Sección de Matemáticas
Universidad de La Laguna

Pablo Javier Sánchez Santana

Planificación de la reposición de inventarios considerando artículos perecederos. Aplicación a hemoderivados.

Dynamic lot-sizing model under perishability. Application to blood products.

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

Dirigido por
José Miguel Gutiérrez Expósito

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

Agradecimientos

Quisiera agradecer a mi familia el apoyo incondicional en todo momento. También a todos mis profesores, en especial a José Miguel, por sus consejos y orientación y a Mari Carmen, por alimentar mi interés por las matemáticas, sin ella tal vez no habría conocido esta carrera.

Pablo Javier Sánchez Santana
La Laguna, 4 de marzo de 2024

Resumen · Abstract

Resumen

*En este trabajo se aborda el problema de la cantidad económica de pedido dinámico (*dynamic EOQ *o* ELSP*) para productos perecederos de vida útil conocida, con aplicación al caso de un banco de sangre. Partiendo de trabajos anteriores, hemos desarrollado un modelo, que se adapta al problema de gestión del inventario de un hemoderivado, y un algoritmo de Programación Dinámica de complejidad computacional considerablemente menor que la del modelo que hemos usado como referencia. Adicionalmente, se aportará un ejemplo numérico que ilustra el algoritmo y el código en* Python *del método solución.*

Palabras clave: *ELSP – EOQ dinámico – Perecederos – Programación Dinámica – Python –*

Abstract

In this study, we address the issue of dynamic Economic Order Quantity (EOQ) or Economic Lot Size (ELS) for perishable products with a known lifespan, specifically applied to the context of a blood bank. Drawing from previous works, we have developed a model that is tailored to the inventory management of a blood derivative, and propose a Dynamic Programming algorithm with significantly lower computational complexity than the reference model used for its resolution. As an additional contribution, a numerical example will be provided to illustrate the algorithm, along with the code of the algorithm in Python.

Keywords: *ELSP – dynamic EOQ – Perishable – Dynamic Programming –* Python –

Contenido

Introducción

A lo largo de la historia, las empresas y diferentes comercios han querido maximizar sus beneficios o, en su defecto, reducir los costes de su negocio. Sin embargo, más allá de estimaciones aproximadas o empíricas, no fue hasta principios del siglo XX, cuando se abordó el problema de optimizar los costes asociados a la reposición de inventarios desde un punto de vista científico, como así se recoge en los trabajos de Harris [12] y Wilson [27]. Es obvio que la falta de la debida metodología a la hora de tratar estos problemas puede conducir a una mala gestión de los inventarios.

La gestión de los inventarios surge como respuesta a la demanda del mercado y debe perseguir la optimalidad. El objetivo principal del control de inventarios es asegurarse de que una empresa tenga la cantidad necesaria de inventario disponible en el momento justo y en el lugar adecuado para satisfacer la demanda de sus clientes y cumplir con sus objetivos operativos y financieros. Por un lado, la falta de inventario genera pérdidas por ventas no realizadas y un posible desgaste en la confianza del cliente. Por otro lado, un exceso de inventario deriva en un coste de mantenimiento que podría evitarse, además del deterioro o incluso la pérdida del producto.

Los modelos para la gestión del inventario se popularizaron especialmete tras la Segunda Guerra Mundial, a pesar de que los primeros estudios al respecto fueron propuestos, respectivamente, en 1913 y 1934 por F.W. Harris [12] y R.H. Wilson [27] para determinar de manera científica la cantidad económica de pedido (o *EOQ*, por sus siglas en inglés). Este modelo supuso el germen de una de las áreas de la *Investigación Operativa* y ha permitido el desarrollado de innumerables variantes, que han ido incluyendo características cada vez más realistas. Una de esas variantes fue desarrollada por Wagner y Whitin en 1958 [26], quienes extendieron el *EOQ* para considerar demanda determinista variable en el tiempo.

Con la llegada de la globalización y las tecnologías avanzadas la gestión de inventarios ha acelerado su evolución. Puesto que se han proporcionado nuevas

herramientas para optimizar la toma de decisiones en la cadena de suministro, anticipar la demanda y gestionar eficientemente los inventarios.

En esta memoria se partirá del trabajo realizado por Wagner y Whitin en 1958 [26], quienes extendieron al caso con demanda variable en el tiempo las aportaciones de Harris-Wilson. Se discutirán algunas formulaciones posteriores que se han ido proponiendo para tratar el problema *EOQ* dinámico con productos perecederos, poniendo especial énfasis en el trabajo desarrollado por Hsu en [13]. La principal aportación de este trabajo será la descripción de un modelo para el caso concreto de un banco de sangre, que se acompañará de un algoritmo de orden cuadrátrico para su resolución. Finalmente, se ilustrará el método solución con un ejemplo numérico y se presentará su codificación en Python.

1

Modelos para el problema *EOQ* con productos perecederos

En este capítulo se repasarán diferentes modelos que se han ido desarrollando en décadas pasadas y que suponen una evolución natural del modelo clásico del *EOQ*. La revisión se inicia con los resultados bien conocidos de Harris-Wilson y concluirá con algunos modelos más recientes del *EOQ* dinámico, que consideran productos perecederos. Además, se presentarán conceptos que serán importantes para el desarrollo del trabajo.

1.1. Primeras aportaciones para productos no perecederos

Los primeros en dar independientemente una respuesta científica al problema del tamaño de pedido (*EOQ*) fueron F.W. Harris en 1915 y R.H. Wilson en 1934.

El modelo del *EOQ* clásico considera que los parámetros son conocidos, la reposición es instantánea, el tiempo de retardo (lapso de tiempo entre la recepción un pedido y su disponibilidad para uso/consumo) es despreciable y la escasez no está permitida. Para formular el problema usaremos la siguiente notación. Sean Q la cantidad a pedir, r la razón de demanda conocida por unidad de tiempo, $T = \frac{Q}{r}$ la duración del periodo de planificación (tiempo entre dos reposiciones consecutivas), $\frac{1}{T} = \frac{r}{Q}$ el número de pedidos, p el costo unitario de reposición (compra), h el coste unitario de mantenimiento y c el coste fijo de pedido. Las fluctuaciones del inventario para el *EOQ* siguen el clásico patrón de diente de sierra que se muestra en la Figura 1.1. Por lo tanto, el coste total por unidad de tiempo puede expresarse como:

$$C(Q) = c\frac{r}{Q} + \frac{Qh}{2} + pr$$

Derivando esta función obtenemos la cantidad óptima a pedir Q^*:

$$Q^* = \sqrt{\frac{2rc}{h}}$$

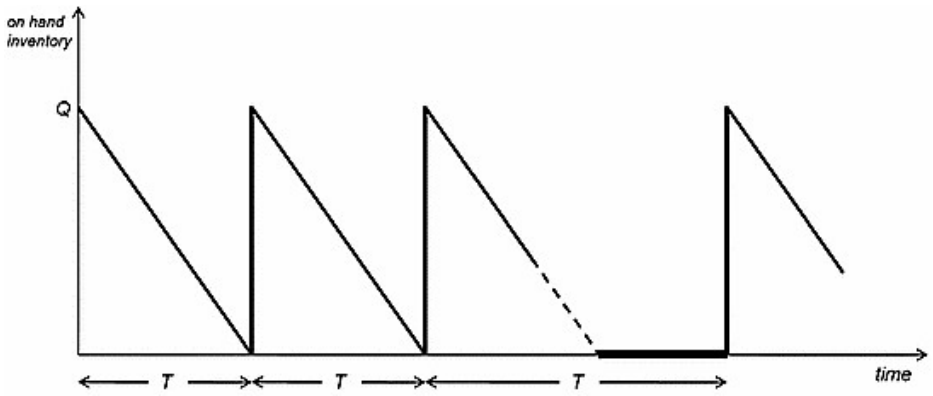

Figura 1.1. Modelo *EOQ* clásico.

Tuvieron que transcurrir más de cuatro décadas hasta que Wagner y Whitin en 1958 [26] propusieron la variante al caso dinámico (demanda y costes variables con el tiempo). Además de tener en cuenta consideraciones que Harris-Wilson no incluyeron en sus modelos, demostraron la optimalidad de la política de cero inventario (ZIO por sus siglas en inglés). Esta política establece que la producción debe llevarse a cabo durante los periodos en los cuales no haya inventario inicial disponible.

El problema original del EOQ dinámico (o $ELSP$ por sus siglas en inglés), introducido en [26] y que supondrá un resultado clave para esta memoria, considera un horizonte temporal finito formado por n periodos y un único artículo. La estructura de coste en cada periodo se define como la suma de un coste fijo y otro lineal de reposición, más un coste lineal de mantenimiento de inventario. En cada periodo, la demanda y los costes son conocidos. El modelo no permite rotura (escasez), por lo que la demanda en cada periodo debe ser satisfecha con la reposición en ese periodo o por el stock existente al final del periodo anterior. Además, en esta versión del problema, se asume que la reposición es instantánea y los tiempos de retardo (tiempo entre la recepción del pedido y su disponibilidad para la venta inmediata al cliente) son despreciables. Por lo tanto, el objetivo es reducir los costes de producción y mantenimiento del inventario, satisfaciendo la demanda en cada periodo sin incurrir en rotura/escasez.

Como ya se ha comentado, Wagner y Whitin utilizaron una estructura de costes basada en la suma de costes fijos más lineales para la producción y funciones lineales para el mantenimiento. Sin embargo, esta estructura se generalizó en estudios posteriores como en [25] y [28] al caso de funciones cóncavas para los costes de producción, mantenimiento y rotura. Esta clase de función es usada tradicionalmente para describir matemáticamente costes en contextos de economías de escala. Como ejemplo, estas economías se caracterizan porque los costes de producción marginales son no crecientes, es decir, aquellas cuyo coste

de producción unitario no aumenta con el incremento del volumen de producción.

Aunque matemáticamente el concepto de función cóncava puede generalizarse a cualquier dimensión d, $(f\colon \mathbb{R}^d \to \mathbb{R})$, en esta memoria nos limitaremos a definirlo para funciones de una única variable $(f\colon \mathbb{R} \to \mathbb{R})$. Gráficamente, una función es cóncava si, dados dos puntos cualesquiera de su gráfica, el segmento de recta que los une queda por debajo de la gráfica de la función (por encima en el caso de las funciones convexas), como se muestra en la Figura 1.2. De manera más rigurosa podremos comprobar si una función $f\colon \mathbb{R} \to \mathbb{R}$ es cóncava si cumple lo siguiente condicición:

Definición 1.1. *Diremos que una función $f(x)$ es cóncava en el intervalo (a, b) si, para cualesquiera dos valores $x_1, x_2 \in (a, b)$ y para cualquier valor $\alpha \in [0, 1]$, se cumple la siguiente desigualdad:*

$$f(\alpha x_1 + (1-\alpha)x_2) \geq \alpha f(x_1) + (1-\alpha)f(x_2)$$

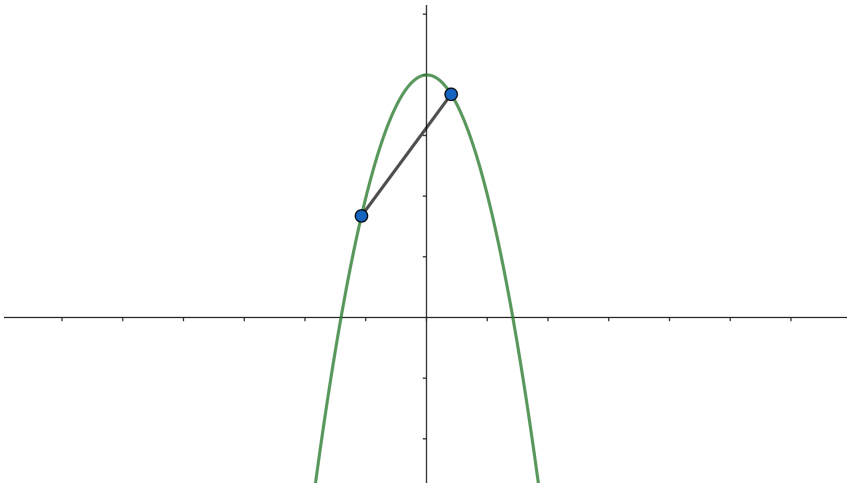

Figura 1.2. Función cóncava

Denotando por d_t, K_t y h_t, respectivamente, a la demanda, al coste de reposición fijo y al coste unitario de mantenimiento del periodo $t = 1, \cdots, n$; y definiendo la siguiente función delta: $\delta(0) = 0$ y $\delta(x) = 1$, si $x > 0$, la formulación original del $ELSP$, introducida en [26], puede expresarse formalmente como sigue:

$$ELSP: \ \text{mín} \sum_{t=1}^{n} (K_t \delta(x_t) + h_t y_t) \tag{1.1}$$

$$s.t.$$

$$y_{t-1} - y_t + x_t = d_t \qquad 1 \leq t \leq n \qquad (1.2)$$
$$x_t, y_t \geq 0 \qquad 1 \leq t \leq n \qquad (1.3)$$

Tanto Wagner y Whitin, como posteriormente Zangwill en [28] y Veinott en [25] para costes cóncavos, con y sin roturas, respectivamente, demostraron que existen soluciones extremas para el $ELSP$ que se corresponden con políticas ZIO. Este resultado permitió resolver el problema en tiempo polinomial haciendo uso de la Programación Dinámica (PD). Las políticas ZIO se corresponden con soluciones extremas del poliedro convexo, acotado y cerrado definido por (1.2) y (1.3), y dado que (1.1) puede considerse cóncava, entonces, por [3], el $ELSP$ tiene una solución óptima que será ZIO. En las siguientes secciones se introducirán los conceptos de Progrmación Dinámica y soluciones extremas de un poliedro.

1.1.1. Programación Dinámica

La Programación Dinámica (PD) es una técnica de optimización para problemas de decisión secuencial basada en la descomposición de problemas complejos en subproblemas más manejables. Las soluciones de estos subproblemas se van almacenando con el fin de utilizarlas para conseguir una solución óptima del problema inicial.

Esta técnica que se utiliza para resolver diversos problemas de optimización como son, por ejemplo, el de la mochila, el de planificación de proyectos, el de optimización de recursos, el del viajante de comercio o el del camino más corto. Veremos a continuación un ejemplo sencillo de este último tipo de problema.

1.1.2. Ejemplo: Problema del camino más corto

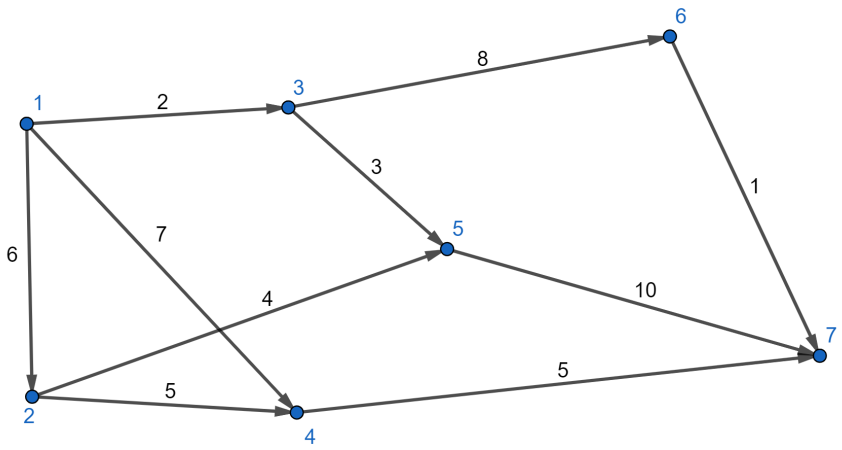

Figura 1.3. Grafo del problema

Sean $N = \{1, \cdots, 7\}$ el conjunto de nodos y $A = \{(i,j) : i,j \in N\}$ el conjunto de arcos del grafo dirigido $G(N,A)$ mostrado en la Figura 1.3, el cual representa una red dirigida donde a cada arco $(i,j) \in A$ le corresponde una distancia $t_{i,j}$. La idea es encontrar el camino más corto entre el nodo 1 (nodo fuente) y el 7 (nodo sumidero) haciendo uso de Programación Dinámica. No obstante, este tipo de problemas puede resolverse, por ejemplo, usando el algoritmo original de Dijkstra en [8], de orden $\mathcal{O}(|N|^2)$.

Denominemos f_i a la distancia mínima para ir del nodo i al nodo 7 y t_{ij} a la distancia entre los nodos i y j. Nótese que con estas definiciones $f_7 = 0$.

Tenemos que $f_i \leq t_{ij} + f_j$ para $i \neq 7$, puesto que el camino más corto de i a 7 debe ser menor que el camino de i a j sumado al camino más corto de j a 7. Esto significa que $f_i = \min\limits_{j:(i,j)\in A} \{t_{ij} + f_j\}$ para $i \neq 7$.

- $f_6 = t_{67} + f_7 = 1$
- $f_5 = t_{57} + f_7 = 10$
- $f_4 = t_{47} + f_7 = 5$
- $f_3 = \text{mín} \begin{cases} t_{35} + f_5 = 3 + 10 = 13 \\ t_{36} + f_6 = 8 + 1 = 9 \end{cases} = 9$
- $f_2 = \text{mín} \begin{cases} t_{24} + f_4 = 5 + 5 = 10 \\ t_{25} + f_5 = 4 + 10 = 14 \end{cases} = 10$
- $f_1 = \text{mín} \begin{cases} t_{12} + f_2 = 6 + 10 = 16 \\ t_{13} + f_3 = 2 + 9 = 11 \\ t_{14} + f_4 = 7 + 5 = 12 \end{cases} = 11$

Llegamos a que el camino más corto (Figura 1.4) entre el primer y el último nodo es el $(1,3,6,7)$ de 11 unidades de distancia. Como describimos anteriormente, para llegar la solución final se ha hecho uso de las soluciones de los subproblemas previos. En este caso, los subproblemas previos consisten en encontrar las distancias más cortas de los nodos $i \in \{2,3,4,5,6\}$ al nodo final 7.

1.1.3. Poliedros

De acuerdo con la geometría clásica, se denomina poliedro a ciertos cuerpos geométricos de caras planas y que encierran un volumen finito. En el contexto de la optimización matemática, el término poliedro se utiliza para referirse a un conjunto definido por un conjunto finito de desigualdades lineales. En otras palabras, un poliedro en este contexto es una región del espacio n-dimensional que está limitada por un conjunto de restricciones lineales. Matemáticamente se puede definir un poliedro n-dimensional como:

$$\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b\} \text{ con } A \in \mathbb{R}^{m\times n} \text{ matriz y } b \in \mathbb{R}^m \text{ vector.}$$

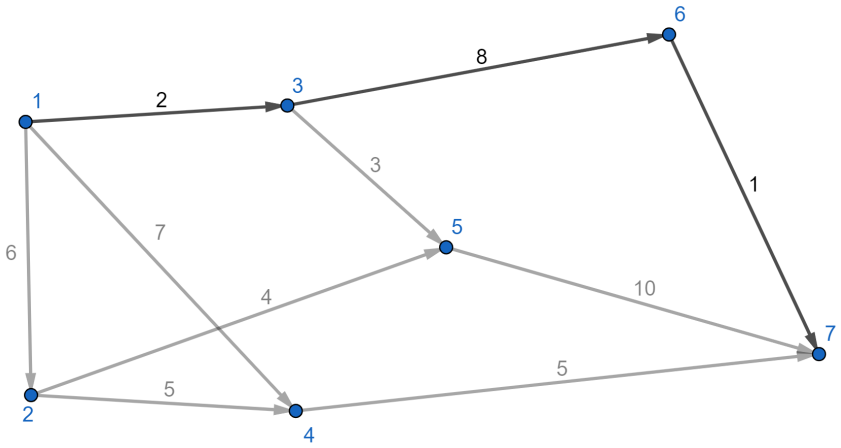

Figura 1.4. Solución del problema

Los poliedros se utilizan para modelar conjuntos factibles en diferentes problemas. Por ejemplo, en un problema de programación lineal, se busca maximizar o minimizar una función lineal sujeta a un conjunto factible representado por un poliedro.

La representación de un conjunto factible como un poliedro facilita la aplicación de métodos de optimización y técnicas algebraicas para resolver problemas de optimización.

Existen poliedros con cualidades que son de especial interés por su aplicación como son los poliedros compactos y los convexos.

Definición 1.2. *Un poliedro $\mathcal{P}$ es compacto en $\mathbb{R}^n$ si:*

1. *$\mathcal{P}$ es un conjunto cerrado en $\mathbb{R}^n$. Esto significa que contiene todos sus puntos límite. En términos matemáticos, para cualquier sucesión de puntos x_i en $\mathcal{P}$ que converge a un punto $x \in \mathbb{R}^n$, se cumple que $x \in \mathcal{P}$.*
2. *$\mathcal{P}$ es acotado. Es decir, existe un número finito $R > 0$ tal que el poliedro $\mathcal{P}$ está contenido en la bola euclidiana de radio R centrada en el origen. Matemáticamente, $\forall x \in \mathcal{P}$, se cumple que $\|x\| \leq R$, donde $\|\cdot\|$ denota la norma euclidiana.*

Definición 1.3. *Un poliedro $\mathcal{P}$ es convexo si, para cualesquiera $x, y \in \mathcal{P}$ el segmento que une x e y también está contenido en $\mathcal{P}$. Matematemáticamente, esto se puede expresar como:*

$$tx + (1-t)y \in \mathcal{P},\ \forall t \in [0,1]$$

Existen poliedros no convexos, que contienen al menos un trozo de segmento entre dos puntos del poliedro que está fuera del poliedro. Suelen ser poliedros con concavidades o agujeros en su estructura como, por ejemplo, el toro.

Los poliedros compactos garantizan la existencia de una solución óptima. Mientras que el especial interés en los poliedros convexos viene de que facilitan la búsqueda de soluciones óptimas, puesto que si existen, al menos una de ellas se encontrará en un punto extremo del poliedro. Además, permiten una base teórica sólida para probemas de optimización más generales

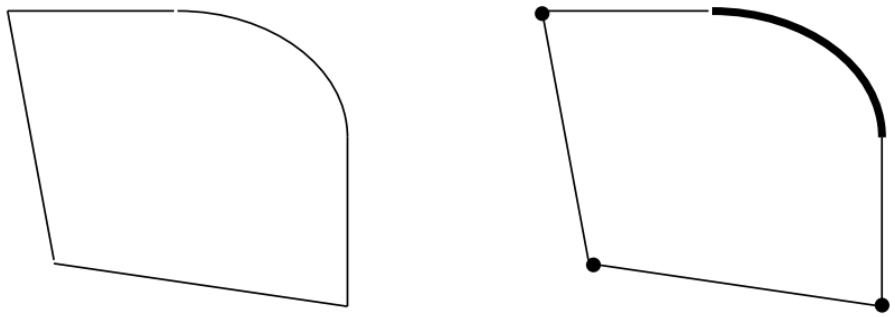

Figura 1.5. Poliedro convexo y sus puntos extremos.

1.2. Introducción al *ELSP* con productos perecederos

Una variante del $ELSP$, que añade complejidad al problema, es la consideración de la caducidad del producto cuya reposición quiere planificarse. La complejidad viene dada por factores como la reducción de la cantidad de producto desperdiciado, los costes de mantenimiento por el uso de refrigeradores u otras técnicas de conservación, o el cumplimento de regulaciones que protegen al consumidor. Además, hay que tener en cuenta el factor estacional, pues la demanda de ciertos productos varía en función de la época del año, al igual que el nivel de deterioro de los productos y la consiguiente preservación. El estudio de modelos clásicos de esta variante será el principal objetivo de esta memoria.

Se debe tener en cuenta que la variedad de modelos $ELSP$ con productos perecederos es considerable, no sólo por la naturaleza de la estructura de costes, sino que también influyen otros factores como el carácter (determinista o estocástico) de los parámetros, las suposiciones sobre las características del modelo (1 o varios productos, tipos de repisición (instantánea o no), si se permite escasez o no, etc) y, por supuesto, el patrón de deterioro de los productos. En relación a este último aspecto, algunos productos dejan de ser aptos para consumo o uso en un plazo fijado, mientras que obtener la vida útil de otros es más difícil de determinar puesto que pueden seguir un patrón de deterioro exponencial o de otra clase de distribución. En este trabajo nos centraremos en el estudio del modelo $ELSP$ con un único artículo y demanda y vida útil conocida. Además,

admitiremos reposición instantánea, tiempos de retardo despreciables y no se permitirá escasez.

1.2.1. Revisión de la literatura del *ELSP* para productos perecederos

Aunque Veinott [25] y S. Nahmias [17], entre otros, abordaron primero las políticas óptimas de pedido, se atribuye a Smith [23] la primera extensión del problema del $ELSP$ para considerar inventarios perecederos. Smith desarrolló un algoritmo de Programación Dinámica utilizando la propiedad ZIO introducida por Wagner y Whitin en [26]. Posteriormente, Friedman y Hoch [9] demostraron que la suposición de que una solución óptima para ese problema siempre es ZIO era incorrecta y, por lo tanto, el algoritmo de Smith tenía fallos. Más recientemente, Önal et al. [20] propusieron algoritmos con complejidades $\mathcal{O}(n^3)$ y $\mathcal{O}(n^4)$ para el caso sin restricciones y para el caso con capacidades de adquisición constantes en el tiempo, respectivamente, aplicando una política de emisión FIFO (se retiran primero del inventario los artÃculos que llevan mÃ¡s tiempo) en ambos casos. Además, se remite al lector a F. Jing y X. Chao [15] y a X. Chao et al. [5] para obtener una revisión actualizada sobre modelos deterministas y estocásticos, respectivamente.

1.3. *EOQ* para productos con vida útil conocida

1.3.1. Demanda determinística y estacionaria

Suponiendo conocida la demanda estacionaria, es decir, aquella que no varía de manera significativa con el paso del tiempo, y la vida útil del producto, expresada en un número fijo de m periodos, se determina de la solución del EOQ clásico.

$$Q^* = \sqrt{\frac{2rc}{h}},\ T^* = Q^*/r$$

Donde T^* representa el periodo óptimo entre pedidos. Si $T^* \leq m$ entonces la solución del problema Q_0 coincide con el EOQ, i.e. $Q_0 = Q^*$. Por el contrario, si $T^* > m$ entonces existirá producto a desechar cuando se realice el siguiente pedido. Por tanto, la cantidad óptima a pedir es $Q_0 = rm$, pues esta cantidad evita el desecho de pedido a la par que reduce los costes de mantenimiento ya que $Q_0 < Q^*$. En resumen, la solución de este problema se reduce a $Q_0 = min(Q^*, rm)$, tal como se recoge en [18].

1.3.2. Demanda determinística y no estacionaria

Este problema puede no resultar tan sencillo de resolver. En este caso las demandas, los costos de mantenimiento y producción no son fijos en cada periodo

del horizonte de planificación. Aplicando el trabajo de Wagner y Whitin a este problema se concluye que la cantidad óptima de pedido en cada periodo es cero o coincide con la demanda a ser satisfecha en el mismo. Si a esto le añadimos la política de cero inventario, el problema se reduce a elegir en qué periodos se debe realizar un pedido y en cuáles no.

No obstante, las políticas ZIO pueden no ser óptimas cuando se introduce el vencimiento de los productos. Esto fue desmostrado por Friedman y Hoch en 1978 con un contraejemplo sencillo en [9], en el que se asumía el deterioro del stock y costes de inventario no dependientes de la edad.

Las soluciones de las políticas ZIO se corresponden con los puntos (soluciones) extremos del poliedro correspondiente.

1.4. Complejidad computacional

La complejidad computacional trata sobre el estudio de la dificultad para resolver los problemas algorítmicos. Dicho de otra manera, se enfoca en analizar la eficiencia de los algoritmos midiendo el tiempo o el espacio (memoria) requeridos para resolver el problema en función del tamaño de la entrada. Este estudio permite buscar soluciones a problemas intratables cuya dificultad impide su resolución mediante una manera secuencial determinista, en la que se verifican todas las soluciones posibles.

1.4.1. Notación O-grande

Para expresar de manera precisa y concisa las características de rendimiento de los algoritmos en términos de sus recursos computacionales existen varias herramientas. Una de ellas (y la que usaremos en este trabajo) es la notación O-grande ($\mathcal{O}$).

Esta notación se utiliza para acotar superiormente el crecimiento del tiempo de ejecución de un algoritmo dado. Diremos que el esfuerzo computacional de un algoritmo es de orden $\mathcal{O}(f(n))$ si, para n suficientemente grande, el tiempo de ejecución es a lo sumo $kf(n)$ para alguna constante k.

Veamos a continuación algunos ejemplos, que se ilustran en la Figura 1.6:

- $\mathcal{O}(1)$ (Constante): la complejidad es constante y no depende del tamaño de entrada. Ejemplos de algoritmos con esta complejidad son el determinar si un número es par o el realizar una operación aritmética simple.
- $\mathcal{O}(\log(n))$ (Logarítmico): la complejidad crece de manera logarítimica con el tamaño de entrada. Se da en algoritmos como la búsqueda binaria en un vector ordenado.
- $\mathcal{O}(n)$ (Lineal): la complejidad crece de manera proporcional al tamaño de entrada. Se da en algoritmos de búsqueda en estructuras de datos lineales no

ordenadas o procesamiento de datos lineales como recorrer una lista de datos una vez.

- $\mathcal{O}(n\log(n))$ (Log-lineal): es una combinación de un componente lineal y otro logarítmico. Por ejemplo, algoritmos de ordenación como Merge Sort o Quick-Sort son de este orden de complejidad.
- $\mathcal{O}(n^2)$ (Cuadrático): la complejidad crece cuadráticamente con el tamaño de entrada. Se da en algoritmos con bucles anidados como el Bubble Sort.
- $\mathcal{O}(n^3)$ (Cúbico): la complejidad crece cúbicamente con el tamaño de entrada. Suele darse en simulaciones científicas y en procesamiento de datos tridimensionales, como el algoritmo de multipliación de matrices tridimensionales.
- $\mathcal{O}(n!)$ (Factorial): la complejidad crece de manera factorial con el tamaño de entrada. Muchos problemas de optimización combinatoria como en logística y planificación de rutas suelen ser de este orden de complejidad. Es el caso del problema clásico del viajante de comercio (TSP).
- $\mathcal{O}(c^n)$ (Exponencial): la complejidad crece de manera exponencial con el tamaño de entrada. Se da en problemas de exploración exhaustiva, particiones y combinaciones como el algoritmo de generación de todos los subconjuntos de un conjunto.

Cabe destacar que estos dos últimos ejemplos se suelen evitar por lo impracticable que puede volverse su uso cuando tenemos entradas grandes. Esto sugiere la importancia del análisis de la complejidad computacional, que es esencial a la hora de tomar decisiones sobre la elección de algoritmos. Puesto que dicha elección puede ser determinante en el rendimiento de ciertos sistemas.

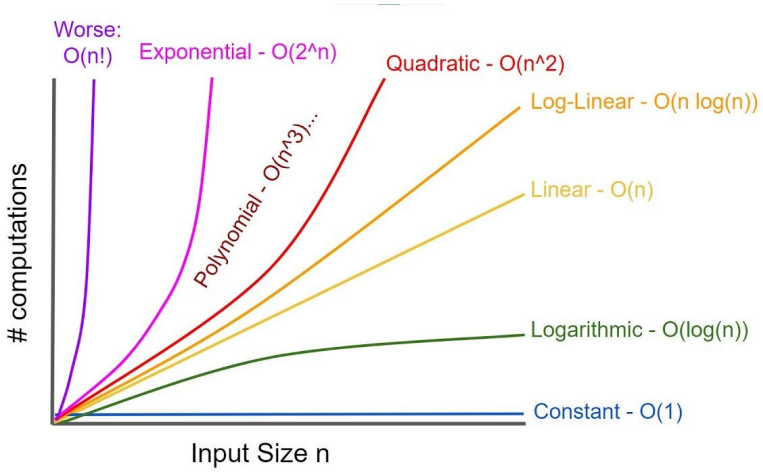

Figura 1.6. Órdenes de complejidad computacional

1.4.2. Clasificación de los problemas según su complejidad computacional

A la hora de resolver un problema podemos encontrarnos con que este no se pueda resolver de manera tradicional, es decir, de manera secuencial determinista. Cuando esto ocurre se recurre a métodos no exhaustivos, en los que se evalúan un subconjunto de las posibles soluciones. Por esta razón es importante la clasificación de los problemas acorde a la teoría de la complejidad.

Definición 1.4. *Se dice que un algoritmo tiene complejidad polinomial si el tiempo de ejecución (orden de complejidad) está acotado por un polinomio en términos del tamaño de la entrada. Si el algoritmo no es polinomial, se denomina exponencial.*

- **Problemas de decisión.** Este tipo de problemas son el origen de la complejidad computacional. En ellos se plantean cuestiones cuya respuesta es sí o no.
  Un ejemplo ilustrativo de este tipo de problemas es el Problema del Viajante de Comercio (TSP), donde la pregunta clave es si existe un recorrido que visite cada ciudad exactamente una vez, volviendo al punto de partida y sin superar cierta longitud límite.
- **Problema de Optimización.** Un problema de optimización es un tipo de problema en el que se busca encontrar la mejor solución de un conjunto de soluciones posibles. En otras palabras, dada una función objetivo $f(x)$, el objetivo es encontrar la solución $\mathbf{x}$ sujeta a ciertas restricciones con el mejor valor objetivo, en el caso de problemas de minimizar la que produce el menor valor, en el caso de problemas de maximizar la que produce el mayor valor. Todo problema de optimización tiene asociado una versión de decisión.
  Un ejemplo de problema de optimización es el Problema de Flujo Máximo que consiste en determinar la máxima cantidad de flujo que puede ser enviada entre dos nodos de una red dirigida, con una capacidad asociada a cada uno de los arcos de la red.
- **Decibilidad sobre espacios continuos.** La teoría de la complejidad fue desarrollada para problemas combinatorios o discretos, pero en la realidad existen muchos problemas de optimización cuyo espacio de soluciones es infinito o continuo.
  Esto puede darse en ciertos problemas de decisión de topología como el problema de Conectividad de Espacios Compactos.
- **Decibilidad en funciones multivaluadas.** El problema radica cuando nos encontramos con funciones multivaluadas, como naturalmente se encuentran en los problemas multiobjetivo. En estos casos podría pasar que un objetivo podría cumplir la condición deseada, mientras que otro no. Esto resulta en dudas a la hora de considerar cuándo el problema está resuelto o incluso en

casos en los que una solución óptima puede implicar que otra sea la peor. El asunto de la decibilidad en funciones multivaluadas no ha sido tan estudiado y sigue siendo un campo de estudio en la teoría de la complejidad.

- **Clase P (Polinomial).** Esta clase contiene problemas de decisión para los que existe un algoritmo determinista que puede resolverlo en tiempo polinomial. Estos problema se consideran fáciles de resolver y se dice que son tratables. Ejemplos de problemas de esta clase son los ya mencionados problemas de ordenación. Algunos algoritmos que los resuelven, basados en la estrategia de *divide y vencerás*, tienen complejidad $\mathcal{O}(n\log(n))$, como el *merge sort*, mientras que otros, que no comparan los elementos entre ellos, son de orden lineal, como por ejemplo el *bucket sort* o *postman's sort*.
- **Clase NP (No Determinista Polinomial).** La clase NP contiene problemas de decisión que pueden ser verificados en tiempo polinomial mediante un algoritmo no determinista. En otras palabras, un problema se considera parte de la clase NP si un algoritmo no determinista puede generar una solución candidata en tiempo polinomial, y además, mediante un algoritmo determinista, es posible verificar en tiempo polinomial si esta solución es válida o no.
  Un ejemplo de problema NP es el problema del Clique, que se formula de la siguiente manera: dado un número entero k y un grafo no dirigido $G = (V, E)$, donde V es el conjunto de vértices y $E = \{(v_i, v_j) \mid v_i, v_j \in V\}$ es el conjunto de aristas, ¿existe un conjunto de k vértices en G tal que cada par de vértices en el conjunto esté conectado por una arista? Es decir, ¿hay un clique de tamaño k en el grafo G? El Problema del Clique está en NP porque, dada una solución propuesta (un conjunto de vértices), se puede verificar en tiempo polinómico si realmente forma un clique, simplemente comprobando si hay una arista que conecta cada par de vértices en el conjunto.

Definición 1.5. *Un problema X es reducible en tiempo polinomial a otro problema Y si se puede hallar un algoritmo polinomial que solucione X asumiendo que existe otro algoritmo polinomial que resuelve Y.*

- **Problema NP-Completo.** Un problema X de la clase NP se dice que es NP-Completo si todos los problemas de la clase NP son reducibles en tiempo polinomial a X.
  Un ejemplo clásico de problema en NP-completo es el Problema de la Mochila (Knapsack Problem). Este problema plantea maximizar el valor del contenido de una mochila sin exceder su peso, conocido el peso y el valor asignado de cada objeto. Este problema es NP porque, dada una combinación de objetos, se puede verificar en tiempo polinómico si su peso total no excede la capacidad y si el valor total es máximo. Sin embargo, la resolución eficiente del problema, es decir, encontrar la combinación óptima, no se ha demostrado que esté en P. La NP-completitud del Problema de la Mochila se evidencia en su capacidad

para ser utilizado como una pieza fundamental en la reducción polinómica de otros problemas en la clase NP-completo.

- **Problema NP-Duro.** Un problema NP-Duro es un problema de optimización cuya versión de decisión es NP-Completo. Dicho de otra forma, un problema X se denomina NP-duro si existe otro problema $Y \in$ NP-completo de forma que Y es reducible en tiempo polinomial a X. Esta clase puede describirse como aquella que abarca los problemas que son al menos tan difíciles como un problema de NP, pero sin encontrarse necesariamente en NP.
  Un ejemplo de NP-duro es el Problema de la Parada: dada una descripción de un programa de computadora (máquina de Turing) y una entrada específica para ese programa, ¿se detendrá el programa en algún momento o entrará en un bucle infinito? El Problema de la Detención es NP-duro porque cualquier problema en NP puede ser reducido a él en tiempo polinómico. Sin embargo, no está en NP porque no se puede verificar en tiempo polinómico si una solución propuesta es correcta.

2

Un modelo de reposición de artículos perecederos con aplicación a bancos de sangre

En este capítulo se enfatiza la problemática de los modelos EOQ dinámicos para productos perecederos, con un enfoque especial en los hemoderivados, que serán nuestro campo de estudio. Por esa razón comenzaremos el capítulo subrayando la importancia de una buena gestión en este tipo de inventarios.

Posteriormente describiremos el modelo de Hsu [13] para el EOQ dinámico con productos perecederos, que usaremos como formulación inicial para nuestro trabajo. Tras presentar algunas propiedades y hacer un par de observaciones sobre el modelo de Hsu, introduciremos nuestra formulación. Demostraremos que en el caso de productos con vida útil fijada y costes (mantenimiento y resposición) cóncavos, el problema se reduce a un Economic Lot Size Problem, o $ELSP$, con capacidad de inventario. Esta propiedad permitirá el diseño de un algoritmo de Programación Dinámica de orden cuadrático, que se ilustrará con un simple ejemplo numérico y se programará en Python.

2.1. Motivación

Como se adelantó en el capítulo anterior, este trabajo se centrará en los modelos $ELSP$ con demanda conocida y no estacionaria considerando productos perecederos con vida útil conocida. En concreto, la tarea principal será la de buscar un modelo aplicable a bancos de sangre y hemoderivados. Precisamente, el cumplimiento de las normas de conservación en el caso de estos productos debe ser tremendamente escrupuloso, ya que, en caso contrario, sus consecuencias podrían ser fatales.

Los bancos de sangre son entidades que se dedican, principalmente, a la recolección, almacenamiento, procesamiento y distribución de sangre y, en especial, de sus componentes (glóbulos rojos, plasma y plaquetas principalmente, aunque existen más productos con los que se trabajan en estos bancos de sangre) para realizar transfusiones médicas. Estas transfusiones se realizan en situaciones médicas diversas como pueden ser ciertas cirugías, trasplantes, tratamientos contra el cáncer o enfermedades hematológicas. Al tratarse de una cuestión de

vital importancia como la salud no hay lugar a fallos por deterioro o falta de producto.

El factor estacional en este caso no es desdeñable. Como muestran, por ejemplo, los datos del Servicio Canadiense de sangre [4], en las épocas vacacionales aumenta la demanda de sangre debido al aumento del tráfico y los consiguientes accidentes.

A pesar de que la sangre y sus derivados se deterioran con el tiempo, los resultados al usarlos en adultos enfermos muestran que, siempre que se utilicen dentro de sus vidas útiles, no habrá problemas. Por ejemplo, en [6] se indica que los glóbulos rojos pueden almacenarse durante aproximadamente 42 días a entre 1 y 6 grados Celsius, mientras que las plaquetas durante unos 5 ó 7 días a temperatura ambiente y el plasma puede llegar a conservarse por más de un año a temperaturas inferiores a los -18 grados Celsius, aunque todos estos plazos dependerán en última instancia de las regulaciones locales.

Por lo tanto, nuestro modelo no es dependiente de la edad en el sentido definido por Hsu [13], ya que no se pierde ninguna fracción del producto en periodos anteriores a su vida útil. Por el contrario, todo el lote se reemplaza por completo en un periodo y aquellas unidades de producto no usadas se desechan por completo después de un número fijo de periodos, que se corresponde con la vida útil del producto.

2.1.1. ¿Cómo pueden las matemáticas ayudar en este tipo de inventarios?

En este contexto, los modelos matemáticos como el $ELSP$ son herramientas de incalculable valor. La estructura formal para abordar la demanda y el factor no estacionario que ofrecen, además de la optimización de los valiosos recursos sanguíneos son aspectos clave para la gestión de los inventarios de este tipo de bancos. Permiten a los bancos de sangre tener una base cuantitativa sobre la que tomar decisiones que les ayuden a encontrar el equilibrio entre la eficiencia operativa y la seguridad de sus pacientes.

Nuestro trabajo busca no solo abordar los desafíos específicos asociados a la gestión de inventarios en bancos de sangre, sino también proporcionar soluciones que respeten las complejidades inherentes a este entorno crucial de la atención médica.

2.2. Formulación del problema

Dado que estamos considerando un modelo determinista en el cual la escasez de productos es crítica, se asume en todo momento que la escasez del producto no está permitida y que el reabastecimiento ocurre de manera ins-

tantánea (es decir, los tiempos de espera son nulos o insignificantes) cuando un periodo es un "periodo de reabastecimiento o producción".

Por comodidad, seguiremos la notación utilizada por Hsu [13]. De acuerdo con esto, consideraremos un horizonte temporal de n periodos. En cada periodo, como se describe en [1], se sigue el siguiente orden de operaciones: cubrir la demanda (aplicando una política de emisión FIFO - *First In First Out* -) al comienzo del periodo, reabastecer con unidades frescas (en caso de periodos de reabastecimiento) y retirar (eliminar) las unidades obsoletas al final del periodo. La estrategia FIFO quedó demostrada en [21] como la óptima a la hora de minimizar el número total de unidades que alcanzan el final de su vida útil, es por ello que será la aplicada en este trabajo. Esta estrategia establece que los bienes adquiridos o producidos en primer lugar son los que se utilizan o venden en primer lugar.

2.2.1. Formulación general: el modelo de Hsu

Además de las consideraciones previas, se supone que la demanda de cada periodo es satisfecha al comienzo del mismo. Para cada periodo $t = 1, \cdots, n$, y para cada periodo i, donde $1 \leq i \leq t \leq n$, introducimos las siguientes tres categorías: parámetros, variables y funciones.

Parámetros:

- $d_t > 0$:= demanda del producto en el periodo t.
- α_{it} := fracción del producto (tasa de deterioro) reabastecido en el periodo i perdida durante el periodo t.

Variables:

- x_t := La cantidad reabastecida de producto en el periodo t.
- z_{it} := La cantidad de producto demandada en el periodo t que se satisface con lo producido en el periodo i.
- y_{it} := La cantidad de producto reabastecida en el periodo i y conservada en el inventario al comienzo del periodo t. Esta cantidad excluye la cantidad z_{it} utilizada para cubrir la demanda en el periodo t.

Funciones:

- $C_t(x_t)$:= El costo de reponer o producir x_t unidades de producto en el periodo t.
- $H_{it}(y_{it})$:= El costo de mantener y_{it} unidades de producto desde el periodo i hasta el periodo t.

Tanto el coste de mantenimiento como el de reposición se suponen funciones cóncavas no decrecientes con $C_t(0) = H_{it}(0) = 0$, para todo $1 \leq i \leq t \leq n$. Esta

suposición modela el comportamiento clásico de cualquier actividad económica en el contexto de economías de escala, ya explicadas en el capítulo anterior.

Las siguientes suposiciones fueron introducidas en [13] como *Suposiciones* 1 y 2, respectivamente. Para $1 \leq i \leq j \leq t \leq n$, $\alpha_{it} \geq \alpha_{jt}$ y $H_{it}(y) \geq H_{jt}(y)$, para $y \geq 0$. Además, se asume, sin pérdida de generalidad, que tanto el inventario disponible al inicio del periodo 1 como el inventario al final del periodo n son iguales a 0, es decir, $y_{01} = y_{in} = 0$. En consecuencia, el modelo $ELSP$ para un único artículo perecedero dependiente de la edad, introducido por Hsu en [13], se formula de la siguiente manera:

$$P : min \sum_{t=1}^{n} (C_t(x_t) + \sum_{i=1}^{t} H_{it}(y_{it})) \tag{2.1}$$

$$s.a :$$

$$x_t - z_{tt} = y_{tt}, \qquad 1 \leq t \leq n \tag{2.2}$$

$$(1 - \alpha_{i,t-1}) y_{i,t-1} - z_{it} = y_{it}, \qquad 1 \leq i < t \leq n \tag{2.3}$$

$$\sum_{i=1}^{t} z_{it} = d_t, \qquad 1 \leq t \leq n \tag{2.4}$$

$$x_t, y_{it}, z_{it} \geq 0, \qquad 1 \leq i \leq t \leq n \tag{2.5}$$

El objetivo (2.1) en P es determinar una política de reabastecimiento óptima con mínimo coste total. El conjunto de restricciones en (2.2) asegura que solo se mantengan y_{tt} unidades como inventario después de utilizar z_{tt} unidades de las x_t unidades de reabastecimiento para cubrir la demanda en el periodo t. Para los periodos restantes $i < t$, las restricciones en (2.3) corresponden a ecuaciones de conservación de flujo en redes generalizadas (o redes con ganancias/pérdidas de flujo, consulte [22] para más detalles). Por lo tanto, cada restricción establece que el inventario disponible al comienzo de t es el inventario reducido $(1 - \alpha_{i,t-1}) y_{i,t-1}$ debido a la obsolescencia al final del periodo anterior menos las z_{it} unidades utilizadas para cubrir la demanda en el periodo t. La demanda en cualquier periodo t debe satisfacerse a partir de los reabastecimientos realizados en ese mismo periodo y en los anteriores, como se expresa en (2.4).

Hsu [13] demostró que P puede transformarse equivalentemente en un problema de flujo de coste mínimo en una red generalizada con pérdida de flujo. Además, se introdujeron algunas propiedades estructurales sobre las que se podría idear un algoritmo de Programación Dinámica polinómico. Más específicamente, se definieron los siguientes coeficientes. Se denota por A_{kt}^{i} a la cantidad de producto a reponer en el periodo $i < t$ y mantenida durante los periodos intermedios k para satisfacer una unidad de demanda en el periodo t, que se puede calcular de la siguiente manera:

$$A^i_{kt} = \frac{1}{\prod_{l=k}^{t-1}(1-\alpha_{il})} \text{ y } A^i_{ii} = 1, \qquad \text{para } 1 \le i \le k < t \le n, \tag{2.6}$$
$$A^i_{kt} = A^i_{kq} A^i_{qt}, \qquad \text{para } k < q < t,$$
$$A^i_{kt} \ge A^j_{kt}, \qquad \text{para } i < j \le k.$$

Se demuestra en [13] las siguientes propiedades estructurales para el problema P:

Propiedad 2.1. Existe una solución óptima Ω^* para P tal que $z^*_{it} = d_t$ para cada t, $1 \le t \le n$, y para algún i, $1 \le i \le t$. En otras palabras, esta propiedad establece que la demanda en un periodo t se satisface por completo por el reabastecimiento en un solo periodo i.

Propiedad 2.2. Existe una solución óptima Ω^* para P en la que si $i < j$ son dos periodos y $z^*_{jk} = d_k$, para algún $k \ge j$, entonces $z^*_{it} = 0$, para todo t, $k \le t \le n$.

Este resultado induce la siguiente propiedad, conocida como Propiedad de División de Intervalos (IDP por sus siglas en inglés).

Propiedad 2.3. (IDP) Se supone que hay R periodos de producción en la solución, denotados por $1 \le i_1 < i_2 < \ldots < i_R \le n$. Por tanto, hay $R+1$ índices, $1 \le j_1 < j_2 \ldots < j_R < j_{R+1} = n+1$, tal que para cada periodo de producción t, $1 \le t \le R$, se tiene:

(a) $i_t \le j_t$,
(b) $z_{i_t k} = d_k$, for $j_t \le k < j_{t+1}$.

Aunque es cierto que el $ELSP$ clásico puede resolverse eficientemente usando la propiedad ZIO, lamentablemente esto no ocurre para el caso del problema P (como demostraron Friedman y Hoch en [9]). En general, para costes de inventario dependientes de la edad (con independencia de si hay o no deterioro del stock), se pueden encontrar ejemplos de P en los que la solución óptima no cumple la propiedad ZIO.

Apoyándose en la propiedad IDP, Hsu elaboró en [13] un algoritmo de Programación Dinámica de complejidad $\mathcal{O}(n^4)$ para los costos previamente dados, y de complejidad $\mathcal{O}(n^3)$ y $\mathcal{O}(n^2)$, respectivamente, para cuando los costes de reposición y mantenimiento se corresponden con la suma de un coste fijo (constante) y un coste lineal, y para cuando las funciones de coste de mantenimiento son puramente lineales.

Desde nuestro punto de vista, se pueden hacer dos críticas al modelo de Hsu. Por un lado, la dificultad de determinar la función $H_{it}(y_{it})$, para cada par

$(i,t) : 1 \leq i \leq t \leq n$. Esta función se podría interpretar como un costo desagregado de una función de costo de mantenimiento más general (digamos $H_t(y_t)$) de mantener y_t unidades al final del periodo t. También vale destacar que la suposición $H_{it}(y) \geq H_{jt}(y)$, para $1 \leq i \leq j \leq t \leq n$ y $y \geq 0$, podría no cumplirse en la práctica, como es el caso de los periodos inflacionarios, es decir, aquellos en los que la tasa de inflación es especialmente elevada y continua. Además, la suposición de que la función de costo $H_{it}(y_{it})$ es cóncava y no decreciente es también cuestionable, puesto que el coste marginal de mantener y_{it} podría aumentar a medida que el inventario de unidades reabastecidas antes del periodo i y retiradas antes de t disminuye a lo largo de los periodos de i a t. Para demostrar esto, supongamos que $h_k(y, Y)$ denota el coste de mantener y unidades de producto en el periodo $k : i \leq k \leq t$, cuando ya hay Y unidades de producto en el inventario que fueron producidas en periodos previos a i y que serán retiradas siguiendo la política FIFO. Para cualquier periodo k, se asume que $h_k(y, Y)$ es una función cóncava no decreciente, con $h_k(0, Y) = 0$ para cualquier $Y \geq 0$. Teniendo en cuenta esta última suposición, está claro que $h_k(y, Y) \leq h_k(y, X)$, para cualquier nivel de inventario $X < Y$. Además, si se asume que $h_k(\cdot) = h(\cdot)$, para todo k, es decir, funciones de coste idénticas a lo largo de todo el horizonte de planificación, entonces también se tiene que $h_k(y, Y) \leq h_{k'}(y, X)$, para cualquier $1 \leq k \leq k' \leq n$.

Por otro lado, teniendo en cuenta la actual definición de tasa de deterioro en (2.6), se tiene que $0 \leq \alpha_{it} < 1$ (para cualquier par $(i,t) : 1 \leq i \leq t \leq n$), y entonces $\lim_{\alpha_{it} \to 1} A^i_{kt} = \infty$ carecería de sentido para una diferencia $t - i$ suficientemente grande. En consecuencia, teóricamente tomaría un número infinito de periodos retirar una cantidad y_{it} del inventario por obsolescencia, manteniendo cierta similitud con con la paradoja de Zenón sobre Aquiles y la tortuga. La paradoja plantea una situación en la que Aquiles, el guerrero más rápido, compite en una carrera contra una tortuga más lenta pero que parte con ventaja. Supongamos que la tortuga tiene una ventaja inicial de cierta distancia sobre Aquiles. Para que Aquiles alcance a la tortuga, primero debe llegar al punto donde estaba la tortuga al principio. Sin embargo, cuando llega a ese punto, la tortuga se ha desplazado de nuevo y deberá desplazarse hasta ese nuevo punto y así sucesivamente. Parece que Aquiles siempre debe recorrer una distancia infinita, para lo que necesitará un tiempo infinito. Por este motivo, Zenón argumentó que Aquiles nunca podría alcanzar a la tortuga.

Por lo tanto, las *Suposiciones* 1 y 2 en [13] o bien podrían no implementarse o bien no cumplirse en la prática. Observemos además que para cada par de periodos $(i,t) : 1 \leq i \leq t \leq n$, la función de coste de mantenimiento desagregada $H_{it}(y_{it})$ puede computarse como $H_{it}(y_{it}) = \sum_{k=i}^{t} h_k(y_{it}, Y_k)$, para una secuencia apropiada de niveles de inventario Y_k. A continuación, nos centraremos en abordar el caso particular de una vida útil fija en lugar de inventarios dependientes de la edad.

2.2.2. Formulación del problema con vida útil conocida: un modelo para bancos de sangre

A partir de ahora, cada vez que usemos el término *producto* nos estaremos refiriendo a unidades de sangre o productos hemoderivados. Como ya adelantamos en la introducción y se muestra en [7], los productos hemoderivados se deterioran con el paso del tiempo, aunque para su uso práctico se mantienen en buen estado durante su vida útil m (expresada como un número fijo de periodos), por esta razón, no es necesario retirarlos antes de que ésta misma acabe. Es decir, en lugar de mantener una fracción $(1 - \alpha_{it})$ de y_{it}, se mantiene el lote hasta su fecha de vencimiento, al final del periodo $t = i + m - 1$ y, por tanto, $\alpha_{i,t} = 0$, para $i \leq t < i + m - 1$, y $\alpha_{i,t} = 1$ cuando $t = i + m - 1$. Observe que la suposición $\alpha_{i,t} \geq \alpha_{j,t}$, para $1 \leq i \leq j \leq t \leq n$, se mantiene en este caso, incluso cuando $1 \leq i \leq t - m + 1$ y $t - m + 1 < j < t$. Es más, los ratios de deterioro se pueden omitir añadiendo una restricción de obsolescencia adecuada, como se muestra en la formulación del problema P', que se deriva de manera trivial del problema P.

$$P' : min \sum_{t=1}^{n} (C_t(x_t) + \sum_{i=1}^{t} H_{it}(y_{it})) \tag{2.7}$$

$s.a$:

$$x_t - z_{tt} = y_{tt}, \qquad 1 \leq t \leq n \tag{2.8}$$

$$y_{i,t-1} - z_{it} = y_{it}, \qquad \text{máx}\{1, t - m + 1\} \leq i < t \leq n \tag{2.9}$$

$$\sum_{i=1}^{t} z_{it} = d_t, \qquad 1 \leq t \leq n \tag{2.10}$$

$$x_t, y_{it}, z_{it} \geq 0, \qquad 1 \leq i \leq t \leq n \tag{2.11}$$

No obstante, para $1 \leq i \leq t \leq n$, las variables z_{it} se pueden omitir en P' porque se asume que la demanda d_t de productos hemoderivados en cualquier periodo t se satisface con el inventario almacenado al final del periodo anterior. Para ello, y_t denotará ahora el inventario disponible al final del periodo t en lugar del inventario inicial. Nótese que y_t podría ser considerado como la suma de las variables y_{it} en [13], para $i = \text{máx}\{1, t - m + 1\}, \ldots, t$. Por otra parte, se asume sin pérdida de generalidad que $y_0 = d_1$ y $y_n = 0$.

Basándonos en las críticas al modelo de Hsu planteadas anteriormente y por simplicidad, denotaremos por $H_t(y)$ al coste marginal de mantener y unidades de producto repuestas desde cualquier periodo i, $\text{máx}\{1, t - m + 1\} \leq i \leq t$, hasta el final del periodo t, de tal forma que $H_{it}(y = y_{it}) = \sum_{l=i}^{t} H_l(y)$ podría ser considerado como el coste desagregado de mantener y_{it} unidades al final del periodo t.

Al contrario que en [13], en este caso los costos de mantenimiento no dependen de la edad del stock. Sin embargo, debido a que se aplica una política FIFO, cuanto más tiempo se mantenga un lote de producto en inventario, más alto podría ser el costo marginal de inventario. Asumiremos nuevamente que las funciones de costo de reabastecimiento, $C_t(\cdot)$, y almacenamiento, $H_t(\cdot)$, son funciones cóncavas no decrecientes para cada $t = 1, \cdots, n$. Además, $D_{tt'}$ representará la demanda acumulada para el producto desde el periodo t hasta el periodo $t' \geq t$, es decir, $D_{tt'} = \sum_{l=t}^{t'} d_l$. En consecuencia, la formulación de este $ELSP$ para un único artículo con vida útil fijada y costes independientes de la edad puede puede expresarse como sigue:

$$P1 : min \sum_{t=1}^{n} (C_t(x_t) + H_t(y_t)) \tag{2.12}$$

$s.a$:

$$y_{t-1} - y_t + x_t = d_t, \qquad 1 \leq t \leq n \tag{2.13}$$

$$y_{t-1} \geq d_t, \qquad 1 \leq t \leq n \tag{2.14}$$

$$y_{t-1} + x_t \leq D_{t,\text{mín}\{n,t+m-1\}}, \qquad 1 \leq t \leq n \tag{2.15}$$

$$x_t, y_t \geq 0, \qquad 1 \leq t \leq n \tag{2.16}$$

La restricción (2.13) es la conocida ecuación de balance de material (conservación de flujo) propia del $ELSP$ clásico. La demanda en cualquier periodo t es cubierta por el inventario disponible al final del periodo anterior, como se expresa en (2.14). Esta restricción garantiza que siempre hay stock suficiente disponible para satisfacer la demanda al comienzo de cada periodo (por ejemplo, para ser utilizado en operaciones quirúrgicas programadas para ese periodo). La restricción (2.15) asegura que ninguna unidad de producto sea desperdiciada debido al vencimiento, al no permitir que el nivel de inventario de ningún periodo t supere la cantidad precisa para cubrir la demanda acumulada hasta el final de su vida útil, es decir $D_{t,\text{mín}\{n,t+m-1\}}$.

Presentamos a continuación una nueva caracterización de las soluciones óptimas del problema $P1$, que lleva a un algoritmo de Programación Dinámica de complejidad $\mathcal{O}(n^2)$. Puesto que $P1$ es un problema de minimización con función objetivo cóncava y su región factible es compacta y no vacía (poliedro definido por las ecuaciones (2.13)-(2.15)), existe una solución óptima al problema $P1$, que es un punto extremo del poliedro.

Teorema 2.4. *Existe una solución óptima Ω^* para el problema P1 tal que $y_{t-1} + x_t = D_{tt'}$ para cada periodo de reposición $1 \leq t \leq n$ y para algún periodo $t \leq t' \leq \text{mín}\{n, t + m - 1\}$.*

Demostración. Teniendo en cuenta que la formulación de $P1$ se corresponde con el modelo $ELSP$ clásico con capacidades de almacenamiento, cuyas soluciones

extremas fueron caracterizadas tanto por Love [16] como por Gutiérrez et al. [10], la demostración sigue de manera trivial.

El Teorema 2.4 da como resultado los Corolarios 2.5, 2.6 y 2.7.

Corolario 2.5. *Existe una solución óptima Ω^* para $P1$ en la que $y_t = D_{tt'}$ para $1 \leq t \leq t' \leq n$.*

Corolario 2.6. *El ELSP uni-artículo de vida útil fija y costes de mantenimiento independientes de la edad se reduce al ELSP clásico con capacidades de almacenamiento dado en $P1$.*

Corolario 2.7. *Si retirásemos el conjunto de restricciones (2.14), existiría una solución óptima Ω^* para $P1$ que sería puramente ZIO.*

De la caracterización dada en el Teorema 2.4 se puede deducir la Propiedad 2.8, que es una adaptación de la propuesta por Love en [16] al contexto del problema $P1$. Con este fin, el concepto de *punto de inventario* o *periodo de inventario* debe ser redefinido. Un periodo t se dice que es un periodo de inventario si el nivel de inventario al final del periodo anterior es, o bien d_t, o bien el máximo de capacidad factible. Claramente, el máximo de capacidad factible en un periodo t se define como $D_{t,\text{mín}\{n,t+m-1\}}$, es decir, el máximo inventario para satisfacer la demanda acumulada de como mucho m periodos consecutivos.

Propiedad 2.8. Existe una solución óptima Ω^* de $P1$ en la que hay como mucho un periodo de reposición entre dos periodos de inventario consecutivos.

Basándonos en la Propiedad 2.8, podemos diseñar una nueva expresión de recursión de Programación Dinámica. Por consiguiente, para $1 \leq i < t \leq n$, denotaremos por $f_{it}(u, v)$ al costo total de reabastecimiento y mantenimiento del problema $P1$ restringido a los periodos desde i hasta t, en los que los periodos extremos se asumen periodos de inventario, y en los que $u, v \in \{0, 1\}$ indican, respectivamente, los niveles de inventario iniciales de i y t. Para ser más precisos, el inventario inicial en el periodo t es $y_{t-1} = d_t$ cuando $v = 0$, e $y_{t-1} = D_{t,\text{mín}\{n,t-1+m-1\}}$ en caso contrario. Denotando máx$(t) =$ mín$\{n, t+m-1\}$, para $t = 1, ..., n$ y máx$(0) = 1$, el costo total de reabastecimiento y mantenimiento $f_{it}(u, v)$ se define como:

$f_{it}(u, v) =$

$$\begin{cases} C_i(D_{i+1,t}) + \sum_{l=i}^{t-1} H_l(D_{it} - D_{il}), & \text{si } u = v = 0 \text{ y } t \leq \text{máx}(i), \\ C_i(D_{i+1,\text{máx}(i)}) + H_i(D_{i+1,\text{máx}(i)}), & \text{si } u = 0, v = 1 \text{ y } t = i+1 < n, \\ \sum_{l=i}^{t-1} H_l(D_{it} - D_{il}), & \text{si } u = 1, v = 0 \text{ y } t = \text{máx}(i-1), \\ \sum_{l=i}^{t-2} H_l(D_{l+1,\text{máx}(i-1)}) & \\ +H_{t-1}(D_{t,\text{máx}(t-1)}) & \\ + C_{t-1}(D_{t,\text{máx}(t-1)} - D_{t,\text{máx}(i-1)}), & \text{si } u = v = 1 \text{ y } 1 < i < t \leq \text{máx}(i) < n, \\ \infty, & \text{en otro caso} \end{cases} \tag{2.17}$$

Ahora, si denotamos por $F_i(u)$ al coste óptimo del problema $P1$ empezando en el periodo i con nivel de inventario inicial u y finalizando en el periodo n, la recursión hacia atrás puede expresarse de la siguiente manera:

$$F_i(u) = \min_{\substack{i<t\leq \text{máx}(i) \\ v\in\{0,1\}}} \{f_{it}(u,v) + F_t(v)\} \tag{2.18}$$

Estableciendo las condiciones iniciales $F_n(0) = 0$ y $F_n(1) = \infty$, el costo mínimo para el problema $P1$ se obtiene cuando se resuelve $F_1(0)$. Obsérvese que haremos que $D_{t,t'} = 0$, if $t > t'$, en el cálculo de las demandas acumuladas del caso $u = v = 1$ en (2.17). Dado que se asume que cada valor $C_t(x)$ y $H_t(y)$ se obtiene en tiempo constante, para $1 \leq t \leq n$ y $x, y \geq 0$, calcular cada $\sum_{l=i}^{t-1} H_l(\cdot)$ en (2.17) llevaría $\mathcal{O}(n)$ tiempo. En consecuencia, determinar todos los valores $f_{it}(u,v)$, para $1 \leq i < t \leq n$ y $u, v \in \{0,1\}$, consumiría un tiempo de orden $\mathcal{O}(n^3)$. De acuerdo a esto, una implementación directa de (2.18) conduce a un algoritmo de complejidad $\mathcal{O}(n^3)$.

Sin embargo, esta complejidad se puede reducir a $\mathcal{O}(n^2)$ si, para cada par (i,t) de periodos, los valores para $H_{i,t} = \sum_{l=i}^{t} H_l(D_{l+1,t} = D_{it} - Di, l)$ son previamente calculados y almacenados en un tiempo y espacio $\mathcal{O}(n^2)$. Así, fijando $H_{i,i} = 0$ para $1 \leq i \leq n$, los valores restantes $(1 \leq i < t \leq n)$ se calculan aplicando la expresión recursiva en (2.19).

$$H_{i,t} = H_i(D_{i+1,t}) + H_{i+1,t} \tag{2.19}$$

Por lo tanto, cuando $u = v = 1$, si denotásemos para cada par de periodos (i,t) el coste acumulado de inventario, $\sum_{l=i}^{t-2} H_l(D_{l+1,\text{máx}(i-1)})$, por $\widehat{H}_{i,t} = H_{i,\text{máx}(i-1)} - H_{t-1,\text{máx}(i-1)}$, las expresiones para $f_{it}(u,v)$ en (2.17) se podrían reescribir de la siguiente manera:

$f_{it}(u,v) =$

$$\begin{cases} C_i(D_{i+1,t}) + H_{i,t} & \text{if } u = v = 0 \text{ and } t \leq \text{máx}(i), \\ C_i(D_{i+1,\text{máx}(i)}) + H_i(D_{i+1,\text{máx}(i)}) & \text{if } u = 0, v = 1 \text{ and } t = i + 1 < n, \\ H_{i,t} & \text{if } u = 1, v = 0 \text{ and } t = \text{máx}(i-1), \\ \widehat{H}_{i,t} + H_{t-1}(D_{t,\text{máx}(t-1)}) + & \text{if } u = v = 1 \text{ and } 1 < i < t \leq \text{máx}(i) < n, \\ + C_{t-1}(D_{t,\text{máx}(t-1)} - D_{t,\text{máx}(i-1)}) & \\ \infty & \text{en otro caso} \end{cases} \tag{2.20}$$

Esta nueva formulación reduce, por lo tanto, la complejidad temporal del algoritmo en [13] de $\mathcal{O}(n^4)$ a $\mathcal{O}(n^2)$. Esta mejora computacional coincide con la del algoritmo propuesto por Hwang y Van den Heuvel en [14] para el $ELSP$ con capacidades de almacenamiento y costes cóncavos, aunque estos autores se valieron de un algoritmo de búsqueda en estructuras de datos ligeramente más sofisticada para su método solución, como son las matrices Monge. Es más, en caso de costes no especulativos, es decir, costes de reabastecimiento fijos más lineales y costes de mantenimiento lineales como en [26], en los que, para todo par (i, t) tal que $1 \leq i < t \leq n$, se cumple que $C_i(x) + H_{i,t}(x) \geq C_t(x)$, pueden usarse algoritmos más eficientes (como, por ejemplo, los propuestos en Gutiérrez et al. [11] y también en [14]), que resuelven de manera óptima el problema en tiempo amortizado lineal. Para el $ELSP$ con capacidades de almacenamiento y costes de reabastecimiento fijos más lineales y costos de mantenimiento lineales hay algoritmos, como los introducidos en [24], [13], [2], [19] y [20] (caso sin capacidad con un mecanismo de asignación FIFO), que también determinan la polítca óptima de inventario en tiempo $\mathcal{O}(n^2)$.

2.3. Ejemplo numérico y algoritmo de programación dinámica

A continuación, en la Tabla 2.1, se recogen los parámetros de un simple ejemplo con $n = 6$ periodos y vida útil $m = 3$. Los valores $f_{it}(u, v)$, para $1 \leq i < t \leq n$ y $u, v \in \{0, 1\}$, que se muestran en la Tabla 2.2, se han obtenido aplicando (2.20), mientras que los valores $F_i(u)$, mostrados en Tabla 2.3, se han calculado usando (2.18).

Funciones	**Demandas**		
$C_t(x_t) = 5\sqrt{x_t}$	$d_1 = 8$	$d_2 = 6$	$d_3 = 9$
$H_t(y_t) = 2\sqrt[3]{y_t}$	$d_4 = 10$	$d_5 = 12$	$d_6 = 7$

Tabla 2.1. Datos del problema.

$f_{it}(u,v)$	$u=v=0$	$u=0,v=1$	$u=1,v=0$	$u=v=1$
$i=1,t=2$	15.881	24.297	∞	∞
$i=1,t=3$	28.457	∞	∞	∞
$i=2,t=3$	19.160	27.131	4.160	21.148
$i=2,t=4$	31.440	∞	∞	33.216
$i=3,t=4$	20.120	29.056	4.308	22.924
$i=3,t=5$	33.635	∞	∞	31.440
$i=4,t=5$	21.899	27.131	4.578	∞
$i=4,t=6$	30.957	∞	∞	∞
$i=5,t=6$	17.054	∞	3.825	∞

Tabla 2.2. Valores de la función $f_{it}(u,v)$.

$F_i(u)$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
$u=0$	78.279	62.397	50.689	30.957	17.054	0
$u=1$	66.031	54.849	35.266	21.633	3.825	∞

Tabla 2.3. Valores de $F_i(u)$.

	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
x	-	6	19	0	19	0	0
y	8	6	19	10	19	7	0

Tabla 2.4. Cantidades óptimas de pedido y nivel de inentario en cada periodo .

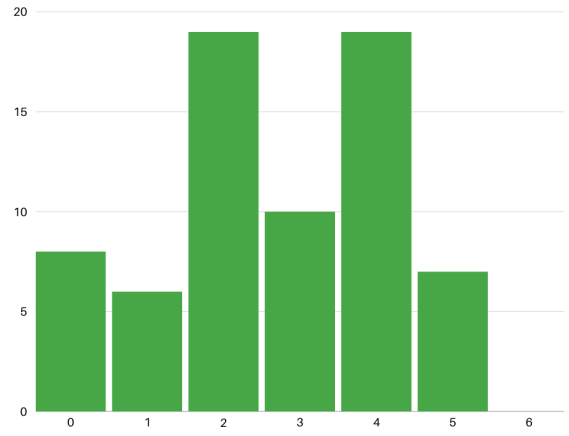

Figura 2.1. Nivel de inventario al final del periodo i

El coste óptimo final es $F_1(0) = 78.279$ y las cantidades óptimas de pedido y los correspondientes niveles de inventario se muestran en la tabla 2.4. Asimismo, las fluctuaciones del nivel de inventario a lo largo del horizonte temporal de 6 periodos se representan en la Figura 2.1.

Presentamos ahora el pseudocódigo de nuestro algoritmo de Programación Dinámica.

Algorithm 1: An $\mathcal{O}(n^2)$ DP algorithm to $P1$

Data: n and, for $t = 1 \dots n$, d_t, $C_t(\cdot)$ and $H_t(\cdot)$
Result: $F_1(0)$ and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$
Initialize $\mathbf{x}$ and $\mathbf{y}$;
$f_{it}(u, v) \leftarrow \infty$, for $1 \leq i < t \leq n$ and $(u, v) \in \{0, 1\}^2$;
$F_{n+1}(0) \leftarrow 0$ and $F_{n+1}(1) \leftarrow \infty$;
for $i \leftarrow 1$ *to* n **do**
  for $t \leftarrow i + 1$ *to* máx(i) **do**
    for $u \leftarrow 0$ *to* 1 **do**
      for $v \leftarrow 0$ *to* 1 **do**
        $f_{it}(u, v)$ is computed according to (2.20);
      end
    end
  end
end
for $i \leftarrow n$ *downto* 1 **do**
  for $u \leftarrow 0$ *to* 1 **do**
    $F_i(u) \leftarrow \infty$;
    for $t \leftarrow i + 1$ *to* $n + 1$ **do**
      for $v \leftarrow 0$ *to* 1 **do**
        if $F_i(u) > f_{it}(u, v) + F_t(v)$ **then**
          $F_i(u) \leftarrow f_{it}(u, v) + F_t(v)$;
          $x[i][u] \leftarrow (D_{jk}, t, v)$ from (2.20) depending on the value of pair (u, v);
          $y[i][u] \leftarrow (D_{j'k'}, t, v)$ from (2.20) depending on the value of pair (u, v);
        end
      end
    end
  end
end
Return $F_1(0)$, $\mathbf{x}$ and $\mathbf{y}$;

Conclusiones

En este trabajo hemos abordado el problema $ELSP$ uni-artículo perecedero con vida útil fija. Específicamente, hemos querido mostrar su aplicación al caso real de un banco de sangre, en el que los productos con los que se trabaja son utilizables durante un periodo conocido de tiempo regulado por protocolos sanitarios.

Hemos partido del modelo de Hsu presentado en [13], que consideraba productos perecederos que se deterioran con el paso del tiempo y costes cóncavos de mantenimiento que dependen de la edad del producto. Desde nuestro punto de vista, el modelo de Hsu hace uso de dos suposiciones que, cuando menos, son cuestionables y que hemos tratado de abordar. La primera de ellas tiene que ver con la definición de su función de coste de mantenimiento, que hemos modificado al tener la percepción de que la suya podría interpretarse como un coste desagregado. Concretamente, esta suposición considera que el coste marginal de mantener hasta un periodo t una cantidad producida en un periodo anterior i es mayor que el de mantener esa misma cantidad, si fuese producida en un periodo posterior j. Sin embargo, esta definición de coste de mantenimiento no tiene en cuenta el nivel de inventario al final de cada periodo t, que claramente influiría en su cálculo, e incluso podría no cumplirse en el caso de periodos inflacionarios. La segunda crítica se centra en la definición de la tasa de deterioro, puesto que genera una situación paradójica en la que tomaría un número infinito de periodos retirar por obsolescencia una cantidad del inventario.

Nuestras principales contribuciones en esta memoria han sido adaptar el modelo de Hsu al caso de artículos con vida útil fija, como lo son la sangre y sus derivados, y demostrar que el nuevo problema se puede reducir a un $ELSP$ con capacidades de almacenamiento. Es importante destacar que si en el modelo no se exige inventario inicial en cada periodo, se puede encontrar una solución ZIO y que, en cualquier caso, los niveles de inventario al final de cada periodo se corresponden con suma de demandas de periodos consecutivos.

Aprovechando las propiedades estructurales de las políticas óptimas del nuevo modelo, hemos diseñado un algoritmo de Programación Dinámica. Nues-

tro algoritmo es de complejidad $\mathcal{O}(n^2)$, lo que representa una mejora considerable con respecto al algoritmo de orden $\mathcal{O}(n^4)$ propuesto por Hsu. Cabe destacar que Hwang y Van den Heuvel en [14] diseñaron un algoritmo de igual complejidad que el nuestro, que puede ser usado también para resolver el nuevo problema. No obstante, su algoritmo requiere trabajar con una matriz inversa de Monge, mientras que el nuestro utiliza una simple fórmula de recursión para calcular los costes de mantenimiento.

A pesar de haber demostrado que el nuevo modelo con vida útil fija se puede reducir a un *ELSP* con capacidades de almacenamiento y de haber propuesto un método solución que rebaja la complejidad del algoritmo de Hsu, nuestro modelo aún presenta ciertas limitaciones en cuanto a su aplicabilidad en casos reales. En primer lugar, el carácter determinista de los parámetros podría cuestionar su idoneidad para casos en los que los parámetros estén sujetos a variaciones aleatorias o a la incertidumbre. De manera similar, las suposiciones de la reposición instantánea y del tiempo de retardo despreciable podrían no tener en cuenta limitaciones prácticas en algunos escenarios. Por último, la restricción de considerar un solo artículo puede no reflejar la realidad común de trabajar con varios artículos, cada uno con características distintas.

Quedan aspectos pendientes para líneas de trabajo futuras, como la incorporación de parámetros estocásticos para mejorar la adaptabilidad en entornos dinámicos, la consideración de tiempos de retardo significativos y limitaciones prácticas en la reposición, así como el desarrollo de un modelo que pueda gestionar eficientemente varios productos, teniendo en cuenta sus características individuales. Estas mejoras podrían enriquecer significativamente la aplicabilidad del modelo y su capacidad para abordar situaciones más realistas y complejas en la gestión de inventarios.

A

Apéndice

A.1. Código para el ejemplo de P1 con n=6

A continuación se muestra detalladamente el código usado para computar el algoritmo del ejemplo de P1 con $n = 6$. Antes de seguir, cabe destacar que hemos incluido un periodo vacío al principio de vectores como el de demanda d puesto que PYTHON numera este primer elemento del vector como 0 y nuestro problema empieza en el periodo 1, aunque por pantalla siempre mostraremos los vectores sin este primer elemento, a excepción del vector y_t que representa el nivel de inventario al final de cada periodo, donde $y_0 = d_1$.

A.1.1. Datos del código

Aquí introduciremos los datos del problema, siendo d la demanda en cada periodo, m los periodos de vida útil del producto y n el número de periodos de planificación. *Dac* es la demanda acumulada del periodo i hasta el final del horizonte de planificación. Además de los datos principales creamos varias funciones como $MAX(i)$, que se corresponde con la función máx(i) definida por máx(i) = mín$\{n, i+m-1\}$ para $i > 0$ y máx(0) = 1 o $D(a, b)$, que nos devolverá la demanda acumulada entre los periodos a y b. Las siguientes dos funciones se corresponden con las de coste de producción $C_t(x)$ y de mantenimiento $H_t(x)$, que en este ejemplo son iguales en todos los periodos, aunque en otros podrían no serlo. La última función llamada aquí H_h representa $H_{i,t}(x) = H_i(D_{i+1,t})+H_{i+1,t}$, definida en (2.19) para reducir la complejidad del algoritmo de $\mathcal{O}(n^3)$ a $\mathcal{O}(n^2)$.

```
d = [None, 8.0, 6.0, 9.0, 10.0, 12.0, 7.0]
m = 3
n = 6
Dac = [0] * (n+2)
for i in reversed(range(n+1)):
    if i>0:
```

```
    Dac[i] = round(Dac[i+1]+ d[i], 3)
d1 = d[1:]
Dac1 = Dac[1:]
print(f"Demanda en cada periodo: {d1}")
print(f"Demanda acumulada del periodo i al final: {Dac1}"
    )

def MAX(i):
  if i == 0:
    return 1
  else:
    return min(n, i+m-1)

def D(a,b):
  return Dac[a] - Dac[b+1]


C=[]
def c(x):
  return 5*(x)**(1/2)
for i in range(0,n+1):
  C.append(c)


H=[]
def h(x):
  return 2*(x)**(1/3)
for i in range(0,n+1):
  H.append(h))

H_h = [[0 for _ in range(n+1)] for _ in range(n+1)]
for i in range(1,n+1):
  H_h[i][i] = 0
for i in range(n-1,0,-1):
  for t in range(i+1,n+1):
    H_h[i][t] = H[i](D(i+1, t)) + H_h[i+1][t]
```

Esta forma de definir C_t y H_t no es necesaria en este ejemplo concreto y podríamos simplemente definir una sola función C y una H. No obstante, queremos que se contemple el caso en el que estas funciones puedan ser distintas en cada periodo t. Para la función Dac redondeamos a las 3 primeras cifras decimales por cuestiones prácticas que tienen que ver con la capacidad de memoria de PYTHON para números con muchos decimales.

Lo que veremos por pantalla será lo siguiente:
Demanda en cada periodo: [8.0, 6.0, 9.0, 10.0, 12.0, 7.0]
Demanda acumulada del periodo i al final: [52.0, 44.0, 38.0, 29.0, 19.0, 7.0, 0]

A.1.2. Función f_{it} y almacenamiento de los valores correspondientes

En esta parte definimos la función $f_{it}(u, v)$ de (2.17) y almacenaremos los valores resultantes necesarios para la recursión del algoritmo en una lista que contendrá los valores de i, t, u, v y el de aplicar la función $f_{it}(u, v)$ con estos parámetros, mostrándolos también por pantalla a medida que se calculan.

```
def f(i,t,u,v):
  if u==0 and v==0 and t <= MAX(i):
    return C[i](D(i+1,t)) + H_h[i][t]
  elif u==0 and v==1 and t == i+1 < n:
    return C[i](D(i+1,MAX(i))) + H[i](D(i+1, MAX(i)))
  elif u == 1 and v == 0 and t == MAX(i-1):
    return H_h[i][t]
  elif u == 1 and v == 1 and 1 < i < t <= MAX(i) < n:
    return H_h[i][MAX(i-1)] - H_h[t-1][MAX(i-1)] + H[t
        -1](D(t, MAX(t-1))) + C[t-1](D(t,MAX(t-1)) - D(t,
        MAX(i-1)))
  else:
    return float('inf')


f_it = []
counter = 1
for i in range(1, n ):
    for t in range(i + 1, MAX(i) + 1):
        for u in range(0, 2):
            for v in range(0, 2):
              print(f"{counter}:f(i,t,u,v)=f({i},{t},{u
                  },{v}):{f(i,t,u,v)}")
              f_it.append((i,t,u,v,f(i,t,u,v)))
              counter += 1
```

Esto es lo que veremos por pantalla:
1: f(i,t,u,v) = f(1, 2, 0, 0): 15.88168989958017
2: f(i,t,u,v) = f(1, 2, 0, 1): 24.297340879698023
3: f(i,t,u,v) = f(1, 2, 1, 0): inf

4: f(i,t,u,v) = f(1, 2, 1, 1): inf
5: f(i,t,u,v) = f(1, 3, 0, 0): 28.45750852580183
6: f(i,t,u,v) = f(1, 3, 0, 1): inf
7: f(i,t,u,v) = f(1, 3, 1, 0): inf
8: f(i,t,u,v) = f(1, 3, 1, 1): inf
9: f(i,t,u,v) = f(2, 3, 0, 0): 19.160167646103808
10: f(i,t,u,v) = f(2, 3, 0, 1): 27.13129801514726
11: f(i,t,u,v) = f(2, 3, 1, 0): 4.160167646103808
12: f(i,t,u,v) = f(2, 3, 1, 1): 21.148191598285788
13: f(i,t,u,v) = f(2, 4, 0, 0): 31.440167395211027
14: f(i,t,u,v) = f(2, 4, 0, 1): inf
15: f(i,t,u,v) = f(2, 4, 1, 0): inf
16: f(i,t,u,v) = f(2, 4, 1, 1): 33.21632510653173
17: f(i,t,u,v) = f(3, 4, 0, 0): 20.120257680905667
18: f(i,t,u,v) = f(3, 4, 0, 1): 29.056157460427922
19: f(i,t,u,v) = f(3, 4, 1, 0): 4.308869380063768
20: f(i,t,u,v) = f(3, 4, 1, 1): 22.924586736999544
21: f(i,t,u,v) = f(3, 5, 0, 0): 33.63501443064125
22: f(i,t,u,v) = f(3, 5, 0, 1): inf
23: f(i,t,u,v) = f(3, 5, 1, 0): inf
24: f(i,t,u,v) = f(3, 5, 1, 1): 31.440167395211027
25: f(i,t,u,v) = f(4, 5, 0, 0): 21.899365045902098
26: f(i,t,u,v) = f(4, 5, 0, 1): 27.13129801514726
27: f(i,t,u,v) = f(4, 5, 1, 0): 4.5788569702133275
28: f(i,t,u,v) = f(4, 5, 1, 1): inf
29: f(i,t,u,v) = f(4, 6, 0, 0): 30.957160380692038
30: f(i,t,u,v) = f(4, 6, 0, 1): inf
31: f(i,t,u,v) = f(4, 6, 1, 0): inf
32: f(i,t,u,v) = f(4, 6, 1, 1): inf
33: f(i,t,u,v) = f(5, 6, 0, 0): 17.054618920867732
34: f(i,t,u,v) = f(5, 6, 0, 1): inf
35: f(i,t,u,v) = f(5, 6, 1, 0): 3.825862365544778
36: f(i,t,u,v) = f(5, 6, 1, 1): inf

A.1.3. F_i y resolución del problema

En esta última parte del código definiremos la función que permite calcular la cantidad de pedido en cada periodo y el algoritmo de recursión F_i de (2.18). Mostrándose por pantalla los valores de $F_i(0)$, $F_i(1)$, además de las soluciones del costo óptimo $F_1(0)$ y la cantidad de pedido óptimo $\mathbf{x}$ en cada periodo i.

```
def X_i(pos,i,t):
    if pos == 0:
        return D(i+1, t)
    elif pos == 1:
        return D(i+1, MAX(i))
    elif pos == 2:
        return 0
    elif pos == 3:
        return D(t, MAX(t-1)) - D(t, MAX(i-1))
    else:
        return None

X = np.zeros((n+1,2,3))

F_i = [[float('inf')] * (n+1) for _ in range(2)]
F_i[0][n] = 0
F_i[1][n] = float('inf')

for i in range(n-1,0,-1):
  for t in range(i+1, MAX(i) + 1):
    for u in range(0,2):
      for v in range(0,2):
        for j in range(len(f_it)-1, -1, -1):
          if f_it[j][0] == i and f_it[j][1] == t and f_it
             [j][2] == u and f_it[j][3] == v:
            if F_i[u][i] > f_it[j][4] + F_i[v][t]:
              F_i[u][i] = f_it[j][4] + F_i[v][t]
              case = 2*u+v
              if case < 3:
                X[i][u][0] = X_i(case,i,t)
                X[i][u][1] = t
                X[i][u][2] = v
              else:
                X[t-1][u][0] = X_i(case,i,t)
                X[t-1][u][1] = t
                X[t-1][u][2] = v
def Sol(tam):
    it = int(1)
    w = int(0)
    sol = np.zeros(n+1)
    while it != 0:
        sol[it] = X[it][w][0]
```

```
            w1 = int(X[it][w][2])
            it = int(X[it][w][1])
            w = w1
        return sol

x = Sol(n)
x[0] = None
y = [None]*(n+1)
y[0] = d[1]
for k in range(1,n+1):
  y[k]=y[k-1]+x[k]-d[k]

F_i0 = F_i[0][1:n+1]
F_i1 = F_i[1][1:n+1]
x1 = x[1:]
print(f"F_i(0) = {F_i0}")
print(f"F_i(1) = {F_i1}")
print(f"El costo pptimo de este ejemplo para el problema
    P1 es F_1(0) = {F_i[0][1]}")
print(f"La cantidad a pedir en cada periodo es: {x1}")
print(f"La cantidad de inventario disponible al final de
    cada periodo es: {y}")
```

Esto es lo que veremos por pantalla:
$F_i(0) = [78.27901767548323, 62.397327775903065, 50.68963335150898, 30.957160380692038, 17.054618920867732, 0]$
$F_i(1) = [\text{inf}, 54.84980099761279, 35.266029760755806, 21.63347589108106, 3.825862365544778, \text{inf}]$
El costo óptimo de este ejemplo para el problema $P1$ es $F_1(0) = 78.27901767548323$
La cantidad a pedir en cada periodo es: [6. 19. 0. 19. 0. 0.]
La cantidad de inventario disponible al final de cada periodo es: [8.0, 6.0, 19.0, 10.0, 19.0, 7.0, 0.0]

Bibliografía

[1] ABBASI, B. y VAKILI, G. y CHESNEAU, S. Impacts of Reducing the Shelf Life of Red Blood Cells: A View from Down Under. *Interfaces*, 2017, vol. 47, No. 4, pp. 336–351.

[2] ATAMTURK, A. y KÜCÜKYAVUZ, S. An $\mathcal{O}(n^2)$ algorithm for lot sizing with inventory bounds and fixed costs. *Operations Research Letters*, 2008, vol. 36, No. 3, pp. 297–299.

[3] BAZARAA, M. S. y SHETTY, C. M. *Nonlinear Programming - Theory and Algorithms*. John Wiley & Sons, Nueva York, 1979.

[4] CANADIAN BLOOD SERVICE. *BloodNotes June 2016 Understanding Hospital Inventory*. 2016. [Fecha de consulta: 17-02-2023]. Disponible en: https://www.blood.ca/en/node/99942.

[5] CHAO, X. y GONG, X. y SHI, C. y ZHANG, H. Approximation Algorithms for Perishable Inventory Systems. *Operations Research*, 2015, vol. 63, No. 3, pp. 585–601.

[6] COOK, L. S., RN, MSRN, CRN. Tratamientos con hemoderivados. *Nursing*, 2009, vol. 27, No. 9.

[7] COOPER, D. J. y MCQUILTEN, Z. K. y NICHOL, A. y ADY, B. y AUBRON, C. y BAILEY, M.lÂ¡ y BELLOMO, R. y GANTNER, D. e IRVING, D. O. y KAUKONEN, K. y MCARTHUR, C. y MURRAY, L. y PETTILÄ, V. y FRENCH, C. Age of Red Cells for Transfusion and Outcomes in Critically Ill Adults. *New England Journal of Medicine*, 2017, vol. 377, No. 19, pp. 1858–1867.

[8] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959, vol. 1, pp. 269–271.

[9] FRIEDMAN, Y. y HOCH, Y. A Dynamic Lot Size Model with Inventory Deterioration. *INFOR*, 1978, vol. 16, pp. 183–188.

[10] GUTIÉRREZ, J. y SEDEÑO-NODA, A. y COLERBROOK, M. y SICILIA, J. A new characterization for the dynamic lot size problem with bounded inventory. *Computers & Operations Research*, 2002, vol. 30, pp. 383–395.

[11] GUTIÉRREZ, J.M y ABDUL-JALBAR, B. y SICILIA, J. y RODRÍGUEZ-MARTÍN, I. Effective Algorithms for the Economic Lot-Sizing Problem with Bounded Inventory and Linear Fixed-Charge Cost Structure *Mathematics*, 2021, vol. 9, pp. 689.

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

[13] HSU, V. N. Dynamic Economic Lot Size Model with Perishable Inventory. *Management Science*, 2000, vol. 46, No. 8, pp. 1159–1169.

[14] HWANG, H.C. y VAN DEN HEUVEL, W. Improved algorithms for a lot-sizing problem with inventory bounds and backlogging. *Naval Research Logistics (NRL)*, 2012, vol. 59, No. 3–4, pp. 244–253.

[15] JING, F. y CHAO, X. A dynamic lot size model with perishable inventory and stockout. *Omega*, 2021, vol. 103, pp. 102421.

[16] LOVE, S. F. Bounded production and inventory models with piecewise concave costs. *Management Science*, 1973, vol. 20, No. 3, pp. 313–318.

[17] NAHMIAS, S. Optimal Ordering Policies for Perishable Inventory II. *Operations Research*, 1975, vol. 23, No. 4, pp. 735–749.

[18] NAHMIAS, S. Perishable Inventory Systems. *International Series in Operations Research & Management Science*, 2011, vol. 160, pp. 49–54.

[19] ÖNAL, M. y VAN DEN HEUVEL, W. y LIU, T. A note on "The economic lot sizing problem with inventory bounds" *European Journal of Operational Research*, 2012, vol. 223, No. 1, pp. 209–294.

[20] ÖNAL, M. y ROMEIJN, H.E. y SAPRA, A. y VAN DEN HEUVEL, W. The economic lot-sizing problem with perishable items and consumption order preference. *European Journal of Operational Research*, 2015, vol. 244, No. 3, pp. 881–891.

[21] PIERSKALLA, W. P. y ROACH, C. D. R. Optimal Issuing Policies for Perishable Inventory. *Management Science*, 1972, vol. 18, No. 11, pp. 603–614.

[22] ROCKAFELLAR, R. T. *Network Flows and Monotropic Optimization.* John Wiley & Sons, Nueva York, 1984.

[23] SMITH, L.A. Simultaneous Inventory and Pricing Decisions for Pperishable Commodities with Pprice Fluctuation Constraints. *INFOR*, 1975, vol. 13, pp. 82–87.

[24] TOCZYLOWSKI, E. An $\mathcal{O}(T^2)$ algorithm for the lot-sizing problem with limited inventory levels. *INRIA/IEEE Symposium on Emerging Technologies and Factory Automation, Paris, France*, 1995, vol. 3, pp. 77–85.

[25] VEINOTT, A.F. JR. *Optimal Ordering Issuing and Disposal of Inventory with Known Demand. Unpublished doctoral dissertation*, 1969.

[26] WAGNER, H.M. y WHITIN, T.M. Dynamic Version of the Economic Lot Size Model. *Management Science*, 1958, vol. 5, pp. 89–96.

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

[28] ZANGWILL, Willard I. From EOQ towards ZI. *Management Science*, 1987, vol. 33, No. 10, pp. 1209–1223.

Dynamic lot-sizing model under perishability. Application to blood products.

Pablo Javier Sánchez Santana
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101158495@ull.edu.es

Abstract

We address the dynamic Economic Order Quantity (EOQ) or Economic Lot Size Problem (ELSP) for perishable products with a known lifespan, specifically applied to the context of a blood bank. Drawing from previous works, we have developed a model that is tailored to the inventory management of a blood product, and propose an Dynamic Programming algorithm with significantly lower computational complexity than the baseline model used for its resolution.

1. Introduction

The objective of the *ELSP* is to minimize the total production and inventory costs. Our model considers a n-period planning horizon for a single perishable item with a m-period lifetime. As we manage hemoderivate products used for medical purposes, backordering is not allowed. We assume that functions for both the replenishment and the inventory costs are general nondecreasing concave functions. An $\mathcal{O}(n^2)$ algorithm is devised, so reducing the $\mathcal{O}(n^4)$ time complexity of the solution method for the baseline model.

2. The model

Now, we proceed to introduce the meaning of the notation used in our model, for each period $t = 1, \cdots, n$:

- $d_t > 0 :=$ known demand for a product in period t.
- $D_{tt'} :=$ cumulative demand from period t through period t', i.e. $D_{tt'} = \sum_{l=t}^{t'} d_l$.
- $x_t :=$ replenishment volume of product in period t.
- $y_t :=$ amount of inventory available at the end of period t. Assuming without loss of generality that $y_0 = d_1$ and $y_n = 0$.
- $C_t(x_t) :=$ cost of replenishing x_t units of product in period t.
- $H_t(y) :=$ marginal cost of holding y units of inventory from some period i, $\max\{1, t-m+1\} \leq i \leq t$, up to the end of period t.

$$P1 : minimize \sum_{t=1}^{n} (C_t(x_t) + H_t(y_t)) \tag{1}$$

$s.t.$

$$y_{t-1} - y_t + x_t = d_t, \qquad 1 \leq t \leq n \tag{2}$$
$$y_{t-1} \geq d_t, \qquad 1 \leq t \leq n \tag{3}$$
$$y_{t-1} + x_t \leq D_{t,\min\{n,t+m-1\}}, \qquad 1 \leq t \leq n \tag{4}$$
$$x_t, y_t \geq 0, \qquad 1 \leq t \leq n \tag{5}$$

Constraints in (2) are the material balance equations related to the *ELS* problems. The demand in any period t is covered by the inventory available at the end of the previous period, as expressed in (3). Constraint set (4) ensures that no hemoderivate unit is wasted by expiration.
As there exists an optimal solution Ω^* to problem $P1$ such that $y_{t-1} + x_t = D_{tt'}$ for each period production period $1 \leq t \leq n$ and for some period $t \leq t' \leq \min\{n, t+m-1\}$, the *ELS* Problem with fixed lifetime and age-independent holding costs reduces to the classical *ELS* problem with Storage Capacities given in $P1$.

3. The Dynamic Programming algorithm

Our algorithm is based on the following Property (Love, 1973): There exists an optimal solution Ω^* to $P1$ in which there is at most one production period between two adjacent inventory periods.
For for $1 \leq i < t \leq n$, we denote:

- $f_{it}(u, v) :=$ the total replenishment and inventory cost of the problem $P1$ restricted to periods i through t
- $u, v \in \{0, 1\} :=$ respectively, the initial inventory levels of i and t. ($y_{t-1} = d_t$ when $v = 0$, and $y_{t-1} = D_{t,\min\{n,t-1+m-1\}}$ otherwise).
- $\max(t) := \min\{n, t+m-1\}$, $t = 1, \cdots, n$, and $\max(0) = 1$.
- $H_{i,t} := H_i(D_{i+1,t}) + H_{i+1,t}$.
- $\widehat{H}_{i,t} := H_{i,\max(i-1)} - H_{t-1,\max(i-1)}$ if $u = v = 1$.

$f_{it}(u, v)$ is defined as follows:

$$\begin{cases} C_i(D_{i+1,t}) + H_{i,t} & \text{if } u = v = 0 \ \& \ t \leq \max(i) \\ C_i(D_{i+1,\max(i)}) + H_i(D_{i+1,\max(i)}) & \text{if } u = 0, v = 1 \ \& \ t = i+1 < n \\ H_{i,t} & \text{if } u = 1, v = 0 \ \& \ t = \max(i-1) \\ \widehat{H}_{i,t} + H_{t-1}(D_{t,\max(t-1)}) + & \text{if } u, v = 1 \ \& \\ + C_{t-1}(D_{t,\max(t-1)} - D_{t,\max(i-1)}) & 1 < i < t \leq \max(i) < n \\ \infty & \text{otherwise} \end{cases} \tag{6}$$

Now, if we denote by $F_i(u)$ the optimal cost of the problem $P1$ starting in period $1 \leq i < \max(i)$ with initial inventory level u and ending in period n, the Dynamic Programming backward recursion can be expressed as:

$$F_i(u) = \min_{\substack{i<t\leq\max(i) \\ v\in\{0,1\}}} \{f_{it}(u, v) + F_t(v)\} \tag{7}$$

Setting the initial conditions $F_n(0) = 0$ and $F_n(1) = \infty$, the minimum cost to problem $P1$ is calculated when $F_1(0)$ is solved.

References

[1] HSU, V. N. Dynamic Economic Lot Size Model with Perishable Inventory. *Management Science*, 2000, vol. 46, No. 8, pp. 1159–1169.

[2] NAHMIAS, S. Perishable Inventory Systems. *International Series in Operations Research & Management Science*, 2011, vol. 160, pp. 49–54.

[3] LOVE, S. F. Bounded production and inventory models with piecewise concave costs. *Management Science*, 1973, vol. 20, No. 3, pp. 313–318.