



UNIVERSIDAD VERACRUZANA

INSTITUTO DE INVESTIGACIONES EN
INTELIGENCIA ARTIFICIAL

*Un método para el acomodo de
objetos basado en intercambios
ponderados, que minimiza el
costo de acceso global a estos*

Trabajo recepcional en la modalidad de:

TESIS

que como requisito parcial para obtener el título de:

Maestría en Inteligencia Artificial



P r e s e n t a

José Alberto López López

Asesor:

Dr. Antonio Marín Hernández

*A mis padres, Blanca López
Velasquez y José Leobardo López
Rodríguez*

Contenido

1	Introducción	1
1.1	Antecedentes	1
1.2	Planteamiento general del problema	3
1.3	Justificación	4
1.4	Hipótesis y objetivos	4
1.4.1	Hipótesis	5
1.4.2	Objetivo general	5
1.4.3	Objetivos particulares	5
1.5	Alcances y limitaciones	6
1.6	Aportaciones	6
2	Trabajos relacionados	8
3	Marco teórico	25
3.1	Definiciones y consideraciones del problema	25
3.2	Especificaciones del problema a tratar	30
3.2.1	Espacio de trabajo	30
3.2.2	Características de los objetos	31
3.2.3	Características de la vecindad	32
3.2.4	Formas de sujeción	34
3.2.5	Restricciones de las formas de sujeción	34
3.2.6	Acciones	35
3.2.7	Especificaciones adicionales	35
3.3	Ánalisis de complejidad del problema	36
3.4	Casos triviales y casos imposibles	38
4	Metodología propuesta	41
4.1	Funciones y procedimientos propuestos	42
4.1.1	Funciones de ponderación	43
4.1.2	Costo de tomar los objetos	45
4.2	Algoritmo principal	49

5 Resultados	53
6 Conclusiones	65
Referencias	67

Capítulo 1

Introducción

En este capítulo se presentarán de forma general las características principales del trabajo realizado, así como las motivaciones y antecedentes históricos que influyeron en su realización.

1.1 Antecedentes

El desarrollo tecnológico ha permitido la creación de máquinas y robots, asimismo ha provisto a algunos de estos con las habilidades para manipular objetos de formas diversas. Con el paso del tiempo estas habilidades se han ido refinando, llegando incluso a igualar o superar a las habilidades humanas en tareas concretas [1][2].

La elaboración y programación de robots para la tarea de manipular objetos comenzó abordando problemas bien delimitados en entornos controlados. En un inicio la manipulación se realizaba por medio de teleoperación, por ejemplo, para manipular material radioactivo. Posteriormente, conforme se desarrollaron robots industriales, las tareas de manipulación comenzaron a aplicarse en líneas de producción, de entre las cuales se pueden mencionar el mover objetos pesados de un lugar a otro, el ensamblado de partes o acciones de tipo *tomar-dejar* [3][4][5].

En la actualidad, la investigación, desarrollo y producción de máquinas y robots capaces de llevar a cabo tareas de manipulación de objetos a un nivel cada vez más alto está en constante auge. Tareas que van desde resolver un cubo Rubik real, auxiliar en una cirugía, hasta construir una casa ya están siendo realizadas por robots; y otras como organizar objetos en estantes o anaqueles, tomar objetos de un refrigerador o reorganizar objetos desacomodados en una superficie están en continuo desarrollo [6][7][8][9].

Se estima que esta tendencia no hará más que incrementarse y diversificarse en los próximos años y no es difícil imaginar que, en un futuro, las máquinas sean

capaces de superar a los humanos, no solo en habilidades que involucran únicamente procesos mentales, sino también en aquellas que requieren de la interacción con el medioambiente.

Gran parte de esta interacción con el medioambiente se lleva a cabo mediante la manipulación de objetos. Una máquina podría sencillamente superar las habilidades de un humano lavando los platos sucios, acomodando los objetos de una habitación desordenada, armando o desarmando cualquier pieza mecánica o simplemente siendo más rápida buscando y tomando un martillo en una caja de herramientas.

Para dar tales habilidades a un robot es necesaria la conjunción de varias subdisciplinas de las ciencias de la computación y la inteligencia artificial (IA), tales como la visión por computadora, el aprendizaje automático, la implementación de algoritmos de planeación, entre otras.

Afortunadamente, la tecnología es cada vez menos limitante en la cuestión de conseguir manipulaciones a nivel humano o superior. Grandes avances se han logrado en las áreas de hardware y control para dar a las máquinas la suficiente capacidad de movilidad y de precisión en sus movimientos para realizar una gran cantidad de labores. Asimismo, con el auge de la IA, se han desarrollado algoritmos cada vez más precisos en campos como la visión por computadora, los cuales son capaces de detectar objetos en escenas del mundo real [10], así como algoritmos de planificación que permiten ejecutar tareas de manipulación de manera rápida y efectiva [11].

Sin embargo, comparado con la investigación hecha en los campos mencionados, las cuales están más enfocadas en mejorar las habilidades del robot o máquina manipuladora, las investigaciones relacionadas con mejorar las características de su entorno y más específicamente, de los objetos a manipular, de forma que el manipulador se pueda desenvolver de mejor manera, son menos abundantes.

Existen pocos estudios acerca de cuál debe ser la disposición adecuada de los objetos, tanto individual como colectivamente, para que un manipulador pueda interactuar con ellos eficientemente, teniendo en cuenta por supuesto, la finalidad de tal interacción, así como las restricciones que esta finalidad y el medioambiente pudieran imponer.

Con el paso del tiempo, los robots y máquinas serán capaces de manipular objetos con mayor precisión y rapidez. Así mismo, serán capaces de manejar un número y diversidad de objetos cada vez mayor. Sin embargo, al crecer el número y diversidad de objetos con los que interactúe el manipulador, una optimización de las condiciones iniciales de estos, siempre que sea posible, puede ayudar a eficientar los procesos de manipulación.

Por más eficiente que sea un robot manipulando objetos individuales, el optimizar la disposición inicial de los objetos con los que va a interactuar, siempre que sea posible, dará mejores resultados que el caso donde no se hiciera, por lo que vale la pena poner más atención en este tema. Una propuesta inicial de cómo lograr esto es la que se desarrolla en el presente trabajo.

1.2 Planteamiento general del problema

En el presente trabajo se plantea una metodología relacionada con la interacción robot-medioambiente, particularmente en el establecimiento de un arreglo inicial eficiente para la posterior manipulación de objetos, con una mínima ejecución de acciones o movimientos, en promedio, para acceder a cada uno de ellos.

En la vida cotidiana existen varias situaciones en las que se pudiera aplicar una metodología como la que se propone. Estas situaciones tienen que ver con tomar un objeto de entre un grupo de elementos, dispuestos en un espacio delimitado, los cuales generalmente obstruyen su acceso. Por ejemplo, cuando se intenta sacar un objeto de un baúl que contiene muchas cosas, cuando se quiere sacar la caja de leche que está al fondo del refrigerador o cuando se quiere sacar un medicamento que está dentro de una caja de medicinas. En este tipo de problemas, el sujeto tiene que elegir cuidadosamente qué objetos retirar para tener un libre acceso al elemento deseado, tratando al mismo tiempo de no desordenar demasiado la configuración de objetos inicial, ni dañar o tirar algún objeto en el proceso.

Para este tipo de situaciones, en las que se tiene un arreglo o acomodo de objetos en un espacio delimitado y se quiere acceder a alguno de estos, se puede idear un método para realizar dicha acción de forma eficiente con respecto a algún criterio, como puede ser el número de obstáculos removidos o el tiempo empleado para acceder al objeto deseado. De manera que, al utilizar dicho método, en el mejor de los casos, este encontraría la solución óptima para tomar el objeto de interés con respecto al criterio establecido y para el arreglo de objetos particular que se le presenta. No obstante, en este punto, surge una cuestión interesante: ¿y si los objetos hubiesen estado inicialmente arreglados de otra forma, podría existir una mejor solución?

Se cree que en los problemas que involucran acciones prensiles, la respuesta a esta pregunta está fuertemente relacionada con las restricciones de sujeción que impone la configuración inicial de objetos, además por supuesto de la metodología empleada para tomar el objeto deseado. Es por ello, que el interés del presente trabajo es idear un método para el acomodo de objetos, de forma que el acceso a estos se pueda hacer de forma eficiente, en términos del número de objetos que hay que retirar antes de poder acceder al objeto de interés. Para ello se toman en cuenta las restricciones de sujeción debidas a las características físicas de los objetos y a su configuración en el espacio.

Al ser un problema para el que actualmente no hay muchas referencias en la literatura, el planteamiento del problema a resolver se realizará de una forma simplificada, como una primera aproximación de las condiciones y variables que se pueden encontrar en una situación del mundo real. Los detalles técnicos de este planteamiento se presentan en el Capítulo 3.

1.3 Justificación

Como se mencionó anteriormente, la manipulación de objetos es un campo en el que la automatización está y seguramente estará ganando cada vez más territorio mediante la refinación de habilidades que los robots y máquinas ya poseen, así como la incorporación de nuevas.

En lo que respecta a la sujeción de objetos que se encuentran inmersos en un espacio o volumen, junto a otros objetos que dificultan su sujeción, existen varias investigaciones que abordan diversos aspectos así como variantes de este tipo de problemas. Particularmente es en el proceso de elección de obstáculos a retirar y en qué orden, dado un arreglo inicial y arbitrario de objetos, donde se enfocan algunas de las investigaciones actuales, de las cuales se hablará en el siguiente capítulo. Sin embargo, la presente propuesta se centra en encontrar un arreglo inicial óptimo de los objetos, el cual permita acceder a estos realizando un menor número de acciones que el caso de tener un arreglo inicial arbitrario. Por lo que, la búsqueda de la solución no se hace dentro de un arreglo ya establecido, sino que es el arreglo inicial mismo el que se trata de encontrar, de manera que este sea el más eficiente posible para la tarea dada. El estado del arte en este aspecto, como se verá más adelante, es bastante reducido en la actualidad, por lo cual se cree que vale la pena investigar más sobre este tema.

Se estima que al encontrar un método que resuelva de forma satisfactoria el problema planteado, será muy factible probarlo en el mundo real. Esto debido a la gran cantidad de modelos de brazos robóticos y dispositivos similares actualmente disponibles, así como de simuladores, los cuales permiten visualizar de forma aproximada cómo el algoritmo sería implementado en condiciones reales. Además, debido a que la interacción con objetos es una actividad muy presente en el día a día de las personas, de seguir desarrollando la metodología propuesta para que pueda ser aplicada en situaciones más complejas, los casos de aplicación en el mundo real aumentarían a medida que se pueda admitir un mayor número de geometrías de objetos y de sus configuraciones en el espacio. Se espera que el método ideado pueda ser aplicado en el mundo real de forma satisfactoria en una gran cantidad de escenarios, además de que pueda ser adaptado de una forma muy sencilla a los diferentes modelos de dispositivos elaborados para la manipulación de objetos.

1.4 Hipótesis y objetivos

A continuación se enuncian la hipótesis y los objetivos planteados para la elaboración del presente trabajo.

1.4.1 Hipótesis

El desarrollo de este trabajo parte de la siguiente hipótesis:

- Existe una metodología para el acomodo de objetos en un espacio delimitado que minimiza el número promedio de acciones necesarias para acceder (tomar) a cualquiera de ellos.

1.4.2 Objetivo general

De acuerdo a la hipótesis planteada, el objetivo principal de este trabajo para comprobarla es el siguiente:

- Diseñar un algoritmo para el arreglo de objetos en un espacio delimitado, con la finalidad de que un manipulador sea capaz de acceder a los objetos realizando un número mínimo de acciones o movimientos.

Las variables independientes que se consideran para dicho algoritmo son: el tamaño del espacio donde se colocarán los objetos, así como las diferentes cantidades de objetos de determinadas clases a colocar en dicho espacio. La variable dependiente, con la que se evalúa la salida del algoritmo, está definida en términos de la cantidad de objetos-obstáculo que hay que retirar antes de acceder a cada objeto, a partir del arreglo inicial encontrado por el algoritmo. Los detalles de esta métrica se pueden consultar en la Sección 4.1.2.

1.4.3 Objetivos particulares

Para llevar a cabo el objetivo general de este trabajo se establecieron los siguientes objetivos particulares:

1. Analizar la literatura relacionada para comparar los diferentes métodos utilizados en problemas similares.
2. Seleccionar y adaptar las metodologías útiles para el problema planteado.
3. Definir las métricas de eficiencia para los algoritmos a implementar.
4. Proponer un procedimiento adecuado para encontrar arreglos eficientes de objetos.
5. Evaluar la metodología propuesta.

1.5 Alcances y limitaciones

El método propuesto para el acomodo de objetos está constituido por cuatro componentes principales:

- La definición de las herramientas teóricas que serán utilizadas.
- Una función de evaluación que pondera los acomodos de objetos de acuerdo a las características del problema.
- Un algoritmo de recocido simulado que utiliza a la función de evaluación para generar los arreglos.
- Una función que calcula el costo real de tomar los objetos de un arreglo.

La teoría desarrollada para la resolución del problema planteado consiste en un conjunto de definiciones matemáticas, las cuales establecen los límites sobre qué tipo de problemas pueden ser abordados con la metodología propuesta y cuáles no. Dichos límites representan una simplificación del caso real.

La simplificación considerada consiste en reducir el número de variables del problema, hasta que este pueda ser tratado de forma sencilla, pero sin que se pierdan sus características esenciales. Lo cual se traduce en este caso, en utilizar cantidades y variedades reducidas de objetos, además de que estos posean geometrías simples.

Los límites impuestos al problema hacen que este, por su naturaleza discreta, pueda ser tratado como un problema de combinatoria.

Existen gran cantidad de metodologías para resolver este tipo de problemas, en este caso se optó por utilizar un algoritmo de recocido simulado, ya que es una de las técnicas que ha demostrado tener muy buenos resultados en problemas donde el espacio de búsqueda de combinaciones crece muy rápido cuando el tamaño del conjunto sobre el que se hace la búsqueda también lo hace. Dicho algoritmo utiliza la función de evaluación antes mencionada y su funcionamiento está basado principalmente en intercambios aleatorios de pares de elementos en un arreglo, los cuales realiza hasta encontrar (en teoría) el acomodo adecuado para las necesidades del problema.

1.6 Aportaciones

En este trabajo se propone un conjunto de herramientas teóricas para el acomodo de objetos de geometría simple en un espacio discreto. Dichas herramientas fueron establecidas con la intención de que metodologías generales para la evaluación y comparación de acomodos, de acuerdo a los fines planteados, se pudieran elaborar de forma sencilla.

La función de evaluación de arreglos de objetos que se propone es una de las componentes de mayor importancia del método, con la cual se define en gran medida si este tendrá éxito o no. Consiste de una heurística simple, la cual ayuda a determinar qué tan fácil es tomar un objeto del arreglo en función de su vecindad.

La función le asigna una puntuación a cada elemento del arreglo y a partir de estas puntuaciones individuales se obtiene una puntuación general de un acomodo. Las puntuaciones se asignan valorando en qué grado dicha vecindad restringe o permite la sujeción del elemento en cuestión.

Esta función de evaluación es utilizada por un algoritmo de recocido simulado para evaluar la eficiencia del arreglo encontrado cada vez que se realiza un cambio de estado, al cual se llega mediante un intercambio de posición de dos elementos del arreglo elegidos de forma aleatoria.

Finalmente, para evaluar los resultados del algoritmo se utiliza una función de costo, inspirada en el trabajo de M. Stilman et al. [12], la cual, mediante una búsqueda exhaustiva para todos los objetos en un acomodo determinado, encuentra cuál es la forma de acceder a estos que implique el menor costo posible. De esta manera se puede saber con certeza qué tan bueno es el resultado al que llegó el algoritmo, al poder comparar su costo asociado con el de otros acomodos.

En el siguiente capítulo se hará una revisión de los trabajos relacionados con esta investigación. En el Capítulo 3 se presentarán las bases teóricas desarrolladas para abordar el problema propuesto. Además, se definirán los límites de los problemas que se abordarán y se calculará la complejidad de su espacio de búsqueda. Posteriormente, en el Capítulo 4, se presentarán los procedimientos y funciones principales de la metodología propuesta, así como el funcionamiento algoritmo de recocido simulado que se elaboró. Luego, en el Capítulo 5, se dan a conocer los resultados obtenidos por la metodología propuesta, comparándolos, en los casos donde fue posible, con los resultados obtenidos al hacer una búsqueda exhaustiva; y en los casos donde no, con los mejores resultados obtenidos al realizar una búsqueda en una muestra del espacio total de configuraciones. Finalmente, en el Capítulo 6, se darán las conclusiones correspondientes tras finalizar el presente proyecto, así como las proyecciones para el trabajo futuro del mismo.

Capítulo 2

Trabajos relacionados

A continuación se hará una revisión de algunos trabajos relevantes para la presente investigación. Como se detallará, existen propuestas que abordan problemas similares al que se plantea en este trabajo; sin embargo, estos se abordan de maneras distintas a la que se propone en esta tesis. De entre ellos se pueden mencionar los siguientes:

- El problema de acomodar objetos en un contenedor, problema de carga del contenedor o *bin-packing problem*, por su denominación en inglés [13][14].
- El problema de tomar objetos de un contenedor o *bin picking problem*, por su denominación en inglés [15].
- Los NAMO (*Navigation Among Movable Obstacles*, por sus siglas en inglés) [16].
- Los MAMO (*Manipulation Among Movable Obstacles*, por sus siglas en inglés) [12].
- El problema de reordenamiento o *rearrangement problem*, por su denominación en inglés.

El problema de carga del contenedor, referido en adelante como problema del contenedor, consiste en colocar un conjunto de objetos en uno o más contenedores, de manera que el volumen total ocupado por dichos objetos se minimice. Para acercar el problema más a la realidad, se pueden agregar algunas restricciones adicionales, por ejemplo, se puede tomar en cuenta el peso de los objetos, su orientación, su separación, la prioridad de envío, etc. [17].

Es relativamente común encontrarse con este problema, sobre todo en ámbitos relacionados con el comercio o la industria, por lo cual se han propuesto diversos enfoques para su solución.

En [18] se propone un algoritmo para administrar los espacios libres en el problema del contenedor, el cual integra un algoritmo genético de clave aleatoria sesgada de múltiples poblaciones. Su proceso genera una lista de espacios máximos en el contenedor cada vez que un nuevo elemento es colocado, tal y como se muestra en la Figura 2.1. Posteriormente, se intenta colocar uno o varios elementos nuevos en los espacios vacíos. Los autores utilizan dos variantes de este enfoque: una en la cual los espacios vacíos tienen soporte completo por debajo y otra donde no lo tienen. El espacio en el cual se colocará un nuevo elemento se selecciona mediante una heurística basada en arreglos rectangulares de cajas del mismo tipo, llamados *capas*. Cada vez que una capa es colocada en el contenedor se generan nuevos espacios máximos. Los resultados muestran una mejora respecto a trabajos previos en la literatura.

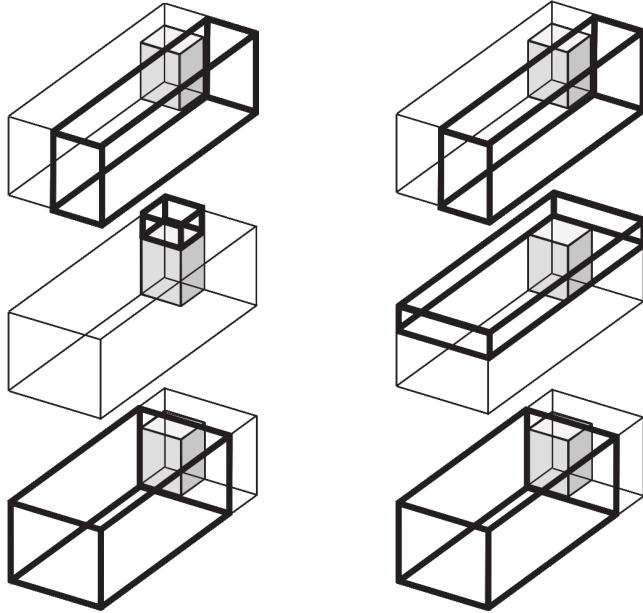


Figura 2.1: Espacios máximos generados: con soporte (izquierda) y sin soporte (derecha) por debajo (imágenes tomadas de [18]).

En [19] se presenta un algoritmo genético para resolver el problema de un solo contenedor, aplicando varias de las restricciones mencionadas en [17]; algunas de las cuales están relacionadas con aspectos como: la orientación de los paquetes, la estabilidad de paquetes que no son colocados directamente en el suelo o el no colocar paquetes encima de otros bajo circunstancias específicas. Para este propósito, se utiliza una estructura de capas para acomodar los objetos, la cual consiste en agregar paredes internas al contenedor, de forma que los espacios generados por estas paredes sean llenados de forma codiciosa. Los resultados obtenidos en este artículo muestran ser mejores que los de otros métodos con los que se comparan.

De forma similar, en [20] se presenta un algoritmo para la solución del problema

del contenedor. En este se establecen tres restricciones: la orientación de las cajas, la estabilidad de la carga y el volumen del contenedor. Dicho algoritmo está basado en el paradigma de búsqueda aleatoria adaptativa codiciosa y está evaluado en términos de las restricciones mencionadas y la comparación con otros nueve algoritmos. La heurística que utilizan se basa en el concepto de espacio vacío: una región con forma de paralelepípedo en la cual ninguna caja ha sido colocada.

El algoritmo utilizado considera todas las orientaciones y posiciones posibles de una caja y selecciona el arreglo cuyo uso del volumen en el contenedor sea menor. Los autores mencionan que su enfoque produce soluciones que superan las de otros en términos de la máxima ocupación del volumen disponible y estabilidad de la carga.

En [21] se muestra una solución al problema del contenedor aplicado en la impresión de modelos 3D, como los que se muestran en la Figura 2.2. Para ello se realiza una partición de un objeto 3D que se desea imprimir, de manera que se puedan acomodar las piezas en un contenedor de espacio mínimo. Con esto se busca economizar espacio y tiempo de producción.

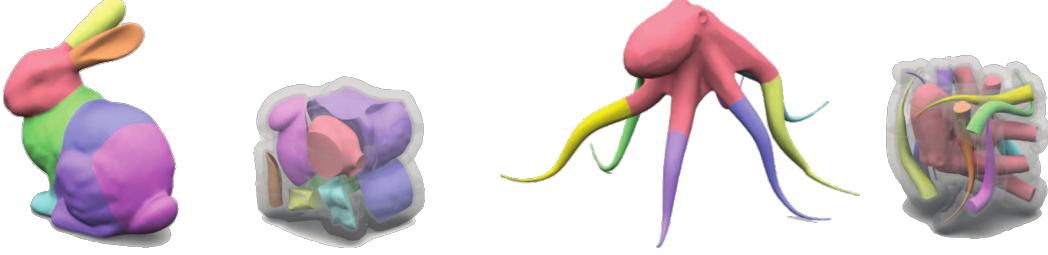


Figura 2.2: Ejemplos de segmentado y acomodo de figuras 3D (imágenes tomadas de [21]).

Una de las características en común del problema del contenedor con respecto al planteado en este trabajo, es que no se tiene certeza del arreglo de objetos al que se va a llegar. Por un lado, mientras que el interés principal del problema del contenedor es asegurar la máxima ocupación del volumen disponible, para de esa forma poder colocar el mayor número de objetos posible en él; en el problema propuesto se da por hecho que todos los objetos disponibles caben en el espacio utilizable, por lo que no hay restricciones sobre la ocupación de este.

El problema de tomar objetos de un contenedor o *bin-picking problem*, se refiere al problema para el cual un robot o manipulador debe tomar un objeto de un contenedor de forma automática. El problema en su forma general, tiene como característica principal que los objetos están colocados de forma aleatoria en un contenedor de dimensiones conocidas.

Este problema hasta la fecha no tiene solución. Esto debido principalmente a cuestiones de oclusión en los objetos [15], donde, por causa de obstrucciones entre objetos o ruido en los datos, no todas las características de los objetos son visibles al

mismo tiempo por las cámaras u otros sistemas de visión, dificultando la estimación de la posición y orientación de los objetos.

El problema de tomar objetos de un contenedor representa un gran reto para la robótica, debido a que en él se involucran diversos subproblemas que requieren un alto dominio de diferentes técnicas de la robótica y la inteligencia artificial. Entre estas técnicas se pueden mencionar la adquisición de datos [22], la localización y estimación de la posición y orientación de un objeto [23] o la prevención de colisiones [24][25]. Sin embargo, debido a que se trata de un tema importante para la industria, se han realizado varios esfuerzos por resolverlo; algunos de los cuales hacen una reducción de su complejidad al tratar solo instancias específicas.

En [25] se utiliza un robot industrial para resolver instancias particulares del problema de tomar objetos de un contenedor. El robot en este trabajo está equipado con un sistema de visión 3D y un efecto final, que puede ser una pinza estándar o una pinza de vacío, dependiendo del tipo de objeto que se vaya a manipular. El sistema de visión se utiliza para calcular la posición de un objeto, así como para detectar de forma general si este se puede sujetar. Después de esto, se hace el plan de una trayectoria libre de colisiones hacia el objeto y este es tomado. Finalmente, el objeto es insertado en una estación de proceso de forma muy precisa y con una orientación particular. Se realizaron experimentos con tuercas como objetos de prueba, tal como se muestra en la Figura 2.3. Como se demuestra, el robot es capaz de tomar todas las tuercas del contenedor.



Figura 2.3: Configuración experimental para la toma de tuercas de un contenedor con un brazo robótico (imagen tomada de [25]).

En [26] se presenta una solución a la combinación de dos problemas: el problema de tomar objetos de un contenedor y el problema del contenedor. El problema consiste en tomar objetos en desorden de un contenedor A y acomodarlos en otro contenedor B , tal como se muestra en la Figura 2.4, intentando maximizar el número de objetos en este último. Para esto se utiliza un brazo robótico con una pinza de vacío que

utiliza únicamente imágenes de profundidad para sensar su ambiente.

La pinza del brazo robótico, además de usarse para trasladar objetos de un lugar a otro, también es utilizada para empujarlos cuando se encuentran dentro del contenedor B . Esto con el objetivo de hacer pequeñas correcciones a sus posiciones, haciendo al arreglo más compacto. Asimismo, se utiliza una función para reconfigurar los objetos dentro del contenedor A , al derribarlos para poder tomarlos de una forma apropiada.

Debido a que todos los objetos manipulables tienen forma de prisma rectangular, la configuración del acomodo que se busca para el contenedor B es a modo de malla. Finalmente, se realizan diversas pruebas con un brazo real, en las que se muestra la dependencia de las funciones para derribar y empujar objetos sobre el arreglo final en B .



Figura 2.4: Configuración de objetos: antes (izquierda) y después (derecha) de la intervención del brazo robótico (imágenes tomadas de [26]).

En [27] se presenta un sistema basado en visión para tratar el problema de tomar objetos de un contenedor. Este sistema es capaz de lidiar con subproblemas relacionados con la caracterización de materiales, problemas de ambiente (polvo, grasa, etc.), estimación de la posición y orientación de los objetos, entre otros.

En este trabajo se presenta un algoritmo de correspondencia de forma; el cual es usado para detectar los objetos de manera confiable. Para la sujeción de objetos se utiliza un brazo robótico real, que toma los objetos del contenedor y realiza un refinamiento de la posición y orientación del objeto mientras este se encuentra en la pinza. También se utiliza una cámara *multiflash*, con la cual se detectan los bordes de los objetos con una mayor facilidad.

Se realizaron pruebas de detección con objetos reales, en las cuales el sistema de visión es capaz de reconocer varios objetos de formas complejas, los cuales se encuentran desordenados en el contenedor, tal y como se muestra en los diferentes ejemplos de la Figura 2.5. El sistema es capaz de detectar y estimar de manera precisa la pose de objetos con reflejos especulares (como se muestra en las Figuras 2.5a a 2.5d); así como objetos sin una textura definida (Figura 2.5e); o bien objetos pintados con una textura engañosa (Figura 2.5f). En los experimentos que realizan los autores se logra un porcentaje de eficiencia en la toma de objetos del 94%.

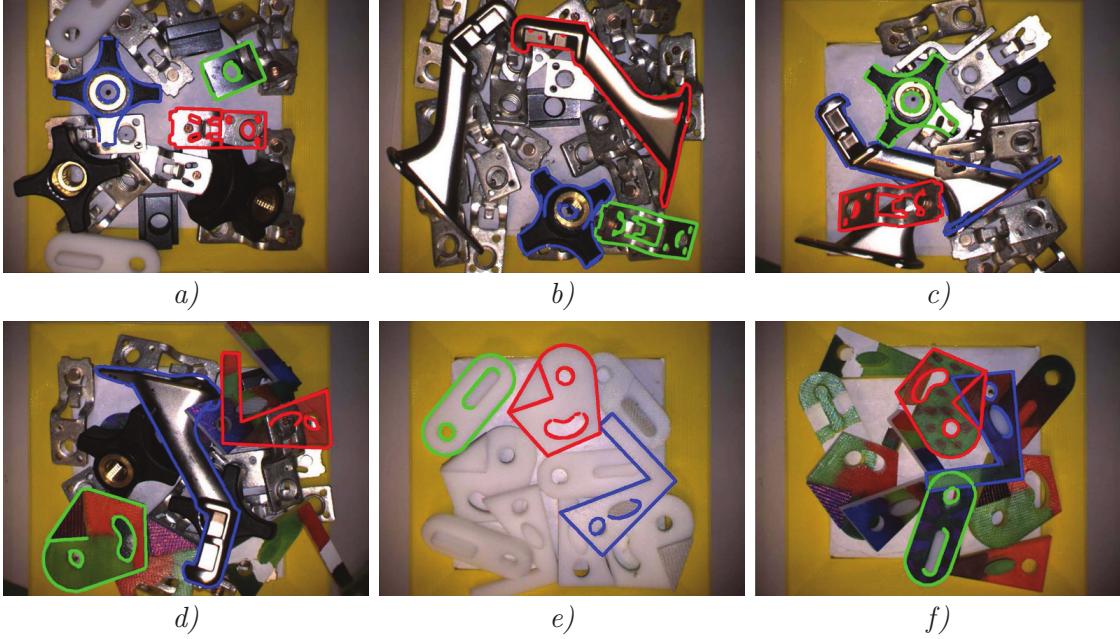


Figura 2.5: Detección de objetos desordenados en un contenedor (imágenes tomadas de [27]).

En [28] se propone un método que modela el problema de tomar objetos de un contenedor con un Proceso de Decisión de Markov Parcialmente Observable de tiempo discreto. Dicho método toma como entrada nubes de puntos de los objetos, obtenidas por una cámara de profundidad, como se muestra en la Figura 2.6c; y como salida entrega una posición y orientación del efecto final para remover un objeto que se encuentra inmerso en un conjunto de objetos desordenados. También se hace uso de un supervisor algorítmico, el cual realiza un cómputo previo de un conjunto de agarrres robustos para cada objeto, utilizando la información disponible acerca de estos. A partir de esto, se generan demostraciones sintéticas de agarre para el robot.

Los autores consideran una acción como el proceso completo de mover la pinza hacia el objeto, cerrarlo para sujetarlo y levantarla. Además, utilizan una simulación dinámica de varios cuerpos para determinar el siguiente estado de los objetos después de tomar uno, así como para determinar si un objeto puede ser levantado o no.

Se realizan experimentos en un robot real utilizando diferentes niveles de ruido en los datos de entrenamiento. Para esto, se utiliza un conjunto de 50 objetos de diferentes formas y tamaños, mostrados en la Figura 2.6b, los cuales tienen que ser trasladados desde un contenedor a otro, tal como se muestra en la Figura 2.6a. El robot debe agarrar objetos del contenedor señalado con borde verde y dejarlos en el contenedor con el borde azul. Sus resultados muestran una tasa aceptable de éxito del 94% en los intentos de mover un objeto de un contenedor a otro.



Figura 2.6: Robot y objetos utilizados en el experimento (imágenes tomadas de [28]).

El problema de tomar objetos de un contenedor es de gran interés para el presente trabajo, debido a que existen dos formas de aplicar la solución propuesta en este tipo de problemas. Por un lado, se facilitaría la tarea de tomar un objeto del contenedor, ya que, mediante el algoritmo propuesto, se podrían preordenar los objetos en el contenedor, en un modo que maximice las posibilidades de sujeción de los objetos. Además, si el problema presenta la tarea adicional de colocar los objetos en otro contenedor, tal como sucede en [26]; entonces, se podría volver a aplicar este preordenamiento, de tal forma que la manera de arreglar los objetos en el nuevo contenedor facilite la sujeción posterior de estos.



Figura 2.7: Ejemplo de problema NAMO. El robot debe manipular los objetos que le impiden el paso con el objetivo de trasladarse hacia un lugar específico (imagen tomada de [16]).

En los problemas NAMO uno o varios robots tienen permitido reconfigurar el ambiente donde se encuentran, al mover los obstáculos presentes en él. Esto con el objetivo de crear un camino por el cual puedan desplazarse sin mayor dificultad hacia una determinada posición final [16]. En la Figura 2.7, se muestra un ejemplo de un problema NAMO encontrado comúnmente en la literatura.

En [29] se presenta un algoritmo para resolver problemas de planeación de movimientos con un robot humanoide. Algunos de estos problemas consisten en situaciones en las que el robot debe realizar tareas de tipo NAMO y/o de reordenamiento de objetos. Entre estas tareas se puede destacar una en la que, en el entorno navegable por el robot, se encuentra una barrera de pilares manipulables, como se muestra en la Figura 2.8, los cuales obstruyen los lugares a los que debe llegar, por lo cual, el robot debe idear un plan para moverlos.

Para cada uno de los problemas se realizaron 50 simulaciones estableciendo un límite de tiempo de 300 segundos. En sus resultados muestran que, incluir información geométrica en la heurística de su algoritmo es crucial para resolver problemas de manipulación, los cuales involucran restricciones geométricas. Adicionalmente, los autores proponen aplicar su método a tareas de manipulación que involucren variables continuas, así como incluir acciones para apilar objetos encima de otros.

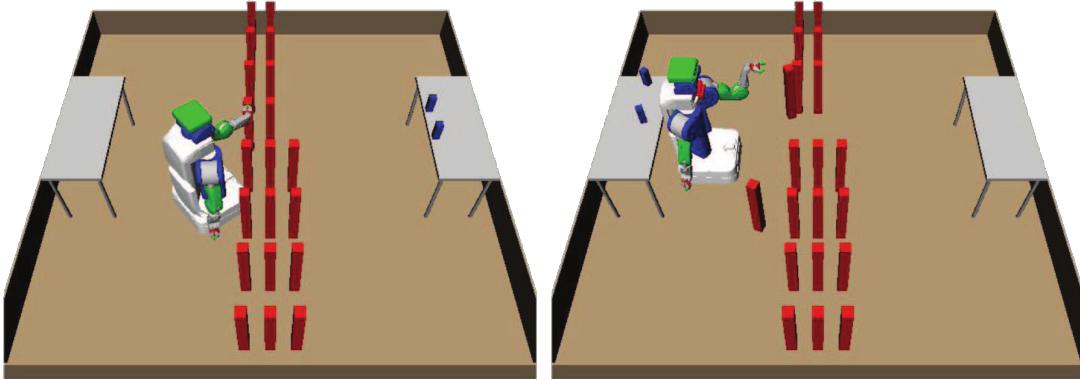


Figura 2.8: Ejemplo de una de las tareas que debe llevar a cabo el robot (imagen tomada de [29]).

En [30] se plantea resolver algunos problemas que involucran planificación de tareas y movimientos. Para ello se establecen secuencias de movimientos de bajo nivel en un robot (por ejemplo, trasladarse a determinado punto mediante una trayectoria libre de colisiones), con las cuales se puede lograr un objetivo de alto nivel (por ejemplo, tomar un objeto). Esto se logra integrando razonamiento geométrico de bajo nivel con razonamiento simbólico de alto nivel. Los autores utilizan un robot humanoide, el cual, junto con una configuración inicial de objetos móviles y obstáculos, debe llegar a una configuración final.

En el trabajo se presenta una nueva forma de representar los estados del mundo

mediante un vector de longitud fija. Los resultados experimentales demuestran que su método es capaz de aprender de una manera más eficiente que los enfoques estándar de aprendizaje por refuerzo, al utilizar menos información durante el entrenamiento.

De forma similar, en [31] se experimenta con la locomoción autónoma de un robot humanoide en un problema NAMO. El robot se encuentra en un ambiente que consiste en una habitación con objetos fijos y móviles. Estos objetos se consideran obstáculos que bloquean el camino para llegar a una posición final, hacia la cual el robot debe trasladarse. Los obstáculos fijos son representados mediante cajas contenadoras aproximadas, mientras que los móviles se representan mediante modelos de mallas 3D de sillas y mesas, debido a que requieren de un mayor nivel de detalle para poder ser manipulados adecuadamente.

Para lograr su objetivo, el robot percibe y construye un modelo de su ambiente, incluyendo los obstáculos en él. Con esto elabora una estrategia de navegación para alcanzar la posición objetivo, que consiste en una secuencia de operaciones de desplazamiento y manipulación de los obstáculos. Los autores consideran que en ambientes con mayor grado de incertidumbre, los robots deben ser capaces de reaccionar a eventos como colisiones no previstas del robot con los obstáculos, así como poder cambiar de decisión acerca de cuál objeto manipular. Por lo cual, estiman que una mejor integración de la replanificación y ejecución en problemas NAMO ayudaría a generar tales comportamientos autónomos complejos en robots humanoides.

En [32] se presenta un problema NAMO para un brazo robótico. Para este problema, al igual que para los problemas NAMO para robots humanoides, se desea que el robot, o más precisamente, su efecto final llegue a una posición determinada. En dicha posición se encuentra un objeto rodeado por varios obstáculos móviles, el cual se desea sujetar; como se muestra en la Figura 2.9. Los autores hacen varias simplificaciones al problema, una de las cuales es que, no se consideran obstáculos inmóviles, debido a que se considera que esto generaría efectos de aglomeración entre el robot y los objetos. La solución que se propone es desplazar los obstáculos con el chasis del brazo hasta llegar al objeto deseado mediante una replanificación en línea utilizando controles óptimos. Esto es, el robot elabora un plan, ejecuta una porción de él, observa el estado resultante y luego replanifica. Los resultados muestran que su enfoque conlleva un menor costo, así como un menor tiempo de ejecución que otros enfoques más ingenuos. Los autores también califican a los trabajos que planifican trayectorias como altamente conservativos y pesimistas.



Figura 2.9: Ejecución del planificador en línea: a) escena inicial; b) el contacto activa la replaneación; c) nuevo plan y d) objeto-meta sujetado (imágenes tomadas de [32]).

Los problemas NAMO tienen mucha similitud con el problema planteado en este trabajo, esto debido a que, en ambos se tiene como objetivo acceder a una cierta posición en un espacio, con obstáculos en él. Particularmente, en el presente caso de estudio, se desea acceder a un objeto rodeado por obstáculos más que a una zona de espacio vacío. Mientras que en los problemas NAMO se deben reacomodar los objetos del entorno para llegar a una zona específica, en la presente investigación se busca minimizar el número de acciones necesarias para realizar dicho reacomodo, con la finalidad de acceder a un objeto obstaculizado más que desbloquear el camino para llegar a una zona específica.

Otra característica de los problemas NAMO es que se enfocan principalmente en los temas de la planificación de movimientos del robot, así como en la planeación de la manipulación de los objetos móviles. En el trabajo aquí desarrollado se da por hecho que el manipulador puede tomar cualquier objeto, para enfocarse, por otro

lado, en que el entorno imponga las restricciones mínimas para tomarlos. Esto es, se brinda más atención a asuntos como minimizar las restricciones de sujeción de los objetos en su acomodo inicial, ya que a partir de ello es más probable que las planificaciones de algoritmos NAMO tengan éxito.

Los problemas MAMO son una generalización de los problemas NAMO. En este dominio, además de las tareas de navegación, el robot requiere remover obstáculos con el objetivo de manipular un objeto, llevándolo desde su posición inicial a una posición final deseada [12].

En [33] se presenta un planificador incremental de tareas y movimientos basado en muestreos y grafos jerárquicos, el cual es utilizado por un brazo robótico para alcanzar objetos *casi-cilíndricos* (como botellas) inmersos en escenas desordenadas. Dichas escenas consisten en los objetos cilíndricos colocados en estantes. El planificador computa un plan para remover obstáculos y generar una trayectoria libre de colisiones para alcanzar el objeto deseado.

Su método consta de dos componentes principales: una planeación de tareas, donde se realiza una expansión incremental de un grafo con la secuencia de obstáculos a remover para alcanzar el objeto meta; y una planeación de movimientos, donde se utiliza otro grafo de estados de colisión, el cual contiene la información de objetos que colisionan con el robot cuando se quiere remover otro objeto. Mediante este último grafo se corrige y optimiza la secuencia de remociones de objetos-obstáculo para alcanzar el objeto meta. Esto es, se intenta minimizar el número de obstáculos a ser removidos, optimizando el orden en el que se retiran.

El método además es capaz de lidiar con occlusiones de objetos. Se demuestra la capacidad de su método para realizar desde tareas simples, hasta movimientos de manipulación complejos, tanto en ambientes reales como virtuales. Comparado con otros enfoques, su método reduce el tiempo de planeación de tareas y movimientos en un 24.6%.

En [34] se aborda el problema de la manipulación de objetos en ambientes desordenados mediante planeación de tareas y movimientos con dos brazos robóticos. La idea es que los brazos colaboren para retirar los obstáculos que bloquean un objeto particular hasta tomarlo, tal como se muestra en la Figura 2.10. También se busca un buen lugar donde dejar los obstáculos retirados, para lo cual se utiliza el razonamiento geométrico sobre la colocación de objetos (Figuras 2.10c a 2.10e). Los autores implementan su solución en problemas de tipo MAMO y NAMO. Aseguran que el principal reto en su investigación fue el proveer al robot la información geométrica de bajo nivel que sirva de guía al planificador de tareas. Su propuesta fue validada tanto en ambientes virtuales como reales, considerando varias clases de problemas de manipulación de objetos que se encuentran sobre una mesa. Los resultados muestran que su método es capaz de resolver dichas tareas de manipulación eficientemente, tanto en términos de tiempo de planeación (16s, en promedio) como en tasa de éxitos (98%, en promedio), mejorando respecto a otras aproximaciones contra las que

se comparan en prácticamente todos los escenarios.

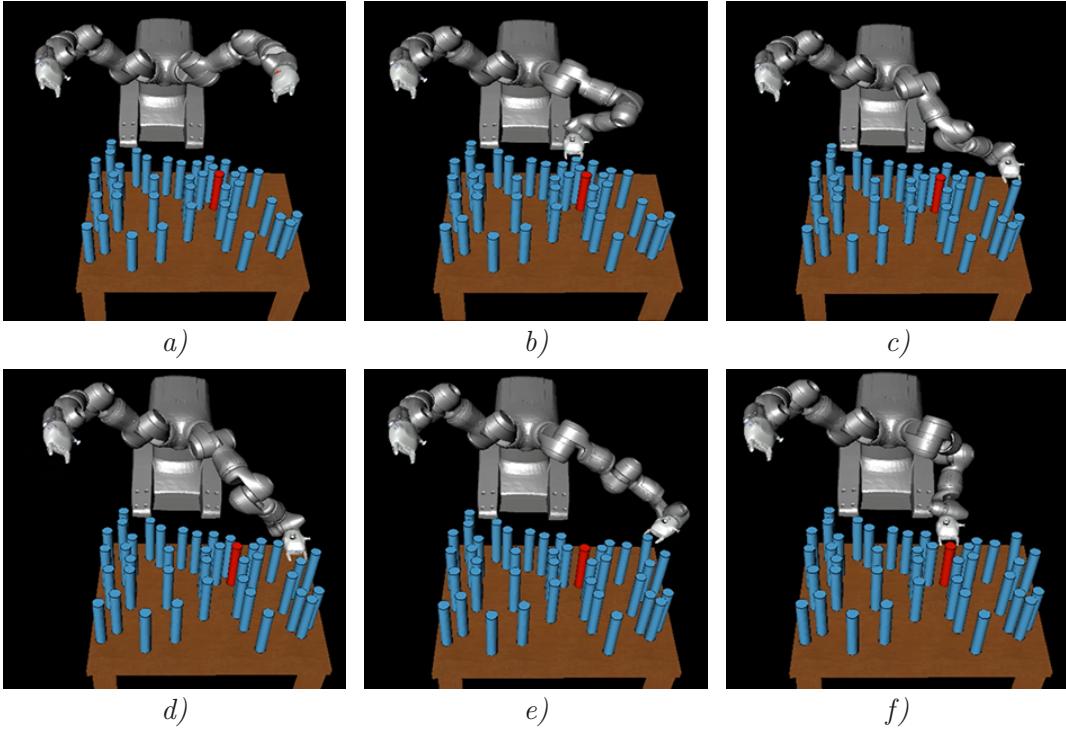


Figura 2.10: Ejecución de un simulador para resolver el problema de manipulación entre objetos desordenados. El robot tiene que sujetar el objeto rojo que se encuentra entre 50 objetos azules (imágenes tomadas de [34]).

En el trabajo desarrollado en [12] se presenta un algoritmo enfocado en la manipulación planificada, el cual genera planes de manipulación rápidos en un espacio de búsqueda no lineal de dimensión exponencial. Para ello se utiliza una búsqueda en profundidad sobre los obstáculos ordenados.

El plan comienza desde el objeto que se desea tomar (raíz) y a partir de este, se identifican los posibles obstáculos (ramas) sobre los cuales se hace el mismo proceso de forma recursiva. Se definen tres funciones: *PlanGrasp*, *PlanManipulation* y *PlanNavigation*. Si alguna de ellas fracasa, entonces el planificador finaliza la rama actual y hace *backtracking* hasta encontrar una solución o hasta que se agoten las opciones de búsqueda.

En la Figura 2.11 se muestran imágenes de la aplicación de este método en una simulación, donde un robot debe retirar los obstáculos que le impiden tomar un objeto particular. Los autores realizan pruebas en escenarios con varios objetos, tanto fijos, como móviles y semifijos. Sus resultados muestran que el 50 % del tiempo empleado por todo el proceso es consumido por la función *PlanManipulation*, debido a las restricciones de manipulación de los objetos, así como a que las geometrías de los

objetos en estas pruebas son complejas.

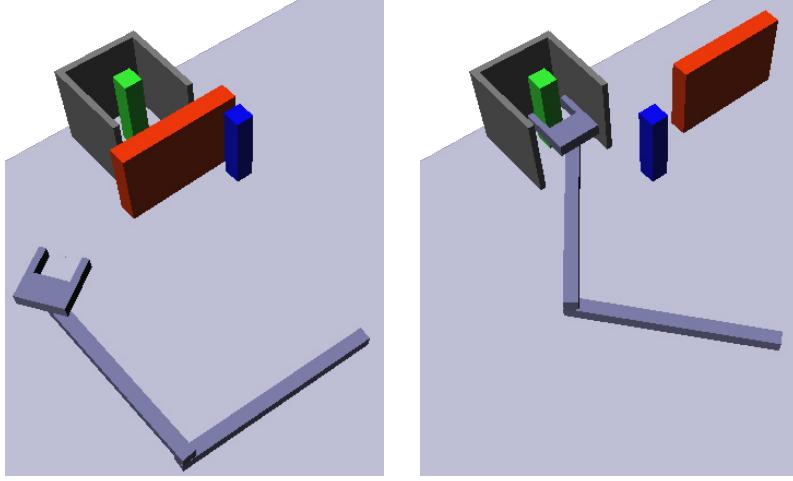


Figura 2.11: Simulación en la que un brazo robótico debe retirar los obstáculos que rodean a un objeto (prisma verde) para posteriormente tomarlo (imágenes modificadas tomadas de [12]).

Al igual que los problemas NAMO, los problemas MAMO se enfocan mayormente en la planeación de movimientos. Debido a que el objetivo del presente trabajo es encontrar una configuración de objetos que minimice el número de acciones para acceder a alguno de estos, se espera que, por razones muy similares a las explicadas para los problemas NAMO, el algoritmo aquí desarrollado se pueda complementar muy bien con los algoritmos de problemas MAMO.

En el problema de reordenamiento, un robot o manipulador debe moverse en un espacio con objetos en desorden, los cuales debe reordenar en una configuración específica. Dentro del espacio de trabajo puede haber o no obstáculos, móviles o inmóviles. Para lograr su objetivo, el robot puede mover los objetos de forma libre, esto es:

- a) De forma no prensil, al empujar los objetos.
- b) De forma prensil, al sujetarlos y trasladarlos de un lugar a otro.
- c) Una combinación de las anteriores.

Una de las características que distingue este tipo de problemas de los problemas MAMO, es que la manipulación y reubicación se hace sobre varios objetos y no solo sobre uno [35]. En la Figura 2.12 se muestra un problema peculiar de este tipo, donde un manipulador debe trasladar dos objetos a zonas determinadas. En este caso particular, el orden en que los objetos son trasladados es muy importante, ya

que uno obstruye la única trayectoria posible del otro.

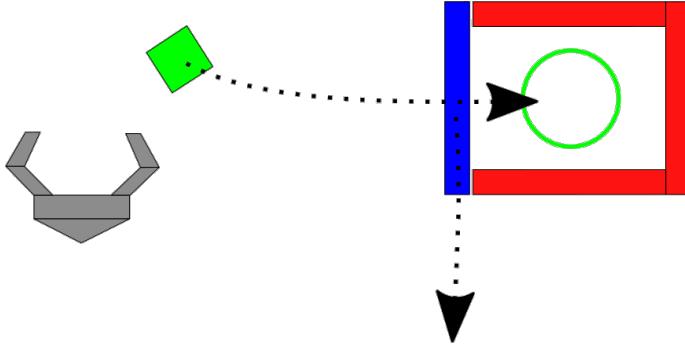


Figura 2.12: Problema donde el manipulador debe reconfigurar los objetos, moviéndolos a donde indican las flechas (imagen tomada de [35]).

En [36] se trata el reordenamiento de objetos mediante interacciones no prensiles. Los autores proponen un planificador para resolver el problema de la planeación del reordenamiento, que considera dos tipos de acciones: acciones centradas en los objetos y acciones centradas en el robot. Las acciones centradas en los objetos guían al planeador a realizar acciones específicas en objetos específicos; mientras que las acciones centradas en el robot mueven al robot sin la intención de manipular un objeto relevante, permitiendo así la interacción simultánea con varios objetos. Tales interacciones pueden ser el empujar los objetos que obstaculicen otro al cual se quiere llegar, o bien simplemente desplazar un objeto de un lugar a otro, al moverlo entre otros que funcionan como obstáculo. Se evalúa el planeador en el brazo de un robot humanoide y en un robot móvil de cuatro ruedas diseñado para empujar objetos. Sus resultados muestran una mejora respecto a otros métodos que solo utilizan un tipo de acción.

En [37] se aborda el problema de reacomodo de objetos de forma no prensil con un robot humanoide; para lo cual se combina un sistema de recompensas con otro de entrenamiento basado en redes Q profundas, los cuales únicamente utilizan información visual, a partir de la cual el robot planea estrategias de reacomodo. Uno de los objetivos de los autores es reducir el número de colisiones con los obstáculos, ya que estas llevan a resultados subóptimos. El sistema de aprendizaje profundo que utilizan consiste en una red convolucional, que recibe como entrada imágenes de los objetos (los cuales tienen forma de cubo) sobre una superficie plana, y da como salida una de cinco direcciones posibles, a lo largo de la cual se debe mover el manipulador para empujar los objetos. El sistema de recompensas premia al manipulador cuando este se acerca al objeto que debe ser desplazado y lo castiga cuando ocurre alguna colisión. Los resultados muestran una tasa de éxito de 85 % en las tareas asignadas.

De igual forma, en [38] se aborda el problema de reordenamiento de objetos de forma no prensil mediante un brazo robótico de siete grados de libertad, utilizando

modelos físicos simples. En sus configuraciones de objetos se pueden incluir tanto obstáculos móviles como inmóviles. Un ejemplo de una configuración del robot con los objetos se muestra en la Figura 2.13. Los modelos físicos que se utilizan reducen la dinámica *quasi*-estática a contactos rígidos, lo cual hace que el método empleado sea más rápido y reduce la complejidad del problema. Sin embargo, al emplear estos modelos de contacto simples y rápidos en las simulaciones, algunas veces estos no imitan lo suficiente la complejidad de los contactos físicos reales; perdiendo en tales ocasiones precisión a cambio de rapidez. Los resultados experimentales muestran que el método propuesto supera a otras propuestas por un factor de 5 en entornos simulados.



Figura 2.13: La meta en este problema de reordenamiento es mover el objetivo a la posición meta deseada (imagen modificada tomada de [38]).

En [39] se utiliza un modelador de física embebido, así como un planificador aleatorio para problemas con restricciones cinemáticas y dinámicas, con el objetivo de resolver el problema de reacomodo de objetos de forma no prensil. La principal acción para manipular los objetos es mediante empujes, y una característica destacable es que en el modelado se incluye la posibilidad de tirar los objetos. Su método está diseñado para que se pueda hacer contacto con cualquier parte del brazo y con varios objetos al mismo tiempo. Se define una *región meta* en la cual se debe colocar el objeto de interés, sin importar la posición final de los demás objetos. La comparación de su método con otras soluciones basadas en funciones básicas de movimiento muestra un incremento en la tasa de éxito, mientras que en el tiempo promedio de planificación se observa un incremento con respecto a los otros enfoques. Como trabajo futuro, los autores proyectan incluir más tipos de interacciones entre el manipulador y los objetos, tales como derribar y rodar.

En [40] se trata el problema de reordenamiento con dos brazos. En el problema que se aborda, los brazos deben coordinarse para manipular un conjunto de objetos desordenados en una mesa y reordenarlos en una configuración específica, tal y como se muestra en la Figura 2.14. Los objetos utilizados tienen forma de cubo y los brazos pueden tomarlos por la parte de arriba. Entre los objetivos principales de la propuesta se encuentra el minimizar el costo del transporte de los objetos cuando estos están siendo reacomodados, así como evitar colisiones con objetos no manipulados. Los autores determinan que los costos involucrados en la solución de su problema son menores al utilizar más de un manipulador. Como trabajo futuro proponen explorar casos más generales con k brazos.

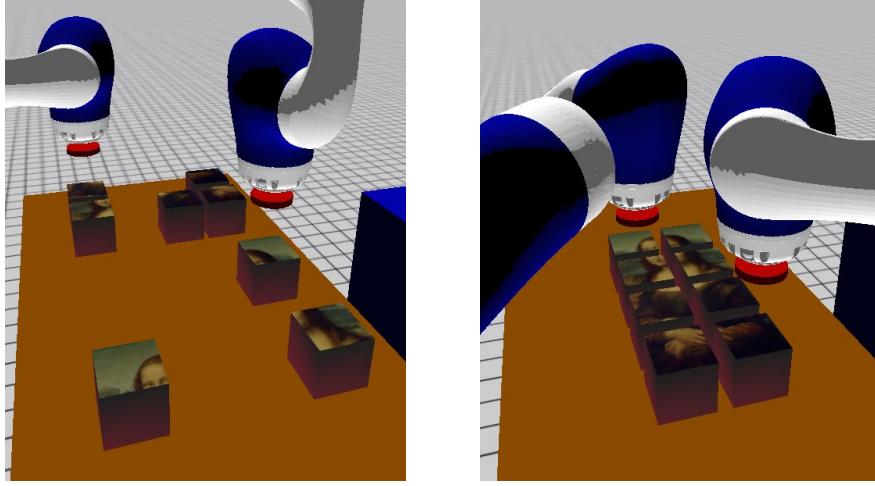


Figura 2.14: Ejemplo de reacomodo de objetos con dos brazos: configuraciones de objetos inicial (izquierda) y final (derecha) (imágenes tomadas de [40]).

De forma distinta, en [35] el reordenamiento de objetos se realiza mediante acciones no prensiles con cualquier parte de un brazo robótico. El robot debe reacomodar algunos objetos que se encuentran entre obstáculos en una superficie plana. Se presenta un algoritmo eficiente de planeación que está diseñado para trabajar con poca información acerca de las habilidades de manipulación del robot, por lo cual, sus creadores aseguran que dicho método es fácilmente adaptable a diferentes arquitecturas de manipuladores. Sin embargo, el hacer que el algoritmo sea agnóstico a las características del manipulador puede llevar a que exista una alta probabilidad de falla, debido a predicciones imprecisas. Los autores afirman que un reto mayor en este tipo de problemas es la necesidad de retirar obstáculos, lo cual requiere de un nivel más alto de lógica. Sus resultados muestran que su algoritmo supera por un factor mayor a dos a otros enfoques de la literatura en lo que al tiempo de planeación se refiere.

Una diferencia importante de este tipo de problemas con el planteado en este

trabajo es el hecho de que se conoce la configuración a la que se debe llegar (al contrario de problemas como el del contenedor). Sin embargo, uno de los trabajos en el tema del reordenamiento que tiene algunas similitudes con el presente es [41]. En él se trata de reordenar, mediante un manipulador prensil, un conjunto de objetos en una superficie plana, los cuales son tomados desde la parte de arriba por dicho manipulador, como se puede apreciar en la Figura 2.15. La similitud recae en que se busca minimizar el número de acciones *tomar-dejar*, ya que, debido a la presencia de obstáculos en las posiciones finales, en ocasiones hay que dejar los objetos en posiciones intermedias antes de colocarlos en su posición final.

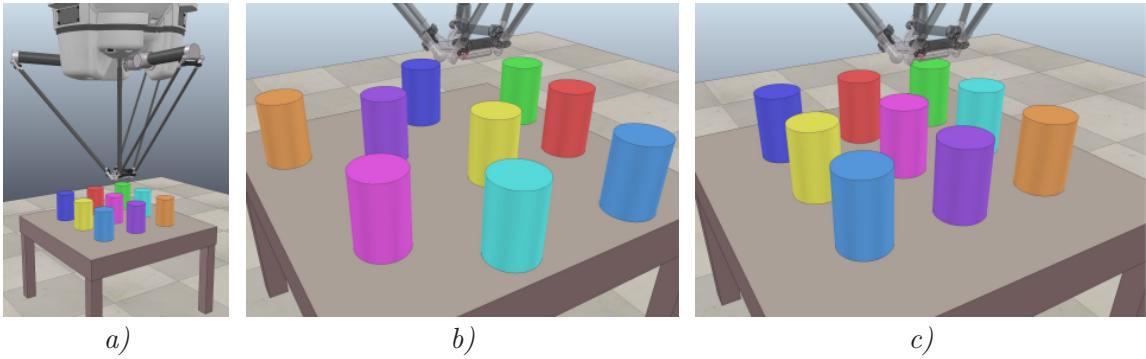


Figura 2.15: Vista de a) manipulador y los objetos, así como de las configuraciones b) inicial y c) final de los objetos (imágenes tomadas de [41]).

Como recordatorio, en el presente trabajo se busca crear un arreglo de objetos que minimice el número de acciones para tomar un objeto, tomando en cuenta sus restricciones de agarre. Donde cada acción consiste en retirar un obstáculo que impida, directa o indirectamente, la sujeción del objeto deseado.

Cabe destacar que ninguno de los trabajos mencionados aborda de manera precisa el problema propuesto, por lo cual no es posible una verdadera comparación entre las investigaciones mencionadas y la presente. Asimismo, hasta la fecha no se conoce una investigación que proponga una solución al problema aquí planteado.

Capítulo 3

Marco teórico

En este capítulo se definirán de manera formal los elementos que componen el problema propuesto, así como las herramientas útiles para resolverlo. De igual manera, se describirán las características de las instancias específicas que se abordarán y se hará un análisis de la complejidad de su espacio de búsqueda asociado.

3.1 Definiciones y consideraciones del problema

Antes de establecer las bases teóricas del problema que se desea abordar, se describirán el contexto y las condiciones iniciales que lo caracterizan. De esta manera, los elementos necesarios para poder definir el problema planteado son:

- Un espacio plano y limitado sobre el cual se pueden colocar objetos.
- Un conjunto* de objetos, los cuales tienen asociados individualmente un conjunto de atributos y una clase, la cual está definida a partir de los primeros.
- Una vecindad asociada a cada objeto colocado en el espacio.
- Un conjunto de formas de sujeción asociadas a cada clase de objeto, definidas en función de su geometría, las cuales pueden ser utilizables o no dependiendo del estado de la vecindad del objeto.
- Un conjunto de acciones, aplicables a cada objeto.

A continuación se definen de forma detallada cada uno de los componentes anteriores

* Debido a cuestiones prácticas, se utilizará la palabra *conjunto*, aunque técnicamente lo correcto sería *multiconjunto*, ya que en los objetos a acomodar están permitidas las repeticiones.

y posteriormente se hará una asignación de valores para tratar un problema en particular, esto con el objetivo de demostrar el funcionamiento del algoritmo propuesto.

Sea E un espacio discreto (malla) de $n \times m$ localidades o celdas como el que se muestra en la Figura 3.1. Cada una de las celdas de E será representada por e_{ij} , con $1 \leq i \leq n$ y $1 \leq j \leq m$.

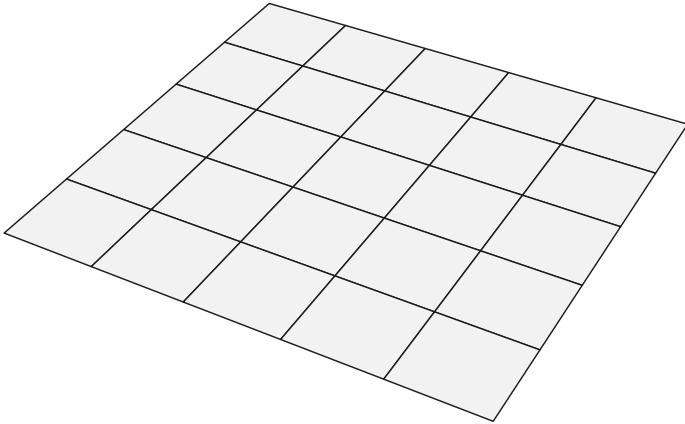


Figura 3.1: Espacio discretizado de dos dimensiones.

Se desea colocar en el espacio E un conjunto de objetos particulares. Sea:

$$O = \{o_1, o_2, \dots, o_N\} \quad (3.1)$$

el conjunto de N objetos disponibles para colocar en las celdas de E , donde o_k representa el k -ésimo objeto, con $1 \leq k \leq N$.

Sea:

$$A = \{a_1, a_2, \dots, a_{N_A}\} \quad (3.2)$$

el conjunto de atributos que puede tener un objeto y que son de utilidad para resolver el problema definido; por ejemplo: el tamaño, el color, la forma, etc. Siendo N_A el número máximo de atributos, y sea:

$$A_k = \{a_{k,1}, a_{k,2}, \dots, a_{k,N_A}\} \quad (3.3)$$

el conjunto de atributos asociado a un objeto o_k . De esta manera, mientras que a_1 representa el nombre de un atributo en particular, por ejemplo *color*; $a_{k,1}$ corresponde al valor o instancia de dicho atributo para el objeto o_k , por ejemplo, *verde*.

La condición necesaria y suficiente para que dos objetos se consideren diferentes es que difieran en al menos el valor de uno de sus atributos.

El conjunto general de atributos A se ha definido para fomentar que los objetos

compartan la mayor cantidad de atributos posible, así como que estos atributos sean lo más generales posible, para que de esta manera el problema pueda ser resuelto con la mínima cantidad de atributos necesarios, reduciendo de esta forma su complejidad.

Así, se motiva a encontrar la representación más simple de los objetos (e.g. mediante su caja contenedora) que satisfaga las necesidades de manipulación del problema. Por ejemplo, si para los fines de manipulación del problema es suficiente que un objeto particular con geometría compleja pueda ser tratado como un cubo u otra figura de geometría más simple, esto ayudará a simplificar la búsqueda de la configuración óptima de objetos, así como la definición de las formas en que estos pueden ser sujetados.

Sea:

$$C = \{C_1, C_2, \dots, C_{N_C}\} \quad (3.4)$$

el conjunto de clases a las que pertenecen los objetos de O , donde N_C es el número máximo de clases.

Cada clase C_K , con $1 \leq K \leq N_C$, representa un conjunto de atributos A_k distinto. De esta manera, si los conjuntos de atributos A_1 , A_2 y A_3 correspondientes a los objetos o_1 , o_2 y o_3 , son iguales, se considera que todos ellos pertenecen a la misma clase, aunque en la realidad puedan diferir en algún atributo no considerado en su conjunto de atributos. En particular, el trabajar con un número elevado de clases de objetos no es, por el momento, de interés en el presente trabajo, ni el objetivo de la definición de clases previa, sino lo contrario: crear un conjunto de clases de atributos que sean lo más similares entre sí para reducir la complejidad del problema.

La manera en que se definen estas clases ayudará a simplificar el problema, ya que permite, cuando sea posible, que objetos con determinadas características en común (sin que necesariamente sean iguales en la realidad) sean identificados como de la misma clase y manejados de forma similar o igual; por ejemplo, en lo que a su sujeción se refiere.

Sea:

$$V_{ij} = \{v_1, v_2, \dots, v_{N_V}\} \quad (3.5)$$

el estado de la vecindad con respecto a la celda e_{ij} ; donde v_{k_V} es el estado individual de una celda vecina, con $1 \leq k_V \leq N_V$, donde N_V es el número de celdas vecinas (tamaño de la vecindad) a considerar.

El número de celdas vecinas N_V y su localización, dependerán del tipo de problema que se vaya a tratar. Según la definición del espacio discretizado E , algunas de las vecindades posibles se pueden conformar, por ejemplo, de 4, 8 o 24 elementos, como se muestra a continuación:

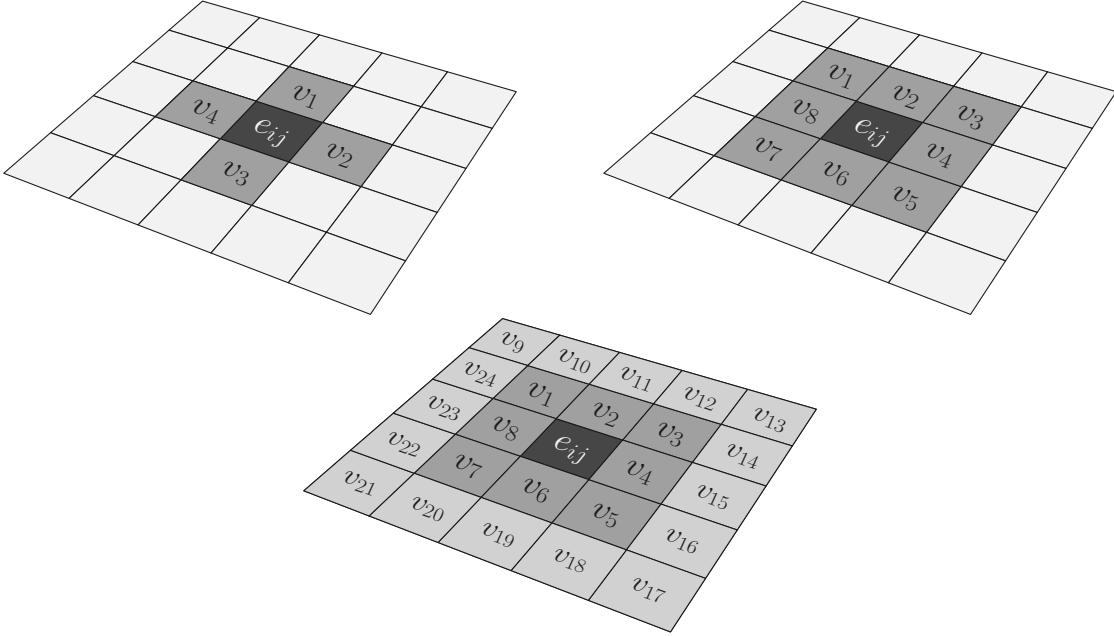


Figura 3.2: Ejemplos de vecindades de 4, 8 y 24 vecinos.

De esta manera, el estado de la vecindad V_{ij} consiste en el listado de los estados individuales de las celdas consideradas como vecinas a e_{ij} , donde cada uno de estos puede ser: contener un objeto de O o no contener ningún objeto (celda vacía), esto es: $v_k \rightarrow O \cup \{\square\}$.

Sea:

$$S = \{s_1, s_2, \dots, s_{N_S}\} \quad (3.6)$$

el conjunto de todas las formas de sujeción de todas las clases de objetos de O . Una forma de sujeción se define como un conjunto de puntos, curvas y/o superficies de contacto de un objeto particular, a partir de las cuales este se puede sujetar. De esta manera, una forma de sujeción s_{k_S} podría ser, por ejemplo, la que especifique el conjunto de puntos de contacto de un objeto particular, sobre los cuales se tendría que ejercer presión, succión o algún otro acto (dependiendo del manipulador utilizado) para sujetarlo.

La definición de las formas de sujeción depende directamente del tipo de manipulador que se vaya a utilizar, de las características del objeto que se vaya a manipular, entre otros factores. Cabe señalar que, en general, las formas de sujeción aplicables a un objeto determinado, no tienen que serlo para otro objeto, debido principalmente a las variaciones en la geometría de estos, pero, en general, puede ser conveniente que, además de que la cantidad de formas de sujeción para cada objeto sean mínimas, estas sean iguales o lo más similares posible (como las utilizadas en el presente trabajo), con el objetivo de reducir la complejidad del problema. Es por ello que, de forma

similar a la definición de los conjuntos de atributos, se define un conjunto general S de formas de sujeción, ya que tanto el preferir geometrías simples para representar las distintas clases de objetos, como el definir formas de sujeción comunes a estas representaciones, simplificará el problema en la medida de lo posible.

Sea:

$$S_K = \{s_{K,1}, s_{K,2}, \dots, s_{K,N_S}\} \quad (3.7)$$

el conjunto de formas de sujeción asociadas a los objetos de la clase C_K , donde cada s_{K,k_S} puede ser restringida en función de la vecindad asociada al objeto en cuestión.

Para el problema particular que se tratará en este trabajo, detallado en la siguiente sección, se han definido formas de sujeción de manera bastante sencilla y muy general, sin embargo, estas formas pueden ser tan complejas como se requiera para una tarea en particular.

Finalmente, sea:

$$W = \{w_1, w_2, \dots, w_{N_W}\} \quad (3.8)$$

el conjunto de acciones que se pueden aplicar a un objeto particular, ya sea dentro del espacio E o fuera de este, tal que modifican el estado de E . Estas acciones se definen según las características del problema a tratar; las cuales pueden ser, por ejemplo: quitar un objeto de E ; añadir un objeto a E , cambiar su posición dentro de E , cambiar su orientación, etc.

Sea t_{ij} la variable que cuantifica el número de acciones necesarias para tomar el objeto en e_{ij} . Esto es, $t_{ij} - 1$ es el número de objetos-obstáculo que impiden tomar el objeto de interés en e_{ij} . Por lo tanto, t_{ij} también se define como el costo asociado para tomar un objeto.

La forma de calcular t_{ij} está en función de la vecindad del objeto en e_{ij} y, recursivamente, de la vecindad de sus vecinos hasta llegar a una condición de paro; lo cual se presenta en la Sección 4.1.2.

Asimismo, se define el costo global T de un acomodo de objetos como la suma de los costos individuales t_{ij} :

$$T = \sum_{i,j} t_{ij} \quad (3.9)$$

El objetivo principal en este trabajo es encontrar una configuración o acomodo de todos los objetos de O en el espacio E ; de forma que el número de acciones t o cambios de estado promedio para acceder y finalmente tomar a cada uno de los objetos sea mínimo. En esta etapa del proyecto, se considera que las acciones que se pueden realizar para acceder a un objeto únicamente consisten en quitar un objeto de E . La acción correspondiente al cambio de posición dentro de E de un objeto, considerado

como obstáculo, no se considera, pero se puede añadir en una posterior extensión del proyecto.

Adicionalmente, tomando en cuenta la posición de un objeto o_k en la malla, su clase C_K y el estado de su vecindad V_{ij} , se considera que, el conjunto de formas de sujeción asociadas S_K puede variar, llegando en ocasiones a restringirse de forma parcial o total las formas en que este se puede sujetar.

3.2 Especificaciones del problema a tratar

Para acotar el problema, en esta sección se definirán instancias específicas de los conjuntos enunciados en la sección anterior, así como de las variables asociadas a ellos. Los detalles del problema particular a tratar se dan a continuación.

3.2.1 Espacio de trabajo

Para evaluar el rendimiento del algoritmo propuesto se utilizaron diferentes tamaños de malla. La Figura 3.3 muestra un ejemplo de la configuración de ejes utilizada para ubicar cada celda e_{ij} en una malla de 5×5 . En las figuras y expresiones matemáticas utilizadas en este documento, la configuración de ejes se mantiene, o bien, es la que más se aproxima a la que muestra la Figura 3.3.

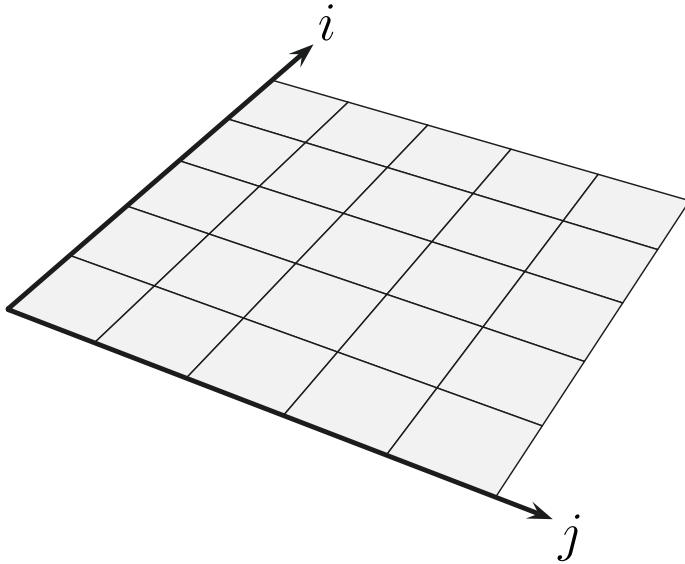


Figura 3.3: Configuración de los ejes para la ubicación de las celdas mediante la notación e_{ij} .

Es importante hacer notar que, el tamaño o discretización de las celdas de E solo permite colocar en cualquiera de estas un y solo un objeto, de cualquier clase.

3.2.2 Características de los objetos

El conjunto de objetos O consiste, en esta fase inicial del proyecto, de objetos pertenecientes a una de dos clases posibles, siendo la clase C_1 un cubo y la clase C_2 un prisma rectangular, ambos objetos de base idéntica correspondiente a una cara del cubo (Figura 3.4). La altura del prisma es igual al doble de la longitud de un lado de un cubo.

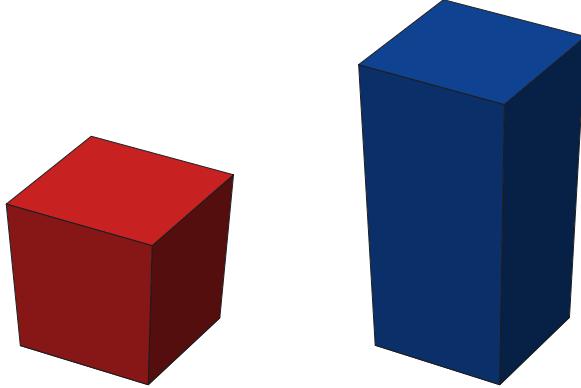


Figura 3.4: Ejemplos de los objetos considerados para las clases C_1 (izquierda) y C_2 (derecha).

De los atributos posibles de estos objetos, el utilizado para diferenciar las clases de objetos es su altura h , el cual es suficiente para abordar el problema.

Así:

$$A_k = \{h_k\} \quad (3.10)$$

son los conjuntos de atributos asociados a cada objeto o_k , donde h_k solo puede tener dos posibles valores, por lo cual las clases quedan definidas de la siguiente forma:

$$\begin{aligned} C_1 &= \{h_1\} \\ C_2 &= \{h_2\} = \{2h_1\} \end{aligned} \quad (3.11)$$

Para evaluar la propuesta en las diferentes configuraciones propuestas del espacio E , se utilizaron diferentes cantidades de objetos, siendo N la cantidad total de objetos en dichos espacios, la cual será la suma de los objetos de ambas clases propuestas, donde N_1 es la cantidad de objetos de la clase C_1 y N_2 los objetos de la clase C_2 .

En los casos tratados solo se permite colocar un objeto por cada celda de la malla y la cantidad N debe cumplir la desigualdad $\lceil \frac{nm}{2} \rceil < N \leq nm$. Esto porque una cantidad de objetos igual o menor a $\lceil \frac{nm}{2} \rceil$ se considera un caso con solución trivial; ya que siempre se dispone de espacio suficiente para acomodarlos en una

configuración en la cual se puedan tomar con una sola acción. Además, el número de objetos disponibles no debe exceder el número total de celdas.

Dicho lo anterior, los casos de interés para validar la propuesta de este trabajo, consisten en todas las configuraciones de objetos N_1 y N_2 que no correspondan a casos triviales ni imposibles (los cuales serán detallados en la Sección 3.4). Es por ello que se obtuvieron las condiciones para que un caso no sea trivial ni imposible, resultando las siguientes:

$$\begin{aligned} \left\lceil \frac{nm}{2} \right\rceil &< N \leq nm \\ N &= N_1 + N_2 \\ 0 \leq N_1 &\leq N - \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor \\ 0 \leq N_2 &\leq N - \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor \end{aligned} \tag{3.12}$$

Un análisis más profundo de estas condiciones se dará en la Sección 3.4.

3.2.3 Características de la vecindad

Para definir el estado de la vecindad V_{ij} de una celda e_{ij} , se considerarán como celdas vecinas de una celda particular e_{ij} , aquellas celdas que se encuentran inmediatamente arriba (dirección positiva de i), abajo, a la derecha (dirección positiva de j) y a la izquierda de esta, como se ilustra en la Figura 3.5.

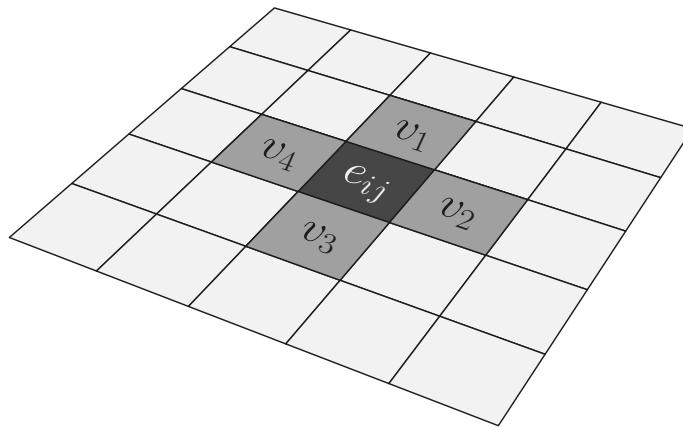


Figura 3.5: Vecindad utilizada para abordar el problema planteado.

El orden en el que se encuentran los estados de las celdas vecinas en el conjunto V_{ij} se obtiene recorriendo las posiciones mencionadas en sentido horario comenzando con

la celda de arriba, es decir:

$$V_{ij} = \{v_1, v_2, v_3, v_4\} \quad (3.13)$$

donde:

v_1 corresponde al estado de la celda $e_{i+1,j}$.

v_2 corresponde al estado de la celda $e_{i,j+1}$.

v_3 corresponde al estado de la celda $e_{i-1,j}$.

v_4 corresponde al estado de la celda $e_{i,j-1}$.

Las características de la vecindad de los elementos en los bordes y esquinas de la malla no difiere de las de los demás elementos. Esto debido a que en todos los arreglos tratados se considera un “espacio extendido” de trabajo, el cual consiste en suponer que el espacio E se encuentra rodeado por celdas vacías, tal y como se muestra en la Figura 3.6. En estas celdas hipotéticas no se puede colocar ningún objeto, ya que únicamente cumplen con la función de actuar como vecinas de los elementos en los bordes y esquinas de la malla, de manera que las mismas reglas de vecindad de los elementos centrales también puedan ser aplicadas en ellos.

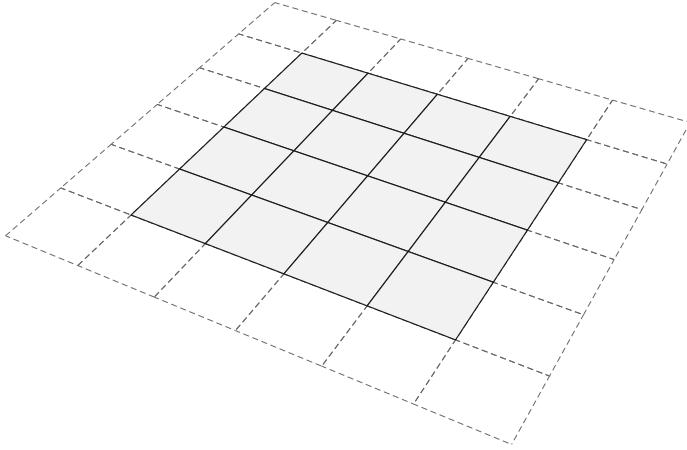


Figura 3.6: Ejemplo de espacio extendido en una malla de 4×4 .

En todos los casos del presente trabajo se supondrá un espacio extendido como el de la Figura 3.6, pero esta característica puede ser modificada según las necesidades del problema que se aborde. Por ejemplo, si se desea realizar el acomodo de objetos dentro de una caja, puede ser más adecuado cambiar las celdas vacías del espacio extendido por elementos fijos de determinada altura, a manera de pared.

3.2.4 Formas de sujeción

Solo se consideran dos formas de agarre, las cuales son comunes a ambas clases de objetos: agarre horizontal H y agarre vertical V ; esto es:

$$S = S_1 = S_2 = \{H, V\} \quad (3.14)$$

Estas formas de agarre se establecieron considerando un manipulador que solo puede tomar los objetos por la parte superior. Un esquema de los puntos que son utilizados, en teoría, para sujetar los objetos se muestra en la Figura 3.7.

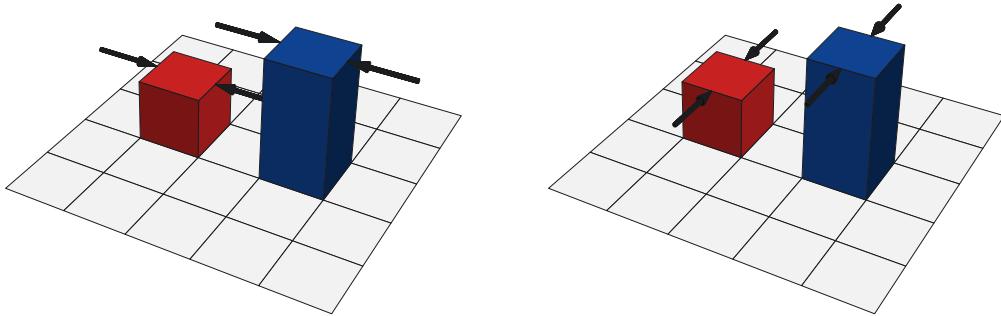


Figura 3.7: Formas de sujeción horizontal (izquierda) y vertical (derecha).

3.2.5 Restricciones de las formas de sujeción

Las formas de sujeción aplicables a cada clase de objeto pueden ser restringidas en función de su vecindad V_{ij} .

Un objeto o_k puede ser sujetado de las formas H y V si sus vecinos horizontales y verticales, respectivamente, tienen una menor altura que el objeto o_k . La definición matemática de la oración anterior se presenta en la Ecuación 4.3 de la Sección 4.1.

Algunos ejemplos de cómo se restringen las formas de sujeción de un objeto, en función de su vecindad, se muestran en la Figura 3.8.

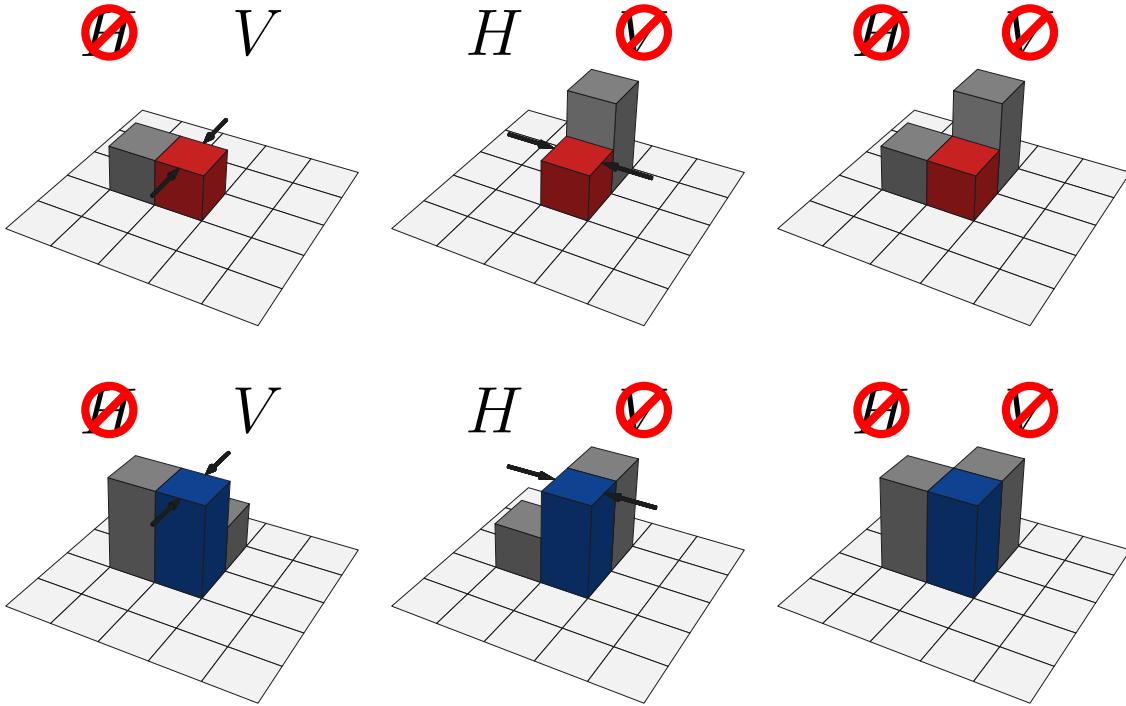


Figura 3.8: Ejemplos de cómo se pueden restringir las formas de sujeción en función de la vecindad.

3.2.6 Acciones

Por el momento solo se consideran dos tipos de acciones: poner un objeto en E (w_1) y quitarlo de E (w_2), es decir:

$$W = \{w_1, w_2\} \quad (3.15)$$

Se considera que estas acciones son suficientes para la primera aproximación propuesta para el problema planteado.

3.2.7 Especificaciones adicionales

Los objetos no se pueden colocar uno encima de otro, por lo cual solo se pueden generar acomodos como el que se muestra en la Figura 3.9.

A diferencia del problema del contenedor, no se toma en cuenta el caso en que haya que dejar objetos sin acomodar por falta de espacio. Ya que, si bien este sería un caso de estudio interesante, en el que se podrían fusionar las técnicas del problema del contenedor con las propuestas, se cree que el añadir este tipo de restricciones disfrazaría la esencia del verdadero problema que se quiere resolver. Por lo cual se da

por hecho que los N objetos se pueden colocar sin problemas en E , es decir, ningún objeto debe quedar fuera de E en el arreglo final.

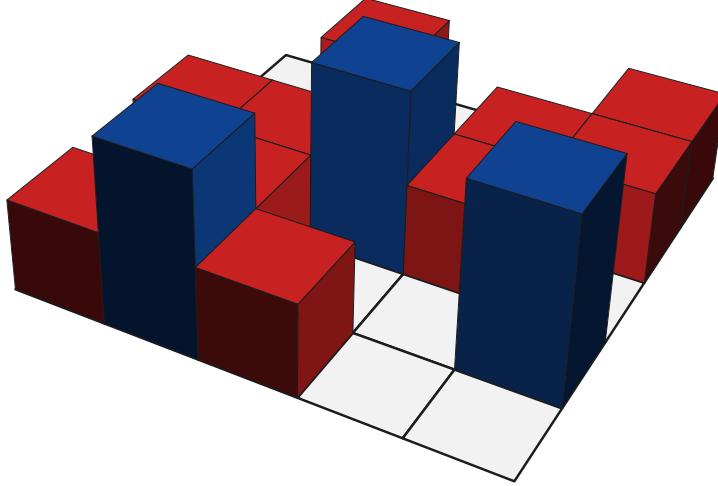


Figura 3.9: Ejemplo de una configuración de objetos en la malla.

3.3 Análisis de complejidad del problema

Debido a que el objetivo final del presente trabajo es encontrar una configuración específica de los objetos de O en E , la complejidad del problema estará entonces relacionada con el total de configuraciones de objetos que se pueden generar a partir de datos: un conjunto O de objetos y un espacio E en el cual colocarlos.

Si se considera el espacio de búsqueda sobre todas las configuraciones posibles de objetos en la malla, utilizando el conjunto O para la representación de los objetos y el símbolo \square para la representación de una celda vacía, se tiene el siguiente espacio de búsqueda:

$$\mathcal{C} = (O \cup \{\square\})^{nm} \quad (3.16)$$

donde se ha hecho explícita la unión con la representación de una celda vacía, al considerarla como un elemento más del conjunto, debido a su recurrente uso.

Al utilizar el conjunto O para la representación de los objetos, implícitamente se está considerando a cada uno de sus elementos como diferente a los demás, ya que, en general, sus respectivos atributos A_k pueden ser diferentes. Por lo cual el número de configuraciones posibles en dicho espacio viene dado por:

$$|\mathcal{C}| = |O \cup \{\square\}|^{nm} = (N + 1)^{nm} \quad (3.17)$$

Con la finalidad de reducir este espacio de búsqueda es que se ha establecido el sistema de clases definido anteriormente. Mediante esta herramienta es posible identificar atributos clave A_k de objetos no necesariamente iguales, los cuales permitan tratarlos como tal únicamente en lo que respecta a su sujeción. Por ejemplo, si al tener una esfera y un cubo en el espacio E se tiene la condición de que el manipulador solo es capaz de tomar estos objetos mediante una única posición y orientación, utilizando puntos de contacto semejantes; entonces, no tendría caso considerar estos objetos como diferentes, en lo que a su manipulación se refiere. Por lo cual, para cuestiones de sujeción, lo conveniente sería representarlos como objetos de la misma clase.

Al hacer esta consideración el espacio de búsqueda puede ser reducido, ya que:

$$|C| \leq |O| \quad (3.18)$$

por lo cual el nuevo espacio de búsqueda sería:

$$\mathcal{C}' = (C \cup \{\square\})^{nm} \quad (3.19)$$

Y de forma similar a la Ecuación 3.17, el número de configuraciones posibles quedaría como:

$$|\mathcal{C}'| = |C \cup \{\square\}|^{nm} = (N_C + 1)^{nm} \quad (3.20)$$

Sin embargo, el espacio de búsqueda de interés para el problema enunciado, es un subconjunto de \mathcal{C}' , para el cual se tiene un número de objetos N fijo, e igualmente las cantidades de objetos de cada clase N_1, N_2, \dots, N_{N_C} son también fijas.

Entonces, se define \mathcal{C}'' como el espacio de todas las configuraciones posibles de N objetos de N_C clases en un E específico, donde N_1, N_2, \dots, N_{N_C} son cantidades fijas. Para lo cual se define a $\mathcal{C}_{\text{ins}}''$ como cualquier instancia de \mathcal{C}'' y a $\mathcal{C}_{\text{ins}}''(C_K)$ como el número de veces que aparece la clase C_K en $\mathcal{C}_{\text{ins}}''$. Por lo tanto

$$\mathcal{C}'' = (C \cup \{\square\})^{nm} \mid \mathcal{C}_{\text{ins}}''(C_K) = N_K, \quad 1 \leq K \leq N_C \quad (3.21)$$

lo cual establece que debe haber exactamente N_1 objetos de la clase C_1 , N_2 de la clase C_2 , etc., en cualquier instancia de \mathcal{C}'' .

El número de configuraciones posibles para \mathcal{C}'' es entonces:

$$|\mathcal{C}''| = \frac{nm!}{N_1!N_2!\cdots N_{N_C}!(nm - N)!} \quad (3.22)$$

En la Tabla 3.1 se muestra el número de configuraciones posibles para algunas combinaciones de valores del tamaño de la malla y del número de objetos de cada clase.

Tabla 3.1: Número de permutaciones de arreglos posibles en función del número de elementos de cada clase presentes en la malla y del tamaño de la malla.

Tamaño malla	N_1	N_2	$nm - N$	$ \mathcal{C}'' $
3×3	3	3	3	1,680
3×3	1	5	3	504
3×3	4	5	0	126
4×4	5	5	6	2,018,016
4×4	1	9	6	80,080
4×4	8	8	0	12,870
5×5	8	8	9	2.62×10^{10}
5×5	10	10	5	0.98×10^{10}
5×5	12	13	0	5,200,300
6×6	12	12	12	3.38×10^{15}
6×6	15	15	6	0.30×10^{15}
6×6	18	18	0	9.07×10^9

Al analizar la Tabla 3.1 se puede observar que, entre más uniforme sea la distribución del número de objetos de cada clase y de celdas vacías ($nm - N$), el número de permutaciones posibles de una malla aumenta. Es decir, que cuando las cantidades de objetos de cada clase y de celdas vacías son lo más parecidas entre sí, es cuando el número de permutaciones es máximo, y disminuye conforme se va perdiendo esta uniformidad.

3.4 Casos triviales y casos imposibles

Existen cantidades especiales de objetos N , así como del número de objetos de cada clase N_1, N_2, \dots, N_{N_C} , para las cuales se pueden tener casos de acomodo triviales o imposibles. Las características de estos casos se definen a continuación.

Un caso trivial es un caso en el cual la cantidad N de objetos es lo suficientemente pequeña como para acomodar los objetos en un patrón simple, el cual permite que todos los objetos puedan ser sujetados con una sola acción. En este caso los objetos pueden ser siempre colocados mediante un mismo patrón, previamente establecido,

siempre que se cumpla la siguiente condición:

$$N \leq \left\lceil \frac{nm}{2} \right\rceil \quad (3.23)$$

Un patrón de acomodo para casos triviales se presenta en la Figura 3.10, dónde $N = \left\lceil \frac{nm}{2} \right\rceil$. En el presente trabajo no se abordan este tipo de casos.

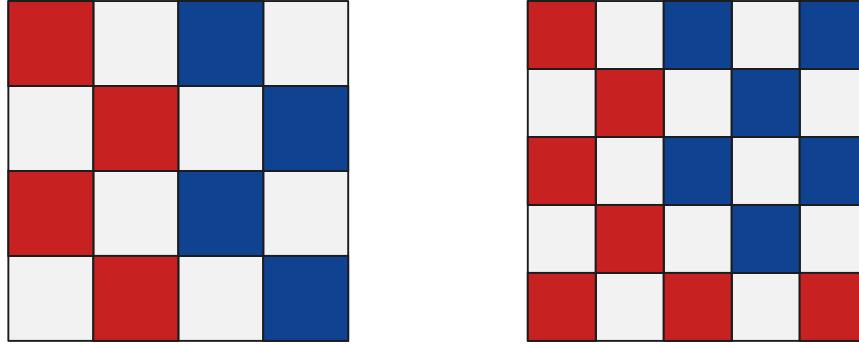


Figura 3.10: Ejemplos de casos triviales en mallas de 4×4 (izquierda) y 5×5 (derecha).

Por otro lado, también existen configuraciones específicas de objetos en la malla, las cuales implican que algunos objetos sean imposibles de tomar, dadas las reglas de sujeción establecidas. Estos casos se presentan cuando al menos cuatro objetos de la misma clase se agrupan en un patrón como el que se muestra en la subfigura izquierda de la Figura 3.11. En este caso ninguno de los cuatro objetos puede ser sujetado mediante las formas de sujeción definidas H y V . Estas agrupaciones de cuatro elementos de la misma clase se puede presentar una o varias veces en un acomodo de objetos, como se muestra en la subfigura derecha de la Figura 3.11.

Se ha denominado como *cuadro imposible* al arreglo de cuatro objetos de la misma clase descrito anteriormente, y como *caso imposible* a un arreglo de objetos en la malla que contenga al menos un cuadro imposible.

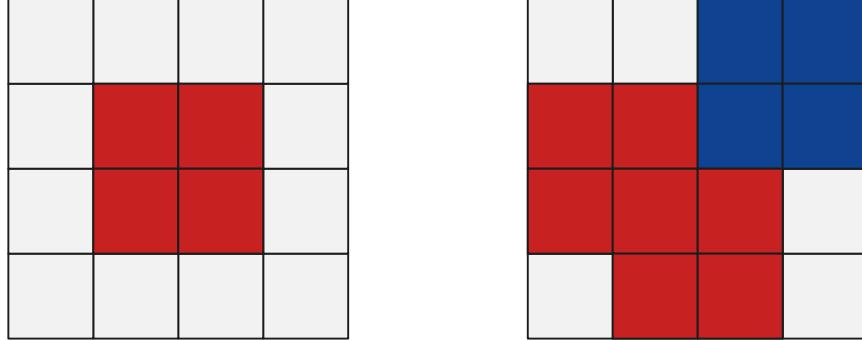


Figura 3.11: Ejemplos de casos imposibles.

Los casos imposibles como los de la Figura 3.11 se pueden evitar con un acomodo correcto de los objetos en la malla. Sin embargo existe una condición, la cual implica que siempre habrá cuadros imposibles en la malla, sin importar la forma en que se acomoden los objetos.

La condición para que exista al menos un cuadro imposible en un arreglo de objetos es:

$$\max_K N_K > nm - \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor \quad (3.24)$$

En el presente trabajo no se abordan este tipo de casos.

En el siguiente capítulo se presentarán los procedimientos y algoritmos desarrollados para abordar el problema propuesto. Al final del capítulo se presentará el algoritmo principal que resuelve el problema planteado, el cual hace uso de los procedimientos que le preceden. Posteriormente, en el Capítulo 5, se presentarán los resultados obtenidos al aplicar el algoritmo principal a diferentes tamaños de malla y diferentes combinaciones de cantidades de objetos en ellas. Finalmente, en el Capítulo 6, se abordarán las conclusiones del proyecto.

Capítulo 4

Metodología propuesta

En este capítulo se establecerá y detallará la propuesta para realizar un acomodo de objetos, con la condición de minimizar el número de acciones que conlleva el acceder a cada uno de ellos. Dicha propuesta se basa en un sistema de puntuaciones, asignadas a cada uno de los elementos en la malla, incluyendo celdas vacías. La puntuación que se asignará a cada elemento dependerá de la clase del elemento en cuestión así como de su vecindad.

La puntuación de un elemento, cuando se trate de un objeto, será máxima cuando este se pueda tomar mediante una sola acción y disminuye con la presencia de vecinos que impiden su agarre o sujeción. En el caso de las celdas vacías, su puntuación se incrementa con respecto al número de celdas vecinas que sean ocupadas por objetos, y disminuye en función del número de celdas vecinas vacías.

Con este sistema de puntuaciones, el cual es parte fundamental de las aportaciones de la propuesta aquí presentada, se busca que la puntuación global de la malla, esto es, la suma de las puntuaciones de los elementos individuales, sea máxima; lo cual es sinónimo de un fácil acceso y posterior sujeción de los objetos.

Los procesos para calcular las puntuaciones mencionadas son integrados en una función, la cual es a su vez utilizada por un algoritmo de recocido simulado, en el cual convergen las aportaciones de este trabajo y que se encarga de hacer los cambios necesarios a un acomodo inicial aleatorio, con la finalidad de que la puntuación global se maximice. De esta forma es como se obtendrá un acomodo de objetos optimizado para realizar el acceso y la sujeción de cada uno de los objetos en una malla, con un número de acciones mínimas en promedio, lo cual en la práctica también podría implicar un tiempo mínimo promedio para acceder a los objetos. Para ello se evalúan los acomodos obtenidos por el algoritmo al calcular el costo de tomar cada objeto de la malla mediante una función de costo.

La definición completa del algoritmo y de las funciones mencionadas se dará en las siguientes secciones.

4.1 Funciones y procedimientos propuestos

En esta sección se iniciará con la definición de los componentes más básicos de la metodología propuesta, posteriormente se definirán las funciones y procedimientos de más alto nivel, que hacen uso de dichos componentes y finalmente, en la Sección 4.2, se describirá el algoritmo principal de la propuesta, el cual es la definición de más alto nivel de esta.

Con la finalidad de simplificar la notación de algunos cálculos posteriores, se define C_0 como la clase de una celda vacía, la cual, al solo tener fines de representación, no tiene ningún atributo asociado. Asimismo se define h_{ij} como la altura del elemento en la celda e_{ij} , donde esta altura es igual a cero cuando se trata de una celda vacía.

Se define la función de estado C_{ij} como la función que retorna la clase del elemento posicionado en la celda e_{ij} .

$$C_{ij} \mapsto C \cup \{C_0\} \quad (4.1)$$

Además, se define la función de sujeción S_{ij} como la función que retorna el subconjunto de formas de sujeción permitidas de un objeto posicionado en la celda e_{ij} .

$$S_{ij} \mapsto S'_K, \quad S'_K \subseteq S_K \quad (4.2)$$

Las expresiones que describen las condiciones bajo las cuales los objetos pueden ser sujetados, correspondientes a la descripción dada en la Sección 3.2.5, se presentan a continuación:

$$\begin{aligned} h_{i+1,j} < h_{ij} \wedge h_{i-1,j} < h_{ij} &\Rightarrow H \in S_{ij} \\ h_{i,j+1} < h_{ij} \wedge h_{i,j-1} < h_{ij} &\Rightarrow V \in S_{ij} \end{aligned} \quad (4.3)$$

Por lo tanto, la condición para que un objeto en una celda e_{ij} sea sujetable es simplemente:

$$|S_{ij}| \geq 1 \quad (4.4)$$

Con fines de legibilidad en definiciones posteriores, se utilizará la función booleana $sujetable(e_{ij})$, la cual simplemente indica si para el elemento de la celda e_{ij} se cumple o no la condición de la Ecuación 4.4. De forma similar, la función $imposible(e_{ij})$ indica si el elemento en la celda e_{ij} pertenece a un cuadro imposible, como los que se describen en la Sección 3.4.

4.1.1 Funciones de ponderación

A continuación se presenta la función que asigna una puntuación (un número entero) a un par de elementos vecinos en E .

Algo a tener en cuenta con esta función es que, si bien por sí sola es una función que solo puntuá un par de elementos en la malla, está pensada para ser una subfunción del proceso de puntuación de más alto nivel de la Función 2, enfocado a puntuar un elemento particular de la malla en función de su vecindad. Particularmente es el elemento del primer argumento de la Función 1 el que será puntuado respecto a su vecindad en la Función 2. Esta es una de las razones por las que en algunos casos el resultado de la función no sea conmutativo respecto al orden de sus argumentos, ya que en el proceso de puntuación de más alto nivel se le da más “atención” al primer argumento de la función, al ser considerado como el elemento a puntuar.

Función 1: $puntuacionPar(e_{i_1j_1}, e_{i_2j_2})$. Función para obtener la puntuación de un par de elementos vecinos.

Datos: Celdas de los elementos vecinos $e_{i_1j_1}$, $e_{i_2j_2}$.

Resultado: Puntuación.

```

1 function puntuacionPar( $e_{i_1j_1}$ ,  $e_{i_2j_2}$ ):
2   switch  $C_{i_1j_1}$ ,  $C_{i_2j_2}$  do
3     case  $C_1, C_0$  do return 3
4     case  $C_1, C_2$  do return 1
5     case  $C_2, C_1$  do return 2
6     case  $C_2, C_0$  do return 1
7     case  $C_0, C_1$  do return 2
8     case  $C_0, C_2$  do return 1
9     case  $C_{k_C}, C_{k_C}$  do return 0
10   end
11 end
```

El sistema de puntuación presentado en la Función 1 fue diseñado con base en heurísticas simples, esto con el propósito de que su implementación fuese eficiente en términos de velocidad de procesamiento. Sin embargo, se intentó mantener un balance entre la eficiencia en el número de operaciones, proporcionada por la simpleza de las reglas de puntuación; y la eficiencia de los arreglos encontrados por el algoritmo, que está definida en términos del número total de acciones necesarias para tomar los objetos.

Dicho lo anterior, y teniendo en cuenta que posteriormente el primer argumento de la Función 1 fungirá como el elemento a ser puntuado en la Función 2, las heurísticas

detrás del sistema de puntuación utilizado en la Función 1 son las siguientes:

- Para los cubos (casos C_1, C_0 y C_1, C_2): la puntuación es máxima cuando su vecino es una celda vacía, debido a que es el elemento que puede permitir su sujeción; y es mínima, pero no cero, cuando su vecino es un prisma, ya que, si bien se restringe la sujeción del cubo, aún hay posibilidad de que el prisma sí pueda ser sujetable.
- Para los prismas (casos C_2, C_1 y C_2, C_0): la puntuación es máxima cuando su vecino es un cubo, pero no tanto como la del par *cubo-celda vacía*. La razón de esto es que, si bien la sujeción del prisma no se restringe, la del cubo sí, mientras que en el caso del par *cubo-celda vacía* no se genera ninguna restricción de sujeción. Por otro lado, la puntuación es mínima, pero no cero, cuando el vecino de un prisma es una celda vacía; esto debido a que, si bien la sujeción del prisma no se restringe, es posible que la celda vacía pueda ser mejor aprovechada al colocarla al lado de un cubo.
- Para las celdas vacías (casos C_0, C_1 y C_0, C_2): la puntuación es máxima cuando su vecino es un cubo, debido a que la celda vacía es el único elemento que permite su sujeción; y es mínima, pero no cero, cuando su vecino es un prisma, ya que si bien permite su sujeción, también existe otro elemento que la permite, pudiendo ser una mejor elección en determinadas situaciones.
- Para vecinos de la misma clase (caso C_{k_C}, C_{k_C}): la puntuación es cero. La razón de ello es que, tratándose de objetos, se restringe la sujeción de ambos, mientras que si se trata de celdas vacías, su capacidad para permitir la sujeción de objetos es desaprovechada.

Otra de las razones para que la función no sea conmutativa, en el caso de vecinos de clase C_1 y C_0 , es para no dar una preferencia excesiva a los pares de vecinos de estas clases, ya que esto puede causar que se generen pares innecesarios de este tipo en el arreglo. Por ejemplo, se podría generar una tendencia a hacer sujetables a los cubos mediante sus dos formas de sujeción H y V . Con esto se podría desaprovechar el uso de espacios vacíos solo en cubos en situaciones donde el número de prismas sea mayor y se requieran pares *prisma-celda vacía*.

En el caso de vecinos de clase C_1 y C_2 la razón es más evidente y tiene que ver con el elemento que se está puntuando, ya que en un caso no se restringe la sujeción de dicho elemento y en el otro sí.

La puntuación de un elemento en una celda e_{ij} de la malla es calculada por la Función 2, haciendo uso de la Función 1. Cuando el elemento corresponde a un objeto y este es sujetable, se calculan las puntuaciones con sus vecinos horizontales y verticales, estas dos puntuaciones se comparan y la puntuación que resulte mayor

es la que se asigna al elemento, añadiendo puntos extra; si el objeto no es sujetable, su puntuación solo es la suma de las puntuaciones con sus cuatro vecinos, restando puntos si el elemento es imposible de tomar. Si el elemento a puntuar se trata de una celda vacía, simplemente se suman las puntuaciones con sus cuatro vecinos, sin realizar ninguna operación adicional.

El proceso detallado para puntuar un elemento de la malla se muestra a continuación en la Función 2. Las funciones *sujetable()* e *imposible()* se describen al inicio de la Sección 4.1.

Función 2: *puntuacionElemento(e_{ij})*. Función para calcular la puntuación de un elemento de la malla.

Datos: Celda del elemento a puntuar e_{ij} .

Resultado: Puntuación p del elemento.

```

1 function puntuacionElemento( $e_{ij}$ ):
2    $p_H = puntuacionPar(e_{ij}, V_{i,j,2}) + puntuacionPar(e_{ij}, V_{i,j,4})$ 
3    $p_V = puntuacionPar(e_{ij}, V_{i,j,1}) + puntuacionPar(e_{ij}, V_{i,j,3})$ 
4    $p = p_H + p_V$ 
5   if  $C_{ij} \neq C_0$  then
6     if sujetable( $e_{ij}$ ) then  $p = \max(p_H, p_V) + 3$ 
7     if imposible( $e_{ij}$ ) then  $p -= 2$ 
8   end
9   return  $p$ 
10 end

```

Finalmente, se define la puntuación global de la malla como la suma de las puntuaciones individuales de todos sus elementos:

$$puntuacionGlobal() = \sum_{i,j} puntuacionElemento(e_{ij}) \quad (4.5)$$

4.1.2 Costo de tomar los objetos

El algoritmo para calcular el costo t_{ij} de tomar un objeto de la malla está basado en el algoritmo RSC, presentado en [12]. Dicho algoritmo consiste básicamente en crear un árbol de objetos-obstáculo que impiden la sujeción de un objeto particular. La raíz del árbol es el objeto que se desea tomar, y a partir de este se ramifican nodos que corresponden a sus objetos-obstáculo más cercanos. A su vez, de cada uno

de estos nodos se desprenden más ramas o nodos, correspondientes a los objetos-obstáculo de cada uno, y así sucesivamente hasta llegar a las hojas del árbol, las cuales corresponden a objetos-obstáculo que sí son sujetables.

Al retirar los objetos correspondientes a los nodos hoja, se pueden hacer sujetables los objetos de sus nodos padres, con lo cual se convierten en nuevos nodos hoja, reduciendo el tamaño del árbol de objetos-obstáculo. Al continuar con este proceso se puede hacer sujetable al objeto del nodo raíz.

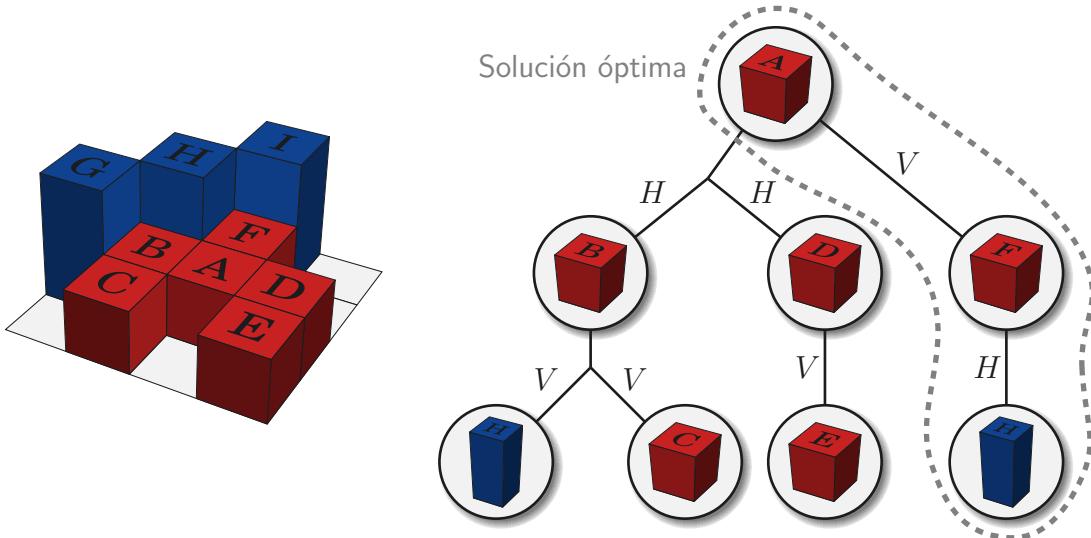


Figura 4.1: Ejemplo de arreglo de objetos (izquierda) y el árbol de objetos-obstáculo correspondiente al cubo A (derecha). Como se puede observar, la solución óptima para tomar este cubo tiene costo 3.

De acuerdo con las formas de sujeción establecidas, existen en general, dos secuencias distintas de objetos-obstáculo que hay que retirar de la malla para hacer sujetable a otro objeto particular. El costo t_{ij} para tomar un objeto en la celda e_{ij} es entonces la longitud de la secuencia con menos objetos a retirar, tal como se muestra en la Figura 4.1, donde esta secuencia corresponde a la rama de menor longitud del árbol generado. El cálculo de t_{ij} se detalla en el Algoritmo 1, el cual utiliza la Función 3 de forma recursiva.

La Función 3 implementa un algoritmo de *backtracking* para encontrar una de las secuencias de objetos-obstáculo para hacer sujetable y tomar un objeto de interés. La secuencia encontrada por la función es considerada como solución al problema de hacer sujetable y tomar un objeto, esto debido a que él o los elementos al final de la secuencia siempre corresponderán a nodos hoja en el árbol de objetos-obstáculo del objeto de interés, como el que se muestra en Figura 4.1. Al retirar estos objetos, se harán sujetables los siguientes objetos de la secuencia y así sucesivamente hasta hacer sujetable al primer elemento de la secuencia, el cual corresponde al objeto que

se desea tomar.

El algoritmo de *backtracking* explora las ramas del árbol de objetos-obstáculo de un objeto particular y determina si al final de la rama existen objetos sujetables o no. Si no es posible retirar ningún objeto de la rama en cuestión, la función falla. Esto sucede cuando el elemento de la malla que se está analizando ya se encuentra agregado en la secuencia, lo cual puede ser indicativo de la existencia de un cuadro imposible en el arreglo. Al fallar una función, todos sus procesos y los de las funciones en recursiones previas que la llamaron directa o indirectamente se invalidan, lo cual se traduce en que la secuencia que se había estado construyendo hasta ese momento sea descartada. Después de esto se procede a explorar otra rama del árbol.

La primera rama para la cual la función finalice con éxito es la que se toma como solución y cuyos objetos son añadidos a la secuencia considerada como resultado de la función.

Función 3: $\text{secuencia}(e_{ij}, l)$. Función de *backtracking* para obtener una secuencia de objetos a retirar para hacer sujetable y tomar un objeto en la celda e_{ij} , el cual también se incluye al final de la secuencia.

Datos: Celda e_{ij} del objeto deseado y lista vacía l .

Resultado: Lista l con la secuencia de elementos a retirar.

Definiciones:

h_{ij} altura del elemento en la celda e_{ij} .

o_{ij} información de interés (clase y ubicación) del objeto en la celda e_{ij} que será añadida a la secuencia.

```

1 function secuencia( $e_{ij}$ ,  $l$ ):
2   if  $C_{ij} = C_0$  then Finalizar recursión con éxito.
3   if  $o_{ij} \notin l$  then
4     append( $o_{ij}$ ,  $l$ )
5     if sujetable( $e_{ij}$ ) then
6       | Finalizar recursión con éxito.
7       | ↳ Ejecutar uno de los siguientes pares de
      recursiones:
8       |   if  $h_{i+1,j} \geq h_{ij}$  then secuencia( $e_{i+1,j}$ ,  $l$ )
9       |   | if  $h_{i-1,j} \geq h_{ij}$  then secuencia( $e_{i-1,j}$ ,  $l$ )
10      | else if Par 1:
11        |   | if  $h_{i,j+1} \geq h_{ij}$  then secuencia( $e_{i,j+1}$ ,  $l$ )
12        |   | if  $h_{i,j-1} \geq h_{ij}$  then secuencia( $e_{i,j-1}$ ,  $l$ )
13      | else if Par 2:
14        |   | if  $h_{i,j+1} \geq h_{ij}$  then secuencia( $e_{i,j+1}$ ,  $l$ )
15        |   | if  $h_{i,j-1} \geq h_{ij}$  then secuencia( $e_{i,j-1}$ ,  $l$ )
16      | else
17        |   | ↳ Si en ambos pares alguna de las dos
          recursiones terminó con fallo:
18        |   | Finalizar recursión con fallo (hacer backtracking e intentar
19        |   | otra alternativa (par) de recursión).
20      | end
21    | else
22    |   | Finalizar recursión con fallo (hacer backtracking e intentar
23    |   | otra alternativa (par) de recursión).
24  | end
25 Finalizar recursión con éxito.
26 end

```

A grandes rasgos, el funcionamiento del Algoritmo 1 consiste en explorar todas las ramas del árbol de objetos-obstáculo generado para un objeto de interés (como el mostrado en la Figura 4.1), mediante la Función 3, encontrando todas sus soluciones posibles; esto es, todas las secuencias de objetos-obstáculo consideradas como solución. El algoritmo cuenta los objetos-obstáculo de cada una de las secuencias y finalmente retorna la longitud de la rama más corta.

Algoritmo 1: Algoritmo para calcular el costo t_{ij} .

Datos: Celda e_{ij} del objeto deseado.

Resultado: Costo t_{ij} de tomar el objeto en e_{ij} .

❑ *Lista para almacenar todas las secuencias posibles para tomar un objeto.*

1 $L = \{\}$

❑ *Función que retorna todas las soluciones posibles de secuencia().*

2 `encontrarTodas()`

3 $L = \text{encontrarTodas}(\text{secuencia}(e_{ij}, l))$

4 $t_{ij} = \min_i |L_i|$

El costo global T , como se definió en la Sección 3.1, es simplemente la suma de los costos individuales t_{ij} :

$$T = \sum_{i,j} t_{ij} \quad (4.6)$$

4.2 Algoritmo principal

La aportación principal de este trabajo consiste en un algoritmo para el acomodo de objetos en un arreglo, que minimiza, en promedio, el costo posterior de acceso a ellos, esto es, minimiza el costo global T previamente definido. Dicho algoritmo, el cual adquiere sus fortalezas de todas las definiciones y condiciones precedentes, es un algoritmo de recocido simulado que utiliza a la función `puntuacionGlobal()` como criterio de evaluación para encontrar un acomodo adecuado de los objetos, acorde a los fines establecidos.

La definición original del algoritmo de recocido simulado se puede encontrar en el artículo [42], en el cual también se puede encontrar una implementación muy parecida

a la realizada en este trabajo, la cual consiste en optimizar la longitud de cableado en la colocación de chips en una tarjeta de circuitos, modelándola como un arreglo en forma de malla 2D.

En el problema propuesto se plantean una serie de restricciones o reglas, las cuales deben ser cumplidas independientemente del método que se utilice para encontrar los acomodos de objetos óptimos. Por ejemplo, una de las reglas básicas del problema, es que siempre se debe de respetar el número de objetos de cada clase que deben estar presentes en el arreglo final, retornado como solución. Es a partir de este tipo de restricciones y reglas, así como de las libertades que en sí mismas se establecen, que se debe elegir una metodología de solución apropiada para el problema planteado.

Una estrategia de búsqueda razonable para encontrar un arreglo de objetos óptimo es partir de un acomodo arbitrario de objetos, el cual cumpla desde un inicio con la restricción descrita del número de objetos de cada clase en la malla y realizar cambios en este que, además de que estén orientados a mejorar el arreglo de acuerdo al criterio establecido, siempre mantengan de igual forma el número de los distintos objetos en la malla del arreglo anterior. La clave del éxito de dicha estrategia de búsqueda dependerá de los criterios bajo los cuales se realicen dichos cambios y de cómo estos cumplirán con las restricciones del problema. En este caso, el criterio principal para generar tales cambios es el sistema de puntuaciones propuesto.

Se eligió un algoritmo de recocido simulado para abordar el problema propuesto debido a que se adecúa muy bien a la búsqueda de vecindad planteada para este problema, la cual consiste en cambios graduales o mínimos en un arreglo dado, considerado como solución actual. Dichos cambios graduales se refieren a los intercambios de pares de elementos en un arreglo, ya que, bajo las reglas establecidas, un intercambio de un par de elementos distintos en un arreglo se considera como el cambio mínimo necesario para hacerlo cambiar de estado, sin alterar el número de objetos de cada clase que hay en él.

Si bien pudieran existir mejores soluciones, al emplear otro tipo de algoritmos o metodologías, el objetivo principal de este trabajo no es encontrar la solución más eficiente, sino encontrar o diseñar una solución aceptable al problema propuesto, del cual no parece haber antecedentes según la investigación realizada. Se cree que, debido al nivel de simplificación y de condiciones particulares que requiere problema, encontrar el método de solución más eficiente no sería de mucha utilidad (más allá de la intelectual) en esta etapa, como si lo sería en etapas posteriores donde se pueda aplicar en situaciones reales.

No obstante, también se consideraron otros algoritmos y metodologías de búsqueda, de entre los cuales se eligió al recocido simulado por el balance entre la simpleza del método, su conocida eficiencia, además de la ya mencionada facilidad con la que se adapta a las reglas de búsqueda del problema propuesto.

Otros métodos de reconocido éxito fueron descartados al considerar que, o bien representaban un mayor costo computacional, o bien su metodología de búsqueda no

se adecuaba a las características del problema. Por ejemplo, en un algoritmo genético, se cree que los cambios en las instancias de arreglos serían más “abruptos” cuando se explora el espacio de búsqueda, debido a que en las recombinaciones de población se realizan muchos cambios para generar la descendencia, que pueden alejarse mucho de los arreglos padres y que, generalmente, no los mejoran en nuestro caso. Además de que la condición en la que el número de elementos de cada clase en el arreglo se debe mantener de una recombinación a otra, es mucho más difícil de cumplir en un algoritmo genético. Por lo cual, en dicho caso, se concluyó que no es la mejor manera de guiar la búsqueda hacia los arreglos óptimos.

En el Algoritmo 2 se presenta el algoritmo de recocido simulado utilizado.

Algoritmo 2: Recocido simulado.

Datos: Temperatura inicial T , constantes α, a, b , malla con objetos.

Resultado: Malla con objetos acomodados (representada por una matriz de enteros).

Parámetros del algoritmo.

```

1  $T = 50,000,000$ 
2  $\alpha = 0.0000001$ 
3  $a = 0.99991$ 
4  $b = 1$ 
```

Puntajes actual y de intercambio.

```

5  $p_A = \text{puntuacionGlobal}()$ 
6  $p_I = 0$ 
```

```

7 while  $T > \frac{b}{1-a}$  do
8     Intercambiar aleatoriamente 2 elementos diferentes en la malla.
9      $p_I = \text{puntuacionGlobal}()$ 
10    if  $p_A \leq p_I$  or  $\text{random}([0, 1]) < e^{-\frac{p_A - p_I}{\alpha T}}$  then
11        Se acepta el intercambio.
12         $p_A = p_I$ 
13    else
14        No se acepta el intercambio.
15    end
16     $T = aT + b$ 
17 end
```

El algoritmo recibe como entrada principal una malla con los objetos acomodados de forma aleatoria. Después de inicializar los parámetros del algoritmo, tales como la temperatura, se inicia un proceso iterativo, en el cual, en primer lugar se calcula la puntuación global de la malla, para luego escoger dos elementos distintos en la malla de forma aleatoria e intercambiarlos de posición. Dicho par de elementos puede ser un par *objeto-objeto* o bien *objeto-celda vacía*. Al simular este intercambio se vuelve a calcular la puntuación global de la malla y esta se compara con la puntuación calculada antes de hacer el intercambio. Si la puntuación después de hacer el intercambio es mejor, entonces el intercambio se acepta, de lo contrario la probabilidad de que se acepte disminuye considerablemente. Este proceso de intercambios se repite hasta que se cumpla la condición de paro, la cual está relacionada con el decremento de la temperatura. Finalmente, la salida del algoritmo será el acomodo encontrado.

Los parámetros iniciales mostrados en el Algoritmo 2 fueron los utilizados para generar los arreglos de mayor tamaño (8×8). Para arreglos de menor tamaño, estos parámetros se pueden relajar hasta cierto punto, sin sacrificar la calidad de los resultados. El esquema de reducción de temperatura utilizado es el esquema aritmético-geométrico [43].

Si bien el proceso de ponderación de un arreglo se ideó con el fin de que el arreglo con mayor puntuación sea el óptimo, este podría no siempre ser el caso. Esto es debido a que existen algunos pocos casos especiales en los que, a causa de cómo están constituidas las reglas de puntuación, el algoritmo no le da la mejor puntuación al arreglo óptimo, sino a uno que tiene, en la mayoría de los casos donde se pudo corroborar, un costo global de una o dos acciones por encima del óptimo. Estos casos especiales se pueden corregir mediante una extensión de las reglas de puntuación que considere las situaciones en las que ocurren, sin embargo, debido a que son muy poco frecuentes, se cree que la ganancia de extender las reglas es poca comparada con la complejidad añadida al modelo, por lo cual en esta ocasión no se considerará tal extensión.

Con la finalidad de evaluar el desempeño de dicho proceso de ponderación y del algoritmo propuesto, también se calculó el costo global real de los arreglos encontrados por el algoritmo propuesto, mediante el proceso descrito en la Sección 4.1.2, el cual es un proceso exhaustivo. Los costos globales de los arreglos encontrados por el algoritmo propuesto se compararon con los costos globales de los arreglos óptimos correspondientes, obtenidos mediante una búsqueda exhaustiva en los casos donde fue posible realizarla. Los resultados de esta comparación se presentan en el siguiente capítulo.

Capítulo 5

Resultados

En este capítulo se presentarán los resultados obtenidos al aplicar la formulación y metodología propuesta a diferentes instancias del problema. Estas instancias consisten en distintos tamaños de mallas, donde, para cada uno de ellos, se probaron todas las combinaciones de objetos de clases C_1 y C_2 posibles, solo descartando las que correspondían a casos triviales o imposibles. Los diferentes tamaños de malla con los que se trabajó fueron:

3×3	5×5	7×7
3×5	6×6	8×8
4×4	5×8	

Para cada una de las combinaciones de objetos en la malla propuestas, se comparó el costo global del acomodo encontrado por el algoritmo, respecto al costo global del mejor acomodo encontrado al realizar una búsqueda exhaustiva en todas las permutaciones posibles de los objetos en la malla, esto para mallas de un tamaño de hasta 4×4 . Para mallas más grandes se realizó una comparación del algoritmo propuesto solo contra un subconjunto de permutaciones y no contra una búsqueda exhaustiva. Lo anterior debido a que la cantidad de permutaciones posibles de elementos en mallas de estos tamaños es muy grande, por lo que una búsqueda exhaustiva es inviable con los recursos computacionales utilizados en cuanto al tiempo de procesamiento requerido.

Los subconjuntos de permutaciones se generaron de manera aleatoria. La cantidad de permutaciones generadas fue de 2,000,000 por cada tamaño distinto de la malla, repartidas en cada una de las combinaciones de objetos probadas. Esto es, para tamaños de malla iguales o mayores a 5×5 , la suma de las permutaciones generadas, de todas las combinaciones permitidas de N_1 y N_2 objetos, es de 2,000,000.

La cantidad de permutaciones generadas para cada caso particular N_1 , N_2 es proporcional al número total de objetos en la malla, es decir a N . Esto debido a que la probabilidad de encontrar el o uno de los arreglos óptimos en cada caso es inversamente proporcional a la cantidad de objetos en la malla.

Los puntos en las gráficas ubicados en el infinito indican que en la muestra generada no se encontró un arreglo que no correspondiera a un caso imposible.

A continuación se presentan los resultados obtenidos para cada tamaño de malla, junto con un ejemplo ilustrativo de un caso particular, en el que se muestra el acomodo aleatorio inicial y el acomodo encontrado por el algoritmo. En cada gráfica se muestran, en los ejes horizontales, el número de objetos de cada clase utilizados para formar el acomodo de objetos en la malla, y en el eje vertical el costo global asociado del arreglo encontrado.

En los casos donde es posible comprobar si el costo global del arreglo encontrado es óptimo, se utilizó una marca diferente en la gráfica, para diferenciarlos de los casos donde, o no es posible comprobarlo o simplemente no lo son. Por ejemplo, en las mallas de tamaño menor a 5×5 , los arreglos encontrados por la búsqueda exhaustiva siempre serán óptimos. Para arreglos de tamaño igual o mayor a 5×5 es posible saber si un arreglo es óptimo si el costo global del arreglo es igual al número de objetos presentes en él.

Para la implementación de las funciones y algoritmos descritos en el Capítulo 4, se utilizaron los lenguajes de programación C++ y Prolog. Mediante el primero se implementó el algoritmo principal, así como las funciones de ponderación que este utiliza, mientras que el segundo se utilizó para implementar las funciones de *backtracking* que calculan el costo global de cada arreglo*. La ejecución de los programas desarrollados se llevó a cabo en una laptop Lenovo modelo Ideapad 320, con un procesador Intel® Core™ i7-7500U CPU @ 2.70 GHz × 4 y 8 GB de memoria RAM.

* El código del proyecto se puede encontrar en <https://github.com/JoseAlbertoLopezLopez>.

Resultados en mallas de 3×3

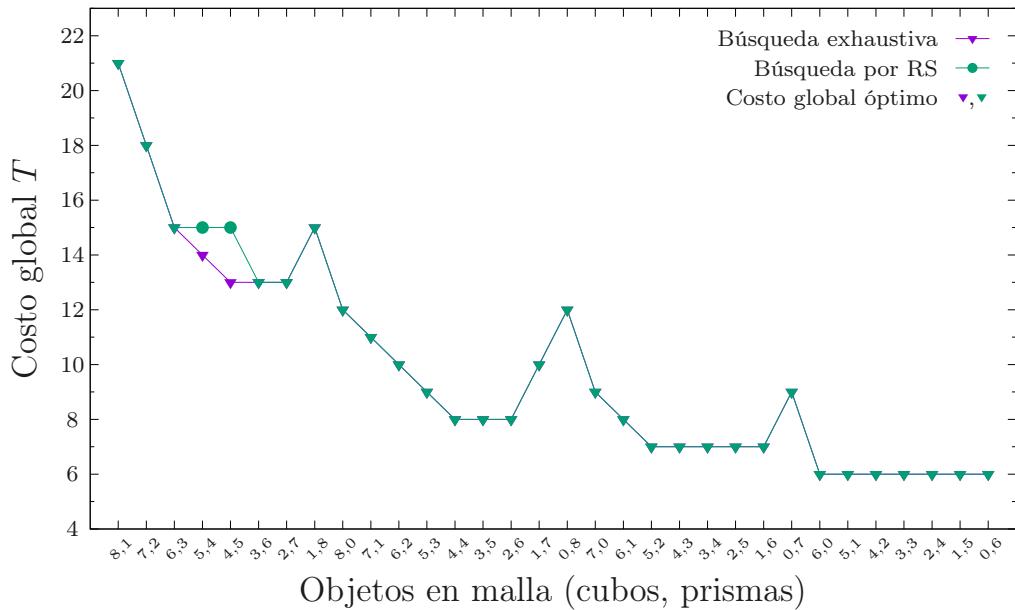


Figura 5.1: Comparación de la evaluación de los mejores acomodos encontrados por una búsqueda exhaustiva contra los encontrados por el algoritmo propuesto.

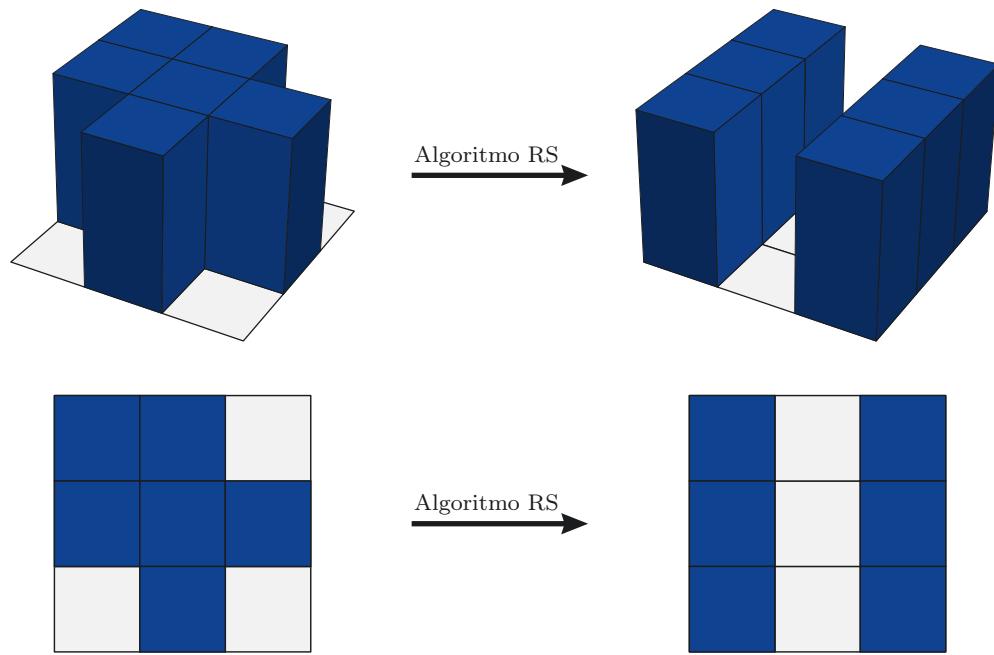


Figura 5.2: Acomodo aleatorio inicial (izquierda) y acomodo encontrado por el algoritmo propuesto (derecha), mostrados en diferentes vistas.

Resultados en mallas de 3×5

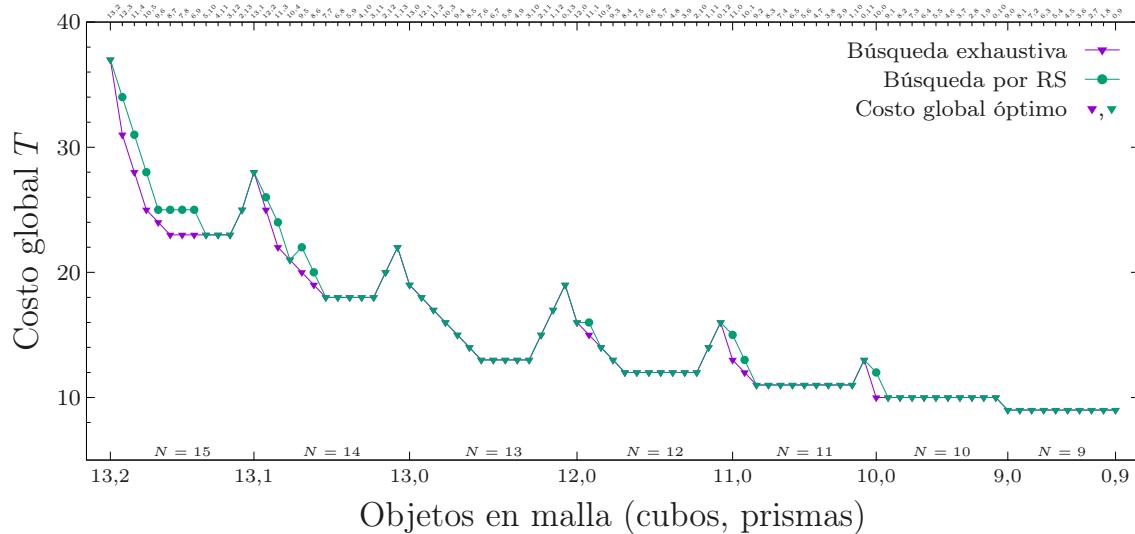


Figura 5.3: Comparación de la evaluación de los mejores acomodos encontrados por una búsqueda exhaustiva contra los encontrados por el algoritmo propuesto.

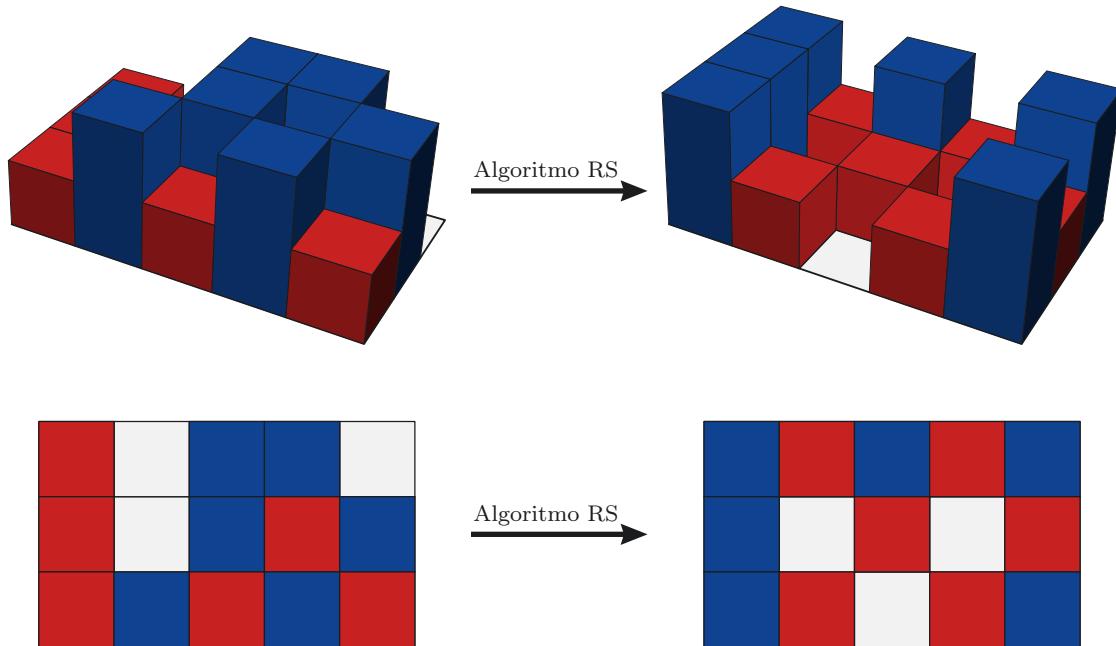


Figura 5.4: Acomodo aleatorio inicial (izquierda) y acomodo encontrado por el algoritmo propuesto (derecha), mostrados en diferentes vistas.

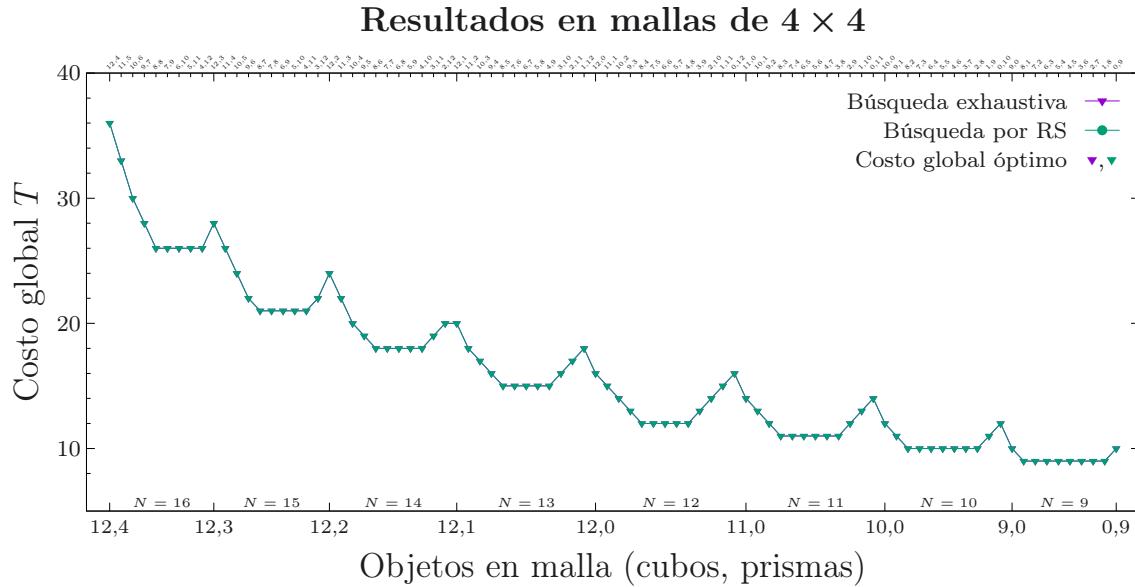


Figura 5.5: Comparación de la evaluación de los mejores acomodos encontrados por una búsqueda exhaustiva contra los encontrados por el algoritmo propuesto.

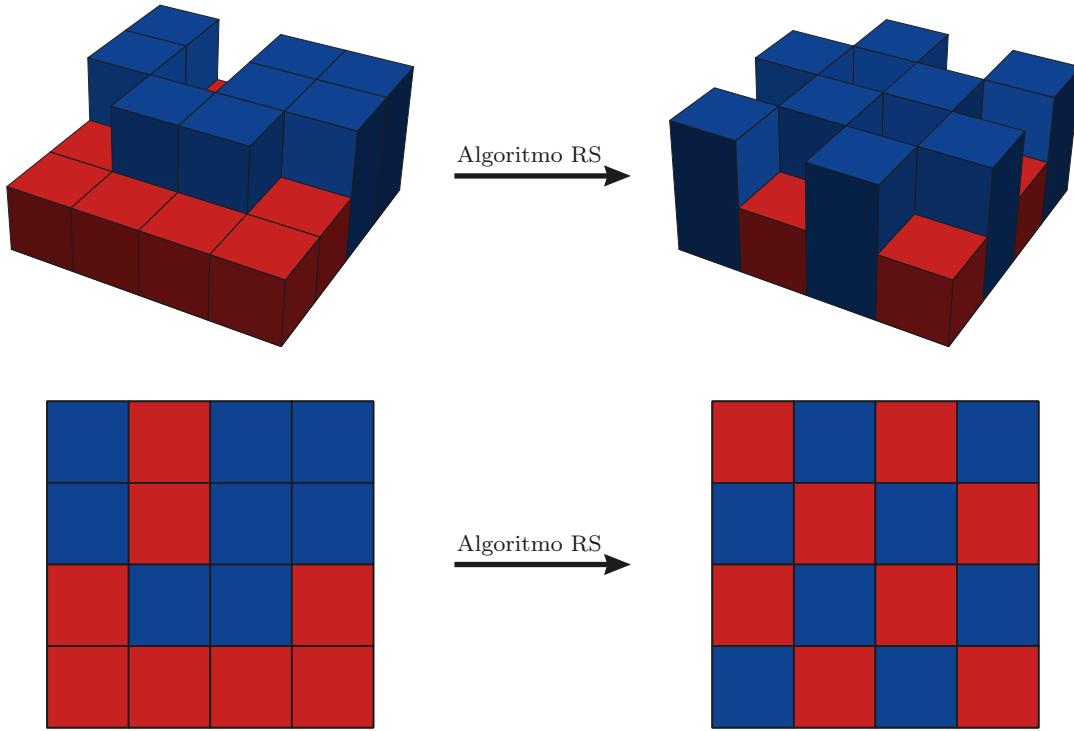


Figura 5.6: Acomodo aleatorio inicial (izquierda) y acomodo encontrado por el algoritmo propuesto (derecha), mostrados en diferentes vistas.

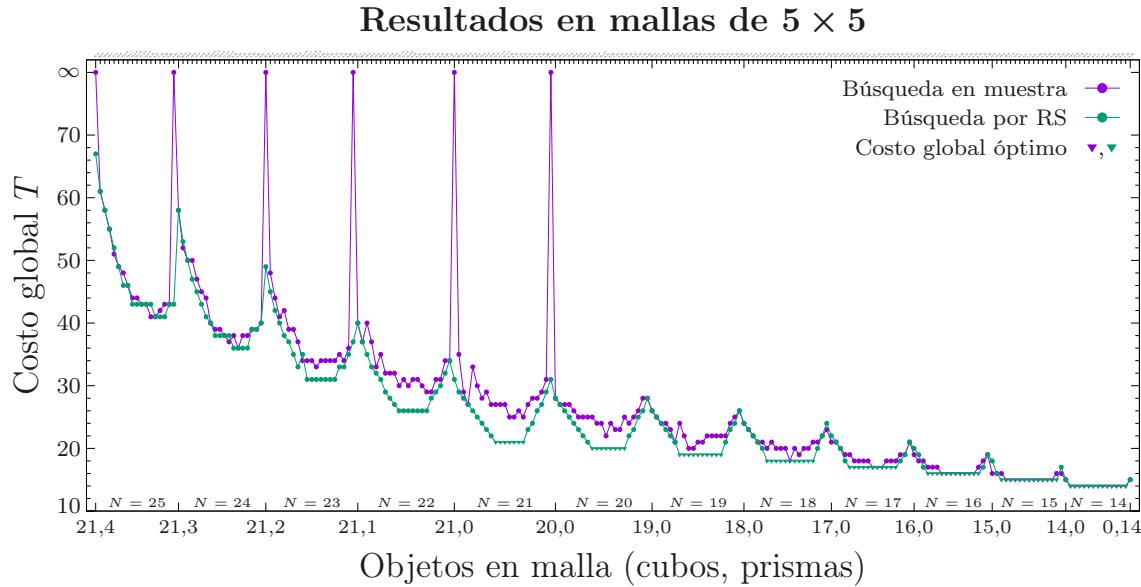


Figura 5.7: Comparación de la evaluación de los mejores acomodos encontrados en una muestra aleatoria y los encontrados por el algoritmo propuesto.

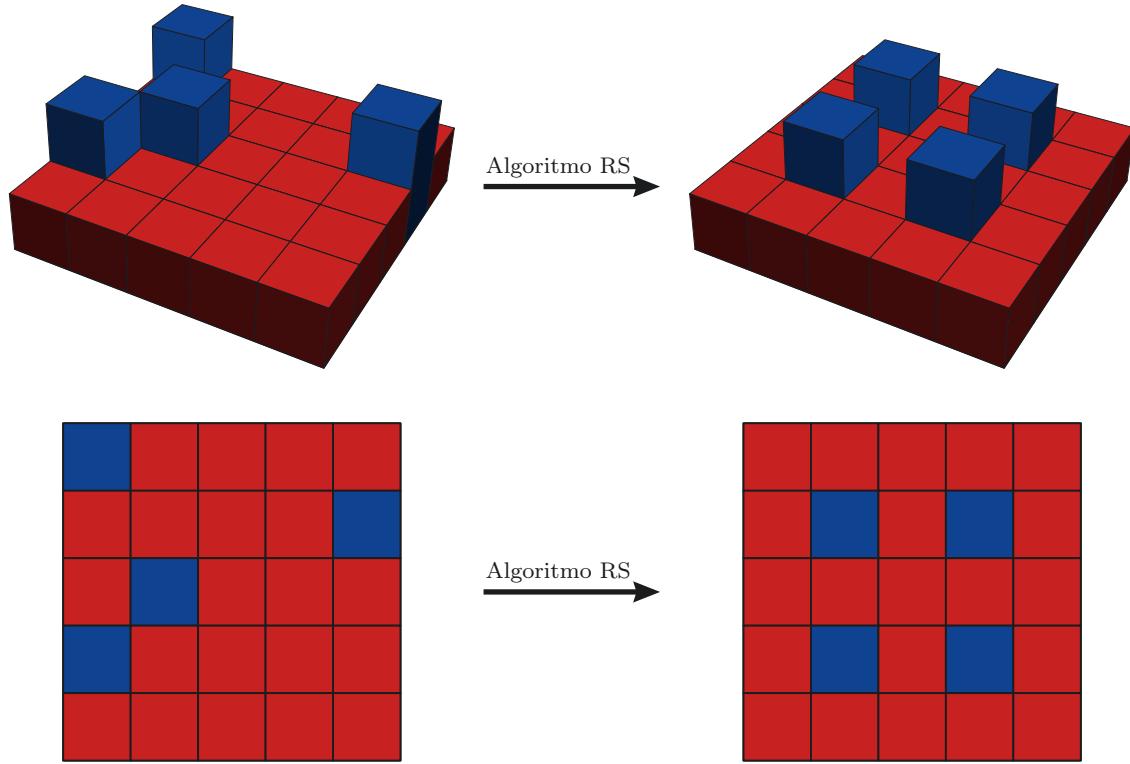


Figura 5.8: Acomodo aleatorio inicial (izquierda) y acomodo encontrado por el algoritmo propuesto (derecha), mostrados en diferentes vistas.

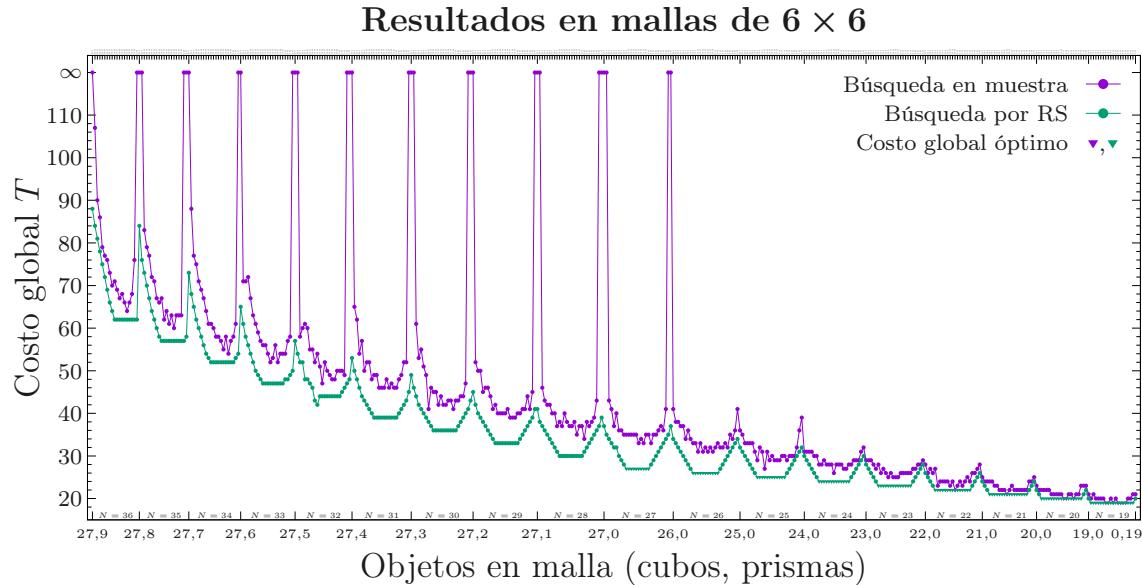


Figura 5.9: Comparación de la evaluación de los mejores acomodos encontrados en una muestra aleatoria y los encontrados por el algoritmo propuesto.

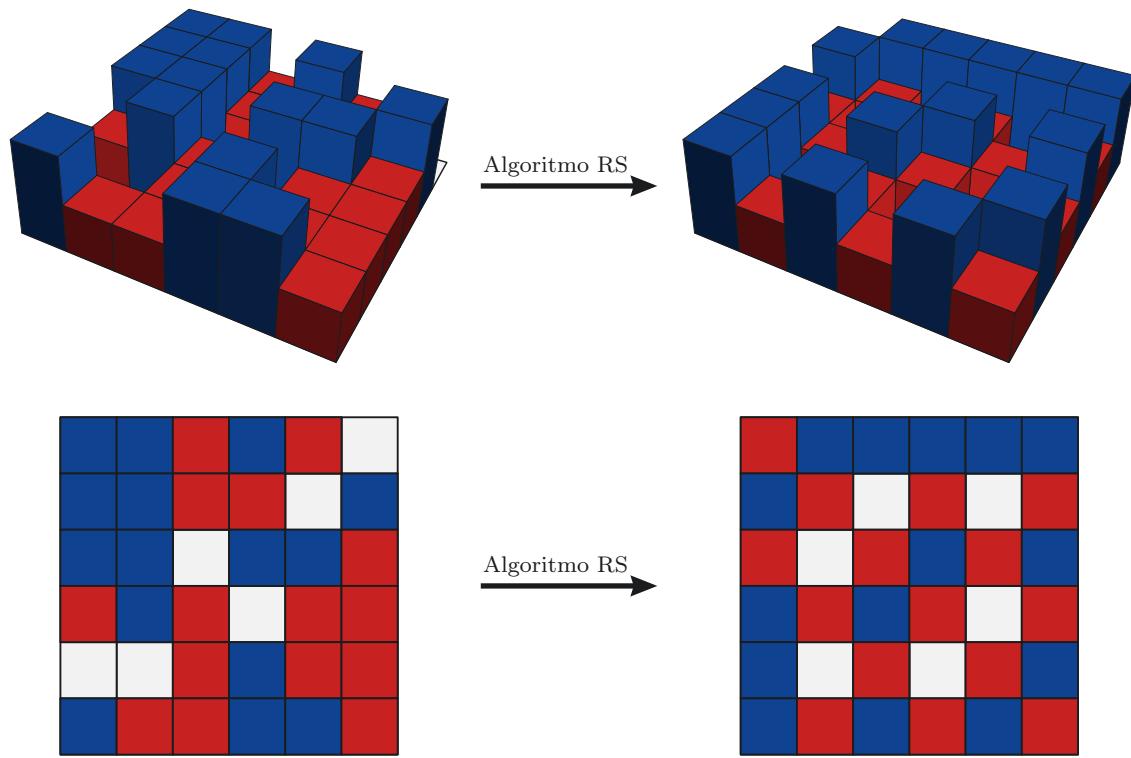


Figura 5.10: Acomodo aleatorio inicial (izquierda) y acomodo encontrado por el algoritmo propuesto (derecha), mostrados en diferentes vistas.

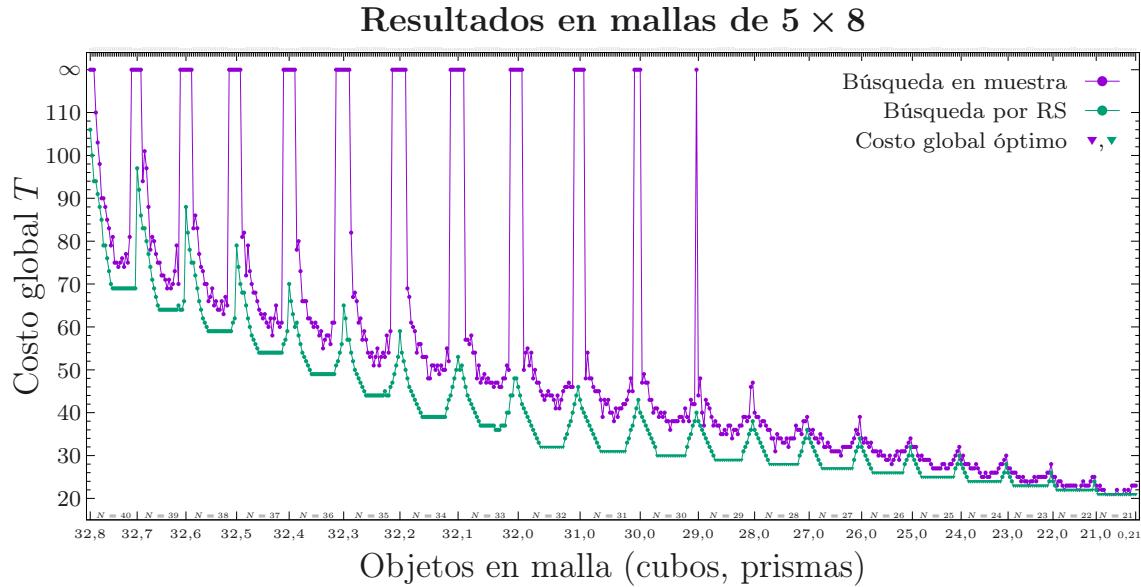


Figura 5.11: Comparación de la evaluación de los mejores acomodos encontrados en una muestra aleatoria contra los encontrados por el algoritmo propuesto.

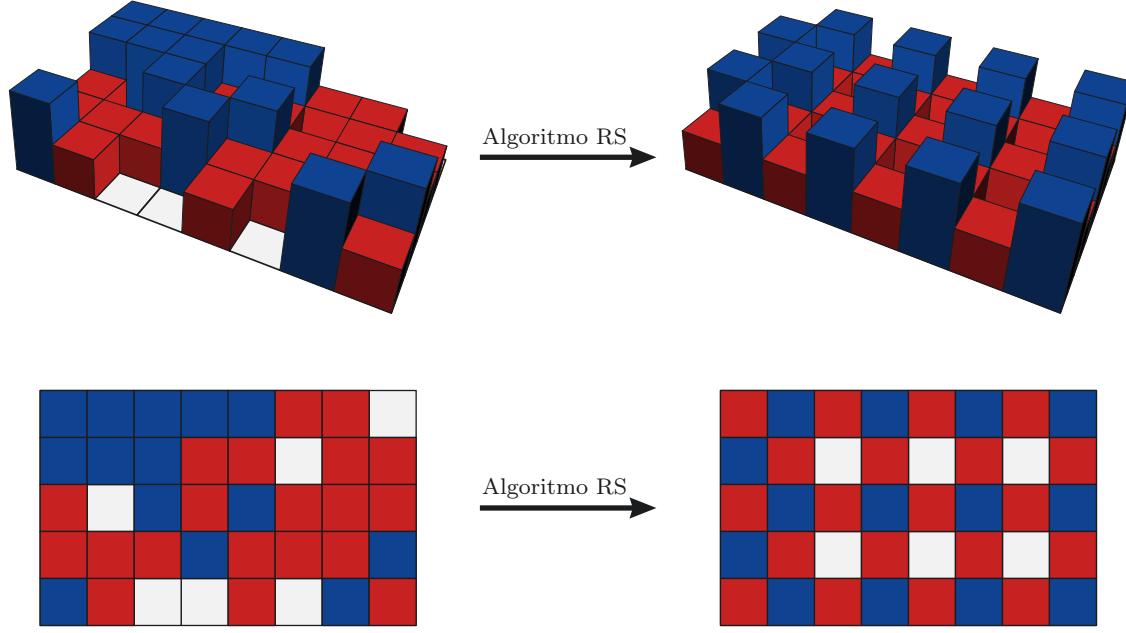


Figura 5.12: Acomodo aleatorio inicial (izquierda) y acomodo encontrado por el algoritmo propuesto (derecha), mostrados en diferentes vistas.

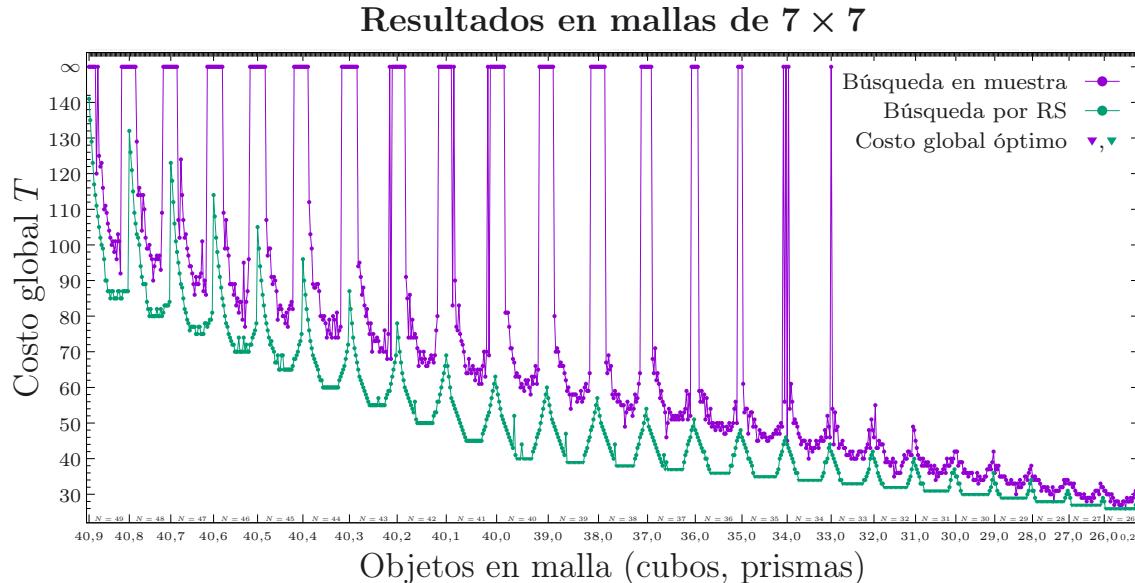


Figura 5.13: Comparación de la evaluación de los mejores acomodos encontrados en una muestra aleatoria y los encontrados por el algoritmo propuesto.

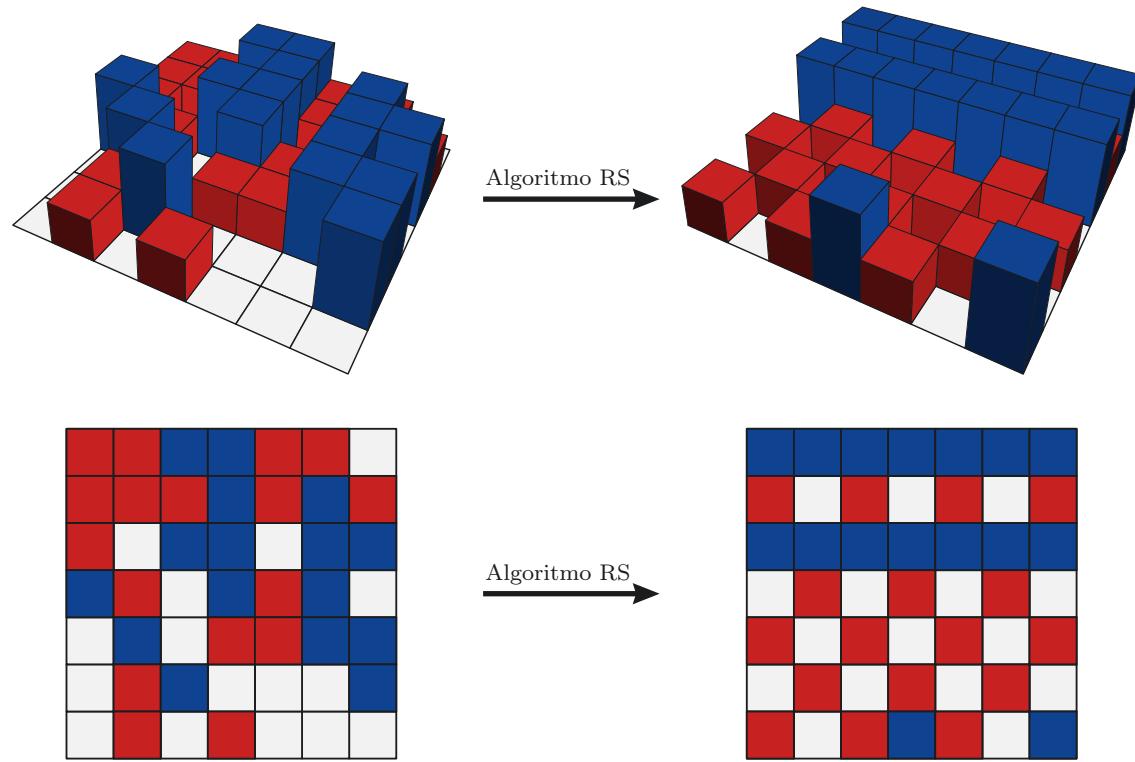


Figura 5.14: Acomodo aleatorio inicial (izquierda) y acomodo encontrado por el algoritmo propuesto (derecha), mostrados en diferentes vistas.

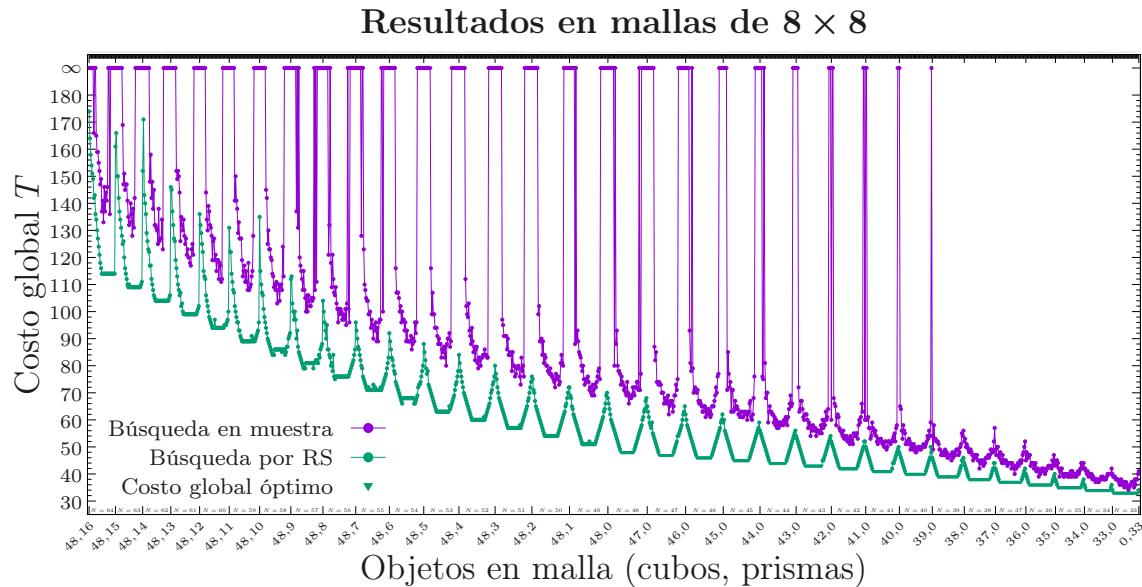


Figura 5.15: Comparación de la evaluación de los mejores acomodos encontrados en una muestra aleatoria y los encontrados por el algoritmo propuesto.

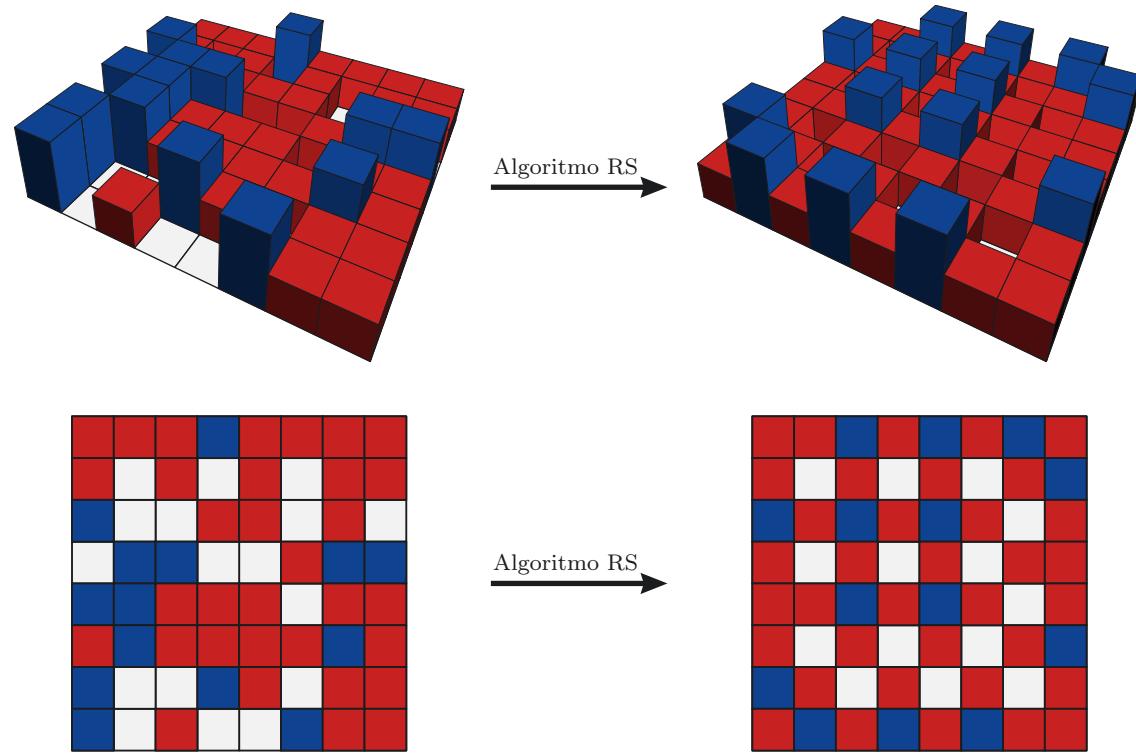


Figura 5.16: Acomodo aleatorio inicial (izquierda) y acomodo encontrado por el algoritmo propuesto (derecha), mostrados en diferentes vistas.

A pesar de ser un algoritmo estocástico, se observó bastante regularidad y consistencia en los resultados obtenidos. En la mayoría de los casos se llegaba al mismo arreglo tras varias ejecuciones del algoritmo, para los mismos datos de entrada. Sin embargo, en ocasiones se llegó a diferentes resultados para una misma entrada. En estos casos los arreglos encontrados obtuvieron la misma puntuación o una muy similar. Esta variabilidad no representa mayor inconveniente en los resultados obtenidos, ya que es poco frecuente y el costo global de los arreglos encontrados se mantiene, pudiendo servir incluso para encontrar diferentes alternativas de arreglos óptimos.

Como se mencionó previamente en este capítulo, a pesar de que a partir de los tamaños de malla de 5×5 no fue posible realizar una búsqueda exhaustiva para comparar los resultados del algoritmo, en algunos casos se puede verificar si el acomodo obtenido por el algoritmo es óptimo. Esto se puede determinar simplemente verificando si el costo global del acomodo obtenido es igual al número de objetos en él. Al observar las gráficas de resultados se puede ver que esto ocurre en varias ocasiones, en las regiones que asemejan valles en la curva generada por los resultados del algoritmo de recocido simulado.

Para los casos donde fue posible comparar el arreglo encontrado por el algoritmo propuesto con el arreglo óptimo, encontrado mediante una búsqueda exhaustiva, se obtuvo la información estadística presentada en la Tabla 5.1.

Tabla 5.1: Resultados del algoritmo propuesto para mallas de tamaño igual o menor a 4×4 .

Malla	# Casos óptimos	# Total casos	% Óptimos	Promedio de acciones extra*
3×3	30	32	93.75 %	0.09
3×5	70	85	82.35 %	0.33
4×4	88	88	100 %	0

* Promedio de acciones (o costo global T) extra por caso. Este valor se calcula mediante la expresión $(\sum T - \sum T_{\text{opt}}) / \# \text{Total casos}$, donde $\sum T_{\text{opt}}$ es la suma de costos óptimos para todos los casos probados.

Como se puede notar, en el 6.25 % de los casos para mallas de 3×3 y en el 17.65 % de los casos para mallas de 3×5 , el algoritmo no es capaz de encontrar el arreglo óptimo. Esto puede resultar confuso, ya que se trata de tamaños de malla más pequeños que el tamaño 4×4 , para el cual el algoritmo siempre encuentra un arreglo óptimo. Esto es debido a la razón explicada al final de la Sección 4.2: se pueden presentar casos especiales en los que las reglas de puntuación establecidas no siempre otorguen la mejor puntuación al arreglo óptimo, lo cual es posible corregir modificando las heurísticas utilizadas para puntuar, probablemente a cambio de un aumento en la

complejidad del modelo.

Sin embargo, en los casos donde no se llega a un arreglo óptimo, el algoritmo encuentra arreglos que no están muy alejados de este, ya que están solo unos pocos movimientos por encima del óptimo. Es decir, por ejemplo, que la única diferencia entre el arreglo encontrado y el óptimo, es que para uno o dos objetos en el arreglo se necesitarían uno o dos movimientos extra para acceder a ellos, manteniendo costos óptimos de acceso para el resto. Lo cual se expresa en la métrica del promedio de acciones extra, mostrada en la Tabla 5.1.

En la gran mayoría de los casos donde es posible comprobarlo, el algoritmo otorga la mejor puntuación al arreglo con costo el global mínimo.

Capítulo 6

Conclusiones

En el presente trabajo se diseñó un método para el acomodo de objetos en un espacio delimitado, el cual tiene como objetivo reducir el posterior costo de acceso a estos. Para ello se abordó el problema de una forma simplificada de como se haría en una situación real. Esto es, discretizando el espacio donde se colocan los objetos, así como reduciendo la complejidad geométrica de los objetos y la cantidad de clases u objetos distintos que se pueden utilizar. Debido a su naturaleza discreta, el problema se puede abordar como una búsqueda en un espacio de permutaciones, para lo cual existen diversas metodologías y algoritmos de búsqueda, de entre los cuales se optó por un algoritmo de recocido simulado.

Dado que no existe a la fecha una investigación que aborde el problema planteado, el aporte principal del presente trabajo es la definición de los parámetros y reglas para evaluar qué tan adecuado es un acomodo, de acuerdo a los objetivos establecidos. Dicha evaluación está basada en la geometría de los objetos y en cómo esta afecta su sujeción cuando los objetos están en determinada configuración. De forma que al combinar esto con el algoritmo de búsqueda elegido, se puede obtener el acomodo requerido.

Los resultados muestran que, en los casos donde fue posible comparar el método propuesto con una búsqueda exhaustiva, se obtuvo en el 91.7 % de los casos, los mismos resultados que dicha búsqueda, esto es, un arreglo óptimo de acuerdo a las reglas establecidas. Mientras que en casos más complejos, en los que no fue posible realizar una búsqueda exhaustiva, los resultados siguen siendo buenos, llegando a ser óptimos en varios casos para los que fue posible determinar tal característica. Con lo cual, se puede aceptar la hipótesis de investigación propuesta, al menos para el problema definido.

Como trabajo futuro se propone expandir las capacidades del método desarrollado. Esto es, adaptarlo para que se puedan abordar casos más complejos y diversos, por ejemplo, con una mayor cantidad de objetos, así como incluir más variedades

Capítulo 6. *Conclusiones*

de geometrías para estos, es decir, incluir una mayor cantidad de clases. Además, se podrían relajar las restricciones acerca de cómo se pueden colocar los objetos en el espacio, permitiendo configuraciones en las que los objetos puedan ser colocados encima de otros y además puedan tener un número mayor de orientaciones.

El método de búsqueda también podría cambiar y mejorarse de acuerdo a estas expansiones. Se podría implementar otra técnica de búsqueda, o bien, utilizar una combinación de estas para diseñar una metodología que se adapte de mejor manera al problema planteado.

Referencias

- [1] L. Acosta, J.J. Rodrigo, J.A. Mendez, G.N. Marichal, and M. Sigut. Ping-pong player prototype. *IEEE Robotics & Automation Magazine*, 10(4):44–52, 2003.
- [2] Bastian Solutions. High-speed pick & place robots by bastian solutions. YouTube. <https://youtu.be/6RKXVefE98w>, April 2012. Fecha de consulta: 08-2023.
- [3] Matthew T. Mason. Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):1–28, 2018.
- [4] Richard M Murray, Zexiang Li, and S Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC press, first edition, 1994.
- [5] Elena Garcia, Maria Antonia Jimenez, Pablo Gonzalez De Santos, and Manuel Armada. The evolution of robotics research. *IEEE Robotics & Automation Magazine*, 14(1):90–103, 2007.
- [6] Zakary Littlefield, Shaojun Zhu, Hristiyan Kourtev, Zacharias Psarakis, Rahul Shome, Andrew Kimmel, Andrew Dobson, Alberto F. De Souza, and Kostas E. Bekris. Evaluating end-effector modalities for warehouse picking: A vacuum gripper vs a 3-finger underactuated hand. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1190–1195, 2016.
- [7] Nichola Abdo, Cyrill Stachniss, Luciano Spinello, and Wolfram Burgard. Robot, organize my shelves! tidying up objects by predicting user preferences. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1564, 2015.
- [8] Jue Kun Li, David Hsu, and Wee Sun Lee. Act to see and see to act: Pomdp planning for objects search in clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5701–5707, 2016.
- [9] Giray Havur, Guchan Ozbilgin, Esra Erdem, and Volkan Patoglu. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 445–452, 2014.

- [10] Georgios Georgakis, Arsalan Mousavian, Alexander Berg, and Jana Kosecka. Synthesizing training data for object detection in indoor scenes. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.
- [11] Joaquim Ortiz-Haro, Jung-Su Ha, Danny Driess, Erez Karpas, and Marc Toussaint. Learning feasibility of factored nonlinear programs in robotic manipulation planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3729–3735, 2023.
- [12] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3327–3332, 2007.
- [13] Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000.
- [14] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., USA, 1990.
- [15] J. Kirkegaard and T.B. Moeslund. Bin-picking based on harmonic shape contexts and graph-based matching. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 2, pages 581–584, 2006.
- [16] MIKE STILMAN and JAMES J. KUFFNER. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 02(04):479–503, 2005.
- [17] E.E. Bischoff and M.S.W. Ratcliff. Issues in the development of approaches to container loading. *Omega*, 23(4):377–390, 1995.
- [18] José Fernando Gonçalves and Mauricio G.C. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39(2):179–190, 2012.
- [19] Andreas Bortfeldt and Hermann Gehring. A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1):143–161, 2001.
- [20] A. Moura and J.F. Oliveira. A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4):50–57, 2005.
- [21] Miaojun Yao, Zhili Chen, Linjie Luo, Rui Wang, and Huamin Wang. Level-set-based partitioning and packing optimization of a printable model. *ACM Trans. Graph.*, 34(6), nov 2015.

- [22] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992. Range Image Understanding.
- [23] Alvaro Collet, Dmitry Berenson, Siddhartha S. Srinivasa, and Dave Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In *2009 IEEE International Conference on Robotics and Automation*, pages 48–55, 2009.
- [24] Yu Feng, Yongbo Yang, Zhendong Xu, and Yanchun Liu. An efficiency collision detection algorithm for rigid objects. In *2012 Sixth International Conference on Internet Computing for Science and Engineering*, pages 42–46, 2012.
- [25] Ales Pochyly, Tomas Kubela, Martin Kozak, and Petr Cihak. Robotic vision for bin-picking applications of various objects. In *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, pages 1–5, 2010.
- [26] Rahul Shome, Wei N. Tang, Changkyu Song, Chaitanya Mitash, Hristiyan Kourtev, Jingjin Yu, Abdeslam Boularias, and Kostas E. Bekris. Towards robust product packing with a minimalistic end-effector. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9007–9013, 2019.
- [27] Ming-Yu Liu, Oncel Tuzel, Ashok Veeraraghavan, Yuichi Taguchi, Tim K Marks, and Rama Chellappa. Fast object localization and pose estimation in heavy clutter for robotic bin picking. *The International Journal of Robotics Research*, 31(8):951–973, 2012.
- [28] Jeffrey Mahler and Ken Goldberg. Learning deep policies for robot bin picking by simulating robust grasping sequences. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 515–524. PMLR, 13–15 Nov 2017.
- [29] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018.
- [30] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Adversarial actor-critic method for task and motion planning problems using planning experience. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):8017–8024, Jul. 2019.

- [31] Mike Stilman, Koichi Nishiwaki, Satoshi Kagami, and James J. Kuffner. Planning and executing navigation among movable obstacles. *Advanced Robotics*, 21(14):1617–1634, 2007.
- [32] Wisdom C. Agboh and Mehmet R. Dogar. Real-time online re-planning for grasping under clutter and uncertainty. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–8, 2018.
- [33] Hao Tian, Chaoyang Song, Changbo Wang, Xinyu Zhang, and Jia Pan. Sampling-based planning for retrieving near-cylindrical objects in cluttered scenes using hierarchical graphs. *IEEE Transactions on Robotics*, 39(1):165–182, 2023.
- [34] Aliakbar Akbari, Fabien Lagriffoul, and Jan Rosell. Combined heuristic task and motion planning for bi-manual robots. *Autonomous Robots*, 43(6):1575–1590, Aug 2019.
- [35] Joshua A. Haustein, Isac Arnekvist, Johannes Stork, Kaiyu Hang, and Danica Kragic. Learning manipulation states and actions for efficient non-prehensile rearrangement planning. *arXiv e-prints*, page arXiv:1901.03557, Jan 2019.
- [36] Jennifer E. King, Marco Cognetti, and Siddhartha S. Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3940–3947, 2016.
- [37] Weihao Yuan, Johannes A. Stork, Danica Kragic, Michael Y. Wang, and Kaiyu Hang. Rearrangement with nonprehensile manipulation using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 270–277, 2018.
- [38] Lerrel Pinto, Aditya Mandalika, Brian Hou, and Siddhartha Srinivasa. Sample-efficient learning of nonprehensile manipulation policies via physics-based informed state distributions. *arXiv e-prints*, page arXiv:1810.10654, Oct 2018.
- [39] Jennifer E. King, Joshua A. Haustein, Siddhartha S. Srinivasa, and Tamim Asfour. Nonprehensile whole arm rearrangement planning on physics manifolds. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2508–2515, 2015.
- [40] Rahul Shome, Kiril Solovey, Jingjin Yu, Kostas Bekris, and Dan Halperin. Fast, high-quality dual-arm rearrangement in synchronous, monotone tabletop setups. In Marco Morales, Lydia Tapia, Gildardo Sánchez-Ante, and Seth Hutchinson, editors, *Algorithmic Foundations of Robotics XIII*, pages 778–795, Cham, 2020. Springer International Publishing.

Referencias

- [41] Shuai Han, Nicholas Stiffler, Athanasios Krontiris, Kostas Bekris, and Jingjin Yu. High-quality tabletop rearrangement with overhand grasps: Hardness results and fast methods. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.
- [42] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [43] Mohammed Ouali, Walid Mahdi, and Seyyid Ahmed Medjahed. Performance analysis of simulated annealing cooling schedules in the context of dense image matching. *Computación y Sistemas*, 21(3):493–501, 2017.