Proyecto Final BOXOP

Autor: Diego Rinaldo Cazon Condori

Materia: Inteligencia Artificial 1

Carrera: Licenciatura en Ciencias de la Computacion - UNCUYO

Año: 2022

Resumen:

Este informe trata el problema de ordenamiento de cajas rectangulares, dentro de un contendor, tratando de minimizar el área desperdiciada. La dificultad aumenta al aumentar la cantidad de cajas y con esto, los posibles formas de ordenar las cajas y poder mantener un buen tiempo de proceso, por lo que para estos casos se hace uso de metaheurísticas para reducir el tiempo de proceso y encontrar buenas soluciones.

Estos problemas los podemos encontrar dentro de las industrias de perfiles metálicos, corte de maderas, papel, plástico o vidrio en donde los componentes rectangulares tienen que ser cortados desde grandes hojas de material y también en el transporte de pallets dentro de contendores, donde no es posible colocar un pallet arriba de otro.

Este informe propone una solución, haciendo uso de 2 estructuras de datos (Árbol Guillotina , Árbol de Ordenamiento) , un algoritmo de ordenamiento simple y un algoritmo Genético, esto para poder encontrar un orden para las cajas, que minimiza el área desperdiciada y poder obtener un buen tiempo de procesamiento.

Índice

1. Introducción

En este capitulo introduciremos el problema de ordenamiento de cajas en 2D, comenzaremos explicando a que tipo de problemas pertenece, en que areas los encontramos, luego explicaremos las dificultades que estos tienen.

Estos problema de ordenamiento, entran dentro de la familia de los problemas de corte y empaquetamiento bidimensional en una sola placa (2D-CSP), donde el objetivo es el de reducir el espacio desperdiciado, para una cantidad finitas de cajas en una placa de mayor tamaño. Estos son encontrados en el área industrias de perfiles metálicos, corte de maderas, papel, plástico o vidrio en donde los componentes rectangulares tienen que ser cortados desde grandes hojas de material , también en el transporte de pallets dentro de contendores, donde no es posible colocar un pallet arriba de otro.Uno de los objetivos, para estos problemas , es

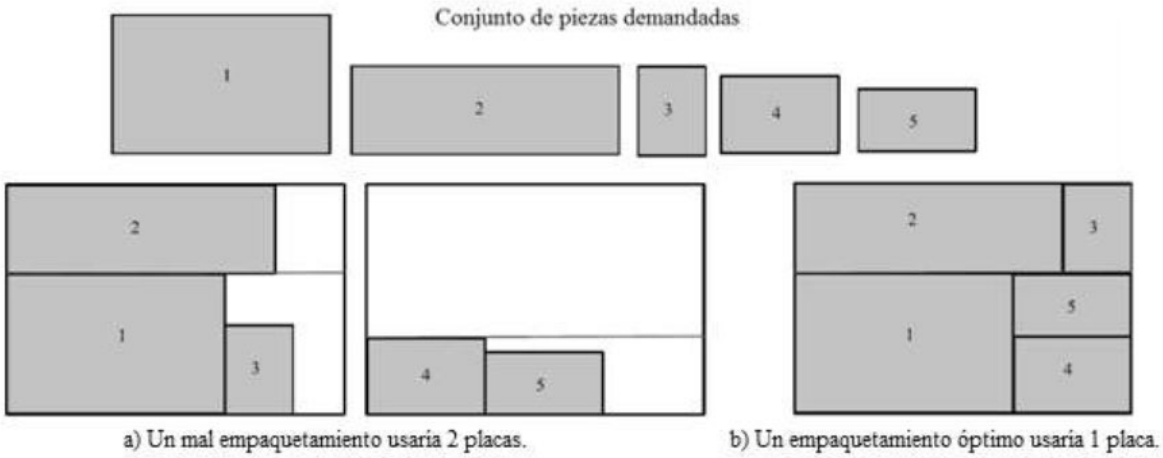


Figura 1. Empaquetamiento inadecuado y adecuado.

el de disminuir el espacio de área desperdiciado y además mantener un buen tiempo de ejecución. El tema ocurre cuando la cantidad de cajas aumenta, haciendo que la cantidad de formas de ordenar las cajas también aumente.

2. Definicion de estructuras de datos , algoritmo de ordenamiento y algoritmo genético

En este capitulo se presentaran las estructuras y algoritmos que usaremos para dar una solución al problema de ordenamiento, luego se presentara la implementación para dar como resultado el algoritmo de ordenamiento para cajas en contendores.

2.1 – Algoritmo y Árbol de ordenamiento

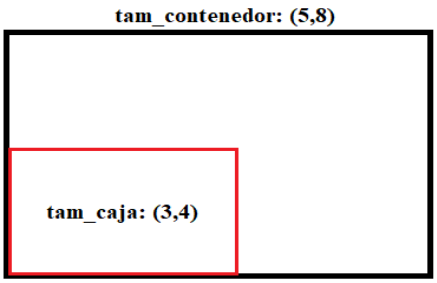
En esta sección presentaremos la estructura Árbol de Ordenamiento y Algoritmo de Ordenamiento, para el ordenamiento de cajas dentro de un contenedor, la estructura Árbol nos servirá para obtener los espacios sin uso, los espacios usados y el orden en el que están colocadas las cajas , el algoritmo de ordenamiento nos dará una guía para elegir que cajas ingresar.

2.1.1 Árbol de Ordenamiento

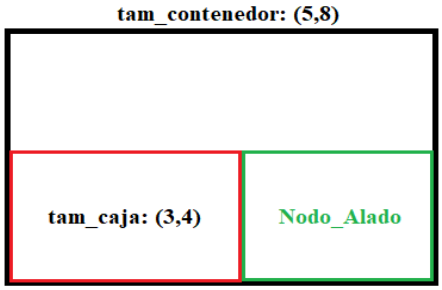
El árbol de ordenamiento es un árbol binario el cual tiene como objetivo dar información de cómo se encuentran ordenadas las cajas, espacios aun no recorridos y espacios desperdiciados

La Estructura cuenta con nodos, los cuales tienen los siguientes atributos:

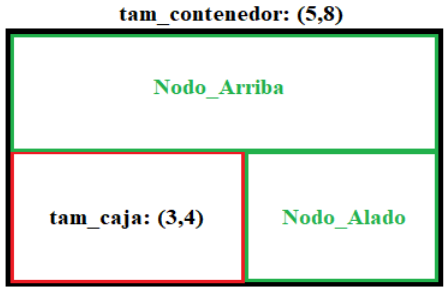
- Tam_Contenedor: Contiene el tamaño del contenedor donde ingresaremos las cajas



- Tam_Caja_Ingresada: Contiene el tamaño de la caja que se ingresó en el contenedor, siempre se ingresa en la posición más a la izquierda y abajo del contenendor



- Nodo_Alado: Contiene una referencia a el nodo de alado que contendrá como espacio de contenedor, el espacio de igual altura que la caja ingresada en el nodo actual, pero de ancho igual a (ancho_tam_contendor – ancho_tam_caja_ingresada).



- Nodo_Arriba: Contiene una referencia a el nodo de arriba que contendrá como espacio de contenedor, el espacio de igual ancho que el contenedor del nodo actual, pero de altura igual a (alto_tam_contendor – alto_tam_caja_ingresada).

A medida que ingresemos cajas, a los contenedores, la estructura ira sumando nodo_Alado y nodo_Arriba, la manera de recorrer el árbol definira el orden de las cajas, para este informe se uso el recorrido primero nodo alado, luego nodo arriba. La estructura Arbol de Ordenamiento resultante nos dará información sobre el ordenamiento de las cajas, y los nodos hojas nos indicaran los espacios desperdiciados.

Implementación Árbol de Ordenamiento y Algoritmo de Ordenamiento.

Ahora mostraremos el algoritmo de ordenamiento que usaremos, luego mostraremos el uso junto al árbol de ordenamiento para poder ordenar varias cajas dentro de un contenedor.

```
Funcion Algoritmo_de_Ordenamiento( Contenedor , Cajas_a_ordenar ):
        Caja_encontrada = Vacio
        Cajas_a_ordenar = Ordenar_Mayor_a_Menor( Cajas_a_ordenar )
        Por cada Caja de Cajas_a_ordenar:
                Caja_Invertida = Invertir_Fila_por_Columna( Caja )
                Si ( Caja ingresa en Contenedor ) o (Caja_Invertida ingresa en Contenedor ) :
                        #Si la caja tiene un dimensiones menores al contenedor  y es posible ingresarla
                        Caja_encontrada = Caja
                        Salir_de_Bucle #Salimos del bucle
```

Con este simple código obtenemos una caja, de la lista de cajas a ordenar, la cual es posible ingresarla en el contenedor, esta caja puede estar girada 90, probamos los 2 casos.

Usaremos un algoritmo recursivo que hará uso de la estructura árbol de ordenamiento y el algoritmo de ordenamiento, para poder ordenar las cajas en el contenedor.

1 – Generamos un nodo que tendrá el “tamaño del contenedor”

2 – Llamamos al algoritmo “ Algoritmo_de_Ordenamiento() ” y obtenemos la caja a ingresar

3 – En Caso de No obtener caja, entonces retornamos None

4 – En Caso de Obtener, borramos la caja obtenida de la lista de cajas y la ingresamos al nodo

5 – Generamos 2 nodos nuevos(Alado y Arriba) y llenamos sus atributos

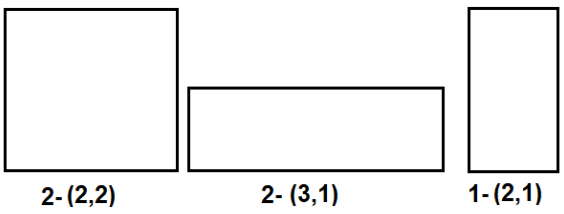
6 – Vamos por el nodo de Alado y ejecutamos el paso 2) con las cajas restantes

7 - Vamos por el nodo de Arriba y ejecutamos el paso 2) con las cajas restantes

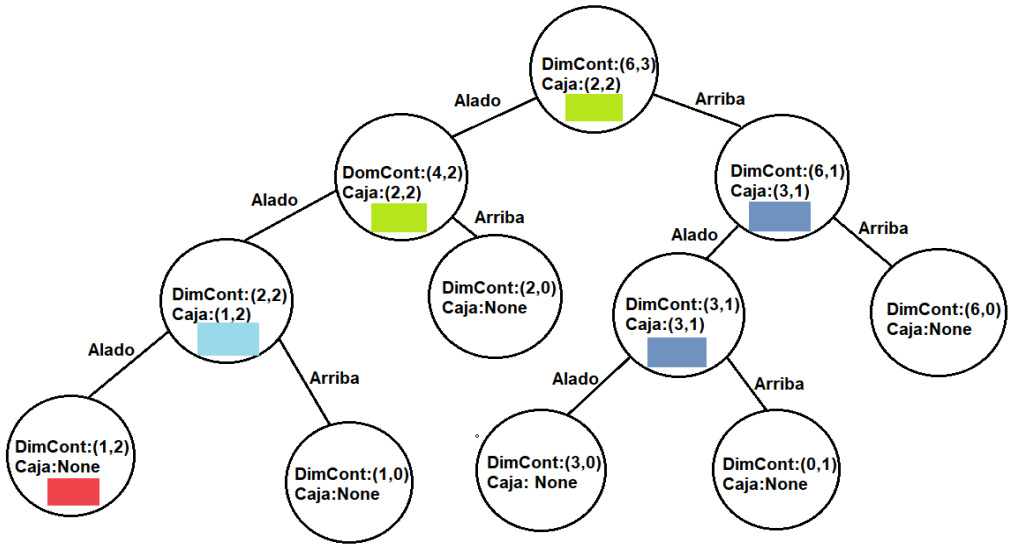
Con este simple algoritmo podremos generar un Arbol de Ordenamiento, con las cajas ordenadas en el contenedor según el algoritmo de ordenamiento, propuesto para este informe.

Para este ejemplo se tendrá que pensar que Dim_Contenedor = Tam_Contenedor y Caja = Tam_Caja.

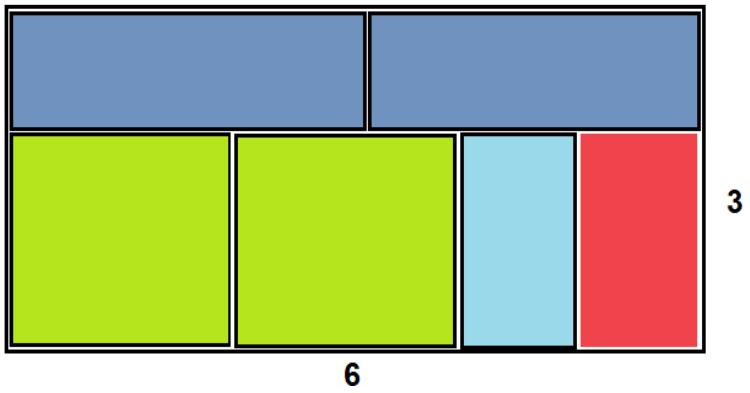
Cajas a Ordenar:



- Aplicando el Algoritmo de Odenamiento, en la estructura Arbol de Ordenamineto , obtendremos la siguiente estructura, donde podremos notar que los nodos hojas no contienen cajas ingresadas, pudiendo obtener de esta forma el espacio desperdiciado del contendor.



- .El resultado de ordenamiento obtenido es el siguiente, donde los espacios de color rojo representan los espacios sin uso.



2.2 – Arbol de Corte Guillotina

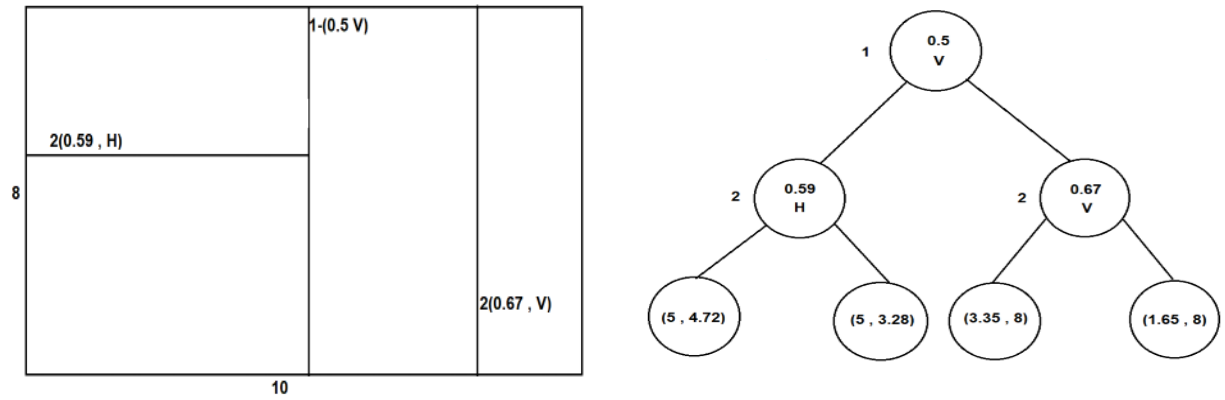
El árbol de Corte Guillotina es un árbol binario completo, esta estructura nos permitirá dividir el contenedor principal en varios subcontenedores, que luego usaremos para ordenar las cajas en estos, logrando dividir el problema en problemas mas pequeños y también obteniendo subcontenedores para un conjunto de grupos de cajas, si se definene buenos subcontenedores para grupos de cajas, estos dejaran menos espacios desperdiciados.

Esta estructura realiza corte de tipo Vertical o Horizontal, corte paralelo a un lado del contenedor, y lo realiza en un punto de la caja, generando 2 nuevos subcontenedores. La altura del árbol define la cantidad de cortes que se realizaran sobre el contenedor, por ejemplo para una altura 2, se generaran 3 cortes guillotina sobre el contenedor, y para una altura 3 , 7 cortes, la cantidad de cortes hace que tengamos subcontenedores de menor tamaño.

El Arbol Corte Guillotina contiene en cada nodo los siguientes atributos:

- Tipo_de_Corte: Define el corte sobre el contenedor, que puede ser Vertical o Horizontal, este corte es paralelo a un lado del contendor.
- Porcentaje_de_Corte: Define el porcentaje de corte que va desde 0 a 1, sin incluir 0 y 1.
- Nodo_ Derecho: Referencia al nodo de contendor derecho en caso de que el corte fuera vertical o contenedor de abajo en caso de que el corte fuera horizontal.
- Nodo_Izquierdo: Referencia al nodo de contendor izquierdo en caso de que el corte fuera vertical o contenedor de arriba en caso de que el corte fuera horizontal.

En este ejemplo podremos ver el contenedor de dimensiones (10,8) y los subcontenedores generados por los cortes realizados con el árbol de corte guillotina de altura 2, podremos ver en la estructura que los nodos hojas contienen los tamaños de los subcontenedores generados por los cortes realizado al contenedor inicial.



Ahora que ya contamos con varios subcontendores, generados por el árbol de corte guillotina, podremos hacer uso del algoritmo y estructura de ordenamiento (capitulo 2.1) para cada uno de los subcontenedores, el reto será encontrar la estructura árbol guillotina que genere menor espacio desperdiciado y esto lo lograremos cambiando los tipos y porcentajes de cortes.

2.3 – Algoritmo Genético

Llamados así porque se inspiran en la evolución biológica. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias, así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.

Un algoritmo genético puede presentar diversas variaciones, dependiendo de cómo se decide el reemplazo de los individuos para formar la nueva población. En general, el pseudocódigo consiste de los siguientes pasos:

- **Inicialización**: Se genera aleatoriamente la población inicial, que está constituida por un conjunto de cromosomas los cuales representan las posibles soluciones del problema.
- **Evaluación**: A cada uno de los cromosomas de esta población se aplicará la función de desempeño para saber cómo de "buena" es la solución que se está codificando.
- **Condición de término**: El AG se deberá detener cuando se alcance la solución óptima, pero esta generalmente se desconoce, por lo que se deben utilizar otros criterios de detención. Normalmente se usan dos criterios: correr el AG un número máximo de iteraciones (generaciones) o detenerlo cuando no haya cambios en la población. Mientras no se cumpla la condición de término se hace lo siguiente:
  - **Selección**: Después de saber el desempeño de cada cromosoma se procede a elegir los cromosomas que serán cruzados en la siguiente generación. Los cromosomas con mejor aptitud tienen mayor probabilidad de ser seleccionados.
  - **Cruzamiento**: La recombinación es el principal operador genético, representa la reproducción sexual, opera sobre dos cromosomas a la vez para generar dos descendientes donde se combinan las características de ambos cromosomas padres.
  - **Mutación**: Modifica al azar parte del cromosoma de los individuos, y permite alcanzar zonas del espacio de búsqueda que no estaban cubiertas por los individuos de la población actual.
  - **Reemplazo**: Una vez aplicados los operadores genéticos, se selecciona los mejores individuos para conformar la población de la generación siguiente.

3 – Implementación de la solución

Ahora haremos uso de las estructuras y algoritmos definidos, en los anteriores capítulos, para encontrar un orden para las cajas, que minimice el espacio desperdiciado, en un buen tiempo de ejecución.

1 - Se define un contenedor y una lista de cajas a ordenar

2 – Definiremos un algoritmo genético, con ‘i’ individuos, los cuales serán Arboles de Corte Guillotina (2.2), y ‘c’ ciclos que será nuestra condición de parada para nuestro algoritmo genetico.

3 – Definimos una unica altura para todos los árboles guillotina, recordar que la altura define la cantidad de cortes en el contenedor principal y por lo tanto la cantidad de los subcontenedores y sus dimenciones, luego definimos por cada individuo, su árbol de corte guillotina con sus tipos y porcentajes de cortes totalmente aleatorio, según lo definido en el capítulo 2.2.

4 – Para cada individuo de la población, ingresamos las cajas en los subcontenedores ,generados por el árbol corte guillotina, usando la estructura y algoritmo de ordenamiento y luego calculamos el área desperdiciada.

5 – Cruzamiento:

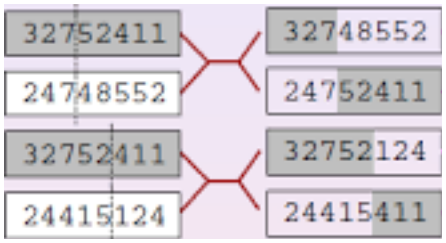
5.1 - Mientras tengamos individuos en la población que aún no se a cruzado:

5.1.1 - Obtendremos 2 individuos aleatorios de la población, serán los padres

5.1.2 – Generamos un numero aleatorio entre 1 y la cantidad de nodos de nuestro árbol corte guillotina, será nuestro número de corte.

5.1.3 – Generaremos 2 individuos nuevos, y sus arboles de corte guillotina.

5.1.3 –Los valores de cada nodo de el individuo nuevo será igual a uno de los padres hasta el numero de corte y el resto de los nodos será igual a los nodos del otro padre, en orden inverso para el otro nuevo individuo



6 – Para cada nuevo individuo de la población, ingresamos las cajas en los subcontenedores generados por el árbol corte guillotina, usando la estructura y algoritmo de ordenamiento y luego calculamos el área desperdiciada.

7 – Ahora tenemos el doble de la población inicial, entonces ordenaremos la población de menor a mayor, según el área desperdiciada de cada individuo, y eliminaremos a la mitad de la población con peor desempeño, que serian los de mayor área desperdiciada.

8 – Ejecutamos desde la instrucción 5) una cantidad de “c” veces.

9 – Luego de ejecutar el proceso “c” veces, de la población obtenida retornamos el individuo de menor área desperdiciada, que será nuestra mejor solucion.

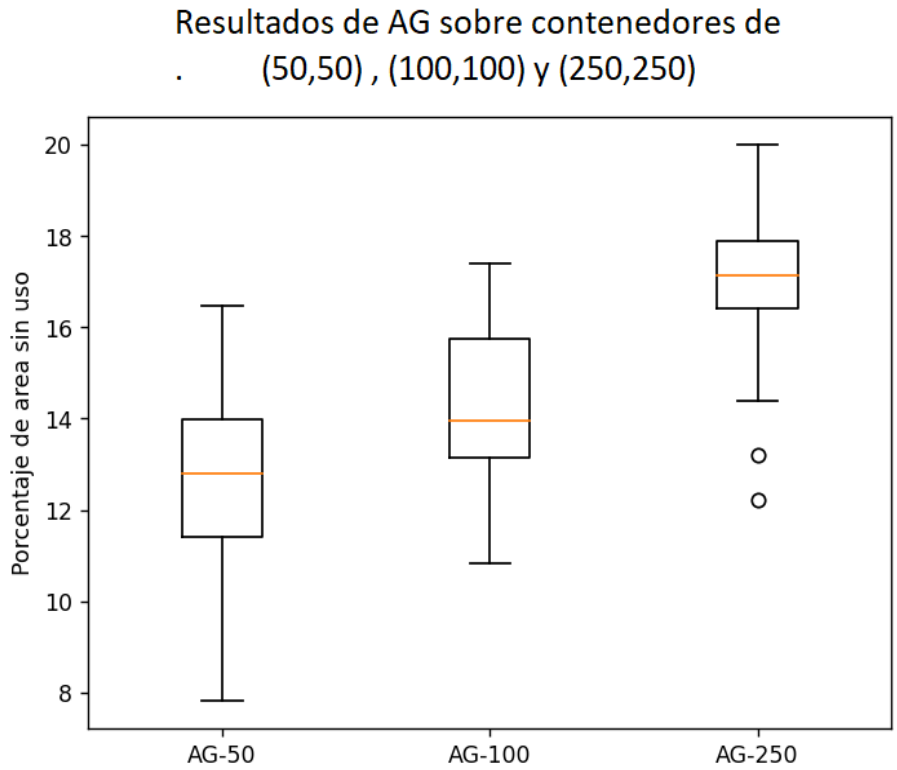
Con el algoritmo propuesto, que hace uso de las estructuras Arbol Guillotina , Árbol de Ordenamiento y Algoritmo de Ordenamiento, podremos encontrar un buen ordenamiento de las cajas en un buen tiempo de ejecución, en el siguiente capitulo mostraremos los resultados obtenidos.

4 – Resultados Obtenidos

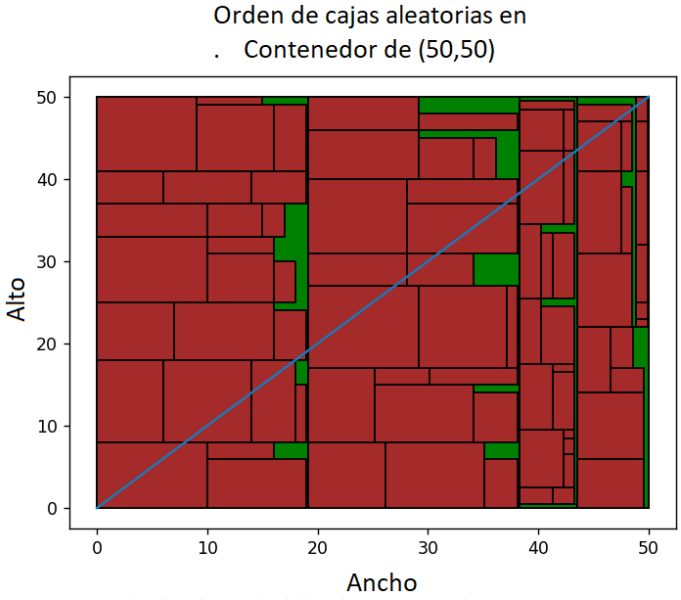
Ahora mostraremos los resultados obtenidos al usar el algoritmo propuesto en el capitulo 3 de este informe.

Se generaron 3 escenarios distintos, uno donde el contenedor es de dimencion 50 de ancho, 50 de alto , otro escenario con un contenedor de 100 de ancho y 100 de alto y otro de 200 de alto y 200 de ancho. Para cada uno de los 3 escenarios, se generaron 30 listas de cajas totalmente aleatorias y de ancho y alto como números enteros positivos.

Definimos para el AG 125 individuos, 50 ciclos y una altura de los arboles, para los 3 escenarios, de 3. Los resultados obtenidos son los siguientes

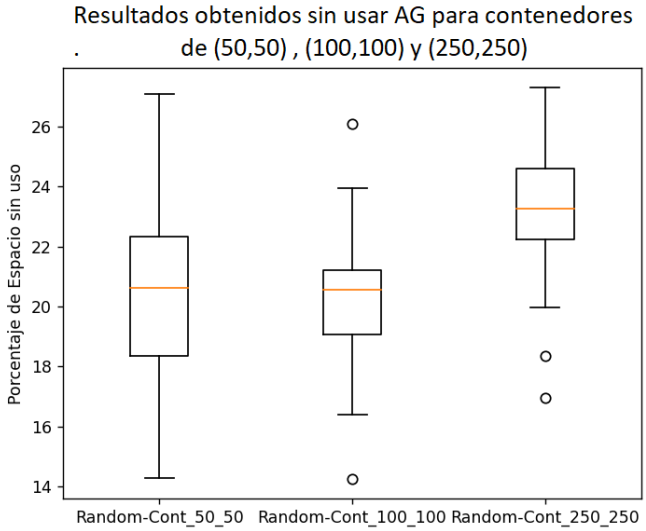


Con los resultados obtenidos vemos que mientras mas aumenta el tamaño de los contenedores para el algoritmo propuesto, obtenemos peores resultados.



Resultado obtenido del ordenamiento de cajas para un contenedor de dimensión 50

Ahora haremos la comparación del algoritmo propuesto con AG y un algoritmo que busca aleatoriamente un árbol guillotina que minimice los espacios desperdiciados, para los mismos escenarios de 50, 100 y 250.



Los resultados obtenidos suelen ser muy dispersos, estos en comparación con los obtenidos con el uso de un Algoritmo Genético suele ser peor, por el hecho de que AG agrega, además de búsquedas aleatorias al inicio, agrega una búsqueda mas profunda a medida que los ciclos pasan, en espacios de soluciones cada vez mas reducidos, con el pasar de los ciclos.

5 – Conclusiones

El algoritmo propuesto que hace uso de un algoritmo genético, tiene mejor desempeño que sin usar AG, los resultados mejoran bastante, y en un tiempo similar, podemos concluir que el uso del AG es importante para mejorar los resultados ya que este en un comienzo realiza una búsqueda aleatoria y luego con el pasar de varios ciclos, busca en espacios de soluciones mas

reducidos, reduciendo con el pasar de los ciclos aún mas los espacios de búsqueda hasta llegar a una población que no mejora. El algoritmo propuesto nos da buenos resultados, a pesar de ser bastante simple, se sabe que estos resultados pueden mejorar aún más, aumentando la cantidad de individuos de la población de AG , implementando programación paralela para reducir los tiempos, implementando PSO para mejorar los porcentajes de corte, mejorando el algoritmo de ordenamiento(2.1).

6 – Bibliografia

1- PROBLEMA DE EMPAQUETAMIENTO RECTANGULAR BIDIMENSIONAL TIPO GUILLOTINA - Universidad Tecnológica de Pereira

2- SOLUCIÓN DEL PROBLEMA DE EMPAQUETAMIENTO ÓPTIMO BIDIMENSIONAL EN UNA SOLA PLACA – David Alvarez Martinez