Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Análisis y estudio de los modelos de programación lineal entera en los talleres de flujo regular con tiempos ociosos no permitidos

Autor: Aníbal Valencia Medina

Tutor: Víctor Fernández-Viagas Escudero

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

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Análisis y estudio de los modelos de programación lineal entera en los talleres de flujo regular con tiempos ociosos no permitidos

Autor:

Aníbal Valencia Medina

Tutor:

Víctor Fernández-Viagas Escudero

Profesor ayudante Doctor

Dpto. Organización Industrial y Gestión de Empresa

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2019

Trabajo de Fin de Grado: Análisis y estudio de los modelos de programación lineal entera en los talleres de flujo regular con tiempos ociosos no permitidos

Autor: Aníbal Valencia Media
Tutor: Víctor Fernández-Viagas Escudero

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Agradecimientos

Este proyecto es la finalización a unos años de gran trabajo y esfuerzo dedicados a este grado, por lo que quiero mostrar mi agradecimiento a las personas que me han acompañado en él. En primer lugar a mi familia, la cuál siempre confió en mí.

A mis amigos, con los que juntos hemos realizado este largo viaje.

A todo el personal docente que me ha ayudado a progresar en estos estudios, y en particular a Víctor Fernández-Viagas Escudero, por su dedicación, tiempo y entrega depositados a guiarme y ayudarme a realizar el presente proyecto.

ÍNDICE

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

En el presente proyecto analizaremos el comportamiento de siete modelos diferentes, inicialmente elaborados para de taller de flujo regular, adaptados para el problema con máquinas sin tiempos ociosos. De esta forma, los conocimientos que han sido necesarios para el estudio, análisis y elaboración de este problema pertenecen al ámbito de la organización de la producción y en especial al de programación de la producción.

La organización de la producción es transcendental para las empresas del sector industrial, y se puede definir de manera amplia como todas las actividades que se incluyen en la organización de recursos para, a través de cualquier tipo de procedimiento, la obtención de productos y servicios (Onieva et al., 2006). Dentro de esta, la programación de la producción es una rama de gran importancia y atañe directamente al problema que nos ocupa.

Podemos definir programación como un proceso de toma de decisiones que se ocupa de la asignación de recursos a tareas durante períodos de tiempo determinados y su objetivo es optimizar uno o más objetivos (Pinedo, 2012). La programación tiene una gran variedad de aplicaciones en diversos ámbitos como podrían ser el de transportes, construcción o fabricación. Un ejemplo de esta aplicación sería tomar como recurso las vías de una estación de trenes, como tareas las salidas y entradas de trenes, y como objetivo la minimización del tiempo de llegada de cada tren.

Nuestro caso, el de la programación de la producción, cuenta con unas determinadas especificaciones y características que hacen que se diferencie con otros ámbitos de decisión.

La programación de la producción no es un proceso que se pueda tomar de manera aislada, sino que forma parte de un todo (empresa), por lo que depende de varios

factores y decisiones generales pertenecientes a la ya comentada gestión u organización de la producción.

La programación de la producción o mundialmente conocida como *Manufacturing Scheduling*, se puede definir como la asignación de los distintos recursos de la empresa a la fabricación de un conjunto de productos. El resultado de esta asignación se denomina programa de producción (*production shedule*), y contiene información sobre cuándo debe comenzar cada recurso a procesar qué producto (Fernández-Viagas et al., 2017). En nuestro problema los recursos serán nombrados como máquinas y los productos como trabajos, por lo que en este proyecto tratamos la programación de la producción como la asignación de cuándo cada máquina comienza a procesar qué trabajo.

Los problemas de la programación de la producción en la realidad son complejos y con multitud de variantes y la solución a esto es modelarlos. De esta manera se consigue simplificar la realidad a un entorno medible y cuantificable, además de poder clasificarlos. Esta clasificación se basa en tres características fundamentales:

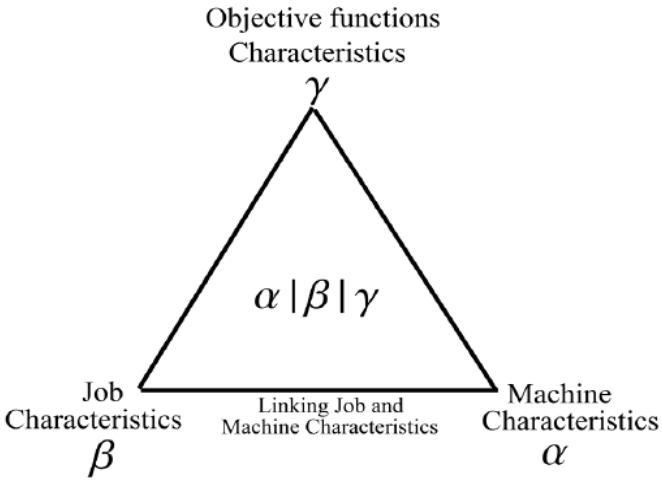

Figura 1.1: Clasificación de los modelos. [Fuente: Fernández-Viagas, 2017]

Características de las máquinas, de los trabajos y de la función objetivo, con símbolos α|β|γ respectivamente (Graham, 1979).

- Características de las máquinas (α): indica con qué tipo y cuántas máquinas se está trabajando. Podemos diferenciar dos grupos:

  -Las máquinas que trabajan en paralelo, por lo que sus trabajos no es necesario que pasen por todas las máquinas. Ejemplo de este tipo podrían ser las *Identical parallel machines* (α=Pm) o las *Uniform parallel machines* (α=Qm)

  -Las máquinas de tipo taller, en las que todas están en serie, cada trabajo debe pasar por cada máquina y cada máquina tiene una finalidad distinta. Hay tres tipos fundamentales, como son el *Job shop* (α=Jm) , *Open Shop* (α=Om) y *Flow Shop* (α=Fm), siendo este último el que nos ocupa y explicaremos más adelante en el capítulo 3.

- Características de los trabajos (β): son las condiciones a las que están sometidas los trabajos, pudiéndolas también reconocer como restricciones. Hay gran variedad de ellas, entre las cuales destacan ser:

  -$\beta = r_j$: fechas de llegada de los trabajos, es decir cuando tenemos a disposición estos trabajos para operar con ellos.

  - $\beta = \bar{d}_j$: fecha obligada de entrega de los trabajos.

  - $\beta = batch$: las máquinas trabajan por lotes de trabajo, es decir, no consideran trabajos de forma individual.

  - $\beta = nwt$: los trabajos no pueden esperar al pasar de una máquina a otra (*no wait*).

  - $\beta = prmu$: la secuencia es la misma para todos los trabajos.

  - $\beta = no - idle$: los trabajos no pueden tener tiempos de inactividad en una máquina. Estas dos últimas restricciones pertenecen también a nuestro problema y serán explicadas en el capítulo 3.

- Características de la función objetivo (γ): que objetivo u optimización se quiere conseguir en el problema. También nos encontramos con múltiples tipos, como podrían ser el sumatorio de los tiempos de terminación de proceso, el tiempo último de finalización de proceso denominado *makespan*, el máximo tiempo en el que un trabajo llega tarde conocido como *maximun tardiness,* o la suma total de los tiempos en el proceso de los trabajos acaban antes de tiempo denominado *total earliness*. Las dos funciones objetivo que nos interesan son el *makespan* y el sumatorio de los tiempos de terminación de proceso, conocido como *sumCi* o $\sum C_i$, que serán también explicadas en el capítulo 3.

Conocemos ya que engloba a un problema de programación de la producción y las características que pueden definir a un problema genérico de esta índole. En los próximos capítulos ahondaremos en detalle en cómo es nuestro problema en particular, cuál es el objeto de este, desarrollándolo y sacando unas conclusiones al respecto.

2. OBJETO DEL PROYECTO

2.1. Objeto del proyecto

En este proyecto abordamos el análisis y estudio de siete modelos distintos de taller de flujo regular los cuales debemos adaptar a la restricción *no-idle*, es decir, en los que las máquinas no presentan periodos de inactividad una vez comience a procesar el primer trabajo. Después debemos evaluar los resultados obtenidos computacionalmente para dos funciones objetivo distintos, siendo estas el tiempo de finalización de todo el proceso, conocido como *makespan*; y la suma de los tiempos de finalización de procesado de cada trabajo, a lo que denominamos *sumCi.* Una vez obtenidos los resultados los comparamos en cada modelo para así sacar conclusiones sobre estos modelos y la implementación de la restricción en ellos para ambas funciones objetivo, pudiendo sacar conclusiones sobre qué modelos son más eficientes sobre otros para sus posibles aplicaciones en un problema de programación de la producción que se pueda dar en la realidad.

Para ello hacemos un estudio de los siete modelos con los que trabajamos en este proyecto, analizando en profundidad cada variable y restricción que los componen y explicándolas con detalle cuales son la función de cada una. Una vez comprendidos los modelos originales, realizamos las modificaciones pertinentes, ya sea modificando, añadiendo o quitando variables y restricciones a los modelos originales para que puedan cumplir con la restricción *no-idle.* Una vez conseguimos adaptar cada modelo a ambas funciones objetivo, los implementamos computacionalmente con una serie de herramientas de software que nos permitirán trabajar de una manera rápida y precisa con los modelos pudiéndolos particularizar a casos concretos con un número determinado de máquinas y trabajos y diferentes tiempos de proceso.

Finalmente, con los resultados obtenidos para la particularización de una gran cantidad de casos para los siete modelos podremos evaluar estos resultados y sacar conclusiones al respecto.

2.2.Justificación

El problema tratado en este proyecto en el que a las máquinas no se les permite tiempos de inactividad en talleres de flujo regular es muy común en la industria y programación de la producción. Existen muchos problemas reales en los que no se permiten o no conviene que las máquinas de un taller estén funcionando periodos donde no procesan ningún trabajo, como podría ser el caso de hornos industriales. Estos al consumir una gran cantidad de energía es conveniente que una vez sean encendidos para recibir el primer producto a tratar reciban todos los productos de forma continuada sin tiempos entre uno y otro hasta que reciban el último producto, para que los hornos estén el menor tiempo posible encendidos y así conseguir un ahorro tanto energético como monetario.

Existen muy pocos estudios para el problema *no-idle*, y de ahí el fin de este proyecto, adecuar varios modelos aplicables para este problema en concreto y evaluar cuáles son más eficientes para que pudieran ser llevados a la práctica.

2.3.Sumario

En los capítulos siguientes trataremos de explicar en profundidad el problema a tratar, los mecanismos y procesos para obtener los resultados de interés y las conclusiones que sacaremos respecto a estos. A continuación, un breve resumen de cada capítulo:

- En el capítulo 3 realizamos una descripción del problema cuestión de este proyecto, como lo abordamos y las características que lo componen.
- En el capítulo 4 describimos en detalle los siete *MILPs* con los que trabajamos en este proyecto.
- En el capítulo 5 realizamos la adaptación de los siete modelos a la restricción *no-idle*.
- En el capítulo 6 explicamos el proceso de implementación de los modelos adaptados al ámbito computacional para poder así trabajar con ellos de una forma rápida y eficaz.
- En el capítulo 7 analizamos los resultados obtenidos para las dos funciones objetivo, comparando los resultados para cada modelo y los resultados entre estas dos funciones objetivo.
- En el capítulo 8 realizamos la valoración tanto de los resultados que han sido analizados en el capítulo anterior como del proyecto en general.
- Finalmente añadiremos la bibliografía utilizada para todo el proyecto, así como un anexo con código de programación que debido a su gran extensión no consideramos incluirlo en los capítulos anteriores.

3. DESCRIPCIÓN DEL PROBLEMA

Según el criterio de clasificación de los modelos de programación de la producción explicado en el capítulo 1, todos los modelos a analizar en este trabajo fueron originalmente propuestos para resolver Fm|prmu|Cmax, es decir, un taller con una estructura de máquinas tipo *Flow shop*, con la restricción *prmu* y cuya función objetivo es minimizar el *makespan*. Estos modelos serán adaptados a la restricción *no-idle,* y además de tener el caso en que la función objetivo es el *makespan,* lo haremos a su vez para la función objetivo $\sum C_i$. Por lo que los modelos de nuestro problema presentan la clasificación de Fm|prmu,no-idle|Cmax y Fm|prmu,no-idle|$\sum C_i$. Seguidamente explicamos en detalle cada característica de nuestros modelos.

En los *Flow shop*, o talleres de flujo regular, cada trabajo debe procesarse en cada una de las máquinas. Todos los trabajos tienen que seguir la misma ruta, es decir, deben procesarse primero en la máquina 1, luego en la máquina 2, y así sucesivamente. Después de finalizar en una máquina, el trabajo se pone en cola en la siguiente máquina. Como norma general, suponemos que todas las colas operan bajo la misma disciplina *First In First Out* (FIFO), es decir, un trabajo no puede adelantarse a otro mientras espera en una cola (Pinedo, 2012).

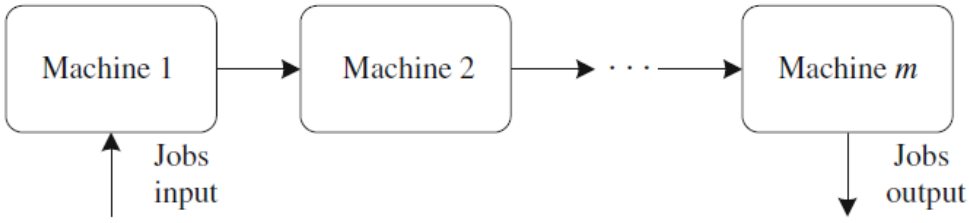

Figura 3.1: Modelo Flow Shop [Fuente: Framinan et al., 2014]

En nuestro caso la regla FIFO está activa, lo que se conoce en *Flow Shop* como taller de flujo de permutación (*prmu*), que es una de las características de los trabajos (β) o restricción con las que cuenta nuestro problema. Definiendo como secuencia

el orden en que se procesan los trabajos dentro de la misma máquina, esta secuencia debe ser la misma para todas las máquinas.

La restricción *no-idle* es la que nos hace replantear los modelos bien sea añadiendo, quitando o modificando las restricciones de cada uno de los modelos. Su traducción directa del inglés es "no-inactivo", haciendo referencia a que las máquinas no pueden tener periodos en los que no estén trabajando, periodos de inactividad. También lo podemos describir como un procesado sin tiempos ociosos entre trabajos.

Proporcionando una definición más amplia, diremos que la secuencia de procesamiento de los trabajos en los que existe una restricción de procesado sin tiempos ociosos debe garantizar que una vez que una máquina comience a procesar, no se detenga hasta que finalice el último trabajo de secuencia (Framinan et al. 2014).

Esta restricción *no-idle* la podemos ver de forma clara ilustrándose gráficamente. En las siguientes figuras se representan dos talleres con restricciones distintas y con el mismo número de máquinas (5), trabajos (3) y tiempos de proceso, siendo estos el tiempo que tarda en procesarse un trabajo *n* en una máquina *m*. Estos tiempos de proceso se muestran en la tabla 3.1:

m/n	Trabajos		
Máquinas	4	5	6
	3	3	4
	2	1	3
	4	5	2
	3	4	9

Tabla 3.1: Tiempos de proceso [Fuente: Elaboración propia]

La figura 3.2 representa un taller de flujo regular únicamente con permutación. Podemos apreciar los distintos tiempos ociosos, con valor total de dos unidades para la segunda máquina y cuatro unidades para la tercera máquina.

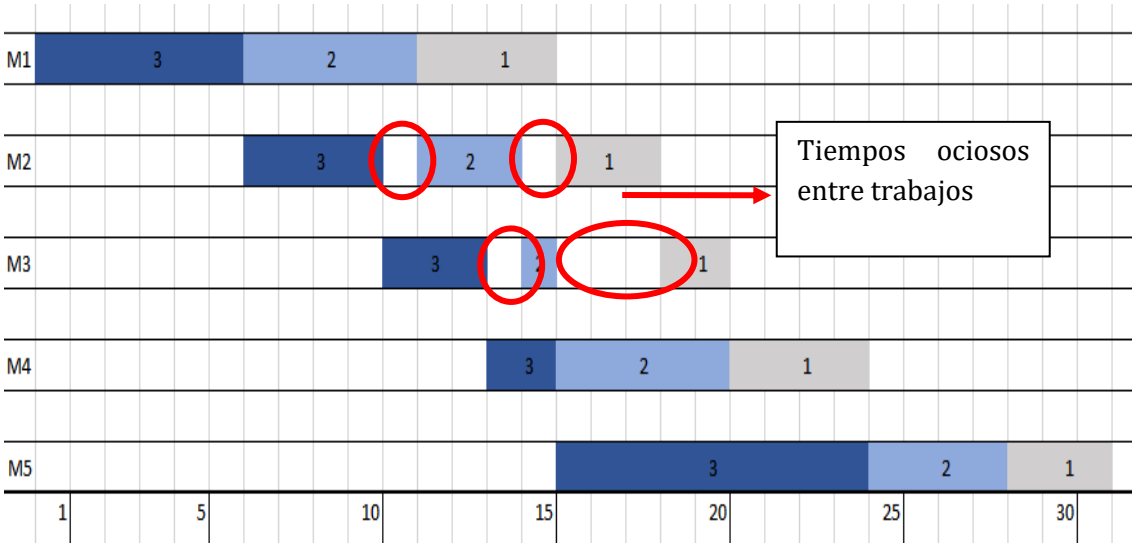

Figura 3.2: Diagrama de Gant- Flow Shop [Fuente: Elaboración propia]

La figura 3.3 representa el mismo taller añadiéndole la restricción *no-idle,* en la que podemos apreciar que el valor de los tiempos occisos es cero. Así se observa claramente que los trabajos son procesados uno detrás de otro en cada máquina, haciendo que estas no tengan periodos de inactividad. Nótese que la secuencia de trabajo de un taller a otro ha cambiado debido a que son con estas secuencias para cada uno de ellos con las que se consigue el óptimo.

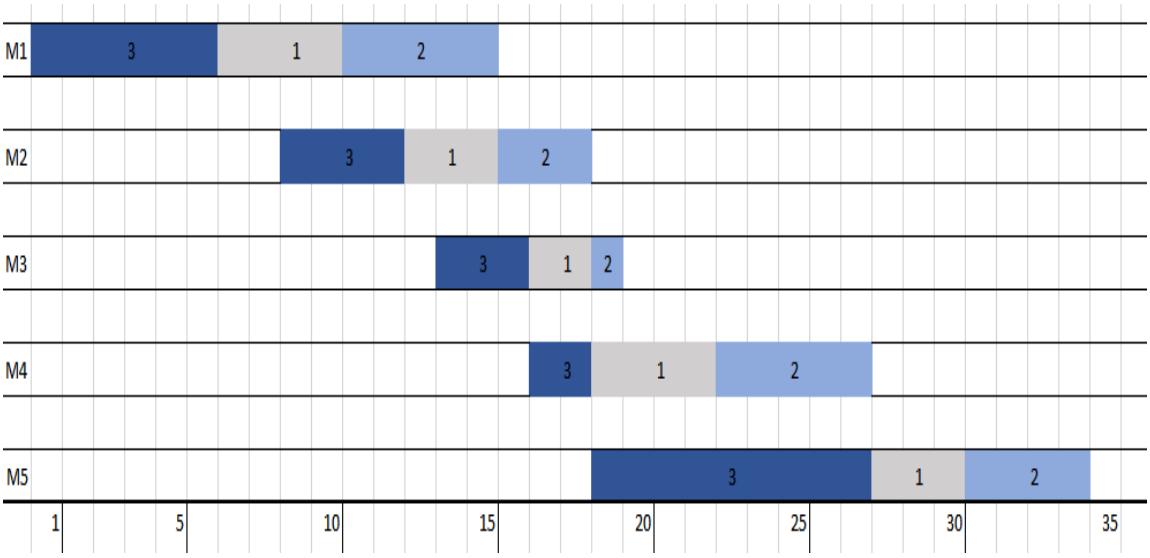

Figura 3.3. Diagrama de Gant- Flow Shop con no-idle [Fuente: Elaboración propia]

Finalmente, la última característica de nuestros modelos, las funciones objetivo a optimizar, serán dos: el *Cmax* o *makespan* y el *sumCi* o $\sum C_i$. El *makespan* es el tiempo de finalización de todo el proceso, es decir, cuando termina el último trabajo en la última máquina, y nuestro objetivo será minimizar este tiempo. En el caso del taller de la figura 3.3 el *makespan* tiene un valor de 34 unidades. En los modelos a adaptar ya vienen por defecto con la función objetivo *makespan,* por lo que únicamente debemos hacer las modificaciones pertinentes en el caso de que sea necesario para adaptar a la restricción *no-idle.* En cuanto al $\sum C_i$, este es el sumatorio de los tiempos de terminación de procesado de los trabajos *i,* tiempo que también debemos minimizar. En el caso de figura 3.3, el *sumCi* sería igual a 27+30+34=91 unidades. Los modelos a los que realizamos la adaptación no cuentan con esta función objetivo, por lo que elaboramos la función desde cero.

Los siete modelos a estudiar son modelos de programación lineal entera mixta, conocidos como MILP *(Mixed Integer Linear Programming)*. Su estructura es la que representamos en la figura 3.4.

$$\begin{aligned} &\min\ c^T x \\ &Ax = b \\ &x \geq 0 \\ &x_i \in \mathbb{Z}\ \ \forall i \in \mathcal{I} \end{aligned}$$

Figura 3.4. Representación MILP [Fuente: Larrosa, 2019]

Donde *min* indica que se pretende minimizar la función objetivo, donde también existe la posibilidad del caso contrario en el que se quiera maximizar dicha función objetivo. Esta queda representada como $c^T x$, donde x son todas las variables representadas de forma genérica en el problema y c^T son los coeficientes asociados a estas variables, también conocidos como costes relativos. En esta estructura le siguen las restricciones donde *A* representa la matriz de coeficientes tecnológicos que van multiplicados por las variables *x* implicadas en las restricciones del modelo y *b* los términos independientes de las restricciones. Estas restricciones pueden ser del tipo =, ≤ ,≥, <, >. Por último, se limitan las variables acotándolas e indicando de que tipo son, libres o binarias.

4. DESCRIPCIÓN DE LOS MILPs A ADAPTAR

En este capítulo presentamos y explicamos en detalle como son los modelos con los que vamos a trabajar y adaptar a la restricción *no-idle*. Son siete modelos en total: tres de la familia Wagner y dos de la Manne, que son descritos y explicados en Stafford et al. (2005), y dos adicionales de la familia Wagner que se encuentran en Tseng y Stafford (2008). Todos los modelos son *MILPs* que resuelven los problemas de taller de flujo regular con permutación para la minimización del *makespan*.

La nomenclatura utilizada por estos artículos es la siguiente:

- T_{ri} ≡ Tiempo de procesado del trabajo *i* en la máquina *r*.
- Z_{ij} ≡ Variable binaria de valor uno si el trabajo *i* está en la posición *j*, cero en caso contrario.
- X_{rj} ≡ Tiempo ocioso antes de que comience un trabajo en la posición *j* en la máquina *r*.
- Y_{rj} ≡ Tiempo de espera del trabajo en la posición *j* después de terminar el procesado en la máquina *r*.
- B_{rj} ≡ Tiempo de inicio de un trabajo en la posición *j* en la máquina *r*.
- E_{rj} ≡ Tiempo de finalización del trabajo en la posición *j* en la máquina *r*.
- C_{ri} ≡ Tiempo de terminación de procesado del trabajo *i* en la máquina *r*.
- D_{ik} ≡ Variable binaria con valor uno si el trabajo *i* está antes del trabajo *k*, y valor cero en caso contrario.
- P ≡ Número suficientemente grande para que se cumpla sólo una de las restricciones del par de restricciones dicotómicas.
- M ≡ Número total de máquinas en el taller.

• $N \equiv$ Número total de trabajos en el taller.

4.1.Familia Wagner

Esta familia de modelos cuenta con cinco miembros los cuales se basan en los primeros modelos desarrollados por Wagner (1959), del que reciben el nombre. Este utilizó el clásico problema de asignación para controlar la asignación de trabajos a posiciones en la secuencia de procesamiento. Cada uno de los cinco miembros de la familia Wagner utiliza el problema de asignación para fines de asignación de trabajo (Staffrod et al., 2005). Es decir, todos los modelos de esta familia van a contar con un par de restricciones para realizar esta asignación de trabajos a posiciones de secuencia, como veremos más adelante.

4.1.1. *Modelo WST*

El primero de los modelos a describir de la familia Wagner es el modelo WST que se basa en cuatro variables de las explicadas al inicio del capítulo. : T_{ri} , Z_{ij}, Y_{rj}, X_{rj}, y se representa de la siguiente manera:

$$Makespan = F_{MAX} = C_{MAX} = C_{MN} = \sum_{i=1}^{N} T_{Mi} + \sum_{p=1}^{N} X_{Mp}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$\sum_{i=1}^{N} T_{r_i} Z_{i,j+1} + X_{r,j+1} + Y_{r,j+1}$$

$$= \sum_{i=1}^{N} T_{r+1,i} Z_{ij} + X_{r+1,j+1} + Y_{rj} \qquad (1 \le r \le M-1; 1 \le j \le N-1) \quad (3)$$

$$X_{r+1,1} = X_{r,1} + Y_{r,1} + \sum_{i=1}^{N} T_{r_i} Z_{i1} \qquad (1 \le r \le M-1) \quad (4)$$

$$Y_{r1} = 0 \qquad (1 \le r \le M-1) \quad (5)$$

Donde sus restricciones tienen las siguientes funciones:

(1) Cada trabajo sólo puede ocupar una posición

(2) No puede haber más de un trabajo en la misma posición.

Este par de restricciones (1) y (2), son las que tienen en común todos los modelos de la familia Wagner y son los que proporcionan la asignación de cada trabajo a cada posición.

(3) El tiempo de proceso del trabajo en una posición más el tiempo ocioso después de esta posición más el tiempo de espera de este trabajo en la máquina anterior tiene que ser igual al trabajo que está situado en la posición inmediatamente posterior pero en la máquina anterior, más el tiempo ocioso existente antes de esta posición más el tiempo de espera que sufre este trabajo en esta máquina.

(4) El tiempo ocioso de una máquina en la primera posición debe de ser igual a el tiempo ocioso en la primera posición de la máquina anterior más el tiempo de espera en la primera posición de la máquina anterior, más el tiempo d proceso del trabajo en la primera posición de la máquina anterior.

Las ecuaciones (3) y (4) son las llamadas *JAML* (*job-adjacency, machine-linkage*) usadas por primera vez por Tseng y Stafford (2001), para vincular las máquinas y los trabajos adyacentes.

5) El tiempo de espera en la posición uno para todas las máquinas debe ser cero.

4.1.2. *Modelo Wilson*

Segundo modelo perteneciente a la familia Wagner cuyas variables características son T_{ri}, Z_{ij}, B_{rj}. En este modelo se sustituyen las ecuaciones *JAML* para usar en su lugar ecuaciones que involucran los tiempos de inicio de los trabajos en cada máquina.

El modelo es como sigue:

$$Makespan = B_{MN} + \sum_{i=1}^{N} T_{Mi} Z_{iN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$B_{1j} + \sum_{i=1}^{N} T_{1i} Z_{ij} = B_{1,j+1} \qquad (1 \leq j \leq N-1)\ (3)$$

$$B_{11} = 0 \qquad (4)$$

$$B_{r1} + \sum_{i=1}^{N} T_{ri} Z_{i1} = B_{r+1,1} \qquad (1 \leq r \leq M-1)\ (5)$$

$$B_{rj} + \sum_{i=1}^{N} T_{ri} Z_{ij} \leq B_{r+1,j} \qquad (1 \leq r \leq M-1; 2 \leq j \leq N)\ (6)$$

$$B_{rj} + \sum_{i=1}^{N} T_{ri} Z_{ij} \leq B_{r,j+1} \qquad (2 \leq r \leq M; 1 \leq j \leq N-1)\ (7)$$

Donde sus restricciones imponen:

(1) Cada trabajo sólo puede ocupar una posición

(2) No puede haber más de un trabajo en la misma posición.

(3) Obliga a cumplir que en la máquina uno no empiece el siguiente trabajo hasta que acabe el anterior.

(4) Obliga a empezar inmediatamente el proceso en la primera máquina.

(5) Los trabajos en la primera posición de cada máquina tienen que empezar inmediatamente en la máquina siguiente justo después de haber terminado en la máquina anterior.

(6) El tiempo de inicio de un trabajo en una posición más el tiempo de proceso de un trabajo en esa posición tiene que ser menor o igual al tiempo de inicio en esa misma posición en la máquina siguiente

(7) El tiempo de inicio de un trabajo en una posición más el tiempo de proceso de un trabajo en esa posición tiene que ser menor o igual al tiempo de inicio del trabajo en la siguiente posición.

4.1.3. *Modelo TS2*

El tercer modelo de la familia Wagner se caracteriza por el uso de las variables T_{ri}, Z_{ij} y E_{rj}, donde existe cierto paralelismo con el modelo Wilson, cambiando la variable de los tiempos de inicio (B_{rj}), por la de los tiempos de finalización (E_{rj}), y con las oportunas restricciones adecuadas a ésta última variable.

Este modelo proviene del modelo TS2 para el problema de flujo regular *SDST* (*sequence-dependent setup times*), tiempos de *setup* que dependen de la secuencia dada (Tseng and Stafford, 2001).

$$Makespan = F_{MAX} = E_{MN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$E_{rj} + \sum_{i=1}^{N} T_{ri} Z_{i,j+1} \leq E_{r,j+1} \qquad (1 \leq r \leq M; 1 \leq j \leq N-1)\ (3)$$

$$E_{rj} + \sum_{i=1}^{N} T_{r+1,i} Z_{ij} \leq E_{r+1,j} \qquad (1 \leq r \leq M-1; 1 \leq j \leq N)\ (4)$$

$$E_{11} \geq \sum_{i=1}^{N} T_{1i} Z_{i1} \qquad (5)$$

Donde sus restricciones nos dicen:

(1) Cada trabajo sólo puede ocupar una posición

(2) No puede haber más de un trabajo en la misma posición.

(3) impone que el tiempo de finalización de un trabajo más el tiempo de proceso del siguiente trabajo en la misma máquina tiene que ser menor o igual al tiempo de finalización del siguiente trabajo.

(4 Establece que el tiempo de finalización de un trabajo más el tiempo de proceso de la misma posición de secuencia en la siguiente máquina tiene que ser menor o igual al tiempo de finalización de este último.

(5) Impone que el tiempo de finalización del primer trabajo en la primera máquina tiene que ser mayor o igual al tiempo de proceso de este mismo.

4.2.Familia Manne

La principal característica de los modelos de la familia Manne es el uso de los tiempos de terminación de procesado (*completion times*). A diferencia de la familia Wagner, no usan las asignaciones de posiciones en una secuencia para los trabajos, sino que directamente son estos tiempos de terminación los que indican en que posición se encuentran cada uno.

4.2.1. *Modelo SGST*

El primer modelo de la familia Manne basa sus restricciones en las variables C_{ri}, T_{ri} y en un número *P,* número suficientemente grande para que se cumpla sólo una de las restricciones del par de restricciones dicotómicas (ecuaciones (3) y (4) que explicaremos a continuación).

El modelo general es el que sigue:

$$Makespan = F_{MAX} = C_{MAX}$$

$$\boldsymbol{s.a}:\ C_{1i} \geq T_{1i} \qquad (1 \leq i \leq N)\ (1)$$

$$C_{r+1,i} - C_{ri} \geq T_{r+1,i} \qquad (1 \leq r \leq M-1; 1 \leq i \leq N)\ (2)$$

$$C_{ri} - C_{rk} + PD_{ik} \geq T_{ri} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (3)$$

$$C_{rk} - C_{ri} + P(1 - D_{ik}) \geq T_{rk} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (4)$$

$$C_{MAX} \geq C_{Mi} \qquad (1 \leq i \leq N)\ (5)$$

Donde sus restricciones imponen:

(1) El tiempo de terminación de procesado de los trabajos en la máquina uno tiene que ser mayor o igual que el tiempo de procesado de estos en esta misma máquina.

(2) El tiempo de finalización de procesado en una máquina de un trabajo menos el tiempo de finalización de procesado de ese mismo trabajo en la máquina anterior tiene que ser mayor o igual al tiempo de procesado del trabajo en la primera máquina antes mencionada.

(3) Una de las restricciones dicotómicas. Impone que El tiempo de terminación de procesado de un trabajo i menos el tiempo de terminación de un trabajo k, más la constante P valor la suma de todos los tiempos de proceso por la variable binaria de valor uno si el trabajo i va antes del trabajo k y cero en caso contrario, debe de ser mayor o igual al tiempo de proceso del trabajo i. la utilidad de esta restricción es para cuando en los casos en los que el trabajo i se encuentre em una posición posterior a k , la diferencia entre estos dos trabajos siempre sea mayor o igual al tiempo de proceso de i. En caso de que i estuviera antes que k, esta ecuación no aportaría nada.

(4) La otra restricción del par dicotómico. Es exactamente igual que la primera pero para el caso contrario, en el que k estaría antes que i.

(5) Obliga a que el máximo tiempo de finalización ha de ser mayor que cada uno de los tiempos de finalización de todos los trabajos en la última máquina.

4.2.2. *Modelo LYeq*

El segundo modelo de la familia Manne proviene del SGST habiéndosele realizado una serie de modificaciones, por lo que comparte variables C_{ri}, T_{ri} y el número P y tres restricciones. A su vez cuenta con una variable Q_{rik}, que es la unión del par de restricciones dicotómicas del modelo SGST unidas en esta variable.

El modelo general es el siguiente:

$$Makespan = F_{MAX} = C_{MAX}$$

$$\boldsymbol{s.a}: \quad C_{1i} \geq T_{1i} \qquad (1 \leq i \leq N)\ (1)$$

$$C_{r+1,i} - C_{ri} \geq T_{r+1,i} \qquad (1 \leq r \leq M-1; 1 \leq i \leq N)\ (2)$$

$$C_{MAX} \geq C_{Mi} \qquad (1 \leq i \leq N)\ (3)$$

$$PD_{ik} + C_{ri} - C_{rk} - T_{r_i} = Q_{rik} \qquad (1 \leq r \leq M; 1 \leq i \leq k \leq N)\ (4)$$

$$Q_{rik} \leq P - T_{ri} - T_{rk} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (5)$$

(1) El tiempo de terminación de procesado de los trabajos en la máquina uno tiene que ser mayor o igual que el tiempo de procesado de estos en esta misma máquina.

(2) El tiempo de finalización de procesado en una máquina de un trabajo menos el tiempo de finalización de procesado de ese mismo trabajo en la máquina anterior tiene que ser mayor o igual al tiempo de procesado del trabajo en la primera máquina antes mencionada.

(3) Obliga a que el máximo tiempo de finalización ha de ser mayor que cada uno de los tiempos de finalización de todos los trabajos en la última máquina.

Estas tres primeras restricciones son las compartidas con el modelo SGST.

(4) El número P igual a la suma de todos los tiempos de proceso por la variable binaria más el tiempo de finalización de procesado del trabajo i menos el tiempo de finalización de procesado del trabajo k menos el tiempo de proceso del trabajo i debe ser igual a la variable Q que aúna las restricciones dicotómicas del modelo anterior.

(5) Marca el límite por arriba de la variable Q como el número P menos el tiempo de proceso del trabajo i menos el tiempo de proceso del trabajo k.

4.3.Modelos adicionales

Estos dos últimos modelos son otros dos modelos pertenecientes a la familia Wagner y se detallan en Tseng y Stafford (2008).

4.3.1. *Modelo TBA*

Las variables utilizadas por parte de este modelo son: T_{ri}, Z_{ij}, X_{rj}. Este modelo proviene de conjuntar varias de las restricciones más significativas de la familia Wagner.

El modelo es el siguiente:

$$Makespan = F_{MAX} = \sum_{p=1}^{M-1} \sum_{i=1}^{N} T_{pi} Z_{i1} + \sum_{i=1}^{N} T_{Mi} + \sum_{s=2}^{N} X_{Ms}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$\sum_{i=1}^{N} T_{r-1,i} Z_{i1} + \sum_{q=1}^{j-1} \sum_{i=1}^{N} (T_{ri} - T_{r-1,i}) Z_{iq} +$$

$$+ \sum_{s=2}^{j} (X_{rs} - X_{r-1,s}) - \sum_{i=1}^{N} T_{r-1,i} Z_{ij} \geq 0 \qquad (2 \leq r \leq M; 2 \leq j \leq N)\ (3)$$

$$X_{1j} = 0 \qquad (2 \leq j \leq N)\ (4)$$

Por lo que las restricciones imponen:

(1) Cada trabajo sólo puede ocupar una posición

(2) No puede haber más de un trabajo en la misma posición.

(3) El primer trabajo de la primera máquina más la diferencia de un trabajo en una máquina menos el mismo trabajo en la máquina anterior, más los tiempos ociosos entre trabajos menos el trabajo en la maquina anterior debe ser mayor igual a cero. O dicho de otra manera, el bloque de trabajos más tiempos ociosos de una máquina, sin contar el trabajo en la última posición, debe de ser mayor o igual al bloque de trabajos más tiempos ociosos de la máquina anterior contando con el trabajo en la última posición.

(4) Todos los tiempos ociosos entre trabajos en la primera máquina deben de ser cero.

4.3.2. *Modelo TS3*

Finalmente llegamos al último modelo el TS3 último de la familia Wagner y último de nuestro estudio. Las variables que utiliza este modelo son T_{ri}, Z_{ij} y Y_{rj}.

$$Makespan = F_{MAX} = \sum_{p=1}^{N} T_{1p} + \sum_{q=2}^{M} \sum_{i=1}^{N} T_{qi} Z_{iN} + \sum_{q=1}^{M-1} Y_{qN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$\sum_{i=1}^{N} T_{1i} Z_{i,j-1} - \sum_{i=1}^{N} T_{r_i} Z_{i,j-1} +$$

$$+ \sum_{q=1}^{r-1} \sum_{i=1}^{N} T_{qi}\left(Z_{ij} - Z_{i,j-1}\right) + \sum_{q=1}^{r-1}\left(Y_{qj} - Y_{q,j-1}\right) \geq 0 \quad (2 \leq r \leq M; 2 \leq j \leq N)\ (3)$$

$$Y_{r1} = O \qquad (1 \leq r \leq M-1)\ (4)$$

Donde sus restricciones nos dicen:

(1) Cada trabajo sólo puede ocupar una posición

(2) No puede haber más de un trabajo en la misma posición.

(3) Muy parecida a la restricción (3) del modelo anterior (TBA), en los que están involucrados los tiempos de espera en vez de los tiempos ociosos entre trabajos.

(4) El tiempo de espera del trabajo en la posición uno para todas las máquinas debe ser cero.

5. ADAPTACIÓN DE LOS MILPs

Habiendo descrito y detallado los modelos a modificar en el capítulo anterior, en este capítulo procedemos a adaptar los modelos a la restricción *no-idle*, tanto para la función objetivo *Cmax* como para $\sum C_i$ para que nuestras soluciones sean las de un taller de permutación sin tiempos ociosos. Así pues, siguiendo el mismo orden que en el capítulo 4, detallaremos modelo a modelo como han sido estas adaptaciones.

5.1. Adaptación familia Wagner

5.1.1. *Modelo WST*

El modelo ya explicado es:

$$Makespan = F_{MAX} = C_{MAX} = C_{MN} = \sum_{i=1}^{N} T_{Mi} + \sum_{p=1}^{N} X_{Mp}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \le i \le N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \le j \le N)\ (2)$$

$$\sum_{i=1}^{N} T_{r_i} Z_{i,j+1} + X_{r,j+1} + Y_{r,j+1}$$

$$= \sum_{i=1}^{N} T_{r+1,i} Z_{ij} + X_{r+1,j+1} + Y_{rj} \qquad (1 \le r \le M-1; 1 \le j \le N-1)\ (3)$$

$$X_{r+1,1} = X_{r,1} + Y_{r,1} + \sum_{i=1}^{N} T_{r_i} Z_{i1} \qquad (1 \le r \le M-1)\ (4)$$

$$Y_{r1} = 0 \qquad (1 \le r \le M-1)\ (5)$$

A la hora de adaptar este modelo a la restricción no-idle, tenemos como ventaja que los tiempos ociosos están representados directamente con una variable (X_{rj}). Esta variable es bidimensional, y representa en este modelo tanto los tiempos de inactividad de las máquinas entre trabajos como los tiempos de inactividad hasta que comience el primer trabajo en la máquina (o lo que es lo mismo, el tiempo ocioso en la primera posición). La clave en este modelo es modificar dicha variable para que únicamente represente los tiempos ociosos en los que la máquina espera que le llegue el primer trabajo, eliminando así los tiempos de inactividad entre trabajos y pasando a ser una variable unidimensional (X_r). De esta manera desaparecen las variables X_{rj} en la ecuación (3), y en la ecuación (4), que cambian a la variable modificada X_r, (para la primera posición).

También eliminamos restricción (5), que de mantenerse podría obligar a las máquinas a tener tiempos de inactividad.

Por último, modificamos también la función objetivo, cambiando la variable X anteriormente explicada.

De esta forma quedaría:

$$Makespan = F_{MAX} = C_{MAX} = C_{MN} = \sum_{i=1}^{N} T_{Mi} + X_M$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$\sum_{i=1}^{N} T_{ri}\, Z_{i,j+1} + Y_{r,j+1}$$

$$= \sum_{i=1}^{N} T_{r+1,i} Z_{ij} + Y_{rj} \qquad (1 \le r \le M-1; 1 \le j \le N-1) \quad (3)$$

$$X_{r+1} = X_r + Y_{r,1} + \sum_{i=1}^{N} T_{r_i} Z_{i1} \qquad (1 \le r \le M-1) \quad (4)$$

A su vez realizamos una nueva función objetivo para el cálculo de $\sum C_i$ que queda de la siguiente forma:

$$sumC_i = \sum_{j=1}^{N} \sum_{i=1}^{N} T_{Mi} Z_{ij} (n+1-j) + n * X_M$$

5.1.2. *Modelo Wilson*

$$Makespan = B_{MN} + \sum_{i=1}^{N} T_{Mi} Z_{iN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \le i \le N) \ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \le j \le N) \ (2)$$

$$B_{1j} + \sum_{i=1}^{N} T_{1i} Z_{ij} = B_{1,j+1} \qquad (1 \le j \le N-1) \ (3)$$

$$B_{11} = 0 \qquad (4)$$

$$B_{r1} + \sum_{i=1}^{N} T_{ri} Z_{i1} = B_{r+1,1} \qquad (1 \le r \le M-1) \ (5)$$

$$B_{rj} + \sum_{i=1}^{N} T_{ri} Z_{ij} \le B_{r+1,j} \qquad (1 \le r \le M-1; 2 \le j \le N) \ (6)$$

$$B_{rj} + \sum_{i=1}^{N} T_{ri} Z_{ij} \leq B_{r,j+1} \qquad (2 \leq r \leq M; 1 \leq j \leq N-1)\ (7)$$

En este modelo la adaptación resulta sencilla, ya que sólo ha hecho falta un par de cambios de signo como veremos a continuación, siendo las restricciones a modificar la (5) y (7). En la restricción (5) será necesario cambiar el signo de = por ≤, para así no obligar a empezar el trabajo en la máquina siguiente inmediatamente después de acabar el trabajo en la máquina anterior, evitando así que pudiera haber tiempos ociosos. Con la (7) se hace lo contrario, cambiando el signo ≤ por el de =, para obligar que, en una misma máquina, el siguiente trabajo empiece justo después, y se evita de esta manera posibles tiempos ociosos en las máquinas. La función objetivo no ha sido necesaria modificarla.

El modelo adaptado quedaría de la siguiente manera:

$$Makespan = B_{MN} + \sum_{i=1}^{N} T_{Mi} Z_{iN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$B_{1j} + \sum_{i=1}^{N} T_{1i} Z_{ij} = B_{1,j+1} \qquad (1 \leq j \leq N-1)\ (3)$$

$$B_{11} = 0 \qquad (4)$$

$$B_{r1} + \sum_{i=1}^{N} T_{ri} Z_{i1} \leq B_{r+1,1} \qquad (1 \leq r \leq M-1)\ (5)$$

$$B_{rj} + \sum_{i=1}^{N} T_{ri} Z_{ij} \leq B_{r+1,j} \qquad (1 \leq r \leq M-1; 2 \leq j \leq N)\ (6)$$

$$B_{rj} + \sum_{i=1}^{N} T_{ri} Z_{ij} = B_{r,j+1} \qquad (2 \leq r \leq M; 1 \leq j \leq N-1)\ (7)$$

Cuya función objetivo para el $\sum C_i$ la representamos de la siguiente manera:

$$sumC_i = \sum_{j=1}^{N} B_{Mj} + \sum_{j=1}^{N} \sum_{i=1}^{N} T_{Mi} Z_{ij}$$

5.1.3. *Modelo TS2*

Sabemos que la estructura del modelo es la siguiente:

$$Makespan = F_{MAX} = E_{MN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$E_{rj} + \sum_{i=1}^{N} T_{ri} Z_{i,j+1} \leq E_{r,j+1} \qquad (1 \leq r \leq M; 1 \leq j \leq N-1)\ (3)$$

$$E_{rj} + \sum_{i=1}^{N} T_{r+1,i} Z_{ij} \leq E_{r+1,j} \qquad (1 \leq r \leq M-1; 1 \leq j \leq N)\ (4)$$

$$E_{11} \geq \sum_{i=1}^{N} T_{1i} Z_{i1} \qquad (5)$$

Al igual que el modelo Wilson, la adaptación no conlleva una gran complejidad. Sólo es necesario modificar un par restricciones, en concreto la (3) y (5), cambiando el signo ≤ y ≥ respectivamente por el de =. De esta manera se obliga a que un trabajo termine inmediatamente antes de empezar el siguiente, evitando así tiempos ociosos en todas las máquinas.

El modelo quedaría de la siguiente manera:

$$Makespan = F_{MAX} = E_{MN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$E_{rj} + \sum_{i=1}^{N} T_{ri} Z_{i,j+1} = E_{r,j+1} \qquad (1 \leq r \leq M; 1 \leq j \leq N-1)\ (3)$$

$$E_{rj} + \sum_{i=1}^{N} T_{r+1,i} Z_{ij} \leq E_{r+1,j} \qquad (1 \leq r \leq M-1; 1 \leq j \leq N)\ (4)$$

$$E_{11} = \sum_{i=1}^{N} T_{1i} Z_{i1} \qquad (5)$$

La función objetivo para $\sum C_i$ es tan simple como:

$$sumC_i = \sum_{j=1}^{N} E_{Mj}$$

5.2.Adaptación familia Manne

La adaptación de los dos modelos de la familia resulta de mayor complejidad que los anteriores.

5.2.1. *Modelo SGST*

Sabemos que este modelo se representa como:

$$Makespan = F_{MAX} = C_{MAX}$$

$$s.a:\ C_{1i} \geq T_{1i} \qquad (1 \leq i \leq N)\ (1)$$

$$C_{r+1,i} - C_{ri} \geq T_{r+1,i} \qquad (1 \leq r \leq M-1; 1 \leq i \leq N)\ (2)$$

$$C_{ri} - C_{rk} + PD_{ik} \geq T_{ri} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (3)$$

$$C_{rk} - C_{ri} + P(1 - D_{ik}) \geq T_{rk} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (4)$$

$$C_{MAX} \geq C_{Mi} \qquad (1 \leq i \leq N)\ (5)$$

En la adaptación comenzamos por añadir otro par de restricciones dicotómicas muy parecidas a las restricciones (3) y (4), además de crear una nueva variable, U_{ki}, que toma el valor 1 si el trabajo i precede al trabajo k, y cero en caso contrario. El nuevo par de restricciones dicotómicas es el siguiente:

$$C_{ri} - C_{rk} + PD_{ik} \leq T_{ri} + P(1 - U_{ki}) \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (7)$$

$$C_{rk} - C_{ri} + P(1 - D_{ik}) \leq T_{rk} + P(1 - U_{ik}) \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (8)$$

De esta manera, junto con el par de ecuaciones dicotómicas que ya existentes en el modelo, obligamos a que el signo de menor o igual se convierta en el de igual sólo cuando el trabajo i precede al trabajo k, es decir, cuando la variable U_{ki} toma el valor de 1. Y para el caso contrario lo mismo, U_{ik} toma valor 1 sólo cuando el trabajo k precede al trabajo i. Así en estas restricciones se tienen en cuenta para

cada par de trabajos si uno es predecesor del otro o viceversa, pero se da también la posibilidad de que no se precedan entre ellos. Por esta razón, hemos añadido una restricción más, que impone cuántas variables binarias U_{ki} y U_{ik} con valor uno debe haber, es decir, cuantos predecesores existen, y este es el número de trabajos menos uno, ya que todos los trabajos deben tener un predecesor menos el último trabajo al que, lógicamente, no le precede ninguno. Por lo que la última restricción a añadir es:

$$\sum_{i=1}^{N} \sum_{k=2}^{N} U_{ik} + U_{ki} = N - 1 \qquad (1 \leq i \leq N)\ (1)$$

Así pues, el modelo adaptado es el siguiente:

$$Makespan = F_{MAX} = C_{MAX}$$

$$\boldsymbol{s.a}: \sum_{i=1}^{N} \sum_{k=2}^{N} U_{ik} + U_{ki} = N - 1 \qquad (1 \leq i \leq N)\ (1)$$

$$C_{1i} \geq T_{1i} \qquad (1 \leq i \leq N)\ (2)$$

$$C_{r+1,i} - C_{ri} \geq T_{r+1,i} \qquad (1 \leq r \leq M-1; 1 \leq i \leq N)\ (3)$$

$$C_{ri} - C_{rk} + PD_{ik} \geq T_{ri} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (4)$$

$$C_{rk} - C_{ri} + P(1 - D_{ik}) \geq T_{rk} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (5)$$

$$C_{MAX} \geq C_{Mi} \qquad (1 \leq i \leq N)\ (6)$$

$$C_{ri} - C_{rk} + PD_{ik} \leq T_{ri} + P(1 - U_{ki}) \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (7)$$

$$C_{rk} - C_{ri} + P(1 - D_{ik}) \leq T_{rk} + P(1 - U_{ik}) \quad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (8)$$

Donde para el $\sum C_i$ tenemos que es la propia definición de esta función objetivo, directamente calcular el sumatorio de los tiempos de terminación de procesado de todos los trabajos.

$$sumC_i = \sum_{i=1}^{N} C_{Mi}$$

A diferencia del resto de modelos, en este caso también se modifican las variables y restricciones para esta función objetivo. Eliminamos la variable C_{MAX} y la restricción (6) ya que son innecesarias.

5.2.2. *Modelo LYeq*

$Makespan = F_{MAX} = C_{MAX}$

$\boldsymbol{s.a}$: $C_{1i} \geq T_{1i} \qquad (1 \leq i \leq N)\ (1)$

$$C_{r+1,i} - C_{ri} \geq T_{r+1,i} \qquad (1 \leq r \leq M-1; 1 \leq i \leq N)\ (2)$$

$$C_{MAX} \geq C_{Mi} \qquad (1 \leq i \leq N)\ (3)$$

$$PD_{ik} + C_{ri} - C_{rk} - T_{r_i} = Q_{rik} \qquad (1 \leq r \leq M; 1 \leq i \leq k \leq N)\ (4)$$

$$Q_{rik} \leq P - T_{ri} - T_{rk} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (5)$$

La adaptación de este modelo una vez realizada la del modelo SGST resulta muy sencilla, ya que sólo es necesario añadir las tres mismas restricciones que añadimos en el modelo SGST, debido a la ya comentada gran semejanza que hay entre estos. Por lo que quedaría:

$Makespan = F_{MAX} = C_{MAX}$

$$\boldsymbol{s.a}: \sum_{i=1}^{N} \sum_{k=2}^{N} U_{ik} + U_{ki} = N - 1 \qquad (1 \leq i \leq N)\ (1)$$

$$C_{r+1,i} - C_{ri} \geq T_{r+1,i} \qquad (1 \leq r \leq M-1; 1 \leq i \leq N)\ (2)$$

$$C_{MAX} \geq C_{Mi} \qquad (1 \leq i \leq N)\ (3)$$

$$PD_{ik} + C_{ri} - C_{rk} - T_{r_i} = Q_{rik} \qquad (1 \leq r \leq M; 1 \leq i \leq k \leq N)\ (4)$$

$$Q_{rik} \leq P - T_{ri} - T_{rk} \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (5)$$

$$C_{ri} - C_{rk} + PD_{ik} \leq T_{ri} + P(1 - U_{ki}) \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (6)$$

$$C_{rk} - C_{ri} + P(1 - D_{ik}) \leq T_{rk} + P(1 - U_{ik}) \qquad (1 \leq r \leq M; 1 \leq i < k \leq N)\ (7)$$

Al igual que en el modelo SGST para $\sum C_i$tenemos:

$$sumC_i = \sum_{i=1}^{N} C_{Mi}$$

Y eliminamos la misma variable C_{MAX} y su correspondiente restricción.

5.3.Adaptación modelos adicionales

5.3.1. *Modelo TBA*

El modelo es el siguiente:

$$Makespan = F_{MAX} = \sum_{p=1}^{M-1} \sum_{i=1}^{N} T_{pi} Z_{i1} + \sum_{i=1}^{N} T_{Mi} + \sum_{s=2}^{N} X_{Ms}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \le i \le N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \le j \le N)\ (2)$$

$$\sum_{i=1}^{N} T_{r-1,i} Z_{i1} + \sum_{q=1}^{j-1} \sum_{i=1}^{N} (T_{ri} - T_{r-1,i}) Z_{iq} +$$

$$+ \sum_{s=2}^{j} (X_{rs} - X_{r-1,s}) - \sum_{i=1}^{N} T_{r-1,i} Z_{ij} \ge 0 \qquad (2 \le r \le M; 2 \le j \le N)\ (3)$$

$$X_{1j} = 0 \qquad (2 \le j \le N)\ (4)$$

A la hora de adaptarlo a la restricción *no-idle*, este es el modelo que más cambios realizamos y en el que más dificultades encontramos. Comenzamos por cambiar la variable X_{rs}, bidimensional, por una variable unidimensional X_r, que representa el tiempo ocioso que existe entre que finaliza el primer trabajo de una máquina hasta que comienza el primer trabajo de la máquina siguiente (No confundir con la X_r del modelo WST, no son iguales). A continuación se ha modificado la restricción (3), cambiando el sumatorio de los tiempos ociosos por introducir únicamente la variable unidimensional ya mencionada. Se deben añadir tiempos de finalización a este modelo (E_{rj}) con sus correspondientes restricciones. Estas se han basado en las del modelo TS2, con las modificaciones correspondientes. Una de ellas ha sido la de dividir la restricción $E_{rj} + \sum_{i=1}^{N} T_{r+1,i} Z_{ij} \leq E_{r+1,j}$ en dos distintas, una particularizando para el primer trabajo en cada máquina añadiéndose los tiempos ociosos X_r (5), y la otra para el resto de trabajos sin estos tiempos ociosos (6). A su vez se ha eliminado la última restricción, innecesaria. Por último, habría que modificar la FO, simplemente cambiando la variable bidimensional por la unidimensional, haciendo sumatorio para todas las máquinas.

El modelo quedaría de la siguiente manera:

$$Makespan = F_{MAX} = \sum_{p=1}^{M-1} \sum_{i=1}^{N} T_{pi} Z_{i1} + \sum_{i=1}^{N} T_{Mi} + \sum_{s=2}^{M} X_r$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$\sum_{i=1}^{N} T_{r-1,i} Z_{i1} + \sum_{q=1}^{j-1} \sum_{i=1}^{N} (T_{ri} - T_{r-1,i}) Z_{iq} +$$

$$+X_r - \sum_{i=1}^{N} T_{r-1,i} Z_{ij} \geq 0 \qquad (2 \leq r \leq M; 2 \leq j \leq N)\ (3)$$

$$E_{rj} + \sum_{i=1}^{N} T_{ri} Z_{i,j+1} = E_{r,j+1} \qquad (1 \leq r \leq M; 1 \leq j \leq N-1)\ (4)$$

$$E_{r1} + \sum_{i=1}^{N} T_{r+1,i} Z_{i1} + X_{r+1} = E_{r+1,1} \qquad (1 \leq r \leq M-1)\ (5)$$

$$E_{rj} + \sum_{i=1}^{N} T_{r+1,i} Z_{ij} \leq E_{r+1,j} \qquad (1 \leq r \leq M-1; 2 \leq j \leq N)\ (6)$$

Para el $\sum C_i$ quedaría:

$$sumC_i = n * \sum_{q=1}^{M-1} \sum_{i=1}^{N} T_{qi} Z_{i1} + \sum_{j=1}^{N} \sum_{i=1}^{N} T_{Mi} Z_{ij} (n+1-j) + n * \sum_{s=2}^{M} X_s$$

5.3.2. *Modelo TS3*

Ya sabemos que este modelo se representa como:

$$Makespan = F_{MAX} = \sum_{p=1}^{N} T_{1p} + \sum_{q=2}^{M} \sum_{i=1}^{N} T_{qi} Z_{iN} + \sum_{q=1}^{M-1} Y_{qN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$\sum_{i=1}^{N} T_{1i} Z_{i,j-1} - \sum_{i=1}^{N} T_{r_i} Z_{i,j-1} +$$

$$+ \sum_{q=1}^{r-1} \sum_{i=1}^{N} T_{qi}\left(Z_{ij} - Z_{i,j-1}\right) + \sum_{q=1}^{r-1}\left(Y_{qj} - Y_{q,j-1}\right) \geq 0 \ \ (2 \leq r \leq M; 2 \leq j \leq N)\ (3)$$

$$Y_{r1} = O \qquad (1 \leq r \leq M-1)\ (4)$$

Y finalmente llegamos a la última adaptación a realizar, la cual resulta ser muy sencilla, pues únicamente cambiamos el signo de la restricción (3) de mayor o igual por el de un igual, desapareciendo así los tiempos ociosos entre máquinas. Además eliminamos la restricción (4), que de mantenerla podría provocar tiempos ociosos entre los primeros y segundos trabajos de cada máquina.

Por lo que el modelo final queda:

$$Makespan = F_{MAX} = \sum_{p=1}^{N} T_{1p} + \sum_{q=2}^{M} \sum_{i=1}^{N} T_{qi} Z_{iN} + \sum_{q=1}^{M-1} Y_{qN}$$

$$\boldsymbol{s.a}: \sum_{j=1}^{N} Z_{ij} = 1 \qquad (1 \leq i \leq N)\ (1)$$

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad (1 \leq j \leq N)\ (2)$$

$$\sum_{i=1}^{N} T_{1i} Z_{i,j-1} - \sum_{i=1}^{N} T_{r_i} Z_{i,j-1} +$$

$$+ \sum_{q=1}^{r-1} \sum_{i=1}^{N} T_{qi} \left(Z_{ij} - Z_{i,j-1} \right) + \sum_{q=1}^{r-1} \left(Y_{qj} - Y_{q,j-1} \right) = 0 \quad (2 \leq r \leq M; 2 \leq j \leq N) \ (3)$$

La función objetivo para $\sum C_i$ es:

$$sumC_i = n * \sum_{q=1}^{M-1} \sum_{i=1}^{N} T_{qi} Z_{i1} + \sum_{j=1}^{N} \sum_{i=1}^{N} T_{Mi} Z_{ij} (n + 1 - j) + n * \sum_{q=1}^{M-1} Y_{q1}$$

6. IMPLEMENTACIÓN DE LOS MILPs

Una vez realizadas las adaptaciones en cada modelo, debemos realizar un análisis y evaluación de estos particularizando con una gran cantidad de datos para cada modelo que nos proporcionan una serie de resultados para poder así sacar conclusiones al respecto. Esta evaluación y conclusiones las realizamos en los capítulos 7 y 8, pero para llegar esta evaluación de datos ha sido necesaria una implementación de estos modelos al ámbito computacional para poder particularizar con estos datos de forma rápida, precisa y eficaz. Esta implementación se basa en una serie de pasos que son explicados a continuación.

6.1. Programación C en Code::Blocks

Una vez sabemos cómo quedan todas las restricciones y funciones objetivo de cada modelo, el primer paso para la implementación se basa en traducir estos modelos a lenguaje C, utilizando el programa Code::Blocks. Este es un IDE (Integrated Development Enviroment o en español entorno de desarrollo integrado) para C, C++ y Fortan (aunque en este proyecto sólo se ha hecho uso del lenguaje C). Es un entorno totalmente configurable sobre un marco para *plugins*, puede extenderse por medio de diferentes *plugins*, incorporando cualquier tipo de funcionalidad mediante la instalación o codificación de un plugin especifico (por ejemplo, la funcionalidad de compilación y *debugging*). (https://www.ucm.es/pimcd2014-free-software /codeblocks)

Entre sus diferentes posibilidades está la del uso de librerías, siendo estos archivos que podemos añadir a nuestro programa para que incluyan diferentes funciones ya elaboradas que podremos utilizar a posteriori. Ejemplos de estas pueden ser *stdlib.h* o *schedule_lib.h,* donde esta última está enfocada a la programación de operaciones. En nuestro caso utilizamos Code::Blocks para leer una batería de

archivos o instancias especificas en la que cada instancia tiene un número determinado de máquinas, trabajos y tiempos de proceso distintos para crear mediante el uso de ficheros archivos de texto los cuales contienen los *MILPs* particularizados a cada instancia. En nuestro problema es necesario crear 14 programas en código C para Code::Blocks, por las dos funciones objetivo para cada uno de los siete modelos con los que trabajamos.

```c
#include <stdio.h>
#include <stdlib.h>
#include <schedule_lib.h>
void WSTmodel(int n, int m, MAT_INT Tpro, FILE *secunfichero);
int main()
{
printf("Modelo WST \n");
int m,n,k,j,i;
int  cont=0;
char  cadenacar[50]={0};
char  cadenacar2[50]={0};
for (j=5;j<=20;)

{
 for (k=2;k<=5;k++)

{
for (i=1;i<=10;i++)
{
sprintf(cadenacar,"Bateria Sin fechas/inst_%d_%d_p_%d",j,k,i);
FILE * primerfichero = fopen(cadenacar,"r");
fscanf(primerfichero,"%d %d",&m,&n);
MAT_INT Tpro = DIM_MAT_INT(m,n);
Tpro=loadTimes_arows(cadenacar,&n,&m,YES);
sprintf(cadenacar2,"gurobi/WSTmodel_%d_%d_%d.lp",j,k,i);
FILE * secunfichero=fopen(cadenacar2,"w");
WSTmodel(n,m,Tpro,secunfichero);

fclose(secunfichero);
fclose(secunfichero);
fflush(stdin);
cont=cont+1;
}
}
  j=j+5;
}
return cont;
}
void WSTmodel(int n, int m, MAT_INT Tpro, FILE *secunfichero)
{
int i,j,r,p;
//FO Cmax
fprintf(secunfichero,"Minimize \n");
for (i=1;i<=n;i++)
```

Lectura de cada instancia

Escribimos el modelo en un archivo de texto en formato LP

De aquí en adelante es la función donde programamos para que escriba en el fichero el modelo particularizado a una instancia

Figura 6.1: Ejemplo de programación en Code::Blocks del modelo WST
[Fuente: elaboración propia]

En la figura 6.1 podemos ver un ejemplo para el caso del modelo WST con función objetivo *makespan*, en el que sólo mostramos el inicio del código el cual es muy similar para todos los programas, que se basa en la lectura de las instancias desde un archivo externo y la creación de un archivo por cada instancia donde se escribe el modelo implementado según la instancia correspondiente a través de la función codificada en el mismo programa para cada modelo.

Así pues, gracias a Code::Blocks generamos archivos de texto en los que aparecen los modelos completos particularizados a cada instancia. Estos archivos de texto deben estar en formato LP (extensión ".lp"), que son los que usara el programa gurobi para las resoluciones de los mismos.

6.2.Gurobi

Gurobi es un *solver* (solucionador) de optimización comercial que trabaja con diferentes lenguajes como Python, MATLAB o el que nos ocupa, en C. Una vez creado los archivos LP por por parte de Code::Blocks, Gurobi se encarga de resolver los modelos. Estos archivos LP deben estar escritos cumpliendo unas características determinadas para su correcta lectura, como se indica en la web oficial de Gurobi (https://www.gurobi.com/documentation/). Entre estas podemos poner de ejemplo que en las funciones objetivo no pueden aparecer términos independientes o que para multiplicar una constante por una variable se ha de escribir poniendo únicamente un espacio entre estas dos.

wtsgurobi - Notepad

File Edit Format View Help

```
Minimize
T_2_1 + T_2_2 + X_2
 Subject To
Z_1_1 + Z_1_2 = 1
Z_2_1 + Z_2_2 = 1
Z_1_1 + Z_2_1 = 1
Z_1_2 + Z_2_2 = 1
 4 Z_1_2  +  5 Z_2_2  + Y_1_2 -  3 Z_1_1 -  4 Z_2_1 - Y_1_1 = 0
X_2 - X_1 - Y_1_1 -  4 Z_1_1  -  5 Z_2_1  = 0
T_2_1 = 3
T_2_2 = 4
Bounds
Y_1_1 >= 0
Y_1_2 >= 0
Generals
Y_1_1
Y_1_2
Binary
Z_1_1
Z_1_2
Z_2_1
Z_2_2
End
```

Figura 6.2: Ejemplo de archivo LP [Fuente: elaboración propia]

La figura 6.2 es un ejemplo para la implementación del modelo WST para la función objetivo *Cmax* con dos máquinas y dos trabajos en formato LP para la lectura de Gurobi. Para la lectura y resolución de estos archivos Gurobi trabaja desde el comando del ordenador (*cmd*), a través del cual llamamos a Gurobi para que nos resuelva el modelo escrito en formato LP, como se puede observar en el ejemplo de la figura 6.3.

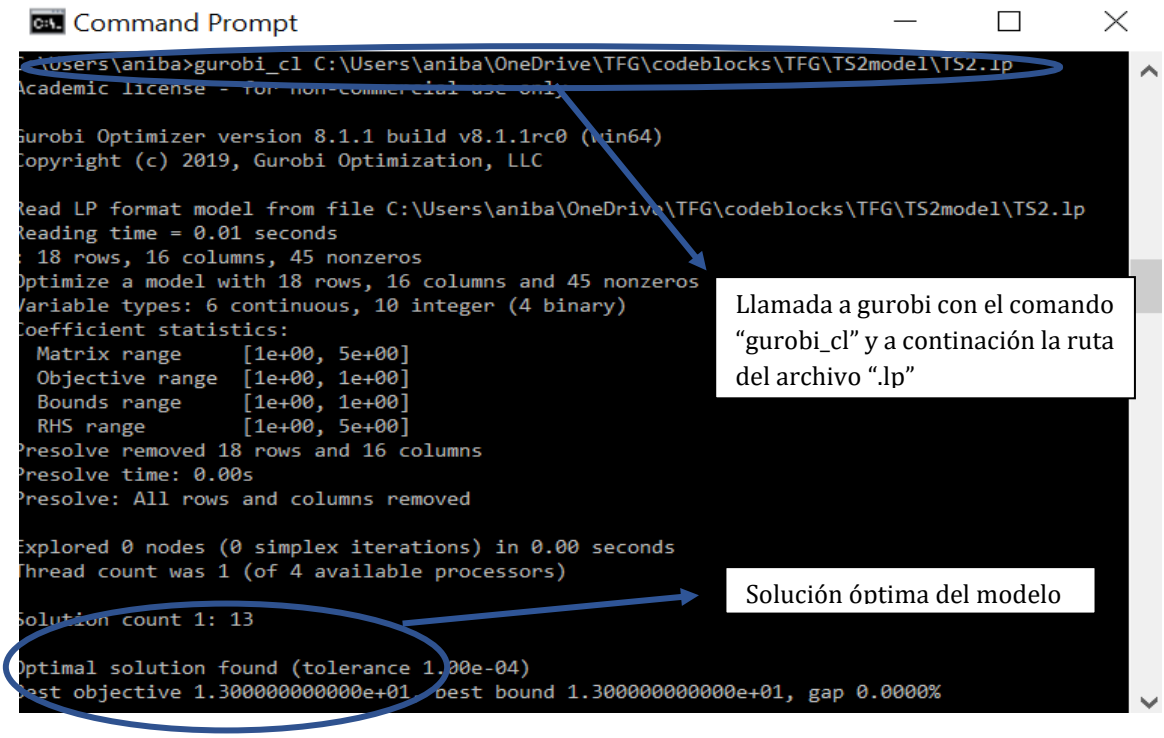

Figura 6.3: Cuadro cmd-llamada a gurobi. [Fuente: Elaboración propia]

Gurobi además nos ofrece otras posibilidades como los archivos SOL (extensión ".sol"), que proporcionan el valor que toman todas las variables en la solución encontrada, una función muy útil para comprobar que nuestros modelos no presentan tiempos ociosos y para corregirlos en el caso de que tuviesen. En la adaptación e implementación de cada modelo, hemos ido comprobando uno a uno si estas eran correctas. Para ello comenzamos resolviendo con Gurobi un caso de un tamaño pequeño, en concreto dos trabajos y dos máquinas, y con los resultados obtenidos con el archivo ".sol", comprobamos dibujando el diagrama de Gant correspondiente si estos resultados cumplen con los requisitos impuestos para cada modelo. Una vez verificados los resultados, seguidamente realizamos el mismo procedimiento con un caso más complejo, tres trabajos y tres máquinas, para tener una mayor seguridad de que las adaptaciones impuestas han sido correctas.

Los tiempos de proceso para el caso de dos máquinas y dos trabajos para cada modelo han sido siempre los representados en la tabla 6.1:

m/n	Trabajos	
Máquinas	4	5
	3	3

Tabla 6.1: Tiempos de proceso 2x2 [Fuente: elaboración propia]

En la figura 6.4 podemos ver el diagrama de *Gant* para este caso:

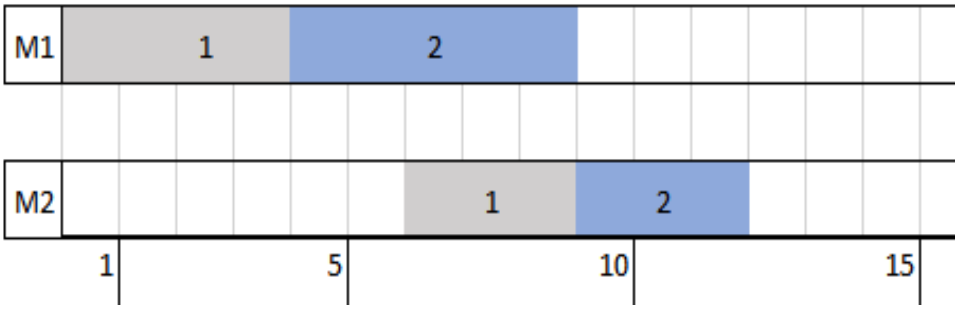

Figura 6.4: Diagrama de Gant 2x2 [Fuente: Elaboración propia]

Para el segundo caso, tres máquinas y tres trabajos, los tiempos de proceso han sido:

m/n	Trabajos		
Máquinas	4	5	6
	3	3	4
	2	1	3

Tabla 6.2: Tiempos de proceso 2x2 [Fuente: elaboración propia]

Y el diagrama de *Gant* para el caso 3x3 es el que aparece en la figura 6.5:

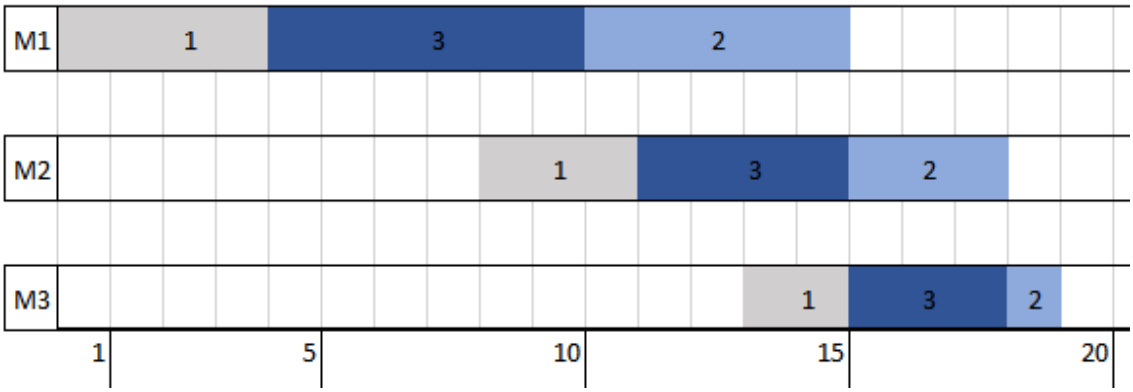

Figura 6.5: Diagrama de Gant 3x3 [Fuente: Elaboración propia]

Sin embargo, esta comprobación con estos pequeños tamaños de problema no es suficiente para asegurar que todos los modelos estén bien adaptados, ya que es al particularizar a instancias con tamaño de problema más grande donde pueden aparecer discordancias entre los resultados de cada modelo y debemos hacer las modificaciones pertinentes.

6.3.Batch y Microsoft Excel

Así pues, una vez particularizados los diferentes modelos en casos de tamaños de problema pequeños, toca trabajar con una gran cantidad de instancias en los que los tamaños de problema varían en gran medida, alcanzando tamaños de problema grandes. Resolver una a una cada instancia se antoja muy tedioso. Por ello utilizamos los archivos *batch*, archivos de procesamiento por lotes. Se trata de archivos de texto sin formato, guardados con la extensión ".bat" que contienen un conjunto de instrucciones MS-DOS (Sistema Operativo de disco de Microsoft). Cuando se ejecuta este archivo, las órdenes contenidas son ejecutadas en grupo, de forma secuencial, permitiendo automatizar diversas tareas. Cualquier orden reconocible por MS-DOS puede ser utilizado en un archivo *batch*

(https://es.wikipedia.org/wiki/Archivo_batch). De esta manera conseguimos realizar todas las llamadas a Gurobi de una vez para cada modelo para la resolución de esta batería de datos, ahorrándonos el proceso de hacerlo individualmente lo que resulta transcendental debido a la gran cantidad de instancias que hay para cada modelo.

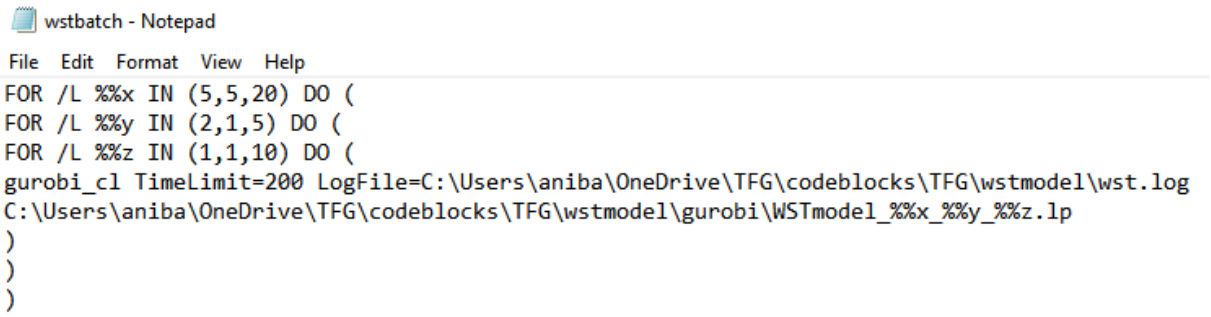

Figura 6.6: Ejemplo de archivo Batch [Fuente: elaboración propia]

En la figura tal mostramos un ejemplo del uso de los archivos *batch* en el que mediante el uso de bucles llamamos a gurobi para que lea y resuelva todos archivos LP del modelo WST con función objetivo *Cmax* particularizado a cada instancia. Estas soluciones se escriben todas en un archivo de texto con extensión ".log", donde nos interesan dos datos en particular: uno de ellos lógicamente la solución obtenida y el otro el tiempo que se ha tardado en encontrar dicha solución, también conocido como el *CPU time.*

Estos dos datos son trabajados en el programa *Microsoft Excel,* el último paso de nuestra implementación. Este es un software de Windows basado en hojas de cálculo que permite trabajar con múltiples aplicaciones, como funciones, herramientas gráficas o tablas de datos. Así pues, una vez obtenidos los resultados los trasladamos a *Excel* para poder trabajar con ellos y evaluarlos.

7. EVALUACIÓN COMPUTACIONAL

Llegados a este punto, procedemos a analizar los siete modelos para las dos funciones objetivo con la batería de datos para cada modelo ya mencionada. Esta batería consta de unos ficheros o instancias que van desde los cinco trabajos y dos máquinas hasta los 20 trabajos y cinco máquinas, avanzando de una en una máquina y de cinco en cinco trabajos, con 10 instancias para cada caso. Es decir, por cada tamaño de problema 10 casos diferentes y un total de 160 instancias. A su vez, imponemos un límite de tiempo para la resolución de cada instancia de 200 segundos. Esta evaluación de los resultados la basamos en tres criterios distintos: número de óptimos, *CPU time average* y el ARPD.

- Número de óptimos: El primero de los criterios se basa en contabilizar el número de óptimos a los que es capaz de llegar cada modelo antes de llegar al límite de tiempo de 200 segundos (óptimo), y en su defecto, en cuántas de las instancias de las que llegan al límite encuentran una solución factible (factible) y cuáles no (infactible).

- CPU times: El segundo criterio para analizar los resultados ha sido el de los CPU times, es decir, los tiempos de computación de cada modelo, lo que tarda el ordenador a través de Gurobi en resolverlos. Como ya dijimos con anterioridad, el límite de este tiempo para cada instancia se ha marcado en 200 segundos.

- ARPD: Como último criterio usamos el Average Relative Percentage Deviation, un indicador en porcentaje que nos informa qué alejada esta la función objetivo de cada instancia de la solución óptima. Para ello a las funciones objetivo de cada instancia en cada modelo le hemos restado el valor mínimo (óptimo del problema) de todos los modelos de la instancia pertinente, dividido entre este mismo valor mínimo.

$$ARPD_{ij} = \frac{FO_{ij} - FO\acute{o}pt_{ij}}{FO\acute{o}pt_{ij}} * 100$$

Dónde el $ARPD_{ij}$ es el Average Relative Percentage Deviation de la instancia *i* del modelo *j*.

7.1. Evaluación para la función Objetivo Makespan

7.1.1. *Análisis del número de óptimos:*

	WST			Wilson			TS2			SGST			LYeq			TBA			TS3		
	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT
5_2	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
5_3	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
5_4	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
5_5	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
10_2	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
10_3	10	0	0	10	0	0	10	0	0	8	2	0	8	2	0	10	0	0	10	0	0
10_4	10	0	0	10	0	0	10	0	0	6	4	0	5	5	0	10	0	0	10	0	0
10_5	10	0	0	10	0	0	10	0	0	3	7	0	5	5	0	10	0	0	10	0	0
15_2	10	0	0	10	0	0	10	0	0	0	10	0	0	10	0	10	0	0	10	0	0
15_3	10	0	0	10	0	0	10	0	0	0	10	0	0	10	0	10	0	0	10	0	0
15_4	10	0	0	10	0	0	10	0	0	0	9	1	0	10	0	10	0	0	10	0	0
15_5	10	0	0	10	0	0	10	0	0	0	8	2	0	10	0	10	0	0	10	0	0
20_2	10	0	0	10	0	0	10	0	0	0	10	0	0	10	0	10	0	0	10	0	0
20_3	10	0	0	10	0	0	10	0	0	0	6	4	0	9	1	10	0	0	10	0	0
20_4	10	0	0	10	0	0	10	0	0	0	8	2	0	10	0	10	0	0	10	0	0
20_5	10	0	0	10	0	0	10	0	0	0	3	7	0	10	0	10	0	0	10	0	0
∑	160	0	0	160	0	0	160	0	0	67	77	16	68	91	1	160	0	0	160	0	0
%	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	41.9	48.1	10.0	42.5	56.9	0.6	100	0.0	0.0	100	0.0	0.0

Tabla 7.1: Nº de Óptimos para Cmax [Fuente: Elaboración propia]

En la tabla 7.1 representamos el número de óptimos para la función objetivo *makespan*, contabilizando cuantos hay en total para cada modelo y representando a su vez que porcentaje hay de cada tipo de solución (óptimo, factible e infactible) dentro de cada modelo. En la figura 7.1 podemos ver estos resultados de una manera más gráfica.

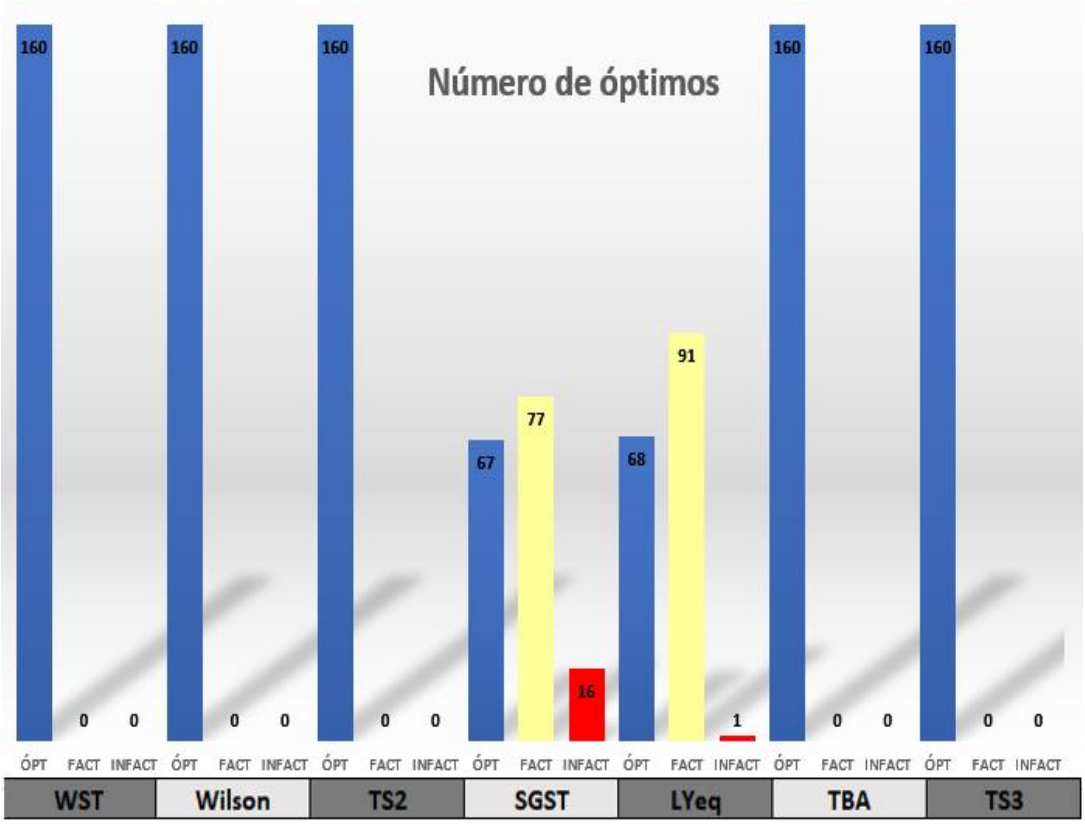

Figura 7.1: Número de óptimos para Cmax [Fuente: elaboración propia]

A la vista de los resultados, podemos ver como los modelos WST, Wilson, TS2, TBA y TS3, todos los de la familia Wagner, llegan al óptimo en todas las instancias para todos los casos. Por otra parte, los resultados para los modelos de la familia Manne, SGST y LYeq, son distintos. En ambos casos, a partir del tamaño de 10 trabajos y tres máquinas, la resolución de varias instancias llega al límite de tiempo, y es a partir del tamaño de 15 trabajos y dos máquinas cuando siempre se llega al límite de tiempo para todas las instancias en ambos casos. Sin embargo, podemos

encontrar diferencias entre estos dos, ya que para el modelo SGST hay hasta 16 instancias que no encuentra solución factible en este límite de tiempo, mientras que para el modelo LYeq sólo ocurre en una ocasión.

7.1.2. *Análisis de los CPU times*

Al igual que en el criterio anterior estos datos han sido agrupados por tamaño de problema con las 10 instancias para cada tamaño y el promedio en el que se resuelve una instancia por tamaño de problema se recogen en la tabla 7.2 que sigue a continuación:

	WST	Wilson	TS2	SGST	LYeq	TBA	TS3
5_2	0.03	0.04	0.02	0.17	0.15	0.03	0.03
5_3	0.18	0.07	0.04	0.26	0.22	0.04	0.04
5_4	0.08	0.10	0.06	0.27	0.29	0.07	0.05
5_5	0.07	0.12	0.08	0.28	0.30	0.11	0.13
10_2	0.01	0.02	0.02	45.26	26.65	0.02	0.02
10_3	0.08	0.20	0.08	80.57	66.55	0.12	0.14
10_4	0.09	0.19	0.12	137.90	143.79	0.12	0.14
10_5	0.33	0.49	0.49	173.02	169.23	0.57	0.78
15_2	0.02	0.02	0.02	200.01	200.02	0.03	0.04
15_3	0.06	0.13	0.09	200.02	200.02	0.15	0.11
15_4	0.25	0.94	0.52	200.01	200.02	1.23	0.83
15_5	1.54	2.09	1.84	200.02	200.03	3.18	3.46
20_2	0.03	0.09	0.02	200.02	200.02	0.05	0.04
20_3	0.19	0.46	0.31	200.02	200.02	0.59	0.31
20_4	1.78	2.20	2.01	200.02	200.03	2.91	1.85
20_5	1.13	1.27	1.06	200.04	200.03	3.38	1.45
Promed.	**0.37**	**0.53**	**0.42**	**127.37**	**125.46**	**0.79**	**0.59**

Tabla 7.2: CPU times para Cmax [Fuente: Elaboración propia]

Sorprende a la velocidad que se resuelven los modelos de la familia Wagner en comparación con los de la familia Manne. Dentro de la primera familia no hay

mucha diferencia entre sus modelos, siendo el WST el de mayor velocidad de computación por parte de Gurobi con una media de 0.37 segundos por instancia, y el TBA el más lento con 0.79 segundos de media. Por otro lado en los dos modelos de la familia Manne se superan los 100 segundos de media, volviendo a salir perdedor el modelo SGST con 127.37 segundos para poder resolver una sola instancia.

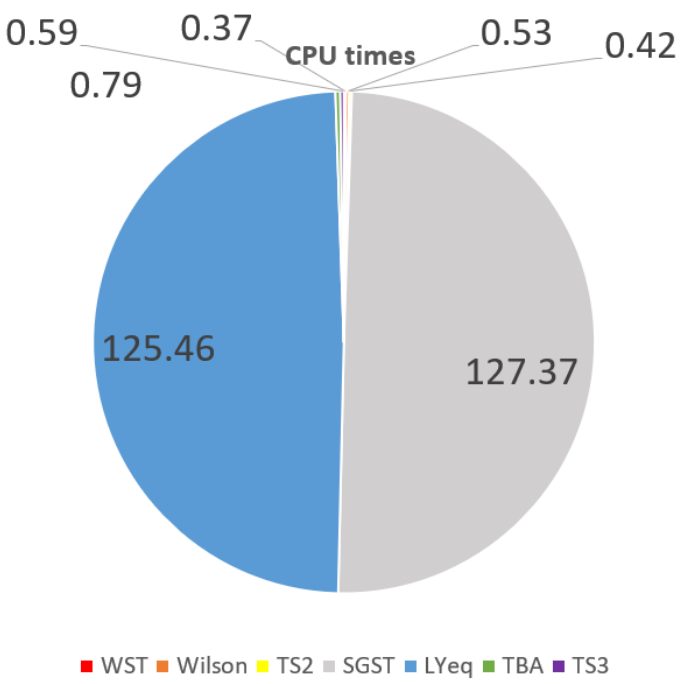

Figura 7.2: CPU times para Makespan [Fuente: elaboración propia]

En la figura 7.2 representamos de forma esquemática una comparativa de lo que tarda de media cada modelo en resolver una instancia y salta a la vista que los tiempos de la familia Wagner son insignificantes respecto a la familia Manne.

7.1.3. *Análisis del ARPD*

Los resultados se reflejan en promedio por tamaño de problema para cada instancia en la tabla 7.3:

	WST	Wilson	TS2	SGST	Lyeq	TBA	TS3
5_2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5_3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5_4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5_5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10_2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10_3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10_4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10_5	0.00	0.00	0.00	0.45	0.21	0.00	0.00
15_2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15_3	0.00	0.00	0.00	0.22	0.05	0.00	0.00
15_4	0.00	0.00	0.00	25.87	2.09	0.00	0.00
15_5	0.00	0.00	0.00	5.95	3.81	0.00	0.00
20_2	0.00	0.00	0.00	0.42	0.00	0.00	0.00
20_3	0.00	0.00	0.00	25.16	43.25	0.00	0.00
20_4	0.00	0.00	0.00	62.94	25.78	0.00	0.00
20_5	0.00	0.00	0.00	9.90	36.29	0.00	0.00
Promed. (%)	**0.00**	**0.00**	**0.00**	**8.18**	**6.97**	**0.00**	**0.00**

Tabla 7.3: ARPD para Cmax [Fuente: elaboración propia]

La figura 7.3 representa estos resultados de forma más gráfica, donde aparece el promedio del ARPD de cada modelo por instancia

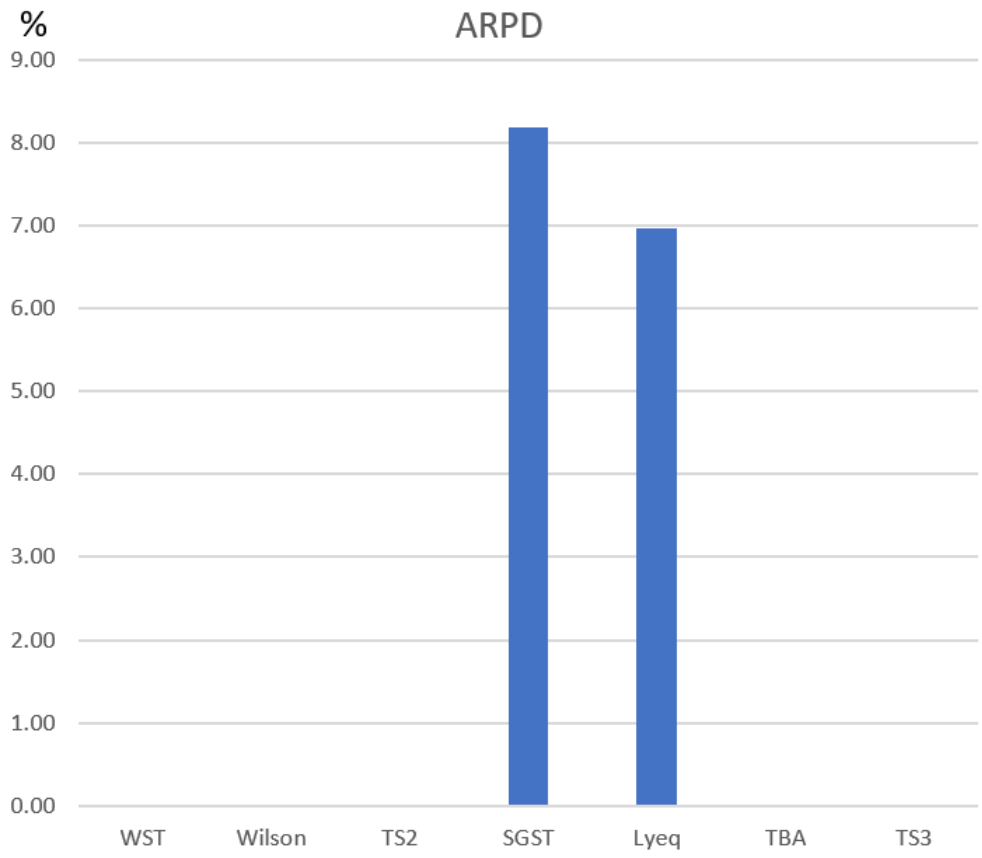

Figura 7.3: ARPD para Makespan [Fuente: elaboración propia]

A la vista de los resultados estas desviaciones sólo se producen, como cabría esperar, en los modelos SGST y LYeq, ya que son los únicos que han alcanzado el límite de tiempo en esta función objetivo. Es a partir del tamaño de problema de 10 trabajos y 5 máquinas cuando ambos modelos comienzan a proporcionar soluciones no óptimas. Dentro de éstos y en concordancia con los resultados en los criterios anteriores, vuelve a resultar perdedor el modelo SGST, con un promedio total de todos los tamaños de problema de 8.18% de desviación. Hemos de destacar los resultados del modelo SGST para el tamaño 20 trabajos y cuatro máquinas que, pese a no ser el tamaño de problema más grande, es donde encontramos la mayor desviación con un 62,94%.

7.2.Evaluación para la función objetivo sumCi

Procedemos a continuación al análisis y evaluación de los resultados obtenidos para la función objetivo $\sum C_i$ con los mismos tres criterios que en la función anterior.

7.2.1. *Análisis del número de óptimos*

	WST			Wilson			TS2			SGST			LYeq			TBA			TS3		
	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT	ÓPT	FACT	INFACT
5_2	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
5_3	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
5_4	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
5_5	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0
10_2	10	0	0	10	0	0	10	0	0	6	4	0	6	4	0	10	0	0	10	0	0
10_3	10	0	0	10	0	0	10	0	0	2	8	0	3	7	0	10	0	0	10	0	0
10_4	10	0	0	10	0	0	10	0	0	0	10	0	0	10	0	10	0	0	10	0	0
10_5	10	0	0	10	0	0	10	0	0	0	10	0	0	10	0	10	0	0	10	0	0
15_2	10	0	0	10	0	0	10	0	0	0	10	0	0	10	0	10	0	0	10	0	0
15_3	10	0	0	10	0	0	10	0	0	0	9	1	0	10	0	10	0	0	10	0	0
15_4	10	0	0	10	0	0	10	0	0	0	5	5	0	9	1	10	0	0	10	0	0
15_5	10	0	0	10	0	0	10	0	0	0	3	7	0	5	5	10	0	0	10	0	0
20_2	10	0	0	10	0	0	10	0	0	0	9	1	0	10	0	10	0	0	10	0	0
20_3	9	1	0	10	0	0	9	1	0	0	1	9	0	5	5	9	1	0	10	0	0
20_4	8	2	0	8	2	0	8	2	0	0	0	10	0	2	8	8	2	0	7	3	0
20_5	8	2	0	8	2	0	8	2	0	0	0	10	0	3	7	8	2	0	8	2	0
∑	155	5	0	156	4	0	155	5	0	48	69	43	49	85	26	155	5	0	155	5	0
%	96.9	3.1	0.0	97.5	2.5	0.0	96.9	3.1	0.0	30.0	43.1	26.9	30.6	53.1	16.3	96.9	3.1	0.0	96.9	3.1	0.0

Tabla 7.4: Número de óptimos para sumCi [Fuente: elaboración propia]

Al igual que en la función objetivo *makespan*, en la tabla 7.4 representamos el número de óptimos esta vez para la función objetivo *sumCi*, contabilizando cuantos hay en total para cada modelo y representando a su vez que porcentaje hay de cada tipo de solución (óptimo, factible e infactible) dentro de cada modelo. En la figura 7.4 podemos ver estos resultados de una manera más gráfica.

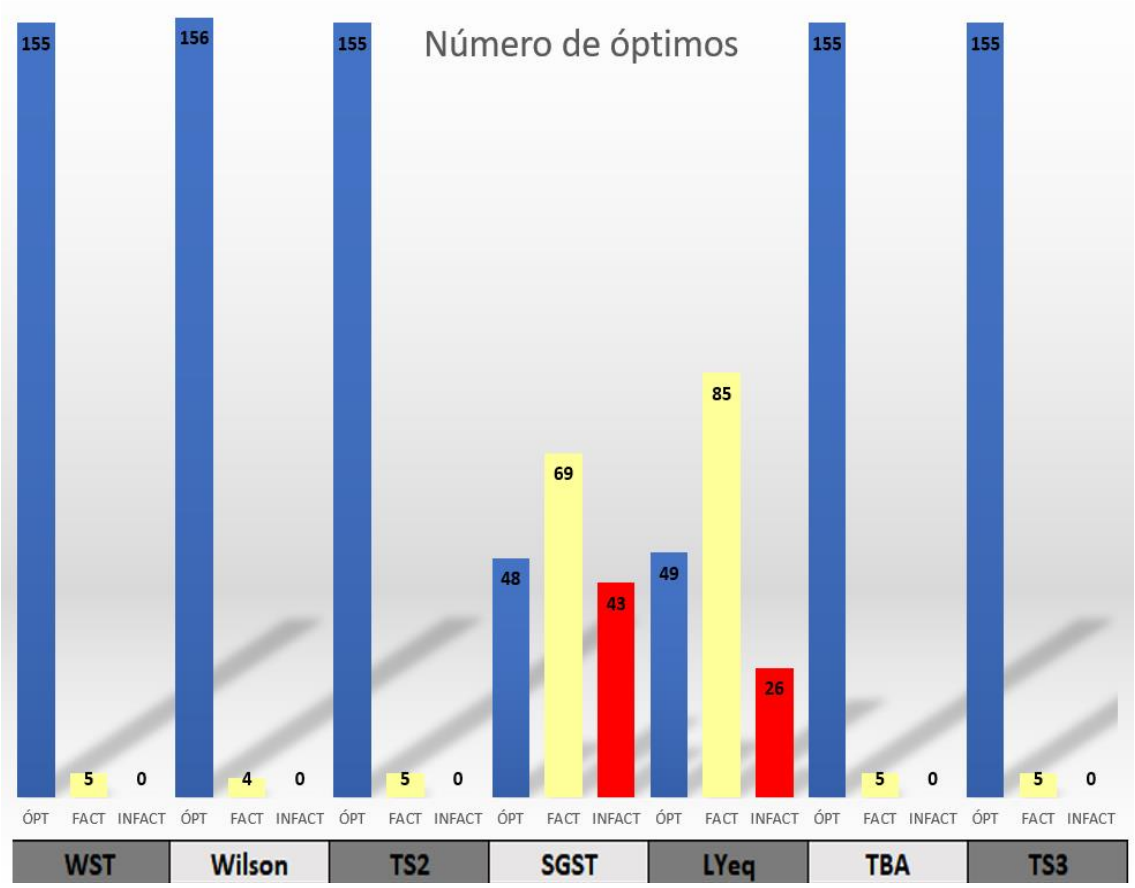

Figura 7.4: Número de óptimos para sumCi [Fuente: elaboración propia]

Los resultados nos proporcionan conclusiones muy similares a las de la función objetivo anterior. Es en los modelos de la familia Manne donde encontramos menor número de óptimos antes de llegar al límite de tiempo, y en los únicos que no se han podido encontrar soluciones factibles. El peor vuelve a resultar ser el SGST, con sólo 48 óptimos encontrados y hasta 43 instancias a las que no se han podido encontrar ninguna solución factible. Los resultados entre los modelos de la familia Wagner son muy parecidos, rondando los 155 óptimos encontrados y cinco

instancias a las que se ha alcanzado el límite de tiempo, pero sí se ha conseguido encontrar una solución factible.

7.2.2. *Análisis de los CPU times*

En la tabla 7.5 recogemos los tiempos que tardan en resolver cada modelo en promedio una instancia por tamaño de problema, con límite de tiempo como ya hemos mencionado de 200 segundos.

	WST	Wilson	TS2	SGST	LYeq	TBA	TS3
5_2	0.05	0.02	0.04	0.26	0.22	0.07	0.08
5_3	0.14	0.05	0.06	0.44	2.17	0.15	0.17
5_4	0.67	0.07	0.07	0.44	0.34	0.30	0.21
5_5	0.14	0.09	0.07	0.55	0.51	0.35	0.39
10_2	0.12	0.19	0.15	135.13	129.87	0.38	0.23
10_3	0.39	0.32	0.60	191.15	176.86	0.79	0.31
10_4	0.35	0.28	0.28	200.02	200.02	0.77	0.38
10_5	0.81	0.90	0.80	200.02	200.03	1.98	0.98
15_2	0.40	0.70	0.49	200.02	200.02	0.99	0.36
15_3	2.84	1.76	1.90	200.03	200.02	4.58	2.96
15_4	8.57	6.07	5.47	200.03	200.03	13.91	6.85
15_5	10.61	6.82	6.69	200.04	200.02	19.50	9.73
20_2	1.77	1.41	1.30	200.03	200.02	2.80	1.62
20_3	33.94	47.16	54.27	200.03	200.02	49.87	31.04
20_4	121.68	120.46	107.29	200.02	200.02	137.08	116.42
20_5	125.49	84.81	79.81	200.02	200.04	110.91	119.56
Promed.	**19.25**	**16.94**	**16.20**	**145.51**	**144.39**	**21.53**	**18.20**

Tabla 7.5: CPU times para sumCi [Fuente: elaboración propia]

La familia Manne vuelve a proporcionar resultados peores que los modelos de la familia Wagner y, otra vez, con mucha diferencia. Dentro de la familia Wagner el modelo que más rápido resuelve cada caso es el Wilson, con un promedio por instancia de 16.94 segundos. El peor parado dentro de la familia Manne y en el problema en general vuelve a ser el SGST, con media por instancia de 145.51 segundos.

Los mismos resultados los representamos en la figura 7.5 de manera más esquemática, en los que al igual que en la función objetivo anterior sigue existiendo una gran diferencia entre la familia Manne y Wagner, aunque los tiempos de resolución en esta última ya no son tan despreciables como en los de la función anterior, aumentando los tiempos en general de una función a otra como veremos en el apartado 7.3.

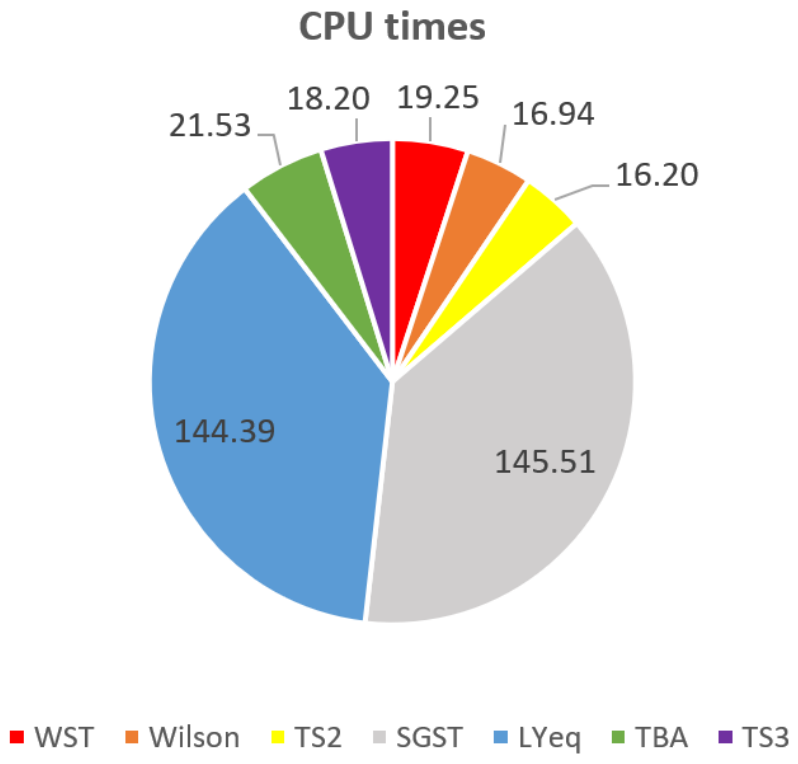

Figura 7.5: CPU times para sumCi [Fuente: elaboración propia]

7.2.3. *Análisis del ARDP*

	WST	Wilson	TS2	SGST	Lyeq	TBA	TS3
5_2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5_3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5_4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5_5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10_2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10_3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10_4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10_5	0.00	0.00	0.00	5.50	3.79	0.00	0.00
15_2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15_3	0.00	0.00	0.00	14.88	285.25	0.00	0.00
15_4	0.00	0.00	0.00	25.82	18.31	0.00	0.00
15_5	0.00	0.00	0.00	104.86	22.17	0.00	0.00
20_2	0.00	0.00	0.00	13.42	499.76	0.00	0.00
20_3	0.00	0.00	0.00	24.58	30.65	0.00	0.00
20_4	0.15	0.01	0.05	-	18.63	0.02	0.13
20_5	0.32	0.44	0.20	-	29.94	0.20	0.14
Promed. (%)	**0.03**	**0.03**	**0.02**	**13.50**	**56.78**	**0.01**	**0.02**

Tabla 7.6: ARPD para sumCi [Fuente: elaboración propia]

Las desviaciones producidas en los modelos de la familia Wagner son prácticamente despreciables, siendo las más altas dentro de estos la de los modelos WST y Wilson con 0.03% de media por instancia. En los modelos de la familia Manne si se producen grandes desviaciones de media con 13.5% para el modelo SGST y 56.78% para el LYeq. Estos resultados son algo confusos y no muy representativos de la realidad del problema, ya que para el modelo SGST en los dos últimos tamaños de problema no se ha encontrado ninguna solución factible, que

de haberse encontrado seguro que distaría en gran medida del óptimo del problema, por lo que la media sería bastante más alta que ese 13,5%.

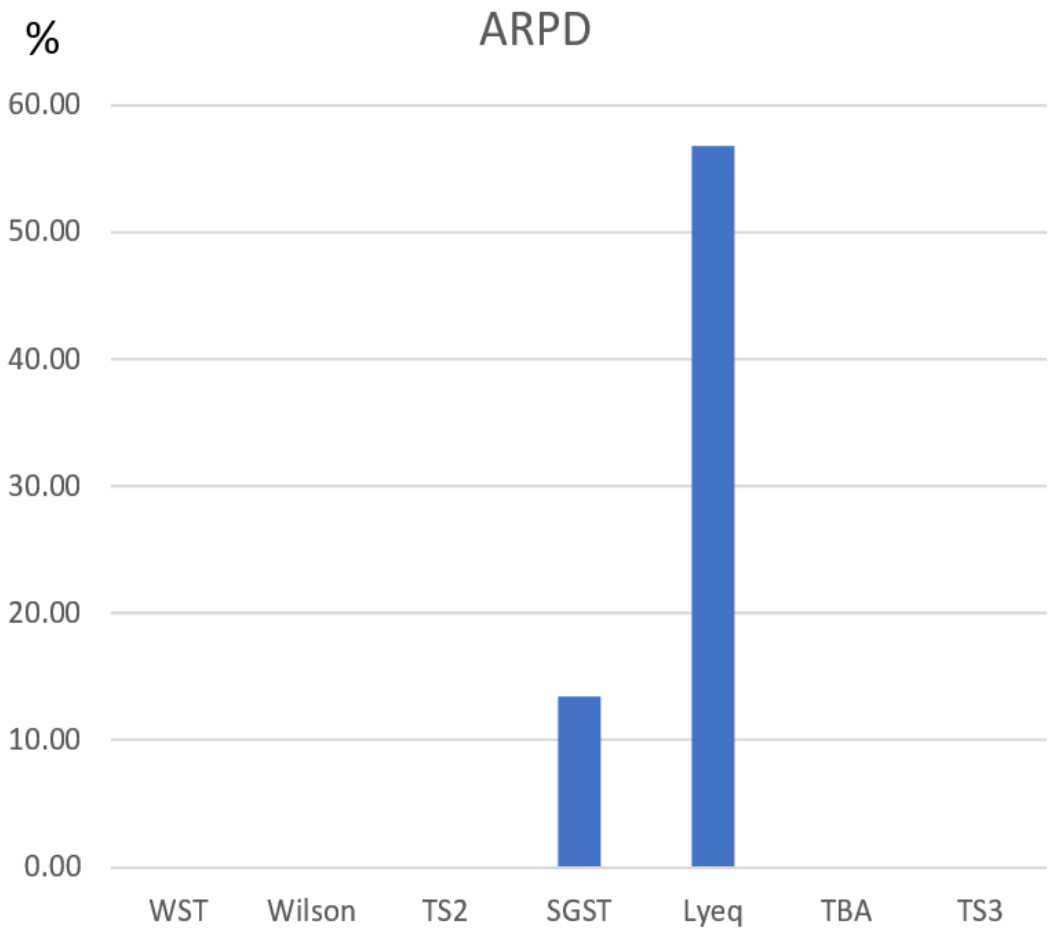

Figura 7.6: ARPD para sumCi [Fuente: elaboración propia]

7.3.Comparación entre las dos funciones objetivo

Hasta ahora hemos realizado la evaluación y análisis de los modelos según tres criterios distintos y para las dos funciones objetivo de manera separada, es decir, dentro de cada función objetivo hemos visto como variaban los resultados según el modelo y los hemos comparado entre estos mismos modelos. A

continuación, estudiamos como varía nuestro problema de una función objetivo a otra, comparando los resultados entre estas dos.

7.3.1. *Numero de óptimos*

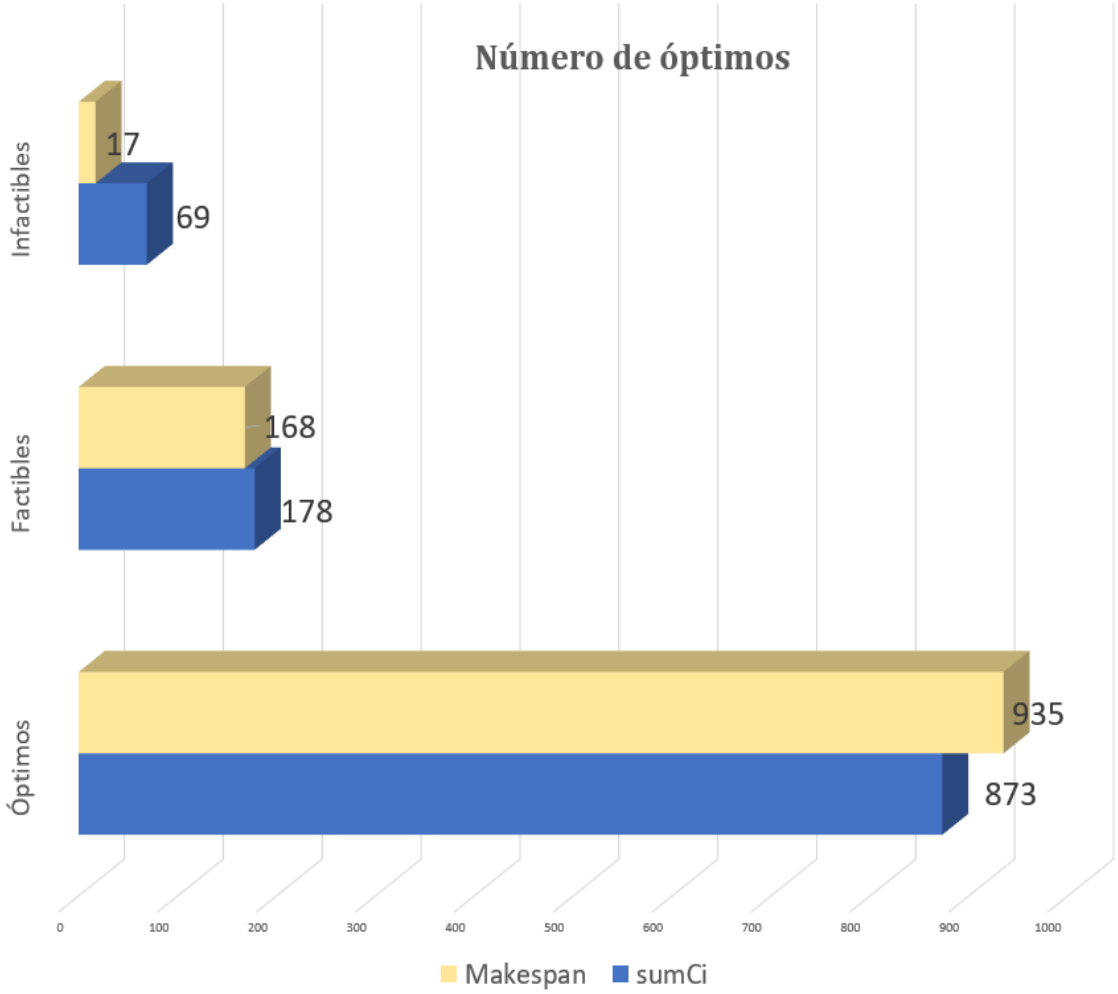

Figura 7.7: Comparación del número de óptimos para las dos funciones objetivo [Fuente: elaboración propia]

Como podemos ver en la figura 7.7, el número de óptimos encontrados en total para la función objetivo *makespan* es mayor que para la función objetivo *sumCi*, con una diferencia de 62 óptimos más encontrados para el problema del *makespan*. Así pues, el número de soluciones factibles e infactibles es mayor para *sumCi*, siendo en las soluciones infactibles donde se encuentra la mayor diferencia, con hasta 69

soluciones infactibles en *sumCi* por las 17 de la función *makespan*. Por lo que se demuestra que cuesta más encontrar soluciones óptimas para la función *sumCi* que para la función *makespan*.

7.3.2. *CPU times*

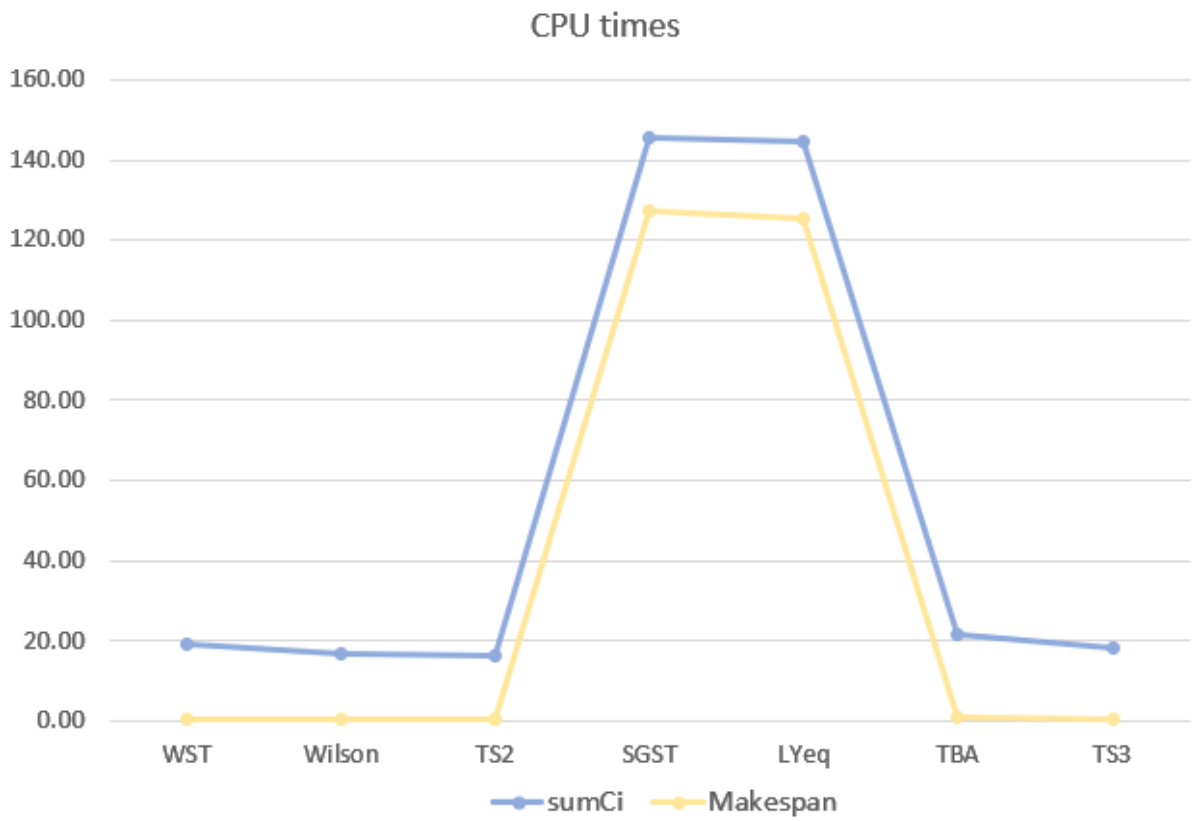

Figura 7.8: Comparación de los CPU times para las dos funciones objetivo
[Fuente: elaboración propia]

En cuanto a los tiempos de CPU, estos aumentan de la función *makespan* a la función *sumCi* en 20 segundos aproximadamente por cada instancia en todos los modelos, por lo que se tarda más en encontrar una solución (ya sea óptima, factible o infactible) para la función objetivo *sumCi* que en la función *makespan*.

7.3.3. *ARPD*

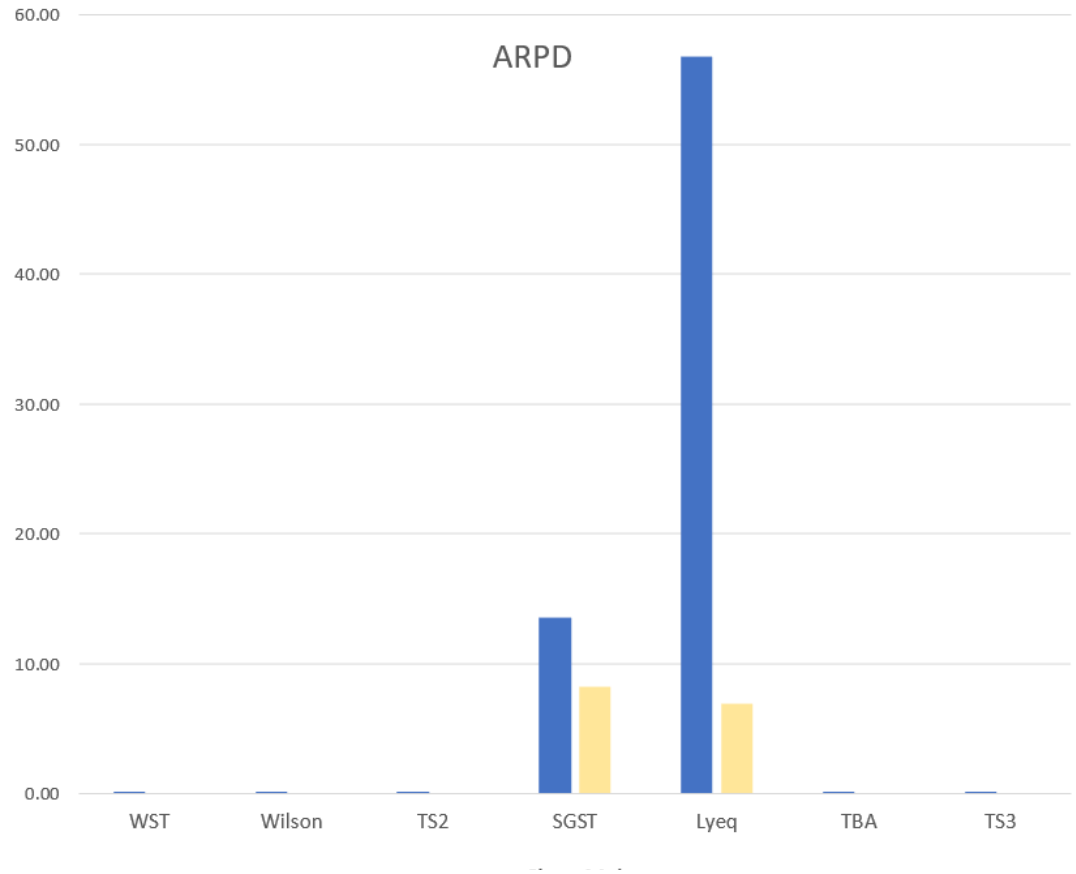

Figura 7.9: Comparación del ARPD para las dos funciones objetivo [Fuente: elaboración propia]

Las diferencias de las desviaciones entre ambas funciones objetivo presentan resultados dispares. Para los modelos de la familia Wagner la diferencia es mínima, puesto que no existen desviaciones para el caso del *makespan* y las desviaciones para *sumCi* son prácticamente inexistentes. Sin embargo, para los modelos de la familia Manne las diferencias son palpables. Para el caso del modelo LYeq, existe una diferencia de más del 40% de desviación por instancia, y para el caso del modelo SGST una diferencia de más del 6%, a pesar de la particularidad de este caso comentada en el apartado 7.2.3.

Con la evaluación de todos los resultados ya elaborada, estamos en condiciones de sacar unas conclusiones para nuestro problema, que serán expuestas en el capítulo siguiente.

8. CONCLUSIONES

En el presente proyecto hemos realizado el estudio de siete modelos distintos ya existentes de programación de la producción para el taller de flujo regular con permutación y su correspondiente adaptación a la restricción *no-idle*, para el posterior análisis de resultados evaluando para cada modelo un total de 160 casos diferentes en los que varían el número de máquinas, de trabajos y los tiempos de proceso. Esta evaluación se ha hecho para dos funciones objetivo diferentes siendo estas el tiempo máximo de finalización o *makespan*, y el tiempo total de finalización, $\sum C_i$, muy habituales en la programación de la producción, lo que implica que la evaluación se ha realizado para un total de 2240 casos distintos. Esta evaluación se ha basado en tres criterios distintos los cuales son el número de óptimos encontrado para cada modelo, el tiempo de resolución de la computadora y el ARPD. Según estos podemos medir de forma cuantificable cuales de los modelos son más eficientes sobre otros para ambas funciones objetivo y qué función objetivo puede implicar en un problema más sencillo o complejo.

Los resultados obtenidos son meridianamente claros. Los modelos de la familia Wagner, tanto los tres primeros como los adicionales son más eficientes que los dos de la familia Manne para ambas funciones objetivo. El número de óptimos encontrado es bastante superior en favor de la primera familia, los CPU times presentan una gran diferencia, siendo mucho más discretos los de la familia Wagner y las desviaciones medidas como ARPD son bastante inferiores en esta misma. La principal diferencia entre estas dos familias es la forma en la que trata los trabajos. Mientras que en la familia Wagner se usa la asignación de trabajos a unas determinadas posiciones en una secuencia de procesamiento, en la familia Manne no se usa esta asignación de los trabajos a las posiciones, sino que se trabaja directamente con el tiempo de finalización de procesado de los trabajos (C_i). Por lo que en la familia Manne al no usar esta asignación, la adaptación a la restricción *no-idle* resulta mucho más complicada, debiendo añadir a los dos modelos

originales de esta familia un gran número de variables y restricciones adicionales, lo que complica la resolución de los mismos para tamaños de problema elevados.

Los resultados dentro de la familia Wagner son muy similares. Para la función objetivo *Cmax* todos los modelos de esta familia encuentran el óptimo en las 160 instancias, siendo el modelo WST el más rápido con 0.37 segundos de media en resolver una instancia. Para la función objetivo *sumCi,* los resultados son también muy similares. El modelo Wilson con 156 óptimos se impone, en este aspecto, sobre los modelos de su familia, los cuáles encuentran sólo un óptimo menos, 155. No obstante, el modelo TS2 es el más rápido con 16.20 segundos de media en resolver una instancia y como ocurre con el resto de criterios, otra vez con muy poca diferencia con el resto de modelos de su familia. En cuanto a la familia Manne, el modelo LYeq se impone sobre el SGST, ya que para ambas funciones objetivo encuentra mayor número de óptimos y soluciones factibles y el tiempo de ejecución del ordenador es menor.

Finalmente comparamos los resultados de manera global entre las dos funciones objetivo, de lo que se deduce que el problema para la función *sumCi* es más complicado que para *Cmax.* Esto se refleja en los tres criterios de la evaluación, ya que para el *makespan* el número de óptimos y soluciones factibles es mayor, los *CPU times* son menores y presentan menos desviaciones para todos los modelos. Futuras líneas de investigación deberían tanto confirmar este hecho mediante la enumeración completa de todas las soluciones en ambos problemas, como justificar estadísticamente las conclusiones obtenidas, mediante la aplicación de contrastes de hipótesis.

Así pues, gracias a este estudio se pueden plantear adicionales líneas de trabajo para este problema en el que las máquinas no tienen tiempos de inactividad, problema que se da con bastante frecuencia en el mundo industrial y de la programación de operaciones. Los estudios existentes sobre este problema en concreto son escasos y este proyecto puede ayudar a modelarlos y encontrar la solución más eficiente para el problema en la realidad. De esta forma, los *MILPs*

elaborados podrán servir como punto de partida para futuras matheuristics o técnicas avanzadas de optimización.

9. BIBLIOGRAFÍA

L. Onieva, P. Cortés, J. Muñuzuri, J. Guadix, J. N. Ibáñez *"Métodos Cuantitativos y Organización de la Producción"* Ed SINTESIS, 2006.

Michael L. Pinedo. "*Scheduling-Theory, Algorithms, and Systems",* Fourth Ed. Springer, 2012.

Fernández-Viagas, Pérez, P.,, V., Framinan, J. M. "*Apuntes de la asignatura Programación de Operaciones",* 2018.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey.", 1979.

Jose M. Framinan, Rainer Leisten, Rubén Ruiz Garcia. *"Manufacturing Scheduling Systems. An Integrated View on Models, Methods and Tools"*, 2014.

Larrosa, Oliveras, Rodríguez-Carbonell. *"Mixed Integer Linear Programming"*, 2019.

Stafford Jr, E. F., Tseng, F. T., Gupta, J. N. D. *"Comparative evaluation of MILP flowshop models".* Journal of the Operational Research Society, 56:88-101*,* 2005.

Tseng, F. T., Stafford Jr., E. F. *"New MILP models for the permutation flowshop problem".* Journal of the Operational Research Society, 59:1373-1386, 2008.

Tseng F. T., Stafford E. F*. "Two MILP models for the N_M SDST flowshop sequencing problem".* Int J Prod Res 39: 1777–1809, 2001.

Wagner HM*, "An integer linear-programming model for machine scheduling."* Nav Res Log Q 6: 131–140, 1959.

https://www.ucm.es/pimcd2014-free-software /codeblocks

http://www.codeblocks.org/

https://www.gurobi.com/documentation/

https://es.wikipedia.org/wiki/Archivo_batch

10. ANEXO

En este anexo incluimos el código de programación en C cada modelo con la función objetivo *makespan*, incluyendo la función objetivo *sumCi* al final de cada modelo puesto que el resto del código es igual.

10.1. Código familia Wagner

10.1.1. *WST model*

```
#include <stdio.h>
#include <stdlib.h>
#include <schedule_lib.h>
void WSTmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero);
int main()
{
	printf("Modelo WST \n");
	int m, n, k, j, i;
	int  cont = 0;
	char  cadenacar[50] = { 0 };
	char  cadenacar2[50] = { 0 };
	for (j = 5; j <= 20;)

	{
		for (k = 2; k <= 5; k++)

		{
			for (i = 1; i <= 10; i++)
			{
				sprintf(cadenacar, "Bateria Sin fechas/inst_%d_%d_p_%d", j, k, i);
				FILE* primerfichero = fopen(cadenacar, "r");
				fscanf(primerfichero, "%d %d", &m, &n);
				MAT_INT Tpro = DIM_MAT_INT(m, n);
				Tpro = loadPTimes_nrows(cadenacar, &n, &m, YES);
				sprintf(cadenacar2, "gurobi/WSTmodel_%d_%d_%d.lp", j, k, i);
				FILE* secunfichero = fopen(cadenacar2, "w");
				WSTmodel(n, m, Tpro, secunfichero);

				fclose(secunfichero);
				fclose(secunfichero);
				fflush(stdin);
				cont = cont + 1;
```

```
                        }
                }
                j = j + 5;
        }
        return cont;
}
void WSTmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero)
{
        int i, j, r, p;
        //FO Cmax
        fprintf(secunfichero, "Minimize \n");
        for (i = 1; i <= n; i++)
        {
                if (i < n)
                {
                        fprintf(secunfichero, "T_%d_%d + ", m, i);
                }
                else
                {
                        fprintf(secunfichero, "T_%d_%d + ", m, i);
                }
        }

        fprintf(secunfichero, "X_%d \n ", m);

        //RESTRICCIONES DEL MODELO
        fprintf(secunfichero, "Subject To \n");
        // Primera restricción
        for (i = 1; i <= n; i++)
        {
                for (j = 1; j <= n; j++)
                {

                        if (j < n)
                        {
                                fprintf(secunfichero, "Z_%d_%d + ", i, j);
                        }
                        else
                        {
                                fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
                        }
                }
        }
        // Segunda restricción
        for (j = 1; j <= n; j++)
        {
                for (i = 1; i <= n; i++)
                {
                        if (i < n)
                        {
```

```
                        fprintf(secunfichero, "Z_%d_%d + ", i, j);
                }
                else
                {
                        fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
                }
        }
}
// Tercera restricción
for (r = 1; r <= m - 1; r++)
{
        for (j = 1; j <= n - 1; j++)
        {
                for (i = 1; i <= n; i++)
                {
                fprintf(secunfichero, " %d Z_%d_%d  + ", Tpro[r - 1][i - 1], i, j + 1);
                }
                fprintf(secunfichero, "Y_%d_%d - ", r, j + 1);
                for (i = 1; i <= n; i++)
                {
                        fprintf(secunfichero, " %d Z_%d_%d - ", Tpro[r][i - 1], i, j);
                }

                fprintf(secunfichero, "Y_%d_%d = 0 \n", r, j);
        }
}
//Cuarta restricción
for (r = 1; r <= m - 1; r++)
{
        fprintf(secunfichero, "X_%d - ", r + 1);
        fprintf(secunfichero, "X_%d - ", r);
        fprintf(secunfichero, "Y_%d_1 - ", r);

        for (i = 1; i <= n; i++)
        {
                if (i < n)
                {
                        fprintf(secunfichero, " %d Z_%d_1  - ", Tpro[r - 1][i - 1], i);
                }
                else
                {
                        fprintf(secunfichero, " %d Z_%d_1  = 0 \n", Tpro[r - 1][i - 1], i);
                }
        }
}

//Tiempo de proceso
for (r = m; r <= m; r++)
{
        for (i = 1; i <= n; i++)
```

```
                {
                        fprintf(secunfichero, "T_%d_%d = %d \n", r, i, Tpro[r - 1][i - 1]);
                }
        }


        //VARIABLES
        fprintf(secunfichero, "Generals \n");
        for (i = 1; i <= m - 1; i++)
        {
                for (j = 1; j <= n; j++)
                {
                        fprintf(secunfichero, "Y_%d_%d \n", i, j);
                }
        }
        fprintf(secunfichero, "Binary \n");
        for (i = 1; i <= n; i++)
        {

                for (j = 1; j <= n; j++)
                {
                        fprintf(secunfichero, "Z_%d_%d \n", i, j);
                }
        }
        //FIN
        fprintf(secunfichero, "End");
}
```

10.1.2. *WST model FO sumCi*

```
//FO sumCi
fprintf(secunfichero, "Minimize \n");
for (j = 1; j <= n; j++)
{
        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "[ %d T_%d_%d * Z_%d_%d ] + ", n - j + 1, m, i, i, j);
        }

}

fprintf(secunfichero, "%d X_%d\n ", n, m);
```

10.1.3. *Wilson model*

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <schedule_lib.h>
void Wilsonmodel(int n, int m, MAT_INT Tpro, FILE* fichero2);
int main()
{
	printf("The Wilson model\n");
	int m, n, k, j, i;
	int  cont = 0;

	char  cadenacar[50] = { 0 };

	char  cadenacar2[50] = { 0 };

	for (j = 5; j <= 20;)

	{
		for (k = 2; k <= 5; k++)

		{
			for (i = 1; i <= 10; i++)
			{
				sprintf(cadenacar, "Bateria Sin fechas/inst_%d_%d_p_%d", j, k, i);
				FILE* primerfichero = fopen(cadenacar, "r");
				fscanf(primerfichero, "%d %d", &m, &n);

				printf("Numero de maquinas es: %d \n", m);
				printf("Numero de trabajos es: %d \n", n);

			sprintf(cadenacar2, "gurobiwilson/Wilsonmodel_%d_%d_%d.lp", j, k, i);
				FILE* secunfichero = fopen(cadenacar2, "w");
				MAT_INT Tpro = DIM_MAT_INT(m, n);
				Tpro = loadPTimes_nrows(cadenacar, &n, &m, YES);
				Wilsonmodel(n, m, Tpro, secunfichero);

				fclose(secunfichero);
				fclose(secunfichero);
				fflush(stdin);
				cont = cont + 1;
			}
		}
		j = j + 5;
	}
	return cont;
}
void Wilsonmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero)
{
	int i, j, r;
	//FO Cmax
	fprintf(secunfichero, "Minimize \n");
	fprintf(secunfichero, "B_%d_%d + ", m, n);
	for (i = 1; i <= n; i++)
```

```
{
        if (i < n)
        {
                fprintf(secunfichero, " %d  Z_%d_%d + ", Tpro[m - 1][i - 1], i, n);
        }
        else
        {
                fprintf(secunfichero, " %d  Z_%d_%d  \n", Tpro[m - 1][i - 1], i, n);
        }
}
//RESTRICCIONES DEL MODELO
fprintf(secunfichero, "Subject To \n");
//Primera restricción
for (i = 1; i <= n; i++)
{
        for (j = 1; j <= n; j++)
        {
                if (j < n)
                {
                        fprintf(secunfichero, "Z_%d_%d + ", i, j);
                }
                else
                {
                        fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
                }
        }
}
//Segunda restricción

for (j = 1; j <= n; j++)
{
        for (i = 1; i <= n; i++)
        {
                if (i < n)
                {
                        fprintf(secunfichero, "Z_%d_%d + ", i, j);
                }
                else
                {
                        fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
                }
        }
}
//Tercera restricción

for (j = 1; j <= n - 1; j++)
{
        fprintf(secunfichero, "B_1_%d + ", j);
        for (i = 1; i <= n; i++)
        {
```

```
                if (i < n)
                {
                        fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[0][i - 1], i, j);
                }
                else
                {
                        fprintf(secunfichero, " %d Z_%d_%d - ", Tpro[0][i - 1], i, j);
                }
        }
        fprintf(secunfichero, "B_1_%d = 0 \n", j + 1);
}
//Cuarta restricción

fprintf(secunfichero, "B_1_1 = 0 \n");
//Quinta restricción

for (r = 1; r <= m - 1; r++)
{
        fprintf(secunfichero, "B_%d_1 + ", r);
        for (i = 1; i <= n; i++)
        {
                if (i < n)
                {
                        fprintf(secunfichero, " %d Z_%d_1 + ", Tpro[r - 1][i - 1], i);
                }
                else
                {
                        fprintf(secunfichero, " %d Z_%d_1 - ", Tpro[r - 1][i - 1], i);
                }
        }
        fprintf(secunfichero, "B_%d_1 <= 0 \n", r + 1);
}
//Sexta restricción

for (r = 1; r <= m - 1; r++)
{
        for (j = 2; j <= n; j++)
        {
                fprintf(secunfichero, "B_%d_%d + ", r, j);
                for (i = 1; i <= n; i++)
                {
                        if (i < n)
                        {
                        fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[r - 1][i - 1], i, j);
                        }
                        else
                        {
                        fprintf(secunfichero, " %d Z_%d_%d - ", Tpro[r - 1][i - 1], i, j);
                        }
                }
```

```
                        fprintf(secunfichero, "B_%d_%d <= 0 \n", r + 1, j);
                }
        }
        //Séptima restricción

        for (r = 2; r <= m; r++)
        {
                for (j = 1; j <= n - 1; j++)
                {
                        fprintf(secunfichero, "B_%d_%d + ", r, j);
                        for (i = 1; i <= n; i++)
                        {
                                if (i < n)
                                {
                fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[r - 1][i - 1], i, j);
                                }
                                else
                                {
                                fprintf(secunfichero, " %d Z_%d_%d - ", Tpro[r - 1][i - 1], i, j);
                                }
                        }
                        fprintf(secunfichero, "B_%d_%d = 0 \n", r, j + 1);
                }
        }

        //Variables
        fprintf(secunfichero, "Integers \n");
        for (r = 1; r <= m; r++)
        {
                for (j = 1; j <= n; j++)
                {
                        fprintf(secunfichero, "B_%d_%d \n", r, j);
                }
        }
        fprintf(secunfichero, "Binary \n");
        for (i = 1; i <= n; i++)
        {
                for (j = 1; j <= n; j++)
                {
                        fprintf(secunfichero, "Z_%d_%d \n", i, j);
                }
        }
        fprintf(secunfichero, "End");
}
```

10.1.4. *Wilson model FO sumCi*

```
//FO sum Ci
fprintf(secunfichero, "Minimize \n");
```

```
for (j = 1; j <= n; j++)
{
        fprintf(secunfichero, "B_%d_%d + ", m, j);
        for (i = 1; i <= n; i++)
        {
                if (j < n)
                {


                        fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[m - 1][i - 1], i, j);
                }

                else
                {
                        if (i < n)
                        {
                                fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[m - 1][i - 1], i, j);
                        }
                        else
                        {
                                fprintf(secunfichero, " %d Z_%d_%d \n", Tpro[m - 1][i - 1], i, j);
                        }

                }


        }

}
```

10.1.5. *TS2 model*

```
#include <stdio.h>
#include <stdlib.h>
#include <schedule_lib.h>
void TS2model(int n, int m, MAT_INT Tpro, FILE* fichero2);
int main()
{
        printf("The TS2 model\n");
        int m, n, k, j, i;
        int  cont = 0;

        char  cadenacar[50] = { 0 };

        char  cadenacar2[50] = { 0 };
```

```
	for (j = 5; j <= 20;)

	{
		for (k = 2; k <= 5; k++)

		{
			for (i = 1; i <= 10; i++)
			{
				sprintf(cadenacar, "Bateria Sin fechas/inst_%d_%d_p_%d", j, k, i);
				FILE* primerfichero = fopen(cadenacar, "r");
				fscanf(primerfichero, "%d %d", &m, &n);
				MAT_INT Tpro = DIM_MAT_INT(m, n);
				Tpro = loadPTimes_nrows(cadenacar, &n, &m, YES);
				sprintf(cadenacar2, "gurobiTS2/TS2model_%d_%d_%d.lp", j, k, i);
				FILE* secunfichero = fopen(cadenacar2, "w");

				TS2model(n, m, Tpro, secunfichero);

				fclose(secunfichero);
				fclose(secunfichero);
				fflush(stdin);
				cont = cont + 1;
			}
		}
		j = j + 5;
	}
	return cont;
}
void TS2model(int n, int m, MAT_INT Tpro, FILE* secunfichero)
{
	int i, j, r;
	//FUNCIÓN OBJETIVO
	//FO Cmax
	fprintf(secunfichero, "Minimize \n");
	fprintf(secunfichero, "E_%d_%d \n", m, n);
	//RESTRICCIONES
	//Primera restricción
	fprintf(secunfichero, "Subject To \n");
	//Primera restricción
	for (i = 1; i <= n; i++)
	{
		for (j = 1; j <= n; j++)
		{
			if (j < n)
			{
				fprintf(secunfichero, "Z_%d_%d + ", i, j);
			}
			else
			{
				fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
```

```
                }
        }
}
//Segunda restricción

for (j = 1; j <= n; j++)
{
        for (i = 1; i <= n; i++)
        {
                if (i < n)
                {
                        fprintf(secunfichero, "Z_%d_%d + ", i, j);
                }
                else
                {
                        fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
                }
        }
}
//Tercera restricción

for (r = 1; r <= m; r++)
{
        for (j = 1; j <= n - 1; j++)
        {
                fprintf(secunfichero, "E_%d_%d + ", r, j);
                for (i = 1; i <= n; i++)
                {
                        if (i < n)
                        {
                fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[r - 1][i - 1], i, j + 1);
                        }
                        else
                        {
                fprintf(secunfichero, " %d Z_%d_%d ", Tpro[r - 1][i - 1], i, j + 1);
                        }
                }
                fprintf(secunfichero, "- E_%d_%d = 0 \n", r, j + 1);
        }
}

//Cuarta restricción

for (r = 1; r <= m - 1; r++)
{
        for (j = 1; j <= n; j++)
        {
                fprintf(secunfichero, "E_%d_%d + ", r, j);
                for (i = 1; i <= n; i++)
                {
```

```
                        if (i < n)
                        {
                        fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[r][i - 1], i, j);
                        }
                        else
                        {
                                fprintf(secunfichero, " %d Z_%d_%d ", Tpro[r][i - 1], i, j);
                        }
                }
                fprintf(secunfichero, "- E_%d_%d <= 0 \n", r + 1, j);
        }
}

//Quinta restricción

fprintf(secunfichero, "E_1_1 ");
for (i = 1; i <= n; i++)
{
        if (i < n)
        {
                fprintf(secunfichero, "- %d Z_%d_1 ", Tpro[0][i - 1], i);
        }
        else
        {
                fprintf(secunfichero, "- %d Z_%d_1 = 0 \n", Tpro[0][i - 1], i);
        }
}
//Tiempos de proceso
for (r = 1; r <= m; r++)
{
        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "T_%d_%d = %d \n", r, i, Tpro[r - 1][i - 1]);
        }
}

//Variables
fprintf(secunfichero, "Generals \n");
for (r = 1; r <= m; r++)
{
        for (j = 1; j <= n; j++)
        {
                fprintf(secunfichero, "E_%d_%d \n", r, j);
        }
}
fprintf(secunfichero, "Binary \n");
for (i = 1; i <= n; i++)
{
        for (j = 1; j <= n; j++)
        {
```

```
                        fprintf(secunfichero, "Z_%d_%d \n", i, j);
                }
        }

        fprintf(secunfichero, "End");
}
```

10.1.6. *TS2 model FO sumCi*

```
//FO sumCj
fprintf(secunfichero, "Minimize \n");
for (j = 1; j <= n; j++)
{
        if (j < n)
        {
                fprintf(secunfichero, "E_%d_%d + ", m, j);
        }
        else
        {
                fprintf(secunfichero, "E_%d_%d\n", m, j);
        }
}
```

10.2. Código familia Manne

10.2.1. *SGST model*

```
#include <stdio.h>
#include <stdlib.h>
#include <schedule_lib.h>
void SGSTmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero);
int main()
{
        printf("Modelo SGST\n");
        int m, n, k, j, i;
        int  cont = 0;

        char  cadenacar[50] = { 0 };

        char  cadenacar2[50] = { 0 };

        for (j = 5; j <= 20;)

        {
                for (k = 2; k <= 5; k++)

                {
```

```
                        for (i = 1; i <= 10; i++)
                        {
                                sprintf(cadenacar, "Bateria Sin fechas/inst_%d_%d_p_%d", j, k, i);
                                FILE* primerfichero = fopen(cadenacar, "r");
                                fscanf(primerfichero, "%d %d", &m, &n);
                                MAT_INT Tpro = DIM_MAT_INT(m, n);
                                Tpro = loadPTimes_nrows(cadenacar, &n, &m, YES);
                                sprintf(cadenacar2,"gurobiSGST/SGSTmodelsum_%d_%d_%d.lp", j, k, i);
                                FILE* secunfichero = fopen(cadenacar2, "w");
                                SGSTmodel(n, m, Tpro, secunfichero);

                                fclose(secunfichero);
                                fclose(secunfichero);
                                fflush(stdin);
                                cont = cont + 1;
                        }
                }
                j = j + 5;
        }
        return cont;
}

void SGSTmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero)
{
        int i, k, r, s;
        //FUNCIÓN OBJETIVO
       //FO Cmax
       fprintf(secunfichero, "Minimize \n");
       fprintf(secunfichero, "CMAX \n", m, n);

        //FO sumCi
        fprintf(secunfichero, "Minimize \n");

        for (i = 1; i <= n; i++)
        {
                if (i < n)
                {
                        fprintf(secunfichero, "C_%d_%d + ", m, i);

                }
                else
                {
                        fprintf(secunfichero, "C_%d_%d\n", m, i);

                }
        }

        //RESTRICCIONES DEL MODELO
        fprintf(secunfichero, "Subject To \n");
        s = 0;
```

```
for (r = 1; r <= m; r++)
{
        for (i = 1; i <= n; i++)
        {
                s = 500 + s + Tpro[r - 1][i - 1];
        }
}

//Primera restricción

for (i = 1; i <= n; i++)
{
        for (k = 2; k <= n; k++)
        {
                if (i < k)
                {
                        if (i < n - 1)
                        {
                                fprintf(secunfichero, "U_%d_%d + U_%d_%d + ", i, k, k, i);
                        }
                        else
                        {
                        fprintf(secunfichero, "U_%d_%d + U_%d_%d = %d \n", i, k, k, i, n - 1);


                        }
                }
        }
}

//Segunda restriccion

for (i = 1; i <= n; i++)
{
        fprintf(secunfichero, "C_1_%d - T_1_%d >= 0 \n", i, i);
}
//Tercera restricción

for (r = 1; r <= m - 1; r++)
{
        for (i = 1; i <= n; i++)
        {
        fprintf(secunfichero, "C_%d_%d - C_%d_%d - T_%d_%d >= 0 \n", r + 1, i, r, i, r + 1, i);
        }
}
//Cuarta restricción

for (r = 1; r <= m; r++)
{
```

```
                for (i = 1; i <= n; i++)
                {
                        for (k = 2; k <= n; k++)
                        {
                                if (k > i)
                                {
                                        fprintf(secunfichero, "C_%d_%d - C_%d_%d + [ P * D_%d_%d ] -
T_%d_%d >= 0 \n", r, i, r, k, i, k, r, i);
                                }
                        }
                }
        }
        //Quinta restricción
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        for (k = 2; k <= n; k++)
                        {
                                if (k > i)
                                {
                                        fprintf(secunfichero, "C_%d_%d - C_%d_%d + P - [ P * D_%d_%d ]
- T_%d_%d >= 0 \n", r, k, r, i, i, k, r, k);
                                }
                        }
                }
        }
        //Sexta restricción

        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        for (k = 2; k <= n; k++)
                        {
                                if (k > i)
                                {
                                        fprintf(secunfichero, "C_%d_%d - C_%d_%d + [ P * D_%d_%d ] -
T_%d_%d + [ P * U_%d_%d ] <= %d \n", r, i, r, k, i, k, r, i, k, i, s);
                                }
                        }
                }
        }
        //Séptima restricción
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        for (k = 2; k <= n; k++)
                        {
```

```
                                if (k > i)
                                {
                                        fprintf(secunfichero, "C_%d_%d - C_%d_%d + P - [ P * D_%d_%d ]
- T_%d_%d + [ P * U_%d_%d ] <= %d \n", r, k, r, i, i, k, r, k, i, k, s);
                                }
                        }
                }
        }

        //Tiempos de proceso
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        fprintf(secunfichero, "T_%d_%d = %d \n", r, i, Tpro[r - 1][i - 1]);

                }
        }
        fprintf(secunfichero, "P = %d \n", s);

        //Variables
        fprintf(secunfichero, "Generals \n");
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        fprintf(secunfichero, "C_%d_%d \n", r, i);
                }
        }

        fprintf(secunfichero, "Binary \n");
        for (i = 1; i <= n; i++)
        {
                for (k = 1; k <= n; k++)
                {
                        if (k > i)
                        {
                                fprintf(secunfichero, "D_%d_%d \n", i, k);
                        }
                }
        }
        for (i = 1; i <= n; i++)
        {
                for (k = 2; k <= n; k++)
                {
                        if (i < k)
                        {
                                fprintf(secunfichero, "U_%d_%d \nU_%d_%d \n", i, k, k, i);

                        }
```

```
		}
	}

	fprintf(secunfichero, "End");
}
```

10.2.2. *SGST model FO sumCi*

```
//FO sumCi
	fprintf(secunfichero, "Minimize \n");

	for (i = 1; i <= n; i++)
	{
		if (i < n)
		{
			fprintf(secunfichero, "C_%d_%d + ", m, i);

		}
		else
		{
			fprintf(secunfichero, "C_%d_%d\n", m, i);

		}
	}
```

10.2.3. *LYeq model*

```
#include <stdio.h>
#include <stdlib.h>
#include <schedule_lib.h>
void LYeqmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero);
int main()
{
	printf("Modelo SGST\n");
	int m, n, k, j, i;
	int  cont = 0;
	char  cadenacar[50] = { 0 };
	char  cadenacar2[50] = { 0 };
	for (j = 5; j <= 20;)

	{
		for (k = 2; k <= 5; k++)

		{
			for (i = 1; i <= 10; i++)
			{
				sprintf(cadenacar, "Bateria Sin fechas/inst_%d_%d_p_%d", j, k, i);
				FILE* primerfichero = fopen(cadenacar, "r");
```

```
                                fscanf(primerfichero, "%d %d", &m, &n);
                                MAT_INT Tpro = DIM_MAT_INT(m, n);
                                Tpro = loadPTimes_nrows(cadenacar, &n, &m, YES);
                                sprintf(cadenacar2, "gurobiLYeq/LYeqmodel_%d_%d_%d.lp", j, k, i);
                                FILE* secunfichero = fopen(cadenacar2, "w");
                                LYeqmodel(n, m, Tpro, secunfichero);

                                fclose(secunfichero);
                                fclose(secunfichero);
                                fflush(stdin);
                                cont = cont + 1;
                        }
                }
                j = j + 5;
        }
        return cont;
}

void LYeqmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero)
{
        int r, i, k, s, j;
        //FO Cmax

        fprintf(secunfichero, "Minimize \n");
        fprintf(secunfichero, "CMAX \n");
        //RESTRICCIONES DEL MODELO
        fprintf(secunfichero, "Subject To \n");
        s = 0;
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        s = 500 + s + Tpro[r - 1][i - 1];
                }
        }
        //Primera restricción

        for (i = 1; i <= n; i++)
        {
                for (k = 2; k <= n; k++)
                {
                        if (i < k)
                        {
                                if (i < n - 1)
                                {
                                        fprintf(secunfichero, "U_%d_%d + U_%d_%d + ", i, k, k, i);
                                }
                                else
                                {
                        fprintf(secunfichero, "U_%d_%d + U_%d_%d = %d \n", i, k, k, i, n - 1);
```

```
                                }
                        }
                }
        }
        //Segunda restriccióon

        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "C_1_%d - T_1_%d >= 0 \n", i, i);
        }
        //Tercera restriccion \n");
        for (r = 1; r <= m - 1; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        fprintf(secunfichero, "C_%d_%d - C_%d_%d - T_%d_%d >= 0 \n", r + 1, i, r, i, r
+ 1, i);
                }
        }
        //Cuarta restricción
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        for (k = 2; k <= n; k++)
                        {
                                if (k > i)
                                {
                                        fprintf(secunfichero, "C_%d_%d - C_%d_%d + [ P * D_%d_%d ] -
T_%d_%d + [ P * U_%d_%d ] <= %d \n", r, i, r, k, i, k, r, i, k, i, s);
                                }
                        }
                }
        }
        //Quinta restricción
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        for (k = 2; k <= n; k++)
                        {
                                if (k > i)
                                {
                                        fprintf(secunfichero, "C_%d_%d - C_%d_%d + P - [ P * D_%d_%d ]
- T_%d_%d + [ P * U_%d_%d ] <= %d \n", r, k, r, i, i, k, r, k, i, k, s);
                                }
                        }
                }
```

```
	}
	//Sexta restricción
	for (r = 1; r <= m; r++)
	{
		for (i = 1; i <= n; i++)
		{
			for (k = 2; k <= n; k++)
			{
				if (k > i)
				{
					fprintf(secunfichero, "[ P * D_%d_%d ] + C_%d_%d - C_%d_%d -
T_%d_%d - Q_%d_%d_%d = 0 \n", i, k, r, i
							, r, k, r, i, r, i, k);
				}
			}
		}
	}
	//Séptima restricción
	for (r = 1; r <= m; r++)
	{
		for (i = 1; i <= n; i++)
		{
			for (k = 2; k <= n; k++)
			{
				if (k > i)
				{
					fprintf(secunfichero, "Q_%d_%d_%d - P + T_%d_%d + T_%d_%d <= 0
\n", r, i, k, r, i, r, k);
				}
			}
		}
	}
	//Octava restricción
	for (i = 1; i <= n; i++)
	{
		fprintf(secunfichero, "CMAX - C_%d_%d >= 0 \n", m, i);
	}
	//Tiempos de proceso

	for (r = 1; r <= m; r++)
	{
		for (i = 1; i <= n; i++)
		{
			fprintf(secunfichero, "T_%d_%d = %d \n", r, i, Tpro[r - 1][i - 1]);

		}
	}
	fprintf(secunfichero, "P = %d \n", s);

	//Variables
```

```
        fprintf(secunfichero, "Generals \n");
        for (i = 1; i <= m; i++)
        {
                for (j = 1; j <= n; j++)
                {
                        fprintf(secunfichero, "C_%d_%d \n", i, j);
                }
        }
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        for (k = 2; k <= n; k++)
                        {
                                if (k > i)
                                {
                                        fprintf(secunfichero, "Q_%d_%d_%d \n", r, i, k);
                                }
                        }
                }
        }
        fprintf(secunfichero, "CMAX \n");
        fprintf(secunfichero, "Binary \n");
        for (i = 1; i <= n; i++)
        {
                for (k = 2; k <= n; k++)
                {
                        if (k > i)
                        {
                                fprintf(secunfichero, "D_%d_%d \n", i, k);
                        }
                }
        }
        for (i = 1; i <= n; i++)
        {
                for (k = 2; k <= n; k++)
                {
                        if (i < k)
                        {
                                fprintf(secunfichero, "U_%d_%d \nU_%d_%d \n", i, k, k, i);


                        }
                }
        }

        fprintf(secunfichero, "End");
}
```

10.2.4. *LYeq model FO sumCi*

```
//FO sumCi
fprintf(secunfichero, "Minimize \n");

for (i = 1; i <= n; i++)
{
	if (i < n)
	{
		fprintf(secunfichero, "C_%d_%d + ", m, i);

	}
	else
	{
		fprintf(secunfichero, "C_%d_%d\n", m, i);

	}
}
```

10.3. Código modelos adicionales

10.3.1. *TBA model*

```
#include <stdio.h>
#include <stdlib.h>
#include <schedule_lib.h>
void TBAmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero);
int main()
{
	printf("Modelo TBA\n");
	int m, n, k, j, i;
	int  cont = 0;
	char  cadenacar[50] = { 0 };
	char  cadenacar2[50] = { 0 };
	for (j = 5; j <= 20;)

	{
		for (k = 2; k <= 5; k++)

		{
			for (i = 1; i <= 10; i++)
			{
				sprintf(cadenacar, "Bateria Sin fechas/inst_%d_%d_p_%d", j, k, i);
				FILE* primerfichero = fopen(cadenacar, "r");
				fscanf(primerfichero, "%d %d", &m, &n);
```

```
                                MAT_INT Tpro = DIM_MAT_INT(m, n);
                                Tpro = loadPTimes_nrows(cadenacar, &n, &m, YES);
                                sprintf(cadenacar2, "gurobiTBA/TBAmodel_%d_%d_%d.lp", j, k, i);
                                FILE* secunfichero = fopen(cadenacar2, "w");
                                TBAmodel(n, m, Tpro, secunfichero);

                                fclose(secunfichero);
                                fclose(secunfichero);
                                fflush(stdin);
                                cont = cont + 1;
                        }
                }
                j = j + 5;
        }
        return cont;
}

void TBAmodel(int n, int m, MAT_INT Tpro, FILE* secunfichero)
{
        int i, j, p, q, r, s;
        //FUNCION OBJETIVO
        //FO Cmax
        fprintf(secunfichero, "Minimize \n");
        for (p = 1; p <= m - 1; p++)
        {
                for (i = 1; i <= n; i++)
                {
                        fprintf(secunfichero, "[ T_%d_%d * Z_%d_1 ] + ", p, i, i);
                }
        }
        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "T_%d_%d + ", m, i);
        }
        for (r = 2; r <= m; r++)
        {
                if (r < m)
                {
                        fprintf(secunfichero, "X_%d + ", r);
                }
                else
                {
                        fprintf(secunfichero, "X_%d \n", r);
                }
        }

        //RESTRICCIONES DEL MODELO
        fprintf(secunfichero, "Subject To \n");

        //Primera restricción
```

```
	for (i = 1; i <= n; i++)
	{
		for (j = 1; j <= n; j++)
		{
			if (j < n)
			{
				fprintf(secunfichero, "Z_%d_%d + ", i, j);
			}
			else
			{
				fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
			}
		}
	}
	//Segunda restricción
	for (j = 1; j <= n; j++)
	{
		for (i = 1; i <= n; i++)
		{
			if (i < n)
			{
				fprintf(secunfichero, "Z_%d_%d + ", i, j);
			}
			else
			{
				fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
			}
		}
	}
	//Tercera restricción

	for (r = 2; r <= m; r++)
	{
		for (j = 2; j <= n; j++)
		{
			for (i = 1; i <= n; i++)
			{
				fprintf(secunfichero, "%d Z_%d_1 + ", Tpro[r - 2][i - 1], i);
			}
			for (q = 1; q <= j - 1; q++)
			{
				for (i = 1; i <= n; i++)
				{
					fprintf(secunfichero, "%d Z_%d_%d - %d Z_%d_%d + ", Tpro[r -
1][i - 1], i, q, Tpro[r - 2][i - 1], i, q);
				}
			}

			fprintf(secunfichero, "X_%d - ", r);
```

```
                        for (i = 1; i <= n; i++)
                        {
                                if (i < n)
                                {
                                        fprintf(secunfichero, "%d Z_%d_%d - ", Tpro[r - 2][i - 1], i,
j);
                                }
                                else
                                {
                                        fprintf(secunfichero, "%d Z_%d_%d >= 0 \n", Tpro[r - 2][i -
1], i, j);
                                }
                        }
                }
        }
        //Cuarta restricción
        for (r = 1; r <= m; r++)
        {
                for (j = 1; j <= n - 1; j++)
                {
                        fprintf(secunfichero, "E_%d_%d + ", r, j);
                        for (i = 1; i <= n; i++)
                        {
                                if (i < n)
                                {
                                        fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[r - 1][i - 1], i,
j + 1);
                                }
                                else
                                {
                                        fprintf(secunfichero, " %d Z_%d_%d ", Tpro[r - 1][i - 1], i, j
+ 1);
                                }
                        }
                        fprintf(secunfichero, "- E_%d_%d = 0 \n", r, j + 1);
                }
        }
        //Quinta restricción
        for (r = 1; r <= m - 1; r++)
        {
                fprintf(secunfichero, "E_%d_1 + ", r);
                for (i = 1; i <= n; i++)
                {
                        if (i < n)
                        {
                                fprintf(secunfichero, " %d Z_%d_1 + ", Tpro[r][i - 1], i);
                        }
                        else
                        {
```

```
                        fprintf(secunfichero, " %d Z_%d_1 ", Tpro[r][i - 1], i);
                }
        }
        fprintf(secunfichero, " - E_%d_1 + X_%d = 0 \n", r + 1, r + 1);
}

//Sexta restricción

for (r = 1; r <= m - 1; r++)
{
        for (j = 2; j <= n; j++)
        {
                fprintf(secunfichero, "E_%d_%d + ", r, j);
                for (i = 1; i <= n; i++)
                {
                        if (i < n)
                        {
                                fprintf(secunfichero, " %d Z_%d_%d + ", Tpro[r][i - 1], i, j);
                        }
                        else
                        {
                                fprintf(secunfichero, " %d Z_%d_%d ", Tpro[r][i - 1], i, j);
                        }
                }
                fprintf(secunfichero, "- E_%d_%d <= 0 \n", r + 1, j);
        }
}
//Tiempos de proceso
for (r = 1; r <= m; r++)
{
        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "T_%d_%d = %d \n", r, i, Tpro[r - 1][i - 1]);
        }
}

//Variables
fprintf(secunfichero, "Generals \n");
for (r = 1; r <= m; r++)
{
        for (j = 1; j <= n; j++)
        {
                fprintf(secunfichero, "E_%d_%d \n", r, j);
        }
}
for (r = 2; r <= m; r++)
{
        fprintf(secunfichero, "X_%d \n", r);
}
fprintf(secunfichero, "Binary \n");
```

```
        for (i = 1; i <= n; i++)
        {
                for (j = 1; j <= n; j++)
                {
                        fprintf(secunfichero, "Z_%d_%d \n", i, j);
                }
        }

        fprintf(secunfichero, "End");
}
```

10.3.2. *TBA model FO sumCi*

```
//FO sumCi
fprintf(secunfichero, "Minimize \n");

for (q = 1; q <= m - 1; q++)
{
        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "[ %d T_%d_%d * Z_%d_1 ] + ", n, q, i, i);
        }
}

for (j = 1; j <= n; j++)
{
        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "[ %d T_%d_%d * Z_%d_%d ] + ", n - j + 1, m, i, i, j);
        }

}
for (q = 2; q <= m; q++)
{
        if (q < m)
        {
                fprintf(secunfichero, "%d X_%d + ", n, q);
        }
        else
        {
                fprintf(secunfichero, "%d X_%d\n ", n, q);
        }
}
```

10.3.3. *TS3 model*

```
#include <stdio.h>
#include <stdlib.h>
#include <schedule_lib.h>
void TS3model(int n, int m, MAT_INT Tpro, FILE* secunfichero);
```

```
int main()
{
	printf("Modelo TS3\n");
	int m, n, k, j, i;
	int  cont = 0;
	char  cadenacar[50] = { 0 };
	char  cadenacar2[50] = { 0 };
	for (j = 5; j <= 20;)

	{
		for (k = 2; k <= 5; k++)

		{
			for (i = 1; i <= 10; i++)
			{
				sprintf(cadenacar, "Bateria Sin fechas/inst_%d_%d_p_%d", j, k, i);
				FILE* primerfichero = fopen(cadenacar, "r");
				fscanf(primerfichero, "%d %d", &m, &n);
				MAT_INT Tpro = DIM_MAT_INT(m, n);
				Tpro = loadPTimes_nrows(cadenacar, &n, &m, YES);
				sprintf(cadenacar2, "gurobiTS3/TS3model_%d_%d_%d.lp", j, k, i);
				FILE* secunfichero = fopen(cadenacar2, "w");
				TS3model(n, m, Tpro, secunfichero);

				fclose(secunfichero);
				fclose(secunfichero);
				fflush(stdin);
				cont = cont + 1;
			}
		}
		j = j + 5;
	}
	return cont;
}

void TS3model(int n, int m, MAT_INT Tpro, FILE* secunfichero)
{
	int i, j, r, p, q;

	//FO Cmax
	fprintf(secunfichero, "Minimize \n");
	for (p = 1; p <= n; p++)
	{
		fprintf(secunfichero, "T_1_%d + ", p);
	}
	for (q = 2; q <= m; q++)
	{
		for (i = 1; i <= n; i++)
		{
			fprintf(secunfichero, "[ T_%d_%d * Z_%d_%d ] + ", q, i, i, n);
```

```
        }
}
for (r = 2; r <= m; r++)
{
        if (r < m)
        {
                fprintf(secunfichero, "Y_%d_%d + ", r - 1, n);
        }
        else
        {
                fprintf(secunfichero, "Y_%d_%d \n", r - 1, n);
        }
}
//RESTRICCIONES DEL MODELO
fprintf(secunfichero, "Subject To \n");

//Primera restricción
for (i = 1; i <= n; i++)
{
        for (j = 1; j <= n; j++)
        {
                if (j < n)
                {
                        fprintf(secunfichero, "Z_%d_%d + ", i, j);
                }
                else
                {
                        fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
                }
        }
}
//Segunda restricción
for (j = 1; j <= n; j++)
{
        for (i = 1; i <= n; i++)
        {
                if (i < n)
                {
                        fprintf(secunfichero, "Z_%d_%d + ", i, j);
                }
                else
                {
                        fprintf(secunfichero, "Z_%d_%d = 1 \n", i, j);
                }
        }
}
//Tercera restricción
for (r = 2; r <= m; r++)
{
        for (j = 2; j <= n; j++)
```

```
		{
			for (i = 1; i <= n; i++)
			{
				if (i < n)
				{
					fprintf(secunfichero, "%d Z_%d_%d + ", Tpro[0][i - 1], i, j - 1);
				}
				else
				{
					fprintf(secunfichero, "%d Z_%d_%d - ", Tpro[0][i - 1], i, j - 1);
				}
			}
			for (i = 1; i <= n; i++)
			{
				if (i < n)
				{
					fprintf(secunfichero, "%d Z_%d_%d - ", Tpro[r - 1][i - 1], i, j - 1);
				}
				else
				{
					fprintf(secunfichero, "%d Z_%d_%d + ", Tpro[r - 1][i - 1], i, j - 1);
				}
			}
			for (q = 1; q <= r - 1; q++)
			{
				for (i = 1; i <= n; i++)
				{
					fprintf(secunfichero, "%d Z_%d_%d - %d Z_%d_%d + ", Tpro[q - 1][i - 1], i, j, Tpro[q - 1][i - 1], i, j - 1);
				}
			}
			for (q = 1; q <= r - 1; q++)
			{
				if (q < r - 1)
				{
					fprintf(secunfichero, "Y_%d_%d - Y_%d_%d + ", q, j, q, j - 1);
				}
				else
				{
					fprintf(secunfichero, "Y_%d_%d - Y_%d_%d = 0 \n", q, j, q, j - 1);
				}

			}
		}
	}
```

```
        //Tiempos de proceso
        for (r = 1; r <= m; r++)
        {
                for (i = 1; i <= n; i++)
                {
                        fprintf(secunfichero, "T_%d_%d = %d \n", r, i, Tpro[r - 1][i - 1]);
                }
        }

        //Variables
        fprintf(secunfichero, "Generals \n");
        for (r = 1; r <= m - 1; r++)
        {
                for (j = 1; j <= n; j++)
                {
                        fprintf(secunfichero, "Y_%d_%d \n", r, j);
                }
        }
        fprintf(secunfichero, "Binary \n");
        for (i = 1; i <= n; i++)
        {
                for (j = 1; j <= n; j++)
                {
                        fprintf(secunfichero, "Z_%d_%d \n", i, j);
                }
        }

        fprintf(secunfichero, "End");
}
```

10.3.4. *TS3 model FO sumCi*

```
//FO sumCi
fprintf(secunfichero, "Minimize \n");
for (q = 1; q <= m - 1; q++)
{
        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "[ %d T_%d_%d * Z_%d_1 ] + ", n, q, i, i);
        }

}
for (j = 1; j <= n; j++)
{
        for (i = 1; i <= n; i++)
        {
                fprintf(secunfichero, "[ %d T_%d_%d * Z_%d_%d ] + ", n - j + 1, m, i, i, j);
        }
```

```
}
for (q = 1; q <= m - 1; q++)
{
        if (q < m - 1)
        {
                fprintf(secunfichero, "%d Y_%d_1 + ", n, q);
        }
        else
        {
                fprintf(secunfichero, "%d Y_%d_1\n ", n, q);
        }
}
```