UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Técnicas inteligentes de optimización de agendas personales

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Rovira Sacie, Joaquín

Tutor: Garrido Tejero, Antonio

Curso 2020-2021

Abstract

The project's aim is to design a schedule optimizer with ample flexibility in order to accurately describe real life enterprises with different long-term goals. We will work on the field of linear programming, focusing on the *resource-constrained project scheduling problem* (RCPSP).

First, a couple of approaches were explored: discrete and continuous. Based on these, the model was expanded in order to better adapt it to the day to day realities of project management.

Then, the model mas implemented using a Python-based approach with help of the Gurobi solver. This formula was selected after various trials with different software solutions. The model was tested throughout various experiments with the main objective of identifying and understanding the model's behavior to changes on each parameter. On one hand, these studies had the primary objective of proving the model solutions' accuracy. The secondary objective was to identify the model's strengths and weaknesses, testing which changes impacted more heavily on its performance.

The model shows promising results. Even tough the maximum project is size limited, it is capable of giving precise answers within a reasonable time frame. At the same time the model is capable of adapting to complex scenarios via the resource, task and mode abstractions it provides.

Keywords: project management, RCPSP, linear programming, optimization, sheduling,

Resumen

El propósito de este proyecto es diseñar un optimizador de tareas con amplia flexibilidad que sea capaz de trabajar con una serie de objetivos para modelar proyectos en el mundo real. Se trabaja en el ámbito de la programación lineal, en concreto el problema conocido como *resource-constrained project scheduling problem* (RCPSP) o planificación de proyectos con restricción de recursos.

En primer lugar, se exploró una serie de modelos: discreto y continuo. Se seleccionó el modelo contínuo, sobre el cual se construyeron varias ampliaciones con el fin de adaptarlo mejor a la realidad del día a día de la gestión de proyectos.

Posteriormente, tras el análisis de diferentes paquetes de software se escogió e implementó el modelo en el resolutor Gurobi. Se probó el modelo, sometiéndolo a una serie de pruebas enfocadas en identificar el comportamiento en función de la variación de diferentes parámetros. Estos estudios tenían el objetivo de poner a prueba la veracidad de las soluciones ofrecidas por el modelo. Como segundo objetivo, identificar los aspectos fuertes y los débiles del modelo, viendo qué cambios afectan con mayor magnitud al tiempo de resolución.

El modelo presenta unos resultados prometedores ya que, aún limitado en el tamaño del proyecto a planificar, ofrece una respuesta exacta siendo capaz de amoldarse a escenarios complejos mediante las abstracciones que recursos, tareas y modos que ofrece.

Palabras clave: gestión de proyectos, RCPSP, programación lineal, optimización, planificación

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Contexto

A lo largo de la historia, el ser humano ha sido capaz de llevar acabo asombrosas proezas de ingenio. Desde las pirámides egipcias hasta el Taj Mahal pasando por la red de calzadas romanas. Resulta difícil de creer que semejantes proyectos que abarcan cientos de años de esfuerzo conjunto por parte de nuestros ancestros hayan sido llevados a cabo sin ningún tipo de gestión de proyectos.

Típicamente los arquitectos o ingenieros de la época eran los encargados de planificar y organizar las tareas a realizar, gestionando recursos y personal. No es hasta el siglo XIX cuando se marca el inicio de la gestión de proyectos moderna tal y como la conocemos hoy en día. Como ejemplo está la construcción del ferrocarril en los Estados Unidos, donde se había de organizar la logística y el trabajo en volúmenes nunca vistos.

Hoy en día el ingeniero Henry Gantt es visto como el fundador de la gestión de proyectos moderna [1]. Desarrolla al inicio del Siglo XX el famoso *«Diagrama de Gantt»*, el cual ha sido utilizado desde la Primera Guerra Mundial hasta hoy día en aplicaciones de software moderno.

Durante los años 50 vemos un auge de estas prácticas, no solo las fuerzas armadas sino que empresas privadas como *Transmountain Oil Pipeline* en Canadá o *Civil & Civic* en Australia comienzan a darle uso en proyectos de construcción.

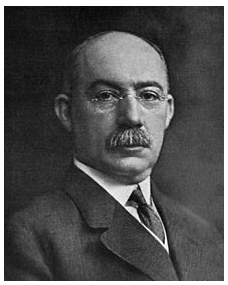


Figura 1.1: Henry Gantt

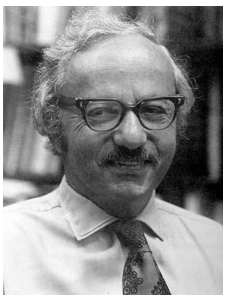


Figura 1.2: George Dantzig

En paralelo, nace la programación lineal moderna con el trabajo de George Dantzig, en 1947 [2]. Dantzig propone que la programación lineal no es únicamente una herramienta para analizar fenómenos económicos, sino que es tremendamente valiosa para dar respuestas verídicas a problemas del mundo real. Bajo esa suposición se aventura a proponer el algoritmo *Simplex* [3], que será hasta hoy día la herramienta primaria para la resolución de problemas de programación lineal. En 1965 [4] se solidifica el concepto de la programación mixta, en la que mezclamos variables continuas y enteras, con la aparición del algoritmo *Branch-and-Bound* que construye sobre el *Simplex*.

A medida que avanza la tecnología de computadores, encontramos la intersección de los dos mundos. Podremos realizar modelos matemáticos que describan el esquema de un proyecto y gracias a los avances en algoritmos de resolución seremos capaces de dar respuestas robustas de manera automática.

1.2. Motivación

El objetivo principal de la gestión de proyectos es *«administrar, planificar, coordinar, seguimiento y control»* de todas las tareas y recursos a asignar, maximizando siempre el beneficio final para la empresa. A niveles bajos de complejidad, una planificación manual de las tareas puede ser relativamente sencilla. A medida que incrementa el número de elementos, la dificultad de la organización aumenta exponencialmente, dado que es necesario mantener las relaciones de precedencia entre actividades, y a su vez asignar una cantidad de recursos limitada (i.e. el número de trabajadores, el presupuesto), que además varía con el tiempo (i.e. el horario de los trabajadores hace variar su disponibilidad).

La falta de una organización óptima a la hora de llevar a cabo un proyecto es decisiva durante la ejecución del proyecto, así como en el resultado final [5]. Una planificación manual de las tareas puede ser relativamente sencilla a niveles bajos de complejidad, cuando el número de tareas y recursos es bajo. Para un proyecto de mayor complejidad, la planificación y órden es crucial para que la secuencia de tareas sea eficaz. Las tareas de un proyecto pueden organizarse sin tener en cuenta la optimización del tiempo. Con una herramienta informática pretendemos planear y realizar el proyecto de una forma rigurosa y eliminar la dedicación laboriosa a la organización de los proyectos que cuenten con muchos elementos. El modelo logra ahorrar en tiempo y cerciora la viabilidad del plan.

Avances en la gestión de proyectos modernos supone la optimización de recursos más eficiente. Desarrollos en la eficiencia de la organización permitirán realizar tareas de mayor magnitud e impacto económico-social, beneficiando así a la comunidad.

1.3. El problema

La habilidad de entregar proyectos a tiempo y ciñéndose a un presupuesto es esencial para ganar ventaja en el mercado competitivo empresarial de hoy en día. La gestión de proyectos entra en este espacio de tremenda complejidad. Por ejemplo, ¿Cómo se planificaría el siguiente conjunto de tareas?

Tarea 1 (t_1)	
Duración:	1.7 hrs.
Predecesores:	$\emptyset$

Tarea 2 (t_2)	
Duración:	2.2 hrs.
Predecesores:	$\emptyset$

Tarea 3 (t_3)	
Duración:	5.2 hrs.
Predecesores:	$\{t_1, t_6\}$

Tarea 4 (t_4)	
Duración:	2.5 hrs.
Predecesores:	$\{t_2\}$

Tarea 5 (t_5)	
Duración:	0.8 hrs.
Predecesores:	$\{t_1, t_4\}$

Tarea 6 (t_6)	
Duración:	3.4 hrs.
Predecesores:	$\{t_2\}$

A esta escala es relativamente sencillo con un poco de esfuerzo obtener la siguiente planificación:

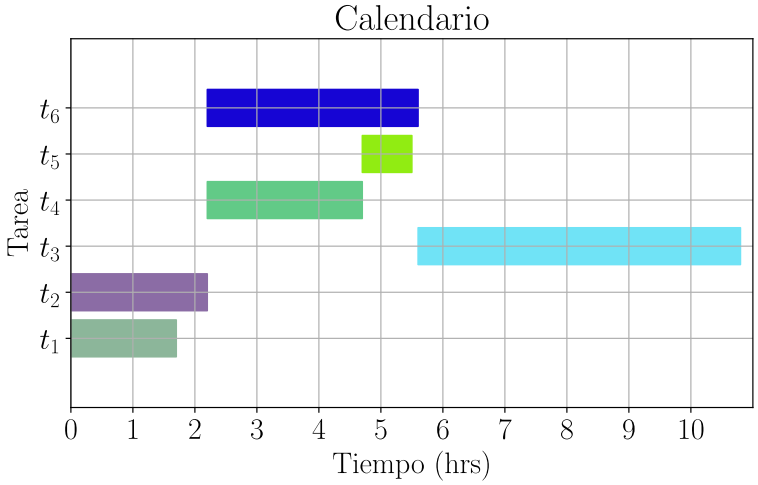


Figura 1.3: Planificación ejemplo

A medida que aumentamos el número de tareas a gestionar intuitivamente se puede entender como la complejidad aumenta a un ritmo insostenible. Además, este problema no es realista. En el mundo real tenemos limitaciones de recursos, tiempo, etc. Al aplicar estas capas de complejidad añadida nos damos cuenta: la gestión de proyectos requiere de una herramienta que ayude a manejar problemas de planificación.

1.4. Objetivos

El objetivo principal del proyecto es diseñar una herramienta que sea capaz de resolver la organización de un proyecto. El modelo debe ser fiel a la realidad para ser capaz de describir proyectos del mundo real. Sin embargo, una respuesta correcta no es suficiente. Además, debe mantener el compromiso con la computabilidad de la respuesta en un tiempo razonable.

Como objetivo transversal, tenemos el análisis de todo un espectro de resolutores con el fin de elegir aquel que mejor se adapte a las necesidades del trabajo. Para completar el proyecto, estudiaremos el comportamiento del modelo viendo debilidades y fortalezas del mismo.

Investigaremos acerca de posibles modelos de la familia de*problemas de programación de*

proyectos con recursos limitados o *resource constrained project scheduling problem* (RCPSP). Ha sido demostrado que este problema existe en la categoría NP-duro[6]. Tienden a explotar en dificultad a medida que crece el número de variables. Es por ello que es crítica la ayuda de los computadores modernos para encontrar soluciones de manera eficiente y en un tiempo razonable. Debemos ser capaces de describir de manera precisa un modelo que sea fiel a la realidad. Paralelamente, es prioritario controlar la complejidad del mismo, de tal manera que sea resoluble en una ventana de tiempo razonable para la gestión del mismo proyecto.

1.5. Metodología

Para abordar el proyecto por partes, definiremos una serie de sub-objetivos. Estos nos permitirán construir sobre los últimos avances en tecnología, verificando y ampliando a medida que se avanza:

- Revisar el estado del arte respecto al RCPSP.
- Describir de manera precisa qué generalizaciones utilizaremos para definir un estándar y construir un supuesto sobre el que desarrollar.
- Definir el modelo matemático que se implementará y analizará a lo largo del proyecto.
- Explorar posibles implementaciones, identificando fortalezas y debilidades y elegir una de ellas.
- Comprobar el funcionamiento correcto y robusto de la implementación y estudiar su comportamiento.
- Sacar conclusiones generales sobre el proyecto además de posible ampliaciones futuras.

Capítulo 2

Revision del estado del arte

2.1. Introducción

Para realizar nuestro optimizador de tareas, investigaremos en el campo de la *programación lineal*. Traduciremos el problema que planteamos a un modelo matemático que exprese correctamente los distintos elementos, relaciones y restricciones. El campo de la programación lineal es considerado un campo de las matemáticas mayormente resuelto por lo que, indagando en la bibliografía encontramos un análogo a nuestro problema.

La programación lineal funciona a través de un sistema de ecuaciones lineales, que se encargan de establecer una serie de restricciones sobre las variables de una función lineal matemática. A través de ello, maximiza o minimiza, optimizando así una función objetivo.

El *problema de programación de proyectos con recursos limitados* o *resource constrained project scheduling problem* (RCPSP) generaliza la idea de tener una serie de tareas a realizar con relaciones entre ellas, además de recursos limitados para realizar las tareas con el objetivo de finalizarlas en el menor tiempo posible.

Existe en la categoría de problemas conocida como NP-duro. Este es especialmente complicado. A medida que el número de tareas crece, el problema tiende a explotar en dificultad de resolución (i.e. tiempo de cálculo). Por eso es importante realizar un modelo eficiente, que minimice el número de variables que explorar, manteniéndose fiel al problema a resolver.

Existen varios modelos en la bibliografía. Vamos a analizar y extraer conclusiones de 2 de ellos para entender la importancia de un buen modelo con el fin de resolver el problema de manera eficiente.

2.2. Programación lineal

Vamos a ver más en detalle cómo nos ayuda la programación lineal. Como hemos visto, en la vida real estamos sujetos a restricciones o condiciones. La planificación de proyectos no es ninguna excepción. Sólo tenemos tanto dinero para gastar, solo tenemos tantos trabajadores, solo disponemos de tanto tiempo. Tenemos la necesidad de usar estas limitaciones en nuestro beneficio, encontrando la manera óptima de repartir y asignar los distintos recursos para completar nuestro proyecto en el menor tiempo posible. En esta búsqueda es en la que nos va a ayudar la programación linear como herramienta de optimización.

La programación lineal busca el máximo o el mínimo de una función lineal. La función lineal, también llamada función objetivo, estará sujeta a una serie de restricciones. Las restricciones se expresan mediante un sistema de in-ecuaciones lineales.

Primero tenemos que identificar nuestras restricciones y traducirlas en un sistema de in-ecuaciones. Tras esto, definimos la función objetivo la cual buscamos maximizar o minimizar. Vamos a ver un ejemplo sencillo.

- *Ikeo S.L.* fabrica dos tipos de muebles: mesas y sillas.
- Cada silla necesita de 2 unidades de madera y 5 horas de trabajo por parte los trabajadores.
- Cada mesa necesita de 4 unidades de madera y 3 horas de trabajo.
- Las sillas se venden a 6€ y las mesas a 10€.
- Cada semana recibe 55 unidades de madera y cuenta con 80 horas de trabajo al tener 2 trabajadores.

El jefe de producción busca obtener el máximo beneficio. Con esta información somos capaces de extraer las restricciones. Primero definimos la notación que vamos a usar:

x_m	número de mesas a fabricar
x_s	número de sillas a fabricar

Una vez definidas las variables, podemos expresar las restricciones como un sistema de in-ecuaciones lineales:

$$2x_s + 4x_m \leq 55 \tag{2.1}$$
$$5x_s + 3x_m \leq 80 \tag{2.2}$$

La in-ecuación (2.1) representa que como mucho se puede usar 55 unidades de madera a la semana. La in-ecuación (2.2) representa que como mucho se puede usar 80 horas de trabajo a la semana. ¿Cual será muestra función objetivo? La función objetivo (2.3) define el beneficio según el número de muebles fabricados multiplicado por su respectivo precio. Además, nuestro objetivo es maximizar por lo tanto nuestra función objetivo será:

$$\text{máx } C = 6x_s + 10x_m \tag{2.3}$$

Si graficamos las restricciones podemos ver el espacio de soluciones que representa nuestro sistema de in-ecuaciones:

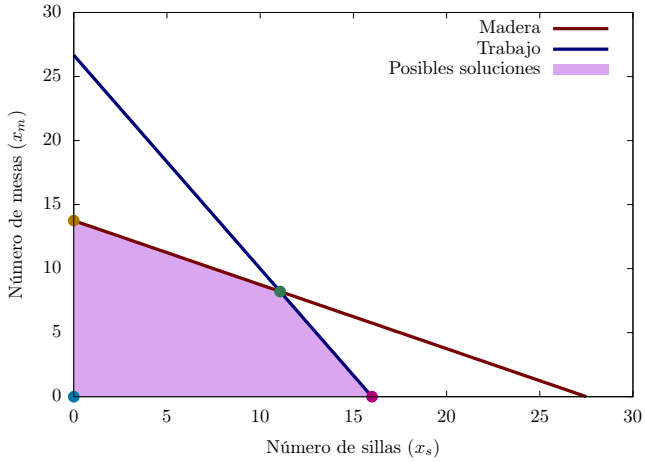


Figura 2.1: Representación gráfica del problema.

Para encontrar la solución óptima existe un algoritmo estándar conocido como el *Simplex* [3]. Se basa en el axioma de que la solución a todo problema de programación lineal se encuentra en los vértices del espacio de soluciones. En este caso, al ser tan sencillo se puede resolver de manera manual buscando el mejor de entre los 4 vértices:

$$\begin{aligned}
(0,0) : C = \quad & 5(0) + 10(0) = 0 \\
(0,13.75) : C = \quad & 5(0) + 10(13.75) = 137.5 \\
(16,0) : C = \quad & 5(16) + 10(0) = 80 \\
(11.07,8.21) : C = \quad & 5(11.07) + 10(8.21) = 137.5
\end{aligned}$$

Por lo tanto podemos concluir que hay dos planes de producción que ofrecen el máximo beneficio para las restricciones actuales. 0 sillas y 13.75 mesas o 11.07 sillas y 8.21 mesas.

Sin embargo, no se pueden producir 13.75 mesas. Hay un método conocido como la programación entera (o mixta) que extiende a la programación lineal. Permite lidiar con este tipo de situaciones y se basa en resolver el problema base e ir añadiendo restricciones que limiten la solución a valores enteros. En este paradigma tenemos dos tipos de variables:

- *Contínuas*, variables que pueden tomar cualquier valor real ($\in \mathbb{R}$).
- *Enteras*, variables que pueden tomar cualquier valor entero ($\in \mathbb{N}$).

Además, del tipo de variables enteras, derivamos las variables *binarias*, muy útiles como mecanismo para expresar la toma de una decisión. Simplemente son variable enteradas limitadas a los valores $[0, 1]$ mediante dos restricciones $x \leq 1$ y $x \geq 0$.

Tendremos que traducir el problema de la planificación de proyectos al marco de la programación lineal. Existe mucha literatura de investigadores que han tratado problemas de características similares, bautizado como *Resource constrained project scheduling problem* (RCPSP) o planificación de proyectos con restricción de recursos.

2.3. Problema de programación de proyectos con recursos limitados (RCPSP)

El RCPSP es una familia de problemas de planificación de proyectos con recursos limitados. Cumple generalmente con las siguientes características:

Consiste en una serie de tareas o actividades que consumen uno o más recursos y tienen relaciones de precedencia entre ellas.

Un **recurso** puede representar, no solo una máquina con capacidad de llevar a cabo múltiples tareas en paralelo, sino que también puede representar entidades como: electricidad, agua, elementos consumibles o incluso habilidades humanas.

Una **solución** es un calendario que posiciona las tareas en un orden tal que no se exceden los límites de recursos y se respetan las relaciones de precedencia.

Generalmente, se asume que una tarea una vez comienza no finaliza hasta que termina su duración (no divisible) y un recurso una vez ha sido utilizado, se libera para su uso futuro (i.e. una máquina industrial). El objetivo clásico del problema es reducir el tiempo de realización de las tareas o *«makespan»*. A partir de esta base puede haber un número infinito de variaciones (tipos de recursos, relaciones o tipos de tareas) en función de las posibles características de los elementos del problema.

2.4. Modelo discreto

Durante los inicios de la programación lineal moderna, entre los 50 y los 70, se explora un primer planteamiento para modelar el problema de la gestión de tareas [7, 8]: el modelo discreto.

El modelo discreto plantea como solución dividir el tiempo en franjas constantes (15 minutos, 1 hora, etc.). La decisión de procesar una tarea o no se tomará únicamente al comienzo de una unidad de tiempo. La decisión se mantendrá necesariamente hasta el comienzo de la siguiente unidad. Estará condicionada por las limitaciones de recursos y relaciones de precedencia con otras tareas.

Pongamos el ejemplo de una situación en la que tenemos 3 tareas x, y, z y dos recursos A, B. Las tareas tienen una duración y demanda. Los recursos una disponibilidad constante. Los datos vienen descritos por las siguientes figuras:

Tarea	Duración (hrs.)
x	1
y	0.5
z	2

Figura 2.2: Duración de las tareas

Recurso	Disponibilidad (uds.)
A	1
B	2

Figura 2.3: Disponibilidad de los recursos

Tareas	Recursos (uds.)	
	A	B
x	0.3	0.5
y	0.5	1.5
z	0.6	0.75

Figura 2.4: Demanda de recurso por cada tarea

La figura 2.2 describe la duración en horas de cada tarea. La figura 2.3 define la disponibilidad de cada recurso. La figura 2.4 describe la demanda del recurso A o B de cada tarea para poder procesarse.

El modelo discreto se basa en definir un unidad de tiempo Δt apropiada para poder solucionar el problema. Usaremos el máximo común divisor (mcd) como estimación del valor de la unidad de tiempo Δt. El máximo común divisor de la duraciones es un heurística que nos asegura que la unidad (Δt) es lo más grande posible sin perder precisión.

$$\Delta t = mcd(1;\ 0.5;\ 2) = 0.5$$

La ecuación anterior muestra el cálculo del máximo común divisor de la duración de las tareas x, y y z.

Además, como tenemos que dividir el tiempo, necesitamos un horizonte de planificación (H). El horizonte de planificación es el límite temporal donde cesa la división del tiempo en unidades temporales. Podemos estimar que la solución se encuentra dentro del horizonte de planificación. Esto es una cosa difícil de saber a priori en problemas más complejos. Generalmente, la duración total de las tareas sirve como límite superior para el horizonte de planificación:

$$H = \sum_{t \in tareas} duracion(t) = 3.5$$

De esta manera, podemos inicializar una tabla que representa la planificación en blanco:

Tareas	Tiempo						
	0.0	0.5	1.0	1.5	2.0	2.5	3.0
x							
y							
z							

Figura 2.5: Representación de problema inicial. $\Delta t = 0.5; H = 3.5$

Como vemos en la figura 2.5, cada fila representa una tarea a realizar. Cada columna representa una unidad de tiempo. La unidad de tiempo (columna) 0.5 representa el intervalo de tiempo t tal que $0.5 \leq t < 1$. De manera general, una unidad de tiempo k representa el intervalo de tiempo t tal que $k \leq t < k + \Delta t$. Además, incluiremos intervalos de tiempo k tal que $k < H$ para todo múltiplo entero de Δt. Es decir:

$$k = n * \Delta t \qquad\qquad k < Hn \in \mathbb{N}$$

Para cada tarea tenemos $\frac{H}{\Delta t} = 7$ decisiones que tomar. En cada unidad habrá que decidir si la tarea en cuestión se está procesando o no, teniendo en cuenta todas las restricciones. En

este sencillo caso, con un poco de esfuerzo se puede encontrar manualmente:

Tareas	Tiempo						
	0.0	0.5	1.0	1.5	2.0	2.5	3.0
x			■	■			
y					■		
z	■	■	■	■			

Figura 2.6: Representación de la solución al problema.

Contrastando la solución (figura 2.6) con las figuras 2.4 y 2.3 vemos que correctamente respeta las limitaciones de recursos:

Recurso	Tiempo							Máx.
	0.0	0.5	1.0	1.5	2.0	2.5	3.0	
A	0.6	0.6	0.9	0.9	0.5	0	0	1
B	0.75	0.75	1.25	1.25	1.5	0	0	2

Figura 2.7: Demanda de recursos en cada unidad de tiempo.

Durante los instantes 1 y 1.5, la demanda de recursos es la suma las tareas que se están ejecutando (x, z). x requiere 0.3 y 0.5 de A y B respectivamente. z requiere 0.6 y 0.75 de A y B respectivamente. Por tanto, la demanda total es 0.9 y 1.25 de A y B respectivamente. No supera la disponibilidad máxima de ambos recursos. De manera similar no se supera para el resto de unidades de tiempo.

Un computador moderno no tendría ningún problema en encontrar la solución a un problema así de sencillo. Sin embargo, podemos anticipar el problema que el mismo Bowman [7] describe como vemos a continuación. En el caso que hemos explorado tenemos, de manera indicativa, 3 tareas con 7 decisiones por tarea. Esto resulta en unas $7 * 3 = 21$ decisiones que tomar. A más tareas tengamos, empeoran otros dos aspectos: H aumentará necesariamente ya que la suma de las duraciones será mayor, Δt generalmente disminuirá en función del *mcd*. Ya que la complejidad de este planteamiento viene dado aproximadamente por *núm. tareas*$\times H \times \frac{1}{\Delta t}$, es de esperar que explote en cuanto añadamos un número realista de tareas, haciendo la computación de soluciones a problemas del mundo real muy lentas si no imposibles.

Además, ya que en un inicio únicamente podemos estimar los parámetros H y Δt, hemos sobre-dimensionado el problema. Esto quiere decir que podríamos haber planteado el problema de otra manera, reduciendo el número de decisiones a tomar. Por ejemplo, si aumentásemos Δt a 1, obtendríamos la siguiente solución equivalente con un número menor de decisiones:

Tareas	Tiempo			
	0.0	1.0	2.0	3.0
x		■		
y			■	
z	■	■		

Figura 2.8: Representación de la solución al problema. $\Delta t = 1; H = 3.5$

Este problema surge si sobre-estimamos el horizonte temporal H o si sub-estimamos la unidad mínima de tiempo Δt. Esto nos puede generar enormes quebraderos de cabeza, ya que

cada problema requerirá de estimaciones que, por su naturaleza, son imprecisas y nos pueden llevar a modelos irresolubles o a un exceso de complejidad innecesario.

2.5. Modelo continuo

Avanzando al siguiente milenio aparece un nuevo planteamiento [9]. El modelo continuo observa que no es necesario analizar cada unidad de tiempo. Los momentos relevantes serán únicamente cuando ocurre un evento significativo. Es decir, cuando comienza o termina el procesamiento de una tarea.

En cada evento, se puede tomar la decisión de procesar una tarea o no, la cual vendrá condicionada por las limitaciones de recursos y relaciones de precedencia con otras tareas. Además, habrá que asociar un valor temporal a ese evento (i.e. cuando ocurre el inicio/fin de una tarea) que no se verá condicionado a divisiones discretas del tiempo.

Repetimos el ejemplo de la situación en la que tenemos 3 tareas x, y, z y dos recursos A, B. Las características del problema son las mismas pero ahora trabajaremos sobre la base del modelo continuo. Nos ceñimos a la definición de eventos y los tiempos asociados a los mismos. Para 3 tareas es fácil apreciar que necesitamos como mucho 3 eventos para marcar el inicio de cada una más 1 evento que marque el final de la planificación:

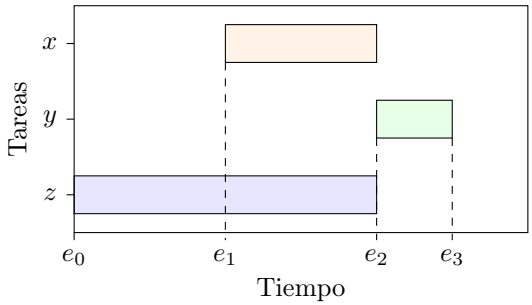


Figura 2.9: Representación de los eventos en la linea temporal.

Evento	e_0	e_1	e_2	e_3
Tiempo asociado	0.0	1.0	2.0	2.5

Figura 2.10: Tiempo asociado a cada evento.

Se puede ver de manera intuitiva como el número de decisiones a tomar se ve importantemente reducido. Donde antes teníamos 7 unidades de tiempo por tarea en las que tomar decisiones, ahora tenemos 4 decisiones por tarea. Esto nos da un total de $4 \frac{decisiones}{tarea} * 3\, tareas = 12\, decisiones$ de manera indicativa, frente a las 21 del modelo discreto. En principio puede no parecer drástica la mejora. Sin embargo, éste es un caso increíblemente sencillo lo cual favorece al modelo discreto y una vez aumentamos la complejidad en 5, 10 ó 15 tareas, la diferencia en cuestión de complejidad es abrumadora.

2.5.1. Contínuo vs. Discreto

Contemplando un ejemplo de mayor complejidad, en el que el número de tareas aumenta a 10, vamos a valorar cómo varía el número de decisiones:

Tarea	Duración (hrs.)
t_0	1
t_1	0.5
t_2	1.3
t_3	2.7
t_4	2.5

Tarea	Duración (hrs.)
t_5	2
t_6	5.3
t_7	1
t_8	0.1
t_9	3.6

Figura 2.11: Duración de las tareas

En esta situación en el modelo discreto tenemos:

$$\textit{núm. tareas} = 10$$

$$H = \sum_{\textit{tarea } t} \textit{duracion}(t) = 20 \textit{ hrs.}$$

$$\Delta t = \textit{mcd}(\textit{duracion de las tareas}) = 0.1 \textit{ hrs.}$$

$$\textit{núm. tareas} \times H \times \frac{1}{\Delta t} = 10 \times 20 \times \frac{1}{0.1} = 2000 \textit{ decisiones}$$

Mientras que en el modelo continuo:

$$\textit{núm. tareas} = 10$$

$$\textit{núm. eventos} = \textit{núm. tareas} + 1 = 11$$

$$\textit{núm. tareas} \times \textit{núm. eventos} = 110 \textit{ decisiones}$$

Este ejemplo ayuda a vislumbrar claramente la diferencia a nivel general en el comportamiento de ambos modelos a medida que aumenta la complejidad del problema a resolver. Además, al eliminar la discretización del tiempo, este enfoque ofrecerá mayor flexibilidad a la hora de crear un modelo robusto a la vez que fiel a la realidad, manteniendo el compromiso con la computabilidad de la solución.

El modelo discreto obliga a establecer un horizonte de planificación. La necesidad de hacerlo recaía en establecer intervalos de tiempo contantes que marcan los instantes en los que puede ocurrir una decisión. Este requisito desaparece con el modelo contínuo, en el que un evento puede ocurrir en cualquier instante a lo largo de la línea temporal. El tiempo de computación se verá menos perjudicado en el modelo contínuo, ya que sólo tiene en cuenta los eventos significativos, mientras que en el modelo discreto debía hacerlo para cada unidad temporal, sea de importancia o no. El modelo continuo tiene varias ventajas frente al discreto:

- Reducimos el número de decisiones por tarea drásticamente.
- Eliminamos la necesidad de estimar el horizonte de planificación. El tiempo de computación no se verá perjudicado en proyectos de larga duración.
- La duración de las tareas podrá ser arbitraria sin afectar al rendimiento, ya que no hace falta estimar la unidad mínima de tiempo.

Por lo tanto, construiremos nuestro modelo sobre el planteamiento básico del modelo continuo.

Capítulo 3

Descripción del modelo

3.1. Introducción

En este apartado procederemos a la definición formal del modelo que utilizaremos. Nos vamos a basar en un supuesto del cual identificaremos las propiedades y comportamientos básicos que debe tener el modelo a fin de describirlo fielmente.

Vamos a enfocar el problema desde el punto de vista de una empresa, la cual tiene una serie de **proyectos** que quiere realizar.

3.2. Supuesto

Proyecto

Un proyecto es un conjunto de **tareas** a realizar, con relaciones de precedencia entre si.

Tareas

Las tareas son todas obligatorias y tienen una duración conocida. Se pueden dividir en dos categorías:

- *Divisibles*: Aquellas que se pueden pausar al final de una jornada y continuarlas en otro momento. Es decir, pueden detenerse y reanudarse.
- *No Divisibles*: Aquellas que, una vez comenzadas, no pueden ser interrumpidas.

Además, cada tarea puede tener una serie de requisitos arbitrarios para poder ser realizada:

- Que la máquina X esté disponible.
- Que dispongamos de Y cantidad de material.
- Que el trabajador Z esté libre.

Este requisito se traducirá a la necesidad de que el **recurso** en cuestión esté disponible para comenzar la tarea. Además, una tarea se podrá realizar de varios **modos**.

3.2.1. Recursos

Un recurso representa de manera abstracta cualquier elemento que pueda ser necesario para realizar una tarea, sin el cual no se puede realizar. Los recursos serán compartidos entre todas las tareas y se dividen en dos categorías:

- *Renovables*: recursos que, una vez la tarea ha hecho uso de ellos, se liberan y vuelven a estar disponibles (i.e. un trabajador).
- *No Renovables*: recursos que, una vez la tarea ha hecho uso de ellos, son gastados para siempre y no pueden volver a ser utilizados (i.e. dinero).

Los recursos renovables pueden tener disponibilidad variable en función del tiempo. Conoceremos a priori los momentos en el tiempo en los cuales la disponibilidad cambia.

Por ejemplo, un trabajador con jornada de 8 a 14 y de 15 a 17 de lunes a viernes. Dentro de su jornada laboral, la disponibilidad del trabajador será de 1, por lo que tareas que requieran su presencia podrán continuar. Lo contrario ocurre fuera de su jornada laboral donde la disponibilidad del trabajador será de 0.

Además, al acabar una tarea, se pueden generar recursos. Por ejemplo, una tarea de ir a sacar 10€ del banco, sin requisito de recursos, genera 10€ del recurso dinero.

Modos

Esta pequeña empresa tiene un departamento de informática en el cual tenemos 3 trabajadores, A, B y C. Si analizamos un supuesto en el que una tarea x de un proyecto requiriese de 2 trabajadores del departamento de informática, vemos como una misma tarea puede realizarse de 3 maneras distintas. La realizan los trabajadores $(A, B), (A, C)$ o (B, C) (Figura 3.1).

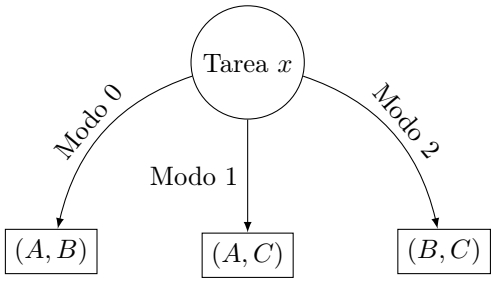


Figura 3.1: Visualización de los distintos modos de una tarea.

Un modo es un conjunto de requisitos de recursos que se debe cumplir durante la duración de la tarea para poder proseguir con ella. Cuando una tarea presenta distintos modos de realización se deberá elegir uno de ellos como configuración con la que realizarla.

A la hora de programar la tarea, se debe elegir un modo de ejecución de manera exclusiva. Cada modo de una tarea podrá tener asociados diferentes costes y duraciones.

3.3. Aplicación al modelo continuo

La formulación basada en eventos resulta la más eficiente en número de decisiones. Esto se traduce a un menor coste computacional. Utilizando la formulación basada en eventos, nos

inspiraremos en el trabajo previo de otros investigadores [9] para construir un modelo que represente adecuadamente el supuesto.

Tendremos tres objetos principales: **tareas, recursos y eventos**. Mediante éstos y las relaciones entre ellos, expresaremos nuestro modelo.

Como vimos en la descripción del modelo continuo, un **evento** es la manera de marcar un instante de tiempo significativo. Tendrá una única propiedad, el instante de tiempo asociado.

Los **recursos** serán divididos en dos categorías:

- *Recursos no renovables*, tendrán una única propiedad, la disponibilidad inicial del recurso. A medida que se vaya realizando tareas que consuman/generen este recurso, la disponibilidad irá variando evento a evento.
- *Recursos renovables*, tendrán como propiedad una lista de instantes de tiempo en los que cambie la disponibilidad (i.e. un calendario). De esta manera, seremos capaces de asociar eventos a esos instantes de tiempo, generando franjas de disponibilidad constante mediante la cual podremos limitar de manera trivial la utilización de los recursos.

Las propiedades de una **tarea** son múltiples:

- Relacionar con otras tareas mediante *relaciones de precedencia* (i, j) que definen que la tarea i debe haber terminado para que la tarea j pueda comenzar.
- Tener una *demanda de recursos* que defina la cantidad necesaria para cada uno de los recursos disponibles.
- Disponer de uno o más *modos* de ejecución. Cada modo ofrece una demanda de recursos diferente a cambio de una duración diferente.
- Además, deberá ser:
  - *Divisible*, permite detener la ejecución temporalmente
  - *No divisible*, indica que una vez comenzada la tarea debe realizarse hasta su fin.

3.3.1. Caso de estudio

A continuación presentaremos un caso concreto sobre el que exponer el modelo extendido. Una empresa, *Proyectos Industriales S.A.* pretende planificar la gestión de los 3 proyectos que tiene por realizar. Cada proyecto se desglosa en tareas de la siguiente manera:

Tareas	Dur. (hrs)	Divisible
a	3	No
b	4.6	No
c	5	Sí
d	1.2	No

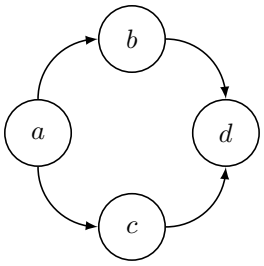


Figura 3.2: Proyecto 1

La tarea d solo puede realizarse una vez han finalizado b y c que a su vez solo pueden llevarse a cabo si ha finalizado a.

	Dur. (hrs)	Divisible
i	3.4	No
j	12.4	Sí
k	6.7	No

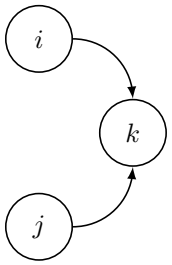


Figura 3.3: Proyecto 2

	Dur. (hrs)	Divisible
x	1.3	No
y	2.5	No
z	13.2	Sí

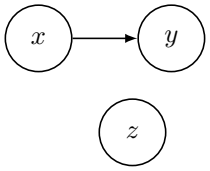


Figura 3.4: Proyecto 3

Las tareas i, j preceden necesariamente a k.
La tarea x precede a y. La tareas z es independiente de ellas.
A lo que recursos respecta, la empresa cuenta con 3 departamentos:

- Departamento A, que cuenta con 3 trabajadores: A_0, A_1 y A_2
- Departamento B, que cuenta con 1 trabajador: B_0
- Departamento C, que cuenta con 2 trabajadores: C_0 y C_1

Además, dispone de 1 máquina industrial M_0 y trabaja con un material no renovable N_0 con una disponibilidad inicial de 10 *uds*. En cuestión de disponibilidad:

- A_0, A_2 y C_1 trabajan de 8 a 16 de lunes a viernes.
- A_1, B_0 y C_0 trabajan de 12 a 20 de lunes a viernes.
- M_0 está disponible lunes, miércoles y viernes.

De esta manera dispondremos de los siguientes conjuntos:

a	$\longleftrightarrow$	t_d^0
b	$\longleftrightarrow$	t_d^1
d	$\longleftrightarrow$	t_d^2
i	$\longleftrightarrow$	t_d^3
k	$\longleftrightarrow$	t_d^4
x	$\longleftrightarrow$	t_d^5
y	$\longleftrightarrow$	t_d^6

Figura 3.5: Tareas divisibles T_d

c	$\longleftrightarrow$	t_n^0
j	$\longleftrightarrow$	t_n^1
z	$\longleftrightarrow$	t_n^2

Figura 3.6: Tareas no divisibles T_n

Además, cada tarea tiene una demanda de recursos. Vamos a definir la demanda de la tarea no en función de los recursos en concreto sino en función del departamento que debe realizarla. Además marcaremos si requiere de la máquina M_0, la cantidad de N_0 que demanda y genera:

A_0	$\longleftrightarrow$	r_r^0
A_1	$\longleftrightarrow$	r_r^1
A_2	$\longleftrightarrow$	r_r^2
B_0	$\longleftrightarrow$	r_r^3
C_0	$\longleftrightarrow$	r_r^4
C_1	$\longleftrightarrow$	r_r^5
M_0	$\longleftrightarrow$	r_r^6

Figura 3.7: Recursos renovables R_r

N_0	$\longleftrightarrow$	r_n^0

Figura 3.8: Recursos no renovables R_n

Tarea	Departamento	M_0	Demanda N_0	Generación N_0
a	A, B	0	0	0
b	C	0	0	0
c	A, C	1	5	0
d	B, C	0	0	5
i	B	1	7	0
j	C	0	0	5
k	A	0	0	0
x	A, C	0	0	0
y	A, B, C	1	4	0
z	A, B	0	0	0

Figura 3.9: Demandas de cada tarea.

El apartado *Departamento* nos genera implícitamente una serie de modos por cada tarea. Por ejemplo, la tarea a procesable por los departamentos A y B nos generará un modo por cada trabajador diferente que lo pueda realizar, así la tarea a tendrá los siguientes modos de ejecución:

Modo	Demanda
m_0	A_0
m_1	A_1
m_2	A_2
m_3	B_0

Figura 3.10: Posibles modos.

Todavía nos hace falta definir el número de eventos. Según el modelo contínuo de *Kone et al.* para un problema con n tareas, el número de eventos necesario es $n+1$, uno por cada tarea con uno extra para marcar el final. Sin embargo, al plantear una disponibilidad de recursos variable, añadimos un «evento significativo» más a tener en cuenta. Éstos serán los eventos de *cambio de recurso* (CR) que marcan momentos en los que la disponibilidad de recursos renovables cambia.

En el caso de los trabajadores, tenemos dos eventos de cambio de recurso diarios. Cuando entran a trabajar, la disponibilidad cambia de 0 a 1. Cuando terminan su horario, la disponibilidad cambia de 1 a 0. En el caso de la máquina tenemos 6 eventos de cambio de recursos por semana, 2 los lunes, miércoles y viernes.

Con esto identificamos un problema, ¿cuándo paramos de generar eventos de cambio de

recurso? Ya que en principio es un calendario semanal que no tiene fin, nos impone la necesidad de añadir un horizonte de planificación H. Esto nos obliga a re-visitar una de las ventajas del modelo continuo que hablábamos en el apartado 2.5:

«Eliminamos la necesidad de estimar el horizonte de planificación. El tiempo de computación no se verá perjudicado.»

Vemos como ya no es del todo cierto. Sin embargo, el resto de ventajas siguen aplicando, igualmente tendremos un número de variables decisión mucho menor que en el modelo discreto.

Vamos a aplicar el mismo criterio de estimación del horizonte de planificación:

$$H = \sum_{t \in T} duracion(t) = 53.3\ hrs. \qquad T \equiv T_d \cup T_n$$

T_d = Conjunto de tareas divisibles

T_n = Conjunto de tareas no divisibles

Es importante entender cómo se comportan los recursos que tenemos a nuestra disposición y cómo va a afectar a nuestro horizonte de planificación. Sería bueno hacer una estimación conservadora de H. Le añadiremos al horizonte de planificación un multiplicador k que nos permita ir aumentando el horizonte si no encontramos solución o reducirlo si sabemos que hemos sobre-estimado el horizonte.

Por lo tanto, si marcamos el horizonte en 53.3 $hrs.$ también hay que definir una fecha de inicio, ya que debemos generar eventos a partir de este. Marcaremos la fecha de inicio en el *lunes, 1 de junio de 2020 a la 00:00* y la unidad de tiempo será en *horas*.

Con todo esto decidido, generamos los eventos de cambio de recurso. Cada evento tiene asociado la nueva disponibilidad de todos los recursos tras el evento. Por ejemplo, en el caso de los trabajadores. Si tenemos dos trabajadores A_0 y A_1 uno con horario de 8h a 16h y el segundo con horario de 12h a 20h tendremos 5 eventos de cambio de recurso. Si empezamos a contar desde las *00:00 h*:

Evento de cambio de recursos	Instante de tiempo asociado ($hrs.$)	Disponibilidad	
		A_0	A_1
cr_0	0	0	0
cr_1	8	1	0
cr_2	12	1	1
cr_3	16	0	1
cr_4	20	0	0

Figura 3.11: Disponibilidad ejemplo tras evento de cambio de recursos (cr).

Mediante este esquema, extraemos los eventos de cambio de recurso para el problema original:

Evento	Tiempo asociado	Disponibilidad
cr_0	0	$[0,0,0,0,0,0,1]$
cr_1	8	$[1,0,1,0,0,1,1]$
cr_2	12	$[1,1,1,1,1,1,1]$
cr_3	16	$[0,1,0,1,1,0,1]$
cr_4	20	$[0,0,0,0,0,0,1]$
cr_5	24	$[0,0,0,0,0,0,0]$
cr_6	32	$[1,0,1,0,0,1,0]$
cr_7	36	$[1,1,1,1,1,1,0]$
cr_8	40	$[0,1,0,1,1,0,0]$
cr_9	44	$[0,0,0,0,0,0,0]$
cr_{10}	48	$[0,0,0,0,0,0,1]$

Figura 3.12: Disponibilidad tras cada evento de cambio de recursos (cr).

El apartado *disponibilidad* tomará la forma de $[r_r^0, r_r^1, r_r^2, r_r^3, r_r^4, r_r^5, r_r^6]$, una lista que representa la disponibilidad de cada recurso renovable en cada evento. Cabe destacar que el evento cr_{11} ya no se calcula puesto que excede el horizonte de planificación $H = 53.3\ hrs$.

3.4. Formulación del modelo

3.4.1. Conjuntos

Como hemos visto en el apartado anterior, hemos identificado los elementos básicos del problema a resolver. Los conjuntos son la representación matemática de los mismos.

Conjunto de todas las tareas	$\boldsymbol{T}$
Conjunto de todas las tareas divisibles	$\boldsymbol{T_d}$
Conjunto de todas las tareas no divisibles	$\boldsymbol{T_n}$
Conjunto de todos los recursos	$\boldsymbol{R}$
Conjunto de todos los recursos renovables	$\boldsymbol{R_r}$
Conjunto de todos los recursos no renovables	$\boldsymbol{R_n}$
Conjunto de todos los eventos	$\boldsymbol{E}$
Conjunto de todas las relaciones de precedencia	$\boldsymbol{P}$
Conjunto de todos los eventos de cambio de recursos	$\boldsymbol{CR}$

Todos los conjuntos serán listas ordenadas de enteros desde 0 hasta el número de elementos del conjunto. Por ejemplo, si tuviéramos 3 eventos, $E = \{0, 1, 2\}$. Por lo tanto, E_{max} será el último elemento del conjunto y N_E será el número de elementos del conjunto. Destacamos que $T \equiv T_d \cup T_n$ y $R \equiv R_r \cup R_n$.

Como definimos en el apartado **2.5 Modelo continuo**, el número de eventos E es derivado como el número de tareas T más el número de eventos de cambio de recursos CR más un evento final.

$$N_E = N_T + N_{CR} + 1$$

Esta asunción da un techo al número de eventos que necesitaremos en el peor de los casos. Por ejemplo, en escenarios donde los recursos estén muy limitados.

3.4.2. Parámetros

Los parámetros del modelo son las propiedades de los elementos básicos que definen el problema. Son valores conocidos antes de comenzar con la resolución del problema. Nos ayudan, junto con los conjuntos, a definir las condiciones concretas de problema a resolver usando el modelo.

Conjunto de modos para la tarea t	$\boldsymbol{modo(t)}$	$\forall\, t \in T$
Duración de la tarea t en modo m	$\boldsymbol{drcn(t,m)}$	$\forall\, t \in T, m \in modo(t)$
Requisito de la tarea t en modo m del recurso k	$\boldsymbol{dmnd(t,m,k)}$	$\forall\, t \in T, m \in modo(t), k \in R$
Tiempo asociado al cr	$\boldsymbol{tmpo(cr)}$	$\forall\, cr \in CR$
Disponibilidad del recurso renovable k_r tras el cr	$\boldsymbol{disp(cr,k_r)}$	$\forall\, cr \in CR, k \in R_r$
Cantidad generada por del recurso no renovable k_n por la tarea t en modo m	$\boldsymbol{gnrd(t,m,k_n)}$	$\forall\, t \in T, m \in modo(t), k_n \in R_n$
Disponibilidad inicial del recurso no renovable k	$\boldsymbol{inic(k_n)}$	$\forall\, k \in R_n$

3.4.3. Variables

Las variables son las incógnitas a resolver. Tenemos dos grupos de variables principales $t(e)$ y $z(t,m,e)$ que nos darán el resultado final de la planificación.

El grupo de variables $t(e)$ indicarán el valor temporal asociado al evento e. Es decir, cuando ocurre.

El grupo de variables $z(t,m,e)$ son binarias por lo que pueden tomar dos valores: 1 o 0. El valor 1 indica que la tarea t está programada para hacerse en modo m en el evento e. El valor 0 indica no est´a programada para realizarse en ese evento.

Los grupos de variables $y(cr,e)$, $d(k_n,e)$ y $x(t,m)$ son variables auxiliares que utilizará el modelo para controlar los límites de recursos renovables, no renovables y la exclusividad entre los distintos modos de una tarea.

El grupo de variables $y(cr,e)$ sirve para que el resolutor asocie eventos de cambio de recursos (cr) con eventos (E) y así saber la disponibilidad de recursos renovables en cada instante de tiempo como veremos más adelante.

El grupo de variables $d(k_n,e)$ realiza el seguimiento del número de recursos no renovables gastado/generado.

El grupo de variables $x(t,m)$ asegura que los modos de las tareas son exclusivos. Si una tarea se programa en modo m en algún evento, no se podrá programar en ningún otro modo en el resto de eventos.

Por último la variable C nos mide el *makespan* o la duración total de la planificación propuesta. La usaremos como variable a minimizar en la función objetivo.

Continua	Tiempo asociado al evento e	$\boldsymbol{t(e)}$	$\forall\, e \in E$
Binaria	Programación de la tarea t en modo m en el evento e	$\boldsymbol{z(t,m,e)}$	$\forall\, e \in E, t \in T, m \in modo(t)$
Binaria	Asocia el evento e al rango temporal de cr	$\boldsymbol{y(cr,e)}$	$\forall\, e \in E, cr \in CR$
Continua	Mide la disponibilidad de recursos no renovable k_n en el evento e	$\boldsymbol{d(k_n,e)}$	$\forall\, e \in E, k_n \in R_n$
Binaria	Define que la tarea t está programada a realizarse en modo m	$\boldsymbol{x(t,m)}$	$\forall\, t \in T, m \in modo(t)$
Continua	Delimita el makespan	$\boldsymbol{C}$	

3.4.4. Restricciones

Como vimos en el apartado **2.2 Programación lineal**, las restricciones son el corazón del modelo. Son éstas las que reflejan las distintas cualidades y atributos del problema general que queremos resolver con el modelo. Es mediante estas que buscamos maneras de reflejar el supuesto dentro de los límites de la programación lineal, expresando de manera algebráica las restricciones en lenguaje natural descritas previamente.

Inicio temporal

Una simple restricción que marca que el inicio de la programación de las tareas ocurre en tiempo 0.

$$t(0) = 0 \tag{3.1}$$

Ordenación

Ordena los eventos secuencialmente en el tiempo. El evento 0 debe ocurrir antes que el evento 1 y así sucesivamente.

$$t(e) \geq t(e-1) \qquad \forall\, e \in E, e > 0 \tag{3.2}$$

Programación

La restricción implica que todas las tareas deben programarse para ser procesadas en al menos un evento. Por lo tanto, para cada tarea al menos una variable $z(t,m,e)$ debe tomar el valor 1.

$$\sum_{e \in E} \sum_{m \in modo(t)} z(t,m,e) \geq 1 \qquad \forall\, t \in T \tag{3.3}$$

Precedencia

Dados un par de tareas $(i,j) \in P$, la tarea i debe finalizar antes de que la tarea j pueda comenzar. Por lo tanto, si en un evento e la tarea j está programada, ni en ese mismo evento

ni en los futuros, puede estar programada la tarea i.

$$z(j, m_j, e) \implies \sum_{e' \geq e} z(i, m_i, e') = 0$$

Es decir, si $z(j, m_j, e) = 1$ la tarea i no puede ser programada a realizarse en eventos siguientes. La suma del número de eventos siguientes en los que la tarea i está programada debe ser 0 ($\sum_{e' \geq e} z(i, m_i, e') = 0$). En caso contrario, $z(j, m_j, e) = 0$ y la suma del número de eventos en los que la tarea i está programada en eventos siguientes, será como mucho N_E:

$$\sum_{e' \geq e} z(i, m_i, e') \leq N_E * (1 - z(j, m_j, e)) \qquad \forall\, e \in E, (i, j) \in P, m_i \in modo(i), m_j \in modo(j) \tag{3.4}$$

Makespan

Define el comportamiento de la variable C. Si hay una tarea programada en algún evento, C tendrá que ser mayor o igual que el valor temporal asociado al inicio del evento siguiente.

$$z(t, m, e) \implies C \geq t(e + 1) \qquad \forall\, e \in E, e < E_{max}, t \in T, m \in modo(t) \tag{3.5}$$

Para ello, añadiremos un evento *dummy* al final. En este evento, no se podrá programar ninguna tarea. Lo definimos de la siguiente forma:

$$z(t, m, E_{max}) = 0 \qquad \forall\, t \in T, m \in modo(t) \tag{3.6}$$

Modos

Como vimos cuando definimos el modelo en el apartado **2.5 Modelo contínuo**, los modos son exclusivos. Si una tarea está programada a realizarse en algún modo, no puede estár programada a realizarse en ningún otro modo. Definimos el comportamiento de la variable auxiliar $x(t, m)$. Como todas las tareas son obligatorias, podemos obligar a que para cada tarea que únicamente una variable $x(t, m)$ esté activa (valor a 1):

$$\sum_{m \in modos(t)} x(t, m) = 1 \qquad \forall\, t \in T \tag{3.7}$$

Ahora, una vez definido el comportamiento de la variable $x(t, m)$ somos capaces de definir una restricción que limite la programación de los demás modos m' si $x(t, m) = 1$. Si una tarea se pausa, no podrá continuar en otro modo que no sea el inicial, como definiremos en el apartado **Duración - No divisibles**. Está restricción sólo es estrictamente necesaria para tareas divisibles.

$$x(t, m) \implies \sum_{m' \neq m} \sum_{e \in E} z(t, m, e) = 0 \qquad \forall\, t \in T, m \in modo(t) \tag{3.8}$$

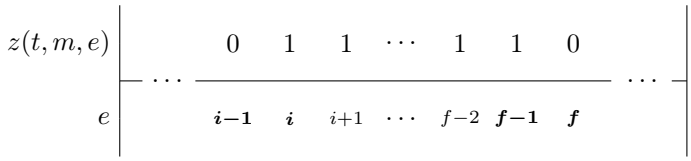
Duración

No divisibles Para las tareas no divisibles, si una tarea empieza a procesarse en un evento i y termina de procesarse en un evento f, el valor temporal asociado a f debe ser mayor que el valor temporal asociado a i más la duración de la tarea.

Dada una variable entera que vale 1 si se cumple esto y es menor en caso contrario (por ahora la llamaremos aux), podemos escribir la restricción como:

$$t(f) \geq t(i) + aux * dr(t)$$

¿Cómo modelamos aux? Cuando una tarea no divisible t en un modo m comienza en un evento i, $z(t, m, i) = 1$ y $z(t, m, i-1) = 0$. Cuando esta termina en un evento f, $z(t, m, f) = 0$ y $z(t, m, f-1) = 1$.



En el esquema anterior, podemos observar el caso en el que $aux = 1$ de manera visual. Desde que empieza hasta que acaba, $z(t, m, e)$ se mantiene a 1 y a 0 fuera de ese rango. Por lo tanto, ese rango es el que debe cumplir la restricción de duración. Si ese rango cumple la restricción, la suma máxima de las 4 variables booleanas vendrá dada por:

$$\begin{array}{lllllllll}
\neg z(t,m,i-1) & + & z(t,m,i) & + & \neg z(t,m,f) & + & z(t,m,f+1) & & = 4 \\
\neg z(t,m,i-1) & + & z(t,m,i) & + & \neg z(t,m,f) & + & z(t,m,f+1) & - & 3 = 1 \\
(1 - z(t,m,i-1)) & + & z(t,m,i) & + & (1 - z(t,m,f)) & + & z(t,m,f+1) & - & 3 = 1 \\
\boldsymbol{-z(t,m,i-1)} & \boldsymbol{+} & \boldsymbol{z(t,m,i)} & \boldsymbol{+} & \boldsymbol{-z(t,m,f)} & \boldsymbol{+} & \boldsymbol{z(t,m,f+1)} & \boldsymbol{-} & \boldsymbol{1 = 1}
\end{array}$$

Con lo que llegamos a la conclusión de que $aux =$

$$z(t,m,i) + z(t,m,f-1) - z(t,m,f) - z(t,m,i-1) - 1 \begin{cases} = 1, \text{si comienza en } i \text{ y termina en } f \\ \leq 0, \text{en otro caso} \end{cases}$$

De este modo tenemos la restricción:

$$t(f) \geq t(i) + (z(t,m,i) + z(t,m,f-1) - z(t,m,f) - z(t,m,i-1) - 1) * dr(t) \tag{3.9}$$

$$\forall\, t \in T, m \in modo(t), i, f \in E, f > i$$

Tenemos un caso especial para el evento inicial $i = e_0$, donde no existe $z(t, m, i-1)$. En ese caso, asumiremos que $z(t, m, i-1) = 0$ ya que ninguna tarea puede haberse programado antes del inicio temporal.

Divisibles Para las tareas divisibles, restringir la duración es más trivial. La diferencia entre el tiempo asociado a un evento e y el siguiente $e+1$ se entiende como la duración del evento. Por

lo tanto, si sumamos las duraciones de los eventos en los que se programa una tarea divisible, obtenemos el tiempo de ejecución de la tarea, el cual debe ser mayor o igual a su duración. Esto sólo será válido si la tarea está programada en modo m según lo marca la variable $x(t, m)$

De esta manera obtenemos:

$$x(t,m) * drcn(t_d) \leq \sum_{e<E_{max}} z(t_d, m, e) * (t(e+1) - t(e)) \tag{3.10}$$

$$\forall\ t_d \in T_d, m \in modo(t_d)$$

Continuidad

Sólo aplica a las tareas no divisibles. Tenemos dos restricciones que describen mediante mecanismos análogos la no divisibilidad.

La primera (3.11) describe como sólo puede haber un inicio de la tarea. Si un evento es inicio de la tarea, el evento no puede haberse programado en eventos previos. Como vimos en el apartado **3.4.4. Duración**, el inicio de una tarea no divisible t en un modo m es el evento en el que es programada por primera vez. Esto ocurre en un evento i cuando $z(t, m, i) = 1$ y $z(t, m, i-1) = 0$.

$$\neg z(t,m,i) \wedge z(t,m,i-1) \implies \sum_{e'<i} z(t,m,e') = 0$$

Expresado como restricción en programación mixta, deberá ser:

- $= 0$ si $z(t, m, i) = 1$ y $z(t, m, i-1) = 0$
- $< N_E$ en otro caso.

$$\sum_{e'<i} z(t,m,e') \leq N_E * (1 + z(t,m,i-1) - z(t,m,i)) \tag{3.11}$$

$$\forall\ t \in T_n, m \in modo(t), e \in E, e > 0$$

La segunda (3.12) describe la misma idea, pero con el final de la tarea en vez de el inicio. Si un evento es fin de la tarea, el evento no puede programarse en eventos posteriores. El fin de una tarea se describe de manera análoga en un evento f cuando $z(t, m, f) = 0$ y $z(t, m, f-1) = 1$.

$$\sum_{e'\geq f} z(t,m,e') \leq N_E * (1 + z(t,m,f) - z(t,m,f-1)) \tag{3.12}$$

$$\forall\ t \in T_n, m \in modo(t), e \in E, e > 0$$

Utilizando estas dos restricciones acotaremos el espacio de búsqueda sin eliminar soluciones óptimas.

Nota: por como está modelada la restricción de duración, esta restricción no es estrictamente necesaria. Sin embargo, restricciones redundantes pueden favorecer la resolución del modelo.

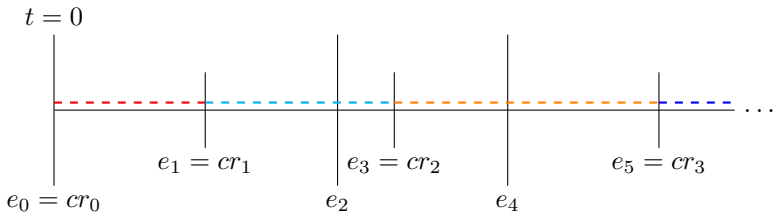
Recursos renovables

Asociación de rangos Vamos a dividir el tiempo en franjas. Dentro de una franja temporal, la disponibilidad de los recursos es constante. Para enlazar una tarea a un instante de tiempo, vamos a introducir un concepto que hará de intermediario entre la realización de las tareas y las distintas franjas de disponibilidad de recursos.

En un evento, puede ocurrir una serie de cosas. Primero, que marque el inicio (o continuación) de una o más tareas. Segundo, que marque el fin de una tarea que estaba realizándose previamente. Tercero, que marque la frontera entre dos franjas de disponibilidad de recursos.

El momento en el que la disponibilidad cambia, lo denominamos un evento del tipo ***resource change*** (***cr***) y tiene un valor temporal conocido ya que es un parámetro del modelo.

Ahora, dentro de una franja la disponibilidad de todos los recursos es fija. Por lo tanto, si asociamos tareas a eventos dentro de esas franjas, para cada evento, el uso por parte de las tareas en proceso debe ser menor que la disponibilidad de recursos de la franja en la que se encuentre.



En este ejemplo tenemos 4 eventos cr asociados a los eventos e_0, e_1, e_3 y e_5.

El evento e_0 pertenece a la franja roja (cr_0).

El evento e_1 y e_2 pertenecen a la franja cian (cr_1).

El evento e_3 y e_4 pertenecen a la franja naranja (cr_2).

El evento e_5 pertenece a la franja azul (cr_3).

En todas ellas se debe respetar los recursos de su franja correspondiente. La restricción (3.13) define que si un evento pertenece a un cr, su valor temporal debe estar dentro del rango. La restricción (3.14) dice que para cada cr al menos un evento debe estar asociado a ese rango. La restricción (3.15) dice que un evento únicamente puede estar asociado a un cr. Una vez hemos definido este comportamiento para las variables $y(cr, e)$ podemos proceder con la restricción de disponibilidad.

$$y(cr,e) \implies \begin{cases} t(e) \geq tmpo(cr) \\ \\ t(e+1) \leq tmpo(cr+1) \end{cases} \qquad \forall\, e \in E, cr \in CR \tag{3.13}$$

$$\sum_{e \in E} y(cr,e) \geq 1 \qquad \forall\, cr \in CR \tag{3.14}$$

$$\sum_{cr \in CR} y(cr,e) = 1 \qquad \forall\, e \in E \tag{3.15}$$

Además, podemos realizar dos asunciones para facilitar la resolución del modelo ya que, de cuanta más información disponga el resolutor, mejores prestaciones obtendremos. Asumimos

que el primer evento está asociado al primer cr y que el último evento está asociado al último cr. Es decir:

$$y(0,0) = 1 \tag{3.16}$$
$$y(CR_{max}, E_{max}) = 1 \tag{3.17}$$

Disponibilidad Una vez asociado cada evento a un cr, simplemente en cada evento, la suma de demandas de recursos de las tareas en proceso debe ser menor o igual a la demanda en esa franja.

$$y(cr,e) \implies \sum_{t \in T} \sum_{m \in modo(t)} dm(t,m,k) * z(t,m,e) \leq bd(cr,k) \qquad \forall\, cr \in CR, e \in E, k \in R_r \tag{3.18}$$

Recursos no renovables

Para restringir los recursos no renovables según su disponibilidad, utilizaremos la variable $d(k_n, e)$ para llevar la cuenta, evento a evento de su disponibilidad. Dada la restricción implícita de no negatividad de todas la variables limitamos la sobreutilización del recurso.

Para el evento e_0 la disponibilidad del recurso k_n será igual al parámetro de disponibilidad inicial, $inic(k_n)$. Además, en cada evento reduciremos la capacidad en función del consumo/demanda de las tareas que inician su programación en ese evento y aumentaremos la capacidad en función de la cantidad generada por las tareas que finalizan su programación.

$$d(k_n, e_i) = d(k_n, e_{i-1}) - consumido + generado \qquad \forall\, e \in E, k_n \in R_n$$

Ahora, ¿qué forma toma el valor *consumido*? Este será igual al valor de la demanda de las tareas que comiencen el este evento. Para las tareas no divisibles, como vimos en la explicación de la restricción de duración de tareas no divisibles (pág. 23), es cuando $z(t_n, m, e) = 1$ y $z(t_n, m, e-1) = 0$:

$$dmnd(t_n, m, k_n) * z(t_n, m, e) * (1 - z(t_n, m, e-1))$$

Para la tareas divisibles, el inicio viene dado por $z(t_d, m, e) = 1$ y todos los eventos previos $z(t_d, m, e') = 0$:

$$dmnd(t_d, m, k_n) * z(t_d, m, e) * \prod_{e' < e} (1 - z(t_d, m, e'))$$

Por lo que el valor consumido será la suma para todas las tareas no divisibles $t_n \in T_n$ de la primera forma más todas las tareas divisibles $t_d \in T_d$ de la segunda forma y teniendo en cuenta sus modos:

$$consumido = \left[\sum_{t_n \in T_n} \sum_{m \in modo(t_n)} dmnd(t_n, m, k_n) * z(t_n, m, e) * (1 - z(t_n, m, e-1))\right] + \left[\sum_{t_d \in T_d} \sum_{m \in modo(t_d)} dmnd(t_d, m, k_n) * z(t_d, m, e) * \prod_{e' < e} (1 - z(t_d, m, e'))\right]$$

De manera análoga podemos definir el valor *generado*. La diferencia está en que ahora en vez del inicio de las tareas, identificaremos el final de las tareas. Para las tareas no divisible, el mecanismo es el mismo que el explicado en la restricción de duración de tareas no divisibles (pág. 23). El final de la tarea no divisible es cuando $z(t_n, m, e) = 0$ y $z(t_n, m, e-1) = 1$:

$$gnrd(t_n, m, k_n) * z(t_n, m, e) * (1 - z(t_n, m, e-1))$$

Para las tareas divisibles, el inicio viene dado por el evento en el que $z(t_n, m, e-1) = 1$ y todos los eventos posteriores $z(t_n, m, e') = 0$:

$$gnrd(t_d, m, k_n) * z(t_d, m, e-1) * \prod_{e' \geq e} (1 - z(t_d, m, e'))$$

Por lo que el valor generado será la suma para todas las tareas no divisibles $t_n \in T_n$ de la primera forma más todas las tareas divisibles $t_d \in T_d$ de la segunda forma y teniendo en cuenta sus modos:

$$generado = \left[\sum_{t_n \in T_n} \sum_{m \in modo(t_n)} gnrd(t_n, m, k_n) * z(t_n, m, e-1) * (1 - z(t_n, m, e))\right] + \left[\sum_{t_d \in T_d} \sum_{m \in modo(t_d)} gnrd(t_d, m, k_n) * \prod_{e' \geq e} (1 - z(t_d, m, e'))\right]$$

Por lo tanto nuestra la restricción será:

$$\begin{aligned} d(k_n, e) = {} & d(k_n, e-1) \\ & - \left[\sum_{t_n \in T_n} \sum_{m \in modo(t_n)} dmnd(t_n, m, k_n) * z(t_n, m, e) * (1 - z(t_n, m, e-1))\right] \\ & - \left[\sum_{t_d \in T_d} \sum_{m \in modo(t_d)} dmnd(t_d, m, k_n) * z(t_d, m, e) * \prod_{e' < e} (1 - z(t_d, m, e'))\right] \\ & + \left[\sum_{t_n \in T_n} \sum_{m \in modo(t_n)} gnrd(t_n, m, k_n) * z(t_n, m, e-1) * (1 - z(t_n, m, e))\right] \\ & + \left[\sum_{t_d \in T_d} \sum_{m \in modo(t_d)} gnrd(t_d, m, k_n) * \prod_{e' \geq e} (1 - z(t_d, m, e'))\right] \end{aligned}$$

$$\forall\, e \in E, e > 0, k_n \in R_n \tag{3.19}$$

Es importante destacar que tendremos un caso especial para e_0, ya que solo hay que detectar el inicio de las tareas y al no haber eventos previos, dependerá únicamente del valor de $z(t, m, 0)$

tanto para tareas divisibles como no divisibles:

$$d(k_n, 0) = inic(k_n) - \left[\sum_{t \in T} \sum_{m \in modo(t)} dmnd(t, m, k_n) * z(t, m, 0)\right]$$

$$\forall\, k_n \in R_n \tag{3.20}$$

A simple vista parece que debería de funcionar este enfoque. Sin embargo, tenemos un problema. La expresiones del tipo:

$$gnrd(t_d, m, k_n) * z(t_d, m, e-1) * \prod_{e' \geq e} (1 - z(t_d, m, e'))$$

no son combinaciones lineales de variables por lo tanto será imposible resolver este modelo. Técnicas modernas de resolución nos permiten resolver expresiones cuadráticas, permitiéndonos expresar restricciones con sumas de multiplicaciones de hasta 2 variables. Se conoce como *Quadratic Programming*. Vamos a transformar la restricción (3.19) en cuadrática mediante el uso de variables auxiliares que definiremos a continuación.

- Por conveniencia definimos $\hat{z}(t, m, e)$ que simplemente será la negación de $z(t, m, e)$.
- El grupo de variables $start(t, m, e)$ es un grupo de variables binarias que servirán para identificar el inicio de las tareas. Valdrá 1 cuando el evento e marque el inicio de la tarea t en modo m.
- El grupo de variables $end(t, m, e)$ es un grupo de variables binarias que servirán para identificar el fin de las tareas. Valdrá 1 cuando el evento e marque el fin de la tarea t en modo m.

Binaria	Negación de $z(t, m, e)$	$\boldsymbol{\hat{z}(t, m, e)}$	$\forall\, e \in E, t \in T, m \in modo(t)$
Binaria	Identifica el inicio de la tarea t en modo m	$\boldsymbol{start(t, m, e)}$	$\forall\, e \in E, t \in T, m \in modo(t)$
Binaria	lIdentifica el fin de la tarea t en modo m	$\boldsymbol{end(t, m, e)}$	$\forall\, e \in E, t \in T, m \in modo(t)$

Para que las variables tengan el comportamiento que se desea, definiremos las restricciones con ese fin. Primero $\hat{z}(t, m, e)$, es sencillamente:

$$\hat{z}(t, m, e) = 1 - z(t, m, e) \qquad \forall\, t \in T, m \in modo(t), e \in E \tag{3.21}$$

Para definir $start(t, m, e)$, usaremos la observación de que, para una tarea divisible, el inicio se da cuando $z(t, m, e) = 1$ y todos los eventos previos están a 0. O lo que es lo mismo, que la suma de $z(t, m, e')$ para todos los eventos $e' < e$ más $\hat{z}(t, m, e)$ valga 0:

$$start(t,m,e) \implies \hat{z}(t,m,e) + \sum_{e'<e} z(t,m,e') = 0 \tag{3.22}$$

$$\forall\, t \in T, m \in modo(t), e \in E$$

Para definir $end(t,m,e)$, usaremos la observación de que, para una tarea divisible, el fin se da cuando $z(t,m,e-1) = 1$ y todos los eventos siguientes están a 0. O lo que es lo mismo, que la suma de $z(t,m,e')$ para todos los eventos $e' \geq e$ más $\hat{z}(t,m,e-1)$ valga 0:

$$end(t,m,e) \implies \hat{z}(t,m,e-1) + \sum_{e'\geq e} z(t,m,e') = 0 \tag{3.23}$$

$$\forall\, t \in T, m \in modo(t), e \in E$$

En el caso del evento 0, ninguna tarea puede haber finalizado por lo que:

$$end(t,m,0) = 0 \tag{3.24}$$

$$\forall\, t \in T, m \in modo(t)$$

Por último, requerimos que se identifique exactamente un inicio y un final para cada tarea. Si no lo requerimos el modelo puede hacer caso omiso de la restricción de recursos no renovables.

$$\sum_{e\in E} \sum_{m\in modo(t)} start(t,m,e) = 1 \qquad \forall\, t \in T \tag{3.25}$$

$$\sum_{e\in E} \sum_{m\in modo(t)} end(t,m,e) = 1 \qquad \forall\, t \in T \tag{3.26}$$

Una vez tenemos las variables auxiliares definidas, procedemos a redeclarar la ecuación (3.19). Quedará de la siguiente manera:

$$\begin{aligned} d(k_n,e) = d(k_n,e-1) \\ - \left[\sum_{t\in T} \sum_{m\in modo(t)} dmnd(t,m,k_n) * start(t,m,e)\right] \\ + \left[\sum_{t\in T} \sum_{m\in modo(t)} gnrd(t,m,k_n) * end(t,m,e)\right] \end{aligned}$$

$$\forall\, e \in E, e > 0, k_n \in R_n \tag{3.27}$$

Una vez definido el comportamiento de la variable $d(k_n,e)$ solo los queda asegurar que la

disponibilidad de los recursos no renovables no sea negativo:

$$d(k_n, e) \geq 0 \qquad \forall\, k_n \in R_n, e \in E \tag{3.28}$$

3.5. Pre-procesado de datos

Una vez llegamos a este punto vamos a realizar un pre-procesado de los datos. Someteremos los datos de entrada a 3 criterios. Si el conjunto de datos supera los 3 criterios, estimaremos que podemos encontrar soluciones factibles. Si no, en función del criterio que se incumpla tendrá un significado diferente como veremos mas adelante.

3.5.1. Grafo acíclico

Tenemos que asegurar que no hay precedencias cíclicas. Por ejemplo, si tenemos dos precedencias (x, y) e (y, x), el problema no tendrá solución puesto que la tarea x no puede empezar hasta que la tarea y acabe y vice versa.

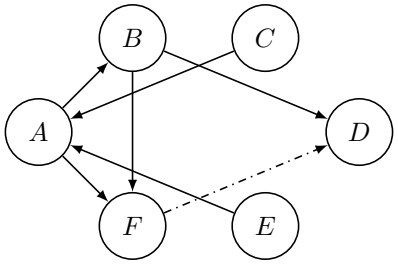
Primero, formaremos un grafo donde los nodos representan las tareas y los vértices dirigidos que los conectan representan las precedencias. Si el grafo tiene orden topológico, no habrá dependencias cíclicas y podemos proceder al resto de pruebas. Si no lo encontramos, podemos asegurar que el grafo presenta ciclos por lo que tendrá precedencias que no se pueden resolver y el problema no tendrá solución.

A continuación explicamos el mecanismo para encontrar el orden topológico del grafo:

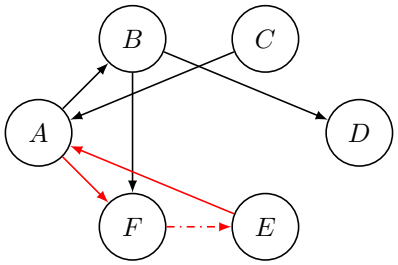
```
L ← Lista vacía donde guardaremos el resultado
Q ← Conjunto de nodos sin vértices entrantes
mientras Q no sea vacía hacer:
    eliminar un nodo n de Q
    insertar n en L
    para cada nodo m con un vértice e de n a m hacer:
        eliminar el vértice e del grafo
        si m no tiene otros vértices entrantes entonces:
            insertar m en Q
si el grafo tiene vértices entonces:
    mensaje error (el grafo contiene ciclos)
si no:
    mensaje (el grafo es acíclico con orden topológico: L)
```

Grafo acíclico con orden topológico:

$$[C, E, A, B, F, D]$$



Grafo cíclico sin orden topológico.



3.5.2. Disponibilidad de recursos no renovables

Vamos a establecer un mecanismo mediante el cual verificaremos que todos los recursos están disponibles en la cantidad mínima para cada tarea mediante todos los modos. Para todos los recursos no renovables siempre se debe cumplir que la capacidad total más la cantidad generada máxima es mayor o igual a la demanda mínima. Es decir:

$$Capacidad + Generado \geq Demanda \qquad \forall\, k_n \in R_n$$

$R_n =$ Recursos no renovables

La *Demanda* total de un recurso no renovable la estimaremos a partir de la demanda de cada tarea. Es importante tener en cuenta que una tarea se puede realizar de varios modos como vimos en apartados anteriores. Por lo tanto, seleccionaremos la demanda mínima de entre todos sus modos. Sumaremos la demanda mínima de todas las tareas. De este modo:

$$Demanda = \sum_{t \in T} \left[\min_{m \in modo(t)} (dmnd(t, m, k_n)) \right]$$

La *Capacidad* hace referencia a la disponibilidad inicial del recurso. Dada por el parámetro $inic(k_n)$.

$$Capacidad = inic(k_n)$$

Por último, el valor *Generado* de un recurso no renovable será la suma de la cantidad máxima que se genere tras cada tarea (*gnrd*). Para una tarea, de entre todos los modos se seleccionará aquel que genere el máximo recurso al finalizar:

$$Generado = \sum_{t \in T} \left[\max_{m \in modo(t)} (gnrd(t, m, k_n)) \right]$$

Si no es cierta la comprobación $Capacidad + Generado \geq Demanda$, podemos asegurar que el problema no tiene solución.

3.5.3. Disponibilidad de recursos renovables

De manera análoga, para estimar si la disponibilidad de los recursos renovables es adecuada, estimaremos que se debe cumplir que la disponibilidad máxima es mayor o igual a la demanda máxima. Es decir:

$$Disponibilidad \geq Demanda \qquad \forall\, k_r \in R_r$$

$R_r =$ Recursos renovables

La *Demanda* de un recurso no renovable la estimaremos a partir de la demanda de cada tarea y su duración. La demanda de cada tarea será la máxima de entre todos los modos de una tarea. Sumaremos la demanda de todas las tareas. De este modo:

$$Demanda = \sum_{t \in T} \left[\max_{m \in modo(t)} (dmnd(t, m, k_n) * drcn(t, m)) \right]$$

La *Disponibilidad* de un recurso renovable viene dada por la suma de la disponibilidad en cada $cr \in CR$ por la duración del mismo cr. La duración del cr será la diferencia entre el tiempo asociado a éste y el siguiente $cr' = cr + 1$. Al tiempo asociado a un cr lo llamaremos $tmpo(cr)$ y a la disponibilidad, $disp(cr)$. De este modo:

$$Disponibilidad = \sum_{cr \in CR} disp(cr, k_r) * (tmpo(cr + 1) - tmpo(cr))$$

La intención es que se cumpla el requisito de la manera más ajustada posible para evitar sub-estimar o sobre-estimar el horizonte de planificación H. Por lo que, mientras se cumpla, reduciremos el horizonte y mientras no se cumpla, lo incrementaremos.

Capítulo 4

Evaluación de los resultados

4.1. Introducción

Una vez tenemos formulado formalmente el modelo matemático que describe el problema, procederemos a la implementación del modelo. Ejecutaremos el caso de estudio y analizaremos los resultados obtenidos con el fin de verificar el correcto funcionamiento del programa. Por último, buscaremos comprender en mayor profundidad el comportamiento del modelo. A partir del caso de estudio, diseñaremos una serie de casos derivados que cambien un parámetro del problema y estudiaremos las diferencias.

4.2. Implementación

Hoy en día, disponemos de un amplio espectro de paquetes de software. Se debe encontrar herramientas para el modelado y la resolución del problema. Podemos dividir el software en dos grandes categorías *libre* y *comercial*. En términos de modelado de problema elegimos de entre los siguientes:

- CPLEX, software comercial de IBM
- PuLP, software libre de Coin-OR
- Gurobi, software comercial de Gurobi
- OR-Tools, software libre de Google

Si indagamos en las propiedades de cada una vemos que el proceso de creación de un modelo es prácticamente idéntico. Los 4 paquetes de software ofrecen las mismas ventajas con interfaces muy similares. Por lo tanto, nuestra decisión se tomará en función del resolutor que queramos utilizar. Tras un análisis de los resolutores que se ofrecen en el mercado encontramos:

- CPLEX, software comercial de IBM
- CBC, software libre de Coin-OR
- Gurobi, software comercial de Gurobi
- GLPK, software libre de FSF (*Free Software Foundation*)

El software libre tiene la ventaja de ser gratuito. Sin embargo, de las dos opciones planteadas, el GLPK tuvo su última actualización en febrero de 2018 y no está al día con los últimos avances. Las demás sí que lo están. A fecha de la redacción de este documento, la última actualización del CBC fue hace 2 días por lo que vemos que sí que está más al día.

El software comercial es por necesidad más actualizado y el mejor mantenido ya que son trabajadores y no voluntarios los que se dedican al proyecto. Los 3 resolutores (CBC, CPLEX y Gurobi) tienen el mismo mecanismo de resolución de problemas. Realizan planos de corte, heurísticas y un preprocesado del modelo. Sin embargo, el software comercial tiene una ventaja sobre CBC. Tienen funciones auxiliares que ayudan al modelado del problema (máx(), mín(), *and*(), *or*(), restricciones $x \implies y \leq z$) que automatiza ciertas técnicas de modelado de forma más eficiente.

A la hora de elegir entre Gurobi y CPLEX, ambas empresas afirman tener el resolutor más rápido. En 2014 decidieron dejar de aparecer en los *benchmarks* de Hans Mittelmann, por lo que no hay información actualizada. Entre los dos, como no ofrecen diferencias evidentes, se eligió Gurobi ya que es muy sencillo conseguir una licencia académica gratuita. Tomando como ejemplo el visto en **3.3.1 Caso de estudio**, veremos los detalles de la implementación.

4.2.1. Datos

Una vez el modelo está definido, la implementación es bastante directa. Veremos por encima una posible implementación en Python. Definiremos primero los datos de entrada: conjuntos y parámetros siguiendo el caso de estudio (pág. 15).

Conjuntos

Los conjuntos T, R y CR serán listas ordenadas de enteros de 0 a N_L para cada conjunto L. Además, como vimos en los detalles del modelo, el conjunto de eventos E lo derivamos del número de tareas y el número cambio de recursos.

```
N_T, N_R, N_CR = (10, 5, 21)
T  = range(N_T)          # Tareas
R  = range(N_R)          # Recursos
CR = range(N_CR)         # Eventos de cambio de recurso
E  = range(N_T + N_CR)   # Eventos
```

Los sub-conjuntos T_d, T_n, R_r y R_n los definiremos como listas de elementos de los conjuntos de los cuales derivan.

```
T_D = [2, 5, 9]              # Tareas divisibles
T_N = [0, 1, 3, 4, 6, 7, 8] # Tareas no divisibles
R_R = [0, 1, 2, 3]           # Recursos renovables
R_N = [4]                    # Recursos no renovables
```

El conjunto de precedencias P lo definimos como una lista de tuplas (i, j) donde la tarea i precede a la tarea j:

```
P = [                   # Precedencias
        (0, 1), (0, 2), (1, 3), (2, 3), ## Proyecto 1
        (4, 6), (5, 6),                 ## Proyecto 2
        (7, 8)                          ## Proyecto 3
]
```

Parámteros

Los parámetros son datos de entrada del problema que van indexados en función de los conjuntos. Para distinguirlos con facilidad de las variables, le añadiremos al nombre de los parámetros el prefijo `p_`.

$modo(t)$ es una lista indexada por el número de tarea donde cada elemento corresponde al número de modos de cada tarea

```
p_modo = [
    2, 1, 2, 2, # Tareas 0,1,2,3    (Proyecto 1)
    1, 1, 1,    # Tareas 4,5,6      (Proyecto 2)
    2, 3, 2     # Tareas 7,8,9      (Proyecto 3)
]
```

$drcn(t, m)$ es una lista de listas. Cada elemento de `p_modo` corresponde con una tarea y cada elemento de cada tarea corresponde a la duración de la tarea ejecutada en ese modo. En este caso, como vimos en el caso de estudio, no varia entre modos.

```
p_modo = [
    [3, 3],             # Duracion de los modos de la tarea 0
    [4.6],              # Duracion de los modos de la tarea 1
    [5, 5],             # Duracion de los modos de la tarea 2
    [1.2, 1.2],         # Duracion de los modos de la tarea 3
    [3.4],              # Duracion de los modos de la tarea 4
    [12.4],             # Duracion de los modos de la tarea 5
    [6.7],              # Duracion de los modos de la tarea 6
    [1.3, 1.3],         # Duracion de los modos de la tarea 7
    [2.5, 2.5, 2.5],    # Duracion de los modos de la tarea 8
    [13.2, 13.2]        # Duracion de los modos de la tarea 9
]
```

$drcn(t, m)$ es una lista de listas de listas. Cada elemento de `p_modo` corresponde con una tarea. Cada elemento de cada tarea corresponde a los modos de la tarea. Cada elemento de cada modo corresponde a las demandas de cada recurso por parte de la tarea t en modo m. Siguiendo el caso de estudio tenemos:

```
p_dmnd = [
    #    A, B, C, M, N
    [# Tarea 0
        (1, 0, 0, 0, 0),
        (0, 1, 0, 0, 0)
    ],
    [# Tarea 1
        (0, 0, 1, 0, 0)
    ],
    [# Tarea 2
        (1, 0, 0, 1, 5),
        (0, 0, 1, 1, 5)
    ],
    [# Tarea 3
        (0, 1, 0, 0, 0),
        (0, 0, 1, 0, 0)
    ],
    [# Tarea 4
        (0, 1, 0, 1, 7)
    ],
    [# Tarea 5
        (0, 0, 1, 0, 0)
    ],
    [# Tarea 6
        (1, 0, 0, 0, 0)
    ],
    [# Tarea 7
        (1, 0, 0, 0, 0),
        (0, 0, 1, 0, 0)
    ],
    [# Tarea 8
        (1, 0, 0, 1, 4),
        (0, 1, 0, 1, 4),
        (0, 0, 1, 1, 4)
    ],
    [# Tarea 9
        (1, 0, 0, 0, 0),
        (0, 1, 0, 0, 0)
    ]
]
```

$tmpo(cr)$ es una lista donde cada elemento corresponde con el tiempo en el que ocurre el evento de cambio de recursos cr.

```
p_tmpo = [
    0, 8, 12, 16, 20, 24,
    32, 36, 40, 44, 48,
    56, 60, 64, 68, 72,
    80, 84, 88, 92, 96
]
```

$disp(cr, k_r)$ es una lista de diccionarios. Cada elemento de la lista corresponde con un diccionario de disponibilidad de los recursos renovables dentro de la franja del evento de cambio de recurso cr. Por ejemplo, `{0: 13, 2: 4}` significa que para un cr dado, el recurso 0 tendría disponibilidad de 13 y el recuso 2 una disponibilidad de 4. Así mismo, según el caso de estudio tenemos:

```
p_disp = [
    {0: 0, 1: 0, 2: 0, 3: 1}, # cr 0
    {0: 2, 1: 0, 2: 1, 3: 1}, # cr 1
    {0: 3, 1: 1, 2: 2, 3: 1}, # cr 2
    {0: 1, 1: 1, 2: 1, 3: 1}, # cr 3
    {0: 0, 1: 0, 2: 0, 3: 1}, # cr 4
    {0: 0, 1: 0, 2: 0, 3: 0}, # cr 5
    {0: 2, 1: 0, 2: 1, 3: 0}, # cr 6
    {0: 3, 1: 1, 2: 2, 3: 0}, # cr 7
    {0: 1, 1: 1, 2: 1, 3: 0}, # cr 8
    {0: 0, 1: 0, 2: 0, 3: 0}, # cr 9
    {0: 0, 1: 0, 2: 0, 3: 1}, # cr 10
    {0: 2, 1: 0, 2: 1, 3: 1}, # cr 11
    {0: 3, 1: 1, 2: 2, 3: 1}, # cr 12
    {0: 1, 1: 1, 2: 1, 3: 1}, # cr 13
    {0: 0, 1: 0, 2: 0, 3: 1}, # cr 14
    {0: 0, 1: 0, 2: 0, 3: 0}, # cr 15
    {0: 2, 1: 0, 2: 1, 3: 0}, # cr 16
    {0: 3, 1: 1, 2: 2, 3: 0}, # cr 17
    {0: 1, 1: 1, 2: 1, 3: 0}, # cr 18
    {0: 0, 1: 0, 2: 0, 3: 0}, # cr 19
    {0: 0, 1: 0, 2: 0, 3: 1}  # cr 20
]
```

$gnrd(t, m, k_n)$ es una lista de listas de diccionarios. Cada elemento de `p_gndr` corresponde con una tarea. Cada elemento de cada tarea corresponde a los modos de la tarea. Cada elemento de cada modo corresponde a la cantidad del recurso k_n generado al finalizar la tarea t en modo m. Al igual que la duración de los distintos modos de una tarea, la generación de recursos no renovables tampoco varia entre modos.

```
p_gnrd = [
    [{4: 0}, {4: 0}],           # Tarea 0
    [{4: 0}],                   # Tarea 1
    [{4: 0}, {4: 0}],           # Tarea 2
    [{4: 5}, {4: 5}],           # Tarea 3
    [{4: 0}],                   # Tarea 4
    [{4: 5}],                   # Tarea 5
    [{4: 0}],                   # Tarea 6
    [{4: 0}, {4: 0}],           # Tarea 7
    [{4: 0}, {4: 0}, {4: 0}],   # Tarea 8
    [{4: 0}, {4: 0}]            # Tarea 9
]
```

$inic(k_n)$ es un diccionario donde para cada recurso no renovable k_n se define la cantidad inicial disponible. En el caso de estudio al solo tener un recurso no renovable tenemos:

```
p_inic = {4: 10}
```

4.2.2. Variables

De manera similar que con los parámetro, prefijaremos las variables con `v_` para diferenciarlas con facilidad de los parámetros. Previo a la definición de las variables necesitaremos una instancia de modelo Gurobi.

```
import gurobipy

model = gurobipy.Model()
```

En la biblioteca `gurobipy` la variables del modelo se declaran como `model.addVar()`. Puede recibir como parámetros opcionales:

- `lb`, o límite inferior. El mínimo valor que puede tomar la variable.
- `ub`, o límite superior. El máximo valor que puede tomar la variable.
- `vtype`, o tipo de variable. Usaremos variables contínuas (`vtype = GRB.CONTINUOUS`) y binarias (`vtype = GRB.BINARY`). Si no se indica nada, por defecto el tipo de las variables es contínuo.

La traducción del modelo a la implementación es bastante directa. Los conjuntos de variables serán diccionarios indexados por tuplas con los índices de sus elementos.

```
v_t = {
        e: model.addVar(lb=0)
        for e in E
}
```

Definimos el tiempo asociado al evento e para cada de elemento de E. Al tener un valor mínimo de 0, en la declaración del conjunto de variables pasamos por parámetro `lb=0`.

```
v_z = {
        (t, m, e): model.addVar(vtype=GRB.BINARY)
        for t in T for e in E for m in p_modo[t]
}
```

Definimos la variable decisión $z(t, m, e)$ para cada tarea, modo y evento. Observamos que la definimos como variable binaria pasándole el parámetro `vtype=GRB.BINARY`.

```
v_y = {
        (cr, e): model.addVar(vtype=GRB.BINARY)
        for cr in CR for e in E
}
```

De igual manera la variable de asociación de eventos a CRs la declaramos pasándole el parámetro `vtype=GRB.BINARY`.

```
v_d = {
        (k_n, e): model.addVar(lb=0)
        for k_n in R_n for e in E
}
```

La variable $d(k_n, e)$ mide la disponibilidad del recursos no renovables k_n en el evento e, por lo que será contínua con un límite inferior de 0, en referencia a la restricción (3.28).

```
v_x = {
        (t, m): model.addVar(vtype=GRB.BINARY)
        for t in T for m in p_modo[t]
}
```

La variable $x(t, m)$ es una variable binaria que representa si la tarea t está programada a realizarse en modo m. Por lo tanto le pasamos el parámetro `vtype=GRB.BINARY`.

```
v_C = model.addVar()
```

La variable C delimita el *makespan* o la duración total de la planificación.

```
v_zneg = {
        (t,m,e): model.addVar(vtype=GRB.BINARY)
        for t in T for m in p_modo[t] for e in E
}
```

La variable $\hat{z}(t, m, e)$ es una variable que definimos por conveniencia para expresar la negación de $z(t, m, e)$ por lo que también será de tipo binaria y recibirá como parámetro `vtype=GRB.BINARY`.

```
v_start = {
        (t,m,e): model.addVar(vtype=GRB.BINARY)
        for t in T for m in p_modo[t] for e in E
}
```

Definimos la variable $start(t, m, e)$, que representa si el evento e es inicio de la tarea t en modo m, como una variable del tipo `vtype=GRB.BINARY`.

```
v_end = {
        (t,m,e): model.addVar(vtype=GRB.BINARY)
        for t in T for m in p_modo[t] for e in E
}
```

Por último, definimos la variable $end(t, m, e)$, que representa si el evento e es fin de la tarea t en modo m, como una variable del tipo `vtype=GRB.BINARY`.

4.2.3. Restricciones

La manera de añadir restricciones al modelo de Gurobi es bastante directa, prácticamente un traducción literal. Mediante la función `model.addConstr()` le podemos pasar como parámetro una expresión lineal para añadir como restricción. Es bastante intuitivo y la lectura es prácticamente literal.

Para traducir a el modelo la biblioteca `gurobipy` utilizaremos una serie de transformaciones. Donde en el modelo teníamos una restricción del tipo:

$$\ldots \qquad\qquad \forall\, e \in E, t \in T, \ldots$$

lo traduciremos a un bucle de la forma:

```
for e in E:
 for t in T:
  #for ... :
   model.addConstr(...)
```

Es decir cada conjunto que aplique a la restricción se traducirá a un bucle anidado que recorra los conjuntos.

Las expresiones del tipo

$$\sum_{t \in T} \sum_{m \in modos(t)} x(t, m)$$

se traducirán utilizando la función `sum()` y el mecanismo de listas de comprensión de python de la siguiente forma:

```
sum(x[t,m] for e in E for m in p_modos[t])
```

El operador `>>` representa la implicación lógica ($\Longrightarrow$) de la forma `(exresion lineal 1) >> (expresion lineal 2)` que añadirá a las restricciones del modelo la expresión lineal 2 si se cumple la expresión lineal 1.

La restricción **Inicio temporal** (pág. 21), tiene una traducción literal, la variable temporal asociada al evento 0 toma el valor 0.

```
model.addConstr(v_t[0] == 0)
```

La restricción **Ordenación** (pág. 21) la implementamos para cada evento a partir del primero (expresado en python `E[1:]`). Además, traducimos $\forall\, e \in E$ como un bucle `for`. Obligamos a los eventos a tener un valor temporal secuencial.

```
# ORDENACIÓN
for e in E[1:]:
    model.addConstr(v_t[e] >= v_t[e-1])
```

La restricción de **Programación** (pág. 21) obliga a programar todas las tareas al menos una vez, es decir, son de obligado cumplimiento. Vemos de nuevo los mecanismos de traducción de conjuntos. En esta restricción traducimos la suma $\sum_{m \in modo(t)} \sum_{e \in E} z(t,m,e)$ utilizando la función `sum` y el mecanismo de listas de comprensión para recorrer ambos conjuntos.

```
# PROGRAMACIÓN
for t in T:
    model.addConstr(
     sum(v_z[(t,m,e)] for m in p_modo[t] for e in E) >= 1
    )
```

La restricción de **Precedencia** (pág. 21) obliga a que la tarea i haya finalizado previo al inicio de la tarea j. La traducimos utilizando la transformaciones ya vistas con un nuevo elemento: traducimos N_E (número de elementos del conjunto E) en python como `len(E)`.

```
# PRECEDENCIA
for e in E:
 for (i,j) in P:
  for m_i in p_modo[i]:
   for m_j in p_modo[j]:
    model.addConstr(
        sum(v_z[(i,m_i,e2)] for e2 in E[e:]) <=
        len(E) * (1 - v_z[(j,m_j,e)])
    )
```

La restricción de **Makespan** (pág. 22) consta de dos sub-restricciones como vimos en la formulación del modelo.

La primera corresponde a la implicación lógica que obliga a la variable C a tomar el valor del final del último evento programado. Vemos como la traducción es literal con el operador `>>` representando la implicación lógica ($\Longrightarrow$).

La segunda sub-restricción define al último evento como un evento *dummy* en el que ninguna tarea puede ser programada.

```
# MAKESPAN
for e in E[:-1]:
 for t in T:
  for m in p_modo[t]:
   model.addConstr((v_z[(t,m,e)] == 1) >> (v_C >= v_t[e+1]))
```

```
## Evento dummy final
for t in T:
 for m in p_modo[t]:
  model.addConstr(v_z[(t,m,E[-1])] == 0)
```

La restricción de **Modos** (pág. 22) está formada de manera muy similar al *Makespan* por dos sub-restricciones.

```
# MODOS
## Tareas en un único modo
for t in T:
 model.addConstr(sum(v_x[t,m] for m in p_modo[t]) == 1)

## Exclusividad entre modos
for t in T:
 for m in p_modo[t]:
  model.addConstr(
   (v_x[(t,m)] == 1) >>
   (sum(v_z[(t,m2,e)] for e in E for m2 in p_modo[t] if m2!=m) == 0)
  )
```

La restricción de **Duración - No divisibles** (pág. 23) asegura que las tareas no divisibles se programan tanto tiempo como manda su duración. Vemos como el primer bucle anidado corresponde con la restricción general y el segundo con la restricción especial para el evento 0.

```
# DURACION - NO DIVISIBLES
## Caso general
for t_n in T_n:
 for m in p_modo[t]:
  for i in E[1:]:
   for f in E[i+1:]:
    model.addConstr(
     v_t[f] >= v_t[i]
     + p_drcn[t_n][m] * (v_z[(t_n,m,i)] + v_z[(t_n,m,f-1)]
     - v_z[(t_n,m,f)] - v_z[(t_n,m,i-1)] - 1)
    )

## Caso especial del evento 0
for t_n in T_n:
 for m in p_modo[t]:
  for f in E[1:]:
   model.addConstr(
    v_t[f] >= v_t[i]
```

```
    + p_drcn[t_n][m] * (v_z[(t_n,m,0)] + v_z[(t_n,m,f-1)]
    - v_z[(t_n,m,f)] - 1)
   )
```

La restricción de **Duración - Divisibles** (pág. 23) asegura que las tareas divisibles se programa tanto tiempo como viene marcado por su duración. Tiene una traducción directa mediante los mecanismos vistos anteriormente.

```
# DURACION - DIVISIBLES
for t_d in T_d:
 for m in p_modo[t]:
  model.addConstr(
   v_x[(t_d,m)] * p_drcn[t_d][m] <=
   sum(v_z[(t_d,m,e)] * (v_t[e+1] - v_t[e]) for e in E[:-1])
  )
```

La restricción de **Continuidad** (pág. 24) asegura que las tareas no divisibles se ejecutan como su nombre indica, sin parar hasta finalizar. El primer bucle anidad corresponde con la continuidad desde el inicio temporal. El segundo bucle corresponde con la continuidad desde el fin temporal.

```
# CONTINUIDAD
## Continuidad - inicio
for e in E[1:]:
 for t in T_n:
  for m in p_modo[t]:
   model.addConstr(
    sum(v_z[(t,m,f)] for f in E[:e]) <=
    len(E) * (1 + v_z[(t,m,e-1)] - v_z[(t,m,e)])
   )

## Continuidad - fin
for e in E[1:]:
 for t in T_n:
  for m in p_modo[t]:
   model.addConstr(
    sum(v_z[(t,m,f)] for f in E[e:]) <=
    len(E) * (1 + v_z[(t,m,e)] - v_z[(t,m,e-1)])
   )
```

La restricción de **Recursos renovables** (pág. 25) asegura que la demanda de recursos en un momento dado no sobrepasa la disponibilidad en cada momento. Los primeros 3 bucles corresponden con la traducción de las restricciones de asociación de rangos. Tras estos, encontramos

las 2 asunciones que hacemos para facilitar la resolución del modelo. El último bucle con la restricción en si, limita que la demanda no sea superior a la disponibilidad.

```
# RECURSOS RENOVABLES
## ASOCIACION DE RANGOS
for e in E:
 for cr in CR:
  model.addConstr(
   (v_y[(cr,e)] == 1) >> (v_t[e] >= p_tmpo[cr])
  )
  model.addConstr(
   (v_y[(cr,e)] == 1) >> (v_t[e+1] <= p_tmpo[cr+1])
  )
for e in E:
 model.addConstr(sum(v_y[(cr,e)] for e in E) >= 1)
for cr in CR:
 model.addConstr(sum(v_y[(cr,e)] for cr in CR) == 1)

## ASUNCIONES
model.addConstr(v_y[0,0] == 1);
model.addConstr(v_y[CR[-1],E[-1]] == 1);

## DISPONIBILIDAD
for e in E:
 for cr in CR:
  for k in R_r:
   model.addConstr(
    (v_y[(cr,e)] == 1) >>
    (sum(v_z[(t,m,e)]*p_dmnd[t][m][k] for t in T for m in p_modo[t])
    <= p_disp[cr][k])
   )
```

La restricción de **Recursos no renovables** (pág. 26), controla el uso de recursos no renovables en función de la disponibilidad en cada momento. Para ello tenemos una serie de sub-restricciones que la componen. Primero, definimos la demanda inicial, seguido de las definiciones del comportamiento de las variables $\hat{z}(t,m,e)$, $start(t,m,e)$ y $end(t,m,e)$. Por último requerimos un inicio y final por cada tarea y aplicamos la restricción que limita la utilización de recursos no renovable. Destacamos como no incluimos la restricción de no negatividad ya que viene implícita en la declaración de la variable `v_d`

```
# RECURSOS NO RENOVABLES
## Disponibilidad inicial
for k in R_n:
 model.addConstr(
  v_d[k,0] == p_inic[k]
  - sum(p_dmnd[t][m][k] * v_z[t,m,0] for t in T for m in p_modo[t])
 )
```

```
# Definición de zneg
for t in T:
 for m in p_modo[t]:
  for e in E:
   model.addConstr(v_zneg[t,m,e] == 1 - v_z[t,m,e])

# Definición de start
for t in T:
 for m in p_modo[t]:
  for e in E:
   model.addConstr(
    (v_start[t,m,e] == 1) >>
    (v_zneg[t,m,e] + sum(v_z[t,m,e2] for e2 in E[:e]) == 0)
   )

# Definición de end
for t in T:
 for m in p_modo[t]:
  for e in E[1:]:
   model.addConstr(
    (v_end[t,m,e] == 1) >>
    (v_zneg[t,m,e-1] + sum(v_z[t,m,e2] for e2 in E[e:]) == 0)
   )
for t in T:
 for m in p_modo[t]:
  model.addConstr(v_end[t,m,0] == 0)

# Requerir un inicio y un final
for t in T:
 model.addConstr(
  sum(v_start[t,m,e] for e in E for m in p_modo[t]) == 1
 )
 model.addConstr(
  sum(v_end[t,m,e] for e in E for m in p_modo[t]) == 1
 )

# Restricción de disponibilidad
for e in E[1:]:
 for k in R_n:
  model.addConstr(
   v_d[k,e] == v_d[k,e-1]
   -sum(p_dmnd[t][m][k]*v_start[t,m,e] for t in T for m in p_modo[t])
   +sum(p_gnrd[t][m][k]*v_end[t,m,e] for t in T for m in p_modo[t])
  )
```

4.2.4. Pre-procesado

Previa a la resolución del caso de estudio, sometemos los datos a las 3 condiciones.

Grafo acíclico

Primero, comprobamos que no hay dependencias cíclicas entre tareas. Para ello implementamos el algoritmo del orden topológico (pág. 30). Si no se supera la comprobación, el caso de estudio no tendría solución.

```
nodes = set(T)
edges = set(P)
L = []
Q = set(n for n in nodes if n not in (e[1] for e in edges))
nodes = nodes.difference(Q)
while len(Q):
 n = Q.pop()
 L.append(n)
 for m in nodes:
  if (n, m) in edges:
   edges.remove((n, m))
   if not len([e for e in edges if e[1] == m]):
     Q.add(m)
if len(edges):
 print("Cyclic Graph \t[success]")
else:
 print("Acyclic Graph \t[error]")
```

Disponibilidad de recursos no renovables

A continuación, comprobamos que la disponibilidad inicial más la cantidad generada por las tareas es mayor que la demanda total (pág. 31). Si no se supera la comprobación, el caso de estudio no tendría solución.

```
for k in R_n:
 ini = p_inic[k]
 generado = sum(max(p_gnrd[t][m][k] for m in p_modo[t]) for t in T)
 demanda = sum(max(p_dmnd[t][m][k] for m in p_modo[t]) for t in T)
 result = "success" if ini + generado >= demanda else "error"
 print(f"{k}: {ini} + {generado} >= {demanda} \t[{result}]")
```

Disponibilidad de recursos renovables

Por último, sometemos el caso de estudio a la comprobación de recursos renovables. Queremos que la disponibilidad total de recursos sea mayor o igual a la demanda máxima de las

tareas (pág. 32). Si no se supera la comprobación, será necesario aumentar el horizonte de planificación hasta que se cumpla. Si se supera con mucha disponibilidad sobrante, se procurará reducir el horizonte de planificación reduciendo así el número de CRs.

```
for k in R_r:
 disponibilidad = sum(
                     p_disp[cr][k] * (p_tmpo[cr+1]-p_tmpo[cr]
                    ) for cr in CR[:-1])

 demanda = sum(
                max(p_dmnd[t][m][k] * p_drcn[t][m] for m in p_modo[t]
               ) for t in T)

 result = "success" if disponibilidad >= demanda else "error"
 print(f"{k}: {disponibilidad: .2f} >= {demanda: .2f} \t[{result}]")
```

Ejecutando las comprobaciones sobre el caso de estudio obtenemos lo siguientes resultados:

```
Pre-procesado de datos

Orden topológico:

    Acyclic graph                [success]

Disponibilidad de recursos no renovables:

    7: 10 + 20 >= 26             [success]

Disponibilidad de recursos renovables:

    0:  32.00 >=  31.30          [success]
    1:  32.00 >=  31.30          [success]
    2:  32.00 >=  29.90          [success]
    3:  32.00 >=  28.30          [success]
    4:  32.00 >=  29.00          [success]
    5:  32.00 >=  31.40          [success]
    6:  48.00 >=  16.20          [success]
```

Vemos que supera las dos primeras comprobaciones y la disponibilidad de recursos renovables está muy ajustada y no se puede reducir más el horizonte de planificación.

4.2.5. Resolución

Una vez hemos implementado el modelo, introducido los datos del caso de estudio y realizado el pre-procesado de los datos, procedemos a la resolución. Dividiremos la resolución en dos pasos.

Primero, optimizamos el modelo sin función objetivo. Priorizaremos encontrar una solución

factible cualquiera con el objetivo de mejorarla una vez encontrada. Ejecutando la función `model.optimize()` obtenemos los siguientes resutlados:

Resolución de modelo I

```
Explored 2586 nodes (53363 simplex iterations) in 30.89 seconds
Thread count was 8 (of 8 available processors)

Solution count 1: 0

Optimal solution found (tolerance 1.00e-04)
Best objective 0.000000000000e+00
```

Tras 30.89 segundos, el resolutor encuentra una primera solución factible. A continuación introducimos la función objetivo, minimizar C. Procedemos a mejorar la solución encontrada. Limitamos de manera indicativa el tiempo de resolución de la segunda ejecución al tiempo de resolución previo.

```
model.setObjective(v_C, GRB.MINIMIZE)
model.params.TimeLimit = model.Runtime
status = model.optimize()
```

Tras la ejecución obtenemos los siguientes resultados:

Resolución de modelo II

```
Changed value of parameter TimeLimit to 30.917278055
Loaded MIP start from previous solve with objective 92

Explored 773 nodes (53927 simplex iterations) in 30.93 seconds
Thread count was 8 (of 8 available processors)

Solution count 4: 86.7 88 92 92

Time limit reached
Best objective 8.670000000000e+01
```

La ejecución previa obtuvo el resultado $C = 92$. Partiendo de ella, encuentra otras 3 soluciones en 30.92 segundos y obtiene un mejor resultado $C = 86.7$, recortando el resutado previo por 5.3 horas.

4.3. Verificación

Una vez tenemos una solución, vamos a verificar que la solución obtenida es correcta analizando el valor de las variables obtenidas. No podemos estudiar las 2930 variable de manera individual así que las representaremos de manera gráfica:

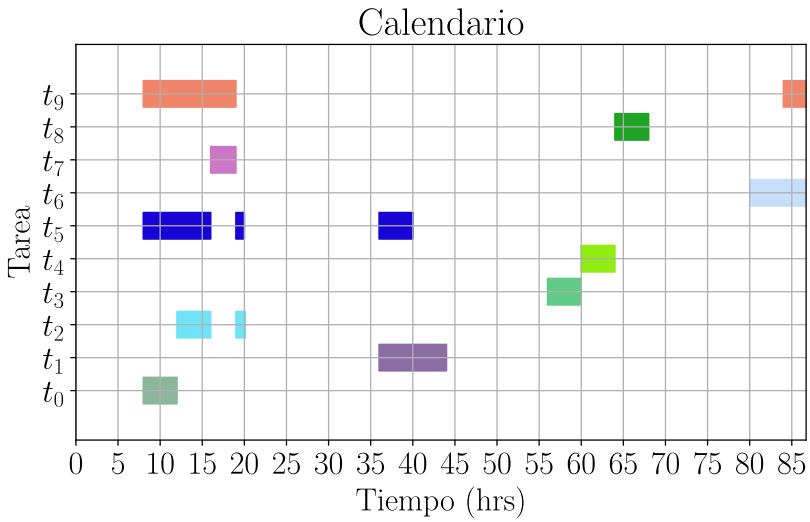


Figura 4.1: Resultados - Calendario

La figura 4.1 es el calendario o solución que representa visualmente los resultados obtenidos. Para cada tarea en el eje vertical tenemos una o más (si es una tarea divisible) barras horizontales que representan el inicio y el final del procesamiento de las tareas.

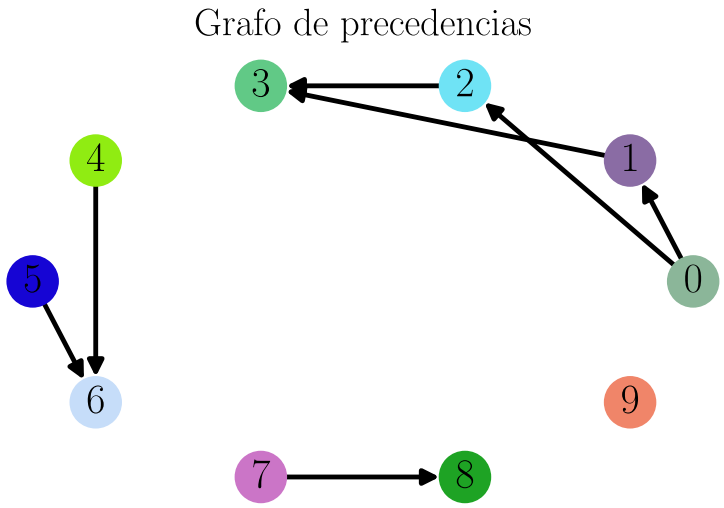


Figura 4.2: Resultados - Grafo de precedencias

Utilizando la información que nos proporciona el grafo de precedencias, verificamos se respetan todas las relaciones de precedencia. Recordamos como las tareas t_2, t_5 y t_9 son las tareas divisibles. Todas ellas aprovechan la propiedad de divisibilidad.

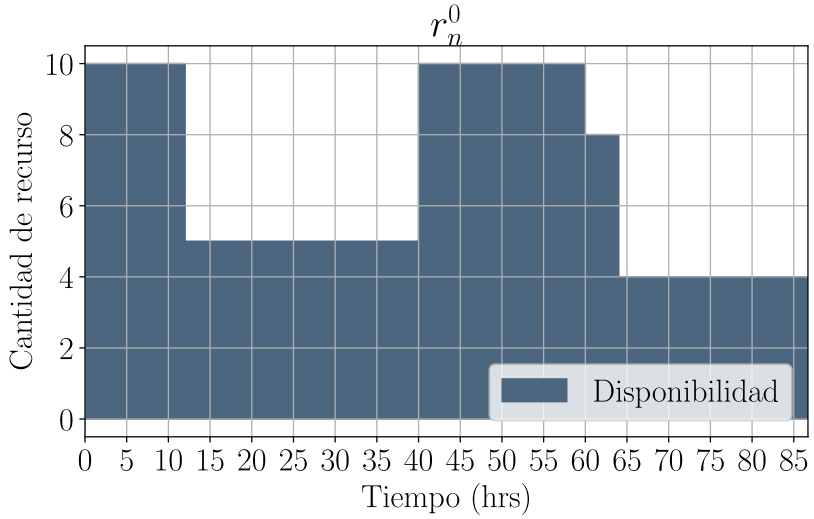


Figura 4.3: Resultados - Recurso no renovables

Las tareas t_2, t_4 y t_8 usan respectivamente $5, 7$ y 4 unidades del recurso no renovable r_n^0. Las tareas t_3 y t_5 generan respectivamente 5 y 5 unidades del recurso no renovable r_n^0. Dada la ordenación de las tareas, vemos en la figura 4.3 como la disponibilidad de recurso evoluciona de la siguiente manera:

- Empezamos con 10 unidades.
- En $t = 12\ hrs$, comienza la tarea t_2 consumiendo 5 unidades del recurso lo que nos deja con 5 unidades.
- En $t = 40\ hrs$ termina la tarea t_5 generando 5 unidades.
- En $t = 60\ hrs$, termina la tarea t_3 generando 5 unidades y comienza la tarea t_4 consumiendo 7 unidades lo que nos deja con 8 unidades.
- Por último, en $t = 64\ hrs$, comienza la tarea t_8 consumiendo 4 unidades lo que nos deja con 4 unidades al final de la planificación.

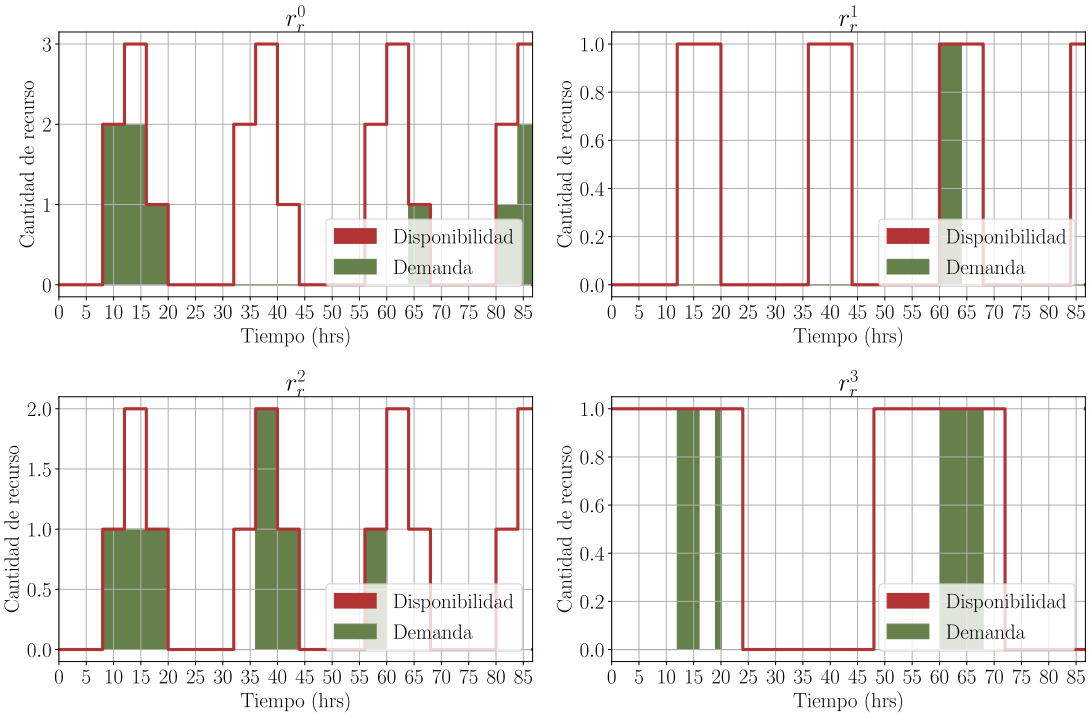


Figura 4.4: Resultados - Recurso renovables

Como última comprobación, en la figura 4.4 vemos como en todo momento la demanda de los recursos renovables se mantiene inferior o igual a la disponibilidad de los recursos. De esta manera hemos conseguido organizar nuestro proyecto de manera automática. Estamos seguros que las respuestas serán correctas aunque, dada la naturaleza del problema, no podemos ser capaces de *asegurar* que es la óptima. Esto es cierto especialmente a medida que van creciendo el número de recursos y tareas.

4.4. Estudio paramétrico

4.4.1. Introducción

Vamos a estudiar cómo afecta el aumento de la complejidad del problema. Estudiaremos diferentes aspectos que puedan afectar a la velocidad de resolución del problema. Recordamos que este problema es NP-Duro, lo que quiere decir que no existe ningún algoritmo capaz de resolverlo en tiempo polinómico. Es inevitable que, a medida que aumenta la complejidad el tiempo de resolución explotará. Por eso, lo mejor que podemos hacer es buscar maneras cada vez más eficientes de plantear el problema, aumentando el umbral a partir del cual deja de ser posible resolver el problema o encontrar soluciones adecuadas en un tiempo razonable.

Vamos a medir la complejidad de un problema en función de cada parámetro:

- Número de tareas
- Número de recursos
- Número de eventos de cambio de recurso (CR)
- % de tareas divisibles
- % de tareas no renovables

Para cada parámetro, diseñaremos 5 ó 6 casos de prueba, procurando variar exclusivamente el parámetro que queremos medir sin modificar el resto. Mediremos como varían el número de variables, restricciones y la velocidad de resolución. Deshabilitaremos todas las optimizaciones y el pre-procesado que realiza de manera automática el resolutor y mediremos cuantos nodos calcula por segundo.

La resolución de un problema se realiza de manera determinista, por lo que repetiremos cada caso de prueba 10 veces cambiando la semilla y sacaremos conclusiones a partir de los resultados.

4.4.2. Resultados

Número de tareas

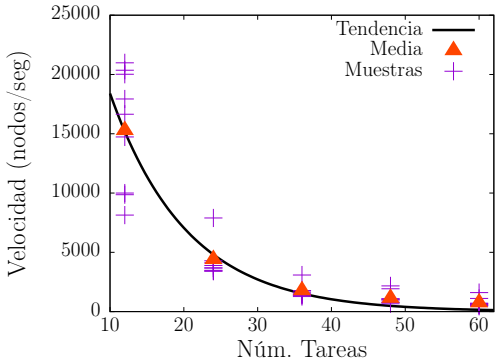


Figura 4.5: Velocidad (nodos/seg) en función de número de tareas

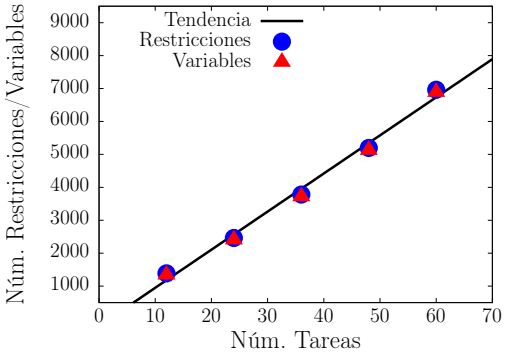


Figura 4.6: Número de restricciones y variables en función del número de tareas

El número de tareas es uno de los factores que más impacto tiene. Es razonable teniendo, en cuenta que el objetivo principal es ordenarlas. Cuantas más tareas tenemos, más difícil resulta su ordenación.

Es importante tener en cuenta que cuando hablamos de número de tareas, nos referimos al número total de tareas y modo. De manera más precisa a la suma de tareas por el número de modos que tiene ($\sum_{t \in T} modos(t)$). Por ejemplo, el caso de estudio tiene 7 tareas. Sin embargo, si las multiplicamos por el número de modos que hay nos sale un total de 35 tareas diferentes.

Para el caso de estudio, una manera de hacer la resolución más fácil es reducir el número de recursos, tratando a todos los trabajadores de un mismo departamento como el mismo recurso. De esta manera reduciríamos el número de modos. Pasaría de haber 35 tareas a 17. Como indican los resultados, potencialmente triplicaría la velocidad de resolución. Esto nos deja ver lo importante de un buen planteamiento del problema en todo momento.

Donde antes tardaba 160 segundos, este segundo planteamiento tarda 35 segundos en dar una respuesta equivalente.

Porcentaje de tareas divisibles

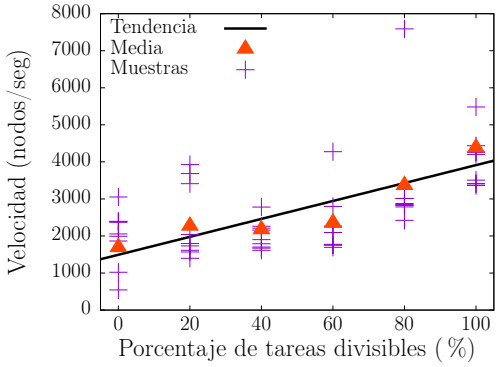


Figura 4.7: Velocidad (nodos/seg) en función del % de tareas divisibles

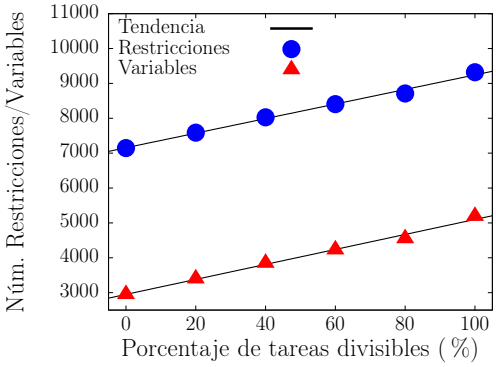


Figura 4.8: Número de restricciones y variables en función del % de tareas divisibles

Como muestran las gráficas, es más fácil trabajar con tareas divisibles. Esto tiene un sentido intuitivo ya que es más fácil para el resolutor respetar las restricciones de recursos si las tareas se pueden dividir en fracciones arbitrariamente pequeñas. Aunque el número de variable y restricciones crezca de manera lineal, al igual que con el número de tareas, no tiene un impacto exponencial en la velocidad de resolución. No presenta una diferencia significativa en términos de complejidad.

Número de recursos

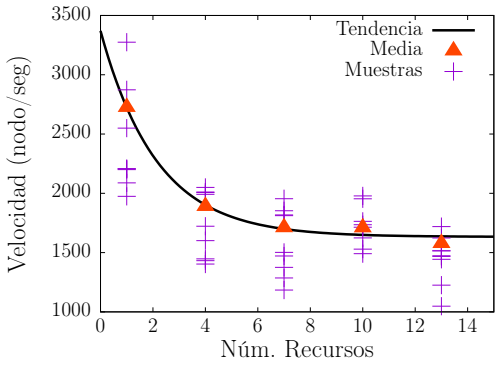


Figura 4.9: Velocidad (nodos/seg) en función del número de recursos

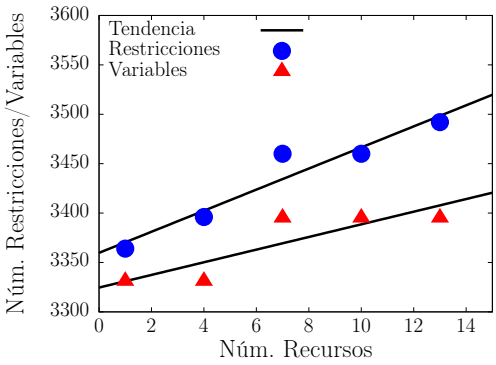


Figura 4.10: Número de restricciones y variables en función del número de recursos

Al igual que con el número de tareas, también tenemos un comportamiento negativo en función del número de recursos. El número de restricciones y variables aumenta linealmente y la velocidad claramente empeora con el aumento del número de recursos. Sin embargo, parece tener un impacto reducido sobre la velocidad de resolución.

La velocidad de resolución no cae en picado sino que parece tener un límite. Sin duda para cada problema el límite tomará otros valores. Sin embargo, lo que sí está claro es que cabe reducir el número de recursos lo máximo posible, como en el ejemplo visto en el apartado *Número de tareas*. Esto es porque indirectamente se reduce el número de tareas al simplificar el número de modos por tarea.

Porcentaje de recursos no renovables

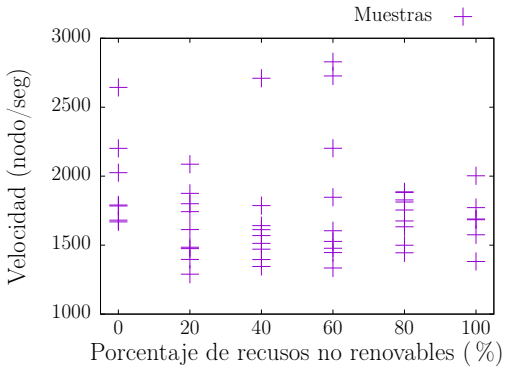


Figura 4.11: Velocidad (nodos/seg) en función del % de recursos no renovables

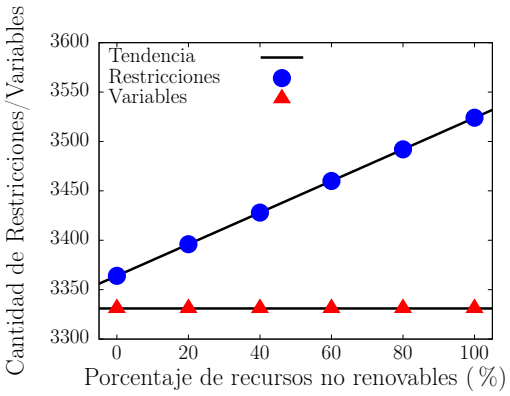


Figura 4.12: Número de restricciones y variables en función del % de recursos no renovables

Al contrario que con el porcentaje de tareas divisibles, no podemos determinar un comportamiento definido en función del número de recursos no renovables. El número de variables es constante indiferentemente del ratio de recursos renovables y no renovables. Aunque crezca el número de restricciones, no empeora ni mejora claramente la velocidad de resolución de las tareas.

Número de eventos de cambio de recurso (CR)

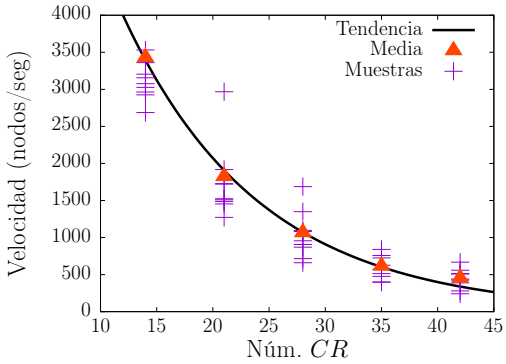


Figura 4.13: Velocidad (nodos/seg) en función del número de eventos de cambio de recurso (CR)

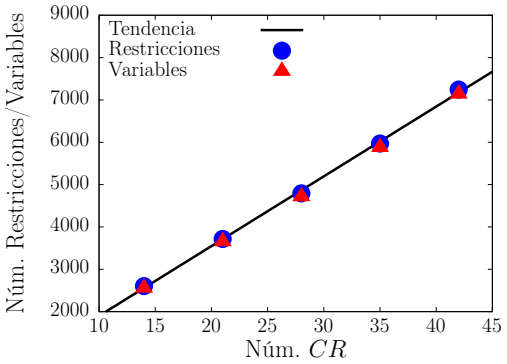


Figura 4.14: Número de restricciones y variables en función del número de eventos de cambio de recurso (CR)

El número de CRs es otro de los factores que más afecta a la velocidad de resolución. Claramente existe un factor de empeoramiento exponencial. Como vimos en el apartado *3.4.4 Restricciones*, la restricción de duración de recursos no renovables aumenta el número de eventos por cada *cr* que se añade. Cada *cr* nos genera una nueva franja de disponibilidad constante. De esta manera, tenemos un aumento exponencial del número posibles de ordenaciónes de los eventos dentro de las franjas de disponibilidad constante.

A medida que aumenta linealmente el número de restricciones y variables, la velocidad de resolución disminuye de manera exponencial. Así vemos como es de máxima importancia ceñir el horizonte de planificación H al máximo posible, con el fin de generar el mínimo número de *CRs* posible.

Tiempo de resolución

El número de tareas y el número de CRs es lo que más impacto tiene sobre la velocidad de resolución, vamos a evaluar su impacto sobre el tiempo de resolución.

Como definimos en el apartado **1.4 Objetivos**, definimos que el modelo «debe mantener el compromiso con la computabilidad de la respuesta en un tiempo razonable». Teniendo en cuenta la aplicación del modelo, un tiempo razonable estaría en el rango de magnitud de *minutos*, 10 minutos como ideal.

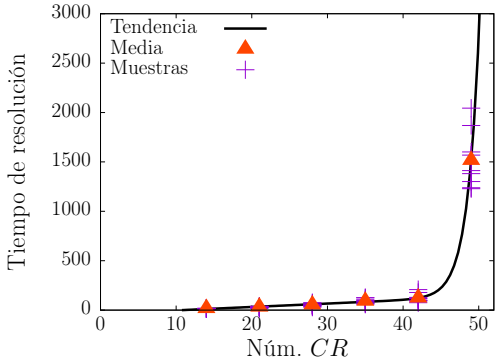


Figura 4.15: Tiempo de resolución del modelo en función del número de eventos de cambio de recursos (CRs)

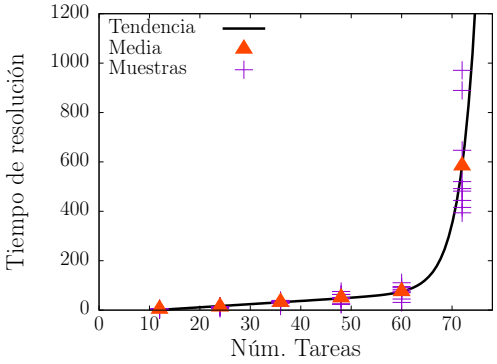


Figura 4.16: Tiempo de resolución del modelo en función del número de tareas.

Para el estudio del impacto del número de CRs sobre el tiempo de resolución modificamos un problema de ejemplo base con 35 tareas y 8 recursos. Modificaremos el número de CRs y mediremos el tiempo que tarda el modelo en encontrar la primera solución factible. Obtenemos los resultado descritos por la figura 4.15.

El modelo se mantiene firmemente por debajo de los 10 minutos hasta sobrepasar los 42 CRs, tras lo cual explota el tiempo de resolución.

Para el estudio del impacto del número de tareas sobre el tiempo de resolución modificamos un problema de ejemplo base con 21 CRs y 8 recursos. Modificaremos el número de tareas y mediremos el tiempo que tarda el modelo en encontrar la primera solución factible. Obtenemos los resultado descritos por la figura 4.16.

El modelo se mantiene por debajo de los 10 minutos hasta sobrepasar las 60 tareas, tras lo cual explota el tiempo de resolución.

Al ser un problema del NP-Duro, la resolución dependerá fuertemente del problema concreto a resolver. Aun así el modelo presenta unos resultados prometedores ya que, aún limitado en el tamaño del proyecto a planificar, ofrece una respuesta exacta siendo capaz de amoldarse a escenarios complejos mediante las abstracciones que recursos, tareas y modos que ofrece.

Capítulo 5

Conclusiones y trabajo futuro

Índice

5.1. Conclusiones generales

Es esencial para el gestor de un proyecto ser capaz de adaptarse a cambios, re-planificar un proyecto en respuesta a los sucesos de día a día. A lo largo del presente trabajo, hemos construido e implementado un modelo matemático que sirve como herramienta auxiliar a la gestión de proyectos. Con esta herramienta se quiere dar una ventaja en el mercado competitivo empresarial de hoy en día donde el más pequeño margen de beneficios cuenta.

Hemos analizado la ventajas y desventajas de los planteamientos históricos de problema clásico de *resource constrained project scheduling problem* (RCPSP) o *problema de programación de proyectos con recursos limitados*. Sobre estos hemos construido sobre la idea básica para ofrecer un modelo que se adapte a las necesidades de los proyectos modernos.

La naturaleza de problema nos fuerza a buscar el planteamiento más eficiente posible, manteniendo el compromiso con la realidad de la gestión de proyectos. El modelo tiene que ser flexible y robusto, cualidades que generalmente se oponen. En nuestro caso, hemos propuesto una simplificación del problema con suficiente capacidad expresiva para ofrecer soluciones de calidad en tiempos de ejecución razonable.

El modelo construido ha resultado ser muy prometedor, es capaz de darnos solución en un marco de tiempo más que aceptable. De esta manera somos capaces de simplificar el proceso manual. Generalmente, al abordar el problema buscando una solución exacta, los límites de resolución de encuentra entorno a la 60 tareas.

Es fundamental el papel que juegan los optimizadores a la hora de resolver modelos como es el optimizador de agendas desarrollado. Principalmente los optimizadores comerciales están es constante avance, extendiendo los límites de los computadores modernos. Un modelo que repre-

sente el problema de manera eficiente junto con un optimizador puntero habilitan el desarrollo de aplicaciones tremendamente valiosas.

El modelo presentado ha mostrado potencial para ser una herramienta valiosa en el día a día de la planificación de proyectos complejos. Sin embargo, el trabajo no tiene porqué acabar aquí. Siempre podremos ir más allá y con ese fin, presentamos alternativas de trabajo futuro.

5.2. Trabajo futuro

Aunque el presente proyecto ha concluido, la investigación y el desarrollo debe continuar. Por lo tanto, esta sección final ofrecerá posibles ampliaciones o caminos a explorar en el futuro.

Primero, sería de increíble utilidad una herramienta que simplifique el proceso de introducción datos. Cada problema será único y debe cumplir con ciertas estructuras no muy intuitivas. Para ello una interfaz que ayude al usuario a traducir la realidad de sus proyectos a los datos relevantes para la resolución del modelo es vital.

También se podría explorar una posible implementación de otros tipos de resolución ya sea algoritmos genéticos o otro tipo de heurísticas a fin de comprobar su utilidad frente a la programación mixta clásica.

Finalmente, siguiendo el objetivo inicial de *se fiel a la realidad*, sería de grán interés hacer ampliaciones a la estructura del modelo, añadiendo parámetros que aporten flexibilidad al modelo. Las ampliaciones propuestas son:

- Implementar una categoría de tareas opcionales que se puedan hacer o no, en función de ciertas situaciones. Esto le permitirá al modelo ayudar al gestor de proyecto con la toma de posibles decisiones.
- Una categoría de tareas recurrentes. A veces, existen actividades que se deben realizar cada cierto tiempo (i.e. mantenimiento de máquinas) que depende de la duración del proyecto.
- Explorar un sistema que precedencias flexible. Permitir expresiones del tipo AND y OR para modelas precedencias más complejas.
- Para las tareas divisibles, sería interesante añadir un parámetro que defina la duración mínima de un fragmento de la tarea completa.
- Desarrollar un mecanismo de *«set up»* y *«shut down»* que considere los tiempos de preparación y suspensión (definitiva o temporal) de las tareas.

En conclusión, esperamos que con futura investigación se perfeccionen las prestaciones del modelo haciéndolo más flexible y útil para la gestión de proyectos.

Bibliografía

[1] Y. C. Chiu. *An Introduction to the History of Project Management: From the Earliest Times to A.D. 1900*. Eburon Uitgeverij B.V., 2010. Google-Books-ID: osNrPO3ivZoC.

[2] J. C. Nash. The (Dantzig) simplex method for linear programming. *Computing in Science Engineering*, 2(1):29–31, January 2000. Conference Name: Computing in Science Engineering.

[3] George B. Dantzig, Alex Orden, and Philip Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955. Publisher: Pacific Journal of Mathematics.

[4] R. Bixby. A Brief History of Linear and Mixed-Integer Programming Computation, 2012.

[5] AK Munns and BF Bjeirmi. The role of project management in achieving project success. *International Journal of Project Management*, 14(2):81–87, April 1996.

[6] Olfa Dridi, Saoussen Krichen, and Adel Guitouni. *Solving resource-constrained project scheduling problem by a genetic local search approach*. April 2013. Journal Abbreviation: 2013 5th International Conference on Modeling, Simulation and Applied Optimization, ICMSAO 2013 Pages: 5 Publication Title: 2013 5th International Conference on Modeling, Simulation and Applied Optimization, ICMSAO 2013.

[7] Edward H. Bowman. The Schedule-Sequencing Problem. *Operations Research*, 7(5):621–624, 1959. Publisher: INFORMS.

[8] Harvey M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800060205.

[9] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, January 2011.

[10] A. Alan B. Pritsker, Lawrence J. Watters, and Philip M. Wolfe. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16(1):93–108, 1969. Publisher: INFORMS.

[11] Alan S. Manne. On the Job-Shop Scheduling Problem. *Operations Research*, 8(2):219–223, 1960. Publisher: INFORMS.

[12] A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, November 1962.

[13] Noemie Balouka and Izack Cohen. A robust optimization approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, September 2019.

[14] Elizabeth Croteau. An On-Event Based Model for Resource Constrained Scheduling of Aircraft Heavy Maintenance Tasks, 2018.

[15] Guler Ozturk and Adalet Oner. Continuous Time MILP Models for Multi-Mode Resource Constrained Project Scheduling Problems. In *2020 9th International Conference on Industrial Technology and Management (ICITM)*, pages 51–56, February 2020.

[16] Victoria G. Achkar, Vanina G. Cafaro, Carlos A. Méndez, and Diego C. Cafaro. Discrete-Time MILP Formulation for the Optimal Scheduling of Maintenance Tasks on Oil and Gas Production Assets. *Industrial & Engineering Chemistry Research*, 58(19):8231–8245, May 2019. Publisher: American Chemical Society.

[17] Christodoulos A. Floudas and Xiaoxia Lin. Mixed Integer Linear Programming in Process Scheduling: Modeling, Algorithms, and Applications. *Annals of Operations Research*, 139(1):131–162, October 2005.

[18] Jiby Joy, Srijith Rajeev, and Vishnu Narayanan. Particle Swarm Optimization for Resource Constrained-project Scheduling Problem with Varying Resource Levels. *Procedia Technology*, 25:948–954, January 2016.

[19] F. Habibi, F. Barzinpour, and S. Sadjadi. Resource-constrained project scheduling problem: review of past and recent developments. *Journal of Project Management*, 3(2):55–88, 2018.

[20] Ignacio E. Grossmann, Ignacio Quesada, Ramesh Raman, and Vasilios T. Voudouris. Mixed-Integer Optimization Techniques for the Design and Scheduling of Batch Processes. In Ginraras V. Reklaitis, Aydin K. Sunol, David W. T. Rippin, and Öner Hortaçsu, editors, *Batch Processing Systems Engineering*, NATO ASI Series, pages 451–494, Berlin, Heidelberg, 1996. Springer.

[21] Kalyanmoy Deb and Koushik Pal. Efficiently Solving: A Large-Scale Integer Linear Program Using a Customized Genetic Algorithm. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation – GECCO 2004*, Lecture Notes in Computer Science, pages 1054–1065, Berlin, Heidelberg, 2004. Springer.

[22] Hans Mittelmann. Benchmarks for Optimization Software.