# RESOLUCIÓN DEL JOB SHOP SCHEDULING PROBLEM MEDIANTE REGLAS DE PRIORIDAD

TRABAJO FIN DE GRADO DE INGENIERÍA INFORMÁTICA DEL SOFTWARE

Universidad de Oviedo

FEBRERO DE 2022


Autor

Cristina Ruiz de Bucesta Crespo

Directores

Francisco Javier Gil Gala

María Rita Sierra Sánchez

*…Tuve otras razones que me hicieron pensar en escribir todas las cosas que me parecieron de alguna importancia a medida que fuese descubriendo la verdad, y poniendo en ello el mismo cuidado que si las tuviera que imprimir, porque así dispondría de mayor espacio para examinarlas bien.*

*Pues es indudable que se pone mayor atención en lo que se cree que han de ver muchos que en lo que se hace únicamente para sí mismo. Tanto, que a veces las cosas que me han parecido ciertas al comenzar a concebirlas me han parecido falsas cuando he intentado estamparlas en papel…*

*… Y, en efecto, quiero que se sepa que lo poco que hasta aquí he aprendido no es casi nada en comparación con lo que ignoro y no desespero de aprender.*

René Descartes, 1637

Discurso del método (Sexta parte)

# Agradecimientos

A mi madre Raquel, y a mi padre Manuel.

A mi abuela Basi.

A mis tíos, María Jesús y José Antonio.

Gracias por apoyarme siempre.

A mis tutores, Fran y María, gracias por vuestra paciencia y dedicación.

# Resumen

Los problemas de *scheduling* (planificación de tareas) consisten en la organización de una serie de operaciones que requieren el uso de un conjunto finito de recursos, cumpliendo una o varias restricciones. Estos problemas tienen gran interés, por su presencia en el mundo industrial, así como en entornos productivos y de servicios, por lo que se puede deducir que el hecho de encontrar buenas soluciones a estos problemas resulta, por tanto, bastante importante.

Dentro del campo de la complejidad computacional y *scheduling*, el problema ***Job Shop Scheduling (JSS)*** es un problema clásico de **optimización** e investigación de operaciones. Problemas tan cotidianos y diarios como puede ser fijar una cita con un grupo de personas o preparar una receta de cocina pueden depender de muchos aspectos interdependientes que pueden llegar a producir conflictos entre ellos, produciendo así la existencia de un conjunto de restricciones que han de ser satisfechas para encontrar una solución al problema.

A pesar de que el ámbito en el que más aparecen este tipo de problemas de secuenciación de tareas resulte ser el ámbito industrial – de ahí la importancia de calcular buenas soluciones que permitan ahorrar la mayor cantidad de recursos posible –, el mero hecho de planificar tareas se manifiesta de manera diaria en nuestras vidas.

Uno de los métodos comúnmente empleados para la resolución de este tipo de problema son las reglas de prioridad. A diferencia de otros métodos, las reglas de prioridad ayudan a construir la planificación de manera incremental, seleccionando en cada paso qué tarea debe ser planificada. El uso de reglas de prioridad aporta la ventaja de que permite una rápida adaptación a las condiciones del sistema, incluso si estas varían.

El objetivo de este trabajo es abordar la resolución del problema *JSS* empleando el algoritmo de Giffler y Thompson (G&T) [1], guiado por diferentes reglas de prioridad clásicas. Tras el estudio de la definición formal del problema, del algoritmo y de las reglas de prioridad, se implementará un prototipo que permita resolver instancias del problema para distintas funciones objetivo. Con él, se diseñará y realizará un estudio experimental que aborda la resolución de instancias del problema con las diferentes reglas de prioridad implementadas, permitiendo comparar las soluciones factibles calculadas por estas y extraer conclusiones al respecto.

# Abstract

Scheduling is a field of computing which consists of assigning a group of tasks to a finite set of resources, meeting one or more type of constraints. Scheduling problems themselves involve such a great interest among the industrial environment, as well as service or production ones, thus it can indeed be assumed that the fact of reaching fair solutions to these problems may be truly important.

Within the field of computational complexity and scheduling, the *Job Shop Scheduling Problem (JSS)* is a classical optimization and operation research problem. Daily issues such as setting a date with a group of people or getting to prepare a cooking recipe may depend on many interdependent aspects that could end up producing some inconveniences among them. This leads to the previously mentioned existence of a set of constraints that must be satisfied in order to find out a solution to the problem.

Even though it is the industrial environment the scope in which this kind of problems can be applied the most – here rises the significance of reaching feasible solutions that may help to save as many resources as possible –, truth is the mere fact of scheduling tasks take place every day in our lives.

One of the methods most commonly used in order to solve this kind of problem are the dispatching rules. Unlike other methods, dispatching rules play a part in developing the final schedule in a gradual way, selecting in each iteration the task to be scheduled. So, the practice of dispatching rules offers the advantage of a faster acclimatization phase to the system conditions, even if there are any variations on them.

The point of this project is to approach the resolution of the *Job Shop Scheduling Problem* using the Giffler and Thompson's algorithm (G&T) [1], led by different traditional dispatching rules. Subsequently, after the analysis of the formal definition of the problem here handled, the algorithm and all the dispatching rules, it will be implemented a prototype that may allow us to solve some problem instances for different criteria chosen beforehand. That way, it will be possible to design and consider a proper experimental study, which will allow to achieve the resolution of the problem from text instances to some feasible solutions, throughout the various dispatching rules implemented, granting the possibility of comparing all the feasible solutions obtained and drawing some conclusions from them.

## Palabras clave

***A***

Análisis experimental, *Apparent Tardiness Cost*, Apache POI, Algoritmo voraz, Aplicación

***C***

Cota inferior, Cota superior

***D***

Determinista, *Due date*, Duración

***E***

Espacio de búsqueda, *Earliest Due Date*

***F***

Fecha límite, Función objetivo

***G***

GitHub

***H***

Heurístico

***I***

Instancia, Inteligencia artificial

***J***

JAR, Java, *Job*, *Job Shop Scheduling Problem*

***L***

*Longest Processing Time*

***M***

*Makespan*, Máquina, *Minimum Completion Time*

***O***

Operación

***P***

Peso, Planificación, Planificación de tareas, Problema NP-Duro, Problema NP-*Hard*

***R***

Regla de prioridad, Requisitos, Resultados, Riesgos

***S***

*Scheduling*, Secuenciamiento, *Shortest Processing Time*

***T***

Tarea, Tardiness, Task, Tiempo de procesamiento, Trabajo

# Índice de contenido

# Índice de figuras

# Índice de tablas

## Declaración de originalidad

De acuerdo al Reglamento sobre la asignatura de Trabajo de Fin de Grado de la Universidad de Oviedo, se incluye la debida declaración de originalidad de la obra. Este proyecto y el presente documento han sido elaborados de manera totalmente original, por el autor y estudiante del Grado de Ingeniería Informática del Software, Cristina Ruiz de Bucesta Crespo, con UO257424.

Así mismo, se asegura que las fuentes utilizadas han sido citadas debidamente para aportar su valor correspondiente al trabajo, las cuales pueden ser consultadas en el apartado Referencias. El autor se declara, además, conocedor de la posibilidad de que se someta el proyecto a revisión mediante herramientas de copia que busquen localizar coincidencias, bien respecto a las correspondientes fuentes citadas, o bien respecto de fuentes alternativas.

# CAPÍTULO 1: Memoria del proyecto

## 1.1 Hoja de identificación

**Título del proyecto**: Resolución del Job Shop Scheduling Problem mediante Reglas de Prioridad

**Suministrador**:

- **Nombre y apellidos**: Cristina Ruiz de Bucesta Crespo
- **DNI**: 15485980B
- **Email**: UO257424@uniovi.es / cristinaruizdebucesta@gmail.com

**Duración estimada: 350 horas**

**Directores:** Francisco Javier Gil Gala y María Rita Sierra Sánchez

## 1.2 Introducción

Este proyecto trata de resolver el problema de planificación de tareas del *Job Shop Scheduling* (JSS), cumpliéndose las restricciones pertinentes asociadas al mismo y mediante el uso de reglas de prioridad. Para ello, se hará uso de un algoritmo voraz guiado por estas reglas de prioridad con el fin de obtener soluciones factibles, es decir, soluciones que cumplan todas las restricciones del problema y, finalmente, realizar un análisis comparativo de las mismas.

Asimismo, se deberán de satisfacer las funcionalidades establecidas en este trabajo, consistentes en el procesamiento de ficheros de datos y carga de instancias a partir de esos datos, planificación de las instancias y generación de resultados.

## 1.3 Objeto

El objeto del proyecto descrito en este documento reside en conseguir la resolución del problema de secuenciación de tareas JSS (*Job Shop Scheduling*) teniendo como objetivo primordial tener soluciones factibles al problema, en las que se planifiquen todos los trabajos, se cumplan las restricciones y se optimice la función objetivo. De esta forma, los objetivos estratégicos a lograr de este proyecto, que determinarán la finalización de este, son:

- Búsqueda y lectura de artículos científicos sobre estrategias de búsqueda y las reglas de prioridad, así como la investigación de los distintos bancos de instancias que existen y el formato que tienen.
- Diseño del algoritmo de planificación a emplear y espacio de búsqueda, así como de las distintas reglas de prioridad.
- Implementación del algoritmo de planificación.
- Implementación de las reglas de prioridad.

- Diseño e implementación de un prototipo que permita realizar un análisis experimental.
- Realización de un plan de pruebas.
- Realización de un análisis experimental que permita dictaminar qué combinación de reglas de prioridad – heurística es la mejor para resolver el problema según la instancia a resolver.

## 1.4 Motivación

La realización de este proyecto viene motivada por el grupo de investigación de la Universidad de Oviedo: Investigación en Optimización y *Scheduling* Inteligentes (iScOp). Este grupo trabaja asiduamente con heurísticos, computación evolutiva y problemas de planificación, entre otros temas. El problema abordado en este proyecto es un problema de planificación de tareas con mucha aplicación en el mundo real y que, por tanto, tiene gran interés para los investigadores que integran iScOp, pues se trata de un problema clásico donde testar diferentes técnicas inteligentes antes de aplicarlas en problemas reales.

## 1.5 Estado del arte

La investigación de *scheduling* viene desarrollándose desde los últimos 40 años y, con ella, se han ido desarrollando distintas técnicas desde reglas de prioridad -*dispatching rules* -, hasta algoritmos de ramificación y poda sofisticados, heurísticas basadas en cuellos de botella - *bottleneck based heuristics* -, y algoritmos genéticos paralelos.

La forma en la que se entiende a día de hoy el problema *Job Shop Scheduling* no debe su origen a un único investigador. Roy y Sussman (1964) [2] fueron los primeros en proponer la representación del problema *Job Shop Scheduling* mediante un grafo disyuntivo, donde los nodos son las tareas y los arcos las restricciones del problema. Fue Balas en 1969 el primero en utilizar esta representación en forma de grafo disyuntivo para resolver el problema mediante un procedimiento de enumeración implícito empleando el concepto de camino crítico [3].

Durante los años 60 se desarrollaron diversas estrategias para abordar este problema mediante métodos exactos (de manera óptima), o mediante métodos aproximados que permiten el cálculo de cotas superiores (soluciones factibles pero no óptimas). Los primeros métodos exactos empleados para la resolución del problema *JSS* fueron modelos matemáticos que empleaban programación entera mixta, considerándose el primero de ellos el propuesto en 1960 por Manne [4]. Sin embargo, estos métodos a pesar de aportar un gran valor teórico y ser una manera natural de abordar el problema, como dijo Rinnooy Kan en [5]: *“a natural way to attack scheduling problems is to formulate them as mathematical programming models”*, solo eran capaces de resolver instancias de pequeño tamaño, por lo que los resultados eran bastante decepcionantes.

Entre los métodos aproximados, las reglas de prioridad son probablemente las estrategias más aplicadas para resolver este tipo de problemas, debido a su fácil implementación y a su baja complejidad temporal [6] [7]. Las reglas de prioridad pueden ser empleadas para modificar el criterio de elección, en distintos algoritmos, a la hora de construir soluciones heurísticas, por ejemplo, podríamos emplearlas en el algoritmo voraz propuesto por Giffler y Thompson en 1960 [1], que construye soluciones en el espacio de búsqueda de las planificaciones activas que es dominante (garantiza contener al menos una solución óptima). Esta característica de dominancia hace que su uso esté ampliamente extendido en diversas estrategias, bien para construir soluciones heurísticas, bien para generar espacios de búsqueda con esta característica.

Aunque entre los métodos exactos para la resolución del problema *JSS*, el más relevante es el algoritmo de ramificación y poda propuesto por Brucker en [8], el algoritmo A* propuesto por Nilsson en 1971 [9] también ha sido empleado para su resolución en algunos trabajos [10] [43] [12]. En cuanto a los métodos no exactos, destacan las técnicas de búsqueda local basadas en las estructuras de vecindad propuestas por Dell'Amico and Trubian en (1993) [13], y que fueron empleadas por muchos otros autores, frecuentemente en combinación con una o varias metaheurísticas tales como Algoritmos Genéticos [14] [15] [16], Búsqueda Tabú o Enfriamiento Simulado [17].

Durante la década de los 70 y mediados de los 80 el énfasis recayó en justificar la complejidad del problema (perteneciente a la clase NP-dura [18] [19]), donde se demostró que solamente un conjunto de instancias determinadas puede ser resuelto en tiempo polinomial. A raíz de esto, Parker se refirió a la época previa a los 70 como BF (*Before Complexity*), y como consecuencia, desde entonces recibe el nombre de AD (*Advanced Difficulty*) [20].

Desde entonces, debido a la gran cantidad de problemas reales que pueden ser formulados como un problema de *scheduling*, ha continuado siendo un campo de estudio para los investigadores. Gracias a ello, se han ido desarrollando otras estrategias de resolución, como pueden ser las redes neuronales u otras variantes de computación evolutiva como por ejemplo la programación genética [21] que permite el cálculo automático de reglas de prioridad. Algunas de estas técnicas, han sido aplicadas por investigadores de otros campos de conocimiento como la biología, la genética y la neurofísica, para resolver problemas de *scheduling* del ámbito de conocimiento, lo que además de contribuir a la teoría de *scheduling* pone de manifiesto la naturaleza multidisciplinar de estos problemas y técnicas.

## 1.6 Requisitos iniciales

La aplicación desarrollada en este trabajo tiene como objeto principal encontrar soluciones a instancias del problema *Job Shop Scheduling*, en el espacio de búsqueda de las planificaciones activas, es decir, aquel que garantiza contener al menos una solución óptima. Para ello, es necesaria la implementación de:

- Carga y lectura de ficheros – instancias.

- Cálculo de la planificación en función de la función objetivo y regla/s de prioridad.
- Exportación de datos en un fichero Excel.

Dentro del cálculo de la planificación se incluye la implementación de un algoritmo voraz basado en el G&T [1], además de una serie de reglas de prioridad clásicas empleadas para guiar dicho algoritmo.

Toda esta funcionalidad se halla más detallada en el apartado: CAPÍTULO 3: Análisis del sistema.

## 1.7 Solución propuesta

Para la realización de las funcionalidades descritas y la resolución del problema de planificación de tareas *(JSS)* mediante el algoritmo G&T guiado con diferentes reglas de prioridad se llevará a cabo el desarrollo de una aplicación Java de escritorio – es decir, un archivo con extensión ".jar" –, con interfaz por consola de comandos con el fin de que pueda ser ejecutable en el clúster del grupo de investigación iScOp (Investigación en Optimización y Scheduling Inteligentes) de la Universidad de Oviedo. Será por tanto conveniente realizar un análisis y una comparativa entre los distintos heurísticos que se empleen. Se podrá, asimismo, comparar los datos obtenidos con cotas conocidas en el estado del arte para las instancias empleadas, o, incluso, en caso de que existan, con las "mejores soluciones encontradas" *(best known solution)* para las mismas.

## 1.8 Alcance del proyecto

Este proyecto constará de los siguientes entregables:

- **Aplicación Java**. Se corresponde con un archivo ".jar" con la citada aplicación totalmente funcional. Asimismo, se aportarán los archivos fuente con el código relativo a la aplicación, así como cualquier otro fichero que precise la citada aplicación para su correcto funcionamiento.
- **Carpeta comprimida con código fuente**. Se adjunta una carpeta comprimida en formato ".zip" que alberga el código fuente del proyecto. Para mayor detalle respecto a la localización de determinadas implementaciones en caso de ampliación del código, se recomienda consultar el apartado 7.4 Manual del programador.
- **Carpeta comprimida con instancias de prueba**. Una carpeta comprimida en formato ".zip" con diferentes directorios en los que se organizan los distintos tipos de instancias con los que probar la funcionalidad de la aplicación.
- **Excel de presupuestos**. Un documento con formato ".xslx" con los presupuestos detallados del proyecto.
- **Documento del proyecto**. Memoria del proyecto en un documento con extensión *".pdf"*.

## 1.9 Definiciones y abreviaturas

- **Algoritmo voraz:** también conocido como "devorador". Utiliza una heurística para, en cada uno de sus pasos, determinar la opción más adecuada, de entre las posibles, para alcanzar una solución no necesariamente óptima.
- **Cota inferior:** coste mínimo posible que tendrá la solución óptima.
- **Cota superior:** coste de la mejor solución alcanzada hasta el momento para un problema, no necesariamente óptima y que puede serlo, pero no estar verificada como tal.
- **Espacio de búsqueda:** conjunto de todos los posibles estados que tendrá un problema, entre los cuales se encontrarán los estados solución al problema.
- **Heurístico:** técnica que, empleando conocimiento sobre el dominio del problema, permite encontrar soluciones buenas en un tiempo razonable.
- **Instancia:** fichero con los datos necesarios para resolver un problema. Es el conjunto de elementos de entrada de un problema. En el caso del JSSP, se trata del conjunto de tareas a planificar.
- **Problema P**: problema que se resuelve mediante un algoritmo deteminista con complejidad acotada por un polinomio.
- **Problema JSS o JSSP:** Job Shop Scheduling Problem.
- **Problema JSSTWT:** Job Shop Scheduling Problem minimizing Total Weighted Tardiness.
- **Problema NP**: problema de decisión que puede ser resuelto en tiempo polinómico por una máquina de Turing no determinista.
- **Problema NP-completo**: problema de decisión que pertenece a la clase NP y al menos es tan difícil como cualquier otro problema en la clase NP.
- **Problema NP-duro o NP-hard**: Un problema *X* pertenece a esta clase si hay un problema *Y* perteneciente a la clase NP-completa, que puede ser reducido a *X* en tiempo polinómico.
- **Regla de prioridad:** expresión aritmética que permite establecer una prioridad para cada tarea, en función de sus atributos.
- **Solución aproximada:** solución que cumple las restricciones del problema (es decir, es factible) y tiene un coste superior a la solución óptima.
- **Solución óptima:** solución factible que tiene el menor coste posible.

## 1.10 Normas y referencias

Este trabajo basa su estructura en la norma ***UNE 157801:2007***, *Criterios generales para la elaboración de proyectos de sistemas de información* con algunas modificaciones debido a que se trata de un trabajo de investigación. Además, también se sigue la normativa vigente sobre la asignatura de Trabajo Fin de Grado de la Universidad de Oviedo.

El estándar ***IEEE 830-1998****, Especificación de Requisitos según el estándar de IEEE 830*, el cual indica la estructura y organización de toda la información que debe incluirse

en un documento de especificación de requerimientos de software y se empleó para la obtención e identificación de requisitos del trabajo.

La guía de ***Java Code Conventions***, proporcionada por Oracle, que se trata de una recopilación de estándares de diseño y convenios que es importante seguir a la hora de realizar un proyecto en Java.

## 1.11 Estudio de alternativas y viabilidad

Se plantean en este apartado una serie de alternativas para la realización del sistema, analizando y valorando cada una de ellas para finalizar realizando una selección de aquellas que se consideren más adecuadas.

En primer lugar, se aborda la elección del lenguaje de implementación entre las siguientes posibilidades: C, Java y Python.

| Alternativa | Implementación en C |
|---|---|
| Ventajas | Apropiado para algoritmos que requieren un alto rendimiento.<br>Alta velocidad para cálculos complejos. |
| Inconvenientes | Limitaciones de conocimiento por parte del autor. |

*Alternativa del lenguaje de programación C.*

| Alternativa | Implementación en Java |
|---|---|
| Ventajas | Lenguaje interpretado.<br>Conocido ampliamente por el autor.<br>Multiplataforma. |
| Inconvenientes | Menos veloz en la realización de cálculos complejos. |

*Alternativa del lenguaje de programación Java.*

| Alternativa | Implementación en Python |
|---|---|
| Ventajas | Implementación sencilla y rápida. |
| Inconvenientes | Más lento que un lenguaje compilado. |

*Alternativa del lenguaje de programación Python.*

Además, también se estudió la posibilidad de diseñar un *framework* como por ejemplo http://ecf.zemris.fer.hr/ o https://deap.readthedocs.io/en/master/, pero debido a la complejidad del trabajo, la inclusión de esta funcionalidad haría que se superasen las horas establecidas para la realización del mismo.

En cuanto a la selección de la alternativa final, se optó por realizar la implementación del trabajo en Java, ya que, a pesar de tener el inconveniente que presenta en la segunda tabla de alternativas, presenta varias ventajas. Además, el desconocimiento por parte del autor de un lenguaje es un inconveniente que tiene mayor peso que los demás, pues para el correcto desarrollo del proyecto sería necesario un período de "formación" en ese lenguaje.

## 1.12 Organización, gestión y planificación del proyecto

Teniendo en cuenta los objetivos marcados en el apartado 1.3 Objeto, se expone aquí la forma de realizar el trabajo que se decidió seguir.

### 1.12.1 Responsables del proyecto

Los dos actores más importantes de este trabajo son los que conforman la unión estudiante – docente investigador. Es decir, respectivamente, el desarrollador o autor del proyecto, encargado del análisis, diseño e implementación del sistema resultado del proyecto; y el tutor asignado para el seguimiento y supervisión del mismo, que actúa también como investigador-cliente del mismo. Pueden considerarse otras entidades implicadas en el proyecto el grupo de investigación de la Universidad de Oviedo: Investigación en Optimización y Scheduling Inteligentes (iScOp), a cuyos integrantes se les debe la idea de la realización de este trabajo, así como a la propia Universidad de Oviedo.

### 1.12.2 Organización del proyecto

La organización del proyecto se ha llevado a cabo según un plan de trabajo adaptado a la naturaleza de un trabajo de investigación y las necesidades que esto conlleva. Teniendo en consideración los plazos y demás aspectos establecidos por parte del autor del proyecto, se diseñó una planificación inicial para la elaboración del proyecto y cada una de sus secciones. A lo largo de la misma, no obstante, esta planificación fue cambiando para adaptarse a la realidad de la evolución del proyecto.

### 1.12.3 Metodología planteada

Dado el plan de trabajo necesario para este tipo de proyecto, lo estipulado en el propio trabajo de fin de grado, así como las necesidades del alumno, la metodología empleada no ha sido una metodología clásica definida formalmente, sino una metodología de trabajo más parecida a la que se emplea en los proyectos de

investigación, siendo los actores participantes el estudiante y el docente investigador, es decir, el autor del proyecto junto con un director o tutor que se encargue de guiar al autor hacia una solución satisfactoria.

### 1.12.4 Planificación temporal

La planificación de este proyecto se realizó al inicio del mismo, momento en el que también se comenzó con la documentación. El comienzo del proyecto se establece el día jueves 18 de febrero de 2021, cuando tuvo lugar la primera reunión con los tutores de este, se explicaron conceptos y se aclaró el camino a seguir en el trabajo. Se plantea un calendario para las funcionalidades de 3 horas diarias y 4 días a la semana y otro calendario distinto para la documentación, siendo de 1 hora y 3 días a la semana, pues es más complicado establecer un horario para la documentación ya que se encuentra supeditada al curso del diseño e implementación del código. Ambos calendarios presentan una pausa durante el mes de agosto. Se establece como fecha de fin del trabajo el jueves 20 de enero de 2022.

| Id | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| **1** | **Trabajo Fin de Grado** | **533 horas** | **jue 18/02/21** | **jue 20/01/22** |
| **1.1** | **Inicio del proyecto** | **44 horas** | **jue 18/02/21** | **mar 16/03/21** |
| 1.1.1 | Primera reunión | 2 horas | jue 18/02/21 | jue 18/02/21 |
| **1.1.2** | **Análisis del problema** | **32 horas** | **jue 25/02/21** | **mar 16/03/21** |
| **1.1.3** | **Estudio del problema** | **12 horas** | **jue 18/02/21** | **jue 25/02/21** |
| 1.1.4 | Estudio y decisión de las tecnologías empleadas | 5 horas | jue 18/02/21 | mar 23/02/21 |
| **1.2** | **Diseño de la solución del problema** | **36 horas** | **mar 16/03/21** | **lun 05/04/21** |
| 1.2.1 | Diseño algoritmo de planificación y espacio de búsqueda | 18,89 horas | mar 16/03/21 | lun 29/03/21 |
| 1.2.2 | Estudio y diseño de reglas de prioridad | 12 horas | mar 30/03/21 | lun 05/04/21 |
| **1.3** | **Implementación del sistema** | **205 horas** | **mar 23/02/21** | **mar 22/06/21** |
| 1.3.1 | Segunda reunión | 2 horas | lun 14/06/21 | mar 15/06/21 |
| 1.3.2 | Implementación básica del sistema | 15 horas | mar 23/02/21 | mié 03/03/21 |
| 1.3.3 | Implementación del algoritmo de planificación | 15 horas | mar 15/06/21 | mar 22/06/21 |
| **1.3.4** | **Implementación de las reglas de prioridad definidas** | **15 horas** | **mar 15/06/21** | **mar 22/06/21** |
| **1.4** | **Análisis experimental** | **135 horas** | **mar 05/10/21** | **jue 16/12/21** |

| Id | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1.4.1 | Tercera reunión | 2 horas | mar 05/10/21 | mar 05/10/21 |
| 1.4.2 | Cambios y mejoras en las reglas de prioridad | 10 horas | mié 06/10/21 | mar 12/10/21 |
| 1.4.3 | Implementación de salida de datos del sistema | 20 horas | mié 13/10/21 | lun 25/10/21 |
| 1.4.4 | Preparación y ejecución de los experimentos mediante la carga de instancias | 10 horas | mar 26/10/21 | lun 01/11/21 |
| 1.4.5 | Cuarta reunión | 3 horas | jue 09/12/21 | jue 09/12/21 |
| 1.4.6 | Cambios en cuanto a la salida de datos del sistema | 2 horas | vie 10/12/21 | lun 13/12/21 |
| 1.4.7 | Crear hojas Excel con los resultados, crear gráficas y analizar resultados, comparándolos con otros estudios | 10 horas | lun 13/12/21 | jue 16/12/21 |
| **1.5** | **Pruebas del sistema** | **231 horas** | **vie 04/06/21** | **mié 17/11/21** |
| 1.5.1 | Pruebas unitarias | 10 horas | jue 04/06/21 | mié 10/06/21 |
| 1.5.2 | Búsqueda y comparativa de resultados con otros estudios | 12 horas | jue 11/11/21 | mié 17/11/21 |
| 1.6 | Documentación | 113 horas | mar 02/03/21 | jue 20/01/22 |

*Tabla 1. Estructura de desglose del trabajo (WBS) simplificada, con la duración del proyecto.*

El trabajo tiene una duración de 533 horas, no obstante, es necesario establecer la diferencia entre “trabajo” y “duración”, siendo el primero la diferencia entre la fecha fin de un evento y la fecha de comienzo en función del calendario establecido y las horas marcadas en el mismo. Duración representa la suma de trabajo de cada tarea de manera individual. En la Tabla 1 se representa una versión simplificada del WBS, con una profundidad máxima de nivel 3 de identificador del esquema. La versión en su totalidad se puede comprobar en Anexo I: Estructura de desglose del trabajo (Work Breakdown Structure).

| Id | Nombre de tarea | Trabajo |
|---|---|---|
| **1** | **Trabajo Fin de Grado** | **346 horas** |
| **1.1** | **Inicio del proyecto** | **86 horas** |
| 1.1.1 | Primera reunión | 2 horas |
| **1.1.2** | **Análisis del problema** | **38 horas** |
| **1.1.3** | **Estudio del problema** | **41 horas** |
| 1.1.4 | Estudio y decisión de las tecnologías empleadas | 5 horas |
| **1.2** | **Diseño de la solución del problema** | **34 horas** |
| 1.2.1 | Diseño algoritmo de planificación y espacio de búsqueda | 22 horas |
| 1.2.2 | Estudio y diseño de reglas de prioridad | 12 horas |
| **1.3** | **Implementación del sistema** | **67 horas** |
| 1.3.1 | Segunda reunión | 2 horas |
| 1.3.2 | Implementación básica del sistema | 15 horas |
| 1.3.3 | Implementación del algoritmo de planificación | 35 horas |
| **1.3.4** | **Implementación de las reglas de prioridad definidas** | **15 horas** |
| **1.4** | **Análisis experimental** | **57 horas** |
| 1.4.1 | Tercera reunión | 2 horas |
| 1.4.2 | Cambios y mejoras en las reglas de prioridad | 10 horas |
| 1.4.3 | Implementación de salida de datos del sistema | 20 horas |
| 1.4.4 | Preparación y ejecución de los experimentos mediante la carga de instancias | 10 horas |
| 1.4.5 | Cuarta reunión | 3 horas |
| 1.4.6 | Cambios en cuanto a la salida de datos del sistema | 2 horas |
| 1.4.7 | Crear hojas Excel con los resultados, crear gráficas y analizar resultados, comparándolos con otros estudios | 10 horas |
| **1.5** | **Pruebas del sistema** | **22 horas** |
| 1.5.1 | Pruebas unitarias | 10 horas |
| 1.5.2 | Búsqueda y comparativa de resultados con otros estudios | 12 horas |
| 1.6 | Documentación | 80 horas |

*Tabla 2. Estructura de desglose del trabajo (WBS) simplificada, con el tiempo de trabajo.*

Asimismo, en la Tabla 2 se muestra el total de trabajo dedicado al proyecto, con una suma de **346 horas**, tiempo consecuente respecto a lo estipulado para los créditos que forman la asignatura de Trabajo Fin de Grado. Se muestra también el diagrama de Gantt, que representa la estructura de la planificación de forma gráfica, en el Diagrama 1.

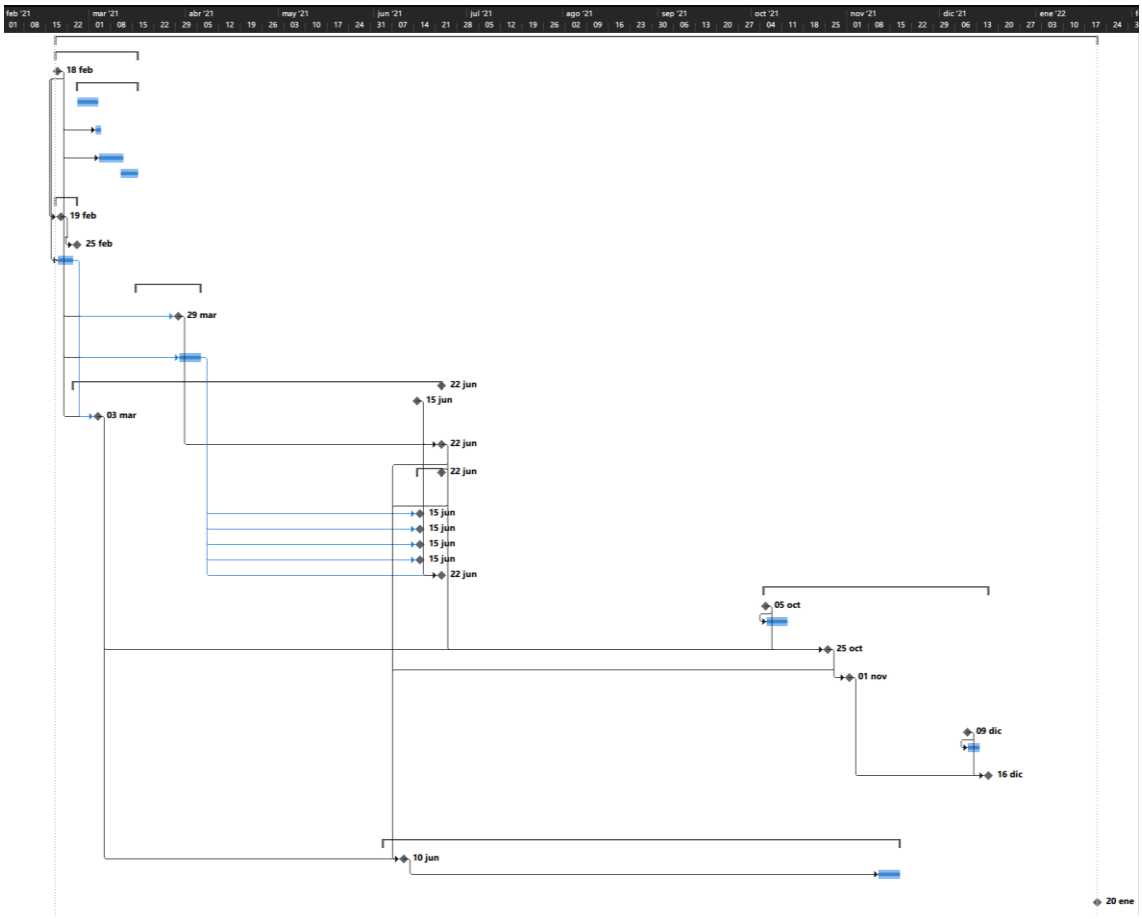



*Diagrama 1. Diagrama de Gantt.*

## 1.13 Análisis de riesgos

En todo proyecto es necesario un análisis cualitativo y cuantitativo de cada una de las situaciones de riesgo que pueden afectar al mismo, así como la forma de gestionarlos con el fin de minimizar su efecto.

Un riesgo es un evento o condición incierta que, si sucede, puede tener un efecto positivo o negativo en los objetivos del proyecto tales como el alcance, el cronograma, el coste y la calidad. En caso de ser positivos reciben el nombre de oportunidades y se realizarán las estrategias necesarias para su aprovechamiento. En cambio, en caso de ser negativos reciben el nombre de amenazas y para evitar los problemas que puedan causar, se han de establecer las estrategias necesarias para eliminarlos, transferirlos, evitarlos o aceptarlos.

### 1.13.1 Identificación y categorización de riesgo

En este apartado relativo a la identificación de riesgos se ha desarrollado un listado con todos aquellos elementos que, durante el proceso del proyecto se han considerado riesgos. Se habla de todo el ciclo de vida del proyecto ya que pueden aparecer nuevos riesgos a lo largo del mismo, así como los ya existentes pueden evolucionar o desaparecer.

Se obtiene como resultado la Tabla 3, que contiene el identificador de cada riesgo encontrado, el nombre del mismo, una breve descripción, y la clase y la categoría a la que pertenecen. La clase se corresponde con las 3 clases principales de riesgos:

- Riesgos del proyecto: afectan a la planificación o a los recursos.
- Riesgos del producto: afectan a la calidad del software que se desarrolla.
- Riesgos del negocio: afectan a la organización que desarrolla el software.

Mientras que la categoría viene de la estructura de desglose de riesgo (*Risk Breakdown Structure* o RBS) de la guía del PMBOK, que lista las categorías y subcategorías en las que los riesgos surgen para un proyecto típico, y la cual se puede encontrar en Anexo II: Estructura de desglose de riesgo (Risk Breakdown Structure).

| Código | Nombre | Descripción | Clase de riesgo | Categoría |
|---|---|---|---|---|
| 1 | No ser capaz de realizar las implementaciones requeridas | No disponer de la suficiente formación para la realización del trabajo, por lo que no es posible completarlo | Producto | Técnico (Complejidad e interfaces) |
| 2 | Cambios en los requisitos u objetivos por parte de los tutores | Modificación por parte de los tutores de los requisitos del proyecto durante la realización del mismo | Negocio | Externo (Subcontratistas y proveedores) |
| 3 | No cumplir con los objetivos marcados | No desarrollar el trabajo según los estándares marcados, por lo que el trabajo realizado no cumple con los objetivos planteados inicialmente | Producto | Técnico (Calidad) |
| 4 | Partes complejas del proyecto que requieran un mayor tiempo | Determinadas partes del proyecto requieren más tiempo del estimado inicialmente, retrasando así la fecha fin del proyecto | Proyecto | Técnico (Complejidad e interfaces) |
| 5 | No dedicar al trabajo las horas necesarias | No es posible dedicar las horas de trabajo necesarias para su finalización según el programa establecido | Proyecto | Gestión del proyecto (Planificación) |
| 6 | No finalizar el trabajo en la convocatoria inicialmente planteada | No es posible terminar el trabajo a tiempo para la presentación y defensa en la convocatoria que se había elegido al inicio | Proyecto | Gestión del proyecto (Estimación) |
| 7 | Software no apto | Alguno de los programas empleados en el proyecto, así como el lenguaje de programación o alguna librería tienen limitaciones que no permiten realizar todas las implementaciones requeridas | Negocio | Técnico (Tecnología) |

| Código | Nombre | Descripción | Clase de riesgo | Categoría |
|---|---|---|---|---|
| 8 | Mala comunicación con los tutores | No lograr comunicarse con los tutores, ya sea por imposibilidad de horarios, vías de comunicación complicadas o caídas, etc | Negocio | Externo (Subcontratistas y proveedores) |
| 9 | Cambio legislativo | Durante el desarrollo del proyecto surge algún cambio legislativo que imposibilita o complica alguna tarea de redacción del mismo | Negocio | Externo (Regulación) |

*Tabla 3. Salida de la identificación de riesgos.*

### 1.13.2 Análisis del impacto y probabilidad de los riesgos

La probabilidad de que un riesgo ocurra debe evaluarse y medirse en todo proyecto, aunque si bien es cierto que, si dicha probabilidad de ocurrencia es muy baja, su importancia en el proyecto también lo será. Como no es posible evaluar y atender cada uno de los riesgos de un proyecto, se ha de establecer prioridades de acuerdo al impacto que tengan en el mismo.

Para calcular el impacto de un riesgo en un proyecto, se ha de hacer un análisis cualitativo del impacto que tendrá en cada uno de los objetivos del proyecto (coste, tiempo, alcance y calidad), además de la probabilidad de que ocurra. Con estos datos, se aportan cifras según el Anexo III: Matriz de Probabilidad e Impacto, y así se podría obtener el valor cuantitativo del impacto de cada riesgo. Para ello, se ha de multiplicar el valor de la probabilidad por el máximo de los valores del impacto de los objetivos del proyecto. Finalmente se obtiene el listado ordenado de mayor a menor impacto que se puede observar en la Tabla 4, donde según los resultados obtenidos habría dos riesgos que habría que priorizar respecto a los demás.

| Código | Nombre del Riesgo | Probabilidad | Impacto | | | | Impacto | 0,50 |
|---|---|---|---|---|---|---|---|---|
| | | | Presup. | Planific. | Alcance | Calidad | | Priorización |
| 1 | Partes complejas del proyecto que requieran un mayor tiempo | Muy alta | Alto | Muy alto | Muy bajo | Muy bajo | 0,72 | |
| 2 | No finalizar el trabajo en la convocatoria inicialmente planteada | Alta | Alto | Muy alto | Medio | Bajo | 0,56 | |
| 3 | No dedicar al trabajo las horas necesarias | Media | Muy alto | Muy alto | Muy bajo | Medio | 0,40 | |
| 4 | Cambios en los requisitos u objetivos por parte de los tutores | Baja | Alto | Muy alto | Alto | Bajo | 0,24 | |
| 5 | No cumplir con los objetivos marcados | Baja | Medio | Muy alto | Medio | Alto | 0,24 | |
| 6 | Mala comunicación con los tutores | Baja | Bajo | Muy alto | Medio | Medio | 0,24 | |
| 7 | No ser capaz de realizar las implementaciones requeridas | Media | Bajo | Alto | Alto | Alto | 0,20 | |
| 8 | Software no apto | Baja | Muy bajo | Medio | Medio | Alto | 0,12 | |
| 9 | Cambio legislativo | Muy baja | Bajo | Alto | Alto | Medio | 0,04 | |

*Tabla 4. Salida del análisis del impacto y probabilidad de los riesgos.*

### 1.13.3 Estrategia y respuesta para los riesgos

Tras la obtención y análisis de los riesgos, es necesario estipular cómo se van a afrontar de cara a intentar no perjudicar el desarrollo del proyecto. Como resultado de este proceso de gestionar cada uno de los riesgos, obtenemos la Tabla 5.

| Código | Nombre del Riesgo | Respuesta al Riesgo | Estrategia |
|---|---|---|---|
| 1 | Partes complejas del proyecto que requieran un mayor tiempo | Dedicar el tiempo necesario a dichas partes para el correcto desarrollo del proyecto | Asumir |
| 2 | No finalizar el trabajo en la convocatoria inicialmente planteada | Procurar tener una buena organización del proyecto desde el primer momento, contando con un tiempo extra para posibles demoras | Mitigar |
| 3 | No dedicar al trabajo las horas necesarias | Planificar el proyecto y replanificar cuando sea necesario para terminarlo en el plazo estimado | Eliminar |
| 4 | Cambios en los requisitos u objetivos por parte de los tutores | Restablecer los requisitos inicialmente establecidos y replanificar el proyecto conforme a las directrices de los tutores | Asumir |
| 5 | No cumplir con los objetivos marcados | Al advertir la posibilidad de no cumplimiento de algún requisito, pedir ayuda a los tutores | Transferir |
| 6 | Mala comunicación con los tutores | Asegurarse de tener diversas vías de comunicación para que así la misma no peligre | Eliminar |
| 7 | No ser capaz de realizar las implementaciones requeridas | Al advertir la posibilidad de no cumplimiento de alguna implementación, pedir ayuda a los tutores | Transferir |
| 8 | Software no apto | Acordar en primera instancia con los tutores qué lenguaje de programación es el mejor, así como demás software necesario | Eliminar |
| 9 | Cambio legislativo | Cambiar las fuentes de información o la manera de adquirir documentación, tipos impositivos, etc | Asumir |

*Tabla 5. Estrategia y respuesta planeada para cada riesgo.*

## 1.14 Resumen del presupuesto

Dado que este proyecto se trata de un trabajo de investigación, resulta algo complicado estimar el coste real de cada aspecto. A pesar de que muchos costes no se hayan producido por tratarse de un trabajo académico, en este apartado se intentará aportar cifras aproximadas al coste real en un proyecto.

El presupuesto de cliente se corresponde con el presupuesto de costes finales obtenido para entregarlo a un tercero. Es necesario pues estimar el beneficio que se desea obtener, que en este caso estimaremos un 25%, así como los impuestos que se imputarán sobre dicho beneficio y sobre el proyecto en sí. En la Tabla 6 se muestra el resumen del presupuesto de cliente, con el importe incrementado con el IVA vigente en el momento de la facturación.

| **Resumen por capítulos** | | |
|---|---|---|
| **Capítulo** | **Descripción** | **Total** |
| **1** | **Estudio del problema** | **1.729,90 €** |
| **2** | **Diseño de la solución al problema** | **3.892,27 €** |
| **3** | **Estudio y decisión de las tecnologías** | **108,12 €** |
| **4** | **Implementación del sistema** | **2.162,38 €** |
| **5** | **Elaboración de la documentación** | **2.594,85 €** |
| **6** | **Licencias** | **418,64 €** |
| **7** | **Reuniones** | **756,84 €** |
| **Coste total del proyecto** | | **11.663,00 €** |
| **IVA (21%)** | | **2.449,23 €** |
| **Coste total del proyecto (IVA incluido)** | | **14.112,23 €** |

*Tabla 6. Resumen por capítulos del presupuesto de cliente.*

El coste total del proyecto asciende a 9330,40€, de los cuales 8090,40€ corresponden a costes directos y 1240 a costes indirectos. A dicho coste total se le sumará el porcentaje del 25% de beneficio que se espera obtener, por lo que el presupuesto de cliente ascenderá a 11663€, importe que se verá incrementado con el correspondiente IVA del 21%, resultando la cifra de **14112,23€**.

# CAPÍTULO 2: Job Shop Scheduling Problem

## 2.1 Introducción

Antes de empezar a hablar sobre el problema *Job Shop Scheduling (JSSP)* es necesario ponerse en el contexto de lo que representa la **complejidad computacional**, la cual es una rama de la teoría de la computación que se centra en la clasificación de los problemas computacionales, organizándolos en función de su dificultad inherente en las llamadas **clases de complejidad**, de las que se hablará más adelante. La dificultad de un problema está relacionada con la cantidad significativa de recursos computacionales que requiera su solución.

Los problemas de *scheduling* en general, y en particular el problema JSS, son problemas de optimización combinatoria[1]. Los problemas pertenecientes a la familia del *scheduling* requieren la asignación, a lo largo del tiempo, de una serie de recursos limitados a tareas, y tiene como fin la optimización de una - o más - funciones objetivo, como pueden ser el tiempo consumido por los recursos o la satisfacción del cliente. La función objetivo a la que los investigadores han prestado mayor atención, es sin duda el *makespan*, o tiempo de finalización de la última tarea, buscando siempre su minimización; sin embargo, este criterio no considera otros aspectos con los que se podría mejorar el servicio al cliente, como, por ejemplo, minimizar el retardo de las tareas, criterio conocido como *tardiness*. En general, el propósito del *scheduling* es construir planificaciones que, cumpliendo las restricciones de los problemas, optimicen algún criterio o función objetivo.

### 2.1.1 Notación y conceptos básicos

Generalmente el número de trabajos en un problema de *scheduling* se denota con la letra *n*, mientras que el número de máquinas se denota con la letra *m*. Así, se habla de problemas de dimensión *nxm*. El subíndice *j* se emplea para especificar un trabajo en concreto, y de igual forma el subíndice *i* se emplea para una máquina en concreto. Aunque existen diferentes tipos de problemas de *scheduling*, todos tienen en común alguna o varias de las siguientes características:

- **Tiempo de procesamiento *(processing time)*** $p_{i\,j}$ – unidades de tiempo de un trabajo *j* para ser ejecutado en una máquina *i*.
- **Tiempo de disponibilidad *(release date)*** $r_j$ – tiempo en el que un trabajo *j* está disponible y es devuelto al sistema.
- **Fecha límite *(due date)*** $d_j$ – tiempo en el que un trabajo *j* debería de haber finalizado ya su ejecución. De lo contrario, hablaríamos de un retardo en el trabajo *(tardiness)*.

[1] Optimización combinatoria: rama de la optimización en matemáticas aplicadas y ciencias de la computación en la cual se resuelven instancias de problemas de la vida real mediante una reducción del tamaño efectivo del espacio y explorando de forma eficiente el espacio de búsqueda.

- **Tiempo de inicio *(starting time)*** $st_j$ – tiempo en el que un trabajo *j* puede comenzar a ejecutarse.
- **Peso *(weight)*** $w_j$ – peso (importancia) de un trabajo *j*. Cuanto mayor sea este valor, más prioritario se considerará un trabajo.

Una vez construida la planificación, es posible obtener una serie de datos referente a cada trabajo ya planificado. Los más comunes son:

- **Fecha de completitud *(Completion time)*** ($C_j$) – el momento del tiempo en el que un trabajo *j* termina su ejecución.

- **Tiempo de retardo de los trabajos *(Tardiness of a job)*** ($T_j$) – las unidades de tiempo que un trabajo *j* siguió ejecutándose después de su fecha límite:

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

- **Tiempo de adelanto *(Earliness)*** ($E_j$) – las unidades de tiempo desde la finalización de un trabajo *j* antes de su fecha límite:

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

- **Tiempo de flujo *(Flowtime)*** ($F_j$) – las unidades de tiempo que un trabajo *j* estuvo en el sistema:

$$F_j = C_j - r_j$$

- **Penalización unitaria *(Unit penalty)*** ($U_j$) – un marcador en función de si un trabajo tiene retardo o no:

$$U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases}$$

### 2.1.2 Tipos de problemas de *scheduling* y restricciones

Existen, como ya se mencionó, distintos tipos de problemas de *scheduling*, que se pueden clasificar en función de la notación α|β|γ [35], [36].

- **α** representa el tipo de entorno de los recursos del problema.
- **β** contiene los detalles sobre las diferentes características y restricciones que presenta el entorno del problema.
- **γ** denota la función objetivo a ser minimizada.

El campo α puede representar los siguientes entornos de recursos:

- ***Una sola máquina (1)*** – consiste en un solo recurso o máquina en la cual se ejecutan todos los trabajos.
- ***Máquinas idénticas en paralelo (Pm)*** – existen *m* máquinas idénticas en paralelo que tienen la misma velocidad de ejecución.

- ***Máquinas en paralelo con diferentes velocidades (Qm)*** – existen *m* máquinas en paralelo con diferentes velocidades de ejecución. También se conoce como *uniform machines*.
- ***Máquinas no relacionadas en paralelo (Rm)*** – consiste en *m* máquinas en paralelo. En este entorno, la velocidad de ejecución de una máquina no solo depende de la propia máquina, sino que también depende del trabajo que ejecuta.
- ***Flow shop (Fm)*** – consiste en *m* máquinas en serie. Cada trabajo ha de ser procesado por cada una de esas máquinas. Todos los trabajos han de seguir la misma ruta preestablecida según la cual se procesan en las máquinas.
- ***Flexible flow shop (FFc)*** – es una generalización del *flow shop*, que consiste en que, en vez de *m* máquinas en serie, existen *c* etapas donde cada una está compuesta por un determinado número de máquinas idénticas en paralelo. Todos los trabajos han de ser procesados en cada una de estas etapas siguiendo la misma ruta y, cualquiera de las máquinas en paralelo de la etapa puede ser usada para procesar un trabajo.
- ***Job shop (Jm)*** – consiste en *m* máquinas en serie donde cada trabajo tiene su ruta predeterminada que seguir. Existe una distinción entre aquellos *job shops* en los que un trabajo visita cada máquina una sola vez y aquellos en donde un trabajo puede visitar cada máquina más de una vez.
- ***Flexible job shop (FJc)***- es una generalización del *job shop* y *máquinas idénticas en paralelo*. En vez de *m* máquinas en serie existen *c* etapas donde en cada etapa hay un número determinado de máquinas idénticas en paralelo. Cada trabajo tiene su propia ruta a seguir. Cada trabajo ha de ser procesado en cada etapa, solamente en una máquina, y cualquier máquina de la etapa puede realizar esta ejecución.
- ***Open shop (Om)*** – consiste en *m* máquinas en serie. Cada trabajo ha de ser procesado por cada una de las máquinas.

Algunas de las posibles entradas incluidas en el campo β, que denota restricciones y características, son:

- ***Fechas de disponibilidad ($r_j$)*** – indica que un trabajo no puede comenzar su ejecución antes de su fecha $r_j$. Si no se encuentra especificada, todos los trabajos se encuentran disponibles desde el comienzo y pueden ser ejecutados en cualquier momento.
- ***Interrupción (prmp)*** – el *preemption* (o interrupción) indica que no es necesario mantener un trabajo en una máquina, una vez comience su ejecución, hasta que se complete. El planificador puede interrumpir el procesamiento de un trabajo *(preempt)* en cualquier momento en el tiempo y comenzar la ejecución de otro trabajo diferente en esa misma máquina.
- ***Restricción de precedencia (prec)*** – denota que determinados trabajos han de ser completados para que otros comiencen con su ejecución.
- ***Tiempos de setup ($s_{i\,j\,k}$)*** – denota un tiempo adicional que ocurre cuando el trabajo *k* es procesado en la máquina *i* después de que el trabajo *j* haya terminado su ejecución. Esta restricción representa, por tanto, un tiempo

adicional necesario para preparar la máquina para la ejecución de un nuevo trabajo.

- ***Familias de trabajos (fmls)*** – denota que existen *F* familias de trabajos. Aquellos trabajos de la misma familia pueden ser ejecutados uno después de otro sin la necesidad de ningún *setup* o preparación. En cambio, cuando trabajos de distintas familias se ejecutan uno después de otro, sí que se requiere de un tiempo de *setup*.
- ***Lote de procesamientos (batch(b))*** – denota que las máquinas pueden procesar lotes de *b* trabajos simultáneamente, y que el tiempo de ejecución del lote depende del trabajo con tiempo de procesamiento más largo.
- ***Breakdowns (brkdwn)*** – denota que algunas máquinas pueden no estar disponibles en determinados periodos de tiempo.
- ***Restricción de elección de máquina ($M_j$)*** – denota que no todas las máquinas pueden procesar todos los trabajos, pero sí que hay máquinas que solamente pueden procesar determinados subconjuntos de trabajos.
- ***Permutación (prmu)*** – denota que el orden en el que se procesan los trabajos en la primera máquina ha de ser mantenido durante el procesamiento en las demás máquinas. Esta restricción solo es aplicable en el entorno del *flow shop*.
- ***Recirculación (rcrc)*** – denota que un trabajo puede visitar la misma máquina o etapa más de una vez.
- ***Restricciones en el número de trabajos (nbr)*** – denota la restricción de un máximo número de trabajos.
- ***Restricciones en el número de operaciones de un trabajo ($n_j$)*** – denota la restricción de un máximo número de operaciones de un trabajo.
- ***Restricciones en el tiempo de procesamiento ($p_j$)*** – denota la restricción en cuanto a los valores de los tiempos de procesamiento.
- ***Fechas límite ($d_j$)*** – denota que cada trabajo ha de ser completado antes de su fecha límite.

Por último, algunas de las funciones objetivo consideradas para estos problemas:

- ***Makespan ($C_{max}$)*** – equivalente al tiempo de completitud del último trabajo que abandona el sistema:

$$C_{max} = \max_j(C_j)$$

- ***Maximum flowtime ($F_{max}$)*** – denota el tiempo máximo de fin conseguido por cualquiera de los trabajos:

$$F_{max} = \max_j(F_j)$$

- ***Maximum tardiness ($T_{max}$)*** – denota el máximo retardo conseguido por cualquiera de los trabajos:

$$T_{max} = \max_j(T_j)$$

- ***Total weighted completion time (Cw)*** – denota la suma con los pesos de todos los tiempos de completitud:

$$Cw = \sum_j w_{Cj} C_j$$

- ***Total weighted tardiness (Twt)*** – denota la suma con los pesos de todos los valores de retardo de todos los trabajos:

$$Twt = \sum_j w_{Tj} T_j$$

- ***Total flowtime (Ft)*** – denota la suma de los tiempos de fin de todos los trabajos:

$$Ft = \sum_j F_j$$

- ***Weighted number of tardy jobs (Nwt)*** – denota la suma con los pesos de todos los trabajos con retardo (con *tardines)*:

$$Nwt = \sum_j w_{Tj} U_j$$

- ***Weighted earliness and weighted tardiness (Etwt)*** – denota la suma del total *tardiness* (las unidades de tiempo desde la finalización de un trabajo *j* después de su fecha límite) con pesos y del total *earliness* (las unidades de tiempo desde la finalización de un trabajo *j* antes de su fecha límite) con pesos:

$$Etwt = \sum_j (w_{Ej} E_j + w_{Tj} T_j)$$

## 2.2 Descripción formal del problema

El problema ***Job Shop Scheduling*** *(JSS)* es un problema de optimización dentro del campo de la Inteligencia Artificial e Investigación Operativa. Se trata de un problema clásico además de uno de los más difíciles de resolver ya que, de hecho, cae en la categoría de problemas NP-duro. Este problema corresponde a un tipo de problema de planificación de tareas. Las soluciones al problema reciben el nombre de planificaciones.

El problema *Job Shop Scheduling* consiste en la planificación de un conjunto de N **trabajos** $J = \{J_1, ..., J_n\}$, sobre un conjunto de M recursos o **máquinas** $M = \{M_1, ..., M_m\}$, ambos conjuntos finitos. Cada trabajo $J_i$ consiste en una serie de tareas u **operaciones** $O_i$

$= \{O_{i,1}, O_{i,2}, \ldots, O_{i,M}\}$, donde $O_{i,1}$ es la primera tarea del trabajo $J_i$ y $M$ es el número de tareas del trabajo $J_i$. Cada tarea $O_{ij}$ tiene un **tiempo de procesamiento** o **tiempo de ejecución** $p_{O_{ij}}$ asignado, además de una máquina especificada *a priori* en la que se llevará a cabo su ejecución. A su vez, a cada operación se le asigna un tiempo de inicio $st_{O_{ij}}$ que se ha de determinar y que será, el instante de tiempo más temprano en el que la tarea podrá comenzar a ser ejecutada. Así, la primera operación de cada trabajo comenzará con un tiempo de inicio igual a 0, y, como mínimo, las operaciones que la suceden tendrán como tiempo de inicio el tiempo de procesamiento de la operación anterior (en el mismo trabajo o misma máquina). El tiempo de inicio de las tareas se actualizará a medida que sus predecesoras en el trabajo o en la máquina sean planificadas.

Estas tareas u operaciones deben ser realizadas de forma secuencial, empleando cada tarea en un único recurso sin expulsión durante un tiempo de procesamiento y cumpliendo una serie de **restricciones**. Esto quiere decir que para resolver este problema se han de encontrar soluciones que cumplan todas las restricciones (soluciones factibles).

Hay tres tipos de restricciones:

- **Secuencial o de precedencia.** Las tareas de los trabajos han de realizarse de forma secuencial y en el orden en que son enunciadas. La tarea $O_{ij}$ ha de procesarse antes que la tarea $O_{i(j+1)}$, es decir:

$$st_{O_{ij}} + p_{O_{ij}} \leq st_{O_{i(j+1)}}$$

- **De capacidad.** Dos tareas que requieren la misma máquina no pueden solaparse en el tiempo, es decir, una máquina no puede procesar más de una tarea al mismo tiempo:

$$st_{O_a} + p_{O_a} \leq st_{O_b} \bigvee st_{O_b} + p_{O_b} \leq st_{O_a} \qquad si\ M_a = M_b$$

- **No interrupción o no expulsión.** Una vez que una tarea es planificada – se le asigna un tiempo de inicio – se procesa de manera ininterrumpida.

Cada trabajo consta de su propia ruta, predeterminada por una secuencia de máquinas que tiene que seguir en cada tarea que lo componga. Es decir, un trabajo consiste en una secuencia de una o más tareas en un determinado orden especificado, y a su vez, cada tarea tiene un tiempo de ejecución en una máquina concreta.

Además de las tres restricciones descritas anteriormente, hay una serie de consideraciones a tener en cuenta:

- Los tiempos de procesamiento de las tareas u operaciones son conocidos *a priori*.

- Todos los trabajos están disponibles para ser procesados en el instante t = 0.
- Cada trabajo representa una entidad, por lo que no pueden procesarse dos tareas de un mismo trabajo de manera simultánea.
- Cada trabajo incluye una y solo una tarea en cada máquina o recurso, por lo que todos los trabajos contienen una cantidad de tareas que no supera al número de máquinas existente.
- Así como existe una relación de precedencia entre las tareas de los trabajos, no existe esta relación de precedencia entre los trabajos en sí (pueden procesarse en distinto orden en el que se enuncian).
- Todas las máquinas están libres en el instante t = 0.
- Un trabajo se considera finalizado o planificado cuando todas sus tareas hayan sido planificadas.

El problema clásico del *job shop* tiene como objetivo la minimización de la función objetivo ***makespan***, que se representa como $J||C_{max}$ y representa el tiempo de finalización de la última tarea planificada, es decir, la duración en el tiempo desde el inicio hasta el final. Equivale a minimizar tiempos muertos o maximizar la utilización de las máquinas, y suele ser este el objetivo más habitual a minimizar, debido a su amplio uso tanto en el ámbito académico como en el industrial.

Además del *makespan*, también existen otras funciones objetivo como lo son el **flujo total** y el ***tardiness***.

La función objetivo del flujo total se representa como $J||\Sigma Ci$, donde Ci es el tiempo de fin del trabajo Ji, y se refiere a la suma de los tiempos de fin de todos los trabajos.

La función objetivo del *tardiness* o tiempo de retardo de los trabajos se representa como $J||\Sigma T_i$, donde $T_i$ es el *tardiness*, y se refiere al tiempo de procesamiento que un trabajo $J_i$ ha empleado después de su fecha límite *(due date)*, es decir, a si el tiempo de fin de un trabajo excede el *due date* establecido para él. Si esta diferencia fuese negativa, es decir, si no se hubiese ejecutado tras su fecha límite, significaría que el *tardiness* es de 0. El *tardiness* de un trabajo se calcula como $T_i = \max (C_i - d_i, 0)$, donde $C_i$ y $d_i$ son respectivamente el tiempo de finalización y la fecha límite o *due date* para el trabajo $J_i$.

## 2.3 Complejidad del problema

Dentro del campo de la complejidad computacional se pueden hallar varios tipos de problemas que se clasifican en conjuntos o clases de complejidad. Una clase de complejidad es un conjunto de problemas de decisión de complejidad relacionada. Un problema de decisión es aquel en que las soluciones posibles son *sí/no*. Por ejemplo, un problema de decisión sería un problema que cuestionase si existe una planificación que obtuviese un beneficio K, mientras que un problema que plantee cuál es la mejor planificación, no lo es. En la teoría de la complejidad se tienen en consideración muchos problemas de decisión, ya que permiten obtener una simple y precisa definición del problema planteado, y, por tanto, muchos problemas pueden convertirse en un problema de decisión equivalente.

Aquellos problemas a los que se está generalmente más acostumbrado a tratar son los problemas de la clase P, NP, NP-completos y NP-duros. Aquí aparece el problema de *¿P=NP?*, del que partirían dos formas diferentes de modelar las clases mencionadas:

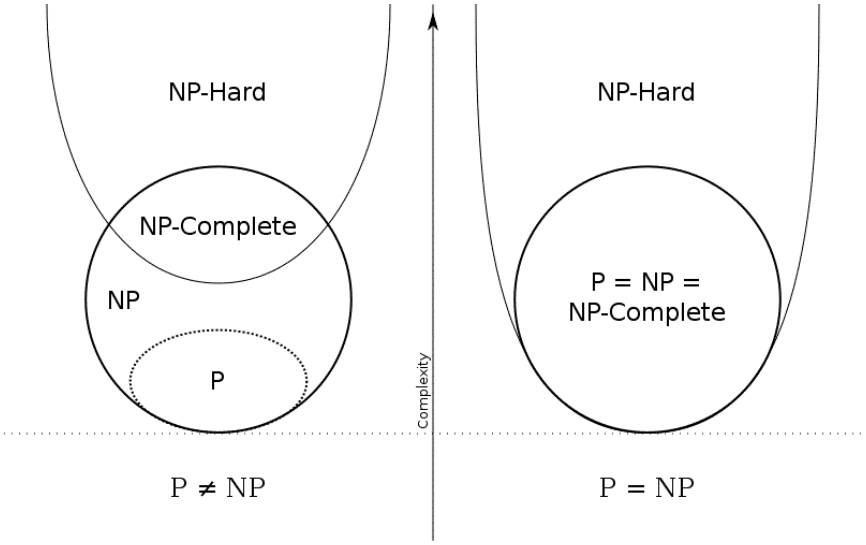



*Figura 1. A la izquierda, el modelo si P≠NP. A la derecha, el modelo si P=NP [46].*

Los problemas de clase **P** son aquellos problemas de decisión que pueden ser resueltos en un tiempo polinómico calculado a partir de la entrada por una máquina de Turing determinista. Paralelamente, los problemas de clase **NP** son aquellos problemas de decisión que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista; por tanto, en todo problema NP resuelto existirá un certificado que permita verificar la respuesta en tiempo polinomial (determinista). Simplemente se necesita percibir que cualquier algoritmo determinista puede ser utilizado en la etapa de verificación de un algoritmo no determinista. La relación entre estas clases, por tanto, es fundamental. De una forma demostrada sí que se puede admitir que, como mínimo, P resulta un subconjunto de NP.

De cualquier modo, para este trabajo no se requiere una rigurosa definición matemática de estos conceptos, tan solo el identificar tanto P como NP.

Paralelamente, también podemos encontrar el tipo de problemas NP-Completos. Aquí, la palabra “completo” hace referencia a la propiedad de poder simular todo en la misma clase de complejidad. Es decir, cada entrada del problema debería asociarse a un conjunto de soluciones de tiempo polinomial, cuya validación debería de poder ser probada rápidamente (en tiempo polinomial). Un problema de decisión es NP-Completo si pertenece a la clase NP y al menos es tan difícil como cualquier otro problema en la clase NP. Aunque no se ha podido demostrar, se puede decir que los problemas de decisión NP-completos son los problemas más difíciles de la clase NP y muy probablemente no formen parte de la clase de complejidad P.

Se dice que la clase que se corresponde con la NP-completa, en el caso de los problemas de optimización, es la clase de problemas NP-duros. La clase de problemas NP-duros contiene aquellos problemas (de decisión o no) para los que, si existiese un algoritmo que los resolviese en tiempo polinomial se verificaría que P = NP. Para demostrar que un problema es NP-duro se utiliza la técnica de la reducibilidad: un problema "X" es NP-duro si hay un problema "Y" NP-completo, que puede ser reducido a "X" en tiempo polinómico. Aplicando esta técnica se encuentran problemas de optimización que son NP-duros, y más en general, ciertos problemas de *scheduling* como el *JSS* son NP-duros.

Aunque se haya podido resolver de manera óptima, bajo ciertas características y circunstancias, instancias concretas del problema *JSS* [37] [38] [39] [40], a medida que el tamaño del problema aumenta, el espacio de búsqueda crece exponencialmente, pues hay $(n!)^{m}$ soluciones posibles. Así, por ejemplo, para un problema de 20x10 (200 tareas), el problema tiene en principio 7.2651×10183 posibles planificaciones, lo que excedería la edad estimada del universo en microsegundos. Aun cuando muchas de estas planificaciones no sean factibles, debido a que incumplan restricciones secuenciales (conjuntivas), restricciones de capacidad (disyuntivas) o ambas, enumerar todas las planificaciones factibles para identificar la óptima es impracticable.

Como resultado de esta intratabilidad, se considera que el problema *JSS* pertenece a la clase de problemas NP que son NP-duros [18] [19]. La clase NP contiene problemas para los que existe un algoritmo no determinista polinomial que los resuelve (pueden resolverse en tiempo polinómico por una MT no determinista), lo que significa que no puede resolverse una instancia arbitraria en tiempo polinomial a no ser que la clase P sea igual a la clase NP [41] [42] [19].

## 2.4 Representación del problema

En todos los problemas del mundo real es necesario encontrar una forma de representarlos que permita abordar su resolución de manera eficiente. Existen varias formas de representar el problema *JSS*, aunque la más extendida es la representación propuesta por Roy y Sussmann en 1964 [2]. El uso del modelo del grafo disyuntivo sirve para modelar diferentes restricciones y características de los problemas. Esta representación garantiza mayor claridad a la hora de representar las soluciones que se pueden obtener. El grafo disyuntivo dirigido **G = (V, A ∪ E)** contiene los siguientes elementos:

- **V**: Conjunto de nodos. Representan las tareas del problema, a excepción de los nodos ficticios "inicio" *(source)* (*O*), que está conectado a través de un arco conjuntivo a la primera operación de cada trabajo y "fin" *(end)* (*), al cual se conecta la última tarea de cada trabajo mediante un arco conjuntivo, y representa el tiempo de finalización del mismo. Ambos se tratan de nodos con tiempo de procesamiento 0.

- **A**: Conjunto de arcos conjuntivos. Conectan dos tareas consecutivas de un trabajo, estableciendo así las restricciones de precedencia o (también denominada de secuencialidad).

- **E**: Conjunto de arcos disyuntivos. Conectan dos tareas que emplean el mismo recurso, es decir, que son procesadas en la misma máquina. Garantiza así la restricción de capacidad, es decir, que esas operaciones no se pueden ejecutar simultáneamente.

En el grafo todos los arcos de los conjuntos A y E están etiquetados con un peso que representa el tiempo de procesamiento de la tarea de la que parte el arco.

La Figura 2 muestra un ejemplo de un grafo disyuntivo dirigido para una instancia del problema *JSS* con 3 trabajos y 3 máquinas (3x3) descrita en la Tabla 7. Aquí, cada nodo representaría una tarea de un trabajo, y constan de un identificador. Mediante el uso de color se han diferenciado las máquinas en las que se procesa cada tarea (por ejemplo, las verdes serán procesadas en la máquina o recurso 1) y, los arcos conjuntivos (en líneas continuas) están etiquetados con la duración de la tarea de la que parten y representan las restricciones de precedencia, mientras que los arcos disyuntivos (en líneas discontinuas) unen las tareas que se ejecutan en la misma máquina y representan las restricciones de capacidad.

| Trabajo | Ruta | | | $p_{ij}$ | | |
|---|---|---|---|---|---|---|
| | $M_1$ | $M_2$ | $M_3$ | $M_1$ | $M_2$ | $M_3$ |
| $J_1$ | 1 | 2 | 3 | 4 | 3 | 1 |
| $J_2$ | 1 | 3 | 2 | 2 | 3 | 3 |
| $J_3$ | 2 | 1 | 3 | 3 | 3 | 2 |

*Tabla 7. Instancia del JSSP con un ejemplo de tres trabajos con 3 tareas cada uno y 3 máquinas para procesarlas.*

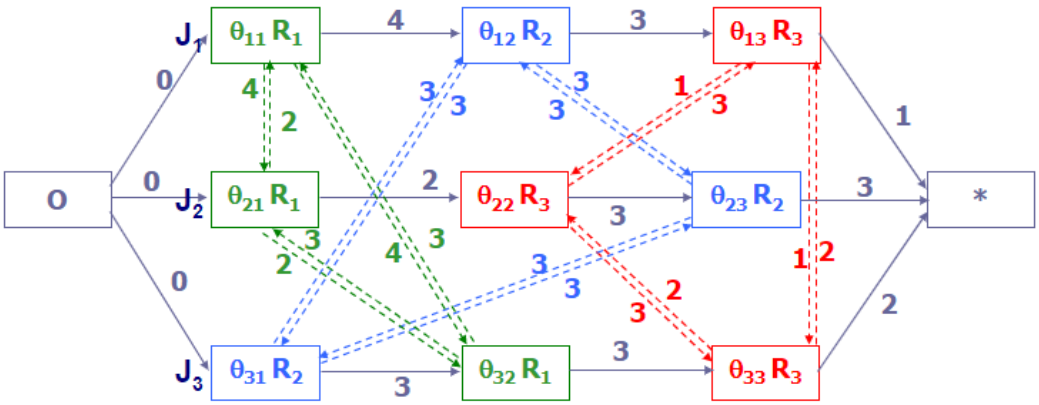



*Figura 2. Grafo de restricciones del JSSP con un ejemplo de tres trabajos con 3 tareas cada uno y 3 máquinas para procesarlas [43].*

## 2.5 Representación de soluciones

La representación de una solución factible, como se propone en este trabajo, se suele realizar mediante un subgrafo del grafo explicado anteriormente o bien mediante un diagrama de Gantt.

En lo que a la representación mediante un grafo respecta, una planificación factible se representaría como un subgrafo sin ciclos $G_s$ de G, Gs = (V, A ∪ H), donde $H = \cup_{i=1,\dots,M} H_i$ , siendo $H_i$ un camino hamiltoniano de $E_i$, si para cada par de tareas *u* y *v* que requieren el recurso $R_i$, uno y solo uno de los pares (*u*, *v*) o (*v*, *u*) pertenece a $H_i$. Así, si (*u*, *v*) ∈ $H_i$, entonces la tarea *u* se procesa antes que *v* en la solución que representa $G_s$. Por lo tanto, encontrar una solución se puede resumir en encontrar un camino hamiltoniano o planificación parcial, que será equivalente a un grafo solución sin ciclos $G_s$.

Según la función objetivo a tratar, el coste de la solución del problema se representará en el grafo de distinta forma. Por ejemplo, cuando la función objetivo es el makespan, el coste del camino crítico será el makespan de la planificación. En un grafo solución, un camino crítico es la ruta más larga desde el nodo inicial hasta el nodo final. Un subconjunto maximal[2] de tareas consecutivas en el camino crítico que se ejecutan sobre la misma máquina recibe el nombre de "bloque crítico" [43].

En la Figura 3, se muestra el grafo de soluciones de una planificación factible para el problema de la Figura 2, donde los arcos en trazo grueso representan un camino crítico cuyo coste, es decir, el *makespan*, es de 12.

Asimismo, también se puede realizar esta representación mediante el empleo de diagramas de Gantt, los cuales pueden estar orientados al recurso o al trabajo. En la Figura 4, se muestra la misma planificación factible que el grafo de la Figura 3, en forma de diagramas de Gantt orientados al trabajo y a los recursos, respectivamente.

[2] Subconjunto maximal: Se dice que A es un subconjunto maximal de B si A ⊂ B y no existe ningún C tal que A ⊂ C y C ⊂ B.

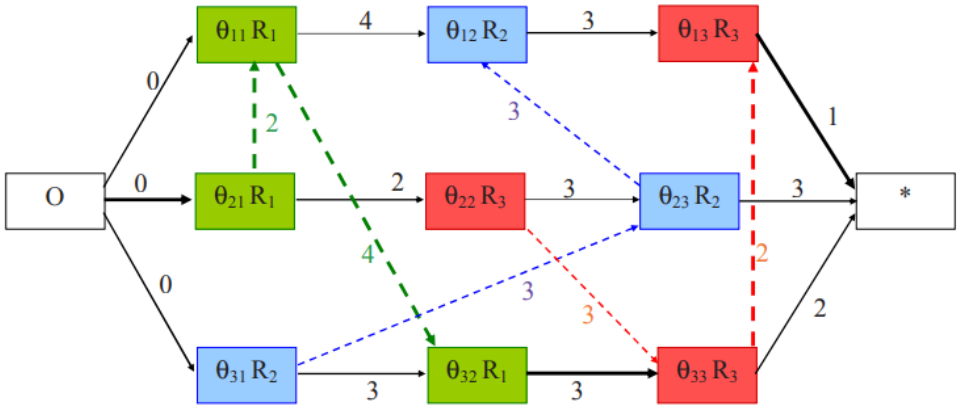



*Figura 3. Grafo de soluciones de una planificación factible para el problema de la Figura 2 [43].*

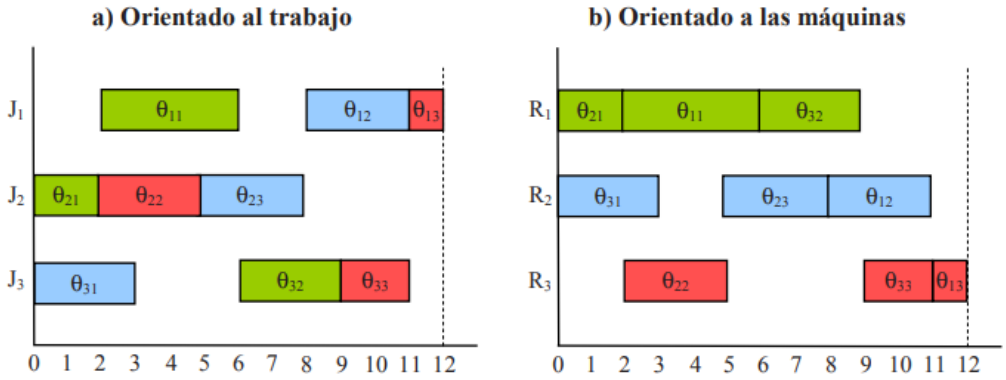



*Figura 4. Diagramas de Gantt de una planificación factible para el problema de la Figura 2 [43].*

## 2.6 Espacio de búsqueda y tipos de planificaciones

Dentro del campo de optimización, se define como **espacio de búsqueda** al dominio de la función a ser optimizada, es decir, el conjunto de soluciones candidatas a un problema [47]. El conjunto de soluciones factibles de un problema representa todas aquellas soluciones que cumplen con las restricciones del mismo. Estas soluciones se clasifican en los tipos de planificaciones, representadas en la Figura 5:

- **Planificaciones semiactivas**. Una planificación semiactiva es aquella en la que para adelantar el tiempo de inicio de una tarea es necesario modificar el orden relativo de al menos dos tareas.

- **Planificaciones activas**. Una planificación activa es aquella en la que, para adelantar el tiempo de inicio de una tarea, al menos otra debe ser retrasada, para obtener una planificación válida.
- **Planificaciones densas**. O *non-delay*. Una planificación es densa si nunca se da la situación de que una máquina esté desocupada en el mismo instante en el que una tarea puede ser procesada en dicha máquina. Las planificaciones densas son un subconjunto de las planificaciones activas, las cuales son a su vez son un subconjunto de las planificaciones semiactivas.
- **Planificaciones óptimas**. Una planificación óptima es aquella que tiene el mínimo coste posible.

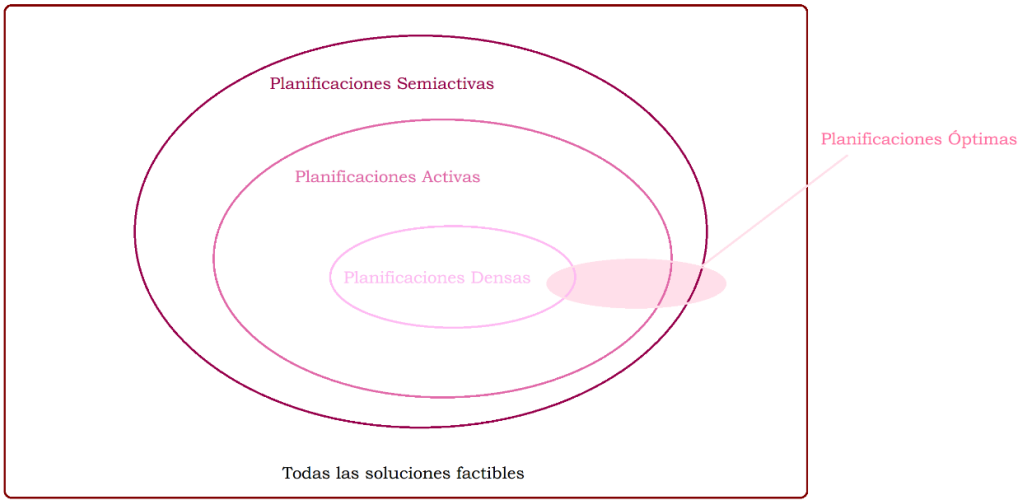



*Figura 5. Espacio de búsqueda con las distintas planificaciones existentes.*

**Planificaciones Densas ⊂ Planificaciones Activas ⊂ Planificaciones Semiactivas ⊂ Planificaciones Factibles**

El espacio de búsqueda formado por todas las soluciones factibles implica un espacio de búsqueda extremadamente grande. Es por ello por lo que se han de explorar otras alternativas que consideren un espacio de búsqueda menor, que a su vez garantice contener al menos una solución óptima. En este trabajo se considerará el espacio de búsqueda de las planificaciones activas. El interés del empleo de este tipo de planificación radica en que constituye un espacio completo más reducido – es decir, contiene al menos una solución óptima – aunque, a pesar de ello, continúa siendo un espacio demasiado grande para muchos problemas.

La experiencia muestra que, en el caso del *makespan*, el coste es mucho mayor para las planificaciones semiactivas que para las activas, y estas, a su vez, tienen un coste menor que las densas. No obstante, las únicas que garantizan encontrar al menos una

solución óptima son las semiactivas y las activas. Se puede ver un ejemplo de un diagrama de una planificación semiactiva y activa en la Figura 6.

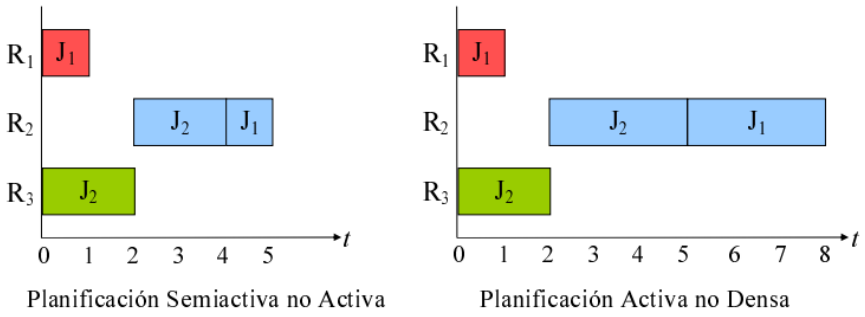



*Figura 6. Ejemplos de planificaciones semiactiva y activa. [43]*

## 2.7 El algoritmo G&T

Antes de introducir el algoritmo G&T, es conveniente recordar que el problema *JSS* puede ser abordado por estrategias exactas y aproximadas. Sin embargo, dada su complejidad, problema NP-duro, son los algoritmos aproximados los más utilizados en su resolución. Estos, aunque no garantizan una solución óptima encuentran buenas soluciones. Entre los algoritmos aproximados, se encuentra el algoritmo G&T [1], uno de los más utilizados en la literatura en el ámbito de los problemas de *scheduling*.

El algoritmo G&T, propuesto por B. Giffler y G. L. Thompson en el año 1960 [1], es un algoritmo voraz, que produce planificaciones activas en N*M pasos y que puede ser guiado por reglas heurísticas. Se trata, de hecho, de la estrategia más apropiada para obtener planificaciones activas. Este algoritmo, en cada iteración, analiza un conjunto de tareas candidatas a ser planificadas a continuación y realiza una elección no determinista sobre el mismo, de modo que, cualquiera que sea la tarea elegida, se garantiza que la planificación resultante es activa.

La Figura 7 muestra un ejemplo para un problema de 4 trabajos y 4 máquinas, en el que para una situación intermedia hay 2 opciones posibles. La tarea $\theta_{32}$ es la que tiene menor tiempo de fin entre las que pueden ser planificadas. La tarea $\theta_{11}$ requiere la misma máquina que $\theta_{32}$ y puede empezar antes del tiempo de fin de la tarea $\theta_{32}$. Por tanto, el algoritmo G&T contemplará en el conjunto B las tareas que pueden empezar en el intervalo [T, C] y que requieren la misma máquina que la tarea $\theta_{32}$, es decir la propia tarea $\theta_{32}$ y la $\theta_{11}$. A continuación, estas son las dos elecciones posibles que puede hacer el algoritmo G&T.

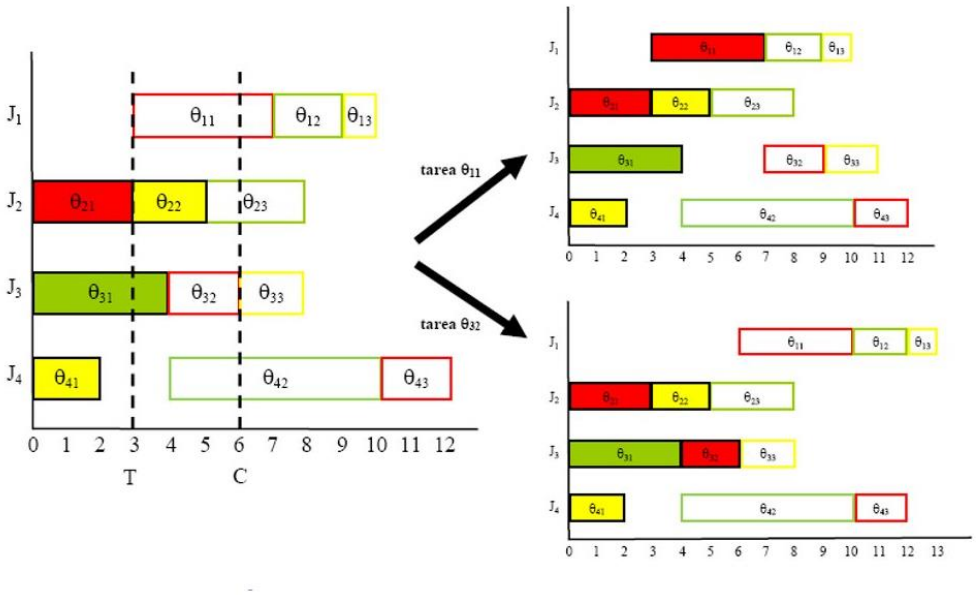



*Figura 7. Elección de las tareas a planificar por el Algoritmo G&T. [43]*

En problemas de gran tamaño el espacio de búsqueda formado por todas las planificaciones activas es excesivamente amplio, por lo que la búsqueda se debe restringir a una parte de este espacio por razones de eficiencia, con la consiguiente pérdida de la completitud (ya no se garantiza obtener una solución óptima). En este sentido, una posibilidad es emplear una variante del algoritmo G&T denominada G&T Híbrido [44] [45].

**Algoritmo 2.1** Algoritmo G&T

Input: *Instancia*

Output: *Planificación*

A: conjunto con las primeras tareas sin planificar de cada trabajo (Job)

*while* A != ∅ *do*:

Determina la operación $\theta' \in A$ con menor tiempo de fin (C):

$$C = st_{\theta'} + p_{\theta'} \leq st_{\theta} + p_{\theta}, \forall \theta \in A$$

Selecciona las operaciones que pueden empezar antes en la misma máquina y que puedan comenzar antes que que C (tiempo de fin de θ'), y que se ejecutan en la misma en la misma máquina que θ'. Con ellas construye el conjunto B:

$$B = \{\theta \in A, M_{\theta} = M_{\theta'} \wedge st_{\theta} < C_{\theta'}\}$$

Selecciona la operación θ'' del conjunto B con algún criterio, por ejemplo, empleando una regla de prioridad, y la planifica.

Borra la operación planificada θ'' de A y añade a A la operación sucesora de θ'', en caso de que exista:

*end while*

*Código 1. Algoritmo G&T (pseudocódigo).*

## 2.8 Reglas de Prioridad

Las **reglas de prioridad** son heurísticos que ayudan en la toma de decisiones en lo relativo a la selección de la siguiente tarea a planificar en el proceso de construcción de una solución, llamada planificación. Son posiblemente las reglas heurísticas aplicadas con mayor frecuencia con el fin de resolver problemas de *scheduling,* debido a su fácil implementación y a su baja complejidad temporal. Estas reglas permiten establecer una prioridad para cada una de las tareas aún no planificadas, lo que permite a la estrategia de construcción de una solución elegir entre ellas. Cada vez que una máquina queda libre, una regla selecciona la siguiente tarea con mayor prioridad para ser planificada. Por ejemplo, en el caso del algoritmo G&T descrito en la sección anterior, las reglas de prioridad se emplearían para elegir, en cada paso del algoritmo, la siguiente tarea a planificar de entre las que se encuentran en el conjunto B.

Generalmente una regla de prioridad se expresa mediante una función de prioridad, encargada de calcular el valor de dicha prioridad para las tareas; y necesita de una mecanismo para construir planificaciones, en el que se emplean dichas reglas como proceso de discriminación entre las tareas, en función de la prioridad asignada por la regla. En el caso de este trabajo, este mecanismo es el algoritmo G&T.

Se pueden distinguir claras ventajas en cuanto al uso de reglas de prioridad en problemas de planificación, como su rápida velocidad de ejecución en comparación con otros algoritmos, su fácil implementación ya mencionada, la amplia aplicabilidad en entornos dinámicos, y la considerable adaptabilidad en cuanto a cambios en las condiciones de un entorno de planificación. Por otro lado, presentan una notable desventaja en cuanto a la obtención de la solución óptima, la cual no puede ser garantizada, al tratarse de un mecanismo que toma decisiones locales y no tiene en cuenta la globalidad del problema.

Las reglas implementadas en este trabajo son las siguientes:

- **Shortest Processing Time (SPT)**. Dado un conjunto de tareas, selecciona aquella con menor tiempo de procesamiento. Calcula, por tanto, la prioridad de la tarea de la forma:

$$\pi_{i,j} = \frac{1}{p_{i,j}}$$

- **Longest Processing Time (LPT)**. Paralelamente al SPT, selecciona la tarea con mayor tiempo de procesamiento. Calcula la prioridad de la forma:

$$\pi_{i,j} = p_{i,j}$$

- **Earliest Due Date (EDD)**. Determina la prioridad de una tarea de la forma:

$$\pi_j = \frac{1}{d_j}$$

- **Apparent Tardiness Cost (ATC)**. Determina la prioridad de una tarea de la forma:

$$\pi_{i,j} = \frac{w_j}{p_{ij}} \exp\left(-\frac{\max(d_j - p_{i,j} - \gamma(\alpha), 0)}{g\overline{p}}\right)$$

En donde $\gamma(\alpha)$ representa el primer instante con capacidad disponible para planificar, al menos, una tarea, es decir, el instante de tiempo en el que se planifica la tarea; $\overline{p}$ indica el tiempo de procesamiento medio de todas las tareas sin planificar hasta ese momento, y $g$ indica que se trata de un parámetro de escala introducido por el usuario.

Se trata de una regla que emplea una función exponencial en lugar de una linear, y que usa el tiempo medio de procesamiento en lugar del tiempo de procesamiento exacto del trabajo. Esta regla de prioridad es una regla de prioridad compuesta que se basa en la combinación de dos reglas básicas: la WSPT (*Weighted Shortest Processing Time* o procesamiento ponderado más corto) y la MS (*Minimum Slack* o mínima holgura), con la asignación de un peso determinado a cada trabajo y con el objetivo de minimizar la suma de los retrasos ponderados de cada trabajo. Al tratarse la ATC de la combinación de ambas reglas, no se realizó la implementación particular de cada una de ellas y se decidió

experimentar con la ATC y distintos valores del parámetro *g*. Por ejemplo, reglas como SPT y EDD son dos reglas básicas que solamente utilizan una variable del dominio para guiar la búsqueda, mientras que ATC es una regla más compleja que, teniendo en cuenta más información sobre el dominio, busca un equilibrio entre SPT y EDD.

Aunque en el estado del arte podemos encontrar diversas reglas, se han elegido estas reglas clásicas por ser de las más empleadas en la literatura para las funciones objetivo abordadas en este trabajo.

# CAPÍTULO 3: Análisis del sistema

## 3.1 Determinación del alcance del sistema

Las funcionalidades a obtener se desprenden de lo ya descrito en el apartado 1.8 Alcance del proyecto, que parten del objetivo de crear un programa capaz de resolver el problema del *job shop scheduling* a partir de unos ficheros de datos con la información de las instancias del problema a resolver. Las funcionalidades del sistema serán las siguientes:

- Carga de instancias (ficheros de datos).
- Elección de regla(s) de prioridad.
- Ejecución del programa.
- Generación de un fichero con la solución.

El sistema pretende asistir a usuarios con un conocimiento base en el estudio y análisis de algoritmos relacionados con el problema JSP y, a pesar de haber sido implementados los objetivos descritos en el trabajo, existen otras funcionalidades que se incluyen en este trabajo como posibles ampliaciones del mismo.

## 3.2 Obtención de los requisitos del sistema

### 3.2.1 Requisitos Funcionales

RF 1. La aplicación podrá resolver el *Job Shop Scheduling Problem* mediante el algoritmo voraz G&T y reglas de prioridad.

RF 1.1. El usuario podrá probar una o varias instancias.

RF 1.1.1. La/s instancia/s a probar tienen que ser ficheros de texto.

RF 1.1.1.1. En el caso de que se trate el problema de minimización del *makespan*, independientemente de que se aborden instancias extendidas o no, como mínimo se requerirá de los siguientes datos:

RF 1.1.1.1.1. Número de trabajos.

RF 1.1.1.1.2. Número de máquinas.

RF 1.1.1.1.3. Tiempos de procesamiento de cada tarea.

RF 1.1.1.1.4. Máquinas asignadas a cada tarea, que serán números desde 1 hasta el especificado en RF 1.1.1.1.2.

RF 1.1.1.2. En el caso de que se trate el problema de minimización del *tardiness*, además de los datos indicados en RF 1.1.1.1. se necesitará:

RF 1.1.1.2.1. Peso de cada trabajo.

RF 1.1.1.2.2. Fecha límite de cada trabajo.

RF 1.1.1.3. Los ficheros de texto tendrán que tener la extensión ".txt".

RF 1.1.2. Si se trata de varias instancias, deberán estar almacenadas en un directorio.

RF 1.1.3. El usuario le proporcionará al programa la ruta de la/s instancia/s.

RF 1.1.3.1. En caso de no existir la ruta proporcionada se mostrará un error al usuario.

RF 1.1.4. El usuario le proporcionará al programa la función objetivo a analizar:

RF 1.1.4.1. *Makespan*.

RF 1.1.4.2. *Tardiness*.

RF 1.2. El usuario podrá probar una de las siguientes reglas de prioridad:

RF 1.2.1. SPT *(Shortest Processing Time)*

RF 1.2.2. LPT *(Longest Processing Time)*

RF 1.2.3. EDD *(Earliest Due Date)*

RF 1.2.4. ATC *(Apparent Tardiness Cost)*

RF 1.3. El usuario podrá probar todas las reglas de prioridad a la vez especificadas en RF 1.2. para la instancia a resolver.

RF 1.4. La instancia a probar deberá poder emplearse con la regla de prioridad elegida.

RF 1.4.1. Una instancia no extendida no podrá ser ejecutada por una regla cuya finalidad sea calcular el *tardiness*.

RF 1.4.1.1. En caso de intentar ejecutar una regla de prioridad con una instancia que carezca de los datos necesarios para su ejecución se mostrará un error.

RF 1.4.2. Una instancia extendida podrá ser ejecutada por cualquier regla.

RF 2. La aplicación exportará un archivo Excel con los resultados de cada instancia.

RF 2.1. Será un archivo de extensión ".xslx".

RF 2.2. Contendrá una instancia por fila y un resultado por cada columna (una por cada regla de prioridad ejecutada).

RF 2.2.1. En caso de que la función objetivo fuese el *makespan* cada celda contendrá el tiempo de completitud ($C_j$) de la instancia.

RF 2.2.2. En caso de que la función objetivo fuese el *tardiness* cada celda será la suma del tiempo de *tardiness* de cada trabajo, respecto a la fecha límite de cada trabajo.

### 3.2.2 Requisitos no funcionales

RNF 1. La aplicación deberá estar correctamente empaquetada para su ejecución.

RNF 1.1. Deberá tratarse de un archivo con extensión ".jar".

RNF 1.2. La aplicación será accesible mediante un entorno Java.

RNF 2. La aplicación deberá seguir las convenciones de código Java de Oracle [24].

RNF 3. Los ficheros de salida tendrán un formato de extensión ".xlsx".

## 3.3 Identificación de subsistemas

### 3.3.1. Descripción de los subsistemas

Este apartado del análisis está dedicado a los diferentes módulos en los que se divide el sistema, para realizar los casos de uso pertinentes. En base a los requisitos establecidos se pueden identificar los siguientes subsistemas:

- **Gestión de lectura de los ficheros y creación de instancias.** Tiene dos funciones principales:
  - Gestionar lo relacionado con la lectura de datos de los ficheros introducidos por el usuario.
  - Cargar las instancias a raíz de estos ficheros. Genera una instancia por fichero de texto y, a su vez, crea las pertinentes máquinas y trabajos con sus operaciones.
- **Generación del grafo y ejecución del algoritmo G&T.** Ejecuta el algoritmo G&T haciendo uso de un grafo de restricciones que crea a raíz de la instancia que se esté ejecutando.
- **Ejecución de las reglas de prioridad.** Ejecuta la/s regla/s de prioridad introducida/s por el usuario.
- **Gestión de escritura de ficheros de salida.** Escribe los datos resultantes de la ejecución en un fichero Excel y lo almacena en una carpeta "out".

### 3.3.2. Descripción de los Interfaces entre subsistemas

La estructura principal de este sistema se divide en capas, por lo que se podrá seguir una clasificación acorde a la finalidad de cada módulo, en la que son esenciales los siguientes subsistemas:

- Subsistema encargado de lo que en otro contexto sería equivalente a la interfaz gráfica, pero con una **consola de línea de comandos**. Aquí se lleva a cabo toda la funcionalidad relacionada con la interacción del usuario con el sistema. Por ende, lo necesario en este subsistema sería una clase encargada de recoger el *input* del usuario y que se encargase de llamar a los métodos pertinentes del *Parser* para la lectura del fichero, así como de los necesarios para invocar al algoritmo G&T para que lleve a cabo la ejecución de la planificación y la generación del *output*.

- Subsistema que contiene toda la **lógica** de negocio del sistema. Contendría distintos paquetes dedicados a diferentes funcionalidades:
  - Contiene las clases necesarias para representar las instancias a planificar.
  - Clase *GTAlgorithm* y las distintas reglas de prioridad implementadas, que heredan de una interfaz *Rule*.
  - Escritura de ficheros con los datos generados.
  - Parser para la lectura de ficheros según una ruta proporcionada por el usuario.

El sistema aquí implementado no requiere de base de datos, por lo que no es necesaria la implementación de una capa de persistencia.

## 3.4 Identificación de actores del sistema

Los roles que interactúan con el sistema aquí implementado son de naturaleza bastante sencilla debido a que no existe un sistema de *login* o registro de usuarios ni requiere de jerarquías de usuarios (como administrador, alumno…). Por lo que solamente existe la figura de **usuario**, del cual se podría hacer la distinción según el conocimiento sobre el tema del mismo:

- **Usuario sin o con escasos conocimientos en problemas de *scheduling.*** Aquella persona que puede hacer uso del sistema con el objeto de estudiar el problema *job shop scheduling* para analizar los resultados de un conjunto de instancias, o incluso de una instancia sola, en función de las distintas reglas de prioridad implementadas.
- **Usuario experimentado en problemas de *scheduling*.** Aquella persona que hace uso del sistema como una herramienta con el objeto de analizar instancias, probablemente de manera masiva, y con el fin de realizar comparativas, gráficas e investigar sobre el potencial de emplear el prototipo implementado en este tipo de problema.

Esta diferencia entre usuarios en función de su conocimiento, respecto al contexto del proyecto, no resulta relevante a la hora de analizar su interacción con el sistema, donde, como se explicó, se tendrá en consideración un solo actor (“Usuario”); mas al analizar la usabilidad del sistema, así como los casos de uso del mismo, sí que resulta un dato interesante a tener en cuenta.

## 3.5 Casos de uso y escenarios

### *Especificación de casos de uso*

A continuación, se exponen los distintos casos de uso para el actor descrito anteriormente “Usuario”. Primero, en el Diagrama 2, se muestran los casos de uso principales integrados en el sistema.

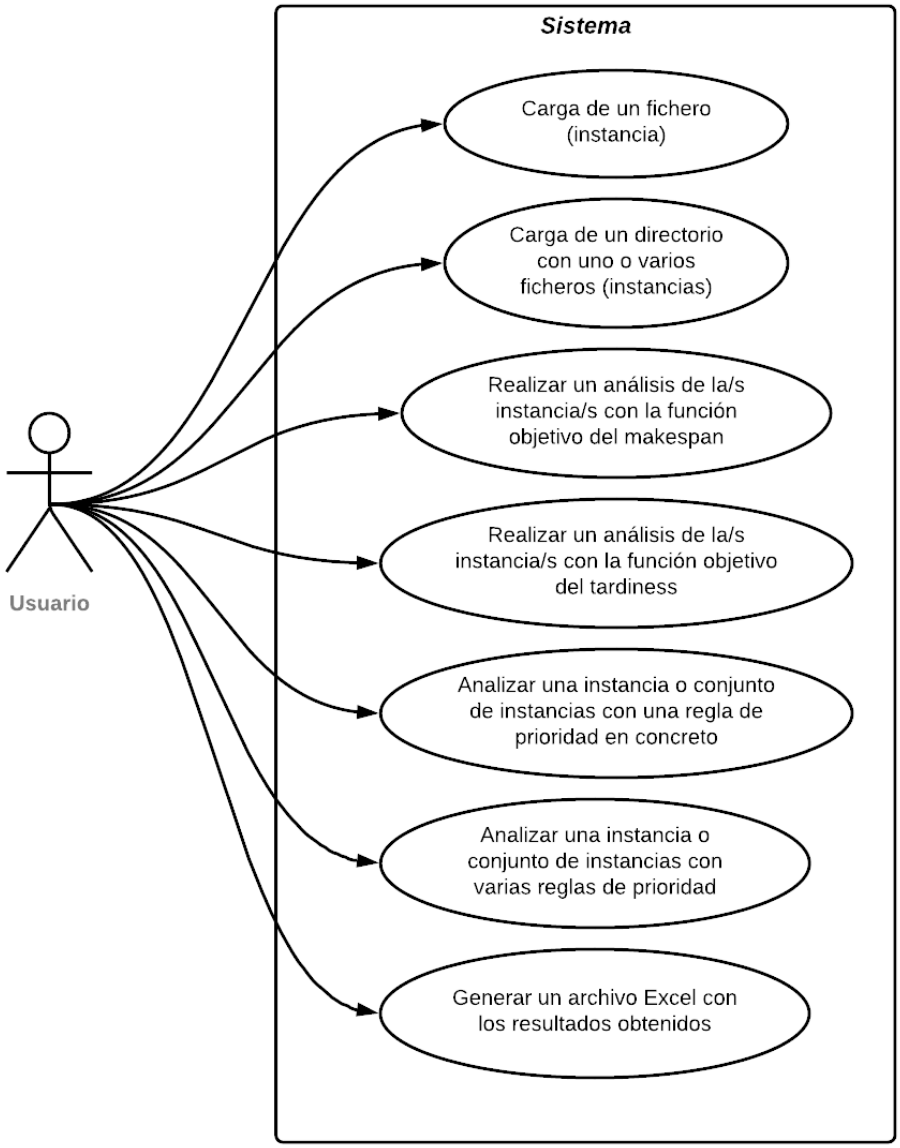



*Diagrama 2. Diagrama de casos de uso principales del sistema.*

A continuación, en el Diagrama 3 y en el Diagrama 4 se muestran respectivamente los casos de uso en dos situaciones distintas: en el primer caso, una instancia o un grupo de instancias no extendidas – por lo que la función objetivo a tener en cuenta ha de ser el *makespan* –, y, en el segundo caso, una instancia o un grupo de instancias extendidas – donde la función objetivo a analizar podría ser tanto el *makespan* como el *tardiness*.

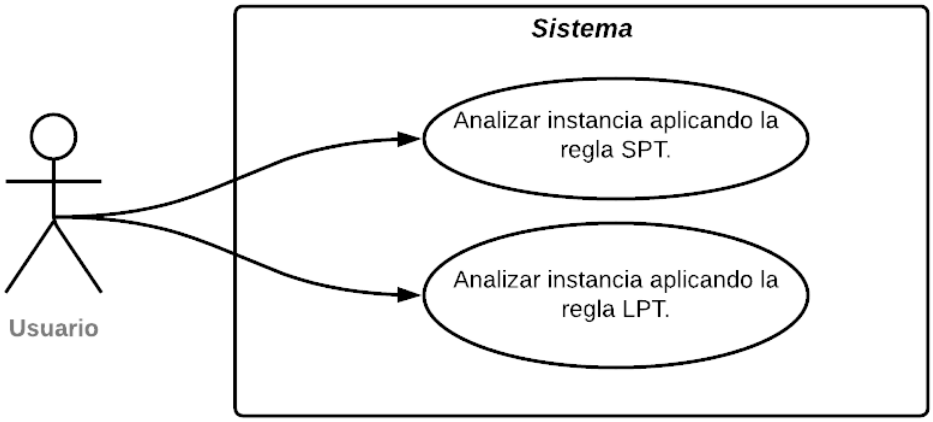



*Diagrama 3. Diagrama de casos de uso al analizar una instancia no extendida.*

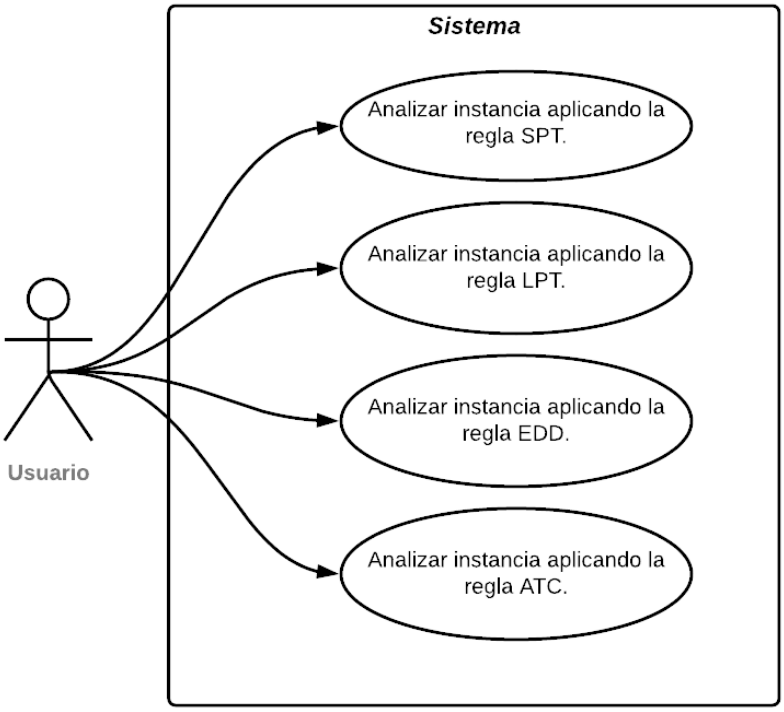



*Diagrama 4. Diagrama de casos de uso al analizar una instancia extendida.*

### *Análisis de los casos de uso*

A continuación, se describe de una forma más exhaustiva cada caso de uso mostrado:

| Cargar un fichero | |
|---|---|
| **Precondiciones** | • La aplicación debe haber sido lanzada.<br>• Ha de existir un fichero con extensión ".txt" con los datos pertinentes de una instancia. |
| **Postcondiciones** | • La aplicación habrá *parseado* y leído el fichero. |
| **Descripción** | El usuario podrá cargar una sola instancia indicando el nombre del fichero (la ruta absoluta de la misma con la extensión ".txt"), junto con la función objetivo e indicando si el fichero es extendido o no. |
| **Variaciones (escenarios secundarios)** | ***Escenario alternativo 1:*** El fichero introducido tiene un formato incorrecto.<br>• Dará un error y no se ejecutará.<br>***Escenario alternativo 2:*** El fichero introducido no tiene formato.<br>• Dará un error y no se ejecutará.<br>***Escenario alternativo 3:*** El fichero introducido no existe (la ruta, por tanto, no existe).<br>• Dará un error y no se ejecutará.<br>***Escenario alternativo 4:*** Se indicó que el fichero era extendido y no lo era.<br>• Dará un error y no se ejecutará.<br>***Escenario alternativo 5:*** Se indicó que el fichero no era extendido y sí lo era.<br>• Dará un error y no se ejecutará.<br>***Escenario alternativo 6:*** Se indicó como función objetivo el *tardiness* y se intentó cargar un fichero no extendido.<br>• Dará un error y no se ejecutará. |

*Tabla 8. Análisis del CU: Cargar un fichero.*

| Cargar un directorio con uno o varios ficheros | |
|---|---|
| **Precondiciones** | • La aplicación debe haber sido lanzada.<br>• Ha de existir una carpeta con uno o varios fichero/s con extensión ".txt" con los datos pertinentes de una instancia. |
| **Postcondiciones** | • La aplicación habrá parseado y leído el/los fichero/s del directorio. |
| **Descripción** | Análogamente al caso anterior, se podrá cargar una carpeta contenedora de instancias indicando, de igual modo, su ruta absoluta. Asimismo, tendrá que especificar la función objetivo y si se trata de instancias extendidas o no. |
| **Variaciones (escenarios secundarios)** | ***Escenario alternativo 1:*** La carpeta introducida está vacía.<br>• El programa se ejecuta, pero al no haber datos no procesa nada ni devuelve ningún *output*.<br><br>*Los demás escenarios son similares al caso de uso anterior, pero tratándose de varios ficheros en lugar de uno solo.* |

*Tabla 9. Análisis del CU: Cargar un directorio con uno o varios ficheros.*

| Realizar análisis con función objetivo *makespan* | |
|---|---|
| **Precondiciones** | • La aplicación debe haber sido lanzada.<br>• Se ha debido de cargar un fichero con extensión ".txt" con los datos pertinentes de una instancia. |
| **Postcondiciones** | • La aplicación habrá ejecutado la instancia y generado resultados. |
| **Descripción** | El usuario podrá analizar una instancia estudiando el *makespan* como función objetivo, obteniendo el tiempo de completitud máximo como resultado. |
| **Variaciones (escenarios secundarios)** | ***Escenario 1:*** El programa se ejecuta, pero no ejecuta todas las instancias (en caso de que sea un directorio con varios ficheros) al desbordarse la memoria.<br>• Se ejecuta y genera el resultado, pero no contendrá datos de instancias que tendrían que haberse ejecutado después de la instancia que desbordó la memoria. |

*Tabla 10. Análisis del CU: Realizar un análisis con función objetivo makespan.*

| Realizar análisis con función objetivo *tardiness* | |
|---|---|
| **Precondiciones** | • La aplicación debe haber sido lanzada.<br>• Se ha debido de cargar un fichero con extensión ".txt" con los datos pertinentes de una instancia. |
| **Postcondiciones** | • La aplicación habrá ejecutado la instancia y generado resultados. |
| **Descripción** | El usuario podrá analizar una instancia estudiando el *tardiness* como función objetivo, obteniendo la suma del tiempo de retardo de cada trabajo de la instancia. |
| **Variaciones (escenarios secundarios)** | ***Escenario 1:*** El programa se ejecuta, pero no ejecuta todas las instancias (en caso de que sea un directorio con varios ficheros) al desbordarse la memoria.<br>• Se ejecuta y genera el resultado, pero no contendrá datos de instancias que tendrían que haberse ejecutado después de la instancia que desbordó la memoria. |

*Tabla 11. Análisis del CU: Realizar análisis con función objetivo tardiness.*

| Analizar una instancia con una regla de prioridad específica | |
|---|---|
| **Precondiciones** | • La aplicación debe haber sido lanzada.<br>• Se ha debido de cargar un fichero con extensión ".txt" con los datos pertinentes de una instancia. |
| **Postcondiciones** | • La aplicación habrá ejecutado la instancia y generado resultados. |
| **Descripción** | El usuario podrá analizar una instancia estudiando una regla de prioridad de las implementadas en específico, obteniendo los resultados correspondientes. |
| **Variaciones (escenarios secundarios)** | ***Escenario 1:*** El programa se ejecuta, pero no ejecuta todas las instancias (en caso de que sea un directorio con varios ficheros) al desbordarse la memoria.<br>• Se ejecuta y genera el resultado, pero no contendrá datos de instancias que tendrían que haberse ejecutado después de la instancia que desbordó la memoria. |

*Tabla 12. Análisis del CU: Analizar una instancia con una regla de prioridad específica.*

| Analizar una instancia con todas las reglas de prioridad implementadas | |
|---|---|
| **Precondiciones** | • La aplicación debe haber sido lanzada.<br>• Se ha debido de cargar un fichero con extensión ".txt" con los datos pertinentes de una instancia. |
| **Postcondiciones** | • La aplicación habrá ejecutado la instancia y generado resultados. |
| **Descripción** | El usuario podrá analizar una instancia estudiando todas las reglas de prioridad de las implementadas, obteniendo los resultados correspondientes. |
| **Variaciones (escenarios secundarios)** | ***Escenario 1:*** El programa se ejecuta, pero no ejecuta todas las instancias (en caso de que sea un directorio con varios ficheros) al desbordarse la memoria.<br>• Se ejecuta y genera el resultado, pero no contendrá datos de instancias que tendrían que haberse ejecutado después de la instancia que desbordó la memoria. |

*Tabla 13. Análisis del CU: Analizar una instancia con todas las reglas de prioridad implementadas.*

| Generar archivo Excel | |
|---|---|
| **Precondiciones** | • La aplicación debe haber sido lanzada.<br>• Se ha debido de cargar un fichero con extensión ".txt" con los datos pertinentes de una instancia. |
| **Postcondiciones** | • La aplicación habrá ejecutado la instancia y generado resultados. |
| **Descripción** | El usuario podrá obtener un archivo de salida que contenga los datos de las soluciones de la instancia a analizar una vez planificada. Este archivo se tratará de un Excel con una instancia por fila y una regla de prioridad por columna. |
| **Variaciones (escenarios secundarios)** | ***Escenario 1:*** Se ha ejecutado dos veces la misma instancia sin cambiar la ruta absoluta.<br>• Se sobrescribe el archivo ya existente, ya que el *output* generado llevaría el mismo nombre que el ya existente.<br>***Escenario 2:*** Se ha ejecutado dos veces el mismo directorio, una vez con unos archivos y la segunda vez con otros distintos.<br>• Se sobrescribe el archivo ya existente, siendo de la forma: si en el primer archivo había menos instancias que en el segundo, resultaría un documento totalmente sobrescrito. En cambio, si en el segundo había menos instancias que en el primero, |

| Generar archivo Excel | |
|---|---|
| | quedarían los datos resultantes guardados y no se borrarían ni se llegarían a sobrescribir. |

*Tabla 14. Análisis del CU: Generar archivo Excel.*

## 3.6 Análisis de clases

Este apartado tiene como objeto llevar a cabo un análisis preliminar de las clases que pueden formar el sistema, sin tener en cuenta la implementación, basándose en los requisitos funcionales especificados.

### *Diagrama de clases*

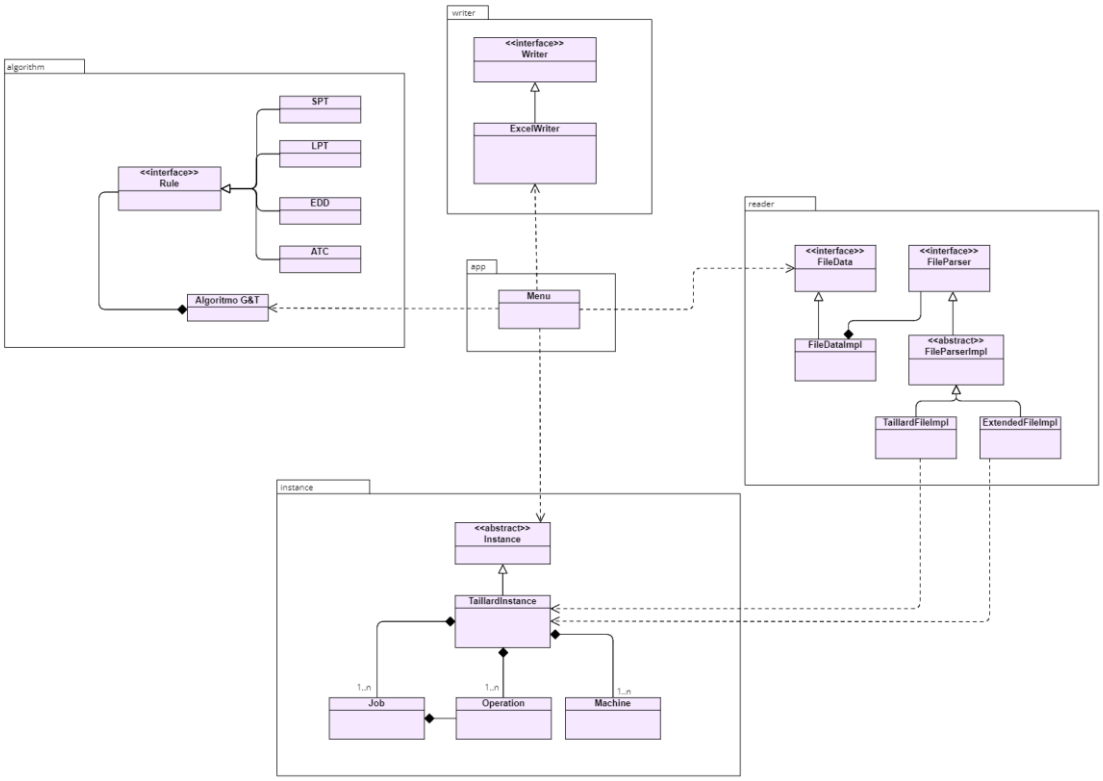



*Diagrama 5. Diagrama de clases del sistema preliminar.*

### *Descripción de las clases*

Se procede a describir las clases mostradas en el Diagrama 5, indicando su propósito, los atributos y los métodos propuestos en este capítulo.

#### *app:*

| **Nombre de la clase** |
|---|
| Menu |
| **Descripción** |
| Recoge el *input* del usuario y se encarga de llamar a los métodos de las clases pertinentes para el análisis deseado por el usuario según los parámetros introducidos y la generación de un archivo Excel con los resultados. |

*Tabla 15. Descripción de clases: Menu.*

#### *reader:*

| **Nombre de la clase** |
|---|
| FileData |
| **Descripción** |
| Interfaz para la obtención de datos de un fichero. |
| **Métodos propuestos** |
| *T getData(String fileName)* – obtiene los datos de un fichero. |

*Tabla 16. Descripción de clases: FileData*

| **Nombre de la clase** |
| --- |
| FileDataImpl |
| **Descripción** |
| Clase que implementa los métodos de la interfaz *FileData*. |
| **Atributos propuestos** |
| *FileParser* – parser para analizar los datos obtenidos del fichero. |
| **Métodos propuestos** |
| *Object getData(String fileName)* – llama al parser para obtener los datos del fichero. |

*Tabla 17. Descripción de clases: FileDataImpl.*

| **Nombre de la clase** |
| --- |
| FileParser |
| **Descripción** |
| Interfaz para la lectura de datos de un fichero. |
| **Métodos propuestos** |
| *readFile(String fileName)* – lee un fichero.<br>*parseFile(String data)* – parsea los datos de un fichero. |

*Tabla 18. Descripción de clases: FileParser.*

| **Nombre de la clase** |
| --- |
| FileParserImpl |
| **Descripción** |
| Clase que implementa los métodos de la interfaz *FileParser.* |
| **Métodos propuestos** |
| *String readFile(String fileName)* – comprueba que sea un fichero de texto y, en caso de serlo, lee todos los *bytes* del mismo.<br>*T parseFile(String data)* – parsea los datos de un fichero. |

*Tabla 19. Descripción de clases: FileParserImpl.*

| **Nombre de la clase** |
| --- |
| TaillardFileImpl |
| **Descripción** |
| Clase que se encarga de crear una instancia de Taillard. Extiende de *FileParserImpl*. |
| **Métodos propuestos** |
| *TaillardInstance parseFile(String data)* – parsea los datos de un fichero creando con ellos una instancia de Taillard. |

*Tabla 20. Descripción de clases: TaillardFileImpl.*

| **Nombre de la clase** |
| --- |
| ExtendedFileImpl |
| **Descripción** |
| Clase que se encarga de crear una instancia extendida, es decir, con datos como *dueDate* y *weight*. |
| **Métodos propuestos** |
| *TaillardInstance parseFile(String data)* – parsea los datos de un fichero creando con ellos una instancia extendida. |

*Tabla 21. Descripción de clases: ExtendedFileImpl.*

*instance:*

| **Nombre de la clase** |
|---|
| Instance |
| **Descripción** |
| Clase abstracta que representa una instancia. |
| **Atributos propuestos** |
| *nJobs* – número de trabajos de la instancia.<br>*nMachines* – número de máquinas de la instancia.<br>*machines* – listado de máquinas que forman la instancia.<br>*totalProcessingTime* – suma de tiempo de procesamiento de cada operación de la instancia.<br>*lowerBound* – cota inferior (si se especificase en el fichero).<br>*upperBound* – cota superior (si se especificase en el fichero). |
| **Métodos propuestos** |
| *List<Job> getJobs()* – método abstracto que devuelve el listado de trabajos que forman la instancia.<br>*void createMachines()* – en función de *nMachines* crea las máquinas que forman la instancia.<br>*int getnJobs()* – devuelve el número de trabajos.<br>*int getnMachines()* – devuelve el número de máquinas.<br>*int getTotalProcessingTime()* – devuelve el tiempo de procesamiento total de la instancia.<br>*int getLowerBound()* – devuelve el valor de la cota inferior.<br>*int getUpperBound()* – devuelve el valor de la cota superior.<br>*List<Machine> getMachines()* – devuelve el listado de máquinas que forman la instancia. |

*Tabla 22. Descripción de clases: Instance.*

| **Nombre de la clase** |
|---|
| TaillardInstance |
| **Descripción** |
| Clase que extiende de la clase abstracta *Instance*. |
| **Atributos propuestos** |
| *timeSeed* – semilla del tiempo.<br>*machineSeed* – semilla de la máquina.<br>*jobs* – listado de trabajos de la instancia. |
| **Métodos propuestos** |
| *List<Job> getJobs()* – devuelve el listado de trabajos que forman la instancia. |

*Tabla 23. Descripción de clases: TaillardInstance.*

| **Nombre de la clase** |
|---|
| Job |
| **Descripción** |
| Clase que representa un trabajo dentro de una instancia. |
| **Atributos propuestos** |
| *jobId* – valor numérico que representa el identificador del trabajo.<br>*operations* – listado de operaciones que conforman el trabajo.<br>*dueDate* – fecha límite del trabajo (en caso de que se trate de una instancia extendida).<br>*weight* – peso del trabajo (en caso de que se trate de una instancia extendida). |
| **Métodos propuestos** |
| *List<Operation> getOperations()* – devuelve el listado de operaciones del trabajo.<br>*int getJobId()* – devuelve el identificador del trabajo.<br>*int getDueDate()* – devuelve la fecha límite del trabajo.<br>*double getWeight()* – devuelve el peso del trabajo.<br>*void resetOperations()* – recorre el listado de operaciones del trabajo y las resetea, es decir, pone el atributo *isScheduled* a *false*. |

*Tabla 24. Descripción de clases: Job.*

| **Nombre de la clase** |
| --- |
| Machine |
| **Descripción** |
| Clase que representa una máquina dentro de una instancia. |
| **Atributos propuestos** |
| *machineId* – valor numérico que representa el identificador de la máquina.<br>*releaseTime* – tiempo de liberación de la máquina.<br>*isBusy* – *true/false* en función de si la máquina está procesando una tarea o no, respectivamente. |
| **Métodos propuestos** |
| *int getMachineId()* – devuelve el identificador de la máquina.<br>*boolean isBusy()* – devuelve *true/false* en función de si la máquina está procesando una tarea o no, respectivamente.<br>*long getReleaseTime()* – devuelve el tiempo de liberación de la máquina.<br>*void setReleaseTime(long newReleaseTime)* – modifica el tiempo de liberación de la máquina por uno pasado por parámetro. |

*Tabla 25. Descripción de clases: Machine.*

| **Nombre de la clase** |
| --- |
| Operation |
| **Descripción** |
| Clase que representa una operación dentro de una instancia. |
| **Atributos propuestos** |
| *operationId* – valor numérico que representa el identificador de la operación.<br>*machineId* – valor numérico que representa el identificador de la máquina en la que tiene que ejecutarse la operación.<br>*jobId* – valor numérico que representa el identificador del trabajo al que pertenece la operación.<br>*processingTime* – tiempo de procesamiento de la operación.<br>*startTime* – tiempo de inicio de la operación. Como mínimo, será la suma de los tiempos de procesamiento de las operaciones previas a ella. En caso de ser la primera del trabajo, en este trabajo será 0.<br>*isScheduled* – *true/false* en función de si está planificada o no, respectivamente. |
| **Métodos propuestos** |
| *long getProcessingTime()* – devuelve el tiempo de procesamiento de la operación.<br>*int getOperationId()* – devuelve el valor del identificador de la operación.<br>*int getJobId()* – devuelve el valor del identificador del trabajo al que pertenece la operación.<br>*int getMachineId()* – devuelve el valor del identificador de la máquina en la que tiene que ejecutarse la operación.<br>*boolean isScheduled()* – devuelve *true/false* en función de si la operación está planificada o no, respectivamente.<br>*long getStartTime()* – devuelve el momento en el tiempo en el que, como mínimo, puede comenzar a ejecutarse una tarea.<br>*long getEndTime()* – devuelve el momento en el tiempo en el que una tarea habría finalizado su ejecución.<br>*void scheduleOperation()* – planifica la operación: modifica su *startTime* en caso de que sea distinto al inicialmente planteado y pone *isScheduled* a *true*.<br>*void setStartTime(long newStartTime)* – asigna un nuevo valor de comienzo a una operación.<br>*void resetOperation()* – resetea la operación: pone *isScheduled* a *false*. |

*Tabla 26. Descripción de clases: Operation.*

*writer:*

| **Nombre de la clase** |
|---|
| Writer |
| **Descripción** |
| Interfaz encargada de escribir un fichero de salida. |
| **Métodos propuestos** |
| *void writeMatrixStartTimes()* – imprime los tiempos de inicio de cada operación en forma matricial.<br>*void write()* – imprime los resultados de cada instancia planificada. |

*Tabla 27. Descripción de clases: Writer.*

| **Nombre de la clase** |
|---|
| ExcelWriterImpl |
| **Descripción** |
| Clase que implementa los métodos de la interfaz *Writer.* |
| **Atributos propuestos** |
| *fileName* – nombre del fichero de salida. |
| **Métodos propuestos** |
| *void writeMatrixStartTimes()* – imprime los tiempos de inicio de cada operación en forma matricial en un fichero Excel.<br>*void write()* – imprime los resultados de cada instancia planificada en un Excel, asignando una fila por cada instancia y una columna por cada regla de prioridad empleada. |

*Tabla 28. Descripción de clases: ExcelWriterImpl.*

*algorithm:*

| **Nombre de la clase** |
|---|
| GTAlgorithm |
| **Descripción** |
| Clase que representa el algoritmo voraz Giffler and Thompson. |
| **Atributos propuestos** |
| *instance – instancia a ejecutar.*<br>*rule* – regla de prioridad a aplicar durante el algoritmo.<br>*setA* – lista de operaciones que conformarán el set A del algoritmo.<br>*setB* – lista de operaciones que conformarán el set B del algoritmo.<br>*results* – lista de operaciones resultantes.<br>*writer* – generador de ficheros de resultados. |
| **Métodos propuestos** |
| *List<ResultTask> run()* – ejecuta el algoritmo.<br>*void initializeASet()* – inicializa el *setA*.<br>*void initializeBSet()* – inicializa el setB.<br>*Operation getOPrimeOperation()* – elige del *setA* la operación $\theta'$.<br>*Operation getOStarOperation()* – elige del *setB* la operación $\theta^*$ mediante una regla de prioridad.<br>*long getLastEndTimeScheduled()* – obtiene el valor de tiempo (mayor valor de unidades de tiempo) a partir del cual puede comenzar la operación en función de las operaciones que tengan la misma máquina que esta o bien la tarea previa del trabajo.<br>*void addResult()* – añade a la lista de operaciones resueltas la operación planificada.<br>*void writeStartingTimeMatrix()* – obtiene los datos de startTime de cada operación de cada trabajo y los imprime en una hoja de cálculo.<br>*void writeAll()* – obtiene los datos resultantes en base a la función objetivo a determinar y los imprime en una hoja de cálculo. |

*Tabla 29. Descripción de clases: GTAlgorithm.*

| **Nombre de la clase** |
|---|
| Rule |
| **Descripción** |
| Interfaz que representa una regla de prioridad. |
| **Métodos propuestos** |
| *Operation run(List<Operation> operations)* – ejecuta el código de la regla de prioridad con las operaciones que le llegan por parámetro. |

*Tabla 30. Descripción de clases: Rule.*

| **Nombre de la clase** |
|---|
| SPTRule |
| **Descripción** |
| Clase que representa la regla de prioridad *ShortestProcessingTime*. |
| **Métodos propuestos** |
| *Operation run(List<Operation> operations)* – ejecuta el código de la regla de prioridad con las operaciones que le llegan por parámetro, devolviendo la que mayor prioridad tenga. |

*Tabla 31. Descripción de clases: SPTRule.*

| **Nombre de la clase** |
|---|
| LPTRule |
| **Descripción** |
| Clase que representa la regla de prioridad *LongestProcessingTime*. |
| **Métodos propuestos** |
| *Operation run(List<Operation> operations)* – ejecuta el código de la regla de prioridad con las operaciones que le llegan por parámetro, devolviendo la que mayor prioridad tenga. |

*Tabla 32. Descripción de clases: LPTRule*

| **Nombre de la clase** |
| --- |
| EDDRule |
| **Descripción** |
| Clase que representa la regla de prioridad *EarliestDueDate*. |
| **Métodos propuestos** |
| *Operation run(List<Operation> operations)* – ejecuta el código de la regla de prioridad con las operaciones que le llegan por parámetro, devolviendo la que mayor prioridad tenga. |

*Tabla 33. Descripción de clases: EDDRule.*

| **Nombre de la clase** |
| --- |
| ATCRule |
| **Descripción** |
| Clase que representa la regla de prioridad *ApparentTardinessCost*. |
| **Atributos propuestos** |
| *jobs* – listado de Jobs de la instancia.<br>*kValue* – valor de escala *k* especificado por el usuario.<br>*auxRule* – regla auxiliar en caso de producirse una indeterminación. |
| **Métodos propuestos** |
| *Operation run(List<Operation> operations)* – ejecuta el código de la regla de prioridad con las operaciones que le llegan por parámetro, devolviendo la que mayor prioridad tenga.<br>*double function(Operation operation, double weight, int dueDate)* – calcula la prioridad de una operación que se le pasa por parámetro siguiendo la fórmula de la regla.<br>*double getAvgProcessingTime()* – calcula el tiempo de procesamiento medio de las tareas que aún no han sido planificadas de la instancia. |

*Tabla 34. Descripción de clases: ATCRule.*

## 3.7 Especificación del plan de pruebas

Por tratarse de un trabajo de investigación resulta complicado realizar un plan de pruebas *a priori*, siguiendo una metodología formal. No obstante, se plantea en este apartado un plan de pruebas enfocado en la funcionalidad principal del sistema.

### 3.7.1 Pruebas unitarias

Las pruebas unitarias son un tipo de pruebas que validan el comportamiento de un objeto y la lógica del sistema. Es una forma de comprobar que el código funciona de forma correcta. Su fin es comprobar que cada unidad del código implementado funciona correcta y eficientemente por separado.

En este proyecto, en el paquete *logic* – donde se encuentra toda la lógica de negocio del sistema – se encuentran distintos submódulos con distintas funcionalidades, algunas de las cuales es más conveniente probar en conjunto, ya que realizar una prueba unitaria no resultaría relevante a la hora de determinar su funcionamiento. En este apartado se someten a pruebas unitarias todas las clases pertenecientes al paquete *instances*, correspondiente a todas las clases implementadas que conforman una instancia, tanto aquellas básicas de Taillard como aquellas instancias extendidas.

| **PU 1. Instancia básica (de Taillard)** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PU1.1 | Comprobar los valores básicos de la instancia | a) Se comprueba que la lista de trabajos no está vacía.<br>b) Se comprueba que la lista de máquinas no esté vacía.<br>c) Se comprueba que la lista de trabajos contiene el número esperado.<br>d) Se comprueba que la lista de máquinas contiene el número esperado. | La lista de trabajos no está vacía y, además, está compuesta por *n* trabajos. La lista de máquinas no está vacía y, además, está compuesta por *m* máquinas. |
| PU1.2 | Comprobar los *id* de los trabajos | Se comprueba que los trabajos creados tienen los *ids* correctos. | Cada trabajo tiene un *id* asignado, en orden ascendente. |

| **PU 1. Instancia básica (de Taillard)** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PU1.3 | Comprobar tiempos de procesamiento | a) Se comprueba el tiempo total de procesamiento (la suma de los tiempos de procesamiento de cada tarea de la instancia).<br>b) Se comprueba el valor de la cota inferior.<br>c) Se comprueba el valor de la cota superior. | El tiempo total de procesamiento es correcto.<br>El valor de la cota inferior y de la cota superior coincide con el valor de la instancia. |
| PU1.4 | Comprobar tiempos de procesamiento de cada operación | Se comprueban:<br>a) Los *ids* de las operaciones, viendo que son correctos y siguen el orden.<br>b) Los tiempos de procesamiento de cada operación de cada trabajo. | Cada operación tiene un *id* asignado, en orden ascendente.<br>El valor del tiempo de procesamiento de cada operación de cada trabajo coincide con el que se muestra en el fichero de la instancia. |
| PU1.5 | Comprobar que no tiene *dueDate* | Se comprueba que, al ser una instancia básica, ningún trabajo tiene *dueDate*, o lo que es lo mismo, es 0. | El valor de *dueDate* de cada trabajo de la instancia es de 0. |
| PU1.6 | Comprobar que no tiene *weight* | Se comprueba que, al ser una instancia básica, ningún trabajo tiene *weight*, o lo que es lo mismo, es 0. | El valor de *weight* de cada trabajo es de 0. |
| PU1.7 | Comprobar que las operaciones no están planificadas | Se comprueba que las operaciones no están planificadas, ya que no se ejecutó ningún algoritmo. | El atributo *isScheduled* de todas las operaciones es *false.* |

| **PU 1. Instancia básica (de Taillard)** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PU1.8 | Comprobar las máquinas de las operaciones | Se comprueba mediante el *id* de las máquinas que las máquinas asignadas a la ejecución de las operaciones de cada trabajo se corresponden con las indicadas en el fichero de la instancia. | El *id* de la máquina corresponde con el asignado en el fichero de la instancia. |
| PU1.9 | Comprobar los *startTimes* de las operaciones | a) Se comprueba que la primera operación de cada trabajo tiene como *startTime* 0.<br>b) El *startTime* de los siguientes es la suma de los tiempos de procesamiento de las operaciones predecesoras (en el trabajo o en la máquina). | Los *startTimes* de las primeras operaciones de cada trabajo son 0.<br>Los siguientes *startTimes* coinciden con la suma de tiempos de procesamiento de las tareas predecesoras (en el trabajo o en la máquina). |
| PU1.10 | Comprobar las máquinas creadas | Se comprueban que los *id* de cada máquina son correctos, que todas las máquinas no están ocupadas (ya que no están procesando ninguna operación) y que el atributo *releaseTime* devuelve correctamente los valores. | Los *id* de cada máquina son correctos.<br>El atributo de las máquinas *isBusy* es *false*. |
| PU1.11 | Comprobar que, al planificar una tarea, esta está planificada | Se simula la planificación de una tarea, pasándole un nuevo *startTime*.<br>a1) Se comprueba que se actualiza el nuevo *startTime*.<br>b2) Se comprueba que el *endTime* es correcto.<br>c2) Se comprueba que está planificada.<br><br>Se simula un reseteo de operaciones del trabajo:<br>a2) Se comprueba que la tarea no está planificada. | El *startTime* se actualiza correctamente con el nuevo valor.<br>El *endTime* se corresponde con la suma del *startTime* de la operación y el *processingTime*.<br>El atributo *isScheduled* es *true* tras haberse producido la planificación de la operación.<br>Tras el reseteo, el atributo *isScheduled* vuelve a ser *false*. |

*Tabla 35.Pruebas unitarias para una instancia básica de Taillard.*

| PU 2. Instancia extendida | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PU2.1 | Comprobar los valores básicos de la instancia | a) Se comprueba que la lista de trabajos no está vacía.<br>b) Se comprueba que la lista de máquinas no esté vacía.<br>c) Se comprueba que la lista de trabajos contiene el número esperado.<br>d) Se comprueba que la lista de máquinas contiene el número esperado. | La lista de trabajos no está vacía y, además, está compuesta por *n* trabajos.<br>La lista de máquinas no está vacía y, además, está compuesta por *m* máquinas. |
| PU2.2 | Comprobar que cada trabajo tiene *dueDate* | Se comprueba que cada *dueDate* de cada trabajo coincide con el del fichero de la instancia. | El valor de *dueDate* de cada trabajo se corresponde con el del fichero. |
| PU2.3 | Comprobar que cada trabajo tiene *weight* | Se comprueba que cada *weight* de cada trabajo coincide con el del fichero de la instancia. | El valor de *weight* de cada trabajo se corresponde con el del fichero. |

*Tabla 36. Pruebas unitarias para una instancia extendida.*

| **PU 3. Tarea planificada *(ResultTask)*** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PU3.1 | Comprobar el *id* del trabajo | Se comprueba que el *id* del trabajo coincide con el correspondiente de la tarea sin planificar inicial. | El *id* del trabajo coincide con el del trabajo de la tarea sin planificar inicial. |
| PU3.2 | Comprobar el *id* de la máquina | Se comprueba el *id* de la máquina en la que se procesó la tarea. | El *id* de la máquina corresponde con aquella en la cual se procesó. |
| PU3.3 | Comprobar el *startTime* | Se comprueba el *startTime* de la tarea con el momento en el tiempo en el que se comenzó a planificar la tarea. | El valor de *startTime* corresponde con el valor en el cual se comenzó a planificar la tarea. |
| PU3.4 | Comprobar el tiempo de procesamiento | Se comprueba que el tiempo de procesamiento coincide con el de la tarea sin planificar inicial. | El tiempo de procesamiento es el mismo que el de la tarea sin planificar inicial. |
| PU3.5 | Comprobar el *endTime* de la tarea | Se comprueba que el *endTime* se corresponde con la suma del *startTime* y el *processingTime* de la tarea. | El *endTime* se corresponde con la suma correcta del *startTime* y el *processingTime*. |

*Tabla 37. Pruebas unitarias para una tarea planificada.*

### 3.7.2 Pruebas de integración y del sistema

Las pruebas de integración son aquellas que prueban que todos los elementos unitarios que componen un sistema pueden funcionar en grupo correctamente, haciendo que cada subsistema cumpla su funcionalidad y todo el sistema pueda funcionar exitosamente.

Aquí se llevan a cabo los *tests* sobre el *parser*, la ejecución del algoritmo planificador, la generación del grafo de restricciones y la generación de *output*. Estas pruebas no se llevaron a cabo después de las pruebas unitarias, ya que fueron necesarias

desde el primer momento, para poder ir asegurándose de que el sistema cumplía con las funcionalidades establecidas y que funcionaba según lo esperado en conjunto.

| **PI 1. Carga de un fichero de instancias** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PI1.1 | Fichero con un formato incorrecto | El usuario selecciona un fichero con un formato incorrecto. | El sistema debe notificar un error al usuario y finalizar. |
| PI1.2 | Fichero sin formato, es decir, sin extensión alguna | El usuario selecciona un fichero sin formato, es decir, sin extensión alguna. | El sistema debe notificar un error al usuario y finalizar. |
| PI1.3 | Fichero inexistente | El usuario introduce un fichero inexistente (su ruta no existe). | El sistema debe notificar un error al usuario y finalizar. |
| PI1.4 | Fichero no extendido, pero sí lo es | El usuario intenta cargar un fichero no extendido e indicó que sí lo era. | El sistema debe notificar un error al usuario y finalizar. |
| PI1.5 | Fichero extendido, pero no lo es | El usuario intenta cargar un fichero extendido y no indicó que lo era. | El sistema debe notificar un error al usuario y finalizar. |
| PI1.6 | *Tardiness* en fichero no extendido | El usuario indica como función objetivo el *tardiness* e intentó cargar un fichero no extendido. | El sistema debe notificar un error al usuario y finalizar. |

*Tabla 38. Pruebas de integración de carga de ficheros de instancias.*

| **PI 2. Carga de un directorio de instancias** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PI2.1 | Directorio vacío | El usuario selecciona un directorio de ficheros vacío. | El sistema se ejecuta, pero no ocurre nada ya que no hay ninguna instancia que analizar. |
| PI2.2 | Instancia no extendida | El usuario intenta cargar en un directorio de instancias extendidas una que no lo era, habiendo indicado que se trataban de instancias extendidas. | Cuando llegue el momento de ejecutar la instancia en cuestión, el sistema debe notificar un error al usuario y finalizar. |
| PI2.3 | Instancia extendida | El usuario intenta cargar en un directorio de instancias no extendidas una que sí lo es, habiendo indicado que se trataban de instancias básicas. | Cuando llegue el momento de ejecutar la instancia en cuestión, el sistema debe notificar un error al usuario y finalizar. |

*Tabla 39. Pruebas de integración de carga de un directorio de instancias.*

| **PI 3. Análisis con función objetivo *makespan*** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PI3.1 | Instancia con la regla SPT | Se prueba una instancia haciendo uso de la regla de prioridad SPT: se carga una instancia, se crea un algoritmo G&T y se ejecuta junto con la regla SPT, obteniendo una lista de tareas planificadas correspondientes a la instancia, donde se comprueba:<br>a) El tamaño de la lista de tareas planificadas coincide con el número de operaciones que tiene la instancia.<br>b) Las operaciones se guardan en la lista según se planifican: se comprueba que el orden es el esperado teniendo en cuenta:<br>• Tiempo de procesamiento de la tarea.<br>• *Id* del trabajo.<br>• *Id* de la máquina en la que se procesó.<br>• *EndTime* de la tarea planificada. | El tamaño de la lista de tareas planificadas coincide con el número de operaciones de la instancia probada.<br>Las operaciones resueltas se encuentran guardadas en la lista en el orden en el que debieron ser planificadas.<br>Las propiedades de las tareas resueltas coinciden con las propiedades de la tarea sin planificar inicial, y su *endTime* coincide con el momento en el tiempo en el que terminó de ejecutarse. |
| PI3.2 | Instancia con un tiempo de procesamiento 0 y la regla SPT | Se prueba una instancia haciendo uso de la regla de prioridad SPT en la cual existe un tiempo de procesamiento de 0, pudiendo provocar una indeterminación.<br>Se carga la instancia, se ejecuta el algoritmo G&T y se comprueba lo mismo que en el caso anterior. | Al igual que en el caso anterior, una ejecución normal de la instancia. |

| **PI 3. Análisis con función objetivo *makespan*** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PI3.3 | Instancia con la regla LPT | Se prueba una instancia haciendo uso de la regla de prioridad LPT: se carga una instancia, se crea un algoritmo G&T y se ejecuta junto con la regla LPT, obteniendo una lista de tareas planificadas correspondientes a la instancia, donde se comprueba:<br>a) El tamaño de la lista de tareas planificadas coincide con el número de operaciones que tiene la instancia.<br>b) Las operaciones se guardan en la lista según se planifican: se comprueba que el orden es el esperado teniendo en cuenta:<br>• Tiempo de procesamiento de la tarea.<br>• *Id* del trabajo.<br>• *Id* de la máquina en la que se procesó.<br>• *EndTime* de la tarea planificada. | El tamaño de la lista de tareas planificadas coincide con el número de operaciones de la instancia probada.<br>Las operaciones resueltas se encuentran guardadas en la lista en el orden en el que debieron ser planificadas.<br>Las propiedades de las tareas resueltas coinciden con las propiedades de la tarea sin planificar inicial, y su *endTime* coincide con el momento en el tiempo en el que terminó de ejecutarse. |

*Tabla 40.Pruebas de integración de carga del análisis con función objetivo makespan.*

| **PI 4. Análisis con función objetivo *tardiness*** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PI4.1 | Instancia con la regla EDD | Se prueba una instancia haciendo uso de la regla de prioridad EDD: se carga una instancia, se crea un algoritmo G&T y se ejecuta junto con la regla EDD, obteniendo una lista de tareas planificadas correspondientes a la instancia, donde se comprueba:<br>a) El tamaño de la lista de tareas planificadas coincide con el número de operaciones que tiene la instancia.<br>b) Las operaciones se guardan en la lista según se planifican: se comprueba que el orden es el esperado teniendo en cuenta:<br>• Tiempo de procesamiento de la tarea.<br>• *Id* del trabajo.<br>• *Id* de la máquina en la que se procesó.<br>• *EndTime* de la tarea planificada. | El tamaño de la lista de tareas planificadas coincide con el número de operaciones de la instancia probada.<br>Las operaciones resueltas se encuentran guardadas en la lista en el orden en el que debieron ser planificadas.<br>Las propiedades de las tareas resueltas coinciden con las propiedades de la tarea sin planificar inicial, y su *endTime* coincide con el momento en el tiempo en el que terminó de ejecutarse. |
| PI4.2 | Instancia con la regla EDD y con el mismo *dueDate* | Se prueba una instancia haciendo uso de la regla de prioridad EDD, pero en la cual los trabajos tienen la misma *dueDate*.<br>Se carga la instancia, se ejecuta el algoritmo G&T y se comprueban los mismos datos que en el caso anterior. | Al igual que en el caso anterior, una ejecución normal de la instancia. |

| **PI 4. Análisis con función objetivo *tardiness*** | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PI4.3 | Instancia con la regla ATC y *g = 0,5* | Se prueba una instancia haciendo uso de la regla de prioridad ATC y valor de *g = 0,5*: se carga una instancia, se crea un algoritmo G&T y se ejecuta junto con la regla ATC, especificando el valor de *g = 0,5*, obteniendo una lista de tareas planificadas correspondientes a la instancia, donde se comprueba:<br>a) El tamaño de la lista de tareas planificadas coincide con el número de operaciones que tiene la instancia.<br>b) Las operaciones se guardan en la lista según se planifican: se comprueba que el orden es el esperado teniendo en cuenta:<br>• Tiempo de procesamiento de la tarea.<br>• *Id* del trabajo.<br>• *Id* de la máquina en la que se procesó.<br>• *EndTime* de la tarea planificada. | El tamaño de la lista de tareas planificadas coincide con el número de operaciones de la instancia probada.<br>Las operaciones resueltas se encuentran guardadas en la lista en el orden en el que debieron ser planificadas.<br>Las propiedades de las tareas resueltas coinciden con las propiedades de la tarea sin planificar inicial, y su *endTime* coincide con el momento en el tiempo en el que terminó de ejecutarse |
| PI4.4 | Instancia con la regla ATC y *g = 0,5* y con el mismo *dueDate* y *weight* | Se prueba una instancia haciendo uso de la regla de prioridad ATC, pero en la cual los trabajos tienen la misma *dueDate* y el mismo *weight*.<br>Se carga la instancia, se ejecuta el algoritmo G&T y se comprueban los mismos datos que en el caso anterior. | Al igual que en el caso anterior, una ejecución normal de la instancia. |

*Tabla 41. Pruebas de integración de carga del análisis con función objetivo tardiness.*

| PI 5. Generar archivo Excel | | | |
|---|---|---|---|
| **ID** | **Prueba** | **Descripción de la prueba** | **Resultado esperado** |
| PI5.1 | Generación de archivo Excel | Se carga una instancia, se ejecuta el algoritmo G&T junto con cualquier regla de prioridad y se llama al método de generar *output.*<br>Además de comprobar que se crea el directorio de salida del fichero Excel, se comprueba que los valores de las celdas son correctos. | Existe un directorio de salida.<br>Existe el archivo Excel generado.<br>Los valores de las celdas son correctos. |

*Tabla 42. Pruebas de integración de la generación de un archivo Excel.*

### 3.7.3 Pruebas de usabilidad

Las pruebas de usabilidad son aquellas pruebas cuyo objeto reside en determinar el grado de satisfacción de un cliente con un sistema, es decir, el grado en que un producto puede ser utilizado por determinados usuarios para lograr los objetivos específicos. Se mide generalmente mediante una serie de indicadores que puedan ser observables y cuantificables de los que se pueda obtener unos resultados más allá de la intuición.

Para estas pruebas se hará uso de un cuestionario que deberá ser redactado previamente con el fin de llevar a cabo un estudio de las interacciones de una muestra representativa de los usuarios con el sistema.

El plan de pruebas para este proyecto se puede encontrar en el apartado 4.3 Especificación técnica del plan de pruebas.

# CAPÍTULO 4: Diseño del sistema

## 4.1 Diseño de casos de uso reales

Haciendo uso de diagramas de secuencia, en este apartado se pretende representar gráficamente los casos de uso del sistema. Así, en el Diagrama 6 se muestra la interacción de un usuario con el sistema de forma genérica, pudiendo hacer uso de cualquier algoritmo de planificación y con las funcionalidades básicas planteadas para el sistema; y en el Diagrama 7 se muestra la interacción de un usuario haciendo uso del algoritmo de planificación planteado en este trabajo, el algoritmo G&T guiado por reglas de prioridad, empleando la nomenclatura utilizada en la implementación.

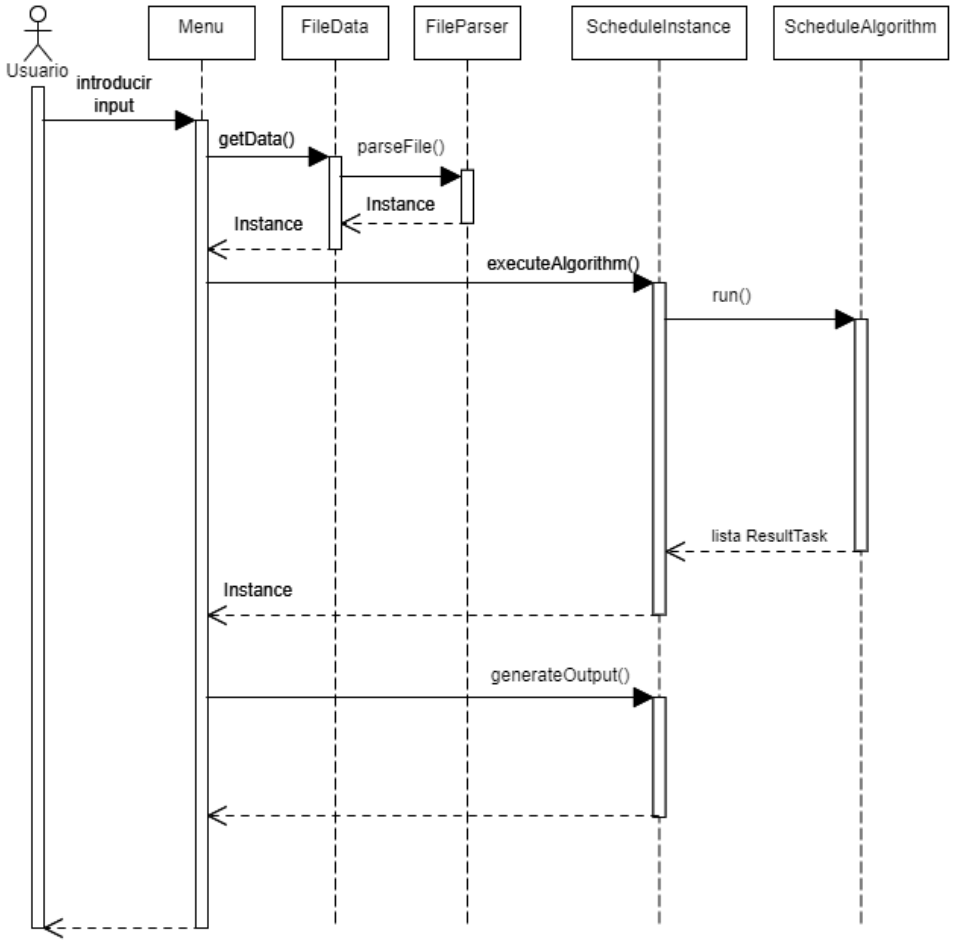



*Diagrama 6. Diagrama de secuencia genérico.*

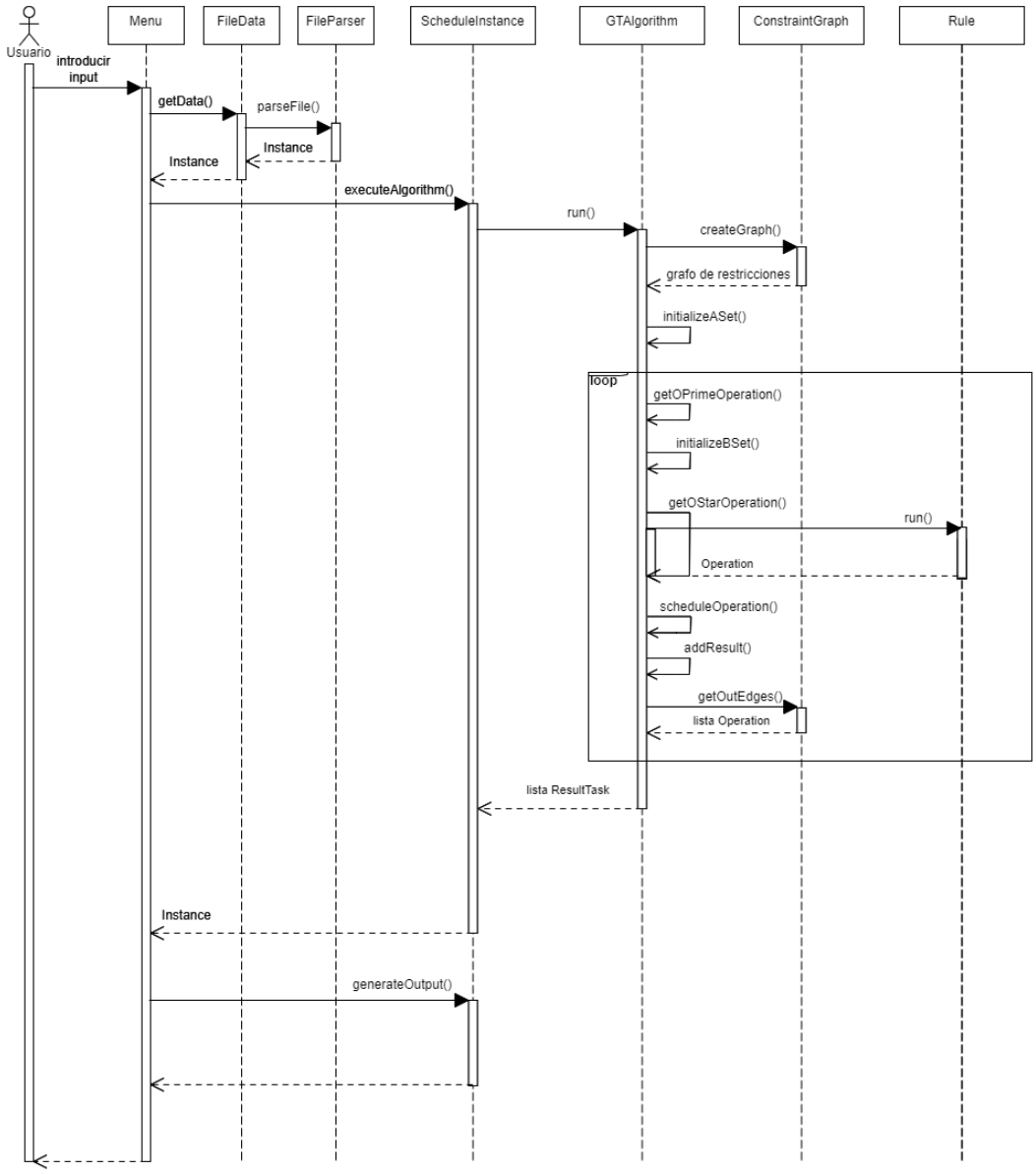



*Diagrama 7. Diagrama de secuencia para el algoritmo G&T.*

## 4.2 Diseño de clases

En este apartado se realizará un análisis sobre el diseño llevado a cabo del sistema junto con los diagramas pertinentes.

### 4.2.1. Diseño de la arquitectura de módulos del sistema

#### *Diagrama de paquetes*

En general, si bien el sistema sigue una lógica secuencial, es imprescindible que exista una comunicación entre los distintos módulos para asegurar el correcto funcionamiento de las distintas tareas. Así, la estructura principal del sistema se encuentra dividida en capas, por lo que en la arquitectura del proyecto existen los siguientes paquetes:

- **application:** El principal paquete de la gestión de la interfaz de línea de comandos. Se trata de la capa en la que se fundamenta la interacción con el usuario y las funcionalidades del sistema. Podría describirse como la capa que contiene las clases que se encargan de guardar el *input* que introduce el usuario y que posteriormente será procesado por el sistema, ejecutado y se obtendrá una futura solución.
- **logic:** El principal paquete que contiene la lógica de negocio del sistema. En él están contenidos los siguientes módulos:
    - **exceptions**: Contiene clases que recogen posibles errores, asignándoles un mensaje personalizado para hacer más usable la aplicación.
    - **graph**: Contiene las clases necesarias para la implementación de un grafo de restricciones. Se trata de una interfaz *Graph* empleable para cualquier tipo de grafo, una clase *GraphImpl* que representa la implementación de un grafo con métodos para ello, pudiendo ser un grafo dirigido o no, así como conexo o no; y una clase *ConstraintGraph* que extiende de esta última mencionada, y representa la estructura de un grafo de restricciones, que será el tipo de estructura utilizada en el sistema.
    - **instances:** Se divide en las distintas clases comunes a un problema de *scheduling*, es decir: *Instance* – que se trata, además, de una clase abstracta –, *Job, Machine, Operation* y *ResultTask* para representar una tarea ya planificada. Además, también consta de otro paquete específico llamado ***Taillard***, que guarda la clase *TaillardInstance*, la cual extiende de la clase abstracta *Instance* y representa concretamente una instancia de Taillard.
    - **output:** Consta de la clase *Writer* y el paquete ***impl*** que almacena la clase *ExcelWriterImpl*, que implementa la primera y se encarga de la escritura de datos, así como de la generación de un archivo Excel que contenga estos datos producto del análisis de las instancias ejecutadas.
    - **parser:** Contiene todo lo necesario para parsear y leer datos de los archivos del *input*. Consta de las interfaces *FileData* y *FileParser*.

Asimismo, existe un paquete ***impl*** contenedor de las clases que implementan dichas interfaces: *FileDataImpl* y *FileParserImpl*, respectivamente. De igual modo, existe la clase *TaillardFileImpl* que extiende de *FileParserImpl* destinada al procesamiento de datos de una instancia de Taillard normal, y la clase *ExtendedFileImpl* destinada al procesamiento de datos de una instancia extendida.

- **schedule:** Elemental para el contexto del proyecto. Consta de lo necesario para la ejecución del algoritmo G&T en el cual se basa este trabajo y las reglas de prioridad incorporadas. Contiene una clase *ScheduleInstance* encargada de contener el algoritmo usado para la planificación junto con el listado de tareas resueltas.
  - **algorithm:** Contiene una interfaz *ScheduleAlgorithm* que contiene los métodos básicos que debería tener cualquier algoritmo de construcción de planificaciones que se implemente. Además, contiene un paquete ***impl*** en el que se encuentra la clase *GTAlgorithm* que implementa de *ScheduleAlgorithm* y en donde se especifica el funcionamiento del algoritmo G&T.
  - **rules:** Contiene una interfaz *Rule* y un paquete ***impl*** donde se almacenan las clases con las distintas reglas de prioridad implementadas.

Como se indicó en la sección 3.3 Identificación de subsistemas, el sistema no requiere de ningún tipo de base de datos o repositorio de información más allá de los ficheros de datos de entrada proporcionados por el usuario, por lo que no se contempla la necesidad de añadir una capa de persistencia al proyecto.

En el Diagrama 8 se muestra la presentación de cómo se organiza el modelo de elementos en paquetes y las dependencias entre ellos, incluyendo las dependencias entre paquetes.

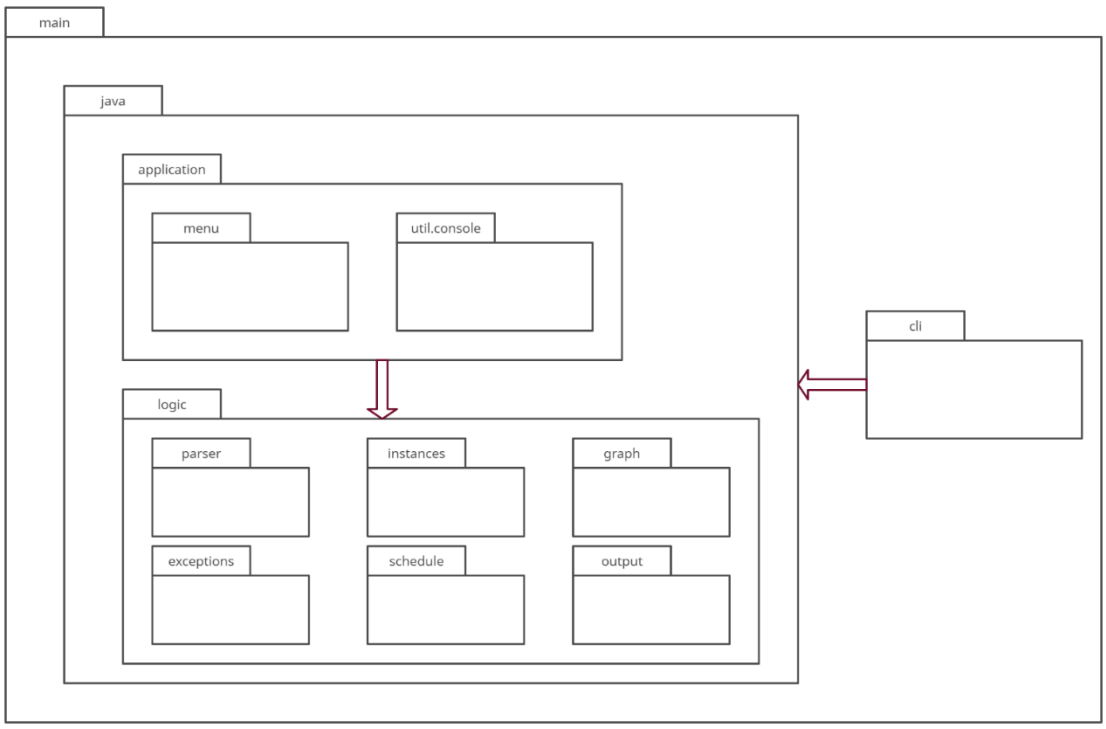



*Diagrama 8. Diagrama de paquetes del sistema.*

### Diagrama de clases

Representación de la estructura del sistema donde se muestran las clases del mismo, sus atributos, sus operaciones (o métodos), y las relaciones que existen entre ellas.

Primeramente, las clases encargadas de la lectura de ficheros, representadas en el Diagrama 9. A continuación, lo crucial del sistema implementado: las clases relacionadas con la implementación y, por tanto, el funcionamiento del algoritmo G&T y las reglas de prioridad, acompañado de la estructura de datos empleada en el proyecto, mostrado respectivamente en el Diagrama 10 y en el Diagrama 11.

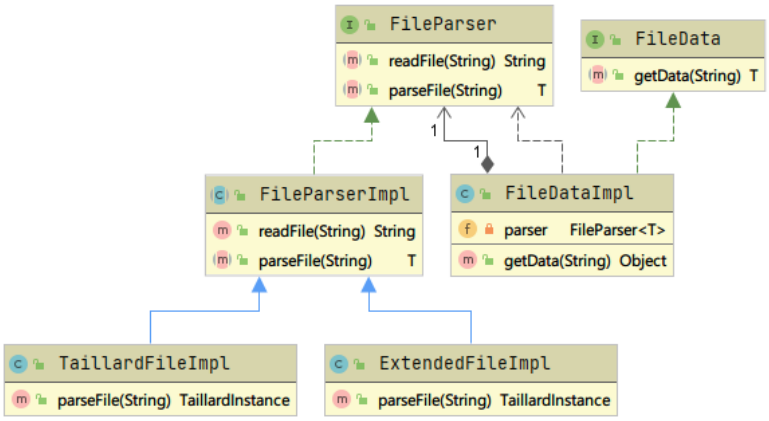



*Diagrama 9. Diagrama del paquete Parser.*

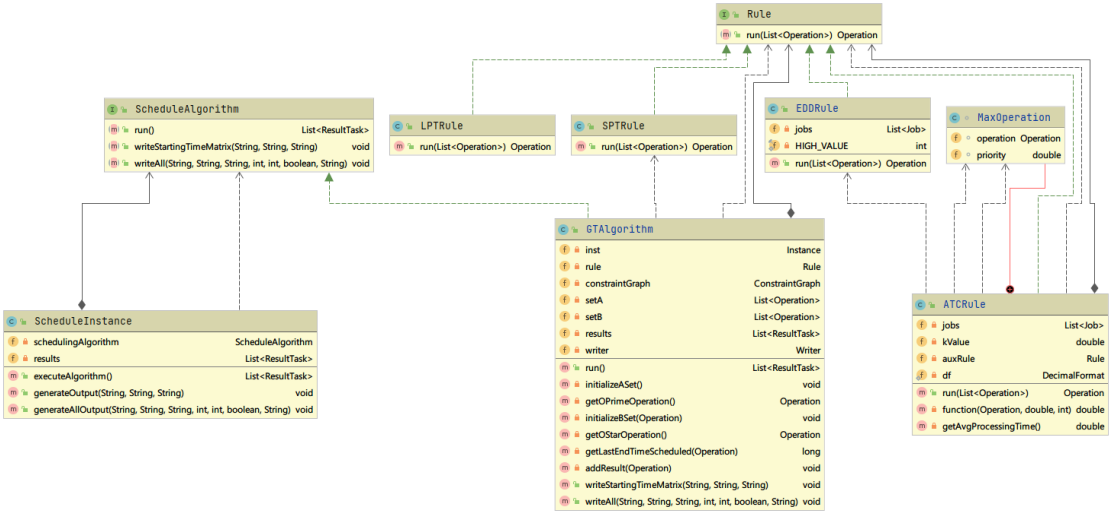



*Diagrama 10. Diagrama de los paquetes Rules y Schedule, que contienen las clases que se encargan del algoritmo GyT y las reglas de prioridad.*

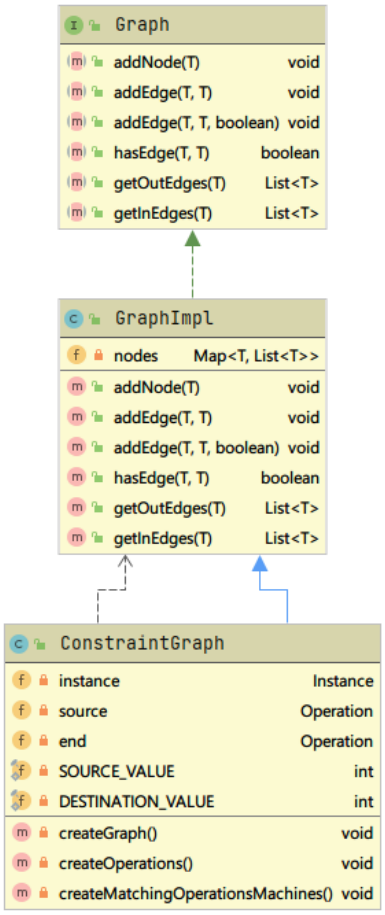



*Diagrama 11. Diagrama del paquete Graph.*

Finalmente, en el Diagrama 12, se encuentran todas las clases empleadas en la lógica del sistema, incluyendo las relaciones entre ellas.

*Diagrama 12. Diagrama de clases del sistema.*

### Diagrama de componentes

Representación de los componentes que conforman el sistema, sus relaciones, interacciones e interfaces públicas más relevantes en el mismo. Se muestran en el Diagrama 13 los diferentes componentes del sistema, como el CLI *(Command Line Interface)*, los componentes necesarios para la lectura y carga *(parser)* de los archivos, los necesarios para la planificación de las tareas y el de escritura de archivos, junto con las relaciones de dependencia entre ellos.

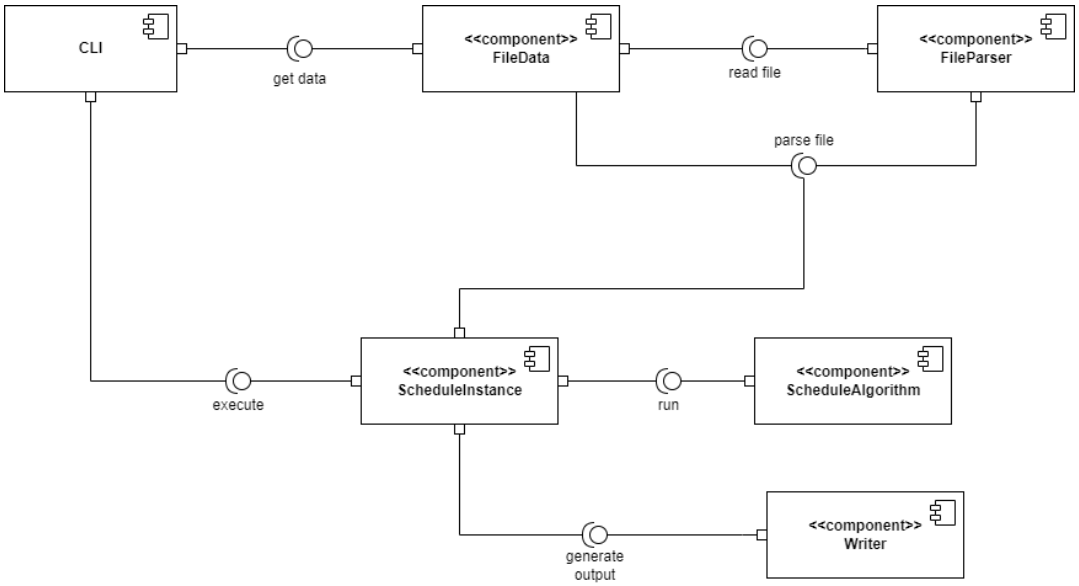



*Diagrama 13. Diagrama de componentes del sistema.*

### Diagrama de despliegue

Aquí, en el Diagrama 14, se resumen tanto los procesos *software* como los entornos implicados para llevar a cabo la ejecución del propio sistema. El único elemento a tener en cuenta es el cliente, ya que el sistema del trabajo propuesto no requiere de un servidor web o un repositorio.

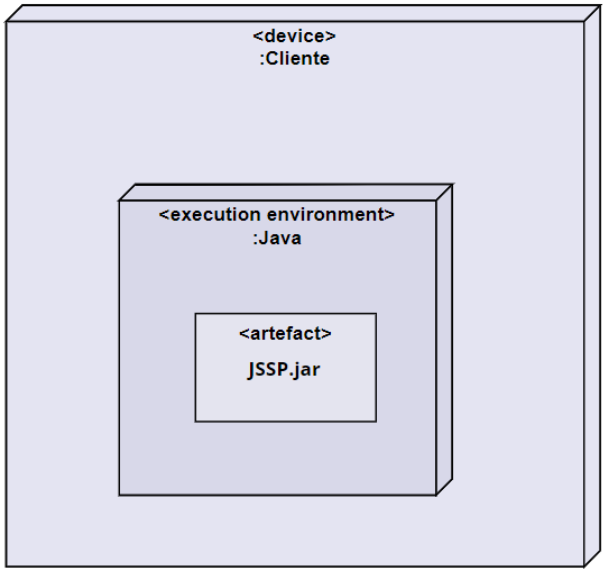



*Diagrama 14. Diagrama de despliegue del sistema.*

## Diagrama de estados

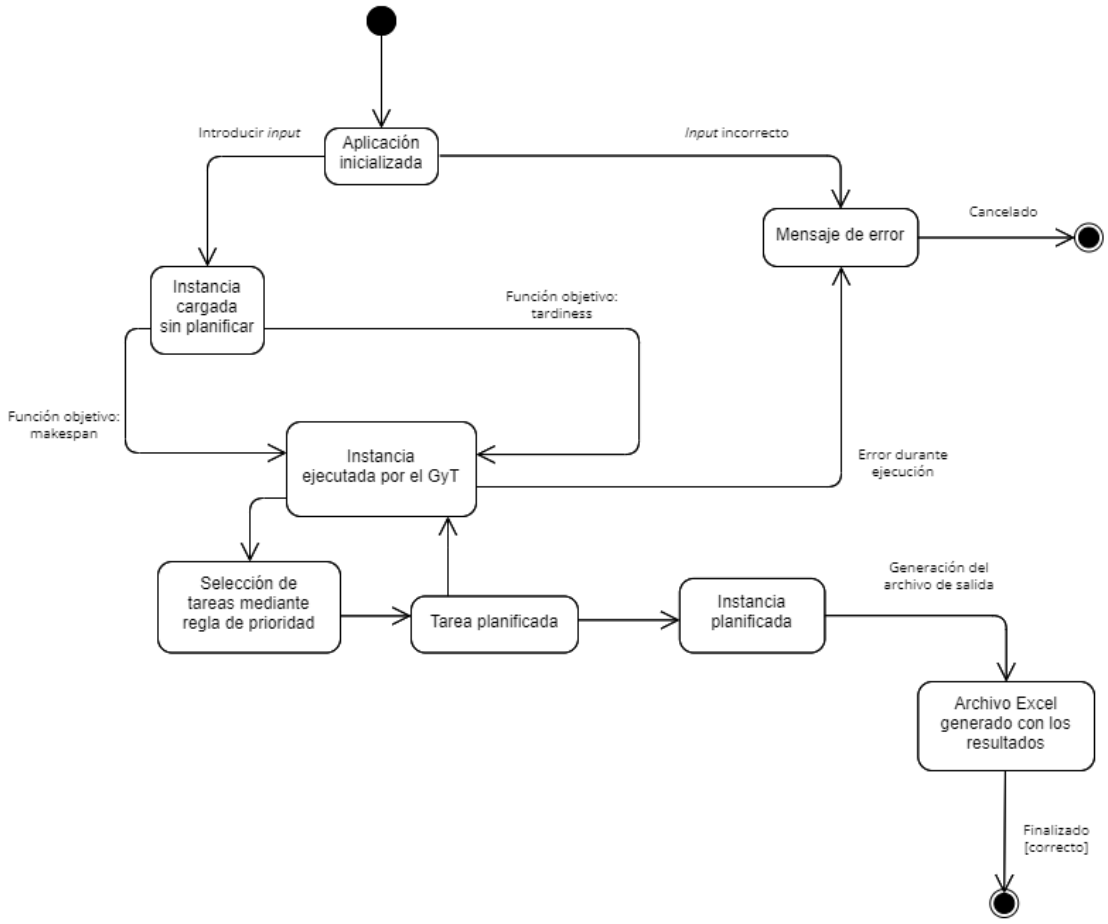



*Diagrama 15. Diagrama de estados del sistema.*

## 4.3 Especificación técnica del plan de pruebas

Se describen aquí los aspectos técnicos del plan de pruebas. Se llevaron a cabo un total de 20 pruebas:

- *InstanceTest:* contiene las pruebas descritas en 3.7.1 Pruebas unitarias relativas a una instancia y sus atributos.
  - 3 pruebas en total.
- *ParserTest:* contiene las pruebas descritas en la Tabla 38 y en la Tabla 39 del apartado 3.7.2 Pruebas de integración y del sistema.
  - 9 pruebas en total.
- *WriterTest:* contiene la prueba descrita en la Tabla 42 del apartado 3.7.2 Pruebas de integración y del sistema.
  - 1 prueba.
- *RulesTest:* contiene las pruebas descritas en la Tabla 40 y en la Tabla 41 del apartado 3.7.2 Pruebas de integración y del sistema.
  - 7 pruebas en total.

### 4.3.1 Pruebas unitarias

Para las pruebas de una instancia básica se usará una instancia de tamaño 2x3, y para las pruebas de una instancia extendida se hará uso de esa misma instancia, pero en su versión extendida. Esta versión extendida se obtendrá siguiendo los pasos descritos en el apartado 6.2.1 Generación de instancias para el JSSPTWT.

Antes de cada prueba y de realizar las comprobaciones pertinentes, se especifica la ruta en la que se encuentra el archivo a *parsear* y se crea la instancia en función de los datos *parseados.*

Los ficheros con los que se realizarán estas pruebas se encuentran incluidos en la carpeta *resources* dentro del código fuente del proyecto y son:

- *tai02x03.txt*
- *tai02x03ext.txt*

| **PU 1. Instancia básica (de Taillard)** | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PU1.1 | Comprobar los valores básicos de la instancia. | OK. |
| PU1.2 | Comprobar los *id* de los trabajos. | OK. |
| PU1.3 | Comprobar tiempos de procesamiento. | OK. |
| PU1.4 | Comprobar tiempos de procesamiento de cada operación. | OK. |
| PU1.5 | Comprobar que no tiene *dueDate.* | OK. |
| PU1.6 | Comprobar que no tiene *weight.* | OK. |
| PU1.7 | Comprobar que las operaciones no están planificadas. | OK. |
| PU1.8 | Comprobar las máquinas de las operaciones. | OK. |
| PU1.9 | Comprobar los *startTimes* de las operaciones. | OK. |
| PU1.10 | Comprobar las máquinas creadas. | OK. |
| PU1.11 | Comprobar que, al planificar una tarea, esta está planificada. | OK. |

*Tabla 43. Resultados para las pruebas unitarias para una instancia básica.*

| **PU 2. Instancia extendida** | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PU2.1 | Comprobar los valores básicos de la instancia. | OK. |
| PU2.2 | Comprobar que cada trabajo tiene *dueDate.* | OK. |
| PU2.3 | Comprobar que cada trabajo tiene *weight.* | OK. |

*Tabla 44. Resultados para las pruebas unitarias para una instancia extendida.*

| PU 3. Tarea planificada *(ResultTask)* | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PU3.1 | Comprobar el *id* del trabajo. | OK. |
| PU3.2 | Comprobar el *id* de la máquina. | OK. |
| PU3.3 | Comprobar el *startTime.* | OK. |
| PU3.4 | Comprobar el tiempo de procesamiento. | OK. |
| PU3.5 | Comprobar el *endTime* de la tarea. | OK. |

*Tabla 45. Resultados para las pruebas unitarias para una tarea planificada.*

### 4.3.2 Pruebas de integración y del sistema

Tanto para las pruebas de la función objetivo *makespan* como para el *tardiness* se hará uso de una instancia de tamaño 2x3, solo que en el caso del *tardiness* se hará uso de la versión extendida de la misma, de la misma forma que en el apartado anterior de pruebas unitarias.

Los ficheros y directorios con los que se realizarán estas pruebas se encuentran incluidos en la carpeta *resources* dentro del código fuente del trabajo fin de grado y son:

- *sinExtension*
- *otraExtension.pdf*
- *invent.txt*
- *tai02x03mal.txt*
- *tai02x03.txt*
- *tai02x03sptAlt.txt*
- *tai02x03ext.txt*
- *tai02x03extAlternative.txt*
- *emptyDir*
- *dir*

| PI 1. Carga de un fichero de instancias | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PI1.1 | Fichero con un formato incorrecto. | OK. |

| **PI 1. Carga de un fichero de instancias** | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PI1.2 | Fichero sin formato, es decir, sin extensión alguna. | OK. |
| PI1.3 | Fichero inexistente. | OK. |
| PI1.4 | Fichero no extendido, pero sí lo es. | OK. |
| PI1.5 | Fichero extendido, pero no lo es. | OK. |
| PI1.6 | *Tardiness* en fichero no extendido. | OK. |

*Tabla 46. Resultados para las pruebas de integración de carga de un fichero de instancias.*

| **PI 2. Carga de un directorio de instancias** | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PI2.1 | Directorio vacío. | OK. |
| PI2.2 | Instancia no extendida. | OK. |
| PI2.3 | Instancia extendida. | OK. |

*Tabla 47. Resultados para las pruebas de integración de carga de un directorio de instancias.*

| **PI 3. Análisis con función objetivo *makespan*** | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PI3.1 | Instancia con la regla SPT. | OK. |
| PI3.2 | Instancia con un tiempo de procesamiento 0 y la regla SPT. | OK. |
| PI3.3 | Instancia con la regla LPT. | OK. |

*Tabla 48. Resultados para las pruebas de integración de carga del análisis con función objetivo makespan.*

| PI 4. Análisis con función objetivo *tardiness* | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PI4.1 | Instancia con la regla EDD. | OK. |
| PI4.2 | Instancia con la regla EDD y con el mismo *dueDate.* | OK. |
| PI4.3 | Instancia con la regla ATC y *g = 0,5.* | OK. |
| PI4.4 | Instancia con la regla ATC y *g = 0,5* y con el mismo *dueDate* y *weight.* | OK. |

*Tabla 49. Resultados para las pruebas de integración de carga del análisis con función objetivo tardiness.*

| PI 5. Generar archivo Excel | | |
|---|---|---|
| **ID** | **Prueba** | **Resultado** |
| PI5.1 | Generación de archivo Excel. | OK. |

*Tabla 50. Resultados para las pruebas de integración de la generación de un archivo Excel.*

### 4.3.3 Pruebas de usabilidad

Se describen aquí las pruebas de usabilidad planteadas, cuyo objeto reside en evaluar el grado de eficiencia del sistema en un entorno real, mediante el estudio de las interacciones de una muestra representativa de los usuarios con el sistema.

Como ya se habló anteriormente, se podría hacer la distinción de dos tipos de usuarios en función de su conocimiento respecto al campo del *scheduling*: aquellos usuarios experimentados en la materia, y aquellos usuarios nuevos sin conocimiento respecto al tema.

Se describe aquí un cuestionario inicial que persigue el objetivo de conocer al usuario que hará uso del prototipo. Este cuestionario constará de:

- Preguntas de carácter general que deberá responder el usuario.
- Una serie de actividades guiadas a realizar por el mismo usuario.
- Un cuestionario a responder por el usuario al finalizar las actividades.
- Un cuestionario a responder por el observador sobre las acciones realizadas por el usuario.

### *Preguntas de carácter general*

| **Cuestionario inicial** |
|---|
| *¿Usa un ordenador frecuentemente?* |
| 1. A diario.<br>2. Varias veces a la semana.<br>3. Ocasionalmente.<br>4. Nunca o casi nunca. |
| *¿Tiene experiencia usando la interfaz de línea de comandos?* |
| 1. Sí, la he usado varias veces.<br>2. Solo la usé una o dos veces.<br>3. Solamente sé cómo ejecutarla.<br>4. No tengo experiencia. |
| *¿Conoce usted algún problema de scheduling?* |
| 1. Sí, poseo un amplio conocimiento del tema.<br>2. He trabajado ocasionalmente con ello.<br>3. Lo conozco, pero nunca he trabajado con ello.<br>4. Carezco de conocimiento o experiencia alguna. |
| *¿Qué interés presenta usted con respecto a este sistema?* |
| 1. Profundizar en la investigación sobre el problema *job shop scheduling*.<br>2. Como herramienta con vistas a realizar análisis a un mayor nivel.<br>3. Probar su uso como parte de introducirme al *scheduling*.<br>4. Solamente por curiosidad. |

*Tabla 51. Pruebas de usabilidad: cuestionario inicial.*

### *Actividades guiadas*

Teniendo en cuenta el abanico de 1 a 5, siendo 1 "más fácil" y 5 "más difícil", el usuario seguirá el siguiente plan de tareas:

| **Tarea** | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| *Cargar una instancia no extendida y calcular el makespan.* | | | | | |
| *Cargar una instancia extendida y calcular el makespan.* | | | | | |
| *Cargar una instancia extendida y calcular el tardiness.* | | | | | |
| *Cargar un directorio de instancias no extendidas y calcular el makespan.* | | | | | |
| *Cargar un directorio de instancias extendidas y calcular el makespan.* | | | | | |
| *Cargar un directorio de instancias extendidas y calcular el tardiness.* | | | | | |

*Tabla 52. Pruebas de usabilidad: actividades guiadas.*

### *Preguntas cortas sobre la aplicación y observaciones*

| **Facilidad de Uso** | **Sí** | **Bastante** | **No demasiado** | **No** |
|---|---|---|---|---|
| *¿Le resulta fácil el uso del sistema?* | | | | |
| *¿Le resulta lógico que la salida se genere en el directorio donde se encuentra la instancia que se planifica?* | | | | |
| **Funcionalidad** | **Sí** | **Bastante** | **No demasiado** | **No** |
| *¿Funciona el sistema como se esperaba?* | | | | |
| *¿Considera que el tiempo de respuesta de la aplicación es muy grande?* | | | | |
| *¿Utilizaría la herramienta frente a otras alternativas?* | | | | |
| *¿Ha echado en falta alguna funcionalidad?* | | | | |
| **Observaciones** | | | | |

*Tabla 53. Pruebas de usabilidad: preguntas cortas sobre la aplicación y observaciones.*

*Cuestionario para el responsable de las pruebas*

| Aspecto observado | Notas |
|---|---|
| *Tiempo en realizar la ejecución de la instancia* | |
| *Errores cometidos* | |
| *Otras observaciones* | |

*Tabla 54. Pruebas de usabilidad: cuestionario para el responsable de las pruebas.*

Es a partir de los resultados obtenidos cuando se podrá elaborar una conclusión respecto a la usabilidad del sistema en cuanto a la facilidad de aprendizaje y manejo, accesibilidad y, por tanto, grado de satisfacción general del usuario.

### 4.3.4 Cobertura de código

Como se indicará más adelante (concretamente en el apartado 5.2 Herramientas empleadas), una de las herramientas empleadas en este proyecto ha sido *CodeCov*, para medir el nivel de cobertura de código que se consiguió en la realización de las pruebas.

En este proyecto se ha conseguido un total de 88% de cobertura de líneas de código sobre el paquete *logic*, que es el que incluye toda la lógica de negocio del sistema. Dentro de él, se obtuvo la siguiente cobertura de código:

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| logic.graph | 100% (2/ 2) | 100% (11/ 11) | 95,1% (58/ 61) |

*Figura 8. Cobertura de código para el paquete graph.*

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| logic.instances | 100% (5/ 5) | 93% (40/ 43) | 96% (95/ 99) |

*Figura 9. Cobertura de código para el paquete instances.*

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| logic.instances.taillard | 100% (1/ 1) | 100% (3/ 3) | 100% (9/ 9) |

*Figura 10. Cobertura de código para el paquete instances.taillard.*

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| logic.output.impl | 100% (1/ 1) | 100% (3/ 3) | 74,2% (69/ 93) |

*Figura 11. Cobertura de código para el paquete output.impl.*

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| logic.parser.impl | 100% (4/ 4) | 100% (8/ 8) | 97,8% (90/ 92) |

*Figura 12. Cobertura de código para el paquete parser.impl.*

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| logic.schedule | 100% (1/ 1) | 100% (4/ 4) | 100% (9/ 9) |

*Figura 13. Cobertura de código para el paquete schedule.*

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| logic.schedule.algorithm.impl | 100% (1/ 1) | 90,9% (10/ 11) | 86,3% (82/ 95) |

*Figura 14. Cobertura de código para el paquete algorithm.impl.*

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| logic.schedule.rules.impl | 100% (5/ 5) | 92,9% (13/ 14) | 90,6% (96/ 106) |

*Figura 15. Cobertura de código para el paquete schedule.rules.impl.*

No se incluye la cobertura de código relativa a la capa del CLI *(Command Line Interface)* ya que las pruebas unitarias están realizadas para comprobar el correcto funcionamiento de la capa de lógica de negocio.

## 4.4 Ejecución de las pruebas del sistema

### Pruebas de usabilidad

Se muestran a continuación los resultados de las pruebas de usabilidad para dos usuarios con las dos distinciones marcadas en 3.4 Identificación de actores del sistema: un usuario con experiencia en cuanto a planificación de tareas y otro sin conocimiento al respecto.

#### *Primera prueba*

**Preguntas de carácter general**

| **Cuestionario inicial** |
|---|
| *¿Usa un ordenador frecuentemente?* |
| 1. A diario.<br>2. Varias veces a la semana.<br>3. Ocasionalmente.<br>4. Nunca o casi nunca. |
| *¿Tiene experiencia usando la interfaz de línea de comandos?* |
| 1. Sí, la he usado varias veces.<br>2. Solo la usé una o dos veces.<br>3. Solamente sé cómo ejecutarla.<br>4. No tengo experiencia. |
| *¿Conoce usted algún problema de scheduling?* |
| 1. Sí, poseo un amplio conocimiento del tema.<br>2. He trabajado ocasionalmente con ello.<br>3. Lo conozco, pero nunca he trabajado con ello.<br>4. Carezco de conocimiento o experiencia alguna. |
| *¿Qué interés presenta usted con respecto a este sistema?* |
| 1. Profundizar en la investigación sobre el problema *job shop scheduling*.<br>2. Como herramienta con vistas a realizar análisis a un mayor nivel.<br>3. Probar su uso como parte de introducirme al *scheduling*.<br>4. Solamente por curiosidad. |

*Tabla 55. Primera prueba de usabilidad: preguntas de carácter general.*

**Actividades guiadas**

| Tarea | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| *Cargar una instancia no extendida y calcular el makespan.* | X | | | | |
| *Cargar una instancia extendida y calcular el makespan.* | X | | | | |
| *Cargar una instancia extendida y calcular el tardiness.* | X | | | | |
| *Cargar un directorio de instancias no extendidas y calcular el makespan.* | X | | | | |
| *Cargar un directorio de instancias extendidas y calcular el makespan.* | X | | | | |
| *Cargar un directorio de instancias extendidas y calcular el tardiness.* | X | | | | |

*Tabla 56. Primera prueba de usabilidad: actividades guiadas.*

**Preguntas cortas sobre la aplicación y observaciones**

| **Facilidad de Uso** | **Sí** | **Bastante** | **No demasiado** | **No** |
|---|---|---|---|---|
| *¿Le resulta fácil el uso del sistema?* | X | | | |
| *¿Le resulta lógico que la salida se genere en el directorio donde se encuentra la instancia que se planifica?* | | X | | |
| **Funcionalidad** | **Sí** | **Bastante** | **No demasiado** | **No** |
| *¿Funciona el sistema como se esperaba?* | X | | | |
| *¿Considera que el tiempo de respuesta de la aplicación es muy grande?* | | | | X |
| *¿Utilizaría la herramienta frente a otras alternativas?* | | X | | |
| *¿Ha echado en falta alguna funcionalidad?* | | X | | |
| **Observaciones** | Estaría bien que cuando el usuario se confunda: por ejemplo, que no introduzca los parámetros, o que vaya a ejecutar una instancia extendida y no ponga la <e>… además de que salte la excepción indicando el error, saliese una pequeña ayuda recordando los argumentos de ejecución y su tipo. | | | |

*Tabla 57. Primera prueba de usabilidad: preguntas cortas sobre la aplicación y observaciones.*

**Cuestionario para el responsable de las pruebas**

| Aspecto observado | Notas |
|---|---|
| *Tiempo en realizar la ejecución de la instancia* | El usuario realiza las ejecuciones en un corto periodo de tiempo. |
| *Errores cometidos* | Se va a ejecutar una instancia extendida y se olvida poner el argumento <e>. |
| *Otras observaciones* | |

*Tabla 58. Primera prueba de usabilidad: cuestionario para el responsable de las pruebas.*

### *Segunda prueba*

**Preguntas de carácter general**

| **Cuestionario inicial** |
|---|
| *¿Usa un ordenador frecuentemente?* |
| 1. A diario.<br>2. Varias veces a la semana.<br>3. Ocasionalmente.<br>4. Nunca o casi nunca. |
| *¿Tiene experiencia usando la interfaz de línea de comandos?* |
| 1. Sí, la he usado varias veces.<br>2. Solo la usé una o dos veces.<br>3. Solamente sé cómo ejecutarla.<br>4. No tengo experiencia. |
| *¿Conoce usted algún problema de scheduling?* |
| 1. Sí, poseo un amplio conocimiento del tema.<br>2. He trabajado ocasionalmente con ello.<br>3. Lo conozco, pero nunca he trabajado con ello.<br>4. Carezco de conocimiento o experiencia alguna. |
| *¿Qué interés presenta usted con respecto a este sistema?* |
| 1. Profundizar en la investigación sobre el problema *job shop scheduling*.<br>2. Como herramienta con vistas a realizar análisis a un mayor nivel.<br>3. Probar su uso como parte de introducirme al *scheduling*.<br>4. Solamente por curiosidad. |

*Tabla 59. Segunda prueba de usabilidad: preguntas de carácter general.*

**Actividades guiadas**

| Tarea | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| *Cargar una instancia no extendida y calcular el makespan.* | X | | | | |
| *Cargar una instancia extendida y calcular el makespan.* | X | | | | |
| *Cargar una instancia extendida y calcular el tardiness.* | X | | | | |
| *Cargar un directorio de instancias no extendidas y calcular el makespan.* | X | | | | |
| *Cargar un directorio de instancias extendidas y calcular el makespan.* | X | | | | |
| *Cargar un directorio de instancias extendidas y calcular el tardiness.* | X | | | | |

*Tabla 60. Segunda prueba de usabilidad: actividades guiadas.*

**Preguntas cortas sobre la aplicación y observaciones**

| **Facilidad de Uso** | **Sí** | **Bastante** | **No demasiado** | **No** |
|---|---|---|---|---|
| *¿Le resulta fácil el uso del sistema?* | X | | | |
| *¿Le resulta lógico que la salida se genere en el directorio donde se encuentra la instancia que se planifica?* | | X | | |
| **Funcionalidad** | **Sí** | **Bastante** | **No demasiado** | **No** |
| *¿Funciona el sistema como se esperaba?* | X | | | |
| *¿Considera que el tiempo de respuesta de la aplicación es muy grande?* | | | | X |
| *¿Utilizaría la herramienta frente a otras alternativas?* | | X | | |
| *¿Ha echado en falta alguna funcionalidad?* | | X | | |
| **Observaciones** | Estaría bien que cuando el usuario se confunda: por ejemplo, que no introduzca los parámetros, o que vaya a ejecutar una instancia extendida y no ponga la <e>… además de que salte la excepción indicando el error, saliese una pequeña ayuda recordando los argumentos de ejecución y su tipo. | | | |

*Tabla 61. Segunda prueba de usabilidad: preguntas cortas sobre la aplicación y observaciones.*

**Cuestionario para el responsable de las pruebas**

| Aspecto observado | Notas |
|---|---|
| *Tiempo en realizar la ejecución de la instancia* | El usuario realiza las ejecuciones en un corto periodo de tiempo. |
| *Errores cometidos* | Como en la primera prueba, se va a ejecutar una instancia extendida y se olvida poner el argumento <e>.<br><br>A la hora de poner el argumento de la ruta del fichero intentó emplear la ruta relativa, y solo se contempla la ruta absoluta o bien el nombre fichero/directorio si se encuentra en el mismo directorio que el archivo ".jar". |
| *Otras observaciones* | |

*Tabla 62. Segunda prueba de usabilidad: cuestionario para el responsable de las pruebas.*

Dado que en ambas pruebas el usuario tuvo problemas en relación al tercer parámetro cuando se trataban de instancias extendidas, se decidió añadir un mensaje de error más concreto que recuerde al usuario los parámetros a introducir y que, en caso de querer cargar instancias extendidas, añada el tercer parámetro necesario para ello.

# CAPÍTULO 5: Implementación del sistema

Para la correcta implementación del sistema se consideró útil, a la par que fundamental, utilizar un tipo de programación orientada a objetos. Como se explicó en capítulos anteriores, el lenguaje de programación elegido por varios motivos fue Java.

El código cumple las funcionalidades requeridas y previamente explicadas en este trabajo, pero para conseguir que el sistema pueda ser ampliado en un futuro, añadiendo nuevas funcionalidades de la manera más simple posible, se trató el proyecto según unos estándares que permiten la interoperabilidad.

## 5.1 Entorno de desarrollo

Después de haber tomado la decisión de emplear Java como lenguaje de programación, se consideraron distintos entornos de desarrollo como posibles alternativas de cara a la realización del proyecto, entre las que estaban *Eclipse*, *IntelliJ IDEA* y *Visual Studio Code*. De ellos, la elección fue el empleo de *IntelliJ* por resultar ser más intuitivo que los otros, además de que es fácil de usar en cuestiones como el completado automático de código o añadir *plugins*. En cuanto al tercero, *Visual Studio*, se descartó inicialmente por la cantidad de extensiones que habría que descargar para tener una experiencia igualable a los otros dos entornos. Además, algo que motivó el uso de *IntelliJ* sobre *Eclipse* fue la licencia de la Universidad de Oviedo para el uso de este software.


*Ilustración 1. Logo de IntelliJ IDEA.*

## 5.2 Herramientas empleadas

### Intellij 2020.3.2

*Intellij IDEA* es un entorno de desarrollo integrado para la creación de programas informáticos. Es desarrollado por *JetBrains*. Se hizo uso de este IDE para el desarrollo del código del proyecto, así como las pruebas unitarias.

### Apache Maven 3.3.0

*Apache Maven* es una herramienta de software para la gestión y construcción principalmente de proyectos Java. Se hizo uso de esta herramienta para la gestión de dependencias del proyecto, concretamente las explicadas a continuación:

#### · *Apache Commons IO 2.6*

Se trata de una librería de utilidades que facilitan el tratamiento de I/O, empleadas tanto en el *Parser* de lectura como en el código de escritura de ficheros.

#### · *Apache POI 4.1.2*

Proporciona bibliotecas Java para gestionar y acceder a archivos de Microsoft. En este proyecto se empleó para archivos Excel, en concreto para la escritura de los mismos con los datos de los resultados de las instancias planificadas.

#### · *Apache Maven Plugins 3.3.0*

Se trata de una dependencia que hace posible trabajar con funcionalidades como copiar, desempaquetar, analizar, etc, proyectos. Se empleó para la creación de un archivo *.jar* del proyecto, que conforma uno de los entregables del trabajo.

#### · *JUnit Jupiter API 5.7.1*

*Junit Jupiter* es una dependencia que contiene todas las clases e interfaces necesarias para la implementación de la automatización de las pruebas tanto unitarias como de integración en un proyecto software, facilitando así la tarea de realizar pruebas en el sistema y asegurando así su consistencia y funcionalidad.

### GitHub

*GitHub,* o *Git* en general, se trata de un sistema de control de versiones de código libre. Es una herramienta que facilita la tarea de gestión del código según se implementa, haciendo posible la creación de distintas ramas en las cuales llevar a cabo distintas partes del mismo.

### Codecov

*Codecov* es una plataforma online cuya función es la detección de errores y problemas sintácticos en un determinado *software*. Se emplea para medir qué líneas de código se ejecutan en un *test*. Aplica la siguiente fórmula:

$$coverage = \frac{hits}{(sum\ of\ hit + partial + miss)}$$

Donde *hits* indica que el código ha sido ejecutado en el *test*, *partial* indica que el código no ha sido enteramente ejecutado en el *test,* es decir, que hay alguna rama que no se ha procesado, y *miss* indica aquel código que no ha sido ejecutado.

### Microsoft Project

*Project* es un software desarrollado por Microsoft creado para la administración y gestión de proyectos y programas que sirve para la evaluación de tareas con el objetivo de realizar una estimación sobre la duración, asignar recursos a estas tareas, y estimar el coste del proyecto en cuestión. Se hizo uso de este software para la planificación entera del proyecto, así como la disgregación de tareas y la estimación de la duración de este.

### Microsoft Word

*Word* es un software desarrollado por Microsoft destinado al procesamiento de texto, mediante el cual es posible crear documentos de texto y un procesamiento de palabras superior. Se hizo uso de este software para la redacción completa del trabajo.

### Microsoft PowerPoint

*PowerPoint* es un software de Microsoft destinado a la creación y gestión de diapositivas con el fin de obtener una presentación con texto esquematizado, animaciones de texto, imágenes prediseñadas o importadas desde el ordenador y demás funcionalidades. El uso de este software en el trabajo es el de la creación de una presentación que resuma el tema tratado con el fin de exponerlo.

### Microsoft Excel

*Excel* es otro software desarrollado por Microsoft que se encarga de gestionar hojas de cálculo. Cuenta con cálculo, gráficas, tablas calculares y un lenguaje de programación macro llamado *Visual Basic*. Consiste en una herramienta avanzada de análisis y visualización de datos. Se utilizó en este trabajo tanto para el guardado de datos resultantes del sistema como para el análisis experimental de estos de las instancias ejecutadas mediante tablas, gráficas y demás.

## 5.3 Problemas encontrados

En esta sección se resumen los problemas encontrados durante el desarrollo del proyecto.

### Apache POI 5.0.0

Empleando una de las últimas versiones de Apache POI para la gestión de escritura de datos en Excel, tras la implementación del pertinente código y la importación de los paquetes necesarios para ello, resultó en un *IOException*. No obstante, tras investigar en dónde se hallaría el error, se concluyó que con que el error solamente se daba al ejecutar la aplicación desde el archivo ".jar", y no desde el IDE, donde funcionaba a la perfección. Leyendo lo que indicaba el log de error, el fallo venía de que podría no haber compilado bien dos clases necesarias para la escritura de datos en una hoja de cálculo. Después de intentar arreglarlo, se pudo encontrar un artículo [25] que explicaba esto mismo aquí descrito y aportaba una solución, consistente en cambiar la versión de la dependencia por una anterior, correspondiente con la 4.1.2, por lo que se tomó la decisión de cambiar la versión y funcionó como se esperaba.

### Orden de elementos en HashSet

Concretamente para el cálculo de la prioridad de una tarea en las reglas de prioridad, específicamente para la ATC *(Apparent Tardiness Cost)* se decidió implementar una estructura de datos en la que se pudiese almacenar la tarea en cuestión junto a la prioridad de la misma para, una vez calculadas todas las prioridades de todas las tareas que le llegasen en esa iteración a la regla, se seleccionase aquella con mayor prioridad. En caso de coincidir estas prioridades, o resultar alguna indeterminación al realizar el cálculo y por tanto ser 0, se aplicaría una regla auxiliar, que podría en alguna situación llegar a quedarse sencillamente con la primera tarea que le llegase de la lista.

Aquí llega el problema, en el que cada vez que realizaba una ejecución de las instancias daban resultados distintos, dentro de un rango más o menos parecido, pero dispares, al fin y al cabo. Esto en el problema tratado no tiene sentido que suceda, ya que se abordan las mismas instancias con el mismo algoritmo y las mismas reglas de prioridad, por lo que los resultados han de ser los mismos en cada ejecución.

Tras depurar varias veces el código e ir comprobando diversos valores reparé en que, en algunas iteraciones variaba el tamaño de la lista de operaciones que le llegaba a la regla de prioridad. Indagando más en todo esto, observé que se debía a una elección errónea de la regla en algún determinado momento. Tras otras tantas comprobaciones siguiendo la ejecución de una instancia en concreto de forma manual y alguna búsqueda en internet por comprender qué podía pasar, pude lograr obtener la respuesta, que se basa en la forma en la que funciona la estructura de datos de *HashSet*. No se garantiza el poder iterar en un orden concreto esta estructura ya que no mantiene el orden de los elementos. Esto provoca que la "primera" tarea del *set* difiriese en una iteración en

concreto de la ejecución de una instancia a otra ejecución distinta de la misma instancia, ya que no guarda orden de inserción de elementos.

A pesar de que este problema podría haber sido resuelto utilizando otras estructuras de datos como *TreeSet* o *LinkedHashList,* la solución adoptada fue implementar en la ATC una clase anidada *MaxOperation* dentro de la clase *ATCRule*, con dos atributos: *operation* y *priority*, la tarea y un número decimal correspondiente a su prioridad, respectivamente.

```
static class MaxOperation {
     Operation operation;
     double priority;

     MaxOperation(Operation op, double priority) {
         this.operation = op;
         this.priority = priority;
     }
 }
```

*Código 2. Clase anidada en ATCRule para gestionar la relación entre una operación y su prioridad.*

### OpenCSV

La primera versión tanto de diseño como de implementación de la escritura de ficheros y generación de *output* en general incluía una dependencia de Maven llamada *OpenCSV*, que ofrece de una manera fácil una forma de guardar datos en un fichero *.csv*. Fue más adelante en el proyecto cuando se propuso una ejecución masiva de instancias – y, por tanto, de generación de hojas de cálculo – y surgió la necesidad de un *plugin* que permitiese guardar datos en distintas hojas de cálculo de un mismo archivo Excel. Por consiguiente, se integró la dependencia *Apache POI*.

### Regla de prioridad MCT

La regla MCT o *Minimum Completion Time* es una regla de prioridad consistente en elegir las tareas calculando su prioridad de la forma:

$$\pi_{i,j} = \frac{1}{\max(mr_i, time) + p_{ij}}$$

donde $mr_i$ representa el momento en el tiempo en el que la máquina *i* queda libre y no está procesando ninguna tarea, y *time* representa el momento actual de tiempo del sistema.

Inicialmente se planteó la implementación e inclusión de esta regla como una más de las reglas a incluir en el estudio experimental. Tras su implementación y utilización con las instancias de los bancos de ejemplo utilizados se comprobó que, sorprendentemente, se obtenían mejores resultados que con otras reglas y especialmente que la regla ATC, algo que no tenía ningún sentido al tratarse la regla ATC una de las reglas con las que se obtienen mejores resultados. Por ello, se procedió a investigar lo que estaba sucediendo.

Al analizar más en profundidad esta regla y su aplicación al problema objeto de estudio, se entendió que esta regla no es posible aplicarla dadas las restricciones presentes en el problema. El parámetro de *time* en este contexto se podría referir al *startTime* de cada operación, el cual se va actualizando en función de dos tipos de hechos:

- Se ha planificado una operación con la que comparte máquina en la que ser procesada y el tiempo de fin de esta operación es superior al *startTime*.
- Se ha planificado la operación inmediatamente anterior del mismo trabajo.

Esto hace que el *startTime* (es decir, el *time* del sistema) de todas las operaciones esté continuamente actualizado, y, por lo tanto, este sea siempre mayor que el $mr_i$ (Diagrama 16) o, en su defecto, igual (en el caso de que se ejecutase justo en el momento de $mr_i$). En el Diagrama 16 se muestra gráficamente que el *startTime* de un trabajo siempre va a ser mayor o igual que el $mr_i$ de la máquina en la que tenga que ejecutarse. Aquí, ocurre con la segunda tarea del trabajo $J_2$, que, por causa de que el tiempo de procesamiento de la operación que la precede (primera operación del $J_2$) es mayor que el tiempo de procesamiento de la operación que se ejecuta antes en la máquina $M_1$ (primera operación del trabajo $J_1$, hay un tiempo *(gap)* hasta que procede a ejecutarse esta segunda tarea del $J_2$. En otra situación en la que el tiempo de procesamiento de la primera operación del $J_2$ fuese menor que el de la primera operación del $J_1$, la segunda operación del $J_2$ se ejecutaría en el instante de tiempo $mr_{\theta_{1,1}}$.

Así, el tiempo de inicio *(startTime)* de $\theta_{2,2}$ ha de ser el máximo entre el tiempo de fin de $\theta_{1,1}$ (*startTime* de $\theta_{1,1}$ + *processingTime* de $\theta_{1,1}$), o lo que es lo mismo, $mr_{\theta_{1,1}}$, y el tiempo de fin de $\theta_{2,1}$ (*startTime* de $\theta_{2,1}$ + *processingTime* $\theta_{2,1}$).

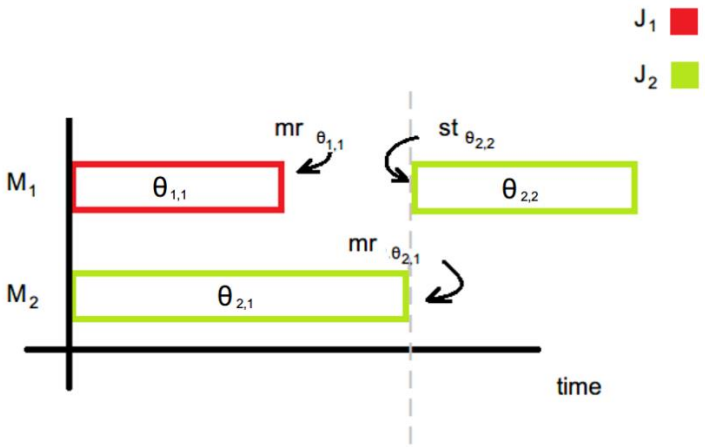



*Diagrama 16. Representación de dos trabajos cuyas operaciones son procesadas en dos máquinas.*

Por ello, el denominador de la fórmula siempre resultaría en: *startTime + processingTime*, lo cual no daría unos resultados fiables. Esta regla de prioridad podría ser más útil en un tipo de problema en el que cada operación pudiera tener varias máquinas en las que ser procesada, ya que daría más "juego" a la hora de seleccionar la alternativa más factible.

# CAPÍTULO 6: Análisis experimental

En el presente capítulo se aborda un estudio experimental que persigue analizar el comportamiento del prototipo implementado.

El estudio aborda la resolución, mediante el algoritmo G&T implementado en el prototipo, de instancias conocidas del problema JSS. Como ya se ha comentado a lo largo del documento, el algoritmo G&T puede ser guiado por diferentes reglas heurísticas, por lo que este estudio permitirá, además de calcular soluciones al problema, realizar una comparativa entre la calidad de las diferentes soluciones alcanzadas dependiendo de la regla de prioridad que haya guiado al algoritmo G&T. Además, estas soluciones serán comparadas con las mejores soluciones conocidas o con las mejores cotas superiores (*Upper Bound*, UB) o inferiores (*Lower Bound*, LB).

En la literatura podemos encontrar numerosos trabajos en los que se aborda la resolución del problema JSS y se proponen bancos de ejemplos, conocidos como *benchmarks*, para evaluar las estrategias propuestas. Aunque la función objetivo más estudiada en la literatura para el problema JSS sea el *makespan*, en este estudio experimental se analizarán también los resultados para la función objetivo *tardiness*.

Para el estudio experimental se empleará el siguiente banco de instancias, compuesto por instancias de Taillard (básicas y extendidas) y las restantes, instancias de Singer y Pinedo [27], detalladas a continuación:

- 80 instancias básicas de Taillard *(tai01 – tai80)*, obtenidas de la OR-library [26]
- 240 instancias de Taillard extendidas *(tai01 – tai80)*, obtenidas de [27]
- 6 instancias extendidas de J. Adams, E. Balas y D. Zawack *(abz5 – abz6)*, obtenidas de [27]
- 3 instancias extendidas de H. Fisher y G.L. Thompson (conocidas como *ft*, pero con nombre alternativo usado en el *benchmark* de *mt10*), obtenidas de [27]
- 27 instancias extendidas de S. Lawrence *(la16 – la24)*, obtenidas de [27]
- 30 instancias extendidas de D. Applegate y W. Cook *(orb1 – orb10)*, obtenidas de [27]

Considerando un total de **386 instancias** en este estudio.

Estas instancias serán resueltas por el algoritmo voraz empleando las reglas de prioridad consideradas en este trabajo. No obstante, no todas las reglas de prioridad pueden ser aplicadas en todas las instancias. Mientras que las reglas SPT *(Shortest Processing Time)* y LPT *(Longest Processing Time)* pueden ser ejecutadas en todas las instancias, las reglas EDD *(Earliest Due Date)* o ATC *(Apparent Tardiness Cost)* precisan de unos datos extra que no todas las instancias del estudio poseen, como ocurre con las 80 instancias de Taillard, que carecen de datos como el peso o la fecha límite. Son precisamente datos como estos los que convierten a una instancia en una instancia "extendida". Así, contaríamos con 80 instancias básicas de Taillard y 306 instancias

extendidas. Más adelante se explicará cómo se pueden generar instancias extendidas de instancias básicas, en el apartado 6.2.1 Generación de instancias para el JSSPTWT.

Es preciso también indicar las características del entorno de experimentación:

**Sistema operativo:** Windows 10 Pro 19041.1415

**Especificaciones:** AMD Ryzen 5 3600 3.60GHz, 16,0 GB RAM

**Java:** JDK 15.0.2

## 6.1 Análisis experimental del JSP con función objetivo del *makespan*

La primera y principal función objetivo a tratar en el problema del *Job Shop* se corresponde con el *makespan* o tiempo máximo de completitud, conocido como $J||C_{max}$ y que persigue minimizar el tiempo de fin de la última tarea procesada.

Probablemente se trate de la función objetivo más sencilla de someter a análisis, ya que las reglas de prioridad empleadas requieren de datos muy básicos, que están definidos en cualquier instancia de este tipo, o en su defecto, pueden ser fácilmente calculados, pero al abordar en este estudio problemas estáticos del problema JSS, toda la información necesaria de las instancias es conocida *a priori*. Es por esto por lo que, en el caso del *makespan,* es posible analizar el banco de instancias completo.

Para esta versión del problema con minimización del makespan, todas las instancias del *benchmark* contienen los datos necesarios para ejecutar las reglas SPT y LPT, que son las apropiadas para esta función objetivo. Sin embargo, dado que los datos manejados por las reglas seleccionadas en este trabajo son comunes tanto en las 240 instancias de Taillard extendidas como en las 80 instancias de Taillard básicas, experimentar con todas sería redundante.

Entonces, se probarán las 80 instancias de Taillard básicas obtenidas de la OR-library [26] [33] [34] para el problema *job shop scheduling*, y que se componen de:

- 10 instancias de tamaño 15x15.
- 10 instancias de tamaño 20x15.
- 10 instancias de tamaño 20x20.
- 10 instancias de tamaño 30x15.
- 10 instancias de tamaño 30x20.
- 10 instancias de tamaño 50x15.
- 10 instancias de tamaño 50x20.
- 10 instancias de tamaño 100x20.

En cuanto a las demás instancias mencionadas anteriormente, al tratarse todas de instancias extendidas (Taillard extendidas y Singer y Pinedo [27]), existen tres archivos por cada instancia, cada una con distintos factores de fechas límite (explicado en detalle en el apartado 6.2.1 Generación de instancias para el JSSPTWT). El resto de datos (*processing time*, número de trabajos, máquinas y operaciones), sin embargo, son los mismos, como se puede apreciar en la Ilustración 20. Ejemplo de la instancia extendida abz5 con los 3 factores de fecha límite. del apartado Anexo V: Formato del fichero de texto I/O, donde

se pueden ver las tres instancias distintas de la instancia *abz5*, cada una con un factor de fecha límite distinto.

Como consecuencia de todo ello, solo se tendrá en cuenta un archivo por cada instancia (por ejemplo, de la instancia *abz5*, el archivo *abz5-1.3*) de las instancias de Singer y Pinedo mencionadas, quedando:

- 2 instancias: *abz5, abz6* ambas de tamaño 10x10.
- 1 instancias *mt10* de tamaño 10x10.
- 9 instancias: *la16 – la24* de tamaño 10x10.
- 10 instancias: *orb1 – orb10* de tamaño 10x10.

Con un total, entre las 80 básicas de Taillard y las 22 de Singer y Pinedo extendidas, de 102 instancias para el *makespan*.

### 6.1.1 Instancias básicas de Taillard

Se comienza el análisis con las instancias de Taillard, en las que contamos con el dato de la cota superior para realizar comparaciones respecto a los resultados obtenidos. Como se comentó anteriormente, para el análisis de la función objetivo del *makespan* se ha hecho uso de la regla SPT y la LTP.

De manera resumida, calculando las medias de cada una de las reglas para las 80 instancias de Taillard, observamos una diferencia entre el rendimiento, en comparación con los datos de las cotas superiores, mostrados en la Figura 16 y en la Figura 17, que representan la mejor solución encontrada hasta el momento.

| SPT | LPT |
|---|---|
| 3915,825 | 3520,588 |

*Tabla 63. Resultados para las reglas empleadas en las instancias básicas de Taillard para el makespan y sus respectivas medias.*

Entendiendo que cuanto menor sea el tiempo de completitud – tiempo de fin de la última tarea – antes se resuelve la instancia, y, por tanto, mejor es el resultado, se observa en la Tabla 63 que el mejor resultado es obtenido por la regla LPT, mientras que el peor resultado (el más largo) lo da la SPT. Se puede observar en la Tabla 81 del Anexo V: Resultados del análisis experimental: Función objetivo: *makespan* los resultados detallados para cada instancia básica de Taillard. Se aprecia que no existe una gran diferencia, por lo general, entre los resultados para una misma instancia entre las dos reglas, aunque, como se manifiesta en las medias ponderadas, se obtienen ligeramente mejores resultados con la LPT.

Todo esto queda reflejado en la Figura 16 y la Figura 17, donde no se puede advertir una diferencia en cuanto a la tendencia de la regla usada. Solo en las últimas instancias es donde se percibe que la diferencia entre los valores llega a ser de mil unidades en el

tiempo entre ambas reglas. Estas unidades de tiempo, cabe aclarar, es el *makespan* resultante para cada instancia.

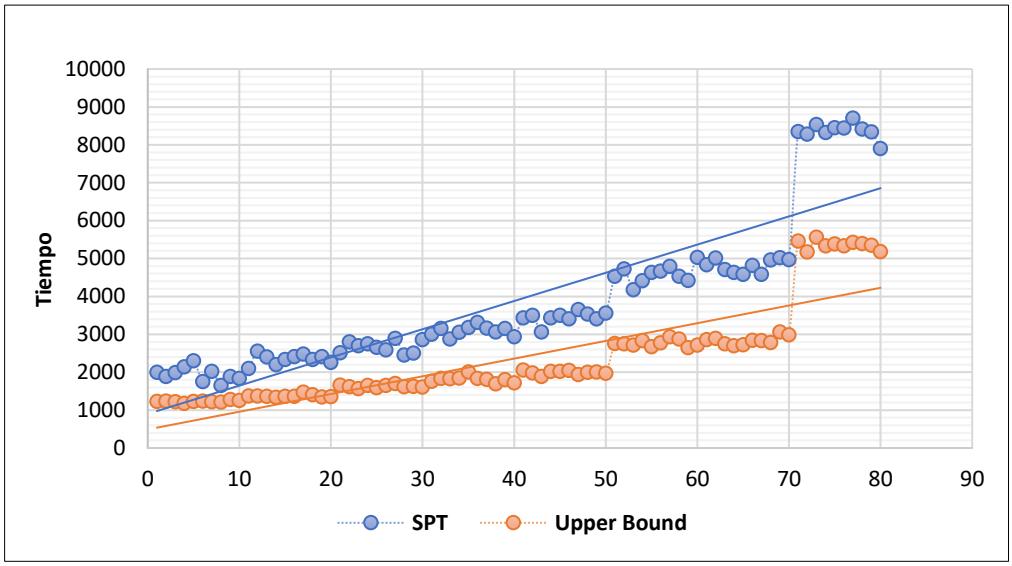



*Figura 16. Gráfica de la tendencia del makespan SPT - UP de las instancias de Taillard.*

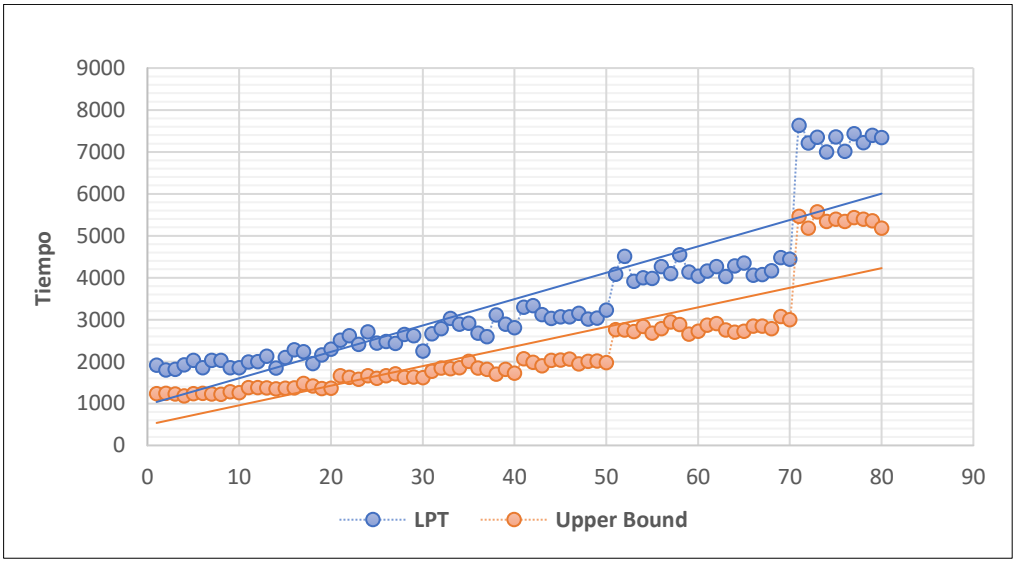



*Figura 17. Gráfica de la tendencia del makespan LPT - UP de las instancias de Taillard.*

### 6.1.2 Instancias extendidas de Singer y Pinedo

A continuación, se llevan a cabo las ejecuciones de las instancias de Singer y Pinedo citadas anteriormente y se comparan con datos de cotas superiores publicados en [28]. Las medias de las reglas se encuentran representadas en la Tabla 64, donde se vuelve a observar, como en el caso anterior de las instancias de Taillard, que no existe una gran diferencia en los resultados obtenidos entre la SPT y la LPT, dando mejores resultados en todos los casos la LPT, exceptuando en las instancias *abz*.

| | **SPT** | **LPT** |
|---|---|---|
| *abz* | 1566,5 | 1608,5 |
| *la* | 1259,1 | 1229,2 |
| *mt* | 1429 | 1355 |
| *orb* | 1338,3 | 1329,8 |

*Tabla 64. Reglas empleadas en las instancias de Singer y Pinedo para el makespan y sus respectivas medias.*

Para analizar mejor los resultados para estas instancias, se calcula la desviación típica (DT) σ, el error absoluto (EA) ε, y el error relativo (ER) δ, respecto a las cotas superiores (UB), detallados en la Tabla 65, junto con el mejor resultado obtenido entre la SPT y la LPT. Se observa que estos datos arrojan unos resultados algo elevados, y con una media de error relativo del 33%. Esto es debido a la diferencia que hay entre el mejor resultado obtenido de las heurísticas empleadas y la cota superior para cada una de las instancias.

| Instancia | UB | Makespan | | | | |
|---|---|---|---|---|---|---|
| | | Mejor | Media | DT | EA | ER |
| *abz5-1.3* | 1258 | 1733 | 1784,5 | 372,3 | 475 | 38% |
| *abz6-1.3* | 952 | 1381 | 1390,5 | 310,1 | 429 | 45% |
| *la16-1.3* | 1012 | 1336 | 1400 | 274,4 | 324 | 32% |
| *la17-1.3* | 787 | 1122 | 1125 | 239,0 | 335 | 43% |
| *la18-1.3* | 954 | 1250 | 1396 | 312,5 | 296 | 31% |
| *la19-1.3* | 861 | 1120 | 1150,5 | 204,7 | 259 | 30% |
| *la20-1.3* | 920 | 1250 | 1343 | 299,1 | 330 | 36% |
| *la21-1.3* | 1092 | 1149 | 1164 | 50,9 | 57 | 5% |
| *la22-1.3* | 955 | 1031 | 1100,5 | 102,9 | 76 | 8% |
| *la23-1.3* | 1049 | 1250 | 1276 | 160,5 | 201 | 19% |
| *la24-1.3* | 971 | 1181 | 1242,5 | 192,0 | 210 | 22% |
| *mt10-1.3* | 970 | 1355 | 1392 | 298,4 | 385 | 40% |
| *orb1-1.3* | 1145 | 1481 | 1485,5 | 240,8 | 336 | 29% |
| *orb2-1.3* | 918 | 1175 | 1254,5 | 237,9 | 257 | 28% |
| *orb3-1.3* | 1098 | 1473 | 1505,5 | 288,1 | 375 | 34% |
| *orb4-1.3* | 1066 | 1618 | 1622 | 393,2 | 552 | 52% |
| *orb5-1.3* | 911 | 1180 | 1204 | 207,2 | 269 | 30% |
| *orb6-1.3* | 1050 | 1527 | 1531,5 | 340,5 | 477 | 45% |
| *orb7-1.3* | 414 | 552 | 565,5 | 107,1 | 138 | 33% |
| *orb8-1.3* | 945 | 1183 | 1261 | 223,4 | 238 | 25% |
| *orb9-1.3* | 978 | 1389 | 1398 | 297,0 | 411 | 42% |
| *orb10-1.3* | 991 | 1473 | 1513 | 369,1 | 482 | 49% |
| | | | | | | 33% |

*Tabla 65. Comparativa del makespan obtenido para las instancias de Singer y Pinedo en función de su UB.*

Se pueden ver los resultados completos de cada instancia y por cada regla en la Tabla 82 del Anexo V: Resultados del análisis experimental: Función objetivo: *makespan*.

## 6.2 Análisis experimental del JSS con función objetivo del *tardiness*

En este apartado se aborda el problema Job Shop Scheduling (JSS) tratando la función objetivo del *tardiness*, conocido como J || $\Sigma T_i$ y denominado *Job Shop Scheduling Problem* con la minimización del *Total Weighted Tardiness*, lo que se abrevia como JSSPTWT.

El *tardiness* o retraso de un trabajo se corresponde con la diferencia entre el tiempo de completitud de un trabajo y la fecha límite *(due date)* del mismo. En caso de ser negativo estaríamos hablando de *earliness*, es decir, la finalización de un trabajo sería anterior a su fecha límite, y no se tendría en cuenta para el cálculo de la función objetivo *Total Weighted Tardiness*. Existen numerosos ámbitos en donde el factor más importante a la hora de la realización de tareas es cumplir con una fecha límite, ya que de no hacerlo podría conllevar grandes pérdidas económicas, por esto esta función objetivo es muy interesante en el ámbito de los problemas reales que pueden ser modelados como un problema JSS.

Para este tipo de función objetivo es necesario proporcionar una serie de datos, en base a la regla de prioridad que se vaya a emplear. Si bien un dato imprescindible es la mencionada fecha límite o *due date* de cada trabajo, a raíz de la cual se podrá calcular el *tardiness*, también se podrán aportar otros datos, como son el peso o *weight* de un trabajo, u otras condiciones de entrada.

### 6.2.1 Generación de instancias para el JSSPTWT

En la mayoría de instancias para el problema JSS no se especifican datos como la fecha de disponibilidad (*release date*) $r_i$, el peso (*weight*) $w_i$ o una fecha límite (*due date*) $d_i$ para cada trabajo. El due date es necesario para calcular el *tardiness* de cada trabajo mediante la expresión $T_i = \max(C_i - d_i,\ 0)$, donde $C_i$ es el tiempo de finalización y $d_i$ es la fecha límite o *due date* del trabajo $J_i$. Es por esto por lo que se ha de buscar una forma de generar estos datos a partir de las instancias ya conocidas.

Con el fin de adaptar las instancias del problema JSS al JSSPTWT, Singer y Pinedo propusieron una manera de incorporar estos datos [35]. Según su enfoque, cada trabajo $j$ recibe una fecha de disponibilidad $r_i = 0$. Y, a continuación, con la ayuda de distintos factores de fechas límites $f \in \{1.3, 1.5, 1.6\}$ se calcula la fecha límite $d_i$ de la forma:

$$d_i = \left\lfloor r_i + f \cdot \sum_{i=1}^{m} p_{ij} \right\rfloor$$

donde $m$ es el número de máquinas, que coincide con el número de operaciones por trabajo de la instancia –, y $p_{ij}$ es el tiempo de procesamiento de un trabajo $i$ y de una operación $j$.

Los tres factores de fecha límites diferentes propuestos por Singer y Pinedo, hacen que las *due dates* generadas sean estrictamente distintas. Así, $f$ = 1.3 producirá fechas límite más rigurosas, es decir, más ajustadas, mientras que $f$ = 1.5 producirá fechas límite más moderadas y $f$ = 1.6 producirá fechas límite más holgadas respecto a las anteriores.

Los pesos o *weights* de cada trabajo se generan de la siguiente manera: el primer 20% de los trabajos reciben un peso de 4, el siguiente 60% de los trabajos recibe un peso de 2, y el 20% final de los trabajos recibe un peso de 1. El peso de un trabajo representa la importancia del trabajo, de manera que cuanto mayor sea su peso más importante es completar el trabajo a tiempo en la medida de lo posible.

A través de este planteamiento, se pueden extender las instancias del problema JSS, calculando estos *due dates* y pesos, al JSSPTWT, como las 80 instancias de Taillard que se pueden encontrar en la OR-library, obteniéndose 3 instancias de cada una de ellas y pasando a ser un total de 240 (cada instancia con distintas variaciones de fechas límite: 1.3, 1.5 y 1.7). Y, por otro lado, contamos con las instancias de Singer y Pinedo ya extendidas que se mencionaron previamente en la descripción del banco de instancias: las instancias de 10x10 de Adams et al. (abz05-abz06), Fisher y Thompson (mt10), Lawrence (la16-la24) y Applegate y Cook (orb1-orb10).

### 6.2.2 Resultados

Para la realización de este estudio se necesitan más datos que en el caso del *makespan*, por lo que es crucial que las instancias a tratar incluyan los datos requeridos. Así pues, para las reglas de prioridad empleadas, los datos de las instancias deberán incluir las fechas límite *(due dates)* y los pesos *(weights)*. Tras la generación de las versiones extendidas, en aquellas instancias que lo requieran, como se explica en el punto 6.2.1 Generación de instancias para el JSSPTWT, el banco de instancias cuenta con:

- 240 instancias de Taillard extendidas:
  - 30 instancias de tamaño 15x15.
  - 30 instancias de tamaño 20x15.
  - 30 instancias de tamaño 20x20.
  - 30 instancias de tamaño 30x15.
  - 30 instancias de tamaño 30x20.
  - 30 instancias de tamaño 50x15.
  - 30 instancias de tamaño 50x20.
  - 30 instancias de tamaño 100x20.

Y 66 de Singer y Pinedo:

- 6 instancias: 3 de *abz5,* 3 de *abz6*; todas de tamaño 10x10.
- 3 instancias *mt10* de tamaño 10x10.
- 27 instancias desde *la16* hasta *la24*, 3 de cada una con tamaño de 10x10.
- 30 instancias desde *orb1* hasta *orb10*, 3 de cada una con tamaño de 10x10.

Con un total de 306 instancias para el *tardiness*.

Comenzando con las instancias de Taillard extendidas, y aplicando las dos reglas de prioridad anteriores (SPT y LPT) junto con otras dos nuevas (EDD y ATC), que operan con parámetros relacionados con la función objetivo a tratar. Nos encontramos, pues, con tres instancias por cada una de las básicas iniciales, una por cada factor de fecha límite $f \in \{1.3, 1.5, 1.6\}$. Esto supone una diferencia en las fechas límite de los trabajos de unas 100 unidades entre un fichero y otro (de la misma instancia *taixx*). Resulta interesante por lo tanto estudiar el comportamiento del algoritmo y las distintas reglas de prioridad en los distintos factores.

Así, realizando la suma de los valores obtenidos del *tardiness* en las instancias de factor 1.3, 1.5 y 1.6 obtenemos la suma de las unidades de retardo totales mostradas en la Tabla 66. Es en la Tabla 67 donde se ve la media de estos datos, y se aprecia de manera más sencilla que el mejor resultado se obtiene en las reglas que evalúan datos que tienen relación con el *tardiness*, como lo es la EDD. También, es preciso puntualizar la diferencia de resultados entre los diferentes factores de fechas límites, obteniendo de manera evidente mejores resultados – menor tiempo de retardo – en las instancias con factor 1.6, y peores resultados en aquellas con factor 1.3, algo lógico si se tiene en cuenta que el factor 1.3 genera *due dates* más ajustados, por lo que es más difícil que no haya retrasos, mientras que el factor 1.6 genera *due dates* más holgados, lo que hace que se produzcan menos retrasos.

| | | | | ATC | | | |
|---|---|---|---|---|---|---|---|
| | **SPT** | **LPT** | **EDD** | **0,25** | **0,5** | **0,75** | **1** |
| ***1.3*** | 6977809 | 7044773 | 4874627 | 5652576 | 5638466 | 5630459 | 5638985 |
| ***1.5*** | 6413208 | 6478688 | 4350394 | 5037331 | 5042274 | 5033793 | 5041490 |
| ***1.6*** | 6137885 | 6201815 | 4101345 | 4770367 | 4745092 | 4744151 | 4773934 |

*Tabla 66. Suma del tardiness de cada instancia del benchmark de Taillard extendidas en función de su factor de fecha límite y por cada regla empleada.*

| | | | | ATC | | | |
|---|---|---|---|---|---|---|---|
| | **SPT** | **LPT** | **EDD** | **0,25** | **0,5** | **0,75** | **1** |
| ***1.3*** | 87222,6 | 88059,6 | 60932,8 | 70657,2 | 70480,8 | 70380,7 | 70487,3 |
| ***1.5*** | 80165,1 | 80983,6 | 54379,9 | 62966,6 | 63028,4 | 62922,4 | 63018,6 |
| ***1.6*** | 76723,6 | 77522,7 | 51266,8 | 59629,6 | 59313,6 | 59301,9 | 59674,2 |

*Tabla 67. Media del tardiness de las instancias del benchmark de Taillard extendidas en función de su factor de fecha límite y por cada regla empleada.*

Además, dentro de los resultados de la regla ATC, es interesante analizar los diferentes resultados en función del parámetro de escala *g* = [0, 1]. En el trabajo se han empleado los diferentes valores para *g* = {0.25, 0.5, 0.75, 1}. Evaluando el promedio de estos valores de todas las instancias ejecutadas, obtenemos peores resultados en los valores más bajos, así como en los más altos, representado en la Tabla 68 e ilustrado en la Figura 18, siendo más favorables aquellos que se encuentran en el intervalo [0.5, 0.75], incluso tendiendo más a aquellos valores más próximos a 0.75.

| ATC | | | |
|---|---|---|---|
| 0,25 | 0,5 | 0,75 | 1 |
| 64417,8 | 64274,3 | 64201,7 | 64393,3 |

*Tabla 68. Promedio de los resultados de tardiness obtenidos en los distintos valores de g empleados en la regla ATC en las instancias de Taillard extendidas.*

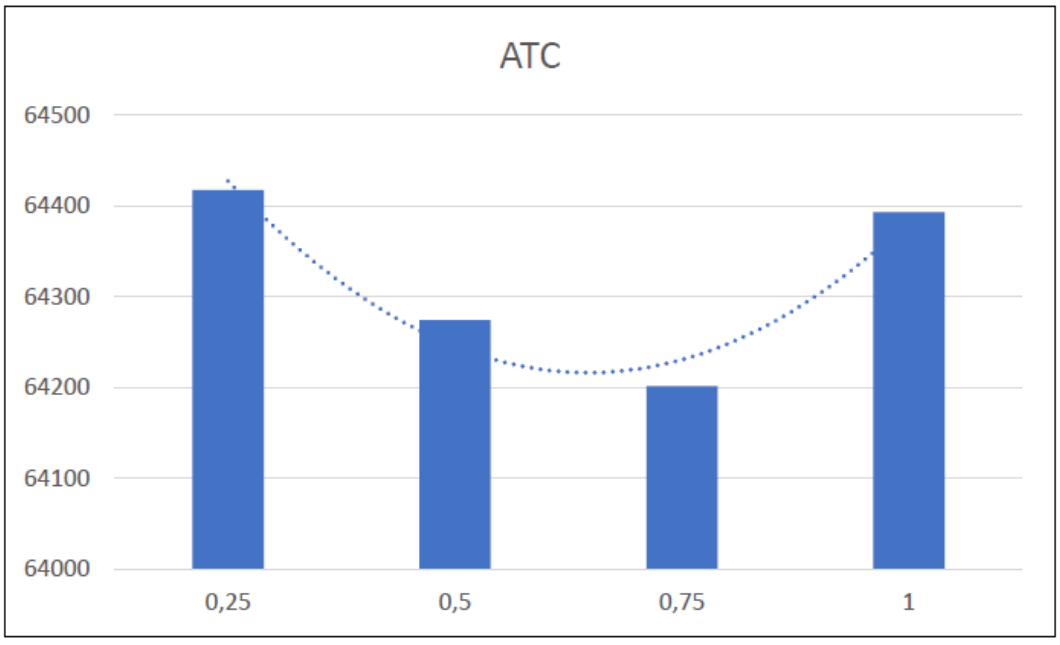



*Figura 18. Promedio de los resultados de tardiness obtenidos en los distintos valores de g empleados en la regla ATC con la tendencia que siguen en las instancias de Taillard extendidas.*

Se realiza el análisis experimental para el grupo de instancias restantes del banco de Singer y Pinedo [27]. Análogamente al análisis de las instancias de Taillard extendidas, se comienza con una comparativa entre los factores de fechas límite de las instancias. Así, separando el tipo de instancia y realizando una suma de los valores obtenidos, llegamos

a la misma situación que con las instancias anteriores: se observan diferencias entre los datos obtenidos en las instancias con factor 1.3 a los obtenidos en las instancias con factor 1.6, plasmados en detalle en la Tabla 69, la Tabla 70, la Tabla 71 y la Tabla 72, obteniéndose los mejores resultados – es decir, menores tiempos de retardo – en las instancias con factor 1.6 y los peores en las instancias con factor 1.3. Asimismo, se vuelve a observar que, por lo general los mejores resultados sin importar el factor de fechas límite los arroja la regla EDD junto con la ATC.

| | | | | ATC | | | |
|---|---|---|---|---|---|---|---|
| *abz* | **SPT** | **LPT** | **EDD** | **0,25** | **0,5** | **0,75** | **1** |
| ***1.3*** | 6483 | 7031 | 4952 | 7243 | 6237 | 5941 | 5834 |
| ***1.5*** | 4197 | 4745 | 3230 | 5110 | 4032 | 3919 | 3586 |
| ***1.6*** | 3162 | 3871 | 2485 | 3848 | 3286 | 3600 | 3036 |

*Tabla 69. Suma de resultados obtenidos del tardiness para las diferentes instancias abz.*

| | | | | ATC | | | |
|---|---|---|---|---|---|---|---|
| *la* | **SPT** | **LPT** | **EDD** | **0,25** | **0,5** | **0,75** | **1** |
| ***1.3*** | 19248 | 25305 | 18136 | 19010 | 18993 | 20237 | 19154 |
| ***1.5*** | 12134 | 17719 | 11669 | 12662 | 11795 | 13298 | 13152 |
| ***1.6*** | 9191 | 14640 | 9000 | 9186 | 9794 | 10337 | 10288 |

*Tabla 70. Suma de resultados obtenidos del tardiness para las diferentes instancias la.*

| | | | | ATC | | | |
|---|---|---|---|---|---|---|---|
| *mt* | **SPT** | **LPT** | **EDD** | **0,25** | **0,5** | **0,75** | **1** |
| ***1.3*** | 2404 | 4803 | 1999 | 2627 | 2526 | 2403 | 2403 |
| ***1.5*** | 1594 | 3896 | 1184 | 1641 | 1753 | 1523 | 1523 |
| ***1.6*** | 1306 | 3446 | 836 | 1217 | 1432 | 1447 | 1447 |

*Tabla 71. Resultados obtenidos del tardiness para la instancia mt.*

| | | | | ATC | | | |
|---|---|---|---|---|---|---|---|
| *orb* | **SPT** | **LPT** | **EDD** | **0,25** | **0,5** | **0,75** | **1** |
| ***1.3*** | 31509 | 38963 | 24361 | 29102 | 28163 | 27711 | 28008 |
| ***1.5*** | 23026 | 30513 | 16876 | 20396 | 20703 | 19280 | 20044 |
| ***1.6*** | 19159 | 26675 | 13479 | 16207 | 16273 | 15707 | 17348 |

*Tabla 72. Suma de resultados obtenidos del tardiness para las diferentes instancias orb.*

Veamos ahora, en la Tabla 73, cómo de lejos se encuentran los mejores resultados obtenidos en este trabajo para cada instancia (el *tardiness* más bajo obtenido), respecto a las BKS (*best known solutions* o mejores soluciones conocidas) publicadas para algunas de las instancias consideradas en [29].

| Inst | $f$ = 1.3 | | $f$ = 1.5 | | $f$ = 1.6 | |
|---|---|---|---|---|---|---|
| | BKS | Mejor | BKS | Mejor | BKS | Mejor |
| *abz5* | 1403 | 3325 | 69 | 2131 | 0 | 1916 |
| *abz6* | 436 | 1627 | 0 | 825 | 0 | 488 |
| *la16* | 1169 | 1980 | 166 | 1231 | 0 | 909 |
| *la17* | 899 | 1839 | 260 | 1109 | 65 | 834 |
| *la18* | 929 | 2099 | 34 | 1315 | 0 | 961 |
| *la19* | 948 | 1795 | 21 | 1063 | 0 | 759 |
| *la20* | 805 | 1872 | 0 | 1105 | 0 | 822 |
| *la21* | 463 | 1284 | 0 | 569 | 0 | 303 |
| *la22* | 1064 | 1991 | 196 | 1319 | 0 | 1026 |
| *la23* | 835 | 1822 | 2 | 1128 | 0 | 847 |
| *la24* | 835 | 1794 | 82 | 1069 | 0 | 780 |
| *mt10* | 1363 | 1999 | 394 | 1184 | 141 | 836 |
| *orb1* | 2568 | 2522 | 1098 | 1694 | 566 | 1298 |
| *orb2* | 1408 | 1966 | 292 | 1238 | 44 | 947 |
| *orb3* | 2111 | 2882 | 918 | 1964 | 422 | 1570 |
| *orb4* | 1623 | 2673 | 358 | 1564 | 66 | 1223 |
| *orb5* | 1593 | 1777 | 405 | 1266 | 163 | 1039 |
| *orb6* | 1790 | 2932 | 426 | 1999 | 28 | 1577 |
| *orb7* | 590 | 1105 | 50 | 716 | 0 | 544 |
| *orb8* | 2429 | 3027 | 1023 | 2268 | 621 | 1913 |
| *orb9* | 1316 | 2177 | 297 | 1419 | 66 | 1107 |
| *orb10* | 1679 | 2616 | 346 | 1798 | 76 | 1419 |

*Tabla 73. Comparativa de instancias de Singer y Pinedo entre el mejor resultado obtenido y la mejor solución conocida.*

Teniendo en consideración las 66 instancias aquí ejecutadas, haciendo el promedio de los resultados del *tardiness* de cada regla para todas las instancias, se observa que al igual que en el caso del *makespan* y como se comentó anteriormente, se obtienen los mejores resultados con la regla EDD, que es la que menores tiempos de retardo ocasiona. Como se observa en la Tabla 74, seguidamente encontramos la regla ATC. Las reglas que peores soluciones proporcionan son, sin duda alguna, la SPT y la LPT, algo totalmente normal pues solo tienen en cuenta el tiempo de procesamiento y no datos importantes en esta versión del problema JSS como son los *due dates*.

| | | | ATC | | | |
|---|---|---|---|---|---|---|
| SPT | LPT | EDD | 0,25 | 0,5 | 0,75 | 1 |
| 2021,4 | 2751,6 | 1639,5 | 1943,1 | 1893,7 | 1900,1 | 1906,4 |

*Tabla 74. Promedio de los resultados del tardiness de las instancias de Singer y Pinedo.*

Analizando la regla ATC, se aprecia visualmente en la Figura 19 que ocurre algo similar a los datos reflejados en la Figura 18, y al igual que ocurría en esta última, los mejores resultados se obtienen con el valor de escala $g$ = [0.5, 0.75]. Debido al tamaño de las instancias aquí empleadas (todas de 10x10) no se aprecia tanto, como en la anterior gráfica, la diferencia de valores, variando esta diferencia en ±60 unidades de tiempo.

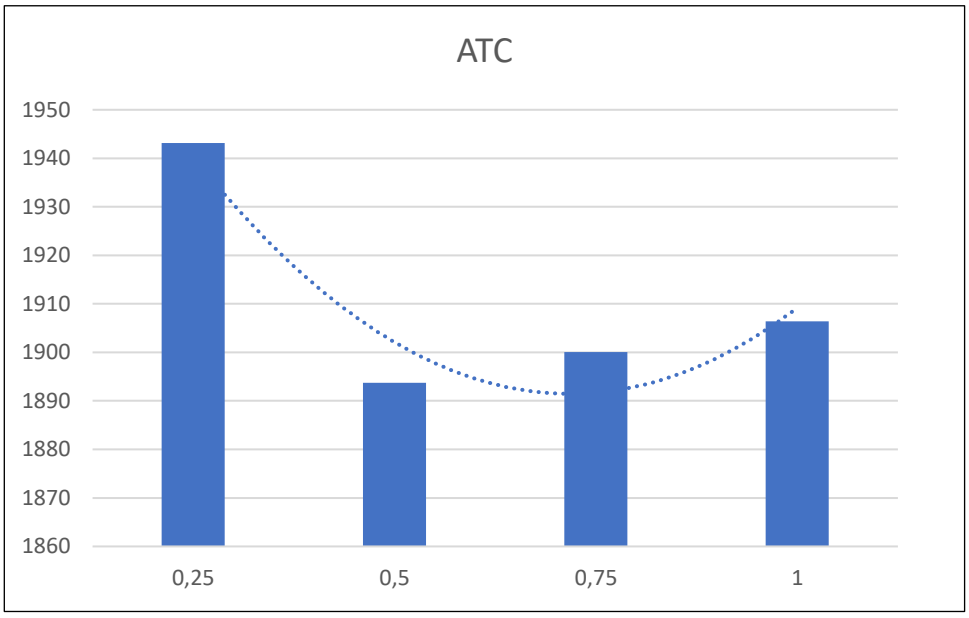



*Figura 19. Promedio de los resultados de tardiness obtenidos en los distintos valores de g empleados en la regla ATC con la tendencia que siguen en las instancias de Singer y Pinedo.*

Análogamente al apartado del *makespan*, se pueden ver los resultados completos de cada instancia y por cada regla en la Tabla 85 y la Tabla 86 del anexo Función objetivo: *tardiness*.

# CAPÍTULO 7: Manuales del sistema

## 7.1 Manual de instalación

Para la correcta ejecución del sistema descrito, al tratarse de un archivo de extensión ".jar", será necesario tener instalado *Java Development Kit (JDK)* desde la página oficial [30]. Deberá ser la versión 15.0.2 o superior. Debido a que se trata de un archivo para ejecutar en la interfaz de línea de comandos, no es necesario tener instalado ningún software o herramienta adicional.

## 7.2 Manual de ejecución

La aplicación consiste en un archivo de extensión ".jar" que, para poder usarlo correctamente con una instancia o un directorio que contenga una o varias instancias, se le puede:

- pasar la ruta absoluta de la instancia o directorio de instancias que se desea probar.
- guardar el fichero o directorio en la misma ruta que el archivo ".jar" e indicarle solamente el nombre del mismo.

Así, se podrá ejecutar el archivo ".jar" haciendo uso de la consola de comandos, como muestra la Ilustración 2.

## 7.3 Manual del usuario

El prototipo puede ejecutarse en dos escenarios distintos: para un solo archivo y para un conjunto de archivos (contenidos en un directorio), por lo que se mostrará a continuación un ejemplo de ambos casos.

La ejecución del archivo ".jar" con el prototipo, archivo "JSSP.jar", se realiza en línea de comandos indicando los argumentos de entrada que recibe separados por espacios en blanco de la forma:

*java -jar JSSP.jar arg1 arg2 [arg3]*

El *arg1* indica la **fución objetivo** {"m" || "t"}: "m" para el *makespan*, "t" para el *tardiness*.

El *arg2* es el **nombre del archivo** con la instancia a ejecutar o el nombre del **directorio** en donde se encuentra la instancia o conjunto de instancias a ejecutar.

El *arg3* es opcional y si se pone contiene el valor <"e">, expresando que las instancias a resolver son instancias extendidas.

A continuación, se muestra un ejemplo de ejecución de una sola instancia con minimización del *makespan*, que quedaría un comando de entrada como el mostrado en la consola de comandos de la Ilustración 3.

Se obtendrá la salida por consola que se ve en la Ilustración 4, donde además se puede ver también que se genera un nuevo directorio “out”, en el mismo directorio en el que se encuentra el ".jar", "JSSP.jar", y que contendrá el archivo Excel con los resultados, nombrado con el nombre del problema a resolver seguido de un espacio y de la letra que indica la función objetivo, en este caso como es el *makespan*, el nombre de la Excel sería: "tai02x03 m.xlsx" (Ilustración 5).

Dentro de este archivo Excel, cada fila contiene los resultados para cada instancia ejecutada. Así, en las filas se muestra el nombre del archivo en el que se encuentra cada instancia, seguido de su dimensión y los resultados de las distintas reglas empleadas en la resolución de la misma. El archivo generado para el ejemplo aquí mencionado se ve en la Ilustración 6, donde se aprecia que la dimensión de la instancia es de 2x3 y, en las siguientes columnas aparecen los resultados obtenidos para las reglas SPT y LPT.

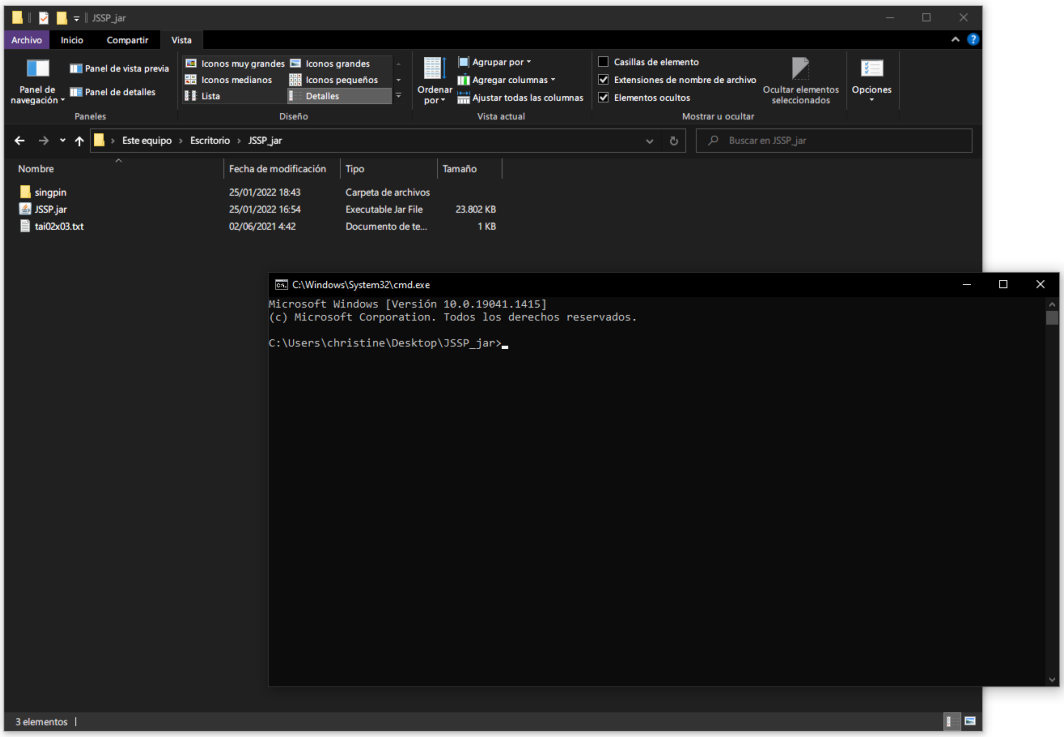



*Ilustración 2. Consola de comandos en el directorio del ".jar".*

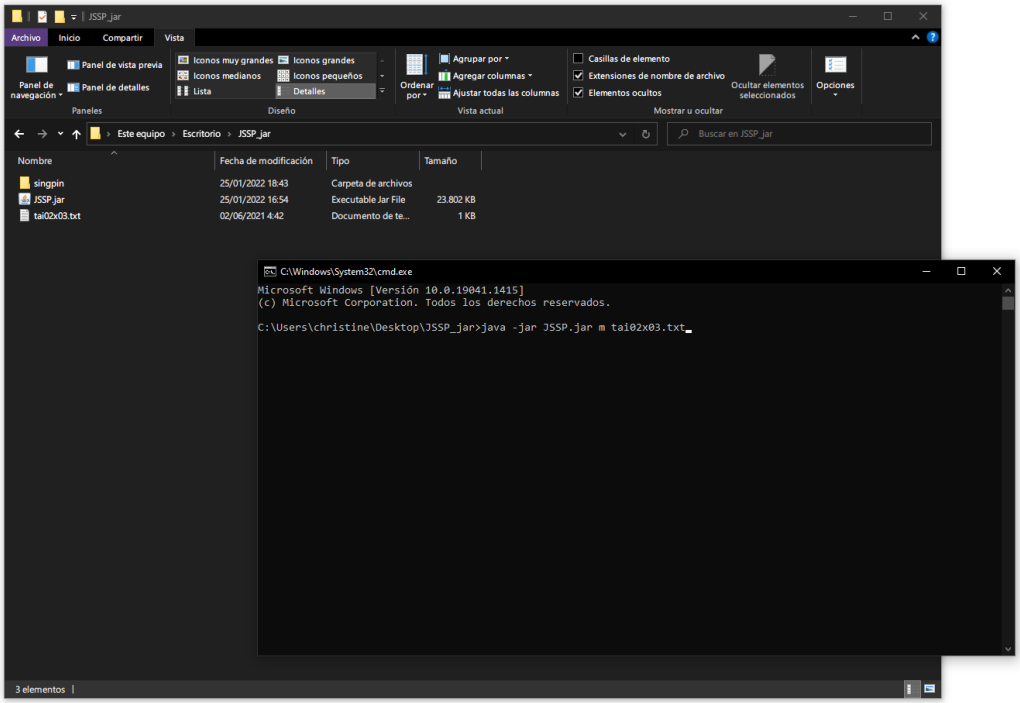



*Ilustración 3. Ejemplo de ejecución del archivo tai02x03.txt con minimización del makespan.*

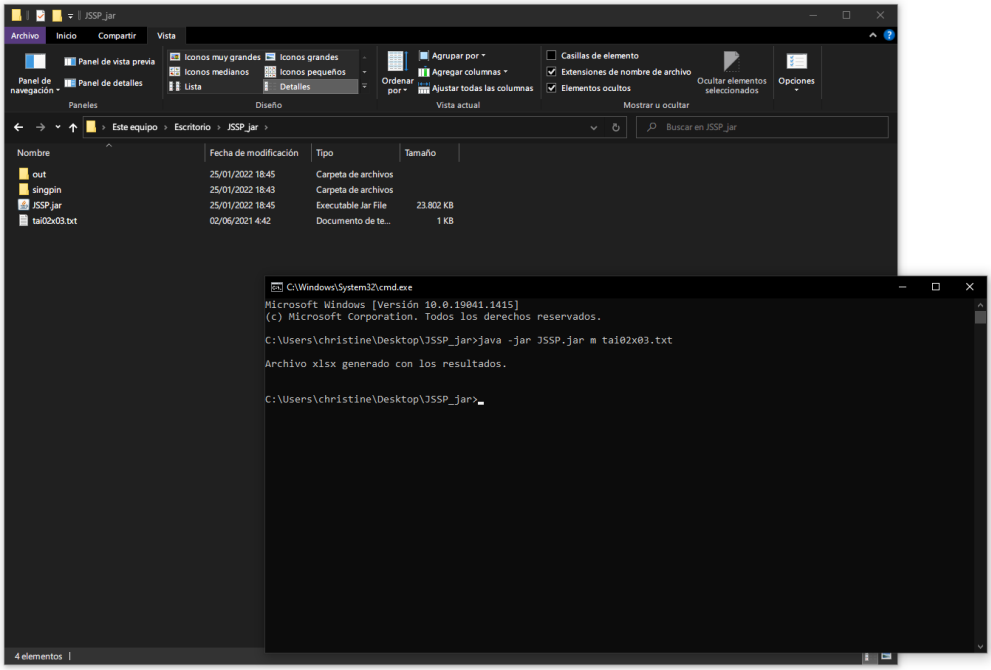



*Ilustración 4. Resultado de la consola de comandos del ejemplo de ejecución del archivo tai02x03.txt con minimización del makespan.*

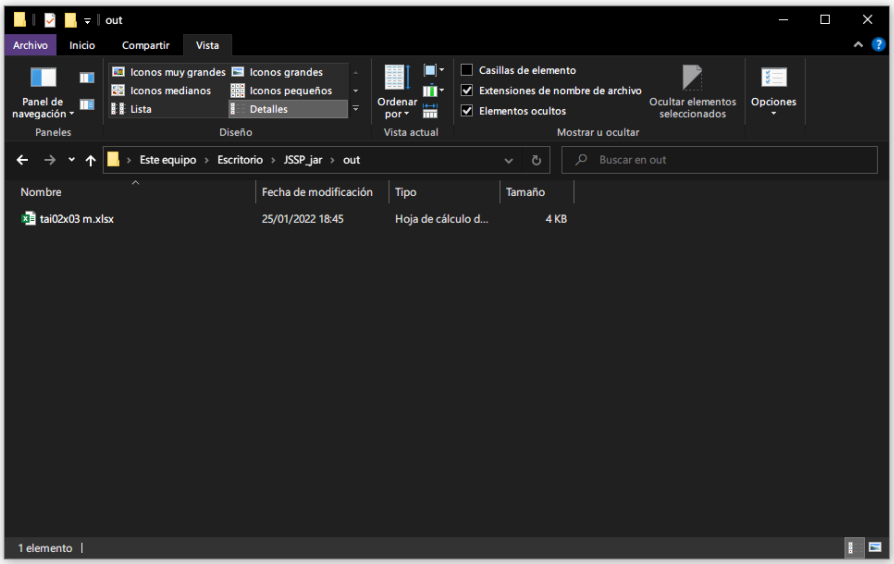



*Ilustración 5. Directorio con el archivo Excel resultante del ejemplo de ejecución del archivo tai02x03.txt con minimización del makespan.*

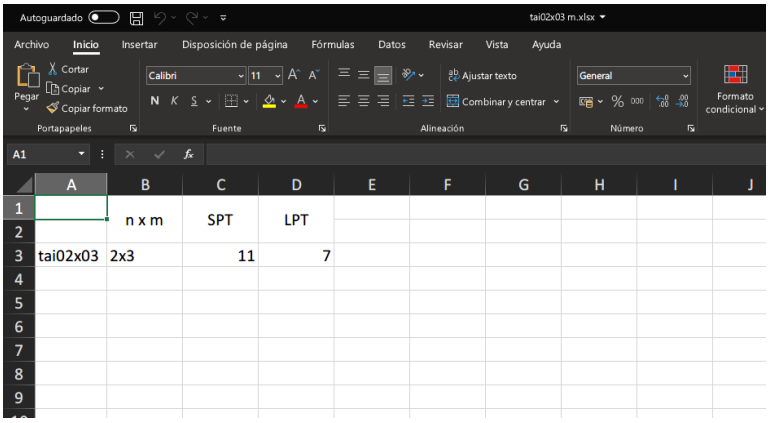



*Ilustración 6. Contenido del archivo Excel resultante del ejemplo del archivo tai02x03.txt con minimización del makespan.*

Para el ejemplo de ejecución de un directorio de instancias se ve en la Ilustración 7 que el cambio respecto a la ejecución de una sola instancia se basa en escribir el nombre de la carpeta contenedora de las instancias en lugar de la instancia junto con la extensión ".txt". En este caso, se ejecutarán las instancias extendidas del problema JSS con función objetivo del *tardiness*, que se encuentran en el directorio "singpin". Por lo que se le pasará la "t" como primer argumento, como segundo el nombre del directorio "singpin", y como tercero la "e" al tratarse de instancias extendidas.

Como se observa en la Ilustración 8, en este caso el directorio "out", que contiene la Excel con los resultados, se genera dentro del directorio donde se encuentran las

instancias, en este caso dentro del directorio "singpin". La Excel con los resultados de estas ejecuciones tendrá como nombre, en este caso, el nombre del directorio en el que se encuentran las instancias ejecutadas seguido de un espacio y la letra que indica la función objetivo, en este caso "t" al tratarse del *tardiness*, es decir "singpin t" como se observa en la Ilustración 15.

Al entrar dentro del archivo Excel (Ilustración 10) observamos una instancia por cada fila y su *tardiness* en función de la regla de prioridad empleada en cada columna, correspondientes con las reglas SPT, LPT, EDD y ATC. Se escribe un valor por cada instancia y por cada regla, a excepción de la regla de prioridad ATC *(Apparent Tardiness Cost)*, en la que se calcula el *tardiness* para los valores de *g* = {0,25; 0,5; 0,75; 1}.

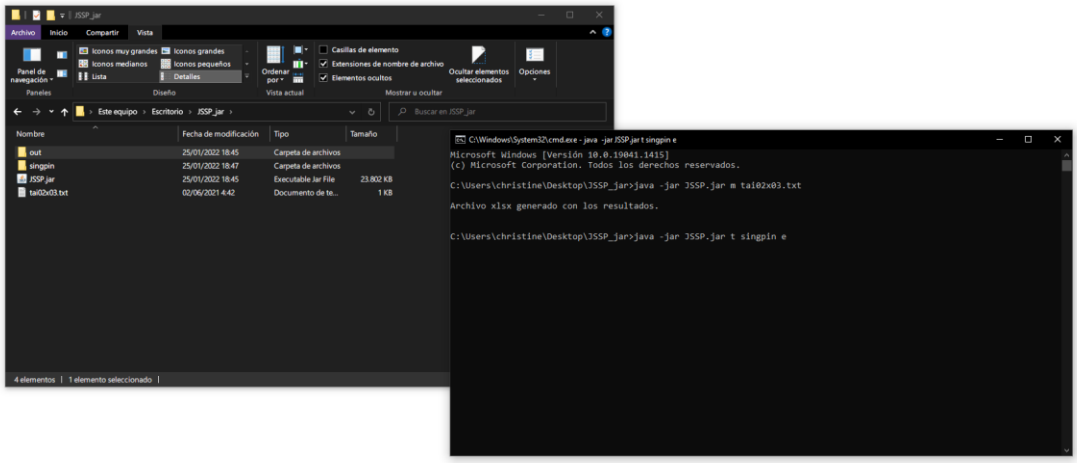



*Ilustración 7. Ejemplo de ejecución del directorio singpin con instancias de Singer y Pinedo extendidas con minimización del tardiness.*

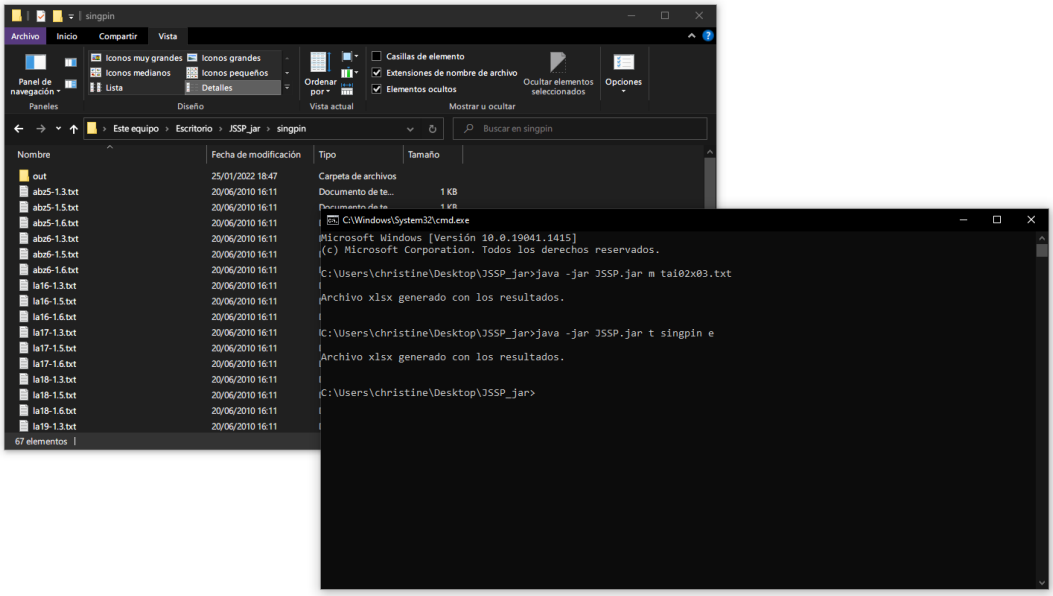



*Ilustración 8. Resultado de la consola de comandos del ejemplo de ejecución del directorio singpin con instancias de Singer y Pinedo extendidas con minimización del tardiness.*

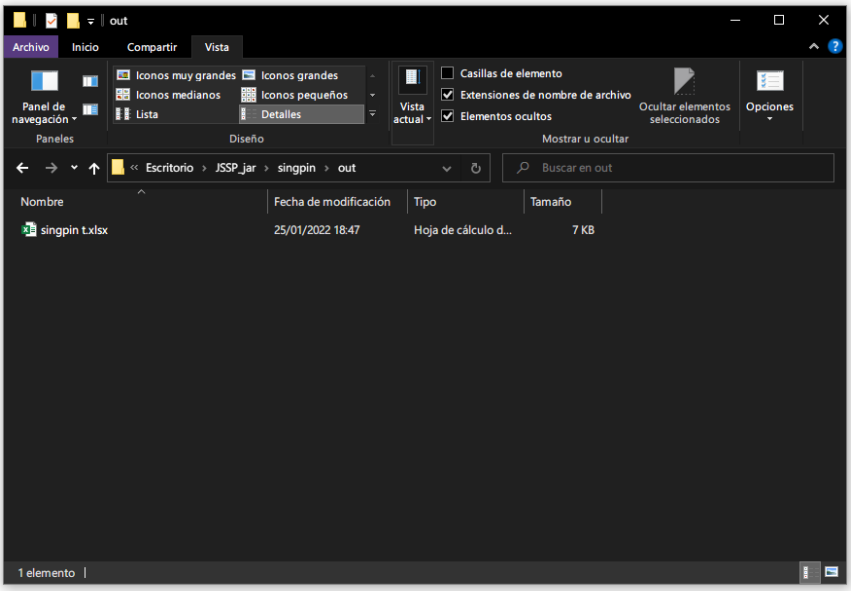
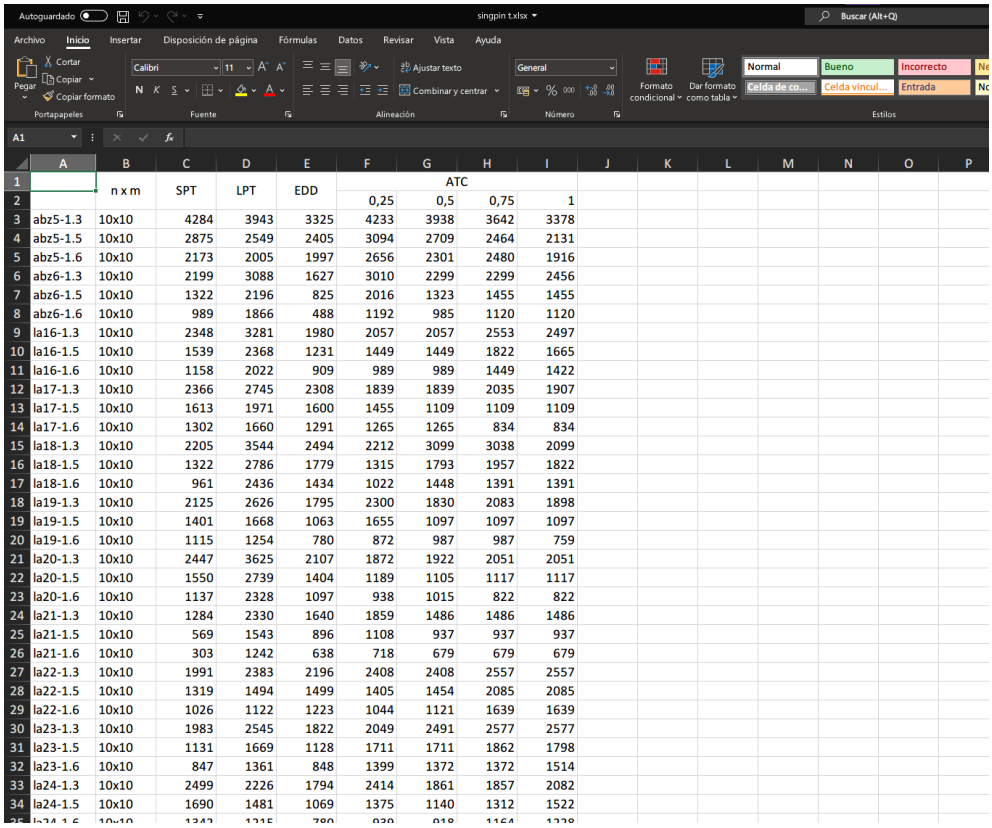



*Ilustración 9. Directorio con el archivo Excel resultante del ejemplo de ejecución del directorio singpin con instancias de Singer y Pinedo extendidas con minimización del tardiness.*



| | n x m | SPT | LPT | EDD | ATC | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 0,25 | 0,5 | 0,75 | 1 |
| abz5-1.3 | 10x10 | 4284 | 3943 | 3325 | 4233 | 3938 | 3642 | 3378 |
| abz5-1.5 | 10x10 | 2875 | 2549 | 2405 | 3094 | 2709 | 2464 | 2131 |
| abz5-1.6 | 10x10 | 2173 | 2005 | 1997 | 2656 | 2301 | 2480 | 1916 |
| abz6-1.3 | 10x10 | 2199 | 3088 | 1627 | 3010 | 2299 | 2299 | 2456 |
| abz6-1.5 | 10x10 | 1322 | 2196 | 825 | 2016 | 1323 | 1455 | 1455 |
| abz6-1.6 | 10x10 | 989 | 1866 | 488 | 1192 | 985 | 1120 | 1120 |
| la16-1.3 | 10x10 | 2348 | 3281 | 1980 | 2057 | 2057 | 2553 | 2497 |
| la16-1.5 | 10x10 | 1539 | 2368 | 1231 | 1449 | 1449 | 1822 | 1665 |
| la16-1.6 | 10x10 | 1158 | 2022 | 909 | 989 | 989 | 1449 | 1422 |
| la17-1.3 | 10x10 | 2366 | 2745 | 2308 | 1839 | 1839 | 2035 | 1907 |
| la17-1.5 | 10x10 | 1613 | 1971 | 1600 | 1455 | 1109 | 1109 | 1109 |
| la17-1.6 | 10x10 | 1302 | 1660 | 1291 | 1265 | 1265 | 834 | 834 |
| la18-1.3 | 10x10 | 2205 | 3544 | 2494 | 2212 | 3099 | 3038 | 2099 |
| la18-1.5 | 10x10 | 1322 | 2786 | 1779 | 1315 | 1793 | 1957 | 1822 |
| la18-1.6 | 10x10 | 961 | 2436 | 1434 | 1022 | 1448 | 1391 | 1391 |
| la19-1.3 | 10x10 | 2125 | 2626 | 1795 | 2300 | 1830 | 2083 | 1898 |
| la19-1.5 | 10x10 | 1401 | 1668 | 1063 | 1655 | 1097 | 1097 | 1097 |
| la19-1.6 | 10x10 | 1115 | 1254 | 780 | 872 | 987 | 987 | 759 |
| la20-1.3 | 10x10 | 2447 | 3625 | 2107 | 1872 | 1922 | 2051 | 2051 |
| la20-1.5 | 10x10 | 1550 | 2739 | 1404 | 1189 | 1105 | 1117 | 1117 |
| la20-1.6 | 10x10 | 1137 | 2328 | 1097 | 938 | 1015 | 822 | 822 |
| la21-1.3 | 10x10 | 1284 | 2330 | 1640 | 1859 | 1486 | 1486 | 1486 |
| la21-1.5 | 10x10 | 569 | 1543 | 896 | 1108 | 937 | 937 | 937 |
| la21-1.6 | 10x10 | 303 | 1242 | 638 | 718 | 679 | 679 | 679 |
| la22-1.3 | 10x10 | 1991 | 2383 | 2196 | 2408 | 2408 | 2557 | 2557 |
| la22-1.5 | 10x10 | 1319 | 1494 | 1499 | 1405 | 1454 | 2085 | 2085 |
| la22-1.6 | 10x10 | 1026 | 1122 | 1223 | 1044 | 1121 | 1639 | 1639 |
| la23-1.3 | 10x10 | 1983 | 2545 | 1822 | 2049 | 2491 | 2577 | 2577 |
| la23-1.5 | 10x10 | 1131 | 1669 | 1128 | 1711 | 1711 | 1862 | 1798 |
| la23-1.6 | 10x10 | 847 | 1361 | 848 | 1399 | 1372 | 1372 | 1514 |
| la24-1.3 | 10x10 | 2499 | 2226 | 1794 | 2414 | 1861 | 1857 | 2082 |
| la24-1.5 | 10x10 | 1690 | 1481 | 1069 | 1375 | 1140 | 1312 | 1522 |

*Ilustración 10. Contenido del archivo Excel resultante del ejemplo del directorio singpin con instancias de Singer y Pinedo extendidas con minimización del tardiness.*

## 7.4 Manual del programador

El sistema se encuentra organizado en secciones claramente definidas en paquetes, por lo que no debería de haber problema en cuanto a la implementación de nuevas funcionalidades o características, como la adición de nuevas reglas de prioridad, otro algoritmo de planificación o un *parser* de otro tipo de ficheros.

### *Graph*

En este trabajo y para este problema concretamente se hizo uso de un grafo de restricciones, implementado en la clase *ConstraintGraph*, que extiende de *GraphImpl* que a su vez implementa de la interfaz *Graph*. A excepción de *ConstraintGraph,* las demás clases son de tipo genérico (como se ve en la Ilustración 11 y la Ilustración 12) por lo que la implementación de otra clase de grafo aplicado a algún otro tipo en concreto que implemente de la interfaz sería bastante sencilla.

```
12      */
13      public interface Graph<T> {
14
```

*Ilustración 11. Declaración de la Interfaz Graph.*

```
13  public class GraphImpl<T> implements Graph<T> {
14
15      private Map<T, List<T>> nodes;
16
```

*Ilustración 12. Declaración de la clase GraphImpl.*

### *Instances*

En este paquete se encuentran las clases básicas necesarias para cualquier problema de planificación de tareas. Se incluye una clase abstracta llamada *Instance* que contiene los atributos básicos que incluye una instancia para estos problemas. No obstante, para especificar más parámetros, se pueden crear clases que extiendan de esta, como la creada para este trabajo: *TaillardInstance*:

```
13  */
14  public class TaillardInstance extends Instance {
15
```

*Ilustración 13. Declaración de la clase TaillardInstance.*

### *Schedule*

Aquí se incluye la interfaz mostrada en la Ilustración 14 *ScheduleAlgorithm*, que contiene métodos básicos que incluiría un algoritmo planificador: *run()*, para la ejecución y dos métodos diferentes de escritura: uno de una matriz de los *startTimes* de una instancia y otro de escritura en función de la función objetivo.

```
10  public interface ScheduleAlgorithm {
11
12      public List<ResultTask> run() throws AlgorithmException;
13
14      public void writeStartingTimeMatrix(String path, String output, String sheet);
15
16      public void writeAll(String path, String output, String inst, int rowNum, int colNum,
17                           boolean extended, String objFun);
18
19  }
```

*Ilustración 14. Interfaz ScheduleAlgorithm.*

En este trabajo se empleó la clase *GTAlgorithm* para la ejecución del algoritmo G&T, que implementa de *ScheduleAlgorithm*, por lo que con vista a la inclusión de un nuevo algoritmo solamente sería necesaria una clase que implementase de dicha interfaz.

### *Rules*

En este paquete se incluye la interfaz *Rule* (Ilustración 15), que incluye un método necesario para ejecutar la regla de prioridad, y de la cual heredan las distintas reglas de prioridad implementadas en este trabajo.

```
8  public interface Rule extends Serializable {
9
10     Operation run(List<Operation> operations);
11 }
```

*Ilustración 15. Interfaz Rule.*

### *Parser*

Como ya se mencionó brevemente antes, existe la posibilidad de leer otro tipo de ficheros diferentes a los empleados en este proyecto solamente extendiendo de la clase *FileParserImpl*, que trabaja con un tipo genérico que posteriormente, al crear un *parser* para una instancia concreta, puede concretarse, como por ejemplo el *parser* de instancias de Taillard de la  Ilustración 16, que emplea el tipo *TaillardInstance*.

```
15 */
16 public class TaillardFileImpl extends FileParserImpl<TaillardInstance> {
17
```

*Ilustración 16. Declaración de la clase TaillardFileImpl.*

### *Writer*

O lo que es lo mismo, el *output* del sistema. Consistente en una interfaz *Writer*, que contiene los dos métodos que incluye *ScheduleAlgorithm* para la escritura de ficheros (Ilustración 17). En este proyecto se incluyó la clase *ExcelWriterImpl*, que especifica el cuerpo de estos métodos para la escritura de datos en Excel, por lo que para realizar la escritura de datos para otro tipo de fichero solamente haría falta heredar de esta interfaz y completar los métodos de la forma pertinente.

```
6  public interface Writer {
7
8      public void writeMatrixStartTimes(String path, String name, String sheetName, List<String[]> data);
9
10     public void write(String path, String name, String instName, int rowNum, int colNum, Instance inst,
11                       long result, boolean extended, String objFunc);
12 }
```

*Ilustración 17. Interfaz Writer.*

# CAPÍTULO 8: Conclusiones y posibles ampliaciones

Se ha desarrollado una aplicación que ofrece la funcionalidad de construir planificaciones para instancias del *JSS* mediante el algoritmo de Giffler y Thompson guiado por una serie de reglas de prioridad, cumpliendo con las restricciones del problema, y con el objeto de calcular soluciones factibles y que posteriormente serán comparadas entre sí y con las mejores soluciones o cotas conocidas.

Por ello, resulta esencial realizar una definición de los requisitos que marcan tanto la funcionalidad como el comportamiento deseado para el sistema, entre los que se encuentran la carga de ficheros (o directorios de ficheros), o la generación de ficheros con los resultados calculados. El algoritmo voraz implementado se ha utilizado para resolver parias instancias del problema JSS, variando en tamaño desde instancias de 10x10 hasta de 100x20. A pesar de haber obtenido una implementación de código capaz de obtener resultados en un tiempo prudencial, aún es posible optimizar este aspecto con el fin de hacer un código más eficiente. Esto podría considerarse una posible ampliación del proyecto.

Se puede concluir, pues, con que la aplicación cumple con los objetivos especificados para el proyecto, y cumple con la funcionalidad de obtener soluciones para las versiones del *JSS* con las funciones objetivo de minimización del *makespan* y/o del *tardiness.* En lo que a un trabajo de investigación se refiere, se puede afirmar que se ha conseguido entender, a pesar de su complejidad, el contexto del problema aquí explicado, así como desarrollar una herramienta ligera a la par que completa que consigue resolver este problema. Aun así, esta herramienta desarrollada es susceptible de ser ampliada y mejorada, con el fin de incrementar su rendimiento y usabilidad; así como de incorporar una interfaz gráfica que facilite su utilización por usuarios no acostumbrados a emplear este tipo de herramientas en línea de comandos.

A pesar del estudio experimental realizado en el trabajo, como se indicó en el CAPÍTULO 6: Análisis experimental, no es posible emitir una conclusión global absoluta. Por ello, sería conveniente, como tarea futura a corto plazo, incluir un validador de soluciones con el fin de comprobar el correcto funcionamiento tanto del algoritmo como de las reglas de prioridad utilizadas, pudiendo así verificar todos los resultados obtenidos. Asimismo, incrementar el banco de instancias de prueba, permitiría extraer mejores conclusiones del análisis experimental, por lo que también se puede proponer como ampliación de este proyecto.

Además, podría contemplarse el cálculo de reglas mediante hiperheurísticos, que consisten en métodos de búsquedas heurísticas que buscan automatizar, generalmente a través de la incorporación de técnicas de aprendizaje automático, el proceso tanto de selección, combinación, generación o adaptación de varias heurísticas más simples con el fin de resolver de manera eficiente problemas computacionales de búsqueda. Esto permitiría seleccionar y aplicar la heurística más apropiadas en cada punto de decisión. Una de las motivaciones y objeto del estudio de estos métodos es la posibilidad de construir sistemas capaces de manejar clases de problemas en lugar de resolver un

problema en concreto, es decir, tienen como objetivo ser métodos genéricos. Son evidentes, pues, las amplias posibilidades de resolución de problemas que habría en el caso de ampliar este trabajo al cálculo de reglas mediante estos hiperheurísticos y, en especial, como se cita en [32] y [21], mediante programación genética, cuya eficacia ha sido demostrada en problemas de optimización "duros". La programación genética puede usarse como hiperheurístico con el fin de generar heurísticos más simples y metaheurísticos. Esto se conoce como GPHH *(Genetic Programming based Hyper-Heuristics)*.

Como otras posibles ampliaciones al proyecto se proponen ideas como: reglas para varios objetivos a la vez (optimización multiobjetivo), implementación de reglas probabilísticas, o reglas para guiar otros algoritmos.

Para finalizar, a modo de reflexión, puedo afirmar que este trabajo, además de suponerme un reto, ha servido para aumentar, en ciertos aspectos, mi crecimiento tanto a nivel académico como personal, pues ha sido necesario leer y entender diversos artículos científicos, siendo este un campo totalmente nuevo para mí, saber interpretarlos y poder trasladar los conocimientos y datos relevantes a este proyecto.

# CAPÍTULO 9: Anexos

## Anexo I: Estructura de desglose del trabajo (Work Breakdown Structure)

En esta sección aparece la versión completa del desglose del trabajo de este proyecto.

| EDT | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| **1** | **Trabajo Fin de Grado** | **533 horas** | **jue 18/02/21** | **jue 20/01/22** |
| **1.1** | **Inicio del proyecto** | **44 horas** | **jue 18/02/21** | **mar 16/03/21** |
| 1.1.1 | Primera reunión | 2 horas | jue 18/02/21 | jue 18/02/21 |
| **1.1.2** | **Análisis del problema** | **32 horas** | **jue 25/02/21** | **mar 16/03/21** |
| 1.1.2.1 | Definición de la organización | 10 horas | jue 25/02/21 | mié 03/03/21 |
| 1.1.2.2 | Análisis y definición de riesgos | 5 horas | mié 03/03/21 | jue 04/03/21 |
| 1.1.2.3 | Análisis de requisitos | 15 horas | jue 04/03/21 | jue 11/03/21 |
| 1.1.2.4 | Definición del presupuesto | 8 horas | jue 11/03/21 | mar 16/03/21 |
| **1.1.3** | **Estudio del problema** | **12 horas** | **jue 18/02/21** | **jue 25/02/21** |
| 1.1.3.1 | Búsqueda de artículos científicos | 16 horas | jue 18/02/21 | vie 19/02/21 |
| 1.1.3.2 | Lectura de artículos | 25 horas | sáb 20/02/21 | jue 25/02/21 |
| 1.1.4 | Estudio y decisión de las tecnologías empleadas | 5 horas | jue 18/02/21 | mar 23/02/21 |
| **1.2** | **Diseño de la solución del problema** | **36 horas** | **mar 16/03/21** | **lun 05/04/21** |
| 1.2.1 | Diseño algoritmo de planificación y espacio de búsqueda | 18,89 horas | mar 16/03/21 | lun 29/03/21 |
| 1.2.2 | Estudio y diseño de reglas de prioridad | 12 horas | mar 30/03/21 | lun 05/04/21 |
| **1.3** | **Implementación del sistema** | **205 horas** | **mar 23/02/21** | **mar 22/06/21** |
| 1.3.1 | Segunda reunión | 2 horas | lun 14/06/21 | mar 15/06/21 |
| 1.3.2 | Implementación básica del sistema | 15 horas | mar 23/02/21 | mié 03/03/21 |
| 1.3.3 | Implementación del algoritmo de planificación | 15 horas | mar 15/06/21 | mar 22/06/21 |
| **1.3.4** | **Implementación de las reglas de prioridad definidas** | **15 horas** | **mar 15/06/21** | **mar 22/06/21** |
| 1.3.4.1 | SPT | 3 horas | mar 15/06/21 | mar 15/06/21 |
| 1.3.4.2 | LPT | 3 horas | mar 15/06/21 | mar 15/06/21 |
| 1.3.4.3 | MCT | 3 horas | mar 15/06/21 | mar 15/06/21 |

| EDT | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1.3.4.4 | EDD | 3 horas | mar 15/06/21 | mar 15/06/21 |
| 1.3.4.5 | ATC | 12 horas | mié 16/06/21 | mar 22/06/21 |
| **1.4** | **Análisis experimental** | **135 horas** | **mar 05/10/21** | **jue 16/12/21** |
| 1.4.1 | Tercera reunión | 2 horas | mar 05/10/21 | mar 05/10/21 |
| 1.4.2 | Cambios y mejoras en las reglas de prioridad | 10 horas | mié 06/10/21 | mar 12/10/21 |
| 1.4.3 | Implementación de salida de datos del sistema | 20 horas | mié 13/10/21 | lun 25/10/21 |
| 1.4.4 | Preparación y ejecución de los experimentos mediante la carga de instancias | 10 horas | mar 26/10/21 | lun 01/11/21 |
| 1.4.5 | Cuarta reunión | 3 horas | jue 09/12/21 | jue 09/12/21 |
| 1.4.6 | Cambios en cuanto a la salida de datos del sistema | 2 horas | vie 10/12/21 | lun 13/12/21 |
| 1.4.7 | Crear hojas Excel con los resultados, crear gráficas y analizar resultados, comparándolos con otros estudios | 10 horas | lun 13/12/21 | jue 16/12/21 |
| **1.5** | **Pruebas del sistema** | **24 horas** | **jue 04/11/21** | **mié 17/11/21** |
| 1.5.1 | Pruebas unitarias | 10 horas | jue 04/11/21 | mié 10/11/21 |
| 1.5.2 | Búsqueda y comparativa de resultados con otros estudios | 12 horas | jue 11/11/21 | mié 17/11/21 |
| 1.6 | Documentación | 113 horas | mar 02/03/21 | jue 20/01/22 |

*Tabla 75. Estructura de desglose de trabajo (WBS).*

## Anexo II: Estructura de desglose de riesgo (Risk Breakdown Structure)

La estructura de desglose de riesgo se trata de una lista jerárquica de los recursos necesarios para la gestión y control del proyecto, donde figuran el personal y los elementos precisos para la realización del mismo [48].

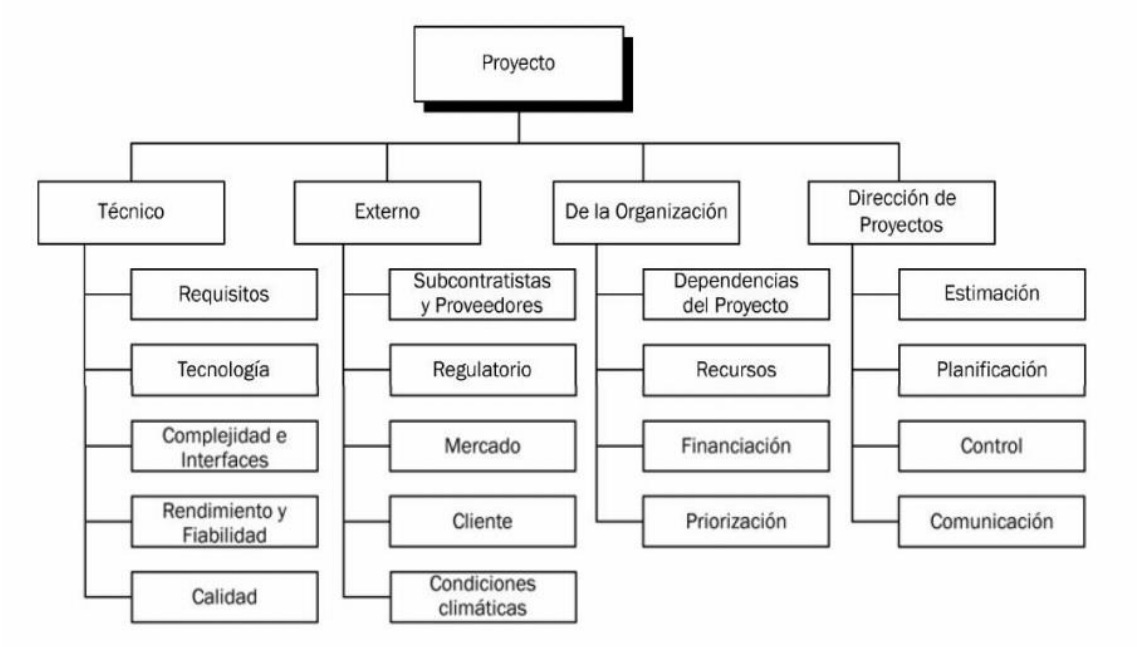



*Ilustración 18. Categorías representadas en el RBS según el PMBOK, 2013.*

## Anexo III: Matriz de Probabilidad e Impacto

La Matriz de Probabilidad e Impacto define los valores usados para priorizar los riesgos de acuerdo a la probabilidad de ocurrencia de un riesgo y a su impacto en el proyecto [48].

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Probabilidad** | **Muy Alta** | ***0,90*** | 0,05 | 0,09 | 0,18 | 0,36 | 0,72 |
| | **Alta** | ***0,70*** | 0,04 | 0,07 | 0,14 | 0,28 | 0,56 |
| | **Media** | ***0,50*** | 0,03 | 0,05 | 0,10 | 0,20 | 0,40 |
| | **Baja** | ***0,30*** | 0,02 | 0,03 | 0,06 | 0,12 | 0,24 |
| | **Muy Baja** | ***0,10*** | 0,01 | 0,01 | 0,02 | 0,04 | 0,08 |
| | | | ***0,05*** | ***0,10*** | ***0,20*** | ***0,40*** | ***0,80*** |
| | | | **Muy Bajo** | **Bajo** | **Medio** | **Alto** | **Muy Alto** |
| | | | **Impacto** | | | | |

*Ilustración 19. Matriz de Probabilidad e Impacto según el PMBOK, 2013.*

# Anexo IV: Presupuestos

## Presupuesto de costes

En este apartado se tendrán en cuenta dos tipos de costes: directos e indirectos. Los costes directos, ilustrados en la Tabla 76, representan todos aquellos aspectos a tener en cuenta en el propio proyecto, como productos o licencias, mientras que los indirectos, ilustrados en la Tabla 77, representan los costes estructurales, financieros, comerciales, etc.

| **Costes directos** | | | | | | |
|---|---|---|---|---|---|---|
| **Capítulo** | **Ítem** | **Descripción** | **Cantidad** | **Unidad** | **Coste/Unidad (€)** | **Coste total (€)** |
| **1** | | **Estudio del problema** | **120** | **Horas** | **10** | **1200** |
| | 1.1 | Búsqueda de artículos científicos relacionados | 20 | Horas | 10 | 200 |
| | 1.2 | Lectura de los artículos | 100 | Horas | 10 | 1000 |
| **2** | | **Diseño de la solución del problema** | **180** | **Horas** | **15** | **2700** |
| | 2.2 | Diseño algoritmo de planificación y espacio de búsqueda | 130 | Horas | 15 | 1950 |
| | 2.3 | Estudio y diseño reglas de prioridad | 50 | Horas | 15 | 750 |
| **3** | | **Estudio y decisión de las tecnologías** | **5** | **Horas** | **15** | **75** |
| **4** | | **Implementación del sistema** | **100** | **Horas** | **15** | **1500** |
| | 4.1 | Implementación del algoritmo de planificación | 70 | Horas | 15 | 1050 |
| | 4.2 | Implementación de las reglas de prioridad definidas | 30 | Horas | 15 | 450 |
| **5** | | **Elaboración de la documentación** | **120** | **Horas** | **15** | **1800** |
| **6** | | **Licencias** | **36** | | | **290,40** |
| | 6.1 | Licencias software | | | | |
| | 6.1.1 | GitHub | 12 | Mes | 7 | 84 |
| | 6.1.2 | Microsoft Project Pro | 12 | Mes | 8,40 | 100,80 |

| Costes directos | | | | | | |
|---|---|---|---|---|---|---|
| **Capítulo** | **Ítem** | **Descripción** | **Cantidad** | **Unidad** | **Coste/Unidad (€)** | **Coste total (€)** |
| | 6.1.3 | Microsoft 365 (aplicaciones) | 12 | Mes | 8,80 | 105,60 |
| **7** | | **Reuniones** | **35** | **Horas** | **15** | **525** |

| | |
|---|---|
| **Total mano de obra** | **7800** |
| **Total materiales / licencias** | **290,40** |
| **Total** | **8090,40** |

*Tabla 76. Presupuesto de costes directos.*

| Costes indirectos | | | | | | |
|---|---|---|---|---|---|---|
| **Capítulo** | **Ítem** | **Descripción** | **Cantidad** | **Unidad** | **Coste/Unidad (€)** | **Coste total (€)** |
| **1** | | **Servicios** | **2** | | | **500** |
| | 1.1 | Amortización de equipos de hardware | 1 | Uds | 300 | 300 |
| | 1.2 | Amortización de licencias de software | 1 | Uds | 200 | 200 |
| **2** | | **Materiales** | **10** | | | **20** |
| | 2.1 | Material de oficina | 10 | Uds | 2 | 20 |
| **3** | | **Comunicaciones** | **12** | **Mes** | **20** | **240** |
| **4** | | **Suministros (Electricidad, agua…)** | **12** | **Mes** | **40** | **480** |
| **Total** | | | | | | **1240** |

*Tabla 77. Presupuesto de costes indirectos.*

En este caso, el porcentaje que representan los costes indirectos frente a los costes directos es del 15%, como se ilustra en la Tabla 78.

| | |
|---|---|
| **Total costes directos (€)** | 8090,40 |
| **Total costes indirectos (€)** | 1240 |
| **Total presupuesto de coste (€)** | 9330,40 |
| **Porcentaje de repercusión de costes indirectos** | 15% |

*Tabla 78. Porcentaje de repercusión de los costes indirectos frente a los costes directos.*

### Presupuesto de cliente

En el presupuesto de cliente no se incluirán los beneficios ni los costes indirectos, por lo que la suma de ambas partidas, que asciende a 3572,60€ como se muestra en la Tabla 79 habrá de ser compensada. Esta cifra, pues, se compensará incrementando cada uno de los elementos reflejados en el presupuesto de costes directos en un 44,16%, computado en el presupuesto de cliente, como se puede apreciar en la Tabla 80.

| | |
|---|---|
| **Coste total del proyecto** | **9.330,40 €** |
| **Margen de beneficio** | **25%** |
| **Beneficio** | **2.332,60 €** |
| **A compensar** | **3.572,60 €** |
| **Total sobre el que se aplica la compensación** | **8.090,40 €** |
| **Porcentaje a incrementar** | **44,16%** |
| **Total con beneficio** | **11.663,00 €** |

*Tabla 79. Cálculo del presupuesto del cliente.*

| Presupuesto de cliente | | | | |
|---|---|---|---|---|
| **Capítulo** | **Ítem** | **Descripción** | **Importe (€)** | **Coste total (€)** |
| **1** | | **Estudio del problema** | | **1729,90** |
| | 1.1 | Búsqueda de artículos científicos relacionados | 288,32 | |
| | 1.2 | Lectura de los artículos | 1441,59 | |
| **2** | | **Diseño de la solución del problema** | | **3892,28** |
| | 2.2 | Diseño algoritmo de planificación y espacio de búsqueda | 2811,09 | |
| | 2.3 | Estudio y diseño reglas de prioridad | 1081,19 | |
| **3** | | **Estudio y decisión de las tecnologías** | | **108,12** |
| **4** | | **Implementación del sistema** | | **2162,38** |
| | 4.1 | Implementación del algoritmo de planificación | 1513,66 | |
| | 4.2 | Implementación de las reglas de prioridad definidas | 648,71 | |
| **5** | | **Elaboración de la documentación** | | **2594,85** |
| **6** | | **Licencias** | | **418,64** |
| | 6.1 | Licencias software | | |
| | 6.1.1 | GitHub | 121,09 | |
| | 6.1.2 | Microsoft Project Pro | 145,31 | |
| | 6.1.3 | Microsoft 365 (aplicaciones) | 152,23 | |
| **7** | | **Reuniones** | | **756,83** |
| **Total costes directos** | | | | **11.663,00 €** |

*Tabla 80. Presupuesto de cliente*

## Anexo V: Resultados del análisis experimental

### Función objetivo: *makespan*

Aquí se encuentran todos los resultados obtenidos para las 80 instancias básicas de Taillard:

| | **n x m** | **SPT** | **LPT** | Lower Bound | Upper Bound |
|---|---|---|---|---|---|
| tai15x15-1 | 15x15 | 2003 | 1913 | 1005 | 1231 |
| tai15x15-2 | 15x15 | 1891 | 1800 | 953 | 1244 |
| tai15x15-3 | 15x15 | 1994 | 1813 | 1036 | 1222 |
| tai15x15-4 | 15x15 | 2144 | 1918 | 973 | 1181 |
| tai15x15-5 | 15x15 | 2307 | 2027 | 940 | 1233 |
| tai15x15-6 | 15x15 | 1761 | 1853 | 1134 | 1243 |
| tai15x15-7 | 15x15 | 2026 | 2026 | 1103 | 1228 |
| tai15x15-8 | 15x15 | 1660 | 2030 | 980 | 1220 |
| tai15x15-9 | 15x15 | 1889 | 1849 | 1020 | 1282 |
| tai15x15-10 | 15x15 | 1840 | 1854 | 940 | 1259 |
| tai20x15-1 | 20x15 | 2106 | 1989 | 1254 | 1376 |
| tai20x15-2 | 20x15 | 2559 | 1993 | 1267 | 1377 |
| tai20x15-3 | 20x15 | 2409 | 2128 | 1243 | 1367 |
| tai20x15-4 | 20x15 | 2208 | 1844 | 1329 | 1345 |
| tai20x15-5 | 20x15 | 2338 | 2099 | 1163 | 1366 |
| tai20x15-6 | 20x15 | 2418 | 2276 | 1211 | 1371 |
| tai20x15-7 | 20x15 | 2480 | 2230 | 1306 | 1480 |
| tai20x15-8 | 20x15 | 2340 | 1949 | 1315 | 1413 |
| tai20x15-9 | 20x15 | 2420 | 2154 | 1202 | 1352 |
| tai20x15-10 | 20x15 | 2262 | 2293 | 1213 | 1362 |
| tai20x20-1 | 20x20 | 2514 | 2510 | 1217 | 1663 |
| tai20x20-2 | 20x20 | 2809 | 2618 | 1314 | 1626 |

| | **n x m** | **SPT** | **LPT** | Lower Bound | Upper Bound |
|---|---|---|---|---|---|
| tai20x20-3 | 20x20 | 2705 | 2409 | 1248 | 1574 |
| tai20x20-4 | 20x20 | 2753 | 2708 | 1284 | 1660 |
| tai20x20-5 | 20x20 | 2662 | 2442 | 1256 | 1598 |
| tai20x20-6 | 20x20 | 2595 | 2476 | 1245 | 1657 |
| tai20x20-7 | 20x20 | 2895 | 2433 | 1403 | 1704 |
| tai20x20-8 | 20x20 | 2458 | 2648 | 1387 | 1626 |
| tai20x20-9 | 20x20 | 2508 | 2616 | 1352 | 1629 |
| tai20x20-10 | 20x20 | 2866 | 2249 | 1277 | 1614 |
| tai30x15-1 | 30x15 | 3006 | 2660 | 1764 | 1770 |
| tai30x15-2 | 30x15 | 3160 | 2781 | 1774 | 1841 |
| tai30x15-3 | 30x15 | 2882 | 3030 | 1733 | 1832 |
| tai30x15-4 | 30x15 | 3056 | 2893 | 1828 | 1851 |
| tai30x15-5 | 30x15 | 3188 | 2912 | 1754 | 2007 |
| tai30x15-6 | 30x15 | 3317 | 2678 | 1777 | 1844 |
| tai30x15-7 | 30x15 | 3171 | 2591 | 1771 | 1815 |
| tai30x15-8 | 30x15 | 3064 | 3112 | 1673 | 1700 |
| tai30x15-9 | 30x15 | 3156 | 2888 | 1764 | 1811 |
| tai30x15-10 | 30x15 | 2943 | 2807 | 1608 | 1720 |
| tai30x20-1 | 30x20 | 3441 | 3293 | 1850 | 2064 |
| tai30x20-2 | 30x20 | 3506 | 3336 | 1761 | 1983 |
| tai30x20-3 | 30x20 | 3071 | 3122 | 1710 | 1896 |
| tai30x20-4 | 30x20 | 3437 | 3026 | 1820 | 2031 |
| tai30x20-5 | 30x20 | 3510 | 3066 | 1785 | 2032 |
| tai30x20-6 | 30x20 | 3410 | 3064 | 1940 | 2057 |
| tai30x20-7 | 30x20 | 3657 | 3150 | 1751 | 1947 |
| tai30x20-8 | 30x20 | 3539 | 3013 | 1770 | 2001 |
| tai30x20-9 | 30x20 | 3412 | 3036 | 1758 | 2013 |

| | **n x m** | **SPT** | **LPT** | Lower Bound | Upper Bound |
|---|---|---|---|---|---|
| tai30x20-10 | 30x20 | 3565 | 3226 | 1678 | 1973 |
| tai50x15-1 | 50x15 | 4535 | 4083 | 2760 | 2760 |
| tai50x15-2 | 50x15 | 4732 | 4511 | 2756 | 2756 |
| tai50x15-3 | 50x15 | 4181 | 3916 | 2717 | 2717 |
| tai50x15-4 | 50x15 | 4417 | 3996 | 2813 | 2839 |
| tai50x15-5 | 50x15 | 4636 | 3984 | 2679 | 2679 |
| tai50x15-6 | 50x15 | 4675 | 4262 | 2781 | 2781 |
| tai50x15-7 | 50x15 | 4802 | 4097 | 2943 | 2943 |
| tai50x15-8 | 50x15 | 4535 | 4550 | 2885 | 2885 |
| tai50x15-9 | 50x15 | 4430 | 4135 | 2655 | 2655 |
| tai50x15-10 | 50x15 | 5035 | 4038 | 2723 | 2723 |
| tai50x20-1 | 50x20 | 4844 | 4158 | 2868 | 2868 |
| tai50x20-2 | 50x20 | 5019 | 4268 | 2848 | 2902 |
| tai50x20-3 | 50x20 | 4717 | 4028 | 2755 | 2755 |
| tai50x20-4 | 50x20 | 4634 | 4282 | 2697 | 2702 |
| tai50x20-5 | 50x20 | 4585 | 4352 | 2725 | 2725 |
| tai50x20-6 | 50x20 | 4823 | 4059 | 2845 | 2845 |
| tai50x20-7 | 50x20 | 4591 | 4072 | 2812 | 2841 |
| tai50x20-8 | 50x20 | 4968 | 4168 | 2764 | 2784 |
| tai50x20-9 | 50x20 | 5029 | 4477 | 3071 | 3071 |
| tai50x20-10 | 50x20 | 4979 | 4440 | 2995 | 2995 |
| tai100x20-1 | 100x20 | 8352 | 7632 | 5464 | 5464 |
| tai100x20-2 | 100x20 | 8289 | 7210 | 5181 | 5181 |
| tai100x20-3 | 100x20 | 8538 | 7352 | 5552 | 5568 |
| tai100x20-4 | 100x20 | 8330 | 6996 | 5339 | 5339 |
| tai100x20-5 | 100x20 | 8460 | 7353 | 5392 | 5392 |
| tai100x20-6 | 100x20 | 8445 | 7011 | 5342 | 5342 |

| | **n x m** | **SPT** | **LPT** | Lower Bound | Upper Bound |
|---|---|---|---|---|---|
| tai100x20-7 | 100x20 | 8709 | 7436 | 5436 | 5436 |
| tai100x20-8 | 100x20 | 8419 | 7216 | 5394 | 5394 |
| tai100x20-9 | 100x20 | 8342 | 7396 | 5358 | 5358 |
| tai100x20-10 | 100x20 | 7904 | 7338 | 5183 | 5183 |

*Tabla 81. Resultados para el makespan de las 80 instancias básicas de Taillard, junto con sus cotas inferiores y superiores.*

A continuación, los resultados obtenidos para las 22 instancias de Singer y Pinedo ejecutadas:

| | **n x m** | **SPT** | **LPT** | Upper Bound |
|---|---|---|---|---|
| abz5-1.3 | 10x10 | 1733 | 1836 | 1258 |
| abz6-1.3 | 10x10 | 1400 | 1381 | 952 |
| la16-1.3 | 10x10 | 1464 | 1336 | 1012 |
| la17-1.3 | 10x10 | 1128 | 1122 | 787 |
| la18-1.3 | 10x10 | 1250 | 1542 | 954 |
| la19-1.3 | 10x10 | 1181 | 1120 | 861 |
| la20-1.3 | 10x10 | 1436 | 1250 | 920 |
| la21-1.3 | 10x10 | 1149 | 1179 | 1092 |
| la22-1.3 | 10x10 | 1170 | 1031 | 955 |
| la23-1.3 | 10x10 | 1250 | 1302 | 1049 |
| la24-1.3 | 10x10 | 1304 | 1181 | 971 |
| mt10-1.3 | 10x10 | 1429 | 1355 | 970 |
| orb1-1.3 | 10x10 | 1481 | 1490 | 1145 |
| orb2-1.3 | 10x10 | 1334 | 1175 | 918 |
| orb3-1.3 | 10x10 | 1473 | 1538 | 1098 |
| orb4-1.3 | 10x10 | 1618 | 1626 | 1066 |

| | **n x m** | **SPT** | **LPT** | Upper Bound |
|---|---|---|---|---|
| orb5-1.3 | 10x10 | 1228 | 1180 | 911 |
| orb6-1.3 | 10x10 | 1527 | 1536 | 1050 |
| orb7-1.3 | 10x10 | 579 | 552 | 414 |
| orb8-1.3 | 10x10 | 1183 | 1339 | 945 |
| orb9-1.3 | 10x10 | 1407 | 1389 | 978 |
| orb10-1.3 | 10x10 | 1553 | 1473 | 991 |

*Tabla 82. Resultados para el makespan de las instancias de Singer y Pinedo, junto con sus cotas sup*

### Función objetivo: *tardiness*

Ahora, los resultados para las 240 instancias extendidas de Taillard para las reglas de prioridad SPT, LPT y EDD.

| | **n x m** | **SPT** | **LPT** | **EDD** |
|---|---|---|---|---|
| *tai01-1.3* | 15x15 | 6573 | 5840 | 4066 |
| *tai01-1.5* | 15x15 | 4353 | 3768 | 2449 |
| *tai01-1.6* | 15x15 | 3266 | 3089 | 1841 |
| *tai02-1.3* | 15x15 | 6721 | 7076 | 4819 |
| *tai02-1.5* | 15x15 | 4550 | 4999 | 3235 |
| *tai02-1.6* | 15x15 | 3562 | 4115 | 2539 |
| *tai03-1.3* | 15x15 | 7078 | 5738 | 5095 |
| *tai03-1.5* | 15x15 | 5100 | 3849 | 3365 |
| *tai03-1.6* | 15x15 | 4161 | 3129 | 2639 |
| *tai04-1.3* | 15x15 | 6947 | 8611 | 5006 |
| *tai04-1.5* | 15x15 | 4884 | 6570 | 3493 |
| *tai04-1.6* | 15x15 | 3874 | 5569 | 2794 |
| *tai05-1.3* | 15x15 | 7092 | 8243 | 6071 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *tai05-1.5* | 15x15 | 5122 | 6124 | 4155 |
| *tai05-1.6* | 15x15 | 4345 | 5139 | 3330 |
| *tai06-1.3* | 15x15 | 6272 | 6100 | 5811 |
| *tai06-1.5* | 15x15 | 4227 | 3967 | 3888 |
| *tai06-1.6* | 15x15 | 3253 | 3143 | 3067 |
| *tai07-1.3* | 15x15 | 6154 | 7181 | 5325 |
| *tai07-1.5* | 15x15 | 3976 | 5250 | 3603 |
| *tai07-1.6* | 15x15 | 2981 | 4349 | 2812 |
| *tai08-1.3* | 15x15 | 4415 | 7351 | 4490 |
| *tai08-1.5* | 15x15 | 2397 | 5291 | 2833 |
| *tai08-1.6* | 15x15 | 1542 | 4282 | 2066 |
| *tai09-1.3* | 15x15 | 6227 | 7853 | 5987 |
| *tai09-1.5* | 15x15 | 4274 | 5681 | 4285 |
| *tai09-1.6* | 15x15 | 3329 | 4753 | 3611 |
| *tai10-1.3* | 15x15 | 7329 | 7835 | 6636 |
| *tai10-1.5* | 15x15 | 5103 | 6049 | 4868 |
| *tai10-1.6* | 15x15 | 4033 | 5242 | 4137 |
| *tai11-1.3* | 20x15 | 12102 | 11901 | 9210 |
| *tai11-1.5* | 20x15 | 9336 | 9636 | 6864 |
| *tai11-1.6* | 20x15 | 8025 | 8611 | 5771 |
| *tai12-1.3* | 20x15 | 12661 | 12576 | 9849 |
| *tai12-1.5* | 20x15 | 9633 | 9528 | 7073 |
| *tai12-1.6* | 20x15 | 8251 | 8144 | 5844 |
| *tai13-1.3* | 20x15 | 11348 | 14947 | 8827 |
| *tai13-1.5* | 20x15 | 8787 | 12034 | 6746 |
| *tai13-1.6* | 20x15 | 7628 | 10658 | 5712 |
| *tai14-1.3* | 20x15 | 10932 | 11929 | 8153 |
| *tai14-1.5* | 20x15 | 8262 | 9381 | 5913 |
| *tai14-1.6* | 20x15 | 7043 | 8193 | 4862 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *tai15-1.3* | 20x15 | 12928 | 13335 | 8837 |
| *tai15-1.5* | 20x15 | 10012 | 10484 | 6512 |
| *tai15-1.6* | 20x15 | 8632 | 9185 | 5479 |
| *tai16-1.3* | 20x15 | 11693 | 13535 | 10553 |
| *tai16-1.5* | 20x15 | 8686 | 10577 | 8002 |
| *tai16-1.6* | 20x15 | 7357 | 9122 | 6790 |
| *tai17-1.3* | 20x15 | 13827 | 14716 | 10866 |
| *tai17-1.5* | 20x15 | 10884 | 11767 | 8338 |
| *tai17-1.6* | 20x15 | 9562 | 10448 | 7230 |
| *tai18-1.3* | 20x15 | 12008 | 14106 | 10106 |
| *tai18-1.5* | 20x15 | 9515 | 11117 | 7667 |
| *tai18-1.6* | 20x15 | 8314 | 9629 | 6599 |
| *tai19-1.3* | 20x15 | 11945 | 14243 | 9784 |
| *tai19-1.5* | 20x15 | 9145 | 11286 | 7253 |
| *tai19-1.6* | 20x15 | 7870 | 9829 | 6234 |
| *tai20-1.3* | 20x15 | 12145 | 16269 | 10364 |
| *tai20-1.5* | 20x15 | 9383 | 13237 | 7927 |
| *tai20-1.6* | 20x15 | 8045 | 11874 | 6876 |
| *tai21-1.3* | 20x20 | 10875 | 12369 | 10861 |
| *tai21-1.5* | 20x20 | 7103 | 8545 | 7729 |
| *tai21-1.6* | 20x20 | 5686 | 6754 | 6323 |
| *tai22-1.3* | 20x20 | 11571 | 11964 | 10086 |
| *tai22-1.5* | 20x20 | 7863 | 8458 | 6939 |
| *tai22-1.6* | 20x20 | 6303 | 6939 | 5372 |
| *tai23-1.3* | 20x20 | 12504 | 13582 | 9930 |
| *tai23-1.5* | 20x20 | 8710 | 9571 | 6823 |
| *tai23-1.6* | 20x20 | 7037 | 7846 | 5448 |
| *tai24-1.3* | 20x20 | 10467 | 12401 | 10499 |
| *tai24-1.5* | 20x20 | 7001 | 8959 | 7309 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *tai24-1.6* | 20x20 | 5540 | 7318 | 5779 |
| *tai25-1.3* | 20x20 | 14381 | 11898 | 9289 |
| *tai25-1.5* | 20x20 | 10725 | 8146 | 6190 |
| *tai25-1.6* | 20x20 | 8977 | 6633 | 4852 |
| *tai26-1.3* | 20x20 | 11913 | 12353 | 10895 |
| *tai26-1.5* | 20x20 | 8151 | 8467 | 7677 |
| *tai26-1.6* | 20x20 | 6527 | 6692 | 6155 |
| *tai27-1.3* | 20x20 | 13648 | 12218 | 8336 |
| *tai27-1.5* | 20x20 | 9506 | 8560 | 5381 |
| *tai27-1.6* | 20x20 | 7550 | 6884 | 4097 |
| *tai28-1.3* | 20x20 | 12848 | 13722 | 8456 |
| *tai28-1.5* | 20x20 | 9139 | 9771 | 5630 |
| *tai28-1.6* | 20x20 | 7563 | 7863 | 4417 |
| *tai29-1.3* | 20x20 | 10412 | 15657 | 10971 |
| *tai29-1.5* | 20x20 | 6689 | 11527 | 7974 |
| *tai29-1.6* | 20x20 | 5129 | 9538 | 6681 |
| *tai30-1.3* | 20x20 | 12594 | 11500 | 8713 |
| *tai30-1.5* | 20x20 | 9049 | 7894 | 5830 |
| *tai30-1.6* | 20x20 | 7579 | 6171 | 4556 |
| *tai31-1.3* | 30x15 | 27598 | 34317 | 22947 |
| *tai31-1.5* | 30x15 | 23412 | 29792 | 18853 |
| *tai31-1.6* | 30x15 | 21360 | 27535 | 17013 |
| *tai32-1.3* | 30x15 | 33637 | 33129 | 22952 |
| *tai32-1.5* | 30x15 | 28988 | 28578 | 18714 |
| *tai32-1.6* | 30x15 | 26675 | 26350 | 16747 |
| *tai33-1.3* | 30x15 | 36002 | 33855 | 26061 |
| *tai33-1.5* | 30x15 | 31258 | 29111 | 21980 |
| *tai33-1.6* | 30x15 | 28921 | 26746 | 19996 |
| *tai34-1.3* | 30x15 | 37056 | 36050 | 24463 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *tai34-1.5* | 30x15 | 32368 | 31354 | 20328 |
| *tai34-1.6* | 30x15 | 30113 | 29016 | 18375 |
| *tai35-1.3* | 30x15 | 29611 | 35385 | 24748 |
| *tai35-1.5* | 30x15 | 25264 | 31131 | 20772 |
| *tai35-1.6* | 30x15 | 23176 | 29121 | 18937 |
| *tai36-1.3* | 30x15 | 33583 | 34352 | 23737 |
| *tai36-1.5* | 30x15 | 28985 | 29754 | 19618 |
| *tai36-1.6* | 30x15 | 26750 | 27468 | 17567 |
| *tai37-1.3* | 30x15 | 33538 | 30011 | 25304 |
| *tai37-1.5* | 30x15 | 28810 | 25400 | 20834 |
| *tai37-1.6* | 30x15 | 26511 | 23307 | 18831 |
| *tai38-1.3* | 30x15 | 31743 | 35729 | 23622 |
| *tai38-1.5* | 30x15 | 27356 | 31378 | 19625 |
| *tai38-1.6* | 30x15 | 25201 | 29246 | 17705 |
| *tai39-1.3* | 30x15 | 33296 | 34768 | 26387 |
| *tai39-1.5* | 30x15 | 28971 | 30593 | 22860 |
| *tai39-1.6* | 30x15 | 26877 | 28515 | 21215 |
| *tai40-1.3* | 30x15 | 30998 | 33436 | 21484 |
| *tai40-1.5* | 30x15 | 26674 | 29115 | 17642 |
| *tai40-1.6* | 30x15 | 24521 | 26986 | 15782 |
| *tai41-1.3* | 30x20 | 30592 | 32461 | 23977 |
| *tai41-1.5* | 30x20 | 24391 | 26415 | 18570 |
| *tai41-1.6* | 30x20 | 21517 | 23433 | 16134 |
| *tai42-1.3* | 30x20 | 32349 | 33115 | 24257 |
| *tai42-1.5* | 30x20 | 26222 | 26988 | 19033 |
| *tai42-1.6* | 30x20 | 23169 | 24032 | 16549 |
| *tai43-1.3* | 30x20 | 32200 | 31699 | 23754 |
| *tai43-1.5* | 30x20 | 26472 | 25922 | 18679 |
| *tai43-1.6* | 30x20 | 23633 | 23136 | 16385 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *tai44-1.3* | 30x20 | 33899 | 32375 | 25024 |
| *tai44-1.5* | 30x20 | 27917 | 26447 | 19771 |
| *tai44-1.6* | 30x20 | 24974 | 23731 | 17337 |
| *tai45-1.3* | 30x20 | 35060 | 34435 | 26051 |
| *tai45-1.5* | 30x20 | 29096 | 28525 | 20965 |
| *tai45-1.6* | 30x20 | 26121 | 25586 | 18687 |
| *tai46-1.3* | 30x20 | 36694 | 34382 | 26420 |
| *tai46-1.5* | 30x20 | 30520 | 28208 | 20960 |
| *tai46-1.6* | 30x20 | 27479 | 25132 | 18403 |
| *tai47-1.3* | 30x20 | 30727 | 36203 | 25746 |
| *tai47-1.5* | 30x20 | 24866 | 30160 | 20632 |
| *tai47-1.6* | 30x20 | 21999 | 27241 | 18245 |
| *tai48-1.3* | 30x20 | 31666 | 30270 | 26896 |
| *tai48-1.5* | 30x20 | 25959 | 24270 | 21711 |
| *tai48-1.6* | 30x20 | 23273 | 21408 | 19234 |
| *tai49-1.3* | 30x20 | 33476 | 33189 | 25358 |
| *tai49-1.5* | 30x20 | 27588 | 27233 | 20393 |
| *tai49-1.6* | 30x20 | 24704 | 24264 | 18102 |
| *tai50-1.3* | 30x20 | 30386 | 30785 | 23323 |
| *tai50-1.5* | 30x20 | 24323 | 24712 | 18402 |
| *tai50-1.6* | 30x20 | 21477 | 21887 | 16232 |
| *tai51-1.3* | 50x15 | 87956 | 99733 | 66782 |
| *tai51-1.5* | 50x15 | 80467 | 92142 | 59526 |
| *tai51-1.6* | 50x15 | 76754 | 88354 | 56009 |
| *tai52-1.3* | 50x15 | 99040 | 109393 | 70649 |
| *tai52-1.5* | 50x15 | 91690 | 101954 | 63602 |
| *tai52-1.6* | 50x15 | 88025 | 98244 | 60203 |
| *tai53-1.3* | 50x15 | 92815 | 92712 | 65656 |
| *tai53-1.5* | 50x15 | 85394 | 85225 | 58679 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *tai53-1.6* | 50x15 | 81731 | 81492 | 55233 |
| *tai54-1.3* | 50x15 | 98688 | 97800 | 68103 |
| *tai54-1.5* | 50x15 | 91326 | 90495 | 61105 |
| *tai54-1.6* | 50x15 | 87698 | 86899 | 57777 |
| *tai55-1.3* | 50x15 | 91351 | 95850 | 70522 |
| *tai55-1.5* | 50x15 | 83968 | 88467 | 63489 |
| *tai55-1.6* | 50x15 | 80287 | 84786 | 60059 |
| *tai56-1.3* | 50x15 | 90479 | 102751 | 68008 |
| *tai56-1.5* | 50x15 | 83030 | 95155 | 60826 |
| *tai56-1.6* | 50x15 | 79315 | 91367 | 57322 |
| *tai57-1.3* | 50x15 | 97351 | 100239 | 68485 |
| *tai57-1.5* | 50x15 | 89667 | 92545 | 61074 |
| *tai57-1.6* | 50x15 | 85887 | 88711 | 57582 |
| *tai58-1.3* | 50x15 | 99851 | 109823 | 75830 |
| *tai58-1.5* | 50x15 | 92204 | 102143 | 68461 |
| *tai58-1.6* | 50x15 | 88436 | 98328 | 64873 |
| *tai59-1.3* | 50x15 | 89594 | 90349 | 67250 |
| *tai59-1.5* | 50x15 | 82210 | 82965 | 60272 |
| *tai59-1.6* | 50x15 | 78532 | 79287 | 56870 |
| *tai60-1.3* | 50x15 | 96621 | 98330 | 67050 |
| *tai60-1.5* | 50x15 | 89073 | 90766 | 59811 |
| *tai60-1.6* | 50x15 | 85367 | 86992 | 56358 |
| *tai61-1.3* | 50x20 | 97233 | 98067 | 71607 |
| *tai61-1.5* | 50x20 | 87026 | 87860 | 62161 |
| *tai61-1.6* | 50x20 | 81937 | 82771 | 57737 |
| *tai62-1.3* | 50x20 | 97222 | 96575 | 71799 |
| *tai62-1.5* | 50x20 | 87055 | 86248 | 62010 |
| *tai62-1.6* | 50x20 | 82024 | 81099 | 57316 |
| *tai63-1.3* | 50x20 | 92316 | 94743 | 66698 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *tai63-1.5* | 50x20 | 82482 | 84909 | 57700 |
| *tai63-1.6* | 50x20 | 77629 | 80003 | 53414 |
| *tai64-1.3* | 50x20 | 87025 | 94398 | 65321 |
| *tai64-1.5* | 50x20 | 77394 | 84696 | 55972 |
| *tai64-1.6* | 50x20 | 72733 | 79856 | 51613 |
| *tai65-1.3* | 50x20 | 91966 | 92504 | 69109 |
| *tai65-1.5* | 50x20 | 82120 | 82658 | 59804 |
| *tai65-1.6* | 50x20 | 77207 | 77816 | 55451 |
| *tai66-1.3* | 50x20 | 94040 | 95261 | 66597 |
| *tai66-1.5* | 50x20 | 83956 | 85176 | 57067 |
| *tai66-1.6* | 50x20 | 78994 | 80145 | 52509 |
| *tai67-1.3* | 50x20 | 92063 | 94612 | 68334 |
| *tai67-1.5* | 50x20 | 82144 | 84693 | 59109 |
| *tai67-1.6* | 50x20 | 77238 | 79743 | 54724 |
| *tai68-1.3* | 50x20 | 94772 | 89779 | 68370 |
| *tai68-1.5* | 50x20 | 84779 | 79787 | 59394 |
| *tai68-1.6* | 50x20 | 79834 | 74928 | 55035 |
| *tai69-1.3* | 50x20 | 98120 | 100844 | 70383 |
| *tai69-1.5* | 50x20 | 88080 | 90657 | 61182 |
| *tai69-1.6* | 50x20 | 83073 | 85609 | 56688 |
| *tai70-1.3* | 50x20 | 98534 | 101637 | 70216 |
| *tai70-1.5* | 50x20 | 88351 | 91454 | 60676 |
| *tai70-1.6* | 50x20 | 83279 | 86369 | 56173 |
| *tai71-1.3* | 100x20 | 410150 | 427660 | 274165 |
| *tai71-1.5* | 100x20 | 389947 | 407457 | 254409 |
| *tai71-1.6* | 100x20 | 379953 | 397386 | 244721 |
| *tai72-1.3* | 100x20 | 405017 | 399339 | 271011 |
| *tai72-1.5* | 100x20 | 385521 | 379843 | 251802 |
| *tai72-1.6* | 100x20 | 375799 | 370121 | 242479 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *tai73-1.3* | 100x20 | 409851 | 413726 | 276917 |
| *tai73-1.5* | 100x20 | 389708 | 393583 | 257357 |
| *tai73-1.6* | 100x20 | 379664 | 383537 | 247777 |
| *tai74-1.3* | 100x20 | 421169 | 411952 | 279494 |
| *tai74-1.5* | 100x20 | 401137 | 391920 | 260026 |
| *tai74-1.6* | 100x20 | 391143 | 381926 | 250406 |
| *tai75-1.3* | 100x20 | 407126 | 415698 | 268206 |
| *tai75-1.5* | 100x20 | 387360 | 395932 | 248817 |
| *tai75-1.6* | 100x20 | 377509 | 386078 | 239347 |
| *tai76-1.3* | 100x20 | 420477 | 399474 | 285785 |
| *tai76-1.5* | 100x20 | 400380 | 379377 | 265940 |
| *tai76-1.6* | 100x20 | 390357 | 369354 | 256401 |
| *tai77-1.3* | 100x20 | 421617 | 423501 | 291584 |
| *tai77-1.5* | 100x20 | 401247 | 403131 | 271380 |
| *tai77-1.6* | 100x20 | 391097 | 392981 | 261608 |
| *tai78-1.3* | 100x20 | 405154 | 404101 | 266758 |
| *tai78-1.5* | 100x20 | 385318 | 384265 | 247340 |
| *tai78-1.6* | 100x20 | 375421 | 374368 | 237904 |
| *tai79-1.3* | 100x20 | 422856 | 398379 | 272795 |
| *tai79-1.5* | 100x20 | 402891 | 378414 | 253238 |
| *tai79-1.6* | 100x20 | 392927 | 368450 | 243719 |
| *tai80-1.3* | 100x20 | 405634 | 388548 | 270741 |
| *tai80-1.5* | 100x20 | 386278 | 369192 | 251872 |
| *tai80-1.6* | 100x20 | 376620 | 359534 | 242648 |

*Tabla 83. Resultados obtenidos para las instancias extendidas de Taillard empleando las reglas de prioridad SPT, LPT y EDD.*

Ahora, los resultados para las 240 instancias extendidas de Taillard para la regla de prioridad ATC, para los valores de *g = {0.25, 0.5, 0.75, 1}.*

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | 0,25 | 0,5 | 0,75 | 1 |
| *tai01-1.3* | 15x15 | 5704 | 4530 | 4530 | 5525 |
| *tai01-1.5* | 15x15 | 3229 | 2762 | 2762 | 4674 |
| *tai01-1.6* | 15x15 | 2498 | 2067 | 2067 | 2968 |
| *tai02-1.3* | 15x15 | 5155 | 5501 | 5111 | 5473 |
| *tai02-1.5* | 15x15 | 3342 | 3192 | 3150 | 3583 |
| *tai02-1.6* | 15x15 | 2197 | 2644 | 2551 | 2866 |
| *tai03-1.3* | 15x15 | 5478 | 5808 | 5557 | 5480 |
| *tai03-1.5* | 15x15 | 3900 | 4028 | 3745 | 3745 |
| *tai03-1.6* | 15x15 | 3172 | 2782 | 2904 | 2904 |
| *tai04-1.3* | 15x15 | 4998 | 4383 | 4741 | 4761 |
| *tai04-1.5* | 15x15 | 4290 | 4130 | 3708 | 3586 |
| *tai04-1.6* | 15x15 | 3578 | 3578 | 2391 | 2593 |
| *tai05-1.3* | 15x15 | 6459 | 5144 | 5534 | 6570 |
| *tai05-1.5* | 15x15 | 4359 | 3425 | 5452 | 5097 |
| *tai05-1.6* | 15x15 | 3077 | 2738 | 3581 | 5002 |
| *tai06-1.3* | 15x15 | 6529 | 6529 | 6147 | 6092 |
| *tai06-1.5* | 15x15 | 4621 | 4621 | 4247 | 4526 |
| *tai06-1.6* | 15x15 | 3618 | 3429 | 3383 | 3335 |
| *tai07-1.3* | 15x15 | 5691 | 5646 | 6195 | 5620 |
| *tai07-1.5* | 15x15 | 3944 | 3371 | 3827 | 4050 |
| *tai07-1.6* | 15x15 | 2909 | 2597 | 3653 | 2903 |
| *tai08-1.3* | 15x15 | 4805 | 4805 | 5007 | 4774 |
| *tai08-1.5* | 15x15 | 2700 | 3144 | 3144 | 2713 |
| *tai08-1.6* | 15x15 | 2091 | 2738 | 2765 | 1796 |
| *tai09-1.3* | 15x15 | 6872 | 6496 | 6496 | 6146 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | **0,25** | **0,5** | **0,75** | **1** |
| *tai09-1.5* | 15x15 | 4371 | 4869 | 4334 | 4572 |
| *tai09-1.6* | 15x15 | 3552 | 3561 | 3508 | 3726 |
| *tai10-1.3* | 15x15 | 6997 | 5329 | 5811 | 7107 |
| *tai10-1.5* | 15x15 | 4414 | 3486 | 4391 | 4669 |
| *tai10-1.6* | 15x15 | 2844 | 2784 | 3447 | 4173 |
| *tai11-1.3* | 20x15 | 10564 | 10564 | 9140 | 9967 |
| *tai11-1.5* | 20x15 | 7125 | 7472 | 8010 | 7439 |
| *tai11-1.6* | 20x15 | 5747 | 5747 | 6250 | 6176 |
| *tai12-1.3* | 20x15 | 11615 | 10232 | 10947 | 11690 |
| *tai12-1.5* | 20x15 | 8994 | 8270 | 8751 | 6378 |
| *tai12-1.6* | 20x15 | 7813 | 7202 | 6871 | 5119 |
| *tai13-1.3* | 20x15 | 8881 | 9003 | 9630 | 9859 |
| *tai13-1.5* | 20x15 | 6870 | 7517 | 7054 | 7196 |
| *tai13-1.6* | 20x15 | 5872 | 5807 | 5851 | 6399 |
| *tai14-1.3* | 20x15 | 10277 | 10064 | 10660 | 9302 |
| *tai14-1.5* | 20x15 | 7849 | 8062 | 7703 | 7698 |
| *tai14-1.6* | 20x15 | 7097 | 6137 | 7404 | 7161 |
| *tai15-1.3* | 20x15 | 9827 | 10336 | 9353 | 10335 |
| *tai15-1.5* | 20x15 | 7687 | 8328 | 7042 | 7612 |
| *tai15-1.6* | 20x15 | 5343 | 7104 | 6215 | 5068 |
| *tai16-1.3* | 20x15 | 10141 | 9934 | 11564 | 9419 |
| *tai16-1.5* | 20x15 | 6821 | 8027 | 9214 | 6904 |
| *tai16-1.6* | 20x15 | 6174 | 6715 | 7720 | 6155 |
| *tai17-1.3* | 20x15 | 11181 | 11171 | 10867 | 10281 |
| *tai17-1.5* | 20x15 | 9109 | 7655 | 8346 | 8110 |
| *tai17-1.6* | 20x15 | 6602 | 6969 | 6682 | 7494 |
| *tai18-1.3* | 20x15 | 12210 | 11084 | 10444 | 10515 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | 0,25 | 0,5 | 0,75 | 1 |
| *tai18-1.5* | 20x15 | 9703 | 8197 | 7766 | 8411 |
| *tai18-1.6* | 20x15 | 8848 | 7403 | 7141 | 7297 |
| *tai19-1.3* | 20x15 | 11196 | 11786 | 11668 | 11668 |
| *tai19-1.5* | 20x15 | 8650 | 9877 | 9279 | 9283 |
| *tai19-1.6* | 20x15 | 7953 | 9200 | 8583 | 8032 |
| *tai20-1.3* | 20x15 | 10504 | 12061 | 10541 | 10640 |
| *tai20-1.5* | 20x15 | 8147 | 7640 | 8324 | 8041 |
| *tai20-1.6* | 20x15 | 6935 | 6187 | 6930 | 6989 |
| *tai21-1.3* | 20x20 | 9988 | 10571 | 11358 | 11409 |
| *tai21-1.5* | 20x20 | 6958 | 8018 | 9484 | 8378 |
| *tai21-1.6* | 20x20 | 5334 | 6056 | 7057 | 7057 |
| *tai22-1.3* | 20x20 | 9728 | 10542 | 10918 | 10143 |
| *tai22-1.5* | 20x20 | 7790 | 7111 | 8306 | 8946 |
| *tai22-1.6* | 20x20 | 6152 | 5196 | 5196 | 6736 |
| *tai23-1.3* | 20x20 | 11072 | 9990 | 9933 | 9925 |
| *tai23-1.5* | 20x20 | 7037 | 6511 | 6701 | 6311 |
| *tai23-1.6* | 20x20 | 4676 | 5132 | 5679 | 5220 |
| *tai24-1.3* | 20x20 | 9874 | 10244 | 10291 | 10179 |
| *tai24-1.5* | 20x20 | 7421 | 7417 | 6818 | 7112 |
| *tai24-1.6* | 20x20 | 5348 | 6149 | 5382 | 5815 |
| *tai25-1.3* | 20x20 | 11155 | 11507 | 11130 | 11563 |
| *tai25-1.5* | 20x20 | 6837 | 8398 | 7826 | 7935 |
| *tai25-1.6* | 20x20 | 5241 | 6973 | 6857 | 6812 |
| *tai26-1.3* | 20x20 | 10974 | 11845 | 11845 | 11938 |
| *tai26-1.5* | 20x20 | 7584 | 7375 | 7319 | 7975 |
| *tai26-1.6* | 20x20 | 6140 | 6344 | 6344 | 7388 |
| *tai27-1.3* | 20x20 | 11408 | 10656 | 9728 | 10022 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | **0,25** | **0,5** | **0,75** | **1** |
| *tai27-1.5* | 20x20 | 7572 | 7750 | 6992 | 7156 |
| *tai27-1.6* | 20x20 | 5908 | 5772 | 6200 | 5488 |
| *tai28-1.3* | 20x20 | 7993 | 9175 | 9892 | 9859 |
| *tai28-1.5* | 20x20 | 5799 | 5531 | 6178 | 6509 |
| *tai28-1.6* | 20x20 | 4679 | 4325 | 5080 | 5515 |
| *tai29-1.3* | 20x20 | 10801 | 11181 | 11230 | 10668 |
| *tai29-1.5* | 20x20 | 7026 | 8411 | 7760 | 7451 |
| *tai29-1.6* | 20x20 | 6650 | 7032 | 5401 | 5998 |
| *tai30-1.3* | 20x20 | 8523 | 10282 | 10437 | 10743 |
| *tai30-1.5* | 20x20 | 6127 | 6348 | 6893 | 7152 |
| *tai30-1.6* | 20x20 | 4760 | 5722 | 5490 | 5989 |
| *tai31-1.3* | 30x15 | 27261 | 23926 | 26250 | 28345 |
| *tai31-1.5* | 30x15 | 20546 | 21048 | 21811 | 21015 |
| *tai31-1.6* | 30x15 | 18931 | 19530 | 18866 | 18425 |
| *tai32-1.3* | 30x15 | 26338 | 24388 | 24736 | 26261 |
| *tai32-1.5* | 30x15 | 22175 | 20684 | 21154 | 21948 |
| *tai32-1.6* | 30x15 | 20185 | 18275 | 19644 | 19956 |
| *tai33-1.3* | 30x15 | 28511 | 27274 | 28903 | 29372 |
| *tai33-1.5* | 30x15 | 21517 | 22762 | 24517 | 21745 |
| *tai33-1.6* | 30x15 | 21522 | 20830 | 21565 | 20628 |
| *tai34-1.3* | 30x15 | 25276 | 26030 | 25906 | 26772 |
| *tai34-1.5* | 30x15 | 23216 | 20528 | 22149 | 22511 |
| *tai34-1.6* | 30x15 | 19913 | 18796 | 19608 | 20834 |
| *tai35-1.3* | 30x15 | 22907 | 24414 | 24817 | 25174 |
| *tai35-1.5* | 30x15 | 20240 | 19903 | 19591 | 21331 |
| *tai35-1.6* | 30x15 | 18224 | 18812 | 18300 | 18510 |
| *tai36-1.3* | 30x15 | 28214 | 26040 | 26351 | 27782 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | 0,25 | 0,5 | 0,75 | 1 |
| *tai36-1.5* | 30x15 | 20020 | 23354 | 21774 | 23004 |
| *tai36-1.6* | 30x15 | 16962 | 19686 | 17977 | 20308 |
| *tai37-1.3* | 30x15 | 25500 | 24062 | 25897 | 26610 |
| *tai37-1.5* | 30x15 | 19573 | 20222 | 21501 | 21889 |
| *tai37-1.6* | 30x15 | 18718 | 17888 | 17908 | 18438 |
| *tai38-1.3* | 30x15 | 26423 | 25881 | 25509 | 26492 |
| *tai38-1.5* | 30x15 | 22226 | 20782 | 21464 | 20866 |
| *tai38-1.6* | 30x15 | 19749 | 19835 | 20329 | 21348 |
| *tai39-1.3* | 30x15 | 27246 | 24806 | 26653 | 24336 |
| *tai39-1.5* | 30x15 | 22910 | 22361 | 23117 | 19447 |
| *tai39-1.6* | 30x15 | 20351 | 18500 | 20138 | 19218 |
| *tai40-1.3* | 30x15 | 26493 | 23783 | 24922 | 24470 |
| *tai40-1.5* | 30x15 | 21605 | 20603 | 20326 | 19392 |
| *tai40-1.6* | 30x15 | 17930 | 18091 | 17606 | 17267 |
| *tai41-1.3* | 30x20 | 26316 | 28129 | 26989 | 29163 |
| *tai41-1.5* | 30x20 | 19350 | 20565 | 21101 | 22083 |
| *tai41-1.6* | 30x20 | 16780 | 17227 | 18948 | 18567 |
| *tai42-1.3* | 30x20 | 26932 | 26068 | 26690 | 26714 |
| *tai42-1.5* | 30x20 | 22240 | 21726 | 21417 | 21421 |
| *tai42-1.6* | 30x20 | 20428 | 18757 | 18007 | 19135 |
| *tai43-1.3* | 30x20 | 24086 | 25218 | 26299 | 26403 |
| *tai43-1.5* | 30x20 | 19100 | 20016 | 22791 | 21676 |
| *tai43-1.6* | 30x20 | 18853 | 18564 | 18984 | 15518 |
| *tai44-1.3* | 30x20 | 26207 | 26078 | 26442 | 24993 |
| *tai44-1.5* | 30x20 | 22035 | 19995 | 20744 | 19620 |
| *tai44-1.6* | 30x20 | 18477 | 19944 | 15072 | 17789 |
| *tai45-1.3* | 30x20 | 24812 | 25576 | 28542 | 29119 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | **0,25** | **0,5** | **0,75** | **1** |
| *tai45-1.5* | 30x20 | 21366 | 19580 | 23087 | 21199 |
| *tai45-1.6* | 30x20 | 18060 | 18136 | 18377 | 18161 |
| *tai46-1.3* | 30x20 | 25988 | 28183 | 29678 | 29251 |
| *tai46-1.5* | 30x20 | 21425 | 22558 | 21855 | 22434 |
| *tai46-1.6* | 30x20 | 18676 | 18654 | 20262 | 18478 |
| *tai47-1.3* | 30x20 | 24131 | 24554 | 23939 | 25523 |
| *tai47-1.5* | 30x20 | 20113 | 19324 | 18387 | 20378 |
| *tai47-1.6* | 30x20 | 18383 | 16479 | 15630 | 15770 |
| *tai48-1.3* | 30x20 | 28831 | 28591 | 26155 | 27238 |
| *tai48-1.5* | 30x20 | 22703 | 21715 | 22410 | 21261 |
| *tai48-1.6* | 30x20 | 19451 | 19244 | 19426 | 18054 |
| *tai49-1.3* | 30x20 | 28332 | 26036 | 28694 | 27334 |
| *tai49-1.5* | 30x20 | 21286 | 21141 | 21503 | 21196 |
| *tai49-1.6* | 30x20 | 19912 | 18019 | 18752 | 19516 |
| *tai50-1.3* | 30x20 | 26800 | 27139 | 26174 | 26223 |
| *tai50-1.5* | 30x20 | 18366 | 20450 | 20697 | 20669 |
| *tai50-1.6* | 30x20 | 18508 | 18032 | 18698 | 18083 |
| *tai51-1.3* | 50x15 | 73346 | 78021 | 75626 | 75123 |
| *tai51-1.5* | 50x15 | 64184 | 65923 | 66789 | 69976 |
| *tai51-1.6* | 50x15 | 63433 | 63660 | 62841 | 62334 |
| *tai52-1.3* | 50x15 | 80227 | 75841 | 78115 | 77996 |
| *tai52-1.5* | 50x15 | 69331 | 71920 | 68556 | 71193 |
| *tai52-1.6* | 50x15 | 67635 | 67000 | 62546 | 68354 |
| *tai53-1.3* | 50x15 | 75219 | 75851 | 75643 | 75117 |
| *tai53-1.5* | 50x15 | 68219 | 67082 | 68511 | 67667 |
| *tai53-1.6* | 50x15 | 61974 | 61640 | 62542 | 63323 |
| *tai54-1.3* | 50x15 | 79737 | 75487 | 76838 | 77666 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | 0,25 | 0,5 | 0,75 | 1 |
| *tai54-1.5* | 50x15 | 70437 | 68837 | 68918 | 70798 |
| *tai54-1.6* | 50x15 | 67955 | 63015 | 64619 | 68006 |
| *tai55-1.3* | 50x15 | 79979 | 78713 | 76247 | 76469 |
| *tai55-1.5* | 50x15 | 70599 | 70351 | 67723 | 70247 |
| *tai55-1.6* | 50x15 | 68068 | 64812 | 66736 | 65092 |
| *tai56-1.3* | 50x15 | 79784 | 79003 | 78933 | 77692 |
| *tai56-1.5* | 50x15 | 70819 | 68789 | 68679 | 70298 |
| *tai56-1.6* | 50x15 | 64907 | 66832 | 65150 | 66853 |
| *tai57-1.3* | 50x15 | 77574 | 76316 | 79555 | 77282 |
| *tai57-1.5* | 50x15 | 66874 | 66124 | 69233 | 68771 |
| *tai57-1.6* | 50x15 | 65017 | 59737 | 65327 | 64158 |
| *tai58-1.3* | 50x15 | 81147 | 80860 | 78188 | 81807 |
| *tai58-1.5* | 50x15 | 70551 | 72465 | 71337 | 71269 |
| *tai58-1.6* | 50x15 | 68067 | 68476 | 67403 | 69635 |
| *tai59-1.3* | 50x15 | 75639 | 76185 | 77193 | 75739 |
| *tai59-1.5* | 50x15 | 68334 | 69753 | 67916 | 68561 |
| *tai59-1.6* | 50x15 | 66863 | 67251 | 65527 | 66560 |
| *tai60-1.3* | 50x15 | 72889 | 74785 | 71372 | 70216 |
| *tai60-1.5* | 50x15 | 65706 | 67115 | 63633 | 64352 |
| *tai60-1.6* | 50x15 | 65459 | 61084 | 62739 | 62061 |
| *tai61-1.3* | 50x20 | 80927 | 77597 | 79814 | 79385 |
| *tai61-1.5* | 50x20 | 63770 | 64621 | 64678 | 67621 |
| *tai61-1.6* | 50x20 | 60157 | 66966 | 66556 | 65795 |
| *tai62-1.3* | 50x20 | 80377 | 76850 | 75782 | 77961 |
| *tai62-1.5* | 50x20 | 68368 | 64849 | 66381 | 65339 |
| *tai62-1.6* | 50x20 | 60682 | 60409 | 60453 | 62138 |
| *tai63-1.3* | 50x20 | 76129 | 81118 | 73563 | 78740 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | **0,25** | **0,5** | **0,75** | **1** |
| *tai63-1.5* | 50x20 | 65418 | 65262 | 67269 | 68815 |
| *tai63-1.6* | 50x20 | 60586 | 59112 | 65140 | 64189 |
| *tai64-1.3* | 50x20 | 75691 | 80708 | 76621 | 75856 |
| *tai64-1.5* | 50x20 | 65103 | 64128 | 64418 | 68537 |
| *tai64-1.6* | 50x20 | 61817 | 59316 | 58215 | 58905 |
| *tai65-1.3* | 50x20 | 72381 | 76384 | 74881 | 72332 |
| *tai65-1.5* | 50x20 | 63435 | 63463 | 65893 | 61582 |
| *tai65-1.6* | 50x20 | 59340 | 58266 | 57812 | 60722 |
| *tai66-1.3* | 50x20 | 76191 | 78837 | 78917 | 75042 |
| *tai66-1.5* | 50x20 | 66219 | 65975 | 65711 | 62884 |
| *tai66-1.6* | 50x20 | 61308 | 62396 | 61021 | 63197 |
| *tai67-1.3* | 50x20 | 78521 | 77119 | 76525 | 78805 |
| *tai67-1.5* | 50x20 | 69497 | 70606 | 66303 | 69365 |
| *tai67-1.6* | 50x20 | 64305 | 63838 | 61934 | 61695 |
| *tai68-1.3* | 50x20 | 77249 | 74663 | 80786 | 78411 |
| *tai68-1.5* | 50x20 | 64065 | 69690 | 64610 | 65454 |
| *tai68-1.6* | 50x20 | 58447 | 63544 | 60109 | 60950 |
| *tai69-1.3* | 50x20 | 78599 | 77151 | 77372 | 76420 |
| *tai69-1.5* | 50x20 | 66861 | 67144 | 70110 | 66902 |
| *tai69-1.6* | 50x20 | 65676 | 62305 | 64142 | 62767 |
| *tai70-1.3* | 50x20 | 75633 | 80167 | 81143 | 80978 |
| *tai70-1.5* | 50x20 | 68548 | 71167 | 70439 | 68602 |
| *tai70-1.6* | 50x20 | 62369 | 64608 | 64247 | 62884 |
| *tai71-1.3* | 100x20 | 339501 | 328377 | 336290 | 327528 |
| *tai71-1.5* | 100x20 | 306214 | 304892 | 306983 | 310055 |
| *tai71-1.6* | 100x20 | 287336 | 299545 | 292274 | 292125 |
| *tai72-1.3* | 100x20 | 323705 | 329349 | 322503 | 331184 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | 0,25 | 0,5 | 0,75 | 1 |
| *tai72-1.5* | 100x20 | 302734 | 299988 | 305280 | 300201 |
| *tai72-1.6* | 100x20 | 298917 | 285688 | 287645 | 289493 |
| *tai73-1.3* | 100x20 | 328194 | 330204 | 324564 | 325055 |
| *tai73-1.5* | 100x20 | 301352 | 302305 | 300546 | 301134 |
| *tai73-1.6* | 100x20 | 293040 | 291089 | 296046 | 286698 |
| *tai74-1.3* | 100x20 | 349820 | 342966 | 333536 | 336972 |
| *tai74-1.5* | 100x20 | 312636 | 319692 | 316467 | 320774 |
| *tai74-1.6* | 100x20 | 310550 | 300835 | 304470 | 301978 |
| *tai75-1.3* | 100x20 | 326796 | 326027 | 319407 | 331445 |
| *tai75-1.5* | 100x20 | 308173 | 306070 | 299465 | 301508 |
| *tai75-1.6* | 100x20 | 291636 | 295388 | 283120 | 296177 |
| *tai76-1.3* | 100x20 | 332666 | 328404 | 326847 | 327925 |
| *tai76-1.5* | 100x20 | 313064 | 305722 | 307835 | 303268 |
| *tai76-1.6* | 100x20 | 297056 | 302012 | 298257 | 297226 |
| *tai77-1.3* | 100x20 | 326299 | 330731 | 323557 | 332180 |
| *tai77-1.5* | 100x20 | 313888 | 309235 | 301944 | 306828 |
| *tai77-1.6* | 100x20 | 295256 | 292631 | 292394 | 299393 |
| *tai78-1.3* | 100x20 | 325832 | 323724 | 339655 | 325900 |
| *tai78-1.5* | 100x20 | 302626 | 308152 | 308181 | 304391 |
| *tai78-1.6* | 100x20 | 286572 | 291386 | 293416 | 294911 |
| *tai79-1.3* | 100x20 | 339350 | 340298 | 338320 | 333994 |
| *tai79-1.5* | 100x20 | 314973 | 322113 | 313933 | 314629 |
| *tai79-1.6* | 100x20 | 307263 | 305656 | 307084 | 309920 |
| *tai80-1.3* | 100x20 | 319970 | 324155 | 326415 | 322517 |
| *tai80-1.5* | 100x20 | 303045 | 298581 | 300130 | 302172 |
| *tai80-1.6* | 100x20 | 295855 | 287206 | 291706 | 292218 |

*Tabla 84. Resultados obtenidos para las instancias extendidas de Taillard empleando la regla de prioridad ATC con distintos valores de g.*

A continuación, los resultados para las 66 instancias extendidas de Singer y Pinedo para las reglas de prioridad SPT, LPT y EDD.

| | **n x m** | **SPT** | **LPT** | **EDD** |
|---|---|---|---|---|
| *abz5-1.3* | 10x10 | 4284 | 3943 | 3325 |
| *abz5-1.5* | 10x10 | 2875 | 2549 | 2405 |
| *abz5-1.6* | 10x10 | 2173 | 2005 | 1997 |
| *abz6-1.3* | 10x10 | 2199 | 3088 | 1627 |
| *abz6-1.5* | 10x10 | 1322 | 2196 | 825 |
| *abz6-1.6* | 10x10 | 989 | 1866 | 488 |
| *la16-1.3* | 10x10 | 2348 | 3281 | 1980 |
| *la16-1.5* | 10x10 | 1539 | 2368 | 1231 |
| *la16-1.6* | 10x10 | 1158 | 2022 | 909 |
| *la17-1.3* | 10x10 | 2366 | 2745 | 2308 |
| *la17-1.5* | 10x10 | 1613 | 1971 | 1600 |
| *la17-1.6* | 10x10 | 1302 | 1660 | 1291 |
| *la18-1.3* | 10x10 | 2205 | 3544 | 2494 |
| *la18-1.5* | 10x10 | 1322 | 2786 | 1779 |
| *la18-1.6* | 10x10 | 961 | 2436 | 1434 |
| *la19-1.3* | 10x10 | 2125 | 2626 | 1795 |
| *la19-1.5* | 10x10 | 1401 | 1668 | 1063 |
| *la19-1.6* | 10x10 | 1115 | 1254 | 780 |
| *la20-1.3* | 10x10 | 2447 | 3625 | 2107 |
| *la20-1.5* | 10x10 | 1550 | 2739 | 1404 |
| *la20-1.6* | 10x10 | 1137 | 2328 | 1097 |
| *la21-1.3* | 10x10 | 1284 | 2330 | 1640 |
| *la21-1.5* | 10x10 | 569 | 1543 | 896 |
| *la21-1.6* | 10x10 | 303 | 1242 | 638 |
| *la22-1.3* | 10x10 | 1991 | 2383 | 2196 |
| *la22-1.5* | 10x10 | 1319 | 1494 | 1499 |
| *la22-1.6* | 10x10 | 1026 | 1122 | 1223 |
| *la23-1.3* | 10x10 | 1983 | 2545 | 1822 |
| *la23-1.5* | 10x10 | 1131 | 1669 | 1128 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *la23-1.6* | 10x10 | 847 | 1361 | 848 |
| *la24-1.3* | 10x10 | 2499 | 2226 | 1794 |
| *la24-1.5* | 10x10 | 1690 | 1481 | 1069 |
| *la24-1.6* | 10x10 | 1342 | 1215 | 780 |
| *mt10-1.3* | 10x10 | 2404 | 4803 | 1999 |
| *mt10-1.5* | 10x10 | 1594 | 3896 | 1184 |
| *mt10-1.6* | 10x10 | 1306 | 3446 | 836 |
| *orb1-1.3* | 10x10 | 4392 | 4379 | 2522 |
| *orb1-1.5* | 10x10 | 3390 | 3490 | 1694 |
| *orb1-1.6* | 10x10 | 2890 | 3045 | 1298 |
| *orb2-1.3* | 10x10 | 2564 | 3049 | 1966 |
| *orb2-1.5* | 10x10 | 1809 | 2198 | 1238 |
| *orb2-1.6* | 10x10 | 1481 | 1863 | 947 |
| *orb3-1.3* | 10x10 | 3816 | 4688 | 2882 |
| *orb3-1.5* | 10x10 | 2925 | 3734 | 1964 |
| *orb3-1.6* | 10x10 | 2500 | 3373 | 1570 |
| *orb4-1.3* | 10x10 | 2673 | 4918 | 2927 |
| *orb4-1.5* | 10x10 | 1761 | 3941 | 2072 |
| *orb4-1.6* | 10x10 | 1392 | 3455 | 1649 |
| *orb5-1.3* | 10x10 | 2546 | 3243 | 1972 |
| *orb5-1.5* | 10x10 | 1752 | 2435 | 1324 |
| *orb5-1.6* | 10x10 | 1439 | 2062 | 1039 |
| *orb6-1.3* | 10x10 | 3971 | 4581 | 2932 |
| *orb6-1.5* | 10x10 | 3017 | 3725 | 1999 |
| *orb6-1.6* | 10x10 | 2551 | 3298 | 1577 |
| *orb7-1.3* | 10x10 | 1137 | 1159 | 1105 |
| *orb7-1.5* | 10x10 | 788 | 716 | 758 |
| *orb7-1.6* | 10x10 | 623 | 544 | 603 |
| *orb8-1.3* | 10x10 | 3857 | 4717 | 3027 |
| *orb8-1.5* | 10x10 | 3012 | 3941 | 2268 |
| *orb8-1.6* | 10x10 | 2593 | 3597 | 1913 |
| *orb9-1.3* | 10x10 | 3077 | 4478 | 2412 |

| | n x m | SPT | LPT | EDD |
|---|---|---|---|---|
| *orb9-1.5* | 10x10 | 2118 | 3568 | 1761 |
| *orb9-1.6* | 10x10 | 1695 | 3118 | 1464 |
| *orb10-1.3* | 10x10 | 3476 | 3751 | 2616 |
| *orb10-1.5* | 10x10 | 2454 | 2765 | 1798 |
| *orb10-1.6* | 10x10 | 1995 | 2320 | 1419 |

*Tabla 85. Resultados obtenidos para las instancias extendidas de Singer y Pinedo empleando las reglas de prioridad SPT, LPT y EDD.*

Finalmente, los resultados para las 66 instancias extendidas de Singer y Pinedo para la regla de prioridad ATC, para los valores de *g = {0.25, 0.5, 0.75, 1}.*

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | 0,25 | 0,5 | 0,75 | 1 |
| *abz5-1.3* | 10x10 | 4233 | 3938 | 3642 | 3378 |
| *abz5-1.5* | 10x10 | 3094 | 2709 | 2464 | 2131 |
| *abz5-1.6* | 10x10 | 2656 | 2301 | 2480 | 1916 |
| *abz6-1.3* | 10x10 | 3010 | 2299 | 2299 | 2456 |
| *abz6-1.5* | 10x10 | 2016 | 1323 | 1455 | 1455 |
| *abz6-1.6* | 10x10 | 1192 | 985 | 1120 | 1120 |
| *la16-1.3* | 10x10 | 2057 | 2057 | 2553 | 2497 |
| *la16-1.5* | 10x10 | 1449 | 1449 | 1822 | 1665 |
| *la16-1.6* | 10x10 | 989 | 989 | 1449 | 1422 |
| *la17-1.3* | 10x10 | 1839 | 1839 | 2035 | 1907 |
| *la17-1.5* | 10x10 | 1455 | 1109 | 1109 | 1109 |
| *la17-1.6* | 10x10 | 1265 | 1265 | 834 | 834 |
| *la18-1.3* | 10x10 | 2212 | 3099 | 3038 | 2099 |
| *la18-1.5* | 10x10 | 1315 | 1793 | 1957 | 1822 |
| *la18-1.6* | 10x10 | 1022 | 1448 | 1391 | 1391 |
| *la19-1.3* | 10x10 | 2300 | 1830 | 2083 | 1898 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | 0,25 | 0,5 | 0,75 | 1 |
| *la19-1.5* | 10x10 | 1655 | 1097 | 1097 | 1097 |
| *la19-1.6* | 10x10 | 872 | 987 | 987 | 759 |
| *la20-1.3* | 10x10 | 1872 | 1922 | 2051 | 2051 |
| *la20-1.5* | 10x10 | 1189 | 1105 | 1117 | 1117 |
| *la20-1.6* | 10x10 | 938 | 1015 | 822 | 822 |
| *la21-1.3* | 10x10 | 1859 | 1486 | 1486 | 1486 |
| *la21-1.5* | 10x10 | 1108 | 937 | 937 | 937 |
| *la21-1.6* | 10x10 | 718 | 679 | 679 | 679 |
| *la22-1.3* | 10x10 | 2408 | 2408 | 2557 | 2557 |
| *la22-1.5* | 10x10 | 1405 | 1454 | 2085 | 2085 |
| *la22-1.6* | 10x10 | 1044 | 1121 | 1639 | 1639 |
| *la23-1.3* | 10x10 | 2049 | 2491 | 2577 | 2577 |
| *la23-1.5* | 10x10 | 1711 | 1711 | 1862 | 1798 |
| *la23-1.6* | 10x10 | 1399 | 1372 | 1372 | 1514 |
| *la24-1.3* | 10x10 | 2414 | 1861 | 1857 | 2082 |
| *la24-1.5* | 10x10 | 1375 | 1140 | 1312 | 1522 |
| *la24-1.6* | 10x10 | 939 | 918 | 1164 | 1228 |
| *mt10-1.3* | 10x10 | 2627 | 2526 | 2403 | 2403 |
| *mt10-1.5* | 10x10 | 1641 | 1753 | 1523 | 1523 |
| *mt10-1.6* | 10x10 | 1217 | 1432 | 1447 | 1447 |
| *orb1-1.3* | 10x10 | 3987 | 3235 | 3848 | 3922 |
| *orb1-1.5* | 10x10 | 2536 | 2924 | 2087 | 2926 |
| *orb1-1.6* | 10x10 | 2083 | 2458 | 1505 | 2932 |
| *orb2-1.3* | 10x10 | 2019 | 2279 | 2288 | 2045 |
| *orb2-1.5* | 10x10 | 1353 | 1461 | 1516 | 1288 |
| *orb2-1.6* | 10x10 | 1159 | 1121 | 1333 | 1258 |
| *orb3-1.3* | 10x10 | 3479 | 3579 | 3234 | 3234 |
| *orb3-1.5* | 10x10 | 2958 | 2458 | 2338 | 2378 |
| *orb3-1.6* | 10x10 | 2244 | 1976 | 1940 | 1980 |
| *orb4-1.3* | 10x10 | 2754 | 2754 | 2913 | 2913 |

| | n x m | ATC | | | |
|---|---|---|---|---|---|
| | | **0,25** | **0,5** | **0,75** | **1** |
| *orb4-1.5* | 10x10 | 1564 | 2042 | 2277 | 2277 |
| *orb4-1.6* | 10x10 | 1223 | 1223 | 1805 | 1805 |
| *orb5-1.3* | 10x10 | 1777 | 2459 | 2431 | 2589 |
| *orb5-1.5* | 10x10 | 1324 | 1662 | 1266 | 1701 |
| *orb5-1.6* | 10x10 | 1039 | 1039 | 1147 | 1390 |
| *orb6-1.3* | 10x10 | 4039 | 4039 | 3263 | 3263 |
| *orb6-1.5* | 10x10 | 2646 | 2646 | 2714 | 2714 |
| *orb6-1.6* | 10x10 | 2190 | 2190 | 2142 | 2252 |
| *orb7-1.3* | 10x10 | 1511 | 1208 | 1328 | 1358 |
| *orb7-1.5* | 10x10 | 850 | 738 | 882 | 821 |
| *orb7-1.6* | 10x10 | 685 | 583 | 583 | 729 |
| *orb8-1.3* | 10x10 | 3858 | 3280 | 3281 | 3163 |
| *orb8-1.5* | 10x10 | 3031 | 2999 | 2522 | 2522 |
| *orb8-1.6* | 10x10 | 2565 | 2664 | 2238 | 2238 |
| *orb9-1.3* | 10x10 | 2413 | 2413 | 2413 | 2177 |
| *orb9-1.5* | 10x10 | 1714 | 1714 | 1714 | 1419 |
| *orb9-1.6* | 10x10 | 1417 | 1417 | 1417 | 1107 |
| *orb10-1.3* | 10x10 | 2420 | 2059 | 1964 | 1998 |
| *orb10-1.5* | 10x10 | 1602 | 1602 | 1597 | 1657 |
| *orb10-1.6* | 10x10 | 3265 | 2917 | 2712 | 3344 |

*Tabla 86. Resultados obtenidos para las instancias extendidas de Singer y Pinedo empleando la regla de prioridad ATC con los distintos valores de g.*

## Anexo V: Formato del fichero de texto I/O

A continuación, se mostrará el formato de los ficheros de instancias del *JSS* empleados en el estudio experimental.

- **Instancia básica de Taillard**

Nb of jobs, Nb of Machines, Time Seed, Machine Seed, Upper Bound, Lower Bound
n m ts ms ub lb
Times
$T_{1,1}$ $T_{1,2}$ $T_{1,3}$…
$T_{2,1}$ $T_{2,2}$ $T_{2,3}$…
Machines
$M_{1,1}$ $M_{1,2}$ $M_{1,3}$…
$M_{2,1}$ $M_{2,2}$ $M_{2,3}$…
…

donde:

- *n* es el número de trabajos.
- *m* es el número de máquinas.
- *ts* es la semilla de tiempo.
- *ms* es la semilla de máquina.
- *ub* es la cota superior.
- *lb* es la cota inferior.
- $T_{i,j}$ es el tiempo de procesamiento del trabajo *i* y de la operación *j*.
- $M_{i,j}$ es la máquina correspondiente al trabajo *i* y la operación *j*.

Un ejemplo de una instancia de Taillard básica es la *tai02x03.txt:*

*Nb of jobs, Nb of Machines, Time seed, Machine seed, Upper bound, Lower bound*

*2 3 1 840612802 20 0*

*Times*

*3 1 3*

*2 2 2*

*Machines*

*1 2 3*

*3 1 2*

- **Instancia extendida**

nJobs, nMachines, nTotalOperations
$M_{st1}$ $d_j$ $w_j$ nOperations $M_{1,1}$ $T_{1,1}$ $M_{1,2}$ $T_{1,2}$ …
$M_{st1}$ $d_j$ $w_j$ nOperations $M_{2,1}$ $T_{2,1}$ $M_{2,2}$ $T_{2,2}$ …
…

Este formato es aplicable tanto a las instancias extendidas de Taillard como a las 66 de Singer y Pinedo.

Un ejemplo de una instancia extendida, concretamente la versión extendida de la instancia básica de Taillard mostrada anteriormente, es la *tai02x03ext.txt:*

2 3 6

0 10 4.0 3 1 3 2 1 3 3

0 7 2.0 3 3 2 1 2 2 2

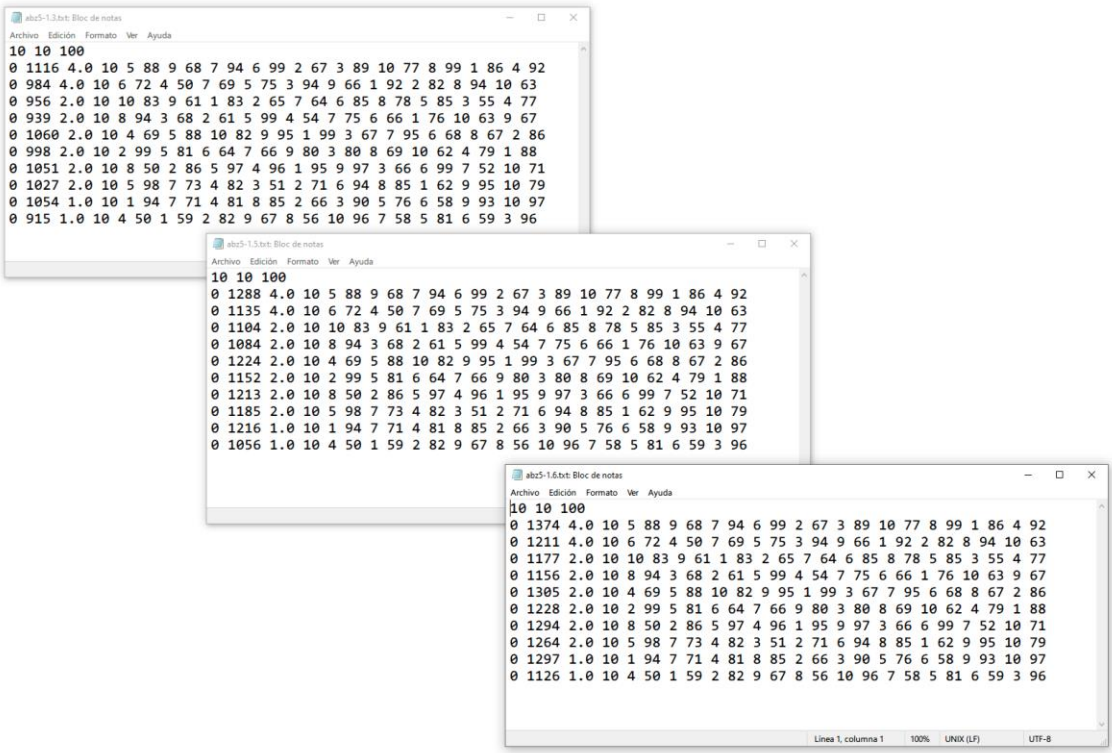



*Ilustración 20. Ejemplo de la instancia extendida abz5 con los 3 factores de fecha límite.*

- **Fichero de salida**

El fichero que se ha de generar tras analizar los datos obtenidos de las instancias será un fichero Excel de formato ".xlsx" con el siguiente formato:

| | ***regla de prioridad 1*** | ***regla de prioridad 2*** | … |
|---|---|---|---|
| *instancia1* | *resultado11* | *resultado12* | |
| *instancia2* | *resultado21* | *resultado22* | |
| … | | | … |

En donde cada fila sería una instancia analizada (o, en el caso de haberse analizado una sola instancia, una sola fila) y cada columna una regla de prioridad aplicada. Las celdas contendrían el resultado de la ejecución en función de la función objetivo que se decidiese tratar.

## Referencias

[1] B. Giffler and G. L. Thomson. Algorithms for solving production scheduling problems. Operations Research, 8:487-503, 1960.

[2] [Roy1964] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec constraintes disjonctives. Note d.s. no. 9 bis, d6c, SEMA, Matrouge, Paris, 1964.

[3] Egon Balas. Machine Sequencing Via Disjunctive Graphs: An Implicit Enumeration Algorithm. Operations Research 17(6):941-957. (1969).

[4] A. S. (1960) On the Job-Shop Scheduling Problem, Operations Research, vol 8, 219-223

[5] A. H. G. Rinnooy Kan. Machine Scheduling Problems. Classification, complexity and computations (1976).

[6] Holthaus, O. and Rajendran, C. (1997). Efficient dispatching rules for scheduling in a job shop. International Journal of Production Economics, 48:87-105.

[7] Holthaus, O. and Rajendran, C. (2000). Efficient jobshop dispatching rules: Further

developments. Production Planning & Control, 11:171-178.

[8] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics, 49:107-127, 1994.

[9] N. J. Nilsson. Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, New York, 1971.

[10] M. Sierra and R. Varela. Optimal scheduling with heuristic best First search. Lecture Notes in Computer Science, 3673:173-176, 2005.

[11] M. Sierra and R. Varela. Pruning by dominance in best-first search for the job shop scheduling problem with total flow time. Journal of Intelligent Manufacturing (JIM), To appear, 2009.

[12] M. R. Sierra and R. Varela. Pruning search spaces by dominance rule: Case study in the job shop scheduling. International.

[13] M. Dell' Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. Annals of Operational Research, 41:231-252, 1993.

[14] D. C. Mattfeld. Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling. Springer-Verlag, 1995.

[15] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop scheduling problem. Management Science, 42:797-813, 1996.

[16] T. Yamada and R. Nakano. Scheduling by genetic local search with multi-step crossover. In Proceedings of Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV 1996), pages 960-969, 1996.

[17] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. Computers and Operations Research, 35:282-294, 2008.

[18] Garey, M. R., Johnson, D. S. and Sethi, R. (1976) The Complexity of Flow Shop and Job-Shop Scheduling, Mathematics of Operations Research, May, 1(2), 117-129.

[19] Garey, M. R. and Johnson, D. S. (1979) Computers and Intertractability: A Guide to the Theory of NPCompleteness, W. H. Freeman, San Francisco.

[20] A. S. Jain, S. Meeran (1998). Deterministic Job-Shop Scheduling: Past, Present and Future. University of Dundee, Dundee, Scotland, UK.

[21] Francisco J. Gil-Gala, María R. Sierra, Carlos Mencía, Ramiro Varela. Genetic programming with local search to evolve priority rules for scheduling jobs on a machine with time-varying capacity. Swarm and Evolutionary Computation, 66: 100944. 2021.

[22] E.K. Burke, M.R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J.R. Woodward, A clas- sification of hyper-Heuristic approaches: Revisited, in: International Series in Op- erations Research & Management Science, 272, Springer International Publishing, 2019, pp. 453–477.

[23] Kaban, A. K., Othman, Z. and Rohmah, D.S. Comparison of Dispatching Rules in Job-Shop Scheduling Problem using Simulation: a case study. 2012.

[24] https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

[25] *https://www.mail-archive.com/user@poi.apache.org/msg13814.html*

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

[27] https://web.imt-atlantique.fr/x-auto/clahlou/mdl/Benchmarks.html

[28] https://repositori.upf.edu/bitstream/handle/10230/1201/1050.pdf.txt;jsessionid=82DE2131DB33568B5C53372EBBEDE08E?sequence=2

[29] https://digibuo.uniovi.es/dspace/bitstream/handle/10651/31599/Gonzalez2012_postprint.pdf?sequence=2&isAllowed=y

[30] https://www.oracle.com/java/technologies/downloads/

[31] Vlasic, Ivan, Marko Đurasević and Domagoj Jakobovic. “Improving genetic algorithm performance by population initialisation with dispatching rules.” Comput. Ind. Eng. 2019

[32] M. Đurasević, D. Jakobović, K. Knežević Adaptive scheduling on unrelated machines with genetic programming Appl Soft Comput, 2016

[33] J.E.Beasley. OR-Library: distributing test problems by electronic mail. Journal of the Operational Research Society 41(11): 1069-1072. 1990.

[34] J. Beasley. Or-library. http://people.brunel.ac.uk/ mastjjb/jeb/info.html. 1990.

[35] Michael L. Pinedo. Scheduling: Theory, Algorithms, and Systems. 5th ed. 2016 edición.

[36] Marko Đurasević. Automated Design of Dispatching Rules in Unrelated Machines Environment. University of Zagreb. (2018)

[37] Johnson, S. M. (1954) Optimal Two- and Three-Stage Production Schedules with Set-Up Times Included, Naval Research Logistics Quarterly, vol 1, 61-68.

[38] Conway, R. W., Maxwell, W. L. and Miller, L. W. (1967) Theory of Scheduling, Addison-Wesley, Reading Massachusetts.

[39] Akers, S. B. (1956) A Graphical Approach to Production Scheduling Problems, Operations Research, vol 4, 244-245.

[40] Hefetz, N. and Adiri, I. (1982) An Efficient Optimal Algorithm for the Two-Machines Unit-Time Job-Shop Schedule-Length Problem, Mathematics of Operations Research, vol 7, 354-360.

[41] Cook, S. A. (1971) The Complexity of Theorem Proving Procedures, Proceedings of the Third Annual ACM Symposium on the Theory of Computing, Association of Computing Machinery, New York, pp. 151-158.

[42] Karp, R. M. (1972) Reducibility Among Combinatorial Problems, in Miller, R. E. and Thatcher, J. W. (eds) Complexity of Computer Computations, Plenum Press, New York, pp. 85-104.

[43] María Rita Sierra Sánchez. Mejora de Algoritmos de Búsqueda Heurística mediante Poda por Dominancia. Aplicación a problemas de Scheduling. Universidad de Oviedo (2009).

[44] C. Bierwirth and D. C. Mattfeld. Production scheduling and rescheduling with genetic algoritms. Evolutionary Computation, 7:117, 1999.

[45] R. Storer and F. Talbot. New search spaces for sequencing problems with application to job shop scheduling. Management Science, 38:14941509, 1992.

[46] https://es.wikipedia.org/wiki/NP-hard

[47]
https://es.wikipedia.org/wiki/Espacio_de_b%C3%BAsquedahttps://es.wikipedia.org/wiki/Espacio_de_b%C3%BAsqueda

[48] A. A. Juan Fuente, B. López Pérez. Dirección y Planificación de Proyectos Informáticos. Guía de Aprendizaje de la asignatura de Dirección y Planificación de Proyectos Informáticos. (2020)