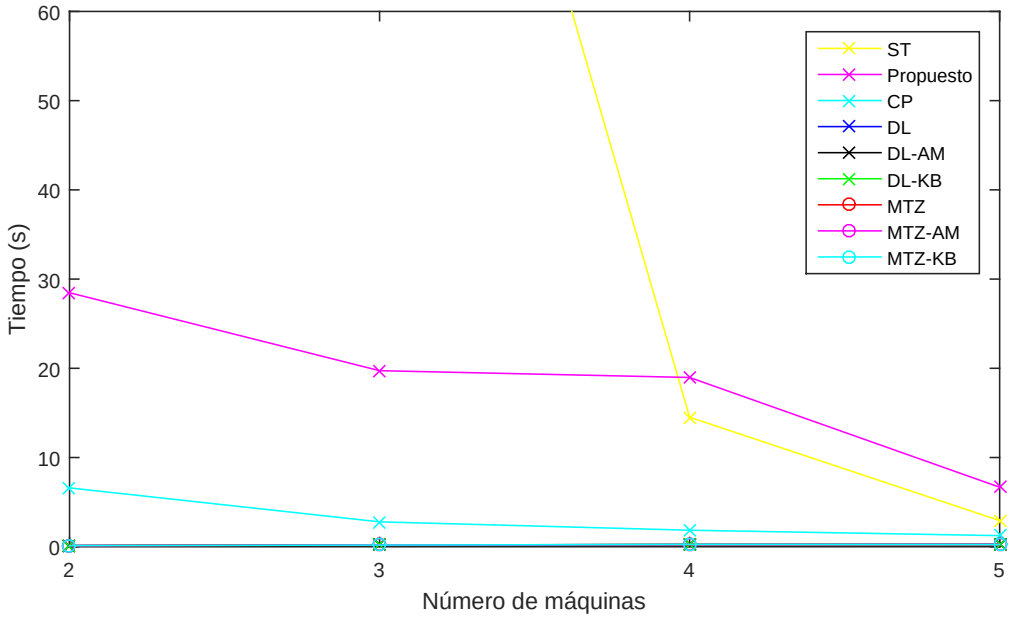



**Figura 3.1** Comparación de los tiempos de los diferentes modelos frente al número de máquinas de las instancias pequeñas.

Para apreciar mejor la diferencia existente entre los modelos basados en DL y MTZ se presenta la figura 3.2, que ignora los otros tres modelos para una mejor comparación. En esta figura vemos cómo los seis modelos tienen un desempeño parejo y el tiempo que necesitan para encontrar una solución crece levemente conforme aumenta el número de máquinas, a diferencia de los tres modelos que se han quedado fuera de esta comparación.

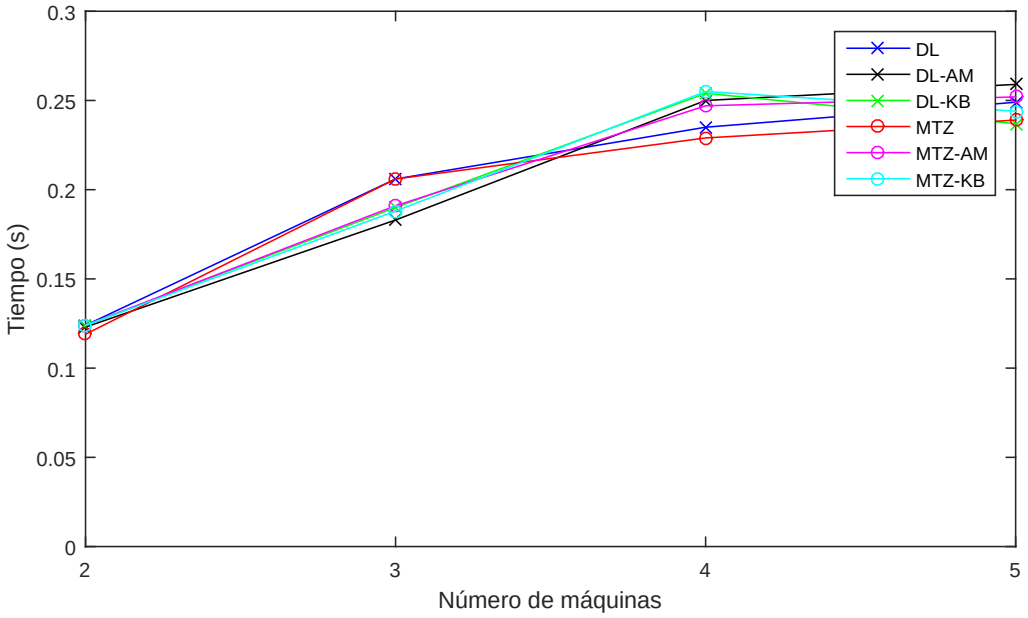



**Figura 3.2** Comparación de los tiempos de los modelos basados en DL y MTZ frente al número de máquinas de las instancias pequeñas.

## 3.2 Instancias medianas

Una vez analizados los modelos sobre las instancias pequeñas, se estudian los resultados de los diferentes modelos sobre el conjunto de instancias mediano. Estas instancias son más atractivas ya que presentan un problema complejo y requiere de modelos más eficientes para la resolución. Además, la duración máxima de los experimentos se ha limitado a una hora (3600 segundos), por lo que los modelos no conseguirán converger siempre a la solución óptima.

Los resultados de las instancias medianas se encuentran recogidos en la tabla 3.6. Estos resultados suponen una media de todas las instancias medianas que, a diferencia de las instancias pequeñas, están distribuidas de forma no uniforme (Figura 2.3), esto hace que, por ejemplo, las instancias con 120 trabajos tengan más repercusión en los resultados que las instancias con 40 trabajos, ya que las primeras comprenderán 60 instancias mientras que las segundas solo 15. Sin embargo, al evaluar todos los modelos con las mismas instancias se consigue obtener una muestra de cómo es el desempeño de éstos ante instancias medianas.

**Tabla 3.6** Resultados de los diferentes modelos ante las instancias medianas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 265,166 | 3601,875 | 3 |
| DL | 20,456 | 2520,555 | 0 |
| DL-AM | 6,260 | 2492,986 | 0 |
| DL-KB | 4,280 | 2392,219 | 0 |
| MTZ | 39,904 | 2574,809 | 0 |
| MTZ-AM | 8,621 | 2474,656 | 0 |
| MTZ-KB | 18,153 | 2428,721 | 0 |
| Propuesto | 35,122 | 3604,328 | 87 |
| CP | 0,006 | 3627,733 | 0 |

Se observa cómo los tiempos han crecido en gran medida en comparación con las instancias pequeñas. Tanto es así, que el tiempo empleado por los modelos ST, CP y el modelo propuesto superan levemente el tiempo máximo especificado. Por cuestiones relativas al solver, esto puede ocurrir y el tiempo límite fijado no es estricto en el proceso de resolución. En lo que respecta a este estudio, no se considera que esto afecte de forma significativa a los resultados.

Se observa también cómo se producen 87 fallos de memoria en el modelo propuesto. Esto significa que el modelo no ha sido capaz de resolver 87 de las 180 instancias debido a problemas de memoria, lo que hace que el modelo sea poco atractivo para implementar en problemas de esta magnitud. Esta cantidad de fallos de memoria puede deberse a lo observado en la tabla 2.3.5, que resaltaba el elevado número de restricciones necesarias para definir el modelo propuesto. El modelo estándar ST sigue teniendo fallos de memoria, pero su número es pequeño en comparación con el número total de experimentos, suponiendo algo menos del 2 % de las instancias. El resto de modelos no experimenta fallos de memoria, así que su desempeño se medirá con las medidas de tiempo y RPD.

El modelo CP consigue en estos experimentos un valor de RPD muy bajo de 0.006, lo que implica que consigue encontrar el mejor valor de la función objetivo de entre los modelos estudiados. El segundo modelo con RPD más bajo es el modelo DL-KB, con un valor de 4.280. Tras esto, le siguen de cerca el modelo DL-AM con RPD=6.260 y el modelo MTZ-AM con RPD=8.621. También hay que destacar los resultados obtenidos por el modelo ST, los cuales son mucho peores a los obtenidos por el resto de los modelos, resultando en un RPD=265.166.

A continuación, al igual que se realizó con las instancias pequeñas, se desglosarán los resultados de las instancias medianas para cada valor posible del número de máquinas.

**Tabla 3.7** Resultados de los diferentes modelos ante las instancias medianas con 2 máquinas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 19,58 | 3602,97 | 3 |
| DL | 0,69 | 1012,85 | 0 |
| DL-AM | 0,70 | 963,72 | 0 |
| DL-KB | 0,69 | 715,93 | 0 |
| MTZ | 0,69 | 1144,76 | 0 |
| MTZ-AM | 0,69 | 898,61 | 0 |
| MTZ-KB | 0,69 | 813,41 | 0 |
| Propuesto | 49,84 | 3604,38 | 12 |
| CP | 0,01 | 3380,72 | 0 |

**Tabla 3.8** Resultados de los diferentes modelos ante las instancias medianas con 4 máquinas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 167,04 | 3600,81 | 0 |
| DL | 2,33 | 3595,33 | 0 |
| DL-AM | 1,85 | 3574,06 | 0 |
| DL-KB | 1,62 | 3581,43 | 0 |
| MTZ | 2,08 | 3593,19 | 0 |
| MTZ-AM | 1,99 | 3600,44 | 0 |
| MTZ-KB | 1,63 | 3569,06 | 0 |
| Propuesto | 106,07 | 3604,15 | 30 |
| CP | 0 | 3596,95 | 0 |

**Tabla 3.9** Resultados de los diferentes modelos ante las instancias medianas con 6 máquinas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 708,37 | 3601,48 | 0 |
| DL | 9,56 | 3600,35 | 0 |
| DL-AM | 24,39 | 3600,30 | 0 |
| DL-KB | 9,35 | 3600,39 | 0 |
| MTZ | 32,47 | 3600,37 | 0 |
| MTZ-AM | 20,75 | 3600,36 | 0 |
| MTZ-KB | 29,12 | 3600,46 | 0 |
| Propuesto | — | — | 30 |
| CP | 0 | 3600,38 | 0 |

**Tabla 3.10** Resultados de los diferentes modelos ante las instancias medianas con 8 máquinas.

| Modelo | RPD | Tiempo(s) | Fallos de memoria |
|---|---|---|---|
| ST | 1003,11 | 3601,66 | 0 |
| DL | 213,58 | 3600,39 | 0 |
| DL-AM | 15,48 | 3600,38 | 0 |
| DL-KB | 22,71 | 3600,52 | 0 |
| MTZ | 402,15 | 3600,42 | 0 |
| MTZ-AM | 50,54 | 3600,35 | 0 |
| MTZ-KB | 149,64 | 3600,47 | 0 |
| Propuesto | — | — | 15 |
| CP | 0 | 3600,50 | 0 |

En las anteriores tablas se puede observa cómo el modelo propuesto no es capaz de resolver ninguna de las instancias con 6 o 8 máquinas debido a problemas de memoria, mientras que también falla en las instancias de 4 máquinas en 30 de las 60 instancias disponibles y en las instancias de 2 máquinas, con 12 fallos de las 75 instancias disponibles. Así, los problemas de memoria hacen que este modelo no sea eficaz cuando el problema es complejo.

Se puede ver también que todos los modelos utilizan los 3600 segundos permitidos para encontrar la solución en los problemas con 6 y 8 máquinas. En las instancias con 4 máquinas algunos obtienen resultados poco antes de la hora y en las instancias de 2 máquinas sí encuentran soluciones mucho antes, a excepción de los modelos estándar, CP y el modelo propuesto.

Así, a diferencia de lo realizado con las instancias pequeñas, en el caso de las instancias medianas el tiempo no parece un factor determinante a la hora de elegir un modelo. Es por

ello que se ha elegido el desempeño en términos de RPD para que la comparación resulte más interesante.

Se observa cómo el modelo CP domina los resultados en todas los conjuntos de instancias, obteniendo siempre las mejores soluciones excepto para el caso de 2 máquinas, donde el RPD es levemente superior a 0. Sin embargo, demuestra ser el modelo con mejor desempeño, obteniendo un RPD de 0.01.

Se incluyen también las figuras 3.3 y 3.4, donde se comparan los RPD obtenidos de forma similar a como se analizaron los tiempos para las instancias pequeñas. Esto es, mostrando cómo evoluciona la métrica RPD en función del número de máquinas consideradas. En ellas se observa cómo la solución de todos los modelos se separa con el tiempo de la solución obtenida por el modelo CP, que se mantiene siempre como la solución más baja.

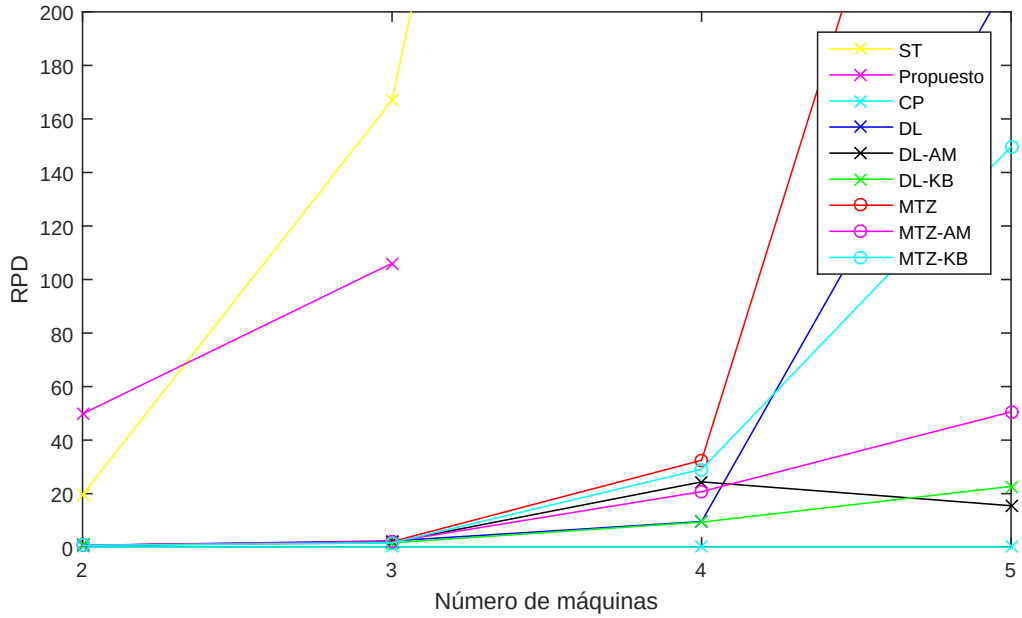



**Figura 3.3** Comparación de las métricas RPD de los diferentes modelos frente al número de máquinas de las instancias medianas.

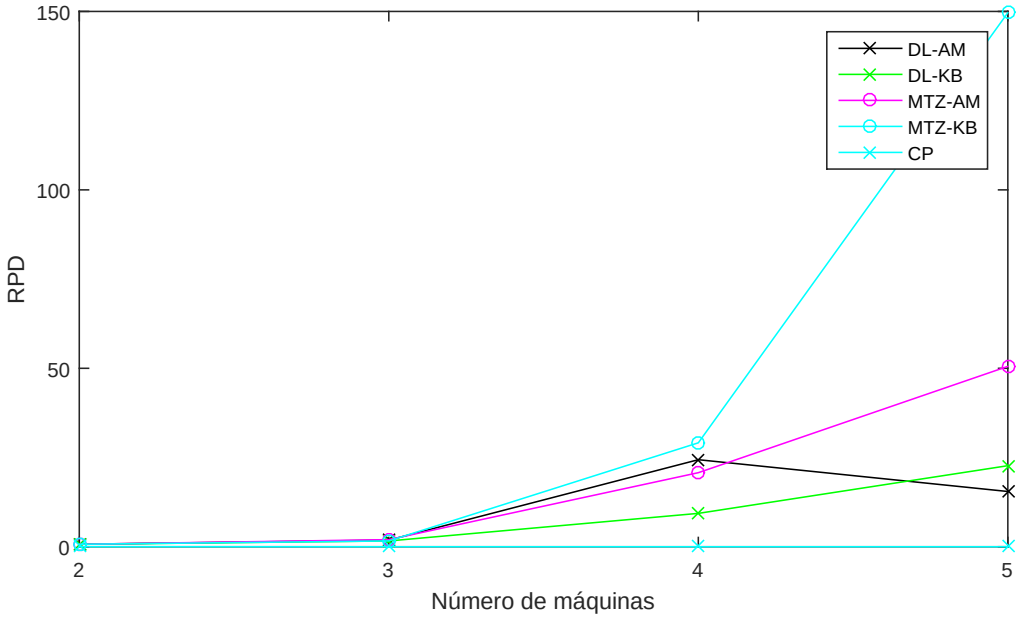



**Figura 3.4** Comparación de las métricas RPD de cinco modelos frente al número de máquinas de las instancias medianas.

# 4 Conclusiones y futuras líneas

En este trabajo se ha estudiado el problema de secuenciación de máquinas paralelas no relacionadas con tiempos de cambio. Para ello, se han explicado los elementos que componen este tipo de problemas, así como problemas similares. Tras ello, se han comentado diferentes acercamientos estudiados en la la literatura y se ha elegido profundizar en este trabajo en el enfoque de modelos exactos.

Se ha definido formalmente el problema, introduciendo la notación más común, para después proponer una notación unificada que abarque todos los modelos tratados en este trabajo. Una vez hecho esto, se han expuesto los diferentes modelos exactos objetivo del trabajo, indicando su formulación matemática, explicando las restricciones de los diferentes problemas y detallando el número de variables y restricciones que posee cada uno de los modelos. Entre los modelos se incluyen siete modelos MILP recogidos de la literatura, un modelo MILP propuesto y un modelo basado en la programación por restricciones.

Para el modelo propuesto se ha comentado el origen de la idea de la que surge, así como la necesidad de adaptar la notación para que se adecúe a él. A diferencia de los anteriores modelos MILP, este modelo define sus variables en función del puesto temporal en el que se procesan y no de la secuencia. El objetivo es el de estudiar si este enfoque mejora al enfoque tradicional.

En cuanto al modelo basado en la programación por restricciones, se ha introducido este paradigma de la formulación y se han definido las restricciones del modelo, explicando qué aporta cada una y comentando las funciones de librería empleadas para su implementación.

Una vez definidos todos los modelos bajo estudio, se han especificado las herramientas empleadas para su implementación. Se han explicado las instancias tratadas para evaluar el desempeño, así como el hardware utilizado. Para ello, se han definido también los ficheros que componen los experimentos y cómo se han obtenido estos.

El siguiente paso ha sido la visualización de los resultados, en la cual hemos podido observar el desempeño de los diferentes modelos con un límite temporal de una hora. A diferencia del trabajo realizado en [17], al limitar el tiempo de ejecución a una hora (en vez de tres), se obseva una diferencia notable entre los diferentes modelos cuando resuelven las instancias medianas. Así, se observa como el modelo propuesto tiene problemas de memoria en estas instancias debido posiblemente al elevado número de restricciones que necesita para definir el problema. En cambio, el modelo basado en programación por restricciones obtiene unos resultados prometedores, obteniendo el mejor desempeño de entre los modelos estudiados para el caso de instancias medianas.

Como líneas futuras se propone evaluar el modelo CP, así como los modelos derivados de DL y MTZ en unas instancias aún más grandes para estudiar si el desempeño del modelo CP sigue mejorando el obtenido por los modelos DL y MTZ.

# Bibliografía


[1] *Soa problem instances*, http://soa.iti.es/problem-instances, Consultado: 10-05-2020.

[2] Krzysztof Apt, *Principles of constraint programming*, Cambridge University press, 2003.

[3] Oliver Avalos-Rosales, Francisco Angel-Bello, and Ada Alvarez, *Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times*, The International Journal of Advanced Manufacturing Technology **76** (2015), no. 9-12, 1705–1718.

[4] Nagraj Balakrishnan, John J Kanet, and V Sridharan, *Early/tardy scheduling with sequence dependent setups on uniform parallel machines*, Computers & Operations Research **26** (1999), no. 2, 127–141.

[5] Philippe Baptiste, Claude Le Pape, and Wim Nuijten, *Constraint-based scheduling: applying constraint programming to scheduling problems*, vol. 39, Springer Science & Business Media, 2012.

[6] S Binato, WJ Hery, DM Loewenstern, and MAURICIO GC Resende, *A grasp for job shop scheduling*, Essays and surveys in metaheuristics, Springer, 2002, pp. 59–79.

[7] Ronald F Boisvert, Sally E Howe, and David K Kahaner, *Gams: A framework for the management of scientific software*, ACM Transactions on Mathematical Software (TOMS) **11** (1985), no. 4, 313–355.

[8] Anthony Brook, David Kendrick, and Alexander Meeraus, *Gams, a user's guide*, ACM Signum Newsletter **23** (1988), no. 3-4, 10–11.

[9] Peter Brucker, *Scheduling algorithms*, Journal-Operational Research Society **50** (1999), 774–774.

[10] Lawrence Davis, *Job shop scheduling with genetic algorithms*, Proceedings of an international conference on genetic algorithms and their applications, vol. 140, 1985.

[11] Mateus Rocha De Paula, Martín Gómez Ravetti, Geraldo Robson Mateus, and Panos M Pardalos, *Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search*, IMA Journal of Management Mathematics **18** (2007), no. 2, 101–115.

[12] Mauro Dell'Amico and Marco Trubian, *Applying tabu search to the job-shop scheduling problem*, Annals of Operations research **41** (1993), no. 3, 231–252.

[13] Martin Desrochers and Gilbert Laporte, *Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints*, Operations Research Letters **10** (1991), no. 1, 27–36.

[14] Emrah B Edis and Ceyda Oguz, *Parallel machine scheduling with flexible resources*, Computers & Industrial Engineering **63** (2012), no. 2, 433–447.

[15] Luis Fanjul-Peyro, Federico Perea, and Rubén Ruiz, *Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources*, European Journal of Operational Research **260** (2017), no. 2, 482–493.

[16] Luis Fanjul-Peyro and Rubén Ruiz, *Size-reduction heuristics for the unrelated parallel machines scheduling problem*, Computers & Operations Research **38** (2011), no. 1, 301–309.

[17] Luis Fanjul-Peyro, Rubén Ruiz, and Federico Perea, *Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times*, Computers & Operations Research **101** (2019), 173–182.

[18] Mary Fenelon, *Ibm ilog cplex optimization studio: The complete optimization development toolkit (presentation)*, IIE Annual Conference. Proceedings, Institute of Industrial and Systems Engineers (IISE), 2011, p. 1.

[19] Ridvan Gedik, Darshan Kalathia, Gokhan Egilmez, and Emre Kirac, *A constraint programming approach for solving unrelated parallel machine scheduling problem*, Computers & Industrial Engineering **121** (2018), 139–149.

[20] Alain Guinet, *Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria*, The International Journal of Production Research **31** (1993), no. 7, 1579–1594.

[21] Magdy Helal, Ghaith Rabadi, and Ameer Al-Salem, *A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times*, International Journal of Operations Research **3** (2006), no. 3, 182–192.

[22] Imdat Kara and Tolga Bektas, *Integer linear programming formulations of multiple salesman problems and its variations*, European Journal of Operational Research **174** (2006), no. 3, 1449–1458.

[23] Dong-Won Kim, Kyong-Hee Kim, Wooseung Jang, and F Frank Chen, *Unrelated parallel machine scheduling with setup times using simulated annealing*, Robotics and Computer-Integrated Manufacturing **18** (2002), no. 3-4, 223–231.

[24] Dong-Won Kim, Dong-Gil Na, and F Frank Chen, *Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective*, Robotics and Computer-Integrated Manufacturing **19** (2003), no. 1-2, 173–181.

[25] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím, *Ibm ilog cp optimizer for scheduling*, Constraints **23** (2018), no. 2, 210–250.

[26] Eugene L Lawler, Jan Karel Lenstra, Alexander HG Rinnooy Kan, and David B Shmoys, *Sequencing and scheduling: Algorithms and complexity*, Handbooks in operations research and management science **4** (1993), 445–522.

[27] Chung-Yee Lee, *Parallel machines scheduling with nonsimultaneous machine available time*, Discrete Applied Mathematics **30** (1991), no. 1, 53–61.

[28] Jan Karel Lenstra, David B Shmoys, and Éva Tardos, *Approximation algorithms for scheduling unrelated parallel machines*, Mathematical programming **46** (1990), no. 1-3, 259–271.

[29] Shih-Wei Lin and Kuo-Ching Ying, *Optimization of makespan for no-wait flowshop scheduling problems using efficient matheuristics*, Omega **64** (2016), 115–125.

[30] Andrew Makhorin, *Modeling language gnu mathprog*, Relatório Técnico, Moscow Aviation Institute **63** (2000).

[31] Mohammad Mahdavi Mazdeh, Farzad Zaerpour, Abalfazl Zareei, and Ali Hajinezhad, *Parallel machines scheduling to minimize job tardiness and machine deteriorating cost with deteriorating jobs*, Applied Mathematical Modelling **34** (2010), no. 6, 1498–1510.

[32] Clair E Miller, Albert W Tucker, and Richard A Zemlin, *Integer programming formulation of traveling salesman problems*, Journal of the ACM (JACM) **7** (1960), no. 4, 326–329.

[33] Anulark Naber and Rainer Kolisch, *Mip models for resource-constrained project scheduling with flexible resource profiles*, European Journal of Operational Research **239** (2014), no. 2, 335–348.

[34] Victor Pillac, Christelle Gueret, and Andrés L Medaglia, *A parallel matheuristic for the technician routing and scheduling problem*, Optimization Letters **7** (2013), no. 7, 1525–1535.

[35] Michael Pinedo, *Planning and scheduling in manufacturing and services*, Springer, 2005.

[36] Michael Pinedo and Khosrow Hadavi, *Scheduling: theory, algorithms and systems development*, Operations Research Proceedings 1991, Springer, 1992, pp. 35–42.

[37] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2019.

[38] Ghaith Rabadi, Reinaldo J Moraga, and Ameer Al-Salem, *Heuristics for the unrelated parallel machine scheduling problem with setup times*, Journal of Intelligent Manufacturing **17** (2006), no. 1, 85–97.

[39] Francisco J Rodriguez, Manuel Lozano, Christian Blum, and Carlos García-Martínez, *An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem*, Computers & Operations Research **40** (2013), no. 7, 1829–1841.

[40] Francesca Rossi, Peter Van Beek, and Toby Walsh, *Handbook of constraint programming*, Elsevier, 2006.

[41] Reza Tavakkoli-Moghaddam, Farid Taheri, Mohammad Bazzazi, M Izadi, and Farrokh Sassani, *Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints*, Computers & Operations Research **36** (2009), no. 12, 3224–3230.

[42] S Ali Torabi, Navid Sahebjamnia, S Afshin Mansouri, and M Aramon Bajestani, *A particle swarm optimization for a fuzzy multi-objective unrelated parallel machines scheduling problem*, Applied Soft Computing **13** (2013), no. 12, 4750–4762.

[43] Eva Vallada and Rubén Ruiz, *A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times*, European Journal of Operational Research **211** (2011), no. 3, 612–622.

[44] Chenhong Zhao, Shanshan Zhang, Qingfeng Liu, Jian Xie, and Jicheng Hu, *Independent tasks scheduling based on genetic algorithm in cloud computing*, 2009 5th International Conference on Wireless Communications, Networking and Mobile Computing, IEEE, 2009, pp. 1–4.

# 5 Anexos

## 5.1 Figuras

| Instance | Objective | Type | Gap (%) | Time (sec.) | |
|---|---|---|---|---|---|
| Type=0 optimal solution found. | | Type=1 time limit exeeded in a solution. | | | Type= |
| I_10_2_S_1-124_1_DLCPLEX.txt | 2,8800000000e+02 | | 0 | 0 | 0,14 |
| I_10_2_S_1-124_10_DLCPLEX.txt | 3,2100000000e+02 | | 0 | 0 | 0,14 |
| I_10_2_S_1-124_2_DLCPLEX.txt | 2,5300000000e+02 | | 0 | 0 | 0,13 |
| I_10_2_S_1-124_3_DLCPLEX.txt | 2,3300000000e+02 | | 0 | 0 | 0,14 |
| I_10_2_S_1-124_4_DLCPLEX.txt | 2,2500000000e+02 | | 0 | 0 | 0,17 |
| I_10_2_S_1-124_5_DLCPLEX.txt | 2,7700000000e+02 | | 0 | 0 | 0,20 |
| I_10_2_S_1-124_6_DLCPLEX.txt | 2,8400000000e+02 | | 0 | 0 | 0,14 |
| I_10_2_S_1-124_7_DLCPLEX.txt | 2,7600000000e+02 | | 0 | 0 | 0,11 |
| I_10_2_S_1-124_8_DLCPLEX.txt | 3,2500000000e+02 | | 0 | 0 | 0,11 |
| I_10_2_S_1-124_9_DLCPLEX.txt | 2,5500000000e+02 | | 0 | 0 | 0,14 |
| I_10_2_S_1-49_1_DLCPLEX.txt | 2,1300000000e+02 | | 0 | 0 | 0,09 |

**Figura 5.1** Extracto del fichero Solution_Report para las instancias pequeñas del modelo DL.

## 5.2 Código

### 5.2.1 Código R para generar los ficheros ".mod" con los modelos CP

```
files <- list.files(path="./Instancias./small_small", ...
    pattern="*.txt", full.names=T, recursive=FALSE)
files

count <- 0
#filename <- files[1]
for(filename in files){
  count <- count + 1
  filename_data <- ...
     paste(substr(filename,1,nchar(filename)-4), ...
     "_cp.dat", sep="")
```

```
9
10   #start creating the executable file
11   filename_exe <- paste("./ficheros/", ...
        substr(filename,29,nchar(filename)-4), ...
        "_cp_mod.mod", sep="")
12   filename_short <- ...
        paste("p",substr(filename,29,nchar(filename)-4),sep="")
13   filename_short <- chartr(old = "-", new = "_", ...
        filename_short)
14   sink(filename_exe)
15   cat("
16       /*********************************************
17  * OPL 12.10.0.0 Model
18  * Author: Vic
19  *********************************************/
20
21  using CP;
22
23 //Parameters
24 int nMachines = ...;
25 int nJobs = ...;
26 range Machs = 0..nMachines-1;
27 range Jobs = 0..nJobs-1;
28 int p[i in Machs][j in Jobs] = ...;
29 int s[i in Machs][j in Jobs][k in Jobs] = ...;
30 //Cmax
31 dvar int LBi[i in 1..2];
32 dvar int LB;
33 dvar int makespan;
34 //interval variables
35 dvar interval Z[j in Jobs];
36 dvar interval Y[i in Machs, j in Jobs] optional size ...
      p[i][j];
37 dvar sequence V[i in Machs] in all(j in Jobs) Y[i][j];
38 tuple Setup_tuple {int id1; int id2; int setup_value;};
39 //{Setup_tuple} Tup[i in Machs] = {<i,j,3> | i in ...
      Jobs, j in Jobs};
40 {Setup_tuple} Tup[i in Machs] = {<j,k,s[i][j][k]> | j ...
      in Jobs, k in Jobs};
41 //cumul function
42 cumulFunction f = sum(i in Machs, j in Jobs) ...
      pulse(Y[i][j],1);
43
44 execute {
45         cp.param.FailLimit = 10000;
46 }
47 //model
48 minimize max(j in Jobs) makespan;
49 subject to {
50   makespan ≥ max(j in Jobs) endOf(Z[j]);
```

```
51    forall (j in Jobs){
52      alternative(Z[j], all(i in Machs) Y[i][j]);
53    }
54    forall (i in Machs)
55      noOverlap(V[i],Tup[i]);
56    f≤nMachines;
57    makespan ≥ LB;
58    makespan ≥ 0;
59    LBi[1]==ceil(1/nMachines*sum(j in Jobs) min(i in ...
         Machs, k in Jobs) (p[i][j]+s[i][j][k]));
60    LBi[2]==max(j in Jobs) min(i in Machs, k in Jobs) ...
         (p[i][j]+s[i][j][k]);
61    LB==max(i in 1..2) LBi[i];
62  }
63
64  execute {
65    var",filename_short, " = new IloOplOutputFile(\" ...
         ",filename_short,".txt \");
66    for (var j in Jobs) {"
67      ,filename_short,".writeln(\"tarea \"+j);
68      writeln(\"tarea \"+j);
69      ",filename_short,".writeln(Z[j].start + \"\");
70      ",filename_short,".writeln(Z[j].end + \"\");
71    }
72    ",filename_short,".writeln(\"makespan \"+makespan);
73    ",filename_short,".close();
74  };
75
76
77  ")
78    sink();
79  }
```

### 5.2.2 Código R para generar las instancias ".dat" a partir de las originales ".txt" para el modelo CP

```
1  #Generate instances
2  library(stringr)
3
4  files <- list.files(path="./Instancias./small_small", ...
       pattern="*.txt", full.names=T, recursive=FALSE)
5  files
6
7  count <- 0
8  #filename <- files[1]
9  for(filename in files){
10   count <- count + 1
```

```
11  # Firt read the Number of jobs, number of machines ...
       and number of stages
12  first_line <- scan(file = filename, what = ...
       integer(), nmax = 3)
13  N_j <- first_line[1] # Number of jobs
14  N_m <- first_line[2] # Number of machines
15  N_s <- first_line[3] # Number of stages
16  # Then initialize the parameters p and S
17  p <- array(NA,dim=c(N_m,N_j)) # Processing time
18  S <- array(NA,dim=c(N_m,N_j,N_j)) # Sequence ...
       dependent setup times
19  # Now, create a loop to read and give the proper ...
       values to the parameters
20  lines_skip <- 1
21  cont_p_i <- 0
22  for(ii in 1:N_j){
23    lines_skip <- lines_skip + 1;
24    data_ii <- scan(file = filename, what = integer(), ...
         nlines = 1,skip=lines_skip)
25    cont_p_i <- cont_p_i + 1
26    cont_p_j <- 0
27    for (jj in seq(2,length(data_ii),2)){
28      cont_p_j <- cont_p_j + 1
29      p[cont_p_j, cont_p_i] <- data_ii[jj]
30    }
31  }
32  lines_skip <- lines_skip + 2
33  cont_S_j <- 0
34  for (ii in 1:(N_m*N_j)){
35    lines_skip <- lines_skip + 1
36    data_ii <- scan(file = filename, what = integer(), ...
         nlines = 1,skip=lines_skip)
37    cont_S_j <- cont_S_j + 1
38    S[((ii-1)%/%N_j)+1, cont_S_j, ] <- data_ii
39    if(ii%%N_j==0){
40      lines_skip <- lines_skip + 1
41      cont_S_j <- 0
42    }
43  }
44
45  filename_data <- ...
       paste(substr(filename,1,nchar(filename)-4), ...
       "_cp.dat", sep="")
46
47  #start creating the executable file
48  filename_exe <- paste("./ficheros/", ...
       substr(filename,29,nchar(filename)-4), ...
       "_cp_data.dat", sep="")
49  filename_short <- substr(filename,29,nchar(filename)-4)
```

```
50    filename_short <- chartr(old = "-", new = "_", ...
          filename_short)
51    sink(filename_exe)
52
53
54    cat("
55    /******************************************** \n
56   * OPL 12.10.0.0 Data \n
57   * Author: Victor Mirasierra \n
58   *********************************************/ \n
59   \n
60
61   nMachines = ",N_m, "; \n
62   nJobs = ",N_j, "; \n
63   p = [")
64   for (ii in 1:N_m){
65     cat("\n [")
66     for (jj in 1:N_j){
67       cat("\t" , p[ii,jj], ", ", sep="")
68     }
69     cat("],")
70   }
71 cat("]; \n")
72 cat("s = [ \n")
73  for (ii in 1:N_m){
74    cat("[")
75     for (jj in 1:N_j){
76       cat(" [")
77       for (kk in 1:N_j){
78         cat("\t" , S[ii,jj,kk], ", ", sep="")
79       }
80       cat("], \n")
81     }
82    cat("] \n,")
83  }
84 cat("];")
85 sink()
86 }
```

### 5.2.3 Código R para generar los archivos de configuración para los experimentos del modelo CP

```
1 files <- list.files(path="./Instancias./small_small", ...
      pattern="*.txt", full.names=T, recursive=FALSE)
2 files
3
4 count <- 0
5 sink("Configuracion_exp.txt")
```

```
#filename <- files[1]
for(filename in files){
  count <- count+1
  filename_exe <- paste("./ficheros/", ...
     substr(filename,29,nchar(filename)-4), ...
     "_cp_mod.mod", sep="")
  filename_short_mod <- ...
     substr(filename,29,nchar(filename)-4)
  filename_short_dat <- ...
     substr(filename,29,nchar(filename)-4)

  filename_short_mod <- chartr(old = "-", new = "_", ...
     filename_short_mod)
  filename_short_dat <- chartr(old = "-", new = "_", ...
     filename_short_dat)

  cat(".\\Instances\\",filename_short_mod,".mod ...
     .\\Instances\\",filename_short_dat,".dat \n", sep="")
}
sink()
```

### 5.2.4 Código R para generar el Solution_Report del modelo CP

```
#Generate instances
library(stringr)

#files <- ...
   list.files(path="./Instancias./medium_rabadi", ...
   pattern="*.txt", full.names=T, recursive=FALSE)
files <- list.files(pattern="*.txt", full.names=T, ...
   recursive=FALSE)

count <- 0
#filename <- files[1]
media_time<-0
media_makespan<-0
media_cota<-0
N<-length(files)

sink("Solution_report_cp.txt")
for(filename in files){
  count <- count + 1
  # Firt read the Number of jobs, number of machines ...
     and number of stages
  line <- scan(file = filename, what = character(), ...
     nmax = 4)
  Time <- as.numeric(line[2])/1000 # Number of jobs
  Makespan <- as.numeric(line[4])
```

```
21
22    cat(filename, Makespan, Time, sep="\t")
23    cat("\n")
24
25    media_time<-media_time+Time/N
26    media_makespan <- media_makespan+Makespan/N
27  }
28  sink()
29  #start creating the executable file
30  sink("Resumen_soluciones.txt")
31  cat("tiempo medio", media_time,
32      "makespan medio", media_makespan)
33  sink()
```

### 5.2.5 Código R para generar los ficheros ".gms" con el modelo propuesto

```
1  #Generate instances
2  library(stringr)
3
4  files <- ...
       list.files(path="./Instancias./medium_rabadi", ...
       pattern="*.txt", full.names=T, recursive=FALSE)
5  files
6
7  count <- 0
8  #filename <- files[1]
9  for(filename in files){
10   count <- count + 1
11   # Firt read the Number of jobs, number of machines ...
        and number of stages
12   first_line <- scan(file = filename, what = ...
        integer(), nmax = 3)
13   N_j <- first_line[1] # Number of jobs
14   N_m <- first_line[2] # Number of machines
15   N_s <- first_line[3] # Number of stages
16   # Then initialize the parameters p and S
17   p <- array(NA,dim=c(N_m,N_j)) # Processing time
18   S <- array(NA,dim=c(N_m,N_j,N_j)) # Sequence ...
        dependent setup times
19   # Now, create a loop to read and give the proper ...
        values to the parameters
20   lines_skip <- 1
21   cont_p_i <- 0
22   for(ii in 1:N_j){
23     lines_skip <- lines_skip + 1;
24     data_ii <- scan(file = filename, what = integer(), ...
          nlines = 1,skip=lines_skip)
25     cont_p_i <- cont_p_i + 1
```

```
26      cont_p_j <- 0
27      for (jj in seq(2,length(data_ii),2)){
28        cont_p_j <- cont_p_j + 1
29        p[cont_p_j, cont_p_i] <- data_ii[jj]
30      }
31    }
32    lines_skip <- lines_skip + 2
33    cont_S_j <- 0
34    for (ii in 1:(N_m*N_j)){
35      lines_skip <- lines_skip + 1
36      data_ii <- scan(file = filename, what = integer(), ...
          nlines = 1,skip=lines_skip)
37      cont_S_j <- cont_S_j + 1
38      S[((ii-1)%/%N_j)+1, cont_S_j, ] <- data_ii
39      if(ii%%N_j==0){
40        lines_skip <- lines_skip + 1
41        cont_S_j <- 0
42      }
43    }
44
45    filename_data <- ...
        paste(substr(filename,1,nchar(filename)-4), ...
        "_gms.gms", sep="")
46
47    #start creating the executable file
48    filename_exe <- paste("./ficheros/", ...
        substr(filename,29,nchar(filename)-4), ...
        "_gms_executable.gms", sep="")
49    filename_short <- substr(filename,29,nchar(filename)-4)
50    filename_short <- chartr(old = "-", new = "_", ...
        filename_short)
51    sink(filename_exe)
52
53    cat("$OFFSYMXREF
54  $OFFSYMLIST
55
56  option limrow=0;
57  option limcol=0;
58  option solprint=on;
59  option  sysout=off;
60  option MIP=convert;
61  option OPTCR=0;
62
63  $TITLE UPMS_Vic \n")
64
65
66    cat("Sets \n", sep="")
67    cat("\t i Machine identifier. / 1 *",N_m,"/ \n")
68    cat("\t j Job identifier. / 1 *",N_j,"/ \n")
69    cat("\t k(j) Job identifier. / 1 *",N_j,"/ \n")
```

```
70    cat("\t t Time step identifier. / 1 *",N_j,"/; \n")
71    cat("Parameters \n")
72    cat("Table S(i,j,k) \"Setup times\"; \n")
73    for (ii in 1:N_m){
74      for (jj in 1:N_j){
75        for (kk in 1:N_j){
76          cat("\t S('",ii,"','",jj,"','",kk,"')=", ...
              S[ii,jj,kk], "; \n", sep="")
77        }
78      }
79    }
80    cat("\n")
81    cat("Table p(i,j) \"Working times\"; \n")
82    for (ii in 1:N_m){
83      for (jj in 1:N_j){
84        cat("\t p('",ii,"','",jj,"')=", p[ii,jj], "; ...
            \n", sep="")
85      }
86    }
87
88    cat("
89  Variables \n
90  X(i,j,t)    1 si la tarea j empieza en la maquina i en ...
      el instante t 0 en caso contrario
91  Sv(i,j,k)   Setup de la tarea j precediendo a la k
92  Cmax        Makespan
93
94  Binary Variable X ;
95
96  Positive Variable Sv ;
97
98  Equations
99  lb_makespan(i)      define lower bounds for the makespan
100 unique_jobs(j)      jobs should only be asigned to one ...
      machine at one timestep
101 oj_machines(i,t)    grant that machines process only ...
      one job at a time
102 setup_costs(i,j,k,t) establish the setup costs
103 order(i,t)          establish the order of the jobs;
104
105 lb_makespan(i) ..          sum((j,k)$(not sameas(j,k)), ...
      Sv(i,j,k)) + sum((j,t), p(i,j)*X(i,j,t)) =L= Cmax ;
106
107 unique_jobs(j) ..    sum((i,t), X(i,j,t))  =E=  1 ;
108
109 oj_machines(i,t) ..    sum(j, X(i,j,t))  =L=  1 ;
110
111 setup_costs(i,j,k,t)$(not sameas(j,k)) ..      ...
      S(i,j,k)*(X(i,j,t)+X(i,k,t+1)-1) =L= Sv(i,j,k);
112
```

```
113 order(i,t)..        sum(j,X(i,j,t)) =G= sum(j,X(i,j,t+1));
114
115 Model M_",filename_short," /all/ ;
116
117 $echo CplexLP M_", filename_short,".lp > convert.opt
118
119 M_",filename_short,".optfile=1;
120
121 Solve M_", filename_short," using mip minimizing Cmax ...
      ;", sep="")
122
123   sink()
124 }
```

### 5.2.6 Código R para generar el fichero ".bat" que permite ejecutar los ficheros ".gms" con el modelo propuesto

```
1 files <- list.files(path="./ficheros", ...
     pattern="*.gms", full.names=T, recursive=FALSE)
2 sink("bat_mediano_gms.bat")
3 for (filename in files){
4   cat("gams ...
       ",substr(filename,12,nchar(filename)),"\n",sep="")
5 }
6 sink()
```