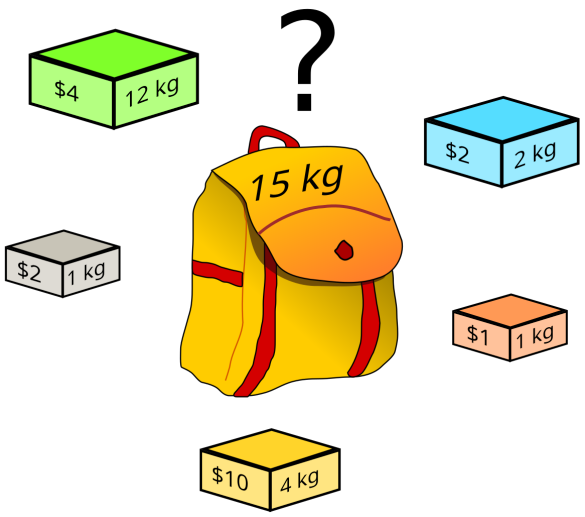
Universidad Alfonso X El Sabio

Grado en Ingeniería Matemática
Técnicas de Optimización y Control

Caso Práctico II



Martina García González

9 de enero de 2025

Abstract

En Técnicas de Optimización, se plantea este caso práctico enfocado en la planificación de una misión espacial. El objetivo es optimizar, mediante programación dinámica, la selección de los artículos que la tripulación debe transportar en su nave, garantizando su seguridad y el cumplimiento de las restricciones de peso y capacidad. El problema se aborda mediante dos enfoques: maximizar la utilidad total y maximizar la cantidad de objetos, respetando restricciones de peso y seguridad. Además de implementar ambos modelos, se realiza una comparativa detallada de su rendimiento y viabilidad. Finalmente, se proponen algoritmos alternativos que pueden emplearse para resolver problemas similares de tipo mochila.

Adicionalmente, se ha entregado un archivo `.ipynb` donde se encuentra todo el código implementado.

Índice

1. Introducción

El problema de la mochila es un modelo de optimización en el que disponemos de una serie de objetos, cada uno con un peso y una utilidad. El objetivo principal es seleccionar qué objetos incluir en una mochila con capacidad limitada, tratando de maximizar, por lo general, la utilidad total de los objetos que llevamos en ella.

1.1. Variantes del Problema de la Mochila

Existen tres variantes principales del problema de la mochila :

- **0/1 Knapsack Problem :** Cada objeto puede seleccionarse o no ($x_i \in \{0, 1\}$).
- **Bounded Knapsack Problem (Mochila con cantidad limitada):** Permite incluir hasta una cantidad máxima c_i de cada objeto ($x_i \in \{0, 1, \ldots, c_i\}$).
- **Unbounded Knapsack Problem (Mochila con cantidad ilimitada):** No hay límite en el número de copias de cada objeto ($x_i \in N$).

1.2. Complejidad Computacional

El problema de la mochila es computacionalmente complejo debido al número de combinaciones posibles. Con n objetos, que podemos decidir si incluir o no (0 ò 1), tendríamos un total de 2^n combinaciones. Esto significa que para nuestro caso $n = 20$, habría que evaluar $2^{20} = 1,048,576$ combinaciones, un número inmanejable para resolver por fuerza bruta en un tiempo razonable.

Para ilustrar esta complejidad, en la Figura 1, se presenta el árbol de decisiones para un caso reducido con $n = 3$. En este ejemplo, cada nivel del árbol representa la decisión de incluir (1) o no incluir (0) un objeto, generando un total de $2^3 = 8$ combinaciones posibles al final de las ramas.

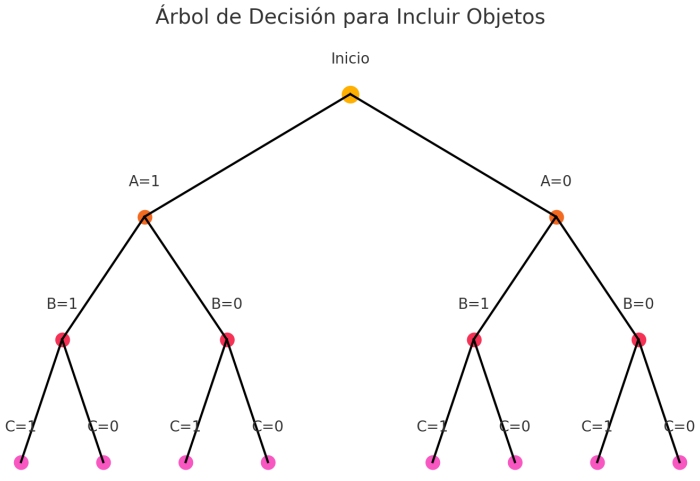


Figura 1: Árbol de decisiones para $n = 3$ en el problema de la mochila.

1.3. Métodos para Resolver el Problema

Para resolver el problema de la mochila, existen dos enfoques principales.

1.3.1. Algoritmos Exactos

Método	Descripción
Programación Dinámica	Divide el problema en subproblemas más pequeños y calcula sus soluciones óptimas de forma iterativa.
Complejidad	$O(nW)$, donde n es el número de objetos y W la capacidad máxima de la mochila.
Ventajas	Garantiza encontrar la solución óptima.
Limitaciones	Puede ser lento si W es muy grande.

1.3.2. Métodos de Aproximación

Método	Descripción
Algoritmos Greedy	Elige la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima.
Complejidad	$O(n \log n)$, debido al ordenamiento inicial.
Ventajas	Son rápidos y fáciles de implementar.
Limitaciones	No garantizan la solución óptima.

1.3.3. Complejidades Métodos

En esta gráfica se comparan las complejidades computacionales de los enfoques .

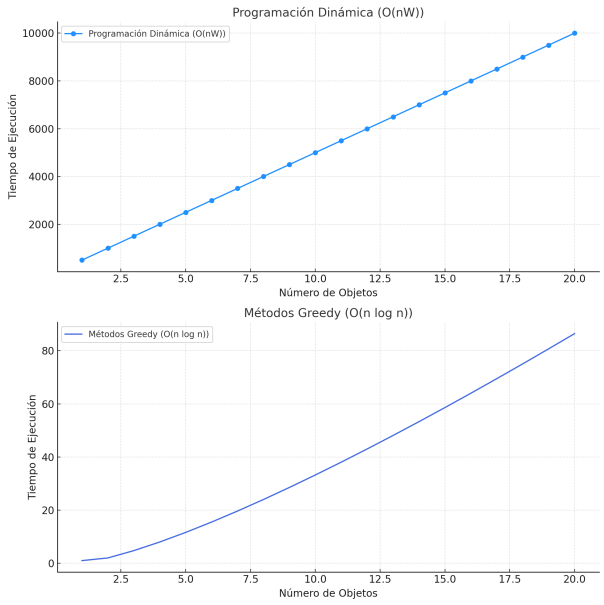


Figura 2: Complejidades Programación Dinámica Vs Algoritmos Greedy

1.4. Fuerza Bruta y Computación Cuántica

Resolver el problema evaluando todas las combinaciones posibles (fuerza bruta) garantiza encontrar la solución óptima, pero es un proceso extremadamente largo debido a la gran variedad de soluciones que deberíamos evaluar.

La **computación cuántica** abre nuevas posibilidades para abordar problemas como el de la mochila. Gracias a su capacidad para explorar múltiples combinaciones simultáneamente, podría reducir significativamente el tiempo necesario para encontrar soluciones. Aunque esta tecnología aún se encuentra en desarrollo, representa una herramienta prometedora para resolver problemas que, actualmente, son intratables debido a su complejidad.

1.5. Aplicaciones Problema de la Mochila

El problema de la mochila tiene aplicaciones en diversas áreas, como:

- **Logística:** Optimización de recursos y espacio en almacenes o envíos.
- **Criptografía:** Implementación de sistemas de cifrado basados en su complejidad.

2. Planteamiento del Caso Prático

En el contexto de las misiones espaciales, la planificación eficiente de los recursos que se pueden transportar en la nave es una tarea vital para garantizar el éxito de la misión. Este problema se modela como una variante del **problema de la mochila**, con las siguientes características:

- Cada producto tiene un peso específico, una utilidad asociada y una cantidad máxima permitida.
- La carga total no puede exceder una capacidad máxima de 500 kg.
- Se debe garantizar que el Índice de Seguridad de la Relación (ISR) sea mayor o igual a 1, asegurando la estabilidad y seguridad del sistema.

El objetivo principal es analizar dos enfoques de optimización:

1. **Maximizar la utilidad total:** Elegir los objetos que ofrezcan el mayor beneficio total cumpliendo con las restricciones.
2. **Maximizar el peso total:** Seleccionar los objetos para llenar la mochila lo máximo posible.

***Adicionalmente, se proporcionará un archivo `.ipynb` que incluye la implementación de los algoritmos, y simulaciones detalladas con los resultados obtenidos.

2.1. Optimización por Utilidad

2.1.1. Modelado Matemático

El problema de optimización por utilidad busca maximizar la utilidad de los objetos transportados en la nave, respetando las restricciones de peso y seguridad. Matemáticamente, el problema se formula como:

$$\text{Maximizar: } Z = \sum_{i=1}^{n} v_i x_i$$

Sujeto a:

$$\sum_{i=1}^{n} w_i x_i \leq W \quad \text{(Restricción de peso)}$$

$$\text{ISR} = \frac{w_{\text{total}}}{w_{\text{carga}}} \geq 1 \quad \text{(Restricción de seguridad)}$$

Donde:

$$w_{\text{total}} = w_{\text{transbordador}} + w_{\text{combustible}} + w_{\text{carga}}$$

$$w_{\text{carga}} = \sum_{i=1}^{n} w_i x_i$$

$$x_i \in \{0, 1, 2, \ldots, c_i\} \quad \forall i = 1, 2, \ldots, n \quad \text{(Restricción de cantidad)}$$

Parámetros:

- v_i: Utilidad asociada al objeto i.
- w_i: Peso del objeto i.
- c_i: Cantidad máxima permitida del objeto i.
- W: Capacidad máxima de la mochila (500kg).
- $w_{\text{transbordador}}$: Peso del transbordador vacío.
- $w_{\text{combustible}}$: Peso del combustible necesario para el viaje.
- w_{carga}: Peso total de los objetos seleccionados.
- n: Número total de objetos disponibles.
- x_i: Número de unidades seleccionadas del objeto i.

2.1.2. Coste Computacional

El problema de optimización propuesto, al ser resuelto mediante programación dinámica, tiene un coste computacional elevado. Esto se debe a que la complejidad del algoritmo depende tanto del número de objetos como de la capacidad de la mochila, siendo aproximadamente $O(n \cdot W)$, donde n es el número de objetos y W es la capacidad máxima de peso.

Durante el desarrollo de este problema, me plantée la siguiente pregunta : **¿sería posible simplificar el modelo para reducir el elevado coste computacional?** Una estrategia intuitiva podría ser ordenar los objetos por utilidad y, después, llenar la mochila empezando por los más útiles. A primera vista, esta idea parece razonable y sencilla, pero ¿es realmente efectiva? Aunque es aparente una respuesta sencilla y válida, esta aproximación presenta varias limitaciones:

- **No garantiza una solución óptima:** Aunque se logra reducir el coste computacional, el algoritmo podría omitir combinaciones que resulten en un mayor valor total, ya que prioriza únicamente un criterio (la utilidad).
- **Ignora el peso relativo:** Este enfoque puede seleccionar objetos de gran utilidad pero con un peso muy elevado, dejando sin espacio objetos más ligeros que podrían maximizar la utilidad total.

Para superar estas desventajas, existen los llamados **algoritmos greedy**. Estos algoritmos simplifican el problema considerando la relación **utilidad/peso** (v_i/w_i) como criterio principal para ordenar los objetos. La mochila se rellena empezando por los objetos con mayor relación utilidad/peso, hasta alcanzar la capacidad máxima. Si bien los algoritmos greedy ofrecen una solución rápida, no siempre garantizan la solución óptima. En la sección de *Algoritmos Alternativos*, se analiza en detalle este tipo de algoritmos.

2.2. Resultados de la Optimización por Utilidad

Producto	**Cantidad**	**Peso Total (kg)**	**Utilidad Total**
Paneles Solares	3	36	12
Baterías de Litio	5	35	10
Sistemas de Navegación	2	22	6
Módulos de Comunicación	4	32	20
Kits de Primeros Auxilios	2	40	14
Equipos de Investigación	1	14	6
Alimentos Liofilizados	6	36	12
Trajes Espaciales	2	26	10
Tanques de Oxígeno	1	15	8
Herramientas de Reparación	3	30	12
Agua Potable	7	70	7
Sistema de Purificación de Agua	1	16	9
Componentes Electrónicos	1	8	3
Generadores de Energía	3	27	15
Antenas de Comunicación	2	24	12
Sensores de Monitoreo	2	22	10
Módulos de Soporte Vital	1	14	7
Dispositivos de Análisis	3	30	12
Total	**49**	**497 kg**	**185**

Cuadro 1: Resultados de la optimización por utilidad.

2.2.1. Aplicabilidad

El enfoque de optimización basado en la utilidad es especialmente útil en contextos donde **se prioriza el valor de los objetos** seleccionados frente a su cantidad. Un ejemplo práctico es el **equipamiento para expediciones científicas**, donde se priorizan instrumentos de mayor precisión y utilidad para maximizar el valor de la investigación con recursos limitados.

2.3. Optimización por Cantidad de Objetos

2.3.1. Modelado Matemático

La optimización por cantidad de objetos tiene como objetivo maximizar el número total de objetos transportados, sin considerar las utilidades asociadas a ellos. Este enfoque prioriza la cantidad sobre el valor.

Matemáticamente, este problema utiliza el mismo algoritmo que la optimización por utilidad, con la diferencia de que se considera que la utilidad de todos los objetos es igual a 1 ($v_i = 1$ para todo i). De esta manera, la función objetivo queda definida como:

$$\text{Maximizar: } Z = \sum_{i=1}^{n} x_i$$

Sujeto a las mismas restricciones:

$$\sum_{i=1}^{n} w_i x_i \leq W \quad \text{(Restricción de peso)}$$

$$\text{ISR} = \frac{w_{\text{total}}}{w_{\text{carga}}} \geq 1 \quad \text{(Restricción de seguridad)}$$

Donde:

$$w_{\text{total}} = w_{\text{transbordador}} + w_{\text{combustible}} + w_{\text{carga}}$$

$$w_{\text{carga}} = \sum_{i=1}^{n} w_i x_i$$

$$x_i \in \{0, 1, 2, \ldots, c_i\} \quad \forall i = 1, 2, \ldots, n \quad \text{(Restricción de cantidad)}$$

En este caso, al asignar un valor unitario ($v_i = 1$) a todos los objetos, la función objetivo busca maximizar la cantidad total de objetos seleccionados.

2.3.2. Coste Computacional

El algoritmo empleado para la optimización por cantidad de objetos es idéntico al del apartado de optimización por utilidad, salvo que en este caso se generaliza la utilidad de todos los objetos a un valor unitario ($v_i = 1$ para todo i). Por lo tanto, el **coste computacional sigue siendo el mismo**, $O(W \cdot n)$.

Para **reducir este coste computacional**, se podrían considerar algoritmos greedy, que seleccionan objetos de manera iterativa según un criterio simplificado, como el peso más bajo en este caso. Estos algoritmos son computacionalmente más eficientes, ya que operan en tiempo $O(n \log n)$.

En este caso particular, donde el objetivo es **maximizar únicamente la cantidad de objetos transportados**, el uso de un **algoritmo greedy proporciona la solución más adecuada**. Este enfoque simplificado asegura que se maximice el número total de objetos seleccionados al priorizar aquellos de menor peso. La implementación y análisis de este algoritmo se detalla en la sección *Algoritmos Alternativos*.

2.4. Resultados de la Optimización por Peso

Producto	**Cantidad**	**Peso Total (kg)**	**Utilidad Total**
Paneles Solares	3	36	12
Baterías de Litio	5	35	10
Sistemas de Navegación	2	22	6
Módulos de Comunicación	4	32	20
Kits de Primeros Auxilios	2	40	14
Equipos de Investigación	1	14	6
Alimentos Liofilizados	6	36	12
Trajes Espaciales	2	26	10
Tanques de Oxígeno	1	15	8
Herramientas de Reparación	3	30	12
Agua Potable	7	70	7
Sistema de Purificación de Agua	1	16	9
Materiales de Construcción	2	14	4
Medicamentos	6	36	12
Antenas de Comunicación	2	24	12
Sensores de Monitoreo	2	22	10
Dispositivos de Análisis	3	30	12
Total	**52**	**498 kg**	**176**

Cuadro 2: Resultados de la optimización por peso.

2.4.1. Adaptabilidad

El enfoque de optimización basado en el peso es útil en contextos donde se priorice m**aximizar la cantidad de objetos transportados, independientemente de su valor**. Un ejemplo práctico sería el aprovisionamiento para refugios en situaciones de emergencia, donde se busca transportar la mayor cantidad de suministros para beneficiar al mayor número de personas posible

3. Comparativa de Métodos de Optimización

3.1. Análisis de los Resultados Obtenidos

A continuación se presentan las diferencias clave entre los dos modelos de optimización:

Modelo	Optimización por Utilidad	Optimización por Cantidad
Utilidad Total	185	176
Cantidad de Objetos Seleccionados	49	52
Peso Total Transportado (kg)	497	498
Variedad de Objetos Seleccionados	18	17

Cuadro 3: Comparación entre modelos de optimización

Aunque ambos modelos son similares en cuanto al peso transportado (solo 1 kg de diferencia), el modelo de optimización por utilidad ofrece una ventaja considerable en el valor total, con un 5.1 % más que el modelo por cantidad de objetos.

Por otro lado, el modelo de optimización por cantidad selecciona más productos (52 frente a 49).Sin embargo, en cuanto a la variedad de objetos, el modelo de optimización por utilidad selecciona más tipos de productos diferentes (18 frente a 17). Esto destaca que, aunque el modelo por cantidad selecciona más objetos, la diversidad de productos es menor. En una misión espacial, lo que realmente le interesaría al astronauta no es simplemente llevar más cantidades de los mismos objetos, sino tener una amplia gama de productos que puedan ser más útiles en situaciones variadas, asegurando así un equipamientomás completo

En nuestro modelo, debido a la distribución relativamente uniforme de la utilidad y el peso de los productos, no se observa una gran diferencia entre los resultados de ambos modelos. Para mostrar mejor las diferencias que se pueden obtener entre ambas optimizaciones, se presenta un ejemplo en el archivo `.ipynb` *Mochila Diaria Astronauta*, donde se puede apreciar como variaría la selección de los artículos.

3.1.1. Viabilidad Técnica y Costes

En ambos casos de optimización, utilizamos programación dinámica, lo que implica que el **coste computacional es el mismo**.vSin embargo, en el caso de la optimización centrada únicamente en maximizar la cantidad de objetos transportados, se podría aplicar un algoritmo *greedy*. Este siempre seleccionará la mayor cantidad de objetos posible, priorizando los objetos de menor peso.

Desde mi punto de vista, utilizaría este algoritmo *greedy* solo si las utilidades de los objetos son muy similares o si hay una gran cantidad de objetos disponibles . En estos casos, el algoritmo *greedy* podría ser una opción debido a su simplicidad y rapidez.

En cuanto a la **viabilidad técnica**, con 20 objetos, resolver el problema utilizando programación dinámica es completamente viable, ya que el tiempo de cálculo es razonable. Sin embargo, si tuviéramos muchos más objetos, sería necesario considerar otros tipos de algoritmos que tengan un coste computacional más bajo. Estos algoritmos no aseguran encontrar la solución óptima, pero pueden ser útiles cuando se prioriza la eficiencia.

En conclusión,si la cantidad de objetos a transportar es considerable, se debe priorizar la eficiencia, lo que implica sacrificar algo de precisión utilizando algoritmos más rápidos como el *greedy*. Por otro lado, si el número de objetos es pequeño y la precisión es crucial, recomedaría utilizar programación dinámica para asegurar una solución óptima, aunque con un coste computacional mayor.

3.1.2. Gráficos Ilustrativos

A continuación, aunque los resultados de ambos modelos son muy similares, se presentan unos gráficos para resumir el análisis anterior.

Comparación de Peso y Utilidad por Modelo

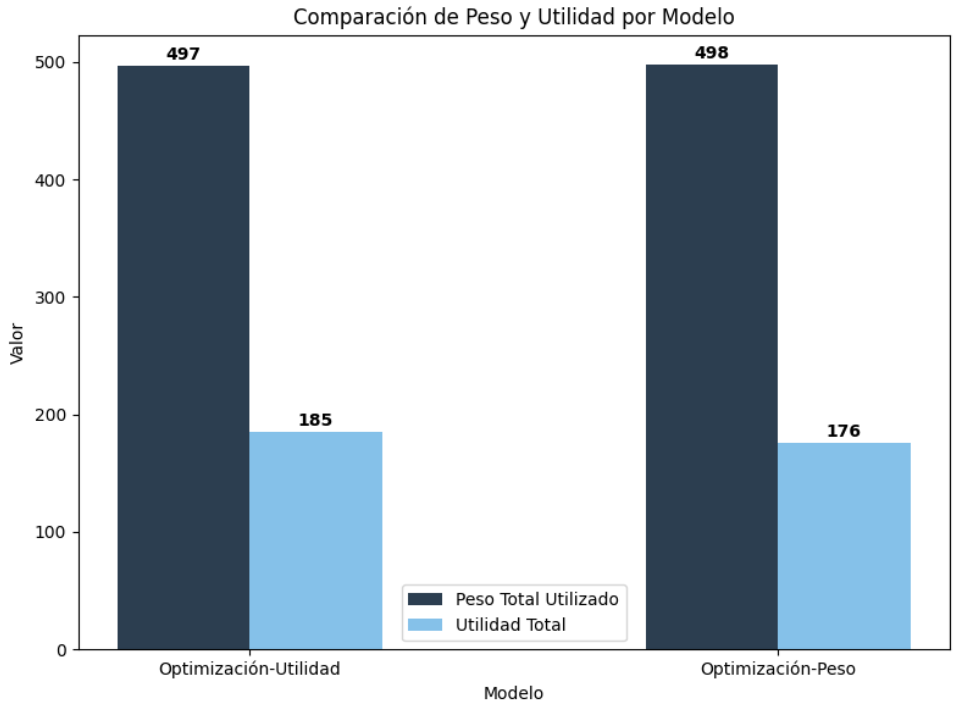


Figura 3: Comparación de Peso y Utilidad por Modelo.

Distribución de Productos en Cada Modelo

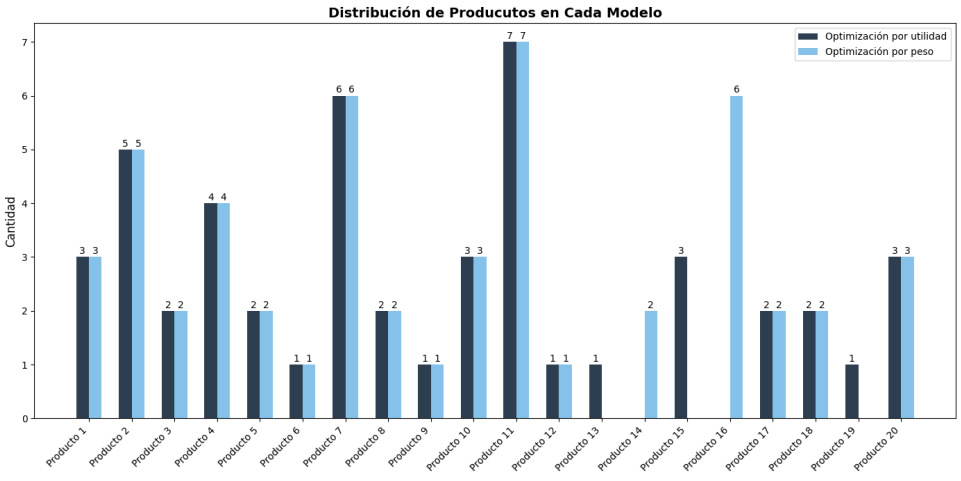


Figura 4: Distribución de Productos en Cada Modelo.

3.1.3. Elección Modelo de Optimización

Si fuera el astronauta encargado de elegir los objetos para la nave espacial, escogería la **optimización por utilidad** debido a los siguientes motivos:

- **Mayor Utilidad Total:** El modelo de optimización por utilidad ofrece una ventaja clara en términos del valor de los objetos transportados. A pesar de transportar una cantidad ligeramente menor de objetos (49 frente a 52), la utilidad total es un 5.11 % mayor, lo que garantiza que los objetos seleccionados sean más útiles para la misión.

- **Enfoque en lo más útil:** En una misión espacial, los recursos son limitados y es crucial asegurar el máximo valor de cada objeto transportado. La optimización por utilidad permite maximizar el valor de los recursos disponibles, lo que es mucho más importante que simplemente llevar una mayor cantidad de objetos. En este sentido, la optimización por cantidad no resulta del todo adecuada, ya que lo que realmente importa para el astronauta es disponer de una mayor variedad de objetos esenciales para la misión, no simplemente transportar más cantidad. Priorizar la cantidad sin considerar la diversidad de objetos podría dejar fuera recursos críticos y útiles.

4. Algoritmos Alternativos

Aunque hemos resuelto este problema utilizando **programación dinámica**, existen otros algoritmos que podrían contemplarse como alternativas.A continuación, se describen algunos de los algoritmos más relevantes que se podrían considerar para resolver el problema .

4.1. Branch and Bound

Este algoritmo lo hemos estudiado en la asignatura de Investigación Operativa. Aunque el árbol de decisiones podría no explorarse completamente gracias a las técnicas de poda, con $n = 20$ objetos el tamaño del árbol sigue siendo tan grande que sería inviable. Para este caso, necesitaríamos realizar muchas relajaciones lineales y explorar un número elevado de ramas, lo cual hace que este algoritmo no sea práctico en este casoo. Aunque no hemos profundizado en este tipo de algoritmos, se sabe que no serían eficientes en este problema debido a la alta complejidad y el tamaño del árbol de decisiones.

4.2. Algoritmo Greedy- Voraz

El algoritmo **Greedy** es una estrategia de búsqueda que consiste elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. Este enfoque es rápido y eficiente, pero no garantiza una solución óptima en todos los casos.

En nuestro caso, hemos planteado un algoritmo greedy para cada tipo de optimización, que se pueden contemplar en el archivo `.ipynb` en la sección *Propuesta Nuevos Algoritmos*. A continuación, se describen brevemente cómo funcionan los algoritmos greedy para la optimización por utilidad y por peso:

4.2.1. Algoritmo Greedy para la Optimización por Utilidad

El algoritmo greedy para la optimización por utilidad sigue estos pasos:

1. **Ordenación:** Los productos se ordenan por la relación $\frac{\text{utilidad}}{\text{peso}}$ de mayor a menor.
2. **Selección:** Se añaden productos a la mochila mientras haya espacio disponible.
3. **Cálculo:** Se calculan el peso total y la utilidad total de los productos seleccionados.

4.2.2. Algoritmo Greedy para la Optimización por Cantidad de Objetos

Para la optimización por peso, el algoritmo sigue los siguientes pasos:

1. **Ordenación:** Los productos se ordenan por peso de menor a mayor
2. **Inicialización:** Se crea un diccionario `seleccionados` en cual se irán introuduciendo los anteriores objetos siempre empezando por los de menor peso hasta agotar estos recursos.Se también inicializa el peso acumulado para supervisar que no se exceda la capacidad máxima de la móchila.
3. **Selección:** Se añaden productos a la mochila, priorizando los de menor peso, hasta llenar la mochila .Y se devuelven los objetos selccionados junto con sus respectivos pesos.

4.3. Aplicaciones de los Algoritmos Greedy

En este apartado, hemos aplicado un algoritmo **Greedy** diseñado específicamente para maximizar la cantidad de objetos transportados, y lo hemos comparado con los resultados obtenidos mediante un enfoque de **Programación Dinámica**. Los resultados se resumen en la siguiente tabla:

Métrica	**Greedy**	**Programación Dinámica**
Peso total utilizado (kg)	493	498
Cantidad de objetos seleccionados	57	52

Cuadro 4: Comparación entre el algoritmo Greedy y la Programación Dinámica.

Al analizar estos resultados, se observa que:
- El **algoritmo Greedy**, al priorizar los productos más ligeros, selecciona una mayor cantidad de objetos (57 frente a 52). Esta estrategia garantiza una solución óptima en cuanto a la maximización de la cantidad de objetos transportados.
- Por otro lado, la **Programación Dinámica** optimiza tanto el peso total transportado como la cantidad de objetos seleccionados. Este enfoque logra llenar mejor la capacidad de la mochila (498 kg frente a 493 kg), utilizando el espacio disponible de manera más eficiente.

Este algoritmo **Greedy** para maximizar la cantidad de objetos es útil en casos donde los productos tienen pesos y cantidades muy similares, y no se considera la utilidad. Sin embargo, es fundamental analizar los datos antes de aplicarlo, ya que, si los artículos más ligeros tienen un stock muy grande, podría seleccionarse una muy baja variedad de objetos. Por otro lado, la **programación dinámica**, al construir las tablas *dp* iterando sobre todos los objetos y posibles pesos, siempre garantiza una solución que maximiza tanto la cantidad de objetos como el peso transportado, aprovechando al máximo la capacidad de carga disponible, lo que la hace más adecuada en escenarios complejos.

5. Conclusiones

En conclusión, el problema de la mochila sigue siendo un desafío interesante, ya que, aunque hay muchos algoritmos disponibles, ninguno logra ser perfecto en términos de **eficiencia y solución óptima** para todos los casos. Esto da pié seguir explorando nuevas ideas y enfoques, en un campo que tiene muchísima relevancia y aplicación en el mundo real.

Referencias

[1] Wikipedia contributors. *Knapsack problem - Greedy approximation algorithm*. Wikipedia. Disponible en: `https://en.wikipedia.org/wiki/Knapsack_problem#Greedy_approximation_algorithm`