**Universidad Peruana de Ciencias Aplicadas**

**Facultad de Ingeniería**



**Nombre del Proyecto:**

**“Empaquetamiento y Corte en 3D”**

**Curso: Complejidad Algorítmica**

**Profesora: Canaval Sanchez, Luis Martin**

**Alumnos:**

|  |  |
| --- | --- |
| Guillermo Gavilano Auris | 20171B294 |
| Diego Alonzo Hilario Callupe | 20171B079 |
| Ricardo Daniel Iglesias Espinoza | 20161C808 |

**Introducción**

El presente trabajo abordará el problema empaquetado tridimensional, que comúnmente presentan las empresas dedicadas a la logística, algunas tareas comunes donde se presentan estos problemas son al llenar contenedores, camiones, barco y/o aviones de carga, donde se desea hacer la menor cantidad de viajes posibles por ende deben asegurarse de enviar la mayor cantidad de elementos en cada transporte que realizan, dejando la menor cantidad de espacio. En la actualidad, existen diferentes heurísticas de las cuales se pueden hacer uso para implementar algoritmos que solucionen el problema.

En este proyecto se buscará y/o implementará tres algoritmos para poder darle solución a un problema real, mostrando su seudocódigo, algoritmo, complejidad, tablas comparativas del consumo de recursos de memoria y tiempo, y como conclusión elegir al mejor algoritmo basándonos en la complejidad, consumo de recursos y tiempo.

Índice

[**1.** **Objetivo del Estudiante (Student Outcome)** 4](#_Toc22847968)

[**2.** **Estado del Arte** 5](#_Toc22847969)

[**2.1 Algoritmo 1:** 5](#_Toc22847970)

[**Heurística Left-Bottom (BL)** 5](#_Toc22847971)

[**Heurística Finite-Next-Fit (FNF)** 5](#_Toc22847972)

[**2.2 Algoritmo 2:** 6](#_Toc22847973)

[**Heurística Next-Fit Decreasing Height (NFDH)** 6](#_Toc22847974)

[**2.3 Algoritmo 3:** 7](#_Toc22847975)

[**3.** **Aporte: Demuestra ética profesional** 8](#_Toc22847976)

[**4.** **Diseño de Aplicativo para Pruebas** 9](#_Toc22847977)

[**4.1 Pseudocódigo 1:** 9](#_Toc22847978)

[**4.2 Pseudocódigo 2: Heurística Next-Fit Decreasing Height (NFDH)** 9](#_Toc22847979)

[**4.3 Pseudocódigo 3:** 9](#_Toc22847980)

[**5.** **Validación de Resultados y Discusión** 10](#_Toc22847981)

[**6.** **Conclusiones y Trabajos Futuros** 11](#_Toc22847982)

[**7.** **Conclusiones** 12](#_Toc22847983)

[**8.** **Anexos** 13](#_Toc22847984)

[**9.** **Bibliografía** 14](#_Toc22847985)

# **Objetivo del Estudiante (Student Outcome)**

Nuestro objetivo como estudiantes del curso de Complejidad algorítmica y de la carrera de Ingeniería de Software es aplicar los conocimientos aprendidos hasta el día de hoy para solucionar y/o mejorar problemas de contexto real, permitiendo nuestro desarrollo en el reconocimiento de responsabilidades de manera profesional en situaciones de ingeniería. Además, buscamos desarrollar la ética y profesionalismo al citar y nombran los autores de los algoritmos y heurísticas de las cuales haremos uso para poder implementar algoritmos en 3D.

# **Estado del Arte**

## **2.1 Algoritmo 1:**

## **Heurística Left-Bottom (BL)**

Este algoritmo de empaquetamiento también sigue la heurística de Strip Packing. Consiste en ordenar todas las piezas a empaquetar de manera decreciente de acuerdo a su área, luego se empaqueta de forma que la siguiente pieza vaya lo más cerca posible de la parte inferior, y luego lo más cerca posible de la izquierda sin superponerse con ninguna otra pieza embalada. Hay que tener en cuenta que este algoritmo no rota las piezas a empaquetar simplemente las acomoda.

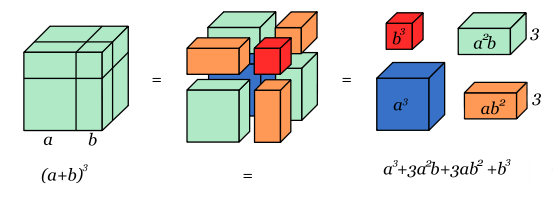
## **Heurística Finite-Next-Fit (FNF)**

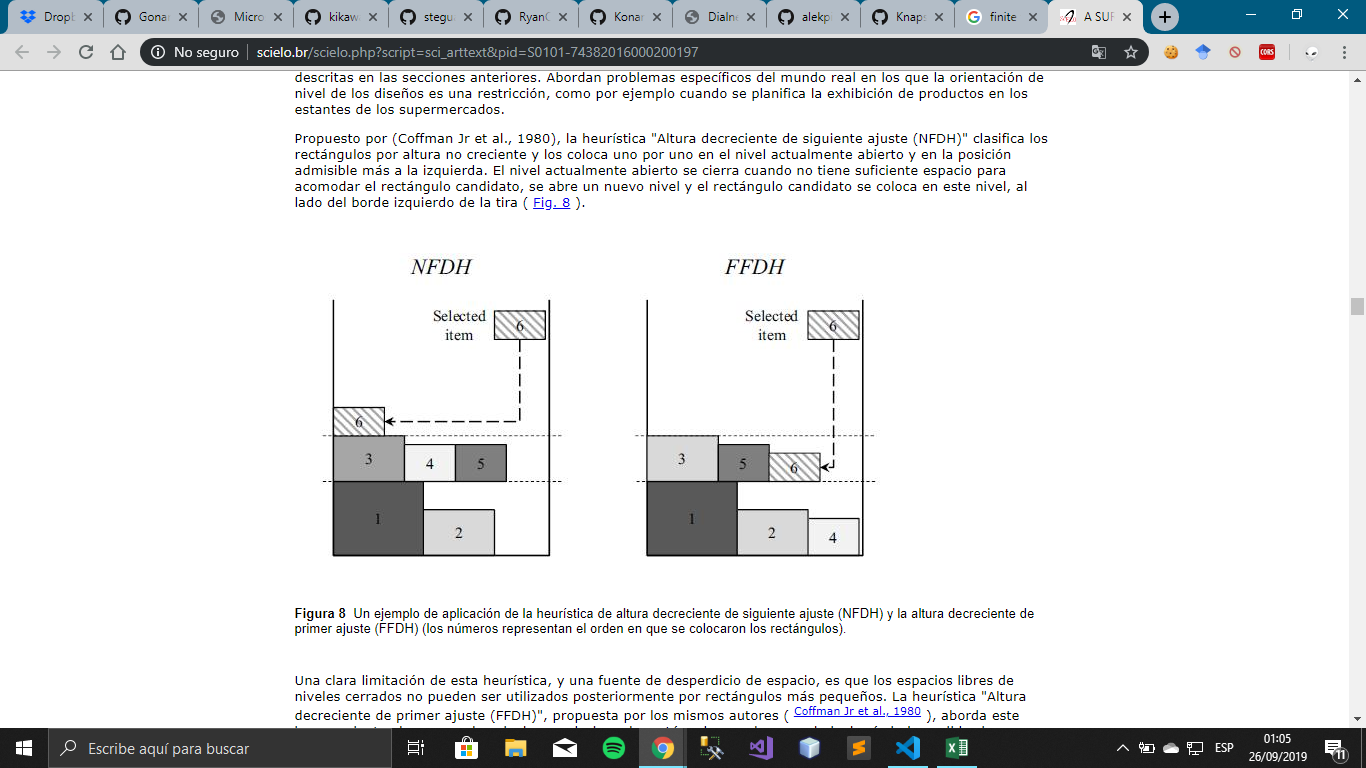
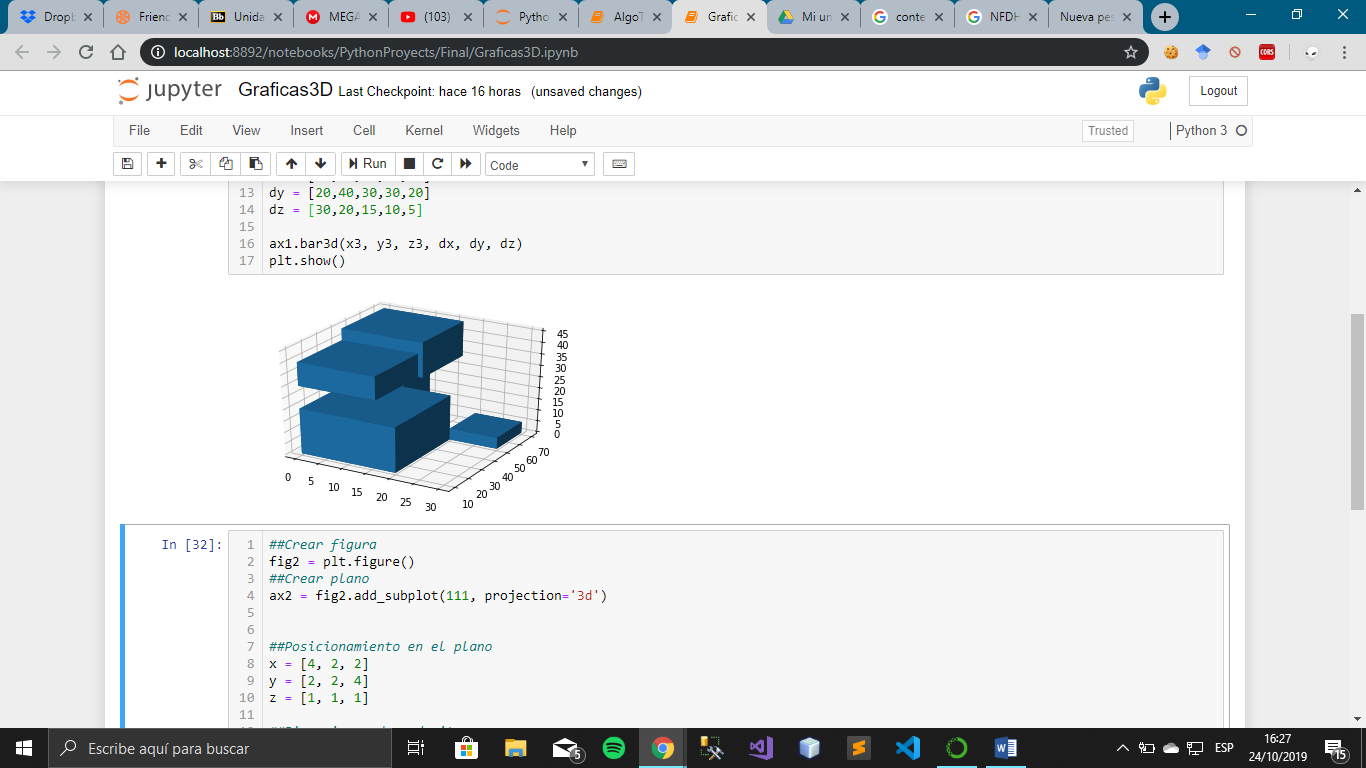
Inicialmente se tienen todos los contenedores vacíos. Se empieza con el primer el primer ítem (i=1). Si el contenedor j tiene capacidad suficiente para albergar a la pieza i, ésta es asignada al contenedor j y se consideraría en ítem i+1. Si, en cambio, no tuviera la capacidad se pasaría a considerar el contenedor j+1. Este proceso continuaría hasta asignar los n ítems. Esta heurística nunca reconsidera los contenedores que han sido rechazados, lo que deja espacio para la mejora.

Las dos heurísticas mencionadas en párrafos anteriores se utilizarán de la siguiente manera para el empaquetado tridimensional. En primer lugar, los ítems serán ordenados de manera decreciente por el volumen de cada uno, como la heurística BL lo hacía en 2D, pero con áreas. En segunda lugar, la heurística de Finite Next Fit será aplicada de igual forma, ya que está no varía.

## **2.2 Algoritmo 2:**

## **Heurística Next-Fit Decreasing Height (NFDH)**

Esta forma parte de la heurística basada en niveles de Strip Packing Problem, que tiene como idea central la colocación de rectángulos en niveles , es decir, cortes paralelos de guillotina en todo el ancho de la tira. Esta heurística fue propuesta por Coffman Jr en 1980. El algoritmo clasifica los ítems (rectángulos) por altura de manera decreciente, donde el ítem con mayor altura define la altura de cada nivel, y los coloca uno por uno en el nivel actual y en la posición más a la izquierda. Se cambiará de nivel cuando en el nivel actual no se tiene suficiente espacio para acomodar al siguiente rectángulo, donde este pasará al nivel superior colocado en la posición más a la izquierda. Para este caso, se realizará la implementación de esta heurística que será dirigida a problemas reales de forma tridimensional (3D), donde tanto el contenedor como los items serán manipulados con los ejes X, Y y Z. Esta heurística permitirá ordenar por altura los ítems y se colocará de manera descendente apegada, en primera instancia, hacia el eje Y y hacia la derecha, apilandolos hasta que ya no quepa otro ítem. Es ahí donde el siguiente ítem pasará a colocarse en la posición X que tendrá el valor del ítem con mayor largo que ha sido colocado hasta el momento y se repetirá el proceso hasta que ya no pueda entrar otro ítem en el contenedor.



## **2.3 Algoritmo 3:**

# **Aporte: Demuestra ética profesional**

# **Diseño de Aplicativo para Pruebas**

## **Pseudocódigo 1:**

## **4.2 Pseudocódigo 2: Heurística Next-Fit Decreasing Height (NFDH)**

**##AlgoritmoDiego**

def nfdhSPP(arrItem, largoBin, anchBin, altoBin)

n = len(arrItem)

quicksort(arrItem, 0, n-1, 2)

bins = 0

y = anchBin

z = 0

x = 0

larC = arrItem[0][0] ##mayor largo

arrItem[0] = (arrItem[0][0], arrItem[0][1], arrItem[0][2], x, y - arrItem[0][1], z, bins)

newEst = arrItem[0][2] ##Z

for i in range(1, n)

varY = arrItem[i-1][4] - arrItem[i][1]

if varY >= 0 ## Si el ancho del item es mayor o igual al 0, agregar

arrItem[i] = (arrItem[i][0], arrItem[i][1], arrItem[i][2], x, arrItem[i-1][4] - arrItem[i][1], z, bins)

if x + arrItem[i][0] > larC

larC = arrItem[i][0]

else

z = newEst ## Actualiza posicion z para un nuevo estante

varZ = z + arrItem[i][2]

varX = larC + arrItem[i][0]

if varZ <= altoBin ## Si el alto del item es menor o igual a altoBin, agregar

newEst = newEst + arrItem[i][2]

arrItem[i] = (arrItem[i][0], arrItem[i][1], arrItem[i][2], x, anchBin - arrItem[i][1], z, bins)

if x + arrItem[i][0] > larC

larC = arrItem[i][0]

elif varZ > altoBin and varX <= largoBin

newEst = arrItem[i][5]

y = anchBin

z = 0

x = larC

larC = x + arrItem[i][0]

arrItem[i] = (arrItem[i][0], arrItem[i][1], arrItem[i][2], x, y - arrItem[i][1], z, bins)

else

newEst = arrItem[i][5]

y = anchBin

x = 0

z = 0

bins = bins + 1

arrItem[i] = (arrItem[i][0], arrItem[i][1], arrItem[i][2], x, y - arrItem[i][1], z, bins)

return arrItem, bins

## **4.3 Pseudocódigo 3:**

# **Validación de Resultados y Discusión**

# **Conclusiones y Trabajos Futuros**

# **Conclusiones**

# **Anexos**

# **Bibliografía**

[Andrea Lodi](https://www.sciencedirect.com/science/article/pii/S0166218X0100347X" \l "!), [Silvano Martello](https://www.sciencedirect.com/science/article/pii/S0166218X0100347X#!), [Daniele Vigo](https://www.sciencedirect.com/science/article/pii/S0166218X0100347X#!) (SF). Recent advances on two-dimensional bin packing problems. Recuperado de <https://www.sciencedirect.com/science/article/pii/S0166218X0100347X#SEC2>