

# Reporte de resultados

## Proyecto Investigación de Operaciones

Kevin Ledezma Jiménez  
Ingeniería en Computación  
Instituto Tecnológico de Costa Rica  
Heredia, Costa Rica  
kledezma204@gmail.com

**Abstract**—Este documento está enfocado en analizar los resultados obtenidos durante este proyecto, en el cual implementamos el algoritmo de la “mochila” en el lenguaje *python*, tanto de manera fuerza bruta así como implementando programación dinámica. También se implementó el análisis de hileras por medio del método *alineamiento* utilizando el algoritmo *Needleman-Wunsch*

### I. INTRODUCCIÓN

El proyecto consta de dos partes: la primera es el programa computacional que resuelve los problemas de contenedor (mochila) y el problema del alineamiento de secuencias. Para esto, nos enfocaremos en dos técnicas para el problema del contenedor: algoritmo de fuerza bruta y algoritmo utilizando programación dinámica. Además, para el problema del alineamiento, también se mostrará el algoritmo *Needleman-Wunsch* que utiliza programación dinámica para resolver y dar puntajes de comparación a secuencias de hileras.

### II. PROBLEMA DEL CONTENEDOR

El problema del contenedor es bastante “simple” pero no fácil. Se trata de simular tener un contenedor (o una mochila) y nos dan un listado de objetos, los cuales tienen un valor (beneficio) y un peso dentro del contenedor. Nuestro objetivo es conseguir la mayor cantidad de objetos maximizando nuestro valor final (ganancia). Para lograr esto, existen dos enfoques: Fuerza bruta y Algoritmo aprovechando recursos de programación dinámica. Para lograr una ejecución de este ejercicio, se necesitarán los siguientes datos: peso máximo que la mochila puede cargar, y la información de cada objeto, tal que: sepamos cuanto pesa cada objeto, cuanto beneficio nos otorga ese objeto además de saber si estamos bajo el escenario donde existen muchas copias de este objeto (es decir, tomar 3 celulares de idénticos).

#### A. Algoritmo Fuerza Bruta

Primero, podemos observar la simpleza que ofrece un algoritmo de fuerza bruta para resolver este ejercicio.

$$V[k, w] = \begin{cases} V[k-1, w] & \text{si } w_i > W \\ \max\{V[k-1, w], V[k-1, w-w_k] + b_k\} & \text{sino} \end{cases}$$

Imagen tomada del curso Investigación de Operaciones, ITCR 2020

En pocas palabras, tenemos dos verificaciones: llevar o no llevar el objeto. Si no es posible llevarlo, lo omitimos y

seguimos al siguiente. Sin embargo, si es posible llevarlo, haremos un *max* de ambos escenarios: tomando el objeto y no llevándolo para analizar más casos.

No obstante, esta solución tiene un complejo costo computacional, ya realiza dos llamadas recursivas por cada elemento que analiza en la mochila, esto provoca una complejidad computacional de  $O(n*k)[1]$ , donde  $k$  es la cantidad de elementos a analizar y su complejidad se vuelve muy complicada entre más objetos deseamos analizar.

#### B. Algoritmo Programación Dinámica

Cuando analizamos este mismo problema, haciendo uso de la programación dinámica *el famoso memoize*, notamos una gran mejoría tanto en rendimiento y hasta en implementación para comprender el funcionamiento de esta solución.

```
for i=1 to n
  for w=0 to W
    if  $w_i > w$ 
       $V[i, w] = V[i-1, w]$ 
    else
      if  $b_i + V[i-1, w-w_i] > V[i-1, w]$ 
         $V[i, w] = b_i + V[i-1, w-w_i]$ 
      else
         $V[i, w] = V[i-1, w]$ 
```

Imagen tomada del curso Investigación de Operaciones, ITCR 2020

Aquí podemos notar una gran diferencia en implementación, empezando por que no hacemos llamadas recursivas, sino que optamos por hacer ciclos iterativos *for* sobre una matriz que nos ayudará a recordar información importante como los objetos que hemos decidido y los que no. Además, nos otorga una mejor duración de ejecución, como podemos observar en las siguientes imágenes, tomadas utilizando el código creado para este proyecto como base de la ejecución.

```
Fuerza Bruta
Resultado: 260
¡Programa Finalizado!
Segundos transcurridos 0.0014951229095458984
```

Imagen tomada como resultado de la implementación de fuerza bruta

```
Ejecutando Programacion Dinamica para Mochila
Resultado: 260
¡Programa Finalizado!
Segundos transcurridos 0.0009868144989013672
```

Imagen tomada como resultado de la implementación de programación dinámica

Tal vez la diferencia en la duración es muy pequeña *también debido a que nuestro computador es bastante robusto en términos de componentes*, pero prestando atención, podemos ver como el tiempo de ejecución del algoritmo de programación dinámica es menor en aproximadamente la mitad del tiempo incluso siendo fracciones de segundo. Esto connota una gran eficiencia computacional, ya que conforme el algoritmo de fuerza bruta consume recursos, podremos ahorrar recursos y tiempo de manera exponencial con el algoritmo de programación dinámica.

### C. Comparativa de Algoritmos

Es muy curioso, que al realizar pruebas de ejecución en estas implementaciones, al ser un ejercicio de pocas entradas (es decir, pocos elementos de donde decidir) parece ser que el algoritmo de programación dinámica no es nada eficaz y hasta menos eficiente que fuerza bruta.

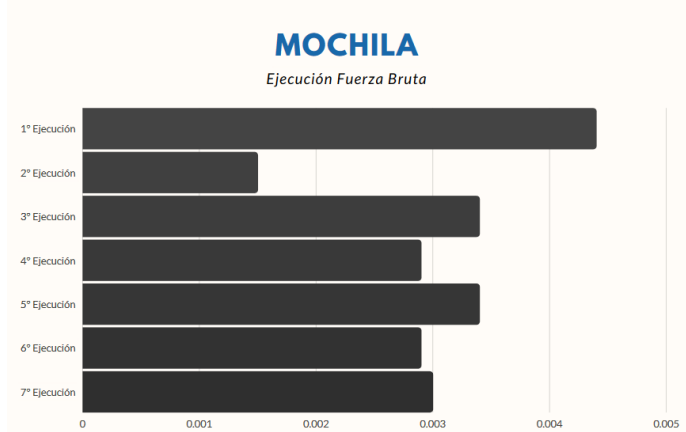


Imagen tomada como resultado de siete ejecuciones usando fuerza bruta en un ejercicio de pocos elementos

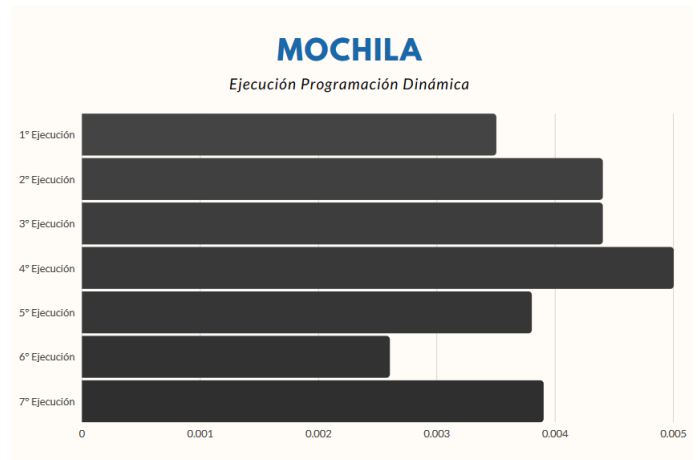


Imagen tomada como resultado de siete ejecuciones usando programación dinámica en un ejercicio de pocos elementos. Nótese que las escalas de la barra horizontal inferior son las mismas y vemos una mayor duración en general por parte del algoritmo de programación dinámica, sin embargo, analicemos que sucede cuando ejecutamos ejercicios con grandes cantidades de elementos.

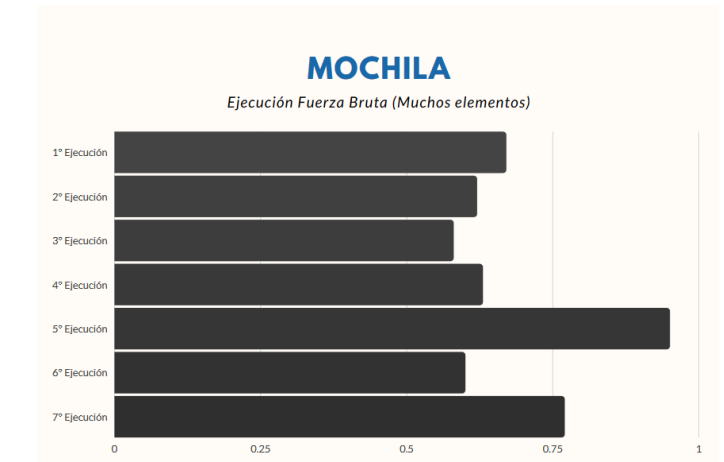


Imagen tomada como resultado de siete ejecuciones usando fuerza bruta en un ejercicio de 40 elementos

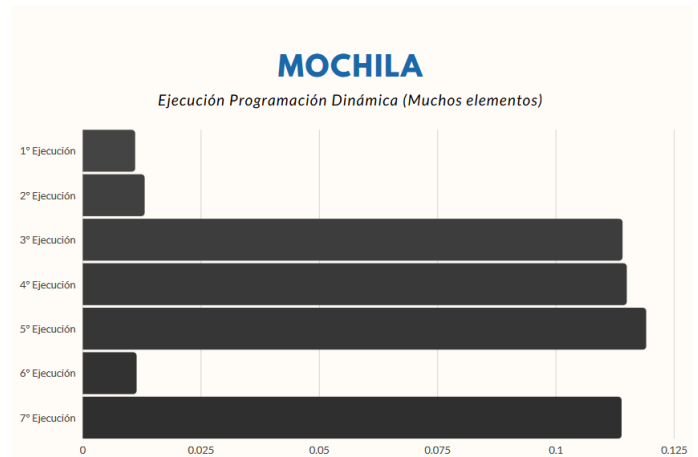


Imagen tomada como resultado de siete ejecuciones usando programación dinámica en un ejercicio de 40 elementos

Aunque las barras de duración parezcan similares, pongamos especial atención a la *barra horizontal inferior* donde vemos que los tiempos de ejecución son verdaderamente inferiores utilizando programación dinámica y podemos afirmar que ese es el verdadero beneficio que otorga este algoritmo.

### III. PROBLEMA DEL ALINEAMIENTO

#### REFERENCES

- [1] Velasco, "NP-Completeness", *Elisa.dyndns-web.com*, 2014. [Online]. Available: <https://elisa.dyndns-web.com/teaching/opt/comb/knapsackOC.pdf>. [Accessed: 14- Dec-2020].