Trabajo Fin de Grado

Ingeniería de Organización Industrial

Resolución de problemas de optimización con Gurobi bajo entorno Python. Aplicación al problema de Steiner

Autor: Rafael María Vizcaíno Primo

Tutor: José Manuel García Sánchez

Ç

Dpto. Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, julio 2021

Trabajo Fin de Grado

Ingeniería de Organización Industrial

Resolución de problemas de optimización con Gurobi bajo entorno Python. Aplicación al problema de Steiner

Autor:

Rafael María Vizcaíno Primo

Tutor:

José Manuel García Sánchez

Profesor titular

Departamento de Organización Industrial y
Gestión de Empresa I

Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Carrera: RESOLUCIÓN DE PROBLEMAS DE OPTIMIZACIÓN CON GUROBI BAJO ENTORNO PYTHON. APLICACIÓN AL PROBLEMA DE STEINER

Autor: Rafael María Vizcaíno Primo

Tutor: José Manuel García Sánchez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

RESUMEN

Este proyecto de fin de grado intentará servir como un modesto manual para la implementación de problemas de optimización con Gurobi bajo entorno Python.

Para ejemplificar el uso de estas dos herramientas trataremos de resolver el problema de Steiner, un problema de grafos con una gran aplicación en diversas industrias.

Estudiaremos la resolución de este problema bajo dos modelos matemáticos, la formulación de Miller, Tucker y Zennit y la variante de esta, la formulación de Desroches y Laporte.

Una vez realizadas las pruebas con ambos modelos, intentaremos obtener respuesta a cuál de los dos es mejor utilizar en función del tipo de grafo a resolver.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN Y OBJETIVOS

En la última década, debido al incipiente aumento del uso de la tecnología en nuestras vidas, la cantidad de datos generados diariamente no ha hecho más que crecer de manera exponencial. Esto, sumado al generalizado uso de las redes sociales, ha provocado un cambio en el enfoque al que estaban dirigidas las herramientas orientadas al análisis de datos. Hoy en día cualquier empresa necesita aplicaciones muy potentes para evaluar o hacer predicciones debido a que contamos con una inmensa cantidad de información que es necesario aprovechar.

Uno de los lenguajes de programación que se ha visto afectado por este cambio de paradigma es Python. Aunque creado en 1990, no ha sido hasta esta última década cuando el número de usuarios de este lenguaje ha aumentado considerablemente, alcanzando a los más conocidos como JavaScript, Java, PHP, etc. Esto es así por el actual uso de Python en la ciencia de datos, además de ser un lenguaje muy versátil y con una rápida curva de aprendizaje. Por todo lo anterior creemos que Python es un lenguaje muy importante de cara al futuro y es la primera causa de su utilización en este trabajo.

Respecto a la segunda causa para utilizar Python, esta es debida al software Gurobi. El uso de este software, al igual que el de Python, ha aumentado en los últimos años por los cambios comentados anteriormente. Cada vez necesitamos programas de optimización más rápidos y con mayor capacidad de uso de datos y Gurobi cuenta con estas características. Además, en las propias instrucciones del software se nos recomienda su uso bajo Python, por tanto, este proyecto nos servirá para hacer una introducción de ambos.

A lo largo de este trabajo intentaremos ofrecer un modesto manual para cualquier usuario que tenga intención de utilizar las herramientas descritas anteriormente para implementar problemas de optimización. Sin ir más lejos, el propio autor de este proyecto ha aprendido desde cero a manejar tanto Python como Gurobi al no haber materias durante la realización del grado universitario que traten sobre las mismas. Así, durante la lectura de este trabajo intentaremos acercar al lector al proceso de crear y resolver problemas de optimización con Gurobi bajo entorno Python.

Para demostrar el potencial de ambas herramientas, en este trabajo resolveremos el problema de Steiner con Python y Gurobi. Este problema tiene una amplia aplicación en nuestros días en ámbitos como infraestructuras, redes de telecomunicaciones o redes biológicas. Además, formularemos el problema de Steiner desde dos modelos matemáticos, el modelo Miller, Tucker, Zennit y su variante de Desroches-Laporte, para encontrar respuesta a cuál de los dos problemas es mejor en función de la situación.

Por tanto, este trabajo cuenta con dos objetivos. El primero será hacer una introducción de dos herramientas que hoy en día no son tan conocidas y en futuro pueden ser esenciales por todos los cambios tecnológicos que se han producido recientemente. El segundo objetivo será resolver el problema de Steiner con ambas herramientas para evaluar dos formulaciones matemáticas similares y estudiar cuál de ellas es mejor.

Para lograr los dos objetivos propuestos, primeros conoceremos todos los elementos de Python y Gurobi que nos servirán para crear y resolver problemas de optimización. Una vez vistos, explicaremos el problema de Steiner y las formulaciones matemáticas utilizadas en proyecto. Finalmente, implementaremos ambos modelos con Gurobi y Python para resolverlos y comentaremos los resultados obtenidos.

2. DESCRIPCIÓN DE LA HERRAMIENTA DE RESOLUCIÓN (PYTHON + GUROBI)

Para el desarrollo del trabajo nos hemos enfocado en el uso de dos herramientas. En primer lugar, nos centraremos en Python, un lenguaje de programación muy utilizado hoy en día en el contexto de las ciencias. En segundo lugar, explicaremos en qué consiste el software para la resolución de problema de optimización GUROBI.

2.1. Python

Python es un lenguaje de programación creado por Guido Van Rosum a principios de la década de 1990. Se trata de un lenguaje interpretado, con tipado dinámico, fuertemente tipado, multiplataforma y multiparadigma. Para la introducción a Python la información ha sido obtenida de [5]. Las características nombradas anteriormente se describen a continuación.

- **Lenguaje interpretado**. Un lenguaje interpretado o de script se ejecuta mediante un programa intermedio denominado intérprete, a diferencia de los lenguajes compilados. La diferencia más notoria entre lenguaje interpretado y compilado reside en el tiempo de ejecución, el cual es menor para los lenguajes compilados. En cambio, los lenguajes interpretados son más flexibles y portables.
- **Tipado dinámico**. No es necesario declarar el tipo de variable antes de asignar a esa variable un tipo de dato. Una vez el dato es asignado a la variable, ésta será del mismo tipo que el dato. Además, si asignamos a una variable definida previamente un tipo de dato distinto, la variable será del tipo de este último dato.
- **Fuertemente tipado**. Definida una variable y su tipo, no es posible tratarla con métodos y funciones propias de otro tipo. A modo de ejemplo, definida una variable como tipo ‘string’ no es posible sumarle un entero, debido a que este entero es de tipo ‘int’. Para realizar esta acción, previamente tenemos que convertir la variable en tipo ‘int’. Esto es posible por lo explicado en el punto anterior.
- **Multiplataforma**. El intérprete de Python está disponible para multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.). Esto permite que un programa desarrollado en cualquiera de estas plataformas pueda ser utilizado en otra sin demasiados cambios.
- **Multiparadigma**. Acepta diferentes paradigmas (técnicas) de programación, tales como la orientación a objetos o aspectos, programación imperativa y programación funcional.
- **Sangría**. Esta característica es una de las principales diferencias con otros lenguajes de programación. Los bloques de código ya sean estructuras condicionales o bucles, son reconocidos por el intérprete de Python mediante la sangría. En otros lenguajes se utilizan elementos, como llaves en C++ o php. Veremos en el apartado de control de flujo cómo realizamos el sangrado para las distintas estructuras.

2.1.1. Tipos de datos

Enteros

En Python este dato es representado mediante la palabra 'int'. Este tipo abarca el conjunto de todos los números positivos y negativos que no tienen parte decimal, además del 0. Para plataformas de 32 bits, el dominio del tipo int es de -2^{31} a 2^{31} -1. Para plataformas de 64 bits este número es incrementado cuatro veces. Al asignar un número a una variable esta pasará a ser del tipo int.

Reales

Este tipo de dato representa los números que tienen parte decimal y se expresa como tipo 'float'. Python implementa este tipo utilizando 64 bits, por tanto, los valores que podemos representar van desde ±2,2250738585072020 x 10^{-308} hasta ±1,7976931348623157×10^{308}.

Cadenas

Una cadena o string consiste en una secuencia de caracteres entre comillas simples ('string') o dobles ("string"). Dentro de las comillas se pueden añadir caracteres especiales escapándolos con \, como \n, el carácter de nueva línea, o \t, el de tabulación.

Booleanos

Este tipo de dato solo admite dos valores posibles, True (verdadero) o False (falso). El tipo booleano es muy importante para el uso de sentencias condicionales y bucles, los cuales explicaremos más adelante.

2.1.2. Operadores

Una vez definidos los tipos de datos admitidos por el lenguaje Python, mostraremos a continuación las posibles operaciones permitidas. En primer lugar, nos hemos centrado en las aplicables a los dos primeros tipos de datos comentados, enteros y reales. Dentro de los distintos operadores aritméticos, el operador de suma también puede ser utilizado con variables de tipo cadena. Su funcionamiento no es el de sumar dos cadenas de caracteres, si no concatenar. Dadas dos cadenas "Hola" y "mundo", mediante el operador + podemos realizar la siguiente operación: r = "Hola" + "mundo".

OPERADOR	DESCRIPCIÓN	EJEMPLO	
+	Suma	r = 3 + 2	# r = 5
-	Resta	r = 4 - 7	# r = -3
-	Negación	r = - 7	# r = -7
*	Multiplicación	r = 2 * 6	# r = 12
**	Exponente	r = 2 ** 6	# r = 64
/	División	r = 3.5 / 2	# r = 1.75
//	División entera	r = 3.5 // 2	# r = 1.0
%	Módulo o resto	r = 7 % 2	# r = 1

Tabla 1 Operadores Aritméticos

Fuente: Python para todos. Raúl González Luque.

OPERADOR	DESCRIPCIÓN	EJEMPLO
and	¿Se cumple a y b?	r = True y False # r = False
or	¿Se cumple a o b?	r = True y False # r = True
not	No a	r = not True # r = False
==	¿Son iguales a y b?	r = 5 == 3 # r = False
¡=	¿Son distintos a y b?	r = 5 ¡= 3 # r = True
<	¿Es a menor que b?	r = 5 < 3 # r = False
>	¿Es a mayor que b?	r = 5 > 3 # r = True
<=	¿Es a menor o igual que b?	r = 5 <= 3 # r = False
>=	¿Es a mayor o igual que b?	r = 5 >= 3 # r = True

Tabla 2 Operadores Lógicos

Fuente: Python para todos. Raúl González Luque.

2.1.3. Colecciones

Las colecciones son un tipo de elemento de Python en el que podemos guardar más de un tipo de dato. En los apartados anteriores siempre nos hemos referido a las variables por su tipo en función del tipo de dato que almacenan.

Para las colecciones, las cuales veremos que son tres distintas, su tipo no viene definido por los datos asignados. Estos tres tipos son lista, tupla y diccionario. A cada uno de estos elementos podemos asignar valores enteros, flotantes, cadenas y booleanos. Por tanto, contamos con tres tipos de datos nuevos.

Listas

Una lista es una secuencia ordenada. Su estructura es similar al vector o array de otros lenguajes de programación. La característica de ordenada es muy importante pues, como veremos en los diccionarios, estos no siguen ningún patrón de orden. Las listas se definen por el uso de corchetes ([ ]), siendo capaces de almacenar cualquier tipo de dato. Para acceder a un elemento de la lista es tan sencillo como llamar a la lista con el nombre dado seguido de corchetes y en su interior el índice del elemento buscado. Un ejemplo sería: lista[2]. Esta sentencia devolverá el tercer elemento de la lista.

La siguiente característica por destacar de las listas es que son mutables. Una vez definida una lista y los elementos que contiene, podemos cambiar el valor de cualquier posición por otro, únicamente haciendo referencia al índice de ese dato. Además, podemos añadir nuevos elementos mediante métodos definidos para la clase lista.

Esto es así porque las listas son una clase de objeto. La programación orientada a objetos se explicará más adelante.

Por último, cabe destacar otra propiedad de las listas, denominada slicing o partición. Existe la posibilidad de acceder a un subconjunto de los elementos contenidos en una lista. Expondremos un ejemplo para ser más concisos. Sea una lista llamada l, formada por 8 valores, con la sentencia l[1:4] accedemos a los valores que van desde la posición 1, es decir, el segundo valor, hasta la posición 4 sin incluir ésta última.

Tuplas

Este tipo de elemento es muy similar a las listas, por tanto, nos centraremos en sus diferencias. Las tuplas se definen mediante paréntesis: tupla = (1, 7.8, "hola", True). Para acceder a un elemento utilizamos el mismo método que en las listas, tupla[1]. Las tuplas son inmutables, es decir, una vez creada no podemos modificar los valores que contiene ni añadir o eliminar nuevos elementos. Esto permite en manejo de memoria que sea un elemento más ligero que las listas.

Diccionarios

También llamados matrices asociativas, se definen mediante llaves, clave y valor d={"a" : 1}, donde "a" es la clave y 1 el valor asociado a dicha clave. Este tipo de colección no es ordenado, como sí lo son listas y tuplas. Para acceder a un elemento de un diccionario utilizamos la clave. Utilizando el ejemplo anterior: d["a"], lo que nos devolvería el valor 1. Podemos utilizar cualquier elemento inmutable como clave en un diccionario. Quedan excluidos tanto listas como diccionarios por este motivo. Al igual que las listas, tenemos la posibilidad de modificar valores de las claves o añadir nuevos elementos al diccionario.

2.1.4. Control de flujo

Hasta ahora hemos visto sentencias secuenciales, en las que el flujo no toma bifurcaciones ni acciones de repetición. En Python podemos diferenciar dos tipos: condicionales y bucles. Ambos bloques de flujo comparten la utilización de la sangría, explicada anteriormente en las características de Python. Entraremos en una de estas estructuras, ya sea un bucle o condición, mediante el uso de sangría y saldremos de la misma estructura eliminando la sangría. Veremos su utilización con la siguiente imagen.

Ilustración 1. Ejemplo de sangría en Python

Fuente: Elaboración propia

La sangría o espaciado, marcada con color rojo, es lo que hace entender al intérprete de Python que esa línea 3 está dentro del bucle for iniciado en la línea 2. En la línea 4 hemos

eliminado la sangría y por tanto esa sentencia se ejecutaría una vez hemos recorrido el bucle for. Por tanto, cada estructura de flujo tendrá tantas líneas de código como líneas espaciadas y finalizará en el momento que la siguiente línea elimine ese espaciado.

Condicional

Las expresiones condicionales nos permiten evaluar una sentencia y en función del resultado, realizar distintas acciones.

Condicional if

```
1   a=5
2   if a >= 3 :
3       print(a)
4
```

Ilustración 2. Ejemplo condicional if Python

Fuente: Elaboración propia

Mediante este ejemplo explicaremos la sintaxis y el control del flujo de este condicional. En primer lugar, asignamos a la variable a el número entero 5. A continuación, mediante la sentencia if, preguntamos si a es mayor o igual que 3. Esta sentencia es evaluada y tiene dos posibles respuestas, True o False. Como sabemos, la respuesta a la pregunta es afirmativa, por tanto, la sentencia print(a) es ejecutada. En caso de ser la respuesta falsa el programa no habría ejecutado ninguna sentencia.

Condicional if-else

Este caso es muy similar al anterior. Modificaremos el ejemplo anterior.

```
1   a=5
2   if a >= 3 :
3       print(a)
4   else :
5       print(-a)
6
```

Ilustración 3 Ejemplo condicional if-else Python

Fuente: Elaboración propia

A diferencia del caso anterior, en este ejemplo ejecutaríamos la sentencia print(a) en caso de que la condición evaluada fuera verdadera y en caso de ser falsa ejecutamos print(-a). Podríamos decir: "haz esto si se cumple la condición o haz esto otro si no se cumple".

Condicional if-elif-elif-…-else

Hemos estudiado el caso de hacer algo si se cumple la condición y el caso de hacer una u otra cosa si cumple condición. Este caso es el último y permite incluir más de dos alternativas. La sintaxis es la siguiente:

```
a=10
if a<5 :
    print ("número pequeño")
elif a>=5 and a<10 :
    print("número mediano")
else :
    print("número grande")

```

Ilustración 4 Ejemplo condicional if-elif-...-else Python

Fuente: Elaboración propia

Con este ejemplo tan sencillo podemos observar la sentencia que se ejecutará en el programa: print("número grande"). No hay un límite de elif's establecido así que podemos incluir tantos caminos alternativos como queramos.

En resumen, las sentencias condicionales nos permiten evaluar una o más condiciones en función de las alternativas que hayamos decidido. Se evalúan por orden de aparición y solo se ejecutará la primera que devuelva el valor booleano True.

Bucles

Los bucles nos permiten ejecutar un cierto fragmento de código un cierto número de veces mientras se cumpla una condición.

While

```
i=5
while i > 0:
    print (i)
    i = i-1

```

Ilustración 5 Ejemplo bucle while Python

Fuente: Elaboración propia

El bucle while consta de dos partes. En primer lugar, la llamada variable de iteración, en este caso i. Por otra parte, tenemos la condición de entrada en el bucle.

El fragmento de código dentro del bucle se repetirá tantas veces como se cumpla la condición. Para no provocar un bucle infinito actualizamos la variable de decisión, en este caso reduciéndola. En el momento en el que la variable i sea menor o igual que 0 el bucle habrá finalizado.

For

```
1  l=[1,2,3,4,5]
2  for num in l :
3      print(num)
4
```

Ilustración 6 Ejemplo bucle for Python

Fuente: Elaboración propia

Al igual que en el bucle while, podemos observar dos partes. También tenemos una variable de iteración, en este caso num. La diferencia radica en la manera de entrar en el bucle. El bucle for no evalúa una condición de entrada, si no que recorre una secuencia. En este caso se trata de una lista de números, pero puede ser una tupla, un diccionario e incluso una cadena de caracteres.

Por último, tenemos que hacer referencia a dos palabras clave para la construcción de condicionales y bucles. Estas son break y continue. El efecto de utilizar break es el de salir del bucle o condición. Es decir, pasa a la siguiente línea de código. El uso de continue es diferente. En vez de finalizar la ejecución, vuelve al principio del bucle sin ejecutar todo lo escrito a continuación de la palabra.

2.1.5. Funciones

Una función es un fragmento de código que recibe uno o más parámetros y devuelve un valor. En los apartados anteriores hemos utilizado una función, print( ), la cual recibe un tipo de dato, ya sea una cadena, una lista o un entero y lo imprime por pantalla. La sintaxis para definir funciones en Python es la siguiente:

```
1  def suma(a,b):
2      c=a+b
3      return c
4
```

Ilustración 7 Ejemplo declaración de funciones Python

Fuente: Elaboración propia

Vamos a comentar el proceso de la función. Creamos la función suma, la cual recibe como parámetro dos valores. Al llamar a la función, esta realiza una suma en su interior y devuelve la suma de esos 2 parámetros recibidos por medio de la variable c. Para nuestro

trabajo no hemos necesitado la creación de funciones ya que Python cuenta con un elevado número de estas para facilitar el trabajo a los usuarios.

Por tanto, vamos a comentar las funciones de Python utilizadas para el desarrollo del trabajo:

- print(). Esta función permite mostrar por pantalla el argumento que recibe. Como argumento puede recibir cualquier tipo de dato, ya sea una cadena, entero, flotante, tupla…
- open(). La función open() permite abrir ficheros. Normalmente se asigna a una variable para tener ese archivo en una variable y poder utilizar la información o incluso escribir en ese archivo. Como argumentos recibe la dirección del archivo y una letra en función de lo que vayamos a hacer con el archivo. En función de esto tendremos:
  - Abrir fichero de lectura: f = open("fichero.txt")
  - Abrir fichero de lectura: f = open("fichero.txt", "r")
  - Abrir fichero de lectura en binario: f = open("fichero.txt", "rb")
  - Abrir fichero para escribir desde cero: f = open ("fichero.txt", "w")
  - Abrir fichero para añadir al final: f = open ("fichero.txt", "a")

Concretamente, en el trabajo desarrollado hemos utilizado la función open para abrir el archivo que contiene los datos del problema y convertirlos en variables de Python y una vez resuelto el problema guardar la solución y el tiempo de ejecución en otro archivo.

- int(). Convierte, cuando sea posible, cualquier argumento que le pasemos en un número entero. Para ello es necesario introducir dígitos y no letras. Puede ser un mismo número entero o un número cualquiera siempre que esté en base 10.
- str(). Esta función actúa de la misma forma que la función anterior. La diferencia es que ahora el argumento pasado se convertirá al tipo string.
- len(). La función len() devuelve la longitud de una cadena de caracteres o el número de elementos de una lista. El argumento de la función len() es la lista o cadena que queremos "medir".
- range(). Retorna una sucesión de números enteros. Cuando se le pasa un único argumento n, la sucesión empieza desde el cero y culmina en n-1. Esta función es muy utilizada cuando usamos un bucle for. La sintaxis utilizada sería del tipo:
  ```
  for i in range(10) :
      print(i)
  ```
  Lo que estamos haciendo aquí es recorrer una lista de número que va desde el 0 hasta el 9 e imprimirla por pantalla. Si se especifican dos argumentos, el primero pasa a indicar el inicio de la sucesión. Ej: range(1, 11). Un tercer argumento indica el intervalo entre dos números de la sucesión resultante. Ej: range(1 ,11, 2).

2.1.6. Orientación a objetos

Una de las características nombradas al principio de este capítulo respecto a Python es que es un lenguaje multiparadigma. Uno de estos paradigmas es la orientación a objetos. La explicaremos brevemente, puesto que, no es tan interesante para nuestro trabajo la

construcción de clases y de objetos y métodos pertenecientes a esa clase como las clases ya definidas por Python y algunos de los métodos que utilizaremos para este trabajo.

A modo de ejemplo, pensemos en una persona. Una persona tiene nombre, apellidos, edad, etc. Una clase sería la clase persona. Dentro de la clase persona tenemos diferentes objetos, los cuales se diferencian por sus atributos. Cada persona tendrá un nombre, apellido y edad.

La sintaxis para crear una clase, con sus atributos y métodos, y posteriormente objetos de esa clase será omitida puesto que no es necesaria para la realización del trabajo.

Una vez explicado en qué consiste una clase, podemos observar que los tipos de datos y colecciones nombrados en los capítulos anteriores son también clases. Por ejemplo, podemos crear dos listas con valores diferentes. Ambas pertenecen a la clase lista, pero difieren en los valores de sus atributos, como podría ser el tamaño.

A continuación, veremos algunos de los métodos utilizados en este trabajo para la construcción de nuestro problema. Dividiremos estos métodos en función de la clase a la que pertenecen, ya que hay métodos únicos en función de nuestro objeto, que puede ser una cadena de texto, una lista o un diccionario.

- Cadena de texto
  - split(). Este método recibe como argumento una cadena de caracteres y devuelve una lista que estará formada por los componentes de la cadena.

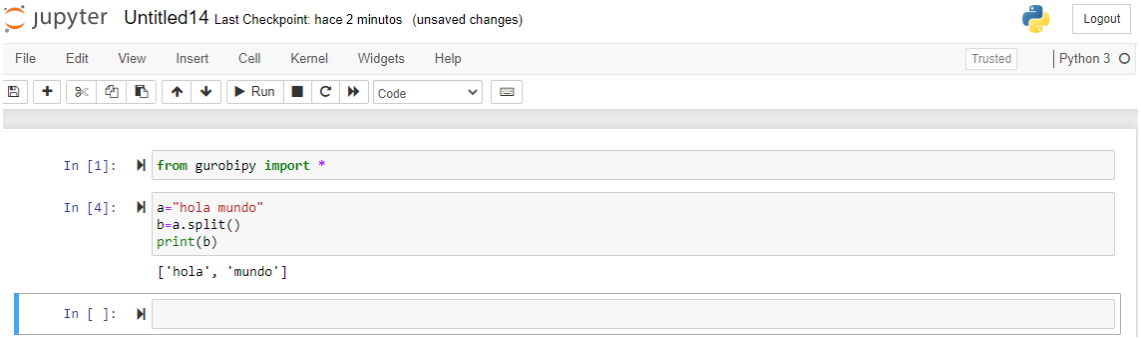


Ilustración 8 Ejemplo método split Python

Fuente: Elaboración propia

Si no especificamos nada la separación de la cadena la hará por cada espacio en blanco. También podemos especificar el número máximo de divisiones y en caso de no hacerlo dividirá la cadena hasta el último espacio en blanco o separador especificado.

  - write(). Mediante este método podemos escribir en archivos. Como argumento recibe una cadena de caracteres.
  - close(). Este método lo utilizaremos para cerrar un archivo una vez escrita la información que queremos.

- Lista
  - append(). Este método nos permite agregar nuevos elementos a la última posición de una lista. Podemos agregar cualquier tipo de elemento a una lista y este elemento será el argumento.

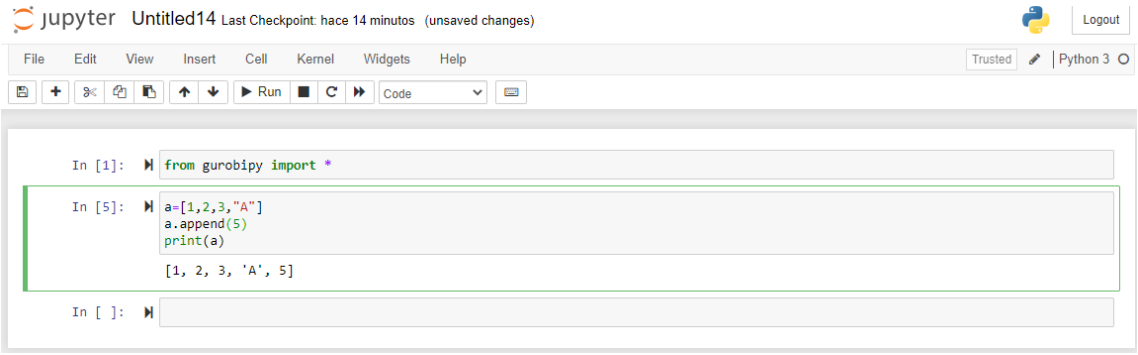


Ilustración 9 Ejemplo método append Python

Fuente: Elaboración propia

  - remove(). Con el método remove podemos eliminar un elemento cualquiera de una lista. El elemento que deseamos eliminar se indica como argumento. En caso de que ese elemento estuviese repetido, remove eliminaría el primero de ellos en la lista.

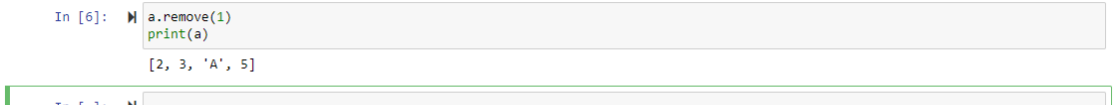


Ilustración 10 Ejemplo método remove Python

Fuente: Elaboración propia

  - sort(). Este método sirve para ordenar una lista en orden ascendente. Si tenemos una lista l = [3, 7, 1, 9] y utilizamos l.sort() el resultado sería l = [1, 3, 7, 9]. También es posible utilizarlo para ordenar listas con cadenas de caracteres en lugar de número.

- Diccionario
  - keys(). Genera una lista con las claves de un diccionario. Este método no recibe ningún argumento.

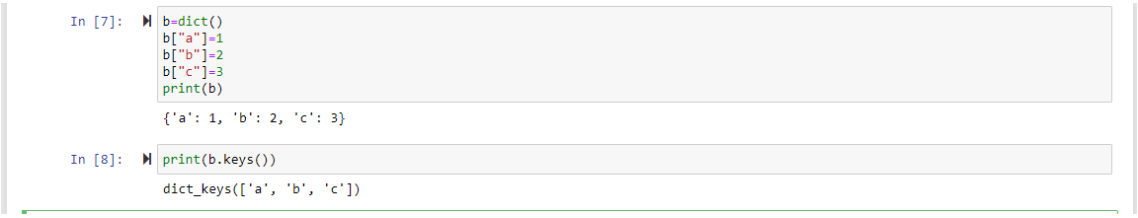


Ilustración 11 Ejemplo método keys Python

Fuente: Elaboración propia

2.2. GUROBI

2.2.1. Introducción

El software elegido para desarrollar este trabajo es Gurobi. Desarrollado en 2008, lleva el nombre de sus creadores: Zonghao **Gu**, Edward **Ro**thberg y Robert **Bi**xby. Este software de optimización permite resolver modelos matemáticos tales como programación lineal (LP), programación cuadrática (QP), programación cuadrática restringida (QCP), programación lineal entera mixta (MILP), programación cuadrática entera mixta (MIQP) y programación cuadrática entera mixta restringida (MIQCP).

Nuestro problema será de la clase MILP, es decir, las variables tomarán tanto valores enteros o continuos como binarios.

Una de las características más importantes de este software es la flexibilidad. Gurobi soporta la mayoría de los lenguajes de programación más utilizados actualmente. Dicho esto, si el usuario no tiene preferencia por usar un lenguaje concreto, es la propia empresa la que recomienda la utilización de Python. Esto es así por dos motivos. El primero se debe a la facilidad de aprendizaje que tiene este lenguaje. El propio autor del texto no había cursado ninguna asignatura en sus años de estudio relacionada con dicho lenguaje.

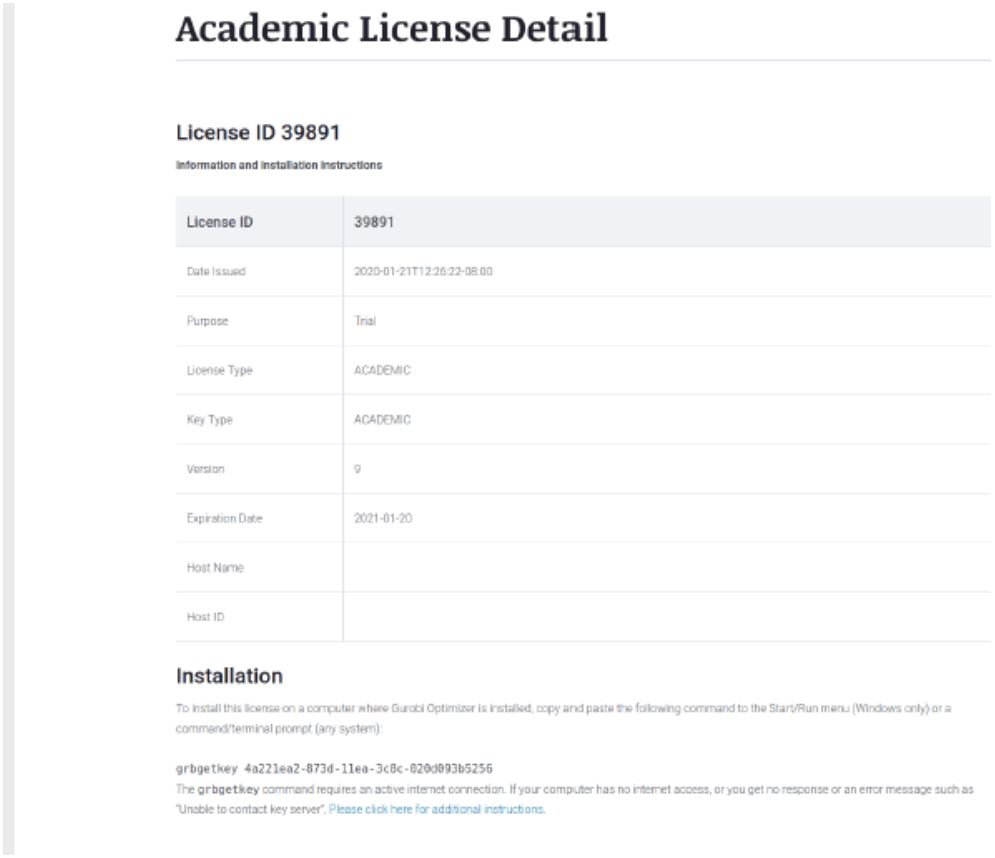
El otro motivo deriva del propio software. La sintaxis utilizada para desarrollar y resolver problemas de optimización en la interfaz de Gurobi y Python es bastante más sencilla e intuitiva respecto a las de los demás lenguajes de programación. En la parte final de este capítulo nos centraremos en esa interfaz. Para introducir al lector a estas dos herramientas hemos seguido la metodología propuesta en [6].

2.2.2. Instalación de gurobi

Para poder hacer uso del optimizador es necesario obtener una licencia. El primer paso de la instalación será el de registrarnos en la plataforma de Gurobi. Siendo estudiantes o perteneciendo a cualquiera de los centros universitarios registrados en la plataforma podemos obtener una licencia de forma gratuita.

Una vez registrados, seleccionamos la versión actual de Gurobi para descargar. Después de aceptar el acuerdo de licencia, podremos descargar e instalar la versión correspondiente a nuestro sistema operativo.

En paralelo a la descarga e instalación, después de registrarnos y tras aceptar las condiciones de la licencia, podemos ver dentro de nuestra sesión la licencia obtenida.



Academic License Detail

License ID 39891

Information and installation instructions

License ID	39891
Date Issued	2020-01-21T12:26:22-08:00
Purpose	Trial
License Type	ACADEMIC
Key Type	ACADEMIC
Version	9
Expiration Date	2021-01-20
Host Name	
Host ID	

Installation

To install this license on a computer where Gurobi Optimizer is installed, copy and paste the following command to the Start/Run menu (Windows only) or a command/terminal prompt (any system):

grbgetkey 4a221ea2-873d-11ea-3c8c-020d093b5256

The grbgetkey command requires an active internet connection. If your computer has no internet access, or you get no response or an error message such as "Unable to contact key server", Please click here for additional instructions.

Ilustración 12 Ejemplo de licencia de Gurobi

Fuente: Elaboración propia

En la imagen se muestra el formato de una licencia para la utilización de Gurobi.

Finalmente, copiamos el número de ésta y abrimos el terminal de nuestro dispositivo. Escribimos el comdando grbgetkey seguido del número de licencia y ya podremos utilizar el software.

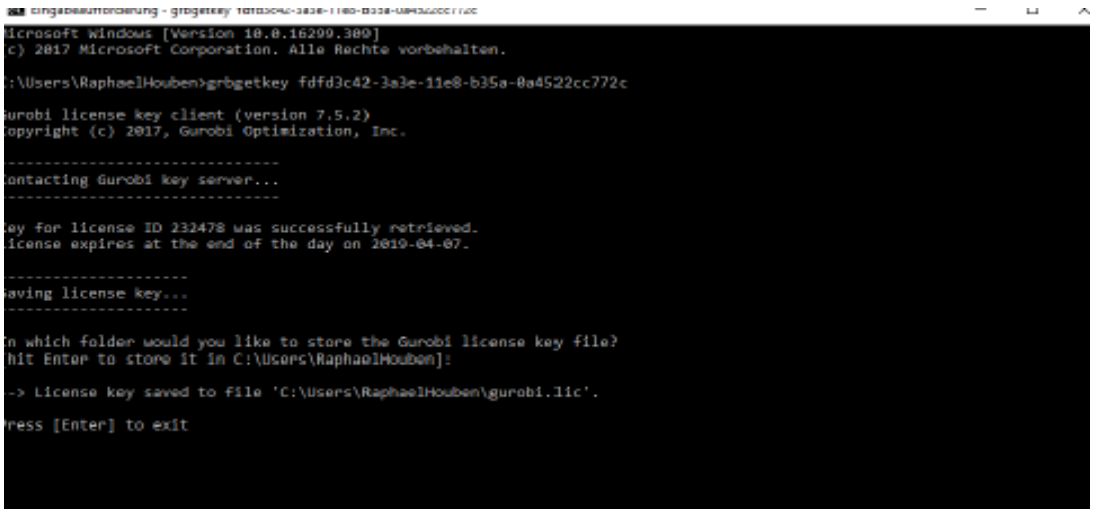


Ilustración 13 Uso de licencia para utilizar Gurobi

Fuente: Elaboración propia

2.2.3. Optimizador de gurobi

La versión del software Gurobi utilizada en este trabajo es la 9.1.1 y mediante este apartado explicaremos su funcionamiento.

Para resolver un problema con este software antes debemos definir dicho problema. Gurobi nos permite dos maneras de hacerlo. La primera consiste en crear el problema en un archivo y una vez realizado este proceso enviarlo al optimizador para que nos devuelva la mejor solución. En función del tipo de archivo utilizado, la definición del problema será de una forma u otra. En concreto Gurobi admite los siguientes formatos: MPS, REW, LP, RLP, IL y OPB.

Una vez definido el problema en cualquiera de los formatos disponibles, obtener el óptimo es muy sencillo. Utilizando la ventana interactiva de Gurobi, la cual es una extensión de la ventana de comandos de Python, leemos el problema para convertirlo en un modelo. Realmente lo que estamos haciendo es crear un objeto de clase modelo, cuyos atributos serán los definidos en el archivo. Este paso permite utilizar métodos de la clase modelo como optimizar, comprobar el valor de las variables, cambiar parámetro como podría ser el tiempo de ejecución, etc.

A continuación, vamos a mostrar un ejemplo del funcionamiento de esta ventana de interactiva. El archivo utilizado está en formato LP y viene dentro de la carpeta de Gurobi al descargar el software.

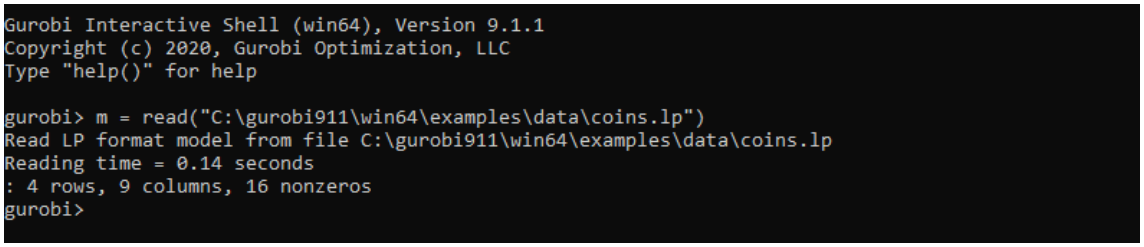


Ilustración 14 ventana interactiva de Gurobi

Fuente: Elaboración propia

El comando ‘read’ es una función global que recibe como parámetro un archivo y devuelve un modelo. Este modelo lo hemos guardado en la variable ‘m’.

```
gurobi> m.optimize()
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 4 rows, 9 columns and 16 nonzeros
Model fingerprint: 0x43bf48c0
Variable types: 4 continuous, 5 integer (0 binary)
Coefficient statistics:
  Matrix range     [6e-02, 7e+00]
  Objective range  [1e-02, 1e+00]
  Bounds range     [5e+01, 1e+03]
  RHS range        [0e+00, 0e+00]
Found heuristic solution: objective -0.0000000
Presolve removed 1 rows and 5 columns
Presolve time: 0.06s
Presolved: 3 rows, 4 columns, 9 nonzeros
Variable types: 0 continuous, 4 integer (0 binary)

Root relaxation: objective 1.134615e+02, 2 iterations, 0.03 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  113.46154    0    1   -0.00000  113.46154      -     -    0s
H    0     0                       113.4500000  113.46154  0.01%     -    0s
     0     0  113.46154    0    1  113.45000  113.46154  0.01%     -    0s

Explored 1 nodes (2 simplex iterations) in 0.25 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 113.45 -0

Optimal solution found (tolerance 1.00e-04)
Best objective 1.134500000000e+02, best bound 1.134500000000e+02, gap 0.0000%
gurobi>
```

Ilustración 15 Salida de datos ventana interactiva de Gurobi

Fuente: Elaboración propia

Utilizamos el método ‘optimize()’ perteneciente a la clase modelo y nos devuelve por pantalla el resultado, que es 113,45, el tiempo de ejecución, las soluciones exploradas, etc.

```
gurobi> m.printAttr('X')

    Variable               X
-------------------------
       Dimes               2
    Quarters              53
     Dollars             100
          Cu           999.8
          Ni            46.9
          Zi              50
          Mn              30
gurobi>
```

Ilustración 16 Ejemplo método printAttr('X')

Fuente: Elaboración propia

Otro método muy interesante es el ‘printAttr()’. Utilizándolo podemos ver el valor que toman las variables distintas de cero en el óptimo.

Estos métodos explicados son los que utilizaríamos para crear un modelo, resolverlo y ver la solución y el valor que toman las variables en el óptimo. Además de esto podemos incluir variaciones en nuestro problema como definir cotas o valores para cualquiera de las variables o incluso establecer un tiempo límite para obtener una solución. Trabajar con la pantalla de comandos de Gurobi es recomendable para resolver problemas sencillos o ya definidos en uno de los formatos que admite. Por tanto, para problemas de mayor complejidad y cuyos datos vienen de una fuente distinta tendremos que utilizar alguna de las interfaces soportadas por Gurobi. En nuestro caso se trata de Jupyter Notebook, la cual explicaremos a continuación.

2.3. Anaconda y Jupyter Notebook

Jupyter Notebook es una de las IDEs pertenecientes a la distribución Anaconda de Python y la herramienta utilizada en este trabajo para crear los modelos matemáticos que luego serán resueltos mediante Gurobi. Antes de adentrarnos en Jupyter Notebook comentaremos brevemente en qué consiste Anaconda y el procedimiento de descarga e instalación.

2.3.1. Distribución Anaconda

Anaconda Distribution es un ecosistema de Python muy popular debido al incipiente desarrollo de la ciencia de datos producido en la última década. Es libre, de código abierto, multiplataforma y cuenta con una gran cantidad de documentación detallada. Esta distribución se compone de distintas IDEs entre las que se encuentra Jupyter Notebook, además de diversas librerías en función de nuestras necesidades, ya sean librerías de visualización, machine learning, etc.

Una de las características que hace Anaconda tan popular es la posibilidad de compartir proyectos de una manera muy sencilla, incluso pueden ser compartidos en vivo, algo esencial en un mundo donde el trabajo en equipo y el desarrollo de proyectos está a la orden del día. También el manejo de las IDEs es muy intuitivo como veremos al adentrarnos en Jupyter Notebook.

2.3.2. Instalación de Anaconda

La versión utilizada en este trabajo es la 3.8. Para descargar Anaconda desde el navegador buscamos la página anaconda.com y seleccionamos la correspondiente con nuestro dispositivo, en este caso Windows. Seguimos las instrucciones del instalador y tendremos ya a nuestra disposición Anaconda.

El último paso es el de descargar e instalar Gurobi en Anaconda. Para ello abrimos el terminal de Windows o el terminal de Anaconda. Una vez abierto, introducimos los siguientes comandos:

-conda config --add channels http://conda.anaconda.org/gurobi [ENTER]
-conda install gurobi [ENTER]

En la siguiente imagen podemos ver el proceso y la respuesta del sistema.

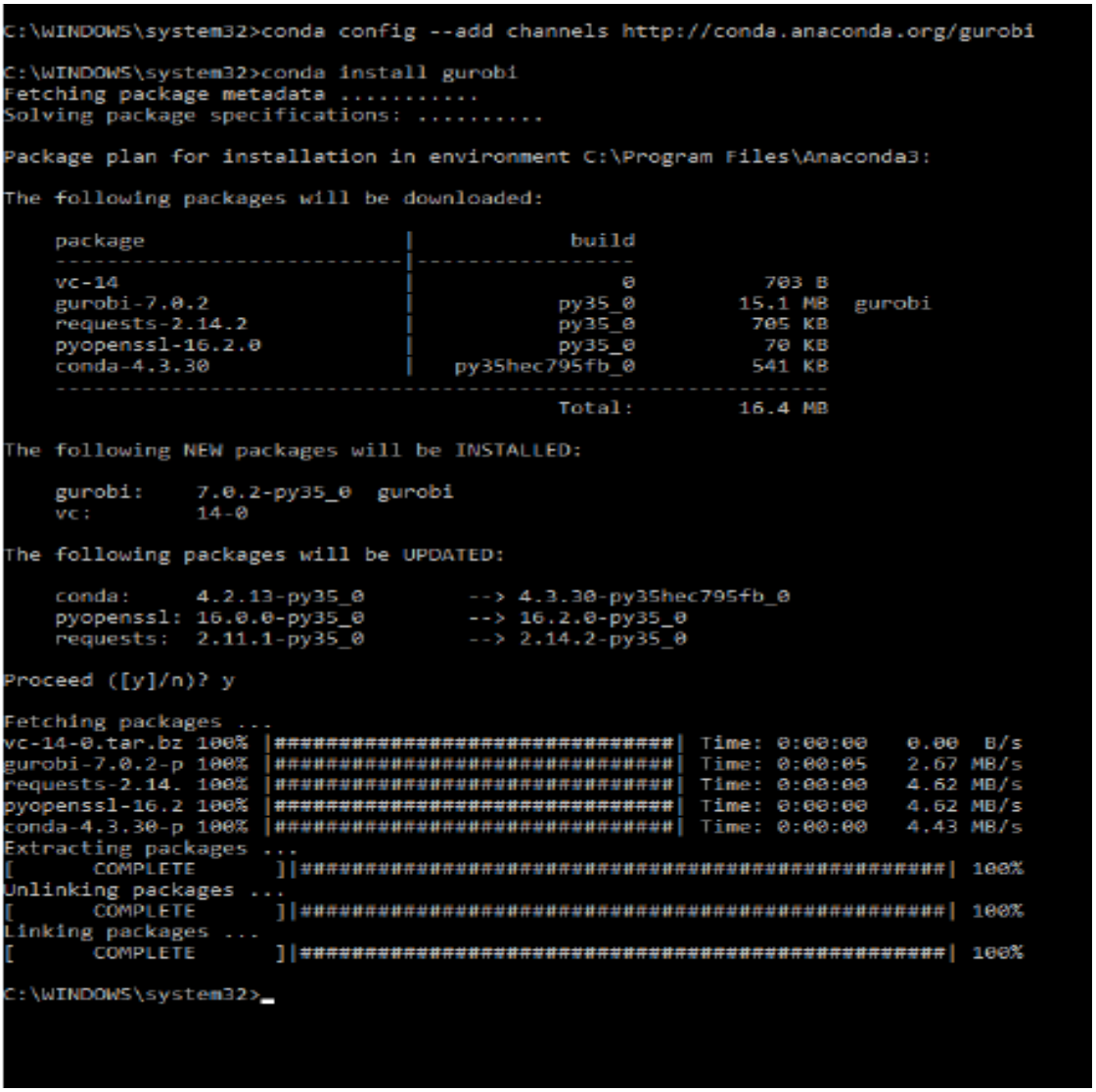


Ilustración 17 Instalación de Gurobi en Anaconda

Fuente: Elaboración propia

Una vez realizado todo lo anterior, podemos crear problemas en Jupyter Notebook y resolverlos con Gurobi. Para acceder a Jupyter primeros tendremos que utilizar el navegador de Anaconda, mostrado en la siguiente imagen.

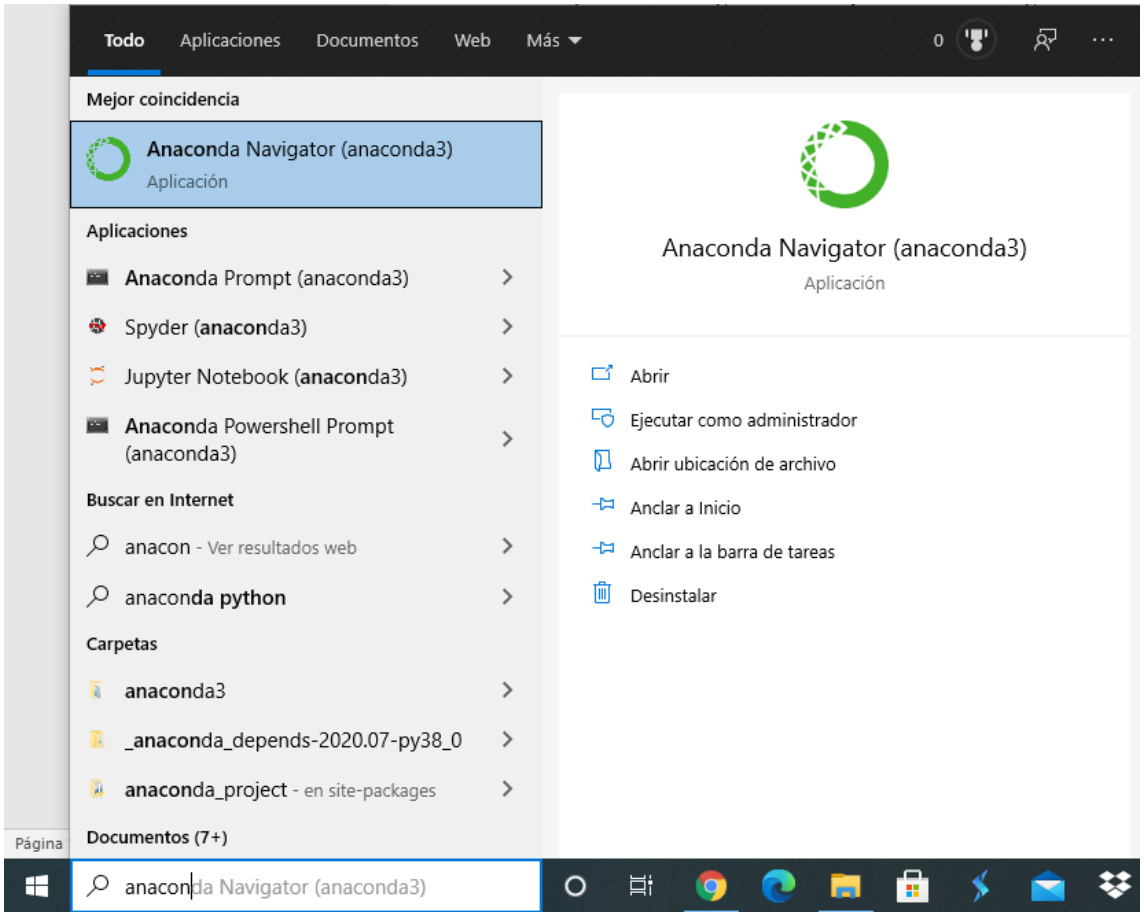


Ilustración 18 Dónde encontar el navegador de Anaconda

Fuente: Elaboración propia

Una vez abierto, esta es la pantalla mostrada por el navegador. Podemos ver las distintas IDEs disponibles, en nuestro caso será Jupyter Notebook.

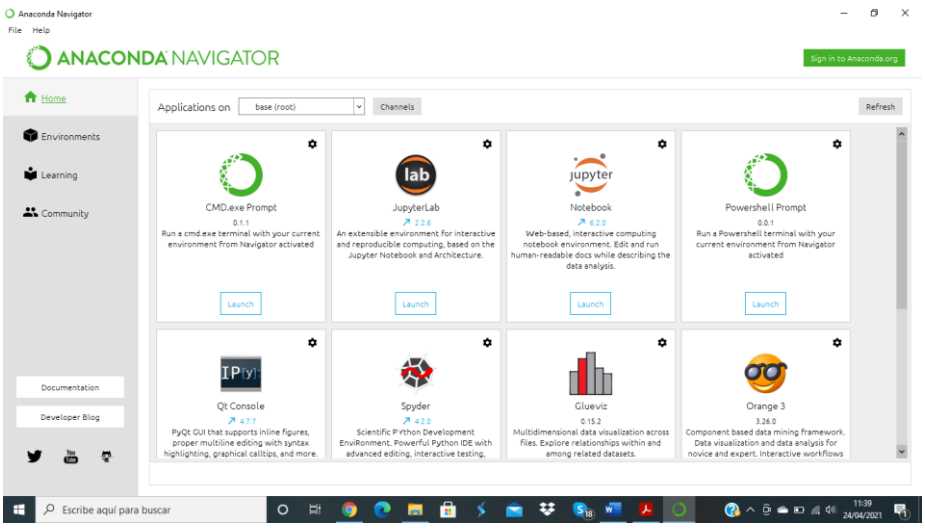


Ilustración 19 Navegador Anaconda

Fuente: Elaboración propia

2.3.3. Jupyter Notebook

Jupyter Notebook es una de las distintas IDEs que encontramos en Anaconda. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Algunas de las otras son JupyterLab, Spyder o Rstudio. Para la realización de este trabajo hemos utilizado Jupyter por recomendación de los creadores de Gurobi.

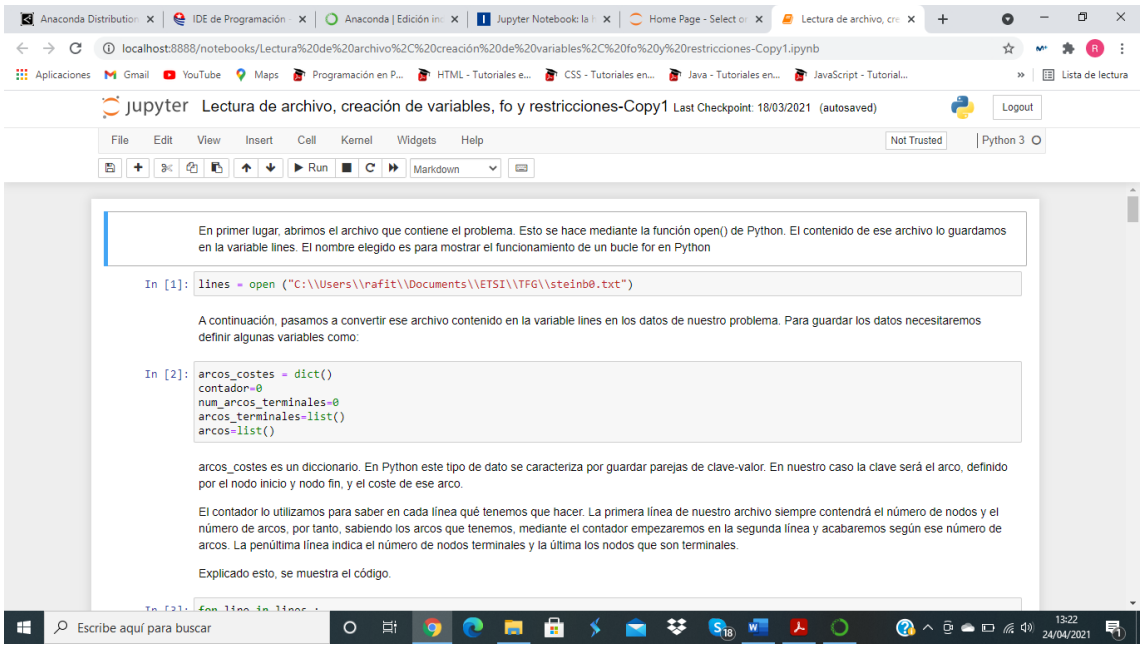


Ilustración 20 Ejemplo Jupyter Notebook

Fuente: Elaboración propia

En la imagen podemos observar un archivo de Jupyter Notebook. Consta de diferentes partes como texto, código en Python y la respuesta del intérprete al código ejecutado. Esta es una de las características más importantes de Jupyter, la posibilidad de intercalar texto, fórmulas matemáticas y código. Esto se consigue mediante la utilización de celdas. En cada celda podemos escribir e indicarle al programa como tiene que interpretar esa celda. Por defecto cada celda es interpretada como si tuviera código escrito. Tan solo indicando que la interprete como “Markdown” conseguimos que el texto no se ejecute y permite realizar breves comentarios o explicaciones para otros usuarios de nuestro archivo. Los archivos pueden ser exportados en distintos formatos como HTML, PDF, Markdown o Python.

Visto el formato de un archivo Jupyter, queda explicar el proceso de creación de un problema de optimización y su posterior resolución con Gurobi.

2.3.4. Creación de un problema en Jupyter Notebook y resolución con Gurobi

Para crear problemas en Jupyter Notebook y obtener una solución tendremos que hacer uso de algunas funciones y métodos propios de Gurobi. Importaremos el módulo Gurobipy en Juptyer Notebook para utilizar estos elementos.

Funciones

- tuplelist(). Esta función es propia de Gurobi y permite crear una lista contenida por tuplas. Como argumento recibe una o un conjunto de tuplas y forma una lista de dimensión igual al número de tuplas que haya recibido como argumento.
- Tupledict(). Es una función que permite crear un diccionario en el que las claves están formadas por tuplas. No necesita de argumento por lo que podemos crear el diccionario recursivamente. Python permite crear diccionarios con la misma composición, pero no permite utilizar algunos métodos que serán necesarios para la creación de problema. Esto solo es posible creando un objeto tupledict. A continuación, veremos cuáles son estos métodos.

Orientación a objetos

- Tupledict. Al utilizar la función tupledict() creamos un objeto de la misma clase. Esta clase permite usar los siguientes métodos:
    - select(). Este método permite seleccionar valores de un diccionario. El argumento que recibe es la clave del diccionario que queremos seleccionar y nos devolverá el valor asociado a dicha clave.
    - sum(). Con el método sum podemos generar un sumatorio de los elementos que formen el diccionario al que se le aplica el método. Podemos decidir los valores a sumar de la misma manera que en el método select().

- Model. Esta clase es, al igual que la función tuplelist, importada del módulo Gurobipy. Mediante el uso de esta clase podremos crear modelos vacíos a los que ir añadiendo variables, restricciones y el criterio a optimizar.
    - addVars(índices, lb = 0.0, ub = float ('inf'), obj = 0.0, vtype = GRB.CONTINUOUS, name = ""). Genera variables asociadas a un índice que se le dará como argumento. Este índice puede ser una lista o un diccionario. Por defecto las variables generadas son continuas, con una cota inferior igual a 0 y una costa superior infinita. Si alguna de estas características de nuestras variables es distinta se indica a través del argumento. El último argumento que recibe es un nombre. Este método devuelve un diccionario con clave igual al a cada uno de los elementos contenidos que hayamos entregado como índice y valor la propia variable asociada a cada elemento.

- setObjective(expresión, criterio(GRB.MAXIMIZE o GRB.MINIMIZE). Establece la función objetivo. Recibe como argumentos la expresión matemática y el sentido a optimizar. No devuelve nada.
- addConstrs(generador, name=" "). Con este método generemos las restricciones del modelo. El primer argumento es una expresión matemática que podrá ser iterable. Esta ventaja nos permite crear restricciones sobre un conjunto de una manera rápida y eficaz. El segundo argumento es opcional. En el siguiente ejemplo podemos ver una restricción por cada elemento del conjunto índice.
- update(). Este método lo utilizamos para actualizar el modelo cada vez que creamos variables, restricciones o declaramos la función objetivo.
- optimize(). El método optimize llama a Gurobi para resolver el modelo. La salida es la siguiente. No recibe argumento.
- objval. Devuelve el valor de la función objetivo en el óptimo.
- runtime. Devuelve el tiempo de resolución para encontrar el óptimo.

Para ilustrar la creación de un problema en Jupyter Notebook y su posterior resolución mediante Gurobi utilizaremos un ejemplo sencillo.

El problema será el siguiente:

- Actividades de decisión.
  - X: variable binaria
  - Y: variable binaria
  - Z: variable binaria
- Restricciones.
  - C1: $X + 2*Y + 3*Z \leq 4$
  - C2: $X + Y \geq 1$
- Función Objetivo.
  - FO = MAX $(X + Y + 2*Z)$

<u>Modelo completo</u>

MAX $(X + Y + 2*Z)$

s.a.

$X + 2*Y + 3*Z \leq 4$

$X + Y \geq 1$

X, Y, Z binarias

Implementación en Jupyter Notebook

```
In [5]: from gurobipy import*

In [6]: m = Model()

In [7]: x=m.addVar(vtype=GRB.BINARY, name="x")
        y=m.addVar(vtype=GRB.BINARY, name="y")
        z=m.addVar(vtype=GRB.BINARY, name="z")

In [8]: m.setObjective(x + y + 2*z, GRB.MAXIMIZE)

In [9]: c1 = m.addConstr(x + 2*y + 3*z <= 4)
        c2= m.addConstr(x + y >= 1)

In [10]: m.optimize()
```

Ilustración 21Creación de un problema en Jupyter Notebook

Fuente: Elaboración propia

El primer paso para crear un modelo es importar el paquete Gurobi. Esto se hace con la primera sentencia. En segundo lugar, creamos un modelo vacío, el cual hemos llamado m. Una vez hemos creado el modelo, podemos hacer uso de los métodos que tiene la clase Model. Por tanto, creamos las variables necesarias para nuestro problema. Esto se hace mediante el método addVar. Como argumento de este método necesitamos el tipo de variable, que en este caso es binaria, y un nombre. Hemos creado 3 variables binarias: x,y,z.

Una vez definidas las variables, utilizamos el método setObjective para establecer la función objetivo. Como argumentos serán necesarios la expresión matemática de la función y el objetivo, es decir, maximizar o minimizar.

Por último, añadimos las restricciones de nuestro problema. En este caso hemos añadido 2, C1 y C2. Utilizamos el método addConstr e introducimos la expresión matemática que caracteriza la restricción.

Finalmente tenemos los elementos necesarios de un problema de optimización y llamamos a Gurobi para su resolución mediante el método optimize, el cual no necesita argumento.

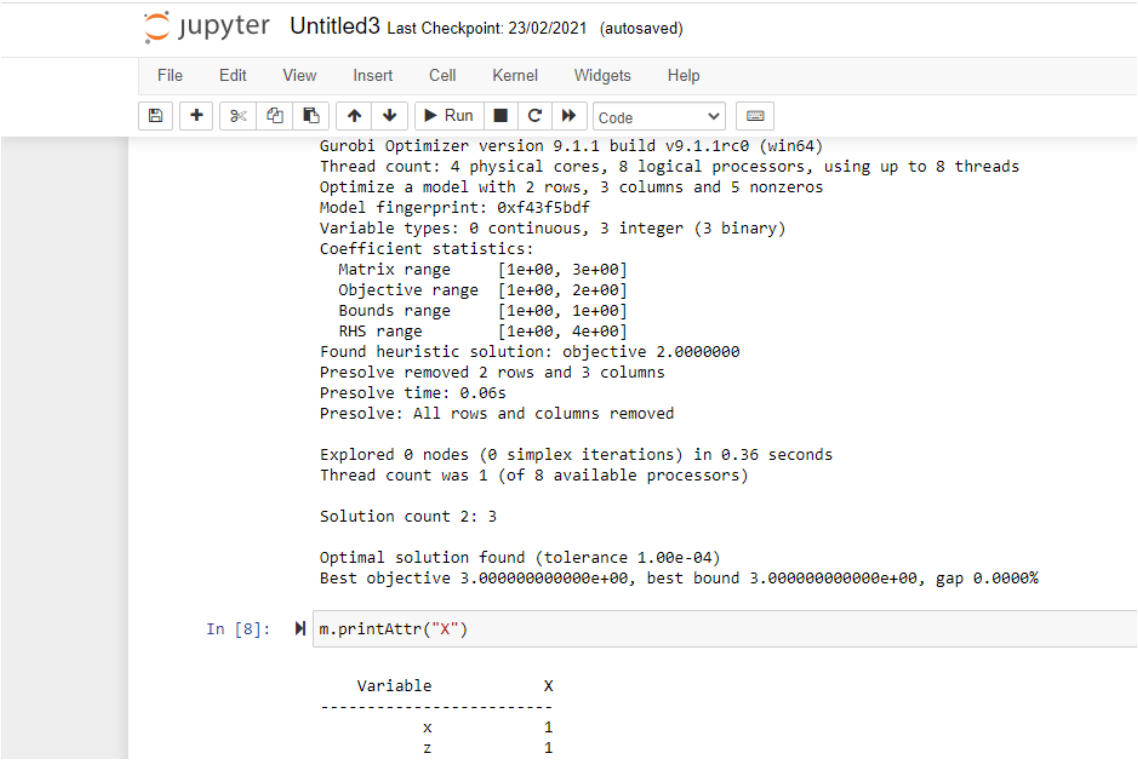


Ilustración 22 Optimización con Gurobi en Jupyter Notebook

Fuente: Elaboración propia

Al utilizar el método optimize podemos ver la solución con el mismo formato de la pantalla interactiva de explicada en el apartado de Gurobi. Además, utilizando el método printAttr veremos el valor de las variables mayores que 0.

Además de esta forma, Python y Jupyter Notebook también permiten modelar problemas de forma matricial para los usuarios que estén más familiarizados con esta manera. Ambas formas de crear un problema son utilizadas para modelos muy sencillos. A la hora de trabajar con modelos con grandes cantidades de datos, variables y restricciones, la forma de generar el modelo será distinta. Utilizaremos los elementos nombrados en el apartado de Python, es decir, listas, tuplas y diccionarios.

Veremos con un ejemplo a continuación cómo crear problemas más complejos. El problema seleccionado ha sido obtenido de la plataforma de Gurobi, donde hay bastantes ejemplos clasificados en función del nivel del usuario. Concretamente, es el primer ejemplo. Para demostrar la potencia de Jupyter Notebook a la hora de desarrollar modelos matemáticos y su posterior resolución será suficiente con este problema.

2.3.5. Modelos complejos con Jupyter Notebook y Gurobi

En primer lugar, comenzaremos describiendo el problema. Una empresa cuenta con 3 puestos de trabajo libres, los cuales son: probador, desarrollador de Java y arquitecto. A su vez, para cubrir los distintos puestos la empresa tiene a su disposición a 3 candidatos: Carlos, Joe y Monika. Cada uno de los candidatos tiene una puntuación respecto a cada trabajo que va desde 0 a 100.

The ability of each resource to perform each of the jobs is listed in the following matching scores table:

Matching Scores	Tester	Java Developer	Architect
Carlos	53%	27%	13%
Joe	80%	47%	67%
Monika	53%	73%	47%

For each resource r and job j, there is a corresponding matching score s. The matching score s can only take values between 0 and 100. That is, $s_{r,j} \in [0, 100]$ for all resources $r \in R$ and jobs $j \in J$.

Ilustración 23 puntuaciones entre recursos y trabajos

Fuente: Gurobi.com

El objetivo del problema será el de asignar a cada puesto de trabajo un candidato intentando maximizar la suma de las puntuaciones. Vamos a expresar el modelo matemáticamente.

- Tabla de elementos

ELEMENTOS	**CJTO**	**NC**	**DATOS**				
			Nombre	**Parámetro**	**Tipo**	**Pertenencia**	**Valor**
Recursos	r=1..3	I_u	Afinidad	S_{rj}	C	C	-
Trabajos	J=1..3	I_u		S_{rj}	C	C	-

Tabla 3 Tabla de elementos

- Actividades de decisión
  - $X_{jr.}$ Es una variable binaria que relaciona a cada recurso con cada trabajo. Valdrá 1 si el trabajador es asignado al trabajo y 0 en caso contrario.
- Restricciones
  - Asociada a los trabajos: $\sum_{r \in R} X_{jr} = 1$
  - Asociada a los recursos: $\sum_{j \in J} X_{jr} \leq 1$
- Función objetivo.
  - FO = $MAX(\sum_{j \in J} \sum_{r \in R} Srj * X_{jr})$

Modelo completo

$MAX(\sum_{r \in R} \sum_{j \in J} Srj * X_{jr})$

s.a.

$\sum_{r \in R} X_{jr} = 1$

$\sum_{j \in J} X_{jr} \leq 1$

X_{jr} binaria.

Una vez tenemos los datos del problema, podemos pasar crear el modelo en Jypyter Notebook. Para hacer uso de todas las clases de Gurobi tenemos que importarlas con el primer comando que vemos en la imagen. En la segunda celda hemos inicializado dos listas, una para los recursos o trabajadores, y otra para los trabajos. Por último, hemos inicializado un diccionario llamado scores cuya clave será la combinación entre trabajador y trabajo y el valor será la puntuación que tiene ese trabajador para ese trabajo.

Gracias a importar las clases y métodos de Gurobi podemos usar la función multidict. Esta función recibe como parámetro un diccionario y permite inicializar más de un diccionario de forma más rápida, además de permitir crear listas. Para este ejemplo hemos inicializado una lista llamada combinations donde cada elemento de la lista será cada clave del diccionario. Esto nos será útil a la hora de crear variables, restricciones y establecer la función objetivo.

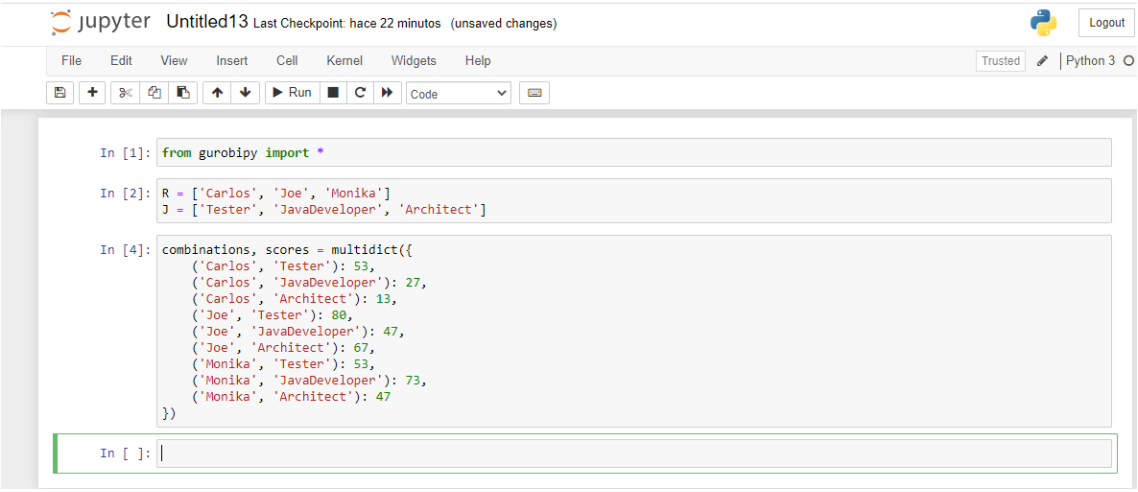


Ilustración 24 Creación de datos en Jupyter Notebook

Fuente: Elaboración propia

El siguiente paso consiste en crear un modelo vacío, al que hemos llamado m. En Gurobi los modelos son objetos que pertenecen a la clase Model. Cada modelo será distinto, pero todos tienen acceso a los mismos métodos. Uno de estos métodos es addVars. Como explicamos en la descripción del problema, tenemos que asignar un trabajador a cada puesto. Por tanto, las variables a utilizar serán de tipo binario, es decir, la actividad de decisión será decidir si el trabajador X es asignado al trabajo Y. Esta variable valdrá 1 en caso afirmativo y 0 en caso contrario. Hemos declarado una variable para cada posible combinación utilizando la lista combinations inicializada anteriormente.

En cuanto al método addVars, recibe como argumentos la lista combinations, así que cada elemento de esa lista tendrá una variable asociada. Además, declaramos el tipo de variable, en este caso binaria y asignamos un nombre a la variable, llamada assign. El método utilizado devuelve una tupledict, es decir un diccionario cuyas claves están formadas por tuplas y el valor es la variable en sí. Esto es fácilmente comprobable mediante el uso de la función print(x), siendo x una variable donde hemos guardado la tupledict devuelta por el método.

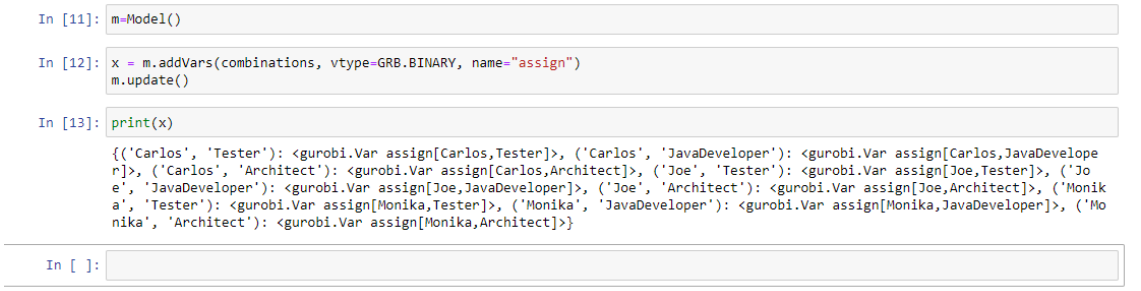


Ilustración 25 Creación de un modelo y variables con Jupyter Notebook y Gurobi

Fuente: Elaboración propia

En la siguiente imagen veremos la construcción de las restricciones del problema. En la descripción del problema no vienen de forma explícita, pero es obvio deducir que para cada trabajo sólo podemos asignar un trabajador. Ésta es la primera de las dos restricciones.

Para realizar esta tarea haremos uso de otro método perteneciente a la clase Model, addConstrs. Como argumento recibe la expresión matemática y un nombre cualquiera. Centrándonos en la expresión matemática, para construirla también hemos utilizado un método, el sumatorio o sum, aplicado a el conjunto de las variables que habíamos guardado en x.

El funcionamiento es el siguiente: mediante un bucle for, recorremos la lista J donde tenemos asignados los distintos tipos de trabajo. Para el primer trabajo Tester, hacemos que el sumatorio de las variables que relacionan a los trabajadores con el trabajo Tester sea igual a 1. Esto es así porque el método sum nos permite hacer referencia a una de las dos componentes de la clave únicamente. En la creación de las variables vimos que para acceder a una variable tenemos que hacer referencia a la clave, que en este caso es una tupla formada por trabajador y trabajo. Mediante el * indicamos que cualquier valor para la primera componente de la clave es válido, es decir, cualquier trabajador, mientras que con el bucle for recorremos cada trabajo. Por tanto, tendremos una restricción para cada uno de los tres trabajos. Comprobamos esto mediante la función print(jobs) y vemos que para cada trabajo hemos incluido una restricción.

Respecto a la segunda restricción implícita, cada trabajador tendrá asignado un trabajo como máximo. El proceso utilizado para crear estas restricciones ha sido el mismo cambiando la lista que recorre el bucle for, en este caso, la lista de recursos o trabajadores, y la componente de la clave sobre la que iteramos.

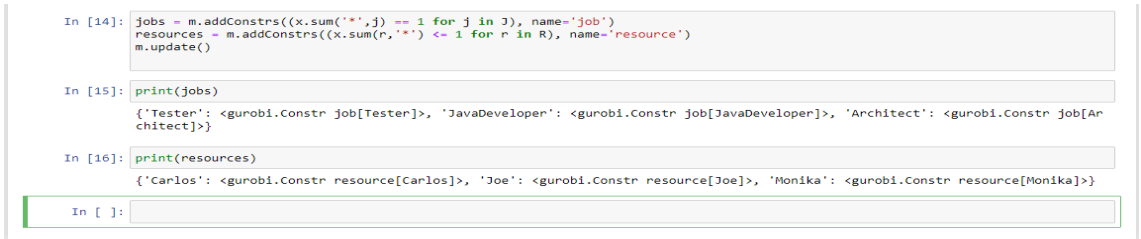


Ilustración 26 Creación de restricciones con Jupyter Notebook y Gurobi

Fuente: Elaboración propia

Finalmente, el último paso consiste en definir la función objetivo. En la descripción del problema el objetivo era maximizar la suma de puntuaciones que tiene cada trabajador con cada trabajo. El método para establecer una función objetivo es setObjective, que recibe como argumentos la expresión matemática y el objetivo, ya sea maximizar o minimizar.

Para crear la función objetivo hemos utilizado el método prod. El funcionamiento de este método es el siguiente: hacemos un sumatorio del producto de cada variable por la puntuación asociada a esa variable. Esta asociación se realiza de forma automática gracias a las estructuras avanzadas de Python de las que hemos hecho uso. Podemos comprobar fácilmente que tanto el diccionario score como el diccionario donde hemos guardado las variables comparten misma estructura de clave. Esta clave es una tupla cuyo primer componente es trabajador y segunda componente trabajo. Por tanto, cada variable será multiplicada una única vez por la puntuación que le corresponde, es decir, cuando coincidan ambas claves.

Una vez creado el modelo, llamamos al método optimize y obtenemos la solución a nuestro problema. En este caso el valor de la función objetivo es 193.

```
In [27]: m.setObjective(x.prod(scores), GRB.MAXIMIZE)
         m.update()

In [28]: m.optimize()

         Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (win64)
         Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
         Optimize a model with 6 rows, 9 columns and 18 nonzeros
         Model fingerprint: 0x8e59071a
         Variable types: 0 continuous, 9 integer (9 binary)
         Coefficient statistics:
           Matrix range     [1e+00, 1e+00]
           Objective range  [1e+01, 8e+01]
           Bounds range     [1e+00, 1e+00]
           RHS range        [1e+00, 1e+00]
         Presolve time: 0.09s
         Presolved: 6 rows, 9 columns, 18 nonzeros
         Variable types: 0 continuous, 9 integer (9 binary)
         Found heuristic solution: objective 193.0000000

         Root relaxation: cutoff, 0 iterations, 0.02 seconds

         Explored 0 nodes (0 simplex iterations) in 0.40 seconds
         Thread count was 8 (of 8 available processors)

         Solution count 1: 193

         Optimal solution found (tolerance 1.00e-04)
         Best objective 1.930000000000e+02, best bound 1.930000000000e+02, gap 0.0000%
```

Ilustración 27 Creación de la fo y optimización con Jupyter Notebook y Gurobi

Fuente: Elaboración propia

Una vez explicado el proceso de creación de un problema más complejo en Jupyter Notebook y su posterior resolución con Gurobi, queda demostrada la potencia de este lenguaje por su sencillez y posterior implementación. Podemos observar que la parte más importante en la construcción del modelo es la de incluir los datos del problema y que una correcta utilización de las estructuras como listas, tuplas y diccionarios nos permite desarrollar cualquier modelo de forma veloz y eficiente.

3. EL PROBLEMA DE STEINER

3.1. Introducción a los grafos

La metodología propuesta para la introducción a los grafos ha sido obtenida de [4]. Un grafo G se define como un par (V, E), donde V es un conjunto cuyos elementos son denominados vértices o nodos y E es un subconjunto de pares no ordenados de vértices y que reciben el nombre de aristas o arcos.

Si V = {v1, · · ·, vn}, los elementos de E se representan de la forma {vi, vj}, donde i ≠ j. Los elementos de una arista o arco se denominan extremos de dicha arista.

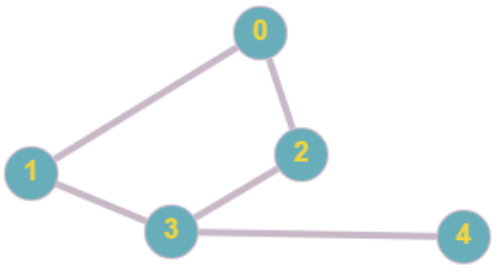


Ilustración 28 Ejemplo de grafo

Fuente: Elaboración propia

En la imagen mostrada podemos ver un grafo cualquiera. Este grafo consta de 5 vértices y 5 aristas. Para este caso V = {0, 1, 2, 3, 4} y E = {(0, 1), (0, 2), (1, 3), (2, 3), (3, 4)}.

3.2. Clasificación de grafos

Dentro de la teoría de grafos podemos encontrar distintas formas de clasificar un grafo. En nuestro caso para el desarrollo de este trabajo nos hemos enfocado en dos formas de clasificación. En primer lugar, nos centraremos en si un grafo es dirigido o no dirigido. La otra tipología de clasificación consistirá en comprobar si se trata de un grafo conexo o no conexo. Estas tipologías de grafo se explicarán a lo largo de este apartado.

3.2.1. Grafo dirigido

Un grafo G dirigido consiste en 2 conjuntos:

- V(G): un conjunto finito no vacío cuyos elementos son llamados vértices de G.
- E(G): un conjunto de pares ordenados de vértices llamados aristas de G. Por la definición de pares ordenados se tiene que el orden en el que son listados los vértices indica la dirección de la arista. Esto significa que dados 2 vértices u y v, la arista (u,v) y la arista (v,u) no son equivalentes, es decir, (u,v) ≠ (v,u). Cuando nos referimos a los pares ordenados de vértices como arcos estamos indicando que se trata de un grafo dirigido.

Los grafos dirigidos también pueden contar con aristas del tipo (v,v). Este tipo de aristas se conocen como lazos. Además, puede darse el caso de que existan más de una arista del tipo (u,v), más conocidas como aristas múltiples o aristas paralelas. Un grafo dirigido que admite aristas múltiples es conocido como multigrafo. Por tanto, cualquier grafo dirigido es un caso especial de multigrafo sin aristas múltiples.

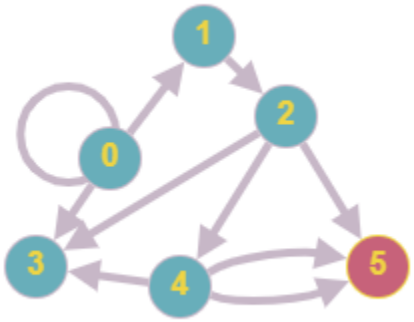


Ilustración 29 Ejemplo de grafo dirigido

Fuente: Elaboración propia

En la imagen mostrada podemos observar un multigrafo en el que también hay lazos. Para la realización del trabajo nos centraremos en los grafos dirigidos simples, esto es, un grafo dirigido libre de lazos.

3.2.2. Grafo no dirigido

Un grafo G para el cual se tiene que las aristas (u,v) y (v,u) son equivalentes, es decir (u,v) = (v,u), es llamado grafo no dirigido. Esto implica que para este tipo de grafos es válido movernos del vértice u al vértice v o del vértice v al vértice u pasando por la arista (u,v). En otras palabras, la arista (u,v) se asume como bidireccional. Un grafo no dirigido libre de lazos se denomina grafo simple. Cuando llamemos a los pares ordenados de vértices aristas nos referimos a un grafo no dirigido.

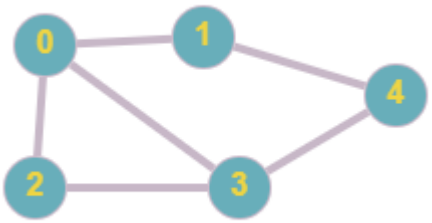


Ilustración 30 Ejemplo de grafo no dirigido

Fuente: Elaboración propia

3.2.3. Grafo conexo

Un grafo se denomina conexo cuando todos los vértices que forman dicho grafo están conectados por un camino (si es un grafo no dirigido) o por un semicamino (si es un grafo dirigido). El grafo no dirigido mostrado en la anterior imagen es conexo por la definición.

Para los grafos dirigidos tendremos distintos tipos de conectividad:

- Grafo débilmente conexo: todos los pares de vértices están débilmente conectados, es decir, unidos por un «semicamino» (camino que no considera la dirección de las aristas). Un ejemplo de grafo dirigido débilmente conexo es el mostrado en la imagen siguiente. Desde el vértice 0 no podemos tomar ningún camino hacia otro vértice que no sean el vértice 1 o el vértice 2. La única manera es despreciando las direcciones.

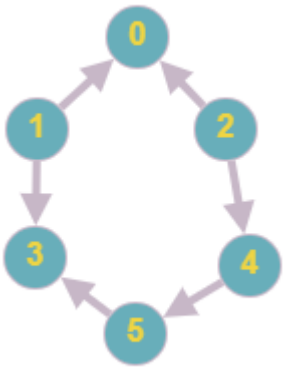


Ilustración 31 Ejemplo de grafo dirigido débilmente conexo

Fuente: Elaboración propia

- Grafo unilateralmente conexo: todos los pares de vértices están unilateralmente conectados, es decir, unidos por un camino que va desde un vértice hasta otro. En la imagen siguiente veremos que para ir de un vértice a otro solo tenemos Un camino. El único vértice que no cumple esta característica es el 0, que puede ir al vértice 4 por dos caminos distintos.

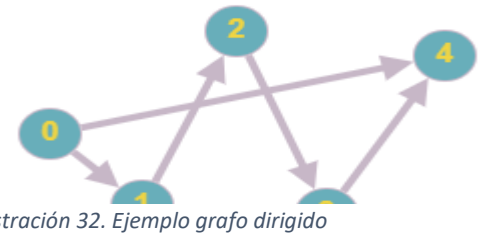


Ilustración 32. Ejemplo grafo dirigido unilateralmente conexo

Fuente: Elaboración propia

- Grafo fuertemente conexo: todos los pares de vértices están fuertemente conectados, es decir, unidos por al menos dos caminos, uno que va desde uno hasta el otro, y viceversa. Podemos comprobarlo en la siguiente imagen. A modo de ejemplo, podemos ir desde el vértice 0 hasta el 4 y también podemos ir desde el 4 hasta el vértice 0. Esto tiene que suceder con todos los vértices.

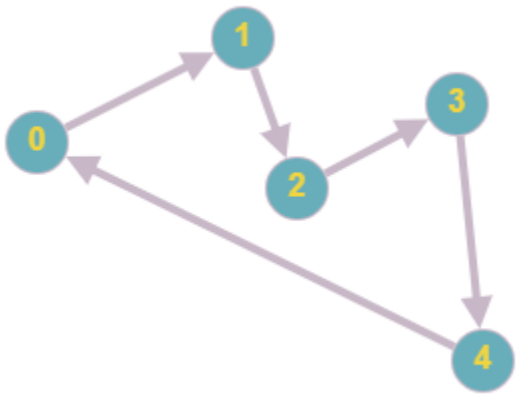


Ilustración 33 Ejemplo Grafo dirigido fuertemente conexo

Fuente: Elaboración propia

- Grafo recursivamente conexo: todos los pares de vértices están recursivamente conectados, es decir, están fuertemente conectados y el camino desde uno hasta el otro usa los mismos vértices y aristas que los del camino inverso.

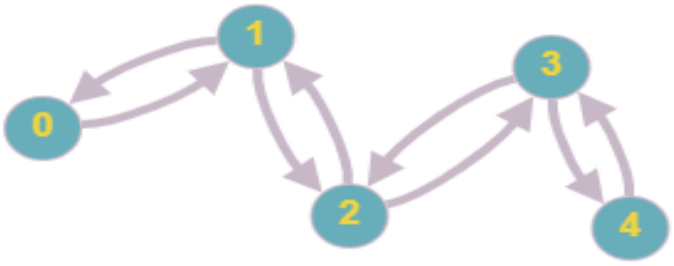


Ilustración 34 Ejemplo grafo dirigido recursivamente conexo

Fuente: Elaboración propia

3.2.4. Grafo no conexo

Una vez se ha explicado las condiciones de conectividad, un grafo no conexo será todo aquel que no cumpla ninguna de estas condiciones. Respecto a un grafo no dirigido, éste será no conexo cuando alguno de los vértices que lo componen no tenga ningún camino hacia alguno de los otros vértices. En la siguiente imagen veremos un ejemplo.

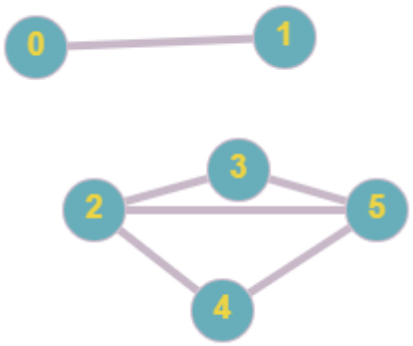


Ilustración 35 Ejemplo grafo no dirigido no conexo

Fuente: Elaboración propia

Podemos observar que no existe ningún camino entre el vértice 0 y cualquiera de los demás vértices que no sean el vértice 1. Lo mismo ocurre para el vértice 1.

En cuanto a los grafos dirigidos, siempre que no se cumpla la condición mínima de conectividad, es decir, que no se trate de un grafo débilmente conexo, será un grafo dirigido no conexo. Podemos observar un grafo dirigido no conexo en la siguiente imagen.

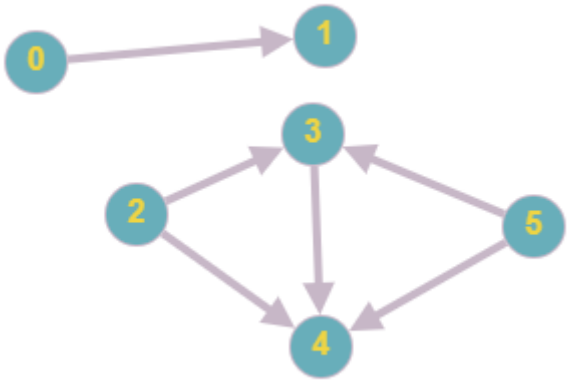


Ilustración 36 Ejemplo grafo dirigido no conexo

Fuente: Elaboración propia

Al no haber ningún semicamino posible entre el vértice 0 o el vértice 1 y los demás vértices se trata de un grafo dirigido no conexo.

3.3. Propiedades de los grafos

En este apartado veremos las propiedades más importantes de los grafos. Algunas de estas nos serán de gran utilidad a la hora de resolver el problema de Steiner.

- Adyacencia: dos vértices son adyacentes si tienen una arista en común. En la imagen siguiente podemos ver que el vértice 0 y el vértice 1 son adyacentes por la arista (0, 1). A su vez, dos aristas son adyacentes si tiene un vértice en común. En la misma imagen vemos que las aristas (0, 1) y (1, 2) son adyacentes al tener en común el vértice 1.

Ilustración 37 Ejemplo propiedad de adyacencia

Fuente: Elaboración propia

- Incidencia: una arista es incidente a un vértice si ésta lo une a otro vértice. Tomando como ejemplo la misma imagen, la arista (0, 1) es incidente al vértice 0 al unirlo con el vértice 1. Por la definición anterior, la misma arista también es incidente al vértice 1 al conectar a éste con el vértice 0. Por tanto, una arista puede ser incidente como máximo a dos nodos o vértices.
- Ponderación: Esta propiedad nos será de gran utilidad a la hora de resolver problemas de grafos. Consiste en adjudicar valores (coste, peso, longitud, etc.) a las aristas de un grafo. Esta adjudicación nos permitirá distinguir entre las distintas aristas y seleccionar aquellas cuyo valor satisfaga nuestro objetivo.

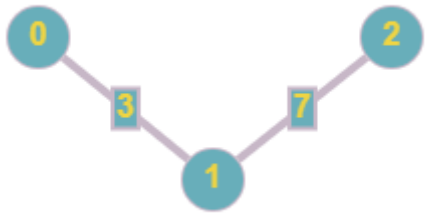


Ilustración 38 Ejemplo propiedad de ponderación

Fuente: Elaboración propia

- Etiquetado: Consiste en seleccionar un nodo o arista y marcarlo para diferenciarlo del resto de nodos o aristas. En la resolución del problema de Steiner veremos su utilización.
- Ciclos: un ciclo es un camino en el que partimos de un vértice inicial y volvemos a él utilizando cada arista una única vez. En la siguiente imagen podemos comprobar que partiendo del vértice 0, podemos volver al mismo utilizando cada arista una única vez. Por tanto, este grafo es dirigido y cíclico. Dentro de los grafos puede haber subgrafos que formen ciclos. Estos ciclos pueden crear bucles e ineficiencias a la hora de resolver problemas de grafos, así que es una característica poco deseada y que intentaremos corregir al optimizar modelos.

  Durante la resolución del problema de Steiner veremos algunos métodos para evitar estos ciclos.

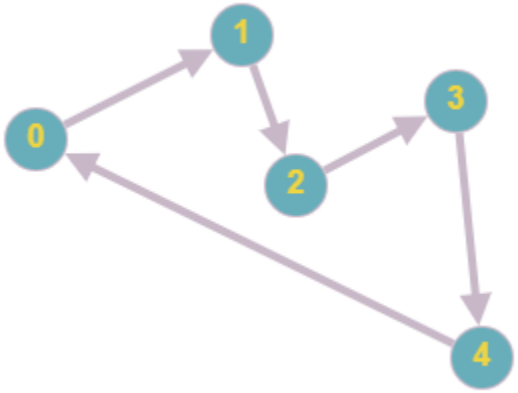


Ilustración 39 Ejemplo ciclo

Fuente: Elaboración propia

3.4. El problema de Steiner

Jacob Steiner, matemático suizo del siglo XIX, formuló un problema de redes de gran importancia en nuestros días. Es por esto por lo que el problema utilizado para el desarrollo del trabajo lleva su nombre en su honor, el problema de Steiner. Dicho problema consta de una serie de nodos llamados terminales y nodos Steiner. Estos nodos, tanto terminales como Steiner, están conectados entre ellos por aristas, cada una con un coste. El objetivo del problema es conectar todos los nodos terminales a mínimo coste, pudiendo utilizar los nodos Steiner que creamos oportuno.

Matemáticamente, tenemos un grafo no dirigido G = (V, E) donde V indica el conjunto de nodos y E el conjunto de aristas que conectan dichos nodos. Tenemos N nodos terminales, $N \in V$, y S nodos Steiner, $S \in V$. Respecto al conjunto de aristas, cada una tendrá un coste $C > 0$.

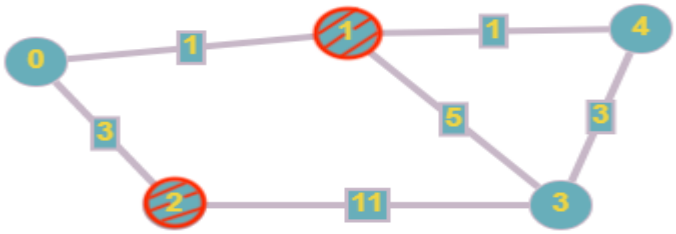


Ilustración 40 Ejemplo representación gráfica problema de Steiner

Fuente: Elaboración propia

A modo de ejemplo, imaginemos que el grafo mostrado es un problema de Steiner. Tenemos un grafo no dirigido G = (5, 6), donde los nodos terminales son 0, 3 y 4, y los nodos Steiner son 1 y 2. También podemos ver el coste de cada arista. Para resolver el problema necesitamos conectar los nodos terminales. Una posible solución sería seleccionar utilizar el nodo Steiner 1 para conectar los nodos terminales. Utilizamos las aristas (0, 1), (1, 4) y (3,4) con sus respectivos costes 1, 1 y 3, lo que daría un total de 5.

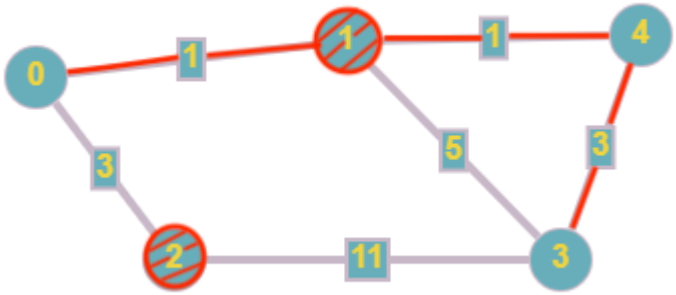


Ilustración 41 Ejemplo representación gráfica solución problema de Steiner

Fuente: Elaboración propia

Al ser un problema de tamaño reducido es bastante simple obtener el óptimo, que es la solución que hemos elegido. Los problemas de Steiner que resolveremos en este trabajo serán mucho más complejos y para su resolución tendremos que contar con el software para la resolución de problemas de optimización Gurobi. Es gracias a la aparición de esta clase de softwares que problemas como el de Steiner han tomado una gran importancia en nuestros días por la rapidez y eficacia con la que podemos resolverlos.

3.5. Aplicaciones del problema de Steiner

Durante el último siglo, debido a los importantes avances en capacidad de computación, el ser humano ha intentado abordar problemas de distinta índole consiguiendo resultados

que antaño hubieran resultado imposibles de plantear. Es aquí donde entra el problema de Steiner debido a su facilidad para ser aplicado en distintos campos. Veremos a continuación algunos de estos campos y la utilización del problema de Steiner para resolver problemas cuya solución es muy importante para mantener nuestra forma de vida.

- Infraestructuras. El ámbito donde más ha sido utilizado el problema de Steiner es en la optimización de caminos. Aunque es creciente su uso en los distintos campos que explicaremos, siguen siendo muy importante en la conexión de ciudades, regiones, países, etc. Algunos ejemplos de la aplicación del problema de Steiner en las infraestructuras son las redes ferroviarias, las redes eléctricas y las redes de minas subterráneas.
- Redes de telecomunicaciones. Como decíamos en la introducción, los avances en el último siglo, donde se ha pasado de la revolución industrial a la era de la información, han provocado un crecimiento espectacular en el campo de las telecomunicaciones. Algunos ejemplos de la aplicación del problema de Steiner en este ámbito son la construcción de redes de fibra óptica o la localización de nodos de red inalámbrica, con el objetivo de minimizar costes, energía o mejorar la conexión.
- Diseño de circuitos integrados. La evolución de los circuitos integrados es una de las principales causas del avance exponencial de la tecnología en los últimos tiempos. La aplicación del problema de Steiner en circuitos integrados es crucial en los problemas VLSI (Very Large Scale Integration). Este problema trata de combinar millares de módulos, en su mayoría transistores, dentro de un chip. Cada vez se necesitan procesadores más pequeños y potentes y es necesario el uso del problema de Steiner para optimizar la colocación de estos módulos.
- Redes biológicas. Aunque es uno de los campos con más potencial en el futuro y todavía no se ha aplicado el problema de Steiner tan a menudo como en los demás, hoy en día ya se han conseguido grandes resultados en experimentos de estructura molecular y redes neuronales.

4. MODELOS MATEMÁTICOS PARA EL PROBLEMA DE STEINER

En este apartado veremos los modelos matemáticos utilizados para resolver el problema de Steiner en este trabajo. En el apartado anterior vimos cómo resolver este tipo de problemas de manera gráfica. Esto es posible para aquellos que cuenten con pocos elementos, es decir, pocos nodos terminales, nodos Steiner y aristas. A la hora de resolver problemas de Steiner más complejos, como los que hemos resuelto a lo largo de este trabajo, necesitaremos modelar el problema matemáticamente para obtener una solución.

En cuanto a los modelos matemáticos utilizados para resolver el problema de Steiner, hemos hecho uso de dos, el modelo MTZ (Miller, Tucker y Zennit) y la variante de este, llamada DL (Desrochers y Laporte). Ambos modelos utilizan mismos elementos y variables, pero difieren en la restricción matemática para evitar bucles.

4.1. Modelado

Las versiones que vamos a implementar son de tipo arborescentes, cambiamos las aristas por arcos para seleccionar un árbol de un grafo dirigido. Es necesario identificar un nodo terminal como nodo raíz del árbol.

Creamos un grafo dirigido G (N, A) donde de cada arista del grafo no dirigido original se generan dos arcos:

Arista (i,j) $\Rightarrow$ Arco (i,j)$\in$A; Arco (j,i) $\in$A

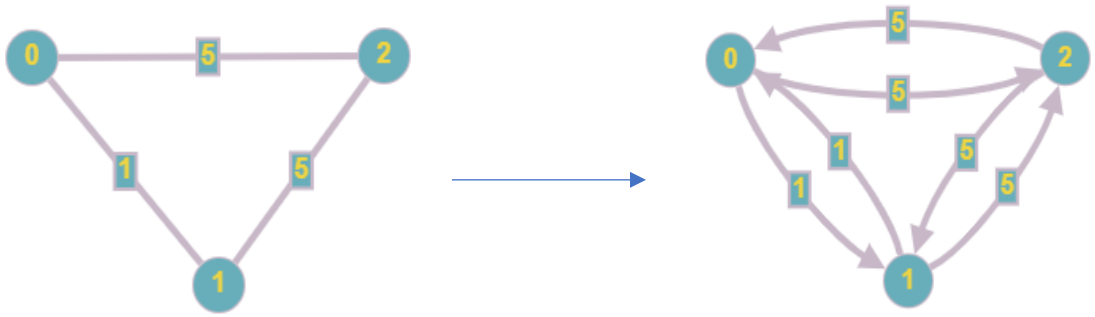


Ilustración 42 Ejemplo paso de grafo no dirigido a grafo dirigido

Fuente: Elaboración propia

Realizamos también un cálculo de datos, el número de arcos entrantes (NE_i)de cada nodo, pues servirá de cota superior en el proceso de modelado

En la estructura arborescente, la decisión principal es la selección de arcos, donde las especificaciones controlarán que tengamos una estructura en árbol de esos arcos. Sin

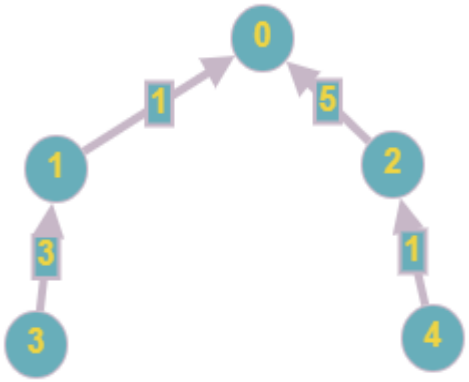
embargo, el modelo necesita también una decisión sobre la profundidad que va a dar cada nodo del árbol. Las variables de profundidad nos servirán para controlar una especificación necesaria para crear un árbol, la no existencia de ciclos.

Aunque la profundidad podría llegar a ser un cálculo lógico no binario en otros problemas de creación de árboles, en el problema de Steiner lo implementamos como actividad de decisión puesto que existe un intervalo para la decisión de la profundidad de cada nodo, ya que no sabemos cuántos nodos formarán parte del árbol.

4.2. Especificaciones

- Todo nodo terminal, excepto el nodo raíz, debe poseer un nodo padre.
- Cada nodo Steiner seleccionado (que es padre de algún nodo), debe poseer un padre.
- La profundidad mínima de un nodo distinto al nodo raíz es 1.
- La profundidad máxima de un nodo distinto al nodo raíz es n-1.
- Especificación anti-bucles:
    - Modelo MTZ: Si seleccionas el arco (i,j) la profundidad de i es mayor que la de j (al menos una unidad más).
      Gracias a esta especificación y a las dos últimas especificaciones, el nodo raíz acabará tomando profundidad 0 y no es necesario considerar esta especificación anti-bucle en los arcos en los que aparece dicho nodo. Si se considera no pasa nada, pero podemos ahorrar restricciones.

    - Modelo DL: Si seleccionas el arco (i,j) o bien el arco (j,i) la profundidad de i más la selección del arco (j,i) es igual a la profundidad de j más la selección del arco (i,j). Ocurre con el nodo raíz lo mismo que en el caso del modelo MTZ.



Nodo raíz, profundidad = 0. El nodo raíz es padre de los nodos 1 y 2

Nodos hijos del nodo raíz, profundidad = 1. El nodo 1 es padre del nodo 3 y el nodo 2 del nodo 4

Nodos hijos, profundidad > 1

Ilustración 43 Ejemplo nodo raíz, padres, hijos y profundidad

Fuente: Elaboración propia

4.3. Tabla de elementos

Para este problema, por comodidad en la identificación, vamos a identificar los arcos con dos índices ordenados (i,j), siendo i e j nodos.

ELEMENTOS	CJTO	NC	DATOS				
			Nombre	**Parámetro**	**Tipo**	**Pertenencia**	**Valor**
Nodos	i,j=1..n	I_M	Terminal	T_i	B	P	-
			Raíz	R_i	B	P	-
			Nº Arcos	NE_i	E	P	-
Arcos	(i,j)	I_U	Coste	C_{ij}	C	P	-

Tabla 4 Tabla de elementos problema de Steiner

4.4. Actividades de decisión

Seleccionar Arcos

$\forall (i,j) \in A : \alpha_{ij} = 1$ si selecciono arco (i,j); 0 si no

Asignar profundidad a Nodos

$\forall i : x_i =$ profundidad del nodo i

4.5. Modelado de especificaciones

El modelado del problema sigue el formato de la metodología descrita en [1]

- **Todo nodo terminal tiene un padre, excepto el nodo raíz**

$$\forall i / T_i = 1 \,\&\, R_i = 0 : \sum_{j/(i,j)\in A} \alpha_{ij} = 1$$

- **Cada nodo Steiner seleccionado (que es padre de algún nodo), debe poseer un padre:**

En primer lugar, declaramos un cálculo lógico binario para saber si cada Steiner es seleccionado:

CÁLCULO LÓGICO BINARIO

$$\forall j / T_j = 0 : \beta_j = 1 \; sss \sum_{i/(i,j)\in A} \alpha_{ij} > 0$$

$$\Rightarrow f_{S_V} \Rightarrow \forall j / T_j = 0 : SI \sum_{i/(i,j)\in A} \alpha_{ij} > 0 \; ENTONCES \; \beta_i = 1$$

$$\Rightarrow f_3, f_7 \Rightarrow \forall j / T_j = 0 : SI \; 1 - \beta_i = 1 \; ENTONCES \sum_{i/(i,j)\in A} \alpha_{ij} \le 0$$

$$\Rightarrow f_{14} \left(UB_{\sum_{i/(i,j)\in A} \alpha_{ij}} = NE_j \right) \Rightarrow \forall j / T_j = 0 : \sum_{i/(i,j)\in A} \alpha_{ij} \le NE_j \beta_i$$

Especificación: Cada Nodo Steiner seleccionado debe poseer un padre:

$$\forall i / T_i = 0 : SI\ \beta_i = 1\ ENTONCES \sum_{j/(i,j)\in A} \alpha_{ij} = 1$$

$$\Rightarrow f_{UB} \Rightarrow \forall i / T_i = 0 : SI\ \beta_i = 1\ ENTONCES \sum_{j/(i,j)\in A} \alpha_{ij} \geq 1$$

$$\Rightarrow f_{15} \Rightarrow \forall i / T_i = 0 : \sum_{j/(i,j)\in A} \alpha_{ij} \geq \beta_i$$

- **Profundidad máxima:**
  $\forall i / R_i = 0 : x_i \leq n-1$
- **Profundidad mínima:**
  $\forall i / R_i = 0 : x_i \geq 1$
- **Especificación Antibucles.**
  Modelo MTZ:

  $\forall (i,j) / R_i = 0 \,\&\, R_j = 0 : Si\ \alpha_{ij} = 1\ entonces\ x_i \geq x_j + 1$

  $\forall (i,j) / R_i = 0 \,\&\, R_j = 0 : Si\ \alpha_{ij} = 1\ entonces\ x_i - x_j \geq 1$

  $\Rightarrow f_{15}\left(LB_{x_i - x_j} = -(n-2)\right) \Rightarrow \forall (i,j) / R_i = 0 \,\&\, R_j = 0 : x_i - x_j \geq \alpha_{ij} - (n-2)(1-\alpha_{ij})$

  $\Rightarrow \forall (i,j) / R_i = 0 \,\&\, R_j = 0 : x_j - x_i + (n-1)\alpha_{ij} \leq (n-2)$

  Modelo DL:

  Se aplica una condición sobre cada arista. Como el problema está configurado con arcos, lo indicamos como $\forall (i,j) / i < j$ **.**

  $\forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 : SI\ \alpha_{ij} + \alpha_{ji} = 1\ ENTONCES\ x_i + \alpha_{ji} = x_j + \alpha_{ij}$

  $\forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 : SI\ \alpha_{ij} + \alpha_{ji} = 1\ ENTONCES\ x_i + \alpha_{ji} - x_j - \alpha_{ij} = 0$

  Asume $\alpha_{ij} + \alpha_{ji}$ como expresión binaria, por lo que aplicamos f_{16}:

$\Rightarrow f_{16}\left(UB_{x_i + \alpha_{ji} - x_j - \alpha_{ij}} = n-2\right) \Rightarrow \forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 :$

$x_i + \alpha_{ji} - x_j - \alpha_{ij} \leq (n-2)(1-\alpha_{ij} - \alpha_{ji})$

$\Rightarrow \forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 : x_i + \alpha_{ji} - x_j - \alpha_{ij} \leq (n-2) - (n-2)\alpha_{ij} - (n-2)\alpha_{ji}$

$\Rightarrow \forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 : x_i - x_j \leq (n-2) - (n-3)\alpha_{ij} - (n-1)\alpha_{ji}$

$\Rightarrow \forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 : x_i - x_j + (n-3)\alpha_{ij} + (n-1)\alpha_{ji} \leq (n-2)$

$\Rightarrow f_{16}\left(LB_{x_i + \alpha_{ji} - x_j - \alpha_{ij}} = -(n-2)\right) \Rightarrow \forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 :$

$x_i + \alpha_{ji} - x_j - \alpha_{ij} \geq -(n-2)(1-\alpha_{ij} - \alpha_{ji})$

$\Rightarrow \forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 : x_i + \alpha_{ji} - x_j - \alpha_{ij} \geq -(n-2) + (n-2)\alpha_{ij} + (n-2)\alpha_{ji}$

$\Rightarrow \forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 : x_i - x_j \geq -(n-2) + (n-1)\alpha_{ij} + (n-3)\alpha_{ji}$

$\Rightarrow \forall (i,j) / i < j \,\&\, R_i = 0 \,\&\, R_j = 0 : x_j - x_i + (n-1)\alpha_{ij} + (n-3)\alpha_{ji} \leq (n-2)$

4.6. Función objetivo

Minimizar el coste total de los arcos seleccionados

$$Min \sum_{(i,j)\in A} C_{ij}\alpha_{ij}$$

4.7. Modelo completo

$$Min \sum_{(i,j)\in A} C_{ij}\alpha_{ij}$$

s.t.

$$\forall i / T_i = 1 \& R_i = 0 : \sum_{j/(i,j)\in A} \alpha_{ij} = 1$$

$$\forall j / T_j = 0 : \sum_{i/(i,j)\in A} \alpha_{ij} \le NE_j \beta_i$$

$$\forall i / T_i = 0 : \sum_{j/(i,j)\in A} \alpha_{ij} \ge \beta_i$$

$$\forall i / R_i = 0 : x_i \le n-1$$

$$\forall i / R_i = 0 : x_i \ge 1$$

Versión MTZ:

$$\forall (i,j) / R_i = 0 \& R_j = 0 : x_j - x_i + (n-1)\alpha_{ij} \le (n-2)$$

Versión DL:

$$\forall (i,j) / i < j \& R_i = 0 \& R_j = 0 : x_i - x_j + (n-3)\alpha_{ij} + (n-1)\alpha_{ji} \le (n-2)$$
$$\forall (i,j) / i < j \& R_i = 0 \& R_j = 0 : x_j - x_i + (n-1)\alpha_{ij} + (n-3)\alpha_{ji} \le (n-2)$$

$$\forall (i,j) \in A : \alpha_{ij} \text{ binarias}$$

$$\forall i / R_i = 0 : x_i \ge 0$$

5. IMPLEMENTACIÓN

Una vez conocidos los modelos matemáticos del problema de Steiner y el funcionamiento de Jupyter Notebook y Gurobi, ya podemos utilizar estas herramientas para crear problemas y resolverlos. Veremos en este apartado la construcción de ambos modelos, aunque difieren únicamente en las restricciones antibucles, en Jupyter Notebook y su posterior resolución mediante el software utilizado. En primer lugar, explicaremos los formatos utilizados donde se presentan los datos de cada problema. También veremos cómo leer esos datos y convertirlos en los elementos necesarios según el modelo matemático. Finalmente crearemos las variables, restricciones y función objetivo para así poder resolver el problema.

5.1. Formato de los problemas de Steiner

En los ejemplos mostrados para comprender el funcionamiento de Jupyter Notebook y Gurobi, los datos necesarios para resolver los problemas han sido introducidos por teclado. Para el desarrollo del trabajo este método es imposible por las grandes dimensiones de los problemas utilizados, por tanto, los datos serán leídos desde un archivo de texto.

Hemos utilizados dos tipos de baterías de problemas, reducidos y no reducidos. La diferencia entre las dos baterías de problemas se debe a los grafos que generan. El formato no reducido crea grafos más extendidos, es decir, con mayor profundidad, y por tanto, su tiempo de resolución es superior al de los problemas reducidos, que generan grafos con menos profundidad. En primer lugar, veremos el formato de los problemas no reducidos y posteriormente el de los reducidos. Ambos formatos son muy similares y difieren en una única línea de texto.

5.1.1. Formato no reducido

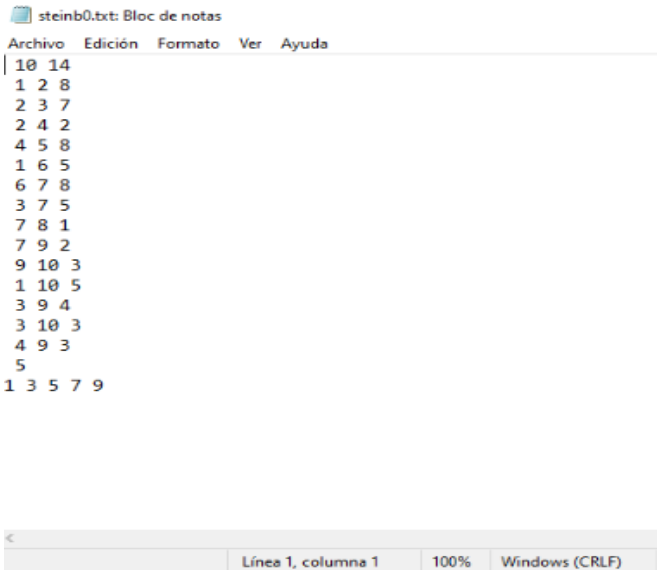


Ilustración 44 Ejemplo formato no reducido

Fuente: Elaboración propia

En la primera línea del archivo de texto tenemos dos números. El primero hace referencia al número de nodos de nuestro problema, que en este caso son 10, y el segundo al número de aristas, es decir, 14. Después de esta línea tendremos un número de líneas igual al número de aristas donde en cada línea se indica los nodos involucrados en la arista y el coste de esa arista. Por ejemplo, la primera arista está formada por los nodos 1 y 2 y tiene un coste de 8. En la penúltima línea se indica el número de nodos terminales, los cuales son 5 para este problema, y en la última línea el conjunto de nodos terminales. Está claro que tendremos tantos números como indique la línea anterior.

5.1.2. Formato reducido

Ilustración 45 Ejemplo formato problemas reducidos

Fuente: Elaboración propia

La diferencia con el formato no reducido radica en la última línea del archivo. En esta tendemos un número producto de la reducción del problema, de aquí viene el nombre de formato reducido. Mediante operaciones el problema es reducido y así conseguimos una reducción del tiempo para resolverlo. El valor obtenido en la última línea deberá ser sumado al obtenido al resolver el problema reducido y así ser equivalente al problema original.

A continuación, mostraremos el proceso de implementación del problema de Steiner no reducido con el ejemplo Steinb0 utilizando el modelo MTZ, comentaremos como implementar el modelo DL y también la implementación del modelo reducido.

5.2. Lectura de datos y creación de variables en Jupyter Notebook

El título de este apartado, que hace referencia a las variables de Jupyter Notebook, no son las mismas que las variables del modelo matemático, sino los datos introducidos en la tabla de elementos del modelo, como los nodos, nodos terminales, arcos y sus costes, nodo raíz y demás.

Dicho esto, mostramos en la siguiente imagen el código en Jupyter Notebook.

```
In [2]: lines = open ("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\steinb0.txt")

In [3]: from gurobipy import *

In [4]: arcos_costes = dict()
        contador=0
        num_nodos_terminales=0
        nodos_terminales=list()
```

Ilustración 46 Abrir archivo en Jupyter Notebook

Fuente: Elaboración propia

En primer lugar, abrimos el archivo que contiene el problema. Esto se hace mediante la función open() de Python. El contenido de ese archivo lo guardamos en la variable lines. El nombre elegido es para mostrar el funcionamiento de un bucle for en Python. Además, importamos gurobipy para hacer uso de todos los elementos necesarios en la creación del modelo.

Respecto a las variables incializadas, arcos_costes es un diccionario. En Python este tipo de dato se caracteriza por guardar parejas de clave-valor. En nuestro caso la clave será el arco, definido por el nodo inicio y nodo fin, y el coste de ese arco. El contador lo utilizamos para saber en cada línea qué tenemos que hacer. También hemos inicializado una variable donde guardaremos el número de nodos terminales y otra donde guardaremos cada uno de esos nodos terminales.

```
In [5]: for line in lines :
            b=line.split()
            print(b)
            if contador == 0 :
                nodes=int(b[0])
                arcs=int(b[1])
            elif contador >=1 and contador <=arcs :
                arcos_costes[(int(b[0]),int(b[1]))]=int(b[2])
                arcos_costes[(int(b[1]),int(b[0]))]=int(b[2])

            elif contador == arcs+1 :
                num_nodos_terminales=int(b[0])
            else :
                for i in range (len(b)):
                    nodos_terminales.append(int(b[i]))
            contador = contador + 1
```

Ilustración 47 Lectura de achivo y creación de datos en Jupyter Notebook

Fuente: Elaboración propia

El bucle for funciona de la siguiente manera: la variable line recorre línea por línea el archivo contenido en la variable lines. La variable lines es de tipo string al haber guardado una cadena de caracteres en ella, es decir, el archivo. Por tanto, line también será de tipo string. Mediante el método split() convertimos una cadena en una lista cuyo tamaño vendrá definido por los elementos que haya separados por espacios en blanco. Esta lista la guardamos en la variable b.

```
['10', '14']
['1', '2', '8']
['2', '3', '7']
['2', '4', '2']
['4', '5', '8']
['1', '6', '5']
['6', '7', '8']
['3', '7', '5']
['7', '8', '1']
['7', '9', '2']
['9', '10', '3']
['1', '10', '5']
['3', '9', '4']
['3', '10', '3']
['4', '9', '3']
['5']
['1', '3', '5', '7', '9']
```

Ilustración 48 Salida por pantalla del archivo leído

Fuente: Elaboración propia

En esta imagen podemos ver la salida por pantalla al imprimir la variable b. El contador los utilizaremos de la siguiente manera: la primera línea de nuestro archivo siempre contendrá el número de nodos y el número de arcos, por tanto, sabiendo los arcos que tenemos, mediante el contador empezaremos en la segunda línea y acabaremos según ese número de arcos. Por tanto, utilizaremos el condicional if para saber en cada momento los datos que tenemos que guardar en nuestras variables incializadas.

```
In [8]:
print(nodes)
print(arcs)
print(arcos_costes)
print(num_nodos_terminales)
print(nodos_terminales)

10
14
{(1, 2): 8, (2, 1): 8, (2, 3): 7, (3, 2): 7, (2, 4): 2, (4, 2): 2, (4, 5): 8, (5, 4): 8, (1, 6): 5, (6, 1): 5, (6, 7): 8,
(7, 6): 8, (3, 7): 5, (7, 3): 5, (7, 8): 1, (8, 7): 1, (7, 9): 2, (9, 7): 2, (9, 10): 3, (10, 9): 3, (1, 10): 5, (10, 1): 5,
(3, 9): 4, (9, 3): 4, (3, 10): 3, (10, 3): 3, (4, 9): 3, (9, 4): 3}
5
[1, 3, 5, 7, 9]
```

Ilustración 49 Impresión por pantalla de las variables creadas

Fuente: Elaboración propia

Imprimimos por pantalla las variables creadas y vemos que hemos guardado correctamente todos los datos necesarios del problema.

Ahora vamos a crear otras variables que no recogemos de leer los datos del problema, sino que tendremos que crearlas a partir de estos datos.

```
In [9]:
nodos = list()
for i in range(1,nodes+1) :
    nodos.append(i)
print(nodos)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [10]:
arcos=tuplelist(arcos_costes.keys())
arcos_entrantes=dict()
for i in nodos :
    seleccionados = tuplelist(arcos.select('*',i))
    arcos_entrantes[i]=len(seleccionados)
print(arcos_entrantes)

{1: 3, 2: 3, 3: 4, 4: 3, 5: 1, 6: 2, 7: 4, 8: 1, 9: 4, 10: 3}
```

Ilustración 50 Creación de variables derivadas de las primeras variables

Fuente: Elaboración propia

Hemos creado una lista con los nodos del problema y una lista con los arcos que entran en cada nodo. El proceso de creación es el siguiente: en primer lugar, creamos una variable llamada arcos donde guardaremos todos los arcos. Un arco está definido por el

nodo origen y el nodo destino, por tanto, su estructura es una tupla, arco (1,2). Lo que estamos haciendo es guardar un conjunto de tuplas en una lista. Esto es posible mediante la clase tuplelist. Este tipo de dato es creación de Gurobi por su utilidad para el modelado de problemas matemáticos y está disponible al importar gurobipy. Dentro de esa tuplelist hemos guardado los arcos como ya hicimos en la creación de las variables gracias al método keys(). La variable arcos_entrantes es un diccionario donde la clave es el nodo y el valor es la suma de arcos entrantes. El bucle for recorre cada nodo, así que explicaremos el nodo 1 ya que los demás serán iguales. La variable seleccionados es otra tuplelist donde introducimos todos los arcos que entran en el nodo 1. Esto se hace mediante el método select, que es propio de esta clase. select('*',i) funciona seleccionando todos los arcos cuyo nodo origen sea cualquiera y el nodo destino sea i, que en este caso es 1. Guardamos ese conjunto de arcos y actualizamos el diccionario. El valor del diccionario se obtiene al utilizar la función len(). Esta devuelve el tamaño de la variable seleccionados, es decir, el número de arcos que entran en el nodo 1. Imprimimos por pantalla y podemos ver los arcos entrantes para cada nodo.

```
In [12]: nodo_raiz=list()
         nodos_terminales.sort()
         nodo_raiz.append(nodos_terminales[0])
         print(nodo_raiz)

         [1]
```

Ilustración 51 Creación de variable nodo raíz

Fuente: Elaboración propia

Otro dato importante para nuestro problema es el nodo raíz. En nuestro caso hemos decidido que sea el número menor de los nodos terminales. En muchos de los problema que hemos utilizado el conjunto de nodos terminales no viene ordenado de menor a mayor, así que mediante el método sort() lo conseguimos y el primer elemento de esa lista ordenada será el nodo raíz.

```
In [41]: nodos_terminales_no_raiz=list()
         for i in nodos_terminales :
             nodos_terminales_no_raiz.append(int(i))
         nodos_terminales_no_raiz.remove(nodo_raiz[0])
         print(nodos_terminales_no_raiz)

         [3, 5, 7, 9]
```

Ilustración 52 Creación de variable nodos terminales

Fuente: Elaboración propia

Además, hemos creado una lista que contenga a todos los nodos terminales que no son el nodo raíz. Esto será necesario para la creación de las restricciones sobre este conjunto.

```
In [117]: nodos_no_raiz=list()
          for i in range(1,nodes+1) :
              nodos_no_raiz.append(i)
          nodos_no_raiz.remove(nodo_raiz[0])
```

Ilustración 53 Creación de variable nodos no raíz

Fuente: Elaboración propia

También hemos creado una lista que contiene a todos los nodos que no son el nodo raíz. Este elemento es necesario para la variable de profundidad, ya que el nodo raíz no tiene esta variable asociada.

```
In [13]:
nodos_steiner = list()
for i in range(1,nodes+1) :
    nodos_steiner.append(i)
print(nodos_steiner)
for i in nodos_terminales :
    nodos_steiner.remove(i)
print(nodos_steiner)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[2, 4, 6, 8, 10]
```

Ilustración 54 Creación de variable nodos steiner

Fuente: Elaboración propia

El último dato necesario para modelar nuestro problema de Steiner es el conjunto de nodos Steiner. Esta lista la hemos creado eliminando de una lista que contiene el conjunto total de nodos todos los nodos terminales. Esta lista es necesaria para las restricciones asociadas a los nodos Steiner.

Una vez tenemos todos los elementos necesarios, podemos continuar con el siguiente paso: la creación de las variables del problema.

5.3. Creación de variables del problema de Steiner

Realmente el proceso más complejo para la elaboración de problemas en Jupyter Notebook y su posterior resolución con Gurobi es el de obtener los datos del problema.

Al conseguir estos, el proceso de crear variables, restricciones y funciones objetivo es muy sencillo al ser bastante similar al modelado matemático que hemos presentado en el apartado 4.

```
In [14]:
m=Model()

Academic license - for non-commercial use only - expires 2021-07-01
Using license file C:\Users\rafit\gurobi.lic

In [15]:
alfa=m.addVars(arcos_costes.keys(), vtype=GRB.BINARY, name="alfa")
m.update()

In [16]:
print(alfa)

{(1, 2): <gurobi.Var alfa[1,2]>, (2, 1): <gurobi.Var alfa[2,1]>, (2, 3): <gurobi.Var alfa[2,3]>, (3, 2): <gurobi.Var alfa[3,
2]>, (2, 4): <gurobi.Var alfa[2,4]>, (4, 2): <gurobi.Var alfa[4,2]>, (4, 5): <gurobi.Var alfa[4,5]>, (5, 4): <gurobi.Var alf
a[5,4]>, (1, 6): <gurobi.Var alfa[1,6]>, (6, 1): <gurobi.Var alfa[6,1]>, (6, 7): <gurobi.Var alfa[6,7]>, (7, 6): <gurobi.Var
alfa[7,6]>, (3, 7): <gurobi.Var alfa[3,7]>, (7, 3): <gurobi.Var alfa[7,3]>, (7, 8): <gurobi.Var alfa[7,8]>, (8, 7): <gurobi.
Var alfa[8,7]>, (7, 9): <gurobi.Var alfa[7,9]>, (9, 7): <gurobi.Var alfa[9,7]>, (9, 10): <gurobi.Var alfa[9,10]>, (10, 9): <
gurobi.Var alfa[10,9]>, (1, 10): <gurobi.Var alfa[1,10]>, (10, 1): <gurobi.Var alfa[10,1]>, (3, 9): <gurobi.Var alfa[3,9]>,
(9, 3): <gurobi.Var alfa[9,3]>, (3, 10): <gurobi.Var alfa[3,10]>, (10, 3): <gurobi.Var alfa[10,3]>, (4, 9): <gurobi.Var alfa
[4,9]>, (9, 4): <gurobi.Var alfa[9,4]>}
```

Ilustración 55 Creación de modelo y variable en Jupyter Notebook

Fuente: Elaboración propia

Antes de crear las variables, hemos inicializado un modelo vacío al que hemos llamado m. Esto es necesario ya que tendremos que añadir variables a ese modelo m mediante el método addVars(), propio de la clase Model de Gurobi.

La variable asociada al modelo matemático que creamos es: $\forall(i, j) \in A: \alpha_{ij}$ *binarias*

Las actividades de decisión son guardadas en la variable alfa. El método addVars() necesita como argumento los índices de los arcos, el tipo de variable y el nombre que

queramos. Como necesitamos las claves pero no los valores de los arcos, es decir, los costes, utilizamos el método keys() para quedarnos solo con las claves. El método update() sirve para actualizar el modelo.

Podemos comprobar que hemos añadido las variables correctamente imprimiendo por pantalla la variable alfa. A cada clave del diccionario arcos_costes, es decir, a cada arco se le ha asociado una variable alfa[i,j], exactamente igual que en nuestro modelo matemático del apartado 4.

La otra variable necesaria para modelar el problema es la variable de profundidad. En el modelo matemático queda representada como: $\forall i / R_i = 0 : x_i \geq 0$

```
In [121]: profundidad = m.addVars(nodos_no_raiz, name="X")
          m.update()
          print(profundidad)

          {2: <gurobi.Var X[2]>, 3: <gurobi.Var X[3]>, 4: <gurobi.Var X[4]>, 5: <gurobi.Var X[5]>, 6: <gurobi.Var X[6]>, 7: <gurobi.Va
          r X[7]>, 8: <gurobi.Var X[8]>, 9: <gurobi.Var X[9]>, 10: <gurobi.Var X[10]>}

In [122]: v=m.getVars()
          print(v)

          [<gurobi.Var alfa[1,2]>, <gurobi.Var alfa[2,1]>, <gurobi.Var alfa[2,3]>, <gurobi.Var alfa[3,2]>, <gurobi.Var alfa[2,4]>, <gu
          robi.Var alfa[4,2]>, <gurobi.Var alfa[4,5]>, <gurobi.Var alfa[5,4]>, <gurobi.Var alfa[1,6]>, <gurobi.Var alfa[6,1]>, <gurob
          i.Var alfa[6,7]>, <gurobi.Var alfa[7,6]>, <gurobi.Var alfa[3,7]>, <gurobi.Var alfa[7,3]>, <gurobi.Var alfa[7,8]>, <gurobi.Va
          r alfa[8,7]>, <gurobi.Var alfa[7,9]>, <gurobi.Var alfa[9,7]>, <gurobi.Var alfa[9,10]>, <gurobi.Var alfa[10,9]>, <gurobi.Var
          alfa[1,10]>, <gurobi.Var alfa[10,1]>, <gurobi.Var alfa[3,9]>, <gurobi.Var alfa[9,3]>, <gurobi.Var alfa[3,10]>, <gurobi.Var a
          lfa[10,3]>, <gurobi.Var alfa[4,9]>, <gurobi.Var alfa[9,4]>, <gurobi.Var X[2]>, <gurobi.Var X[3]>, <gurobi.Var X[4]>, <gurob
          i.Var X[5]>, <gurobi.Var X[6]>, <gurobi.Var X[7]>, <gurobi.Var X[8]>, <gurobi.Var X[9]>, <gurobi.Var X[10]>]
```

Ilustración 56 Creación de variable profundidad

Fuente: Elaboración propia

Al igual que con la variable alfa, hemos introducido como argumento los elementos a los que va asociada la variable profundidad, que en este caso es a cada nodo menos el nodo raíz, además de un nombre. No ha sido necesario especificar el tipo de variable ya que se trata de una variable continua y Gurobi por defecto interpreta que las variables son de este tipo si no lo especificamos. Imprimimos por pantalla la variable profundidad y vemos que cada nodo tiene una variable asociada.

También podemos utilizar el método getVars() propio de la clase modelo para guardar todas las variables creadas e imprimiarlas. Vemos que tenemos tanto el conjunto de variables asociadas a los arcos como las variables de profundidad asociada a los nodos.

La última variable necesaria para modelar el problema es una variable auxiliar. No es una variable de decisión como tal, sino que es derivada del modelado de especificaciones visto en el apartado 4. Esta variable representa la decisión de seleccionar un nodo Steiner, así que recibe como argumentos el conjunto de nodos Steiner, el tipo binario y el nombre que le hemos dado.

```
In [65]: auxiliar= m.addVars(nodos_steiner, vtype=GRB.BINARY, name="B")
         m.update()
         print(auxiliar)

         {2: <gurobi.Var B[2]>, 4: <gurobi.Var B[4]>, 6: <gurobi.Var B[6]>, 8: <gurobi.Var B[8]>, 10: <gurobi.Var B[10]>}

In [66]: p=m.getVars()
         print(p)

         [<gurobi.Var alfa[1,2]>, <gurobi.Var alfa[2,1]>, <gurobi.Var alfa[2,3]>, <gurobi.Var alfa[3,2]>, <gurobi.Var alfa[2,4]>, <gu
         robi.Var alfa[4,2]>, <gurobi.Var alfa[4,5]>, <gurobi.Var alfa[5,4]>, <gurobi.Var alfa[1,6]>, <gurobi.Var alfa[6,1]>, <gurob
         i.Var alfa[6,7]>, <gurobi.Var alfa[7,6]>, <gurobi.Var alfa[3,7]>, <gurobi.Var alfa[7,3]>, <gurobi.Var alfa[7,8]>, <gurobi.Va
         r alfa[8,7]>, <gurobi.Var alfa[7,9]>, <gurobi.Var alfa[9,7]>, <gurobi.Var alfa[9,10]>, <gurobi.Var alfa[10,9]>, <gurobi.Var
         alfa[1,10]>, <gurobi.Var alfa[10,1]>, <gurobi.Var alfa[3,9]>, <gurobi.Var alfa[9,3]>, <gurobi.Var alfa[3,10]>, <gurobi.Var a
         lfa[10,3]>, <gurobi.Var alfa[4,9]>, <gurobi.Var alfa[9,4]>, <gurobi.Var X[1]>, <gurobi.Var X[2]>, <gurobi.Var X[3]>, <gurob
         i.Var X[4]>, <gurobi.Var X[5]>, <gurobi.Var X[6]>, <gurobi.Var X[7]>, <gurobi.Var X[8]>, <gurobi.Var X[9]>, <gurobi.Var X[1
         0]>, <gurobi.Var B[2]>, <gurobi.Var B[4]>, <gurobi.Var B[6]>, <gurobi.Var B[8]>, <gurobi.Var B[10]>]
```

Ilustración 57 Creación de variable auxiliar

Fuente: Elaboración propia

5.4. Creación de las restricciones del problema de Steiner

Vamos a seguir el orden de las restricciones mostradas en el apartado 4.

La primera restricción obliga a que todo nodo terminal tenga un padre, a excepción del nodo raíz.

La restricción en forma matemática es la siguiente: $\forall i / T_i = 1 \& R_i = 0 : \sum_{j/(i,j)\in A} \alpha_{ij} = 1$

```
In [67]: m.addConstrs((alfa.sum(i,'*') == 1 for i in nodos_terminales_no_raiz), name="C")

Out[67]: {3: <gurobi.Constr *Awaiting Model Update*>,
          5: <gurobi.Constr *Awaiting Model Update*>,
          7: <gurobi.Constr *Awaiting Model Update*>,
          9: <gurobi.Constr *Awaiting Model Update*>}
```

Ilustración 58 Creación de restricciones en Jupyter Notebook

Fuente: Elaboración propia

Como argumento el método addConstrs() recibe una expresión matemática, el conjunto al que se aplica esa expresión y un nombre. El conjunto de nodos terminales que no son el raíz está formado por 4 nodos, así que tendremos 4 restricciones distintas. El método sum(i,*) funciona realizando un sumatorio de las variables alfa asociadas a los arcos donde el nodo origen es el i y el nodo destino * es cualquiera que no sea él mismo.

La siguiente especificación es que cada nodo Steiner seleccionado debe poseer un padre. Tras modelar esta especificación obtenemos dos restricciones:

- $\forall j / T_j = 0 : \sum_{i/(i,j)\in A} \alpha_{ij} \leq NE_j \beta_i$

- $\forall i / T_i = 0 : \sum_{j/(i,j)\in A} \alpha_{ij} \geq \beta_i$

En la imagen siguiente podemos ver como han sido creadas las restricciones en Jupyter Notebook.

```
In [70]: m.addConstrs((alfa.sum('*',i) <= arcos_entrantes[i]*auxiliar[i] for i in nodos_steiner), name="C1")

Out[70]: {2: <gurobi.Constr *Awaiting Model Update*>,
          4: <gurobi.Constr *Awaiting Model Update*>,
          6: <gurobi.Constr *Awaiting Model Update*>,
          8: <gurobi.Constr *Awaiting Model Update*>,
          10: <gurobi.Constr *Awaiting Model Update*>}

In [71]: m.addConstrs((alfa.sum(i,'*') >= auxiliar[i] for i in nodos_steiner), name="C2")

Out[71]: {2: <gurobi.Constr *Awaiting Model Update*>,
          4: <gurobi.Constr *Awaiting Model Update*>,
          6: <gurobi.Constr *Awaiting Model Update*>,
          8: <gurobi.Constr *Awaiting Model Update*>,
          10: <gurobi.Constr *Awaiting Model Update*>}
```

Ilustración 59 Creación de restricciones nodo padre

Fuente: Elaboración propia

El proceso de creación es idéntico al de las primeras restricciones, con la diferencia del conjunto al que se le aplica, que en este caso es el conjunto de nodos Steiner.

Las siguientes restricciones que crear son las de profundidad máxima y profundidad mínima:

- $\forall i / R_i = 0 : x_i \leq n-1$

- $\forall i / R_i = 0 : x_i \geq 1$

```
In [128]: m.addConstrs((profundidad[i]<=nodes-1 for i in nodos_no_raiz ), name="C4")

Out[128]: {2: <gurobi.Constr *Awaiting Model Update*>,
           3: <gurobi.Constr *Awaiting Model Update*>,
           4: <gurobi.Constr *Awaiting Model Update*>,
           5: <gurobi.Constr *Awaiting Model Update*>,
           6: <gurobi.Constr *Awaiting Model Update*>,
           7: <gurobi.Constr *Awaiting Model Update*>,
           8: <gurobi.Constr *Awaiting Model Update*>,
           9: <gurobi.Constr *Awaiting Model Update*>,
           10: <gurobi.Constr *Awaiting Model Update*>}

In [129]: m.addConstrs((profundidad[i]>=1 for i in nodos_no_raiz ), name="C5")

Out[129]: {2: <gurobi.Constr *Awaiting Model Update*>,
           3: <gurobi.Constr *Awaiting Model Update*>,
           4: <gurobi.Constr *Awaiting Model Update*>,
           5: <gurobi.Constr *Awaiting Model Update*>,
           6: <gurobi.Constr *Awaiting Model Update*>,
           7: <gurobi.Constr *Awaiting Model Update*>,
           8: <gurobi.Constr *Awaiting Model Update*>,
           9: <gurobi.Constr *Awaiting Model Update*>,
           10: <gurobi.Constr *Awaiting Model Update*>}
```

Ilustración 60 Creación de restricciones de profundidad máxima y mínima

Fuente: Elaboración propia

Finalmente, la última restricción del modelo es la restricción antibucles. La expresión matemática es: $\forall (i, j) / R_i = 0 \,\&\, R_j = 0 : x_j - x_i + (n-1)\alpha_{ij} \leq (n-2)$

La creación del conjunto de estas restricciones podemos verla en la imagen siguiente:

```
In [130]: m.addConstrs((profundidad[i]-profundidad[j]+(nodes-1)*alfa[i,j]<=(nodes-2)
                        for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
                       name="C6")

Out[130]: {(2, 3): <gurobi.Constr *Awaiting Model Update*>,
           (3, 2): <gurobi.Constr *Awaiting Model Update*>,
           (2, 4): <gurobi.Constr *Awaiting Model Update*>,
           (4, 2): <gurobi.Constr *Awaiting Model Update*>,
           (4, 5): <gurobi.Constr *Awaiting Model Update*>,
           (5, 4): <gurobi.Constr *Awaiting Model Update*>,
           (6, 7): <gurobi.Constr *Awaiting Model Update*>,
           (7, 6): <gurobi.Constr *Awaiting Model Update*>,
           (3, 7): <gurobi.Constr *Awaiting Model Update*>,
           (7, 3): <gurobi.Constr *Awaiting Model Update*>,
           (7, 8): <gurobi.Constr *Awaiting Model Update*>,
           (8, 7): <gurobi.Constr *Awaiting Model Update*>,
           (7, 9): <gurobi.Constr *Awaiting Model Update*>,
           (9, 7): <gurobi.Constr *Awaiting Model Update*>,
           (9, 10): <gurobi.Constr *Awaiting Model Update*>,
           (10, 9): <gurobi.Constr *Awaiting Model Update*>,
           (3, 9): <gurobi.Constr *Awaiting Model Update*>,
           (9, 3): <gurobi.Constr *Awaiting Model Update*>,
           (3, 10): <gurobi.Constr *Awaiting Model Update*>,
           (10, 3): <gurobi.Constr *Awaiting Model Update*>,
           (4, 9): <gurobi.Constr *Awaiting Model Update*>,
           (9, 4): <gurobi.Constr *Awaiting Model Update*>}
```

Ilustración 61 Creación de restricciones antibucles

Fuente: Elaboración propia

Esta restricción es la más compleja de todas las del modelo al tener que incluir al operador condicional if. Aún así, podemos comprobar visualmente la similitud entre las expresiones matemáticas y las expresiones creadas con Python. Esta es una de las ventajas más importantes a la hora de decidir utilizar Python y Jupyter Notebook en este trabajo.

5.5. Creación de la función objetivo

El último paso antes de poder resolver el problema de Steiner es establecer la función objetivo. En nuestro caso es:

$$Min \sum_{(i,j)\in A} C_{ij}\alpha_{ij}$$

En la siguiente imagen veremos cómo hemos construido esta función objetivo en Jupyter Notebook.

```
In [131]: fun_obj = 0
          for i,j in alfa :
              fun_obj = fun_obj + alfa[i,j]*arcos_costes[i,j]
          m.setObjective(fun_obj, GRB.MINIMIZE)
          m.update()
```

Ilustración 62 Creación de función objetivo en Jupyter Notebook

Fuente: Elaboración propia

Hemos inicializado la variable función objetivo y la hemos actualizado mediante un bucle for incrementando en cada iteración la función objetivo al multiplicar la variable binaria alfa por el coste asociado a esa variable que tenemos guardado en el diccionario arcos_coste. El bucle for recorre en este caso el conjunto alfa donde tenemos las variables asociadas a los arcos.

Tras obtener la expresión de la función objetivo, la pasamos como argumento al método setObjective, además del criterio de optimización, que en este caso es minimizar.

Por último, actualizamos el modelo y ya podemos llamar a Gurobi para resolver.

5.6. Optimización con Gurobi y obtención del resultado

```
In [132]: m.optimize()

Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 54 rows, 42 columns and 131 nonzeros
Model fingerprint: 0x8250d105
Variable types: 9 continuous, 33 integer (33 binary)
Coefficient statistics:
  Matrix range     [1e+00, 9e+00]
  Objective range  [1e+00, 8e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 9e+00]
Found heuristic solution: objective 32.0000000
Presolve removed 31 rows and 17 columns
Presolve time: 0.07s
Presolved: 23 rows, 25 columns, 75 nonzeros
Variable types: 6 continuous, 19 integer (19 binary)

Root relaxation: objective 1.928571e+01, 10 iterations, 0.04 seconds


    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   19.28571    0    4   32.00000   19.28571  39.7%     -    0s
H    0     0                      27.0000000   19.28571  28.6%     -    0s
     0     0   20.28571    0    2   27.00000   20.28571  24.9%     -    0s
H    0     0                      26.0000000   20.28571  22.0%     -    0s
     0     0   21.50000    0    6   26.00000   21.50000  17.3%     -    0s
     0     0   22.00000    0    6   26.00000   22.00000  15.4%     -    0s
     0     0   23.66667    0    6   26.00000   23.66667  8.97%     -    0s
H    0     0                      24.0000000   23.66667  1.39%     -    0s

Cutting planes:
  Learned: 3
  Gomory: 2
  Cover: 2
  Implied bound: 2
  MIR: 5

Explored 1 nodes (28 simplex iterations) in 0.48 seconds
Thread count was 8 (of 8 available processors)

Solution count 4: 24 26 27 32

Optimal solution found (tolerance 1.00e-04)
Best objective 2.400000000000e+01, best bound 2.400000000000e+01, gap 0.0000%
```

Ilustración 63 Optimización y salida de datos con Gurobi y Jupyter Notebook

Fuente: Elaboración propia

Utilizamos el método optimize() y Gurobi nos devuelve por pantalla lo mostrado en la imagen. Podemos obtener el valor de la función y el tiempo de ejecución mediante los comandos objVal y runtime propios de la clase model,

```
In [133]: print(m.objVal)
          print(m.runtime)

          24.0
          0.501190185546875
```

Ilustración 64 Salida por pantalla valor fo y tiempo de ejecución

Fuente: Elaboración propia

En este caso, el valor de la función objetivo es de 24 unidades y el tiempo de ejecución es de 0.501190185546875 segundos.

Una vez explicado la resolución del problema no reducido steinb0 mediante el modelo MTZ, vamos a explicar como aplicar el modelo DL.

5.7. Variación modelo DL

La única diferencia entre el modelo MTZ y el modelo DL radica en las restricciones utilizadas para evitar los bucles.

En el modelo MTZ teníamos la expresión matemática:

$$\forall(i,j)/R_i=0\,\&\,R_j=0: x_j-x_i+(n-1)\alpha_{ij}\le(n-2)$$

Mientras que el modelo DL utiliza dos restricciones:

$$\forall(i,j)/i<j\,\&\,R_i=0\,\&\,R_j=0: x_i-x_j+(n-3)\alpha_{ij}+(n-1)\alpha_{ji}\le(n-2)$$
$$\forall(i,j)/i<j\,\&\,R_i=0\,\&\,R_j=0: x_j-x_i+(n-1)\alpha_{ij}+(n-3)\alpha_{ji}\le(n-2)$$

Por tanto, el único cambio que tenemos que realizar en Jupyter Notebook es prescindir de la restricción antibucle MTZ y crear las dos restricciones utilizadas en el modelo DL.

En la imagen se muestra la creación de estas restricciones:

```
In [ ]: m.addConstrs((profundidad[i]-profundidad[j]+(nodes-3)*alfa[i,j]+(nodes-1)*alfa[j,i]<=(nodes-2)
                           for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
                name="C6")
        m.addConstrs((profundidad[j]-profundidad[i]+(nodes-1)*alfa[i,j]+(nodes-3)*alfa[j,i]<=(nodes-2)
                           for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
                name="C7")
```

Ilustración 65 Creación restricciones antibucle modelo DL

Fuente: Elaboración propia

5.8. Variación problemas reducidos

Respecto a al formato utilizado para los problemas reducidos, vimos que añadían una línea más en los archivos de texto donde se daba un valor, el cual tenemos que sumar al valor obtenido al resolver el problema para conseguir el valor inicial del problema sin reducir. Para guardar este valor, simplemente haremos una modificación en el bucle for utilizado para leer el archivo de texto y guardar los datos del problema.

```
In [ ]: for line in lines :
            b=line.split()
            if contador == 0 :
                nodes=int(b[0])
                arcs=int(b[1])
            elif contador >=1 and contador <=arcs :
                arcos_costes[(int(b[0]),int(b[1]))]=int(b[2])
                arcos_costes[(int(b[1]),int(b[0]))]=int(b[2])

            elif contador == arcs+1 :
                num_nodos_terminales=int(b[0])
            elif contador == arcs+2 :
                for i in range (len(b)):
                    nodos_terminales.append(int(b[i]))
            else :
                diferencia_fo=int(b[0])
            contador = contador + 1
```

Ilustración 66 Variación lectura de datos modelo reducido

Fuente: Elaboración propia

Introducimos una condición y más y cuando lleguemos a la última línea guardamos el valor en una variable que hemos llamado diferencia_fo. El proceso de creación del problema es idéntico al de los problemas no reducidos, con la diferencia de cuando obtengamos el valor de la función objetivo, tendremos que sumarle la variable diferencia_fo.

5.9. Automatización para resolución de batería de problemas

En los apartados anteriores hemos visto cómo crear y resolver el modelo MTZ para un problema no reducido, cómo crear y resolver el modelo DL para el mismo tipo de problemas y cómo variar el código para crear y resolver ambos modelos para los problemas reducidos. Para el desarrollo de este trabajo no hemos tenido que resolver únicamente un problema no reducido y uno reducido, sino una batería de problemas para cada tipo. Por tanto, en este apartado explicaremos cómo hemos automatizado el proceso para crear y resolver la batería de problemas de cada tipo.

Respecto a la automatización, hemos dividido en cuatro grupos distintos: modelo MTZ con problemas no reducidos, modelo MTZ con problemas reducidos, modelo DL con problemas no reducidos y modelo DL con problemas reducidos.

Realmente el modo de conseguir esta automatización es prácticamente idéntico, así que explicaremos el primer grupo y comentaremos las diferencias entre grupos para finalizar el apartado.

En la siguiente imagen se mostrará gráficamente la automatización para explicarla posteriormente.

```
In [ ]:
from gurobipy import*
problemas=["steinb0.txt","steinb1.txt","steinb2.txt","steinb3.txt","steinb4.txt","steinb5.txt","steinb6.txt","steinb7.txt"
          ,"steinb8.txt","steinb9.txt","steinb10.txt","steinb11.txt","steinb12.txt","steinb13.txt","steinb14.txt","steinb15.
          ,"steinb16.txt","steinb17.txt","steinb18.txt","steinc1.txt","steinc2.txt","steinc3.txt","steinc4.txt","steinc5.txt
          ,"steinc6.txt","steinc7.txt","steinc8.txt","steinc9.txt","steinc10.txt","steinc11.txt","steinc12.txt","steinc13.tx
          ,"steinc14.txt","steinc15.txt","steinc16.txt","steinc17.txt","steinc18.txt","steinc19.txt","steinc20.txt"]
for problema in problemas :
    lines = open ("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\"+problema)
    arcos_costes = dict()
    contador=0
    num_nodos_terminales=0
    nodos_terminales=list()
    arcos=list()
    for line in lines :
        b=line.split()
        if contador == 0 :
            nodes=int(b[0])
            arcs=int(b[1])
        elif contador >=1 and contador <=arcs :
            arcos_costes[(int(b[0]),int(b[1]))]=int(b[2])
            arcos_costes[(int(b[1]),int(b[0]))]=int(b[2])

        elif contador == arcs+1 :
            num_nodos_terminales=int(b[0])
        else :
            for i in range (len(b)):
                nodos_terminales.append(int(b[i]))
```

Ilustración 67 Automatización para resolución batería de problemas

Fuente: Elaboración propia

La principal diferencia que podemos observar respecto al proceso de creación del problema en los apartados anteriores es que todo el código está escrito en la misma celda. Durante la explicación de Jupyter Notebook comentamos que una de las características más importantes era la posible intercalación de código y texto. Para la automatización esto no es posible debido al bucle for utilizado. Si separamos en distintas celdas el bucle y el código de creación del problema, primero se ejecutaría el bucle for hasta terminar de leer todos los problemas de la batería y después empezaría a crear el problema, que en este caso sería el último de todos ellos, es decir, el steinc20.txt.

Al ser nuestro objetivo que el bucle for lea un problema, lo cree, lo resuelva y pase con el siguiente, tendremos que utilizar la misma celda.

Comentada la principal diferencia, vamos a explicar el funcionamiento del bucle. Importado el módulo de gurobipy, creamos una lista que contiene todos los problemas de la batería de problemas no reducidos e iniciamos un bucle for que recorrerá esta lista empezando por el primero. El código leerá la información de cada problema, guardando sus datos, creando variables, restricciones y función objetivo y optimizándolo.

Además de la lista de problemas y el bucle for que la recorre, después de optimizar el problema hemos añadido unas líneas de código que mostraremos a continuación.

```
m.optimize()
optimo=str(m.objVal)
tiempo=str(m.runtime)
f=open("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\Optimos_MTZ_NoReducidos.txt", "a")
f.write(problema  )
f.write(optimo )
f.write(tiempo)
f.write("\n")
f.close()
```

Ilustración 68 Automatización para recoger resultados

Fuente: Elaboración propia

Después del método optimize() hemos creado dos variables, optimo y tiempo, donde asignamos la solución y el tiempo de ejecución de cada problema. Estos dos valores, además del nombre del problema, son guardados en un archivo de texto automáticamente mediante la función open, que permite tanto leer como escribir. Hemos utilizado la función str() para convertir en cadenas de texto tanto la solución como el tiempo de ejecución ya que el método write() solo admite de argumento este tipo de dato.

En la imagen siguiente se muestra la solución del problema steinb0 no reducido mediante el modelo MTZ en el archivo de texto.

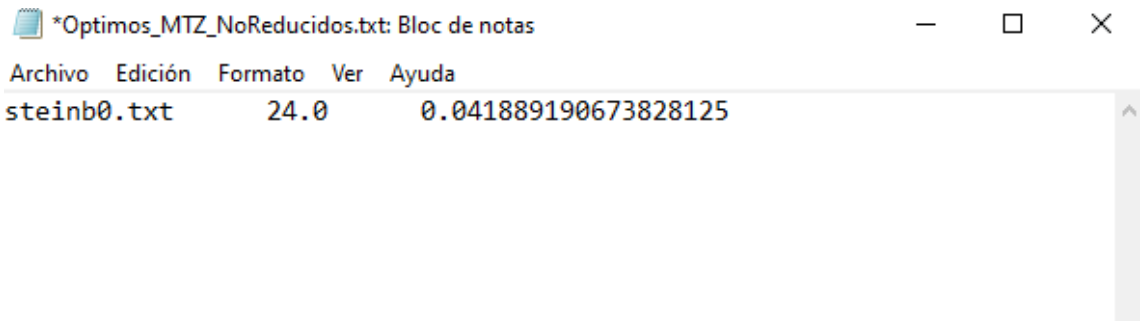


Ilustración 69 Formato recogida de resultados

Fuente: Elaboración propia

Para automatizar el modelo DL con problemas no reducidos basta con utilizar el mismo formato cambiando las dos restricciones antibucle que incluye este modelo por la restricción antibucle utilizada por el modelo MTZ.

Respecto a los problemas reducidos, tenemos que sumar al óptimo obtenido en la optimización el valor guardado al leer los datos del problema.

```
optimo=str(m.objVal + diferencia_fo)
tiempo=str(m.runtime)


f=open("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\Optimos_DL_Reducidos_Gurobi.txt", "a")
f.write(problema)
f.write("    ")
f.write(optimo)
f.write("    ")
f.write(tiempo)
f.write("\n")
f.close()
```

Ilustración 70 Automatización reocgida de datos para problemas reducidos

Fuente: Elaboración propia

Una vez explicada la automatización para resolver las dos baterías de problemas mediante ambos modelos matemáticos, podemos evaluar los resultados obtenidos de esta actividad en el siguiente apartado.

6. RESULTADOS COMPUTACIONALES

En este apartado veremos los resultados obtenidos tras resolver la batería de problemas. Como explicamos en el apartado anterior, tenemos cuatro modelos distintos: MTZ con batería de problemas no reducidos, DL con batería de problemas no reducidos, MTZ con batería de problemas reducidos y DL con batería de problemas reducidos. Para abreviar, a partir de ahora los modelos serán mencionados como MTZ-NR, DL-NR, MTZ-R y DL-R respectivamente.

Respecto a las baterías de problemas, dentro de los no reducidos contamos con problemas del tipo SteinerB y SteinerC. Ambos tipos se diferencian en el tamaño, ya que los problemas SteinerC tienen todos 500 nodos, mientras que los problemas SteinerB tienen entre 10 y 100 nodos. Para los problemas reducidos tenemos los tipos mencionados anteriormente y además hemos añadido los SteinerD. Estos problemas son de un tamaño más considerable y por este motivo no han sido resueltos como no reducidos. Por tanto, todos los modelos resolverán los problemas SteinerB y SteinerC, con la diferencia de que en un caso serán no reducidos y en otro reducidos, pero el valor obtenido será siempre el mismo, como explicamos en el apartado de la diferencia entre problemas reducidos y no reducidos. Además, los modelos MTZ-R y DL-R resolverán los problemas SteinD también.

Para mostrar los resultados computacionales hemos utilizado una tabla que consta del problema que se resuelve con sus características, es decir, número de nodos, número de aristas y número de nodos terminales, además del valor obtenido al resolver dicho problema y el tiempo de resolución en segundos tanto del modelo MTZ como del modelo DL. Así, tendremos una tabla para la batería de problemas no reducidos y otra tabla para la batería de problemas reducidos.

Finalmente, mostraremos una tabla comparativa de la rapidez de ambos modelos. Hemos diferenciado entre la batería de problemas no reducidos y reducidos, y a su vez, entre los distintos tipos de problemas dentro de cada batería. Por tanto, para cada modelo tendremos cinco valores distintos, los cuales serán: tiempo en resolver todos los problemas SteinerB no reducidos, tiempo en resolver todos los problemas SteinerC no reducidos, tiempo en resolver todos los problemas SteinerB reducidos, tiempo en resolver todos los problemas SteinerC reducidos y tiempo en resolver todos los problemas SteinerD reducidos.

6.1. Resultados computacionales de la batería de problemas no reducidos

	NODOS	ARCOS	NODOS TERMINALES	VALOR FO	TIEMPO MTZ (s)	TIEMPO DL (s)
steinb1	50	63	9	82	0,21	0,17
steinb2	50	63	13	83	0,47	0,31
steinb3	50	63	25	138	0,35	0,32

steinb4	50	100	9	59	0,38	0,21
steinb5	50	100	13	61	0,09	0,09
steinb6	50	100	25	122	0,73	0,43
steinb7	75	94	13	111	0,26	0,13
steinb8	75	94	19	104	0,69	0,27
steinb9	75	94	38	220	0,23	0,11
steinb10	75	150	13	86	0,28	0,19
steinb11	75	150	19	88	0,83	0,78
steinb12	75	150	38	174	0,36	0,28
steinb13	100	125	17	165	0,43	0,29
steinb14	100	125	25	235	1,25	1,87
steinb15	100	125	50	318	0,13	0,13
steinb16	100	200	17	127	0,83	0,69
steinb17	100	200	25	131	1,11	1,02
steinb18	100	200	50	218	0,39	0,27
steinc1	500	625	5	85	3,97	2,14
steinc2	500	625	10	144	4,22	3,79
steinc3	500	625	83	754	11,32	9,49
steinc4	500	625	125	1079	8,52	7,84
steinc5	500	625	250	1579	2,05	2,27
steinc6	500	1000	5	55	13,05	13,69
steinc7	500	1000	10	102	5,53	5,64
steinc8	500	1000	83	509	10,80	10,50
steinc9	500	1000	125	707	29,18	12,29
steinc10	500	1000	250	1093	30,50	103,50
steinc11	500	2500	5	32	292,24	327,11
steinc12	500	2500	10	46	28,82	16,82
steinc13	500	2500	83	258	90,96	87,17
steinc14	500	2500	125	323	6,35	14,92
steinc15	500	2500	250	556	920,81	505,53
steinc16	500	12500	5	11	13,08	13,73
steinc17	500	12500	10	18	12,86	16,22
steinc19	500	12500	125	146	3,62	9,36
steinc20	500	12500	250	267	2,23	5,57

Tabla 5 Resultados computacionales batería de problemas no reducidos

6.2. Resultados computacionales de la batería de problemas reducidos

	NODOS	ARCOS	NODOS TERMINALES	VALOR FO	TIEMPO MTZ (s)	TIEMPO DL (s)
steinb1	13	19	8	82	0,05	0,17

steinb2	15	21	11	83	0,04	0,06
steinb3	20	25	15	138	0,03	0,05
steinb4	40	80	9	59	0,29	0,40
steinb5	39	80	12	61	0,05	0,10
steinb6	45	87	25	122	0,27	0,50
steinb7	22	33	11	111	0,05	0,09
steinb8	26	38	15	104	0,16	0,29
steinb9	27	35	23	220	0,07	0,07
steinb10	55	121	13	86	0,16	0,27
steinb11	62	129	19	88	0,44	0,86
steinb12	63	125	36	174	0,15	0,23
steinb13	36	56	14	165	0,15	0,22
steinb14	42	65	21	235	0,12	0,16
steinb15	48	69	38	318	0,05	0,13
steinb16	77	166	17	127	0,35	0,68
steinb17	74	153	23	131	0,47	1,05
steinb18	82	166	45	218	0,17	0,26
steinc1	143	260	5	85	0,65	1,37
steinc2	128	234	10	144	0,76	0,99
steinc3	178	295	75	754	2,67	5,10
steinc4	193	314	102	1079	1,93	2,22
steinc5	223	341	180	1579	3,38	5,26
steinc6	366	837	5	55	5,29	5,88
steinc7	383	866	10	102	1,77	2,70
steinc8	387	867	79	509	8,78	19,79
steinc9	418	903	124	707	9,67	8,95
steinc10	427	891	242	1093	9,08	15,11
steinc11	499	2005	5	32	180,12	79,74
steinc12	499	2065	10	46	3,90	8,51
steinc13	498	2026	83	258	23,43	22,29
steinc14	499	1968	125	323	2,24	4,87
steinc15	500	1815	250	556	282,40	493,84
steinc16	500	3517	5	11	1,26	1,48
steinc17	500	3463	10	18	4,30	2,12
steinc19	500	3352	125	146	1,45	1,00
steinc20	500	3121	250	267	1,06	1,16
steind1	272	504	5	106	3,77	3,10
steind2	283	519	10	220	4,99	3,43
steind3	350	585	148	1565	1,58	2,12
steind4	359	590	207	1935	0,98	1,71
steind5	470	708	377	3250	1,05	0,88
steind6	759	1730	5	67	62,17	66,16
steind7	749	1721	10	103	3,66	4,85

steind9	802	1768	246	1448	15,95	25,68
steind10	836	1781	485	2110	7,64	7,17
steind11	993	4442	5	29	36,90	25,33
steind12	1000	4437	10	42	11,34	13,05
steind13	998	4354	167	500	16,80	35,67
steind15	996	3921	498	1116	10,03	16,12
steind16	1000	8048	5	13	35,64	30,59
steind17	1000	8061	10	23	19,25	22,11
steind18	1000	7755	167	223	49,59	95,50
steind19	1000	7552	250	310	6,31	12,11
steind20	1000	6887	500	539	1,64	2,67

Tabla 6 Resultados computacionales batería de problemas reducidos

6.3. Comparación de resultados entre modelo MTZ y modelo DL

	COMPARATIVA MODELOS MTZ Y DL				
	NO REDUCIDOS		REDUCIDOS		
	steinb	**steinc**	**steinb**	**steinc**	**steind**
MTZ	9,07	1490,10	3,09	544,14	289,29
DL	7,84	1167,60	5,58	682,39	368,26

Tabla 7 Comparativa de tiempos modelo MTZ y modelo DL

Una vez mostrados los resultados, podemos afirmar que el modelo DL es más rápido para resolver la batería de problemas no reducidos, ya que resuelve tanto el tipo SteinB como el SteinC en menor tiempo. Respecto a la batería de problemas reducidos, el modelo MTZ es mejor al ser más rápido tanto resolviendo los SteinB, como los SteinC y los SteinD. Por tanto, podemos concluir afirmando que para resolver problemas cuyos grafos sean más extendidos, es decir, con mayor profundidad, convendría utilizar el modelo Desroches-Laporte, mientras que, a la hota de resolver problemas de grafos con menor profundidad, como son los grafos reducidos, debemos utilizar el modelo matemático propuesto por Miller, Tucker y Zennit. Igualmente, las diferencias de tiempos entre ambos modelos no son notables y también podemos decir que las dos formulaciones matemáticas funcionan de manera adecuada para resolver las dos baterías de problemas propuestas.

7. CONCLUSIONES

Respecto a los modelos matemáticos, podemos concluir afirmando que el modelo MTZ es mejor que el modelo DL para problemas de grafos con menor profundidad, mientras que para grafos más extendidos el modelo DL se comporta mejor que el modelo MTZ. Cabe destacar que las diferencias entre ambos modelos no son muy significativas y por tanto sería posible usar cualquiera de los dos modelos para cualquiera de los dos tipos de grafos generados en este trabajo sin que los tiempos de ejecución variasen demasiado.

En el caso del software utilizado para resolver las baterías de problemas sí podemos afirmar que los resultados que hemos obtenidos se han conseguido en un tiempo notablemente superior a otra aplicación que no ha sido nombrada a lo largo del trabajo, pero de la cual tenemos constancia al haberla estudiado durante el grado. Esta aplicación es Lingo, un software que hemos visto en la carrera por la asignatura de métodos cuantitativos de gestión. El tutor de este trabajo ya tenía los resultados de las baterías de problemas con Lingo y hemos tenido la oportunidad de comparar tiempos entre ambas aplicaciones. Tras la comparativa, Gurobi ha demostrado ser bastante más rápido gracias a las buenas características del software que se han explicado a lo largo de este trabajo.

Considero que los conocimientos que he adquirido durante la realización de la carrera me han sido muy útiles para el desarrollo de este proyecto, aunque he tenido que formarme en estas herramientas de forma autónoma puesto que no formaban parte de los contenidos en ninguna de las asignaturas de mi grado. Esto para mi no ha sido un problema sino todo lo contrario. Creo que aprender a utilizar Gurobi y Python ha sido una experiencia enriquecedora y me será útil en un futuro. Además, al no haber mucha documentación en nuestro idioma respecto a la implementación de problemas de optimización bajo estos dos elementos, espero que esta tarea pueda ser útil a aquellos que quieran iniciarse en este modo de resolver problemas.

8. BIBLIOGRAFÍA

[1] José Manuel García Sánchez, 2021. "Modelling in Mathematical Programming," International Series in Operations Research and Management Science, Springer, number 978-3-030-57250-1, November.

[2] Aplicación de la formulación de R.K. Martin al problema de Steiner en grafos, Trabajo de Fin de Grado, Universidad de Sevilla, Autor: Sergio Flores Medina, Tutor: José Manuel García Sánchez, Año: 2019.

[3] Aplicaciones en la industria del problema de Steiner y su resolución mediante algoritmos genéticos, Trabajo de Fin de Grado, Universidad de Sevilla, Autor: Virginia Martínez Lacañina, Tutor: Pablo Cortés Achedad, Año: 2018.

[4] Pérez Aguila, R. (2013). Una introducción a las matemáticas discretas y teoría de grafos. El Cid Editor. https://elibro--net.us.debiblio.com/es/ereader/bibliotecaus/36562?page=1

[5] González Luque, Raúl. (28/10/2008). *Python para todos*. España: Otros.

[6] https://www.gurobi.com/documentation/9.1/quickstart_windows/index.html

9. ANEXOS

En el apartado de anexos mostraremos el código utilizado para automatizar la resolución de las dos baterías de problemas propuestas con los modelos MTZ Y DL.

9.1. Código en Jupyter Notebook para automatizar la resolución de la batería de problemas no reducidos con el modelo MTZ

```
from gurobipy import*

problemas=["steinb0.txt","steinb1.txt","steinb2.txt","steinb3.txt","steinb4.txt","steinb5.t
xt","steinb6.txt","steinb7.txt","steinb8.txt","steinb9.txt","steinb10.txt","steinb11.txt","st
einb12.txt","steinb13.txt","steinb14.txt","steinb15.txt","steinb16.txt","steinb17.txt","stei
nb18.txt","steinc1.txt","steinc2.txt","steinc3.txt","steinc4.txt","steinc5.txt","steinc6.txt",
"steinc7.txt","steinc8.txt","steinc9.txt","steinc10.txt","steinc11.txt","steinc12.txt","stein
c13.txt","steinc14.txt","steinc15.txt","steinc16.txt","steinc17.txt","steinc19.txt","steinc2
0.txt"]

for problema in problemas :
    lines = open ("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\"+problema)
    arcos_costes = dict()
    contador=0
    num_nodos_terminales=0
    nodos_terminales=list()
    arcos=list()
    for line in lines :
        b=line.split()
        if contador == 0 :
            nodes=int(b[0])
            arcs=int(b[1])
        elif contador >=1 and contador <=arcs :
            arcos_costes[(int(b[0]),int(b[1]))]=int(b[2])
            arcos_costes[(int(b[1]),int(b[0]))]=int(b[2])


        elif contador == arcs+1 :
```

```
        num_nodos_terminales=int(b[0])
    else :
        for i in range (len(b)):
            nodos_terminales.append(int(b[i]))
    contador = contador + 1
nodos=list()
for i in range(1,nodes+1) :
    nodos.append(i)
arcos=tuplelist(arcos_costes.keys())
arcos_entrantes=dict()
for i in nodos :
    seleccionados = tuplelist(arcos.select('*',i))
    arcos_entrantes[i]=len(seleccionados)
nodo_raiz=list()
nodos_terminales.sort()
nodo_raiz.append(nodos_terminales[0])
nodos_steiner = list()
for i in range(1,nodes+1) :
    nodos_steiner.append(i)
for i in nodos_terminales :
    nodos_steiner.remove(i)
nodos_terminales_no_raiz=list()
for i in nodos_terminales :
    nodos_terminales_no_raiz.append(int(i))
nodos_terminales_no_raiz.remove(nodo_raiz[0])
nodos_no_raiz=list()
for i in range(1,nodes+1) :
    nodos_no_raiz.append(i)
nodos_no_raiz.remove(nodo_raiz[0])
m=Model()
```

```
    alfa=m.addVars(arcos_costes.keys(), vtype=GRB.BINARY, name="alfa")
    m.update()
    profundidad = m.addVars(nodos_no_raiz, name="X")
    m.update()
    auxiliar= m.addVars(nodos_steiner, vtype=GRB.BINARY, name="B")
    m.update()
    m.addConstrs((alfa.sum(i,'*') == 1 for i in nodos_terminales_no_raiz), name="C")
    m.addConstrs((alfa.sum('*',i) <= arcos_entrantes[i]*auxiliar[i] for i in nodos_steiner),
    name="C1")
    m.addConstrs((alfa.sum(i,'*') >= auxiliar[i] for i in nodos_steiner), name="C3")
    m.addConstrs((profundidad[i]<=nodes-1 for i in nodos_no_raiz ), name="C4")
    m.addConstrs((profundidad[i]>=1 for i in nodos_no_raiz ), name="C5")
    m.addConstrs((profundidad[i]-profundidad[j]+(nodes-1)*alfa[i,j]<=(nodes-2)
             for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
            name="C6")
    fun_obj = 0
    for i,j in alfa :
        fun_obj = fun_obj + alfa[i,j]*arcos_costes[i,j]
    m.setObjective(fun_obj, GRB.MINIMIZE)
    m.update()
    m.optimize()
    optimo=str(m.objVal)
    tiempo=str(m.runtime)

f=open("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\OptimosTFG_MTZ_NoReducidos.tx
t", "a")
    f.write(problema)
    f.write("      ")
    f.write(optimo)
    f.write("      ")
    f.write(tiempo)
```

```
    f.write("\n")
    f.close()
```

9.2. Código en Jupyter Notebook para automatizar la resolución de la batería de problemas no reducidos con el modelo DL

```
from gurobipy import*

problemas=["steinb0.txt","steinb1.txt","steinb2.txt","steinb3.txt","steinb4.txt","steinb5.t
xt","steinb6.txt","steinb7.txt","steinb8.txt","steinb9.txt","steinb10.txt","steinb11.txt","st
einb12.txt","steinb13.txt","steinb14.txt","steinb15.txt","steinb16.txt","steinb17.txt","stei
nb18.txt","steinc1.txt","steinc2.txt","steinc3.txt","steinc4.txt","steinc5.txt","steinc6.txt",
"steinc7.txt","steinc8.txt","steinc9.txt","steinc10.txt","steinc11.txt","steinc12.txt","stein
c13.txt","steinc14.txt","steinc15.txt","steinc16.txt","steinc17.txt","steinc19.txt","steinc2
0.txt"]

for problema in problemas :
    lines = open ("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\"+problema)
    arcos_costes = dict()
    contador=0
    num_nodos_terminales=0
    nodos_terminales=list()
    arcos=list()
    for line in lines :
        b=line.split()
        if contador == 0 :
            nodes=int(b[0])
            arcs=int(b[1])
        elif contador >=1 and contador <=arcs :
            arcos_costes[(int(b[0]),int(b[1]))]=int(b[2])
            arcos_costes[(int(b[1]),int(b[0]))]=int(b[2])


        elif contador == arcs+1 :
            num_nodos_terminales=int(b[0])
        else :
            for i in range (len(b)):
                nodos_terminales.append(int(b[i]))
```

```
    contador = contador + 1
nodos=list()
for i in range(1,nodes+1) :
    nodos.append(i)
arcos=tuplelist(arcos_costes.keys())
arcos_entrantes=dict()
for i in nodos :
    seleccionados = tuplelist(arcos.select('*',i))
    arcos_entrantes[i]=len(seleccionados)
nodo_raiz=list()
nodos_terminales.sort()
nodo_raiz.append(nodos_terminales[0])
nodos_steiner = list()
for i in range(1,nodes+1) :
    nodos_steiner.append(i)
for i in nodos_terminales :
    nodos_steiner.remove(i)
nodos_terminales_no_raiz=list()
for i in nodos_terminales :
    nodos_terminales_no_raiz.append(int(i))
nodos_terminales_no_raiz.remove(nodo_raiz[0])
nodos_no_raiz=list()
for i in range(1,nodes+1) :
    nodos_no_raiz.append(i)
nodos_no_raiz.remove(nodo_raiz[0])
m=Model()
alfa=m.addVars(arcos_costes.keys(), vtype=GRB.BINARY, name="alfa")
m.update()
profundidad = m.addVars(nodos_no_raiz, name="X")
m.update()
```

```
    auxiliar= m.addVars(nodos_steiner, vtype=GRB.BINARY, name="B")
    m.update()
    m.addConstrs((alfa.sum(i,'*') == 1 for i in nodos_terminales_no_raiz), name="C")
    m.addConstrs((alfa.sum('*',i) <= arcos_entrantes[i]*auxiliar[i] for i in nodos_steiner),
    name="C1")
    m.addConstrs((alfa.sum(i,'*') >= auxiliar[i] for i in nodos_steiner), name="C3")
    m.addConstrs((profundidad[i]<=nodes-1 for i in nodos_no_raiz ), name="C4")
    m.addConstrs((profundidad[i]>=1 for i in nodos_no_raiz ), name="C5")
    m.addConstrs((profundidad[i]-profundidad[j]+(nodes-3)*alfa[i,j]+(nodes-
1)*alfa[j,i]<=(nodes-2)
                for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
          name="C6")
    m.addConstrs((profundidad[j]-profundidad[i]+(nodes-1)*alfa[i,j]+(nodes-
3)*alfa[j,i]<=(nodes-2)
                for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
          name="C7")
    fun_obj = 0
    for i,j in alfa :
       fun_obj = fun_obj + alfa[i,j]*arcos_costes[i,j]
    m.setObjective(fun_obj, GRB.MINIMIZE)
    m.update()
    m.optimize()
    optimo=str(m.objVal)
    tiempo=str(m.runtime)
    f=open("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\Optimos_DL_NoReducidos.txt",
   "a")
    f.write(problema)
    f.write("      ")
    f.write(optimo)
    f.write("      ")
    f.write(tiempo)
```

```
    f.write("\n")
    f.close()
```

9.3. Código en Jupyter Notebook para automatizar la resolución de la batería de problemas reducidos con el modelo MTZ

```
from gurobipy import*

problemas=["steinb1.txt","steinb2.txt","steinb3.txt","steinb4.txt","steinb5.txt","steinb6.t
xt","steinb7.txt","steinb8.txt","steinb9.txt","steinb10.txt","steinb11.txt","steinb12.txt","s
teinb13.txt","steinb14.txt","steinb15.txt","steinb16.txt","steinb17.txt","steinb18.txt","ste
inc1.txt","steinc2.txt","steinc3.txt","steinc4.txt","steinc5.txt","steinc6.txt","steinc7.txt","
steinc8.txt","steinc9.txt","steinc10.txt","steinc11.txt","steinc12.txt","steinc13.txt","stein
c14.txt","steinc15.txt","steinc16.txt","steinc17.txt","steinc19.txt","steinc20.txt",
"steind1.txt","steind2.txt","steind3.txt","steind4.txt","steind5.txt","steind6.txt","steind7.t
xt","steind9.txt","steind10.txt","steind11.txt","steind12.txt","steind13.txt","steind15.txt"
,"steind16.txt","steind17.txt","steind18.txt","steind19.txt","steind20.txt"]

for problema in problemas :
    lines = open ("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\Reducidos\\"+problema)
    arcos_costes = dict()
    contador=0
    num_nodos_terminales=0
    nodos_terminales=list()
    arcos=list()
    for line in lines :
        b=line.split()
        if contador == 0 :
            nodes=int(b[0])
            arcs=int(b[1])
        elif contador >=1 and contador <=arcs :
            arcos_costes[(int(b[0]),int(b[1]))]=int(b[2])
            arcos_costes[(int(b[1]),int(b[0]))]=int(b[2])


        elif contador == arcs+1 :
            num_nodos_terminales=int(b[0])
```

```
    elif contador == arcs+2 :
        for i in range (len(b)):
            nodos_terminales.append(int(b[i]))
    else :
        diferencia_fo=int(b[0])
    contador = contador + 1
nodos=list()
for i in range(1,nodes+1) :
    nodos.append(i)
arcos=tuplelist(arcos_costes.keys())
arcos_entrantes=dict()
for i in nodos :
    seleccionados = tuplelist(arcos.select('*',i))
    arcos_entrantes[i]=len(seleccionados)
nodo_raiz=list()
nodos_terminales.sort()
nodo_raiz.append(nodos_terminales[0])
nodos_steiner = list()
for i in range(1,nodes+1) :
    nodos_steiner.append(i)
for i in nodos_terminales :
    nodos_steiner.remove(i)
nodos_terminales_no_raiz=list()
for i in nodos_terminales :
    nodos_terminales_no_raiz.append(int(i))
nodos_terminales_no_raiz.remove(nodo_raiz[0])
nodos_no_raiz=list()
for i in range(1,nodes+1) :
    nodos_no_raiz.append(i)
nodos_no_raiz.remove(nodo_raiz[0])
```

```
    m=Model()
    alfa=m.addVars(arcos_costes.keys(), vtype=GRB.BINARY, name="alfa")
    m.update()
    profundidad = m.addVars(nodos_no_raiz, name="X")
    m.update()
    auxiliar= m.addVars(nodos_steiner, vtype=GRB.BINARY, name="B")
    m.update()
    m.addConstrs((alfa.sum(i,'*') == 1 for i in nodos_terminales_no_raiz), name="C")
    m.addConstrs((alfa.sum('*',i) <= arcos_entrantes[i]*auxiliar[i] for i in nodos_steiner),
    name="C1")
    m.addConstrs((alfa.sum(i,'*') >= auxiliar[i] for i in nodos_steiner), name="C3")
    m.addConstrs((profundidad[i]<=nodes-1 for i in nodos_no_raiz ), name="C4")
    m.addConstrs((profundidad[i]>=1 for i in nodos_no_raiz ), name="C5")
    m.addConstrs((profundidad[i]-profundidad[j]+(nodes-1)*alfa[i,j]<=(nodes-2)
             for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
            name="C6")
    fun_obj = 0
    for i,j in alfa :
       fun_obj = fun_obj + alfa[i,j]*arcos_costes[i,j]
    m.setObjective(fun_obj, GRB.MINIMIZE)
    m.update()
    m.optimize()
    optimo=str(m.objVal+diferencia_fo)
    tiempo=str(m.runtime)

f=open("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\OptimosTFG_MTZ_Reducidos.txt",
    "a")
    f.write(problema)
    f.write("      ")
    f.write(optimo)
    f.write("      ")
```

```
    f.write(tiempo)
    f.write("\n")
    f.close()
```

9.4. Código en Jupyter Notebook para automatizar la resolución de la batería de problemas reducidos con el modelo DL

```
from gurobipy import*

problemas=["steinb1.txt","steinb2.txt","steinb3.txt","steinb4.txt","steinb5.txt","steinb6.t
xt","steinb7.txt","steinb8.txt","steinb9.txt","steinb10.txt","steinb11.txt","steinb12.txt","s
teinb13.txt","steinb14.txt","steinb15.txt","steinb16.txt","steinb17.txt","steinb18.txt","ste
inc1.txt","steinc2.txt","steinc3.txt","steinc4.txt","steinc5.txt","steinc6.txt","steinc7.txt","
steinc8.txt","steinc9.txt","steinc10.txt","steinc11.txt","steinc12.txt","steinc13.txt","stein
c14.txt","steinc15.txt","steinc16.txt","steinc17.txt","steinc19.txt","steinc20.txt",
"steind1.txt","steind2.txt","steind3.txt","steind4.txt","steind5.txt","steind6.txt","steind7.t
xt","steind9.txt","steind10.txt","steind11.txt","steind12.txt","steind13.txt","steind15.txt"
,"steind16.txt","steind17.txt","steind18.txt","steind19.txt","steind20.txt"]

for problema in problemas :
    lines = open ("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\Reducidos\\"+problema)
    arcos_costes = dict()
    contador=0
    num_nodos_terminales=0
    nodos_terminales=list()
    arcos=list()
    for line in lines :
        b=line.split()
        if contador == 0 :
            nodes=int(b[0])
            arcs=int(b[1])
        elif contador >=1 and contador <=arcs :
            arcos_costes[(int(b[0]),int(b[1]))]=int(b[2])
            arcos_costes[(int(b[1]),int(b[0]))]=int(b[2])


        elif contador == arcs+1 :
            num_nodos_terminales=int(b[0])
```

```
    elif contador == arcs+2 :
        for i in range (len(b)):
            nodos_terminales.append(int(b[i]))
    else :
        diferencia_fo=int(b[0])
    contador = contador + 1
nodos=list()
for i in range(1,nodes+1) :
    nodos.append(i)
arcos=tuplelist(arcos_costes.keys())
arcos_entrantes=dict()
for i in nodos :
    seleccionados = tuplelist(arcos.select('*',i))
    arcos_entrantes[i]=len(seleccionados)
nodo_raiz=list()
nodos_terminales.sort()
nodo_raiz.append(nodos_terminales[0])
nodos_steiner = list()
for i in range(1,nodes+1) :
    nodos_steiner.append(i)
for i in nodos_terminales :
    nodos_steiner.remove(i)
nodos_terminales_no_raiz=list()
for i in nodos_terminales :
    nodos_terminales_no_raiz.append(int(i))
nodos_terminales_no_raiz.remove(nodo_raiz[0])
nodos_no_raiz=list()
for i in range(1,nodes+1) :
    nodos_no_raiz.append(i)
nodos_no_raiz.remove(nodo_raiz[0])
```

```
    m=Model()
    alfa=m.addVars(arcos_costes.keys(), vtype=GRB.BINARY, name="alfa")
    m.update()
    profundidad = m.addVars(nodos_no_raiz, name="X")
    m.update()
    auxiliar= m.addVars(nodos_steiner, vtype=GRB.BINARY, name="B")
    m.update()
    m.addConstrs((alfa.sum(i,'*') == 1 for i in nodos_terminales_no_raiz), name="C")
    m.addConstrs((alfa.sum('*',i) <= arcos_entrantes[i]*auxiliar[i] for i in nodos_steiner),
    name="C1")
    m.addConstrs((alfa.sum(i,'*') >= auxiliar[i] for i in nodos_steiner), name="C3")
    m.addConstrs((profundidad[i]<=nodes-1 for i in nodos_no_raiz ), name="C4")
    m.addConstrs((profundidad[i]>=1 for i in nodos_no_raiz ), name="C5")
    m.addConstrs((profundidad[i]-profundidad[j]+(nodes-3)*alfa[i,j]+(nodes-
1)*alfa[j,i]<=(nodes-2)
                for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
          name="C6")
    m.addConstrs((profundidad[j]-profundidad[i]+(nodes-1)*alfa[i,j]+(nodes-
3)*alfa[j,i]<=(nodes-2)
                for i,j in alfa if i != nodo_raiz[0] if j!=nodo_raiz[0]),
          name="C7")
    fun_obj = 0
    for i,j in alfa :
       fun_obj = fun_obj + alfa[i,j]*arcos_costes[i,j]
    m.setObjective(fun_obj, GRB.MINIMIZE)
    m.update()
    m.optimize()
    optimo=str(m.objVal+diferencia_fo)
    tiempo=str(m.runtime)
    f=open("C:\\Users\\rafit\\Documents\\ETSI\\TFG\\OptimosTFG_DL_Reducidos.txt",
    "a")
```

```
f.write(problema)
f.write("      ")
f.write(optimo)
f.write("      ")
f.write(tiempo)
f.write("\n")
f.close()
```