UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ADE 20 Años
FACULTAD DE ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS. UPV

Facultad de Administración y Dirección de Empresas
Universitat Politècnica de València

Optimización del problema de asignación de contenedores en una terminal portuaria

TRABAJO FINAL DE GRADO

Grado en Administración y Dirección de Empresas

Autor: Carlos Romero López

Tutoras: Eva Vallada Regalado, Mª Fulgencia Villa Juliá

Curso 2019-2020

Resumen

El puerto tiene una gran importancia económica, pues es el nexo entre mar y tierra, y el medio mediante el cual llegan muchas de las mercancías necesarias para gran cantidad de procesos productivos. A su vez, una gran parte del comercio internacional depende de él.

Así pues, es necesario llevar a cabo tareas de optimización en los procesos que se realizan en la terminal portuaria, siendo el patio de contenedores el principal cuello de botella del sistema, ya que todos los contenedores han de pasar por el mismo. Una terminal con un alto flujo de contenedores se considerará más productiva, y por tanto, se obtendrá un mayor beneficio.

En este Trabajo de Fin de Grado nos centramos en el problema de la asignación de los contenedores a los bloques situados en el patio de contenedores. Para ello propondremos una serie de métodos que ofrecerán distintas soluciones a este problema, así como un modelo matemático para resolver un caso simplificado y facilitar la comprensión del problema.

Analizaremos los resultados obtenidos con los diferentes métodos propuestos, haciendo uso de una serie de instancias generadas aleatoriamente, tratando de encontrar cuál de ellos es el que resuelve el problema de una manera más eficiente.

Palabras clave: Puerto, mar, transporte, asignación, contenedores, algoritmo, investigación operativa

Resum

El port té una gran importància econòmica, ja que és el nexe entre la part marítima i la part terreste, a més de ser el mitjà pel qual arriven moltes de les mercancies necessaries per a una gran quantitat de processos productius. A més, una gran part del comerci internacional depén del mateix.

Així doncs, és necessari dur a terme tasques d'optimització als processos que es realitzen a la terminal portuària, on el pati de contenidors és el principal coll d'ampolla del sistema, ja que tots el contenidors han de pasar pel mateix. Una terminal amb un ample fluix de contenidors es considerarà més productiva, y per tant, obtindrà un major benefit.

En aquest treball de Fi de Grau ens centrarem en el problema d'asignació dels contenidors als blocs situats al pati de contenidors. Proposarem una sèrie de mètodes que oferiràn diferents solucions a aquest problema, així como un model matemàtic per a resoldre un cas simplificat y facilitar la comprensió del problema.

Finalment, analisarem els resultats obtinguts amb el diferents mètodes proposats, fent ús d'una sèrie d'instàncies generades aleatoriament, tractant de trobar el que proporcione la solució més eficient.

Paraules clau: Port, mar, transport, asignació, contenidors, algoritme, investigació operativa

Abstract

Ports are usually very important for economics, since they are the link between sea and land, moreover, lots of industries use this mean to get the goods necessaries to their productive processes. Also, an important part of international trades rely on it.

This is why it is necessary to optimize all the processes in the port, where the yard is one of the main bottlenecks of the system, since all the containers must go thought the yard to get their final destination.

In this Bachelor thesis, we are going to focus on the containers allocation problem on a yard. Thus, a few methods will be purposed to show different solutions to this problem. Also a mathematical model will be used to solve a simplified case.

We will analyse the results we get with these methods, by using some random instances and we will find out which solution is more efficient.

Key words: Port, sea, transport, allocation, containers, algorithm, operational research

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

El transporte de mercaderías por vía marítima ha facilitado las operaciones entre países durante siglos, sin embargo, este tipo de transporte requiere de una infraestructura extensa y compleja para dar cabida a las embarcaciones, cada vez de mayor tamaño, en los distintos puntos de las costas, además de las grandes cantidades de combustible que son necesarias. Sin duda, actualmente, el transporte marítimo es una pieza clave en la economía mundial.

Los puertos convencionales han comenzado una transformación para convertirse en puertos inteligentes, donde se utilicen las nuevas tecnologías disponibles para mejorar la eficiencia y productividad de los servicios proporcionados.

Esta transformación ha de llevar consigo una transición energética hacia energías más limpias, tratar de implementar sistemas digitalizados para sus operaciones, apostar por la innovación e invertir en ello, además de tener un gran cuidado en nuevas áreas, como la ciberseguridad.

Estos cambios de estructuras conllevan una serie de beneficios, como son el aumento de rentabilidad de sus operaciones, la simplificación y automatización de los procesos y la mejora en la calidad de las operaciones.

Entre los puertos inteligentes (*Smartport*), cabe destacar que el plan de la nueva terminal de Valencia hará uso de las nuevas tecnologías, convirtiendo el puerto de Valencia en un *Smartport* [14]

Nuestro trabajo se centra en la gestión y optimización de los procesos en una terminal portuaria con unas características que se asemejen a las del puerto de Valencia, debido a la cercanía a nuestro entorno. Principalmente, nos centraremos en los movimientos de los contenedores, para ello, necesitaremos introducir la estructura general de un puerto.

Un manejo eficiente de la asignación de los contenedores en los patios puede reducir de manera notoria los costes para las empresas y los tiempos de espera de las embarcaciones [20]

1.1 Objetivos

El objetivo fundamental de este Trabajo de Fin de Grado consiste en resolver eficientemente el problema de asignación de contenedores a los bloques en una terminal portuaria. Para ello, hemos de analizar y conocer el funcionamiento de una terminal portuaria. Así pues, podremos diseñar metodologías, que teniendo en cuenta las características y particularidades de la terminal, permitan tomar decisiones respecto a la asignación de contenedores de una manera rápida y racional.

El trabajo se realizará tomando como referencia un entorno cercano a nuestra situación, pues tomaremos como referencia principal la estructura de la nueva terminal del Puerto de Valencia, una configuración europea, concepto que más adelante introduciremos, pues es el tipo de configuración que mejor se adapta a las nuevas tecnologías que se están introduciendo y al nuevo concepto de *Smartport*.

A lo largo del trabajo se realizará la formulación de distintas aproximaciones y modelos algorítmicos que permitan minimizar los tiempos necesarios para las labores de asignación de contenedores. Además, identificaremos el orden de asignación de cada contenedor a cada bloque y realizaremos un análisis y una comparación de los diferentes resultados obtenidos para cada aproximación o modelo utilizado.

Para conseguir los objetivos previamente descritos, seguiremos la siguiente planificación:

- Introduciremos algunos de los puntos más importantes de la evolución del transporte marítimo a lo largo de la historia, lo que nos permitirá conocer el contexto en el cual nos encontramos actualmente y de dónde provienen las estructuras y medios utilizados a día de hoy.
- Identificaremos el impacto medioambiental y económico del transporte marítimo, motivos por los cuales, principalmente, es interesante realizar labores de optimización. Esto avalará la importancia de este trabajo.
- Mostraremos los medios de transporte e infraestructuras necesarias en una terminal portuaria y cuál es su labor dentro de la terminal portuaria, además de la estructura general de la misma e introduciremos el puerto de Valencia, principal referente en nuestro trabajo. Esto permitirá conocer mejor la complejidad del problema y las tareas a optimizar.
- Realizaremos una introducción a los distintos problemas que surgen en una terminal portuaria, los cuales son objeto de investigación. Así, obtendremos una primera visión de la gran complejidad que hay en la gestión de una terminal portuaria.
- Describiremos más a fondo el problema de asignación de contenedores y explicaremos los principales problemas que surgen a la hora de asignar los contenedores a los diferentes bloques del patio. Ofreceremos una serie de soluciones para estos problemas.
- Desarrollaremos una serie de algoritmos que permitan la mejora de los tiempos de asignación de los contenedores a los bloques.
- Analizaremos los resultados e interpretaremos las soluciones obtenidas con el fin de conocer qué modelos algoritmicos nos ofrecen resultados aceptables dentro de los rangos establecidos.
- Realizaremos una serie de conclusiones e introduciremos las futuras líneas de trabajo dentro de este campo de investigación.

1.2 Motivación

Las necesidades en el transporte marítimo se han visto notablemente incrementadas a lo largo de los años. Dada la limitación existente en cuanto a espacio y los requerimientos constantes de materiales y productos para las diferentes industrias, es esencial tratar de reducir los tiempos de parada en el puerto. Con una mejor gestión de la asignación

de los contenedores dentro de la terminal portuaria, podríamos reducir estos tiempos notablemente, lo cual generaría un ahorro importante para las empresas.

A lo largo de los últimos años, se han realizado diversos estudios que tratan sobre diferentes problemas de optimización dentro de la terminal portuaria y se han publicado distintos artículos tratando la problemática de las operaciones en una terminal portuaria, lo que demuestra el creciente interés en este campo.

Determinar cuál es el bloque idóneo para cada contenedor conlleva una serie de ventajas importantes para el rendimiento de una terminal. A su vez, un incremento del rendimiento de una terminal portuaria afecta positivamente a la economía de las ciudades cercanas, ya que favorece al comercio y la industria que rodea esta zona.

La optimización de las operaciones en una terminal portuaria puede extrapolarse a muchos otros campos, resultando un problema muy interesante de trabajar. Debido a la creciente competitividad en las diferentes industrias, muchas empresas consideran necesario ahorrar lo máximo posible en recursos.

Además del ahorro económico que supone la reducción de tiempos para las empresas, cabe destacar la contribución positiva que supone para el medioambiente. Esto puede ser relacionado con los Objetivos de Desarrollo Sostenible.

La introducción de las nuevas tecnologías en el campo de la investigación operativa, en concreto en las operaciones realizadas en el puerto, es un aliciente para la realización de este trabajo, pues permitirá en un futuro aplicar nuestras conclusiones en otros campos.

Académicamente, se aplican conceptos estudiados durante el Grado de Administración y Dirección de Empresas. En concreto, las nociones aprendidas en el campo de Investigación Operativa sustentan principalmente este trabajo. Además herramientras como el *Solver* de *Microsoft Excel* nos han ayudado a la realización del mismo.

Conceptos relacionados con la economía, aprendidos en asignaturas como Economía Española o Economía Mundial nos han servido para conocer la importancia del comercio marítimo, el transporte, la globalización y las relaciones internacionales.

1.3 Estructura del documento

Este Trabajo de Fin de Grado seguirá la siguiente estructura, con el fin de aclarar el contenido del mismo:

- En el primer y actual capítulo se expone cuál es el objetivo del trabajo, cuál ha sido la motivación para la realización del mismo y cómo es la estructura que sigue.
- El segundo capítulo muestra un resumen de la evolución a lo largo de la historia del transporte marítimo, además de cuál es su impacto sobre el medioambiente y la economía.
- En el tercer capítulo, introduciremos la estructura general de una terminal portuaria y definiremos sus principales zonas. Además, mostraremos los elementos principales de la terminal. Introduciremos el puerto de Valencia y su importancia para la ciudad, además de su influencia en nuestro trabajo. Expondremos algunos de los posibles problemas que aparecen dentro de una terminal portuaria.
- El cuarto capítulo presenta una descripción del problema a tratar en el trabajo, el problema de asignación, mostraremos la notación y formulación de una versión

simplificada y desarrollaremos una serie de algoritmos que permitirán aproximarnos a una solución de la versión más realista del problema. Seguidamente mostraremos los resultados de aplicar estos métodos a una serie de instancias aleatorias y se realizará un análisis computaciónal, mostrando la eficiencia de los algoritmos diseñados..

- El quinto capítulo muestra qué conclusiones hemos podido obtener con este trabajo, además de exponer cuáles serán las posibles líneas de investigación futura.

CAPÍTULO 2

La evolución del transporte marítimo y su relación con el medioambiente y la economía

El transporte marítimo surge aproximadamente en el año 3500 a.C., con rutas de corta distancia por las costas para el transporte de pequeñas cargas y personas. Desde entonces, su evolución ha sido constante. Las grandes dimensiones de las embarcaciones actuales hacen que este tipo de transporte tenga, globalmente, una gran influencia económica. Por otra parte, también influye en el medioambiente, pues los combustibles utilizados en estos medios de transporte son realmente contaminantes. Para entrar en el contexto del trabajo, hemos de describir un poco la historia de este tipo de transporte y entrar un poco en detalle sobre su relación con la economía y el medioambiente.

2.1 Historia del transporte marítimo

Desde principio de la historia, las ciudades más importantes se han construido alrededor de las zonas cercanas a ríos o mares principales. Muestra de ello son ciudades como Atenas y Roma, que se fundaron próximas al Mar Mediterráneo o el imperio egipcio, que se fundó siguiendo las orillas del río Nilo.

El transporte marítimo ha hecho referencia desde la antigüedad a la forma utilizada para llevar mercancías o personas de un lugar a otro, separados por amplias zonas de agua, que hacían inviables el uso de otros medios de transporte.

Fue el hecho de que las principales ciudades estuvieran cerca de mares o ríos, lo que propició el desarrollo del transporte marítimo, pues nació una necesidad de transportar tanto personas, como mercancías a lugares alejados, la cual fue creciendo a lo largo de la historia.

La evolución en el transporte supuso una gran ventaja para el desarrollo de la sociedad. Con el paso de los siglos se comenzaron a realizar exploraciones a través de la mar, lo cual sirvió para descubrir nuevos territorios, hasta el momento inexplorados. Nuevas culturas comenzaron a aparecer y, poco a poco, las civilizaciones más importantes fueron imponiendo sus costumbres en los territorios recién descubiertos, consiguiendo ganar poder y convirtiéndose en referentes a nivel mundial. Un ejemplo de esto es la colonización, pues Cristóbal Colón impuso sus costumbres en América, territorio hasta entonces inexplorado. El imperio español abarcaba por aquel entonces diversos territorios, tanto por Europa, como por América.

Con la invención de la máquina de vapor en el siglo XVIII y su posterior incorporación en las embarcaciones, sus dimensiones aumentaron drásticamente, la cantidad de carga que se podía transportar aumentó notablemente, lo que supuso un gran avance en el transporte marítimo. Poco a poco se fue llegando a tamaños extremadamente grandes.

Los puertos comenzaron a tomar importancia en el transporte marítimo, pues las labores de carga y descarga de mercancías, cada vez, se complicaban más. En la actualidad, con la llegada de la Industria 4.0, los puertos están comenzando a digitalizarse, utilizando nuevas tecnologías que permiten aumentar su eficiencia notablemente, llegando a convertirse en puertos inteligentes o *Smartports*. El uso del *Big Data* y el Internet de las cosas ha acompañado esta evolución. Como referencias de *Smartports* en España, podemos destacar los puertos de Barcelona, Vigo, La Coruña y Tarragona [7].

Nuestro trabajo se centra en la importancia que toma el puerto en esta cadena de transporte de mercancías, principalmente en los movimientos de las mercancías que llegan y salen del puerto, y cómo ha influido la era tecnológica en sus procesos.

2.2 El impacto mediambiental del transporte marítimo

Los hábitats de distintas especies marítimas se ven afectados debido a las construcciones de los puertos, pues se está tomando terreno al mar para realizar una obra extensa. Pese a ello, hoy en día es impensable tratar con las operaciones entre países sin hacer uso del transporte marítimo, por lo que han de estudiarse métodos eficaces para reducir la huella ecológica de estas construcciones.

Cada vez son mayores las exigencias relativas al tamaño y capacidad de los barcos. Así pues, cuanta mayor capacidad tenga un barco, menor será el impacto que genere el combustible por tonelada transportada. Ya a mediados del año 2013, comenzó a circular por los mares una nueva clase de buques portacontenedores, el Triple E, con una capacidad de aproximadamente 18.000 contenedores. Con estas medidas podemos observar que las dimensiones en aquellas fechas ya eran inmensas y con el paso de los años han ido aumentando [1].

A su vez, el funcionamiento de los barcos requiere de grandes cantidades de combustible, tanto para su movimiento, como mientras se encuentran atracados en puerto, pues las embarcaciones no detienen los motores en ningún momento durante las operaciones. A esto debemos sumar el rastro de combustible que se genera con el movimiento de las embarcaciones por el mar, lo cual resulta muy nocivo para las distintas especies que se encuentran en el fondo del mar. Durante los últimos años, las empresas se han centrado en la eficiencia de las operaciones, lo que está directamente relacionado con la minimización del consumo de combustibles y la cantidad de gases de efecto invernadero que se generan. La eficiencia en las operaciones de las embarcaciones está muy ligado al tiempo que la embarcación pasa en el mar y el tiempo que permanece en el puerto [10].

Con el paso del tiempo, comienza a ser más habitual tratar de buscar soluciones lo más ecológicas posibles.

Durante los últimos años se ha visto como en diversas ciudades costeras con puerto, han tratado de implementar un sistema de economía circular, como es el caso de Rotterdam, uno de los puertos más grandes de Europa, el cual también es uno de los más sostenibles. Como podemos ver en [12], en este puerto se han comenzado a adoptar una serie de medidas para realizar su transición a un puerto sostenible, además de seguro:

- Gestión del riesgo de inundaciones: Han tratado de adaptar sus instalaciones al nivel creciente del mar en Ámsterdam, situándolo unos metros por encima para evitar posibles futuros problemas.

- Descuento para embarcaciones limpias: Los buques que cumplen los estándares legales en cuanto a materia de medioambiente cuentan con un descuento del 20 % de las cuotas portuarias.

- Naturaleza alrededor del puerto: La zona portuaria de la ciudad de Rotterdam cuenta con aproximadamente 20 hectáreas de naturaleza, la cual se mantiene gracias a las autoridades portuarias.

- Rastreo de emisiones: se han instalado una serie de sensores con la finalidad de controlar los olores en la terminal.

- Energía solar y eólica: Se ha dedicado un espacio para placas solares y energía eólica, lo cual proporciona una parte importante de la energía utilizada.

- Red de tuberías: con el fin de aprovechar el calor desprendido, se han instalado tuberías que permiten reutilizar esta energía en industria, invernaderos e, incluso, hogares.

- Almacenaje de CO_2: se ha construido una red de tuberías para conseguir almacenar todas estas emisiones de CO_2 y evitar su impacto en el medioambiente.

- Luces LED: se han comenzado a utilizar luces LED, lo cual llevará a un ahorro de aproximadamente un 50 % de electricidad.

- Huella de carbono: han comenzado a utilizar tanto embarcaciones, como vehículos híbridos, lo que permitirá reducir la huella de carbono.

Sin duda alguna, el puerto de Rotterdam es un claro ejemplo a seguir. Las autoridades portuarias de diversas ciudades comienzan a intentar imitar este tipo de iniciativas y comienzan a controlar sus emisiones de CO_2, preocuparse por la naturaleza que rodea los puertos y comenzar la transformación hacia puertos más sostenibles.

En España también existen diversos puertos que han apostado por la digitalización y la optimización de procesos, con el beneficio para el medio ambiente que esto ha supuesto. Ejemplo de ello son los puertos de Barcelona, Vigo, Tarragona, La Coruña y Sevilla. En el puerto de Barcelona se ha creado una herramienta llamada *Ecocalculadora* que permite obtener la ruta que menor impacto medioambiental produzca para realizar los envíos.

Siguiendo la línea del trabajo, en referencia al puerto de Valencia, cabe destacar que durante los últimos años han consolidado una serie de políticas de sostenibilidad. Gracias a ello han conseguido reducir un 17 % la huella de carbono. Además, entre sus planes de futuro, están una serie de proyectos los cuales cuentan con fuentes de energía renovables y permitirían que el puerto de Valencia se convirtiera en un puerto autosuficiente y con cero emisiones en el año 2023. Vehículos híbridos y paneles fotovoltaícos son ejemplos de los cambios que se están llevando a cabo [13].

Si además de realizar acciones que apoyen los Objetivos de Desarrollo Sostenible, se optimizan las labores que se realizan en el puerto, se conseguirá reducir notablemente el impacto medioambiental, por ello consideramos de gran importancia trabajar en la eficiencia de las operaciones.

2.3 El impacto económico del transporte marítimo

Debido a la globalización, cada vez es más habitual transportar toda clase de mercancias alrededor del planeta, lo cual ha supuesto un reto para el transporte.

Es muy común que un producto se fabrique en un lugar del mundo diferente de donde provienen las materias primas. También, que el consumidor final esté localizado en un tercer lugar, también alejado. Las distancias son extradamente grandes y requieren de transportes lo más rápidos posibles.

Además de la distancia, las cantidades de materia transportadas aumentan por momentos y se requiere de grandes espacios para ello. El transporte marítimo ofrece la capacidad necesaria para tal cantidad de mercancías, por lo que durante los últimos años ha crecido notablemente [6].

Es común que ciudades con gran importancia se hayan desarrollado alrededor de puertos, algo que afecta directamente a la economía de estas ciudades. Los puertos han generado empleo y riqueza en su entorno. Las ciudades con puerto han conseguido atraer inversiones y empleo, ya sea directamente en el puerto o en industrias dependientes de él, que muchas veces se han situado en zona próximas. Contar con un puerto permite obtener bienes a precios bajos y facilita las exportaciones a su vez [4].

Estudios, como [2], han demostrado analíticamente, con una muestra de 562 regiones europeas, que existe evidencia de que los puertos marítimos tienen un impacto positivo en el empleo de las ciudades donde se sitúan. Se demostró pues, que en una región con 1 millón de empleos, al producirse un aumento de 1 millón de toneladas en las mercancias con las que trabajan, se incrementarían los empleos entre 400 y 600.

Con los nuevos avances en las tecnologías se ha conseguido incrementar la productividad de los puertos más importantes y reducir los costes operativos, lo que conlleva a unos márgenes más elevados. Es por ello que la tendencia hacia la digitalización de la terminal es clave para la economía de las ciudades con puerto. Siguiendo esta línea, este Trabajo de Fin de Grado pretende aportar herramientas que permitan mejorar la productividad de las terminales, y por tanto, la eficiencia tanto económica como medioambiental.

Un puerto activo permite atraer inversión, y esto, en materia económica, conlleva muchos beneficios para las ciudades que lo rodean. Así pues, un mayor rendimiento del puerto implica un mayor impacto económico para la ciudad. Por ello consideramos de gran importancia la optimización de las operaciones en la terminal portuaria.

CAPÍTULO 3
La terminal portuaria

La terminal portuaria juega un papel fundamental en el transporte y comercio marítimo. Es el lugar por el que todas las mercancías han de pasar, sin excepción. La complejidad de las operaciones que se llevan a cabo en una terminal es muy grande. Para entender mejor los problemas que surgen, hemos de definir una serie de conceptos y estructuras. Las tecnologías avanzan y, gracias a ello, se consigue una mejora en la eficiencia de los procesos. Por ello, es importante introducir la figura del puerto de Valencia, el cual cuenta con un plan de desarrollo de una nueva terminal. Esta nueva terminal contará con nuevas tecnologías, que la acercarán al concepto de *Smartport*.

3.1 Estructura

Dentro de una terminal portuaria cabe diferenciar entre la zona de mar y la zona de tierra, las cuales están conectadas entre sí mediante un patio de contenedores. Nuestro trabajo se centrará, principalmente, en el patio de contenedores. En la figura 3.1 podemos observar la estructura general de una terminal portuaria:

Así pues, distinguiremos las siguientes partes:

- Zona de mar o *Sea-side*: se trata de la parte de la terminal donde se encuentra la zona de atraque y las grúas dedicadas a la carga y descarga de contenedores de las embarcaciones que atracan en el puerto. Es ahí donde se cargan y descargan los contenedores, los cuales son, o bien tomados de los puntos de entrada/salida o bien son llevados a ellos.
- Zona de almacenamiento o *Stacking Area*: es la zona dedicada al depósito de los contenedores que llegan al puerto, bien por mar o bien por tierra. Como vemos en la figura 3.1, es en esta parte donde se encuentran localizados los bloques donde se colocan los contenedores, las grúas que permiten su movimiento dentro de los bloques y los puntos de entrada/salida conectados con las zonas de mar y tierra. Es la unión de la zona de mar con la zona de tierra y, una parte fundamental, tanto en la terminal portuaria, como para nuestro trabajo, pues es una de las zonas donde más labores de optimización deben hacerse.
- Zona de tierra o *Land-side*: es la parte interior de una terminal, la que está en contacto con el transporte interior. Es ahí donde se realizan las cargas y descargas de mercancías en los distintos medios de transporte utilizados, ya sean camiones o ferrocarriles.

En la zona de almacenamiento se encuentra el patio de contenedores, el cual está formado por una serie de bloques en los que se colocan los contenedores. Estos conte-

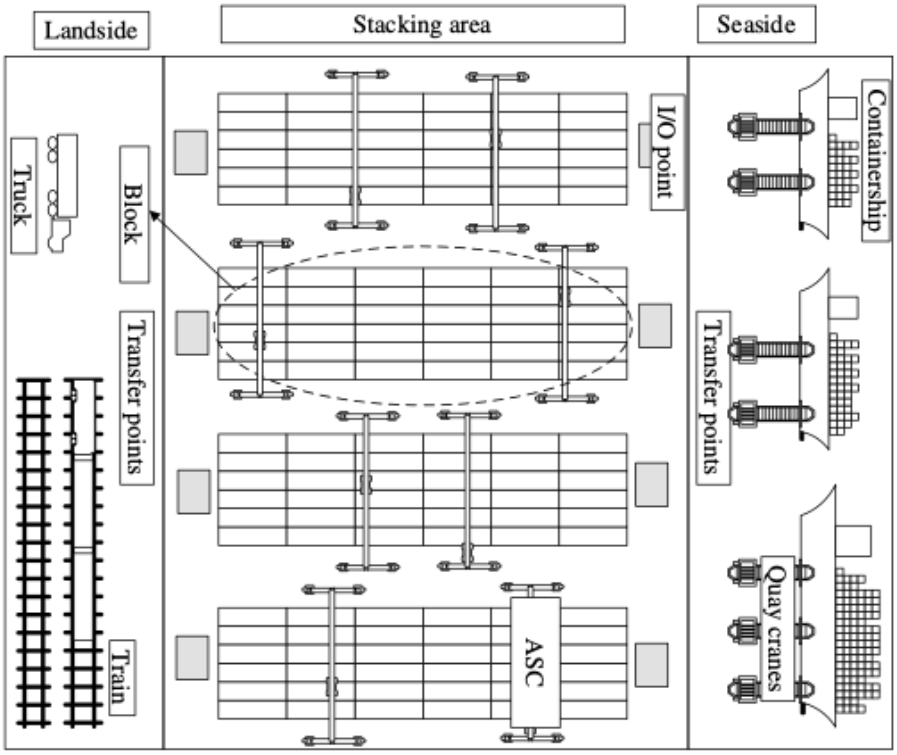

Figura 3.1: Estructura de una terminal portuaria
Fuente: htpp://soa.iti.es

nedores pueden ser apilados unos encima de otros y requieren del uso de grúas para su colocación. Estos bloques siguen una estructura BAROTI normalmente, pues cuentan con distintas bahías, filas y alturas.

Los patios de contenedores pueden presentar distintas configuraciones, asiática o europea. La principal diferencia que existe entre las dos configuraciones es la localización de los puntos de entrada y salida de los contenedores al bloque (*input/output points*).

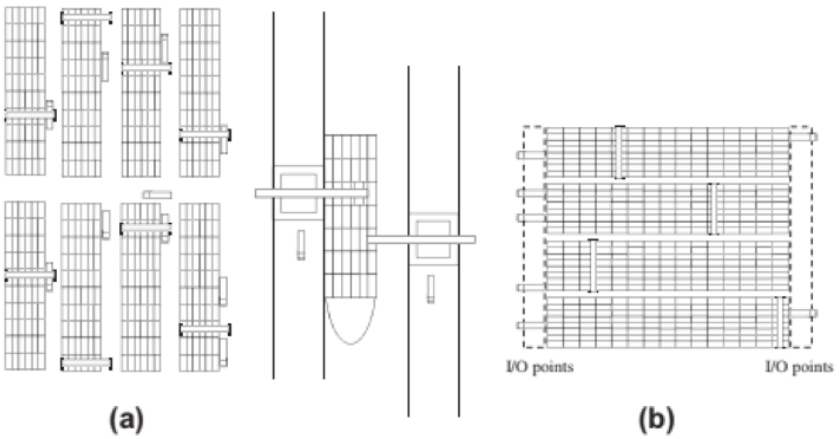

Figura 3.2: Configuración asiática (a) y configuración europea (b)
Fuente: (Carlo, Vis & Roodbergen, 2014)

En la figura 3.2 podemos observar la diferencia entre la configuración asiática (a) y la configuración europea (b).

En primer lugar, la configuración europea, la cual, como vemos en la parte derecha de la figura 3.2 cuenta con bloques perpendiculares al muelle. Los puntos de entrada y salida de los contenedores se sitúan en los finales de los bloques de almacenaje, tanto en el lado de la mar, como en el lado de tierra. Como mencionaremos más adelante en la sección 3.3 y podemos ver en la figura 3.5, la nueva terminal de Valencia contará con una terminal con esta estructura.

En segundo lugar, la configuración asiática que, como vemos en la parte izquierda de la figura 3.2, cuenta con bloques paralelos al muelle y en cada bloque hay reservada una fila que desempeña la función de pista para el movimiento de los camiones y demás vehículos de transporte. En la figura 3.3 podemos ver una imagen de la terminal portuaria de Shanghai, una de las más importantes del mundo, la cual tiene una configuración asiática. Siguiendo con la referencia que hemos tomado durante todo el trabajo, vemos otro ejemplo de configuración asiática es la actual terminal de Valencia en la figura 3.4.

Actualmente la tendencia está siendo las terminales con configuración europea. Es debido a ello y a la situación de la ciudad de Valencia, ciudad la cual hemos tomado como referencia en todo momento, por lo que nuestro trabajo va a centrarse en la configuración europea. La terminal portuaria de Valencia es la más cercana a nuestro entorno y próximamente se verá ampliada con una nueva terminal que contará con tecnologías punteras que permitirán la digitalización y optimización de los procesos. En la figura 3.5 podemos ver una simulación de lo que será nueva terminal del puerto de Valencia, la cual contará con una configuración europea.

Figura 3.3: Terminal portuaria de Shanghai.
Fuente: http://ingenieriaporelmundo.blogspot.com/

Optimizar los procesos en los puertos es muy importante debido a la limitación de espacio para almacenar contenedores, además del coste que ocasiona contar con una embarcación a la espera en puerto, lo cual, como hemos dicho antes, no sólo tiene en cuenta el coste económico, sino también el coste medioambiental.

En este trabajo nos centraremos en la zona de almacenamiento de una terminal con configuración europea, especialmente en el patio, el cual es el principal cuello de botella de una terminal portuaria, pues, como hemos visto previamente, cada vez son mayores las capacidades de los buques portacontenedores, por lo que llega un mayor número de contenedores a puerto, siendo el espacio una limitación. Así pues, es crucial organizar los contenedores de manera eficiente, y para ello tendremos que tener en cuenta distintos factores.

Figura 3.4: Puerto de Valencia en la actualidad.
Fuente: https://www.valenciaport.com

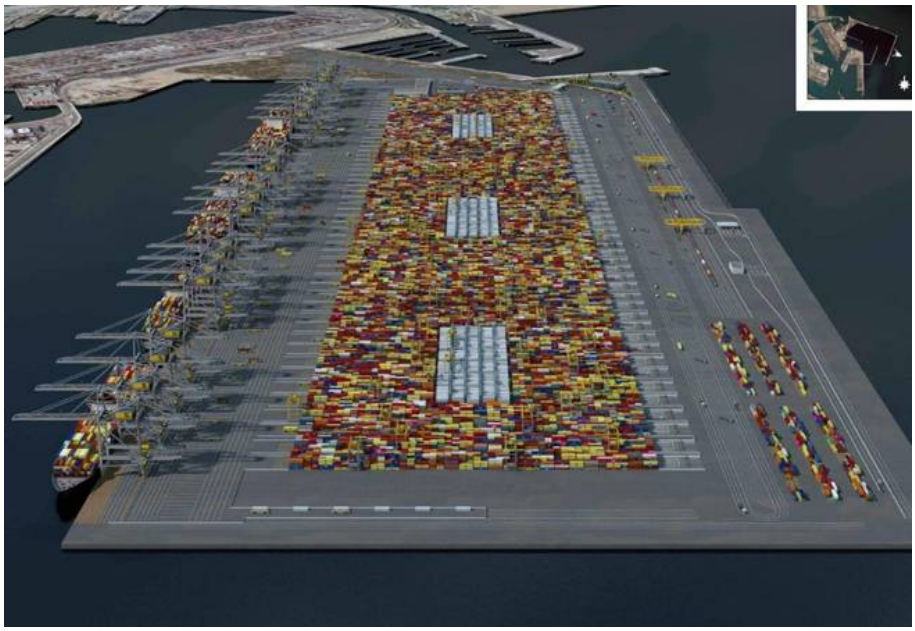

Figura 3.5: Simulación de la nueva terminal del puerto de Valencia.
Fuente: https://www.elconfidencial.com

Las operaciones realizadas en los puertos y, en especial, las realizadas en el patio, son determinantes a la hora de cuantificar la competitividad de un puerto, pues, como hemos visto previamente, las exigencias son cada día mayores. Así pues, un manejo eficiente de la ordenación de los contenedores en los patios puede reducir de manera notoria los costes para las empresas y los tiempos de espera de las embarcaciones [20].

3.2 Elementos

Una vez expuesta la estructura general de una terminal portuaria, hemos de presentar los elementos principales de la misma. A continuación veremos los más importantes.

Contenedores

Los contenedores son la parte más importante de los procesos en una terminal portuaria. Todas las operaciones que se realizan en el puerto tratan de conseguir la mayor eficiencia en sus movimientos. Distinguiremos tres tipos de contenedores: import, export y transshipment, en la figura 3.6 vemos un esquema de los movimientos que realizan cada uno de ellos.

- Contenedores import: Son aquellos que llegan por la parte de mar y se almacenan en el patio de contenedores a la espera de ser recogidos por los distintos medios de transporte terrestre.
- Contenedores export: Son aquellos que llegan por la parte de tierra y se almacenan en el patio de contenedores a la espera de ser cargados a las embarcaciones correspondientes.
- Contenedores transhipment: Son aquellos que llegan por la parte de mar y son temporalmente almacenados en el patio de contenedores, a la espera de ser cargados en otra embarcación.

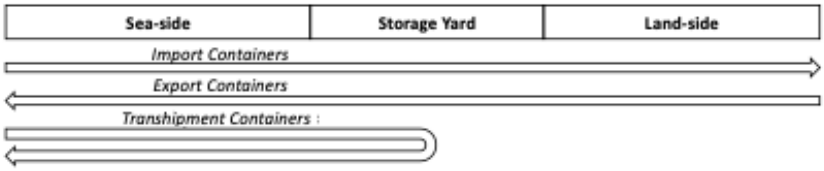

Figura 3.6: Movimientos de los contenedores
Fuente: Elaboración propia

Nuestro trabajo consistirá en decidir cuál será el bloque, elemento que definiremos a continuación, más adecuado para cada contenedor dependiendo de una serie de factores.

Para poder operar con estos contenedores de manera adecuada, los puertos cuentan con distintos elementos para su transporte: grúas y vehículos de transporte.

Bloques

En una terminal portuaria, los contenedores se agrupan en bloques, que cuentan con diferentes filas, bahías y alturas. Nuestro trabajo se centra en asignar los contenedores a estos bloques.

Así pues, en la figura: 3.7 podemos ver la estructura de un bloque de contenedores de una terminal portuaria. Vemos que un bloque está compuesto por una serie de filas, bahías y alturas.

Según [11], normalmente, un bloque cuenta con 40 bahías, 7 filas y entre 3 y 6 alturas, en función de la altura que pueda alcanzar la grúa que se encargue de depositarlos.

La decisión a la hora de asignar un contenedor a un bloque está condicionada por diversos factores. Así pues, una vez se ha colocado en un bloque, no se podrá cambiar.

Uno de los posibles factores a tener en cuenta es la ocupación del bloque, pues será adecuado tratar de equilibrar la congestion de los mismos. Estudios como [15] han tratado el tema, concluyendo que en función de la cantidad de mercancías que pasen por la terminal a diario, será más o menos beneficioso que los bloques tengan una mayor ocupación para el rendimiento de la terminal.

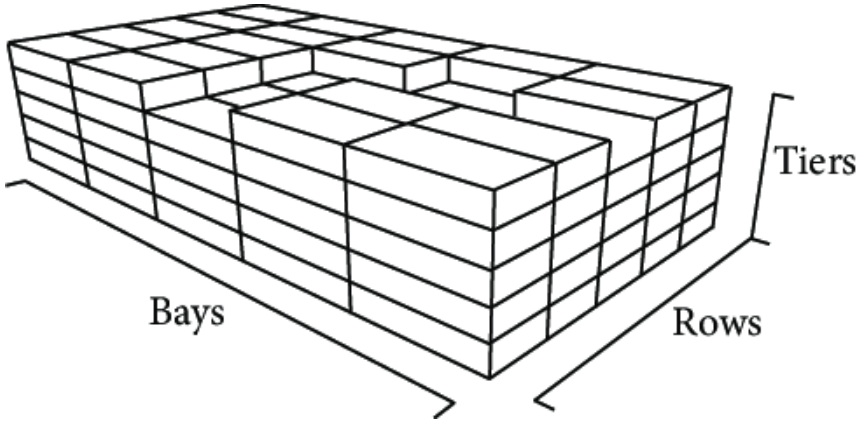

Figura 3.7: Estructura de un bloque
Fuente: https://www.researchgate.net

Realizar movimientos con los contenedores una vez están colocados en un bloque resulta muy complejo, por lo que es muy importante realizar una correcta asignación de los mismos.

Grúas

Las grúas son las encargadas de realizar los movimientos de los contenedores desde las embarcaciones hasta los vehículos de transporte y a través de la terminal.

Distinguiremos principalmente dos tipos de grúas: las grúas pórtico o *yard cranes* y las grúas de muelle o *quay cranes*. Las grúas pórtico son utilizadas para el manejo de las mercancías y el almacenamiento dentro de la terminal. Son las más frecuentemente utilizadas y existen dos tipos, por un lado, las que cuentan con neumáticos de goma y, por otro lado, las que realizan sus movimientos siguiendo raíles. Las primeras cuentan con una mayor flexibilidad de movimientos, no son automáticas y pueden realizar giros de 90 grados, mientras que las segundas, limitan su movimiento a los raíles por donde se mueven. En la figura 3.8 vemos la estructura de una grúa pórtico con raíles.

En segundo lugar, las grúas de muelle, también llamadas *quay cranes*, que son las que se utilizan para cargar y descargar los contenedores de las embarcaciones. Normalmente son manejadas por un operario. En la figura 3.9 podemos ver un ejemplo de grúa de muelle.

Vehículos de transporte

En cuanto a los vehículos de transporte utilizados en una terminal portuaria, por un lado podemos distinguir aquellos que se centran en la parte terrestre y el intercambio de mercancías en dicha parte, como serían los camiones o ferrocarriles.

Por otro lado, contamos con unos vehículos para el transporte de contenedores en el interior de la terminal, donde cabe diferenciar entre aquellos que no pueden levantar las cargas, como serían los vehículos de guiado automático (AGV) o camiones con trailer y los que sí pueden levantar las cargas, como serían los *straddle carriers* (SC), vehículos dirigidos por operarios, cuya automatización es posible y se da en lo que serían los vehículos de elevación automatizados (ALV).

En la figura 3.10 mostramos un ejemplo de vehículos de guiado automático, mientras que en la figura 3.11 podemos ver una *straddle carrier*.

Figura 3.8: Grúa pórtico con raíles.
Fuente: www.noticiasmaquinaria.com

Figura 3.9: Grúa de muelle.
Fuente: www.nst.com.my

Generalmente se utilizan vehículos automatizados guiados (AGVs) para realizar los movimientos de los contenedores en el lado del mar y camiones externos para realizar los movimientos en el lado de la tierra. Estas configuraciones se comenzaron a utilizar en puertos europeos, de ahí su nombre. [3].

Figura 3.10: Vehículos de guiado automático.
Fuente: www.marineinsight.com

Figura 3.11: Straddle carrier.
Fuente: www.seanews.com.tr

3.3 El puerto de Valencia

A día de hoy, las actividades realizadas en el puerto de Valencia, representan una gran parte de la actividad económica de la ciudad. Y no sólo eso, sino que del puerto de Valencia dependen diversas provincias españolas. Esto se puede ver con las estadísticas recogidas, pues el año 2019, pasaron por el puerto de Valencia más de 81 millones de toneladas en mercancías. Además, tiene conexión con más de 1.000 puertos en todo el mundo, destacando las relaciones que mantiene con China y Estados Unidos.

Durante abril de 2020 un total de 352 empresas operaban diariamente en el puerto de Valencia. Se ha conseguido convertir en uno de los puertos más importantes de España, lo cual se puede ver reflejado en que aproximadamente un 60 % de las mercancías que se importan y exportan en España, pasan por Valenciaport. Hablando de cifras concretas, el año pasado pasaron por el puerto de Valencia un total de 81 millones de toneladas de mercancías, 5,4 millones de contenedores.

El puerto de Valencia tiene conexiones establecidas por todo el mundo, desde Estados Unidos hasta China, pasando por diversos países europeos. Anualmente se intercambian alrededor de 8 millones de toneladas con Estados Unidos y cerca de 6,5 millones de toneladas con China [8]

En la tabla de la figura 3.12 podemos ver el impacto económico que general la actividad de Valenciaport, tanto en cifras absolutas como en cifras relativas respecto a la Comunidad Valenciana.

	Inicial	Directo	Indirecto	Inducido	Totales	Total+ Inicial	% s/Comunidad Valenciana
Salarios Brutos	401.895	452.311	99.379	289.740	841.430	1.243.325	2,62%
Beneficio Bruto	293.158	410.797	94.931	277.339	783.067	1.076.225	2,35%
Ingresos Fiscales	-12.425	98.601	24.579	69.618	192.798	180.373	1,60%
VABpm	682.626	961.709	218.889	636.697	1.817.295	2.499.921	2,39%
Empleos*	7.304	16.367	3.815	11.380	31.562	38.866	2,09%

Figura 3.12: Impacto económico obtenido por la actividad de Valenciaport en miles de euros.
Fuente: https://www.valenciaport.com

El puerto de Valencia cuenta con una gran importancia en la Unión Europea, y, como vemos en la figura 3.13, fue uno de los diez puertos con mayor tráfico de contenedores en 2018, junto con Barcelona y Algeciras.

Figura 3.13: Puertos de europa con mayor tráfico en miles de TEU en 2018
Fuente: http://www.canaryports.es

Aumentar el rendimiento de un puerto es un punto clave para conseguir un buen funcionamiento, sobre todo cuando se trata de un puerto con gran importancia, como es el de Valencia. Así pues, en este trabajo expondremos una serie de soluciones adaptables a las necesidades operativas del puerto de Valencia.

La nueva terminal portuaria de Valencia generará alrededor de 15.000 nuevos empleos, además de los cerca de 30.000 empleos que se esperan generar debido a la creciente actividad del puerto [17]. La configuración que se utilizará en la nueva terminal del Puerto de Valencia será una configuración europea.

Recientemente, como hemos mencionado en la sección 2.1, debido a la aparición de la Industria 4.0 y la logística inteligente en los puertos, la tendencia está siendo hacia los puertos inteligentes, también llamados *Smartports*. Valencia será un ejemplo de ello, es por ello que se ha decidido trabajar con una configuración europea, pues se adapta mejor a las necesidades de los *Smartports*. Entre sus ventajas, podemos destacar que con la configuración europea se consiguen mayores velocidades en las grúas y tiene unos costes operativos menores, además de una mayor capacidad en los bloques, debido a que no hay ninguna fila destinada únicamente al movimiento de los camiones. Tras el desarrollo de su nueva terminal comenzará a hacer uso de nuevas tecnologías para la gestión de las operaciones en su terminal.

Sumado a la importancia de la parte económica, hay que tener en cuenta que un buen proceso de optimización de los movimientos en las operaciones portuarias podrá convertirse en una reducción importante de los gases de efecto invernadero producidos. Estos gases son producidos por el combustible utilizado por los barcos mientras permanecen en la terminal durante sus labores de carga y descarga.

La Autoridad Portuaria de Valencia (APV) apostó en 2011 por un modelo sostenible medioambientalmente y, para ello, propuso un plan para minimizar los impactos negativos que pueden derivar de su actividad en la calidad de las aguas, del aire y del ruido [16]. En relación con lo expuesto en el punto 2.2 y según vemos en [18], durante el periodo transcurrido entre 2008 y 2016, en el puerto de Valencia se consiguió reducir la huella de carbono en un 19 %, haciendo, a su vez, aumentar el tráfico de mercancías en un 24 %. Además, se produjo una reducción del gasto energético de un 27 %. En la figura 3.14 podemos observar la estructura actual del puerto de Valencia.

En julio de 2020, el Jefe de Planificación Estratégica e Innovación de la APV anunció la elaboración de un Plan Estratégico para el Puerto de Valencia con una duración hasta 2030, siendo revisado en 2025. Este plan contará con una cifra de 511.514 euros y su puesta en marcha se realizará alrededor de noviembre de 2020. El Plan contará con distintas líneas, entre ellas la transformación energética con nuevos combustibles y la construcción de un ferrocarril que refuerce el transporte hacia el interior.

La intención del Jefe de Planificación de la APV es realizar acciones que favorezcan la transformación digital del Puerto de Valencia y convertir el actualmente llamado *Valenciaport* en *Valencia Smartport* [5].

Figura 3.14: Estructura del puerto de Valencia
Fuente: Las Provincias: "El coronavirus ocasionará una «importante» caída del tráfico portuario valenciano, según la estiba"

3.4 Problemas de optimización en una terminal portuaria

Las operaciones realizadas en los puertos y, en especial, las realizadas en el patio son determinantes a la hora de cuantificar la competitividad de un puerto. Minimizar los tiempos en las operaciones de una terminal portuaria, hará que ésta sea más competitiva y generará un impacto positivo.

La optimización de los procesos en los puertos es muy importante debido a la limitación de espacio para almacenar contenedores, además del coste que ocasiona contar con una embarcación a la espera en puerto, lo cual no sólo tiene en cuenta el coste económico, sino también el coste medioambiental.

Entre los distintos problemas que pueden surgir en una terminal portuaria, podemos destacar:

- Planificación del atraque: consiste en asignar un tiempo y posición a las embarcaciones que llegan al puerto.
- Asignación de grúas: se trata de asignar una cantidad de grúas de muelle a cada embarcación, con la finalidad de minimizar los movimientos realizados y tiempos de espera en el puerto.
- Asignación de contenedores: los contenedores que llegan a puerto han de ser llevados al patio de contenedores, donde se decidirá en qué bloque hay que colocarlos.
- Localización de contenedores: una vez decidido a qué bloque llevar el contenedor, se ha de decidir en que posición exacta del bloque hay que colocarlo.

Como vemos, la problemática dentro del puerto es muy amplia, hay multitud de opciones y se ha de intentar minimizar los tiempos en las operaciones realizadas, pues, una reducción de tiempos, supone en la mayoría de casos, un aumento del beneficio.

En nuestro trabajo, trataremos el problema de asignación de contenedores a bloques, un problema general de flujo, centrándonos principalmente en una terminal con configuración europea.

CAPÍTULO 4
El problema de asignación de contenedores

En este capítulo mostraremos el problema de asignación de contenedores a los distintos bloques del patio de una terminal. Para comenzar, en la sección 4.1 introduciremos el problema, sus características y la notación utilizada. En la sección 4.2 introduciremos una versión simplificada del problema de asignación. Seguidamente, en la sección 4.3, se describirán los algoritmos utilizados para resolver el problema de asignación y se utilizarán para resolver unas situaciones sencillas. Finalmente, en la sección 4.4 analizaremos el comportamiento de los algoritmos propuestos aplicándolos a una batería de problemas diseñada para tal fin.

4.1 Descripción del problema de asignación

4.1.1. Características del problema de asignación

El problema a tratar es el problema de asignación de contenedores a los diferentes bloques de una terminal portuaria. Trabajaremos con terminales cuyo patio de contenedores tenga una configuración europea. Hablaremos de una asignación adecuada cuando se consiga un coste temporal del proceso bajo en comparación con otras formas de asignar los contenedores. Así pues, cada contenedor que llega al puerto, tanto por tierra, como por mar, ha de ser asignado a uno de los bloques disponibles dentro del patio de contenedores de la terminal portuaria. Este proceso necesita un tiempo determinado, pues ni todos los contenedores están disponibles a la vez, ni tardan lo mismo en ser trasladados a los diferentes bloques.

Los contenedores permanecerán en los bloques durante un tiempo determinado, hasta que sean transportados hacia su próximo destino, ya sea por vía marítima, como por vía terrestre. Esta decisión ha de ser tomada con precaución, pues una vez colocado el contenedor, no es posible cambiar de bloque.

Trataremos de asignar los contenedores de una manera eficiente, esto es, intentando minimizar el tiempo total del proceso, teniendo en cuenta distintas caracteristicas, como si su siguiente destino es tierra o mar.

Además de minimizar los costes temporales, una buena asignación puede facilitar abordar otros problemas directamente relacionados, como puede ser el problema de localización de contenedores dentro de un bloque.

Generalmente, se dispone de una serie de datos respecto a los contenedores que llegarán. Esta información muestra, normalmente, datos como el tiempo de llegada previsto,

su próximo destino o el coste temporal de asignar el contenedor a cada bloque. Esto permite a los agentes de la terminal portuaria realizar planes de asignación.

Por lo general, una terminal portuaria tiene un esquema similar al de la figura 4.1. En ella existen tres partes diferenciadas: una zona de mar (*Sea*), una zona de tierra (*Land*) y el patio de contenedores (*Yard*). La configuración del patio de contenedores con la que vamos a trabajar es europea, por tanto, existen distintos bloques de contenedores, los cuales tienen dos puntos de entrada y salida de contenedores (*I/O Point*), uno en la parte más cercana al lado de mar y otro en la más cercana al lado de tierra, mostrados de color naranja y morado, respectivamente. Por un lado, los contenedores que llegan por la parte de mar, import o transhipment, son descargados de las embarcaciones y se decide a qué bloque se destinan. Una vez decidido el bloque, se colocan en los vehículos internos correspondientes para su transporte hacia el bloque que se ha decidido, depositándolos en el *I/O Point* más cercano a la parte de mar. Por otro lado, a los camiones externos que llegan por la parte de tierra se les indica a qué bloque han de llevar su mercancía, depositando los contenedores del tipo export en el *I/O Point* más cercano a la parte de tierra del bloque indicado. Una vez los contenedores han llegado al *I/O Point* permanecen temporalmente a la espera de que la grúa esté libre y pueda colocar los contenedores dentro de los bloques.

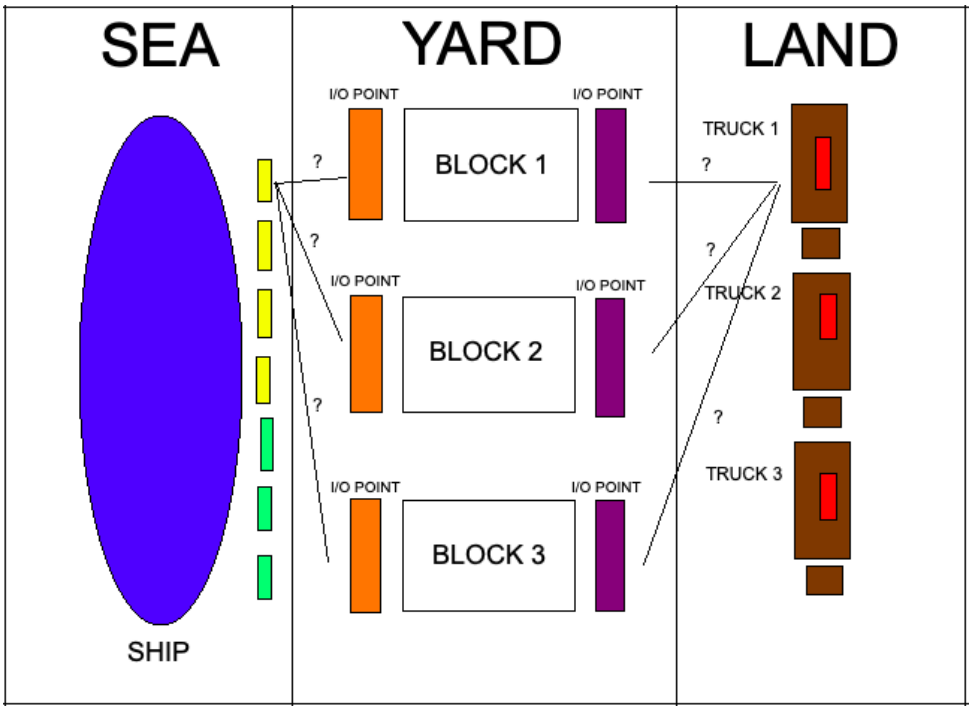

Figura 4.1: Flujo de contenedores en una terminal con sus diferentes parte dentro del patio. Figura de elaboración propia

Nuestro trabajo partirá de la premisa de que no todos los contenedores se encuentran disponibles en un mismo momento del tiempo, sino que llegan en distintos instantes. Comenzaremos a asignar contenedores a medida que vayan estando disponibles en la terminal. Con cada asignación incrementaremos el tiempo total del proceso de asignación, según nuestra elección del bloque.

Partiremos de una serie de datos iniciales. En concreto conoceremos la estructura de los bloques y el espacio libre en el momento de iniciar el proceso. También tendremos datos sobre los instantes de salida de algunos contenedores de cada uno de los bloques, contenedores que estaban ya dentro del bloque antes de comenzar el proceso. Dichas salidas aumentarán la capacidad libre de los bloques.

Aparte, tendremos información de todos los contenedores a asignar, como su instante de llegada o su instante de salida. Conocemos el lugar por el que llegan a la terminal y cuál será su próximo destino (tierra o mar). Gracias a ello, podemos saber qué tipo de contenedores son: import, export o transhipment.

La asignación de contenedores se llevará a cabo por orden de llegada a la terminal, así pues, los contenedores serán organizados en una lista ordenada de menor a mayor instante de llegada. El proceso comenzará cuando el primer contenedor de la lista esté disponible y finalizará cuando el último de los contenedores haya sido asignado a un bloque.

La finalidad de este proceso es asignar todos los contenedores de la lista en el menor tiempo posible.

4.1.2. Notación empleada

Lo primero de todo, antes de formular nuestras propuestas algorítmicas, será definir cuál es la notación que seguiremos, lo cual se muestra en la tabla 4.1, donde se pueden ver los principales elementos que formarán parte de nuestro problema.

Notación	Significado
N	Conjunto de contenedores para asignar: 1,2,...,N1,...,N2...,N. de 1 hasta N1 son import, de N1+1 hasta N2 son export y de N2+1 hasta N serán transhipment.
B	Conjunto de bloques disponibles: 1,2,...,B.
i	Identifica contenedor $i \in N$
j	Identifica bloque $j \in B$
C_i	Identifica la parte por la que entra el contenedor i a la terminal. Siendo 1 si entra por mar y 2 si entra por tierra.
r_i	Instante de llegada del contenedor i.
d_i	Instante de salida del contenedor i.
D_i	Próximo destino del contenedor i, siendo 1 si es mar y 2 si es tierra.
LT_{jt}	Capacidad libre del bloque j en el lado de tierra en un instante t
LM_{jt}	Capacidad libre del bloque j en el lado de mar en un instante t
T_{ij}	Tiempo en asignar un contenedor i al bloque j

Tabla 4.1: Notación utilizada para la formulación del modelo.
Tabla de elaboración propia

Teniendo en cuenta la notación de la tabla 4.1, un contenedor identifica su tipo según sus parámetros C_i y D_i siguiendo la tabla 4.2:

Además, los parámetros LM_{jt} y LT_{jt} guardan cierta relación con el parámetro d_i, ya que si tenemos en cuenta la capacidad libre de los bloques, esta, se ha de actualizar a lo largo del horizonte temporal teniendo en cuenta el instante de salida de los contenedores. Por ejemplo, si un contenedor tiene una salida prevista por la parte de mar en el instante

Tipo de contenedor	C_i	D_i
Transhipment	1	1
Import	1	2
Export	2	1

Tabla 4.2: Tipos de contenedores según C_i y D_i

10, la capacidad libre del lado de mar en el bloque que está asignado aumentará en una unidad en el instante 10, ya que se libera un espacio dentro del bloque

4.1.3. Ejemplo de estudio

Para comenzar a comprender mejor el problema, introducimos un ejemplo número sencillo, el cual se mostrará a continuación.

En primer lugar, la tabla 4.3 nos muestra una serie de datos generales del problema. En concreto, podemos ver el número de bloques y la cantidad de contenedores a asignar en nuestro problema.

Para comenzar con la resolución del problema, hemos de tener una serie de datos sobre los contenedores, los cuales se muestran en las tablas 4.4 y 4.5, donde podemos ver una serie de parámetros correspondientes a cada contenedor y sus tiempos de asignación a los respectivos bloques.

En último lugar, en la tabla 4.6 podemos ver el espacio libre en cada lado (mar o tierra) de cada uno de los bloques. Cabe destacar que los bloques tienen un máximo de ubicaciones para introducir contenedores. Consideramos que la mitad de ellas corresponden al lado de mar del bloque y la otra mitad al lado de tierra.

Este ejemplo se utilizará y se resolverá mediante un modelo simplificado que propondremos en la sección 4.2, además de servirnos para comprender mejor los algoritmos planteados en la sección 4.3.

El objetivo principal nuestro problema es obtener la asignación de contenedores a bloques que minimice el tiempo total de dicha asignación.

Contenedores a asignar	5
Bloques	2

Tabla 4.3: Información del ejemplo de estudio
Tabla de elaboración propia.

Contenedor	C_i	r_i	d_i	D_i
A	1	5	–	2
B	2	6	–	1
C	1	4	–	2
D	2	8	–	1
E	2	2	12	1

Tabla 4.4: Lista de contenedores con su correspondiente información
Tabla de elaboración propia.

Contenedor	T_{i1}	T_{i2}
A	3	2
B	6	4
C	7	2
D	2	5
E	5	3

Tabla 4.5: Tiempo que tarda cada contenedor en ser asignado a cada bloque.
Tabla de elaboración propia.

Bloque	LT_{j0}	LM_{j0}
B_1	126	139
B_2	147	139

Tabla 4.6: Datos sobre el espacio libre de cada lado (tierra o mar) en cada bloque. Tabla de elaboración propia.

4.2 Simplificación matemática del problema

Dada la complejidad del problema de asignación de contenedores en una terminal portuaria, una manera de comprender mejor cómo trabajar con él, es proponer un modelo simplificado, que nos permita resolverlo de una manera sencilla y nos proporcione pistas para resolver situaciones de mayor complejidad. En la sección 4.2.1 exponemos la simplificación del problema y, seguidamente, en la sección 4.2.2 mostramos la solución propuesta para el ejemplo de la sección 4.1.3 utilizando esta simplificación.

4.2.1. Modelo simplificado

Para realizar una simplificación a nuestro problema, supondremos que todos los contenedores se encuentran disponibles en un mismo instante del tiempo $t = 0$ y que al bloque únicamente llegan nuevos contenedores, lo cual hará que la disponibilidad de huecos solo disminuya con el tiempo. Esto hará que el parámetro d_i y r_i no se tendrán en cuenta. Estas suposiciones en la realidad no se dan, ya que los contenedores se encuentran disponibles en distintos momentos del tiempo y existe la posibilidad de que algunos de ellos abandonen el bloque dentro del horizonte temporal contemplado. Así, con estas simplificaciones podremos tratar el problema como un problema de asignación.

Debido a la distinción que hacemos según el tipo de contenedor, contamos con 3 conjuntos de variables de decisión: Y_{ij}, X_{ij} y S_{ij}.

El conjunto de variables Y_{ij} indica si un contenedor i del tipo import se asiga a un bloque j, $\forall i \in \{1, ..., N1\}$, $\forall j \in B$. 0 en caso contrario.

El conjunto de variables X_{ij} indica si un contenedor i del tipo export, se asigna a un bloque j, $\forall i \in \{N1 + 1, ..., N2\}$, $\forall j \in B$. 0 en caso contrario.

El conjunto de variables S_{ij} indica si un contenedor i del tipo transhipment, se asigna a un bloque j, $\forall i \in \{N2 + 1, ..., N\}$, $\forall j \in B$. 0 en caso contrario.

La formulación matemática de esta simplificación será la siguiente:

Función objetivo:

$$Min \sum_{i=1}^{N1} \sum_{j=1}^{B} T_{ij} * Y_{ij} + \sum_{i=N+1}^{N2} \sum_{j=1}^{B} T_{ij} * X_{ij} + \sum_{i=N2+1}^{N} \sum_{j=1}^{B} T_{ij} * S_{ij} \tag{4.1}$$

Sujeto a:

$$\sum_{j=1}^{B} Y_{ij} = 1 \quad \forall i \in \{1, ...N\} \tag{4.2}$$

$$\sum_{j=1}^{B} X_{ij} = 1 \quad \forall i \in \{N1 + 1, ..., N2\} \tag{4.3}$$

$$\sum_{j=1}^{B} S_{ij} = 1 \quad \forall i \in \{N2+1, ..., N\} \tag{4.4}$$

$$\sum_{i=1}^{N1} Y_{ij} \leq LT_{j0} \quad \forall j \in B \tag{4.5}$$

$$\sum_{i=N1+1}^{N2} X_{ij} + \sum_{i=N2+1}^{N} S_{ij} \leq LM_{j0} \quad \forall j \in B \tag{4.6}$$

$$Y_{ij} \in \{0,1\} \quad \forall i \in \{0, ..., N1\}, \forall j \in B \tag{4.7}$$

$$X_{ij} \in \{0,1\} \quad \forall i \in \{N1+1, ..., N2\}, \forall j \in B \tag{4.8}$$

$$S_{ij} \in \{0,1\} \quad \forall i \in \{N2+1, ..., N\}, \forall j \in B \tag{4.9}$$

Nuestra función objetivo 4.1 muestra el coste temporal total de asignar todos los contenedores a los diferentes bloques de la terminal.

Las restricciones 4.2, 4.3 y 4.4 nos permiten asegurar que cada uno de los contenedores que llegan al puerto es asignado a uno de los bloques de la terminal.

Mediante la restricción 4.5 indicamos que la suma de todos los contenedores del tipo import asignados al bloque dado no puede superar la capacidad libre de ese bloque en el lado de tierra.

La siguiente restricción, 4.6 nos indica que la suma de todos los contenedores del tipo export y transhipment asignados al bloque dado, no puede superar la capacidad libre de ese bloque en el lado de mar.

Finalmente, con las restricciones 4.7, 4.8, 4.9 aseguramos que los conjuntos de variables de decisión Y_{ij}, X_{ij} y S_{ij} son variables binarias, es decir, solo pueden tomar 0 o 1 como valor.

4.2.2. Resolución de la formulación simplificada

Para obtener la solución de nuestro modelo simplificado hemos utilizado la herramienta *Solver*, disponible como complemento en *Microsoft Excel*. Mediante el uso de esta herramienta podemos resolver problemas de optimización, buscando maximizar o minimizar nuestra función objetivo.

Para resolver el modelo propuesto, utilizaremos los datos expuestos en la sección 4.1.3, considerando las simplificaciones mencionadas: un instante del tiempo igual a 0 para todo el problema y que únicamente pueden entrar contenedores al bloque, es decir, que las variables LM_{j0} y LT_{j0} sólo pueden disminuir su valor.

Los tiempos que emplea cada contenedor en ser asignado a cada bloque son los dados en la tabla 4.5. A su vez, el espacio libre de los bloques disponibles se muestra en la tabla 4.6.

En la tabla representada en la figura 4.2 se muestra la matriz con las variables de decisión, donde cada casilla representará X_{ij}, Y_{ij}, S_{ij}, según sea el tipo de contenedor. La suma de las columnas representa la ocupación de los bloques, diferenciando entre las partes de mar y tierra de cada bloque, este valor no puede superar el espacio libre disponible, definido en la tabla 4.6. Esto hace referencia a las restricciones 4.6 y 4.5. La suma de cada una de las filas ha de ser uno, lo que implica que un contenedor tiene que estar asignado a un solo bloque, según indicamos en las restricciones 4.3, 4.2, 4.4.

La solución óptima se obtendría asignando todos los contenedores al bloque 2, obteniendo un tiempo total del proceso de 16 unidades temporales.

Dada la matriz obtenida, podemos representar la función objetivo tomando los tiempos de la tabla 4.5. Así pues, la función objetivo resultante será la mostrada en la ecuación 4.10.

$$2 * Y_{A1} + 4 * X_{B2} + 2 * Y_{C1} + 5 * X_{D1} + 3 * X_{E1} = 16u.t. \tag{4.10}$$

		BLOQUES DISPONIBLES (j)						
		B1 T	B1 M	B2 T	B2 M			
CONTENEDORES (i)	A	0	0	1	0	1	=	1
	B	0	0	0	1	1	=	1
	C	0	0	1	0	1	=	1
	D	0	0	0	1	1	=	1
	E	0	0	0	1	1	=	1
		0	0	2	3			
		<=	<=	<=	<=			
		126	139	147	139			
F.O.	16	u.t.						

Figura 4.2: Matriz resultado del ejemplo 4.1.3 con el modelo simplificado.
Figura de elaboración propia

4.3 Algoritmos propuestos

En la sección 4.3.1 introduciremos el concepto de heurística, cuál es su importancia y por qué haremos uso de ella. A continuación, en las secciones 4.3.2, 4.3.3 y 4.3.4, expondremos los algoritmos utilizados para dar solución al problema de asignación, mostraremos, además, cuáles han sido los resultados obtenidos.

4.3.1. Introducción a la heurística

Mediante los problemas de optimización se trata, normalmente, de conseguir que el valor de una función objetivo dada tome su máximo o mínimo, en función de cuál sea la naturaleza del problema. Para ello se tienen en cuenta una serie de restricciones que limitan los valores de la función objetivo.

Existen algunos problemas de optimización, los que entran dentro de la clase de complejidad P, que se pueden resolver en tiempo polinómico.

Sin embargo, existen otro tipo de problemas, de la clase *NP*, donde no se puede garantizar su resolución en un tiempo polinómico, es decir, no se resuelven en un tiempo razonable.

Para abordar este tipo de problemas se recurre a los métodos heurísticos, que según [9] son procedimientos, los cuales resuelven el problema aportando soluciones de una calidad alta y que se ofrecen en un tiempo razonable. Estas soluciones no se puede garantizar que sean óptimas, pero sí que ofrecen un resultado factible y en muchas ocasiones, de calidad. A la hora de diseñar un método heurístico específico para un problema, se ha de tener en cuenta todos los datos disponibles.

En la sección 4.2 propusimos un modelo simplificado para el problema de asignación de contenedores. En concreto, asumimos que el instante de llegada de todos los contenedores era 0, que, en los bloques, únicamente, entraban contenedores y que, en ningún caso, abandonaban el bloque. Estas simplificaciones nos ayudan a resolver de manera óptima, a través de un modelo matemático, una versión simplificada del problema. Sin embargo, al introducir restricciones más realistas, tales como instantes de llegada distintas para los contenedores o instantes de salida de los mismos, los modelos matemáticos son muy complejos y no resuelven el problema en un tiempo razonable. Es por ello que se propone utilizar técnicas heurísticas para la resolución del mismo.

Para el diseño de nuestros métodos heurísticos tomaremos una serie de datos y dejaremos de lado las simplificaciones anteriores. Ahora el instante de tiempo no será cero en todo momento y, por tanto, tendremos que tener en cuenta el valor de r_i y d_i para mantener actualizada la cantidad de espacios libres de los bloques, LM_{jt} y LT_{jt}.

Gracias a la utilización de los métodos heurísticos podemos obtener soluciones sencillas, que nos permitan entender el problema. Podremos también introducir modificaciones en nuestras soluciones al problema, de manera que podamos comparar fácilmente distintos puntos de vista para abordarlo.

4.3.2. Algoritmo de asignación aleatoria

En primer lugar, introduciremos un método heurístico de asignación aleatoria de contenedores a los diferentes bloques. La solución que nos proporciona es una solución funcional, la cual no será buena a priori, pero nos permite obtener una solución al problema de asignación de contenedores sin utilizar demasiados recursos.

Este tipo de algoritmos pueden llegar a tener un rendimiento similar a otros de mayor complejidad. Además, sirven como referencia para saber si merece la pena desarrollar otro tipo de algoritmos, pues si no obtenemos mejoras respecto a una asignación aleatoria, no resultará de gran utilidad.

Asumiremos que realizaremos una asignación aleatoria del bloque, estableciendo previamente un orden en los contenedores a asignar. La ordenación de los contenedores se realizará según su r_i, es decir, de menor a mayor instante de llegada a la terminal, ya sea por mar o por tierra. Así pues, una vez tengamos el primer contenedor disponible, se realizará una asignación aleatoria a uno de los bloques del conjunto B.

A continuación, en el algoritmo 1, encontramos el pseudocódigo de este método planteado.

Para realizar los cálculos, hemos de conocer el conjunto de contenedores N para asignar, con información sobre su instante de llegada, de salida, el lugar de llegada, su destino y su coste temporal de asignación a cada uno de los bloques. También tendremos que saber el número de bloques B disponibles, con información sobre el espacio libre en

el momento inicial, tanto en la parte cercana a mar LM_{jt}, como en la parte cercana a tierra LT_{jt}.

Algoritmo 1: Algoritmo de asignación aleatoria

```
ts = 0;
solucion = ∅;
Ordenar de menor a mayor la lista de contenedores según su r_i;
for cada contenedor i hasta N do
    asignado = 0;
    while ts < r_i do
        Comprobamos salidas previstas entre ts y ts + 1;
        ts = ts + 1;
    end
    while asignado == 0 do
        if D_i == 1 then
            if LM_jts > 0 then
                Asignamos al contenedor i un bloque aleatorio j ∈ B que tenga
                 espacio en la parte de mar;
                Reducimos en 1 el espacio libre del bloque j ∈ B en la parte de mar
                 en instante ts;
                asignado = 1;
            end
        else
            if LT_jts > 0 then
                Asignamos al contenedor i un bloque aleatorio j ∈ B que tenga
                 espacio en la parte de tierra;
                Reducimos en 1 el espacio libre del bloque j ∈ B en la parte de
                 tierra en instante ts;
                asignado = 1;
            end
        end
        if asignado == 1 then
            Comprobamos salidas previstas en ∀j ∈ B entre ts y ts + T_ij;
            Actualizamos el tiempo →ts = ts + T_ij;
            Actualizamos LT_jts y LM_jts;
            Añadimos a solucion el contenedor i, el bloque asignado j y ts;
        end
    end
end
return solucion
```

El primer paso será inicializar la variable *ts*, que indicará el instante de tiempo en el que nos encontramos y la variable *solucion*, que será una lista de listas, la cual tendrá elementos formados por un contenedor, un bloque asignado y un tiempo en el que finaliza su asignación.

A continuación ordenaremos la lista de contenedores para asignar, de manera que el orden sea creciente según su instante de llegada, dado por el componente r_i de cada contenedor. Tras ello procedemos a realizar la asignación de cada uno de los contenedores de la lista ordenada. Para ello utilizaremos el primer bucle for.

En cada iteración del bucle for se inicializará la variable *asignado* a 0, para indicar que todavía no se ha asignado un bloque a dicho contenedor.

La variable *ts* nos va a permitir saber en qué instante del tiempo ha finalizado la asignación de cada contenedor para lo cual habrá de tener en cuenta el tiempo que tarda cada contenedor en llegar a cada bloque, que viene dado por el parámetro T_{ij}. Para que un contenedor pueda ser asignado, el valor de *ts* ha de ser mayor o igual que el valor de la r_i correspondiente al contenedor. Dicha comprobación se realizará en un bucle *while* y si el contenedor no se encuentra disponible se incrementará el valor de *ts* en una unidad temporal, hasta que el contenedor se encuentre disponible. Mientras, si no hay contenedores disponibles, debemos comprobar si hay salidas planificadas en los bloques, dato que se nos proporcionaba como *input*. De ser así, tendremos que actualizar los espacios libres de cada bloque.

Una vez hemos comprobado que el contenedor i está disponible y no está asignado a un bloque, elegiremos un bloque aleatorio de entre todos los bloques disponibles. Hemos de comprobar que esa parte del bloque asignado no esté ocupado al completo, ya que en caso de estar ocupado, el contenedor no podrá ser asignado a ese bloque y tendremos que elegir otro bloque para asignarlo. Tras elegir a qué bloque lo asignaremos, comprobaremos qué tipo de contenedor es, para asignarlo a un bloque, en el lado de mar o en el lado de tierra del bloque, según sea el valor de D_i.

Seguidamente, si el bloque no está completamente ocupado, se llevará el contenedor al *I/O* correspondiente del bloque asignado y reduciremos en 1 el valor de LT_{jt} o LM_{jt}, según el tipo de contenedor que sea, con la finalidad de controlar la capacidad de los bloques y no permitir que un contenedor se asigne a un bloque ocupado por completo. Indicaremos además que el contenedor i ha sido ya asignado, actualizando la variable *asignado* a 1.

Durante el tiempo que dura el proceso hemos de comprobar si existen contenedores con una salida prevista en los diferentes bloques. Por tanto, una vez asignemos el contenedor y, antes de que este llegue al bloque correspondiente, comprobamos si hay salidas en ese intervalo de tiempo, con el fin de actualizar la ocupación de los bloques.

Incrementaremos el tiempo *ts* en el valor T_{ij} y añadiremos a nuestra lista *solucion* un nuevo componente cuyos elementos serán el contenedor que hemos asignado, el bloque elegido y el tiempo *ts* en ese instante.

Este proceso se llevará a cabo con cada uno de los contenedores y se obtendrá finalmente una lista de listas que contenga la información del contenedor, el bloque asignado y el tiempo en el que terminó el proceso de asignación del contenedor i. El valor de la función objetivo (tiempo total de la asignación), será el tiempo en el que termina el proceso de asignación del último contenedor de la lista.

La solución al problema vendrá dada por la variable *solucion*, la cual cambiará cada vez que se ejecute el algoritmo, ya que se trata de un algoritmo aleatorio.

Una vez hemos descrito el funcionamiento del algoritmo aleatorio, mostraremos un ejemplo numérico, que ayudará a entender mejor este método. Para ello utilizaremos los datos del ejemplo de estudio de la sección 4.1.3. Nuestra labor será asignar los 5 contenedores mostrados en la tabla 4.4 a uno de los bloques, cuya información encontramos en la tabla 4.6.

A continuación tendremos que diferenciar qué tipo de contenedor es cada uno de los disponibles. En la tabla 4.7 vemos el tipo de cada uno de los 5 contenedores a asignar en el ejemplo. El tipo de cada contenedor es conocido según el valor de C_i y D_i, según se explicó en la tabla 4.2.

Para comenzar con el proceso, hemos de ordenar los contenedores según su tiempo de llegada al puerto. Tomamos los datos de la tabla 4.4 y generamos una nueva con los contenedores ordenados (4.8).

Contenedor	Tipo
A	Import
B	Export
C	Import
D	Export
E	Export

Tabla 4.7: Información sobre el tipo de cada contenedor.
Tabla de elaboración propia.

Contenedor	Tipo	r_i
E	Export	2
C	Import	4
A	Import	5
B	Export	6
D	Export	8

Tabla 4.8: Lista de contenedores ordenados de menor a mayor instante de llegada (r_i).
Tabla de elaboración propia.

Con esta lista de contenedores ordenados podemos proceder a su asignación. Comenzamos con un ts a 0, y comparando con el tiempo r_i del primer contenedor, vemos que no está disponible todavía. Por lo tanto, hemos de esperar hasta el instante de tiempo $ts = 2$. En ese momento comenzamos el proceso de asignación del contenedor E. Como vemos, el contenedor E es del tipo export. Al ser el primer contenedor y partir de que ningún bloque está ocupado al completo, podemos asignarlo a cualquier bloque.

Asignamos aleatoriamente el contenedor E al bloque 2. Al ser el contenedor E de tipo export, llega por tierra y es transportado al I/O de tierra del bloque 2. Posteriormente será ubicado por la grúa de patio en la parte de mar dentro del bloque, ya que su destino es salir por mar. Así, tendremos que reducir en 1 el espacio libre del bloque 2 en el lado de mar en el momento ts, sumar el tiempo empleado en llevar el contenedor, que es 3, al bloque a la variable ts, siendo ahora 5 y añadir a la variable $solucion$ el siguiente elemento: [E,B_2,5].

El próximo contenedor a asignar es el contenedor C, que es del tipo import. La variable ts tras la asignación del contenedor E al bloque 2 toma el valor 5. Comprobaremos que el contenedor C se encuentre disponible y así es, pues su r_i es igual a 4. Tras esta comprobación, asignamos aleatoriamente el contenedor C al bloque 1, el cual tiene espacio en el lado de tierra, ya que el contenedor C llega por mar pero su destino es ir a tierra. Reducimos en 1 el espacio libre del bloque 1 en el lado de tierra en el instante de tiempo ts, sumamos el tiempo que tarda el contenedor en ser asignado al bloque, el cual tarda 7 u.t. e incluimos en la solución un nuevo elemento: [C,B_1,12].

Repetiremos el mismo proceso con los contenedores restantes. Los resultados obtenidos son que el contenedor A se asigna al bloque 1, ocupando lugar en su lado de tierra, el contenedor B se asigna al bloque 2 ocupando su lado de mar y el contenedor D se asigna al bloque 2, lado de mar. En ningún momento se llega a ocupar por completo ningún bloque, por lo que las asignaciones iniciales se llevan siempre a cabo.

Una vez hemos terminado la asignación de todos los contenedores y el algoritmo ha finalizado, obtenemos una solución factible, dada por la variable $solucion$, cuyos componentes mostraremos en la tabla 4.9. El tiempo total empleado en realizar la asignación es de 24 u.t., que es el tiempo en el que finaliza la asignación del último contenedor.

Contenedor	Bloque	Tiempo llegada
E	B_2	5
C	B_1	12
A	B_1	15
B	B_2	19
D	B_2	24

Tabla 4.9: Solución del ejemplo de la sección 4.1.3 aplicando el algoritmo aleatorio. Tabla de elaboración propia.

4.3.3. Algoritmo iterativo de asignación aleatorio

Tras plantear un método heurístico de asignación aleatoria, introducimos una variante, con la cual se realizan una serie de iteraciones sobre el método planteado en la sección anterior. La finalidad de esta variación es obtener un resultado distinto con cada iteración y obtener como solución final la mejor solución obtenida entre todas las iteraciones.

Se espera obtener una mejor solución, pero, debido a la repetición del método durante un número dado de iteraciones, el coste computacional es mayor.

Para el desarrollo de este método hemos de contar con dos listas donde guardemos los resultados: una lista con el resultado de la iteración actual y otra variable que indique la mejor solución encontrada en todas las iteraciones.

En el algoritmo 2 mostraremos el pseudocódigo de este método.

Para realizar estos cálculos, son necesarios los mismos datos que en la sección 4.3.2. Además, necesitaremos saber cuál es el número máximo de iteraciones $MAXIT$.

Como primer paso inicializaremos las variables *tmejorsol* e *iteracion* a 0, además de la lista *mejorsol*, inicialmente vacía. La variable *tmejorsol* nos indicará el tiempo total de la mejor solución hasta el momento, mientras que *iteracion* nos mostrará en qué iteración nos encontramos. La lista *mejorsol* hace referencia a la mejor solución, donde se mostrará toda la información de ella.

A continuación ordenaremos el conjunto de contenedores N, según su instante de llegada, r_i. Tras esto, iniciaremos un bucle que estará iterando mientras no se alcance el número máximo de iteraciones dado previamente.

Para cada iteración se inicializará la variable *ts* a 0, ya que así podemos almacenar el tiempo total que obtenemos como solución en esa iteración. Tras ello, el funcionamiento es el mismo que el algoritmo presentado en la sección 4.3.2.

Una vez ha finalizado el bucle for se ha de comprobar si es la primera iteración, ya que si se trata de la primera, hemos de almacenar el resultado obtenido como en las variables *tmejorsol* y *mejorsol*, ya que no hay ninguna otra solución. En caso contrario, si no es la primera iteración, habría que comparar las variables *tmejorsol* y *ts* y si el tiempo obtenido en esta iteración es mejor, actualizaríamos las variables *tmejorsol* y *mejorsol* con el resultado de esta iteración.

Utilizaremos el mismo ejemplo numérico que en la sección 4.3.2 para comprender mejor este método. Para ello, tomaremos como primera iteración el ejemplo de la sección 4.3.2 y se dará a la variable $MAXIT$ un valor de 4.

Realizaremos 3 iteraciones más y los resultados serán los mostrados en las tablas 4.10, 4.11 y 4.12. En la tabla 4.13 vemos un resumen de los tiempos de las 4 iteraciones, indicando cuál es la mejor solución.

Algoritmo 2: Algoritmo iterativo de asignación aleatoria

```
solucion = ∅;
mejorsol = ∅;
tmejorsol = 0;
iteracion = 0;
Ordenar de menor a mayor la lista de contenedores según su r_i;
while iteracion < MAXIT do
    ts = 0;
    for cada contenedor i hasta N do
        asignado = 0;
        while ts <r_i do
            Comprobamos salidas previstas entre ts y ts + 1;
            ts = ts + 1;
        end
        while asignado == 0 do
            if D_i == 1 then
                if LM_j > 0 then
                    Asignamos al contenedor un bloque aleatorio j ∈ B que tenga
                     espacio en la parte de mar;
                    Reducimos en 1 el espacio libre del bloquej ∈ B en la parte de
                     mar;
                    asignado = 1;
                end
            else
                if LT_j > 0 then
                    Asignamos al contenedor un bloque aleatorio j ∈ B que tenga
                     espacio en la parte de tierra;
                    Reducimos en 1 el espacio libre del bloque j ∈ B en la parte de
                     tierra;
                    asignado = 1;
                end
                if asignado == 1 then
                    Comprobamos salidas previstas en ∀j ∈ B entre ts y ts + T_ij
                     Actualizamos el tiempo →ts = ts + TT_ij;
                    Actualizamos LT_jts y LM_jts;
                    Añadimos a solucion el contenedor i, el bloque asignado j y ts;
                end
            end
        end
        if iteracion == 1 then
            tmejorsol = ts;
            mejorsol = solucion;
        else
            if tmejorsol > ts then
                tmejorsol = ts;
                mejorsol = solucion;
            end
        end
        iteracion = iteracion + 1
    end
end
return solucion
```

Contenedor	Bloque	Tiempo de asignación
E	B_1	7
C	B_2	9
A	B_2	11
B	B_1	17
D	B_1	19

Tabla 4.10: Solución para la iteración 2 con el algoritmo aleatorio.
Tabla de elaboración propia.

Contenedor	Bloque	Tiempo de asignación
E	B_1	7
C	B_1	14
A	B_1	17
B	B_2	21
D	B_2	26

Tabla 4.11: Solución para la iteración 3 con el algoritmo aleatorio.
Tabla de elaboración propia.

Contenedor	Bloque	Tiempo de asignación
E	B_2	5
C	B_1	12
A	B_1	15
B	B_1	21
D	B_1	23

Tabla 4.12: Solución para la iteración 4 con el algoritmo aleatorio.
Tabla de elaboración propia.

Iteración 1	Iteración 2	Iteración 3	Iteración 4	Mejor solución
24	19	19	19	19

Tabla 4.13: Resultado de las 4 iteraciones con el algoritmo iterativo de asignación aleatoria
Tabla de elaboración propia.

Después de realizar las 4 iteraciones, llegamos a un resultado final para la variable *tmejorsol* de 19 u.t., resultado que se obtuvo en la segunda iteración.

Se puede observar que gracias a la utilización del algoritmo iterativo se ha conseguido mejorar el tiempo en 5 u.t. respecto a la primera iteración, a pesar de que el coste computacional se ha elevado al respecto con el algoritmo de asignación aleatoria, ya que, éste, se ha realizado 4 veces.

4.3.4. Algoritmo de proximidad

Los algoritmos presentados en 1 y 2 nos ofrecen soluciones para el problema de asignación, pero realmente lo que se está buscando es minimizar el coste temporal del proceso de asignación. Así pues, dado que tenemos como datos del problema los tiempos que tarda cada contenedor i en ser asignado a cada bloque j, podemos considerar que una buena forma para asignar cada contenedor a un bloque es escogiendo el bloque más cercano a dicho contenedor, es decir, que su T_{ij} sea el menor.

Llevaremos a cabo el desarrollo de nuestro método heurístico tomando como referencia el tipo de algoritmos llamados de vecindad o K-NN, comúnmente utilizados para tareas de clasificación. En nuestro problema, para cada uno de los contenedores, seleccionaremos la posición más cercana. Una posible situación es que para un dato, la proximidad sea igual para más de una posición. En ese caso habría que decidir con una regla cuál será la posición a elegir [19]. En nuestro caso, elegiremos el bloque cuyo índice sea menor, es decir, entre el bloque 1 y bloque 2, nos quedaríamos con el bloque 1.

Partiendo de este tipo de algoritmos, hemos desarrollado un método heurístico, el cual se muestra en el pseudocódigo del algoritmo 3.

Para realizar los cálculos, son necesarios los mismos *inputs* que en la sección 4.3.2.

Algoritmo 3: Algoritmo de proximidad

```
ts = 0;
solucion = ∅;
Ordenar de menor a mayor la lista de contenedores según su r_i;
for cada contenedor i hasta N do
    asignado = 0;
    while ts < r_i do
        Comprobamos salidas previstas entre ts y ts + 1;
        ts = ts + 1;
    end
    while asignado == 0 do
        if D_i == 1 then
            Seleccionar mínimo T_ij del contenedor i ∀j ∈ B;
            Bloque j = posición del mínimo T_ij;
            if LM_jts > 0 then
                LM_jts = LM_jts − 1;
                asignado = 1;
            else
                Elegir un bloque B aleatorio de los restantes que tenga espacio;
                asignado = 1;
            end
        else
            Seleccionar mínimo T_ij del contenedor i ∀j ∈ B;
            Bloque j = posición del mínimo T_ij;
            if LT_jts > 0 then
                LT_jts = LT_jts - 1;
                asignado = 1;
            else
                Elegir un bloque B aleatorio de los restantes que tenga espacio;
                asignado = 1;
            end
        end
        if asignado == 1 then
            Comprobamos salidas previstas en ∀j ∈ B entre ts y ts + T_ij;
            Actualizamos el tiempo →ts = ts + T_ij;
            Actualizamos LT_jts y LM_jts;
            Añadimos a solucion el contenedor i, el bloque asignado j y ts;
        end
    end
end
return solucion
```

Dado el conjunto de contenedores N, ordenamos los contenedores según el instante de llegada, r_i, de menor a mayor. Tras esto, iniciaremos un bucle for que recorrerá todo el conjunto de contenedores. En este bucle, lo primero que se hará será inicializar la variable *asignado* a cero, indicando que el contenedor i no ha sido todavía asignado a un bloque.

Tras esto, mediante un bucle *while* nos aseguraremos que el proceso de asignación no comience hasta que el instante de tiempo en el que nos encontremos sea mayor o igual que el instante de llegada del contenedor i. Mientras, si no hay contenedores disponibles,

debemos comprobar si hay salidas planificadas en los bloques, dato que se nos proporcionaba como *input*. De ser así, tendremos que actualizar los espacios libres de cada bloque.

Una vez el contenedor se encuentra listo para ser asignado, y mientras que el contenedor no esté ya asignado ($asignado = 0$), comprobaremos qué clase de contenedor es, mediante una condición. Sabemos que si el contendor i tiene una D_i igual a 1, se trata de un contenedor de tipo export o transhipment y, por tanto, su destino es la parte de mar del bloque j. De ser así, seleccionaremos el bloque cuyo tiempo sea el mínimo en el proceso de asignación y si dicho bloque no está completo en la parte de mar, el contenedor i será asignado al bloque j. En caso de que el bloque j estuviera completo en el lado de mar, elegiremos uno de los bloques aleatoriamente, comprobando que esté libre. Tras esto reduciremos el espacio libre en el lado de mar del bloque j en 1 unidad.

En caso de que el contenedor sea del tipo import, su D_i será igual a 2, y, por tanto, se ha de realizar la parte del pseudocódigo correspondiente a *else*. Siguiendo un proceso similar al anterior, se elegirá el bloque cuyo tiempo sea el mínimo y si todavía hay espacio disponible en la parte de tierra de ese bloque se asignará, reduciendo en 1 la capacidad libre del bloque j en el lado de tierra.

Una vez hemos asignado el contenedor, independientemente del tipo, incrementaremos el tiempo ts según el T_{ij} del bloque j escogido e indicaremos que está ya asignado, cambiando el valor de *asignado* a 1.

Durante el tiempo que dura el proceso hemos de comprobar si existen contenedores con una salida prevista en los diferentes bloques. Por tanto, una vez asignemos el contenedor y, antes de que este llegue al bloque correspondiente, comprobamos si hay salidas en ese intervalo de tiempo, con el fin de actualizar la ocupación de los bloques.

Una vez que el contenedor se ha asignado a un bloque, añadiremos en nuestra lista de soluciones $solucion$ un nuevo elemento formado por el contenedor i, el bloque j y el tiempo ts.

Para comprender mejor este método haremos uso del ejemplo 4.1.3. Habrá que asignar 5 contenedores, con 2 bloques disponibles.

Tras ordenar los contenedores según su r_i, de menor a mayor, el orden resultante es el mismo que obtuvimos en la tabla 4.8 utilizada en 4.3.2.

Comenzando a ordenar por el primero de los contenedores de la lista ordenada, elegimos el contenedor E. Para este contenedor, seleccionamos el mínimo de las distancias a los bloques, dato que podemos obtener de la tabla 4.5. El bloque más cercano es el bloque 2, tardaría 3 u.t. y todavía queda espacio libre. Por tanto, el contenedor E se asigna al bloque 2. Como el contenedor E estuvo disponible en el instante 2 y tarda 3 u.t. en ser asignado, el proceso finalizará en el instante 5, valor que tomará la variable ts. Además hemos de reducir el espacio disponible del bloque 2 en la parte de mar en una unidad. A continuación añadiremos a nuestra solución el elemento [E,B_2,5]. Realizaremos el mismo proceso para el resto de contenedores del conjunto.

El siguiente sería el contenedor C, que es de tipo import. Así pues, el contenedor C se encuentra disponible en el instante 4 y actualmente estamos en el instante 5, por lo que ya se encuentra disponible. Escogemos el bloque cuya distancia es la mínima, siendo este el bloque 2 y cuya distancia es 2 u.t. Así pues, comprobamos que el bloque 2 tenga suficiente capacidad en el lado de tierra y, tras la comprobación, incrementamos el valor de ts en 2 u.t. y reducimos el espacio libre del bloque 2 en el lado de tierra. Tras esto añadiremos a nuestra lista de soluciones el elemento [C, B_2, 7].

El proceso se repetirá con el resto de contenedores, obteniendo finalmente una lista $solucion$ la cual mostraremos en la tabla 4.14 y un tiempo final ts igual a 15 u.t..

Contenedor	Bloque	Tiempo de asignación
E	B_2	5
C	B_2	7
A	B_2	9
B	B_2	13
D	B_1	15

Tabla 4.14: Lista de contenedores ordenados de menor a mayor instante de llegada (r_i). Tabla de elaboración propia.

Comparando el tiempo final obtenido en los tres algoritmos propuestos 4.3.2, 4.3.3, 4.3.4, vemos que el que nos ofrece una mejor solución es el algoritmo de proximidad.

4.4 Análisis computacional

Una vez hemos mostrado cuáles serán los métodos que utilizaremos para la asignación de contenedores a los distintos bloques de la terminal portuaria en la sección 4.3, realizaremos pruebas haciendo uso de una serie de instancias generadas, las cuales se utilizarán en cada uno de los métodos. En primer lugar, en la sección 4.4.1 mostramos la manera en la que hemos generado estas instancias y, más adelante, en la sección 4.4.2 mostraremos cuáles han sido los resultados obtenidos tras realizar las pruebas en las que utilizamos las instancias generadas para cada uno de los métodos.

4.4.1. Generación de instancias

Con la finalidad de realizar pruebas con los métodos expuestos en la sección 4.3, se han creado un total de 25 instancias, generadas aleatoriamente. Estas instancias corresponden a 5 tamaños distintos, siendo 500, 1000, 1500, 2000 y 2500 el número posible de contenedores. En todos los casos se ha considerado que el número de bloques era 5.

Para cada tamaño, se han generado 5 instancias distintas, con una información para los contenedores aleatoria. Las instancias se han nombrado como *Ins_BN_NCA_D*, siendo A el número de contenedores de la instancia, N el número de bloques y D el número de instancia con ese tamaño de contenedores. Así pues, la instancia *Ins_B*5_*NC*500_1 contendrá 500 contenedores, 5 bloques y será la primera de las instancias con esa cantidad.

Cada contenedor de las diferentes instancias nos proporciona información sobre su coste temporal hasta cada uno de los bloques, lo cual permitirá obtener el resultado final.

Además, en cada bloque conoceremos las próximas salidas de contenedores ya asignados previamente y el espacio libre en cada lado (tierra o mar) antes de comenzar el proceso, y sabremos si salen del lado de mar o del lado de tierra, con el fin de poder actualizar los espacios libres de manera adecuada.

4.4.2. Resultados

A la hora de realizar las pruebas computacionales, se ha utilizado un el sistema operativo *macOS Catalina*, en un dispositivo *Apple*, modelo *MacBook Air* con 8GB de RAM y un procesador Intel Core i5 de doble núcleo de 1,6 GHz. La implementación se ha realizado utilizando *Python 2.7*.

A lo largo de esta sección explicaremos las pruebas realizadas con los algoritmos planteados previamente. Indicaremos el desempeño de los algoritmos mediante un parámetro que mide la desviación porcentual relativa, RPD (*Relative Percentage Deviation*), sobre la mejor solución conocida para esa instancia.

Mediremos la eficacia o calidad de los algoritmos mediante la expresión mostrada en 4.11. Necesitaremos saber cuál ha sido la mejor solución obtenida (s^*) de entre todos los algoritmos planteados y la solución obtenida para cada uno de los algoritmos propuestos (s). Con esto, podremos sacar la desviación relativa respecto a la mejor solución conocida s^*. Multiplicaremos por 100 para obtener el porcentaje de desviación obtenido (RPD).

$$RPD = \frac{s - s^*}{s^*} * 100 \tag{4.11}$$

Para las pruebas hemos contado con instancias con diferentes cantidades de contenedores, unas con 500, otras con 1000, 1500, 2000 y 2500. En total había 5 instancias de cada tamaño.

Realizaremos dos grupos de instancias según los tamaños, consideraremos que las instancias de 500 y 1000 contenedores son pequeñas y formarán un grupo. Las instancias de 1500, 2000 y 2500 contenedores serán consideradas como las instancias grandes y formarán otro grupo.

En la tabla 4.15 mostraremos los datos sobre las instancias pequeñas y en la tabla 4.16 los de las instancias grandes. Las columnas indicarán, en primer lugar, el nombre de la instancia y el coste temporal total de la mejor solución (s^*) de los algoritmos utilizados. Después indicaremos para cada uno de los algoritmos su RPD, calculado mediante la fórmula mostrada en la ecuación 4.11, su coste temporal (s) y el tiempo empleado en realizar el cálculo, en milisegundos (t). Corresponderán a los algoritmos aleatorio, iterativo de asignación aleatoria y de proximidad, respectivamente.

Los algoritmos aleatorio y de proximidad se van a ejecutar una única vez para cada una de las instancias, mientras que el algoritmo iterativo se ejecutará con 10 iteraciones para cada instancia.

Los resultados únicamente serán comparables entre sí, pues no tenemos datos sobre otros métodos que proporcionen una solución a este problema.

Como podemos observar en las tablas 4.15 y 4.16, el algoritmo que mejor solución (s^*) nos ha proporcionado en todos los casos, es el algoritmo de proximidad. Este nos servirá como referencia para comparar con los algoritmos aleatorio e iterativo.

En primer lugar, analizamos las instancias más pequeñas, las de 500 y 1000 contenedores. En la tabla 4.15 podemos ver que las columnas correspondientes a la s del algoritmo de proximidad y la columna de s^* son iguales. Esto es debido a que la mejor solución se obtiene con este algoritmo.

		ALEATORIO			ITERATIVO			PROXIMIDAD		
INSTANCIA	s*	RPD	s	t	RPD	s	t	RPD	s	t
Ins_B5_NC500_1	1137	139,75	2726	0,02	141,07	2741	0,34	0	1137	0,03
Ins_B5_NC500_2	1078	154,45	2743	0,02	142,76	2617	0,32	0	1078	0,03
Ins_B5_NC500_3	1101	149,86	2751	0,03	147,77	2728	0,47	0	1101	0,04
Ins_B5_NC500_4	1085	143,96	2647	0,03	143,69	2644	0,45	0	1085	0,03
Ins_B5_NC500_5	1062	151,88	2675	0,03	145,57	2608	0,35	0	1062	0,04
Ins_B5_NC 1000_1	2276	145,47	5587	0,08	138,97	5439	1,59	0	2276	0,09
Ins_B5_NC 1000_2	2132	156,85	5476	0,08	146,06	5246	1,34	0	2132	0,08
Ins_B5_NC 1000_3	2212	152,76	5591	0,10	144,98	5419	1,23	0	2212	0,08
Ins_B5_NC 1000_4	2152	152,88	5442	0,07	149,54	5370	1,07	0	2152	0,08
Ins_B5_NC 1000_5	2194	146,81	5415	0,08	142,39	5318	1,12	0	2194	0,08
Promedio		149,47	4105,30	0,05	144,28	4013,00	0,83	0,00	1642,90	0,06

Tabla 4.15: RPD, tiempos de asignación y tiempos de computación de las instancias pequeñas. Tabla de elaboración propia

Lo primero que podemos pensar es que cuando elegimos un método de asignación aleatorio, el resultado esperable no va a ser bueno. En las tablas podemos corroborarlo. Observamos que en las instancias más pequeñas, el algoritmo aleatorio es un 149,47 % peor que el el algoritmo de proximidad, y el algoritmo iterativo es un 144,48 % peor.

Respecto a los tiempos de computación que observamos, el algoritmo de proximidad ya no es el mejor. Esto se debe a que la asignación de contendores a los bloques se realiza mediante un proceso donde hay que extraer los datos de cada contenedor y elegir el de mínimo coste temporal. El algoritmo que mejores resultados computacionales ofrece es el aleatorio, ya que el cálculo necesario es poco complejo, simplemente selecciona un bloque aleatorio, aún así no queda muy distante del tiempo de computación del algoritmo de proximidad y en términos de milisegundos, la diferencia no es notable. Cabe destacar que existe una gran diferencia con el algoritmo iterativo, pues este su tiempo de computación es cercano a 1 milisegundo. El motivo de esta gran diferencia es que el algoritmo iterativo realiza un mismo proceso durante un número dado de veces, en este caso 10.

Pese a las diferencias en cuanto a los valores del tiempo de computación, no deberían servir como guía para elegir un algoritmo u otro, ya que estamos hablando de milisegundos.

A la hora de comprobar cuál es la solución que más conviene (s^*), vemos que el algoritmo de proximidad ofrece un tiempo de asignación total muy inferior a ambos algoritmos, lo que nos permite concluir que en instancias pequeñas, el algoritmo de proximidad ofrece mejores resultados que los algoritmos aleatorio e iterativo.

Tras analizar las instancias más pequeñas, pasamos a ver qué sucede con aquellas instancias que hemos considerado grandes, las de 1500, 2000 y 2500 contenedores. En la tabla 4.16 podemos ver los resultados obtenidos.

Podemos destacar que en promedio el algoritmo aleatorio es un 140,86 % peor que el de proximidad y que el iterativo es un 136,46 % peor. Por tanto, en cuanto a la solución, el mejor algoritmo para instancias grandes es el de proximidad.

A la hora de comparar los tiempos de computación de los algoritmos, vemos que el algoritmo de proximidad sigue siendo el mejor y que conforme crece el número de contenedores, esta diferencia se hace más notable. Al igual que sucedía con las instancias más pequeñas, el algoritmo iterativo ofrece un tiempo de computación bastante mayor que los otros dos algoritmos.

Respecto a la solución que ofrece, s, observamos que los resultados del algoritmo de proximidad sigue siendo el mejor.

		ALEATORIO			ITERATIVO			PROXIMIDAD		
INSTANCIA	s*	RPD	s	t	RPD	s	t	RPD	s	t
Ins_B5_NC 1500_1	3318	151,63	8349	0,17	146,11	8166	1,89	0	3318	0,17
Ins_B5_NC 1500_2	3298	150,97	8277	0,17	149,73	8236	2,22	0	3298	0,17
Ins_B5_NC 1500_3	3287	152,72	8307	0,19	144,36	8032	2,00	0	3287	0,18
Ins_B5_NC 1500_4	3267	151,12	8204	0,25	144,60	7991	2,01	0	3267	0,17
Ins_B5_NC 1500_5	3163	154,79	8059	0,22	155,20	8072	1,85	0	3163	0,17
Ins_B5_NC 2000_1	4512	148,07	11193	0,54	137,03	10695	3,28	0	4512	0,28
Ins_B5_NC 2000_2	4495	145,98	11057	0,76	144,25	10979	3,46	0	4495	0,31
Ins_B5_NC 2000_3	4480	147,12	11071	0,44	137,99	10662	2,90	0	4480	0,30
Ins_B5_NC 2000_4	4733	134,61	11104	0,64	130,36	10903	2,95	0	4733	0,29
Ins_B5_NC 2000_5	4413	143,15	10730	0,51	145,46	10832	2,85	0	4413	0,29
Ins_B5_NC 2500_1	5796	136,51	13708	1,68	132,04	13449	5,72	0	5796	0,45
Ins_B5_NC 2500_2	6362	115,14	13687	1,33	112,40	13513	6,33	0	6362	0,45
Ins_B5_NC 2500_3	5987	128,36	13672	1,38	128,26	13666	7,36	0	5987	0,45
Ins_B5_NC 2500_4	6301	117,81	13724	1,30	112,41	13384	7,27	0	6301	0,44
Ins_B5_NC 2500_5	5948	134,97	13976	1,34	126,63	13480	8,39	0	5948	0,47
Promedio		140,86	11007,87	0,73	136,46	10804,00	4,03	0,00	4624,00	0,31

Tabla 4.16: RPD, tiempos de asignación y tiempos de computación de las instancias grandes. Tabla de elaboración propia.

En cuanto a los resultados ofrecidos en función de los contendores a asignar, vemos que el tiempo de asignación total crece linealmente, con un crecimiento entre 2000 y 3000 u.t. por cada 500 contenedores, esto se puede observar en el gráfico de la figura 4.3. Vemos también que la diferencia entre las soluciones ofrecidas aumenta conforme mayor es el número de contenedores. El algoritmo de proximidad ofrece muy buenas soluciones cuando se trata de instancias grandes, mientras que los algoritmos aleatorio e iterativo ofrecen unas soluciones similares.

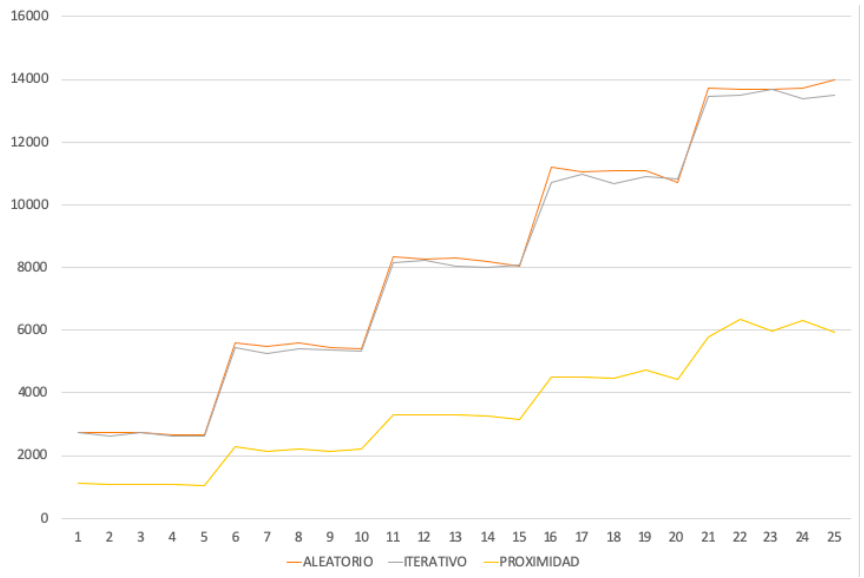

Figura 4.3: Evolución de los tiempos totales de asignación s de las instancias según el algoritmo utilizado.
Figura de elaboración propia

De los tiempos de computación, cabe destacar que crecen de una manera exponencial en función del número de contenedores. Esto lo podemos ver en la gráfica de la figura 4.4. Cabe destacar que el tiempo de computación del algoritmo de proximidad es el mejor y, esta diferencia, es más notable cuanto mayor sea el número de contenedores de la instancia.

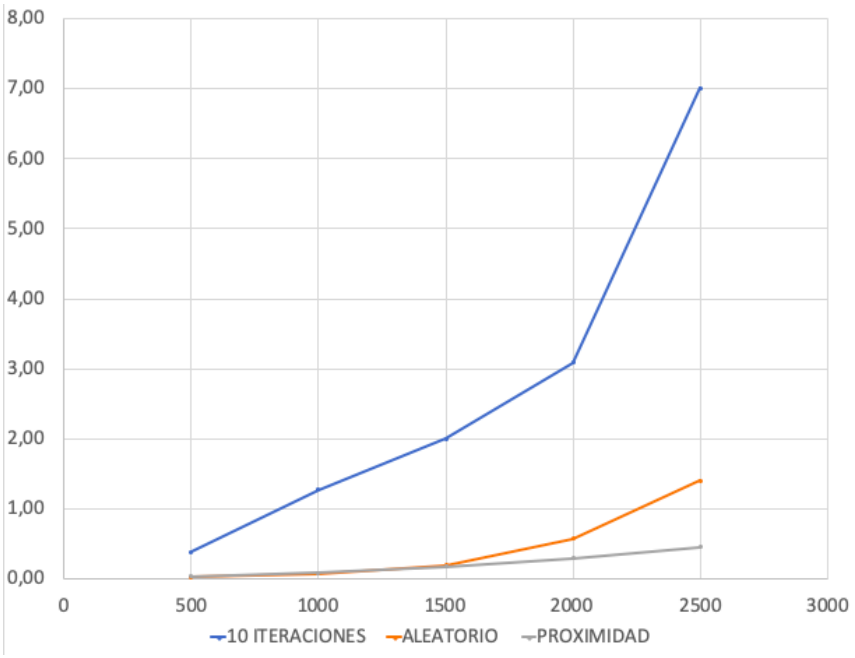

Figura 4.4: Tiempos de cómputo promedios de las instancias para cada tamaño de contenedores
Figura de elaboración propia

Con el gráfico de dispersión de la figura 4.5 podemos observar que con instancias pequeñas se obtienen peores resultados con el algoritmo aleatorio, sin embargo, conforme aumenta el número de contenedores, la diferencia comienza a ser menos destacada.

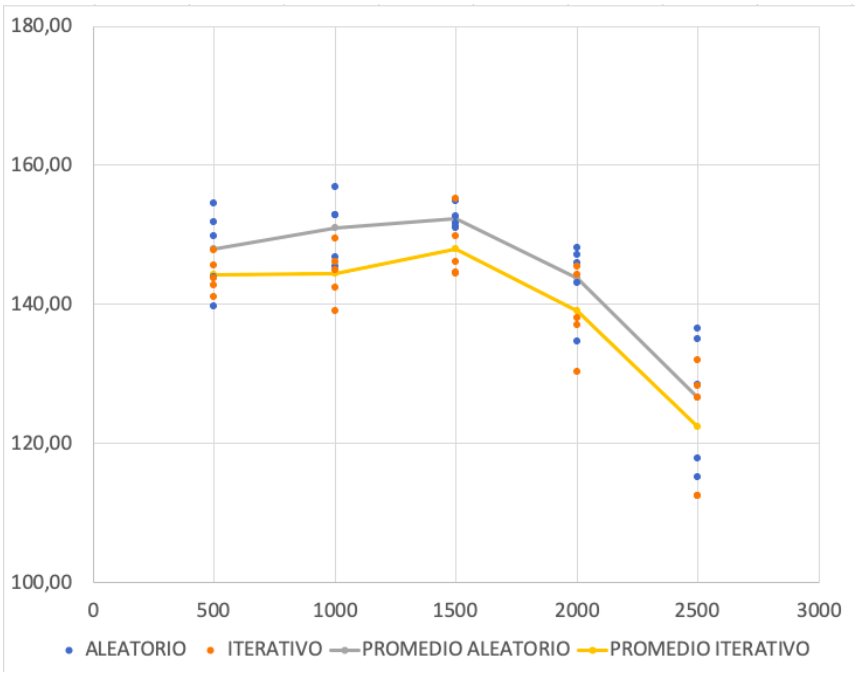

Figura 4.5: Gráfico de dispersión del RPD de los algoritmos aleatorio e iterativo
Figura de elaboración propia

La finalidad del desarrollo de estos algoritmos es obtener una buena solución para el problema de asignación. Así pues, pese a que es interesante tener datos sobre el tiempo de cómputo no resulta de gran utilidad a la hora de comparar los algoritmos en este

caso, ya que todos son muy rápidos. Hemos podido comprobar que estos tiempos no son significativos, puesto que hablamos de milisegundos.

La parte del análisis donde debemos centrarnos se trata de aquella que muestra la solución obtenida por cada algoritmo. Tras comprobar los resultados en las tablas y gráficas anteriores obtenemos como conclusión que la aplicación del algoritmo de aproximación proporciona un mejor resultado que el resto.

De tal manera, se puede comprobar que la aplicación de métodos heurísticos para la resolución de problemas de optimización permite obtener unos mejores resultados. En concreto, aplicando el algoritmo de proximidad hemos podido obtener un amplio margen de mejora respecto a los algoritmos aleatorio e iterativo.

CAPÍTULO 5
Conclusiones y futuras líneas de investigación

Durante este trabajo hemos mostrado el gran impacto socio-económico que representa una terminal portuaria para el comercio y la economía mundiales. Hemos mostrado la estructura de una terminal portuaria y sus elementos, además de los diferentes problemas a estudiar dentro de ella. En concreto hemos analizado con mayor profundidad el problema de asignación de contenedores en una terminal portuaria y la importancia de la optimización de este proceso.

Hemos expuesto tres métodos heurísticos que permiten ofrecer una solución real al problema de asignación de contenedores de una manera rápida. Se ha utilizado la desviación porcentual sobre la mejor solución para comparar los métodos y el mejor ha resultado ser el algoritmo de proximidad. La razón principal por la cual se ha conseguido una mejor solución ha sido la implementación de una manera correcta de los datos disponibles del problema. Esto ha provocado un mejor funcionamiento.

En el contexto de una terminal portuaria, una gestión racional de los recursos existentes permitiría aumentar el rendimiento de una terminal, permitiendo obtener una competitividad mayor.

Los métodos presentados se han apoyado en un ejemplo númerico, igual para todos ellos, lo cual ha permitido una mejor comprensión de su funcionamiento y han permitido realizar una primera comparación.

Una de las limitaciones importantes del trabajo es el hecho de haber considerado la asignación de manera secuencial. Por ejemplo, llega un camión a la terminal, se asigna un bloque al contenedor que tiene que descargar el camión y hasta que no termina no se comienza con la asignación del siguiente camión. En un entorno real, los camiones pueden funcionar de manera independiente. Si sus contenedores son asignados a distintos bloques, podrían desplazarse a cada bloque de manera simultánea varios camiones, sin esperar a que termine uno para comenzar el siguiente.

Una de las futuras líneas será aplicar otros tipos de métodos para realizar una mejor asignación, teniendo en cuenta otros factores. Por ejemplo, teniendo en cuenta que contenedores con un mismo destino no deben ir a un mismo bloque, ya que esto generaría un cuello de botella al tener que abandonar el bloque, debido a que la grúa del patio de contenedores suele ser el cuello de botella a la hora de introducir/sacar contenedores del mismo. Así, estando en diferentes bloques, serán diferentes grúas de patio las que trabajen con contenedores con un mismo destino.

Dada la importancia de estos procesos de optimización y los actuales métodos utilizados en las terminales portuarias de diferentes lugares, otra de las futuras líneas de

este trabajo consistirá en la implementación de los métodos diseñados en una aplicación móvil multiplataforma. Esta aplicación móvil podría ser utilizada por los operarios del puerto para facilitar sus labores. A su vez, este trabajo de fin de grado servirá como base para el posterior desarrollo del trabajo de fin de grado del grado en Ingeniería Informática, el cual se trata del desarrollo de dicha aplicación móvil multiplataforma.

Bibliografía

[1] BBC. Barcos cargueros, los grandes monstruos del océano, 2013.

[2] Anna Bottasso, Maurizio Conti, Claudio Ferrari, Olaf Merk, and Alessio Tei. The impact of port throughput on local employment: Evidence from a panel of European regions. *Transport Policy*, 27:32–38, may 2013.

[3] Héctor J. Carlo, Iris F.A. Vis, and Kees Jan Roodbergen. Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research*, 235(2):412–430, jun 2014.

[4] Fundación de las cajas de ahorros. *Papeles de economía española*. 2012.

[5] Diario El Canal. El puerto de Valencia propone convertirse en smart port con su Plan Estratégico 2030, 2020.

[6] González Laxe Fernando. Economia maritima y tipologias portuarias, 2002.

[7] JCV. Smart Ports, el futuro del transporte marítimo, 2019.

[8] V J.M. Conectados con mil puertos, 2020.

[9] Belén Melian, Jose A. Moreno Pérez, and J. Marcos Moreno Vega. *Metaheuristics: A global view*.

[10] Daniel Seong-Hyeok Moon and Jong Kyun Woo. The impact of port operations on efficient ship operation from both economic and environmental perspectives. *Maritime Policy & Management*, 41(5):444–461, jul 2014.

[11] Matthew E.H. Petering and Katta G. Murty. Effect of block length and yard crane deployment systems on overall performance at a seaport container transshipment terminal. *Computers & Operations Research*, 36(5):1711–1725, may 2009.

[12] PortOfRotterdam. Building a sustainable port.

[13] Andrea Puchades. MEDIO AMBIENTE | Valenciaport consolida sus políticas de sostenibilidad, may 2020.

[14] RFL. La evolución de los puertos hacia los smarts ports, 2020.

[15] S. Saurí and E. Martín. Space allocating strategies for improving import yard performance at marine terminals. *Transportation Research Part E: Logistics and Transportation Review*, 47(6):1038–1057, nov 2011.

[16] ValenciaPort. Plan estratégico, sep 2011.

[17] ValenciaPort. La nueva terminal de contenedores creará 14.453 puestos de trabajo, feb 2020.

[18] Esteban Chapapría Vicent, Domingo Aleixandre Jesús, Puertas Medina Rosa Mª, and Martí Selva Mª Luisa. *Puerto de Valencia: La nueva terminal en la ampliación norte. Sostenibilidad, efectos socioeconómicos y necesidades.* 2020.

[19] Andrés Felipe Zapata-Tapasco, Sandra Milena Pérez-Londoño, and Juan José Mora-Flórez. A fault location method applied in power distribution systems based on k-NN classifiers parameterized using genetic algorithms and the reactance estimation, 2014.

[20] Lu Zhen, Xinjia Jiang, Loo Hay Lee, and Ek Peng Chew. A Review on Yard Management in Container Terminals. *Industrial Engineering and Management Systems*, 12(4):289–304, dec 2013.

APÉNDICE A
Código de los algoritmos

Listing A.1: Algoritmo aleatorio de asignación

```
import sys
import csv
import random
from time import time
import pandas as pd
import itertools
import bisect
import operator


listado = sys.argv[1]

# Indica si esta en el lado de tierra o no
def esLadoTierra(posicion,bloque):
    if salientesTipos[bloque][2] < posicion :
        listaBloques[bloque][1] = listaBloques[bloque][1] - 1
        return True
    else: return False

# Indica si esta en el lado de mar o no
def esLadoMar(posicion,bloque):
    if salientesTipos[bloque][2] >= posicion :
        listaBloques[bloque][0] = listaBloques[bloque][0] - 1
        return True
    else: return False

# Saca un contenedor del bloque en un instante
def salida(tAnt, tAct, listaSalientes1D, listaSalientes2D, nB, lB, sTipos):
    cont = 0
    sacar = []
    # Metemos en array sacar aquellos tiempos que hay que sacar del bloque
    for listaSalientes1DIndex in range(len(listaSalientes1D)):
        if ((listaSalientes1D[listaSalientes1DIndex] <= tAct) and
        (listaSalientes1D[listaSalientes1DIndex] >= tAnt)):
            sacar.append(listaSalientes1D[listaSalientes1DIndex])
    # Para cada tiempo de sacar, hay que buscar a que bloque corresponde
    # y si es a lado de mar o a lado de tierra.
    for s in range(len(sacar)):
        # hay que buscarlo en tiemposSalientes, para encontrar el bloque.
        indiceSacar = 0
        for nbl in range(numBloques):
            # Si este es el bloque
            if ((sacar[s] in tiemposSalientes[nbl]) and
            esLadoTierra(tiemposSalientes[nbl].index(int(sacar[s])),nbl)):
                if((sacar[s] not in listaSacados)):
                    for contar in range(contar_veces(sacar[s],tiemposSalientes1)):
                        listaSacados.append(sacar[s])

            if ((sacar[s] in tiemposSalientes[nbl]) and
            esLadoMar(tiemposSalientes[nbl].index(int(sacar[s])),nbl)):
                if((sacar[s] not in listaSacados)):
                    for contar in range(contar_veces(sacar[s],tiemposSalientes1)):
                        listaSacados.append(sacar[s])

# Saca un contenedor de los que ha entrado nuevos en el bloque en un instante
def salidaNuevo(tAntN, tActN, listaSalientes1DN, nBN, lBN, sNuevo, sNTierra, sNMar):
    contN = 0
    sacarN = []
    # Metemos en array sacar aquellos tiempos que hay que sacar del bloque
    for listaSalientes1DIndexN in range(len(listaSalientes1DN)):
        if ((listaSalientes1DN[listaSalientes1DIndexN][0] <= tActN) and
        (listaSalientes1DN[listaSalientes1DIndexN][0] >= tAntN)):
            sacarN.append(listaSalientes1DN[listaSalientes1DIndexN])

    # Para cada tiempo de sacar hay que buscar a que bloque corresponde y
    # si es al lado de mar o de tierra.
    # Para cada elemento de sacarN
    for sN in range(len(sacarN)):
        bloqueN = 0
        tipo = 0
        for lN in sacarN:
            # Sacamos su bloque y el tipo de contenedor que es
```

```
73              bloqueN =  sacarN[sacarN.index(lN)][2]
74              tipo = sacarN[sacarN.index(lN)][1]
75
76              if sacarN[sacarN.index(lN)] not in sNuevo :
77
78                  if sacarN[sacarN.index(lN)][1] == 1: #Si es 1 hay que sacarlo del lado de mar
79                      # El espacio disponible aumenta
80                      lBN[2][0] = lBN[2][0] + 1
81                      # Quitamos el contenedor de proximasSalidas
82                      listaSalientes1DN = listaSalientes1DN[1:]
83                      sNuevo.append(sacarN[sacarN.index(lN)])
84                      sNMar.append(sacarN[sacarN.index(lN)])
85
86                  elif sacarN[sacarN.index(lN)][1] == 2: #Si es 2 hay que sacarlo del lado de tierra
87                      # El espacio disponible aumenta
88                      lBN[2][1] = lBN[2][1] + 1
89                      # Quitamos el contenedor de proximasSalidas
90                      listaSalientes1DN = listaSalientes1DN[1:]
91                      sNuevo.append(sacarN[sacarN.index(lN)])
92                      sNTierra.append(sacarN[sacarN.index(lN)])
93
94
95
96
97  # Devuelve una lista con los repetidos
98  def encontrarTiemposRep(lista):
99      repetido = []
100     unico = []
101     for x in lista:
102         if x not in unico:
103             unico.append(x)
104         else:
105             repetido.append(x)
106     return repetido
107
108 # Devuelve el numero de veces que se repite un tiempo de salida
109 def contar_veces(elemento, lista):
110     veces = 0
111     for i in lista:
112         if elemento == i: veces += 1
113     return veces
114
115 for i in range(1):
116
117     # Abro archivo con nombre de listas
118     with open(listado, "r") as f:
119         listaInstancias = f.readlines()
120         listaInstancias = [x.strip() for x in listaInstancias]
121         tiemposInstancias = [[]]
122         # para cada uno de los archivos
123         for i in listaInstancias:
124
125             tiempoInicialInstancia = time()
126             tInicialProceso = time()
127
128             # abro archivo
129             with open(i) as datos:
130
131                 # inicializacion de variables necesarias:
132                 numBloques = 0
133                 numCont = 0
134                 numRows = 0
135                 numCol = 0
136                 bloquesMar = []
137                 bloquesTierra = []
138                 saleSea = 0
139                 saleLand = 0
140                 saleMarPrimeros = 0
141                 salientesTipos = [[]]
142                 tiemposSalientes = [[]]
143                 tSalientesRes = []
144                 TsalenSea = []
145                 TsalenLand = []
146                 listaContenedores = []
147                 listaBloques = [[]]
148                 listaSacados = []
149                 listaSacadosLand = []
150                 listaSacadosSea = []
151                 contenedoresAsignados = []
152                 solucion = []
153                 tiemposRepetidos = []
154                 proximasSalidas = []
155                 proximasSalidasDatos = [[]]
156                 listaSacadosNuevos = [[]]
157                 listaSacadosNMar = [[]]
158                 listaSacadosNTierra = [[]]
159                 listaSacadosNuevos = listaSacadosNuevos [1:]
160                 listaSacadosNTierra = listaSacadosNTierra [1:]
161                 listaSacadosNMar = listaSacadosNMar [1:]
162                 tiemposAsignados = []
163
164                 # para cada linea del archivo
165                 for linea in datos:
166
167                     # Numero de bloques
168                     if linea.startswith("number␣of␣blocks"):
169                         instanciaNumBloques = next(datos).strip()
170                         instanciaNumBloques = instanciaNumBloques.replace("\t",",")
171                         numBloques = int(instanciaNumBloques)
172
173                     # Numero de contenedores
```

```
174                     if linea.startswith("number␣of␣containers"):
175                         instanciaNumCont = next(datos).strip()
176                         instanciaNumCont = instanciaNumCont.replace("\t",",")
177                         numCont = instanciaNumCont
178 
179                     # Numero de filas
180                     if linea.startswith("number␣of␣rows"):
181                         instanciaNumRows = next(datos).strip()
182                         instanciaNumRows = instanciaNumRows.replace("\t",",")
183                         numRows = instanciaNumRows
184 
185                     # Numero de columnas
186                     if linea.startswith("number␣of␣columns"):
187                         instanciaNumCol = next(datos).strip()
188                         instanciaNumCol = instanciaNumCol.replace("\t",",")
189                         numCol = instanciaNumCol
190 
191                     # Datos sobre los bloques
192                     for b in range(int(numBloques+1)):
193                         if linea.startswith("#Block␣␣␣␣" + str(b)):
194                             instanciaFreeSea = next(datos).strip()
195                             instanciaFreeSea = instanciaFreeSea.replace("\t",",")
196                             instanciaFreeLand = next(datos).strip()
197                             instanciaFreeLand = instanciaFreeLand.replace("\t",",")
198                             instanciasSaleMar = next(datos).strip()
199                             instanciasSaleMar = instanciasSaleMar.replace("\t",",")
200                             instanciasSaleLand = next(datos).strip()
201                             instanciasSaleLand = instanciasSaleLand.replace("\t",",")
202                             instanciasSaleMarPrimeros = next(datos).strip()
203                             instanciasSaleMarPrimeros = instanciasSaleMarPrimeros.replace("\t",",")
204                             instanciaFreeSea = list(map(str,instanciaFreeSea.split(",")))
205                             instanciaFreeLand = list(map(str,instanciaFreeLand.split(",")))
206                             instanciasSaleMar = list(map(str,instanciasSaleMar.split(",")))
207                             instanciasSaleLand = list(map(str,instanciasSaleLand.split(",")))
208                             instanciasSaleMarPrimeros = list(map(str,instanciasSaleMarPrimeros.split(",")))
209                             instanciaBloque = [instanciaFreeSea[1],instanciaFreeLand[1]]
210                             instanciaBloquesMar = instanciaFreeSea
211                             instanciaBloquesTierra = instanciaFreeLand
212                             instanciaBloque = [int(x) for x in instanciaBloque]
213                             bloquesMar.append(instanciaBloquesMar[1])
214                             bloquesTierra.append(instanciaBloquesTierra[1])
215                             listaBloques.append(instanciaBloque)
216                             bloquesMar = [int(x) for x in bloquesMar]
217                             bloquesTierra = [int(x) for x in bloquesTierra]
218                             saleSea = int(instanciasSaleMar[1])
219                             saleLand = int(instanciasSaleLand[1])
220                             saleMarPrimeros = int(instanciasSaleMarPrimeros[2])
221                             salientesTipos.append([int(saleSea),int(saleLand),int(saleMarPrimeros)])
222                             instanciatiempos = []
223                             instanciaTSaliente = next(datos).strip()
224                             for j in range((int(saleSea)+int(saleLand))):
225                                 instanciaTSaliente = next(datos).strip()
226                                 instanciaTSaliente = instanciaTSaliente.replace("\t",",")
227                                 instanciatiempos.append(instanciaTSaliente)
228                                 tSalientesRes.append(instanciaTSaliente)
229                             instanciatiempos = [int(x) for x in instanciatiempos]
230                             TsalenSea.append(instanciatiempos[:int(saleMarPrimeros)])
231                             TsalenLand.append(instanciatiempos[int(saleMarPrimeros):])
232                             tiemposSalientes.append(instanciatiempos)
233 
234                     # Datos sobre los contenedores
235                     if linea.startswith("#Type"):
236                         for n in range(int(numCont)):
237                             instanciaCont = next(datos).strip()
238                             instanciaCont = instanciaCont.replace("\t",",")
239                             instanciaCont = list(map(int,instanciaCont.split(",")))
240                             listaContenedores.append(instanciaCont)
241                         pass
242                     pass
243 
244                 tiemposSalientes = tiemposSalientes[1:]
245                 salientesTipos = salientesTipos[1:]
246                 listaBloques = listaBloques[1:]
247                 listaCDesordenada = listaContenedores
248                 listaContenedores.sort(key = operator.itemgetter(1, 2))
249                 tiemposSalientes1 =    sorted(list(itertools.chain.from_iterable(tiemposSalientes)))
250                 tiemposRepetidos = encontrarTiemposRep(tiemposSalientes1)
251                 tActual = 0
252                 tAnterior = 0
253                 tProximo = 0
254 
255                 for c in listaContenedores:
256                     # Esperamos hasta que haya contenedor disponible
257                     while (tActual<c[1]):
258                         salida(tAnterior,tActual,tiemposSalientes1,tiemposSalientes,numBloques,
259                         listaBloques, salientesTipos)
260                         tAnterior = tActual
261                         tActual = tActual + 1
262 
263                     # Definimos bloqueAsignado para indicar si el bloque ya ha sido asignado.
264                     # Estara a 0 cuando aun no haya sido asignado.
265                     bloqueAsignado = 0
266 
267                     # Mientras que no este asignado seguiremos iterando
268                     while bloqueAsignado == 0:
269                         # Elegimos un bloque aleatorio entre 1 y numBloques
270                         bloqueAleatorio = random.randint(1,numBloques)
271 
272                         # Sabemos el lado al que va segun su Ci, si es 1 es a mar, si es 2 es a tierra
273                         if c[0] == 1:
274                             # Hay que mirar que haya hueco en el bloque elegido en la parte de mar
```

```
275                             if listaBloques[bloqueAleatorio-1][0] > 0 :
276                             # Hay hueco
277                             # Restamos un espacio libre en el lado de mar de ese bloque
278                             listaBloques[bloqueAleatorio-1][0] = int(listaBloques[bloqueAleatorio-1][0]) - 1
279                             # Decimos que el contenedor tiene bloque asignado
280                             bloqueAsignado = 1
281                             tiemposAsignados.append(c[1])
282                             # Anyadimos el tiempo ri a una lista que nos mostrara
283                             #los contenedores que si hemos asignado
284                             bisect.insort(proximasSalidas,([tActual+c[2],c[0],bloqueAleatorio]))
285                             #Meto el contenedor en la lista de proximas salidas,
286                             #sabre por donde llega, tiempo de salida y bloque en el que esta asigando
287                             tProximo = tActual + c[3+int(bloqueAleatorio)]
288
289                         # Si no ha ido a mar, habra ido a tierra
290                         elif c[0] == 2:
291                             # Hay que mirar que haya hueco en el bloque elegido en la parte de tierra
292                             if listaBloques[bloqueAleatorio-1][1] > 0 :
293                                 # Hay hueco
294                                 # Restamos un espacio libre en el lado de mar de ese bloque
295                                 listaBloques[bloqueAleatorio-1][1] = int(listaBloques[bloqueAleatorio-1][1]) - 1
296                                 # Decimos que el contenedor tiene bloque asignado
297                                 bloqueAsignado = 1
298                                 tiemposAsignados.append(c[1])
299                                 # Anyadimos el tiempo ri a una lista que nos
300                                 #mostrara los contenedores que si hemos asignado
301                                 bisect.insort(proximasSalidas,([tActual+c[2],c[0],bloqueAleatorio]))
302                                 #Meto el contenedor en la lista de proximas salidas,
303                                 #sabre por donde llega, tiempo de salida y bloque en el que esta asigandox
304                                 tProximo = tActual + c[3+int(bloqueAleatorio)]
305
306                         # Tenemos que ver si va a haber salidas entre el instante tActual y
307                         #el instante en el que llegue el contenedor al bloque, tProximo
308                         salida(tActual, tProximo, tiemposSalientes1, tiemposSalientes, numBloques,
309                             listaBloques, salientesTipos)
310                         salidaNuevo(tActual, tProximo, proximasSalidas, numBloques,
311                             listaBloques, listaSacadosNuevos, listaSacadosNTierra, listaSacadosNMar)
312                         # Actualizamos los tiempos
313                         tAnterior = tActual
314                         tActual = tProximo
315                         # Anyadimos el componente a la solucion
316                         solucion.append([c,bloqueAleatorio,tActual])
317
318                 tiempoFinalInstancia = time()
319                 tiempoTotalInstancia = tiempoFinalInstancia -                 tiempoInicialInstancia
320                 tiemposInstancias.append([str(i),solucion[-1][-1],tiempoTotalInstancia])
321                 print("El tiempo del proceso de asignacion de esta instancia es: " + str(solucion[-1][-1]))
322
323         tiemposInstancias = tiemposInstancias[1:]
324         df = pd.DataFrame(tiemposInstancias, columns = ["Instancia", "Tiempo u.t.", "Tiempo"])
325         df.to_excel ("TiemposRandom.xlsx", index = False, header=True)
326
327 tFinalProceso = time()
328 tTotalProceso = tFinalProceso - tInicialProceso
329 print("\nEl tiempo total del proceso de asignacion: " + str(tTotalProceso) + " ms.")
```

Listing A.2: Algoritmo iterativo de asignación aleatoria

```
1
2  import sys
3  import csv
4  import random
5  from time import time
6  import pandas as pd
7  import numpy as np
8  import itertools
9  import bisect
10 import operator
11
12
13 listado = sys.argv[1]
14 iteraciones = sys.argv[2]
15
16 # Indica si esta en el lado de tierra o no
17 def esLadoTierra(posicion,bloque):
18     if salientesTipos[bloque][2] < posicion :
19         listaBloques[bloque][1] = listaBloques[bloque][1] - 1
20         return True
21     else: return False
22
23 # Indica si esta en el lado de mar o no
24 def esLadoMar(posicion,bloque):
25     if salientesTipos[bloque][2] >= posicion :
26         listaBloques[bloque][0] = listaBloques[bloque][0] - 1
27         return True
28     else: return False
29
30 # Saca un contenedor del bloque en un instante
31 def salida(tAnt, tAct, listaSalientes1D, listaSalientes2D, nB, lB, sTipos):
32     cont = 0
33     sacar = []
34     #print("Tiempo anterior: " + str(tAnt))
35     #print("Tiempo actual: " + str(tAct))
36     # Metemos en array sacar aquellos tiempos que hay que sacar del bloque
37     for listaSalientes1DIndex in range(len(listaSalientes1D)):
38         if ((listaSalientes1D[listaSalientes1DIndex] <= tAct) and
39         (listaSalientes1D[listaSalientes1DIndex] >= tAnt)):
40             sacar.append(listaSalientes1D[listaSalientes1DIndex])
41     # Para cada tiempo de sacar, hay que buscar a que bloque corresponde y
```

```
42      #si es a lado de mar o a lado de tierra.
43      for s in range(len(sacar)):
44          # sacar[s] es el valor que queremos sacar
45          # hay que buscarlo en tiemposSalientes, para encontrar el bloque.
46          indiceSacar = 0
47          for nbl in range(numBloques):
48          # Si este es el bloque
49              if ((sacar[s] in tiemposSalientes[nbl]) and
50              esLadoTierra(tiemposSalientes[nbl].index(int(sacar[s])),nbl)):
51                  if((sacar[s] not in listaSacados)):
52                      for contar in range(contar_veces(sacar[s],tiemposSalientes1)):
53                          listaSacados.append(sacar[s])
54
55              if ((sacar[s] in tiemposSalientes[nbl]) and
56              esLadoMar(tiemposSalientes[nbl].index(int(sacar[s])),nbl)):
57                  if((sacar[s] not in listaSacados)):
58                      for contar in range(contar_veces(sacar[s],tiemposSalientes1)):
59                          listaSacados.append(sacar[s])
60
61  # Saca un contenedor de los que ha entrado nuevos en el bloque en un instante
62  def salidaNuevo(tAntN, tActN, listaSalientes1DN, nBN, lBN, sNuevo, sNTierra, sNMar):
63      contN = 0
64      sacarN = []
65      # Metemos en array sacar aquellos tiempos que hay que sacar del bloque
66      for listaSalientes1DIndexN in range(len(listaSalientes1DN)):
67          if ((listaSalientes1DN[listaSalientes1DIndexN][0] <= tActN) and
68          (listaSalientes1DN[listaSalientes1DIndexN][0] >= tAntN)):
69              sacarN.append(listaSalientes1DN[listaSalientes1DIndexN])
70      # Para cada tiempo de sacar hay que buscar a que bloque corresponde y
71      # si es al lado de mar o de tierra.
72      # Para cada elemento de sacarN
73      for sN in range(len(sacarN)):
74          bloqueN = 0
75          tipo = 0
76          for lN in sacarN:
77              # Sacamos su bloque y el tipo de contenedor que es
78              bloqueN =  sacarN[sacarN.index(lN)][2]
79              tipo = sacarN[sacarN.index(lN)][1]
80              if sacarN[sacarN.index(lN)] not in sNuevo :
81                  if sacarN[sacarN.index(lN)][1] == 1:
82                      # El espacio disponible aumenta
83                      lBN[2][0] = lBN[2][0] + 1
84                      # Quitamos el contenedor de proximasSalidas
85                      listaSalientes1DN = listaSalientes1DN[1:]
86                      sNuevo.append(sacarN[sacarN.index(lN)])
87                      sNMar.append(sacarN[sacarN.index(lN)])
88                  elif sacarN[sacarN.index(lN)][1] == 2:
89                      # El espacio disponible aumenta
90                      lBN[2][1] = lBN[2][1] + 1
91                      # Quitamos el contenedor de proximasSalidas
92                      listaSalientes1DN = listaSalientes1DN[1:]
93                      sNuevo.append(sacarN[sacarN.index(lN)])
94                      sNTierra.append(sacarN[sacarN.index(lN)])
95
96  # Devuelve una lista con los repetidos
97  def encontrarTiemposRep(lista):
98      repetido = []
99      unico = []
100     for x in lista:
101         if x not in unico: unico.append(x)
102         else: repetido.append(x)
103     return repetido
104
105 # Devuelve el numero de veces que se repite un tiempo de salida
106 def contar_veces(elemento, lista):
107     veces = 0
108     for i in lista:
109         if elemento == i:
110             veces += 1
111     return veces
112
113 tiemposInstancias = []
114
115 for it in range(int(iteraciones)):
116     tInicialProceso = time()
117     # abro archivo con nombre de listas
118     with open(listado, "r") as f:
119         listaInstancias = f.readlines()
120         listaInstancias = [x.strip() for x in listaInstancias]
121         # para cada uno de los archivos
122         for i in listaInstancias:
123             tiempoInicialInstancia = time()
124             # abro archivo
125             with open(i) as datos:
126                 # inicializacion de variables necesarias:
127                 numBloques = 0
128                 numCont = 0
129                 numRows = 0
130                 numCol = 0
131                 bloquesMar = []
132                 bloquesTierra = []
133                 saleSea = 0
134                 saleLand = 0
135                 saleMarPrimeros = 0
136                 salientesTipos = []
137                 tiemposSalientes = []
138                 tSalientesRes = []
139                 TsalenSea = []
140                 TsalenLand = []
141                 listaContenedores = []
142                 listaBloques = [
```

```
143                 listaSacados = []
144                 listaSacadosLand = []
145                 listaSacadosSea = []
146                 contenedoresAsignados = []
147                 solucion = []
148                 tiemposRepetidos = []
149                 proximasSalidas = []
150                 listaSacadosNuevos = []
151                 listaSacadosNMar = []
152                 listaSacadosNTierra = []
153                 tiemposAsignados = []
154                 # para cada linea del archivo
155                 for linea in datos:
156                     # Numero de bloques
157                     if linea.startswith("number␣of␣blocks"):
158                         instanciaNumBloques = next(datos).strip()
159                         instanciaNumBloques = instanciaNumBloques.replace("\t",",")
160                         numBloques = int(instanciaNumBloques)
161                     # Numero de contenedores
162                     if linea.startswith("number␣of␣containers"):
163                         instanciaNumCont = next(datos).strip()
164                         instanciaNumCont = instanciaNumCont.replace("\t",",")
165                         numCont = instanciaNumCont
166                     # Numero de filas
167                     if linea.startswith("number␣of␣rows"):
168                         instanciaNumRows = next(datos).strip()
169                         instanciaNumRows = instanciaNumRows.replace("\t",",")
170                         numRows = instanciaNumRows
171                     # Numero de columnas
172                     if linea.startswith("number␣of␣columns"):
173                         instanciaNumCol = next(datos).strip()
174                         instanciaNumCol = instanciaNumCol.replace("\t",",")
175                         numCol = instanciaNumCol
176                     # Datos sobre los bloques
177                     for b in range(int(numBloques+1)):
178                         if linea.startswith("#Block␣␣␣␣␣" + str(b)):
179                             instanciaFreeSea = next(datos).strip()
180                             instanciaFreeSea = instanciaFreeSea.replace("\t",",")
181                             instanciaFreeLand = next(datos).strip()
182                             instanciaFreeLand = instanciaFreeLand.replace("\t",",")
183                             instanciasSaleMar = next(datos).strip()
184                             instanciasSaleMar = instanciasSaleMar.replace("\t",",")
185                             instanciasSaleLand = next(datos).strip()
186                             instanciasSaleLand = instanciasSaleLand.replace("\t",",")
187                             instanciasSaleMarPrimeros = next(datos).strip()
188                             instanciasSaleMarPrimeros =
189                             instanciasSaleMarPrimeros.replace("\t",",")
190                             instanciaFreeSea = list(map(str,instanciaFreeSea.split(",")))
191                             instanciaFreeLand = list(map(str,instanciaFreeLand.split(",")))
192                             instanciasSaleMar = list(map(str,instanciasSaleMar.split(",")))
193                             instanciasSaleLand = list(map(str,instanciasSaleLand.split(",")))
194                             instanciasSaleMarPrimeros = list(map(str,instanciasSaleMarPrimeros.split(",")))
195                             instanciaBloque = [instanciaFreeSea[1],instanciaFreeLand[1]]
196                             instanciaBloquesMar = instanciaFreeSea
197                             instanciaBloquesTierra = instanciaFreeLand
198                             instanciaBloque = [int(x) for x in instanciaBloque]
199                             bloquesMar.append(instanciaBloquesMar[1])
200                             bloquesTierra.append(instanciaBloquesTierra[1])
201                             listaBloques.append(instanciaBloque)
202                             bloquesMar = [int(x) for x in bloquesMar]
203                             bloquesTierra = [int(x) for x in bloquesTierra]
204                             saleSea = int(instanciasSaleMar[1])
205                             saleLand = int(instanciasSaleLand[1])
206                             saleMarPrimeros = int(instanciasSaleMarPrimeros[2])
207                             salientesTipos.append([int(saleSea),int(saleLand),int(saleMarPrimeros)])
208                             instanciatiempos = []
209                             instanciaTSaliente = next(datos).strip()
210                             for j in range((int(saleSea)+int(saleLand))):
211                                 instanciaTSaliente = next(datos).strip()
212                                 instanciaTSaliente = instanciaTSaliente.replace("\t",",")
213                                 instanciatiempos.append(instanciaTSaliente)
214                                 tSalientesRes.append(instanciaTSaliente)
215                             instanciatiempos = [int(x) for x in instanciatiempos]
216                             TsalenSea.append(instanciatiempos[:int(saleMarPrimeros)])
217                             TsalenLand.append(instanciatiempos[int(saleMarPrimeros):])
218                             tiemposSalientes.append(instanciatiempos)
219                     # Datos sobre los contenedores
220                     if linea.startswith("#Type"):
221                         for n in range(int(numCont)):
222                             instanciaCont = next(datos).strip()
223                             instanciaCont = instanciaCont.replace("\t",",")
224                             instanciaCont = list(map(int,instanciaCont.split(",")))
225                             listaContenedores.append(instanciaCont)
226                         pass
227                     pass
228
229                 listaCDesordenada = listaContenedores
230                 #Ordenamos los contenedoers por tiempo de llegada y salida
231                 listaContenedores.sort(key = operator.itemgetter(1, 2))
232                 tiemposSalientes1 = sorted(list(itertools.chain.from_iterable(tiemposSalientes)))
233                 tiemposRepetidos = encontrarTiemposRep(tiemposSalientes1)
234                 tActual = 0
235                 tAnterior = 0
236                 tProximo = 0
237
238                 for c in listaContenedores:
239                     # Esperamos hasta que haya contenedor disponible
240                     while (tActual<c[1]):
241                         salida(tAnterior, tActual, tiemposSalientes1, tiemposSalientes,
242                             numBloques, listaBloques, salientesTipos)
243                         tAnterior = tActual
```

```
                        tActual = tActual + 1
                    # Definimos bloqueAsignado para indicar si el bloque ya ha sido asignado.
                    # Estara a 0 cuando aun no haya sido asignado.
                    bloqueAsignado = 0
                    # Mientras que no este asignado seguiremos iterando
                    while bloqueAsignado == 0:
                        # Elegimos un bloque aleatorio entre 1 y numBloques
                        bloqueAleatorio = random.randint(1,numBloques)
                        # Sabemos el lado al que va segun su Ci, si es 1 es a mar, si es 2 es a tierra
                        if c[0] == 1:
                            # Hay que mirar que haya hueco en el bloque elegido en la parte de mar
                            if listaBloques[bloqueAleatorio-1][0] > 0 :
                                # Hay hueco
                                # Restamos un espacio libre en el lado de mar de ese bloque
                                listaBloques[bloqueAleatorio-1][0] = int(listaBloques[bloqueAleatorio-1][0]) - 1
                                # Decimos que el contenedor tiene bloque asignado
                                bloqueAsignado = 1
                                tiemposAsignados.append(c[1])
                                # Anyadimos el tiempo ri a una lista que nos mostrara
                                # los contenedores que si hemos asignado
                                bisect.insort(proximasSalidas,([c[2],c[0],bloqueAleatorio]))
                                #Meto el contenedor en la lista de proximas salidas,
                                #sabre por donde llega, tiempo de salida y bloque en el que esta asigando
                                tProximo = tActual + c[3+int(bloqueAleatorio)]

                        # Si no ha ido a mar, habra ido a tierra
                        elif c[0] == 2:
                                # Hay que mirar que haya hueco en el bloque elegido en la parte de tierra
                                if listaBloques[bloqueAleatorio-1][1] > 0 :
                                # Hay hueco
                                # Restamos un espacio libre en el lado de mar de ese bloque
                                listaBloques[bloqueAleatorio-1][1] = int(listaBloques[bloqueAleatorio-1][1]) - 1
                                # Decimos que el contenedor tiene bloque asignado
                                bloqueAsignado = 1
                                tiemposAsignados.append(c[1])
                                # Anyadimos el tiempo ri a una lista que nos
                                # mostrara los contenedores que si hemos asignado
                                bisect.insort(proximasSalidas,([c[2],c[0],bloqueAleatorio]))
                                #Meto el contenedor en la lista de proximas salidas,
                                #sabre por donde llega, tiempo de salida y bloque en el que esta asigandox
                                tProximo = tActual + c[3+int(bloqueAleatorio)]

                        # Tenemos que ver si va a haber salidas entre el instante tActual y
                        # el instante en el que llegue el contenedor al bloque, tProximo
                        salida(tActual, tProximo, tiemposSalientes1, tiemposSalientes, numBloques,
                            listaBloques, salientesTipos)
                        salidaNuevo(tActual, tProximo, proximasSalidas, numBloques,
                            listaBloques, listaSacadosNuevos, listaSacadosNTierra, listaSacadosNMar)
                        # Actualizamos los tiempos
                        tAnterior = tActual
                        tActual = tProximo
                        # Anyadimos el componente a la solucion
                        solucion.append([c,bloqueAleatorio,tActual])

                tiempoFinalInstancia = time()
                tiempoTotalInstancia = tiempoFinalInstancia -                tiempoInicialInstancia
                tiemposInstancias.append([str(i),solucion[-1][-1],tiempoTotalInstancia])

df = pd.DataFrame(tiemposInstancias, columns = ["Instancia", "Tiempo u.t.", "Tiempo"])
df.to_excel ("Tiempos_"+str(it+1)+"Iteraciones_"+str(i)+".xlsx", index = False, header=True)
tFinalProceso = time()
tTotalProceso = tFinalProceso - tInicialProceso
tMejor = []

for tm in range(len(tiemposInstancias)):
        tMejor.append(tiemposInstancias[tm][1])

print("\nEl tiempo total del proceso de asignacion: " + str(tTotalProceso) + " ms.")
print("El mejor tiempo de solucion ha sido: " + str(min(tMejor)))
```

Listing A.3: Algoritmo de proximidad

```

import sys
import csv
import random
from time import time
import pandas as pd
import numpy as np
import itertools
import bisect
import operator

listado = sys.argv[1]

# Indica si esta en el lado de tierra o no
def esLadoTierra(posicion,bloque):
    if salientesTipos[bloque][2] < posicion :
        listaBloques[bloque][1] = listaBloques[bloque][1] - 1
        return True
    else: return False

# Indica si esta en el lado de mar o no
def esLadoMar(posicion,bloque):
    if salientesTipos[bloque][2] >= posicion :
        listaBloques[bloque][0] = listaBloques[bloque][0] - 1
        return True
    else: return False

```

```
28  # Saca un contenedor del bloque en un instante
29  def salida(tAnt, tAct, listaSalientes1D, listaSalientes2D, nB, lB, sTipos):
30      cont = 0
31      sacar = []
32      # Metemos en array sacar aquellos tiempos que hay que sacar del bloque
33      for listaSalientes1DIndex in range(len(listaSalientes1D)):
34          if (listaSalientes1D[listaSalientes1DIndex] <= tAct)
35              and (listaSalientes1D[listaSalientes1DIndex] >= tAnt):
36                  sacar.append(listaSalientes1D[listaSalientes1DIndex])
37          # Para cada tiempo de sacar, hay que buscar a que bloque corresponde
38          #y si es a lado de mar o a lado de tierra.
39          for s in range(len(sacar)):
40          # sacar[s] es el valor que queremos sacar
41          # hay que buscarlo en tiemposSalientes, para encontrar el bloque.
42          indiceSacar = 0
43          for nbl in range(numBloques):
44              # Si este es el bloque
45              if ((sacar[s] in tiemposSalientes[nbl]) and
46              esLadoTierra(tiemposSalientes[nbl].index(int(sacar[s])),nbl)):
47                  if((sacar[s] not in listaSacados)):
48                      for contar in range(contar_veces(sacar[s],tiemposSalientes1)):
49                          listaSacados.append(sacar[s])
50
51              if ((sacar[s] in tiemposSalientes[nbl]) and
52              esLadoMar(tiemposSalientes[nbl].index(int(sacar[s])),nbl)):
53                  if((sacar[s] not in listaSacados)):
54                      for contar in range(contar_veces(sacar[s],tiemposSalientes1)):
55                          listaSacados.append(sacar[s])
56
57  # Saca un contenedor de los que ha entrado nuevos en el bloque en un instante
58  def salidaNuevo(tAntN, tActN, listaSalientes1DN, nBN, lBN, sNuevo, sNTierra, sNMar):
59      contN = 0
60      sacarN = []
61      # Metemos en array sacar aquellos tiempos que hay que sacar del bloque
62      for listaSalientes1DIndexN in range(len(listaSalientes1DN)):
63          if ((listaSalientes1DN[listaSalientes1DIndexN][0] <= tActN) and
64          (listaSalientes1DN[listaSalientes1DIndexN][0] >= tAntN)):
65              sacarN.append(listaSalientes1DN[listaSalientes1DIndexN])
66      # Para cada tiempo de sacar hay que buscar a que bloque corresponde y
67      #si es al lado de mar o de tierra.
68      # Para cada elemento de sacarN
69      for sN in range(len(sacarN)):
70          bloqueN = 0
71          tipo = 0
72          for lN in sacarN:
73              # Sacamos su bloque y el tipo de contenedor que es
74              bloqueN =  sacarN[sacarN.index(lN)][2]
75              tipo = sacarN[sacarN.index(lN)][1]
76              if sacarN[sacarN.index(lN)] not in sNuevo :
77                  if sacarN[sacarN.index(lN)][1] == 1: #Si es 1 hay que sacarlo del lado de mar
78                      # El espacio disponible aumenta
79                      lBN[2][0] = lBN[2][0] + 1
80                      # Quitamos el contenedor de proximasSalidas
81                      listaSalientes1DN = listaSalientes1DN[1:]
82                      sNuevo.append(sacarN[sacarN.index(lN)])
83                      sNMar.append(sacarN[sacarN.index(lN)])
84                  elif sacarN[sacarN.index(lN)][1] == 2: #Si es 2 hay que sacarlo del lado de tierra
85                      # El espacio disponible aumenta
86                      lBN[2][1] = lBN[2][1] + 1
87                      # Quitamos el contenedor de proximasSalidas
88                      listaSalientes1DN = listaSalientes1DN[1:]
89                      sNuevo.append(sacarN[sacarN.index(lN)])
90                      sNTierra.append(sacarN[sacarN.index(lN)])
91
92  # Devuelve una lista con los repetidos
93  def encontrarTiemposRep(lista):
94      repetido = []
95      unico = []
96      for x in lista:
97          if x not in unico: unico.append(x)
98          else: repetido.append(x)
99      return repetido
100
101 # Devuelve el numero de veces que se repite un tiempo de salida
102 def contar_veces(elemento, lista):
103     veces = 0
104     for i in lista:
105         if elemento == i:
106             veces += 1
107     return veces
108
109 def noHaySitiosTierra():
110     hay = True
111     for bloq in range(numBloques):
112         if(listaBloques[bloq][1] > 0): hay = False
113
114 def noHaySitiosMar():
115     hay = True
116     for bloq in range(numBloques):
117         if(listaBloques[bloq][0] > 0): hay = False
118
119 def buscarAleatorioMar():
120     bl = random.randint(0,numBloques-1)
121     while listaBloques[bl][0] == 0 : bl = random.randint(0,numBloques-1)
122     listaBloques[bl][0] = listaBloques[bl][0] -1
123     return bl
124
125 def buscarAleatorioTierra():
126     bl = random.randint(0,numBloques-1)
127     while listaBloques[bl][1] == 0 : bl = random.randint(0,numBloques-1)
128     listaBloques[bl][1] = listaBloques[bl][1] -1
```

```
129         return bl
130
131     for i in range(1):
132         tInicialProceso = time()
133         # abro archivo con nombre de listas
134         with open(listado, "r") as f:
135             listaInstancias = f.readlines()
136             listaInstancias = [x.strip() for x in listaInstancias]
137             tiemposInstancias = []
138             # para cada uno de los archivos
139             for i in listaInstancias:
140                 tiempoInicialInstancia = time()
141                 # abro archivo
142                 with open(i) as datos:
143                     # inicializacion de variables necesarias:
144                     numBloques = 0
145                     numCont = 0
146                     numRows = 0
147                     numCol = 0
148                     bloquesMar = []
149                     bloquesTierra = []
150                     saleSea = 0
151                     saleLand = 0
152                     saleMarPrimeros = 0
153                     salientesTipos = []
154                     tiemposSalientes = []
155                     tSalientesRes = []
156                     TsalenSea = []
157                     TsalenLand = []
158                     listaContenedores = []
159                     listaBloques = []
160                     listaSacados = []
161                     listaSacadosLand = []
162                     listaSacadosSea = []
163                     contenedoresAsignados = []
164                     solucion = []
165                     tiemposRepetidos = []
166                     proximasSalidas = []
167                     proximasSalidasDatos = [[]]
168                     listaSacadosNuevos = []
169                     listaSacadosNMar = []
170                     listaSacadosNTierra = []
171                     tiemposAsignados = []
172                     bloquesAsignados = []
173                     res = []
174
175                     # para cada linea del archivo
176                     for linea in datos:
177
178                         # Numero de bloques
179                         if linea.startswith("number␣of␣blocks"):
180                             instanciaNumBloques = next(datos).strip()
181                             instanciaNumBloques = instanciaNumBloques.replace("\t",",")
182                             numBloques = int(instanciaNumBloques)
183
184                         # Numero de contenedores
185                         if linea.startswith("number␣of␣containers"):
186                             instanciaNumCont = next(datos).strip()
187                             instanciaNumCont = instanciaNumCont.replace("\t",",")
188                             numCont = instanciaNumCont
189
190                         # Numero de filas
191                         if linea.startswith("number␣of␣rows"):
192                             instanciaNumRows = next(datos).strip()
193                             instanciaNumRows = instanciaNumRows.replace("\t",",")
194                             numRows = instanciaNumRows
195
196                         # Numero de columnas
197                         if linea.startswith("number␣of␣columns"):
198                             instanciaNumCol = next(datos).strip()
199                             instanciaNumCol = instanciaNumCol.replace("\t",",")
200                             numCol = instanciaNumCol
201
202                         # Datos sobre los bloques
203                         for b in range(int(numBloques+1)):
204                             if linea.startswith("#Block␣␣␣␣␣" + str(b)):
205                                 instanciaFreeSea = next(datos).strip()
206                                 instanciaFreeSea = instanciaFreeSea.replace("\t",",")
207                                 instanciaFreeLand = next(datos).strip()
208                                 instanciaFreeLand = instanciaFreeLand.replace("\t",",")
209                                 instanciasSaleMar = next(datos).strip()
210                                 instanciasSaleMar = instanciasSaleMar.replace("\t",",")
211                                 instanciasSaleLand = next(datos).strip()
212                                 instanciasSaleLand = instanciasSaleLand.replace("\t",",")
213                                 instanciasSaleMarPrimeros = next(datos).strip()
214                                 instanciasSaleMarPrimeros = instanciasSaleMarPrimeros.replace("\t",",")
215                                 instanciaFreeSea = list(map(str,instanciaFreeSea.split(",")))
216                                 instanciaFreeLand = list(map(str,instanciaFreeLand.split(",")))
217                                 instanciasSaleMar = list(map(str,instanciasSaleMar.split(",")))
218                                 instanciasSaleLand = list(map(str,instanciasSaleLand.split(",")))
219                                 instanciasSaleMarPrimeros = list(map(str,instanciasSaleMarPrimeros.split(",")))
220                                 instanciaBloque = [instanciaFreeSea[1],instanciaFreeLand[1]]
221                                 instanciaBloquesMar = instanciaFreeSea
222                                 instanciaBloquesTierra = instanciaFreeLand
223                                 instanciaBloque = [int(x) for x in instanciaBloque]
224                                 bloquesMar.append(instanciaBloquesMar[1])
225                                 bloquesTierra.append(instanciaBloquesTierra[1])
226                                 listaBloques.append(instanciaBloque)
227                                 bloquesMar = [int(x) for x in bloquesMar]
228                                 bloquesTierra = [int(x) for x in bloquesTierra]
229                                 saleSea = int(instanciasSaleMar[1])
```

```
230                         saleLand = int(instanciasSaleLand[1])
231                         saleMarPrimeros = int(instanciasSaleMarPrimeros[2])
232                         salientesTipos.append([int(saleSea),int(saleLand),int(saleMarPrimeros)])
233                         instanciatiempos = []
234                         instanciaTSaliente = next(datos).strip()
235                         for j in range((int(saleSea)+int(saleLand))):
236                             instanciaTSaliente = next(datos).strip()
237                             instanciaTSaliente = instanciaTSaliente.replace("\t",",")
238                             instanciatiempos.append(instanciaTSaliente)
239                             tSalientesRes.append(instanciaTSaliente)
240                         instanciatiempos = [int(x) for x in instanciatiempos]
241                         TsalenSea.append(instanciatiempos[:int(saleMarPrimeros)])
242                         TsalenLand.append(instanciatiempos[int(saleMarPrimeros):])
243                         tiemposSalientes.append(instanciatiempos)
244
245                     # Datos sobre los contenedores
246                     if linea.startswith("#Type"):
247                         for n in range(int(numCont)):
248                             instanciaCont = next(datos).strip()
249                             instanciaCont = instanciaCont.replace("\t",",")
250                             instanciaCont = list(map(int,instanciaCont.split(",")))
251                             listaContenedores.append(instanciaCont)
252                         pass
253                     pass
254
255                 listaCDesordenada = listaContenedores
256                 listaContenedores.sort(key = operator.itemgetter(1, 2))
257                 tiemposSalientes1 = sorted(list(itertools.chain.from_iterable(tiemposSalientes)))
258                 tiemposRepetidos = encontrarTiemposRep(tiemposSalientes1)
259                 tActual = 0
260                 tAnterior = 0
261                 tProximo = 0
262                 contenedoresAs = 0
263                 noAs = 0
264
265                 for c in listaContenedores:
266                     # Esperamos hasta que haya contenedor disponible
267                     while (tActual<c[1]):
268                         salida(tAnterior, tActual, tiemposSalientes1, tiemposSalientes, numBloques,
269                             listaBloques, salientesTipos)
270                         tAnterior = tActual
271                         tActual = tActual + 1
272
273                     # Definimos bloqueAsignado para indicar si el bloque ya ha sido asignado.
274                     # Estara a 0 cuando aun no haya sido asignado.
275                     bloqueAsignado = 0
276                     bloque = 0
277
278                     # A partir de la 4 posicion tenemos las distancias
279                     time1 = time()
280                     dist = c[4:]
281                     dist = [int(x) for x in dist]
282                     # Las ordenamos de menor a mayor instante de tiempo
283                     dist.sort()
284
285                     # Mientras que no este asignado seguiremos iterando
286                     while (bloqueAsignado == 0) and (bloque < numBloques):
287
288                         # Sacamos el bloque que mas cerca esta
289                         bloqueMasCerca = c[4:].index(dist[bloque])
290
291                         if c[0] == 1: # Si viene por mar va a lado de mar
292                             if listaBloques[bloqueMasCerca][0] > 0 :
293                                 # Asignamos
294                                 listaBloques[bloqueMasCerca][0] = listaBloques[bloqueMasCerca][0] -1
295                                 bloqueAsignado = 1
296                                 contenedoresAs = contenedoresAs + 1
297                                 # Tiempos que han sido asignados
298                                 tiemposAsignados.append(c[1])
299                                 # Meto el contenedor en la lista de proximas salidas (Lo ultimo dice que bloque es)
300                                 bisect.insort(proximasSalidas,([c[2],c[0],str(c[4:].index(int(dist[bloque]))+1)]))
301                                 # Sabre por donde llega, tiempo de salida y bloque en el que esta asigandox
302                                 tProximo = tActual + c[4+int(bloqueMasCerca)]
303
304                             else:
305                                 # Busco un bloque aleatorio en el lado de mar
306                                 blq = buscarAleatorioMar()
307                                 bloqueAsignado = 1
308                                 bloqueMasCerca = blq
309                                 contenedoresAs = contenedoresAs + 1
310                                 tiemposAsignados.append(c[1])
311                                 bisect.insort(proximasSalidas,([tActual+c[2],c[0],str(c[4:].index(int(dist[blq]))+1)]))
312                                 tProximo = tActual + c[4+int(blq)]
313
314
315                         elif c[0] == 2: # Si viene por tierra va a lado de tierra
316                             if listaBloques[bloqueMasCerca][1] > 0 :
317                                 # Asignamos
318                                 listaBloques[bloqueMasCerca][1] = listaBloques[bloqueMasCerca][1] -1
319                                 bloqueAsignado = 1
320                                 contenedoresAs = contenedoresAs + 1
321                                 # Tiempos que han sido asignados
322                                 tiemposAsignados.append(c[1])
323                                 # Meto el contenedor en la lista de proximas salidas (Lo ultimo dice que bloque es)
324                                 bisect.insort(proximasSalidas,([tActual+c[2],c[0],str(c[4:].index(int(dist[bloque]))+1)]))
325                                 # Sabre por donde llega, tiempo de salida y bloque en el que esta asigandox
326                                 tProximo = tActual + c[4+int(bloqueMasCerca)]
327
328                             else:
329                                 # Busco un bloque aleatorio en el lado de tierra
330                                 blq = buscarAleatorioTierra()
```

```
331                                 bloqueAsignado = 1
332                                 bloqueMasCerca = blq
333                                 contenedoresAs = contenedoresAs + 1
334                                 tiemposAsignados.append(c[1])
335                                 bisect.insort(proximasSalidas,([c[2],c[0],str(c[4:].index(int(dist[blq]))+1)))
336                                 tProximo = tActual + c[4+int(blq)]
337 
338                         salida(tActual, tProximo, tiemposSalientes1, tiemposSalientes,
339                             numBloques, listaBloques, salientesTipos)
340                         salidaNuevo(tActual, tProximo, proximasSalidas, numBloques,
341                             listaBloques, listaSacadosNuevos, listaSacadosNTierra, listaSacadosNMar)
342 
343                         # Anyadimos el componente a la solucion
344                         solucion.append([c,c[0],c[4:],bloqueMasCerca+1,c[4],c[5],c[6],c[7],c[8],tActual,tProximo])
345                         res.append([c,bloqueMasCerca,tActual])
346                         # Actualizamos los tiempos
347                         tAnterior = tActual
348                         tActual = tProximo
349 
350                         if(bloque < numBloques):
351                             bloquesAsignados.append(c[4:].index(int(dist[bloque]))+1)
352                         else:
353                             noAs = contenedoresAs
354 
355                     tiempoFinalInstancia = time()
356                     tiempoTotalInstancia = tiempoFinalInstancia - tiempoInicialInstancia
357                     tiemposInstancias.append([str(i),solucion[-1][-1],tiempoTotalInstancia])
358 
359                     print("\nTiempo solucion para la instancia " + str(i) + ": " + str(solucion[-1][-1]))
360                     print("El tiempo total del proceso de asignacion: " + str(tiempoTotalInstancia) + " ms.")
361 
362 # Export los datos a una hoja excel
363 df = pd.DataFrame(tiemposInstancias, columns = ["Instancia", "Tiempo en u.t.", "Tiempo"])
364 df.to_excel ("TiemposMENORDISTANCIAS.xlsx", index = False, header=True)
```