Sección de Matemáticas
Universidad de La Laguna

Agoney Bentejuí Pérez Medina

Introducción al Problema de Reposición Conjunta de Artículos

Introduction to the Joint Replenishment Problem (JRP)

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Julio de 2024

Dirigido por
José Miguel Gutiérrez Expósito

José Miguel Gutiérrez Expósito
Departamento de Matemáticas,
Estadística e Investigación
Operativa
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

Quiero agradecer a mi familia y amigos por apoyarme siempre, sin ellos no hubiera sido posible.

También a José Miguel, por haber hecho una labor excelente como tutor y ayudarme en esta última etapa.

Agoney Bentejuí Pérez Medina
La Laguna, 4 de julio de 2024

Resumen · Abstract

Resumen

Esta memoria pretende ser una introducción a los Modelos de Tamaño del Lote para varios artículos considerando costes de reposición conjunta (varios tipos de artículos) y por tipo de artículo (marginales). Teniendo en cuenta que se incurrirán en los primeros cuando se reponga al menos un artículo.
En el primer capítulo se repasarán los conceptos básicos de la Gestión de Inventarios y el Modelo Clásico de Tamaño del Lote (EOQ). Además, se presentarán conceptos, como la Complejidad Computacional, y métodos, como la Programación Dinámica, que nos ayudarán a entender la dificultad del modelo propuesto en el capítulo 2 y a diseñar algoritmos para su resolución. Asimismo, discutiremos la versión dinámica del modelo EOQ, introduciendo la caracterización de un tipo de soluciones óptimas y presentando la correspondiente expresión de recursión de la Programación Dinámica. Finalmente, describiremos también la técnica de Ramificación y Acotación/Poda (Branch and Bound) para la resolución de Problemas de Programación Lineal Entera Mixta, que se empleará en el segundo capítulo.
Precisamente, será en el segundo capítulo, donde se estudiará el Modelo del Tamaño del Lote con Varios Artículos con costos conjuntos, también conocido como Problema de Planificación de Pedidos con Multi-artículo, presentándose una formulación alternativa para el problema y varios métodos de resolución. Se verán además las diferencias de los distintos métodos respecto de su complejidad computacional.
El objetivo de esta memoria es presentar varios métodos para la resolución de un Problema de tamaño de lote haciendo uso de la Programación Dinámica y la Optimización. Destacará la complejidad computacional del problema debido su elevada dificultad, llegándose incluso a ser intratable por una computadora en un tiempo computacionalmente razonable.

Palabras clave: *Gestión de inventarios – Optimización – Programación Dinámica – Modelos de tamaño de lote.*

Abstract

This work is intended to be an introduction to the Multi-Product Lot-Size Models with joint replenishment costs (different types of articles) and for types of articles (marginal). Taking into account that the first will incurred when at least one product is replenished.
In the first chapter basic concepts about Inventory Management and the Economic Order Quantity Model (EOQ) will be reviewed. In addition, concepts will be presented, such as Computational Complexity, and methods, such as Dynamic Programming, that will help us to understand the difficulty of the model proposed in chapter 2 and to design algorithms for its resolution. Likewise, we will discuss the dynamic version of the EOQ model, introducing the characterization of a type of optimal solutions and presenting the corresponding recursion expression of Dynamic Programming. Finally, we will describe the Branch and Bound Algorithm for Mixed-Integer Lineal Programming Problems, which will be used in the second chapter.
Precisely, it would be in the second chapter where the Multi-Product Lot-Size Model with joint replenishment costs will be studied, presenting an alternative formulation for solve the problem and various resolution methods. They will also see the differences between the different methods regarding their computacional complexity.
The aim of this work is to present differents resolution methods for a Lot Size Problem using Dynamic Programming and Optimization. Computacional complexity of the problem will highlight due to its high difficulty, reaching even to be intractable in a computacionally reasonable time.

Keywords: *Inventory management – Optimization – Dynamic Programming – Lot-Size Models.*

Contenido

Introducción

En las últimas décadas se ha producido un crecimiento demográfico mundial sin precedentes, alcanzando la cifra de casi 8000 millones de personas. Por suerte, los avances en tecnología y ciencia también se han desarrollado vertiginosamente. Todo ello ha permitido que gran parte de la demanda de productos a nivel mundial se pueda abastecer por la numerosa oferta de empresas y distribuidores. Por ello, la Investigación Operativa, como rama de las Matemáticas que aborda los modelos y técnicas de resolución a problemas de decisión, ha tomado especial relevancia en el sector empresarial.

En particular, destaca la Gestión de Inventarios, la disciplina de la Investigación Operativa que se encarga de gestionar de manera óptima los bienes del inventario desde el origen hasta su destino. De esta manera, el objetivo es buscar el óptimo rendimiento del inventario respecto a mantenimiento de productos, reposición y pérdida de los mismos así como sus costos asociados. Todo ello forma parte de lo que denominamos una política óptima de inventario.

La correcta resolución de un problema de inventario consiste en buscar la política óptima. Para poder ajustarnos a la realidad, son diversos los modelos y formulaciones matemáticas que existen dado que aspectos como la demanda, los periodos, los costos de reposición, productos perecederos y muchas más variables se ven involucradas. Así, una correcta gestión del inventario indica cuántos productos se deben reponer en cada periodo y cada cuánto tiempo se realizarán dichas reposiciones, además de los costos asociados a la reposición, mantenimiento y pérdida de los productos.

Cabe destacar que, para resolver los modelos de gestión de inventarios más complejos, se emplean diversas técnicas matemáticas tales como métodos heurísticos, algoritmos de la rama de la Optimización y Programación Dinámica entre muchos otros. En esta memoria veremos varios métodos utilizando las disciplinas indicadas. Por todo ello, las Matemáticas ayudan a las empresas a obtener mayores beneficios.

Por otro lado, destaca la complejidad computacional de algunos modelos de gestión de inventarios, por su dificultad a la hora de resolverlos en un tiempo

computacionalmente razonable. Resulta muy interesante a la vez que desafiante trabajar con problemas intratables, esto es, problemas que no se pueden resolver computacionalmente en un tiempo polinomial. En efecto, la dificultad de estos problemas motiva a usar diferentes métodos de resolución en búsqueda de una política óptima mejor que la anterior.

En lo que sigue, introduciremos conceptos básicos sobre la Gestión de Inventarios y la Complejidad Computacional, abarcaremos la Programación Dinámica para la resolución de Problemas de Decisión y estudiaremos el Algoritmo Branch and Bound para la resolución de Problemas de Programación Entera Lineal Mixta. Posteriormente, en el segundo capítulo se estudiarán los Modelos Clásicos de Tamaño de Lote para Varios Artículos con Costos de Reposición Conjunta donde haremos uso de todo lo introducido en el primer capítulo para resolver un particular problema de este tipo. Además se ilustrará un ejemplo numérico haciendo uso de un código en Python.

1

Modelos de gestión de inventarios y conceptos básicos

El constante crecimiento de la humanidad ha generado una inmensa demanda de productos a nivel mundial. Con el fin de abastecerla, las empresas y los distribuidores deben generar tanta oferta como para satisfacer dicha demanda. Por ello, la Investigación Operativa y, en particular, la Gestión de Inventarios son de especial utilidad a la hora de encontrar políticas óptimas de control de inventarios.

¿Qué impacto económico tendría no aplicar un control eficiente de las existencias (*stock*)? Es fácil llegar a la conclusión de que si tenemos menos artículos de los que se demandan, estaríamos perdiendo beneficios y confianza del cliente, que resulta más difícil de estimar. Así mismo, si tuviéramos más artículos de los que vayamos a vender esto podría generar costos adicionales. Todo aquello que implique un costo extra en nuestra política es un problema, pues nuestro objetivo será optimizar el costo, bien sea minimizándolo o maximizando el beneficio. Además, no es fácil determinar una política óptima, pues debemos tener en cuenta factores como que la demanda, que podría variar con el tiempo, productos perecederos, periodos no constantes, etc. Por ende, la Gestión de Inventarios debería estar asociada al plan de beneficios de una empresa [18].

El control de los inventarios empezó a estudiarse de manera científica a comienzos del siglo pasado. Los trabajos iniciales, desarrollados independientemente por F.W. Harris [8] y R.H. Wilson [24], dieron como fruto la famosa fórmula del *Economic Order Quantity* (o *EOQ*), que calcula la cantidad óptima de pedido que minimiza la suma de los costes de mantenimiento y reposición/producción de un único artículo, cuando los parámetros de entrada son estacionarios (no cambian con el tiempo).

Desde ese momento y hasta la actualidad, el desarrollo de nuevos modelos de inventarios más realistas, y por lo tanto más complejos, ha sido imparable, hasta tal punto que la Gestión de Inventarios se ha consolidado como una de las ramas más relevantes dentro la *Investigación Operativa*. A continuación, se introducirá tanto el modelo *EOQ* como su versión dinámica, además de discutir algunos conceptos y técnicas clásicas de optimización.

1.1. Modelo clásico de tamaño de lote (*EOQ*)

En este apartado definiremos conceptos básicos sobre la gestión de inventarios así como la formulación matemática del *EOQ*.

1.1.1. Conceptos básicos

Para describir el modelo clásico de la cantidad óptima de pedido debemos introducir primero la siguiente notación, que será usada también a lo largo de esta memoria.

- **Inventario:** es el conjunto de artículos mantenidos por cierto tiempo en almacén, que poseen un valor económico y que será usado para satisfacer la demanda.
- **Demanda(r):** es una tasa que representa las unidades de artículos que los consumidores demandan por unidad de tiempo. Estas cantidades son satisfechas con unidades del inventario.
- **Tamaño de lote(Q):** es la cantidad de artículos a pedir, comprar o producir en un solo lote o pedido de reposición.
- **Periódo de gestión(T):** es el tiempo que transcurre entre dos pedidos consecutivos.
- **Tiempo de retardo:** es el tiempo que transcurre entre la realización de un pedido a un proveedor externo y la disposición de los artículos en el almacén para su venta.
- **Nivel inicial de inventario(S):** es la cantidad de stock del producto al principio del ciclo de inventario.
- **Punto de pedido(s):** es el nivel de stock del producto en el qué se debe solicitar una reposición del mismo.

La estructura de costes del *EOQ* estará formada por los siguientes componentes:

- **Coste de mantenimiento del inventario(C_1):** es el costo relacionado con el mantenimiento y conservación (*carrying* o *holding* cost) de los artículos en el almacén. Puede contener una componente fija (*personal de almacén, seguridad, impuestos, cuota préstamo/leasing, etc)* y otra variable (*gastos corrientes (agua, energía), coste de oportunidad, etc*), que suele imputarse a cada unidad de producto mantenida en el almacén.
- **Coste de rotura(C_2):** se incurre en él cuando la demanda de un determinado artículo no puede satisfacerse de manera inmediata. La rotura (*stockout* o *shortage* en inglés) suele resolverse mediante uno de los siguientes enfoques. O bien se asume que el cliente está dispuesto a esperar hasta que se reponga el inventario (*backlogging*) para así satisfacer su demanda; o bien se considera

que el cliente es impaciente, por lo que la venta se pierde (*lost sale*). La determinación del coste de rotura en el primer escenario no es sencilla, ya que debe medir con cierta precisión el grado de insatisfacción del cliente. Es de esperar que si esta situación (*backlogged demand* o demanda acumulada) se repite con frecuencia, el cliente termine perdiendo completamente la confianza (y la paciencia) en el proveedor, por lo que la venta podría perderse finalmente.

- **Coste de reposición(C_3):** es el coste relacionado con la reposición de artículos. Al igual que el coste de mantenimiento, puede estar formado por una componente fija (por ejemplo, costes de *personal*) y otra variable (por ejemplo, *gastos corrientes*), que suele imputarse a cada unidad de artículo del pedido. Puede variar según el artículo o bien la empresa.

El modelo *EOQ* es una idealización de la realidad, ya que, en su versión original (ver [8] y [24]), sólo considera un único artículo, asume que la reposición es instantánea, no permite roturas, los parámetros son conocidos y estacionarios (no cambian con el tiempo). No obstante, su influencia a nivel de la gestión del inventario sigue manteniéndose a día de hoy, ya que es un modelo bastante robusto a pequeñas variaciones de los parámetros y es muy fácil de implementar.

A la hora de calcular los costes, necesitaremos saber el nivel medio de inventario, la cantidad media de rotura y el número medio de reposiciones, que denotaremos por I_1, I_2 y I_3, respectivamente. Además debemos conocer el costo unitario de mantenimiento, costo unitario y de rotura y costo unitario de reposición, denotados como C_1, C_2 y C_3, respectivamente.

En el Capítulo 2 se estudiará en detalle una variante más realista y compleja del modelo *EOQ*. En particular, se considerará que los parámetros (demanda y costes) no son estacionarios (pueden variar en cada periodo del horizonte temporal finito), lo cual nos lleva a tratar una versión dinámica del *EOQ*, introducida por H.M. Wagner y T.M. Whitin en [25], en la que además se contemplan varios arículos y se incorpora un coste adicional de pedido conjunto, como así fue estudiado originalmente por E.P.C. Kao en [11]. En presencia de este tipo de costes de pedido conjunto, bastaría con reponer un único artículo para que se active el coste de reposición/pedido en ese periodo.

1.1.2. Formulación matemática del *EOQ*

Una vez conocidos estos conceptos básicos sobre la gestión de inventarios, estamos en disposición de introducir el módelo clásico de la cantidad económica de pedido o *EOQ*. A continuación se detallan de las principales características del modelo:

- Razón de demanda r determinista y constante
- Tamaño de lote o cantidad económica de pedido $Q > 0$ (variable a determinar), que está relacionada con el periodo de gestión según la igualdad $Q = Tr$

- No se permiten roturas ($C_2 = 0$)
- Se asume reposición instantánea
- Punto de reposición $s = 0$
- Tiempo de retardo nulo
- Patrón de demanda uniforme

Teniendo en cuenta estas consideraciones, las fluctuaciones del nivel de inventario en el modelo EOQ se muestran en la Figura 1.1. Asimismo, la formulación del problema viene dada en la expresión (1.1), en la que el objetivo es determinar la cantidad óptima de pedido (Q) que minimice el coste total por unidad de tiempo (C):

$$C : \min_{Q>0} C_1(Q) + C_3(Q) \tag{1.1}$$

donde $C_1(Q)$=$c_1 I_1(Q)$, $C_3(Q)$=$c_3 I_3(Q)$

La gráfica de nuestro problema viene dada por:

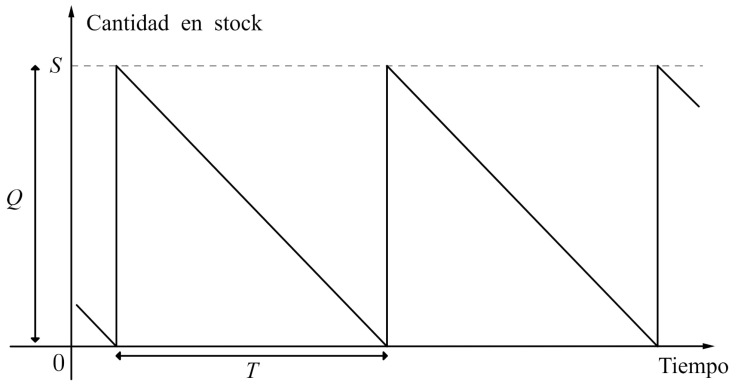


Figura 1.1: Gráfica EOQ

Para calcular la cantidad media en el inventario (stock) por unidad de tiempo, $I_1(Q)$, tomaremos el área bajo del triángulo de la gráfica y dividiremos entre el periodo T, luego: $I_1(Q) = \dfrac{T\frac{Q}{2}}{T} = \frac{Q}{2}$
Por otro lado, $I_3(Q) = \frac{1}{T} = \frac{r}{Q}$ En efecto, se tiene que:

$$C(Q) = c_1 \frac{Q}{2} + c_3 \frac{r}{Q} \tag{1.2}$$

Ahora, debemos obtener el valor Q_0 que minimice la función (1.2). Para ello, derivaremos respecto de Q e igualaremos a cero, lo que devuelve la cantidad óptima de pedido:

$$Q_0 = \sqrt{\frac{2c_3 r}{c_1}},$$

cuyo coste mínimo viene dado por

$$C_0 = C(Q_0) = c_1 \frac{Q_0}{2} + c_3 \frac{r}{Q_0} = \sqrt{2rc_1c_3},$$

y donde el periodo de reposición óptimo se determina como:

$$T_0 = \frac{Q_0}{r} = \sqrt{\frac{2c_3}{rc_1}}$$

Como discutimos anteriormente, las suposiciones del modelo *EOQ* original difícilmente se darán en la práctica, pues rara vez encontraremos una tasa de demanda constante en el mercado. Asimismo, será complicado encontrar ejemplos del mundo real en los que la reposición ocurra de manera instantánea.

Lo normal es que, una vez tramitado un pedido, se deben seleccionar y preparar los productos (*packaging*) en las instalaciones del proveedor, para su posterior distribución, tramitación aduanera (como es el caso de Canarias) y almacenamiento en el minorista. Incluso en el caso de la compra de aplicaciones informáticas por vía telemática, donde prima la inmediatez, el tiempo entre la realización del pedido online y la disponibilidad para su uso por el cliente final no es nulo. Seguramente requerirá de un proceso de autenticación/registro por parte del cliente en la web del proveedor, verificación de la forma de pago, tiempo de descarga del programa y registro del producto.

Quizás, un ejemplo en el que se podrían asumir tiempos de retardo despreciables (casi nulos) es el de las operaciones bursátiles, en las que las compra/venta de activos se hacen a una velocidad frenética, por lo que los tiempos de latencia (medidos en milisegundos o en microsegundos) son prácticamente despreciables.

En base a lo indicado anteriormente, la realidad es que el *EOQ* no se ajusta a las exigencias del mercado actual. Es por ello que este modelo ha tenido que evolucionar para adaptarse mejor a diferentes situaciones que pueden plantearse en la práctica, dando lugar a un abanico amplio de variantes del *EOQ*. En el Capítulo 2 de esta memoria revisaremos algunas de ellas, como es el caso de la versión dinámica del *EOQ*, introducida por que Wagner y Whitin en [25] y que fue resuelta utilizando Programación Dinámica. Asimismo, también repasaremos una extensión de ese modelo dinámico, presentada por Kao en [11], que considera varios artículos y costes de reposición conjunta. Sin embargo, debemos introducir previamente conceptos y metodologías de optimización usadas en la resolución de las dos variantes para poder avanzar adecuadamente.

1.2. Complejidad computacional

La Optimización tiene una estrecha relación con las ciencias de la computación, pues es importante estudiar la complejidad de los algoritmos de resolución de problemas. La Complejidad Computacional estudia la clasificación de los problemas computacionales según su dificultad (Michael Sipser, 2012) [20]. Esta teoría nos indica que ciertos problemas son intratables de forma determinista por una computadora en un tiempo computacionalmente razonable.

1.2.1. Órdenes de complejidad

En computación se utilizan varias notaciones (Ω, $\mathcal{O}$ y Θ) para representar la complejidad de un algoritmo, de modo que se pueda relacionar el aumento de iteraciones como función del aumento del tamaño n de los datos de entrada. Así, por ejemplo, la notación $\Omega()$ indica la cota inferior del tiempo de ejecución de un algoritmo (es decir, proporciona la complejidad en el mejor caso), mientras que la notación $\Theta()$ devuelve la complejidad media de un algoritmo, dado que establece cotas superiores e inferiores para los tiempos de ejecución. Por su lado, la notación $\mathcal{O}()$ grande, representa la complejidad del algoritmo en el peor caso, por lo que supone una cota superior para los tiempos de ejecución, y suele ser la notación más usada para mostrar el comportamiento asintótico de un algoritmo. En concreto, diremos que un algoritmo es de orden $f(n) = \mathcal{O}(g(n))$ si podemos encontrar un número real positivo M y un entero n_0 tal que $|f(n)| \leq Mg(n)$, para todo $n \geq n_0$. A continuación, se muestran en la Tabla 1.1 diferentes órdenes de complejidad tipo $\mathcal{O}$ grande.

Orden	**Nombre**
$\mathcal{O}(1)$	Constante
$\mathcal{O}(log(n))$	Logarítmica
$\mathcal{O}(n)$	Lineal
$\mathcal{O}(nlog(n))$	Casi lineal
$\mathcal{O}(n^2)$	Cuadrática
$\mathcal{O}(c^n)$	Exponencial
$\mathcal{O}(n!)$	Factorial

Tabla 1.1: Órdenes de complejidad computacional

Veamos además en la gráfica 1.2 como varía el tiempo de las iteraciones con respecto al tamaño n de entrada:

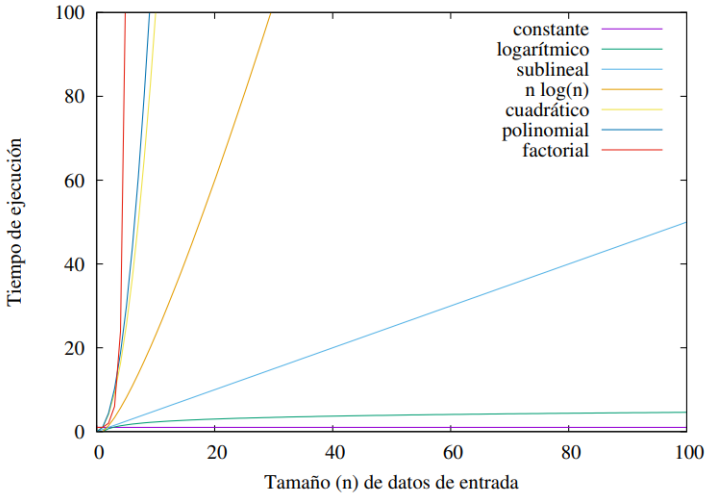


Figura 1.2: Representación gráfica de los órdenes

En esta gráfica podemos observar que las funciones ilustradas en la Tabla 1.1 están ordenadas de menor a mayor respecto al tiempo de ejecución. Ahora, veremos una clasificación general de los problemas bajo la Teoría de la Complejidad Computacional, la cuál se estudia en profundidad en [20].

1.2.2. Problemas Clase *P*

Los problemas de Clase P son aquellos problemas de decisión que pueden ser resueltos en tiempo polinomial con un algoritmo determinista, se les consideran sencillos de resolver y se les denomina tratables. Cuando hablamos de problemas de decisión, entendemos problemas que tienen la característica de que su respuesta es sí o no. Para resolverlos, deberemos construir un algoritmo determinista que encuentre su solución óptima.

1.2.3. Problemas Clase *NP*

Los problemas de Clase NP son todos aquellos problemas de decisión que se pueden verificar en tiempo polinomial con un algoritmo no determinista. Para resolverlos, debemos buscar un algoritmo no determinista que lo resuelva y comprobar si es solución o no en tiempo polinomial.

1.2.4. Problemas Clase *NP-Completo*

Los problemas de Clase *NP-Completo* son todos aquellos problemas en los que se ha probado que no existe un algoritmo determinista polinomial que los resuelva. Diremos que un problema de decisión A pertenece a la Clase *NP-Completo* si se cumple que:

- $A \in$ Clase-*NP*
- $\forall S \in$ Clase-*NP* es reducible en tiempo polinomial a A

1.2.5. Problemas Clase *NP-Duro*

Los problemas de Clase *NP-Duro* son aquellos problemas de optimización que son, al menos, tan difíciles como los problemas Clase *NP*. Por lo tanto, para estos problemas no existe un algoritmo determinista polinomial que los resuelva. Diremos que un problema de decisión B pertenece a la Clase *NP-Duro* si se cumple que:

- $B \in$ Clase-*NP-Completo*
- $\forall S' \in$ Clase-*NP-Completo*, $S' \leq A$

En el segundo capítulo trataremos un problema de tipo *NP-Duro* y buscaremos diferentes algoritmos de resolución con sus respectivos órdenes de complejidad computacional.

1.3. Introducción a la Programación Dinámica

La Programación Dinámica (o *PD* para abreviar) es una técnica de resolución de problemas de decisión secuencial, en la que el problema original es descompuesto en subproblemas más pequeños que pueden abordarse de manera más sencilla, consiguiendo así una reducción del tiempo de ejecución. Fue ideada por el matemático Richard Bellman en 1953 y su eficiencia se basa en el *Principio de Optimalidad*, que establece que una política (solución) óptima global de un problema está conformada por políticas óptimas locales de los subproblemas que contiene. Esto quiere decir que si tenemos la solución óptima de un problema, cualquier subconjunto de la misma también será óptimo para el subproblema que resuelve. No obstante, la contrapartida de esta técnica es que debe resolver todos los posibles subproblemas que se puedan generar para asegurar la optimalidad de la solución del problema original, lo cual podría llevar a lo que se conoce como *the curse of dimensionality* en inglés, es decir, a una explosión combinatoria del número de subproblemas a resolver, dejando, por tanto, de ser atractiva como método solución. Esto ocurre en problemas de complejidad computacional de orden exponencial. En esos casos, otro tipo de técnicas serán más recomendables, como, por ejemplo, la Ramificación y Poda/Acotación (Branch and Bound), que se discutirá más adelante.

En particular, esta técnica será usada en el Capítulo 2 para diseñar un algoritmo que permita resolver el problema de reposición conjunta de múltiples artículos a lo largo de un horizonte temporal finito. Además, también será usada a continuación para la resolución del modelo clásico de tamaño de lote en su versión dinámica (Wagner-Whitin).

1.3.1. Características principales de la Programación Dinámica

Introduzcamos los conceptos básicos con los que trabajaremos en esta sección:

- **Etapa:** es una unidad básica de tiempo o progreso en la resolución del problema (dividiremos el problema en varias etapas). Cada etapa tiene cierto número de estados.
- **Estado:** son las condiciones en la que se encuentra el problema en cada etapa.
- **Variable de decisión:** representa la decisión que tomaremos para llegar a la etapa siguiente.
- **Función de transición:** devuelve el resultado (expresado en las unidades usadas en el problema) de transitar de un estado del problema a otro, que estarán en diferentes etapas, cuando se toma una determinada decisión. Esta información suele almacenarse en tablas, evitando así la repetición de cálculos.

El objetivo de la *PD* será determinar una solución óptima del problema original, que vendrá dada como un conjunto de decisiones secuenciales. La técnica puede apoyarse en la Teoría de Grafos para modelar el conjunto de etapas y estados, así como las transiciones entre ellos. Es por ello que a continuación definiremos conceptos básicos de Grafos.

Definición 1.1. *Un grafo $G(V, E)$ viene definido por un conjunto de vértices/nodos V unidos entre sí mediante aristas/arcos del conjunto E. Permiten establecer relaciones binarias entre los elementos de un conjunto.*

Veáse un ejemplo en la Figura 1.3

Definición 1.2. *Diremos que un grafo $G(V, A)$ es directo o dirigido si sus nodos están conectados entre sí por arcos de A que especifican el sentido a tomar cuando se atraviesan. Además, diremos que el grafo es cíclico si existe alguna secuencia de arcos que permitan regresar al mismo vértice partiendo de él mismo.*

Definición 1.3. *Diremos que un grafo $G(V, E)$ es una red si los nodos y/o las aristas(arcos) tienen asociadas determinadas magnitudes.*

En particular, cada nodo del grafo puede tener asociada una etiqueta, que almacenará información relevante para ese nodo (valor/coste asociado, demanda, oferta, etc.), asimismo cada arco que conecta un nodo i con un otro j también puede tener asociado una etiqueta (coste, peso, capacidad, etc.).

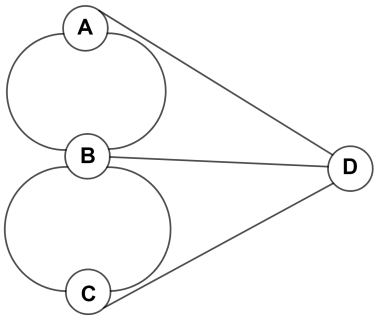


Figura 1.3: Ejemplo de grafo (Puentes de Könisberg)

Para familiarizarnos con estos conceptos, podemos destacar el conocido problema de los puentes de Könisberg, resuelto por el gran matemático Leonhard Euler en 1736, que dió origen a la actual Teoría de Grafos. Este problema nace en la actual ciudad de Kaliningrado, famosa por sus siete puentes que unen ambos lados del río Pregel con dos de sus islas. El problema se plantea de la siguiente manera: ¿es posible partir de un lugar cualquiera (nodo/vértice inicial), cruzar por todos los puentes (arcos/aristas) una única vez cada uno y regresar al mismo punto de partida?. Euler resolvió el problema demostrando que el grafo asociado al esquema de los puentes (Figura 1.3) no tiene solución. A continuación aplicaremos la *PD* en un problema clásico de Grafos, el problema de camino mínimo.

1.3.2. Problema del camino mínimo entre dos nodos

Dado un grafo $G(V, E)$ (dirigido o no), con dos nodos especiales: inicial (s_0) y final (t_0), el problema del camino mínimo entre dos nodos consiste en determinar el subconjunto de aristas en E, que permitan ir desde s_0 a t_0, visitando vértices intermedios en V, empleando el menor tiempo(coste) posible. Supongamos que el problema cuenta con n etapas, entonces, para cada etapa $i = 1, \cdots, n$, se introduce la siguiente notación:

- s_i = estado del problema en la etapa i
- $f_i(s_i, x_i)$ = contribución a la función objetivo del subproblema con estado inicial s_i, en el que se ha tomado la decisión x_i.
- x_i^* = decisión óptima en la etapa i
- $f_i^*(s_i)$= solución óptima del (sub)problema que comienza en el estado s_i de la etapa i.

Ilustremos la técnica de PD mediante el siguiente ejemplo.

Ejemplo 1
Los alumnos de Sicue de la Universidad de La Laguna han decidido realizar un tour por la zona metropolitana de Tenerife. Desean visitar lugares de interés entre San Cristóbal de La Laguna (**LL**) y Santa Cruz de Tenerife (**SC**). Sin embargo, no disponen de mucho tiempo, por lo que deben realizar la ruta más corta, sin repetir lugares ya visitados.

Así, sea $V = \{a, b, c, d, e, f, g, h, i, j\}$ el conjunto de vértices(nodos), donde a y j son los nodos inicial y final, respectivamente, del grafo dirigido de la Figura 1.4, en el que también se muestran los correspondientes arcos y distancias del conjunto E:

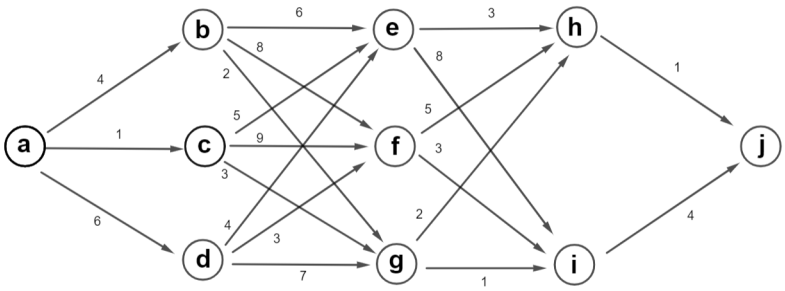


Figura 1.4: Grafo del Problema 1

A partir del grafo de la Figura 1.4, se puede apreciar que la excursión que realizarán los alumnos se divide en cuatro etapas. En este caso, partiremos del nodo final j (Santa Cruz de Tenerife) y el objetivo será hallar el camino mínimo hasta el nodo inicial a (San Cristóbal de La Laguna) mediante un método recursivo hacia atrás (*método backward*). También podríamos hacerlo mediante un método recursivo hacia adelante (*método forward*) de manera análoga pero empezando por el nodo inicial. Tenemos que:

- S = conjunto de nodos (conjunto de lugares de interés)
- $D(S)$ = decisiones posibles con origen S (lugares posibles que podemos visitar partiendo de algún lugar de interés de S,)
- $x_i \in D(S)$ = decisión tomada en la etapa i (a qué lugar de interés deciden ir en la etapa i)
- $f_i^*(s_i) = \text{mín}_{x_i \in D(S)}[f(s_i, x_i) + f_{i+1}^*(s_{i+1})]$ tiempo mínimo del camino que comienza en s_i y finaliza en j. Observe que $f(s_i, x_i)$ es el valor asociado al desplazamiento desde el lugar de interés s_i al s_{i+1}, cuando se toma la decisión x_i

Dado que se implementará el método backward, la solución óptima $f_5^*(j)$ será 0. Observe que la etapa 5 es ficticia y sólo se utiliza para fines computacionales. Ahora, iremos hacia atrás para identificar los lugares de interés en la etapa 4 (nodos h e i) desde los que podemos llegar al nodo j. Las posibles soluciones para obtener $f_4^*(S)$ vienen recogidas en la Tabla 1.2.

Estado s_4	$f_4^*(s_4)$	x_4^*
h	1	j
i	4	j

Tabla 1.2: Etapa 4

De aquí obtenemos que $f_4^*(h) = 1$ y $f_4^*(i) = 4$. Pasamos a la etapa 3, en la que los nodos desde los que podemos llegar a h, i son e, f, g. La información relevante de esta etapa se recoge en la Tabla 1.3.

$f_3^*(s_3) = \min_{x_3}\{f(s_3, x_3) + f_4^*(s_4)\}$				
Estado s_3	h	i	$f_3^*(s_3)$	x_3^*
e	4	12	4	h
f	6	7	6	h
g	3	5	3	h

Tabla 1.3: Etapa 3

Veamos un ejemplo que nos ayudará a entender cómo se calculan los valores de $f_3^*(s_3)$ en la cuarta columna de la Tabla 1.3. La entrada en esa tabla, correspondiente a la combinación de lugares de interés (nodos) (e, h), recoge la suma del tiempo de desplazamiento entre esos dos lugares (3) más el tiempo mínimo del camino que comienza en h y termina en j (1). Asimismo, en la entrada para la combinación de estados (e, i) se almacenará la suma del tiempo de desplazamiento entre ese par de lugares y el tiempo mínimo del camino que comienza en el nodo i. Por lo tanto, tenemos que $f_3^*(e) = \min\{f(e, h) + f_4^*(h), f(e, i) + f_4^*(i)\} = \min\{4, 12\} = 4$.

Para los otros valores resolvemos de manera análoga. De aquí, se tiene que $f_3^*(e) = 4, f_3^*(f) = 6$ y $f_3^*(g) = 3$. Retrocedemos ahora a la etapa 2, en la que los lugares de interés son b, c y d. Los tiempos mínimos desde cada uno de ellos se recogen en la Tabla 1.4.

$f_2^*(s_2) = \min_{x_2}\{f(s_2, x_2) + f_3^*(s_3)\}$					
Estado s_2	e	f	g	$f_2^*(s_2)$	x_2^*
b	10	14	5	5	g
c	9	15	6	6	g
d	8	9	10	8	e

Tabla 1.4: Etapa 2

Obteniendo que $f_2^*(b) = 5, f_2^*(c) = 6$ y $f_2^*(d) = 8$. Finalmente, retrocediendo, llegaríamos al único estado (lugar de interés) de la etapa 1, cuya información relevante se muestra en la Tabla 1.5.

$f_1^*(s_1) = \min_{x_1}\{f(s_1, x_1) + f_2^*(s_2)\}$					
Estado s_1	b	c	d	$f_1^*(s_1)$	x_1^*
a	9	7	14	9	c

Tabla 1.5: Etapa 1

Concluimos la búsqueda del camino de mínimo tiempo por PD asegurando que $f_1^*(a) = 9$. Interpretemos las tablas empezando por la última. Hemos obtenido que el valor mínimo en la tabla 1.5 se obtiene cuando vamos de a hacia c. Ahora, podemos determinar a qué lugar de interés en la etapa 3 debemos desplazarnos desde s viendo la tabla 1.4. Se comprueba fácilmente que debemos movernos de c a g. Luego, de g nos desplazaremos a h, según la información de la Tabla 1.3 y, por último, de h a j. En efecto, el camino mínimo es:

$$a \to c \to g \to h \to j \text{ con un costo asociado de } f_1^*(a) = 9$$

Contextualizando la solución al problema planteado, si los alumnos de Sicue de la Universidad de La Laguna quieren minimizar el tiempo de su excursión, deberán visitar en orden los lugares interés c, g, h para finalizar su trayecto en j (Santa Cruz de Tenerife).

1.4. Modelo dinámico de tamaño de lote (Wagner-Whitin)

Una vez repasados los conceptos básicos del *EOQ* clásico y de la Programación Dinámica, podemos abordar la versión dinámica del *EOQ*, tambien conocida como *planificación de pedidos/producción*, por su traducción libre del inglés (*lot sizing problems*). Este problema clásico de la Gestión de Inventarios fue introducido y bien resuelto (*polinomialmente*) por H.M. Wagner y T.M. Whitin en [25].

Para su resolución, los autores desarrollaron un algoritmo basado en Programación Dinámica con búsqueda hacia adelante (*forward*) de la solución óptima. Las características y la formulación del modelo se muestran a continuación.

¿Por qué utilizaríamos este modelo y no el modelo clásico de tamaño de lote? Se observa que cuando las cantidades demandadas en cada periodo son conocidas pero diferentes y además los costos de inventario varían de periodo en periodo, la fórmula obtenida en el *EOQ* no nos asegura una solución de costo mínima.

Por esta razón, se presenta el Modelo dinámico de tamaño de lote (Harvey M. Wagner and Thomson M. Whitin) [25], el cuál requiere una demanda conocida y dependiente del tiempo.

1.4.1. Formulación matemática del *EOQ* Dinámico

El modelo asume un horizonte temporal finito dividido en T periodos, siendo los parámetros del problema conocidos en cada uno de ellos. De manera específica, se considera que la demanda del único artículo es conocida y variable en el tiempo, así como los costes de pedido/producción y los de mantenimiento de inventario. No se permitirán roturas (escasez) y la reposición/producción se considerará instantánea. Asimismo, el modelo no fija limitaciones ni en la cantidad de reposición/producción ni en la cantidad de almacenaje.
Para cada periodo $t = 1, \cdots, T$, se introduce la siguiente notación:

- d_t = cantidad demandada en el periodo t
- h_t = coste unitario de almacenamiento entre el periodo t y $t+1$
- s_t = coste fijo de pedido en el periodo t
- x_t = cantidad pedida o tamaño de lote en el periodo t
- y_t = nivel de inventario al final del periodo t

Por lo tanto, el problema del *EOQ* Dinámico se puede formalizar como sigue:

$$EOQ\ Dinámico : min \sum_{t=1}^{T} [S_t \delta(x_t) + h_t y_t] \tag{1.3}$$

$$s.t.$$

$$y_{t-1} + x_t - y_t = d_t \qquad t = 1, \cdots, T \tag{1.4}$$

$$x_t, y_t \in \mathbb{R}^+ \qquad t = 1, \cdots, T \tag{1.5}$$

$$y_0 = y_T = 0 \qquad t = 1, \cdots, T \tag{1.6}$$

donde la función delta en (1.3) se define como

$$\delta(x_t) = \begin{cases} 0 & \text{si } x_t = 0 \\ 1 & \text{si } x_t > 0 \end{cases} \tag{1.7}$$

El problema del EOQ Dinámico consiste en determinar una política de reposición/producción $(x_1, \cdots, x_T)$ que minimice el coste total, y que deberá verificar la ecuación de balance de materiales (conservación del flujo) en (1.4), que establece que el nivel de inventario al final de un periodo coincide con el del periodo anterior más la diferencia entre la cantidad de pedido/producción y la demanda en ese periodo. Esta solución no tiene por qué ser única. Wagner y Whitin en [25] demostraron que entre las soluciones óptimas siempre existe al menos una que cumple la propiedad *Zero Inventory Order* (*ZIO*), es decir se pide/produce en un periodo t sólo cuando el nivel de inventario disponible al final del periodo anterior es cero. Esta propiedad se puede expresar matemáticamente como $y_{t-1}x_t = 0$, y permite desarrollar un algoritmo de Programación Dinámica de complejidad cuadrática (i.e., $\mathcal{O}(T^2)$).

Sin pérdida de la generalidad se puede asumir que $y_0 = y_T = 0$, mientras que para el resto de periodos el nivel de inventario final se puede calcular como:

$$y_{t-1} = y_0 + \sum_{j=1}^{t-1} x_j - \sum_{j=1}^{t-1} d_j \tag{1.8}$$

Haciendo uso de la propiedad ZIO, denotaremos ahora por $f(t, t')$ al coste del subproblema que abarca los periodos desde t hasta t', en el que se realiza un único pedido en el periodo t para satisfacer las demandas desde ese periodo hasta el $t' - 1$:

$$f(t, t') = s_t + \sum_{j=t}^{t'-2} h_j D_{j+1,t'-1} \tag{1.9}$$

donde $D_{t_1,t_2} = \sum_{j=t_1}^{t_2} d_j$ representa la demanda acumulada entre los periodos t_1 y t_2. Observe que, para cualesquiera $1 \leq t_1 \leq t_2 \leq T$, el valor D_{t_1,t_2} se computa en tiempo constante si previamente se calculan en $\mathcal{O}(T)$ las demandas acumuladas $D_{j,T}$, para $j = 1, \cdots, T$, ya que $D_{t_1,t_2} = D_{t_1,T} - D_{t_2+1,T}$.

En este punto podemos definir el coste mínimo del (sub)problema que comienza en el periodo t y finaliza en T como sigue:

$$F(t) = \min_{t<t' \leq T+1} \{f(t, t') + F(t')\} \tag{1.10}$$

donde $F(T + 1) = 0$ para el periodo ficticio $T + 1$ y donde la solución óptima se obtiene al resolver $F(1)$.

Es fácil concluir que una implementación directa de este algoritmo lleva a una complejidad temporal de $\mathcal{O}(T^2)$. No obstante, existen algoritmos más eficientes que resuelven el problema en orden $\mathcal{O}(T \log T)$, como aquellos propuestos por Aggarwal y Park [1], Wagelmans et al. [22] y Federgruen y Tzur [7].

1.5. Algoritmo Branch and Bound

Introduciremos el algoritmo Branch and Bound (*B&B*), también conocido como Ramificación y Poda/Acotación. Este método se plantea para resolver problemas de programación lineal entera mixta (Mixed-Integer Linear Programming, *MILP*).

Partiremos de un problema de máximo (o mínimo) con restricciones lineales y variables que pueden ser enteras (mixto). Se tiene que la formulación del problema lineal entero la denotaremos como:

$$MILP : \{\text{máx}\, cx + hy,\ (x, y) \in S\} \tag{1.11}$$

donde $S := \{(x, y) \in \mathbb{Z}^n \times \mathbb{R}^p : Ax + Gy \leq b\}$

De manera análoga se podría estudiar el mínimo, pues se tiene que máx $Z =$ mín$(-Z)$, siendo $Z = cx + hy$ la función objetivo asociada al problema.

Ahora, sea $P^0 := \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^p : Ax + Gy \leq b\}$ el poliedro asociado al problema relajado de (1.11), que denotaremos por LP_0 y que se obtiene eliminando la restricción de integralidad de las variables x. Además, sean (x^0, y^0) y z^0 la solución óptima y el valor óptimo de LP_0, respectivamente. Supongamos que $(x^0, y^0) \notin S$, por lo tanto debe existir al menos un $j : 1 \leq j \leq n$ tal que $f := x_j^0 \notin \mathbb{Z}$, lo que permitirá la *ramificación* del problema relajado en dos nuevos problemas. Para ello, definiremos una partición del conjunto S, formada por dos subconjuntos S_1 y S_2, de manera que

$$S_1 := S \cap \{(x, y) : x_j \leq \lfloor f \rfloor\} \tag{1.12}$$
$$S_2 := S \cap \{(x, y) : x_j \geq \lceil f \rceil\} \tag{1.13}$$

donde $\lfloor f \rfloor, \lceil f \rceil$ denotan, respectivamente, la parte entera inferior y superior de f y además $S_1 \cup S_2 = S$.

La partición del conjunto factible entero S en dos subconjuntos permite definir dos nuevos problemas enteros $MILP_1$ y $MILP_2$ para los elementos de S_1 y S_2, respectivamente. Obviamente, la solución del MILP original será la mejor de las soluciones óptimas de $MILP_1$ *y* $MILP_2$, para los que se definen los siguientes poliedros relajados, a partir de P^0:

$$P^1 := P^0 \cap \{(x, y) : x_j \leq \lfloor f \rfloor\} \tag{1.14}$$
$$P^2 := P^0 \cap \{(x, y) : x_j \geq \lceil f \rceil\} \tag{1.15}$$

En cada uno de estos problemas se procede como se hizo con el problema *padre* (LP_0). Nótese que la solución óptima no entera de cualquiera de los problemas *hijos* representa una *cota superior* (UB) para el problema MILP original.

Por el contrario, si la solución óptima es factible (entera) para el *MILP* original, obtendríamos una nueva *cota inferior* (LB) para ese problema. En caso de que la nueva LB mejore (sea mayor que) la actual candidata, ésta última se deberá actualizar con el valor de la nueva. Si en la exploración del árbol de problemas que genera esta técnica se diera el caso de que la cota superior de alguno de ellos fuese inferior que la cota superior candidata (hasta ese momento), entonces se procedería a *podar* esa rama, ya que podemos asegurar que ningún problema hijo de ese problema tendría mayor solución que la actual candidata. Otra situación que puede darse en la resolución de un problema hijo es que fuera *no factible*, lo que indicaría que la región factible (P^i, para algún subproblema i) asociada a ese problema relajado es vacía.

A partir de (1.14) y (1.15), se pueden definir los correspondientes problemas lineales, hijos de LP_0:

$$LP_1 : max\{cx + hy : (x, y) \in P^1\} \tag{1.16}$$
$$LP_2 : max\{cx + hy : (x, y) \in P^2\} \tag{1.17}$$

A continuación presentamos el pesudocódigo del método de Ramificación y Poda:

- **PASO 1.** Si alguno de los problemas lineales LP_i cumple $P^i = \emptyset$, entonces $S_i = \emptyset$. Por lo tanto, $MILP_i$ será infactible y lo descartaríamos. En este caso diremos que podaremos esta rama por ser no factible.
- **PASO 2.** Sea (x^i, y^i) una solución optima de LP_i y z_i el costo asociado, con $i = 1, 2$. Luego:
  - **PASO 2.1**. Si x^i es un vector entero entonces es solución de $MILP_i$ y una solución factible de MILP. Diremos que $MILP_i$ está resuelto y podaremos. Además, z_i es una cota inferior del valor óptimo del MILP original.
  - **PASO 2.2**. Si x^i no es un vector entero y z_i es más pequeño o igual de la mejor cota inferior conocida del MILP, entonces S_i no puede contener una mejor solución y por lo tanto podaremos.
  - **PASO 2.3**. Si x^i no es un vector entero y z_i es mejor que la cota inferior conocida entonces S_i podría contener una solución óptima para el MILP. Ahora, consideramos x^i_j una componente no entera del vector x^i y sea $f := x^i_j$. Se obtienen los siguientes conjuntos:
    $S_{i_1} := S_i \cap \{(x, y) : x_j \leq \lfloor f \rfloor\}$, $S_{i_2} := S_i \cap \{(x, y) : x_j \geq \lceil f \rceil\}$
    Con estos nuevos conjuntos repetimos el proceso.

Ilustremos el algoritmo con un ejemplo. Se considera el siguiente problema entero:

$$MILP : \text{máx}\, 5.5x_1 + 2.1x_2 \tag{1.18}$$
$$-x_1 + x_2 \leq 2$$
$$8x_1 + 2x_2 \leq 17$$
$$x_1, x_2 \geq 0$$
$$x_1, x_2 \text{ enteros}$$

En primer lugar debemos obtener una solución del problema relajado, por ejemplo, utilizando el método Simplex (no lo abarcaremos en esta memoria). Se obtiene $x_1 = 1.3$ y $x_2 = 3.3$ con valor objetivo y cota superior de la solución óptima del problema $z = 14.08$. Tenemos:

P^0
$x_1 = 1.3\ ,\ x_2 = 3.3$
$z = 14.08$

Tabla 1.6

Ahora, como $x_1 = 3.3$ no es entero, ramificamos tomando nuevas restricciones. Añadiendo $x_1 \leq \lfloor 3.3 \rfloor = 3$ obtenemos:

P^1
$x_1 = 1\ ,\ x_2 = 3$
$z = 11.8$

Tabla 1.7

Por otro lado, añadiendo la restricción $x_1 \geq \lceil 3.3 \rceil = 4$ se tiene:

P^2
$x_1 = 2\ ,\ x_2 = 0.5$
$z = 12.05$

Tabla 1.8

Ahora, en P^0 tenemos dos variables enteras luego se toma como nueva cota inferior $z = 11.8$ (**PASO 2.1**). Por otro lado, dado que tenemos una variable racional $x_2 = 0.5$ y un costo superior a la cota inferior en P^1 debemos continuar ramificando (**PASO 2.3**).

Añadiremos las nuevas restricciones correspondientes. Por un lado consideramos el nuevo problema P^{22} considerando $x_2 \geq 1$. Sin embargo, se tiene que es no factible, por lo tanto, podamos esta rama. No obstante, se considera P^{21} añadiendo $x_2 \leq 0$. Se tiene:

P^{21}
$x_1 = 2.125\ ,\ x_2 = 0$
$z = 11.6875$

Tabla 1.9

Observamos que tenemos una variable racional y un costo asociado menor que la cota inferior, por lo tanto, podamos (**PASO 2.2**). En efecto, terminamos el algoritmo y se tiene como solución óptima $(x^*, y^*) = (1, 3)$ y costo asociado óptimo $z^* = 11.8$ para el problema de programación lineal entera 1.18.

Finalmente, tenemos el siguiente esquema para simplificar la ramificación.

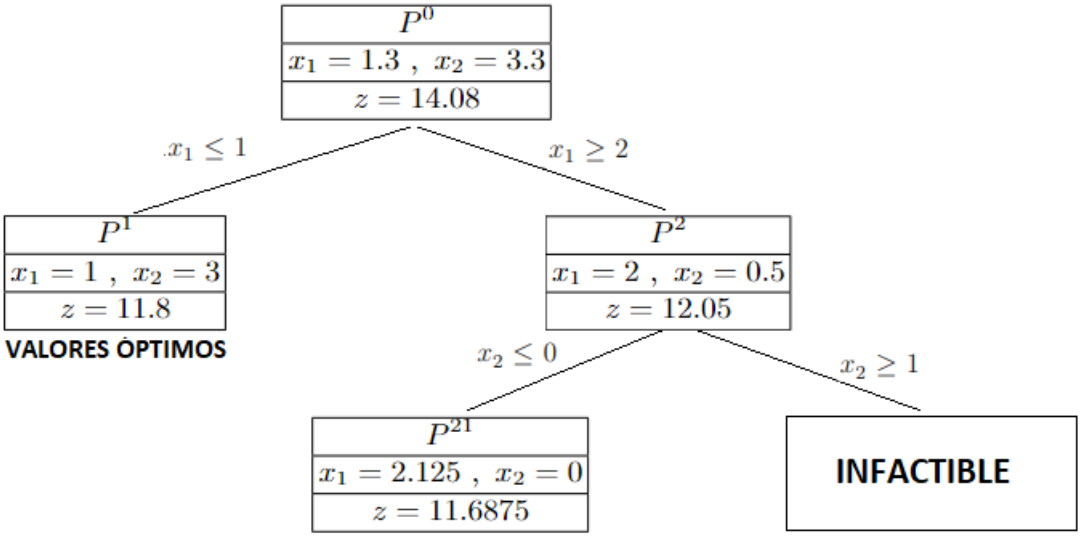


Figura 1.5: Problema ramificado

Con esto, ya hemos ilustrado los conceptos básicos para adentrarnos de lleno en el segundo capítulo, donde abordaremos varios metodos de resolución para la planificación de la reposición conjunta de artículos.

2

Modelos para planificar la reposición conjunta de múltiples artículos

2.1. Introducción

En el capítulo anterior hemos introducido conceptos básicos en el ámbito de la Gestión de Inventarios como el Modelo *EOQ* (1.1) y la versión dinámica del Modelo *EOQ* (1.4). Ahora, nos centraremos mayormente en la generalización de los modelos clásicos de reposición conjunta de varios artículos con costes de reposición marginales y conjuntos.

En la literatura especializada en gestión de inventarios podemos encontrar diversos modelos que consideran varios artículos. En la mayoría de ellos el objetivo consiste en decidir las cantidades óptimas de distintos productos a pedir al mismo proveedor, o bien cuándo conviene repartir la distribución de un pedido de producción en diferentes lotes. Nos referiremos a este problema como Planificación de la Reposición Conjunta de Varios Artículos, o *JRP* por sus siglas en inglés (Joint Replenishment Problem), y en esta memoria nos centraremos en este tipo de problemas.

Las principales características de esta familia de modelos multi-artículos se enumeran a continuación:

- **Economías de escala:** combinando varios artículos en un lote se puede aprovechar economías de escala en producción, transporte y gestión de los inventarios, esto es, se logra producir mayor cantidad de productos a menor coste marginal por unidad de producto.
- **Reducción de costos de pedido:** agrupando varios artículos en un lote podemos reducir costes asociados con la reposición de pedidos.
- **Flexibilidad en la gestión de inventarios:** los modelos de varios artículos proporcionan mayor flexibilidad al permitir ajustes dinámicos en la cantidad de cada artículo en un lote en función de su demanda, simplificando la planificación del inventario.

Asimismo, en la práctica, los mercados no suelen trabajar con un solo artículo por lo que estos modelos son de gran aplicabilidad en la gestión de inventarios.

2.1.1. Revisión literaria de los Modelos Dinámicos de Tamaño de Lote para Varios Artículos

Desde mediados de la década de los 60 del siglo pasado, se ha estudiado en profundidad el modelo *JRP*, proponiéndose diversas maneras de resolver el problema. Los métodos más destacados son los heurísticos (Silver, 1976) [19], métodos alternativos como la política 'Power-Of-Two' (PoT) (Lee and Yao, 2003) [13], computación evolutiva (Olsen, 2005) [15], *JRP* bajo demanda estocástica [10] y *JRP* bajo demanda dinámica [4]. Especial relevancia tienen los algoritmos propuestos en [23] por Webb en 1997 y los más recientes presentados en [16] por Robinson en 2007.

Destacamos también las labores hechas por O. Kirca [12], quién propone un método interactivo donde se escoge un producto en base a un estricto criterio de selección y el heurístico de observación para varios artículos (One-Way Eyeballing Heuristic, OWEH), (E.U. Choo, G.H. Chan) [5], el cuál simplemente requiere de la comparación visual de un número predeterminado de secuencias de valores de corte con la secuencia de demandas conocidas.

Es conocido además que Zangwill [26] propone una formulación de Programación Dinámica, la cuál usa el periodo como espacio de estado y los niveles de inventario de todos los productos en cada periodo como la variable de estado. Por otro lado, Edward P.C.Kao [11] trabaja con una red acíclica para resolver el problema como un problema de camino mínimo. Más recientemente, S.S Ergenguc [6] presentó un algoritmo Branch and Bound en el cuál se introduce un método heurístico para la obtención de la solución factible y la cota superior del problema óptimo.

2.1.2. Formulación matemática del modelo *JRP*

En adelante consideraremos las siguientes suposiciones. La demanda se asume conocida con patrón de demanda uniforme, no se permitirán roturas y el coste de mantenimiento será lineal. Asimismo, seguiremos la siguiente notación:

- T = tiempo entre reposiciones sucesivas (años)
- S = coste fijo de reposición conjunta (€/pedido)
- TC = coste anual total de mantenimiento y reposición para todos los productos (€/ año)
- n = número de artículos
- i = índice del artículo i-ésimo, $i = 1, \cdots, n$
- D_i = demanda anual del producto i-ésimo (unidades/año)
- h_i = coste anual de mantenimiento del producto i-ésimo (€/unidad/año)
- s_i = coste marginal de pedido que se activará si se incluyen unidades del producto i-ésimo en la reposición conjunta (€/pedido)
- Q_i = cantidad de pedido del producto i-ésimo

- T_i = intervalo de tiempo entre sucesivas reposiciones del producto i-ésimo (años)

En general, las estrategias que se siguen para resolver el *JRP* se clasifican en dos tipos: una estrategia de agrupación directa (Direct Grouping Strategy, abreviado *DGS*) y una estrategia de agrupación indirecta (Indirect Grouping Strategy, *IGS*).

En la estrategia *DGS*, los productos son divididos en un número predeterminado de conjuntos y los productos de cada conjunto se reponen conjuntamente en el mismo periodo. Por otro lado, bajo la estrategia *IGS*, se realiza un reabastecimiento en intervalos de tiempo regulares y cada producto tiene una cantidad de reposición suficiente como para abastecer exactamente un múltiplo entero de el intervalo de tiempo regular.

Los estudios sugieren que la táctica *IGS* supera a *DGS* en cuanto a costes de pedidos dado que muchos productos se pueden reponer conjuntamente bajo la estrategia *IGS* (van Eijs et al., 1992) [9].

En efecto, veamos como formular el problema mediante la estrategia *IGS*. Bajo esta política, el periodo de tiempo de cada artículo i será un múltiple entero k_i del periodo de reposición T. Así, el ciclo de tiempo para el producto i es

$$T_i = k_i T \tag{2.1}$$

y el tamaño de lote i-ésimo es

$$Q_i = T_i D_i = T k_i D_i \tag{2.2}$$

Por otro lado, los costos anuales de mantenimiento serán

$$C_H = (\sum_{i=1}^{n} Q_i h_i)/2 = \frac{T}{2} \sum_{i=1}^{n} k_i D_i h_i \tag{2.3}$$

Además, los costos anuales de pedido serán

$$C_0 = \frac{S}{T} + (\sum_{i=1}^{n} s_i)/k_i T = (S + \sum_{i=1}^{n} s_i/k_i)/T \tag{2.4}$$

En la ecuación (2.4), los ciclos sin reposiciones (p.e $k_i \geq 2, i = 1, \cdots, n$) se incluyen en el costo principal de reposición S. Finalmente, los costos anuales totales serán

$$TC(T, K) = C_H + C_0 = \frac{T}{2} \sum_{i=1}^{n} k_i D_i h_i + (S + \sum_{i=1}^{n} s_i/k_i)/T \tag{2.5}$$

donde K es un conjunto de múltiplos enteros. Esta política, basada en un ciclo básico de tiempo y un conjunto de múltiplos enteros K es conocida como política cíclica.

De esta manera, para $K = (k_1, \cdots, k_n) \in \mathbb{N}^n$), la solución óptima del problema viene dada por:

$$\min_{T>0} TC(T, K) = T(K) = [2(S + \sum_{i=1}^{n} s_i/k_i)/\sum_{i=1}^{n} D_i h_i k_i]^{1/2} \tag{2.6}$$

2.1.3. Nuestra contribución

En las siguientes secciones de este capítulo abordaremos el Modelo de Kao, propuesto en [11], así como una formulación más robusta (*stronger*) del mismo, de forma que nos permita usar varios métodos de resolución. De esta manera, transformaremos la formulación del Problema de Planificación de Pedidos con Múltiples Productos (o *MPLS*, por sus siglas en inglés) a un Problema de Programación No Entera Mixta (Mixed Integer Non Linear Programming, *MINLP*). En efecto, podemos aplicar diferentes técnicas de resolución para nuestra formulación alternativa.

Dado que se trata de un problema *NP-Duro* (ver en 1.2.5), propondremos algoritmos de resolución eficientes que se ajusten al problema. En primer lugar tomaremos la formulación propuesta por A.F.Veinott en [21] donde se generarán todos los posibles patrones de producción conjunta. En segundo lugar, haremos uso de la Programación Dinámica y finalmente, dado que para un número elevado tanto de artículos como de periodos el algoritmo de Programación Dinámica resulta intratable, se propone el algoritmo Branch and Bound para resolver el *MILNP* junto con un sencillo ejemplo donde usaremos un código en Python para ilustrar el método.

2.2. Modelos clásicos de reposición de inventario con múltiples artículos

Ahora, veremos la formulación propuesta por Edward P.C.Kao [11] y posteriormente se introducirán varios métodos de resolución.

2.2.1. Formulación general: Modelo de Kao

En lo que sigue, estudiaremos el Modelo de Planificación de la Reposición Conjunta de Múltiples Artículos con costos de pedido unitarios y conjuntos, introducido en [11]. Veremos que los prodecimientos para alcanzar la política óptima se verán afectados por el número de artículos y el número de periodos, pues para valores grandes tendremos que utilizar métodos más eficientes para resolver el problema.

Por conveniencia, usaremos la notación empleada en (Kao, 1979) [11]. Para cada periodo $t \in \{1, \cdots, T\}$ y para cada artículo $k \in \{1, \cdots, K\}$, introduciremos

las 3 siguientes categorías: parámetros en la Tabla 2.1, variables en la Tabla 2.2 y funciones en la Tabla 2.3.

Parámetros	
$d_t^k \geq 0$	= demanda conocida del producto k en el periodo t.
S_t	= coste de reposición conjunta en el periodo t, el cual se activará cuando se repone (o produce) al menos uno de los artículos en el periodo t.

Tabla 2.1: Parámetros del problema.

Variables	
x_t^k	= cantidad de pedido para el artículo k en el periodo t.
y_t^k	= nivel de inventario para el artículo k al final del periodo t.

Tabla 2.2: Variables del problema.

Funciones	
$C_t^k(x_t^k)$	= función coste de reponer/producir x_t^k unidades del producto k en el periodo t.
$H_t^k(y_t^k)$	= función coste de mantener y_t^k unidades del producto k en el inventario al final del periodo t.
$\delta(q)$	= función delta que toma los valores $\delta(0) = 0$ y $\delta(q) = 1$, para $q \neq 0$.

Tabla 2.3: Funciones del problema.

Para cada periodo t, las funciones de reposición/producción y mantenimiento de la Tabla 2.3 se asumen como funciones cóncavas no decrecientes en $[0, \infty)$, donde $C_t^k(0) = H_t^k(0) = 0$. Esta suposición modela el comportamiento clásico de cualquier actividad económica en el contexto de las economías de escala, donde el coste marginal de una actividad (e.g., reposición o mantenimiento) no aumenta como resultado de un crecimiento marginal en esa actividad económica (p.e. un aumento marginal en la cantidad de reposición o en el nivel de inventario). En adelante se tendrán en cuenta las siguientes consideraciones del modelo:

- Las demandas d_t^k son constantes conocidas
- No se admiten roturas (no se permite stock negativo)
- El tiempo de entrega es insignificante (reposición instantánea)
- Los inventarios al principio y al final del horizonte temporal para cada producto $k = 1, \cdots, K$ es cero, denotado como $y_0^k = y_T^k = 0$

Así, el modelo *MPLS* presentado por Edward P.C. Kao se formularía como sigue:

$$P : min \sum_{t=1}^{T} [S_t \delta(\sum_{k=1}^{K} x_t^k) + (\sum_{k=1}^{K} C_t^k(x_t^k) + H_t^k(y_t^k))] \tag{2.7}$$

$s.t.$

$$y_{t-1}^k + x_t^k - y_t^k = d_t^k \qquad t = 1, \cdots, T;\ k = 1, \cdots, K \tag{2.8}$$

$$x_t^k, y_t^k \in \mathbb{R}^+ \qquad t = 1, \cdots, T;\ k = 1, \cdots, K \tag{2.9}$$

$$y_0^k = y_T^k = 0 \qquad k = 1, \cdots, K \tag{2.10}$$

Dadas las demandas para los K productos en los T periodos, $D = (d_1^1, \cdots, d_T^1; \cdots; d_1^K, \cdots, d_T^K)$, el objetivo en P será encontrar una política óptima de reposición $X = (x_1^1, \cdots, x_T^1; \cdots; x_1^K, \cdots, x_T^K)$ que minimice la función de costo total (2.7).

Las restricciones consideradas en (2.8) corresponden a las ecuaciones de equilibrio de material (conservación del flujo). Además, las variables del problema son no negativas, tal y como tenemos en (2.9) y (2.10).

Debemos destacar que la función objetivo (2.7) es cóncava pues es suma de funciones cóncavas. Por lo tanto, aplicando el resultado en [3], existen soluciones óptimas de P que pueden ser encontradas en los puntos extremos del poliedro convexo ($\mathcal{P}$) definido por el conjunto de restricciones (2.8)-(2.10). Por otro lado, sea $A \in \mathcal{M}_{KT \times 2KT}(\mathbb{Z})$ la matriz formada por los coeficientes del lado izquierdo de las restricciones de (2.8). Es un resultado bien conocido que A es *totalmente unimodular*, por lo que, siempre y cuando la demanda de cada periodo y producto sea entera, cada solución extrema (o vértice de $\mathcal{P}$) será entera. Conviene recordar en este punto que una matriz $A \in \mathcal{M}_{m \times n}(\mathbb{Z})$ es totalmente unimodular si para cualquiera submatriz cuadrada B de A se tiene que $|B| \in \{0, 1, -1\}$

2.2.2. Caso especial: estructura fija lineal de costos

Un caso especial en la práctica es cuando $C_t^k(0) = 0, C_t^k(x) = s_t^k + c_t^k x$, para $x > 0$, y $H_t^k(y) = h_t^k y$, donde s_t^k, c_t^k, y h_t^k son el coste fijo de activación de pedido (set-up cost), el coste unitario de reposición/producción y el coste unitario de mantenimiento para el producto k en el periodo t, respectivamente. Por conveniencia, denotaremos por $D_{t,t'}^k = \sum_{i=t}^{t'} d_i^k$ la demanda acumulada desde el periodo t hasta el periodo t' para el producto k.

Tanto Zangwill en [26] como Veinott en [21] se apoyaron en la caracterización de las soluciones extremas, descrita en la Propiedad 2.1, y utilizaron técnicas diferentes para encontrar soluciones óptimas para P, incluso para el caso con roturas.

Propiedad 2.1. (**Solución tipo *ZIO***) Existe solución óptima para P tal que $y_{t-1}^k x_t^k = 0$, para todo t y k. Coloquialmente, esta propiedad establece que siempre existe una política de costo mínimo teniendo en cuenta que x_t^k tiene uno de los siguientes valores: $0, D_{t,t}^k, D_{t,t+1}^k, \cdots, D_{t,T}^k$.

Aprovechando la Propiedad 2.1, Zangwill desarrolló una formulación basada en programación dinámica que usa el periodo como variable de etapa y el inventario de inicio del periodo para los K artículos como variable de estado. Por otro lado, en el procedimiento de Veinott, se consideran 2^{T-1} posibles patrones de reposición y, tras resolver de manera independiente K problemas uniartículo para cada patrón (usando algoritmos discutidos en la sección 1.4), el procedimiento elige el de menor coste. Este último procedimiento requiere considerablemente menos espacio de almacenamiento en un ordenador. Sin embargo, enumerar 2^{T-1} patrones puede resultar intratable para valores grandes de T.

2.2.3. Formulación alternativa para el MPLS

Cabe destacar que la función delta en la formulación del problema P puede ser reemplazada usando variables binarias auxiliares como sigue:

$$P' : min \sum_{t=1}^{T} [S_t z_t + \sum_{k=1}^{K} (C_t^k(x_t^k) + H_t^k(y_t^k))] \tag{2.11}$$

$s.t.$

$$y_{t-1}^k + x_t^k - y_t^k = d_t^k \qquad t = 1, \cdots, T;\ k = 1, \cdots, K \tag{2.12}$$

$$\sum_{k=1}^{K} x_t^k \leq \sum_{k=1}^{K} D_{t,T}^k z_t \qquad t = 1, \cdots, T \tag{2.13}$$

$$x_t^k, y_t^k \in \mathbb{R}^+ \qquad t = 1, \cdots, T;\ k = 1, \cdots, K \tag{2.14}$$

$$z_t \in \{0, 1\} \qquad t = 1, \cdots, T \tag{2.15}$$

$$y_0^k = y_T^k = 0 \qquad k = 1, \cdots, K \tag{2.16}$$

Claramente, la formulación de P' corresponde ahora a un Problema de Programación No Lineal Entera Mixta (*MINLP*), donde z_t en (2.13) tomará valor 1 cuando se reponga al menos una unidad de algún producto en el periodo t y $z_t = 0$ en otro caso. Observamos que hemos obtenido una formulación de P más robusta (*stronger*), esto es, $P' \subset P$ tal que esta alternativa del problema inicial

proporciona una mejor cota y facilita la búsqueda de la solución óptima. Las formulaciones en general pueden ser mejoradas a priori o bien dinámicamente, añadiendo cortes o variables adicionales. De hecho, reducimos el tiempo invertido en resolver el algoritmo.

Además, la Propiedad 2.1 sigue siendo válida para nuestra nueva formulación (incluso cuando la estructura de costes en (2.11) es más general que la estructura fija lineal de costos).

Para demostrar esto último, asumiremos X una solución óptima para P', pero X no es *ZIO*. De este modo, debería haber al menos un periodo t en X tal que $y_{t-1}^k x_t^k \neq 0$ para algún producto k. Ahora, sea $\hat{t}$ el periodo de producción anterior a t en X y sea X_1 una solución factible alternativa a X, donde una cantidad extra $q = \text{mín}\{y_{t-1}^k, x_t^k\}$ se produce en el periodo $\hat{t}$ y se sustrae de la producción del periodo t. Por otro lado, sea X_2 otra solución factible obtenida de X reduciendo la producción en el periodo $\hat{t}$ en q unidades, que serán añadidas al pedido en el periodo t.

Se observa que $X = (X_1 + X_2)/2$. Por ello, X no es una solución extrema. Además, la solución X es óptima y por [3], X_1 y X_2 son soluciones óptimas donde una de ellas (dependiendo del valor de q) verifica la propiedad *ZIO*. Para ser más preciso, si $q = y_{t-1}^k$ entonces X_2 es *ZIO* y cuando $q = x_t^k$ entonces X_1 es *ZIO*.

Haciendo uso de la propiedad 2.1, se puede obtener una formulación más robusta (*stronger*) que la del problema P' introduciendo nuevas variables. De este modo, para cada par de periodos $1 \leq t < \bar{t} \leq T+1$ y para cada producto k, sea $z_{t,\bar{t}}^k$ una variable binaria tal que $z_{t,\bar{t}}^k = 1$ cuando se realizan dos pedidos consecutivos en los periodos t y $\bar{t}$ para el producto k, y $z_{t,\bar{t}}^k = 0$ en otro caso. Si $z_{t,T+1}^k = 1$ entonces t será el último (o el único) periodo de producción para el producto k en la política. Observamos que si $z_{t,\bar{t}}^k = 1$ entonces la demanda del artículo k para los periodos entre t y $\bar{t}$ (ambos inclusive) está cubierta por la producción del periodo t. Estas variables permiten reformular el *MPLS* (o P'' para abreviar) de la siguiente manera:

$$P'' : min \sum_{t=1}^{T} [S_t z_t + \sum_{k=1}^{K} (C_t^k(x_t^k) + H_t^k(y_t^k))] \tag{2.17}$$

s.t.

$$y_{t-1}^k + x_t^k - y_t^k = d_t^k \qquad t = 1, \cdots, T;\ k = 1, \cdots, K \tag{2.18}$$

$$x_t^k \leq D_{t,\bar{t}-1}^k z_{t,\bar{t}}^k \qquad 1 \leq t < \bar{t} \leq T+1; k = 1, \cdots, K \tag{2.19}$$

$$\sum_{k=1}^{K} z_{t,\bar{t}}^k \leq K z_t \qquad 1 \leq t < \bar{t} \leq T+1 \tag{2.20}$$

$$x_t^k, y_t^k \in \mathbb{R}^+; z_{t,\bar{t}}^k \in \{0,1\} \qquad 1 \leq t < \hat{t} \leq T+1;\ k = 1, \cdots, K \tag{2.21}$$

$$z_t \in \{0,1\} \qquad t = 1, \cdots, T \tag{2.22}$$

$$y_0^k = y_T^k = 0 \qquad k = 1, \cdots, K \tag{2.23}$$

Se tiene que, incorporando las nuevas variables $z_{t,\bar{t}}^k$, la formulación de P'' se corresponde con un Problema de Flujo de Costos Mínimo (Minimum Cost Flow Problem, *MCFP*) o bien un Problema de Localización (Uncapacitated Facility Location Problem, *UFLP*) bajo una red generalizada $G(V, E)$ sin capacidades.

Sea $V = \{0, 1^1, \cdots, T^1, (T+1)^1; \cdots; 1^K, \cdots, T^K, (T+1)^K, F\}$ el conjunto de vértices del grafo, donde 0 es el vértice de partida y F el vértice final. Además, sea $E = \{(t^k, \bar{t}^k) \mid t^k, \bar{t}^k \in V, 1 \leq t < \bar{t} \leq T+1\ ,\ k = 1, \cdots, K\} \cup \{(0, 1^k) \mid k = 1, \cdots, K\} \cup \{((T+1)^k, F) \mid k = 1, \cdots, K\}$ el conjunto correspondiente de los arcos de la red.

Así, el costo asociado a cada arco viene dado de la siguiente manera: para los arcos del tipo $(0, 1^k)$ y $((T+1)^k, F)$, los costes $\mathcal{C}_{0,1}^k$ y $\mathcal{C}_{T+1,F}^k$ son 0, para todo k, mientras que para los arcos $(t^k, \bar{t}^k)$ el coste será $\mathcal{C}_{t,\bar{t}}^k = C_t^k(D_{t,\bar{t}-1}^k) + H_{t,\bar{t}}^k$, donde $H_{t,\bar{t}}^k$ se puede computar recursivamente de acuerdo a la siguiente fórmula:

$$H_{t,\bar{t}}^k = H_t^k(D_{t+1,\bar{t}-1}^k) + H_{t+1,\bar{t}}^k$$

con $H_{t,t}^k = 0$ para $t = 1, \cdots, T$ y $k = 1, \cdots, K$. Notemos que todos los valores $H_{t,\bar{t}}^k$ pueden ser determinados en un tiempo $\mathcal{O}(KT^2)$ (ver en 1.1). Por lo tanto, denotando $E^- = E \setminus \{(0, 1^k) \cup ((T+1)^k, F)\}$ con $k = 1, \cdots, K$, el problema *MPLS* puede ser formulado como un problema *MCFP* como sigue:

$$MCFP: min \sum_{t=1}^{T} S_t z_t + \sum_{(t^k, \bar{t}^k) \in E} \mathcal{C}_{t,\bar{t}}^k z_{t,\bar{t}}^k \tag{2.24}$$

$s.t.$

$$z_{0,1}^k = 1 \qquad k = 1, \cdots, K \tag{2.25}$$

$$z_{T+1,F}^k = 1 \qquad k = 1, \cdots, K \tag{2.26}$$

$$\sum_{(t^k, \bar{t}^k) \in E} z_{t,\bar{t}}^k = \sum_{(\bar{t}^k, t^k) \in E} z_{\bar{t},t}^k \qquad k = 1, \cdots, K \tag{2.27}$$

$$z_{t,\bar{t}}^k \leq z_t \qquad (t^k, \bar{t}^k) \in E^- \tag{2.28}$$

$$z^k_{t,\bar{t}} \in \{0,1\} \qquad (t^k, \bar{t}^k) \in E^- \qquad (2.29)$$

$$z_t \in \{0,1\} \qquad t = 1, \cdots, T \qquad (2.30)$$

Dado que el problema *MPLS* inicial lo podemos transformar en un problema *MCFP* o *UFLP*, se tiene que es computacionalmente tan complicado como estos dos problemas, por lo que el *MPLS* es también *NP-duro*.

2.3. Métodos de resolución

2.3.1. Patrones de producción conjunta

Este método fue propuesto por A.F. Veinott en [21]. Teniendo en cuenta que el periodo 1 será siempre un punto de partida, la clave de este algoritmo es generar todos los 2^{T-1} patrones de producción conjunta $\mathbf{p} = (p_1, \cdots, p_T)$, donde $p_t \in \{0,1\}$ para $t = 1, \cdots, T$. Claramente, $p_t = 1$ significa que el periodo t es un punto de regeneración (periodo en el que se repone al menos un artículo) y $p_t = 0$ en otro caso. Luego, para cada patrón, se tienen K problemas de tipo *ELSP* (Problema de Planificación del Lote Económico, abreviado del inglés como *ELSP*) independientes que pueden ser resueltos en un tiempo $\mathcal{O}(KT \log T)$ usando algoritmos de Programación Dinámica y técnicas geométricas. (Mirar [22]).

En consecuencia, este método se ejecuta en tiempo $\mathcal{O}(2^{T-1}KT \log T)$, pues tenemos 2^{T-1} patrones de producción conjunta. Consideramos f^k_t el costo mínimo de *ELSP* para el producto k, comenzando en el periodo t y terminando en el periodo T. Por ello, se obtiene $f^k_{T+1} = 0$ de manera recursiva para k,$\mathbf{p}$ fijos como sigue:

$$f^k_t = \begin{cases} \min\limits_{\bar{t}=t+1,\cdots,T+1} \{\mathcal{C}^k_{t,\bar{t}} + f^k_{\bar{t}}\} & \text{si } t \text{ es un punto de regeneración en } \mathbf{p} \\ \infty & \text{otro caso} \end{cases}$$

Además, se consigue una solución óptima resolviendo el siguiente problema

$$\min_{(1,p_2,\cdots,p_T)} \{\sum_{t=1}^{T} S_t p_t + \sum_{k=1}^{K} f^k_1\} \qquad (2.31)$$

2.3.2. Algoritmo de Programación Dinámica

Sea $\mathbf{w}_t = (w^1_t, \cdots, w^K_t)$ el patrón de inventario al principio del periodo t (o equivalentemente al final del periodo $t-1$), donde $w^k_t \in \{0,1\}$ para cada producto $k = 1, \cdots, K$. Por ello, el nivel de inventario al final del periodo t

para el producto k estará completamente agotado cuando $w_t^k = 0$, mientras que si $w_t^k = 1$ entonces quedarán unidades en stock del artículo k al comienzo del periodo t.

Observamos que se pueden generar 2^K patrones de inventario en cada periodo. Por otro lado, consideramos $f_t(\mathbf{w}_t)$ la función de coste óptima para el problema *MPLS* comenzando en el periodo t y acabando en el periodo $T+1$, con un patrón de inventario inicial $\mathbf{w}_t$. Se considera además el conjunto de los índices de los productos $\mathcal{K} = \{1, \cdots, K\}$.

Ahora, dado un periodo t, sea $\mathcal{K}_0(\mathbf{w}_t) \subseteq \mathcal{K}$ un subconjunto de $\mathcal{K}$ incluyendo aquellos productos k tales que $w_t^k = 0$, esto es, t es un periodo de regeneración para todos los productos en $\mathcal{K}_0(\mathbf{w}_t)$. Por simplicidad, sea $I(t, \bar{t}) = \mathcal{K}_0(\mathbf{w}_t) \cap \mathcal{K}_0(\mathbf{w}_{\bar{t}})$ la intersección del conjunto de los índices de los productos que deberían ser pedidos en ambos periodos, a saber t y $\bar{t}$, de acuerdo con sus respectivos patrones de nivel de inventario.

Asimismo, denotando por $\mathbf{0}$ al vector nulo en $\mathbb{R}^K$, tendremos además que $f_{T+1}(\mathbf{0}) = 0$ y $f_{T+1}(\mathbf{w}_{T+1}) = \infty$, para todo patrón $\mathbf{w}_{T+1} \neq \mathbf{0}$, por lo que la fórmula recursiva puede ser expresada como sigue, para $t = 1, \cdots, T$.

$$f_t(\mathbf{w}_t) = \begin{cases} z_t + \min\limits_{\mathbf{w}_{t+1}, \cdots, \mathbf{w}_T} \{ \sum\limits_{\bar{t}=t+1}^{T+1} \sum\limits_{k \in I(t,\bar{t})} \mathcal{C}_{t,\bar{t}}^k + \delta(|\mathcal{K}_0(\mathbf{w}_{\bar{t}})|) f_{\bar{t}}(\mathbf{w}_{\bar{t}}) \} & \text{si } |\mathcal{K}_0(\mathbf{w}_t)| \neq 0 \\ \min\limits_{\mathbf{w}_{t+1}} f_{t+1}(\mathbf{w}_{t+1}) & \text{otro caso} \end{cases} \tag{2.32}$$

Cabe destacar de (2.32) que los próximos periodos de regeneración para los periodos con índices en $\mathcal{K}_0(\mathbf{w}_t)$ están distribuidos en el intervalo $[t+1, T]$. Por esa razón, consideramos $p = |\mathcal{K}_0(\mathbf{w}_t)|$ el número de productos diferentes pedidos en el periodo t, de acuerdo con el patrón de nivel de inventario $\mathbf{w}_t$ y sea $x_{\bar{t}} \in \{0, 1, \cdots, p\}$ el número de productos con índices en $I(t, \bar{t})$, para $\bar{t} = t+1, \cdots, T$. Por lo tanto, el número de periodos consecutivos de producción, t y $\bar{t}$ para los productos en $\mathcal{K}_0(\mathbf{w}_t)$ corresponde con el número de soluciones de la siguiente ecuación diofántica:

$$x_{t+1} + \cdots + x_T = p \tag{2.33}$$

Debemos notar que el número de soluciones no negativas enteras en (2.33) es equivalente al número de maneras de colocar p objetos idénticos en $T - t$ cajones (similar al *Problema combinatorio de las estrellas y las barras*), tal y como se argumenta en [17]. Además, el número de estas soluciones es conocido como Sylvester's *Denumerant* $E(\mathbf{a}; p)$, donde $\mathbf{a} \in \mathbb{N}^{T-t}$ es el vector de coeficientes de la parte izquierda de (2.33).

Dado que $\mathbf{a} = \mathbf{1}$ en la expresión anterior, se tiene que $mcd(\mathbf{a}) = 1$ y por ello $E(\mathbf{1}; p)$ es una función *quasi-polinomial* en la variable p de grado $T -$

$t-1$ (mirar p.e. [2] para más detalles). Asimismo, la expresión en (2.33) está estrechamente relacionada con problemas conocidos más generales, como por ejemplo el *Problema de la Mochila con igualdad* y el *Problema de Partición de Números Enteros*.

Se puede tomar la siguiente expresión: dados enteros positivos $a_1, \cdots, a_n$ y b, se considera también un $0-1$ n-vector x tal que:

$$a^T x = b \Leftrightarrow a_1 x_1 + \cdots + a_n x_n = b$$

También es conocido que el Problema de la Mochila es un problema *NP-Completo*, a pesar de que se puede resolver en tiempo *pseudopolinomial* usando un algoritmo de Programación Dinámica. No obstante, como se afirmó arriba, (2.33) es un caso particular del Problema de la Mochila, que podemos resolver en tiempo *quasi-polinomial*.

Sea m_0 el número de variables $x_{\bar{t}}$ que toman valor 0 y, para $i = 1, \cdots, p$, sea $m_i \leq \lfloor \frac{p}{i} \rfloor$ ($\lfloor x \rfloor$ denota el entero más grande por debajo o igual a x) el número de veces que se repite i en la ecuación (2.33), esto es, el número de variables $x_{\bar{t}}$ tomando el valor i. Luego, el número de soluciones de dicha ecuación viene dado por:

$$\sum_{(m_0,\cdots,m_p)} \binom{T-t}{m_0, m_1, \cdots, m_p} = \sum_{(m_0,\cdots,m_p)} \frac{(T-t)!}{\prod_{i=0}^{p} m_i!} = \binom{p+T-t-1}{T-t-1}$$

donde $(m_0, \cdots, m_p)$ denota todos los vectores enteros no negativos, tal que $\sum_{i=0}^{p} m_i = p$ y, para una combinación fija $m_0, \cdots, m_p$, $\binom{T-t}{m_0,\cdots,m_p}$ representa las $(T-t)$-permutaciones del *multiconjunto* que surje de los posibles valores de $i = 0, 1, \cdots, p$.

Observamos que el coeficiente binomial en la expresión de arriba viene dado en [17] y [14]. Ahora, ilustraremos el método para contar todas las posibles soluciones en un simple ejemplo. Sea $p = |\mathcal{K}_0(\mathbf{w}_t)| = 3$ y $T - t = 4$, se tiene entonces la siguiente ecuación diofántica $x_1 + x_2 + x_3 + x_4 = 3$, cuyas soluciones se muestran en la siguiente tabla 2.4.

Como el $\text{mcd}(x_{t+1}, \cdots, x_T) = 1$ en (2.33) divide a p, siempre habrá un número finito de soluciones enteras no negativas, el cuál puede ser determinado recursivamente, por ejemplo, usando el *Algoritmo de Euclídes*. Como mencionamos arriba, resolver $f_1(\mathbf{0})$ de (2.32) es tan complicado como resolver los problemas *MCFP* o *UFLP*. Efectivamente, cuando $p = \mathcal{K}_0(\mathbf{0}) = T$ en $t = 1$, se tiene la siguiente expresión:

$$\binom{2T}{T} = \frac{(2T)!}{T!T!} = \frac{2^{T/2} \cancel{\prod_{i=0}^{\frac{T-2}{2}} (T-i)} \prod_{\substack{i=2n-1 \\ n=1\cdots T}} (2T-i)}{\cancel{T!}} = 2^{T/2} P(T)$$

$\bar{t}$	
1 2 3 4	*Permutaciones*
3 0 0 0 0 3 0 0 0 0 3 0 0 0 0 3	$\frac{4!}{1!(3)3!(0)} = 4$
2 1 0 0 2 0 1 0 2 0 0 1 1 2 0 0 1 0 2 0 1 0 0 2 0 2 1 0 0 2 0 1 0 0 2 1 0 1 2 0 0 1 0 2 0 0 1 2	$\frac{4!}{1!(2)1!(1)2!(0)} = 12$
1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1	$\frac{4!}{3!(1)1!(0)} = 4$

Tabla 2.4: Soluciones enteras no negativas de la ecuación $x_1 + x_2 + x_3 + x_4 = 3$.

donde $P(T)$ es un polinomio en T de grado $K = T$. Claramente, el problema generará una explosión combinatoria de soluciones para valores de entrada suficientemente grandes, y por eso la recurrencia (2.32) solo será útil para valores pequeños de entrada.

En lo referente a la complejidad computacional de (2.32), debemos notar que se tienen que resolver $2^{K/2}\mathcal{O}(T^K)$ problemas para determinar el coste mínimo para el problema comenzando en el periodo t con un patrón de nivel de inventario dado $\mathbf{w}_t$. Como quiera que en el peor de los casos se consideran 2^K patrones en cada periodo $t = 1, \cdots, T$, el esfuerzo computacional para cada periodo será $2^{3K/2}\mathcal{O}(T^K)$, lo cual lleva a que la complejidad del Algoritmo de Programación Dinámica en (2.32) sea $\mathcal{O}(T2^{3K/2}T^K)$.

Mediante una simple comparación en una hoja de cálculo, en donde T varía en $[K, 2K^2]$ para un K fijo, se puede probar que (2.32) supera a (2.31) cuando $\frac{T}{K} \gtrapprox 2\ln K + 3.6576$. En efecto, el nuevo algoritmo será útil cuando el número de periodos es relativamente más grande (en términos de una función *log-lineal* en K) que el número de productos.

Dado que este problema es intratable en la práctica con el Algoritmo de Programación Dinámica para valores grandes de K y T, debemos usar otros métodos. En concreto, se propone un algoritmo de Ramificación y Acotación (*Branch and Bound*) a continuación.

2.3.3. Algoritmo Branch and Bound

Tal y como explicamos en el capítulo anterior, la metodología Branch and Bound (B&B) consiste en particionar la región factible de una versión relajada del Problema *MCFP*, definido por las restricciones (2.25)-(2.28), donde se permite que las variables $z^k_{t,\bar{t}}$ y z_t sean no binarias para todo $(t^k, \bar{t}^k) \in E^-$ y $t = 1, \cdots, T$. Las particiones entonces van resolviéndose iterativamente, pudiéndose obtener así estos cuatro posibles resultados:

- El problema es no factible porque el poliedro asociado es vacío.
- La solución obtenida (cota inferior *LB*, si la solución no es entera, o cota superior *UB* en otro caso) es mayor que la *UB* actual, en cuyo caso ese problema (rama) en el árbol se poda.
- La solución obtenida representa una nueva *UB* mejor (más pequeña) que la actual, por lo que se debe actualizar ese último valor.
- La solución representa una cota inferior (solución no entera) que es menor que la actual *UB*, en cuyo caso habría que seguir explorando (ramificando) ese subproblema.

Para cualquier instancia, se define como cota superior (Upper Bound, *UB*) inicial el coste de la solución factible que consiste en hacer un único pedido en el periodo 1 para cubrir las demandas de todos los artículos. Resolviendo el problema relajado, el algoritmo genera un árbol de soluciones usando una búsqueda en ramificaciones, en donde cada nodo representa un problema descendiente del problema inicial. Por otro lado, en caso de tener que ramificar un subproblema, se tomará la primera variable $z^k_{t,\bar{t}}$, para $t = 1, \cdots, T$, $\bar{t} = t+1, \cdots, T+1$ y $k = 1, \cdots, K$, que resulte no entera.

2.4. Ejemplo de resolución con Branch and Bound

En lo que sigue, se computará un Problema *MPLS* utilizando un código en Python (ver en Apéndice A). Se tratará de planificar la reposición de pedidos para $K = 2$ productos y $T = 5$ periodos. En primer lugar, se generarán aleatoriamente los costes de reposición marginales s^k_t, así como las correspondientes demandas d^k_t y el vector de demandas acumuladas $\mathcal{D}^k$, para cada producto $k = 1, 2$ y periodos $t = 1, \cdots, 5$. Además, obtendremos las matrices de costos de reposición $\mathcal{C}^k_{t,\bar{t}}$ para cada producto $k = 1, 2$ y cada combinación de periodos $(t, \bar{t})$.

En este caso se han generado los siguientes costes de reposición conjunta $S = (7, 9, 5, 2, 4)$ y las demandas para cada producto:

$$d^1_t = \big(113, 111, 196, 444, 310\big) \ , \ d^2_t = \big(222, 485, 103, 239, 465\big)$$

Se tiene, por tanto, las siguientes demandas acumuladas:

$$\mathcal{D}^1 = \big(1174, 1061, 950, 754, 310\ 0\big)\ ,\ \mathcal{D}^2 = \big(1514, 1292, 807, 704, 465\ 0\big)$$

Por otro lado, se tienen las siguientes matrices de costes, obtenidas a partir de las expresiones $C_t^k(D_{t,\bar{t}-1}^k) = \sqrt{D_{t,\bar{t}-1}^k}$ y $H_t^k(I) = \sqrt[3]{I}$, para todo artículo k y todo periodo t:

$$\mathcal{C}^1 = \begin{pmatrix} 0 & 10.63015 & 19.77253 & 33.04868 & 54.73014 & 70.16312 \\ 0 & 0 & 10.53565 & 23.3302 & 43.651 & 58.2731 \\ 0 & 0 & 0 & 14 & 32.9271 & 46.6917 \\ 0 & 0 & 0 & 0 & 21.07131 & 34.22696 \\ 0 & 0 & 0 & 0 & 0 & 17.60682 \end{pmatrix}$$

$$\mathcal{C}^2 = \begin{pmatrix} 0 & 14.89966 & 34.4463 & 41.52577 & 54.97374 & 75.75505 \\ 0 & 0 & 22.02272 & 28.93626 & 41.95662 & 61.89781 \\ 0 & 0 & 0 & 10.14889 & 24.69906 & 45.05098 \\ 0 & 0 & 0 & 0 & 15.45962 & 34.28031 \\ 0 & 0 & 0 & 0 & 0 & 21.56386 \end{pmatrix}$$

Dados estos datos, resolveremos mediante el Algoritmo Branch and Bound. El código que hemos implementado realizará las modificaciones necesarias para tratar con un problema de Programación Entera Lineal Mixta. Se tomará un problema inicial, en este caso P^0. Si las variables no son enteras, pues tomará restricciones para ramificar y se resolverán los problemas resultantes hasta obtener una solución óptima con valores enteros.

Como cota superior inicial tenemos $UB = 152.91817$. Ahora, tomamos las restricciones $z_{1,4}^1 \leq 0$:

P^1
$z_{1,4}^1 \leq 0$
$z = 150.2280$

Tabla 2.5

y $z_{1,4}^1 \geq 1$.

P^2
$z_{1,4}^1 \geq 1$
$z = 152.0369$

Tabla 2.6

Dados estos dos subproblemas, se tiene que P^2 es factible y con variables enteras, luego el costo $z = 152.0369$ es candidato a óptimo. Por otro lado, para P^1 debemos ramificar. Ahora, teniendo en cuenta que $z^1_{1,4} \leq 0$, tomamos $z^1_{1,3} \leq 0$:

P^3
$z^1_{1,4} \leq 0,\ z^1_{1,3} \leq 0$
$z = 151.3966$

Tabla 2.7

y $z^1_{1,3} \geq 1$:

P^4
$z^1_{1,4} \leq 0,\ z^1_{1,3} \geq 0$
$z = 153.4869$

Tabla 2.8

Observamos que P^4 tiene un coste (LB) asociado mayor que la cota superior actual, luego debemos podar esa rama porque ninguna posible solución entera (UB factible para el MPLS original) obtenida a partir de ese problema P^4 va a ser mejor que la UB actual. Sin embargo, debemos ramificar nuevamente P^3 en busca de una solución óptima. Tomaremos, teniendo en cuenta que $z^1_{1,4} \leq 0, z^1_{1,3} \leq 0$, la restricción $z^1_{1,2} \leq 0$:

P^5
$z^1_{1,4} \leq 0, z^1_{1,3} \leq 0, z^1_{1,2} \leq 0$
$z = 152,8963$

Tabla 2.9

y por último $z^1_{1,2} \geq 1$:

P^6
$z^1_{1,4} \leq 0, z^1_{1,3} \leq 0, z^1_{1,2} \geq 1$
$z = 156.1602$

Tabla 2.10

De aquí, observamos que el costo asociado a P^6 es $z = 156.1602$, nuevamente supera la cota superior por lo que debemos podar esta rama. Por otro lado, el algoritmo para en P^5. En efecto, el costo óptimo obtenido es el costo $z = 152.0369$ asociado a P^2, pues es factible y se cumplen las condiciones requeridas.

A parte de resolver el problema, el código nos proporciona una visión gráfica de la solución. Tenemos:

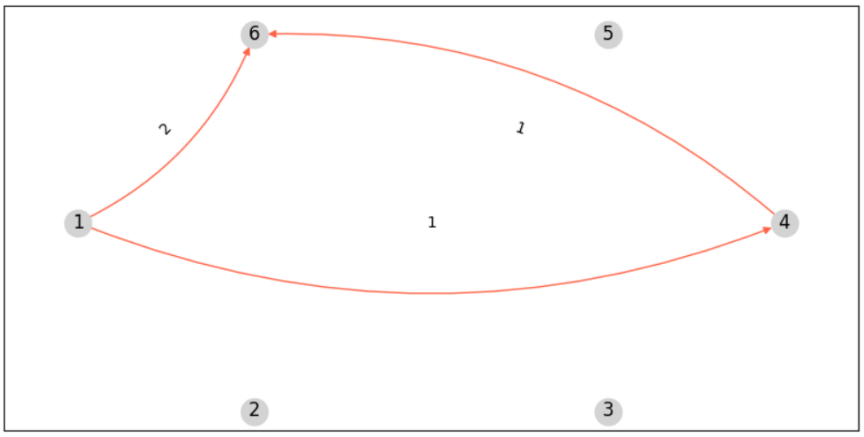
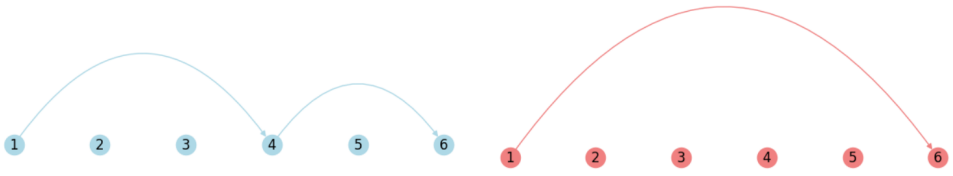


Figura 2.1: Gráfica reposición conjunta

Este esquema nos indica la política óptima. Los nodos representan los periodos. Observamos que para el segundo artículo se debe reponer una sola vez en el primer periodo, mientras que para el primer artículo se repone en el primer periodo para satisfacer la demanda acumulada hasta el periodo 3, y en el cuarto periodo para poder satisfacer el resto de demandas, como se muestra en las Figuras 2.1 y 2.2. En la Figura 2.2, el plan de reposición se muestra de manera independiente para cada artículo:



(a) Reposiciones para artículo 1 (b) Reposiciones para artículo 2

Figura 2.2: Gráfica política óptima

Observamos nuevamente que para el artículo 1 se repone en periodo $t = 1$ para abastecer la demanda acumulada $D^1_{1,3}$ y nuevamente reponemos en $t = 4$

para poder satisfacer la demanda acumulada $D^1_{4,6}$ mientras que para el artículo 2 basta con reponer en $t = 1$.

3

Conclusiones

A lo largo de esta memoria, el tópico general ha sido la Gestión de Inventarios. Hemos abarcado varios modelos clásicos de inventarios, sin embargo, se ha focalizado más en los Modelos de Tamaño de Lote Para Varios Artículos con Reposición Conjunta. Esta última variante se asemeja más a la realidad, siendo así un modelo útil a la hora de la práctica por ventajas como las economías de escala, flexibilidad en la gestión de inventarios y reducción de costos de pedido y almacenamiento entre otras. De esta manera, resalta la importancia de las Matemáticas en el ámbito empresarial pues nos permite mantener un control eficiente del inventario.

Ahora, ¿qué sucedería si consideramos demandas no constantes, existencia de roturas o tiempo de entrega? Es decir, ¿qué consecuencias tendría modificar las hipótesis de nuesta formulación? Se tiene que, si cambiamos las hipótesis de nuestro problema, surgirían nuevas formulaciones y por tanto nuevos complejos algoritmos de resolución. Por ello, la Investigación Operativa es imprescindible en el mercado, pues nos permite estudiar diversidad de problemas aplicados a la realidad donde se ven involucradas numerosas variables.

A

Apéndice: código en Python del Algoritmo Branch and Bound

Se adjunta el código en Python utilizado en 2.4 para resolver el Problema de Programación No Lineal Entera Mixta:

```
import pulp
import numpy as np
import random
import math
import networkx as nx
import matplotlib.pyplot as plt
#DATA PROBLEM
T = 5
K = 2
Tp = T + 2
MAX_D = 500
MAX_SETUP = 10
epsilon = math.pow(10, -5)
set_K = range(1,K+1)
set_T = range(1,T+1)
set_TP = range(2,T+2)
Keys = []
for k in range(1,K+1):
    for t in range(Tp+1):
        if t == 0:
            Keys.append("P"+str(k)+":"+str(t)+","+str(1))
        elif t == Tp - 1:
            Keys.append("P"+str(k)+":"+str(t)+","+str(Tp))
        else:
            for tp in range(t+1,Tp):
                Keys.append("P"+str(k)+":"+str(t)+","+str(tp))
# FILES
```

```
f_i = open("input_dat.txt", "w")
f_i.write("Instance of the MPLS with %s products and %s periods
\n" %(K,T))
f_o = open("output_dat.txt", "w")
# DATA STRUCTURES TO STORE THE RELEVANT INFORMATION
setup_values = random.sample(range(1, MAX_SETUP), T+1)
setup_values[0] = 0
f_i.write("Setup cost vector \n")
f_i.write(str(setup_values))
f_i.write("\n")
CC = np.zeros((K+1,T+2,Tp+1))
CH = np.zeros((K+1,T+1,Tp+1))
D = np.zeros((K+1,T+1))
CD = np.zeros((K+1,Tp+1))
f_i.write("Demand vectors: \n")
for k in range(1,K+1):
    for t in range(1,T+1):
        D[k][t] = np.random.randint(1,MAX_D)
    f_i.write("D[%s] = " % k)
    f_i.write(str(D[k]))
    f_i.write("\n")
#FUNCTIONS
rad = .2
conn_style = f'arc3,rad={rad}'
def offset(d, pos, dist = rad/2, loop_shift = .2, asp = 1):
    for (u,v),obj in d.items():
        if u!=v:
            par = dist*(pos[v] - pos[u])
            dx,dy = par[1]*asp,-par[0]/asp
            x,y = obj.get_position()
            obj.set_position((x+dx,y+dy))
        else:
            x,y = obj.get_position()
            obj.set_position((x,y+loop_shift))
def sub(a,b):
    return a-b
def get_aspect(ax):
    # Total figure size
    figW, figH = ax.get_figure().get_size_inches()
    # Axis size on figure
    _, _, w, h = ax.get_position().bounds
    # Ratio of display units
```

```
    disp_ratio = (figH * h) / (figW * w)
    # Ratio of data units
    # Negative over negative because of the order of subtraction
    data_ratio = sub(*ax.get_ylim()) / sub(*ax.get_xlim())
    return disp_ratio / data_ratio
def isBinary(key):
    if key <= 1 and (key-math.floor(key)) <= epsilon:
        return True
    else:
        return False
def CDem():
    for k in range(1,K+1):
        CD[k][T] = D[k][T]
        for t in range(T-1,0,-1):
            CD[k][t] = D[k][t] + CD[k][t+1]
CDem()
f_i.write("Cumulate Demand vectors: \n")
for k in range(1,K+1):
    f_i.write("D[%s] = " % k)
    f_i.write(str(CD[k]))
    f_i.write("\n")
def CD_(k,t,tp):
    if t > tp:
        return 0
    else:
        return CD[k][t] - CD[k][tp+1]
def H(k,t,I):
    return math.pow(I,1/3) # o bien return math.cbrt(I)
def C(k,t,X):
    return math.sqrt(X)
def CHold():
    for k in range(1,K+1):
        for t in range(1,T+1):
            CH[k][t][t] = 0
        for t in range(T-1,0,-1):
            for tp in range(t+1,T+2):
                CH[k][t][tp] = H(k, t, CD_(k, t+1, tp-1)) +
                CH[k][t+1][tp]
CHold()
Values= []
def CCost():
    for k in range(1,K+1):
```

```
        for t in range(Tp+1):
            if t == 0:
                CC[k][0][1] = 0
                Values.append(CC[k][0][1])
            elif t == Tp - 1:
                CC[k][T+1][Tp] = 0
                Values.append(CC[k][T+1][Tp])
            else:
                for tp in range(t+1,Tp):
                    CC[k][t][tp] = round(C(k, t, CD_(k, t, tp-1))
                    + CH[k][t][tp],5)
                    Values.append(CC[k][t][tp])
CCost()
def Value_UB():
    cost = 0
    for k in range (1,K+1):
        cost += CC[k][1][T+1]
    cost += setup_values[1]
    return cost
# Write in file "input_data" the cost matrix for each product k
for k in range(1,K+1):
    f_i.write("Cost matrix C_{t,tp}^{k} for product %s \n" % k)
    for t in range(1,T+1):
        f_i.write(str(CC[k][t]))
        f_i.write("\n")
List_pairs = zip(Keys,Values)
Data = dict(List_pairs)
# DECISION VARIABLES SHOULD BE DEFINED AS DICTIONARIES FOR PuLP
z_sum = {}
Z_sum = {}
z_vars = pulp.LpVariable.dicts("z_vars",(Keys), lowBound=0,
upBound=1, cat='Continuos')
Z_vars = pulp.LpVariable.dicts('Z_vars',((t) for t in set_T),
lowBound=0, upBound=1, cat='Continuos')
Z_sum = pulp.lpSum([Z_vars[t] * setup_values[t]] for t in set_T)
z_sum = pulp.lpSum([z_vars[t] * Data[t]] for t in Keys)
# MODEL
model = pulp.LpProblem("Joint setup K products ELS Model",
pulp.LpMinimize)
# OBJECTIVE FUNCTION
model += Z_sum + z_sum
# CONSTRAINTS
```

```
for k in set_K:
    model += z_vars["P%s:0,1" % k] == 1
    model += z_vars["P%s:%s,%s" % (k,T+1,T+2)] == 1
set_Left = []
set_Right = []
for k in set_K:
    for tp in range(1,T+2):
        if tp == 1:
            set_Left = ["P%s:%s,%s" % (k,0,1)]
            for t in range(tp+1,T+2):
                set_Right.append("P%s:%s,%s" % (k,tp,t))
        elif tp == T+1:
            set_Right = ["P%s:%s,%s" % (k,T+1,T+2)]
            for t in range(1,tp):
                set_Left.append("P%s:%s,%s" % (k,t,tp))
        else:
            for t in range(1,tp):
                set_Left.append("P%s:%s,%s" % (k,t,tp))
            for t in range(tp+1,T+2):
                set_Right.append("P%s:%s,%s" % (k,tp,t))
        model += pulp.lpSum([z_vars[key] for key in set_Left])
        - pulp.lpSum([z_vars[key] for key in set_Right]) == 0
        set_Left = []
        set_Right = []
for k in set_K:
    for t in range(1,T+1):
        for tp in range(t+1,T+2):
            model += z_vars["P%s:%s,%s" % (k,t,tp)] <=
            Z_vars[t]
# SOLVE MODEL
Prob_list = []
Keys_z =[]
for k in range(1,K+1):
    for t in range(1,T+1):
        for tp in range(t+1,T+2):
            Keys_z.append("P%s:%s,%s" % (k,t,tp))
Prob_list.append(model)
UB = Value_UB()
f_o.write("UB = %s" % UB)
f_o.write("\n")
Var_UB = []
cont = 0
```

```
while Prob_list:
    model_active = Prob_list.pop(0)
    model_active.solve()
    cont += 1
    model_active.writeLP("Active_model_%s.txt" % str(cont),
    writeSOS=1,mip=1)
    f_o.write("Solver relevant information: \n")
    f_o.write(str(pulp.LpStatus[model_active.status]))
    f_o.write("\n")
    f_o.write("Objective value: %s" %
    str(model_active.objective.value()))
    f_o.write("\n")
    if model_active.status == 1 and
    model_active.objective.value() <= UB:
        Binary = True
        Keys_z_aux = Keys_z.copy()
        while Binary and Keys_z_aux:
            key = Keys_z_aux.pop(0)
            if not isBinary(z_vars[key].varValue):
                Binary = False
        if Binary == False:
            model_branch_left = pulp.LpProblem("Son left problem",
            pulp.LpMinimize)
            model_branch_left = model_active.copy()
            model_branch_left += z_vars[key] <= 0
            Prob_list.append(model_branch_left)
            model_branch_left.writeLP("Model_branch_left_%s.txt"
            % str(cont),writeSOS=1,mip=1)
            model_branch_right = pulp.LpProblem("Son right
            problem", pulp.LpMinimize)
            model_branch_right = model_active.copy()
            model_branch_right += z_vars[key] >= 1
            Prob_list.append(model_branch_right)
            model_branch_right.writeLP("Model_branch_right_%s.txt"
            % str(cont),writeSOS=1,mip=1)
        else:
            UB = model_active.objective.value()
            model_better = model_active.copy()
if cont ==1:
    model_final = model_active.copy()
else:
    model_final = model_better.copy()
```

```
model_final.solve()
# the graphs with optimal decisions are plotted
G = nx.MultiDiGraph()
[G.add_node(t) for t in range(1,Tp)]
edge_labels = {}
for key in Keys_z:
    if z_vars[key].varValue == 1:
        k = int(key[1])
        t = int(key[3])
        tp = int(key[5])
        G.add_edge(t, tp)
        edge_labels[(t,tp)] = str(k)
plt.figure(figsize = (10,5))
nr_nodes = G.number_of_nodes()+1
posx = (nr_nodes)*[0]
for t in range(nr_nodes):
    posx[t] = 2*t+1
posy = (nr_nodes)*[0]
pos = nx.shell_layout(G, scale=1)
nx.draw_networkx_nodes(G, pos, node_size=300,
node_color='lightgray')
nx.draw_networkx_labels(G, pos=pos, font_color='black')
nx.draw_networkx_edges(G, pos=pos, edgelist=G.edges(),
edge_color='tomato',connectionstyle=conn_style)
d = nx.draw_networkx_edge_labels(G, pos=pos,
edge_labels=edge_labels)
offset(d, pos, asp = get_aspect(plt.gca()))
plt.show()
G1 = nx.DiGraph()
G2 = nx.DiGraph()
[G1.add_node(t) for t in range(1,Tp)]
[G2.add_node(t) for t in range(1,Tp)]
for key in Keys_z:
    if z_vars[key].varValue == 1:
        k = int(key[1])
        t = int(key[3])
        tp = int(key[5])
        if k == 1:
            G1.add_edge(t, tp)
        else:
            G2.add_edge(t, tp)
nr_nodes = G1.number_of_nodes()+1
```

```
posx = (nr_nodes)*[0]
for t in range(nr_nodes):
    posx[t] = 2*t+1
posy = (nr_nodes)*[0]
pos = {t:[posx[t],posy[t]] for t in range(nr_nodes)}
nx.draw(G1, pos, connectionstyle="arc3,rad=-0.7",
node_color='lightblue', edge_color='lightblue', with_labels=True)
plt.show()
nx.draw(G2, pos, connectionstyle="arc3,rad=-0.7",
node_color='lightcoral', edge_color='lightcoral', with_labels=True)
plt.show()
f_o.write("Solver relevant information: \n")
f_o.write(str(pulp.LpStatus[model_final.status]))
f_o.write("\n")
f_o.write("Objective value: %s" % str(model_final.objective.value()))
f_o.write("\n")
v = model_final.variables()
for t in range(T):
    f_o.write("Z[%s] = " % str(t+1))
    f_o.write(str(v[t].varValue))
    f_o.write(" ")
f_o.write("\n")
i = T+1
for k in range(K):
    for t in range(T):
        for tp in range(t+1,Tp-1):
            f_o.write("z[%s][%s][%s] = " %(k+1,t+1,tp+1))
            f_o.write(str(v[i].varValue))
            f_o.write(" ")
            i+=1
        f_o.write("\n")
    i+=2
for v in model_final.variables():
    f_o.write(str(v.name))
    f_o.write(" = ")
    f_o.write(str(v.varValue))
    f_o.write("\n")
f_i.close()
f_o.close()
```

Bibliografía

[1] A.Aggarwal, J.K. Park, Improved Algorithms for Economic Lot Size Problems. *Operations Research*, 41(3) (1993), pp. 549-571.

[2] V.Baldoni and N.Berline and Jesús De Loera and Brandon Dutra and Matthias Koppe and Michéle Vergne, Coefficients of Sylvester's. Denumerant [en línea]. 2005. *Cornell University*. Disponible en: https://arxiv.org/abs/1312.7147.

[3] M.S. Bazaraa and C.M. Shetty, Nonlinear Programming - Theory and Algorithms. *John Wiley & Sons*, 1979.

[4] Boctor, F.F., Laporte, G., Renaud, J., 2004. Models and algorithms for the dynamic-demand joint replenishment problem. *International Journal of Production Research 42*, 2667-2678

[5] E.U. Choo, G.H. Chan, Eyeballing heuristics for dynamic lot size problems. *Computers and Operations Research*, 16 (3) (1989), pp. 189-193

[6] S.S. Ergenguc, Multiproduct Dynamic Lot-Dizing Model With Coordinated Replenishments. *Naval Research Logistics*, 35 (1988), pp. 1-22

[7] A. Federgruen, M. Tzur, A Simple Forward Algorithm to Solve General Dynamic Lot Sizing Models with n Periods in 0(n log n) or 0(n) Time. *Management Science*, 37(8) (1991), pp. 909-925.

[8] Harris, F.W. How Many Parts To Make At Once. *Factory, The Magazine of Management*, 1913, vol. 10(2), pp. 135–136.

[9] M.J.G., Heuts, R.M.J., Kleijnen and J.P.C., Analysis and Comparison of Two Strategies for Multi-Item Inventory systems with joint replenishment costs. *European Journal of Operational Research*, 1992, pp. 405–412.

[10] Johansen, S.G., Melchiors, P., 2003. Can-order policy for the periodic-review joint replenishment problem. *Journal of the Operational Research Society 54*, 283-290.

[11] E.P.C. Kao, A Multi-Product Dynamic Lot-Size Model with Individual and Joint Set-Up Costs. *Operations Research*, Mar. - Apr., 1979, Vol. 27, No. 2 (Mar. - Apr., 1979), pp. 279-289.

[12] O. Kirca, M. Kokten, A new heuristic approach for the multi-item dynamic lot sizing problem. *European Journal of Operational Research*, 75 (1994), pp. 332-341

[13] Lee, F.C., Yao, M.J., A Global Optimum Search Algorithm for the Joint Replenishment Problem Under Power-of-Two Policy. *Computers and Operations Research 30* pp. 1319-1333.

[14] V.N. Murty, Counting the Integer Solutions of a Linear Equation with Unit Coefficients. *Mathematics Magazine*, vol. 54(2), pp. 79–81

[15] Olsen, A.L., An Evolutionary Algorithm to Solve the Joint Replenishment Problem Using Direct Grouping. *Computers and Industrial Engineering 48*, 2005, pp. 223-235.

[16] Robinson, E.P., A. Narayanan, and L.L. Gao. Efective Heuristics for the Dynamic Demand Joint Replenishment Problem. *Journal of the Operational Research Society 58 (6)*, 2007, pp. 808-815.

[17] S.M.Ross, A First Course in Probability. *CRC Press, New York*, 1976.

[18] S. Shin, Kevin L. Ennis, W. Paul Spurlin, Effect of Inventory Management Efficiency on Profitability: Current Evidence From The U.S Manufactuting Industry *Journal of Economics and Economic Education. Research*,(2015), Vol.16

[19] Silver, E.,A Simple Method of Determining Order Quantities in Joint Replenishments Under Deterministic Demand. *Management Science 22*,1976, pp. 1351–1361.

[20] M. Sipser, Introduction To The Theory Of Computation Third Edition. *Cengage Learning*,(2012) ep.7

[21] A.F.Veinott, Minimum Concave-Cost Solution of Leontief Substitution Models of Multi-Facility Inventory Systems. *Informs: Operations Research*, 1969, vol. 17, pp. 262–291.

[22] A.Wagelmans and S.V.Hoesel and A.Kolen, Economic Lot Sizing: An O(n log n) Algorithm That Runs in Linear Time in the Wagner-Whitin Case. *Informs: Operations Research*, 1992, vol.40.

[23] Webb, I.R., Buzby, B.R., Campbell, G.M., Cyclical schedules for the Joint Replenishment Problem with Dynamic Demands. *Naval Research Logistics 44*, 1997, pp. 577-589.

[24] Wilson, R. H. A Scientific Routine for Stock Control. *Harvard Business Review*, 1934, vol. 13, pp. 116–128.

[25] H.M.Wagner and T.M.Whitin, Dynamic Version of the Economic Lot Size Model. *Management Science 5*, 1958, vol.5, No.1, pp. 89–96.

[26] W. I. Zangwill, A Deterministic Multiproduct, Multifacility Production and Inventory Model. *Informs: Operations Research*, 1996, vol. 14, pp. 486-507.