



## Algoritmos Avanzados

### Laboratorio N°3: Método Backtracking

Integrantes: Christian Méndez  
Israel Arias  
Sección: 0-A-1  
Profesor(a): Aileen Esparza

23 de Julio de 2021

# Tabla de contenidos

<b>1. Introducción</b>	<b>1</b>
1.1. Contexto . . . . .	1
1.2. Motivación . . . . .	1
1.3. Teoría relevante . . . . .	1
1.4. Objetivos . . . . .	2
1.5. Estructura del informe . . . . .	3
<b>2. Método</b>	<b>4</b>
2.1. Recursos utilizados . . . . .	4
2.2. Procedimientos . . . . .	4
2.2.1. Implementación del algoritmo . . . . .	4
2.2.2. Test de la implementación . . . . .	6
2.3. Resultados y análisis . . . . .	6
<b>3. Discusión</b>	<b>10</b>
3.1. Cálculo de complejidad . . . . .	10
3.2. Resultados . . . . .	14
3.3. Recomendaciones . . . . .	15
<b>4. Conclusiones</b>	<b>16</b>
<b>5. Apéndice</b>	<b>17</b>
5.1. Pseudocódigo implementado . . . . .	17
5.2. Instrucciones de uso . . . . .	20
<b>Referencias</b>	<b>22</b>

# **1. Introducción**

## **1.1. Contexto**

En el presente laboratorio, se plantea una situación en la cual, con el fin de abrir un nuevo negocio, tenemos un presupuesto el cual será invertido en productos, también se tiene la información del precio de venta y volumen de cada uno de los productos ofrecidos, por lo que se busca determinar la combinación de productos más lucrativa dado el volumen disponible del container en que serán importados.

## **1.2. Motivación**

Existen muchos algoritmos los cuales son capaces de solucionar problemas tanto industriales como cotidianos, estos algoritmos presentan una alta utilidad en muchos tipos de problemas en los cuales la mente humana no resulta eficiente debido a la alta cantidad de datos que necesitan ser analizados, por ejemplo para el problema planteado es posible el determinar la mejor combinación de productos, lo que pareciera ser una tarea sencilla, sin embargo al darse cuenta la gran cantidad de combinaciones posibles de distintos productos esta tarea parece ser interminable si es ejecutada por una persona, por lo cual resulta más eficiente el modelar computacionalmente el problema, lo que permite resolverlo en tiempos más acotados. La resolución del modelado del problema se efectúa a través de un algoritmo, por lo que el aprendizaje de distintos tipos de algoritmos para la solución de diversas problemáticas se vuelve vital en una sociedad donde los problemas que involucran grandes cantidades de datos se vuelven más comunes.

## **1.3. Teoría relevante**

El algoritmo que se usará para resolver el problema planteado se creará mediante la técnica de backtracking, esta técnica consiste en construir un subconjunto solución incorporando elementos candidatos dado un conjunto de datos, por cada incorporación al subconjunto se verifica si se cumple las condiciones del problema y se repite el proceso hasta llegar a una solución, en caso de que no se llegue a una solución se vuelve atrás y se intenta

con otro elemento. Por ejemplo, para un laberinto que se puede apreciar en la figura 1 el algoritmo para determinar como llegar a la salida del laberinto mediante backtracking consistiría en avanzar por alguna dirección hasta que no sea posible seguir avanzando, en caso de no llegar a la salida se vuelva atrás y se intenta con otra dirección. El método de backtracking se puede resumir como “ es una técnica de programación para hacer búsqueda sistemática a través de todas las configuraciones posibles dentro de un espacio de búsqueda.” [1]

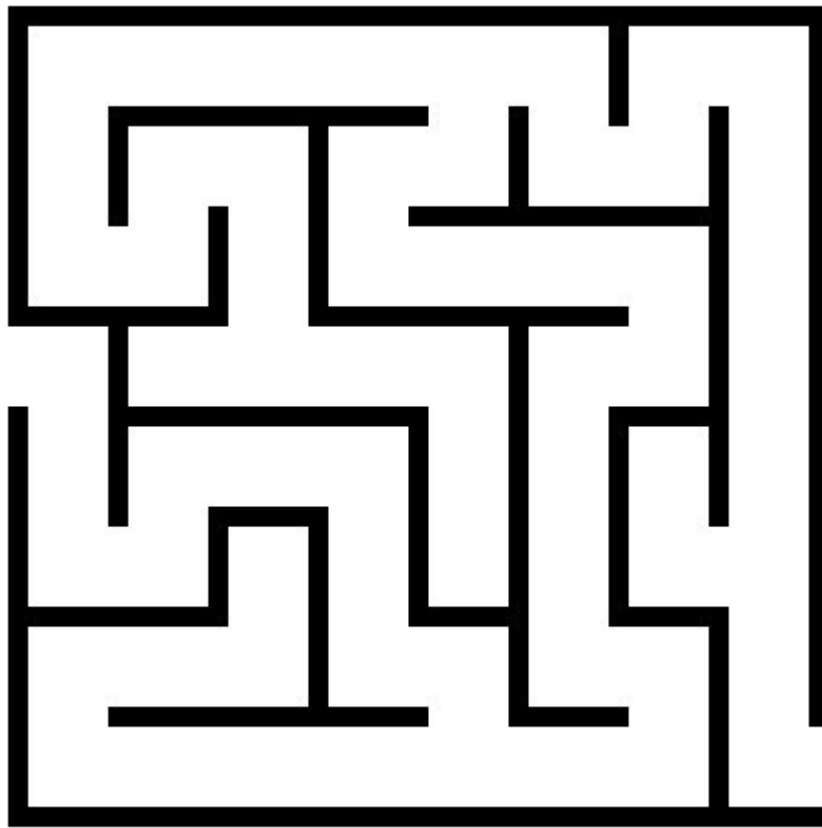


Figura 1: Ejemplo de utilización de Backtracking

#### 1.4. Objetivos

Los objetivos definidos para esta experiencia de laboratorio son:

- Comprensión del método Backtracking.

- Creación de la solución para el problema planteado usando el método Backtracking.
- Implementación de la solución en un programa creado en el lenguaje de programación C.
- Trabajar en pareja de manera efectiva.

## **1.5. Estructura del informe**

Lo presentado hasta ahora ha sido la introducción del informe, a continuación se seguirá con la sección de Método, en la cual se detallaran los recursos utilizados y procedimientos para luego presentar los resultados con su respectivo análisis de estos. Posteriormente se encontrara la sección de Discusión en la cual se se discutirá respecto a el cálculo de complejidad del algoritmo implementado, los resultados conseguidos al implementar el algoritmo y se darán recomendaciones respecto a lo observado. La cuarta sección corresponde a la sección de Conclusiones en la cual se listaran las conclusiones conseguidas de esta experiencia de laboratorio. La quinta sección corresponde al Apéndice, en el cual se adjuntaran anexos que ayuden a la comprensión del problema, algoritmo o de la solución implementada. Finalmente esta la sección de Referencias la cual contiene todas las fuentes que fueron citadas a lo largo del informe.

## 2. Método

### 2.1. Recursos utilizados

Los equipos utilizados para el desarrollo y posteriores pruebas de la solución implementada en el lenguaje de programación C son:

- Intel Core i5 6600, 16GB de ram, S.O: Windows 10.
- Intel Xeon E5 2620 V2, 16GB de ram, S.O: Windows 10.

### 2.2. Procedimientos

#### 2.2.1. Implementación del algoritmo

Para el problema descrito en la introducción, se proporciona un dataset en un archivo .csv el cual contiene toda la información de los paquetes que se ofrecen a la venta, con la información del beneficio monetario que deja cada paquete y el volumen en  $m^3$  que ocupa. Para modelar el problema se implementó la estructura: “Paquete”. La estructura “Paquete” contiene los datos referentes al id, precio, volumen y ponderación de un paquete, cabe destacar que la ponderación consiste en la razón entre el precio de un paquete y el volumen de este. La figura 2 permite observar la idea de cómo se representa un paquete de forma gráfica. Con la estructura paquete definida, se realiza la obtención del conjunto de paquetes solución mediante el siguiente algoritmo: los paquetes validados son incorporados a un arreglo de paquetes, luego este arreglo es ordenado de mayor a menor según la ponderación de los paquetes, además se inicializan dos arreglos del mismo tamaño que el arreglo de paquetes, mochila y mejorMochila. Luego se ejecuta la función recursiva de backtracking en donde en un inicio se llenara el arreglo mochila con los primeros paquetes disponibles en el arreglo de paquetes, esta sera una mochila inicial que contiene los paquetes con mejor ponderación, luego de esto el algoritmo de backtracking buscará “devolverse” sacando el último paquete ingresado, al devolverse calculará el posible beneficio máximo (Posible Beneficio Máximo = beneficio actual + capacidad restante \* ponderación del ítem que se desea agregar) que podría llegar a obtener si agregase el paquete siguiente al que ya fue removido de la mochila, si este

beneficio máximo resulta ser mayor al mejor beneficio actual encontrado, entonces se generan todas las posibles combinaciones que deriven de agregar ese paquete a la actual mochila con paquetes, en caso de que este beneficio no sea mayor al beneficio actual encontrado, no se creara esa combinación ni ninguna de las otras combinaciones derivadas de agregar ese paquete, lo que disminuye el espacio de posibles combinaciones y por ende permite no tener generar todas las posibles combinaciones, mejorando así los tiempos de ejecución. Luego de que el algoritmo recursivo genere todas las posibles mochilas, retornara el arreglo mejorMochila que contendrá un 1 en el índice del paquete que se encuentra en la mejor combinación encontrada. Finalmente se escribe la solución escribiendo en un documento de texto plano, recorriendo el arreglo de mejorMochila que indica con un 1 si el paquete es incorporado a la solución, de esta forma se busca el paquete que posea el id correspondiente a la posición del arreglo, por cada uno de los paquetes encontrados se escribe sus respectivas características. El algoritmo implementado se encuentra detallado en la sección apéndice del presente informe.

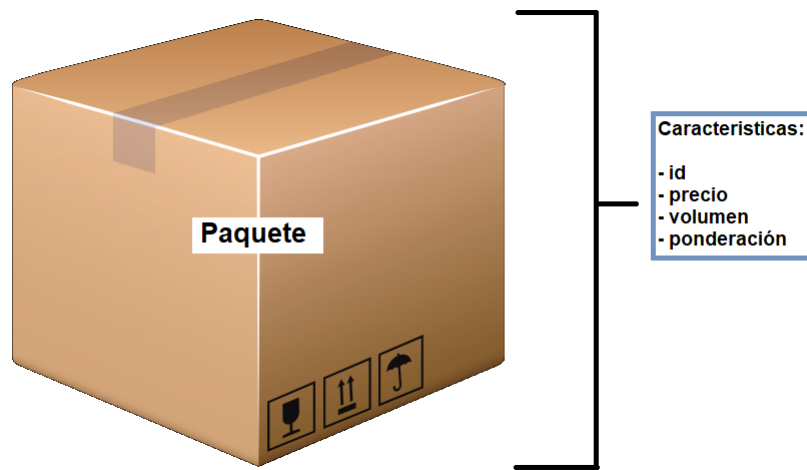
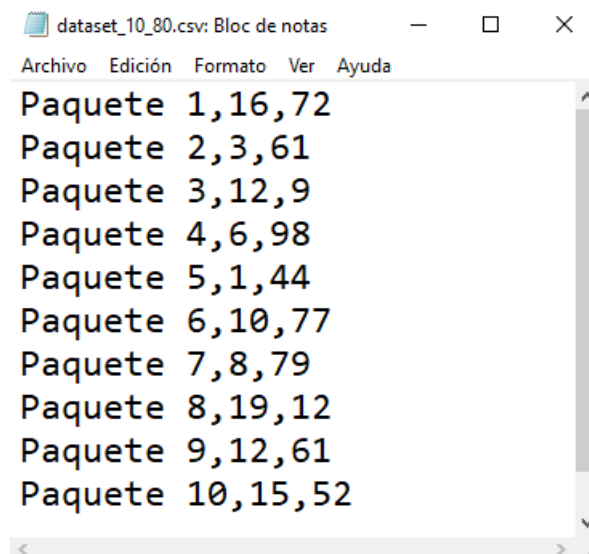


Figura 2: Paquete representado de forma gráfica.

### 2.2.2. Test de la implementación

Para probar la efectividad del programa se dispone de 5 datasets con distinta cantidad de paquetes, en cada uno de los datasets se indica la información relevante de los paquetes, señalando el id, volumen y beneficio de cada uno de los paquetes que posee el dataset. Para cada dataset se registra el tiempo de ejecución que toma el programa implementado en resolver el problema para los datos entregados.



Paquete	id	volumen	beneficio
1	16	72	
2	3	61	
3	12	9	
4	6	98	
5	1	44	
6	10	77	
7	8	79	
8	19	12	
9	12	61	
10	15	52	

Figura 3: Ejemplo de un Dataset de 10 paquetes.

## 2.3. Resultados y análisis

Los resultados son presentados en ambos equipos mediante una tabla y gráfico con sus respectivos tiempos de ejecución en los que se resolvió el problema, para el equipo con procesador **Intel core i5** corresponden las figuras 4 y 5, para el equipo con el procesador **Intel Xeon E5 2620 V2** corresponden las figuras 6 y 7. Además en esta sección se adjuntan los archivos de salida.txt a modo de ejemplo generados por el programa que contienen la mejor combinación para los datasets de 10, 20 y 30 paquetes.



Cantidad de paquetes	Tiempo de ejecución [s]
10	0,001
20	0,002
30	0,001
40	0,002
50	0,002

Figura 4: Tabla al testear en el procesador Intel Core I5

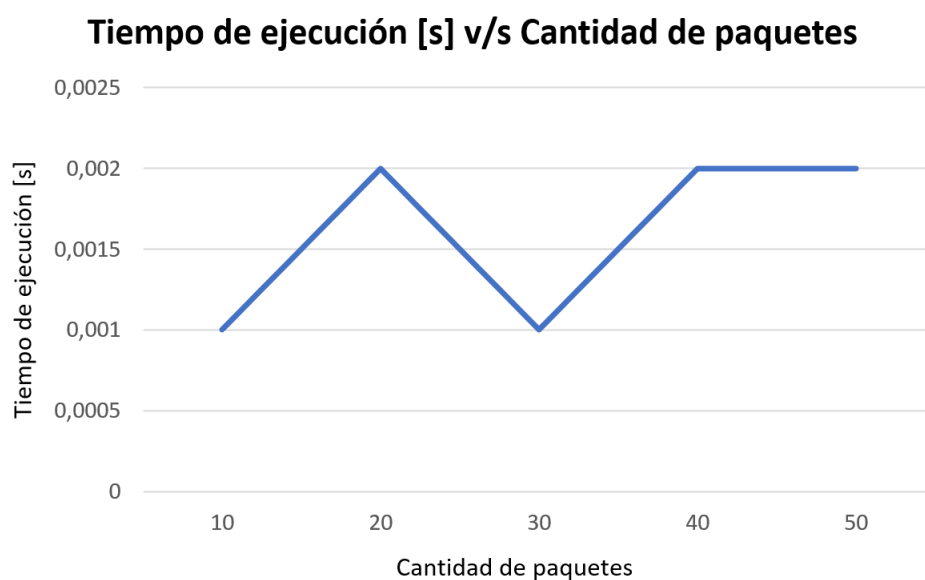


Figura 5: Gráfico de tiempo en el procesador Intel Core I5

Cantidad de paquetes	Tiempo de ejecución [s]
10	0,001
20	0,001
30	0,001
40	0,002
50	0,028

Figura 6: Tabla al testear en el procesador Intel Xeon

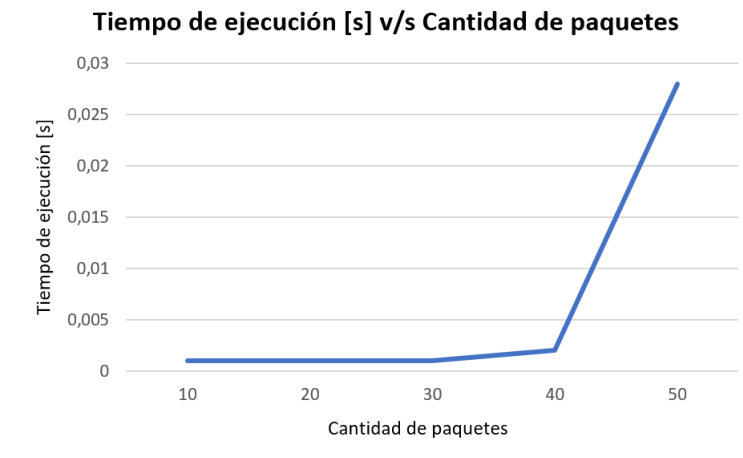


Figura 7: Gráfico de tiempo en el procesador Intel Xeon

```
salida.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
Paquete 1
Paquete 2
Paquete 4
Paquete 5
Paquete 6
Paquete 7
Paquete 9
Paquete 10
Beneficio: 544
Volumen: 71m3
```

Figura 8: salida.txt para el dataset de 10 paquetes

```
salida: Bloc de notas
Archivo Edición Formato Ver Ayuda
Paquete 2
Paquete 7
Paquete 9
Paquete 13
Paquete 15
Paquete 18
Paquete 20
Beneficio: 525
Volumen: 149m3
```

Figura 9: salida.txt para el dataset de 20 paquetes

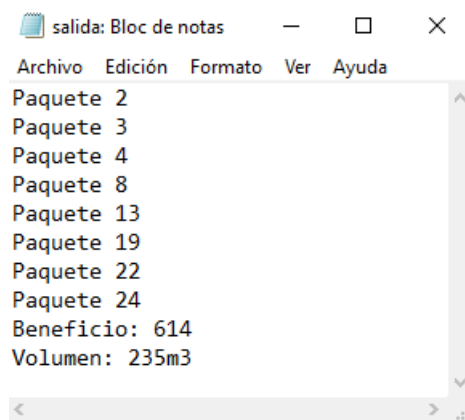


Figura 10: salida.txt para el dataset de 30 paquetes

### 3. Discusión

#### 3.1. Cálculo de complejidad

A continuación, se presenta el cálculo de complejidad del algoritmo creado.

Por cada línea en el archivo:

Se lee volumen del Paquete de la línea (segundo dígito)

Si volumen Paquete < Capacidad Container:

Calcular ponderación -> beneficio/volumen

Generar paquete según estructura

Guardar paquete en arreglo inicial

Cerrar archivo

Orden:  $O(n)$

Si largo arreglo inicial == 0:

Mostrar por pantalla “No hay solución válida”

Fin de Programa

Orden:  $O(1)$

Realizar un ordenamiento por Quicksort de la lista de paquetes de mayor a menor según su ponderación, usando la función nativa `qsort()`; disponible en la librería estándar de C `<stdio.h>`.

La función de comparación garantiza que nunca ocurra el peor caso.

Orden:  $O(n \log n)$

Se inicializan dos arreglos con la misma cantidad de espacio  
que el largo del arreglo de paquetes: mochila y mejorMochila

Estos arreglos se llenan inicialmente con Ceros

Se define una variable auxiliar  $i = 1$

Orden:  $O(n)$ , debido a que se recorren linealmente dos arreglos

Si  $i > \text{largoArregloInicial}$ :

Para cada elemento dentro de mochila:

Insertar elemento dentro de mejorMochila

retornar mejorMochila

Si  $\text{pesoActual} + \text{peso del item } i \text{ del arreglo inicial} \leq \text{capacidad Total}$ :

Se agrega el item al arreglo mochila

$\text{pesoActual} = \text{pesoActual} + \text{pesoItem}$

$\text{beneficioActual} = \text{beneficioActual} + \text{beneficioItem}$

Se llama recursivamente a esta función backtracking,  
aumentando en 1 el valor de  $i$  //  $\text{backtracking}(i+1)$

Se quita el item del arreglo mochila

$\text{pesoActual} = \text{pesoActual} - \text{pesoItem}$

$\text{beneficioActual} = \text{beneficioActual} - \text{beneficioItem}$

Si  $\text{posibleBeneficio al añadir próximo ítem} > \text{beneficioActual}$ :

Se quita el item del arreglo mochila

Se llama recursivamente a esta función backtracking,  
aumentando en 1 el valor de  $i$  //  $\text{backtracking}(i+1)$

Orden:  $O(n \cdot 2^n)$  Esto debido a que en el peor caso, la poda aplicada

no eliminara ninguna posible combinación a generar, lo que provocara que se generen todas las posibles combinaciones para los  $n$  paquetes del dataset

Y matemáticamente la suma de todas las posibles combinaciones desde cero

hasta  $n$  sin repetición está dada por la expresión:  $\sum_{i=0}^n \binom{n}{i} = 2^n$

y debido a que las llamadas recursivas se ejecutan como si estuvieran en

un ciclo anidado de complejidad  $O(n)$  que recorre todo el arreglo de paquetes iniciales

y desde ahí genera todas las posibles combinaciones y podas, la complejidad final es de  $O(n \cdot 2^n)$

Abrir archivo con nombre salida.txt

Se inicializan las variables beneficioTotal y volumenTotal ambas en cero

Para cada elemento de mejorMochila:

    Si elemento == 1:

        Encontrar Paquete con el id de la posición de elemento

        Leer numero de id del Paquete encontrado

        Escribir “Paquete” + id en archivo

        beneficioTotal = beneficioTotal + beneficio Paquete encontrado

        volumenTotal = volumenTotal + volumen Paquete encontrado

Escribir “Beneficio: ” + beneficioTotal en archivo

Escribir “Volumen: “ + volumenTotal en archivo Cerrar archivo

Fin de Programa

Orden:  $O(n^2)$

El orden de este algoritmo se debe a que se realiza una búsqueda en un arreglo de tamaño n dentro de un ciclo que se repite n veces.

### Complejidad total

$$O(n) + O(1) + O(n \log n) + O(n) + O(n \cdot 2^n) + O(n^2)$$

Por cota superior:  $O(n \cdot 2^n) \geq O(n^2) \geq n \log n \geq n \geq 1$ .

Por lo cual la complejidad del algoritmo es  $O(n \cdot 2^n)$ .

## 3.2. Resultados

Tomando en consideración los resultados conseguidos, es posible observar que se logró obtener una solución para todos los datasets sin mayor problema. Respecto al algoritmo implementado, es posible observar que siempre entrega una solución en un tiempo acotado, sin embargo, este algoritmo no asegura el óptimo del problema debido a la aplicación de poda en el backtracking, “Las funciones de acotación se consideran buenas si reducen considerablemente el número de nodos que hay que explorar. Sin embargo, las buenas funciones de acotación suelen consumir mucho tiempo en su evaluación, por lo que hay que buscar un equilibrio entre el tiempo de evaluación de las funciones de acotación y la reducción del número de nodos generados.” [2] Entonces se puede determinar que la principal ventaja de realizar un algoritmo mediante backtracking es el que entregará una solución en un tiempo acotado y su principal desventaja es que no se puede asegurar que la solución entregada sea la mejor o la óptima. Realizando una comparación con la técnica de fuerza bruta/búsqueda exhaustiva que fue implementada en la sesión número n°1 de laboratorio, se puede apreciar una gran similitud en los métodos, pero los tiempos de backtracking resultan muy acotados en comparación a fuerza bruta debido a la aplicación de las podas para disminuir la cantidad de combinaciones generadas, por ejemplo para el tercer dataset de 30 paquetes, en la implementación de fuerza bruta tardo 23 minutos en arrojar un resultado, en cambio en esta implementación mediante backtracking, solo tardo 2 milésimas de segundo. Comparando el método backtracking con goloso, se puede apreciar tiempos bastante similares, también una mejoría en la calidad de las soluciones, aún así ambos métodos no aseguran el óptimo. En caso de no realizar el método de poda o incluso tener un volumen que pueda contener todos los paquetes, sería posible observar que el algoritmo crece de manera exponencial. Por otra parte, el precio a pagar de backtracking por aplicar una poda se ve reflejado en la diferencia de las soluciones entregadas en comparación al algoritmo de fuerza bruta, mientras back-



tracking entrega una solución aproximada superior a goloso, fuerza bruta siempre entregó la mejor solución para cada caso.

### **3.3. Recomendaciones**

Tomando los resultados en consideración, podría recomendarse un algoritmo basado en backtracking para tanto grandes como pequeñas cantidades de datos, ya que se obtendrá una solución aproximada en una baja cantidad de tiempo. Sin embargo, es necesario tener en consideración que los resultados con este método pueden no ser el óptimo del problema debido al criterio aplicado en la poda, por lo que en caso de tener una cantidad pequeña de datos es recomendable ocupar el algoritmo de fuerza bruta para asegurar el mejor resultado.

## 4. Conclusiones

En el presente informe se efectuó el estudio del algoritmo mediante la técnica de backtracking a través del problema de la combinación y selección de paquetes. Se cumplió el objetivo de lograr comprender el método de backtracking, siendo capaces de destacar sus fortalezas y debilidades, además se cumplió el objetivo de idear una solución para el problema planteado utilizando un algoritmo basado en backtracking, el cual luego fue implementado bajo la creación de un programa en el lenguaje de programación C. Las pruebas efectuadas en el programa arrojaron resultados satisfactorios logrando encontrar soluciones aproximadas que pueden ser o no las óptimas, además fue posible encontrar solución para todos los datasets entregados para el testing. En conclusión, se recomienda el uso de esta técnica cuando se tiene claro los criterios de poda a aplicar, ya que se asegura una calidad óptima de solución, además de disminuir la cantidad de combinaciones a generar, lo que permite reducir los tiempos de ejecución, en caso de que se tengan pocos datos se recomienda la búsqueda de otro algoritmo que siempre pueda asegurar resultados óptimos.

Para finalizar, el trabajo en grupo para la realización de este laboratorio fue muy efectivo, logrando una buena comunicación y distribución del trabajo.

## 5. Apéndice

### 5.1. Pseudocódigo implementado

Pseudocódigo del programa de backtracking:

1. Se Solicita el nombre del archivo por teclado.
2. Se verifica si existe el dataset según el nombre entregado, en caso de no existir vuelve al paso 1.
3. Se almacena la capacidad máxima del container por el nombre del archivo.
4. Se almacena la cantidad de paquetes por el nombre del archivo.
5. Se abre el archivo dataset (archivo .csv).

---

Nota: La estructura que se utilizar y que se le har referencia a lo largo del pseudocódigo es la siguiente:

```
struct Paquete{  
    int id; //id del paquete  
    int beneficio; //beneficio del paquete  
    int volumen; //volumen que ocupa el paquete  
    float ponderacin; //resultado de dividir beneficio entre volumen  
};
```

---

6. Lectura, validación y creación de paquetes, creación del arreglo inicial que contiene todos los paquetes leídos y validados.

---

Se crea un arreglo de paquetes iniciales vaco

Por cada linea en el archivo:

Se lee volumen del Paquete de la linea (segundo dgito)

Si volumen Paquete < Capacidad Container:

Calcular ponderacin -> beneficio/volumen

Generar paquete segn estructura

Guardar paquete en arreglo inicial

Cerrar archivo

---

7. Se verifica en caso de que no se haya guardado ningún paquete válido.

---

Si largo arreglo inicial == 0:

Mostrar por pantalla No hay solución válida

Fin de Programa

---

8. Realizar un ordenamiento por Quicksort de la lista de paquetes de mayor a menor según su ponderación, usando la función nativa qsort(); disponible en la librería estándar de C stdio.h.

9. Se inicializan dos arreglos los cuales serán ocupados dentro del backtracking.

---

Se inicializan dos arreglos con la misma cantidad de espacio que el largo del arreglo de paquetes: mochila y mejorMochila

Se define una variable auxiliar i = 1

---

10. Se realiza el backtracking con poda para determinar la solución.

---

Si i > largoArregloInicial: //Condición de salida de la recursión

Para cada elemento dentro de mochila:

Insertar elemento dentro de mejorMochila

retornar mejorMochila

Si pesoActual + peso del item i del arreglo inicial <= capacidad Total:

Se agrega el item al arreglo mochila

pesoActual = pesoActual + pesoItem

beneficioActual = beneficioActual + beneficioItem

Se llama recursivamente a esta función backtracking,

aumentando en 1 el valor de i // backtracking(i+1)

Se quita el item del arreglo mochila

pesoActual = pesoActual - pesoItem

```
beneficioActual = beneficioActual - beneficioItem
```

Si posibleBeneficio al aadir prximo tem > beneficioActual:

Se quita el item del arreglo mochila

Se llama recursivamente a esta funcin backtracking,

aumentando en 1 el valor de i // backtracking(i+1)

---

Nota: posibleBeneficio se calcula de la siguiente manera:

$\text{posibleBeneficio} = \text{beneficioActual} + \text{capacidadRestante} * P_i$

( $P_i$  = Ponderación del item que se desea agregar)

11. Se escribe un archivo de salida .txt que posee la información de la solución encontrada, la cual fue determinada en el paso 9 según el índice de tope de la lista de paquetes calculado, el cual no excede el volumen máximo del container. Se escribirán los paquetes de la solución encontrada, incluyendo el beneficio y el volumen que ocupa.

---

Abrir archivo con nombre salida.txt

Se inicializan las variables beneficioTotal y volumenTotal ambas en cero

Para cada elemento de mejorMochila:

Leer numero de id del elemento actual

Escribir Paquete + id en archivo

beneficioTotal = beneficioTotal + beneficio elemento

volumenTotal = volumenTotal + volumen elemento

Escribir Beneficio : + beneficioTotal en archivo

Escribir Volumen : + volumenTotal en archivo

Cerrar archivo

Fin de Programa

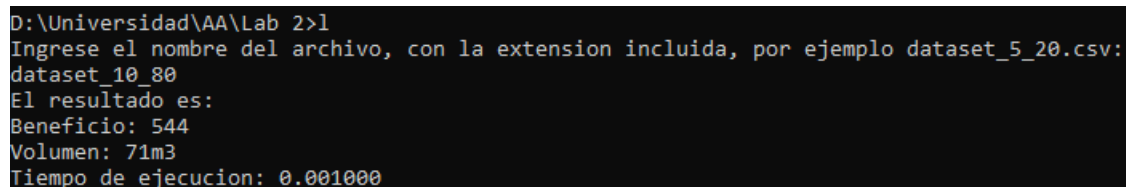
---

## 5.2. Instrucciones de uso

El programa implementado busca una solución para cualquier dataset entregado por el usuario en tiempo de ejecución, para dar un dataset de forma correcta al programa, este debe encontrarse en la misma carpeta del ejecutable, además, el nombre del archivo debe contener el siguiente formato:

- dataset\_**P**\_**V**.csv.
- P: Cantidad de Paquetes que contiene el dataset.
- V: Volumen máximo del container.

El programa finaliza cuando se indica por pantalla la mejor combinación de paquetes, dando a conocer la cantidad de tiempo en segundos que demora generar las combinaciones, el volumen de la solución y beneficio de la solución. Además, se genera un archivo de salida “salida.txt” en el directorio del programa, el cual contiene los paquetes que conforman la mejor combinación, beneficio y volumen totales.



```
D:\Universidad\AA\Lab 2>l
Ingrese el nombre del archivo, con la extension incluida, por ejemplo dataset_5_20.csv:
dataset_10_80
El resultado es:
Beneficio: 544
Volumen: 71m3
Tiempo de ejecucion: 0.001000
```

Figura 11: Ejemplo de ingreso de un dataset y termino de ejecución del programa.

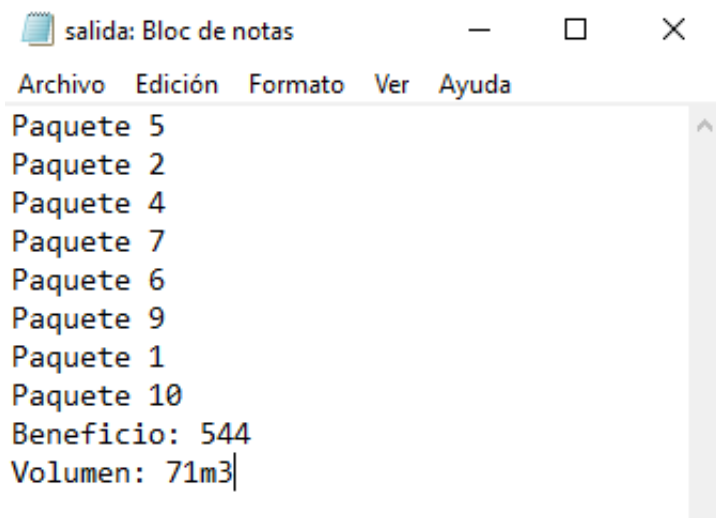


Figura 12: Ejemplo de archivo de salida al termino de la ejecución del programa.

## Referencias

- [1] J. B. Aranda. Backtracking. PUC. [Online]. Available: <https://jabaier.sitios.ing.uc.cl/iic2552/backtracking.pdf>
- [2] Eficiencia de eficiencia de backtracking. [Online]. Available: <http://elvex.ugr.es/decsai/algorithms/slides/5/%20backtracking.pdf>