Grado en Ingeniería del Software

Investigación Operativa

Presentación

URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Profesores

- Curso académico 2024/2025

Nicolás Rodríguez Uribe
nicolas.rodriguez@urjc.es
Despacho 114, Departamental II, Móstoles

- En caso de querer concertar una tutoría con cualquiera de los dos se recomienda escribir con tiempo a través de Correo de Aula Virtual.
- Recordad también que el correo electrónico es un medio de comunicación asíncrono. No hay ninguna obligación de responder inmediatamente y menos aún fuera de la jornada laboral.

Temario

BLOQUE I. INTRODUCCIÓN.

Tema 1. Introducción a la investigación operativa. Historia y desarrollo de la Investigación Operativa. Metodología y formulación de modelos. Aplicaciones.

BLOQUE II. MODELOS DETERMINISTAS.

Tema 2. Optimización Lineal. Introducción. Formulación de modelos. Solución gráfica. Algoritmos de resolución.

Software de optimización. Análisis de la solución. Dualidad e interpretación económica.

Tema 3. Optimización Lineal Entera y Combinatoria. Introducción. Modelos de programación entera, binaria y mixta.

Aplicaciones. Modelizaciones fuertes y modelizaciones débiles. Preproceso. Algoritmos de resolución.

BLOQUE III. MODELOS ESTOCÁSTICOS.

Tema 4. Fenómenos de espera. Introducción al estudio de los fenómenos de espera. Elementos de la teoría de colas: los procesos estocásticos. Medidas de eficiencia. Modelos poissonianos. Redes de colas.

Tema 5. Simulación. Introducción. Simulación de sistemas de sucesos discretos. Generación de números y variables aleatorias. Simulación de Montecarlo. Aplicaciones a la fiabilidad y los fenómenos de espera.

Evaluación

Convocatoria ordinaria

La asignatura se evaluará mediante evaluación continua. Consultar la Guía Docente para mayor detalle:

- Examen (40%): Todo el temario. Fecha oficial convocatoria ordinaria.
- Ejercicio 1 (10%): Bloque II. Fecha tentativa: 18 de noviembre de 2024.
- Ejercicio 2 (10%): Bloque III. Fecha tentativa: 13 de diciembre de 2024.
- Práctica 1 (20%): Bloque II. Fecha tentativa: 22-25 de noviembre de 2024.
- Práctica 2 (20%): Bloque III. Fecha tentativa: 16-20 de diciembre de 2024.
- Cada uno de los apartados antes mencionados tiene una nota mínima de 4.

Evaluación

Cálculo de la nota final

La nota final se calcula como la media ponderada de las notas de las pruebas evaluables según los porcentajes indicados, siempre y cuando se hayan superado con la nota mínima indicada para cada una de ellas.

Si alguna de las pruebas evaluables no se ha superado con la nota mínima necesaria para hacer media, la nota final de la asignatura será el mínimo entre 4,9 y el valor que da la media ponderada.

Si no se ha presentado a alguna de las pruebas evaluables en la convocatoria, dicha prueba se puntuará con un 0 y la nota final de la asignatura será el mínimo entre 4,9 y el valor que da la media ponderada.

Si no se ha presentado a ninguna prueba evaluable, la nota final será "No presentado".

En todo caso, para aprobar la asignatura será necesario tener una nota final superior o igual a 5 puntos.

Evaluación

Convocatoria extraordinaria

En convocatoria extraordinaria los estudiantes solo se podrán presentar a la revaluación de las pruebas no superadas (las no presentadas y las que no alcanzaron la nota mínima). El cálculo de la nota final se realizará con los mismos pesos y las mismas notas mínimas que en el apartado anterior.

En el caso especial de que en la convocatoria ordinaria se alcanzasen las notas mínimas de todas las pruebas, pero la nota final fuera inferior a 5 puntos, para la convocatoria extraordinaria los estudiantes se podrán presentar a subir nota a alguna o a todas las pruebas reevaluables en las que hubieran sacado menos de 5 puntos, según su elección, para conseguir aprobar la asignatura.

En el caso especial de que incluso pudiendo sacar un 10 en todas las pruebas revaluadas en la convocatoria extraordinaria fuera imposible obtener una nota final igual o superior a 5 puntos, los estudiantes se podrán presentar a subir nota a alguna o a todas las pruebas superadas en la convocatoria ordinaria, según su elección.

En todos los casos, para el cálculo de la nota final de la convocatoria extraordinaria se utilizará la calificación de las pruebas superadas en convocatoria ordinaria y las notas obtenidas en las pruebas revaluadas, incluso si estas últimas fueran inferiores a las obtenidas en la convocatoria ordinaria.

La revaluación del examen se realizará en la fecha oficial indicada para la convocatoria extraordinaria. En cuanto a la práctica, de los problemas y de los casos prácticos, se planificarán plazos de entrega a determinar dentro de las fechas de exámenes de la convocatoria extraordinaria.

Objetivo

- Una fábrica produce dos tipos de cerveza: rubia y tostada.
- La fábrica debe decidir cuántos litros de cada tipo debe producir semanalmente teniendo en cuenta que 1.000 litros de cerveza rubia se venden a 100 euros y 1.000 litros de cerveza tostada a 125 euros.
- Para producir 1.000 litros de cerveza rubia se necesitan 3 empleados; para lo mismo de cerveza tostada, 5.
- La fábrica sólo dispone de 15 empleados.
- La compra de materias primas supone un precio de 90€ por cada 1.000 litros de cerveza rubia, y 85€ para la tostada. Se dispone de 350€ semanales para este concepto.

Objetivo

Variables de decisión

- x_1 = litros de cerveza rubia (miles)
- x_2 = libros de cerveza tostada (miles)

Restricciones

- No emplear más empleados de los 15 disponibles:

$$3x_1 + 5x_2 \leq 15$$

- No superar el presupuesto en la compra de materias primas

$$90\ x_1 + 85x_2 \leq 350$$

- La producción es positiva

$$x_1, x_2 \geq 0$$

Función Objetivo

- Maximizar el beneficio obtenido

$$\max 100x_1 + 125x_2$$

Grado en Ingeniería del Software

Investigación Operativa

Presentación

URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Grado en Ingeniería del Software

Investigación Operativa

BLOQUE I. INTRODUCCIÓN

Tema 1: Introducción a la Investigación Operativa

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Índice

Introducción

- La Investigación Operativa(IO) es una disciplina científica que utiliza modelos matemáticos, estadísticos y técnicas analíticas avanzadas para ayudar en la toma de decisiones óptimas.
- Su objetivo es estudiar sistemas complejos y encontrar la mejor manera de organizar, gestionar y utilizar recursos limitados para maximizar resultados, minimizar costes, o alcanzar otros objetivos específicos.

Historia de la Investigación Operativa

- Siglo XVII: Se desarrollan conceptos fundamentales sobre la probabilidad y toma de decisiones bajo incertidumbre, por parte de Blaise Pascal y Pierre de Fermat.

- Siglos XVIII y XIX: Leonhard Euler y Carl Friedrich Gauss hicieron contribuciones importantes en la teoría de redes y estadísticas.

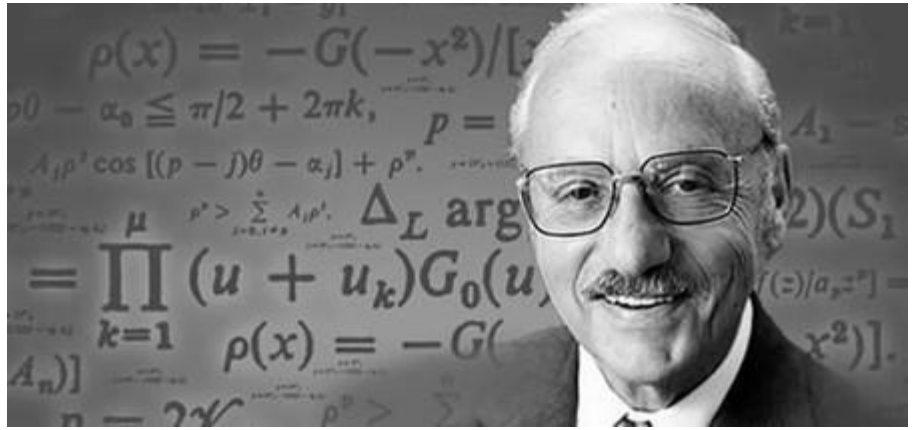
Historia de la Investigación Operativa

- Segunda Guerra Mundial: Auge de la IO.
- La IO nació como una herramienta para mejorar el uso de **recursos limitados**, como aviones, submarinos, radares, y personal militar.
- El primer caso de éxito fue en el Reino Unido, donde se utilizó para optimizar la ubicación y uso de los radares durante la Batalla de Inglaterra, lo que permitió una defensa más eficiente contra los bombardeos alemanes.

Historia de la Investigación Operativa

Postguerra: expansión y formalización (1940s-1960s)

- **Programación lineal:** Formulada por George Dantzig en 1947, es uno de los métodos más importantes de la IO para optimizar problemas con restricciones lineales.



Historia de la Investigación Operativa

Postguerra: expansión y formalización (1940s-1960s)

- **Teoría de colas:** Desarrollada por Agner Krarup Erlang y utilizada para optimizar sistemas de espera y servicios
- **Teoría de juegos:** John von Neumann y Oskar Morgenstern, tiene aplicaciones en la toma de decisiones estratégicas.
- **Simulación por Monte Carlo:** Stanislaw Ulam et al., (Proyecto Manhattan), se usa para modelar sistemas complejos y tomar decisiones bajo incertidumbre.

Historia de la Investigación Operativa

Crecimiento y aplicación en el sector privado (1960s-1980s)

- La programación lineal y la teoría de inventarios son esenciales en la planificación de producción.
- La simulación y la optimización son esenciales en redes de transporte y la gestión de recursos.

Historia de la Investigación Operativa

Era de la informática y la globalización (1980s-presente)

- **Optimización combinatoria:** planificación de rutas y diseño de redes de distribución.
- **Algoritmos genéticos y redes neuronales:** hibridación de técnicas de inteligencia artificial.
- **Big Data y análisis predictivo:** integración de IO con análisis de grandes volúmenes de datos para mejorar la predicción y la toma de decisiones en tiempo real.

Metodología de la Investigación Operativa

- **Definición del problema.**
- Modelado del problema.
- Búsqueda de la solución óptima.
- Validación del modelo.
- Implementación de la solución.

Metodología de la Investigación Operativa

- **Objetivos**: ¿Qué se desea optimizar?
  - Minimización de costes.
  - Maximización de beneficios.
  - Reducción de tiempos.
  - Mejora en la calidad.

- **Variables y parámetros:** Identificación de las variables que controlan el sistema y las relaciones entre ellas.
  - Variables de decisión (las que pueden controlarse)
  - Parámetros que afectan al sistema pero no pueden modificarse.

Metodología de la Investigación Operativa

- **Restricciones**: Limitaciones del sistema que restringen las decisiones posibles. Pueden ser:
  - Físicas (capacidad de producción).
  - Económicas (presupuesto disponible).
  - Legales (regulaciones).
  - Recursos (número limitado de trabajadores o materiales).
- **Ejemplo**:
  - Supongamos que el problema es optimizar la distribución de productos de una fábrica a múltiples tiendas, minimizando los costes de transporte. En este caso:
  - Función objetivo: **minimización de costes**.
  - Variables: rutas de transporte y la cantidad de productos enviados.
  - Restricciones: capacidad de los vehículos y la demanda de las tiendas.

Metodología de la Investigación Operativa

- Definición del problema.
- **Modelado del problema.**
- Búsqueda de la solución óptima.
- Validación del modelo.
- Implementación de la solución.

Metodología de la Investigación Operativa

- Modelos matemáticos:
  - Se representan mediante ecuaciones matemáticas que describen las relaciones entre las variables.
  - Formulación de una **función objetivo** (que se quiere optimizar) y un conjunto de **restricciones** que limitan el espacio de soluciones.
- Tipos de modelos:
  - Lineales, donde la función objetivo y las restricciones son lineales*.
  - No lineales, donde alguna relación no es lineal.
  - Estocásticos, si el problema incluye incertidumbre o probabilidades.
- ***Las ecuaciones de la función objetivo y las restricciones no contienen potencias, productos entre variables, ni funciones no lineales como logaritmos o exponenciales.**

Metodología de la Investigación Operativa

- Definición del problema.
- Modelado del problema.
- **Búsqueda de la solución óptima.**
- Validación del modelo.
- Implementación de la solución.

Metodología de la Investigación Operativa

- Algoritmos de optimización:
  - **Método Simplex:** Muy utilizado en programación lineal.
  - **Programación entera**: Para problemas donde algunas o todas las variables deben tomar valores enteros.
  - **Heurísticas y metaheurísticas**: Utilizadas para problemas grandes o complejos donde los métodos exactos son computacionalmente ineficientes.

- Soluciones factibles vs óptimas:
  - La solución **factible** es aquella que satisface todas las restricciones.
  - La solución **óptima** es la mejor solución factible en términos de la función objetivo.

Metodología de la Investigación Operativa

- Definición del problema.
- Modelado del problema.
- Búsqueda de la solución óptima.
- **Validación del modelo.**
- Implementación de la solución.

Metodología de la Investigación Operativa

- Comprobación de resultados:
  - Se comparan los resultados del modelo con los datos históricos o con situaciones reales para verificar su consistencia.
- Análisis de sensibilidad:
  - Se evalúa cómo la solución óptima varía en respuesta a cambios en los parámetros del modelo.
  - Es importante entender la robustez del modelo y las posibles implicaciones de la incertidumbre en los datos o en las condiciones cambiantes del sistema.
- Ejemplo:
  - En el caso de la distribución de productos, se puede verificar si la solución es efectiva comparándola con datos de entregas anteriores.
  - También se podría realizar un análisis de sensibilidad para observar qué sucede si cambian los costes de transporte o las demandas de las tiendas.

Metodología de la Investigación Operativa

- Definición del problema.
- Modelado del problema.
- Búsqueda de la solución óptima.
- Validación del modelo.
- **Implementación de la solución.**

Metodología de la Investigación Operativa

- Aplicación práctica:
  - Adaptación de la solución del modelo matemático al entorno real, considerando las limitaciones operativas y prácticas.
- Monitorización y ajustes:
  - Después de la implementación, es necesario la monitorización el desempeño del sistema y ajustar la solución si es necesario.
  - En algunos casos, los datos en tiempo real pueden retroalimentar el modelo para mejorar continuamente la optimización.
- Ejemplo:
  - En la distribución de productos, la implementación implicaría organizar las rutas de los camiones según la solución obtenida, ajustando las rutas si las condiciones cambian (como el clima o el tráfico) y midiendo los ahorros obtenidos en costes de transporte.

Casos de éxito

- **American Airlines (Optimización de rutas y precios):**
  - **Optimización de precios:** En los años 90, American Airlines implementó un sistema de fijación de precios dinámico basado en modelos de yield management, lo que les permitió ajustar los precios de los billetes en tiempo real, según la demanda. Este sistema optimizado generó más de **500 millones de dólares** adicionales en ingresos durante su primer año de implementación.
  - **Planificación de rutas:** Utilizando algoritmos de optimización de redes, American Airlines puede calcular las rutas más eficientes para sus vuelos, minimizando los costes de combustible y tiempo de vuelo, mientras maximiza la ocupación de los asientos.

Casos de éxito

- **Amazon: Optimización de la cadena de suministro**
  - **Almacenamiento y distribución**: Amazon utiliza modelos de programación lineal y algoritmos de optimización estocástica para decidir la ubicación de sus centros de distribución y optimizar la gestión de inventarios en tiempo real. Esto le permite prever la demanda y reubicar productos estratégicamente, asegurando que siempre haya inventario cerca del cliente.
  - **Rutas de entrega**: El sistema de planificación de rutas de Amazon, basado en algoritmos de optimización de redes, permite a la compañía organizar rutas de entrega para que sus conductores cubran las menores distancias posibles, reduciendo costes de transporte y tiempos de entrega.
- Amazon ha sido capaz de reducir el tiempo de entrega a **un solo día** en muchas áreas, gracias a la integración de la IO en sus operaciones logísticas.

Casos de éxito

- **UPS: Reducción de costes de combustible con ORION (On-Road Integrated Optimization and Navigation)**, que utiliza modelos de **programación lineal** y **teoría de grafos** para planificar las rutas de entrega de sus vehículos de manera más eficiente.
  - **Ahorro de combustible**: ORION optimiza las rutas de los conductores para minimizar el número de giros a la izquierda (que en los EE. UU. requieren más tiempo y consumo de combustible). Esta optimización ha permitido a UPS ahorrar **más de 10 millones de galones de combustible** al año y reducir sus emisiones de carbono significativamente.
  - **Reducción de costes**: El uso de ORION ha ayudado a UPS a reducir sus costes operativos en más de **300 millones de dólares** al año, mejorando la eficiencia y reduciendo los tiempos de entrega.

Casos de éxito

- **Google: Gestión de energía en centros de datos**
  - **Optimización del uso de energía**: Google utiliza **algoritmos de aprendizaje automático** combinados con modelos de programación no lineal para predecir el uso de energía y ajustar en tiempo real la refrigeración y el uso de equipos en los centros de datos. Esto ha permitido a Google reducir el consumo de energía en sus centros de datos en más del **40%**, manteniendo un alto nivel de eficiencia operativa.

Casos de éxito

- **NHS (National Health Service) en el Reino Unido: Optimización de recursos sanitarios**
  - **Gestión de camas hospitalarias**: El NHS utiliza modelos de simulación y optimización para gestionar las camas en los hospitales de manera más eficiente, equilibrando la demanda y la disponibilidad de camas para maximizar la capacidad sin comprometer la calidad del servicio. Esto ha permitido reducir los tiempos de espera de los pacientes y mejorar el flujo de trabajo en hospitales de alta demanda.
  - **Optimización de programación quirúrgica**: Mediante la aplicación de algoritmos de programación, el NHS ha logrado optimizar las listas de espera y la programación de cirugías, lo que ha permitido realizar más operaciones dentro de un tiempo determinado, aumentando la productividad de los quirófanos y mejorando la atención al paciente.

Casos de éxito

- **Walmart: Optimización del inventario**
  - **Gestión de inventarios**: Walmart utiliza modelos de **programación estocástica** y simulaciones para optimizar el inventario en sus tiendas, asegurando que los productos estén disponibles cuando los clientes los necesitan, pero sin acumular exceso de inventario. Esta optimización ha ayudado a Walmart a reducir sus costes de almacenamiento mientras mejora la disponibilidad de productos.
  - **Precios dinámicos**: Walmart también aplica la IO para ajustar los precios en tiempo real, basándose en la demanda, competencia y costes, lo que le permite maximizar su margen de ganancias mientras mantiene precios competitivos.

Bibliografía

- Hillier & Liebermann (2010). Introducción a la investigación de operaciones, 9ªed. Capítulos 1 y 2.
- González y García (2015). Manual práctico de investigación de operaciones 4ª ed. Capítulo 1.
- Investigación Operativa (2004). Modelos determinísticos y estocásticos. Sixto Ríos Insua, Alfonso Mateosos Caballero, Mª Concepción Bielza Lozoya y Antonio Jiménez Martín, Editorial Centro de Estudios Ramón Areces, S.A., Madrid.

Bibliografía

- Investigación Operativa. Optimización, Sixto Ríos Insua. Editorial Centro de Estudios Ramón Areces, S.A., Madrid.
- Problemas de Investigación Operativa. Programación Lineal y Extensiones. Sixto Ríos Insua, David Ríos Insua, Alfonso Mateos Caballero, Jacinto Martín Jiménez y Antonio Jiménez Martín. Editorial Ra-Ma, Madrid 2006.
- Model Building in Mathematical Programming. Paul W. Williams. Editorial JOHN WILEY AND SONS
- Apuntes de Victoria Ruiz y Antonio Alonso
- Apuntes de Alberto Herrán y José J. Ruz Ortiz
- Apuntes de Gerardo Reyes y Salvador Sánchez

Grado en Ingeniería del Software

Investigación Operativa

BLOQUE I. INTRODUCCIÓN

Tema 1: Introducción a la Investigación Operativa

URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Grado en Ingeniería del Software

Investigación Operativa

BLOQUE II. MODELOS DETERMINISTAS

Tema 2: Optimización Lineal Continua

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Índice

Propiedades del modelo lineal

- Proporcionalidad
  - La contribución al coste y a las restricciones es directamente proporcional al valor de las variables de decisión.
- Aditividad
  - El coste y las restricciones son la suma directa de los valores aportados por las variables de decisión.
- Divisibilidad
  - Las variables de decisión pueden dividirse en cualquier tipo de fracción, es decir, toman como valores números reales.
- Determinismo
  - Los valores que se relacionan con las variables de decisión mantienen su valor constante.

Asignación de recursos

- En una fábrica de cerveza se producen tres tipos distintos: rubia, negra y de baja graduación, y para ello se utilizan dos materias primas: malta y levadura.
- En la siguiente tabla se especifican:
  - a) la cantidad de materias primas para producir una unidad de cada tipo de cerveza;
  - b) las cantidades disponibles de cada materia prima; y
  - c) el precio unitario de venta de cada tipo de cerveza.

	Consumo de materias primas por cada tipos de cerveza			
Materia prima	*rubia*	*negra*	*baja*	Disponibilidad
malta	1	2	2	30
levadura	2	1	2	45
Precio de venta	7	4	3	

Asignación de recursos

- Se busca maximizar el beneficio
- Variables de decisión
  - x_1 = producción de cerveza rubia
  - x_2 = producción de cerveza negra
  - x_3 = producción de cerveza de baja graduación
- Función objetivo
  - $\max 7x_1 + 4x_2 + 3x_3$
- Restricciones:
  - $x_1 + 2x_2 + 2x_3 \leq 30$
  - $2x_1 + x_2 + 2x_3 \leq 45$
  - $x_1, x_2, x_3 \geq 0$

```python
# Importar el módulo gurobipy
from gurobipy import Model, GRB

# Crear un nuevo modelo
modelo = Model("Asignación de recursos")

# Definir las variables de decisión
# x: miles de litros de cerveza rubia a producir (>= 0)
# y: miles de litros de cerveza negra a producir (>= 0)
# z: miles de litros de cerveza baja a producir (>= 0)
x = modelo.addVar(name="Cerveza_Rubia", lb=0)
y = modelo.addVar(name="Cerveza_Negra", lb=0)
z = modelo.addVar(name="Cerveza_Baja", lb=0)

# Establecer la función objetivo: Maximizar Z = 7x + 4y + 3z
modelo.setObjective(7 * x + 4 * y + 3 * z, GRB.MAXIMIZE)

# Agregar las restricciones

# Restricción de presupuesto para materias primas: x + 2y + 2z ≤ 30
modelo.addConstr(x + 2 * y + 2 * z <= 30, name="Restricción_Malta")
modelo.addConstr(2 * x + y + 2 * z <= 45, name="Restricción_Levadura")

# Optimizar el modelo
modelo.optimize()

# Verificar si se encontró una solución óptima
if modelo.status == GRB.OPTIMAL:
    # Imprimir los resultados
    print(f"Litros de cerveza rubia a producir: {x.X:.2f} litros")
    print(f"Litros de cerveza negra a producir: {y.X:.2f} litros")
    print(f"Litros de cerveza baja a producir: {z.X:.2f} litros")
    print(f"Beneficio máximo: {modelo.ObjVal:.2f} euros")
else:
    print("No se encontró una solución óptima.")
```

Fábrica de cerveza

- Una fábrica de cerveza produce dos tipos: rubia y negra. Cada una de ellas tiene un proceso de fabricación distinto.
- La fábrica debe decidir cuántos litros de cerveza debe producir semanalmente teniendo en cuenta que 1000 litros de cerveza rubia se venden a 100 euros y 1000 litros de cerveza negra a 125 euros.
- Para producir 1000 litros de cerveza rubia se necesitan 3 empleados y para la cerveza negra 5 empleados. La fábrica solo dispone de 15 empleados.
- La compra de materias primas supone para el fabricante un precio de 90 euros por cada 1000 litros de cerveza rubia. Para la cerveza negra este coste asciende a 85 euros. Se dispone de 350 euros semanales para este concepto.
- El problema que se plantea la gerente de la fábrica es determinar cuántos litros de cerveza debe producir teniendo en cuenta las condiciones anteriores.

Fábrica de cerveza

$$Maximizar\ Z = 100x + 125y$$
$$Sujeto\ a:$$
$$3x + 5y \leq 15$$
$$90x + 85y \leq 350$$
$$x \geq 0$$
$$y \geq 0$$

Litros de cerveza rubia a producir: 2.44 litros
Litros de cerveza negra a producir: 1.54 litros
Ingresos máximos: 435.90 euros

```python
# Importar el módulo gurobipy
from gurobipy import Model, GRB

# Crear un nuevo modelo
modelo = Model("Produccion_Cerveza")

# Definir las variables de decisión
# x: Miles de litros de cerveza rubia a producir (>= 0)
# y: Miles de litros de cerveza negra a producir (>= 0)
x = modelo.addVar(name="Cerveza_Rubia", lb=0)
y = modelo.addVar(name="Cerveza_Negra", lb=0)

# Establecer la función objetivo: Maximizar Z = 100x + 125y
modelo.setObjective(100 * x + 125 * y, GRB.MAXIMIZE)

# Agregar las restricciones
# Restricción de empleados: 3x + 5y ≤ 15
modelo.addConstr(3 * x + 5 * y <= 15, name="Restriccion_Empleados")

# Restricción de presupuesto para materias primas: 90x + 85y ≤ 350
modelo.addConstr(90 * x + 85 * y <= 350, name="Restriccion_MateriasPrimas")

# Optimizar el modelo
modelo.optimize()

# Verificar si se encontró una solución óptima
if modelo.status == GRB.OPTIMAL:
    # Imprimir los resultados
    print(f"Litros de cerveza rubia a producir: {x.X * 1000:.2f} litros")
    print(f"Litros de cerveza negra a producir: {y.X * 1000:.2f} litros")
    print(f"Ingresos máximos: {modelo.ObjVal:.2f} euros")
else:
    print("No se encontró una solución óptima.")
```

Petroquímica

- Una petroquímica se dedica a producir plástico para envases (PET-envase) y fibra sintética (PET-fibra), a partir de PTA (Ácido Tereftálico Purificado) y de MEG (Monoetilenglicol).
- Las cantidades, expresadas en toneladas, de cada materia prima necesarias para producir una tonelada de cada uno de estos productos son:

Materia prima	PET-envase	PET-fibra
PTA	0.95	0.92
MEG	0.37	0.34

- La venta en el mercado de PET-envase y PET-fibra proporciona un beneficio de 32 euros y 38 euros por tonelada, respectivamente.
- La petroquímica dispone de 280 toneladas de PTA y 170 toneladas de MEG.
- El problema al que se enfrenta la petroquímica es distribuir los recursos entre las actividades de producción de PET-envase y PET-fibra, de manera que los beneficios de la venta de su producción en el mercado sean máximos.

Petroquímica

$$Maximizar\ Z = 32x + 38y$$
$$Sujeto\ a:$$
$$0.950x + 0.920y \leq 280$$
$$0.370x + 0.340y \leq 170$$
$$x \geq 0$$
$$y \geq 0$$

Toneladas de PET-envase a producir: 0.00 toneladas
Toneladas de PET-fibra a producir: 304.35 toneladas
Beneficio máximo: 11565.00 euros

El PET-fibra tiene un **mayor beneficio por tonelada** y **consume menos materias primas** por tonelada producida en comparación con el PET-envase. Por lo tanto, es más rentable y eficiente producir PET-fibra bajo las condiciones actuales.

```python
# Importar el módulo gurobipy
from gurobipy import Model, GRB

# Crear un nuevo modelo
modelo = Model("Producción_PET")

# Definir las variables de decisión
# x: Toneladas de PET-envase a producir (>= 0)
# y: Toneladas de PET-fibra a producir (>= 0)
x = modelo.addVar(name="PET_envase", lb=0)
y = modelo.addVar(name="PET_fibra", lb=0)

# Establecer la función objetivo: Maximizar Z = 32x + 38y
modelo.setObjective(32 * x + 38 * y, GRB.MAXIMIZE)

# Agregar las restricciones
# Restricción de PTA: 0.950x + 0.920y ≤ 280
modelo.addConstr(0.950 * x + 0.920 * y <= 280, name="Restricción_PTA")

# Restricción de MEG: 0.370x + 0.340y ≤ 170
modelo.addConstr(0.370 * x + 0.340 * y <= 170, name="Restricción_MEG")

# Optimizar el modelo
modelo.optimize()

# Verificar si se encontró una solución óptima
if modelo.status == GRB.OPTIMAL:
    # Imprimir los resultados
    print(f"Toneladas de PET-envase a producir: {x.X:.2f} toneladas")
    print(f"Toneladas de PET-fibra a producir: {y.X:.2f} toneladas")
    print(f"Beneficio máximo: {modelo.ObjVal:.2f} euros")
else:
    print("No se encontró una solución óptima.")
```

Petroquímica

- **Análisis de Resultados:**
- Consumo de PTA: $0.950x + 0.920y = 0.950(0) + 0.920(304.35)=280$ T
- Se utiliza toda la disponibilidad de PTA (280 toneladas).
- Consumo de MEG: $0.370x + 0.340y= 0.370(0) + 0.340(304.35)=103.48$ T
- Se utiliza 103.48 toneladas de MEG, quedando una disponibilidad restante de: $170 - 103.48 = 66.52$.
- Hay MEG disponible que no se utiliza, pero la disponibilidad de PTA limita la producción adicional.
- ¿Qué precio tendríamos que poner al PET-envase para que fuera rentable de fabricar?

Petroquímica

- **Paso 1: Entender por qué no se produce PET-envase**
- Beneficio por tonelada:
  - PET-envase: 32 euros/tonelada
  - PET-fibra: 38 euros/tonelada
- Consumo de materia:

Materia prima	PET-envase	PET-fibra
PTA	0.95	0.92
MEG	0.37	0.34

- Disponibilidad de materias primas:
  - PTA: 280 toneladas
  - MEG: 170 toneladas
- El PET-fibra tiene un mayor beneficio por tonelada y consume menos materias primas por tonelada producida en comparación con el PET-envase. Por lo tanto, es más rentable y eficiente producir PET-fibra bajo las condiciones actuales.

Petroquímica

- **Paso 2: Calcular el beneficio por unidad de materia prima**

- Para que el PET-envase sea competitivo, su beneficio por unidad de materia prima debe ser al menos igual al del PET-fibra.
- $\text{Beneficio}_{\text{Envase}} = \frac{32}{0.95} \approx 33.68$ euros/tonelada de PTA
- $\text{Beneficio}_{\text{Fibra}} = \frac{38}{0.92} \approx 41.3$ euros/tonelada de PTA

Petroquímica

- **Paso 3: Determinar el precio necesario para el PET-envase**

- Para que el PET-envase sea tan atractivo como el PET-fibra en términos de beneficio por tonelada de PTA, necesitamos encontrar el nuevo beneficio por tonelada de PET-envase (PP) que satisfaga la siguiente condición:

$$\frac{P}{0.95} \geq 41.30$$

Despejamos *P*

$$P \geq 41.30 * 0.95 \approx 39.24\ €/T$$

Petroquímica

$$Maximizar\ Z = 39.24x + 38y$$
$$Sujeto\ a:$$
$$0.950x + 0.920y \leq 280$$
$$0.370x + 0.340y \leq 170$$
$$x \geq 0$$
$$y \geq 0$$

Toneladas de PET-envase a producir: 294.74 toneladas
Toneladas de PET-fibra a producir: 0.00 toneladas
Beneficio máximo: 11565.47 euros

```python
# Importar el módulo gurobipy
from gurobipy import Model, GRB

# Crear un nuevo modelo
modelo = Model("Producción_PET")

# Definir las variables de decisión
# x: Toneladas de PET-envase a producir (>= 0)
# y: Toneladas de PET-fibra a producir (>= 0)
x = modelo.addVar(name="PET_envase", lb=0)
y = modelo.addVar(name="PET_fibra", lb=0)

# Establecer la función objetivo: Maximizar Z = 32x + 38y
modelo.setObjective(39.24 * x + 38 * y, GRB.MAXIMIZE)

# Agregar las restricciones
# Restricción de PTA: 0.950x + 0.920y ≤ 280
modelo.addConstr(0.950 * x + 0.920 * y <= 280, name="Restricción_PTA")

# Restricción de MEG: 0.370x + 0.340y ≤ 170
modelo.addConstr(0.370 * x + 0.340 * y <= 170, name="Restricción_MEG")

# Optimizar el modelo
modelo.optimize()

# Verificar si se encontró una solución óptima
if modelo.status == GRB.OPTIMAL:
    # Imprimir los resultados
    print(f"Toneladas de PET-envase a producir: {x.X:.2f} toneladas")
    print(f"Toneladas de PET-fibra a producir: {y.X:.2f} toneladas")
    print(f"Beneficio máximo: {modelo.ObjVal:.2f} euros")
else:
    print("No se encontró una solución óptima.")
```

Explotación minera

- Una empresa minera produce lignito y antracita. Puede vender toda la producción que obtenga de ambos minerales con un beneficio unitario de 24 y 18 euros por tonelada vendida, respectivamente. El proceso de producción está dividido en tres fases:
  1. Se corta el mineral
  2. Se procede al tamizado y a la selección
  3. Se produce el lavado.
- La producción requiere del uso de maquinaria. Las necesidades en cada una de las tres fases durante los tiempos y la disponibilidad máxima de cada tipo de máquinas son:

	Corte	Tamizado	Lavado
Lignito	3	3	4
Antracita	4	3	2
Disponibilidad máxima	12	10	8

- ¿Cuántas toneladas de cada clase de carbón debe producir al día?

Explotación minera

$$Maximizar\ Z = 24x + 18y$$
$$Sujeto\ a:$$
$$3x + 4y \leq 12$$
$$3x + 3y \leq 10$$
$$4x + 2y \leq 8$$
$$x \geq 0$$
$$y \geq 0$$

Toneladas de Lignito a producir: 0.80 toneladas
Toneladas de Antracita a producir: 2.40 toneladas
Beneficio máximo: 62.40 euros

```python
# Importar el módulo gurobipy
from gurobipy import Model, GRB

# Crear un nuevo modelo
modelo = Model("Explotación minera")

# Definir las variables de decisión
# x: Toneladas de Lignito (>= 0)
# y: Toneladas de Antracita (>= 0)
x = modelo.addVar(name="PET_envase", lb=0)
y = modelo.addVar(name="PET_fibra", lb=0)

# Establecer la función objetivo: Maximizar Z = 24x + 18y
modelo.setObjective(24 * x + 18 * y, GRB.MAXIMIZE)

# Agregar las restricciones
# Restricción del corte: 3x + 4y ≤ 12
modelo.addConstr(3 * x + 4 * y <= 12, name="Restricción_Corte")

# Restricción del tamizado 3x + 3y ≤ 10
modelo.addConstr(3 * x + 3 * y <= 10, name="Restricción_Tamizado")

#Restricción del lavado 4x + 2y ≤ 8
modelo.addConstr(4 * x + 2 * y <= 8, name="Restricción_Lavado")

# Optimizar el modelo
modelo.optimize()

# Verificar si se encontró una solución óptima
if modelo.status == GRB.OPTIMAL:
    # Imprimir los resultados
    print(f"Toneladas de Lignito a producir: {x.X:.2f} toneladas")
    print(f"Toneladas de Antracita a producir: {y.X:.2f} toneladas")
    print(f"Beneficio máximo: {modelo.ObjVal:.2f} euros")
else:
    print("No se encontró una solución óptima.")
```

Aceites

- Una fábrica produce aceite mezclando aceites refinados, dos de origen vegetal y tres de origen no vegetal.
- En un mes sólo es posible refinar 200 toneladas de vegetal y 250 de no vegetal.
- El aceite resultante debe cumplir un valor de dureza comprendido entre 3 y 6. El coste de una tonelada para cada aceite refinado junto con su dureza aparecen en la siguiente tabla:

	VEG_1	*VEG_2*	*NOVEG_1*	*NOVEG_2*	*NOVEG_3*
costo	110	120	130	110	115
dureza	8,8	6,1	2,0	4,2	5,0

- Se trata de refinar las cantidades apropiadas de cada aceite a fin de maximizar el beneficio de la producción final sabiendo que una tonelada del aceite producido se vende a 150.

Aceites

$$Maximizar\ Z =$$
$$150t - 110v_1 - 120v_2 - 130nv_1 - 110nv_2 - 115nv_3$$
$$Sujeto\ a:$$
$$v_1 + v_2 \leq 200$$
$$nv_1 + nv_2 + nv_3 \leq 250$$
$$8.8v_1 + 6.1v_2 + 2nv_1 + 4.2nv_2 + 5nv_3 \leq 6t$$
$$8.8v_1 + 6.1v_2 + 2nv_1 + 4.2nv_2 + 5nv_3 \geq 3t$$
$$v_1 + v_2 + nv_1 + nv_2 + nv_3 = t$$
$$v_1, v_2, nv_1, nv_2, nv_3 \geq 0$$

Solución Óptima:

v1 = 159.26
v2 = 40.74
nv1 = 0.00
nv2 = 250.00
nv3 = 0.00
y = 450.00
Valor óptimo de la función objetivo Z = 17592.59

Límite superior de dureza del aceite producido:

$$8.8v_1 + 6.1v_2 + 2nv_1 + 4.2nv_2 + 5nv_3 \leq 6t$$

- La **dureza total** del aceite producido no puede superar **6 unidades por unidad de producto**.
- La expresión del lado izquierdo representa la **dureza total aportada por cada tipo de aceite**, y al dividir por t (producción total), se obtiene la **dureza promedio**, que debe ser menor o igual a 6.
- De manera análoga, se repite para la cota inferior de dureza.

Relación entre variables y producción total:

$$v_1 + v_2 + nv_1 + nv_2 + nv_3 = t$$

- La **producción total** de aceite (t) es igual a la suma de las cantidades producidas de cada tipo de aceite.

Aceites

```python
# Importar el módulo gurobipy
from gurobipy import Model, GRB

# Crear un nuevo modelo
model = Model("Optimization_Model")

# Definir las variables de decisión (todas no negativas)
v1 = model.addVar(name="v1", lb=0)
v2 = model.addVar(name="v2", lb=0)
nv1 = model.addVar(name="nv1", lb=0)
nv2 = model.addVar(name="nv2", lb=0)
nv3 = model.addVar(name="nv3", lb=0)
y = model.addVar(name="y", lb=0)

# Establecer la función objetivo
model.setObjective(
    150 * y - 110 * v1 - 120 * v2 - 130 * nv1 - 110 * nv2 - 115 * nv3,
    GRB.MAXIMIZE
)

# Agregar las restricciones

# Restricción 1: v1 + v2 <= 200
model.addConstr(v1 + v2 <= 200, name="Capacidad_Vegetal")

# Restricción 2: nv1 + nv2 + nv3 <= 250
model.addConstr(nv1 + nv2 + nv3 <= 250, name="Capacidad_NoVegetal")

# Restricción 3: 8.8*v1 + 6.1*v2 + 2*nv1 + 4.2*nv2 + 5*nv3 <= 6*y
model.addConstr(
    8.8 * v1 + 6.1 * v2 + 2 * nv1 + 4.2 * nv2 + 5 * nv3 <= 6 * y,
    name="Dureza_Superior"
)
```

```python
# Restricción 4: 8.8*v1 + 6.1*v2 + 2*nv1 + 4.2*nv2 + 5*nv3 >= 3*y
model.addConstr(
    8.8 * v1 + 6.1 * v2 + 2 * nv1 + 4.2 * nv2 + 5 * nv3 >= 3 * y,
    name="Dureza_Inferior"
)

# Restricción 5: v1 + v2 + nv1 + nv2 + nv3 == y
model.addConstr(v1 + v2 + nv1 + nv2 + nv3 == y, name="Produccion_Total")

# Optimizar el modelo
model.optimize()

# Verificar si se encontró una solución óptima
if model.status == GRB.OPTIMAL:
    print("\nSolución Óptima:")
    print(f"v1 = {v1.X:.2f}")
    print(f"v2 = {v2.X:.2f}")
    print(f"nv1 = {nv1.X:.2f}")
    print(f"nv2 = {nv2.X:.2f}")
    print(f"nv3 = {nv3.X:.2f}")
    print(f"y  = {y.X:.2f}")
    print(f"Valor óptimo de la función objetivo Z = {model.ObjVal:.2f}")
else:
    print("No se encontró una solución óptima.")
```

Whiskies

- Una destilería produce whisky mezclando diferentes tipos de whiskies refinados: uno de origen escocés y dos de origen irlandés. En un mes, solo es posible refinar 200 barriles de whisky escocés y 250 barriles de whisky irlandés.
- El whisky resultante debe cumplir un valor de graduación alcohólica comprendido entre 40% y 46%. El coste por barril y la graduación alcohólica de cada tipo de whisky refinado aparecen en la siguiente tabla:

Tipo de Whisky	Origen	Coste (€)	Graduación (%)
Whisky Escocés	Escocés	300	48
Whisky Irlandés Tipo 1	Irlandés	250	38
Whisky Irlandés Tipo 2	Irlandés	270	42

- Se busca determinar las cantidades apropiadas de cada tipo de whisky a mezclar para maximizar el beneficio de la producción final, sabiendo que un barril del whisky producido se vende a 500 euros

Whiskies

$$Maximizar\ Z = 500B - 300we - 250wi_1 - 270wi_2$$

$$Sujeto\ a:$$

$$w_e \leq 200$$

$$w_{i1} + wi_2 \leq 250$$

$$48we + 38wi_1 + 42wi_2 \geq 40B$$

$$48we + 38wi_1 + 42wi_2 \leq 46B$$

$$w_e + wi_1 + wi_2 = B$$

$$w_e, wi_1, wi_2 \geq 0$$

Solución Óptima:

we (Whisky Escocés) = 200.00 barriles
wi1 (Whisky Irlandés Tipo 1) = 250.00 barriles
wi2 (Whisky Irlandés Tipo 2) = 0.00 barriles
B (Producción Total) = 450.00 barriles
Beneficio Máximo Z = 102500.00 euros

Análisis de resultados:

- ¿Cómo se calcula el beneficio total de Z?
- ¿Cuál es la graduación alcohólica promedio de la mezcla?
- Dado el modelo anterior, ¿qué modificaciones harías para que se utilice algún barril de whisky irlandés tipo 2?

Whiskies

Análisis de Resultados:

- ¿Cómo se calcula el beneficio total de Z?

$$Z=500B-300we-250wi_1-270wi_2$$
$$=500(450)-300(200)-250(250)-270(0)$$
$$=225000-60000-62500-0=$$
$$225000-122500$$
$$=102500 \text{ euros}$$

- ¿Cuál es la graduación alcohólica promedio de la mezcla?

$$\text{Graduación Promedio}=\frac{48we+38wi_1+42wi_2}{B}=\frac{19100}{450}\approx 42.44\%$$

Whiskies

Análisis de Resultados:

- Ajustar parámetro de coste o precio de venta. ¿Reducir coste de w_{i2} a 240?
- Ajustar parámetro de coste o precio de venta. ¿Aumentar coste de w_{i1} a 270?
- Imponer una demanda mínima para w_{i2}

$$w_{i2} \geq m$$

- Limitar la capacidad de producción de w_{i1}

$$w_{i1} \leq 200$$

- Limitar la capacidad de producción de w_{i1}

$$\frac{w_{i2}}{B} \geq p$$

Análisis de sensibilidad

- Es una técnica que permite evaluar cómo cambios en los parámetros de un modelo de programación lineal (como los coeficientes de la función objetivo o los recursos disponibles) afectan la solución óptima.
- Este análisis es fundamental para entender la robustez de la solución y para tomar decisiones informadas en situaciones de incertidumbre o cambios en el entorno. Tiene los siguientes objetivos.
- Evaluar la robustez de la solución óptima:
  - Determinar si la solución óptima permanece válida ante pequeños cambios en los parámetros.
  - Identificar los rangos de variación de los coeficientes dentro de los cuales la solución óptima no cambia.
- Identificar parámetros críticos:
  - Detectar qué parámetros tienen mayor impacto en la solución.
  - Priorizar la atención en los parámetros más sensibles para una gestión más efectiva.
- Facilitar la toma de decisiones bajo incertidumbre:
  - Preparar planes de contingencia para diferentes escenarios posibles.
  - Evaluar el efecto de cambios futuros en el mercado o en la disponibilidad de recursos.

Análisis de sensibilidad

Componentes

- Variación en los coeficientes de la función objetivo:
  - Analizar cómo cambios en los beneficios o costes unitarios afectan la solución.
  - Determinar los rangos de variación para los cuales la solución básica óptima permanece inalterada.
- Variación en los recursos disponibles:
  - Evaluar el impacto de cambios en las restricciones de disponibilidad de recursos.
  - Calcular el valor sombra de los recursos, que indica el incremento en el valor de la función objetivo por unidad adicional de recurso.
- Análisis de rangos:
  - Rango de optimalidad: Intervalo en el cual los coeficientes de la función objetivo pueden variar sin cambiar la solución óptima.
  - Rango de factibilidad: Intervalo en el cual los recursos pueden variar sin que la solución se vuelva infactible.
- Informes de sensibilidad del software de optimización:
  - Herramientas como Gurobi, CPLEX o Excel Solver proporcionan informes detallados que incluyen valores sombra, rangos de variación y análisis de sensibilidad.
  - Se puede realizar un análisis manual utilizando las propiedades de la dualidad y las tablas del método Simplex, es posible calcular manualmente los valores sombra y los rangos de validez.

Valores sombra

Definición

- Los valores sombra (o precios dual) representan la tasa de cambio del valor óptimo de la función objetivo por unidad adicional de un recurso disponible.
- Indican cuánto aumentaría (o disminuiría) el beneficio total si se incrementa (o reduce) en una unidad el recurso asociado a una restricción activa, manteniendo todo lo demás constante.

Interpretación económica

- Valor económico de los recursos escasos:
  - Los valores sombra reflejan el valor marginal de un recurso.
  - Si un recurso es escaso (la restricción está activa), su valor sombra es positivo, indicando que obtener más de ese recurso aumentaría el beneficio total.
- Decisiones de inversión:
  - Ayudan a decidir si es rentable invertir en obtener más de un recurso limitado
  - Ejemplos: contratar más personal o comprar más maquinaria.

Valores sombra

Cálculo de los valores sombra

- Mediante el problema dual:
  - Cada restricción en el problema primal se corresponde a una variable en el problema dual.
  - Los valores de estas variables duales en la solución óptima son los valores sombra.
- Utilizando el método Simplex:
  - En la última tabla óptima del Simplex, los coeficientes de las variables básicas en las ecuaciones correspondientes a las restricciones proporcionan los valores sombra.

Características de los valores sombra

- Aplicables a restricciones activas:
  - Los valores sombra son generalmente no cero solo para restricciones que están activas (se cumplen con igualdad en la solución óptima).
- Rangos de validez:
  - Los valores sombra son válidos dentro de ciertos rangos de variación del recurso.
  - Fuera de estos rangos, el valor sombra puede variar y la solución óptima puede ser diferente.

Restricciones activas

Definición

- Una **restricción activa** es aquella que se cumple con igualdad en la solución óptima. Es decir, el recurso asociado está completamente utilizado, y la restricción limita directamente la solución.
- Una **restricción no activa** (o con holgura) es aquella que no se utiliza completamente en la solución óptima; hay recursos sobrantes y la restricción no limita la solución.

Identificación de restricciones activas

- Si en la solución óptima, la restricción se cumple exactamente (el recurso total disponible es igual al recurso utilizado), es una restricción activa.

Restricciones activas

Importancia

- **Limitan la solución óptima:** Acotan la región factible donde se encuentra la solución óptima.
- Afectan directamente el valor de la función objetivo: Cualquier cambio en una restricción activa puede alterar la solución óptima y el valor de la función objetivo.
- Asociadas a valores sombra no cero: Las restricciones activas suelen tener valores sombra distintos de cero, indicando el valor económico del recurso limitado.

Restricciones no activas

- **Recursos con holgura:** Indican que hay más recurso disponible de lo que se utiliza en la solución óptima.
- Valor sombra cero: Añadir más de este recurso no mejorará el valor de la función objetivo, ya que no es un recurso limitante.
- Posibilidad de reasignación: Se puede considerar reducir el recurso disponible (si es posible) para ahorrar costes, ya que no afecta la solución óptima.

Aleaciones

- Una industria siderúrgica va a producir un nuevo tipo de hierro mezclando 4 tipos diferentes de hierro, cada uno de ellos con distinta composición.
- Cada tipo de hierro tiene un contenido distinto de Carbono, Silicio y Manganeso.
- La tabla siguiente indica estos contenidos y el precio unitario para los cuatro tipos de hierro:

Hierro	Carbono	Silicio	Manganeso	Precio
#1	2	1	0	0.05
#2	5	0	1	0.04
#3	0	60	40	0.25
#4	0	20	80	0.35

- Las exigencias del mercado imponen que el hierro que se va a producir debe cubrir los mínimos y máximos que son los siguientes:
  - Mínimos: 2 de Carbono y 2 de Silicio
  - Máximos: 4 de Carbono, 5 de Silicio y 1 de Manganeso
- La siderúrgica debe producir un lote de 1000 libras de este nuevo tipo de hierro.

Aleaciones

```python
# Importar el módulo gurobipy
from gurobipy import Model, GRB

# Crear un nuevo modelo
model = Model("Mezcla_Hierros")

# Definir las variables de decisión (cantidades en libras de cada tipo de hierro)
x1 = model.addVar(name="x1", lb=0)  # Hierro Tipo 1
x2 = model.addVar(name="x2", lb=0)  # Hierro Tipo 2
x3 = model.addVar(name="x3", lb=0)  # Hierro Tipo 3
x4 = model.addVar(name="x4", lb=0)  # Hierro Tipo 4

# Actualizar el modelo para integrar las nuevas variables
model.update()

# Establecer la función objetivo (minimizar el coste total)
model.setObjective(
    0.05 * x1 + 0.04 * x2 + 0.25 * x3 + 0.35 * x4,
    GRB.MINIMIZE
)

# Agregar las restricciones

# Restricción 1: Producción total de 1000 libras
model.addConstr(
    x1 + x2 + x3 + x4 == 1000,
    name="Produccion_Total"
)

# Restricciones de Carbono
# Restricción 2: Contenido mínimo de Carbono (2% de 1000 libras)
model.addConstr(
    0.02 * x1 + 0.05 * x2 >= 20,
    name="Carbono_Minimo"
)

# Restricción 3: Contenido máximo de Carbono (4% de 1000 libras)
model.addConstr(
    0.02 * x1 + 0.05 * x2 <= 40,
    name="Carbono_Maximo"
)

# Restricciones de Silicio
# Restricción 4: Contenido mínimo de Silicio (2% de 1000 libras)
model.addConstr(
    0.01 * x1 + 0.60 * x3 + 0.20 * x4 >= 20,
    name="Silicio_Minimo"
)

# Restricción 5: Contenido máximo de Silicio (5% de 1000 libras)
model.addConstr(
    0.01 * x1 + 0.60 * x3 + 0.20 * x4 <= 50,
    name="Silicio_Maximo"
)

# Restricción 6: Contenido máximo de Manganeso (1% de 1000 libras)
model.addConstr(
    0.01 * x2 + 0.40 * x3 + 0.80 * x4 <= 10,
    name="Manganeso_Maximo"
)

# Optimizar el modelo
model.optimize()
# Verificar si se encontró una solución óptima
if model.status == GRB.OPTIMAL:
    print("\nSolución Óptima:")
    print(f"x1 (Hierro Tipo 1) = {x1.X:.2f} libras")
    print(f"x2 (Hierro Tipo 2) = {x2.X:.2f} libras")
    print(f"x3 (Hierro Tipo 3) = {x3.X:.2f} libras")
    print(f"x4 (Hierro Tipo 4) = {x4.X:.2f} libras")
    print(f"coste Mínimo Total = ${model.ObjVal:.2f}")
else:
    print("No se encontró una solución óptima.")
```

Aleaciones

$$Minimizar\ Z = 0.05x + 0.04y + 0.25z + 0.35a$$
$$Sujeto\ a:$$
$$0.02x + 0.05y \geq 0.02$$
$$0.02x + 0.05y \leq 0.04$$
$$0.01x + 0.6z + 0.2a \geq 0.02$$
$$0.01x + 0.6z + 0.2a \leq 0.05$$
$$0.01y + 0.4z + 0.8a \leq 0.01$$
$$x + b + z + a = 1$$
$$x, y, z, a \geq 0$$

Solución Óptima:
x1 (Hierro Tipo 1) = 787.88 libras
x2 (Hierro Tipo 2) = 191.92 libras
x3 (Hierro Tipo 3) = 20.20 libras
x4 (Hierro Tipo 4) = 0.00 libras
coste Mínimo Total = $52.12

Análisis de resultados

¿Qué restricciones debo cambiar para que se utilice también el hierro de tipo 4?

Aleaciones

Solución óptima (GRB.OPTIMAL):
x = 787.88 libras
y = 191.92 libras
z = 20.20 libras
a = 0.00 libras
coste mínimo total = $52.12

Valores sombra de las restricciones (constr.Pi):
Produccion_Total: 0.0439
Carbono_Minimo: 0.0000
Carbono_Maximo: 0.0000
Silicio_Minimo: 0.6061
Silicio_Maximo: 0.0000
Manganeso_Maximo: -0.3939

Holguras de las restricciones constr.Slack:
Produccion_Total: 0.0000
Carbono_Minimo: -5.3535
Carbono_Maximo: 14.6465
Silicio_Minimo: 0.0000
Silicio_Maximo: 30.0000
Manganeso_Maximo: 0.0000

Costes reducidos de las variables (var.RC):
x: 0.0000
y: 0.0000
z: 0.0000
a: 0.5000

```python
# Imprimir los valores sombra de las restricciones
print("\nValores sombra de las restricciones:")
for constr in model.getConstrs():
    print(f"{constr.ConstrName}: {constr.Pi:.4f}")

# Imprimir los costes reducidos de las variables
print("\nCostes reducidos de las variables:")
for var in model.getVars():
    print(f"{var.VarName}: {var.RC:.4f}")

# Imprimir las holguras de las restricciones
print("\nHolguras de las restricciones:")
for constr in model.getConstrs():
    print(f"{constr.ConstrName}: {constr.Slack:.4f}")

# Verificar si se encontró una solución óptima
if model.status == GRB.OPTIMAL:
    print("\nSolución óptima:")
    for var in model.getVars():
        print(f"{var.VarName} = {var.X:.2f} libras")
    print(f"coste mínimo total = ${model.ObjVal:.2f}")
else:
    print("No se encontró una solución óptima.")
```

Aleaciones

Valores sombra de las restricciones

- **Produccion_Total: 0.0439**
  - **En un problema de minimización**, un valor sombra **positivo** indica que **aumentar** el lado derecho de la restricción en una unidad **incrementará** el coste mínimo total.
  - Por cada libra adicional que se requiera producir, el coste mínimo total aumentaría en aproximadamente **$0.0439**.
- **Carbono_Mínimo (Máximo), Silicio_Máximo : 0.0000**
  - El valor sombra es cero, lo que indica que no está limitando la solución óptima.
  - Aumentar o disminuir el requerimiento mínimo (máximo) de carbono y no afectará el coste mínimo total dentro de ciertos rangos. Tampoco el silicio máximo.

Aleaciones

Valores sombra de las restricciones

- **Silicio_Minimo: 0.6061**
  - Un valor sombra positivo indica que **aumentar** el requerimiento mínimo de silicio en una unidad **incrementará** el coste mínimo total en aproximadamente **$0.6061**.
  - Esta restricción es **activa** y está limitando la solución óptima.
- **Manganeso_Maximo: -0.3939**
  - Un valor sombra **negativo** en un problema de minimización indica que **aumentar** el límite máximo de manganeso en una unidad **reduciría** el coste mínimo total en aproximadamente **$0.3939**.
  - Esta restricción es crítica y está limitando la solución óptima.

Aleaciones

Costes reducidos de las variables

- Los **costes reducidos** muestran cuánto debe cambiar el coeficiente de una variable no básica (no incluida en la solución óptima) en la función objetivo para que esa variable entre en la solución óptima.
- **x1, x2, x3: Coste Reducido = 0.0000**
  - Estas variables están en la solución básica (tienen valores positivos en la solución óptima).
  - No hay necesidad de cambiar sus costes unitarios para que formen parte de la solución.
- **x4: Coste Reducido = 0.5000**
  - Para que el Hierro Tipo 4 entre en la solución óptima, su **coste unitario** tendría que **disminuir** en al menos **$0.50** por libra. ¿Tiene sentido?
  - Alternativamente, podrían modificarse las restricciones para hacerlo necesario.

Aleaciones

Holgura de las restricciones

- Produccion_Total: Holgura = 0.0000
- Silicio_Minimo: Holgura = 0.0000
- Manganeso_Maximo: Holgura = 0.0000
  - La restricción se cumple exactamente (es una **restricción activa)**.
- **Carbono_Maximo: Holgura = 14.6465**
  - Hay una **holgura positiva** en una restricción de tipo ≤, indicando que el contenido de carbono está **14.6465 unidades** por debajo del máximo permitido.
  - La restricción no es activa.
- **Silicio_Maximo: Holgura = 30.0000**
  - Hay una holgura significativa en esta restricción.
  - El contenido de silicio está **30 unidades** por debajo del máximo permitido.

Aleaciones

Holgura de las restricciones

- Carbono_Minimo: Holgura = -5.3535
  - ¿Tiene sentido?
- En Gurobi, la holgura de una restricción (atributo .Slack) se calcula como RHS - LHS (lado derecho menos lado izquierdo) para todo tipo de restricciones, **sin importar la dirección de la desigualdad**.
- **Cálculo de la Holgura:**
  - Valores de las vars. en la solución óptima: x1=787.88, x2=191.92

$$LHS = 0.02 * 787.88 + 0.05 + 191.92 \approx 15.76 + 9.6 \approx 25.35$$
$$RHS = 20$$
$$Holgura = RHS - LHS = 20 - 25.35 = -5.35$$

Aleaciones

Reducir el coste unitario del Hierro Tipo 4

- Disminuir el coste unitario de **$0.35** a **$0.35 - $0.50 = -$0.15** por libra.
- Un coste unitario negativo no es realista.
- No es viable reducir el coste unitario lo suficiente para incluir x4 en la solución óptima bajo las condiciones actuales.

Modificar las restricciones activas

- Aumentar el límite máximo de Manganeso
  - ¿En cuánto tiene que aumentar este límite máximo para que haya al menos una unidad de Manganeso?
- Aumentar el requerimiento mínimo de Silicio
  - Lo mismo pero para esta restricción.

Aleaciones

Aumentar el límite máximo de Manganeso

- Cálculo especialmente complicado.
- Se utiliza el precio sombra y se obtiene una cota inferior de 11.27.
- Es una cota inferior y es insuficiente. El valor está cerca de 16.

Aumentar el requerimiento mínimo de Silicio

- Primera prueba. Si x_4 aporta 0.20, ¿con 20.20 se utiliza x_4? No.
  - Aumenta x_3 con más Silicio y más costes.
- Con $x1$ en su máximo (1000 - $x2$ - $x3$), podemos intentar minimizar $x2$ y maximizar $x3$ hasta alcanzar el límite de manganeso. Complicado.
- ¿Alternativa? Aumentar *mucho* el requerimiento mínimo.

Aleaciones

Aumentar *mucho* el requerimiento mínimo de Silicio

- Máximo Silicio aportado por x_3 :

$$Máximo\ x3 = \frac{Límite\ de\ Manganeso\ -\ 0.01}{0.4} = 0.4167$$

- Coste de x_4 : \$0.35 por libra
- Silicio aportado por x_4 : 0.20 libras por libra
- Coste por libra de silicio de $x4$

$$\frac{0.35}{0.2} = 1.75$$

Es más rentable el silicio a través de x_3 que de x_4, por lo que se prefiere

Es complicado hacerlo cambiando una única restricción. Además, el coste de x_3 respecto a x_4 no va a permitir cambios sencillos.

PCIngredients

- La empresa PCIngredients vende ordenadores y debe hacer una planificación semanal de la producción.
- La compañía produce tres tipos de ordenadores: de mesa (A), portátil normal (B) y portátil de lujo (C). El beneficio neto por la venta un ordenador es 350, 470 y 610 euros, respectivamente.
- Cada semana se venden todos los equipos que se montan.
- Los ordenadores pasan un control de calidad de una hora y la empresa dispone de 120 horas para realizar los controles de los ordenadores A y B y 48 para los C.
- El resto de las operaciones de montaje requieren 10, 15 y 20 horas, respectivamente, y la empresa dispone de 2000 horas a la semana.

PCIngredients

$$Maximizar\ Z = 350x + 470y + 610z$$
$$Sujeto\ a:$$
$$x + y \leq 120$$
$$z \leq 48$$
$$10x + 15y + 20z \leq 2000$$
$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$

Solución Óptima:
x (Ordenador A) = 120.00 unidades
y (Ordenador B) = 0.00 unidades
z (Ordenador C) = 40.00 unidades
Beneficio Máximo Total = 66400.00 euros

```python
from gurobipy import Model, GRB

# Crear un nuevo modelo
model = Model("PCIngredients_Produccion")

# Definir las variables de decisión
xA = model.addVar(name="xA", lb=0)  # Unidades de Ordenador A
xB = model.addVar(name="xB", lb=0)  # Unidades de Ordenador B
xC = model.addVar(name="xC", lb=0)  # Unidades de Ordenador C

# Establecer la función objetivo
model.setObjective(
    350 * xA + 470 * xB + 610 * xC,
    GRB.MAXIMIZE
)

# Agregar las restricciones

# Restricción 1: Control de Calidad para A y B
model.addConstr(
    xA + xB <= 120,
    name="Control_Calidad_AB"
)

# Restricción 2: Control de Calidad para C
model.addConstr(
    xC <= 48,
    name="Control_Calidad_C"
)

# Restricción 3: Tiempo de Montaje Total
model.addConstr(
    10 * xA + 15 * xB + 20 * xC <= 2000,
    name="Tiempo_Montaje"
)

# Optimizar el modelo
model.optimize()

# Verificar si se encontró una solución óptima
if model.status == GRB.OPTIMAL:
    print("\nSolución Óptima:")
    print(f"xA (Ordenador A) = {xA.X:.2f} unidades")
    print(f"xB (Ordenador B) = {xB.X:.2f} unidades")
    print(f"xC (Ordenador C) = {xC.X:.2f} unidades")
    print(f"Beneficio Máximo Total = {model.ObjVal:.2f} euros")
else:
    print("No se encontró una solución óptima.")
```

PCIngredients

Análisis de sensibilidad

- Identificación de las restricciones críticas
- Análisis de variación en los coeficientes de la función objetivo
  - Cambios en el beneficio por unidad del ordenador Y
  - Cambios en el beneficio por unidad del ordenador X
- Análisis de variación en los recursos disponibles
  - Tiempo de montaje total
  - Tiempo de control de calidad para ordenadores X e Y
  - Tiempo de control de calidad para ordenadores Z
- Análisis de cambios en el tiempo de montaje
  - Aumento del tiempo de montaje total
  - Disminución del tiempo de montaje total
- Análisis de cambios simultáneos en recursos y beneficios
  - Incrementar el beneficio por unidad Y y el tiempo de montaje

PCIngredients

Análisis de las restricciones

- Restricciones activas (holgura = 0):
  - Tiempo de control de calidad para A y B: Se utiliza todo el tiempo disponible.
  - Tiempo de montaje total : Se utiliza toda la capacidad disponible.
- Restricciones no activas (holgura > 0):
  - Tiempo de control de calidad para C: Hay una holgura de 8 horas.

PCIngredients

Análisis de los costes reducidos

- x_A y x_C: Coste reducido de 0.0000
  - Estas variables están en la solución óptima .
- x_B: Coste reducido de -32.5000.
  - El coste reducido negativo indica que, para que la producción del ordenador B sea rentable y entre en la solución óptima, el beneficio por unidad podría **disminuir hasta en $32.50** sin afectar la solución actual.
  - **Como el beneficio por unidad de B es de $470**, una disminución hasta $437.50 aún mantendría la solución óptima sin producir B.

PCIngredients

Valores sombra

- Control_Calidad_AB (45.0000):
  - Por cada hora adicional de tiempo de control de calidad para los ordenadores A y B, el beneficio total aumentaría en $45.00.
  - Es un recurso crítico; aumentar su disponibilidad podría incrementar el beneficio.
- Control_Calidad_C (0.0000):
  - No hay incremento en el beneficio total al aumentar el tiempo de control de calidad para el ordenador C, ya que existe una holgura de 8 horas.
  - No es necesario aumentar este recurso; actualmente hay tiempo de control de calidad sin utilizar para C.
- Tiempo_Montaje (30.5000):
  - Por cada hora adicional, el beneficio total aumentaría en **$30.50**.
  - Es un recurso crítico; aumentar su disponibilidad podría incrementar el beneficio.

PCIngredients

Conclusiones

- Recursos críticos
  - Tiempo de control de calidad para A y B (valor sombra alto $45.00)
  - Tiempo de montaje (valor sombra significativo $30.50)
- Coste reducido negativo (x_B=−32.5):
  - El ordenador B no es competitivo en comparación con A y C en las condiciones actuales. ¿Aumentar beneficio?¿Reducir costes?
- Utilización de holguras
  - Control de calidad para C (holgura de 8 horas).
  - ¿Reasignar tiempos?
- ¿Cómo podría producir PC de tipo B?
- ¿Cómo podría producir PC de tipo B, además de A y C?

PCIngredients

Razones por las que xB=0 en la solución óptima

- Menor rentabilidad por hora de montaje:
  - Tipo A: \$350 / 10 horas = \$35/hora
  - Tipo B: \$470 / 15 horas ≈ \$31.33/hora
  - Tipo C: \$610 / 20 horas = \$30.50/hora
  - El PC Tipo A tiene la mayor rentabilidad por hora de montaje, por lo que se prioriza su producción.
- Los recursos limitados (tiempo de montaje y control de calidad) se asignan a los productos más rentables.
- Coste reducido de xB :–\$32.50. Indica que aumentar xB disminuiría el beneficio total, por lo que no se incluye en la solución óptima.

PCIngredients

¿Cómo podría producir PC de tipo B?

- Ajuste de los beneficios por unidad:
  - PC Tipo B: Aumentar de $470 a $503 (rentabilidad $33.53/hora)

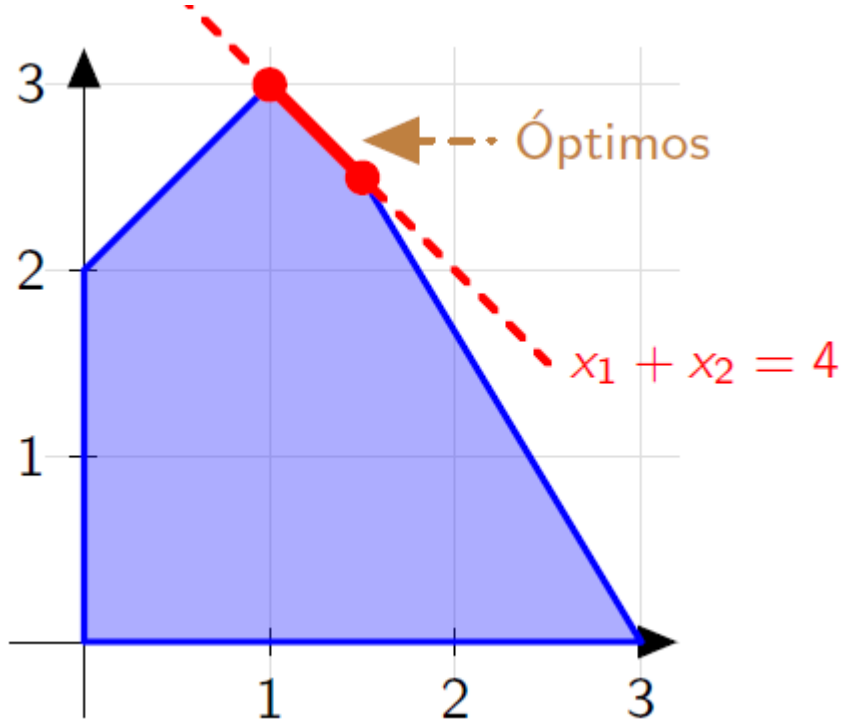
¿Cómo podría producir PC de tipo B, además de A y C?

- Ajuste de los beneficios por unidad:
  - PC Tipo A: Aumentar de $350 a $400
  - PC Tipo B: Aumentar de $470 a $475
  - PC Tipo C: Reducir de $610 a $550
- Ajuste de restricciones:
  - Control de calidad A y B: Reducir a 100

Algoritmos de resolución

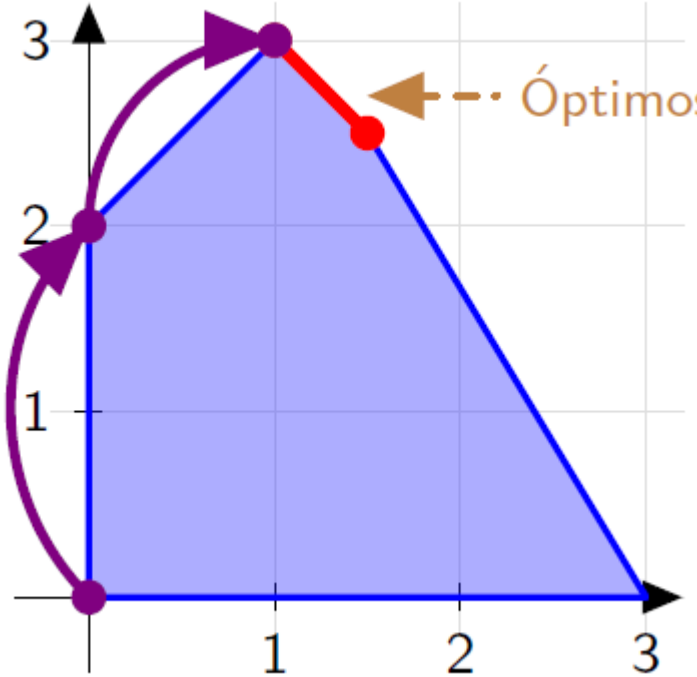
- Algoritmo del Simplex (Dantzig, 1949).
- Métodos de punto interior (Karmarkar, 1986).

$$
\begin{aligned}
\text{máx} \quad & x_1 + x_2 , \\
\text{s.a} \quad & -x_1 + x_2 \leq 2, \\
& x_1 + x_2 \leq 4, \\
& 5x_1 + 3x_2 \leq 15, \\
& x_1 , \quad x_2 \geq 0.
\end{aligned}
$$



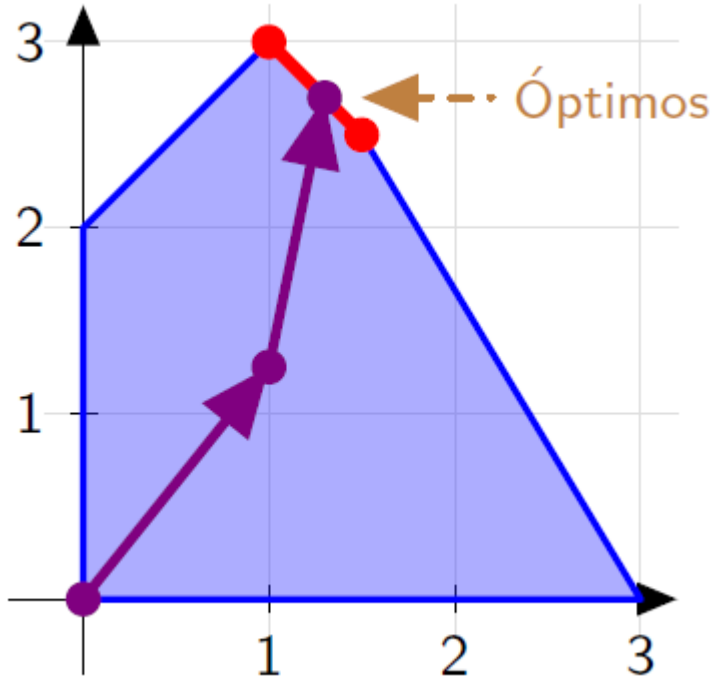
Algoritmos de resolución

- El algoritmo del Simplex se mueve por la frontera de la región factible, hasta llegar a un punto óptimo.
- Se basa en el siguiente resultado:
  - *Si el problema de programación lineal tiene solución, entonces se alcanza al menos en un punto extremo.*



Algoritmos de resolución

- Los métodos de punto interior se mueven por el interior de la región factible.
- Utilizan técnicas de programación no lineal.
- Se basan en el uso de funciones barrera que penalizan la violación de alguna restricción.



Conclusiones

- La programación lineal se ha convertido en una de las herramientas más importantes en la optimización matemática debido a dos razones fundamentales:
  - Es posible resolver problemas de grandes dimensiones.
  - Prácticamente todos los modelos se pueden resolver, independientemente de su formulación.

- El éxito de la programación lineal tiene una pega: los modelos utilizan unas hipótesis de partida que muchas veces suponen una simplificación excesiva del problema.
  - Suponen que los costes son lineales: ¿producir el doble -> cuesta el doble?
  - Las economías de escala suelen indicar que a mayor producción, menor coste unitario

Conclusiones

- Las hipótesis anteriores son muy restrictivas. El avance tecnológico (potencia y velocidad de computación) ha permitido el desarrollo de alternativas dentro de la Investigación Operativa que permiten abordar nuevos problemas:
  - Programación entera: cantidades no divisibles y enteras y decisiones lógicas.
  - Programación no lineal: relaciones no proporcionales y no aditivas.
  - Programación multiobjetivo: varios objetivos.
  - Programación por metas: permite trabajar con modelos en los que las restricciones no sean tan rígidas.
  - Programación estocástica, permite incorporar la incertidumbre inherente en muchas situaciones reales al modelo.
  - Programación semi-infinita, permite trabajar con problemas en los que algunos de los parámetros del problema no están controlados por el modelizador.
  - Programación difusa, permite trabajar con problemas en los que las restricciones no son rígidas (modelizan relaciones vagamente definidas).

Bibliografía

- Hillier & Liebermann (2010). Introducción a la investigación de operaciones, 9ªed. Capítulos 1 y 2.
- González y García (2015). Manual práctico de investigación de operaciones 4ª ed. Capítulo 1.
- Investigación Operativa (2004). Modelos determinísticos y estocásticos. Sixto Ríos Insua, Alfonso Mateosos Caballero, Mª Concepción Bielza Lozoya y Antonio Jiménez Martín, Editorial Centro de Estudios Ramón Areces, S.A., Madrid.

Bibliografía

- Investigación Operativa. Optimización, Sixto Ríos Insua. Editorial Centro de Estudios Ramón Areces, S.A., Madrid.
- Problemas de Investigación Operativa. Programación Lineal y Extensiones. Sixto Ríos Insua, David Ríos Insua, Alfonso Mateos Caballero, Jacinto Martín Jiménez y Antonio Jiménez Martín. Editorial Ra-Ma, Madrid 2006.
- Model Building in Mathematical Programming. Paul W. Williams. Editorial JOHN WILEY AND SONS
- Apuntes de Victoria Ruiz y Antonio Alonso
- Apuntes de Alberto Herrán y José J. Ruz Ortiz
- Apuntes de Gerardo Reyes y Salvador Sánchez

Grado en Ingeniería del Software

Investigación Operativa

BLOQUE II. MODELOS DETERMINISTAS

Tema 2: Optimización Lineal Continua

URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Grado en Ingeniería del Software

Investigación Operativa

BLOQUE II. MODELOS DETERMINISTAS

Tema 3: Optimización Lineal Entera y Combinatoria

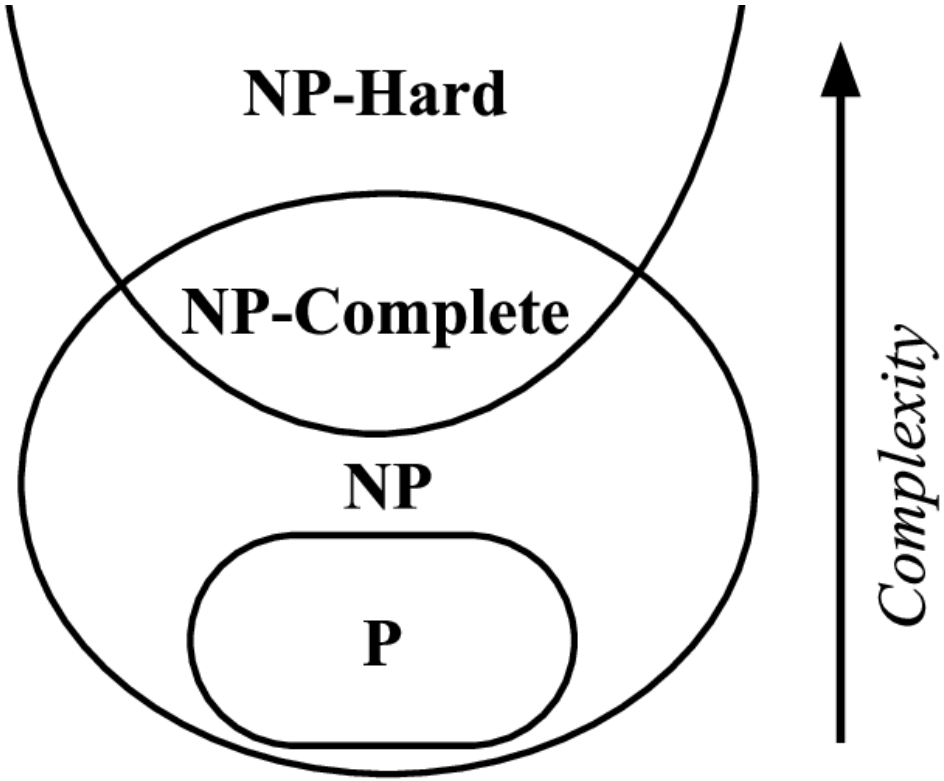
URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Índice

Introducción

- En la teoría de la complejidad computacional, las clases P, NP, NP-duro y NP-completo son categorías que nos permiten clasificar problemas según la dificultad de resolverlos y verificar sus soluciones.
- Estas clasificaciones son fundamentales para entender qué problemas pueden ser resueltos de manera eficiente y cuáles representan desafíos significativos en computación.
- ¿P = NP?
  - https://www.youtube.com/watch?v=UR2oDYZ-Sao&ab_channel=Derivando



Introducción

Clase P (tiempo polinomial)

- **P** es el conjunto de problemas de decisión que pueden ser resueltos por una máquina determinista en **tiempo polinomial** respecto al tamaño de la entrada. Esto significa que existe un algoritmo que resuelve el problema en un tiempo que es una función polinomial de la longitud de la entrada.
- Los problemas en **P** se consideran **eficientemente resolubles**, ya que los tiempos polinomiales son prácticos para tamaños de entrada razonables. Ejemplos:
  - **Ordenamiento de números** (algoritmos como Quicksort).
  - **Búsqueda de caminos más cortos** en grafos (algoritmo de Dijkstra).
  - **Multiplicación de matrices**.

Introducción

Clase NP (tiempo polinomial no determinista)

- **NP** es el conjunto de problemas de decisión para los cuales, si se presenta una solución candidata, esta puede ser **verificada** en **tiempo polinomial** por una máquina determinista.
- No se requiere que el problema pueda ser resuelto en tiempo polinomial, solo que las soluciones puedan ser verificadas rápidamente.
- Los problemas en **NP** pueden ser difíciles de resolver, pero una vez que se tiene una solución, es **rápido** verificar su corrección.

Introducción

Clase NP-completo

- Un problema es NP-completo si:
  - Está en NP.
  - Es tan difícil como cualquier otro problema en NP, en el sentido de que cualquier problema en NP puede ser transformado en él mediante una reducción en tiempo polinomial.
- Los problemas NP-completos son los más difíciles dentro de NP
- Ejemplos:
  - **Problema SAT**: El primer problema demostrado como NP-completo (Teorema de Cook-Levin).
  - **Problema del Clique**: Determinar si existe un clique (subgrafo completamente conectado) de tamaño k en un grafo.
  - **Problema de coloreado de grafos**: Determinar si un grafo puede ser coloreado con k colores sin que dos nodos adyacentes compartan color.

Introducción

Clase NP-duro

- Un problema es NP-duro si es al menos tan difícil como los problemas en NP.
- Formalmente, si todo problema en NP puede ser reducido a él en tiempo polinomial.
- Un problema NP-duro no tiene que estar en NP (sus soluciones no necesitan ser verificables en tiempo polinomial). Ejemplos:
  - **Problema del viajero de comercio (TSP)**:Encontrar la ruta más corta posible que visita cada ciudad exactamente una vez. No es un problema de decisión y no está en NP, pero es NP-duro.
  - **Problema de parada (Halting Problem)**:Determinar si un programa arbitrario se detendrá o continuará ejecutándose indefinidamente. Es indecidible y se considera NP-duro.

Introducción

- * Un problema NP que también es P es resoluble en tiempo P.
- ** Un problema NP-duro que también es NP-completo es verificable en tiempo P.
- *** Los problemas NP-Completos (todos los cuales forman un subconjunto de NP-duros) pueden serlo. El resto de NP-duros no lo son.

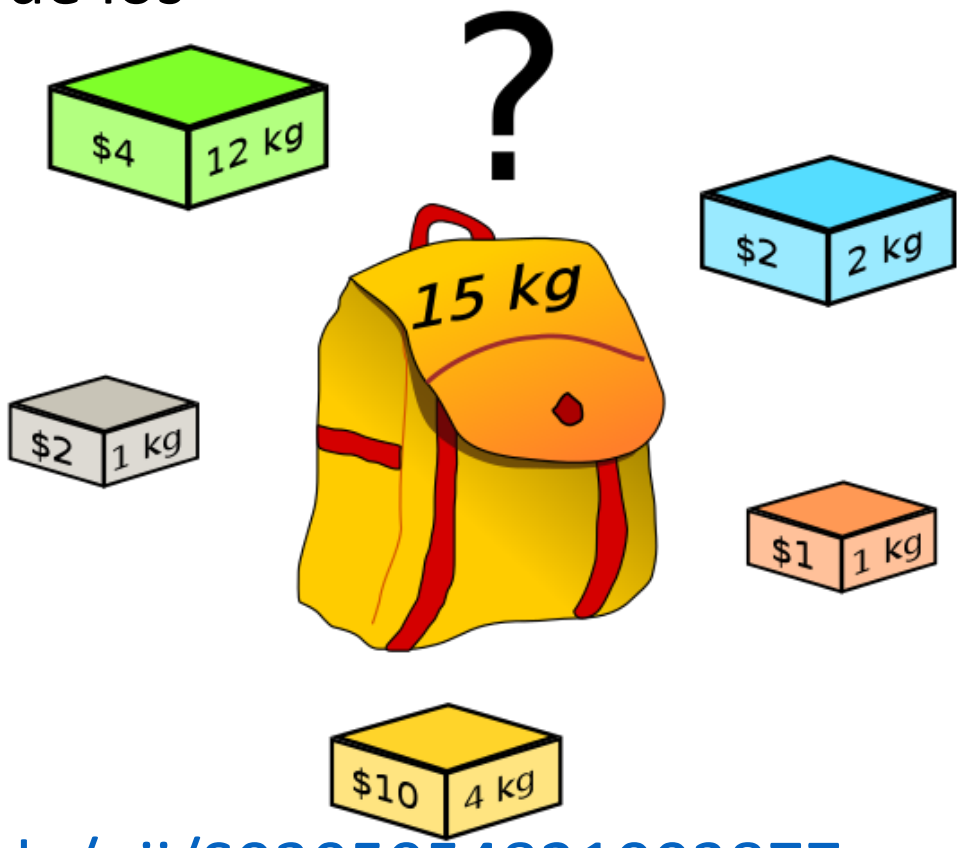
Problem Type	Verifiable in P time	Solvable in P time	Increasing Difficulty
P	Yes	Yes	
NP	Yes	Yes or No *	
NP-Complete	Yes	Unknown	
NP-Hard	Yes or No **	Unknown ***	
			V

Introducción

- Los modelos de programación entera son una extensión de los modelos lineales en los que algunas variables toman valores enteros.
- Con frecuencia las variables enteras sólo toman valores en 0 y 1, ya que este tipo de variables permiten representar condiciones lógicas.
- Este tipo de modelos permite representar sistemas mucho más complejos.
- A cambio, la resolución de estos se complica excesivamente.
- Problemas con unas solas decenas de variables pueden ser casi imposibles de resolver.

Problema de la mochila

- El problema de la mochila (knapsack) es uno de los problemas más clásicos y estudiados en el campo de la optimización combinatoria y la teoría de la complejidad computacional.



- Se trata de seleccionar un subconjunto de objetos con diferentes pesos y valores para incluir en una mochila, de manera que el valor total sea maximizado sin exceder la capacidad máxima de peso de la mochila.
- https://www.sciencedirect.com/science/article/pii/S030505482100387 7

Fuente: https://inviarobotics.com/blog/knapsack-problem-and-warehouse-optimization/

Problema de la mochila

Formulación del problema

- Supongamos que se tiene **N** objetos, donde cada objeto **i** tiene:
  - Un **valor** v_i.
  - Un **peso** w_i.
- Una mochila con una capacidad máxima de peso **W**.
- El objetivo es seleccionar una combinación de objetos para maximizar el valor total sin exceder el peso máximo **W**.
- Cada objeto puede ser incluido o no en la mochila (decisión binaria).
- No se permiten fracciones de objetos.

Problema de la mochila

Formulación del problema

- Supongamos que se tiene **N** objetos, donde cada objeto **i** tiene:
  - Un **valor** v_i.
  - Un **peso** w_i.

Objeto	Valor (v_i)	Peso (w_i)
1	60	10
2	100	20
3	120	30

- Maximizar el valor total sin exceder 50 kg.

Problema de la mochila

Función objetivo:

$$Maximizar\ Z = \sum_{i=1}^{N} v_i x_i$$

Sujeto a:

$$\sum_{i=1}^{N} w_i x_i \leq W$$

$$x_i \in \{0, 1\}, \forall_i = 1, 2, \dots, N$$

Variables de decisión:

$$x_i \in \{0, 1\}, donde:$$

$x_i = 1$ *si el objeto i es incluido en la mochila*

$x_i = 0$ *si el objeto i no es incluido*

Problema de la mochila

- Datos del problema

```
# Importar la librería gurobipy
from gurobipy import Model, GRB

# Datos del problema
objetos = [1, 2, 3]
valores = {1: 60, 2: 100, 3: 120}
pesos = {1: 10, 2: 20, 3: 30}
capacidad = 50
```

- Creación del modelo

```
# Crear un nuevo modelo
modelo = Model("Mochila_0-1")
```

- Variables de decisión **binarias**

```
# Definir las variables de decisión
# x[i] = 1 si el objeto i es incluido en la mochila, 0 en caso contrario
x = {}
for i in objetos:
    x[i] = modelo.addVar(vtype=GRB.BINARY, name=f"x_{i}")
```

Problema de la mochila

- Maximizar la función objetivo

```
# Establecer la función objetivo: maximizar el valor total
modelo.setObjective(
    sum(valores[i] * x[i] for i in objetos),
    GRB.MAXIMIZE
)
```

- Restricción capacidad **W**

```
# Agregar la restricción de capacidad
modelo.addConstr(
    sum(pesos[i] * x[i] for i in objetos) <= capacidad,
    name="Restriccion_Capacidad"
)
```

- Imprimir solución

```
# Optimizar el modelo
modelo.optimize()
# Imprimir la solución
if modelo.status == GRB.OPTIMAL:
    print(f"\nValor óptimo total: ${modelo.ObjVal}")
    print("Objetos seleccionados:")
    for i in objetos:
        if x[i].X > 0.5:
            print(f" - Objeto {i}: Valor = ${valores[i]}, Peso = {pesos[i]} kg")
else:
    print("No se encontró una solución óptima.")
```

Valor óptimo total: $220.0
Objetos seleccionados:
 - Objeto 2: Valor = $100, Peso = 20 kg
 - Objeto 3: Valor = $120, Peso = 30 kg

Selección de inversiones

- Se están considerando cuatro posibles inversiones.
- La primera de ellas se prevé que proporcione unos beneficios netos de 16.000 euros, la segunda, 22.000 euros, la tercera 12.000 euros, y la cuarta 8.000 euros.
- Cada una de las inversiones requiere una cantidad de dinero en efectivo: 5.000, 7.000, 4.000 y 3.000 euros, respectivamente.
- Si solo se dispone de 14.000 euros para invertir. ¿Qué modelo de programación lineal entera permite obtener la combinación de inversiones que prevea los máximos beneficios?

Selección de inversiones

$$Función\ objetivo:$$

$$Maximizar\ Z = 16000x_1 + 22000x_2 + 12000x_3 + 8000x_4$$

$$Sujeto\ a:$$

$$5000x_1 + 7000x_2 + 4000x_3 + 3000x_4 \leq 14000$$

$$x_i \in \{0, 1\}, \forall_i = 1, 2, 3,4$$

Variables de decisión:

$$x_i \in \{0, 1\}, donde:$$

$$x_i = 1\ si\ se\ realiza\ la\ inversión$$

$$x_i = 0\ si\ no\ se\ realiza\ la\ inversión$$

Selección de inversiones

```python
# Importar la librería gurobipy
from gurobipy import Model, GRB

# Datos del problema
inversiones = [1, 2, 3, 4]
beneficios = {
    1: 16000,
    2: 22000,
    3: 12000,
    4: 8000
}
capital_requerido = {
    1: 5000,
    2: 7000,
    3: 4000,
    4: 3000
}
capital_disponible = 14000

# Crear un nuevo modelo
modelo = Model("Seleccion_Inversiones")

# Desactivar la salida de Gurobi (opcional)
modelo.Params.OutputFlag = 0

# Definir las variables de decisión
# x[i] = 1 si se realiza la inversión i, 0 en caso contrario
x = {}
for i in inversiones:
    x[i] = modelo.addVar(vtype=GRB.BINARY, name=f"x_{i}")

# Establecer la función objetivo: maximizar el beneficio total
modelo.setObjective(
    sum(beneficios[i] * x[i] for i in inversiones),
    GRB.MAXIMIZE
)

# Agregar la restricción de capital disponible
modelo.addConstr(
    sum(capital_requerido[i] * x[i] for i in inversiones) <= capital_disponible,
    name="Restriccion_Capital"
)

# Optimizar el modelo
modelo.optimize()

# Imprimir la solución
if modelo.status == GRB.OPTIMAL:
    total_beneficio = modelo.ObjVal
    total_capital = sum(capital_requerido[i] * x[i].X for i in inversiones)
    print(f"Beneficio máximo total: {total_beneficio} euros")
    print(f"Capital invertido total: {total_capital} euros")
    print("Inversiones seleccionadas:")
    for i in inversiones:
        if x[i].X > 0.5:
            print(f" - Inversión {i}: Beneficio = {beneficios[i]} euros, Capital = {capital_requerido[i]} euros")
else:
    print("No se encontró una solución óptima.")
```

Selección de inversiones

- Beneficio máximo total: 42000.0 euros
- Capital invertido total: 14000.0 euros
- Inversiones seleccionadas:
- - Inversión 2: Beneficio = 22000 euros, Capital = 7000 euros
- - Inversión 3: Beneficio = 12000 euros, Capital = 4000 euros
- - Inversión 4: Beneficio = 8000 euros, Capital = 3000 euros

Mercancías

- En un camión se desean cargar mercancías de 5 tipos diferentes en cuanto a su peso, valor y volumen, según se especifica en la siguiente tabla:

Tipo	Peso	Volumen	Valor
1	5	1	4
2	8	8	7
3	3	6	6
4	2	5	5
5	7	4	4

- El camión sólo se admite un peso máximo de 112 y un volumen máximo de 109.
- ¿Cuántos productos de cada tipo debe llevar para maximizar el valor de la carga?

Mercancías

Función objetivo:

$$Maximizar\ Z = \sum_{i=1}^{N} c_i x_i$$

Sujeto a:

$$\sum_{i=1}^{N} w_i x_i \leq W$$

$$\sum_{i=1}^{N} v_i x_i \leq V$$

$$x_i \in \mathbb{Z}, \forall_i = 1, 2, \dots, N$$

Mercancías

```python
# Importar la librería gurobipy
from gurobipy import Model, GRB

# Datos del problema
mercancias = [1, 2, 3, 4, 5]
pesos = {1: 5, 2: 8, 3: 3, 4: 2, 5: 7}
volumenes = {1: 1, 2: 8, 3: 6, 4: 5, 5: 4}
valores = {1: 4, 2: 7, 3: 6, 4: 5, 5: 4}
peso_maximo = 112
volumen_maximo = 109

# Crear un nuevo modelo
modelo = Model("Carga_Camion")

# Desactivar la salida de Gurobi (opcional)
modelo.Params.OutputFlag = 0

# Definir las variables de decisión
# x[i] = número de unidades de la mercancía i a cargar
x = {}
for i in mercancias:
    x[i] = modelo.addVar(vtype=GRB.INTEGER, lb=0, name=f"x_{i}")

# Establecer la función objetivo: maximizar el valor total
modelo.setObjective(
    sum(valores[i] * x[i] for i in mercancias),
    GRB.MAXIMIZE
)
```

```python
# Agregar la restricción de peso máximo
modelo.addConstr(
    sum(pesos[i] * x[i] for i in mercancias) <= peso_maximo,
    name="Restriccion_Peso"
)

# Agregar la restricción de volumen máximo
modelo.addConstr(
    sum(volumenes[i] * x[i] for i in mercancias) <= volumen_maximo,
    name="Restriccion_Volumen"
)

# Optimizar el modelo
modelo.optimize()

# Imprimir la solución
if modelo.status == GRB.OPTIMAL:
    total_valor = modelo.ObjVal
    total_peso = sum(pesos[i] * x[i].X for i in mercancias)
    total_volumen = sum(volumenes[i] * x[i].X for i in mercancias)
    print(f"Valor máximo total de la carga: {total_valor} euros")
    print(f"Peso total de la carga: {total_peso} kg")
    print(f"Volumen total de la carga: {total_volumen} m³")
    print("Unidades de mercancías a cargar:")
    for i in mercancias:
        cantidad = int(round(x[i].X))
        if cantidad > 0:
            print(f" - Mercancía {i}: {cantidad} unidades")
else:
    print("No se encontró una solución óptima.")
```

Mercancías

- Valor máximo total de la carga: 151.0 euros
- Peso total de la carga: 108.0 kg
- Volumen total de la carga: 109.0 m^3
- Unidades de mercancías a cargar:
- - Mercancía 1: 14 unidades
- - Mercancía 4: 19 unidades

Problema de asignación

- El problema de asignación es un problema fundamental en el campo de la Investigación Operativa y las matemáticas aplicadas.
- Se trata de asignar un conjunto de recursos a un conjunto de tareas de manera óptima, generalmente minimizando costes o maximizando beneficios.
- Este problema tiene aplicaciones en diversas áreas como logística, programación de personal, asignación de maquinaria, planificación de horarios y muchas otras. https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109

		machines			
		I	II	III	IV
jobs	A	10	12	19	11
	B	5	10	7	8
	C	12	14	13	11
	D	8	15	11	9

Problema de asignación

- Juan es el jefe de un bufete de jóvenes abogados y abogadas, y está interesado en la utilización más efectiva de sus recursos de personal buscando la forma de hacer las mejores asignaciones de abogado/a-cliente.
- El 25 de octubre llegan nuevos clientes.
- Revisando a su personal encuentra que 4 (**N**) abogados: Ana, Bruno, Carmen y Domingo.
- Todos pueden ser asignados a los casos.
- Cada uno de ellos sólo se puede hacer cargo de un caso.

Problema de asignación

- Para decidir la mejor asignación Juan tiene en cuenta una tasa de efectividad (de 1 a 9) construida sobre actuaciones (**M**) anteriores de dichos abogados, ya que no todos son especialistas en todo tipo de procesos:

Abogado/a	Divorcio	Empresariales	Desfalco	Herencias
Ana	6	2	8	5
Bruno	9	3	5	8
Carmen	4	8	3	4
Domingo	6	7	6	4

- ¿Qué asignaciones se deben realizar?

Problema de asignación

$Función\ objetivo$:

$$Maximizar\ Z = \sum_{i=1}^{N} \sum_{j=1}^{M} e_i x_i$$

$Sujeto\ a$:

$$\sum_{j=1}^{M} x_{ij} = 1\ \forall_i\ \in \{Ana, Bruno, Carmen, Domingo\}$$

$$\sum_{i=1}^{N} x_{ij} = 1\ \forall_i\ \in \{1, 2, 3, 4\}$$

$$x_{ij} \in \{0, 1\}, \forall_{ij}$$

Problema de asignación

```python
# Importar la librería gurobipy
from gurobipy import Model, GRB

# Definir los datos
abogados = ['Ana', 'Bruno', 'Carmen', 'Domingo']
casos = {
    1: 'Divorcio',
    2: 'Fusión Empresarial',
    3: 'Desfalco',
    4: 'Herencias'
}

# Tasa de efectividad e_{ij}
efectividad = {
    ('Ana', 1): 6,
    ('Ana', 2): 2,
    ('Ana', 3): 8,
    ('Ana', 4): 5,
    ('Bruno', 1): 9,
    ('Bruno', 2): 3,
    ('Bruno', 3): 5,
    ('Bruno', 4): 8,
    ('Carmen', 1): 4,
    ('Carmen', 2): 8,
    ('Carmen', 3): 3,
    ('Carmen', 4): 4,
    ('Domingo', 1): 6,
    ('Domingo', 2): 7,
    ('Domingo', 3): 6,
    ('Domingo', 4): 4
}
# Crear un nuevo modelo
modelo = Model("Asignacion_Abogados_Casos")

# Desactivar la salida de Gurobi (opcional)
modelo.Params.OutputFlag = 0
```

```python
# Definir las variables de decisión
x = {}
for i in abogados:
    for j in casos:
        x[i, j] = modelo.addVar(vtype=GRB.BINARY, name=f"x_{i}_{j}")

# Actualizar el modelo para integrar las variables
modelo.update()

# Establecer la función objetivo: maximizar la efectividad total
modelo.setObjective(
    sum(efectividad[i, j] * x[i, j] for i in abogados for j in casos),
    GRB.MAXIMIZE
)

# Agregar las restricciones

# 1. Cada abogado debe ser asignado a exactamente un caso
for i in abogados:
    modelo.addConstr(
        sum(x[i, j] for j in casos) == 1,
        name=f"Asignacion_Unica_Abogado_{i}"
    )

# 2. Cada caso debe ser asignado a exactamente un abogado
for j in casos:
    modelo.addConstr(
        sum(x[i, j] for i in abogados) == 1,
        name=f"Asignacion_Unica_Caso_{j}"
    )

# Optimizar el modelo
modelo.optimize()
```

Problema de asignación

- Efectividad total máxima: 30.0
- Asignaciones óptimas:
- - Ana asignado al caso 'Desfalco' con efectividad 8
- - Bruno asignado al caso 'Herencias' con efectividad 8
- - Carmen asignado al caso 'Fusión Empresarial' con efectividad 8
- - Domingo asignado al caso 'Divorcio' con efectividad 6

Problema de cubrimiento

- El problema de cubrimiento (set covering) consiste en seleccionar el número mínimo de subconjuntos de un conjunto universal para cubrir todos los elementos del conjunto universal.
- En otras palabras, todos los elementos de un conjunto deben estar contenidos en al menos uno de los subconjuntos seleccionados.
- Aplicaciones:
  - Planificación de rutas: Determinar el mínimo número de vehículos necesarios para cubrir todas las rutas.
  - Ubicación de instalaciones: Colocar el mínimo número de instalaciones para cubrir todas las áreas de servicio.
  - Asignación de recursos: Asignar recursos limitados para cubrir todas las necesidades.

https://www.sciencedirect.com/science/article/pii/S0167637724000944

Problema de cubrimiento

- Dados los siguientes subconjuntos:

Subconjunto (S_j)	Zonas cubiertas (e_i)	Coste (c_j)
S_1	{1, 2}	3
S_2	{2, 3}	2
S_3	{3, 4}	4
S_4	{4, 5}	3
S_5	{1, 5}	5

- Siendo el conjunto de las zonas: {1, 2, 3, 4, 5}, la solución óptima es:
  - Subconjunto 1: Cubre elementos [1, 2]
  - Subconjunto 2: Cubre elementos [2, 3]
  - Subconjunto 4: Cubre elementos [4, 5]
  - Coste mínimo total: 8.0

Problema de cubrimiento (ejemplo)

Función objetivo:

$$Minimizar\ Z = 3x_1 + 2x_2 + 4x_3 + 3x_4 + 5x_5$$

Sujeto a:

$$x_1 + x_5 \geq 1$$
$$x_1 + x_2 \geq 1$$
$$x_2 + x_3 \geq 1$$
$$x_3 + x_4 \geq 1$$
$$x_4 + x_5 \geq 1$$

Variables de decisión:

$$x_i \in \{0, 1\}, \forall_i$$

Problema de cubrimiento

Sea:

- $E = \{e_1, e_2, \dots, eM\}$: $conjunto\ de\ elementos\ a\ cubrir$
- $S = \{S_1, S_2, \dots, SN\}$: $conjunto\ de\ subconjuntos\ con\ elementos\ \subset E$
- c_j: $coste\ asociado\ a\ seleccionar\ el\ subconjunto\ Sj.$
- $a_{ij} = \begin{cases} 1\ si\ ei\ \in S_j \\ 0\ si\ no \end{cases}$

Variables de decisión:

$$x_j \in \{0, 1\}, \forall_i = 1, 2, \dots, N$$

$$x_j = 1\ si\ el\ subconjunto\ S_j\ es\ seleccionado$$

$$x_j = 0\ en\ caso\ contrario$$

Problema de cubrimiento

Función objetivo:

$$Minimizar\ Z = \sum_{j=1}^{N} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{N} a_{ij} x_j \geq 1, \forall_i = 1, 2, \dots, M$$

Variables de decisión:

$$x_j \in \{0, 1\}, \forall_j = 1, 2, \dots, N$$

Problema de cubrimiento

```python
from gurobipy import Model, GRB

# Datos
elementos = [1, 2, 3, 4, 5]
subconjuntos = [1, 2, 3, 4, 5]

# Coste de los subconjuntos
costes = {1: 3, 2: 2, 3: 4, 4: 3, 5: 5}

# Elementos cubiertos por cada subconjunto
cobertura = {
    1: [1, 2],
    2: [2, 3],
    3: [3, 4],
    4: [4, 5],
    5: [1, 5]
}

# Crear modelo
modelo = Model("Problema_de_Cubrimiento")

# Variables de decisión
x = modelo.addVars(subconjuntos, vtype=GRB.BINARY,
name="x")

# Función objetivo
modelo.setObjective(
    sum(costes[j] * x[j] for j in subconjuntos),
    GRB.MINIMIZE
)
```

```python
# Restricciones de cobertura
for i in elementos:
    modelo.addConstr(
        sum(x[j] for j in subconjuntos if i in cobertura[j]) >= 1,
        name=f"Cobertura_{i}"
    )

# Optimizar modelo
modelo.optimize()

# Imprimir solución
if modelo.status == GRB.OPTIMAL:
    print(f"\nCoste mínimo total: {modelo.ObjVal}")
    print("Subconjuntos seleccionados:")
    for j in subconjuntos:
        if x[j].X > 0.5:
            print(f" - Subconjunto {j}: Cubre elementos {cobertura[j]}")
else:
    print("No se encontró una solución óptima.")
```

Problema de partición

- El problema de partición consiste en dividir un conjunto universal en subconjuntos disjuntos de manera que cada elemento del conjunto universal pertenezca a exactamente uno de los subconjuntos seleccionados.
- En otras palabras, todos los elementos de un conjunto deben estar contenidos en uno de los subconjuntos seleccionados.
- Aplicaciones
  - Programación de horarios: Asignar tareas o turnos de manera que cada tarea se realice una sola vez y los recursos se utilicen eficientemente.
  - Agrupamiento de datos: Dividir un conjunto de datos en grupos exclusivos.
  - Asignación de tripulaciones: Asignar tripulaciones a vuelos sin superposiciones.

https://epubs.siam.org/doi/abs/10.1137/1018115

Problema de partición

- Dados los siguientes subconjuntos:

Subconjunto (S_j)	Zonas cubiertas (e_i)	Coste (c_j)
S_1	{1, 2}	3
S_2	{2, 3}	2
S_3	{3, 4}	4
S_4	{4, 5}	3
S_5	{1, 5}	5

- Siendo el conjunto de las zonas: {1, 2, 3, 4, 5}, la solución óptima es:
  - No hay soluciones factibles

Problema de partición

- Dados los siguientes subconjuntos:

Subconjunto (S_j)	Zonas cubiertas (e_i)	Coste (c_j)
S_1	{1, 2}	3
S_2	{2, 3}	2
S_3	{3, 4}	4
S_4	{4, 5}	3
S_5	{1, 5}	5
S_6	{5}	2

- Siendo el conjunto de las zonas: {1, 2, 3, 4, 5}, la solución óptima es:
  - Subconjunto 1: Cubre elementos [1, 2]
  - Subconjunto 3: Cubre elementos [3, 4]
  - Subconjunto 6: Cubre elementos [5]
  - Coste mínimo total: 9.0

Problema de partición (ejemplo)

Función objetivo:

$$Minimizar\ Z = 3x_1 + 2x_2 + 4x_3 + 3x_4 + 5x_5 + 2x_6$$

Sujeto a:

$$x_1 + x_5 = 1$$
$$x_1 + x_2 = 1$$
$$x_2 + x_3 = 1$$
$$x_3 + x_4 = 1$$
$$x_4 + x_5 + x_6 = 1$$

Variables de decisión:

$$x_i \in \{0, 1\}, \forall_i$$

Problema de partición

Sea:

- $E = \{e_1, e_2, \dots, eM\}$: $conjunto\ de\ elementos\ a\ cubrir$
- $S = \{S_1, S_2, \dots, SN\}$: $conjunto\ de\ subconjuntos\ con\ elementos\ \subset E$
- c_j: $coste\ asociado\ a\ seleccionar\ el\ subconjunto\ Sj.$
- $a_{ij} = \begin{cases} 1\ si\ ei\ \in S_j \\ 0\ si\ no \end{cases}$

Variables de decisión:

$$x_j \in \{0, 1\}, \forall_i = 1, 2, \dots, N$$

$$x_j = 1\ si\ el\ subconjunto\ S_j\ es\ seleccionado$$

$$x_j = 0\ en\ caso\ contrario$$

Problema de partición

Función objetivo:

$$Minimizar\ Z = \sum_{j=1}^{N} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{N} a_{ij} x_j = 1, \forall_i = 1, 2, \dots, M$$

Variables de decisión:

$$x_j \in \{0, 1\}, \forall_j = 1, 2, \dots, N$$

Problema de partición

```python
from gurobipy import Model, GRB

# Datos
elementos = [1, 2, 3, 4, 5]
subconjuntos = [1, 2, 3, 4, 5, 6]

# Coste de los subconjuntos
costes = {1: 3, 2: 2, 3: 4, 4: 3, 5: 5, 6: 2}

# Elementos cubiertos por cada subconjunto
cobertura = {
    1: [1, 2],
    2: [2, 3],
    3: [3, 4],
    4: [4, 5],
    5: [1, 5],
    6: [5]  # Nuevo subconjunto que cubre solo el elemento 5
}

# Crear modelo
modelo_particion = Model("Problema_de_Particion")

# Variables de decisión
x = modelo_particion.addVars(subconjuntos, vtype=GRB.BINARY, name="x")

# Función objetivo
modelo_particion.setObjective(
    sum(costes[j] * x[j] for j in subconjuntos),
    GRB.MINIMIZE
)

# Restricciones de partición
for i in elementos:
    modelo_particion.addConstr(
        sum(x[j] for j in subconjuntos if i in cobertura[j]) == 1,
        name=f"Particion_{i}"
    )

# Optimizar modelo
modelo_particion.optimize()

# Imprimir solución
if modelo_particion.status == GRB.OPTIMAL:
    print(f"\nCoste mínimo total: {modelo_particion.ObjVal}")
    print("Subconjuntos seleccionados:")
    for j in subconjuntos:
        if x[j].X > 0.5:
            print(f" - Subconjunto {j}: Cubre elementos {cobertura[j]}")
else:
    print("No se encontró una solución óptima.")
```

Diferencias entre problemas

Restricciones de cobertura:

- Cubrimiento: Cada elemento debe estar cubierto al menos una vez (≥1).
- Partición: Cada elemento debe estar cubierto exactamente una vez (=1).

Solapamiento de subconjuntos:

- Cubrimiento: Los subconjuntos seleccionados pueden solaparse.
- Partición: Los subconjuntos seleccionados son disjuntos.

Problema de conjunto de empaquetamiento

- El problema de conjunto de empaquetamiento (set packing problema) es un problema clásico en optimización combinatoria y teoría de grafos.
- Consiste en seleccionar un conjunto de subconjuntos disjuntos de manera que se maximice algún criterio.
- Aplicaciones
  - Asignación de tareas: Asignar tareas a recursos limitados sin conflictos.
  - Programación de horarios: Seleccionar actividades que no se solapen en el tiempo.
  - Selección de equipos: Formar equipos donde los miembros no pertenezcan a más de un equipo.
  - Problemas de emparejamiento máximo: Encontrar el máximo emparejamiento en un grafo.

Problema de conjunto de empaquetamiento

- Dados los siguientes subconjuntos:

Tarea (S_j)	Actividades (e_i)	Beneficio (c_j)
S_1	{A, B}	5
S_2	{C, D}	7
S_3	{B, D}	4
S_4	{E}	3
S_5	{A, E}	6

- Siendo el conjunto de las zonas: {A, B, C, D, E}, la solución óptima es:
  - Subconjunto 1: Elementos ['A', 'B'], Beneficio 5
  - Subconjunto 2: Elementos ['C', 'D'], Beneficio 7
  - Subconjunto 4: Elementos ['E'], Beneficio 3
  - Beneficio máximo total: 15.0

Problema de conjunto de empaquetamiento

Sea:

- $E = \{e_1, e_2, \dots, eM\}$: $conjunto\ de\ elementos$
- $S = \{S_1, S_2, \dots, SN\}$: $conjunto\ de\ subconjuntos\ E$
- c_j: $beneficio\ asociado\ a\ seleccionar\ el\ subconjunto\ Sj$.
- $a_{ij} = \begin{cases} 1\ si\ ei\ \in S_j \\ 0\ si\ no \end{cases}$

Variables de decisión:

$$x_j \in \{0, 1\}, \forall_i = 1, 2, \dots, N$$

$$x_j = 1\ si\ el\ subconjunto\ S_j\ es\ seleccionado$$

$$x_j = 0\ en\ caso\ contrario$$

Problema de conjunto de empaquetamiento

Función objetivo:

$$Minimizar\ Z = \sum_{j=1}^{N} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{N} a_{ij} x_j \leq 1, \forall_i = 1, 2, \dots, M$$

Variables de decisión:

$$x_j \in \{0, 1\}, \forall_j = 1, 2, \dots, N$$

Problema de conjunto de empaquetamiento

```python
# Importar la librería gurobipy
from gurobipy import Model, GRB

elementos = ['A', 'B', 'C', 'D', 'E']

# Conjunto de subconjuntos y sus elementos
subconjuntos = {
    1: {'elementos': ['A', 'B'], 'beneficio': 5},
    2: {'elementos': ['C', 'D'], 'beneficio': 7},
    3: {'elementos': ['B', 'D'], 'beneficio': 4},
    4: {'elementos': ['E'], 'beneficio': 3},
    5: {'elementos': ['A', 'E'], 'beneficio': 6},
}

# Crear el modelo
modelo = Model("Problema_de_Conjunto_de_Empaquetamiento")

# Desactivar la salida de Gurobi (opcional)
modelo.Params.OutputFlag = 0

# Variables de decisión: x_j = 1 si el subconjunto j es seleccionado
x = modelo.addVars(subconjuntos.keys(), vtype=GRB.BINARY, name="x")

# Función objetivo: maximizar el beneficio total
modelo.setObjective(
    sum(subconjuntos[j]['beneficio'] * x[j] for j in subconjuntos),
    GRB.MAXIMIZE
)
```

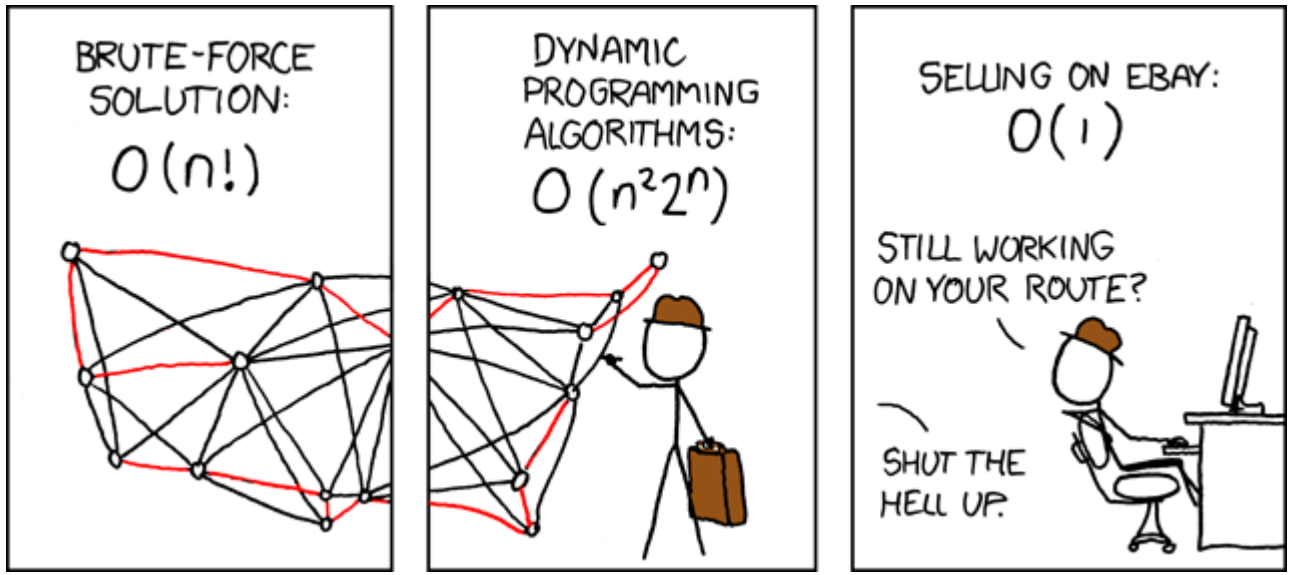
```python
# Restricciones: cada elemento puede pertenecer a como máximo un subconjunto seleccionado
for e in elementos:
    modelo.addConstr(
        sum(x[j] for j in subconjuntos if e in subconjuntos[j]['elementos']) <= 1,
        name=f"Elemento_{e}"
    )

# Optimizar el modelo
modelo.optimize()

# Verificar si se encontró una solución óptima
if modelo.status == GRB.OPTIMAL:
    print(f"\nBeneficio máximo total: {modelo.ObjVal}")
    print("Subconjuntos seleccionados:")
    for j in subconjuntos:
        if x[j].X > 0.5:
            elems = subconjuntos[j]['elementos']
            ben = subconjuntos[j]['beneficio']
            print(f" - Subconjunto {j}: Elementos {elems}, Beneficio {ben}")
else:
    print("No se encontró una solución óptima.")
```

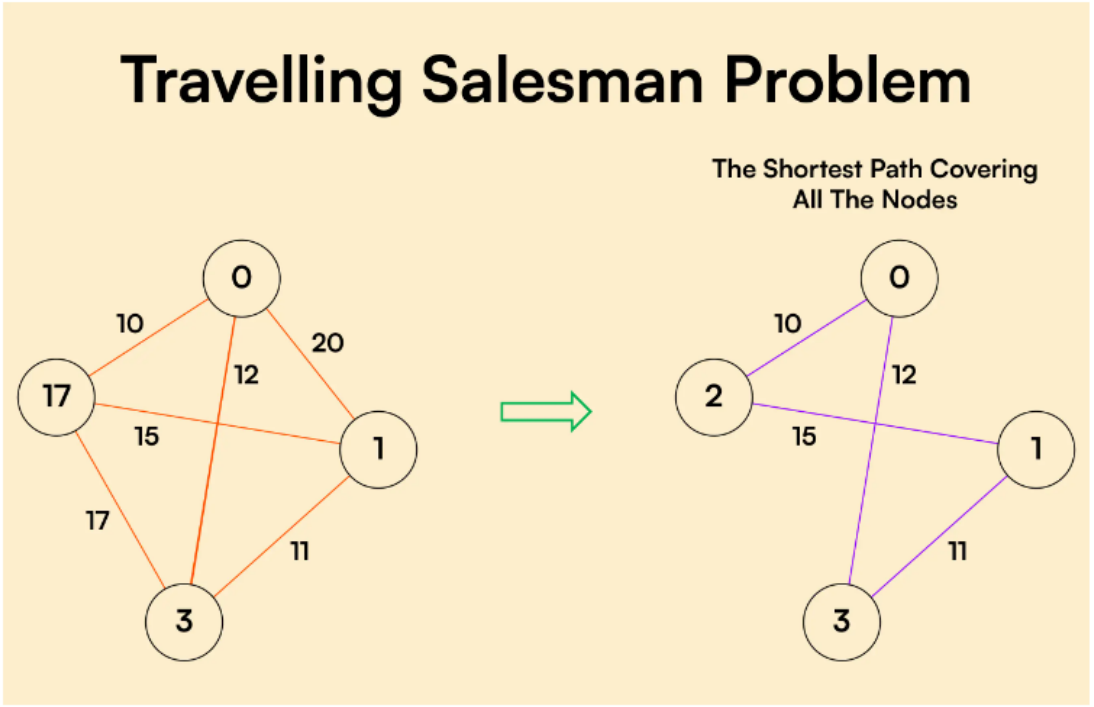
Problema del viajero

- El problema del viajero (Traveling Salesman Problem - TSP) es uno de los problemas más clásicos y estudiados en el campo de la optimización combinatoria y la investigación operativa.
- Consiste en encontrar la ruta más corta posible que permita a un viajero visitar un conjunto de ciudades exactamente una vez y regresar a la ciudad de origen. A pesar de su enunciado sencillo, es un problema de gran complejidad computacional y es NP-duro.
- Aplicaciones:
  - Logística y distribución: Planificación de rutas de vehículos para entregas.
  - Manufactura: Optimización de rutas en máquinas CNC.
  - Bioinformática: Secuenciación de genes y proteínas.
  - Planificación de circuitos: Diseño de circuitos impresos y chips.

https://www.sciencedirect.com/science/article/pii/S0377221724002959

Problema del viajero





Problema del viajero

- Dados la siguiente tabla con ciudades y distancias:

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	0	10	8	9	7
Ciudad 2	10	0	10	5	6
Ciudad 3	8	10	0	8	9
Ciudad 4	9	5	8	0	6
Ciudad 5	7	6	9	6	0

- La solución óptima es:
  - La ruta óptima es: Ciudad 1 → Ciudad 5 → Ciudad 4 → Ciudad 2 → Ciudad 3 → Ciudad 1.
  - El coste mínimo del recorrido es 34 unidades.

Problema del viajero

$Función\ objetivo$:

$$Minimizar\ Z = \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij} x_{ij}$$

$Sujeto\ a$:

$$\sum_{j=1, j\neq i}^{N} x_{ij} = 1, \forall_i = 1, 2, \dots, N\ (Sale\ una\ vez\ de\ cada\ ciudad)$$

$$\sum_{j=1, j\neq i}^{N} x_{ji} = 1, \forall_i = 1, 2, \dots, N (Entra\ una\ vez\ a\ cada\ ciudad)$$

Problema del viajero

- Para evitar ciclos más pequeños que incluyan solo un subconjunto de ciudades se agregan restricciones adicionales.
- Hay que garantizar que para cualquier conjunto de nodos el número máximo de arcos incluidos en la solución sea de uno menos que el número de nodos.
- Formulación de Miller-Tucker-Zemlin (MTZ).

Problema del viajero

Sujeto a (continuación):

$$u_i - uj + nxij \leq n - 1, \qquad \forall_i \neq j; \quad i, j = 2, \dots, N$$

Variables binarias:

$$x_{ij} \in \{0, 1\}, \qquad \forall_{i,\, j} = 1, 2, \dots, N; \quad i \neq j$$

Variables continuas (MTZ):

$$2 \leq u_i \leq n, \qquad \forall_{ij} = 2, \dots, N$$

Problema del viajero

```
from gurobipy import Model, GRB, quicksum

# Datos
n = 5  # Número de ciudades
ciudades = range(1, n + 1)

# Distancias entre ciudades (matriz c_{ij})
c = {
    (1, 2): 10, (1, 3): 8,  (1, 4): 9,  (1, 5): 7,
    (2, 1): 10, (2, 3): 10, (2, 4): 5,  (2, 5): 6,
    (3, 1): 8,  (3, 2): 10, (3, 4): 8,  (3, 5): 9,
    (4, 1): 9,  (4, 2): 5,  (4, 3): 8,  (4, 5): 6,
    (5, 1): 7,  (5, 2): 6,  (5, 3): 9,  (5, 4): 6
}

# Crear el modelo
modelo = Model("Problema_del_Viajero")

# Variables de decisión: x[i,j] = 1 si se viaja de ciudad i a ciudad j
x = modelo.addVars(ciudades, ciudades, vtype=GRB.BINARY, name="x")

# Variables auxiliares para eliminar subtours (MTZ)
u = modelo.addVars(ciudades, vtype=GRB.CONTINUOUS, lb=1, ub=n, name="u")

# Establecer la función objetivo
modelo.setObjective(
    quicksum(c[i, j] * x[i, j] for i in ciudades for j in ciudades if i != j),
    GRB.MINIMIZE
)
```

- Datos del problema

- Variables de decisión
- Variables auxiliares
- Función objetivo

Problema del viajero

```
# Restricciones de flujo

# 1. Sale exactamente una vez de cada ciudad
for i in ciudades:
    modelo.addConstr(
        quicksum(x[i, j] for j in ciudades if j != i) == 1,
        name=f"Salida_{i}"
    )

# 2. Entra exactamente una vez a cada ciudad
for j in ciudades:
    modelo.addConstr(
        quicksum(x[i, j] for i in ciudades if i != j) == 1,
        name=f"Entrada_{j}"
    )

# Restricciones de eliminación de subtours (MTZ)
for i in ciudades:
    if i != 1:
        modelo.addConstr(u[i] >= 2, name=f"u_min_{i}")
        modelo.addConstr(u[i] <= n, name=f"u_max_{i}")

for i in ciudades:
    for j in ciudades:
        if i != j and i != 1 and j != 1:
            modelo.addConstr(
                u[i] - u[j] + n * x[i, j] <= n - 1,
                name=f"MTZ_{i}_{j}"
            )
```

- Restricciones para visitar cada ciudad una única vez.
- Para evitar ciclos dentro del propio grafo.

Problema del viajero

```
# Optimizar el modelo
modelo.optimize()

# Imprimir la solución
if modelo.status == GRB.OPTIMAL:
    print(f"\nCoste mínimo del recorrido: {modelo.ObjVal}")
    ruta = []
    ciudad_actual = 1
    while True:
        ruta.append(ciudad_actual)
        for j in ciudades:
            if j != ciudad_actual and x[ciudad_actual, j].X > 0.5:
                siguiente_ciudad = j
                break
        if siguiente_ciudad == 1:
            ruta.append(1)
            break
        else:
            ciudad_actual = siguiente_ciudad
    print("Ruta óptima:")
    print(" -> ".join(map(str, ruta)))
else:
    print("No se encontró una solución óptima.")
```

- Resolver el problema
- Imprimir resultado

Modelado de condiciones lógicas

- El uso de variables binarias permite modelizar condiciones lógicas que permiten obtener modelos más complejos.
- **Costes fijos:** la realización de una actividad conlleva un gasto fijo, independientemente del nivel de actividad.
- **Variables semicontinuas:** si se realiza una actividad, se hace a un nivel mínimo.
- **Implicaciones:** si se hace una actividad, se deben hacer otras o se deben impedir otras.
- **Funciones no lineales:** aproximación por funciones lineales por partes.
- Etc.

Aceites

- Una fábrica produce aceite mezclando aceites refinados, dos de origen vegetal y tres de origen no vegetal.
- En un mes sólo es posible refinar 200 toneladas de vegetal y 250 de no vegetal.
- El aceite resultante debe cumplir un valor de dureza comprendido entre 3 y 6. El coste de una tonelada para cada aceite refinado junto con su dureza aparecen en la siguiente tabla:

	VEG_1	*VEG_2*	*NOVEG_1*	*NOVEG_2*	*NOVEG_3*
costo	110	120	130	110	115
dureza	8,8	6,1	2,0	4,2	5,0

- Se trata de refinar las cantidades apropiadas de cada aceite a fin de maximizar el beneficio de la producción final sabiendo que una tonelada del aceite producido se vende a 150.

Aceites

- (I) El alimento final no puede contener más de tres tipos de aceite diferentes.
- (II) Si el producto final contiene un cierto tipo de aceite, debe contener al menos 20 toneladas de este.
- (III) Si la mezcla contiene algún tipo de aceite vegetal (VEG1 o VEG2), entonces también debe contener aceite no vegetal de tipo 3 (OIL3).

Aceites

- (I) El alimento final no puede contener más de tres tipos de aceite diferentes.

$$\delta = \begin{cases} 1\ si\ se\ toma\ la\ decisión \\ 0\ no\ se\ toma\ la\ decisión \end{cases}$$

$$\delta_i = \begin{cases} 1\ si\ se\ utiliza\ el\ aceite\ de\ tipo\ i \\ 0\ no\ se\ utiliza\ el\ aceite \end{cases} \quad i = 1, \dots, 5$$

- **Si no se utiliza un tipo de aceite, no se puede mezclar nada de ese tipo de aceite.**
- Si $\delta_i = 0$, entonces $x_i = 0$.
- Si $\delta_i = 1$, entonces $x_i \geq 0$.

Aceites

- Esto se puede representar de la siguiente forma:

$$x_1 \leq 200\ \delta_1$$
$$x_2 \leq 200\ \delta_2$$
$$x_3 \leq 250\ \delta_3$$
$$x_4 \leq 250\ \delta_4$$
$$x_5 \leq 250\ \delta_5$$

- Sólo se puede mezclar una cantidad positiva de aceite ($x_i > 0$) se previamente se ha decidido usar ese tipo de aceite en la mezcla ($\delta_i = 1$)
- Sólo tres variables δ pueden ser igual a 1.
- Esto se representa limitando a que la suma de las variables sea menor o igual que 3:

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 \leq 3$$

Aceites

- Hay distintas opciones de selección múltiple.

- El aceite final debe contener **al menos** dos tipos de aceite diferentes:

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 \geq 2$$

- El aceite final debe contener **exactamente** dos tipos de aceite diferentes:

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 = 2$$

- El aceite final debe contener **como mucho** dos tipos de aceite diferentes:

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 \leq 2$$

Aceites

- (II) Si el producto final contiene un cierto tipo de aceite, debe contener al menos 20 toneladas de este.

$$Si\ \delta_i = 1, entonces\ xi \geq 20$$

$$20\delta_i \leq xi, i = 1, \dots, 5$$

- Las variables x_i se denominan **variables semicontinuas**, ya que no son estrictamente continuas: pueden tomar valores en el intervalo [20, 200] o [20, 250], pero además pueden tomar el valor 0 (pero no pueden tomar ningún valor en (0, 20)).

Aceites

- (III) Si la mezcla contiene algún tipo de aceite vegetal (VEG1 o VEG2), entonces también debe contener aceite no vegetal de tipo 3 (OIL3).

$$Si\ \delta_1 = 1\ o\ \delta_2 = 1, entonces\ \delta_5 = 1$$

- Que una de las dos variables δ_1 y δ_2 tomen el valor 1, se sabe si la suma de las dos es mayor o igual que 1:

$$Si\ \delta_1 + \ \delta_2 \geq 1, entonces\ \delta_5 = 1$$

- A partir de una única restricción:

$$\delta_1 + \ \delta_2 \leq 2\delta_5$$

- A partir de dos restricciones:

$$\delta_1 \leq \delta_5$$
$$\delta_2 \leq \delta_5$$

Aceites

- El uso de las variables binarias permite definir muchas condiciones lógicas.
- Por ejemplo, los aceites VEG1 y VEG2 son incompatibles (no se pueden utilizar simultáneamente):

$$\delta_1 + \delta_2 \leq 1$$

- Otro ejemplo, si se utiliza VEG1 y VEG2, entonces, no puede utilizarse OIL3:

$$\delta_1 + \delta_2 \leq 2 - \delta_5$$

Condiciones disyuntivas

- En general las condiciones disyuntivas (o se cumple una cosa o la otra, o ambas):

$$\sum_i a_i x_i \leq a \quad o \quad \sum_j b_j y_j \geq b$$

- Se pueden modelar utilizando una nueva variable binaria δ:

$$\sum_i a_i x_i \leq a + M\delta \quad o \quad \sum_j b_j y_j \geq b + m(1-\delta)$$

- Siendo M suficientemente grande y m suficientemente pequeño (negativo).
- De esta forma, si δ = 0 al menos se cumple la primera restricción y si δ = 1 al menos se cumple la segunda.

Condiciones disyuntivas

- Las implicaciones simples se pueden modelar como condiciones disyuntivas sin más que aplicar resultados básicos de lógica

$$A \Rightarrow B \equiv \neg B \Rightarrow \neg A \equiv \neg A \vee B$$

- Por tanto, las siguientes expresiones son equivalentes:

$$\sum_i a_i x_i \leq a \Rightarrow \sum_j b_j y_j \geq b$$

$$\sum_i a_i x_i > a \; o \sum_j b_j y_j \geq b$$

Aceites

$$Maximizar\ Z =$$
$$150t - 110v_1 - 120v_2 - 130nv_1 - 110nv_2 - 115nv_3$$
$$Sujeto\ a:$$
$$v_1 + v_2 \leq 200$$
$$nv_1 + nv_2 + nv_3 \leq 250$$
$$8.8v_1 + 6.1v_2 + 2nv_1 + 4.2nv_2 + 5nv_3 \leq 6t$$
$$8.8v_1 + 6.1v_2 + 2nv_1 + 4.2nv_2 + 5nv_3 \geq 3t$$
$$v_1 + v_2 + nv_1 + nv_2 + nv_3 = t$$
$$20\delta_1 \leq v_1 \leq 200\delta_1$$
$$20\delta_2 \leq v_2 \leq 200\delta_2$$
$$20\delta_3 \leq nv_1 \leq 250\delta_3$$
$$20\delta_4 \leq nv_2 \leq 250\delta_4$$
$$20\delta_5 \leq nv_3 \leq 250\delta_5$$
$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 \leq 3$$
$$\delta_1 \leq \delta_5$$
$$\delta_2 \leq \delta_5$$
$$v_1, v_2, nv_1, nv_2, nv_3 \in \mathbb{Z}$$

Solución Óptima:

v_1 = 155

v_2 = 0

nv_1 = 0

nv_2 = 230

nv_3 = 20

y = 405.00

Valor óptimo de la función objetivo Z = 16100

Aceites

```
# Importar la librería gurobipy
from gurobipy import Model, GRB

# Crear un nuevo modelo
model = Model("Problema_de_Optimización")

# Desactivar la salida de Gurobi (opcional)
model.Params.OutputFlag = 1  # Puedes cambiar a 0 para desactivar la salida

# Variables enteras
v1 = model.addVar(vtype=GRB.INTEGER, name="v1")
v2 = model.addVar(vtype=GRB.INTEGER, name="v2")
nv1 = model.addVar(vtype=GRB.INTEGER, name="nv1")
nv2 = model.addVar(vtype=GRB.INTEGER, name="nv2")
nv3 = model.addVar(vtype=GRB.INTEGER, name="nv3")
t = model.addVar(vtype=GRB.INTEGER, name="t")

# Variables binarias
delta1 = model.addVar(vtype=GRB.BINARY, name="delta1")
delta2 = model.addVar(vtype=GRB.BINARY, name="delta2")
delta3 = model.addVar(vtype=GRB.BINARY, name="delta3")
delta4 = model.addVar(vtype=GRB.BINARY, name="delta4")
delta5 = model.addVar(vtype=GRB.BINARY, name="delta5")

# Actualizar el modelo para integrar las variables
model.update()

# Establecer la función objetivo
model.setObjective(
    150 * t - 110 * v1 - 120 * v2 - 130 * nv1 - 110 * nv2 - 115 * nv3,
    GRB.MAXIMIZE
)
```

```
# Agregar las restricciones

# Restricción 1
model.addConstr(v1 + v2 <= 200, name="R_Vegetal")

# Restricción 2
model.addConstr(nv1 + nv2 + nv3 <= 250, name="R_NoVegetal")

# Restricción 3
model.addConstr(
    8.8 * v1 + 6.1 * v2 + 2 * nv1 + 4.2 * nv2 + 5 * nv3 <= 6 * t,
    name="Dureza_Max"
)

# Restricción 4
model.addConstr(
    8.8 * v1 + 6.1 * v2 + 2 * nv1 + 4.2 * nv2 + 5 * nv3 >= 3 * t,
    name="Dureza_Min"
)

# Restricción 5
model.addConstr(
    v1 + v2 + nv1 + nv2 + nv3 == t,
    name="Total_Aceite"
)
```

Aceites

```
# Restricciones 6 a 10 (Vinculación con variables binarias)
model.addConstr(v1 >= 20 * delta1, name="V1_inferior")
model.addConstr(v1 <= 200 * delta1, name="V1_superior")

model.addConstr(v2 >= 20 * delta2, name="V2_inferior")
model.addConstr(v2 <= 200 * delta2, name="V2_superior")

model.addConstr(nv1 >= 20 * delta3, name="NV1_inferior")
model.addConstr(nv1 <= 250 * delta3, name="NV1_superior")

model.addConstr(nv2 >= 20 * delta4, name="NV2_inferior")
model.addConstr(nv2 <= 250 * delta4, name="NV2_superior")

model.addConstr(nv3 >= 20 * delta5, name="NV3_inferior")
model.addConstr(nv3 <= 250 * delta5, name="NV3_superior")

# Restricción 11
model.addConstr(
    delta1 + delta2 + delta3 + delta4 + delta5 <= 3,
    name="Total_Aceite_Max"
)
# Restricciones 12 y 13
model.addConstr(delta1 <= delta5, name="Si_vegetal1_no_vegetal")
model.addConstr(delta2 <= delta5, name="Si_vegetal2_no_vegetal")
```

```
# Optimizar el modelo
model.optimize()

# Imprimir la solución
if model.status == GRB.OPTIMAL:
    print(f"\nValor óptimo de la función objetivo: {model.ObjVal}")
    print("Valores de las variables de decisión:")
    print(f"v1 = {v1.X}")
    print(f"v2 = {v2.X}")
    print(f"nv1 = {nv1.X}")
    print(f"nv2 = {nv2.X}")
    print(f"nv3 = {nv3.X}")
    print(f"t = {t.X}")
    print(f"delta1 = {delta1.X}")
    print(f"delta2 = {delta2.X}")
    print(f"delta3 = {delta3.X}")
    print(f"delta4 = {delta4.X}")
    print(f"delta5 = {delta5.X}")
else:
    print("No se encontró una solución óptima.")
```

Aceites

Variables enteras no negativas:

$x_k \geq 0$, para $k \in K=\{1,2,3,4,5\}$ donde:

- $x_1=v_1$
- $x_2=v_2$
- $x_3=nv_1$
- $x_4=nv_2$
- $x_5=nv_3$

Variables binarias:

$$\delta_k \in \{0,1\}, para\ k\ \in K = \{1,2,3,4,5\}$$

Variable entera no negativa:

$$t\ \geq 0$$

Aceites

Costes asociados a cada variable x_k:

c_k, para $k \in K=\{1,2,3,4,5\}$ donde:

- $c_1=110$
- $c_2=120$
- $c_3=130$
- $c_4=110$
- $c_5=115$

Dureza asociada a cada variable x_k:

a_k, para $k \in K$ donde:

- $a_1=8.8$
- $a_2=6.1$
- $a_3=2$
- $a_4=4.2$
- $a_5=5$

Aceites

Límites superiores (cantidad) U_k:

- Para k=1,2: U_k=200
- Para k=3,4,5:U_k=250

Conjuntos:

- V={1,2} (variables v_1,v_2)
- NV={3,4,5} (variables nv_1,nv_2,nv_3)

Aceites

$$Maximizar\ Z = 150t - \sum_{k=1}^{N} c_k x_k$$

$Sujeto\ a$:

$$\sum_{i \in V} x_i \leq 200$$

$$\sum_{j \in NV} x_j \leq 250$$

$$\sum_{k=1}^{N} a_k x_k \leq 6t$$

$$\sum_{k=1}^{N} a_k x_k \geq 3t$$

$$\sum_{k=1}^{N} x_k = t$$

$$20\delta_k \leq x_k \leq U_k \delta_k, \forall_K \in K$$

$$\sum_{k=1}^{N} \delta_k \leq 3$$

$$\delta_1 \leq \delta_5$$

$$\delta_2 \leq \delta_5$$

$$t, xk \in \mathbb{Z}$$

$$\delta_k \in \{0, 1\}, \forall_K \in K$$

Aceites

```python
# Importar la librería gurobipy
from gurobipy import Model, GRB

# Datos
K = [1, 2, 3, 4, 5]  # Conjunto de aceites
V = [1, 2]           # Aceites vegetales
NV = [3, 4, 5]       # Aceites no vegetales

# Costes asociados a cada aceite
c = {1: 110, 2: 120, 3: 130, 4: 110, 5: 115}

# Coeficientes de consumo
a = {1: 8.8, 2: 6.1, 3: 2, 4: 4.2, 5: 5}

# Límites superiores de producción
U = {1: 200, 2: 200, 3: 250, 4: 250, 5: 250}

# Crear el modelo
model = Model("Modelo_Simplificado")

# Desactivar la salida de Gurobi (opcional)
model.Params.OutputFlag = 1  # Cambia a 0 para desactivar la salida

# Variables de decisión
x = model.addVars(K, vtype=GRB.INTEGER, lb=0, name="x")
delta = model.addVars(K, vtype=GRB.BINARY, name="delta")
t = model.addVar(vtype=GRB.INTEGER, lb=0, name="t")

# Función objetivo
model.setObjective(150 * t - sum(c[k] * x[k] for k in K), GRB.MAXIMIZE)

# Restricciones

# 1. Capacidad de aceites vegetales
model.addConstr(sum(x[i] for i in V) <= 200, name="Capacidad_Vegetal")

# 2. Capacidad de aceites no vegetales
model.addConstr(sum(x[j] for j in NV) <= 250, name="Capacidad_No_Vegetal")

# 3. Restricciones de consumo

# Consumo máximo
model.addConstr(sum(a[k] * x[k] for k in K) <= 6 * t, name="Consumo_Maximo")

# Consumo mínimo
model.addConstr(sum(a[k] * x[k] for k in K) >= 3 * t, name="Consumo_Minimo")

# 4. Relación entre x_k y t
model.addConstr(sum(x[k] for k in K) == t, name="Produccion_Total")

# 5. Vinculación con variables binarias
for k in K:
    model.addConstr(x[k] >= 20 * delta[k], name=f"Vinculacion_Min_{k}")
    model.addConstr(x[k] <= U[k] * delta[k], name=f"Vinculacion_Max_{k}")

# 6. Restricción en el número máximo de aceites utilizados
model.addConstr(sum(delta[k] for k in K) <= 3, name="Max_Aceites_Usados")

# 7. Restricciones lógicas
model.addConstr(delta[1] <= delta[5], name="Logica_delta1_delta5")
model.addConstr(delta[2] <= delta[5], name="Logica_delta2_delta5"
```

Aceites

```
# Optimizar el modelo
model.optimize()

# Imprimir la solución
if model.status == GRB.OPTIMAL:
    print(f"\nValor óptimo de la función objetivo: {model.ObjVal}")
    print("Valores de las variables de decisión:")
    for k in K:
        print(f"Aceite {k}: x_{k} = {x[k].X}, delta_{k} = {int(delta[k].X)}")
    print(f"Producción total (t): {t.X}")
else:
    print("No se encontró una solución óptima.")
```

Valor óptimo de la función objetivo: 16100.0
Valores de las variables de decisión:
Aceite 1: x_1 = 155.0, delta_1 = 1
Aceite 2: x_2 = -0.0, delta_2 = 0
Aceite 3: x_3 = -0.0, delta_3 = 0
Aceite 4: x_4 = 230.0, delta_4 = 1
Aceite 5: x_5 = 20.0, delta_5 = 1
Producción total (t): 405.0

Process finished with exit code 0

Aceites

Costes fijos

- Supongamos que la empresa tiene que comprar los depósitos en los que va a almacenar los aceites que utilice, con un coste de 750 euros por depósito.
- En este caso, aparece un elemento nuevo, un coste fijo (compra del depósito) que es independiente de la cantidad de aceite que vaya a contener.
- La función de coste de cada tipo de aceite es de la forma: entonces es de la forma:

$$C = \begin{cases} c_1 + c_2 x \; si \; x > 0 \\ 0 \; si \; x = 0 \end{cases}$$

Aceites

Costes fijos

- En general, es suficiente con añadir a la función objetivo el coste fijo asociado a la variable binaria (y mantener el de la variable continua):

$$C = c_1\delta_i + c_2 x_i$$

- Y mantener las retricciones que ligan las variables binarias con las variables continuas

$$x_i \leq M\delta_i$$

- La función objetivo queda, por tanto:

$$Maximizar\ Z = 150t - \sum_{k=1}^{N} (ckxk + 750\delta_i)$$

Aceites

Costes fijos

- Supongamos que la empresa está considerando 2 modelos diferentes de depósito:

Modelo	Capacidad	Coste
I	200	750
II	100	450

- Construir un modelo que contemple la decisión sobre qué tipo de depósito adquirir para cada uno de los aceites considerados, considerando que solo puede comprar un depósito para cada tipo de aceite.

Aceites

Nuevas variables

$$\delta_{Ii} = \begin{cases} 1, si\ el\ aceite\ de\ tipo\ i\ se\ almacena\ en\ un\ depósito\ I \\ 0, en\ otro\ caso \end{cases}$$

$$\delta_{IIi} = \begin{cases} 1, si\ el\ aceite\ de\ tipo\ i\ se\ almacena\ en\ un\ depósito\ II \\ 0, en\ otro\ caso \end{cases}$$

Además, es necesario actualizar la función objetivo.

Si se utiliza un tipo de aceite, hay que comprar uno de los dos depósitos:

$$\delta_{Ii} + \delta_{IIi} = \delta_i, \forall i = 1, \dots, N$$

Ahora la cantidad de aceite comprada depende del depósito.

$$x_i \leq 200\delta_{Ii} + 100\delta_{IIi}, \forall i = 1, \dots, N$$

Aceites

Variables binarias de elección de depósito:

- $\delta_I^i \in \{0,1\}$ Indica si el aceite de tipo i se almacena en un depósito de tipo I.
- $\delta_{II}^i \in \{0,1\}$ Indica si el aceite de tipo i se almacena en un depósito de tipo II

Parámetros

- $f_I = 750\ coste\ fijo\ del\ depósito\ de\ tipo\ I$
- $f_{II} = 450\ coste\ fijo\ del\ depósito\ de\ tipo\ II$
- $C_I = 200\ capacidad\ del\ depósito\ de\ tipo\ I$
- $C_{II} = 100\ capacidad\ del\ depósito\ de\ tipo\ II$

Aceites

$$Maximizar\ Z = 150t - \sum_{k=1}^{N} (ckxk + f_I \delta_I^i + f_{II}\delta_{II}^i)$$

Sujeto a:

$$\sum_{i \in V} x_i \leq 200$$

$$\sum_{j \in NV} x_j \leq 250$$

$$\sum_{k=1}^{N} a_k x_k \leq 6t$$

$$\sum_{k=1}^{N} a_k x_k \geq 3t$$

$$\sum_{k=1}^{N} x_k = t$$

$$20\delta_k \leq x_k \leq U_k \delta_k, \forall_K \in K$$

$$\sum_{k=1}^{N} \delta_k \leq 3$$

$$\delta_1 \leq \delta_5$$

$$\delta_2 \leq \delta_5$$

$$x_i \leq C_I \delta_I^i + C_{II}\delta_{II}^i, \forall_i \in K$$

$$t, xk \in \mathbb{Z}$$

$$\delta_k \in \{0, 1\}, \forall_K \in K$$

Aceites

```python
# Importar la librería gurobipy
from gurobipy import Model, GRB

# Datos
K = [1, 2, 3, 4, 5]  # Conjunto de aceites
V = [1, 2]           # Aceites vegetales
NV = [3, 4, 5]       # Aceites no vegetales

# Costes variables por tonelada de cada aceite
c = {1: 110, 2: 120, 3: 130, 4: 110, 5: 115}

# Coeficientes de dureza
a = {1: 8.8, 2: 6.1, 3: 2, 4: 4.2, 5: 5}

# Costos fijos y capacidades de los depósitos
f_I = 750    # Coste del depósito tipo I
f_II = 450   # Coste del depósito tipo II

C_I = 200    # Capacidad del depósito tipo I
C_II = 100   # Capacidad del depósito tipo II

# Crear el modelo
model = Model("Modelo_Con_Nuevas_Variables_Delta")

# Desactivar la salida de Gurobi (opcional)
model.Params.OutputFlag = 1  # Cambia a 0 para desactivar la salida

# Variables de decisión
x = model.addVars(K, vtype=GRB.INTEGER, lb=0, name="x")
delta = model.addVars(K, vtype=GRB.BINARY, name="delta")
delta_I = model.addVars(K, vtype=GRB.BINARY, name="delta_I")
delta_II = model.addVars(K, vtype=GRB.BINARY, name="delta_II")
t = model.addVar(vtype=GRB.INTEGER, lb=0, name="t")

# Función objetivo
model.setObjective(
    150 * t - sum(c[i] * x[i] + f_I * delta_I[i] + f_II * delta_II[i] for i in K),
    GRB.MAXIMIZE
)

# Restricciones

# 1. Capacidad de aceites vegetales
model.addConstr(sum(x[i] for i in V) <= 200, name="Capacidad_Vegetal")

# 2. Capacidad de aceites no vegetales
model.addConstr(sum(x[i] for i in NV) <= 250, name="Capacidad_No_Vegetal")

# 3. Restricciones de consumo

# Dureza máxima
model.addConstr(sum(a[i] * x[i] for i in K) <= 6 * t, name="Consumo_Maximo")

# Dureza mínima
model.addConstr(sum(a[i] * x[i] for i in K) >= 3 * t, name="Consumo_Minimo")

# 4. Relación entre x_i y t
model.addConstr(sum(x[i] for i in K) == t, name="Produccion_Total")

# 5. Mínimo de producción si el aceite es utilizado
for i in K:
    model.addConstr(x[i] >= 20 * delta[i], name=f"Produccion_Min_{i}")

# 6. Capacidad de los depósitos
for i in K:
    model.addConstr(x[i] <= C_I * delta_I[i] + C_II * delta_II[i],
name=f"Capacidad_Deposito_{i}")
```

Aceites

```
# 7. Elección única de depósito por aceite
for i in K:
    model.addConstr(delta_I[i] + delta_II[i] == delta[i],
name=f"Deposito_Unico_{i}")

# 8. Restricción en el número máximo de aceites utilizados
model.addConstr(sum(delta[i] for i in K) <= 3,
name="Max_Aceites_Usados")

# 9. Restricciones lógicas
model.addConstr(delta[1] <= delta[5],
name="Logica_delta1_delta5")
model.addConstr(delta[2] <= delta[5],
name="Logica_delta2_delta5")

# Optimizar el modelo
model.optimize()

# Imprimir la solución
if model.status == GRB.OPTIMAL:
    print(f"\nValor óptimo de la función objetivo:
{model.ObjVal}")
    print("Valores de las variables de decisión:")
    for i in K:
        print(f"Aceite {i}: x_{i} = {x[i].X}, delta_{i} = {int(delta[i].X)},
delta_I_{i} = {int(delta_I[i].X)}, delta_II_{i} = {int(delta_II[i].X)}")
    print(f"Producción total (t): {t.X}")
else:
    print("No se encontró una solución óptima.")
```

Valor óptimo de la función objetivo: 13800.0
Valores de las variables de decisión:
Aceite 1: x_1 = 0.0, delta_1 = 0, delta_I_1 = 0, delta_II_1 = 0
Aceite 2: x_2 = 200.0, delta_2 = 1, delta_I_2 = 1, delta_II_2 = 0
Aceite 3: x_3 = -0.0, delta_3 = 0, delta_I_3 = 0, delta_II_3 = 0
Aceite 4: x_4 = 200.0, delta_4 = 1, delta_I_4 = 1, delta_II_4 = 0
Aceite 5: x_5 = 50.0, delta_5 = 1, delta_I_5 = 0, delta_II_5 = 1
Producción total (t): 450.0

Aceites

- Se definen dos nuevas variables binarias: δ_I^i y δ_{II}^i

```
delta_I = model.addVars(K, vtype=GRB.BINARY, name="delta_I")
delta_II = model.addVars(K, vtype=GRB.BINARY, name="delta_II")
```

- Se actualiza la función con los costes fijos de los depósitos

```
model.setObjective(
    150 * t - sum(c[i] * x[i] + f_I * delta_I[i] + f_II *
delta_II[i] for i in K),
    GRB.MAXIMIZE
)
```

- Restricciones de capacidad de los depósitos

```
for i in K:
    model.addConstr(x[i] <= C_I * delta_I[i] + C_II * delta_II[i], name=f"Capacidad_Deposito_{i}")
```

- Restricción de elección única de depósito por aceite:

```
for i in K:
    model.addConstr(delta_I[i] + delta_II[i] == delta[i], name=f"Deposito_Unico_{i}")
```

PCIngredients

- La empresa PCIngredients vende ordenadores y debe hacer una planificación semanal de la producción.
- La compañía produce tres tipos de ordenadores: de mesa (A), portátil normal (B) y portátil de lujo (C). El beneficio neto por la venta un ordenador es 350, 470 y 610 euros, respectivamente.
- Cada semana se venden todos los equipos que se montan.
- Los ordenadores pasan un control de calidad de una hora y la empresa dispone de 120 horas para realizar los controles de los ordenadores A y B y 48 para los C.
- El resto de las operaciones de montaje requieren 10, 15 y 20 horas, respectivamente, y la empresa dispone de 2000 horas a la semana.

PCIngredients

$$Maximizar\ Z = 350x + 470y + 610z$$
$$Sujeto\ a:$$
$$x + y \leq 120$$
$$z \leq 48$$
$$10x + 15y + 20z \leq 2000$$
$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$

Solución Óptima:
x (Ordenador A) = 120.00 unidades
y (Ordenador B) = 0.00 unidades
z (Ordenador C) = 40.00 unidades
Beneficio Máximo Total = 66400.00 euros

```python
from gurobipy import Model, GRB

# Crear un nuevo modelo
model = Model("PCIngredients_Produccion")

# Definir las variables de decisión
xA = model.addVar(name="xA", lb=0)  # Unidades de Ordenador A
xB = model.addVar(name="xB", lb=0)  # Unidades de Ordenador B
xC = model.addVar(name="xC", lb=0)  # Unidades de Ordenador C

# Establecer la función objetivo
model.setObjective(
    350 * xA + 470 * xB + 610 * xC,
    GRB.MAXIMIZE
)

# Agregar las restricciones

# Restricción 1: Control de Calidad para A y B
model.addConstr(
    xA + xB <= 120,
    name="Control_Calidad_AB"
)

# Restricción 2: Control de Calidad para C
model.addConstr(
    xC <= 48,
    name="Control_Calidad_C"
)

# Restricción 3: Tiempo de Montaje Total
model.addConstr(
    10 * xA + 15 * xB + 20 * xC <= 2000,
    name="Tiempo_Montaje"
)

# Optimizar el modelo
model.optimize()

# Verificar si se encontró una solución óptima
if model.status == GRB.OPTIMAL:
    print("\nSolución Óptima:")
    print(f"xA (Ordenador A) = {xA.X:.2f} unidades")
    print(f"xB (Ordenador B) = {xB.X:.2f} unidades")
    print(f"xC (Ordenador C) = {xC.X:.2f} unidades")
    print(f"Beneficio Máximo Total = {model.ObjVal:.2f} euros")
else:
    print("No se encontró una solución óptima.")
```

PCIngredients

- La empresa **PCIngredients** ha revisado sus costes de producción de ordenadores y ha llegado a las siguientes conclusiones:
  1. **Producción mínima**: No es rentable iniciar la producción de ordenadores si no se producen al menos **10 unidades**.
  2. **Producción en instalaciones propias**:
     1. **Capacidad máxima**: Las instalaciones actuales permiten producir hasta **80 unidades** sin problemas.
     2. **Coste fijo**: Incurre en un **coste fijo de 1000 euros** en concepto de mantenimiento de las instalaciones (independientemente de la cantidad producida en estas instalaciones).
  3. **Producción en instalaciones externas**:
     1. **Capacidad adicional**: Puede recurrir a instalaciones externas para incrementar su producción por encima de las **80 unidades**.
     2. **Coste adicional**: Incurre en un **coste adicional de 800 euros** en concepto de alquiler (independientemente de la cantidad producida en instalaciones externas).
- El objetivo es **actualizar el modelo de optimización** para incluir estas nuevas condiciones.

PCIngredients

1. Variables de producción (enteras):

- $x_i \geq 0$: Número total de ordenadores del tipo *i* a producir, donde *i* $\in$ {a,b,c}.

2. Variables binarias:

- $y_i \in \{0,1\}$: Indica si se produce ordenadores a.
  - $y_i = 1$ si $x_i \geq 10$.
  - $y_i = 0$ si $x_i = 0$.
- $y_{ext} \in \{0,1\}$: Indica si se utilizan instalaciones externas para ordenadores a.
  - $y_{ext} = 1$ si $x_a > 80$.
  - $y_{ext} = 0$ si $x_a \leq 80$.

3. Variables de producción en instalaciones:

- $x_i^{int} \geq 0$: Número de ordenadores *i* producidos en instalaciones internas.
- $x_i^{ext} \geq 0$: Número de ordenadores *i* producidos en instalaciones externas.

PCIngredients

$$Maximizar\ Z = \sum_{i \in \{a,b,c\}}^{N} (BiXi - CFiYi - CEiYexti)$$

$Sujeto\ a$:

$\forall_i \in \{a, b, c\}$:

$$x_i = x_i^{int} + x_i^{ext}$$

$$x_i \geq YiProduccion_{\min_i}$$

$$x_i \leq YiM$$

$(donde\ M\ es\ un\ número\ suficietemente\ grande)$

$$x_i^{int} \leq Y_i Capacidad_{inti}$$

$$x_i^{ext} \geq X_i - YiCapacidadint_i$$

$$x_i^{int} \leq MY_{exti}$$

$$x_i^{ext} \leq MY_{exti}$$

$$h_a x_a + hbxb + hcxc \leq Hmon_{taje}$$

$$c_a x_a + c_b x_b + hcxc \leq H_{calidad\ \_\ ab}$$

$$c_c x_c \leq H_{calidad\ \_\ c}$$

$$x_i, x_i^{int}, x_i^{ext} \in \mathbb{Z}, \forall_i \in \{a, b, c\}$$

$$y_i, yext_{\ \_\ i} \in \{0, 1\}, \forall_i \in \{a, b, c\}$$

PCIngredients

```python
# Importar la librería gurobipy
from gurobipy import Model, GRB

# Crear el modelo
model = Model("PCIngredients_Actualizado")

# Desactivar la salida de Gurobi (opcional)
model.Params.OutputFlag = 0

# Conjuntos
productos = ['a', 'b', 'c']

# Parámetros
beneficio = {'a': 350, 'b': 470, 'c': 610}
coste_fijo = {'a': 1000, 'b': 1000, 'c': 1000}
coste_ext = {'a': 800, 'b': 800, 'c': 800}
horas_montaje = {'a': 10, 'b': 15, 'c': 20}
horas_calidad = {'a': 1, 'b': 1, 'c': 1}
capacidad_int = {'a': 80, 'b': 80, 'c': 80}
produccion_min = {'a': 10, 'b': 10, 'c': 10}

# Horas disponibles
H_montaje = 2000
H_calidad_ab = 120
H_calidad_c = 48

# Número grande M
M = 10000

# Variables de decisión
x = model.addVars(productos, vtype=GRB.INTEGER, lb=0, name="x")
x_int = model.addVars(productos, vtype=GRB.INTEGER, lb=0, name="x_int")
x_ext = model.addVars(productos, vtype=GRB.INTEGER, lb=0, name="x_ext")
y = model.addVars(productos, vtype=GRB.BINARY, name="y")
y_ext = model.addVars(productos, vtype=GRB.BINARY, name="y_ext")

# Actualizar el modelo para integrar las variables
model.update()

# Función objetivo
model.setObjective(
    sum(beneficio[i] * x[i] - coste_fijo[i] * y[i] - coste_ext[i] * y_ext[i] for i in productos),
    GRB.MAXIMIZE
)

# Restricciones

# 1. Relación entre producción total y producción en instalaciones
for i in productos:
    model.addConstr(x[i] == x_int[i] + x_ext[i], name=f"Produccion_total_{i}")

# 2. Producción mínima
for i in productos:
    model.addConstr(x[i] >= produccion_min[i] * y[i], name=f"Produccion_minima_{i}")
    model.addConstr(x[i] <= M * y[i], name=f"Produccion_maxima_{i}")

# 3. Capacidad de instalaciones internas
for i in productos:
    model.addConstr(x_int[i] <= capacidad_int[i] * y[i], name=f"Capacidad_interna_{i}")

# 4. Uso de instalaciones externas
for i in productos:
    model.addConstr(x_ext[i] >= x[i] - capacidad_int[i] * y[i], name=f"Produccion_externa_min_{i}")
    model.addConstr(x_ext[i] <= M * y_ext[i], name=f"Produccion_externa_max_{i}")

# 5. Si no se produce el tipo de ordenador, no hay producción interna ni externa
for i in productos:
    model.addConstr(x_int[i] <= M * y[i], name=f"Produccion_interna_si_se_produce_{i}")
    model.addConstr(x_ext[i] <= M * y[i], name=f"Produccion_externa_si_se_produce_{i}")
```

PCIngredients

```
# 7. Horas de montaje
model.addConstr(sum(horas_montaje[i] * x[i] for i in productos) <= H_montaje, name="Horas_Montaje")

# Optimizar el modelo
model.optimize()

# Imprimir la solución
if model.status == GRB.OPTIMAL:
    print(f"\nBeneficio máximo total: {model.ObjVal} euros")
    print("Plan de producción óptimo:")
    for i in productos:
        print(f" - Ordenadores {i}: {x[i].X} unidades")
        print(f"   - Producción interna ({i}_int): {x_int[i].X} unidades")
        print(f"   - Producción externa ({i}_ext): {x_ext[i].X} unidades")
        print(f"   - Se produce ({i}): {int(y[i].X)}")
        print(f"   - Se usan instalaciones externas ({i}_ext): {int(y_ext[i].X)}")
else:
    print("No se encontró una solución óptima.")
```

Beneficio máximo total: 63600.0 euros
Plan de producción óptimo:
- Ordenadores a: 120.0 unidades
  - Producción interna (a_int): 80.0 unidades
  - Producción externa (a_ext): 40.0 unidades
  - Se produce (a): 1
  - Se usan instalaciones externas (a_ext): 1
- Ordenadores b: 0.0 unidades
  - Producción interna (b_int): 0.0 unidades
  - Producción externa (b_ext): -0.0 unidades
  - Se produce (b): 0
  - Se usan instalaciones externas (b_ext): 0
- Ordenadores c: 40.0 unidades
  - Producción interna (c_int): 40.0 unidades
  - Producción externa (c_ext): -0.0 unidades
  - Se produce (c): 1
  - Se usan instalaciones externas (c_ext): 0

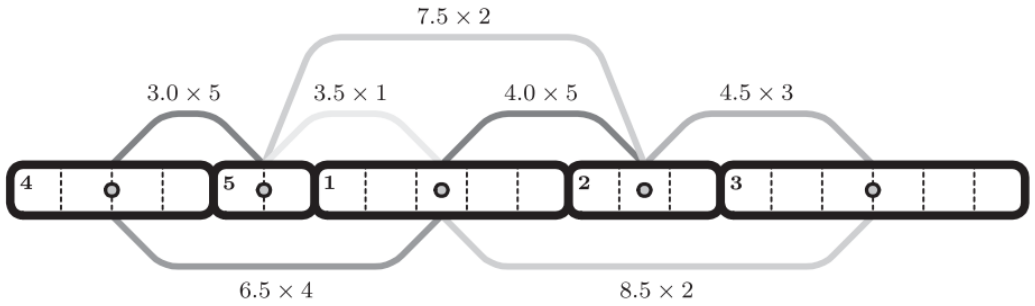
PCIngredients

```
# 7. Horas de montaje
model.addConstr(sum(horas_montaje[i] * x[i] for i in productos) <= H_montaje, name="Horas_Montaje")

# Optimizar el modelo
model.optimize()

# Imprimir la solución
if model.status == GRB.OPTIMAL:
    print(f"\nBeneficio máximo total: {model.ObjVal} euros")
    print("Plan de producción óptimo:")
    for i in productos:
        print(f" - Ordenadores {i}: {x[i].X} unidades")
        print(f"   - Producción interna ({i}_int): {x_int[i].X} unidades")
        print(f"   - Producción externa ({i}_ext): {x_ext[i].X} unidades")
        print(f"   - Se produce ({i}): {int(y[i].X)}")
        print(f"   - Se usan instalaciones externas ({i}_ext): {int(y_ext[i].X)}")
else:
    print("No se encontró una solución óptima.")
```

Beneficio máximo total: 63600.0 euros
Plan de producción óptimo:
- Ordenadores a: 120.0 unidades
  - Producción interna (a_int): 80.0 unidades
  - Producción externa (a_ext): 40.0 unidades
  - Se produce (a): 1
  - Se usan instalaciones externas (a_ext): 1
- Ordenadores b: 0.0 unidades
  - Producción interna (b_int): 0.0 unidades
  - Producción externa (b_ext): -0.0 unidades
  - Se produce (b): 0
  - Se usan instalaciones externas (b_ext): 0
- Ordenadores c: 40.0 unidades
  - Producción interna (c_int): 40.0 unidades
  - Producción externa (c_ext): -0.0 unidades
  - Se produce (c): 1
  - Se usan instalaciones externas (c_ext): 0

Single Row Facility Layout Problem

- El problema de diseño de instalaciones en una fila (SRFLP, por sus siglas en inglés) es un problema clásico en la optimización combinatoria y la investigación de operaciones.
- Consiste en determinar el orden óptimo de una serie de instalaciones (o departamentos) a lo largo de una línea (fila), minimizando el coste total asociado a las distancias entre las instalaciones.
- Aplicaciones:
  - Diseño de líneas de producción en fábricas.
  - Diseño de almacenes y centros de distribución.
  - Diseño de circuitos integrados y VLSI.
  - Diseño de instalaciones de transporte:
  - Diseño de plantas químicas y petroquímicas:
  - Diseño de data centers y servidores:



Fuente: https://www.sciencedirect.com/science/article/pii/S0950705116301204

https://www.tandfonline.com/doi/full/10.1080/00207543.2021.1897176

Single Row Facility Layout Problem

Descripción del Problema

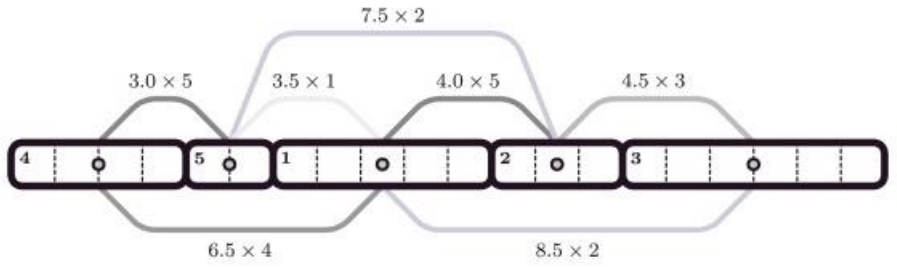
- **Objetivo**: Determinar la disposición lineal de **N** instalaciones en una fila, minimizando el coste total.
- **Datos del problema**:
  - **Instalaciones**: Hay **N** instalaciones, cada una con una longitud determinada.
  - **Flujos**: Hay un flujo o interacción entre cada par de instalaciones.
  - **Coste**: El coste entre dos instalaciones es proporcional al flujo entre ellas multiplicado por la distancia que las separa.

Facility lengths:

i	1	2	3	4	5
l_i	5	3	6	4	2

Weights between facilities:

c_{ij}	1	2	3	4	5
1	0	5	2	4	1
2	5	0	3	0	2
3	2	3	0	0	0
4	4	0	0	0	5
5	1	2	0	5	0



Solution: $\pi = \langle 4, 5, 1, 2, 3 \rangle$
Cost: $C(\pi) = (3.0 \times 5) + (6.5 \times 4) + (3.5 \times 1) + (7.5 \times 2) + (4.0 \times 5) + (8.5 \times 2) + (4.5 \times 3) = 110$

Fuente: https://www.sciencedirect.com/science/article/pii/S0950705116301204

Single Row Facility Layout Problem

Formulación Matemática

- **Variables de decisión**
  - **Variables de posicionamiento**:
    - x_i: Posición de inicio de la instalación *i* en la fila.
- **Variables binarias de orden**:
  - $y_{ij} \in \{0,1\}$: Variable que indica si la instalación *i* está a la izquierda de la instalación *j*.
  - $y_{ij}=1$ si *i* está a la izquierda de *j*.
- **Parámetros**
  - l_i: Longitud de la instalación $_i$.
  - w_{ij}: Flujo (o peso) entre las instalaciones *i* y *j*.

Single Row Facility Layout Problem

Función objetivo:

$$Minimizar\ Z = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} d_{ij}$$

$$donde\ dij = \left| x_i + \frac{l_i}{2} - xj - \frac{l_j}{2} \right|$$

$Sujeto\ a$:

$$x_i + li\ \leq xj + M(1 - yij), \forall_i\ \neq j$$

$$x_j + lj\ \leq xi + Myij, \forall_i\ \neq j$$

$$(Donde\ M\ es\ un\ número\ suficietemente\ grande)$$

$$\mathrm{y}_{ij} + yji = 1, \forall_i \neq j$$

$$y_{ij} \in \{0, 1\}, \forall_i \neq j$$

$$x_i \geq 0, \forall_i$$

Single Row Facility Layout Problem

Función objetivo:

$$Minimizar\ Z = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} d_{ij}$$

$$donde\ dij = \left| x_i + \frac{l_i}{2} - xj - \frac{l_j}{2} \right|$$

$Sujeto\ a$:

$$x_i + li \leq xj + M(1 - yij), \forall_i \neq j$$

$$x_j + lj \leq xi + Myi_j, \forall_i \neq j$$

$$(donde\ M\ es\ un\ número\ suficientemente\ grande)$$

$$y_{ij} + yj_i = 1, \forall_i \neq j$$

$$x_i \geq 0, \forall_i$$

Single Row Facility Layout Problem

```python
# Importar la librería gurobipy
from gurobipy import Model, GRB, quicksum


# Función para leer los datos desde un archivo
def leer_datos_desde_archivo(nombre_archivo):
    # Abrir el archivo y leer las líneas
    with open(nombre_archivo, 'r') as f:
        lines = f.readlines()

    # Remover saltos de línea y espacios en blanco
    lines = [line.strip() for line in lines if line.strip() != '']

    # Línea 0: Número de instalaciones
    n = int(lines[0])

    # Línea 1: Anchos de las instalaciones
    l = list(map(float, lines[1].split()))

    # Verificar que el número de anchos coincide con n
    if len(l) != n:
        raise ValueError("El número de anchos proporcionados no coincide
con el número de instalaciones.")
```

```python
# Matriz de costes: líneas desde la tercera hasta el final
    w = []
    for line in lines[2:]:
        row = list(map(float, line.split()))
        w.append(row)

    # Convertir la matriz de costes a una matriz completa
    # Asumiendo que la matriz proporcionada es triangular inferior
    W = [[0.0 for _ in range(n)] for _ in range(n)]
    for i in range(len(w)):
        for j in range(len(w[i])):
            W[i][j] = w[i][j]
            W[j][i] = W[i][j]  # Matriz simétrica
    return n, l, W
```

Single Row Facility Layout Problem

```
def main():
    # Nombre del archivo con los datos
    nombre_archivo = 'S9.txt'
    # nombre_archivo = 'Am12a.txt'
    # nombre_archivo = 'Am15.txt'
    # nombre_archivo = 'H20.txt'

    # Leer los datos desde el archivo
    n, l, w = leer_datos_desde_archivo(nombre_archivo)
    I = range(n)

    # Número grande M
    M = sum(l) * 2  # Un valor suficientemente grande

    # Crear el modelo
    model = Model("SRFLP")
    model.Params.OutputFlag = 0

    # Variables de decisión
    x = model.addVars(n, lb=0, vtype=GRB.CONTINUOUS, name="x")
    y = model.addVars(n, n, vtype=GRB.BINARY, name="y")

    # Variables auxiliares para las distancias
    d = model.addVars(n, n, lb=0, vtype=GRB.CONTINUOUS, name="d")

    # Función objetivo
    model.setObjective(
        quicksum(
            w[i][j] * d[i, j]
            for i in I for j in I if i < j
        ),
        GRB.MINIMIZE
    )
```

```
# Restricciones
    # 1. No superposición de instalaciones
    for i in I:
        for j in I:
            if i != j:
                # Si y_ij = 1, entonces x_i + l_i <= x_j
                model.addConstr(
                    x[i] + l[i] <= x[j] + M * (1 - y[i, j]),
                    name=f"NoOverlap_{i}_{j}"
                )
                # Si y_ij = 0, entonces x_j + l_j <= x_i
                model.addConstr(
                    x[j] + l[j] <= x[i] + M * y[i, j],
                    name=f"NoOverlap_{j}_{i}"
                )
                # Relación entre y_ij y y_ji
                model.addConstr(
                    y[i, j] + y[j, i] == 1,
                    name=f"Order_{i}_{j}"
                )
    # 2. Cálculo de distancias
    for i in I:
        for j in I:
            if i < j:
                # d_ij >= (x_j + l_j/2) - (x_i + l_i/2)
                model.addConstr(
                    d[i, j] >= (x[j] + l[j] / 2) - (x[i] + l[i] / 2),
                    name=f"Dist1_{i}_{j}"
                )
                # d_ij >= (x_i + l_i/2) - (x_j + l_j/2)
                model.addConstr(
                    d[i, j] >= (x[i] + l[i] / 2) - (x[j] + l[j] / 2),
                    name=f"Dist2_{i}_{j}"
                )
```

Single Row Facility Layout Problem

```
# Optimizar el modelo
    model.optimize()

    # Imprimir la solución
    if model.status == GRB.OPTIMAL:
        print("\nPosiciones óptimas de las instalaciones:")
        for i in I:
            inicio = x[i].X
            fin = x[i].X + l[i]
            print(f"Instalación {i + 1}: Inicio en {inicio:.2f}, Fin en {fin:.2f}, Longitud: {l[i]}")
        print("\nOrden de las instalaciones:")
        sorted_facilities = sorted(I, key=lambda i: x[i].X)
        for idx, i in enumerate(sorted_facilities):
            print(f"Posición {idx + 1}: Instalación {i + 1}")
    else:
        print("No se encontró una solución óptima.")


if __name__ == "__main__":
    main()
```

Posiciones óptimas de las instalaciones:
Instalación 1: Inicio en 10.00, Fin en 12.00, Longitud: 2.0
Instalación 2: Inicio en 48.00, Fin en 56.00, Longitud: 8.0
Instalación 3: Inicio en 39.00, Fin en 48.00, Longitud: 9.0
Instalación 4: Inicio en 0.00, Fin en 7.00, Longitud: 7.0
Instalación 5: Inicio en 7.00, Fin en 10.00, Longitud: 3.0
Instalación 6: Inicio en 20.00, Fin en 24.00, Longitud: 4.0
Instalación 7: Inicio en 24.00, Fin en 30.00, Longitud: 6.0
Instalación 8: Inicio en 12.00, Fin en 20.00, Longitud: 8.0
Instalación 9: Inicio en 30.00, Fin en 39.00, Longitud: 9.0

Orden de las instalaciones:
Posición 1: Instalación 4
Posición 2: Instalación 5
Posición 3: Instalación 1
Posición 4: Instalación 8
Posición 5: Instalación 6
Posición 6: Instalación 7
Posición 7: Instalación 9
Posición 8: Instalación 3
Posición 9: Instalación 2

Algoritmos de resolución

Métodos de planos de corte

- El **método de planos de corte** es una técnica iterativa utilizada para resolver problemas de programación entera.

- Se basa en resolver una serie de problemas de programación lineal (PL) y agregar gradualmente restricciones adicionales, llamadas **planos de corte**, para eliminar soluciones no enteras del espacio factible sin eliminar ninguna solución entera factible.

Algoritmos de resolución

Funcionamiento General

1. Resolución del relajado lineal:

1. Se comienza resolviendo la **relajación lineal** del problema, es decir, el problema original sin la restricción de que las variables sean enteras.
2. Esta solución proporcionará un límite inferior (para problemas de minimización) o un límite superior (para problemas de maximización) para el problema entero.

2. Verificación de entereidad:

1. Si la solución obtenida es entera, entonces es la solución óptima del problema entero.
2. Si no es entera, se procede al siguiente paso.

Algoritmos de resolución

Funcionamiento General

1. Generación de planos de corte:

1. Se generan cortes, que son restricciones lineales adicionales que eliminan la solución fraccionaria actual y potencialmente otras soluciones fraccionarias, pero que no eliminan ninguna solución entera factible.
2. Estos cortes se basan en propiedades matemáticas del problema, como la estructura de las restricciones y variables.

2. Actualización del modelo:

1. Se agrega el plano de corte al modelo y se resuelve nuevamente el problema lineal actualizado.
2. Este proceso se repite hasta que se encuentra una solución entera o se determina que el problema es infactible.

Algoritmos de resolución

- **Cortes de Gomory:**
  - Son cortes generales derivados de la solución básica actual.
  - Se basan en la representación de la solución fraccionaria y generan una desigualdad que excluye esta solución.
- **Cortes de Plano de Chvátal-Gomory:**
  - Se derivan combinando las restricciones originales del problema y redondeando los coeficientes para obtener una nueva restricción válida.
- **Cortes de Plano de Cobertura:**
  - Utilizados en problemas de mochila y similares, se basan en identificar subconjuntos de variables que no pueden tomar ciertos valores simultáneamente.
- **Cortes de Plano de Florencia y Padberg:**
  - Aplicados en problemas de rutas y particionamiento, se centran en eliminar soluciones no factibles relacionadas con subciclos o particiones inválidas.

Algoritmos de resolución

Métodos de ramificación y acotación (Branch and Bound)

- El **método de ramificación y acotación** es un algoritmo de enumeración implícita que explora sistemáticamente todas las posibles soluciones enteras de un problema.

- Utiliza límites para descartar grandes subconjuntos del espacio de búsqueda donde no puede existir una solución óptima, reduciendo así el esfuerzo computacional.

Algoritmos de resolución

Funcionamiento general

- Resolución del relajado lineal:
  - Al igual que en el método de planos de corte, se comienza resolviendo la relajación lineal del problema.
- Verificación de entereidad:
  - Si la solución es entera, se ha encontrado la solución óptima. Si no es entera, se procede a ramificar.
- Ramificación:
  - Se selecciona una variable fraccionaria en la solución actual.
  - Se crean dos nuevos subproblemas (nodos hijos) añadiendo restricciones que obligan a esa variable a tomar valores enteros:
    - En un subproblema, se agrega la restricción de que la variable debe ser menor o igual a la parte entera inferior.
    - En el otro, se agrega la restricción de que la variable debe ser mayor o igual a la parte entera superior.
- Acotación:
  - Se resuelven las relajaciones lineales de los subproblemas.
  - Se utilizan los valores de las funciones objetivo para establecer límites superiores e inferiores.

Algoritmos de resolución

Funcionamiento general

- Poda (Pruning):
  - Un subárbol puede ser podado (es decir, no se explora más) en los siguientes casos:
    - Infactibilidad: Si el subproblema es infactible.
    - Acotación (Bound): Si el valor óptimo del subproblema es peor que el mejor valor entero encontrado hasta el momento.
    - Optimalidad: Si la solución es entera y mejora la mejor solución encontrada, se actualiza la solución óptima actual.
- Iteración:
  - Se repite el proceso de ramificación y acotación en los subproblemas que no han sido podados hasta que se explora todo el espacio de búsqueda o se cumple un criterio de parada.

Estrategias de ramificación y selección de nodos

- Estrategias de selección de variables:
  - Variable más fraccionaria, orden natural y criterios avanzados (basados en estimaciones de impacto en el objetivo=.
- Estrategia de selección de nodos:
  - Profundidad primero, anchura primero, mejor límite

Algoritmos de resolución

Comparación entre los algoritmos de resolución

- Métodos de planos de corte:
  - Buscan mejorar la relajación lineal añadiendo restricciones.
  - Pueden ser más eficientes en problemas donde los cortes son efectivos y reducen significativamente el espacio factible.
- Métodos de ramificación y acotación:
  - Exploran el espacio de soluciones posibles mediante un árbol de decisiones.
  - Garantizan encontrar la solución óptima pero pueden ser más lentos si el espacio de búsqueda es muy grande.
- Combinación de métodos:
  - En la práctica, muchos solucionadores modernos (como Gurobi, CPLEX) combinan ambos métodos en algoritmos conocidos como Branch and Cut, que integran la generación de planos de corte dentro del marco de ramificación y acotación para mejorar la eficiencia.

Bibliografía

- Hillier & Liebermann (2010). Introducción a la investigación de operaciones, 9ªed. Capítulos 1 y 2.
- González y García (2015). Manual práctico de investigación de operaciones 4ª ed. Capítulo 1.
- Investigación Operativa (2004). Modelos determinísticos y estocásticos. Sixto Ríos Insua, Alfonso Mateosos Caballero, Mª Concepción Bielza Lozoya y Antonio Jiménez Martín, Editorial Centro de Estudios Ramón Areces, S.A., Madrid.

Bibliografía

- Investigación Operativa. Optimización, Sixto Ríos Insua. Editorial Centro de Estudios Ramón Areces, S.A., Madrid.
- Problemas de Investigación Operativa. Programación Lineal y Extensiones. Sixto Ríos Insua, David Ríos Insua, Alfonso Mateos Caballero, Jacinto Martín Jiménez y Antonio Jiménez Martín. Editorial Ra-Ma, Madrid 2006.
- Model Building in Mathematical Programming. Paul W. Williams. Editorial JOHN WILEY AND SONS
- Apuntes de Victoria Ruiz y Antonio Alonso
- Apuntes de Alberto Herrán y José J. Ruz Ortiz
- Apuntes de Gerardo Reyes y Salvador Sánchez

Grado en Ingeniería del Software

Investigación Operativa

BLOQUE II. MODELOS DETERMINISTAS

Tema 3: Optimización Lineal Entera y Combinatoria

URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Grado en Ingeniería del Software

Investigación Operativa

BLOQUE III. MODELOS ESTOCÁSTICOS

Tema 4: Fenómenos de espera

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Índice

Introducción

Conceptos básicos

- La **teoría de colas** emerge como una herramienta vital en la investigación operativa para analizar y mejorar sistemas donde los recursos son limitados y la demanda es variable.
- Se centra en proporcionar una visión detallada y profunda de los conceptos y modelos que permiten entender y gestionar las líneas de espera en diversos contextos.
- Los **fenómenos de espera** no solo afectan la satisfacción del cliente, sino que también influyen en la productividad y rentabilidad de las organizaciones.

Introducción

Objetivos

- Comprender los fundamentos de los fenómenos de espera y cómo se aplican en diferentes sistemas.
- Analizar medidas de eficiencia que permitan evaluar y mejorar el desempeño de sistemas de colas.
- Estudiar modelos poissonianos, fundamentales en la modelización de llegadas y servicios en sistemas de colas.
- Explorar redes de colas, entendiendo la complejidad añadida de sistemas interconectados y cómo se gestionan.

Introducción

Importancia

- Optimizar recursos y minimizar costes operativos.
- Mejorar la experiencia del cliente al reducir tiempos de espera.
- Toma de decisiones informadas basadas en modelos matemáticos sólidos.
- Anticipar y gestionar la congestión en sistemas complejos.

Introducción

Naturaleza de los fenómenos de espera

- Los fenómenos de espera surgen cuando existe un desequilibrio temporal entre la demanda de un servicio y la capacidad para proporcionarlo.
- Los desequilibrios pueden deberse a:
  - Variabilidad en la llegada de clientes: Los clientes o tareas llegan al sistema de manera aleatoria, generando picos y valles en la demanda.
  - Variabilidad en el tiempo de servicio: El tiempo que toma atender a cada cliente puede variar significativamente.
  - Capacidad limitada de recursos: Los recursos (personas, máquinas, líneas de comunicación) son finitos y no pueden atender instantáneamente toda la demanda.

Introducción

Ejemplos

- Supermercados: Los clientes hacen fila en las cajas registradoras cuando la afluencia supera la capacidad de los cajeros.
- Transporte público: Pasajeros esperando autobuses o trenes durante horas pico.
- Restaurantes: Comensales esperando por una mesa o por la preparación de sus pedidos.
- Tráfico vehicular: Atascos en carreteras y semáforos debido al volumen de vehículos.
- Sistemas informáticos: Procesos en espera para acceder a recursos compartidos como CPU, memoria u otros dispositivos.

Introducción

Historia

- Agner Krarup Erlang (1878-1929): Matemático danés considerado el pionero de la teoría de colas. En 1909, Erlang publicó sus primeros trabajos mientras trabajaba en la compañía telefónica de Copenhague, donde buscaba optimizar el número de líneas telefónicas y operadores necesarios para manejar las llamadas entrantes.
- Publicación de "The Theory of Probabilities and Telephone Conversations" (1917): Erlang desarrolló fórmulas para calcular la probabilidad de pérdida de llamadas y tiempos de espera, sentando las bases de la teoría de colas.

Introducción

Historia

- Décadas de 1920-1940: Se expandieron los modelos de Erlang para incluir diferentes distribuciones de llegadas y tiempos de servicio. Se aplicaron métodos de procesos estocásticos y teoría de probabilidades.
- Segunda Guerra Mundial: La teoría de colas se utilizó en logística militar y gestión de recursos, impulsando su desarrollo.
- Años 50 y 60: Se formalizaron modelos matemáticos más complejos, como los procesos de Markov y cadenas de Markov, aplicados a sistemas de colas.

Introducción

Aplicaciones

- **Telecomunicaciones**: Diseño y gestión de redes de comunicación, dimensionamiento de ancho de banda y routers.
- **Tecnología de la información**: Gestión de recursos en centros de datos, balanceo de carga en servidores y optimización de SSOO.
- **Manufactura y producción**: Optimización de líneas de ensamblaje, gestión de inventarios y planificación de producción.
- **Sector servicios**: Mejora en la atención al cliente, optimización de recursos humanos y reducción de tiempos de espera.
- **Transporte y logística**: Gestión del tráfico aéreo, planificación de rutas y programación de horarios.

Sistemas de colas

Proceso de llegadas

Describe cómo los clientes o unidades de demanda llegan al sistema.

- **Tasa de llegada (λ, lambda)**: Número promedio de llegadas por unidad de tiempo.
- **Distribución de llegadas**: Puede ser determinística o estocástica. Se modela con una distribución de Poisson para llegadas aleatorias.
- **Patrones de llegada**:
  - **Llegadas individuales**: Los clientes llegan uno por uno.
  - **Llegadas en lotes**: Los clientes llegan en grupos o paquetes.
- **Fuente de clientes**:
  - **Infinita**: Las llegadas no afectan la tasa de llegada futura.
  - **Finita**: Cada llegada reduce temporalmente la población fuente.

Sistemas de colas

Proceso de servicio

Describe la forma en que los clientes son atendidos en el sistema.

- **Tasa de servicio (μ, mu)**: Número promedio de clientes que pueden ser atendidos por unidad de tiempo.
- **Distribución de tiempos de servicio**: Puede ser constante o seguir una distribución probabilística, como la exponencial.
- **Número de servidores**:
  - **Sistema de servidor único**: Un solo servidor atiende a los clientes.
  - **Sistema multiservidor**: Varios servidores operan en paralelo.
- **Estructura de servicio**:
  - **En paralelo**: Los clientes pueden ser atendidos por cualquier servidor disponible.
  - **En serie**: Los clientes deben pasar por una secuencia de servidores en orden específico.

Sistemas de colas

Disciplina de la cola

Regla que determina el orden de la cola.

- **FIFO**: El primer cliente en llegar es el primero en ser atendido. Es la disciplina más común.
- **LIFO**: El último cliente en llegar es el primero en ser atendido. Ejemplo: Pila de documentos en un escritorio.
- **SIRO**: Los clientes son seleccionados al azar para el servicio.
- **Prioridad**:
  - **Preemptiva**: Un cliente de mayor prioridad puede interrumpir el servicio de uno de menor prioridad.
  - **No preemptiva**: Los clientes de mayor prioridad se atienden primero, pero no interrumpen servicios en curso.

Sistemas de colas

Capacidad y población del sistema

- **Capacidad del sistema**:
  - **Capacidad Infinita**: No hay límite en el número de clientes que pueden esperar en cola.
  - **Capacidad Finita (K)**: Existe un límite máximo de clientes en el sistema. Si se alcanza este límite, los nuevos clientes son rechazados o bloqueados.
- **Bloqueo o pérdida de clientes**:
  - En sistemas con capacidad limitada, los clientes que llegan cuando el sistema está lleno pueden ser perdidos, lo que afecta el rendimiento y la satisfacción.
- **Población fuente**:
  - **Población infinita**: El número de posibles clientes es tan grande que la tasa de llegada no se ve afectada por el número de clientes en el sistema.
  - **Población finita**: El número total de clientes es limitado, y la tasa de llegada puede depender del número de clientes en el sistema.

Medidas de eficiencia

Conceptos básicos

- Las **medidas de eficiencia** son métricas cuantitativas que describen el comportamiento y desempeño de un sistema de colas. Estas medidas evalúan aspectos como los siguientes:
- Tiempo que los clientes pasan en cola y en el sistema.
- Número promedio de clientes en cola y en el sistema.
- Grado de utilización de los recursos del sistema.

Medidas de eficiencia

Relevancia

- **Permiten identificar cuellos de botella** y áreas donde el sistema puede mejorar.
- **Ayudan a equilibrar recursos** para satisfacer la demanda de manera eficiente.
- **Contribuyen a mejorar la satisfacción del cliente**, al reducir tiempos de espera.
- **Facilitan la toma de decisiones estratégicas y operativas**, como la asignación de personal o la inversión en infraestructura.
- **Proporcionan una base para comparar diferentes configuraciones** y seleccionar la más adecuada.

Medidas de eficiencia

Tiempo promedio en cola (Wq)

- El tiempo promedio en cola (Wq) es el tiempo promedio que un cliente pasa esperando en la cola antes de ser atendido.
- Wq **refleja la eficiencia del sistema** en la gestión de la espera.
- Un Wq alto indica que los clientes esperan mucho tiempo, lo cual puede disminuir la satisfacción y afectar la productividad.
- Un Wq bajo es deseable, pero puede implicar costes más altos si se requieren más recursos para lograrlo.

Medidas de eficiencia

Número promedio de clientes en cola (Lq)

- El número promedio de clientes en cola (Lq) es el número promedio de clientes que están esperando en la cola en un momento dado.
- Lq proporciona una medida de la saturación en la cola.
- Un Lq alto puede indicar necesidad de aumentar la capacidad de servicio.
- Un Lq bajo sugiere que el sistema está manejando eficientemente la demanda.

Medidas de eficiencia

Tiempo promedio en el sistema (W)

- El tiempo promedio en el sistema (W) es el tiempo total promedio que un cliente pasa en el sistema, incluyendo tanto el tiempo en cola (Wq) como el tiempo de servicio (Ws).

$$W = Wq + \frac{1}{\mu}$$

Donde $\frac{1}{\mu}$ es el tiempo promedio de servicio (Ws).

- W refleja la experiencia completa del cliente en el sistema.
- Un W alto puede afectar negativamente la satisfacción del cliente.
- Es crucial equilibrar Wq y Ws para optimizar W.

Medidas de eficiencia

Número promedio de clientes en el sistema (L)

- El número promedio de clientes en el sistema (L) es el número promedio de clientes presentes en el sistema, incluyendo tanto los que están en cola como los que están siendo atendidos.

$$L = Lq + \rho L$$

Donde ρ es la utilización del servidor.

- L indica el nivel general de actividad en el sistema.
- Un L alto puede señalar saturación y potenciales demoras.
- Ayuda en la planificación de recursos y espacio físico necesario.

Medidas de eficiencia

Utilización del servidor (ρ)

- La utilización del servidor (ρ) es la proporción del tiempo que el servidor está ocupado atendiendo clientes. Se calcula como:

$$\rho = \frac{\lambda}{\mu}$$

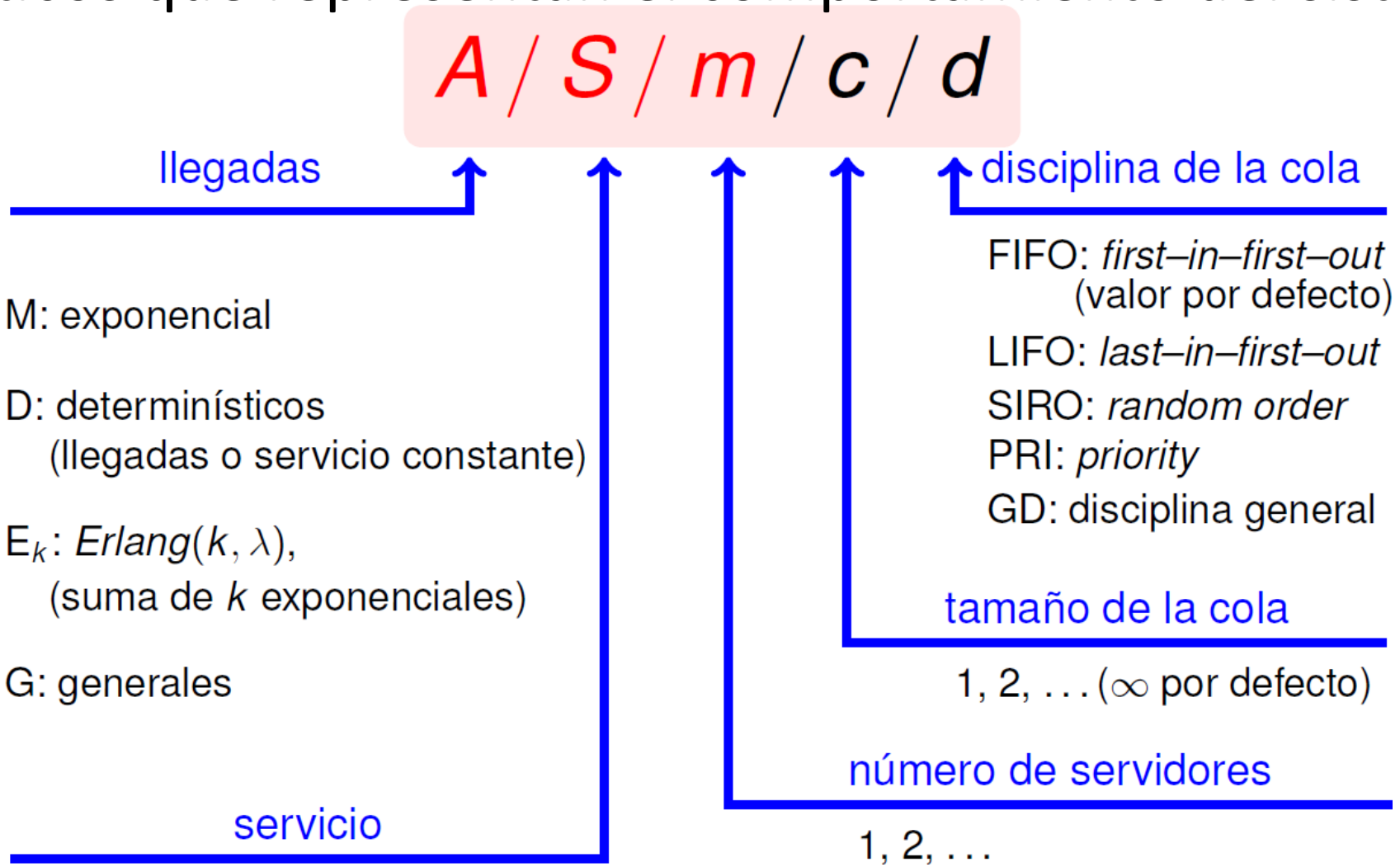
Donde:

  - λ es la tasa promedio de llegadas.
  - μ es la tasa promedio de servicio.

- ρ refleja la eficiencia en el uso del recurso de servicio.
- Un ρ cercano a 1 indica que el servidor está casi siempre ocupado.
- Un ρ muy bajo puede implicar subutilización y costes innecesarios

Modelos

- Las medidas de eficiencia se calculan utilizando modelos matemáticos que representan el comportamiento del sistema de colas.



Modelos

Modelo M/M/1

- Llegadas: Proceso de Poisson con tasa λ
- Tiempos de servicio: Distribución exponencial con tasa μ
- Un solo servidor.
- Capacidad infinita y población fuente infinita.
- Disciplina FIFO.
- Para que el sistema sea estable, la tasa de llegada debe ser menor que la tasa de servicio (condición de estabilidad):

$$\rho = \frac{\lambda}{\mu} < 1$$

Modelos

Modelo M/M/1

- Utilización del servidor (ρ):

$$\rho = \frac{\lambda}{\mu}$$

- Número promedio de clientes en cola (Lq):

$$Lq = \frac{\rho^2}{1-\rho}$$

- Número promedio de clientes en el sistema (L):

$$L = \frac{\rho}{1-\rho}$$

Modelos

Modelo M/M/1

- Tiempo promedio en cola (Wq):

$$Wq = \frac{p}{\mu(1-\mathrm{p})} = \frac{Lq}{\lambda}$$

- Tiempo promedio en el sistema (W):

$$W = \frac{1}{\mu(1-\mathrm{p})} = \frac{L}{\lambda}$$

Modelos

Modelo M/M/1

- λ=5 clientes/hora
- μ=8 clientes/hora
- Utilización del servidor (ρ):

$$\rho = \frac{5}{8} = 0.625$$

- Número promedio de clientes en cola (Lq):

$$Lq = \frac{(0.625)^2}{1-0.625} = \frac{0.390625}{0.375} \approx 1.0417$$

- Número promedio de clientes en el sistema (L):

$$L = \frac{0.625}{1-0.625} = \frac{0.625}{0.375} \approx 1.6667$$

Modelos

Modelo M/M/1

- Tiempo promedio en cola (Wq):

$$Wq = \frac{1.0417}{5} = 0.2083 \approx 12.5\ min$$

- Tiempo promedio en el sistema (W):

$$W = \frac{1.6667}{5} = 0.3333\ h\ \approx 20 min$$

¿Interpretación de las métricas?

Modelos

```python
def mm1(rho, lam):
    # Cálculos
    Lq = rho**2 / (1 - rho)
    L = rho / (1 - rho)
    Wq = Lq / lam
    W = L / lam

    # Resultados
    results = {
        'Utilización del servidor (rho)': rho,
        'Número promedio de clientes en cola (Lq)': Lq,
        'Número promedio de clientes en el sistema (L)': L,
        'Tiempo promedio en cola (Wq)': Wq,
        'Tiempo promedio en el sistema (W)': W
    }
    return results

# Datos de entrada
lam = 5  # lambda
mu = 8   # mu

# Cálculo de rho
rho = lam / mu

# Llamar a la función mm1
resultado_mm1 = mm1(rho, lam)

# Imprimir resultados
print("Resultados para el modelo M/M/1:")
for key, value in resultado_mm1.items():
    if 'Tiempo' in key:
        print(f"{key}: {value:.4f} horas ({value*60:.2f} minutos)")
    else:
        print(f"{key}: {value:.4f}")
```

Resultados para el modelo M/M/1:
Utilización del servidor (rho): 0.6250
Número promedio de clientes en cola (Lq): 1.0417
Número promedio de clientes en el sistema (L): 1.6667
Tiempo promedio en cola (Wq): 0.2083 horas (12.50 minutos)
Tiempo promedio en el sistema (W): 0.3333 horas (20.00 minutos)

Modelos

Modelo M/M/c

- Llegadas: Proceso de Poisson con tasa λ.
- Tiempos de servicio: Distribución exponencial con tasa μ.
- c servidores en paralelo.
- Capacidad infinita y población fuente infinita.
- Disciplina FIFO.
- Para que el sistema sea estable, la tasa de llegada debe ser menor que la tasa de servicio (condición de estabilidad):

$$\rho = \frac{\lambda}{c\mu} < 1$$

Modelos

Modelo M/M/c

- Probabilidad de que todos los servidores estén ocupados (P0):

$$P0 = \frac{1}{\sum_{n=0}^{c-1} \frac{(\lambda/\mu)^n}{n!} + \frac{(\lambda/\mu)^c}{c!} * \frac{c\mu}{c\mu - \lambda}}$$

- Número promedio de clientes en cola (Lq):

$$Lq = \frac{\lambda\mu(\lambda/\mu)^c}{c!(c\mu - \lambda)^2} P0$$

- Número promedio de clientes en el sistema (L):

$$L = Lq + \frac{\lambda}{\mu}$$

Modelos

Modelo M/M/c

- Tiempo promedio en cola (Wq):

$$Wq = \frac{Lq}{\lambda}$$

- Tiempo promedio en el sistema (W):

$$W = Wq + Ws$$

Hacer para:

- λ=10 clientes/hora.
- μ=6 clientes/hora.
- c=2 servidores.

Modelos

```python
import math
def mmc(lam, mu, c):
    rho = lam / (c * mu)

    # Verificar condición de estabilidad
    if rho >= 1:
        return "El sistema es inestable (rho >= 1)."

    # Calcular P0
    sumatoria = sum((lam / mu) ** n / math.factorial(n) for n in range(c))
    pnumerator = (lam / mu) ** c / math.factorial(c)
    pdenominator = 1 / (1 - rho)
    P0 = 1 / (sumatoria + pnumerator * pdenominator)

    # Calcular Lq
    Lq = (pnumerator * rho) / ((1 - rho) ** 2) * P0
    L = Lq + (lam / mu)
    Wq = Lq / lam
    W = Wq + (1 / mu)

    # Resultados
    results = {
        'Utilización del sistema (rho)': rho,
        'Probabilidad de que todos los servidores estén Ocupados (P0)': P0,
        'Número promedio de clientes en cola (Lq)': Lq,
        'Número promedio de clientes en el sistema (L)': L,
        'Tiempo promedio en cola (Wq)': Wq,
        'Tiempo promedio en el sistema (W)': W
    }
    return results
```

```python
# Datos de entrada
lam = 10  # lambda
mu = 6    # mu
c = 2     # número de servidores

# Llamar a la función mmc
resultado_mmc = mmc(lam, mu, c)

# Verificar si el sistema es estable
if isinstance(resultado_mmc, str):
    print(resultado_mmc)
else:
    # Imprimir resultados
    print("\nResultados para el Modelo M/M/c:")
    for key, value in resultado_mmc.items():
        if 'Tiempo' in key:
            print(f"{key}: {value:.4f} horas ({value*60:.2f} minutos)")
        else:
            print(f"{key}: {value:.4f}")
```

Resultados para el Modelo M/M/c:
Utilización del sistema (rho): 0.8333
Probabilidad de que todos los servidores estén Ocupados (P0): 0.0909
Número promedio de clientes en cola (Lq): 3.7879
Número promedio de clientes en el sistema (L): 5.4545
Tiempo promedio en cola (Wq): 0.3788 horas (22.73 minutos)
Tiempo promedio en el sistema (W): 0.5455 horas (32.73 minutos)

Fórmula de Little

Características

- Establece una relación directa y sencilla entre tres variables clave en cualquier sistema de colas en estado estacionario:
  - El número promedio de clientes en el sistema.
  - La tasa promedio de llegada de clientes al sistema.
  - El tiempo promedio que un cliente pasa en el sistema.

Formulación de la relación de Little

$$L=\lambda W$$

- Donde:
  - L: Número promedio de clientes en el sistema.
  - λ: Tasa promedio de llegada al sistema (clientes por unidad de tiempo).
  - W: Tiempo promedio que un cliente pasa en el sistema.
- Se puede aplicar específicamente a colas.

Fórmula de Little

Interpretación

- El número promedio de clientes en el sistema es igual a la tasa de llegada multiplicada por el tiempo promedio en el sistema.
- Por ejemplo, si a un restaurante llegan en promedio 30 clientes por hora (λ=30 clientes/hora) y cada cliente pasa en promedio 0.5 horas en el restaurante (W=0.5, entonces el número promedio de clientes en el restaurante es:

$$L=\lambda W= 30 * 0.5 = 15$$

Condiciones de aplicabilidad

- La Ley de Little es válida bajo las siguientes condiciones:
  - Sistema estable: El sistema debe estar en estado estacionario, es decir, las medidas promedio no cambian con el tiempo, y la tasa de llegada es menor que la tasa de servicio.
  - Medidas promedio de largo plazo: Las variables L, λ y W son promedios a largo plazo.
  - No dependiente: La ley es independiente de las distribuciones de llegadas y tiempos de servicio.

Fórmula de Little

Aplicación de la Ley de Little

Ejemplo 1

- Datos:
  - Tasa de llegada (λ): 20 clientes/hora.
  - Tiempo promedio en el sistema (W): 0.25 horas.
  - Cálculo del número promedio de clientes en el sistema (L)=λW=20×0.25=5 clientes
- En promedio, hay 5 clientes en el sistema.

Ejemplo 2

- Datos:
  - Número promedio de clientes en cola (Lq): 2 clientes.
  - Tasa de llegada (λ): 10 clientes/hora.
  - Cálculo del tiempo promedio en cola (Wq):
  - Wq=Lq / λ => 2 / 10 => 0.2 horas => 12 minutos
- Los clientes esperan en promedio 12 minutos antes de ser atendidos.

Fórmula de Little

Aplicación de la Ley de Little

Ejercicio 1

- Una clínica atiende en promedio a 15 pacientes por hora. Si el paciente promedio pasa 20 minutos en la clínica, ¿cuántos pacientes hay en promedio en la clínica?

Ejercicio 2

- En un taller mecánico, hay en promedio 4 vehículos esperando reparación. Si los vehículos llegan a una tasa de 2 por hora, ¿cuánto tiempo espera en promedio un vehículo antes de ser atendido?

Fórmula de Little

Aplicación de la Ley de Little

Ejercicio 1

- Una clínica atiende en promedio a 15 pacientes por hora. Si el paciente promedio pasa 20 minutos en la clínica, ¿cuántos pacientes hay en promedio en la clínica? **Solución:**
  - Convertir el tiempo a horas:
    - W=20 / 60= 0.3333 horas
  - Aplicar la Ley de Little:
    - $L=\lambda W$=15×0.3333=5 pacientes

Ejercicio 2

- En un taller mecánico, hay en promedio 4 vehículos esperando reparación. Si los vehículos llegan a una tasa de 2 por hora, ¿cuánto tiempo espera en promedio un vehículo antes de ser atendido? Solución:
  - Wq = Lq / λ => 4 / 2 => 2 horas

Modelos Poissonianos

Características

- El proceso de Poisson es un modelo probabilístico que describe la ocurrencia de eventos aleatorios en el tiempo o en el espacio.
- Un proceso de Poisson es un proceso de conteo $\{N(t), t \geq 0\}$ que satisface las siguientes condiciones:
  - Incrementos independientes: El número de eventos ocurridos en intervalos disjuntos de tiempo es independiente.
  - Incrementos estacionarios: La distribución del número de eventos en un intervalo de tiempo solo depende de la longitud del intervalo, no del punto de inicio.
  - Probabilidad de un solo evento: La probabilidad de que más de un evento ocurra en un intervalo infinitesimal es insignificante.
  - Homogeneidad en el tiempo: La tasa promedio de ocurrencia de eventos es constante a lo largo del tiempo.

Modelos Poissonianos

Propiedades

- Distribución de Poisson: El número de eventos en un intervalo de tiempo t sigue una distribución de Poisson con parámetro λt, donde λ es la tasa promedio de eventos por unidad de tiempo.

$$P[N(t)=k]=\frac{(\lambda t)^k e^{-\lambda t}}{k!}$$

- Intervalos entre eventos: El tiempo entre eventos sucesivos sigue una distribución exponencial con parámetro λ

$$f_T(t)=\lambda e^{-\lambda t}, t \geq 0$$

- Propiedad de sin Memoria: La distribución exponencial es "sin memoria", lo que significa que el tiempo hasta el próximo evento no depende de cuánto tiempo ha pasado desde el último evento.

Modelos Poissonianos

Distribución exponencial

- La distribución exponencial es una distribución de probabilidad continua que modela el tiempo entre eventos en un proceso de Poisson. Tiene la siguiente función de densidad:

$$f_T(t) = \lambda e^{-\lambda t}, t \geq 0$$

- Donde λ es el parámetro de tasa, que representa el número promedio de eventos por unidad de tiempo.
- Propiedades:
  - Media: $E[T] = 1 / \lambda$
  - Varianza: $\text{Var}(T) = 1 / \lambda^2$
  - Propiedad de sin Memoria:

$$P[T > s+t | T > s] = P[T > t]$$

Esto significa que el tiempo adicional de espera es independiente del tiempo ya transcurrido.

Modelos Poissonianos

Notación Kendall

- La notación de Kendall es una forma estándar de describir sistemas de colas y se expresa como $A/S/c/K/N/D$, donde:
  - A: Distribución de los tiempos entre llegadas (Arrivals).
  - S: Distribución de los tiempos de servicio (Service).
  - c: Número de servidores.
  - K: Capacidad del sistema (opcional).
  - N: Tamaño de la población fuente (opcional).
  - D: Disciplina de la cola (opcional).
- Para el modelo M/M/1, la notación es:
  - **M**: Llegadas según un proceso de Poisson (Markoviano).
  - **M**: Tiempos de servicio exponenciales (Markoviano).
  - **1**: Un solo servidor.

Modelos Poissonianos

Modelo M/M/1/K (Capacidad Limitada)

- Representa un sistema de colas con un solo servidor y una capacidad máxima de K clientes en el sistema (incluyendo al que está siendo atendido y a los que están en cola).
- Si un cliente llega y el sistema ya tiene K clientes, ese cliente es rechazado o bloqueado
- Características principales:
  - Llegadas: Siguen un proceso de Poisson con tasa λ.
  - Servicio: Los tiempos de servicio son independientes y exponencialmente distribuidos con tasa μ.
  - Servidores: Un único servidor.
  - Capacidad del sistema: Limitada a K clientes.
  - Disciplina de cola: Generalmente FIFO (First In, First Out).
  - Población fuente: Infinita.
  - Bloqueo de llegadas: Si el sistema está lleno, las nuevas llegadas son rechazadas.

Modelos Poissonianos

Modelo M/M/1/K – Formulación

Factor de utilización: $\rho = \frac{\lambda}{\mu}$

Probabilidades de estado: $P_n = \frac{\rho^n}{\sum_{i=0}^{K} \rho^i}$

La suma del denominador es una serie geométrica:

$$S = \sum_{i=0}^{K} \rho^i = \frac{1-\rho^{k+1}}{1-\rho}, si\ \rho \neq 1$$

$$S = K + 1\ , si\ \rho = 1$$

Probabilidad de que el sistema esté lleno (PK): $P_K = \frac{\rho^k}{S}$

Tasa efectiva de llegada: $\lambda_e = \lambda(1 - PK)$

Modelos Poissonianos

Modelo M/M/1/K – Formulación

Número promedio de clientes en el sistema (L):

$$L = \sum_{i=0}^{K} nP_n$$

Tiempo promedio en el sistema (W): $W = \frac{L}{\lambda_e}$

Tiempo promedio en cola (Wq): $W_q = W - \frac{1}{\rho}$

Número promedio de clientes en cola (Lq): $Lq = L - (1 - P_0)$

Modelos Poissonianos

Modelo M/M/1/K – Ejercicio

- $\lambda = 4$
- $\mu = 2$
- $K = 5$
- ¿Probabilidad de que el sistema esté lleno (P_5)?

Modelos Poissonianos

Modelo M/M/1/K – Ejercicio

- Factor de utilización $\rho = \lambda / \mu = 4 / 2 = 2$.
  Nota: ρ>1 indica que la demanda excede la capacidad de servicio. En sistemas con capacidad limitada, esto es manejable debido al rechazo de llegadas cuando el sistema está lleno.

- Cálculo de la suma (S). Dado que $\rho = 2$, calculamos:

$$\sum_{i=0}^{5} 2^i = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 =$$
$$1 + 2 + 4 + 8 + 16 + 32 = 63$$

Modelos Poissonianos

Modelo M/M/1/K – Ejercicio

- Probabilidades de estado (P_n):
- $P_n = \frac{2^n}{63}$
- Calculamos P_0 a P_5:

$$P_0 = \frac{1}{63} \approx 0.0159$$

$$P_1 = \frac{2}{63} \approx 0.0317$$

Modelos Poissonianos

$$P_2 = \frac{4}{63} \approx 0.0635$$

$$P_3 = \frac{8}{63} \approx 0.1270$$

$$P_4 = \frac{16}{63} \approx 0.2540$$

$$P_5 = \frac{32}{63} \approx 0.5079$$

Modelos Poissonianos

Modelo M/M/1/K – Ejercicio

- Tasa efectiva de llegada (λ_e)

$$\lambda_e = \lambda(1 - PK) =$$
$$4(1 - 0.5079) =$$
$$4 \times 0.4921 \approx 1.9684\ coches/hora$$

- Número promedio de autos en el sistema (L)

$$L = \sum_{i=0}^{K} nP_n$$
$$= 0 * P_0 + 1 * P_1 + 2 * P_2 + 3 * P_3 + 4 * P_4 + 5 * P_5$$

L=(0)(0.0159)+(1)(0.0317)+(2)(0.0635)+(3)(0.1270)+(4)(0.2540)+(5)(0.5079)

L=0+0.0317+0.1270+0.3810+1.0160+2.5395 = 4.0952 COCHES

Modelos Poissonianos

Modelo M/M/ ∞ /

- Es un caso especial en la teoría de colas donde se asume que hay un número infinito de servidores disponibles.
- Esto significa que cada cliente que llega es atendido inmediatamente sin esperar.
- Este modelo es útil para analizar sistemas donde la capacidad de servicio es muy alta o prácticamente ilimitada. estar siendo

Modelos Poissonianos

Modelo M/M/ ∞ /

- Llegadas: Siguen un proceso de Poisson con tasa λ.
- Servicio: Los tiempos de servicio son independientes y exponencialmente distribuidos con tasa μ.
- Servidores: Número infinito de servidores.
- Capacidad del sistema: Infinita; no hay límite en el número de clientes que pueden estar siendo atendidos a la vez.
- Disciplina de servicio: Dado que no hay cola, la disciplina es irrelevante.

Modelos Poissonianos

Modelo M/M/ ∞ /

- Probabilidad de n clientes en el sistema (P_n)

$$P_n = \frac{p^n e^{-p}}{n!}$$

- Número promedio de clientes en el sistema (L)

$$\mathrm{L} = \rho = \frac{\lambda}{\mu}$$

- Tiempo promedio en el sistema (W)

$$W = \frac{1}{\mu}$$

- ¿Tiempo promedio en cola (Wq), y número promedio en cola (Lq)?

Modelos Poissonianos

Modelo M/M/ ∞ / - Ejercicio

- Tasa de llegada (λ): 1000 correos por segundo.
- Tasa de servicio (μ): 500 correos por segundo por servidor (pero hay infinitos servidores).
- ¿Métricas de eficiencia aplicables?
- Probabilidad de que haya n correos en el sistema (P_n)

Modelos Poissonianos

Modelo M/M/ ∞ / - Ejercicio

- Factor de utilización (p)

$$\rho = \frac{\lambda}{\mu} = \frac{1000}{500} = 2$$

- Número promedio de correos en el sistema (L)

$$L = \rho = 2$$

- Tiempo promedio en el sistema (W)

$$W = \frac{1}{\mu} = \frac{1}{500} = 0.002$$

- Probabilidad de que haya exactamente 3 correos en el sistema:

$$P_3 = \frac{p^3 e^{-p}}{3!} = \frac{2^3 e^{-2}}{6} \approx 0.1804$$

Redes de colas

Conceptos fundamentales

- Es un conjunto de uno o más nodos (o estaciones) de cola interconectados, donde cada nodo es un sistema de colas individual.
- Los clientes pueden:
  - Llegar desde el exterior a una o varias estaciones.
  - Moverse entre estaciones según ciertas probabilidades de enrutamiento.
  - Salir del sistema después de completar su servicio.
- Estas redes modelan situaciones reales donde las entidades deben pasar por varios procesos o etapas antes de completar su servicio.

Redes de colas

Elementos clave

- **Estaciones** (Nodos): Puntos donde se ofrece un servicio. Cada estación tiene su propia disciplina de cola, tasa de servicio y capacidad.
- **Clientes**: Entidades que requieren servicio y se mueven a través de la red.
- **Enrutamiento**: Las reglas o probabilidades que determinan cómo los clientes se mueven de una estación a otra.
- **Redes Abiertas, Cerradas y Mixtas**: Clasificación basada en cómo los clientes entran y salen del sistema.

Redes de colas

Red abierta

- Los clientes pueden:
  - **Entrar** al sistema desde el exterior.
  - **Moverse** entre las estaciones dentro de la red.
  - **Salir** del sistema al completar su servicio.
- Características:
  - **Fuente infinita de clientes**: Se asume que hay un número ilimitado de clientes potenciales que pueden entrar al sistema.
  - **Tasas de llegada externas**: Los clientes llegan a ciertas estaciones desde fuera del sistema con una tasa determinada.
  - **Probabilidades de enrutamiento**: Después de ser atendido en una estación, un cliente puede moverse a otra estación o salir del sistema, según probabilidades específicas.
  - **Estado estacionario**: El análisis suele enfocarse en el comportamiento a largo plazo cuando el sistema alcanza un estado estable.

Redes de colas

Red abierta

- Tasa de llegada a cada estación (λ_i):

$$\lambda_i = ei + \sum_{j=1}^{N} \lambda_j P_{ji}$$

- Donde:
  - e_i: Tasa de llegadas externas a la estación i.
  - P_{ji}: Probabilidad de que un cliente vaya de la estación j a la estación i.
  - N: Número total de estaciones.
- Utilización de la estación i (ρ_i):

$$\rho_i = \frac{\lambda_i}{\mu_i}$$

- Donde μi es la tasa de servicio de la estación i.

Redes de colas

Red abierta - Ejemplo

- Una empresa de servicios tiene tres departamentos: **A**, **B** y **C**.
- Llegadas externas:
  - Los clientes llegan al departamento **A** con una tasa de **5 clientes/hora**.
- Enrutamiento después del servicio en A:
  - **50%** de los clientes van al departamento **B**.
  - **30%** de los clientes van al departamento **C**.
  - **20%** de los clientes **salen del sistema**.
- Salida del sistema:
  - Después de ser atendidos en **B** o **C**, todos los clientes **salen del sistema**.

Redes de colas

Red abierta - Ejemplo

- Determinar las tasas de llegada a cada departamento.
- Calcular la tasa total de llegadas a B y C.
- Calcular las utilizaciones y tiempos de espera en cada departamento, asumiendo tasas de servicio conocidas.
- Implementar el análisis en Python.

Redes de colas

Red abierta - Ejemplo

Determinación de las tasas de llegada a cada departamento

- Llegadas a departamento A
  - λ_A : Tasa de llegadas externas al departamento A => λ_A=5 clientes/hora
- Enrutamiento A-B y A-C
- Después de ser atendidos en A, los clientes se dirigen a otros departamentos o salen del sistema según las siguientes probabilidades:
  - Probabilidad de ir a B: P_{AB} = 0.5
  - Probabilidad de ir a C: P_{AC} = 0.3
  - Probabilidad de salir del sistema: P_{A0} = 0.2

Redes de colas

Red abierta - Ejemplo

Determinación de las tasas de llegada a cada departamento

- Cálculo de las Tasas de Llegada a B y C
- Las tasas de llegada a B y C desde A son:
  - λ_{AB}: Tasa de llegada a B desde A. $\lambda_{AB} = \lambda_A \times P_{AB} = 5 \times 0.5 = 2.5$ clientes/hora
  - λ_{AC}: Tasa de llegada a C desde A. $\lambda_{AC} = \lambda_A \times P_{AC} = 5 \times 0.3 = 1.5$ clientes/hora
  - Nota: No hay llegadas externas directas a B o C, y no hay retroalimentación hacia A.
- Tasas totales de llegada a B y C
- Dado que después de B y C los clientes salen del sistema, las tasas totales de llegada a B y C son:
  - $\lambda_B = \lambda_{AB} = 2.5$ clientes/hora
  - $\lambda_C = \lambda_{AC} = 1.5$ clientes/hora

Redes de colas

Red abierta - Ejemplo

Calcular las utilizaciones y tiempos de espera

- Para calcular las utilizaciones y tiempos de espera, necesitamos conocer las tasas de servicio (μ) de cada departamento. Supongamos que las tasas de servicio son las siguientes:
  - Departamento A: μ_A = 6 clientes/hora
  - Departamento B: μ_B = 5 clientes/hora
  - Departamento C: μ_C = 4 clientes/hora
- Además, asumimos que cada departamento se comporta como un sistema M/M/1, es decir, llegadas Poisson, tiempos de servicio exponenciales y un solo servidor.

Redes de colas

Red abierta - Ejemplo

Cálculo de las utilizaciones (*p*)

- La utilización de cada departamento es:

$$\rho_i = \frac{\lambda_i}{\mu_i}$$

$$\rho_A = \frac{\lambda_A}{\mu_A} = \frac{5}{6} \approx 0.8333$$

$$\rho_B = \frac{\lambda_B}{\mu_B} = \frac{2.5}{5} \approx 0.5$$

$$\rho_C = \frac{\lambda_C}{\mu_C} = \frac{1.5}{4} \approx 0.375$$

Redes de colas

- Departamento A

$$L_A = \frac{0.8333}{1 - 0.8333} \approx 5$$

$$L_{qA} = \frac{(0.8333)^2}{1 - 0.8333} \approx 4.1667$$

$$W_A = \frac{1}{6 - 5} = 1h$$

$$W_{qA} = WA - \frac{1}{\mu A} = 1 - \frac{1}{6} \approx 0.8333h$$

Redes de colas

- Departamento B

$$L_B = \frac{0.5}{1 - 0.5} = 1$$

$$L_{qB} = \frac{(0.5)^2}{1 - 0.5} = 0.5$$

$$W_B = \frac{1}{5 - 2.5} = 0.4h$$

$$W_{qB} = WB - \frac{1}{\mu B} = 0.4 - \frac{1}{5} = 0.2h$$

Redes de colas

- Departamento C

$$L_C = \frac{0.375}{1 - 0.375} = 0.6$$

$$L_{qC} = \frac{(0.375)^2}{1 - 0.375} = 0.225$$

$$W_C = \frac{1}{4 - 1.5} = 0.4h$$

$$W_{qC} = WC - \frac{1}{\mu C} = 0.4 - \frac{1}{4} = 0.15h$$

Redes de colas

Red abierta - Ejemplo

Interpretación de resultados

- Departamento A
  - Alta utilización (83.33%): El departamento A está muy ocupado, lo que resulta en un tiempo promedio en el sistema de 1 hora y un tiempo de espera en cola de aproximadamente 50 minutos.
  - Número promedio en el sistema: Hay en promedio 5 clientes en el departamento A, de los cuales 4.1667 están esperando en cola.
- Departamento B
  - Moderada utilización (50%): El departamento B tiene una utilización moderada.
  - Tiempo en el sistema: Los clientes pasan en promedio 24 minutos en B, con 12 minutos de espera en cola.
  - Número promedio en el sistema: En promedio, hay 1 cliente en B.
- Departamento C
  - Baja utilización (37.5%): El departamento C está menos ocupado.
  - Tiempo en el sistema: Los clientes pasan en promedio 24 minutos en C, con 9 minutos de espera en cola.
  - Número promedio en el sistema: En promedio, hay 0.6 clientes en C.

Bibliografía

- Hillier & Liebermann (2010). Introducción a la investigación de operaciones, 9ªed. Capítulos 1 y 2.
- González y García (2015). Manual práctico de investigación de operaciones 4ª ed. Capítulo 1.
- Investigación Operativa (2004). Modelos determinísticos y estocásticos. Sixto Ríos Insua, Alfonso Mateosos Caballero, Mª Concepción Bielza Lozoya y Antonio Jiménez Martín, Editorial Centro de Estudios Ramón Areces, S.A., Madrid.

Bibliografía

- Investigación Operativa. Optimización, Sixto Ríos Insua. Editorial Centro de Estudios Ramón Areces, S.A., Madrid.
- Problemas de Investigación Operativa. Programación Lineal y Extensiones. Sixto Ríos Insua, David Ríos Insua, Alfonso Mateos Caballero, Jacinto Martín Jiménez y Antonio Jiménez Martín. Editorial Ra-Ma, Madrid 2006.
- Model Building in Mathematical Programming. Paul W. Williams. Editorial JOHN WILEY AND SONS
- Apuntes de Victoria Ruiz y Antonio Alonso
- Apuntes de Alberto Herrán y José J. Ruz Ortiz
- Apuntes de Gerardo Reyes y Salvador Sánchez

Grado en Ingeniería del Software

Investigación Operativa

BLOQUE III. MODELOS ESTOCÁSTICOS

Tema 4: Fenómenos de espera

URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es