eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Algoritmos de asignación de recursos para la guía de un brazo robótico

Autor

Asier Esteban Sauce

2022

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Algoritmos de asignación de recursos para la guía de un brazo robótico

Autor

Asier Esteban Sauce

Director

Jose A. Pascual

Resumen

La organización de diferentes objetos en una caja, conocido como *Bin Packing*, es un problema importante cuando se trata de automatizar la producción de un almacén. Hoy en día, a menudo se usan algoritmos que controlan un brazo robótico diciéndole donde poner cada objeto. Sin embargo, estas soluciones, debido a la complejidad del problema, rara vez pueden ser aplicadas en un caso práctico y por eso Tekniker ha querido mejorar una aplicación que desarrollaron en el pasado. En este trabajo se propone una nueva implementación basada en el control de los espacios libres que generan los objetos con el fin de reducir el coste computacional sin perder la calidad de los resultados. Los experimentos realizados muestran que este nuevo enfoque más eficiente que la aplicación original y que otros métodos alternativos.

Índice general

Resumen I

Índice general III

Índice de figuras V

Índice de tablas VII

1. Introducción 1

2. Documento de objetivos del proyecto 3

3. Estado del arte 5

4. Algoritmos para solucionar el 3D Bin Packing Problem 7

4.1. Definición del problema 7

4.2. Algoritmo de la aplicación original 8

4.3. Basado en la regla de puntos extremos con una matriz de ocupación . . . 8

4.3.1. Puntos extremos 9

4.3.2. Descripción del algoritmo 10

4.3.3. Debilidades 12

4.4. Algoritmo basado en el control de espacios residuales 12

4.4.1. Espacios Residuales 13
4.4.2. Descripción del algoritmo 13
4.4.3. Análisis de complejidad 16
4.5. Extensiones 17

5. Experimentos **19**
5.1. Según forma y tamaño 19
5.1.1. Análisis de los resultados 20
5.2. Según cantidad de objetos 22
5.2.1. Análisis de los resultados 22

6. Conclusiones y Trabajo Futuro **25**

Bibliografía **27**

Índice de figuras

4.1. Ejemplo de los Puntos Extremos en 2D y 3D. Figura de [Crainic et al., 2008] 9

4.2. Proyecciones de los puntos del objeto 10

4.3. Matriz de ocupación 2D al insertar un nuevo objeto 11

4.4. Espacio residual que se genera en el eje Z 13

4.5. Representación del suelo estable de un espacio residual 14

4.6. Actualización de un espacio residual 15

4.7. Actualización de la base de un espacio residual 16

5.1. Tiempos de ejecución según formas y tamaños 21

5.2. Ratio de utilización de la caja 22

5.3. Tiempos de ejecución según la cantidad de objetos 23

Índice de tablas

5.1. Combinación de Forma y Dimensiones para generar objetos 20

1. CAPÍTULO

Introducción

En los últimos años, se ha creado mucho interés sobre como automatizar las producciones de un almacén. Concretamente, la tarea de empaquetar los productos en cajas para su envío o almacenamiento. Esta operación se conoce como *Bin Packing*. Una de las soluciones más habituales es usar un robot que se encargue de coger los objetos de una cinta y agruparlos de forma óptima.

Sin embargo, encontrar la posición idónea para las piezas debido al gran número de posibilidades para cada artículo y las restricciones a tener en cuenta para cada posición, como puede ser evitar una superposición entre los objetos, convierten esta tarea en un problema NP-Hard.

Actualmente, la mayoría de las soluciones planteadas, solo se encuentran en un marco teórico y no son aplicables en un caso práctico. Es por eso que el centro de investigación Tekniker, quiere desarrollar una aplicación que genere un *planning* que especifique en que lugar de la caja y en que orden se debe colocar cada objeto. Todo esto, de una manera lo suficientemente estable para que un brazo robótico pueda reproducirlo.

Para conseguir la estabilidad necesaria, se deben de tener en cuenta ciertos aspectos de los objetos que a menudo son ignorados en este tipo de problema, por ejemplo, el peso y la rigidez que tendrá un objeto. Estas variables suponen una gran diferencia en que el resultado final sea reproducible por el brazo robot.

Además, la calidad final del *planning* también será determinada por el tipo de pinza que use el robot para colocar los objetos. Considerando que en muchos casos no se tiene en

cuenta el espacio que ocuparía la pinza en el momento de colocar el objeto, generando resultados que no serán reproducibles para un brazo robótico.

Por otro lado, con la intención de automatizar y optimizar la ta tarea de *packing* en los centros de distribución, Tekniker desarrolló un prototipo de aplicación híbrida, donde los humanos y los robots colaborarían en un entorno seguro combinando automatización, flexibilidad y seguridad.

Una parte de ese proyecto incluía un algoritmo que generaba un *planning* offline. Es decir, se calculaba previamente en que lugar de la caja y en que orden se debía colocar cada objeto y más adelante las personas trabajadoras del centro de distribución reproducían este *planning*.

Por lo tanto, en el desarrollo de este Trabajo de Fin de Grado, utilizando como base la aplicación creada por Tekniker, se busca modificar el algoritmo para que añadiendo diferentes propiedades a los objetos, como el peso o una forma irregular, se puedan optimizar los tiempos de ejecución permitiendo calcular el *planning* rápidamente. Asimismo, favoreciendo la obtención de soluciones más estables que puedan ser reproducidas un brazo robótico sin perder rendimiento en el uso del espacio disponible de la caja.

Sin embargo, debido a la complejidad de generar un *planning* rápido y eficiente para formas irregulares, se ha tomado la decisión de simplificarlo con el fin de obtener un resultado funcional en un margen de tiempo razonable. Es por eso que este proyecto se centrará diseñar e implementar algoritmo nuevo que permita realizar los cálculos necesarios de manera rápida y eficiente con objetos de forma regular. Adicionalmente, será necesario implementarlo de manera modular para facilitar futuras modificaciones, por ejemplo, ser ejecutado en tiempo real.

Para explicar el desarrollo de este proyecto, la memoria está estructurada de la siguiente manera. Primero se definen los objetivos que deberá cumplir este trabajo. A continuación se realiza un análisis del estado del arte y se explica en profundidad en que consiste el problema y como funcionan las soluciones planteadas. Por último, se realizan y analizan los experimentos que analizan la eficiencia de los nuevos algoritmos.

2. CAPÍTULO

Documento de objetivos del proyecto

Con el fin de conseguir una aplicación de paletizado en tiempo real, Tekniker quiere modificar una aplicación que tenían en un proyecto anterior, que realizaba un empaquetado de 6 cajas de medicamentos de diferentes tamaños. Sin embargo, ese *planning* se calcula de forma offline, sin tener en cuenta que un brazo robótico va a tener que colocar los objetos uno a uno, es decir, sin valorar criterios como el peso, o restricciones a la hora de rotar ciertas caras. Debido a ello, la modificación de dicha aplicación para que funcione en tiempo real requeriría de cambios considerables. Por eso, se ha decidido diseñar la aplicación desde cero.

Para empezar a plantear este proyecto sin reinventar la rueda, primero hay que estudiar que investigaciones se han realizado sobre el tema. Para eso, se realizará un análisis del estado del arte sobre las técnicas más usadas en el campo de *Bin Packing Problem*, soluciones en 3D o en 2D que sirvan de base para el proyecto, en especial buscar trabajos que hayan tratado con objetos con formas más irregulares.

Después analizamos la aplicación original de Tekniker, que al ser una parte de una aplicación más grande, tiene bastantes funcionalidades que a este proyecto le sobran, así que lo primero es simplificar la arquitectura, dejando solo lo necesario para una solución que resuelva el problema en 3 dimensiones.

Con el objetivo de que un robot sea capaz de reproducir el *planning*, se añadirán propiedades a los objetos que influyan en el algoritmo, como el peso o la rigidez de los objetos. De esta manera, el algoritmo generará una solución más acorde con el mundo real en el que los objetos que más estabilidad generen se colocaran en la parte inferior de la caja y

que los objetos más frágiles acaben en la parte superior.

Por último, el algoritmo de Tekniker es demasiado lento a la hora de calcular las soluciones, que normalmente son poco estables para ser reproducidas paso a paso. Es por eso que se ha decidido diseñar un algoritmo nuevo más simple que genere soluciones estables y reproducibles rápidamente. Con el fin de confirmar que la nueva implementación cumple los objetivos, se realizaran varios experimentos que evaluaran el rendimiento.

Resumiendo lo anterior, se van a realizar las siguientes tareas en el desarrollo del proyecto:

- Estudiar el estado del arte.
- Analizar la aplicación de Tekniker.
- Añadir características a los objetos para calcular el *planning* teniendo en cuenta la aplicación real.
- Diseñar e implementar un algoritmo nuevo más eficiente.
- Evaluar la solución propuesta.

3. CAPÍTULO

Estado del arte

El empaquetado de objetos ha sido un problema clásico de combinatoria muy estudiado en el campo de la robótica. En concreto, encontrar la configuración óptima para que el set de objetos maximice el espacio disponible en la caja. En muchos casos este problema se plantea desde un punto de vista teórico con objetos rectangulares.

Aunque actualmente existan algoritmos que resuelvan el problema óptimamente, por ejemplo, el algoritmo de *branch&bound* propuesto por [Martello and Vigo, 1998], este problema pertenece al grupo de problemas NP-Hard y el coste temporal de encontrar la solución es demasiado alto.

Es por eso que [Den Boef et al., 2005] han usado varios heurísticos como *Bottom Left First* y *Best Fit First* con la intención de conseguir una aproximación óptima y en un tiempo razonable. El uso de metaheurísticos como el *Guided Local Search* [Faroe et al., 2003] o el *Tabu Search* [Crainic et al., 2009] han encontrado soluciones buenas para el *Bin Packing* en 2D y 3D. Además, [Hu et al., 2017] han aplicado diferentes estrategias de aprendizaje como el *Deep Reinforcement Learning* que han conseguido soluciones buenas generando un plan con formas cúbicas.

Con el fin de mejorar los heurísticos [Crainic et al., 2008] plantean una norma que optimiza los resultados y el tiempo de ejecución de diferentes heurísticos como *First Fit Decreasing* o *Best Fit Decreasing*. Sin embargo, estos algoritmos están planteados para encontrar una solución offline en objetos regulares y no tienen en cuenta las restricciones que genera un brazo robótico a la hora de replicar el *planning* calculado.

Por otro lado, enfocándose en un conjunto de objetos más irregulares, [Lutters et al., 2012]

propone un algoritmo basado en el *Brazil Nut Effect* [Jia and Williams, 2001] en el que, partiendo de una solución válida, trata cada objeto como una partícula y le aplica vibraciones con el fin de optimizar la solución. Lo interesante de esta investigación es la simplificación de los objetos irregulares sin convertirlos directamente a cubos. No obstante, es un enfoque experimental y los resultados que obtienen son muy similares a otras prácticas más establecidas.

Otra solución que plantean [Wang and Hauser, 2019] para el empaquetado de objetos en almacenes, consiste en minimizar un *heightmap* del estado de la caja, permitiendo así encontrar una solución estable y un uso del espacio mejor que otros algoritmos más comunes. Con este método son capaces de encontrar la orientación y posición de los objetos a empaquetar de manera óptima para las limitaciones de un brazo robot. A pesar de ser muy similar al objetivo de este proyecto, solo está demostrado para pequeños empaquetados de pocos objetos.

Partiendo de que el *Bin Packing Problem* es en esencia un problema de asignación de recursos, se pueden encontrar planteamientos interesantes en la manera en la que se gestionan las aplicaciones que usan los nodos en la red de un supercomputador. En la investigación de [Zhu, 1992] plantean las estrategias de *Best Fit* y *First Fit* para asignar cada aplicación a un subconjunto de nodos que estén libres.

Asimismo, [KIM and YOON, 1996] usando una lista global que contenga la información global de los nodos que estén libres y una nueva estrategia de *Best Fit* para evitar la mayor fragmentación posible. En este caso se puede interpretar que la red es el equivalente a la caja y que cada nodo libre representa un espacio disponible en el que asignar los objetos.

Siguiendo con la idea de utilizar el espacio libre en lugar de controlar los elementos que se insertan [Agarwal et al., 2020] proponen un algoritmo que, conociendo de antemano los objetos a insertar, genera un *planning offline* que pueda ser realizado por un brazo robot. Agarwal et al., también proponen un sistema de ajuste online en caso de variaciones por parte del entorno real y el brazo robot.

4. CAPÍTULO

Algoritmos para solucionar el 3D Bin Packing Problem

En este capítulo se explica en detalle en que consiste el *Bin Packing Problem*, además de describir el algoritmo desarrollado por Tekniker y proponer dos soluciones de mejor rendimiento.

La primera solución se basa en la investigación realizada por [Crainic et al., 2008] y utiliza una matriz de ocupación con el fin de reducir el coste computacional de recorrer la lista entera de objetos varias veces. En cambio, el segundo algoritmo, de manera similar al método que usan [Agarwal et al., 2020], mantiene un control de los espacios libres que generan los objetos, evitando así por completo recorrer la lista de objetos insertados.

4.1. Definición del problema

El problema que trata de solucionar este proyecto se llama *3D Bin Packing Problem* que consiste en empaquetar un conjunto de varios objetos de diferentes dimensiones (w_o, h_o, l_o) de forma regular, dentro de una caja 3D de tamaño fijo $W \times H \times L$.

Se asume que los objetos se colocaran paralelos a los ejes de la caja, y que pueden ser rotados en cualquier dirección siempre que se mantengan alineados con los ejes. También se asume que las dimensiones de cada objeto deberán de ser inferiores a las dimensiones de la caja $w_o \leq W, h_o \leq H$ y $l_o \leq L$. Asimismo, los objetos no se pueden superponer entre ellos.

Con la intención de conseguir un resultado más próximo a la realidad, cada objeto debe

de tener un mínimo del 60 % de su base colocada sobre una superficie para garantizar que el *planning* generado sea estable. Como los objetos deberán de ser empaquetados por un brazo robótico, también se tendrán en cuenta las limitaciones como restringir rotaciones en caras específicas, así como un offset generado por la pinza del robot a la hora de colocar el objeto.

4.2. Algoritmo de la aplicación original

A la hora de diseñar la aplicación, [Susperregi et al., 2020] se enfocaron en conseguir un cálculo del *planning* rápido que permitiera un uso dinámico en caso de algún error o variación del robot en el momento de reproducirlo. Este problema es NP-completo, por lo que el coste de los algoritmos más usados crece exponencialmente con el número de cajas a organizar. Para afrontar el problema, Tekniker decide usar el heurístico propuesto por [Dósa, 2007], *First Fit Decreasing (FFD)* ya que se puede implementar en $O(nlogn)$ aunque sin garantizar un resultado óptimo. Asimismo, el heurístico original solo tiene en consideración el volumen de los objetos, por lo que modifican el algoritmo para que valore también las formas de las cajas y sus limitaciones a la hora de rotarlas.

Para generar el *planning*, el algoritmo recibe una lista con los diferentes tipos de cajas, la cantidad de cada uno de ellos y si existe una restricción para rotar alguna cara. Con esa información se ordena la lista por el volumen de mayor a menor. Luego, buscando la primera posición más a la izquierda de la caja posible, comprueba que las dimensiones del objeto y se rota comprobando la lista de objetos insertados hasta que entre en la caja sin solapar ningún objeto. En el caso de que no entre, el objeto se descarta.

Con este método logran obtener resultados considerablemente rápidos cuando se trata de organizar cantidades pequeñas, habiendo así modificado el heurístico para que sea útil en una aplicación funcional del mundo real.

4.3. Basado en la regla de puntos extremos con una matriz de ocupación

Analizando el algoritmo original (ver Sección 4.2), hay varios puntos posibles para mejorar la aplicación. Usando la regla de puntos extremos [Crainic et al., 2008] buscamos optimizar el encontrar la posición para los objetos, proporcionando una serie de puntos

posibles en los que colocar los siguientes objetos. Con el fin de no recorrer la lista de objetos insertados cada vez que insertamos un objeto nuevo, se plantea el uso de una matriz de ocupación, en la cual, se guardará el espacio ocupado por cada objeto.

4.3.1. Puntos extremos

La idea principal de los *Puntos Extremos (PE)* es extender la idea de los de los *Puntos de esquina* para optimizar el uso del espacio libre que se define por los objetos que están dentro de la caja. En la Figura 4.1 se muestran los *PE* en 2D y 3D.

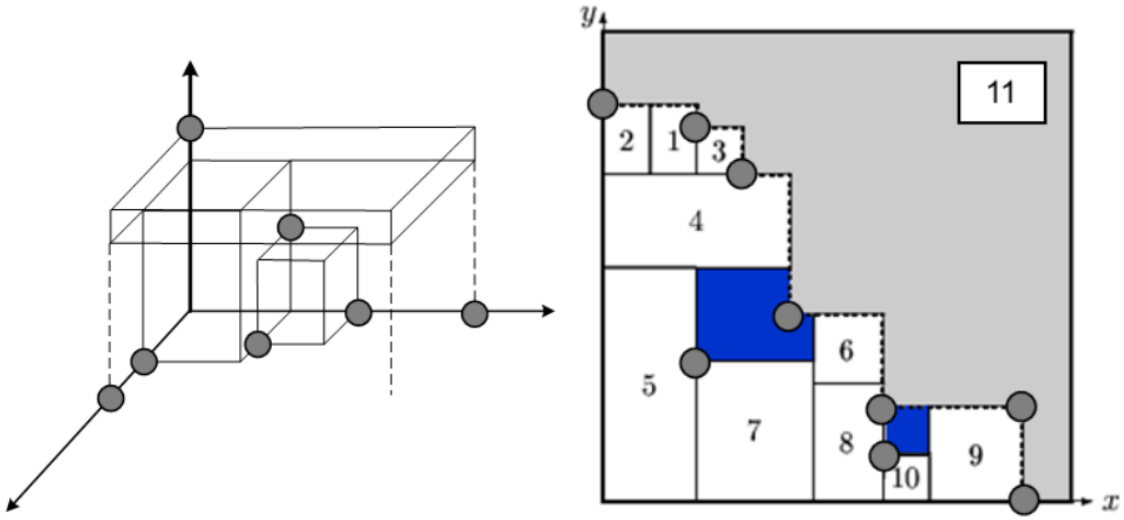


Figura 4.1: Ejemplo de los Puntos Extremos en 2D y 3D. Figura de [Crainic et al., 2008]

Los *Puntos Extremos (PE)* consisten en que por cada objeto o de dimensiones w_o, h_o y l_o y apoyándolo sobre la esquina inferior izquierda se generan un máximo de 6 potencialmente puntos nuevos donde colocar los siguientes objetos. Para obtener estos puntos se proyectan 3 esquinas sobre los ejes X, Y y Z (ver Figura 4.2) donde intercepta algún objeto o en su defecto, la pared de la caja.

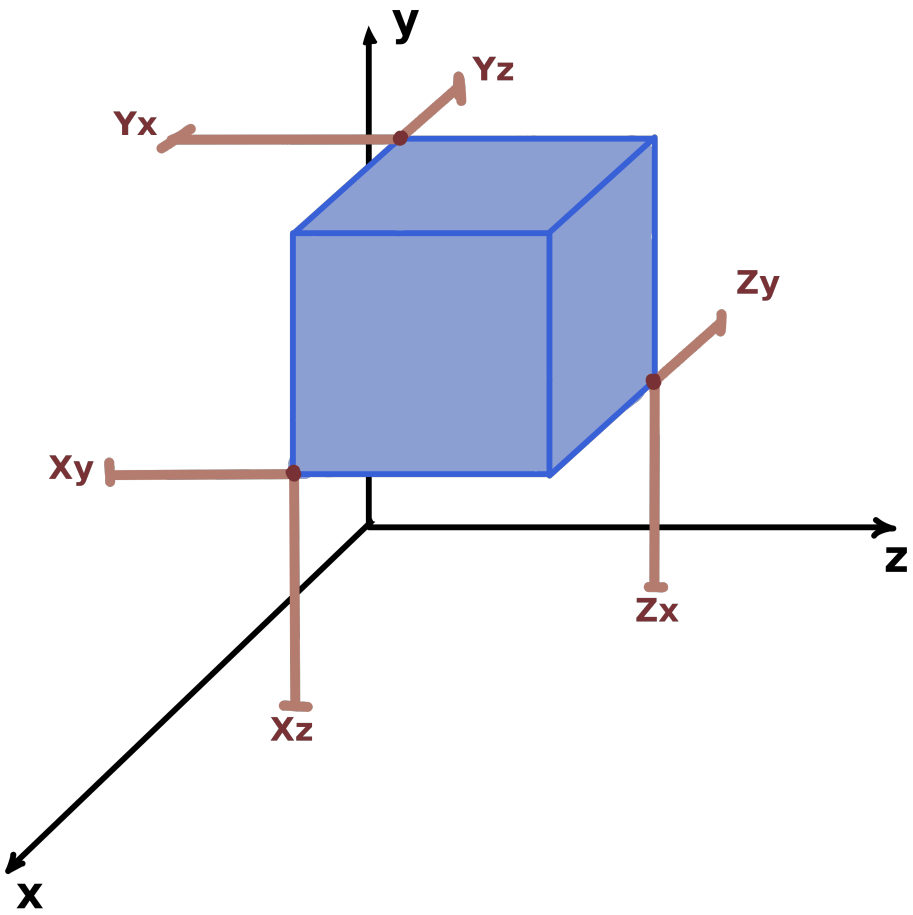


Figura 4.2: Proyecciones de los puntos del objeto

4.3.2. Descripción del algoritmo

Lo primero de todo se realiza una organización de los objetos de menor a mayor, basándose en un heurístico que tendrá en cuenta el volumen, el peso y la rigidez del objeto. Luego, siguiendo la regla de los puntos extremos, se inserta el primer objeto en la posición $(0,0,0)$ sobre la esquina inferior izquierda, y se crearan 3 nuevos PE en las posiciones $(x_o+w_o,y_o,z_o),(x_o,y_o+h_o,z_o)$ y (x_o,y_o,z_o+l_o) con los que se iniciará el bucle.

Para los siguientes objetos se recorre la lista de puntos posibles, asegurándose de que el objeto cumple las restricciones en esa posición, es decir, si está dentro de la caja y no se sobrepone con ningún objeto. Asimismo, para garantizar una estabilidad mínima se analizan las celdas inferiores y se rota el objeto en todas las direcciones posibles, obteniendo así la orientación más estable para la posición. En la Figura 4.3 se representa en 2D la matriz de ocupación.

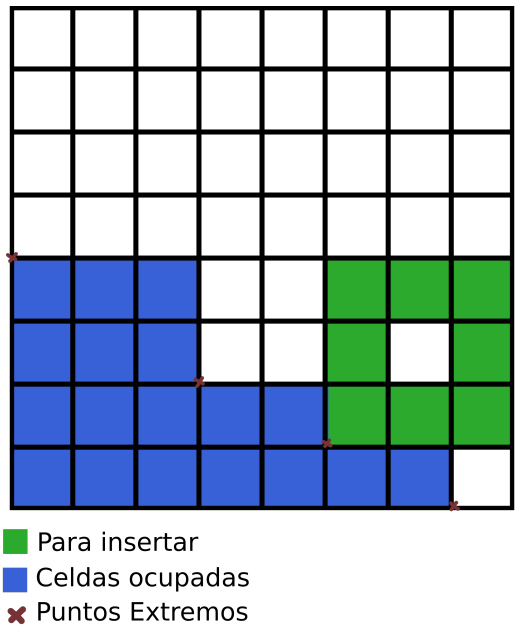


Figura 4.3: Matriz de ocupación 2D al insertar un nuevo objeto

Una vez comprobado que el objeto se puede insertar en esa posición con una estabilidad mínima del 60 %, se puntúan más alto las posiciones que tengan la altura más baja, y se multiplica la puntuación por el porcentaje de superficie que esté en contacto en su base.

$$estabilidad = puntuar\_altura() * \frac{area\_suelo}{base\_objeto}$$

Una vez puntuados todos los objetos, se obtiene la posición con la puntuación más alta. En el caso de que esa puntuación sea 0 significaría que no se ha encontrado una posición válida para el objeto, en cuyo caso habría que descartarlo y continuar con el siguiente. Después de insertar el objeto, se actualiza la lista de PE con el nuevo objeto.

Como se ha mencionado al principio de la sección, con el fin de reducir el número de veces que se recorre la lista de objetos insertados en la caja, se usa una matriz de ocupación que representara con una resolución de milímetros el espacio que ocupan los objetos dentro de la caja, por ejemplo, un objeto de 25 mm de largo ocupará 25 posiciones en la matriz. De esta manera, para comprobar si un objeto se solapará con otro objeto, recorremos el perímetro del objeto en la matriz, comprobando que el espacio que ocupara estará libre.

4.3.3. Debilidades

En este apartado se proceden a explicar las debilidades de este algoritmo que justifican la decisión de descartarlo y buscar una alternativa más óptima y simple. Concretamente, se han encontrado las siguientes 3 limitaciones:

1. Con el fin de tener precisión, la resolución de la matriz es milimétrica. El problema de utilizar una matriz de ocupación con esa resolución es que se requiere de mucha memoria, y en este caso en un PC con 8 GB de memoria RAM no es capaz de solucionar problemas con matrices de tamaño superior a $1000x1000x1000$.

2. Los objetos reales no son objetos milimétricos, por lo que su tamaño en milímetros ocupa muchas celdas. Es por eso que aunque comprobar sí se solapa con otros objetos, tiene un coste constante independientemente de cuantos objetos haya dentro, sigue siendo un coste considerablemente alto como para que sea una implementación viable.

3. Para obtener los PE de cada objeto hay que proyectar el punto comprobando si se proyecta sobre algún objeto que haya dentro, recorriendo la lista de objetos insertados un total de 6 veces dos por cada eje, lo que hace que al insertar muchos objetos el coste computacional se incremente considerablemente.

4.4. Algoritmo basado en el control de espacios residuales

La razón de que los anteriores algoritmos no sean eficientes a la hora de empaquetar una gran cantidad de elementos se debe a que por cada nuevo elemento hay que recorrer una lista con todos los elementos insertados previamente, es decir, que el coste computacional de la ejecución va creciendo con cada elemento nuevo que se mete.

Sin embargo, si en lugar de recorrer la lista de los objetos insertados, bien para comprobar que no solapa ningún otro objeto o bien para encontrar una posición válida, se mantiene un control sobre el espacio residual que generan los objetos, se obtiene una variable que se va reduciendo a medida que el algoritmo avanza reduciendo así el coste computacional.

4.4.1. Espacios Residuales

Los espacios residuales que genera cada objeto se representan como volúmenes de forma rectangular. Como los objetos nuevos se insertan en los espacios libres, al insertar un objeto se fracciona ese espacio en 3 nuevos espacios residuales en los puntos $(x_{rs}+w_o, y_{rs}, z_{rs}), (x_{rs}, y_{rs}+h_o, z_{rs})$ y $(x_{rs}, y_{rs}, z_{rs}+l_o)$. Por ejemplo, en la Figura 4.4 se ve que el espacio residual que se genera en el punto $(x_{rs}, y_{rs}, z_{rs}+l_o)$ tendrá las mismas dimensiones que el espacio libre original, excepto en el eje Z, donde se le resta el ancho del objeto.

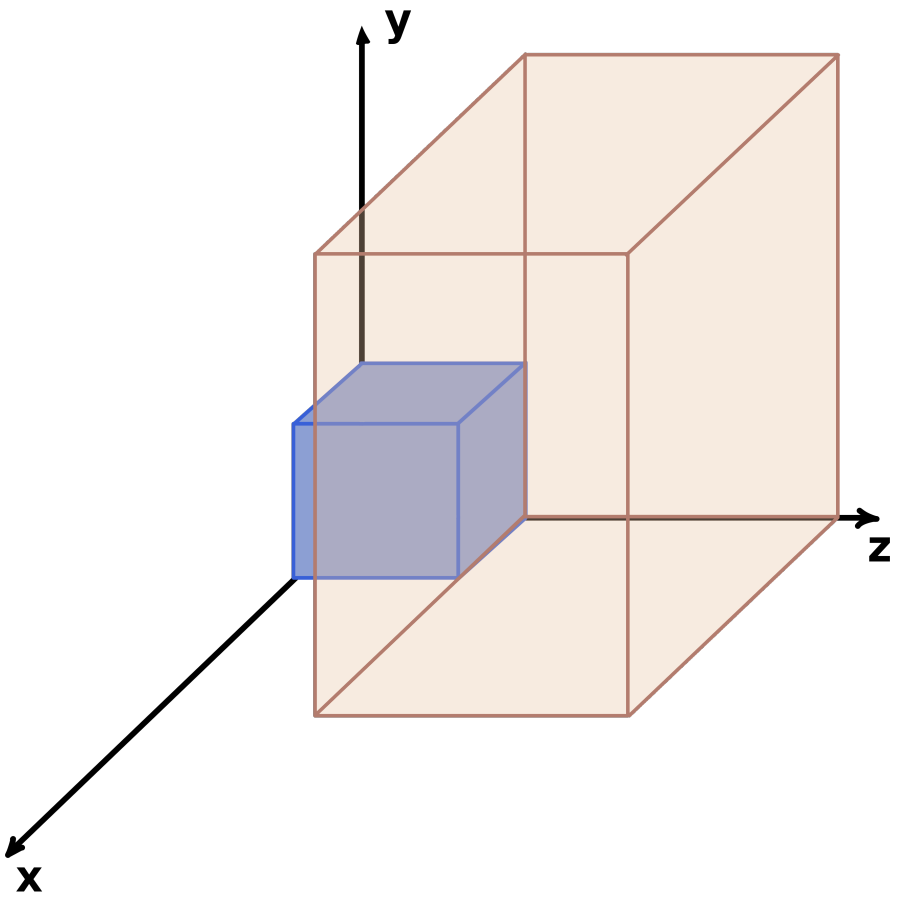


Figura 4.4: Espacio residual que se genera en el eje Z

4.4.2. Descripción del algoritmo

La estructura de este algoritmo es igual que en el anterior, primero se ordenan los objetos a insertar mediante un heurístico con el que se puntúan las propiedades del objeto, es decir, volumen, peso y rigidez. Luego, empezando por la esquina inferior izquierda de

la caja, se inserta el primero y se generan una serie de nuevos espacios posibles donde colocar los siguientes objetos.

Para los siguientes objetos, por cada espacio libre se comprueba si el objeto está dentro del espacio y no sobresale en ningún eje. Para los espacios que no cumplen esta condición no se tienen en consideración a la hora de valorar la posición del objeto.

Para asegurar un mínimo de estabilidad, cada espacio dispone de un parámetro en el que guardan dos puntos que representan la superficie de contacto que proporciona el espacio residual. En la Figura 4.5 aparece un caso en el que al tener un área superior a la de la base del objeto, el espacio le proporciona una estabilidad del 100 %.

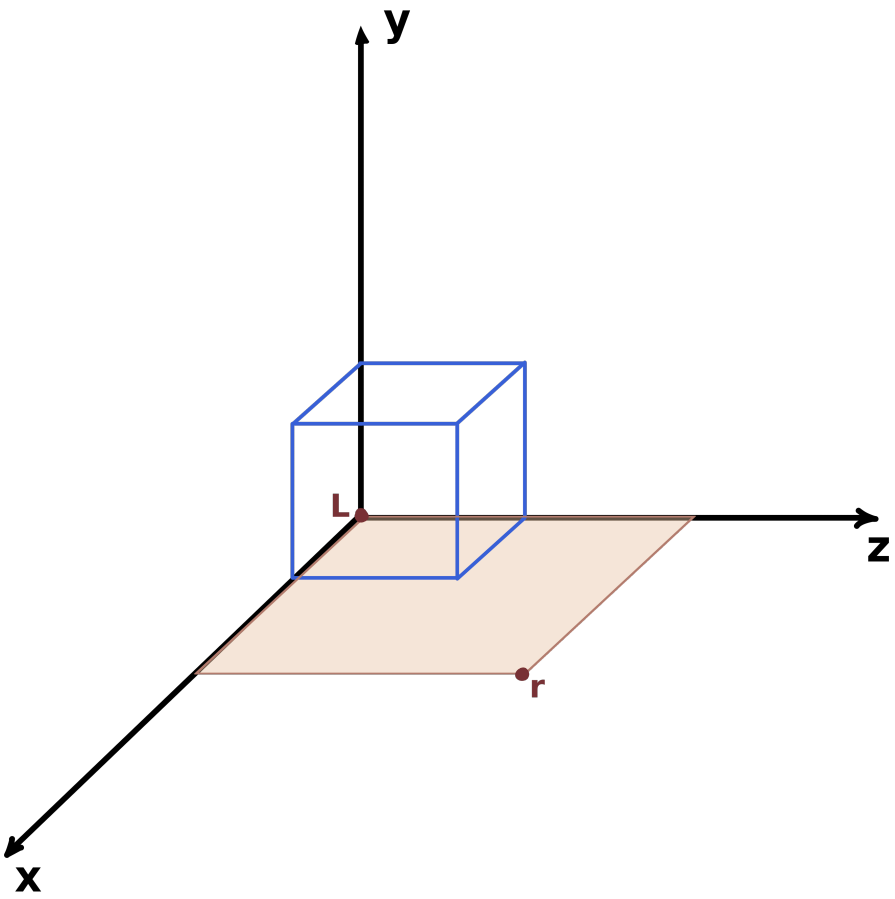


Figura 4.5: Representación del suelo estable de un espacio residual

Una vez que se han valorado todos los espacios, se coloca el objeto en el que tenga la puntuación más alta y se borra de la lista. Con el objeto ya colocado, se recorren todos los espacios residuales, ajustando las dimensiones de cada uno en el caso de que el objeto lo solape o la superficie de contacto en caso de que la altura del nuevo objeto coincida con la base del espacio.

En la Figura 4.7 se muestra un caso en el que el objeto crea dos opciones a las que actualizar las dimensiones. En estos casos siempre se actualizará a la opción que deje un volumen más grande.

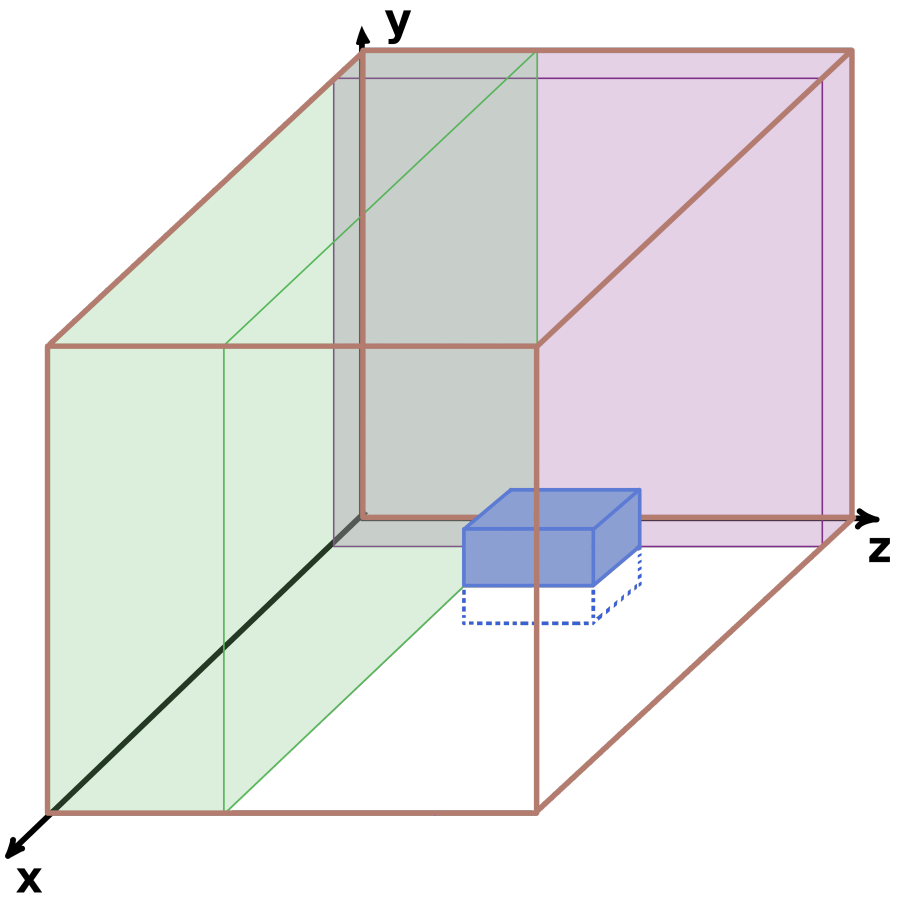


Figura 4.6: Actualización de un espacio residual

Para actualizar el área de base estable que ofrece cada espacio residual, el objeto nuevo tiene que estar a la misma altura que la base del espacio y en contacto con alguno de los vértices que forman la superficie. Después se le suma las dimensiones a la superficie anterior. Como se ve en la Figura 4.7 esto genera un pequeño margen de error sobre la estabilidad real, sin embargo, no es lo suficientemente relevante como para que las soluciones generadas sean inestables.

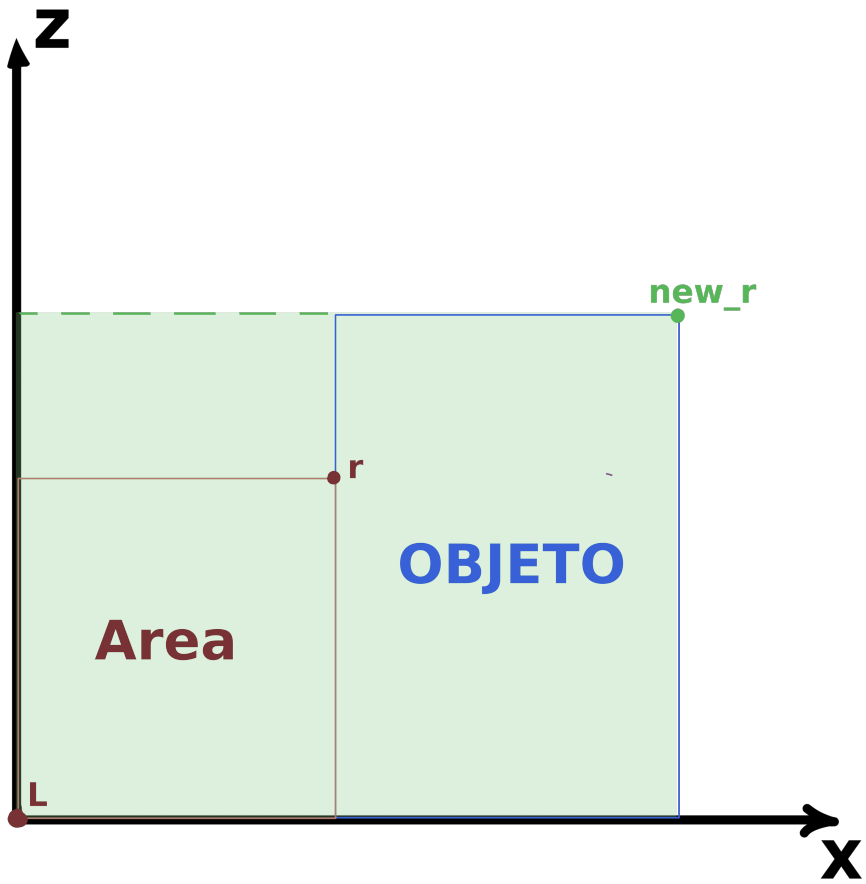


Figura 4.7: Actualización de la base de un espacio residual

Por último, se fragmenta el espacio residual en el que se ha insertado el objeto en 3 nuevos espacios y se borra de la cualquier espacio duplicado que se haya podido generar.

4.4.3. Análisis de complejidad

Tal y como se explica en la Sección 4.4 el algoritmo empieza ordenando la lista de objetos con un coste $O(nlogn)$. Luego se recorre la lista entera de n objetos una vez. Como se ve en el Algoritmo 1 por cada objeto se realizan dos bucles. El primero consiste en recorrer los espacios residuales para encontrar la posición más óptima para el objeto.

Por cada objeto que se inserta, se borra el espacio libre que ocupa y se fragmenta en tres nuevos espacios residuales. Por lo tanto, en el último objeto los dos bucles interiores son recorridos un total de $2n+1$ veces, que supone un coste de $O(n)$.

Algorithm 1 Algoritmo de control de Espacios Residuales

Require: $Objetos$: Lista de objetos a insertar
Require: RSL: Lista de espacios residuales dentro de la caja
  for $o \in Objetos$ **do**
    for $rs \in RSL$ **do**
      $Puntuaciones \leftarrow PuntuarEspaciosResiduales(o, rs)$
    end for
    $maxRs \leftarrow MaximioEspacioResidual(Puntuaciones)$
    $InsertarObjetoEnCaja(o)$
    $BorrarEspacioResidual(maxRs)$
    for $rs \in RSL$ **do**
      $ActualizarEspaciosResiduales(o, rs)$
    end for
    $RSL \leftarrow FragmentarEspacioResidual(o, maxR)$
  end for

En el análisis estático este algoritmo tiene un coste lineal de $O(n)$ por cada objeto que se inserta en la caja. Sin embargo, en la ejecución, el coste es inferior porque a medida que avanza el algoritmo, el espacio libre de la caja disminuye y en consecuencia el número de espacios residuales se reduce.

4.5. Extensiones

En esta sección se explican las diferentes funcionalidades que complementan el algoritmo. Estas extensiones permiten influir en el algoritmo para facilitar el uso práctico de la aplicación.

En un caso real, dependiendo del tipo de pinza o agarre que se use en el robot, la colocación y la orientación de los objetos se ve limitada, es por eso que la aplicación dispone de una opción en la que se restringen las rotaciones sobre los ejes seleccionados evitando así que esa cara se tenga en cuenta a la hora de calcular el *planning*.

Además, también se permite añadir un offset a las dimensiones del objeto que representara el espacio que ocuparía la pinza del robot en los ejes laterales a la hora de colocar los objetos. De esta manera, a la hora de generar el *planning* se ajustará mejor a la ejecución del mundo real. Para ello, en el momento de comprobar cuanto espacio ocupara el objeto, se modifica el *bounding box* del objeto que posteriormente se eliminara, representando así con exactitud el comportamiento de un brazo robot.

Como primer paso a un generador de *plannings* online, se ha añadido la opción de generar el plan desde una caja que no este vacía. Para lograr esto, sabiendo la posición de cada objeto en la caja, se reproduce el algoritmo actualizando y creando los espacios residuales. De esta manera se consiguen el estado correcto de la caja, permitiendo continuar el algoritmo correctamente.

Para obtener una aplicación más genérica, se ha implementado una función que permite importar archivos *.stl* de un modelo 3D de los objetos que se quieran empaquetar. En este momento esa función solamente calcula el *bounding box* del modelo, aunque más adelante se podrá modificar para obtener las dimensiones exactas de los objetos y calcular el *planning* con formas irregulares.

5. CAPÍTULO

Experimentos

En este capítulo se analizan las pruebas realizadas a los algoritmos explicados en el Capítulo 4. Para ello se han definido dos tipos de experimentos, el primero consiste en analizar el comportamiento de cada algoritmo al tratar con diferentes formas y tamaños. En el segundo se quiere ver como afecta el coste computacional con grandes cantidades de objetos.

Cada experimento se ha ejecutado 10 veces con objetos generados aleatoriamente, usando los mismos objetos en cada algoritmo y generando nuevos para cada iteración. Posteriormente, se hará la media de esas 10 iteraciones para obtener los resultados finales.

El código desarrollado se encuentra alojado en el repositorio privado de Tekniker y no es posible su difusión pública.

5.1. Según forma y tamaño

Como se ha explicado antes, este experimento analiza como afectan los diferentes tamaños y formas a cada algoritmo. Con ese objetivo se usan dos dimensiones de cajas: una de forma cuadrada y otra con una forma más irregular de dimensiones $400x400x400$ $600x400x320$ respectivamente.

En cuanto a los objetos, se han establecido 3 formas diferentes que representan los diferentes tipo de objetos cúbicos, una forma rectangular y regular con dimensiones cercanas a un cubo; otra con un carácter más plano, es decir, la altura del objeto será el 30 % del

lado más pequeño. Por último, una forma que representa un listón tendrá dos de los lados con una dimensión del 20 % del tamaño del lado restante.

Con el fin de analizar como influyen las dimensiones, se generan los objetos de manera aleatoria en 3 rangos diferentes: un rango de $[50, 100]$ para representar objetos pequeños, otro de $[100, 200]$ para objetos grandes y $[200, 400]$ para los objetos muy grandes.

Para tener una imagen completa realizamos 9 combinaciones de esas características (ver Tabla 5.1) y medimos el tiempo de ejecución de cada algoritmo y el ratio de uso de la caja, es decir, el porcentaje del volumen ocupado.

Experimento	Tamaños	Forma
1	Grande, Muy grande	Rectangular
2	Grande, Muy grande	Plana, Listón
3	Grande, Muy grande	Rectangular, Plana, Listón
4	Pequeño	Rectangular
5	Pequeño	Plana, Listón
6	Pequeño	Rectangular, Plana, Listón
7	Grande, Pequeño	Rectangular
8	Grande, Pequeño	Plana, Listón
9	Grande, Pequeño	Rectangular, Plana, Listón

Tabla 5.1: Combinación de Forma y Dimensiones para generar objetos

5.1.1. Análisis de los resultados

Como se ve en la Figura 5.1 el algoritmo basado en los espacios residuales obtiene tiempos de ejecución considerablemente menores que el resto de algoritmos en todos los casos. No obstante, se puede observar que empaquetar objetos de tamaño grande al ocupar mucho más espacio por cada objeto, la caja se llena antes y se obtienen tiempos inferiores en todas las aplicaciones.

Por otro lado, con la matriz de ocupación el tamaño de los objetos tiene una gran influencia en su rendimiento, es por eso que cuanto más pequeños sean los objetos, el algoritmo obtiene mejores tiempos.

En la última fila de la Figura 5.1 se enseñan los resultados de los experimentos que empaquetan más objetos. En estas pruebas ya se puede observar la mejora en el rendimiento que se obtiene controlando los espacios residuales en comparación con el algoritmo de Tekniker.

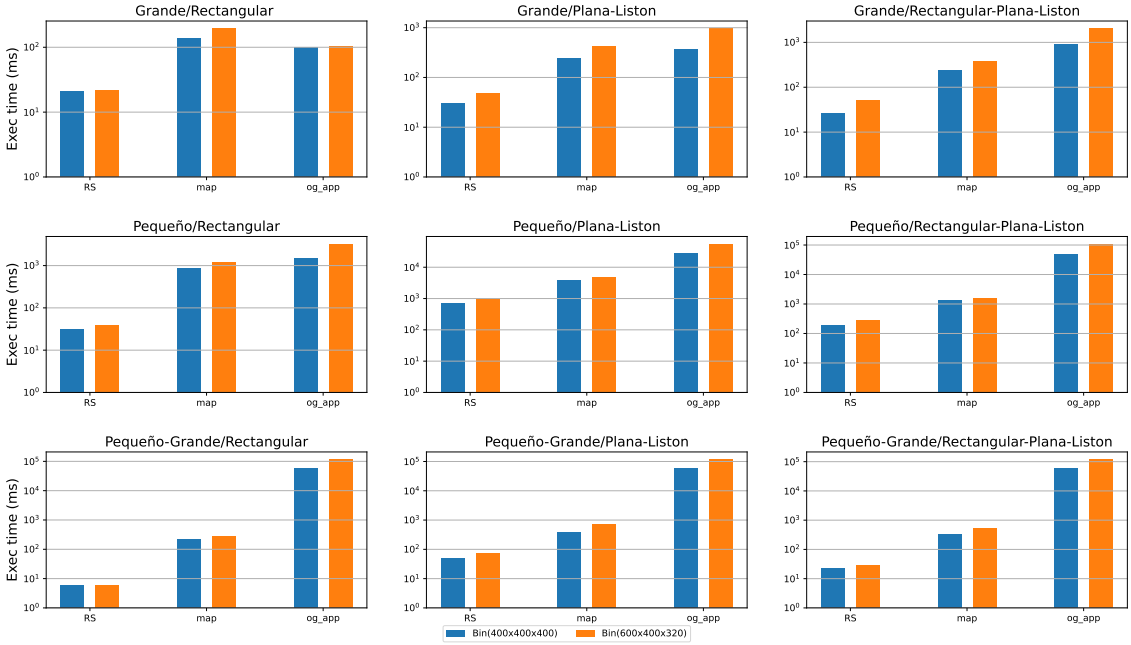


Figura 5.1: Tiempos de ejecución según formas y tamaños

En los resultados del ratio de uso de la caja que se muestran en la Figura 5.2 se puede observar como al usar objetos de gran tamaño la caja se usa de manera menos eficiente aprovechando únicamente un 70 % de la caja, no obstante esto se debe a que el tamaño de los objetos impide que entren más y no tiene nada que ver con la calidad de los algoritmos. De hecho, en la mayoría de los casos se obtienen resultados similares.

Al empaquetar objetos más pequeños se puede observar una gran diferencia en el ratio de uso, empaquetando solo objetos con forma regular o mezclando las diferentes formas, en el que el espacio disponible de la caja se utiliza de manera más eficiente. Por otro lado, las diferencias en los resultados de cada algoritmo no son muy significativas. Sin embargo, el algoritmo de la matriz de ocupación obtiene resultados ligeramente superiores, aunque debido al alto coste computacional de ese algoritmo no compensa el uso de la caja con el rendimiento en su tiempo de ejecución.

Por último, cuando medimos los tiempos de ejecución (ver Figura 5.1) no se aprecia ninguna diferencia en los resultados que indique que la forma de los objetos que se empaquetan tenga alguna influencia. Sin embargo, al analizar la Figura 5.2 se ve que al insertar objetos de formas planas, los algoritmos utilizan mejor el espacio disponible.

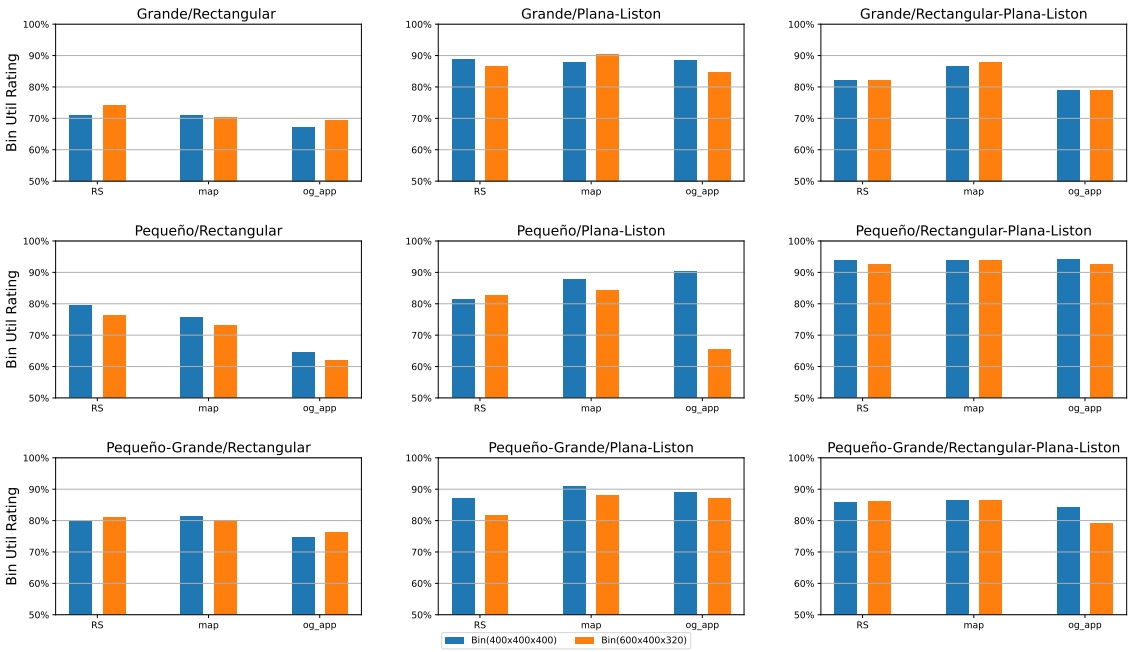


Figura 5.2: Ratio de utilización de la caja

5.2. Según cantidad de objetos

Con la intención de llevar los algoritmos al límite y ver como les afecta una gran cantidad de objetos, se ha definido una caja de dimensiones $1000 \times 1000 \times 1000$ y se han generado objetos de manera aleatoria en un rango de $[50, 100]$ con la intención de que entre una gran cantidad de ellos y con todas las formas definidas en la sección anterior (ver 5.1).

Cada algoritmo se ha ejecutado 4 veces insertando cada vez más objetos, concretamente $60, 300, 600, 3000$, siendo 3000 una aproximación del máximo de objetos de este tipo en una caja de este tamaño.

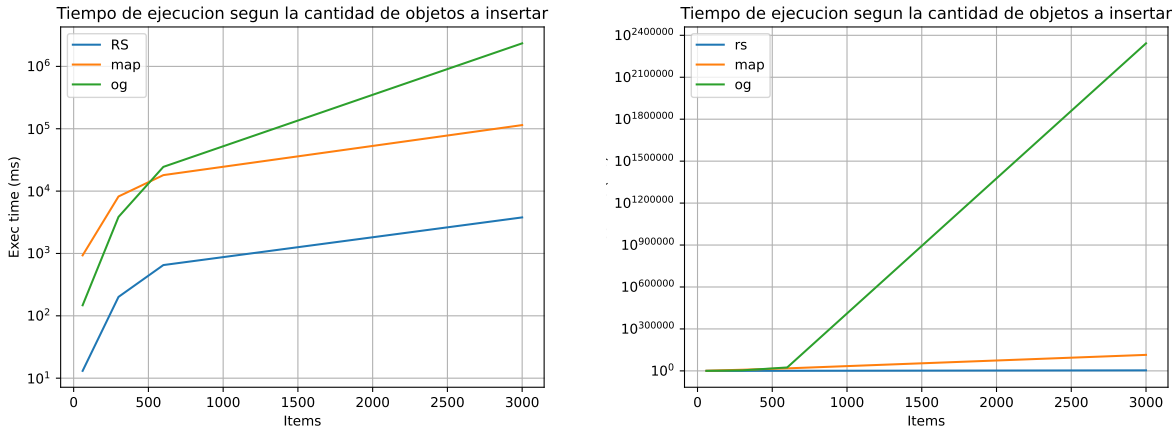
5.2.1. Análisis de los resultados

En la Figura 5.3 se muestra como la aplicación original tiene un crecimiento en el tiempo de ejecución muy superior que cualquiera de las propuestas. En el gráfico de la izquierda se muestran los resultados en escala logarítmica, donde se puede ver como los dos algoritmos propuestos tienen un coste menos lineal.

Sin embargo, en el gráfico de la derecha se observa claramente la gran mejora que se obtiene con los nuevos algoritmos. Por ejemplo, en el experimento número 3, insertando 600

objetos, la aplicación de Tekniker tardaba una media de $2,45$ segundos. No obstante, el algoritmo del mapa y el de los espacios residuales, la superan con un tiempo de ejecución de $1,52$ y $0,63$ segundos respectivamente.

Asimismo, en la Figura 5.3 (a), se aprecia que en el primer experimento, con una cantidad pequeña de objetos, tarda aproximadamente 1 segundo en comparación con los 124 milisegundos en los que se ejecuta la aplicación original, debido al número de iteraciones que requiere cada objeto. No obstante, cuando el número de objetos se incrementa, se obtiene un tiempo claramente inferior.



(a) Escala logaritmica **(b)** Escala normal

Figura 5.3: Tiempos de ejecución según la cantidad de objetos

6. CAPÍTULO

Conclusiones y Trabajo Futuro

La automatización de los paletizados en almacenes es un problema que ha sido investigado a lo largo de los años generando soluciones cada vez más eficientes a la hora de ser aplicadas. A menudo, la mayoría de las soluciones que se plantean rara vez salen del marco teórico, ignorando factores importantes si se quisiera aplicar la solución en un entorno real.

Con la intención de desarrollar un algoritmo que genere una solución que pueda ser realizada por un brazo robot, Tekniker quiere mejorar su aplicación hibrida para eliminar por completo la interacción humana en el proceso de automatización. La aplicación original, a pesar de generar soluciones finales óptimas, no tiene en consideración la estabilidad de dicha solución y al tener una gran cantidad de objetos, el tiempo de ejecución es considerablemente alto.

Teniendo como objetivo principal optimizar el tiempo de ejecución del algoritmo original, así como plantar las bases para un generador de mosaicos de objetos irregulares, en este trabajo se plantean dos algoritmos: utilizando la norma de puntos extremos con una matriz de ocupación y manteniendo un control de los espacios libres que genera la inserción de los objetos.

El primero, mediante los puntos extremos, consigue optimizar la elección de las posiciones de los objetos reduciendo el coste computacional. Además, usando una matriz de ocupación se controla que espacios de la caja están ocupados y proporciona una opción viable para objetos irregulares. No obstante, debido a la alta resolución que necesita la matriz, este método consume una gran cantidad de memoria RAM y el coste computacio-

nal de insertar objetos grandes es muy alto.

Por otro lado, manteniendo un control de los espacios libres, se elimina la necesidad de una matriz de ocupación, reduciendo considerablemente el tiempo de ejecución de una operación que con los métodos anteriores, debido al gran número de recorridos, tenían un coste computacional muy alto. También se simplifican los puntos extremos, por lo que ya no se recorre la lista de objetos insertados, consiguiendo así un algoritmo rápido y eficiente.

En este trabajo, se han realizado una serie de experimentos que determinan empíricamente que los cambios realizados optimizan la aplicación original sin perder calidad en los resultados obtenidos. En lo que se refiere a la eficiencia con diferentes formas, se observa que el nuevo algoritmo que controla los espacios libres obtiene resultados similares en un tiempo mucho más reducido. Del mismo modo, en el segundo experimento se demuestra que a diferencia de los otros dos, a la hora de insertar una gran cantidad de objetos, su coste computacional tiene un crecimiento muy inferior.

Sin embargo, el proyecto es una implementación parcial fácilmente ampliable del objetivo final de Tekniker. Empezando por transformar el algoritmo para generar el *planning* de manera online. De esta manera, en caso de que el robot falle y los objetos colocados no estén alineados a los ejes de la caja, se podría implementar un sistema que corrija las dimensiones obteniendo un *bounding box* que simule la alineación con el fin de no obstaculizar los futuros objetos.

Además, Tekniker quiere ser capaz de planificar con objetos de forma irregular. Para conseguir esta funcionalidad, se puede implementar un sistema que pre agrupe los objetos para aproximarlos a una forma regular, o especificar para cada objeto que se quiera insertar como se fragmenta el espacio libre en el que se inserta.

Bibliografía

[Agarwal et al., 2020] Agarwal, M., Biswas, S., Sarkar, C., Paul, S., and Paul, H. S. (2020). Jampacker: An efficient and reliable robotic bin packing system for cuboid objects. *IEEE Robotics and Automation Letters*, 6(2):319–326.

[Crainic et al., 2008] Crainic, T. G., Perboli, G., and Tadei, R. (2008). Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing*, 20(3):368–384.

[Crainic et al., 2009] Crainic, T. G., Perboli, G., and Tadei, R. (2009). Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744–760.

[Den Boef et al., 2005] Den Boef, E., Korst, J., Martello, S., Pisinger, D., and Vigo, D. (2005). Erratum to "the three-dimensional bin packing problem": robot-packable and orthogonal variants of packing problems. *Operations Research*, 53(4):735–736.

[Dósa, 2007] Dósa, G. (2007). The tight bound of first fit decreasing bin-packing algorithm is ffd (i)$\leq$ 11/9opt (i)+ 6/9. In *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer.

[Faroe et al., 2003] Faroe, O., Pisinger, D., and Zachariasen, M. (2003). Guided local search for the three-dimensional bin-packing problem. *Informs journal on computing*, 15(3):267–283.

[Hu et al., 2017] Hu, H., Zhang, X., Yan, X., Wang, L., and Xu, Y. (2017). Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*.

[Jia and Williams, 2001] Jia, X. and Williams, R. A. (2001). A packing algorithm for particles of arbitrary shapes. *Powder technology*, 120(3):175–186.

[KIM and YOON, 1996] KIM, G. and YOON, H. (1996). Free submesh list strategy: A best fit submesh allocation in mesh connected multicomputers. *Parallel Processing Letters*, 06(01):75–86.

[Lutters et al., 2012] Lutters, E., Ten Dam, D., and Faneker, T. (2012). 3d nesting of complex shapes. *Procedia Cirp*, 3:26–31.

[Martello and Vigo, 1998] Martello, S. and Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management science*, 44(3):388–399.

[Susperregi et al., 2020] Susperregi, L., Fernandez, A., Molina, J., Iriondo, A., Sierra, B., Lazkano, E., Martinez Otzeta, J. M., Altuna, M., Zubia, L., and Bautista, U. (2020). Rsaii: Flexible robotized unitary picking in collaborative environments for order preparation in distribution centers. In *Bringing Innovative Robotic Technologies from Research Labs to Industrial End-users*, pages 129–151. Springer.

[Wang and Hauser, 2019] Wang, F. and Hauser, K. (2019). Stable bin packing of non-convex 3d objects with a robot manipulator. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8698–8704. IEEE.

[Zhu, 1992] Zhu, Y. (1992). Efficient processor allocation strategies for mesh-connected parallel computers. *Journal of Parallel and Distributed Computing*, 16(4):328–337.