Universidad de Castilla-La Mancha
Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado
Grado en Ingeniería Informática
Computación

Algoritmos heurísticos y metaheurísticos para problemas de carga de camiones

Rubén Castillo Carrasco

Junio, 2023

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Computación

Algoritmos heurísticos y metaheurísticos para problemas de carga de camiones

Autor: Rubén Castillo Carrasco

Tutor: Francisco Parreño Torres

Junio, 2023

Dedicado a mi familia y a mis amigos.

Declaración de autoría

Yo, Rubén Castillo Carrasco, con DNI 49427163W, declaro que soy el único autor del trabajo fin de grado titulado “Algoritmos heurísticos y metaheurísticos para problemas de carga de camiones ”, que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual, y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 12 de Junio de 2023

Fdo.: Rubén Castillo Carrasco

Resumen

La carga eficiente de camiones es un desafío de la logística y transporte de mercancías que en la actualidad tiene un impacto significativo tanto económico como ambiental.

El problema de carga de camiones, también conocido como CLP (siglas en inglés de Container Loading Problem), es un problema de optimización que busca maximizar el volumen total cargado dentro del contenedor de un camión mediante una disposición eficiente de un conjunto de instancias de entrada, todo ello manteniendo una serie de condiciones que garanticen la seguridad de la carga.

El objetivo de este proyecto será explorar los distintos métodos heurísticos y metaheurísticos, enfoques no exactos que permiten abordar problemas complejos como el CLP, además de aplicarlos al problema en cuestión.

Agradecimientos

A mi familia, por apoyarme en todo momento durante estos años, siendo un pilar fundamental para desarrollarme tanto como estudiante como persona.

A mis amigos, tanto a los que he podido conocer en el grado como a los de toda la vida, que siempre han estado de mi lado.

A mi tutor, por dedicar gran parte de su tiempo, guiándome y ayudándome en todo lo necesario para poder llevar a cabo este trabajo de fin de grado de la mejor forma posible.

A todos los profesores que me han impartido clases a lo largo de la carrera, por brindarme una formación de calidad que me permitirá hacer frente a cualquier obstáculo que se presente en mi camino profesional.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Índice de listados de código

1. Introducción

En la actualidad, la logística y el transporte de mercancías suponen un pilar fundamental para el desarrollo adecuado de la economía. En nuestro país, como indica un informe realizado por el ministerio de industria [Minsterio de industria, 2023], esta actividad representa el 10% del PIB, alcanzando una cifra de negocio anual de 111.000 millones de euros y generando cerca de un millón de puestos de trabajo , además de afectar de forma clave a otros sectores económicos, siendo esencial para el desarrollo de una economía globalizada.

Podemos determinar que, en nuestro país, como indica un informe realizado por el observatorio de transporte y logística de España [OTLE, 2023], la mayor proporción de transporte de mercancías se realiza de forma terrestre. Según datos del ministerio de transportes, movilidad y agenda urbana, en el año 2020 un 75% de las toneladas transportadas en nuestro país se llevó a cabo haciendo uso de la carretera como modo de transporte, seguida del transporte marítimo con un 23% de las toneladas transportadas.

Es importante destacar que el coste de transporte de bienes ha aumentado significativamente en los últimos meses. Según el INE este coste se incrementó en un total de un 26.2% en el último año, debido a diferentes factores como el aumento en el precio del gasóleo.

De acuerdo con [MITMA, 2022], en el año 2022, los costes medios tomando como vehículo de referencia el vehículo articulado de carga general siguen la distribución de la figura 1.1. Cabe destacar el gran impacto que tienen los costes variables, dependientes del uso del vehículo, como el costo del combustible que toma de media más del 30% del gasto o el del personal de conducción, con más de un 20%, frente a gastos fijos como el de la amortización del vehículo, que ocupa poco más del 10%.

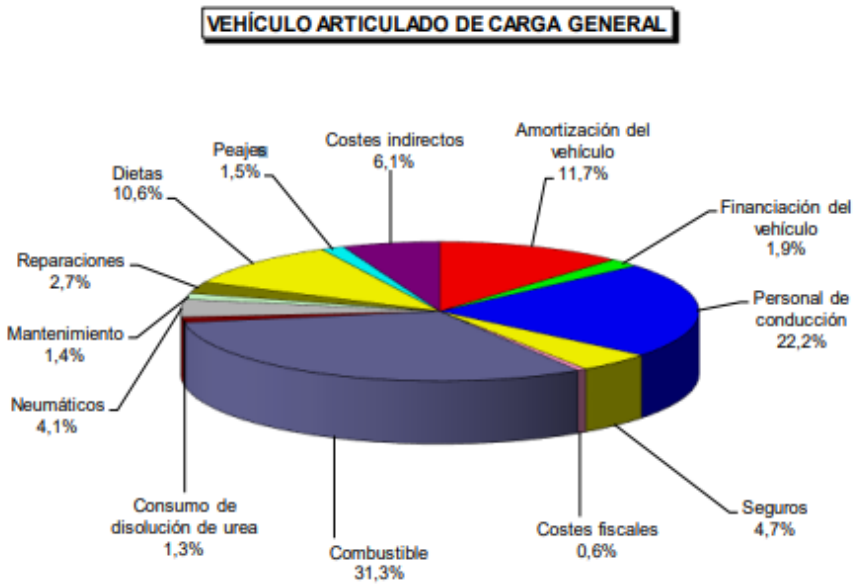


Figura 1.1: Diagrama de los costes, obtenido en [MITMA, 2022]

Como ya se ha indicado, uno de los mayores costes del transporte es el combustible, en el caso de los camiones generalmente se trata del gasóleo. Este, tomando como referencia datos de [CETM, 2023], ha incrementado su valor en los últimos 5 años un 63%, por lo que una colocación adecuada de los productos en el camión será un factor indispensable para reducir los costes del transporte.

Todo ello, sumado a la falta de transportistas en la que se encuentra el sector, donde se estima que existen unos 20.000 puestos vacantes en nuestro país [laSexta, 2022], y que en el futuro esta situación se verá agravada con la jubilación de los transportistas (según la DGT el 72% de estos tienen una edad superior a los 50 años) hace necesaria la obtención de soluciones inmediatas para que no se vea dañada la cadena de suministro, que tendría consecuencias fatales en la economía.

A esto hay que sumarle el impacto ambiental negativo producido por los vehículos en el transporte de bienes. En la actualidad, en España los camiones de combustión solo suponen el 2% de la flota de vehículos en circulación, pero producen hasta un 23% de las emisiones de CO_2 totales a la atmósfera[NexoTrans, 2022].

Esto, por lo tanto, sitúa a esta actividad económica como una de las mayores responsables de las emisiones de este gas a la atmósfera, ya que, según fuentes oficiales[MITMA, 2020], en el año 2019 hasta un 71.7% de las emisiones de CO_2 en nuestro país tuvieron como origen el transporte por carretera.

La emisión en masa de gases de efecto invernadero a la atmósfera, como lo es el CO2, produce el denominado cambio climático, responsable directo de alteraciones en el clima, entre las que se encuentran desde el aumento de las temperaturas hasta el aumento del riesgo de desertificación, mayores precipitaciones anuales o aumento en el riesgo de inundaciones, como indica el parlamento europeo [Europeo, 2018].

Asimismo, según indica el informe del IPCC del año 2023 [IPCC, 2023], la quema de combustibles fósiles durante más de un siglo ha causado un incremento de la temperatura global de 1.1ºC frente a niveles previos a la revolución industrial. Una gráfica que representa este ascenso se puede observar en la figura 1.2.

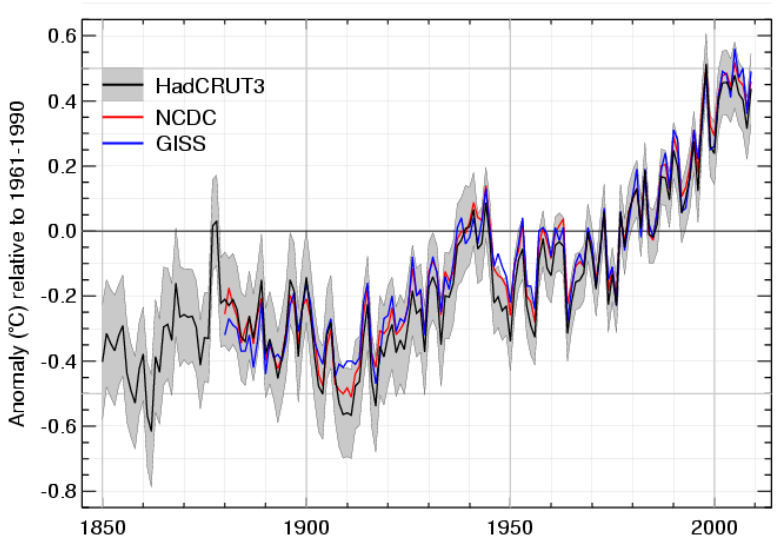


Figura 1.2: Diagrama del aumento de la temperatura desde 1850, obtenida en [SINC, 2009]

Es por todo ello que, una optimización del transporte de mercancías afectará positivamente tanto en el correcto desempeño de una actividad como la logística, motor de la economía, como en la reducción de la contaminación de nuestro planeta.

1.1. Objetivos

El objetivo principal consistirá en optimizar la carga de los camiones de transporte de mercancías, los cuales transportarán en su interior pallets. Nuestro estudio se enfocará en la búsqueda del mejor subconjunto de pallets y su disposición, de tal manera que se maximice el volumen total de carga transportada en el camión, todo ello teniendo en cuenta una serie de restricciones relacionadas con el peso, dimensiones y compatibilidad de los pallets, que serán especificadas a posteriori, necesarias para un transporte óptimo y seguro de la carga.

1.2. Estructura de la memoria

Este documento se organiza en un total de 6 capítulos, dentro de los cuales se encuentran los siguientes:

- Capítulo 1: introducción al problema a tratar, objetivos del trabajo y estructura del documento.
- Capítulo 2: estado del arte y definición formal del problema. En esta sección se abordarán tanto el marco teórico necesario para entender el problema como algoritmos que permiten resolver problemas similares a la optimización de la carga en camiones.
- Capítulo 3: modelización del problema y definición de los distintos algoritmos heurísticos planteados.
- Capítulo 4: implementación de algoritmos heurísticos y metaheurísticos definidos en el segundo capítulo, adaptándolos al problema a tratar.
- Capítulo 5: comparativa de los resultados obtenidos por los distintos algoritmos implementados y sus parámetros, analizando su eficiencia, precisión y aplicabilidad en casos reales.
- Capítulo 6: conclusiones, trabajo futuro y competencias adquiridas, identificando posibles mejoras y desarrollos adicionales para optimizar aún más el proceso de carga de camiones, además de la bibliografía, incluyendo las fuentes consultadas y referencias utilizadas.

2. Estado del arte

Este apartado queda dividido en dos partes. En la primera de ellas se estudiará el concepto de optimización, los algoritmos más usados para resolver problemas de este tipo y su clasificación. La segunda parte se centrará en el estado del arte del problema anterior, conocido como el problema de carga de camiones o CLP (por sus siglas en inglés, provenientes de container loading problem). Este se trata un problema de optimización combinatoria que busca maximizar el volumen de la selección de un conjunto de objetos cargados y su posicionamiento dentro de un contenedor, de tal forma que estos no se solapen entre sí ni salgan de las dimensiones del camión. Cabe destacar que el CLP es un problema de gran dificultad, al ser $NP-duro$, cuya definición y demostración se verán en este mismo capítulo.

2.1. Introducción a la optimización y algoritmos que la abordan

Podemos definir un problema computacional como una tarea a resolver mediante un algoritmo tal que a partir de una entrada se produzca una salida deseada que cumpla una serie de propiedades. Dentro de estos, podemos distinguir 3 tipos principalmente [UIUC, 2020]: los problemas de búsqueda, los de decisión y los de optimización . En este caso se hará énfasis en los de optimización, que se corresponden con el tipo del problema a resolver.

2.1.1. Problemas de optimización

Se puede definir un problema de optimización como aquel en el que se trata de encontrar el mejor candidato entre un conjunto de instancias, a su vez cumpliendo un conjunto de requisitos. Formalmente podemos definir un problema de optimización como una cuádrupla (I, f, v, c) donde:

- I es un conjunto de instancias a evaluar por el algoritmo.
- v es la función de valor o función objetivo, una aplicación de la forma $v\colon I \to R$, que devuelve el valor a optimizar de una instancia determinada para el problema a tratar.

- f es la función de factibilidad, una aplicación de la forma $f\colon I \to \{0,1\}$, que nos devuelve un valor binario que indica si la instancia en cuestión es factible, es decir, si puede conformar una solución al problema o no.
- c es el criterio de optimización, que se usa junto con la función de valor e indica qué es lo que se quiere optimizar de esta. Su valor puede ser maximizar, minimizar, etc.

Para ejemplificar lo anterior se toma como referencia el problema de la mochila binaria, o en inglés 0/1 knapsack problem (0/1 KP), que busca llenar una mochila con objetos, optimizando el valor total cargado de estos y usando como restricción un límite de peso. Para ello tenemos que I es el total de conjuntos de instancias evaluables, es decir, el conjunto de todos los conjuntos de objetos a cargar en la mochila. Si tomamos una representación binaria donde 0 indica no tomado y 1 indica tomado, dada una entrada de 2 objetos se tiene que

$$I = \{(0,0),(0,1),(1,0),(1,1)\}$$

donde la instancia $(1,0)$ indicaría que el primer objeto es tomado y el segundo no tomado.

En cuanto a la función de factibilidad f , dado un conjunto que $x^* \in I$ y un valor M, usando t_i como variable binaria que indica que el objeto i es tomado o no tomado, w_i indica el peso del ítem i, y n es el tamaño de la entrada, se tiene que:

$$f(x^*) = \begin{cases} 1 & \text{si } \sum_{i=0}^{n} w_i \cdot t_i \leq M, \\ 0 & \text{en caso contrario.} \end{cases}$$

Respecto a la función de valor o función objetivo, como se ha indicado es la suma del valor de los objetos cargados, por lo tanto, se puede expresar tal que:

$$v(x^*) = \sum_{i=0}^{n} valor_i \cdot t_i$$

Por último, como criterio de optimización, en este caso es maximizar. Esto implica que lo que se busca es aquella instancia tal que:

$$\hat{x} = \arg\max_{x^* \in I} \sum_{i=0}^{n} v(i) \cdot t_i \text{ sujeto a } f(x^*) = 1$$

Para resolver este tipo de problemas, podemos agrupar los algoritmos que los resuelven en dos grupos, atendiendo a como es la solución obtenida para cualquier instancia de problema de entrada para cada uno de estos. Según este criterio, distinguimos por un lado entre algoritmos exactos, que siempre devuelven la mejor solución, denominada solución óptima y no exactos, que no siempre encuentran esta última. Además, dentro de los dos grupos anteriores se pueden distinguir otros subgrupos, como se observa en la figura 2.1.

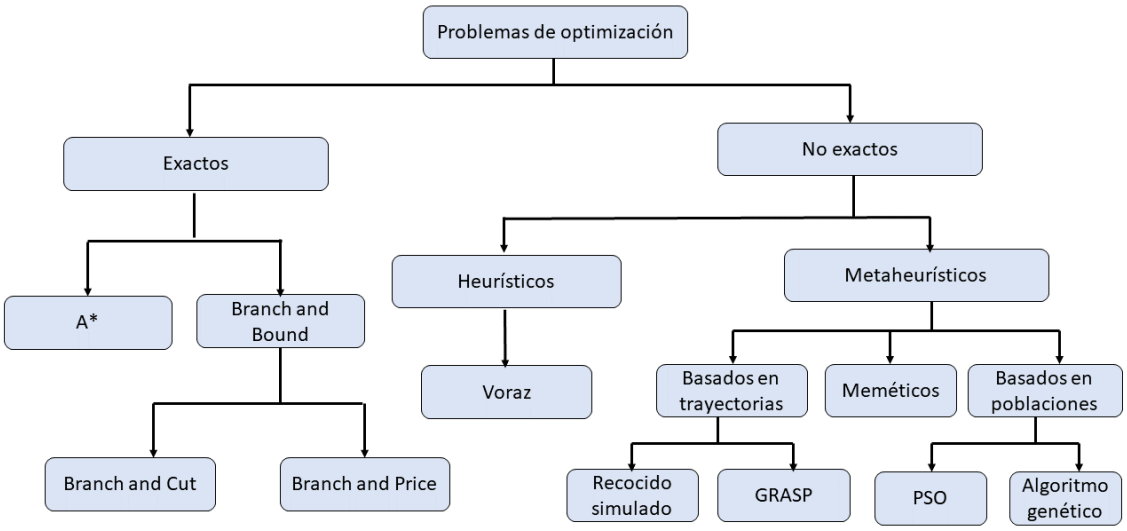


Figura 2.1: Clasificación de algoritmos para resolver problemas de optimización

Antes de profundizar en estos grupos es necesario conocer la propia clasificación de los problemas, en este caso atendiendo a la dificultad que presentan. Esto es imprescindible para poder determinar cuáles serían los algoritmos oportunos para resolver el problema en cuestión.

2.1.2. Clasificación de los problemas atendiendo a su dificultad

De acuerdo con la "dificultad" que puedan presentar los problemas a nivel computacional, que tiene como consecuencia directa la complejidad computacional para resolverlos, es habitual distinguir entre problemas P, NP, $NP-completos$ y $NP-duros$.

- Problema P: categoría de problemas resolubles mediante algoritmos de complejidad polinomial mediante una máquina de Turing determinista. Un problema de completitud P podría ser la ordenación de una lista, resoluble por algoritmos como merge sort, de complejidad temporal $O(n \cdot \log n)$ [Mellon, 2020]
- Problema NP: categoría de problemas resolubles mediante algoritmos de complejidad polinomial usando una máquina de Turing no determinista, donde sus soluciones pueden verificarse en tiempo polinomial mediante una máquina determinista. Un subconjunto de este grupo de problemas es el de los problemas $NP-completos$ [Naseera, 2020].

- Problema $NP - duro$: conjunto de problemas tales que todos los problemas en NP pueden reducirse a estos en tiempo polinomial. Estos no tienen por qué pertenecer a NP, es decir, no necesariamente han de ser resolubles en tiempo polinomial mediante una máquina de Turing no determinista [Mann, 2017].
- Problema $NP - completo$: aquellos problemas que pertenecen tanto a las categorías de NP como a a la de $NP - duro$. En caso de encontrar un algoritmo capaz de resolver un problema $NP - completo$ en tiempo polinomial para el peor de sus casos, se demostraría que $P = NP$, dado que todos los $NP - completos$ podrían ser transformados a este $NP - completo$ en tiempo polinomial, implicando a su vez que cualquier problema $NP - completo$ sería resoluble en tiempo polinomial, algo que todavía no ha sido demostrado.

  Un problema $NP - completo$ es SAT o problema de la satisfactibilidad booleana, problema de búsqueda que trata de hallar los valores de un conjunto de variables que hacen cierta una expresión. Todo aquel problema transformable a SAT en tiempo polinomial será, por lo tanto, $NP - completo$, dado que todos los problemas de NP se podrían transformar en este en tiempo polinomial [Darmann and Döcker, 2019].

Con lo anterior se pasa a determinar el conjunto de algoritmos que se usan en la actualidad para resolver problemas de optimización y sus respectivas ventajas e inconvenientes teniendo en cuenta las características de la solución obtenida, el tipo de problema de optimización a resolver y su dificultad.

2.1.3. Algoritmos exactos

Los algoritmos exactos podrían definirse como aquel grupo de algoritmos que llevan a cabo una búsqueda dentro del espacio de soluciones de forma que siempre encuentren la instancia solución que optimice la función de valor de acuerdo con el criterio de optimización.

El hecho de alcanzar siempre el óptimo suele traer consigo un problema, la complejidad temporal necesaria para alcanzarlo. Por ejemplo, tomando como referencia problemas $NP - completos$ como el problema de la mochila, si se cumple que $P \neq NP$, el orden de complejidad de un algoritmo exacto ascendería a uno exponencial, ya que este siempre devuelve la solución óptima para cualquier entrada, y si se diera el caso de que este fuera capaz de resolverlo en tiempo polinómico quedaría demostrado que $P = NP$.

A su vez, los algoritmos de este grupo pueden clasificarse atendiendo a como exploran en espacio de soluciones.

Por un lado, se encuentran los algoritmos que realizan una búsqueda exhaustiva del espacio de soluciones, como los algoritmos de búsqueda en profundidad o en anchura, resultando en el mayor de los costes temporales, pero siendo a su vez de mayor facilidad a la hora de implementar.

Por otro lado, existen métodos de resolución como backtracking o branch and bound que realizan una poda intermedia para evitar expandir soluciones que no alcanzan la óptima, reduciendo así el tiempo de obtención de la solución. En este caso nos centraremos en branch and bound (B&B) y sus variantes, que serían las más adecuadas para resolver el problema a tratar.

Podemos definir Branch and bound como la estrategia algorítmica que permite abordar problemas de optimización, que intenta explorar de la forma más breve posible el espacio de soluciones de tal forma que siempre alcance la solución óptima. Para ello, el procedimiento seguido es el siguiente [de la Ossa, 2022]:

- Selección: se toma el estado no explorado almacenado en una estructura de datos, que puede seguir criterios como el de primero mejor.
- Ramificación: a partir del estado seleccionado, este se expande generando otros estados a visitar y se almacenan en la estructura de datos determinada.
- Poda: se eliminan aquellos estados que no pueden alcanzar el óptimo. Para ello se hace uso de cotas, dentro de las cuales se pueden encontrar las de factibilidad, la cota optimista o la pesimista. La poda por factibilidad es aquella que impide expandir aquellos estados que nunca alcanzan una solución factible o posible. Otro criterio de poda es la optimista, que dado un estado este no se expande si su cota optimista, que representa el valor máximo que espera obtenerse al expandirlo, es inferior a una cota pesimista, que garantiza el peor resultado esperado del espacio de soluciones.

2.1.4. Algoritmos no exactos

Por otro lado, los problemas de optimización también pueden resolverse aplicando técnicas no exactas, es decir, que no recorren el espacio de soluciones de forma que se garantice encontrar siempre la solución óptima. A cambio, estos evitan complejidades temporales elevadas que para problemas de gran dificultad como los $NP - completos$ o $NP - duros$, donde la complejidad de un exacto ascendería a una exponencial, podrían dar lugar a no encontrar la solución en tiempo aceptable.

Dentro de este grupo encontramos dos grupos mayoritarios, las técnicas heurísticas y las metaheurísticas.

Técnicas heurísticas

Las técnicas heurísticas pueden definirse como aquellos enfoques específicos a un problema que hacen uso de reglas bien definidas obtenidas en función de este, utilizándolas para explorar sistemáticamente el espacio de búsqueda del problema para llegar a una solución aceptable en un tiempo mínimo de consumo[Hilbig, 2012].

Uno de los algoritmos más destacados que hace uso de técnicas heurísticas es el algoritmo voraz.

Algoritmos voraces

Son algoritmos cuyo funcionamiento consiste en ir formando soluciones comenzado con un conjunto vacío al que se le van añadiendo elementos, que se corresponden con el óptimo local en cada etapa, de tal forma que se escoge en cada momento la mejor solución para el subproblema a tratar sin tener en cuenta las consecuencias en etapas futuras, todo esto hasta alcanzar una solución completa que, si bien no tiene por qué ser la óptima, generalmente es de buena calidad, además de obtenerse de forma rápida. Para ello, es necesario definir una heurística que aporte información del problema, de tal forma que la selección de la decisión ante el subproblema a tratar permita acercarse a la solución óptima. En su implementación, se hace uso de los siguientes elementos:

- Función de selección: obtiene el elemento del conjunto de elementos no explorados con estructura óptima para el subproblema a tratar.
- Función de factibilidad: comprueba que al añadir el elemento anterior se cumplen las restricciones del problema
- Función de valor o función objetivo: devuelve el valor a optimizar por el problema de una solución.
- Adicionalmente se puede hacer uso de una función que comprueba si un estado actual es solución final.

Para ejemplificarlo, se toma como referencia el problema de la mochila (0/1 KP), ya definido en este mismo capítulo. Para el problema de la mochila, una heurística comúnmente utilizada es la relación $valor/peso$ de cada ítem. Por ello, un algoritmo voraz que resuelva un problema como el anterior seguiría el siguiente esquema algorítmico 2.1, tomando C como el conjunto de elementos que pueden conformar la solución, donde x es el estado actual:

```
1: función ALGORITMO VORAZ(elems)
2:     x ← ∅
3:     mientras elems ≠ ∅ & !esSOLUCION(x) hacer
4:         sel ← SELECCION(elems)
5:         elems ← elems − sel
6:         si esFACTIBLE( x ∪ sel) entonces
7:             x = x ∪ sel
8:     devolver x
```

Algoritmo 2.1: Algoritmo voraz

2.1.5. Técnicas metaheurísticas

La palabra metaheurística procede de la combinación del prefijo griego meta, que significa más allá de, y la palabra heurística. Estas se basan en procesos de resolución de forma inteligente que combinan la exploración y explotación del espacio de búsqueda, teniendo como objetivo encontrar soluciones que se acerquen al óptimo. Estas técnicas son de propósito general y son ampliamente usadas para resolver una gran variedad de problemas de optimización complejos[Osman and Kelly, 1996].

El hecho de que sean capaces de realizar una búsqueda eficiente del espacio de soluciones los hace especialmente útiles para resolver problemas difíciles como los $NP - completos$ o $NP - duros$, para los cuales, a pesar de no garantizar una solución óptima, generalmente obtienen una solución aceptable que se acerca, en un intervalo de tiempo significativamente inferior al proporcionado por un algoritmo exacto, en especial para tamaños de entrada grandes, que imposibilitan la obtención de una solución por parte de estos últimos en una cantidad de tiempo aceptable. Además, son métodos muy generales, aplicables a una gran variedad de problemas y fáciles de implementar.

Para una buena aplicación de estas técnicas es necesario lograr un equilibrio en su búsqueda entre explotación, que se refiere al esfuerzo empleado para la búsqueda en una región, y exploración, que hace referencia al empleado para explorar regiones remotas, con ello tratando de evitar pasar mucho tiempo explorando espacios dentro del espacio de soluciones no prometedores y a su vez evitando reexplorar zonas[Flores María Julia, 2021a].

Los algoritmos metaheurísticos pueden clasificarse de una gran variedad de formas. Como indica [Nesmachnow, 2014], la principal forma de clasificar las técnicas metaheurísticas es atendiendo a la forma de recorrer el espacio de soluciones, donde se dividen en dos grandes grupos, los poblacionales y los basados en trayectorias. Otras formas de clasificación, también observables en la figura 2.2, pueden ser su fuente de inspiración, donde se encuentran fenómenos de la naturaleza, como los movimientos de las colonias de hormigas o la genética, o fenómenos físicos, como el enfriamiento del acero, entre otros. Una última forma de clasificación es bien si hacen uso o no de memoria para almacenar resultados.

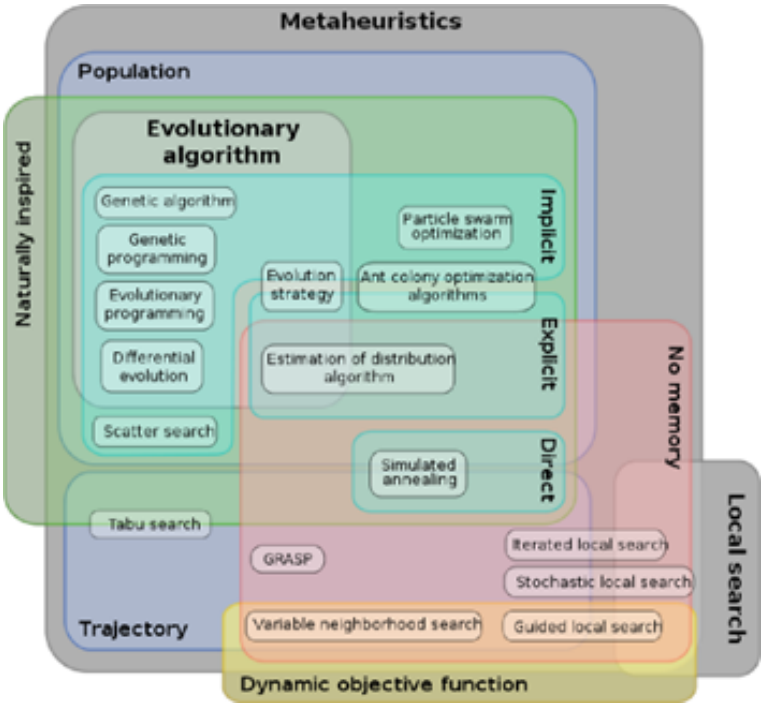


Figura 2.2: Posible clasificación de las técnicas metaheurísticas

Metaheurísticas basadas en trayectorias

Agrupa a aquellas técnicas metaheurísticas que, de forma iterativa mejoran una única solución, realizando una búsqueda en forma de trayectoria dentro del espacio de soluciones. En este grupo se encuentran algoritmos como el de recocido simulado (SA), el de búsqueda Tabú (TS), el de la búsqueda iterada con reinicios (ILS) o GRASP[Shi and Zhang, 2018].

Recocido simulado

También conocido como simulated annealing (SA) es un metaheurístico de búsqueda local que introduce de movimientos ascendentes, es decir movimientos que no son necesariamente óptimos, para evitar quedarse estancado prematuramente en un óptimo local, buscando soluciones mejores a largo plazo, y potenciando así la exploración del espacio de búsqueda[Millán Páramo et al., 2014].

Al igual que otras técnicas metaheurísticas, está basado en un proceso físico, en este caso el del enfriamiento del metal, como propuso [Kirkpatrick et al., 1983]. Este proceso tiene dos fases, la primera se basa en como se incrementa la temperatura del acero en un baño de calor, y en la segunda como disminuye la temperatura de este último lentamente hasta que sus partículas se reorganizan y la energía del sistema es mínima. El procedimiento del algoritmo se divide en las siguientes fases:

En primera instancia, se genera una solución, ya sea de forma aleatoria o voraz, dando lugar a un estado inicial.

A partir de esta, se genera un vecino de forma aleatoria. Para ello, es necesario definir el concepto de vecino, que hace referencia a aquellos estados que pertenecen al conjunto de estados obtenibles mediante un cambio mínimo en una configuración dada.

Por ejemplo, para el caso de una configuración representada por una permutación de elementos, un estado S' vecino de S si cambia de orden la posición de dos de sus elementos. De esta forma, dado un estado S', este estado es vecino del estado S si cumple la expresión siguiente:

$$S' \in N\,(S)$$

Donde N es una aplicación que genera los vecinos dada una configuración.

Además, para aplicar esta estrategia metaheurística es necesario definir un operador de transacción que dado un estado a explorar y el estado actual devuelva la probabilidad de tomar el primero como el estado actual. El más utilizado es el propuesto por Metropolis–Hastings [Bertsimas and Tsitsiklis, 1993], que consiste en que, cuando la función de valor del vecino mejora al actual, se continua la exploración a partir de este siguiendo el procedimiento anterior y se almacena como mejor estado encontrado en el caso de serlo. En cambio, cuando el nuevo estado no mejora al actual, la probabilidad de transición del actual al nuevo viene dada por la siguiente expresión

$$P(S_{i+1} = S') = e^{\left(\frac{1}{T}\cdot \max\{0,\Delta J\}\right)}$$

Donde ΔJ representa la diferencia de la función de evaluación entre el estado actual y el vecino estudiado.

Esta expresión hará más probable una transición al nuevo estado en el caso de que la temperatura sea alta, es decir, cuando el algoritmo se encuentra en sus primeras fases, o si bien la diferencia de valores de la función de valor entre el estado actual y el nuevo es escasa. En cambio, cuando la temperatura se ha reducido o bien por el uso de un parámetro T pequeño o por un numero de iteraciones elevado, o cuando la diferencia de costes es mayor, la probabilidad de transición se reduce.

Este proceso se repite durante N iteraciones, y una vez finalizadas estas se ejecuta la función de enfriado, que decrementa el valor de T. Este último proceso, de igual forma, se repite hasta que se alcance la condición de parada, que podría ser un número de iteraciones determinado o un número de iteraciones en el que el mejor valor obtenido no ha mejorado.

Para disminuir valor de T, que evita en estados avanzados la transición a estados de menor valor, necesario para obtener un equilibro entre explotación y exploración del espacio de soluciones, es importante definir una función de enfriamiento adecuada. La más ampliamente usada en la literatura es la que reduce la temperatura de forma lineal en función de α [[Millán Páramo et al., 2014], de tal forma que $T(t+1) = T(t) \cdot \alpha$, donde α es una constante de reducción de la temperatura tal que $\alpha \in (0, 1)$. En la literatura, de forma menos habitual también se hace uso de funciones que la reducen de forma exponencial.

Con lo anterior, el esquema algorítmico que capturaría su funcionamiento sería el que aparece en el pseudocódigo 2.2, donde T es el parámetro de temperatura inicial, N el número de iteraciones por fase y α representa una constante para decrementar T con el tiempo, también conocido como cooling rate:

```
1:  función RECORRIDO SIMULADO(it,T,α )
2:      s ← GENERASOLINICIAL()                          ▷ Bien de forma informada o aleatoria
3:      s' ← s
4:      mientras !CRITERIOPARADA() hacer                ▷ Atendiendo al criterio a usar
5:          para cada it hacer
6:              vecina ← GENERAVECINO(s)
7:              si vecina.val ≥ s.val entonces
8:                  s ← vecina ACTUALIZAMEJORSOL()(s,s')   ▷ Si mejora la mejor solucion
9:              si vecina.val < s.val entonces
10:                 s ← CALCULATRANSICCION(s, vecina, T)
11:         T ← ACTUALIZATEMPERATURA(T)
12:     devolver s'
```

Algoritmo 2.2: Pseudocódigo de un algoritmo de recorrido simulado

GRASP

GRASP o Greedy Randomized Adaptive Search Procedures es una técnica metaheurística perteneciente al grupo de las metaheurísticas basadas en trayectorias. Esta combina una búsqueda heurística aleatorizada con una fase de búsqueda local para explorar el espacio de soluciones.

La ejecución de GRASP, al igual que sucede con SA se divide en dos fases. En la primera de ellas, denominada fase constructiva, se obtiene una solución mediante un algoritmo voraz, pero introduciendo no determinismo, de tal forma que en cada etapa de la construcción de la solución mediante un método voraz se escoja de forma probabilística entre las mejores decisiones posibles [Feo and Resende, 1995], obteniendo así una buena solución diferente a la obtenida por un voraz determinista.

Una vez obtenida una solución tiene lugar la segunda fase, denominada fase de búsqueda local, donde trata de mejorar la anterior aplicando procedimientos de búsqueda local, remplazando de forma iterativa la solución actual con la mejor solución vecina hasta que esta no sea mejorable, de tal forma que se logra una solución que, a diferencia de la anterior, se trata de la óptima local [Feo and Resende, 1995].

Dado que en la primera fase se producen soluciones de forma no determinista, la repetición de este algoritmo no siempre produce los mismos resultados en cada iteración, por lo que se podría llevar una ejecución iterada que podría mejorar los resultados, pero por el contrario aumentaría la cantidad de tiempo necesario para su ejecución.
El esquema algorítmico se corresponde con el fragmento de pseudocódigo 2.3

```
1:  función GRASP(n_elems,it)
2:      s' ← ∅
3:      valor' ← 0
4:      para cada it hacer                          ▷ Se ejecuta it iteraciones
5:          s ← GREEDYADAPTATIVESEARCH(n_elems)
6:          s ← LOCALSEARCH(s)
7:          value ← FUNCIONVALOR(s)
8:          si value > value' entonces
9:              s' ← s
10:             value' ← value
11:     devolver s'
```

Algoritmo 2.3: Pseudocódigo de GRASP

Donde la función que genera la selección voraz sigue el esquema algorítmico del pseudocódigo 2.4.

```
1: función GREEDYADAPTATIVESEARCH(n)                ▷ n también conocido como Random Factor
2:     s ← ∅
3:     restantes ← CONJUNTOCANDIDATOS()
4:     mientras !ESSOLUCION(s) hacer
5:         candidatos ← MEJORESCANDIDATOS(n)        ▷ Donde n son los elementos por torneo
6:         elem ← SELECCIONALEATORIA(candidatos)    ▷ Dependiente del criterio de seleccion
7:         s ← s ∪ elem
8:         restantes ← restantes − elem
9:     devolver s
```

Algoritmo 2.4: Fase de construcción voraz aleatorizada en GRASP

Metaheuristicas basadas en poblaciones

Se pueden definir las metaheurísticas basadas en poblaciones como aquellas técnicas que imitan el aprendizaje, la evolución y adaptación de las especies. Dentro de este grupo destacan una gran variedad de algoritmos, como indica [He et al., 2018], como el algoritmo genético (GA), el particle swarm optimization (PSO) y el ant colony optimization (ACO).

En este caso el estudio se centrará en el algoritmo genético clásico, que será el implementado.

Algoritmo genético clásico

Algoritmo inspirado en la teoría de la evolución de Darwin, donde solo los individuos más fuertes y adaptados al entorno tienen mayores probabilidades de supervivencia y reproducción [Chahar et al., 2021].

Este algoritmo se basa en poblaciones, conformadas por una cantidad o bien fija o variable de individuos, que representan configuraciones. Estas, atendiendo al problema a resolver, pueden configurarse de forma binaria, numérica, o en forma de permutaciones.

Como indica [Chahar et al., 2021], se conforma una población generando configuraciones ya sea de forma informada, haciendo uso para ello de una heurística definida, o bien de forma aleatoria. Tras ello, de forma iterativa se producen nuevas poblaciones mediante operadores genéticos, definidos posteriormente, haciendo uso de los individuos presentes en la población actual con el objetivo de explorar el espacio de soluciones de forma eficiente.

Entre los operadores genéticos destacan el de mutación y el de cruce. El operador de cruce obtiene dos nuevos individuos a partir de dos padres dados, que se corresponden con dos configuraciones de la población. Este se aplica de forma estocástica a dos individuos de acuerdo a una probabilidad pasada por parámetro, la cual en ocasiones suele ser igual o cercana a 1 [Flores María Julia, 2021b].

Es necesario definir su aplicación, que para representación entera o binaria suele ser por 1 o n puntos, o bien mediante una máscara, mientras que para configuraciones formadas por permutaciones se hace uso de una mayor variedad de operadores, donde se encuentra el 2PCX, que a partir de dos puntos de cruce genera dos individuos donde entre fuera de estos puntos las configuraciones mantienen los valores, mientras que dentro de estos contiene los mismos valores pero en el orden del padre dado.[Flores María Julia, 2021b]

En cuanto al operador de mutación, este se aplica a una configuración dada de forma probabilística, de nuevo, en función de un parámetro de entrada. Este puede realizarse de una amplia gama de formas, entre las que se encuentran la mutación por desplazamiento y la mutación por inversión, y, como resultado, obtiene una configuración similar a la anterior, pero permitiendo explorar configuraciones vecinas en el espacio de búsqueda [Chahar et al., 2021].

Para la selección de los individuos de la nueva población, que será la usada en la próxima iteración, es necesario tomar un subconjunto de tamaño predefinido. Esta obtiene un conjunto de tamaño n, definido como parámetro, y se puede realizar aplicando diversos métodos como la selección mediante una ruleta, por ranking, o por torneo, donde se encuentra entre otros el propuesto en[Chahar et al., 2021].

El algoritmo llega a su fin cuando se alcanza la condición de parada, donde entre las más usadas se encuentran la no mejora del mejor individuo en las últimas n interacciones o la ejecución de un número n de interacciones determinadas.

Como se observa en la 2.3, la cantidad de variantes del algoritmo atendiendo a los operadores y codificación elegida es muy amplia y quedan en mano del programador, por lo que la elección adecuada de estos en función del problema es fundamental para obtener buenos resultados.

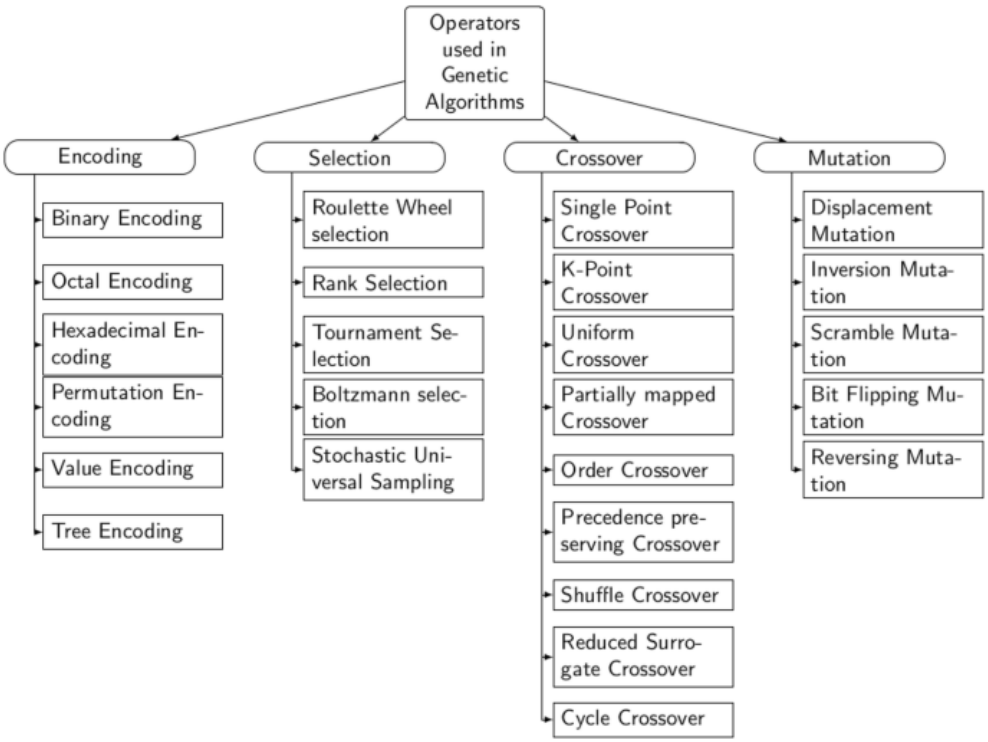


Figura 2.3: Algunas de las variantes en los operadores del algoritmo genético clásico

Con todo lo anterior, el funcionamiento de este algoritmo lo captura el fragmento de pseudocódigo 2.5.

La principal ventaja de este algoritmo es su capacidad para realizar una búsqueda más global, gracias a la introducción y uso de una gran variedad de operadores, y a la mantenimiento de una población, frente a la búsqueda más local que realizan los algoritmos basados en trayectorias. En cuanto a su principal desventaja, resalta la necesidad de definir una gran cantidad de parámetros como la probabilidad de mutación y de cruce, el tamaño de la población o la condición de parada, que afectan de forma directa al rendimiento del algoritmo[Rao and Pawar, 2020].

```
1: función ALGORITMO GENÉTICO(tamPop,pMut,pCruce,nTorneo )
2:     pop ← GENERAPOBINICIAL(tamPop)
3:     mientras !CRITERIOPARADA() hacer
4:         pop' ← SELECCIONAR(pop,nTorneo)          ▷ Atendiendo el criterio de selección
5:         pop' ← OPERACIONCRUCE(pop,pCruce)        ▷ Atendiendo al operador de cruce
6:         pop ← OPERACIONMUTACION(pop,pMut)        ▷ Atendiendo al operador de mutación
7:         pop ← COMBINARPOBLACIONES(pop',pop)
8:     devolver mejorSol                            ▷ Aquella mejor de todas las iteraciones realizadas
```

Algoritmo 2.5: Algoritmo genético clásico

2.2. Propuestas para el problema de carga de camiones

El problema a tratar, denominado problema de la carga de camiones, que en la literatura podemos encontrar como CLP o container loading problem, se trata de una variante del problema de la carga de un container tridimensional o 3D-BPP, distando de este último en una serie de restricciones como la forma de colocación de los ítems, la cantidad de peso soportado o la orientación, entre otros, como indica [Correcher et al., 2017].
En estos se han de cargar un conjunto de ítems de formas ortogonales dentro de un contenedor, también de forma ortogonal, de tal forma que no exista solapamiento entre ítems, donde además cada uno de estos se encuentre dentro de las dimensiones del contenedor y el posicionamiento de sus lados sea paralelo a los lados del contenedor.

La cantidad de variantes atendiendo a las restricciones consideradas de cara a solucionar el problema es muy extensa. La principal clasificación del problema como indica [Ali et al., 2022] es la siguiente:

- Criterios relativos al conocimiento de las instancias a colocar: se distingue entre problema de carga “on-line”, donde llegan de uno en uno y deben ser cargados en su llegada, sin conocer los posteriores, y los problemas de carga “off-line”, donde se conocen la totalidad de ítems en todo momento. La mayor parte de la investigación hace uso del segundo.

- Criterios relativos a las restricciones y atributos considerados: autores, como Bortfeldt [Bortfeldt et al., 2003] diferencian entre dos grandes grupos.
  Por un lado distinguen entre restricciones relativas a la seguridad de la carga, donde se encuentran restricciones como el peso limite, la distribución del peso o la orientación de la carga, y por otro agrupan las relativas a la logística, donde se encuentran tanto las relativas a los envíos completos de todos los ítems a un cliente, como las restricciones relativas al posicionamiento de los ítems para cada cliente, que son denominadas como multi drop constraints, entre otras, todo ello da lugar a una gran variedad de problemas distintos.

La tabla 2.1 muestra la cantidad de papers que abordan las restricciones más comunes, de nuevo, presentada por el autor anterior [Bortfeldt et al., 2003], y donde cabe destacar que existen varias restricciones que apenas se tienen en cuenta en la literatura, por lo que encontrar un artículo donde las restricciones consideradas sean idénticas a las del problema en cuestión es extremadamente improbable.

Tipo de restricción	Número de papers	
	Total	Porcentaje
sin restricciones	35	22.2
peso límite	22	13.9
distribución de peso	19	12
prioridad de carga	2	1.3
orientación de los objetos	112	70.1
apilamiento	24	15.2
envío completo	1	0.6
colocación dentro del camión	26	16.5
estabilidad	59	37.5
complejidad de colocación	15	9.5

Tabla 2.1: Número de papers que abordan el problema por restricción considerada

Tomando como referencia el CLP, tenemos que, en la actualidad, diversos autores hacen uso de métodos exactos para la resolución de este problema. Entre las distintas propuestas se puede encontrar la de [Alonso Martínez et al., 2016], donde se resuelve mediante técnicas de programación lineal, [Ndèye Fatma Ndiaye, 2014], con el uso de técnicas Branch and Cut, variante de Brach and Bound, o [Fanslau and Bortfeldt, 2010], con la exploración del árbol de soluciones, de igual forma con una variante del Branch and Bound. Otros autores también hacen uso de técnicas aproximadas, que, si bien no producen siempre el resultado óptimo, obtienen un buen rendimiento para el problema en cuestión, destacando propuestas como [Jansen and Solis-Oba, 2006] o [Miyazawa and Wakabayashi, 1997].

En cuanto a las técnicas exactas cabe destacar que, como indica [Bortfeldt et al., 2003], no son por el momento técnicas buenas para la resolución del problema, ya que no permiten obtener soluciones en tiempo aceptable para problemas de tamaño de entrada realistas, especialmente si se tienen en cuenta restricciones.

Es por ello por lo que, por el momento, las técnicas heurísticas y metaheurísticas son las únicas aplicables en la práctica para instancias reales.

En la literatura, existen diversas propuestas que hacen uso de algoritmos metaheurísticos para la resolución del problema, como la presente en [Ramos et al., 2017]. Se pueden encontrar propuestas de una gran variedad de algoritmos distintos como GRASP, propuesta en [Parreño et al., 2008] algoritmos genéticos [Gonçalves and Resende, 2012], búsqueda Tabú [Bortfeldt et al., 2003], o búsqueda local en [Mack et al., 2004], obteniendo resultados aceptables y respondiendo de forma adecuada a diferentes tamaños de entrada.

Entre las soluciones aportadas donde se tienen en cuenta las "multi drop constraits" o restricciones de la colocación de objetos atendiendo a los clientes a los que van dirigidos, las más restrictivas del problema, se pueden encontrar las propuestas de algoritmos genéticos en [Kirke et al., 2013], GRASP en [Alonso Martínez et al., 2020] o búsqueda local, como propone [Ceschia and Schaerf, 2010].

Para la simplificación del problema sin restricciones, en este caso el 3D-BPP y para sus otras variantes de 1 y 2 dimensiones 2D-BPP y BPP, en la actualidad se hace uso tanto de técnicas exactas mediante algoritmos como Branch and cut (B&C) o Branch and proce(B&P) como en [Pisinger and Sigurd, 2005] o [Martello and Vigo, 1998], además de otras no exactas, donde se encuentran las técnicas heurísticas y metaheurísticas [Fernández et al., 2010] [Hopper and Turton, 2001].

Cabe destacar el uso de heurísticas para la colocación, que presentan enfoques específicos al problema permitiendo obtener soluciones aceptables de forma rápida. Entre las heurísticas presentes en la literatura, resaltan de forma significativa las siguientes, introducidas por [Divakaran, 2019]:

- First fit: consisten en la colocación del ítem en el primer espacio donde cabe. Es la de menor complejidad temporal.
- Best fit: consiste en colocar el ítem la posición en la que menos espacio deja libre
- Worst Fit: coloca el ítem en el espacio que deja más hueco libre, de tal forma que otros ítems puedan colocarse en este.

3. Descripción del problema

Una vez definidos los algoritmos que se pueden usar para resolver este tipo de problemas, pasamos a tratar de definir de forma minuciosa el problema a tratar.

Para ello, se toma como referencia el challenge ROADEF 2022 [ROADEF, 2022a], que propone la resolución del CLP en su versión off-line para un conjunto de datos de prueba extraídos de fuentes reales, donde en este caso se busca el llenado de un subconjunto de camiones, sometido a un amplio rango de restricciones que se verán a continuación. El estudio del problema se centrará en concreto en una variante de este, donde se ignorarán algunas restricciones del problema original como las fechas de entrega o los puntos de carga, y cuyo objetivo final será el llenado de un único camión.

Este capítulo queda dividido en dos secciones, donde en la primera se presentan de forma más detallada el problema a tratar, estudiando su formulación y en la segunda se describe una metodología de colocación de pallets que permite abordar el problema, estudiando su optimalidad y complejidad.

3.1. Problema a tratar, variables y restricciones consideradas

Como ya se ha indicado, se trata de maximizar el volumen total cargado en un camión, todo ello cumpliendo una serie de restricciones que garanticen el transporte seguro de la carga y permitan mantener un orden en las entregas. Por ello, hemos de identificar los distintos elementos que se nos proporcionan como entradas. Estos son el camión y los pallets.

Notación utilizada

Para la notación de cada uno de los elementos, de tal forma que se facilite la identificación de la gran cantidad de parámetros y variables existentes, se ha seguido el siguiente convenio:

- Conjuntos: se han usado mayúsculas con virgulilla, como sucede con el conjunto de pallets, denotado como $\widetilde{P}$
- Constantes: se usan letras mayúsculas para designar, que almacenan aquellos parámetros del problema que no cambian, como la masa máxima a cargar de un camión, designada como TM, los pallets a nivel individual, designados como P_i, o sus dimensiones $L_i = (L_i^a, L_i^l, L_i^h)$, donde i es el identificador del pallets al que hacen referencia.
- Variables: se han nombrado usando letras minúsculas, tanto para las binarias como para las numéricas. Un ejemplo podría ser la variable orientación elegida, designada como d_i, donde i es el identificador del pallet al que hace referencia.

El camión lo podemos definir con una estructura con forma de prisma recto de base rectangular u ortoedro, y del cual conocemos sus medidas de ancho L_{cam}^a, largo L_{cam}^l y alto L_{cam}^h del contenedor.

Además, para este también se ha de conocer una serie de constantes necesarias para calcular las restricciones de masa, que son:

- TM: masa total autorizada, supone la masa máxima cargable en el camión.
- CM, EM: masas del tractor y del contenedor vacío respectivamente.
- CJ_{fm},CJ_{fc},CJ_{fh}: distancias entre el eje frontal y medio, entre el eje frontal y el centro de gravedad y entre el eje frontal y enganche, respectivamente.
- EJ_{hr},EJ_{cr},EJ_{eh}: distancias entre el enganche el enganche y el eje trasero, entre el centro de gravedad y el eje trasero, y entre el comienzo del contenedor y el enganche, respectivamente.
- EM_{mr},EM_{mm}: peso máximo cargable en el eje trasero y medio respectivamente.

La figura 3.1 ofrece una representación gráfica de estos para una mayor compresión.

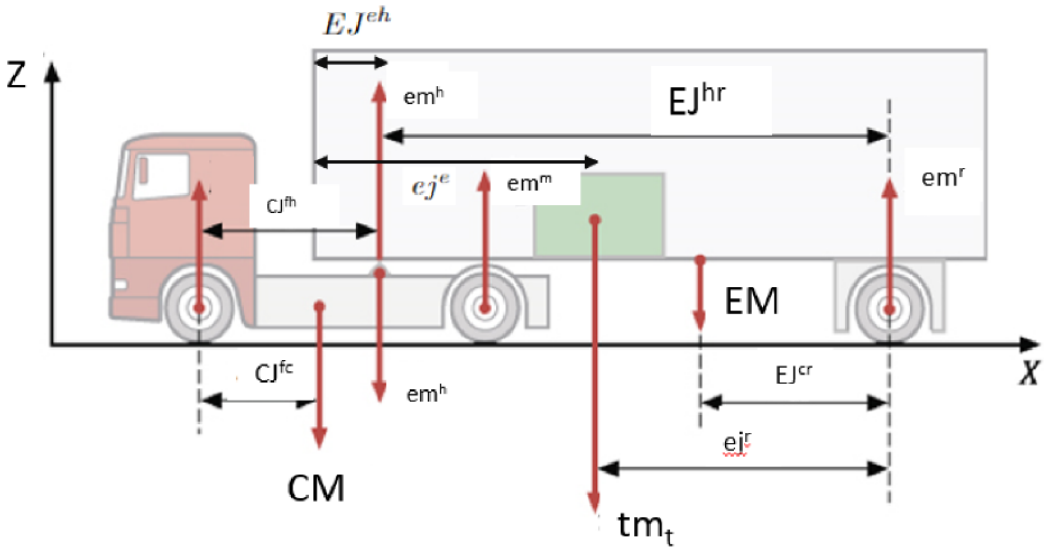


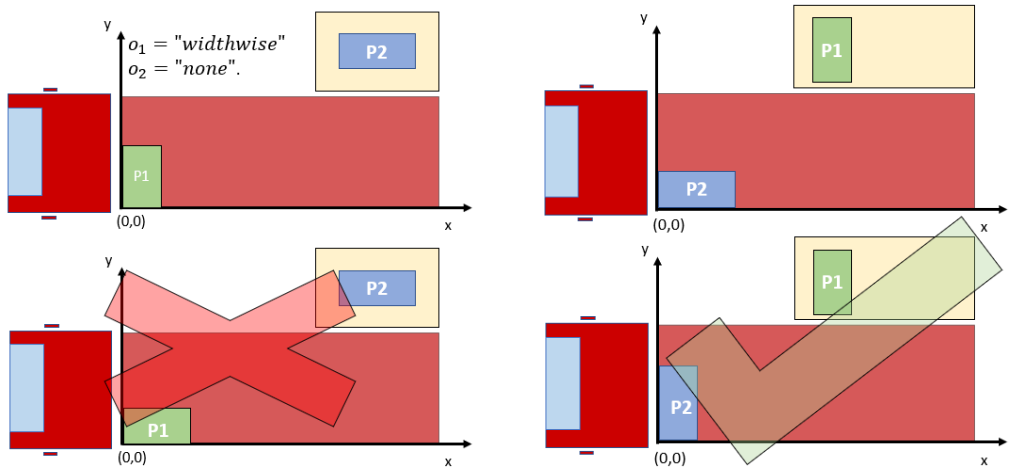
Figura 3.1: Representación gráfica de las constantes relativas a la masa del camión

Como entrada conocemos el conjunto de pallets $\widetilde{P} = \bigcup_i^N P_i$, siendo P_i una instancia de pallet y N el número total de pallets.

Para cada instancia de pallet hemos de conocer tanto sus dimensiones, es decir su ancho L_i^a largo L_i^l y alto L_i^h, además de su peso W_i.

Además, también hemos de conocer si este permite un cambio de orientación en su colocación o, por el contrario, está forzado a colocarse con una orientación determinada. Por ello para cada pallet conocemos su atributo O_i, el cual tiene como valor $widthwise$ en el caso de deber de colocarse su anchura L_i^a y largo L_i^l en el eje x e y del camión respectivamente, o bien contiene como valor $none$ que indica que el ancho se puede colocar en la dirección tanto del eje x como del eje y. Para una mejor comprensión, en la figura 3.2 se pude observar las distintas colocaciones de los pallets P_1, de color verde y con $O_1 = widthwise$, representado en la subfigura 3.2a y P_2, de color azul y donde se tiene que $O_2 = none$, representada en la subfigura 3.2b.

Por último, también se ha de conocer el cliente al que va destinado el pallet en cuestión, cuya importancia se verá detallada en las restricciones del problema. Esto se capturará con el atributo entero C_i, que almacenará el identificador del cliente del pallet P_i.



(a) Orientación fija **(b)** Orientación alternativa

Figura 3.2: Colocaciones posibles atendiendo al valor de O

Para poder formular el problema, se han de introducir diversas variables:

- Variable de selección: s_i, variable de tipo binaria que contiene un 0 en caso de que el pallet P_i no sea seleccionado para ser transportado o un 1 en el caso de sí serlo.
- Posición en el plano: pos_i, indica el punto de apoyo sobre el que se coloca el pallet dentro del contenedor del camión.
  Para su expresión, se hace uso de coordenadas. Asignamos el origen de coordenadas en el punto inferior a la izquierda en la zona más alejada desde la puerta del contenedor, como se puede observar en la figura inferior. Además, determinamos que el eje x se corresponde con la dirección en la que el camión se desplaza, el eje y con la anchura de este, y el z con la altura. Algunas representaciones de puntos las podemos observar en la figura 3.3.

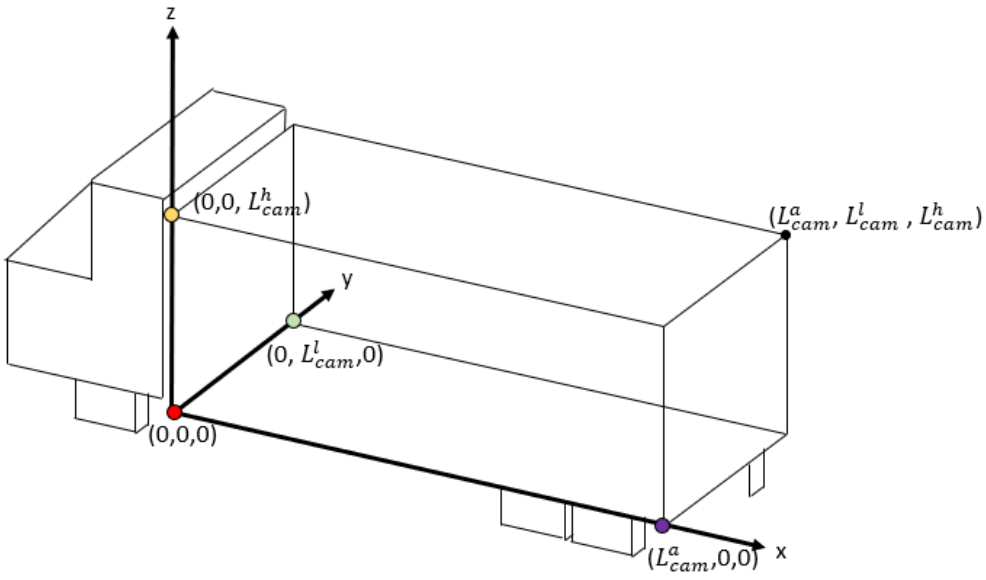


Figura 3.3: Ejemplos de puntos dentro del contenedor

Dado que los objetos no pueden colocarse unos encima de otros y estos siempre han de estar a ras de suelo, es decir con valor 0 en el vector de posición para el eje z, se omitirá la expresión este último para indicar los puntos de apoyo, de tal forma que podemos expresar la posición de un punto sobre el contenedor del camión de la forma (pos_i^x, pos_i^y) en lugar de $(pos_i^x, pos_i^y, 0)$. Un ejemplo podría ser el de la figura 3.4.

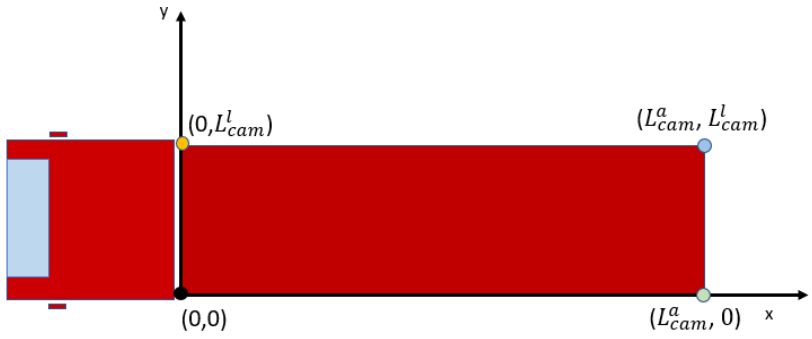


Figura 3.4: Ejemplos de puntos en la superficie del contenedor

- Orientación elegida: d_i variable binaria que almacena un 0 cuando la orientación en la que un ítem es colocado es la natural, es decir, la dimensión L_i^a del pallet situada en paralelo al eje x del camión, la L_i^l paralela al eje y y la L_i^h al z, o bien un 1, que indica que la dimensión L_i^a y L_i^l se sitúan de forma paralela al eje y y al eje x respectivamente.

Con lo anterior, al tener como función de evaluación la cantidad de volumen cargado, y como criterio de optimización maximizar, se busca el máximo de la siguiente expresión:

$$\sum_{i \in P} L_i^a \cdot L_i^l \cdot L_i^h \cdot s_i$$

El objetivo es conocer el conjunto de pallets y sus posiciones que conforman la expresión que optimiza la función de evaluación, cumpliendo a la vez con las restricciones del problema.

Las restricciones que se deben cumplir para determinar que una solución es correcta, garantizando las condiciones de seguridad de la carga y, permitiendo la descarga eficiente de la mercancía son las siguientes:

- Restricción que impide que la carga salga por el ancho, largo o alto del camión. Se puede formalizar de la sigueinte forma:

$$(0 \leq L_i^a + \mathrm{pos}_i^x \leq L_{cam}^a) \ \forall i : (s_i = 1) \wedge (d_i = 0) \quad (1.1.1)$$

$$\left(0 \leq L_j^l + \mathrm{pos}_i^x \leq L_{cam}^a\right) \ \forall i : (s_i = 1) \wedge (d_i = 0) \quad (1.1.2)$$

$$\left(0 \leq L_i^l + \mathrm{pos}_i^y \leq L_{cam}^l\right) \ \forall i : (s_i = 1) \wedge (d_i = 0) \quad (1.2.1)$$

$$\left(0 \leq L_j^a + \mathrm{pos}_i^y \leq L_{cam}^l\right) \ \forall i : (s_i = 1) \wedge (d_i = 1) \quad (1.2.2)$$

$$\left(0 \leq L_i^h \leq L_{cam}^h\right) \ \forall \, i : \ (s_i = 1) \quad (1.3)$$

- Restricción que impide el solapamiento de los distintos pallets: para facilitar la formulación de esta, se añaden las siguientes variables binarias $iz_{i,j}$ y $\mathrm{ab}_{i,j}$. La primera indica si un pallet P_i esta completamente a la izquierda de otro pallet P_j, o lo que es lo mismo, que el fin en el eje x de P_i es anterior o igual al comienzo de P_j en el mismo eje. Sucede lo mismo con ab, salvo que esta indica que P_i esta completamente debajo de P_j.

$$iz_{i,j} = 1 \Rightarrow \left[\left(L_i^a + \mathrm{pos}_i^x \leq \mathrm{pos}_j^x\right) \wedge (d_i = 0)\right] \vee \left[\left(L_i^l + pos_i^x \leq pos_j^x\right) \wedge (d_i = 1)\right]$$

$$\mathrm{ab}_{i,j} = 0 \Rightarrow \left[\left(L_i^l + \mathrm{pos}_i^y \leq \mathrm{pos}_j^y\right) \wedge (d_i = 0)\right] \vee \left[\left(L_i^a + pos_i^y \leq pos_j^y\right) \wedge (d_i = 1)\right]$$

Usando estas se ha de cumplir la siguiente expresión, relativa al no solapamiento de los pallets elegidos:

$$(iz_{i,j} + iz_{j,i} + ab_{i,j} + ab_{j,i}) \geq 1 \ \forall i, j \ \neq j, s_i = 1, s_j = 1 \qquad (2)$$

- Restricción que impide que la carga se desplace hacia adelante al frenar debido a la inercia: esto se podrá evitar si no existe espacio tras el objeto en la dirección de frenado del camión, es decir, si en el eje x, inmediatamente antes del inicio del objeto se sitúa otro objeto, impidiendo que se desplace al frenar.

$$(pos_i^x = 0) \vee \left(\sum_{j=1|j\neq i}^{N} \left(ady_{i,j}^{izq} \cdot s_j \right) \geq 1 \right) \forall i : s_i = 1 \qquad (3)$$

Donde $ady_{i,j}^{izq}$ es una variable de tipo binaria que indica que el pallet P_i es adyacente a P_j por su lado izquierdo, lo que quiere decir que el fin de P_j en el eje x coincide con el inicio de P_i. Esto se calcula mediante la siguiente expresión:

$$ady_{i,j}^{izq} = \begin{cases} pos_j^x + L_j^a = pos_i^x & \text{si } d_j = 0 \\ pos_j^x + L_j^l = pos_i^x & \text{si } d_j = 1 \end{cases}$$

Para una mejor comprensión, en la figura 3.5 se muestran los pallets P_1, P_2 y P_3, donde los dos primeros cumplen las restricciones, mientras que P_3 no lo hace. Por un lado, cumplen P_2 por tener $pos_2^x = 0$, es decir, ser adyacente al camión por su izquierda, de tal forma que al frenar no se deslice, y P_1 por ser adyacente un pallet, en este caso P_2, por su izquierda, al cumplir que $(pos_2^x + L_2^a = pos_1^x) \wedge (d_1 = 0)$.En cambio, P_3 no es adyacente ni al camión ni a ningún pallet por su izquierda y, por lo tanto, una solución de estas características no sería factible al incumplir, al menos, una de las restricciones.

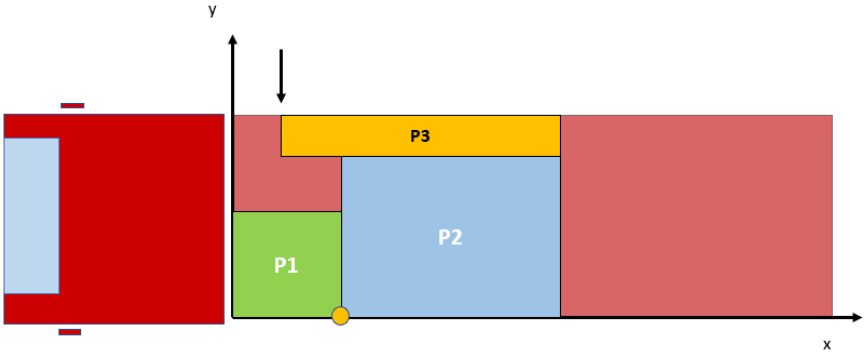
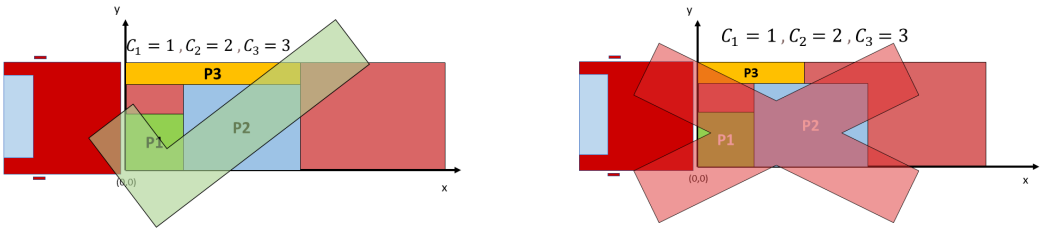


Figura 3.5: Un ejemplo de distribución de pallets que no cumple la condición (3)

- Restricción en el orden de entrega: dado que existe un orden de entrega predeterminado atendiendo al cliente al que pertenecen, los objetos deben de ser extraíbles del camión en un orden dado. Para que esto suceda, los pallets tendrán fin en el eje x menor o igual que el fin de otros pallets de clientes de índices mayores, de tal forma que los pallets de clientes más bajos sean extraidos en último lugar. Un ejemplo de esto se ve en la 3.6a, donde en su parte izquierda muestra una disposición correcta de pallets, ya que el fin en x de los pallets del cliente 3 es mayor o igual a los del cliente 1 y 2, y el fin de los del cliente 2 es mayor a los del 1. En la 3.6a se muestra una disposición incorrecta, ya que el fin del pallet P_3 tiene un fin en el eje x al de P_2, mientras que $C_3 < C_2$, incumpliendo la restricción.



(a) Restricción cumplida

(b) Restricción incumplida

Figura 3.6: Ejemplos de la restricción 4

Esta restricción se puede formalizar mediante la siguiente expresión:

$$\forall i, j : (s_i = 1) \wedge (s_j = 1) \wedge (s_i \neq s_j) \Rightarrow (C_j \leq C_i) \vee (izeq_{i,j}) \qquad (4)$$

- Restricción en el peso de los ejes: la carga debe de cumplir una serie de restricciones de peso en los ejes en cada uno de los desplazamientos. Se distinguen dos grupos de restricciones de este tipo, la de masa general y la de masa en un eje:

  Por un lado, la restricción de masa general indica que la masa total cargada no ha de superar la del valor de la constante TM del camión. Se comprueba mediante la siguiente expresión

$$\sum_{i}^{N} W_i \cdot s_i \leq TM \qquad (5)$$

  Por otro lado, la restricción de masa en ejes comprueba que la masa cargada tanto por el eje delantero como por trasero supera los límites establecidos por ley para el camión dado, para ello respetando las fórmulas ya definidas en [ROADEF, 2022a]. Estas son:

$$em^m \leq EM^{mm} \qquad (6.1)$$

$$em^r \leq EM^{mr} \qquad (6.2)$$

  Donde los valores de em^m, y de em^r se calculan de la siguiente forma:

$$em^r = tm_t + EM - em_h$$

$$em^r = \frac{CM \cdot CJ^{fc} + em^h \cdot CJ^{fh}}{CF^{fm}}$$

$$em^h = \frac{\left(\sum_i^N W_i * s_i\right) * ej^r + EM * EJ^{er}}{EJ^{hr}}$$

$$ej^r = EJ^{eh} + EJ^{hr} - ej^e$$

$$ej^e = \frac{\sum_i^N s_i \cdot W_i \cdot centro_i}{\sum_i^N W_i \cdot s_i}$$

  donde $centro_i$ indica el punto de equilibrio en el eje x para P_i, conociendo tanto su posición pos_i^x, además de su orientación d_i.

- Por último, se ha de respetar la restricción que captura que ninguno de los pallets con orientación fija se coloca con una orientación distinta a esta. Esto se comprueba mediante la siguiente expresión:

$$d_i = 0 \, \forall i : \; (s_i = 1) \wedge (d_i = 0) \qquad (7)$$

Complejidad temporal

El problema de carga de camiones se trata de una variante del 3D-BPP, problema del empaquetamiento en tres dimensiones, donde el primero difiere de este último en alguna de sus restricciones y la inclusión de alguna variable como la del cliente. Como ya demostró [Martello and Vigo, 1998], el problema 3D-BPP es $NP - duro$, y, dado que el problema se trata de una simple variante del anterior, donde se introducen variables que aumentan aún más su complejidad, se puede determinar que este problema también es $NP - duro$.

Esto implica que no existe un algoritmo de complejidad polinomial capaz de resolver el problema para cualquiera de sus instancias de entrada. Sabiendo esto, se pasa a determinar cuál es la cota superior de un algoritmo exacto que lo resuelva, que será exponencial.

Para ello primero cabe destacar que el tamaño de entrada se corresponde con el número de ítems a colocar. Antes de calcular el tamaño del espacio de soluciones es necesario determinar las estrategias de colocación, que permiten limitar el número de posiciones de apoyo de cada ítem a las de un conjunto finito.

3.2. Estrategias para la colocación de los pallets y sus características

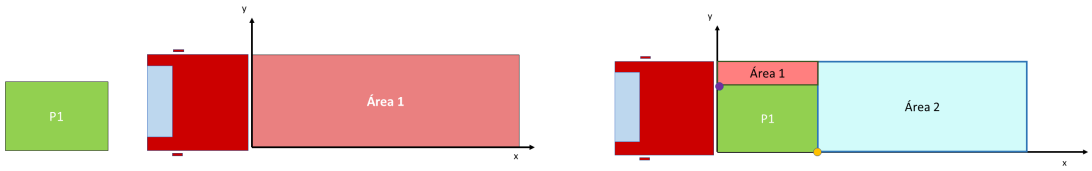
En primera instancia se describirán las estrategias propuestas, y, tras esto, se evaluará su optimalidad y tamaño del espacio de soluciones generado de la estrategia seleccionada.

3.2.1. Estrategia utilizada

Para la estrategia de colocación se han valorado las siguientes alternativas:

- Estrategia usando áreas: es la usada en la mayoría de las propuestas de la literatura. Esta consiste en la generación de áreas donde colocar cada ítem. El algoritmo comienza con un área grande que supone las dimensiones totales del contenedor. En cada área que se decide colocar un ítem, esta se divide hasta en dos nuevas, que se corresponden con los dos espacios que deja el ítem al apoyarse en la esquina inferior izquierda de esta. Por lo tanto, si al colocar el ítem este deja espacio en su lado derecho se genera una nueva área, y si lo deja en la parte superior se genera otra. En el caso de no dejar ningún espacio no se genera ninguna nueva área.

El procedimiento continua hasta no quedar ningún ítem o área disponible .Una representación visual se encuentra en la 3.7, donde la colocación del ítem 1 deja dos nuevas áreas.



(a) Estado previo **(b)** Estado al colocar ítem

Figura 3.7: Ejemplo de metodología usando áreas

El gran problema que presenta es el posible incumplimiento de la restricción relativa a la seguridad de la carga en el frenado (6), ya que es común que se generen áreas a partir de las cuales existan bloqueos, como se observa en la 3.8, donde salvo que se logre colocar un ítem o conjunto de ítems tal que la suma de sus anchos sea exactamente la del área 2, algo muy improbable, no se podrán colocar ítems en las áreas 3 y 4.

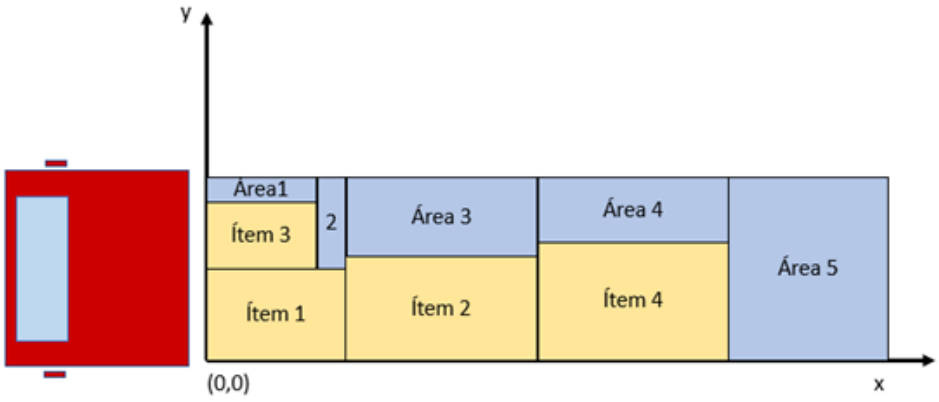


Figura 3.8: Un ejemplo de un bloqueo usando la estrategia de áreas

- Estrategia usando corners: esta estrategia, elegida para la implementación, consiste en definir conjunto finito de puntos a partir de los cuales un nuevo pallet, en este caso su extremo inferior izquierdo, puede colocarse. Por lo tanto, dado un punto de apoyo $pos_i = (pos_i^x, pos_i^y)$ sobre el que se coloca P_i con $O_i = 0$, de dimensiones $\left(L_i^a, L_i^l, L_i^h\right)$, el pallet abarcaría desde este punto hasta el punto $(L_i^a + pos_i^x, L_i^l + pos_i^y, L_i^h)$, siendo este el extremo superior derecho de su parte superior. El algoritmo se inicializa con un único punto de inserción, el origen de coordenadas $(0, 0)$ y cada vez que se añade un pallet se añaden dos puntos nuevos: El primero de ellos en el extremo superior derecho del pallet colocado, es decir, en $(L_i^a + pos_i^x,\ pos_i^y)$, como se observa en la figura 3.9.

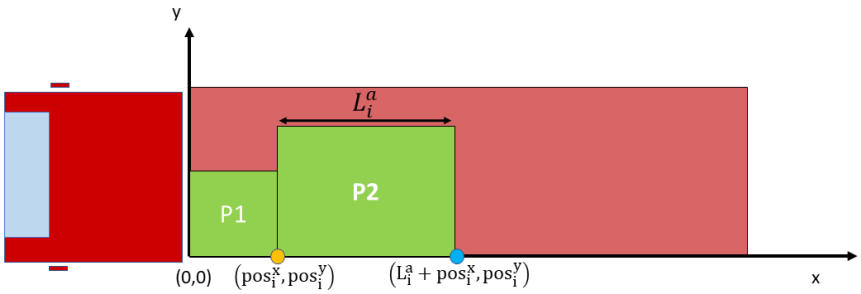


Figura 3.9: Primer punto generado mediante la estrategia de puntos

Otro segundo punto en la proyección hacia el eje y del extremo superior izquierdo del pallet, que formalizándolo, el pallet buscado en caso de existir sería tal que:

$$arg\max_i \left((pos_i^x + L_i^x) \cdot s_i\right)$$

Esto hace referencia al primer punto que se encuentra a la izquierda desde el extremo superior izquierdo, es decir, desde el punto $\left(pos_i^x, L_i^h + pos_i^y\right)$ en caso colocarlo con su orientación habitual. Un ejemplo se encuentra en la figura 3.10, donde al colocar el pallet P_5 se generan tanto el punto morado (correspondiente con el primero punto), como el punto azul oscuro, obtenido mediante la proyección del punto azul claro.

Además, para este último si no se encuentra ningún pallet al realizar la proyección desde el extremo superior izquierdo, el nuevo punto se apoyará en la pared izquierda del camión, es decir, obteniendo un punto con $pos_i^x = 0$. Esto puede observar en la figura 3.11, donde al colocar P_4 se obtiene el punto azul oscuro.

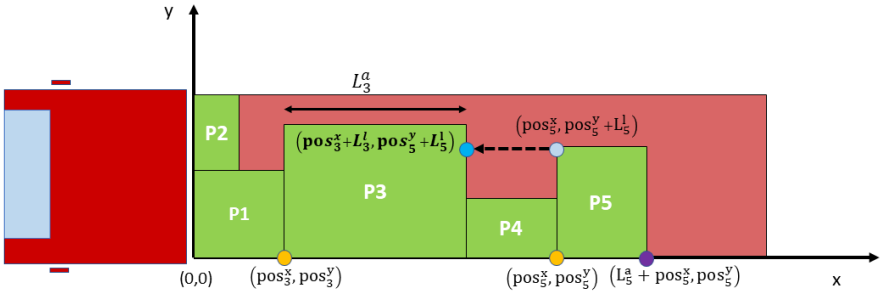


Figura 3.10: Segundo punto generado mediante la estrategia de puntos

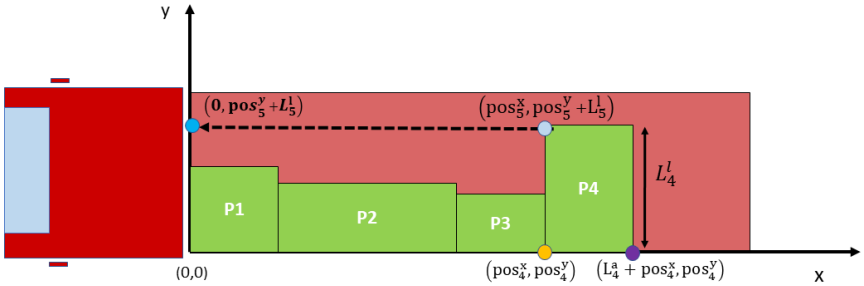


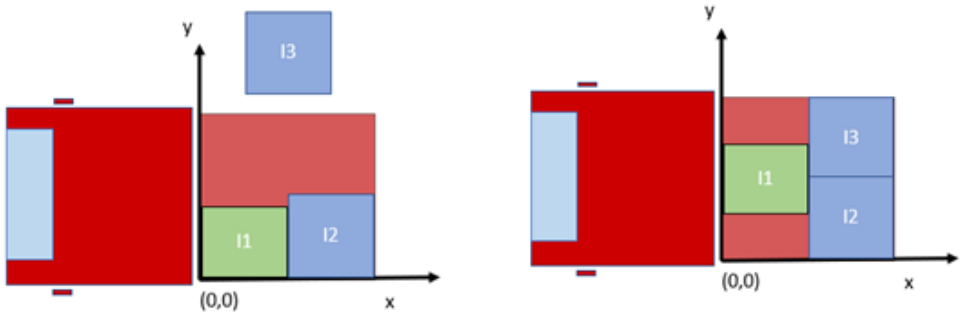
Figura 3.11: Segundo punto generado mediante la estrategia de puntos

3.2.2. Optimidad de la estrategia utilizada

Por último, se estudiará si la solución óptima es siempre aplicando la metodología de colocación anterior. Para ello se tratará de demostrar la verdad del enunciado “dada una solución obtenida mediante la metodología de puntos no existe otra solución factible obtenida por cualquier metodología con función de valor mejor que la obtenida por puntos” o lo que es lo mismo, “la solución óptima del problema siempre se puede obtener mediante la metodología de puntos”.

Por lo tanto, si se logra demostrar que “existe un caso donde para la mejor solución obtenible por la metodología existe al menos otra factible que es mejor”, quedará demostrado que la afirmación es falsa.

Dado la instancia de problema donde existen 3 ítems de entrada, donde el ítem 1 pertenece al cliente 1 y los ítems 2 y 3 al cliente 2, donde además se tiene que las dimensiones de los tres son iguales salvo la altura del primero que es ligeramente inferior, y donde las dimensiones del camión son el doble tanto en el eje x e y, se demuestra que la metodología impide alcanzar el óptimo, ya que la mejor solución factible, representada en la figura 3.12a, es peor que la mejor obtenible factible, representada en la figura3.12b, por ende, no siendo óptima la mejor solución obtenible por la metodología.



(a) Óptima por puntos **(b)** Solución óptima

Figura 3.12: Ejemplos del cumplimiento y no cumplimiento de (4)

Por lo tanto, queda demostrado que el uso de la heurística de colocación no siempre garantiza alcanzar el óptimo, aun haciendo una exploración exhaustiva de su espacio de soluciones.

3.2.3. Tamaño del espacio de soluciones para la estrategia considerada

Una vez seleccionada la estrategia que permite definir los puntos donde se puede colocar un ítem, se pasa a determinar cuál sería el tamaño del espacio de soluciones a explorar por un algoritmo que realice una búsqueda exhaustiva.

Cabe destacar que una solución óptima podrá contener desde 0 a n pallets, siendo n el número de pallets de entrada, por lo que los conjuntos de menos de n elementos también tendrán que ser explorados.

En cuanto al orden de selección y los puntos de colocación elegidos, dado que el algoritmo se inicializa con un único punto de colocación, $(0, 0)$, las posibilidades de selección de pallets y puntos en el primer movimiento son, como máximo $2n$, dado que hay n pallets, de los cuales cada uno puede colocarse como máximo en 2 orientaciones diferentes. Una vez colocado, se desbloquean dos nuevos puntos de colocación, eliminando el existente, pasando a tener $2\,(n-1)\,\cdot 2$ colocaciones distintas, dado que se han de tener en cuenta la colocación de todos los pallets restantes en sus dos orientaciones posibles con los dos los puntos disponibles.

Para que sea más fácilmente comprensible, se ha elaborado el árbol de expansión para una entrada de tres pallets, que puede observarse en la figura 3.13.

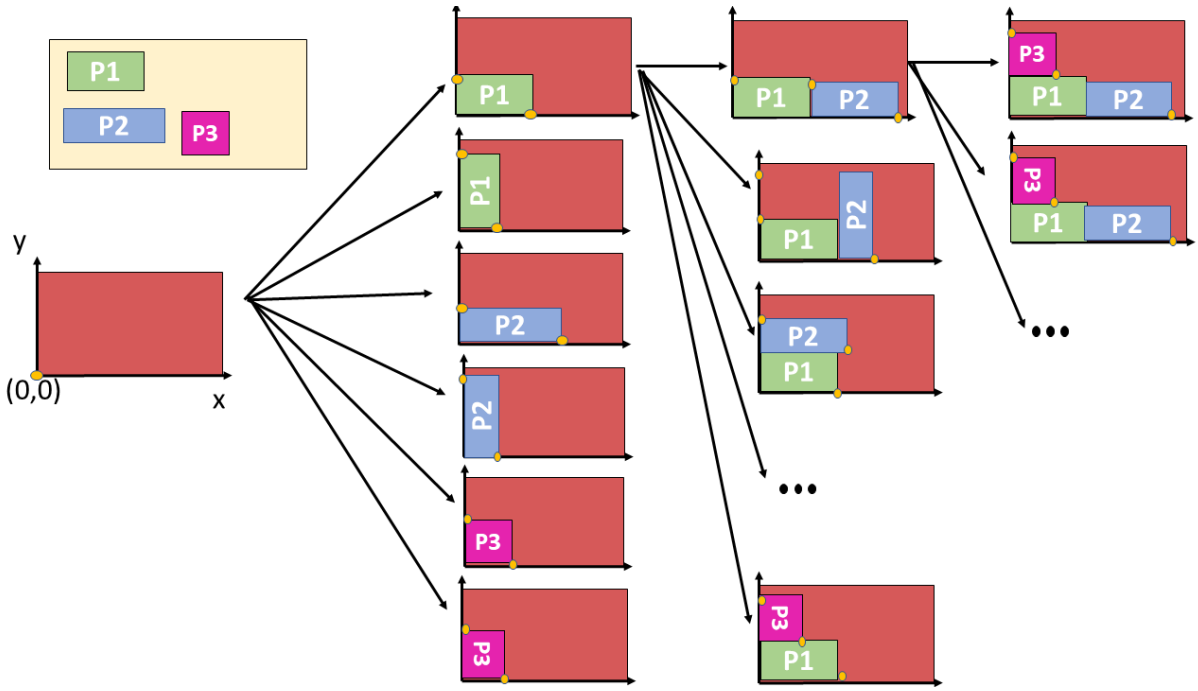


Figura 3.13: Árbol de expansión para la estrategia utilizada

Este patrón lo podemos formalizar mediante la siguiente expresión, donde m es el número de pallets que queremos colocar en total, que indica la cantidad de nodos con profundidad m del árbol de expansión.

$$\prod_{i=1}^{m} 2\,(n-i+1)\cdot(i)$$

Por lo tanto, la cantidad de combinaciones posibles para el problema en cuestión la podemos expresar mediante la siguiente expresión, ya teniendo en cuenta que debemos de estudiar tanto las soluciones que contienen desde 1 pallet hasta N.

$$\sum_{m=0}^{N}\prod_{i=1}^{m} 2\,(N-i+1)\cdot(i)$$

Tomando una cota superior, tenemos que:

$$\sum_{m=0}^{N}\prod_{i=1}^{m} 2\,(N-i+1)\cdot(i) \leq (N)\cdot\prod_{i=1}^{N}(2N-i+1)\cdot(i)$$

Que mediante cotas superiores se obtiene que:

$$(N)\cdot\prod_{i=1}^{N}(2N-i+1)\cdot(i) \leq (N)\cdot(2N\cdot N)^{N}\,N\cdot(N\cdot N)^{N} = N\cdot N^{2N} = N^{2N+1}$$

Con esto se puede determinar que, haciendo uso de la metodología propuesta de colocación, el tamaño del espacio de soluciones es descomunal, donde si usa un algoritmo que realice una búsqueda exhaustiva para recorrerlo este tendría una complejidad de $O(N^N)$, siendo esta extremadamente alta, incluso comparándola con problemas $NP-completos$ como el 0/1 KP, donde la complejidad complejidad sería de $O(2^N)$.

Es por ello por lo que las técnicas exactas no son adecuadas para resolver este problema, para el que deberían de tratar de aplicarse técnicas heurísticas o metaheurísticas que, si bien no garantizan la obtención de la solución óptima, generan una que se acerca en tiempo adecuado si son correctamente implementadas.

4. Implementación

Para la implementación y testeo de los algoritmos, se ha decidido seguir siguiente metodología, de tal forma que permitirá seguir un procedimiento ordenado, evitando errores y dando lugar a un código más limpio y ordenado. En esta se consideran las siguientes etapas:

1. Obtención del benchmark propuesto, implementación de una función de generación de instancias de problema auxiliar y análisis de los pallets que lo componen.
2. Implementación del método para la representación gráfica de los datos y definición de las estructuras de datos comunes.
3. Implementación de las heurísticas de colocación, algoritmos voraces y testing de estos.
4. Implementación de los algoritmos metaheurísticos.

4.1. Obtención de benchmarks de entrada

Las entradas de datos del challenge ROADEF, en los que se basa la implementación, son accesibles en los directorios dataset.zip de su repositorio en Github[ROADEF, 2022b].

Dentro de este repositorio se pueden encontrar conjuntos de directorios que representan diferentes instancias del problema. En cada instancia se almacenan tres ficheros csv, donde uno de ellos, denominado input_trucks, contiene instancias de camiones, entre las cuales se tomará la primera, otro, denominado input_items almacena el conjunto de pallets y sus características y el tercero, denominado input_parameters, contiene atributos propios del challenge propuesto, pero que para el problema propuesto carecen de relevancia y, por lo tanto, no será necesario su uso. Para probar el algoritmo se han usado los ficheros del directorio dataset_C1 para su instancia del problema “AS”.

4.1.1. Instancia de camión

Dentro del fichero que almacena los camiones, será necesario tomar el subconjunto de las variables relevantes para el planteamiento del problema propuesto. Por lo tanto, los atributos que serán necesarios cargar serán: Length, Width, Height, EMmm, EMmr, CM, CJfm, CJfc, CJfh, EM, EJhr, EJcr y EJeh, cuyo significado se vio en el apartado anterior. Para un mejor procesamiento este se almacena en una lista que contiene por un lado una tupla con las dimensiones espaciales y un diccionario con las restricciones de masa. Un ejemplo se puede observar en la 4.1.

```
([14940.0, 2500.0, 2950.0], {'Max weight': 24000.0, 'EMmm': 12000.0, 'E
Mmr': 31500.0, 'EJhr': 7630.0, 'EJeh': 1670.0, 'EM': 7300.0, 'EJcr': 23
50.0, 'CM': 7808.0, 'CJfc': 1040.0, 'CJfh': 3330.0, 'CJfm': 3800.0})
```

Figura 4.1: Ejemplo de una instancia de camión almacenada en una estructura de datos

4.1.2. Instancias de pallets

Estas quedan contenidas en los ficheros input_items, del que, de nuevo, se toma el subconjunto de variables necesarios.

En este caso para su almacenamiento se hace uso de Dataframes de la librería de python pandas, de tal forma que se pueda operar con ellos de forma rápida, de vital importancia para tamaños de entrada grandes. Una vez extraídos se lleva a cabo un pequeño preprocesamiento donde modifican tipos y variables para facilitar el uso de estos por el algoritmo, donde por ejemplo se sustituyen los valores de la orientación $none$ por el entero 0, y $widthwise$ por 1.

Si se tratan de visualizar los datos del benchmark es importante destacar que, la altura, una de las variables que influyen directamente en los resultados del problema, se repite en muchas de las instancias. En la figura 4.2 se puede observar la distribución de alturas para las instancias de primer csv del benchmark, donde la inmensa mayoría de los valores únicos de altura se repite en 50 o más instancias.

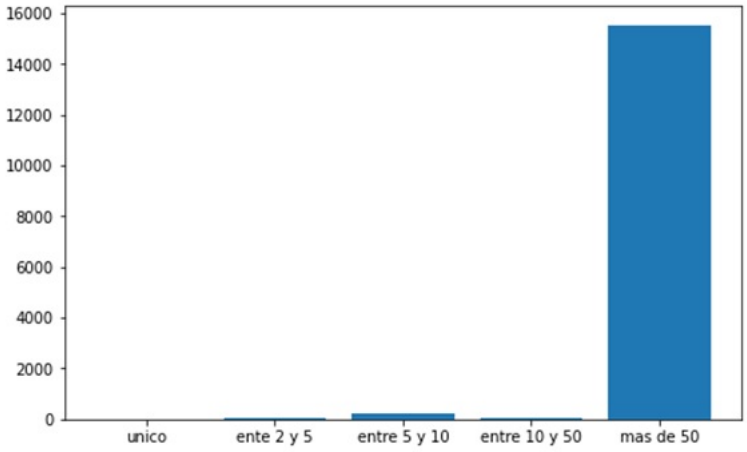


Figura 4.2: Número de veces que se repite un valor único de altura

Además, al ser un challenge las soluciones de este tampoco están disponibles, siendo necesario hacer uso de algoritmos exactos para evaluar la optimalidad de las soluciones generadas por algoritmos heurísticos, algo que no es viable, y más aún para tamaños de problema de este tamaño.

Es por ello por lo que, para un mejor testeo, se ha implementado un generador de instancias adicional, que aporte datos más variados y que permita validar el funcionamiento del algoritmo, al generarse a partir de unas distribuciones estadísticas conocidas.

4.1.3. Fuente alternativa de instancias

Para ello, se implementa la función generaPales con los siguientes parámetros.

- Semilla: usada para garantizar la reproducibilidad del experimento.
- Tamaño de entrada: número total de pallets que se deseen generar.
- Número de clientes: número de clientes totales.
- Dimensiones del camión

Con esto se generan pallets que siguen las siguientes distribuciones:

- Altura: se ha decidido hacer uso de una distribución normal o gaussiana, con media del 60% de la altura del camión, y que con un 95% de probabilidad este no sobrepase la altura del camión, de tal forma que la gran mayoría de pallets tengan alturas al 50%, mientras que solo unos pocos tengan valores buenos. Con esto se tiene que:

$$P\left(Z \leq \frac{\mu - L^z_{\text{cam}}}{\sigma}\right) = 0.95$$

Por lo que se tiene que:

$$\frac{L^z_{\text{cam}} \cdot 0.6 - L^z_{\text{cam}}}{\sigma} = 1.64$$

La distribución de alturas obtenida es la presente en la figura 4.3, donde se marcan de rojo los pallets que, por su altura, no cabrían en el contenedor.

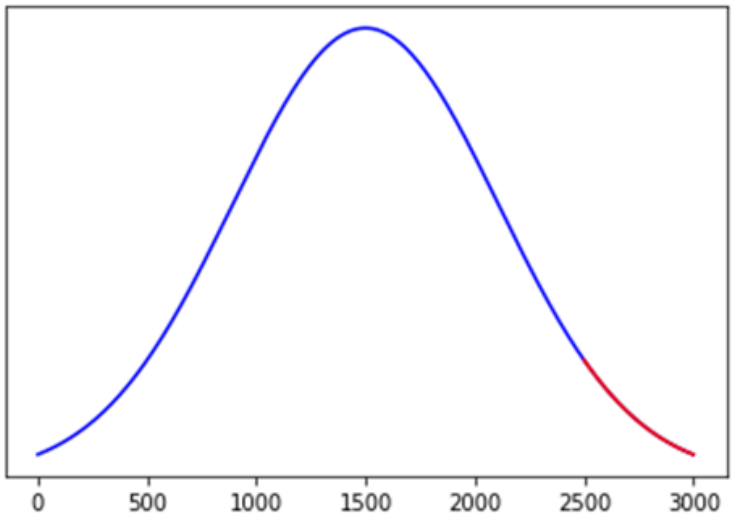


Figura 4.3: Distribución que genera las alturas para el conjunto de entrada propuesto

- Anchura: se genera mediante una distribución normal con media $\mu_x = 0.8 \cdot L^y_{cam}$ y varianza $\sigma_x = L^y_{cam}/8$. Cabe destacar que, para evitar los casos donde la distribución obtiene valores negativos se usa el valor absoluto de la cuantía devuelta por la distribución
- Longitud: igual a la de anchura, pero con media $\mu_x = 0.7 \cdot L^y_{cam}$ y $\sigma_x = L^y_{cam}/8$.
- Id_cliente: distribución uniforme atendiendo al parámetro de entrada.

- Orientación permitida: distribución uniforme discreta, donde la probabilidad de orientación fija y alternativa es la misma.

Un ejemplo de la llamada a esta función usando como parámetros 10 pallets, 5 clientes y semilla=100 y como camión uno con dimensiones $(10, 10, 10)$ obtiene como resultado el conjunto de pallets de la figura 4.4.

	Item ident	Supplier code	Length	Width	Height	Weight	Forced orientation
0	0	4	8.404253	4.407450	12.670802	47.0	0
1	1	4	7.430422	3.092090	3.903528	24.0	0
2	2	5	7.950117	5.785968	8.617742	37.0	1
3	3	5	8.478036	6.070865	8.585334	40.0	0
4	4	2	6.865948	3.487192	4.412297	42.0	1
5	5	2	8.352323	2.841427	8.727270	47.0	1
6	6	5	8.572578	4.444161	2.308517	34.0	1
7	7	2	8.974378	4.138133	6.454156	40.0	1
8	8	5	7.886949	5.967257	0.445754	49.0	0
9	9	1	7.369955	3.750290	6.653570	49.0	1

Figura 4.4: Ejemplo de una llamada a la función generaPallets()

4.2. Implementación del método para la representación gráfica y definición de las estructuras de datos comunes

Antes de comenzar a implementar algoritmos capaces de resolver este problema, se considerado necesario definir como se estructurarán tanto la solución parcial como la solución final, además de la función de evaluación, comunes a todos los algoritmos. También se describirá la función usada para la representación gráfica de soluciones.

4.2.1. Solución parcial

Almacena las decisiones que han sido tomadas hasta el momento tras la exploración de los pallets, almacenando tanto los pallets rechazados en la solución como los aceptados para que formen parte de esta.

Para este problema, podemos definir una solución parcial como un conjunto de pallets junto la posición de cada uno de ellos en el plano del contenedor, expresados en el apartado anterior como tuplas de la forma (pos_i^x, pos_i^y) y su orientación elegida d_i, necesaria al menos, para aquellos con orientación no fija.

En el código, para los pallets que se ha decidido no colocar se ha almacena como un booleano que contiene $False$, mientras que para las posiciones sin explorar se almacena un $None$.

Con ello, una solución parcial será una lista de tamaño n, siendo n el número de pallets de entrada, donde en cada tupla se almacena un elemento del conjunto siguiente:

$$\{False, None, (id_i, (pos_i^x, pos_i^y), d_i)\}$$

Cabe observar que para acelerar cálculos se introduce el identificador en la tupla de pallets colocados, de tal forma que se pueda acceder a variables del dataframe como sus dimensiones de forma directa, sin tener que almacenar las dimensiones de este u otros atributos en la solución parcial. Un ejemplo de la solución parcial podría ser el siguiente, que corresponde con la no elección del pallet de $id = 1$, la colocación del pallet de $id = 2$ en el punto $(0, 0)$ con orientación alternativa, y la no exploración del pallet 3:

$$[False, (2, (0, 0), 1), None]$$

4.2.2. Solución final

En lo relativo a las soluciones finales o soluciones, estas difieren de las soluciones parciales en que en ellas se ha explorado la colocación de la totalidad de los pallets, por lo que en cada posición se almacena o bien $False$ si el pallet no se ha elegido, o bien una tupla $(id_i, (pos_i^x, pos_i^y), d_i)$, si ha sido elegido.

Una solución parcial donde, a partir de un conjunto de tres pallets, se ha decidido colocar el primero en la posición $(1, 2)$ con orientación normal y no colocar tanto el segundo como el tercero sería la siguiente:

$$S = [(1, (1, 2), 0), False, False]$$

4.2.3. Función de evaluación

Se encarga de evaluar como de buena o mala es una solución dada. Esta lo hace atendiendo a la variable a optimizar en el problema de optimización presentado, que en este caso es el volumen, y el criterio de optimización, que es maximizar. De tal forma que el resultado obtenido es la cantidad de volumen cargado. Por ello, dado una solución de entrada esta función devolverá como salida la suma de los volúmenes de aquellos pallets colocados, es decir, los que en la lista solución su contenido no es $None$.

4.2.4. Interfaz gráfica

Para poder evaluar las soluciones obtenidas por cada uno de los algoritmos de forma sencilla se ha implementado un algoritmo que dibuje en 2D una solución pasada por parámetro, usando para ello la librería turtle [documentacion python, 2023].

Dado que esta hace uso pixeles en lugar de porcentajes de pantalla, se han implementado dos funciones. La primera se encarga de obtener el tamaño de la pantalla principal, para que el dibujo realizado tenga un tamaño proporcional a esta.

La segunda obtiene dado un punto (pos_i^x, pos_i^y) sobre el que se sitúa un pallet en el camión, la posición que ocupa en la pantalla. Esto permite pasar de puntos en $([0, L_{cam}^x], [0, L_{cam}^y])$ a puntos en

$$\left(\left[\frac{-resol_x}{2}, \frac{resol_x}{2}\right], \left[\frac{-resol_y}{2}, \frac{resol_y}{2}\right]\right)$$

Donde resol indica la resolución de la pantalla, de tal forma que la representación logre adaptarse a esta.

Haciendo uso de las anteriores, el algoritmo de dibujo es capaz de dibujar el camión y su contenido haciendo uso siempre de $2/3$ partes del ancho de la pantalla, independientemente de la resolución de esta, y por ello comportándose de forma responsiva a la pantalla en cuestión.

Un ejemplo de una representación gráfica de una solución se puede observar en 4.5.

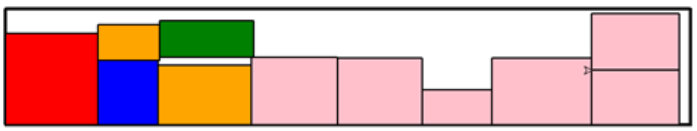


Figura 4.5: Ejemplo de la representación gráfica para una solución con 5 clientes

4.3. Implementación de las heurísticas de colocación, algoritmos voraces y testing de estos

Como primera aproximación al problema, se implementará un algoritmo voraz, que, como se ha indicado anteriormente, destaca por la obtención de un resultado de forma rápida, aunque este no garantiza la obtención de la solución óptima, frente a otros que sí que lo logran, pero su orden de complejidad asciende a uno exponencial.

Como se trata de un algoritmo voraz, se han de definir una serie de elementos y funciones característicos de estos adaptados al problema a tratar. Entre estos se encuentran las soluciones parciales y finales, ya definidas anteriormente, y las funciones de selección, factibilidad y esSolución.

4.3.1. Función de selección

Esta recibe como entrada un estado y devuelve los pallets y posiciones tomados, haciendo uso para ello de dos funciones heurísticas.

En este caso, a diferencia de otros problemas como el de la mochila, explicado en el capítulo 2, la función de selección queda dividida en dos fases. En la primera de ellas se selecciona el ítem a colocar y en la segunda se selecciona la posición de colocación.

4.3.2. Heurística de selección de ítem

Para ello se han llevado a cabo diversas aproximaciones, ya que, como se verá, no existe una heurística trivial que permita obtener el óptimo para el subproblema a tratar sin provocar bloqueos en las siguientes colocaciones.

1. Por altura: se hace uso de la función heurística que obtiene el óptimo local para el subproblema a tratar, por ello ordenándolo por la altura, variable que maximiza el volumen ocupado por superficie de la base. Como cabe intuir, esta se comportará de forma correcta ante instancias donde hay muy pocos clientes o los de las mejores alturas son los de los primeros clientes. En cambio, cuando existen muchos clientes y pocas instancias y son los de estos últimos los de las mejores alturas, se pueden generar bloqueos al incumplir la restricción (4) relativa al orden de los clientes. Un caso de esto ocurre cuando el pallet del último cliente es el de la mejor altura. Como se puede observar en la 4.6, al colocar el pallet 10 ninguno de los otros puede ser colocado, dando lugar a una solución con función de evaluación nefasta.

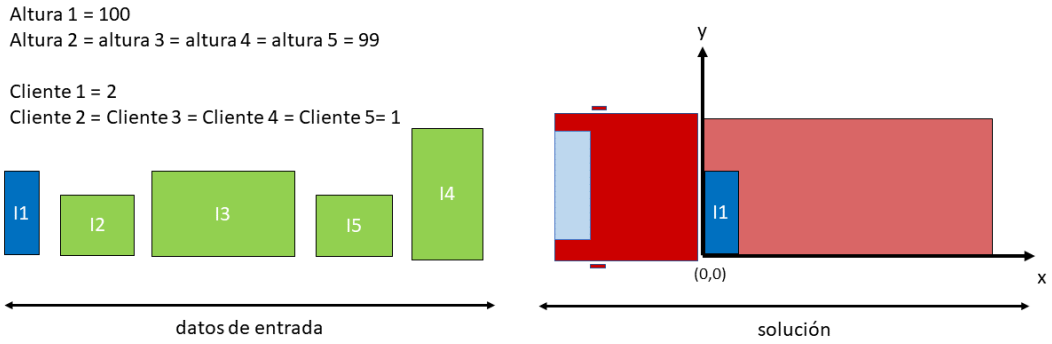


Figura 4.6: Ejemplo de una mala solución obtenida por la función de selección

2. Por cliente: como segunda aproximación, se ordena primero por clientes, desempatando por la altura de estos. Esta favorecería el llenado del camión ante bloqueos por la restricción, pero daría lugar a la toma de items independientemente de su altura, afectando seriamente a la función de evaluación, sumando a que si los mejores ítems pertenecieran a los últimos clientes estos casi nunca serían elegidos.
3. Ordenación mixta: se trata de ordenar dando la misma importancia a la altura como al id del cliente, de tal forma que, tras la normalización de los atributos, la heurística que evalúa el valor de un ítem dado es la siguiente:

$$heuristica = (1 - id\_cliente)/2 + altura/2$$

A pesar de ello, esta continúa siendo sensible a bloqueos producidos por otras variables, como lo es el peso, por lo que cabe destacar que quizá el uso de una estrategia voraz no sea lo más adecuado para el problema a tratar, siendo más convenientes las técnicas metaheurísticas, como se verá a continuación.

En cuanto a la complejidad de esta heurística de selección de pallets, esta podría tener lugar en una única ocasión al inicio del algoritmo voraz, con una complejidad temporal de $O(n \cdot \log(n))$ si se hiciera uso de un algoritmo de ordenación como mergesort.

4.3.3. Heurísticas de colocación de ítem

Para la implementación de las heurísticas de colocación, se ha decidido hacer uso de las tres heurísticas más ampliamente utilizadas en la literatura para resolver el 2D-BPP. Estas son las siguientes:

- First fit: coloca el pallet en la primera posición que encuentra donde cabe de forma completa.
- Best fit: coloca el ítem en la posición donde al colocarse deja menos espacio libre.
- Worst fit: coloca el ítem en aquella posición que dejaría más espacio libre, de tal forma que este hueco pueda ser utilizado por otros ítems.

En cuanto a la complejidad temporal, la implementación de estos tres consiste en primero la construcción de una lista que contenga aquellas áreas donde quepa el ítem y posteriormente la selección del área correspondiente.

Por un lado, la construcción de la lista mediante comprensión de listas da lugar a una complejidad de $O(n)$, donde la lista creada no está ordenada. Tras ello, tiene lugar la selección del área, que para first fit resulta en una complejidad de $O(1)$, al tomar siempre la primera de ellas, mientras que para los dos últimos asciende a $O(n)$. Es por ello por lo que la complejidad de las tres heurísticas asciende a $O(n)$, aunque la constante multiplicadora es superior en los dos últimos.

4.3.4. Función de factibilidad

Esta recibe como entrada una solución parcial y un pallet con su orientación y posición, devolviendo cierto si la inclusión de este último en la solución no infringe ninguna de las restricciones definidas.

En este caso, de entre las restricciones definidas en el capítulo anterior, será necesario comprobar las restricciones (1) relativa a las dimensiones del camión, (4) correspondiente con el orden de clientes y las (5) y (6), correspondientes con las masas cargadas.

Con esto, cabe observar que tanto la restricción (3) relacionada con la restricción de apoyo en frenado, como la restricción (7),responsable de comprobar que los objetos que tienen orientación fija solo pueden colocarse de esta forma, nunca se incumplirán, ya que con el método de puntos estas siempre quedan garantizadas, y por lo tanto no es necesaria su comprobación. Además, al filtrar los puntos en la función de selección por las dimensiones máximas de ancho y largo la restricción de solapamiento (2) no es necesaria comprobarla, al cumplirse siempre.

La función implementada sigue el procedimiento implementado en el pseudocódigo 4.1

```
1: función esFactible(sol)
2:     si cumpleTamCamion(sol) entonces
3:         devolver Falso
4:     si cumplePesos(sol) entonces
5:         devolver Falso
6:     si cumpleClientes(sol) entonces
7:         devolver Falso
8:     devolver Cierto
```

Algoritmo 4.1: Función de factibilidad

Además, para reducir el tiempo de ejecución se ha hecho uso de estructuras de datos que agilicen la comprobación de las restricciones de orden de clientes (4) y de cargas de masa (5) y (6).

Para la primera se hace un uso de un diccionario que almacena el inicio y fin en el eje x de sus pallets, por ello no reduciendo la cota superior de su complejidad temporal, pero sí agilizando la comprobación de esta, que tendrá lugar en $O(n)$, siendo menor en el caso de que la entrada contenga menos clientes que ítems.

En lo relativo a la comprobación de la restricción de a la masa, se almacena la masa total cargada y el centro de masa dentro del bin en cada inserción de un pallet, calculándose de forma dinámica en función del anterior, de tal forma que la comprobación se puede realizar en $O(1)$ en lugar de $O(n)$.

Con los cambios obtenidos se tiene que la función de factibilidad tiene como orden de complejidad temporal $O(n)$.

4.3.5. Función esSolucion

Por último, la función esSolución consistirá en la comprobación de la no existencia de un pallet a explorar dentro de la solución parcial. En este caso se implementa mediante la instrucción de python in, que en la documentación oficial se indica que tiene complejidad $O(n)$ para listas [documentacion python listas, 2023]

Con todo lo anterior, el esquema algorítmico del algoritmo voraz a implementar sería el siguiente:4.2

```
1:  función VORAZ(camion,items)
2:      s ← ∅
3:      items ← ORDENAHEURISTIC(items)
4:      puntos ← [0, 0]
5:      espacios ← GENERAESPACIOS(camion)                    ▷ Para filtrar por tamaño
6:      mientras !ESSOLUCION(s) hacer
7:          item ← SELECCIONITEM(items) puntosPosible ← FILTRAESPACIOS(espacios)
8:          factible ← Falso
9:          mientras !factible ∧ QUEDAN(puntosPosible) hacer
10:             punto ← SELECCION(puntosPosible)
11:             si ESFACTIBLE(s,vapunto) entonces
12:                 factible ← Cierto
13:                 ACTUALIZAEEDD(s,puntos,espacios)
14:         si !factible entonces
15:             INSERTANONE(s)                    ▷ Se indica que el pale explorado no se instera
16:     devolver s
```

Algoritmo 4.2: Función de factibilidad

4.3.6. Testing de las funciones

Una de técnicas que permiten asegurar la correcta implementación de software es el testing. Atendiendo a que se encargan de probar, podemos clasificar un test en unitario, de integración, o funcional [Dto.software, 2021].

Los tests unitarios pueden definirse como aquellas pruebas encargadas de testear el componente de software más pequeño, en este caso una función, para asegurar que cada unidad funciona de acuerdo con su especificación [Dto.software, 2021].

En consecuencia, se ha considerado implementar tests para aquellas funciones de más difícil evaluación, siendo testeadas las funciones que son llamadas desde la función de factibilidad y las auxiliares necesarias para calcular la posición de los puntos, las dimensiones máximas permitidas por punto y la actualización las proyecciones al colocar un ítem.

Por ello, para cada una de las anteriores se implementan entradas que generen condiciones específicas que permitan evaluar la totalidad de los fallos, además de comprobar que la función se comporte de forma correcta ante entradas válidas. Un ejemplo se puede observar en la fragmento de código 4.1:

```
def test_dameAreas():
    camion = (1000,500,500)
    # Caso de prueba 3, donde a partir del punto 2 se genera una
       area que colisiona en x, pero en y no
    # mientras que desde el punto 1 colisiona en y pero en x no
    p1 = (150,10)
    p2 = (100,50)
    proy = [((151,160),(0,40)),((100,125),(60,70))]
    expected1 = (1,490)
    expected2 = (900,10)
    #Se añaden a la lista de entradas(omitido por falta de espacio)

    # Caso de prueba 4, ambas areas generadas colisionan tanto en x
        como en y. En ambos casos colisionan con multiples pales y
       se comprueba
    p1 = (150,10)
    p2 = (100,50)
    proy = [((0,160),(1,10)),((200,260),(1,11)),((260,280),(1,51)),
        ((0,100),(0,60)),((0,151),(100,110)),((100,125),(110,120))]
    expected1 = (50,90)
    expected2 = (160,50)
    #Se añaden a la lista de entradas(omitido por falta de espacio)


    #Por último, se comprueba que cada resultado equivale con el
       esperado
    for i, data in enumerate(zip(punto1,punto2,proyecciones)):
        p1,p2,proy = data
        a1,a2 = dameAreas(p1,p2,proy,camion)
        e1= expected1[i]
        e2= expected2[i]
        if(e1==a1 and e2==a2):
            aciertos +=1
```

Código 4.1: Fragmento del método de pruebas unitarias para el testeo de la función dameÁreas en python

4.4. Implementación de algoritmos heurísticos y metaheurísticos

A lo largo de esta sección se explicaran otros algoritmos implementados que hacen uso de algunas estructuras de datos y funciones anteriores para resolver el problema de optimización a tratar. Entre estos se encuentran tanto un algoritmo de búsqueda local como técnicas metaheurísticas.

4.4.1. Algoritmo de búsqueda local

Una estrategia implementada para resolver este problema de optimización es mediante un algoritmo de búsqueda local, que, como se ha indicado, se basa en explorar las configuraciones vecinas de una dada con el objetivo de moverse en la dirección de aquella con mejore en mayor medida la función de evaluación, resultando en el alcance de la óptima local.

Para su implementación es necesario definir una serie de componentes, como la configuración o el operador de vecindad:

- Configuración: permutación de tamaño n, que indica el orden de inserción de los elementos haciendo uso del método voraz, pero sustituyendo la ordenación realizada por la función de selección. Por ello, se mantiene la parte de la función de selección relativa a la colocación dentro del contenedor, lo que implica que es necesario definir cuál de las heurísticas de colocación es la usada (firts-fit, best-fit o worst-fit). Un ejemplo de una configuración para una entrada de 7 ítems es el de la 4.7, donde se indica que el primero en ser colocado es el ítem de $id = 7$, tras ello, se trata de colocar el de $id = 2$ y así con el resto, tratando de colocar en última instancia el de $id = 4$.

Figura 4.7: Ejemplo de una configuración de 7 items

- Operador de vecindad: dada una configuración este genera un conjunto de configuraciones que difieren de forma mínima de la primera. Han sido dos los operadores de vecindad considerados. El primero de ellos consiste en el intercambio de dos ítems en la permutación, de forma que se obtienen una cantidad de vecinos tal que:

$$C_2^n = \frac{n \cdot (n-1)}{2}$$

  Por otro lado, se ha considerado el operador que devuelva todas las posibles inserciones de cada ítem en cada posición, resultando en $(n-1)^2$ vecinos.

Además, para su implementación se añade un parámetro que indica si es informado, es decir, si su configuración inicial se toma de forma aleatoria o bien mediante una heurística. Además, se incluye el número de reinicios, que por defecto almacena 0 y se encarga de repetir el algoritmo tantas veces como se indique, de cara a tener mayor probabilidad de alcanzar una mejor solución. Cabe destacar que si la configuración inicial se obtiene de forma totalmente informada, este segundo parámetro siempre obtendría la misma solución en cada iteración.

Con lo anterior cabe destacar que la complejidad temporal de este algoritmo ascendería hasta $O(n^4)$ para cualquiera de los operadores elegidos, viéndose incrementada de forma lineal con el aumento de el número de reinicios, cuyo efecto se verá en el capítulo de evaluación de resultados.

4.4.2. Estrategias metaheurísticas

Como se ha observado en la implementación de la función de selección en el apartado de los algoritmos voraces, la gran cantidad de variables que pueden producir bloqueos hacen difícil la obtención de valores aceptables para cualquier entrada. Además, se busca un valor cercano al óptimo global en lugar del local, objetivo no siempre alcanzable mediante una búsqueda local.

Es por ello que se ha decidido implementar algoritmos metaheurísticos para la obtención de buenas soluciones, ya que estas técnicas en el caso de ser bien implementadas son capaces de explorar el espacio de soluciones de forma eficiente.

Para ello se han implementado tanto dos metaheurísticas basados en trayectorias, GRASP y simulated annealing, como una poblacional, el algoritmo genético.

4.4.3. Recocido simulado

Como ya se vio en el apartado anterior, este algoritmo se divide en dos fases, una primera de generación de una solución inicial y otra segunda fase de mejora de esta última. Además, este procedimiento completo puede iterarse.

Para la fase de generación, se ha de definir cómo se crea la configuración inicial, indicando si se hace de forma informada o aleatoria. En el caso de hacerse de forma informada, es necesario definir que heurística de selección de ítems es la usada, e independientemente de si se genera de forma informada o aleatoria es necesario definir la heurística de colocación a implementar por el algoritmo.

Por otro lado, para la fase de enfriamiento es necesario ajustar diversos parámetros, donde se encuentran por un lado la temperatura inicial y α, que se corresponde con el factor de enfriamiento, además de la propia función de enfriamiento que en este caso se toma la siguiente:

$$cooling(T) = \alpha \cdot T$$

Esta decrementa la temperatura de forma lineal. Además es necesario definir el número de iteraciones a ejecutar.

Por último, para la probabilidad de transición se implementará la función de Boltzmann, donde se tiene que:

$$P_{transicion} = e^{(-\Delta J/T))}$$

El siguiente capítulo evaluará varias combinaciones de parámetros para determinar cuál se adapta mejor al problema en cuestión.

4.4.4. GRASP

Como se vio en el capítulo anterior GRASP queda dividido en dos fases, donde en una primera genera una configuración de forma informada mediante una heurística, introduciendo no determinismo para elegir entre los mejores ítems en cada movimiento, y una posterior mejora de la solución obtenida aplicando búsqueda local. Este proceso de igual manera puede repetirse de forma iterada para mejorar la solución. Con ello, los operadores y parámetros que es necesario definir para el problema en cuestión han sido los siguientes:

- Heurística de selección de ítem y de posicionamiento
- Método de selección: es necesario definir el método de selección entre ítems, donde en este caso se implementarán dos, por un lado la selección mediante ruleta, donde influye la función de valor de cada ítem, y por otro lado se implementará la selección de torneo aleatorio, donde será necesario introducir un nuevo parámetro que indique la cantidad de configuraciones que participan, y donde de los n mejores individuos se elige uno al azar.

- Operador de vecindad implementado: se hace uso del intercambio de posición entre dos elementos de una configuración, de la misma forma que se vio en el subapartado de búsqueda local.
- Número de iteraciones a total a ejecutar
- Factor de aleatorización: en la fase de construcción es necesario definir cuantos elementos de la configuración se tienen en cuenta para seleccionar en cada fase del método voraz de construcción, donde si se toma como valor 1 se comporta como un método completamente voraz y no adaptativo, mientras que si se toma n, donde n es el número de ítems, se genera la solución inicial de forma no informada, obteniendo una configuración totalmente aleatoria.

4.4.5. Algoritmo genético

En este caso, los parámetros y operandos a definir son los siguientes para cada una de las fases del algoritmo:

- Generación de la población inicial: se hace uso tanto de un parámetro que indique el tamaño de la población a generar como de otro parámetro que indique si esta generación se realiza de forma aleatoria o bien se realiza de forma parcialmente informada.
- Cruce: en la implementación es necesario definir tanto el operador de cruce utilizado como el parámetro probabilidad de cruce. Asimismo, se han implementado dos operadores distintos, de entre los cuales se elige entre uno de ellos durante la ejecución basándose en el valor de otro parámetro. El primero de ellos, denominado en la literatura 2PCX, obtiene a partir de dos puntos de cruce generados de forma aleatoria dos individuos; en estos individuos, fuera de estos puntos de cruce se retienen los valores de su primer padre, mientras que en el interior de estos se mantienen los mismos valores, pero en el orden del padre dado. Una representación gráfica se observa en la figura 4.8.

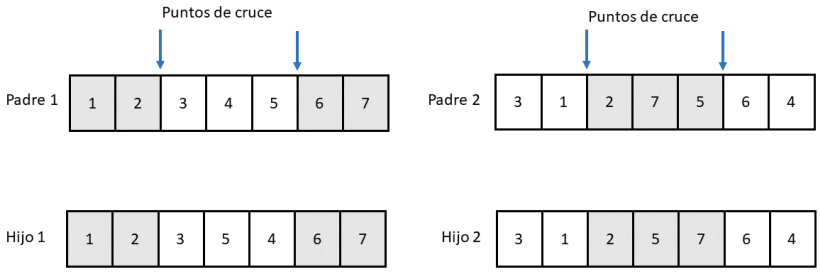


Figura 4.8: Ejemplo de aplicación del operador de cruce 2PCX

El segundo operador de cruce implementado OX1, que como se vio en el capítulo anterior, consiste en la generación de una máscara la cual toma tal cual de un padre los elementos que almacenan un 1 y el resto de elementos a colocar se sitúan en las posiciones de la máscara con 0 en el orden de los del segundo padre. Un ejemplo de este se puede observar en 4.9.

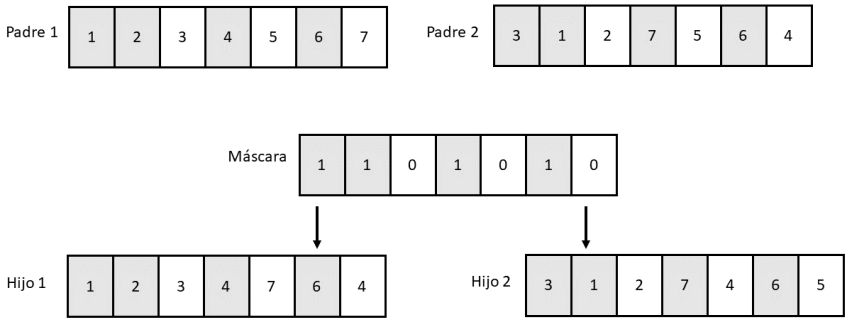


Figura 4.9: Ejemplo de aplicación del operador de cruce OX1

- Mutación: es necesario definir tanto el parámetro de probabilidad de mutación como el operador de mutación. Para el operador de mutación, se ha implementado un método único, que consiste en el intercambio de dos elementos en la configuración con la probabilidad dada.
- Selección de la población : tiene como objetivo reducir las configuraciones almacenadas a un total de n. Para su implementación, se han llevado a cabo dos operadores de selección distintos. El primero de ellos consiste en la selección mediante ruleta, donde la probabilidad de selección viene en función de su función de valor, siendo esta la de la siguiente expresión:

$$P_{seleccion}(i) = \frac{valor\,(i)}{\sum_{c \in Pob} valor\,(c)}$$

El segundo método implementado consiste en la selección mediante torneo, que requiere un parámetro adicional donde indique la cantidad de configuraciones que participan en cada combate, y donde se mantiene al mejor individuo de los seleccionados.

- Criterio de parada: en este caso se han implementado dos distintos. El primero consiste en la no mejora de la población en las últimas n iteraciones, donde este valor n se pasa por parámetro. El segundo consiste en la ejecución de un número de iteraciones fijas, donde, de nuevo, este se pasa por parámetro.

Como se comporta el algoritmo para este problema atendiendo a los distintos parámetros y operadores usados se verá detalladamente en el siguiente apartado.

4.5. Código

Se puede acceder al código en el siguiente repositorio público de github, programado en una libreta de jupyter: `https://github.com/ruben2567/TFG`

5. Evaluación de resultados

Este apartado queda dividido en dos partes, en la primera se estudia que ajuste de parámetros se comporta mejor para cada uno de los algoritmos atendiendo al tamaño y número de clientes del problema en cuestión, con el objetivo de obtener la combinación de los que responde mejor a cada situación. En la segunda parte se hace una comparativa global entre las distintas técnicas heurísticas y metaheurísticas utilizadas, haciendo uso para cada una de la mejor combinación de parámetros encontrada, para así poder comparar los resultados producidos para la misma entrada de problemas.

5.1. Evaluación de los parámetros de cada uno de los algoritmos

En primera instancia se evalúa el algoritmo voraz, tras este el de búsqueda local y, por último, cada uno de las técnicas metaheurísticas implementadas.

Para lograr una evaluación reproducible y evitar que los resultados dependan de la aleatoriedad del conjunto de entrada, para cada una de las situaciones a evaluar se ha hecho uso de un conjunto de semillas fijo para generar los pallets de entrada, calculando la media entre los resultados obtenidos por las semillas para cada instancia de prueba.

5.1.1. Algoritmo voraz

Para el algoritmo voraz en primer lugar se ha buscado cuáles son las heurísticas de colocación y de ordenación que se comportan de mejor forma ante una entrada según su tamaño y número de clientes. Para ello, se ha evaluado este algoritmo tomando como tamaños de entrada 20, 40, 60, 80, 100 y 150 pallets y como número de clientes 1, 2, 5, 10 y 20 respectivamente. Los resultados obtenidos han sido los de la figura 5.1.

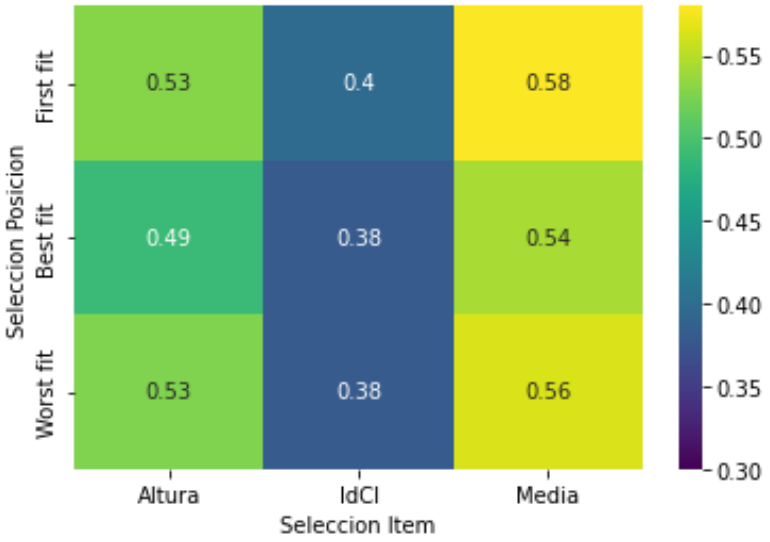


Figura 5.1: Función de evaluación por heurísticas elegidas

Esto muestra que la heurística de selección de ítem que mejor se comporta es la que otorga la misma importancia a la altura y al orden del cliente normalizados, frente a la que ordena por altura y la que ordena por el id del cliente, siendo esta última la que obtiene peores resultados.

En cuanto a la heurística de selección del punto de colocación, los resultados no muestran una diferencia importante, a pesar de que en principio se pudiera pensar que best fit se pudiera comportar de mejor manera que worst fit, o viceversa. Investigando como resuelven este algoritmo instancias con cada heurística de colocación, para ello depurando el código, se ha podido determinar que esto sucede porque las dimensiones del camión son mayores en el eje x que en el y, por lo que al colocar un ítem, worst fit tiende a elegir aquellas posiciones con comienzo mayor en el eje x, provocando que las posiciones anteriores a esta no queden sometidas a bloqueos por el id del cliente. En cambio, best fit tiende a elegir estas últimas, resultando en que si el id cliente es alto se bloquearán los valores de x posteriores debido a la restricción de orden de entrega, al mismo tiempo haciendo un mejor aprovechamiento de las áreas. Esto hace que, de forma conjunta, los resultados acaben siendo prácticamente los mismos.

La figura 5.2 muestra un ejemplo de este comportamiento de las heurísiticas, donde worst fit tiende a colocar pallets sobre el punto amarillo, ocupando área 2 y haciendo un peor uso del espacio, pero al mismo tiempo produciendo bloqueos por el id de cliente en una menor cantidad de posiciones. En cambio, si se hace uso de best fit el pallet se colocaría sobre el área 1, aprovechando mejor el espacio, pero generando bloqueos más restrictivos por la restricción del orden de entrega, que aparecería desde el principio del contenedor.

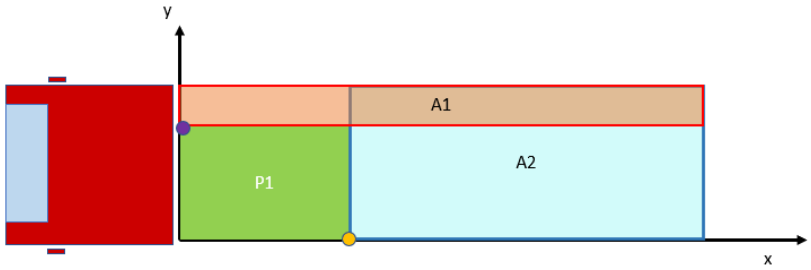


Figura 5.2: Bloqueos por las heurísticas de selección

Los tiempos de ejecución medios tras ejecutar 100 semillas para cada uno de los tamaños de entrada según el tipo de heurística se muestran en la figura 5.3, donde es importante destacar que los valores son bajos para cualquier tamaño de entrada, en especial en comparación con otros algoritmos implementados, como se verá posteriormente en este mismo capítulo.

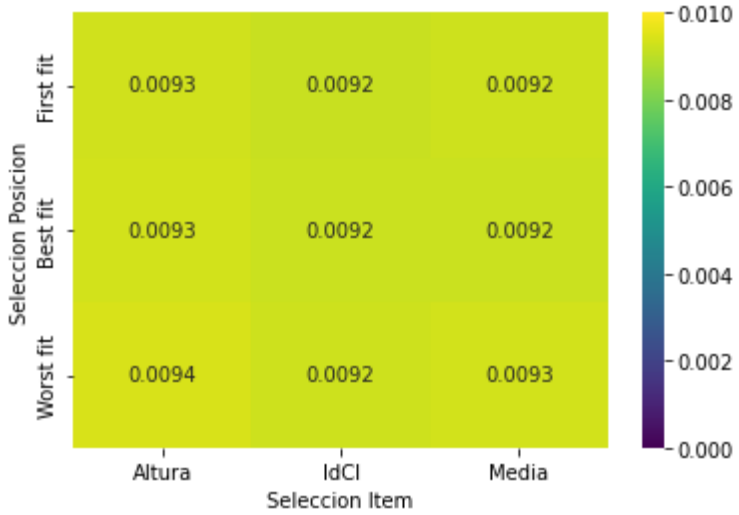


Figura 5.3: Tiempos de ejecución del algoritmo voraz

A pesar de que en el capítulo anterior se pudiera pensar que existen diferencias en tiempo de ejecución de first fit frente a worst y best fit por la ordenación realizada de estos últimos, estos en la práctica no muestran esta diferencia.

Esto puede ser debido a la gran optimización de la ordenación de la lista de candidatos mediante programación funcional con funciones como sorted y la construcción de listas mediante comprensión de listas para realizar la ordenación, que reducen el tiempo de ejecución de forma muy destacable, siendo otras fases del algoritmo mucho más costosas y por ello provocando que el aumento de tiempo no sea apreciable.

En cuanto a las heurísticas de selección de ítems, como era de esperar, no dan lugar a diferencias temporales aparentes.

En este algoritmo, cabe destacar que la complejidad temporal no crece de forma exponencial frente a tamaños de entrada mayores, sino que lo hace de forma polinómica. Esto se puede observar en la figura 5.4.

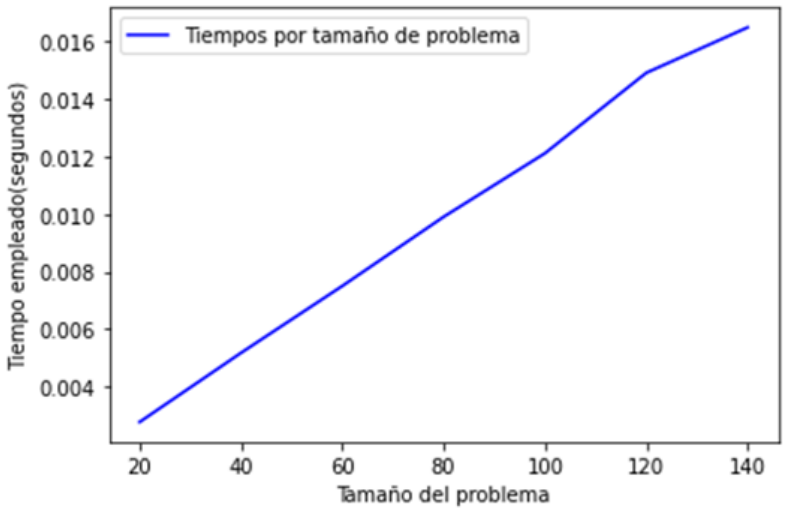


Figura 5.4: Tiempos de ejecución en función del tamaño de problema

En lo que concierne al funcionamiento en general del algoritmo voraz destaca que la solución proporcionada, si bien es obtenida de forma rápida, no se acerca lo suficiente al óptimo como otras estrategias que se verán a continuación. Este toma decisiones en cada etapa en base al subóptimo local para construir la solución, sin tener en cuenta las consecuencias futuras que puede conllevar la colocación del pallet, lo que resulta en que no termina de aprovechar lo suficientemente bien los espacios dentro del contenedor, dejando generalmente grandes huecos. Esto se puede observar en la figura 5.5.

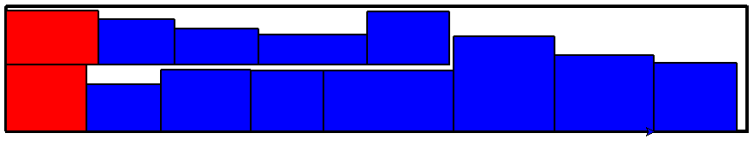


Figura 5.5: Problema del voraz: deja muchos espacios libres

5.1.2. Algoritmo de búsqueda local

En esta caso se han considerado un total de dos casos de prueba, expuestos a continuación.

En primer lugar, se evalúa como mejora la función de evaluación y cómo aumenta el coste temporal en función del número de reinicios, usando como configuración de inicio una no informada, todo ello teniendo en cuenta el tamaño del problema de entrada.

Para ello, se evalúa como se comporta usando un total de 1, 2,4 y 6 iteraciones por ejecución respectivamente. Los resultados de la función de evaluación para tamaños de problema de 20, 40, 60 y 80 pallets pueden observarse en la figura 5.6, donde cabe destacar que las mejoras producidas son algo limitadas ante el aumento del número de iteraciones.

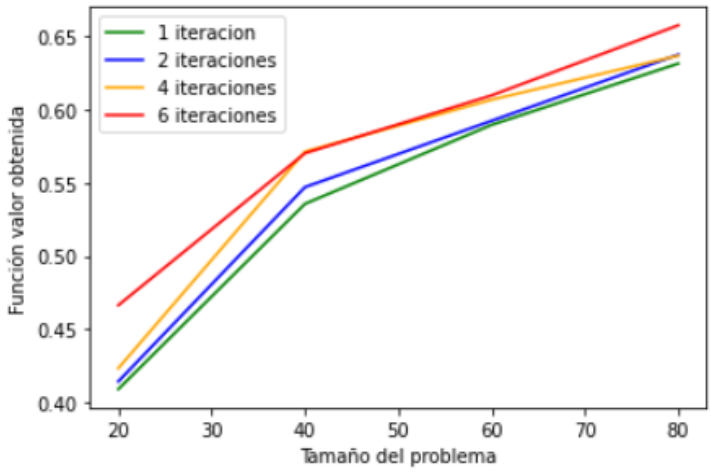


Figura 5.6: Función de evaluación por número de iteraciones

En cuanto a la diferencia de tiempos, como se observa en la figura 5.7 esta crece de forma importante atendiendo al número de iteraciones realizadas, resultando en costes demasiado elevados para más de dos iteraciones si el tamaño de entrada es grande.

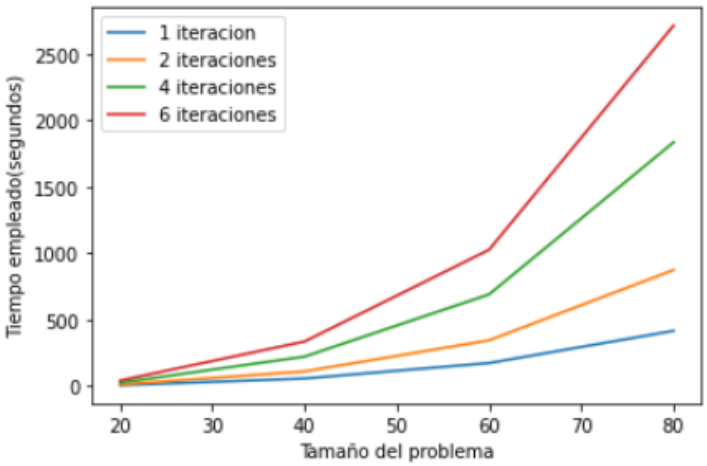


Figura 5.7: Coste temporal por número de iteraciones

Por otro lado, se estudia la diferencia entre la ejecución con una configuración de inicio informada frente a una no informada, obtenida de forma aleatoria. Para obtener la configuración informada se ha hecho uso de la heurística de selección de ítems que otorga la misma importancia al orden de clientes y a la altura, y la heurística de colocación first fit, ya que la combinación de estas dos es la que ha ofrecido mejores resultados para el algoritmo voraz, como se vio en el subapartado anterior.

Los resultados se pueden observar en la figura 5.8, donde destaca que la solución obtenida es mejor usando como configuración inicial una informada, por lo que el uso de una buena heurística permite obtener, de forma general, mejores resultados al algoritmo local.

En general, aunque este algoritmo genere resultados aceptables, este se queda estancado en el óptimo local, y tiende a dejar huecos dentro del contenedor, aunque no tanto como sucedía en el voraz. Por ello, es necesario hacer uso de técnicas metahuerísticas para realizar una mejor exploración del espacio de soluciones, para así de aproximarse en mayor medida al óptimo global.

Asimismo, a nivel temporal es muy costoso cuando el tamaño del problema es grande, siendo demasiado lento a partir de una entrada de más de 80 pallets, debido a la gran combinación de vecinos a evaluar por iteración.

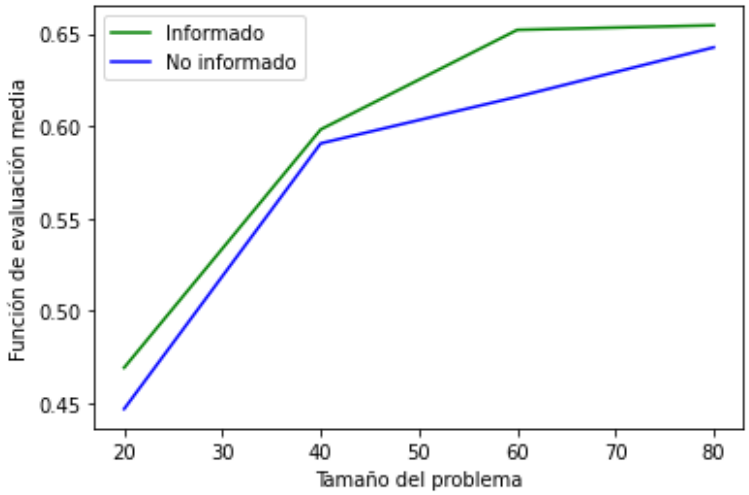


Figura 5.8: Evolución de la función de evaluación en búsqueda local

5.1.3. GRASP

Para evaluar cuál es el subconjunto de parámetros que mejor se comporta se ha decidido ajustar en primera instancia el valor del parámetro randomFactor, que indica entre cuantos pallets se elige en cada etapa de la fase constructiva voraz. Para ello, se prueba con los valores 1, 2, 3, 5 y 30, fijando el tamaño del problema a 80. En cuanto al número de iteraciones, este se fija a 3. Los resultados se muestran en la figura 5.9.

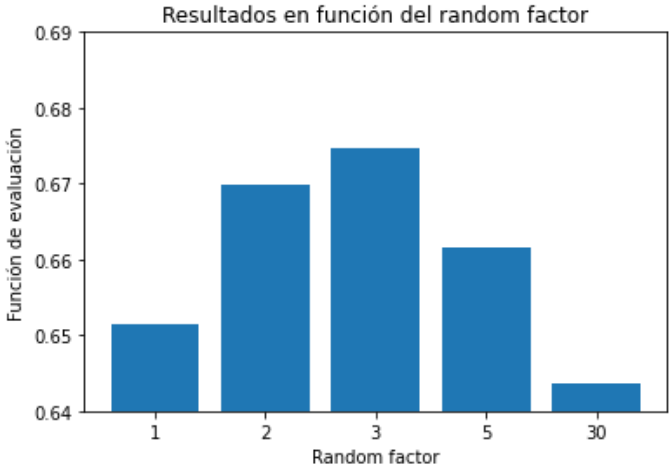


Figura 5.9: Evolución de la función de evaluación atendiendo al randomFactor

Respecto a estos, cabe destacar que los valores que mejor se comportan para el problema en cuestión son 2 y 3, es decir, aquellos que en la fase constructiva eligen entre los 2 y 3 mejores pallets disponibles. Los resultados son mejores que para configuración completamente informada, obtenida con randomFactor 1, que se obtiene de forma completamente determinista y, por lo tanto, la solución obtenida en cada una de las iteraciones es la misma. Del mismo modo, estos valores se comportan mejor que 30, donde se agrega una mayor cantidad de no determinismo en cada selección, haciendo que el algoritmo se comporte de una forma parecida a la del algoritmo local no informado.

En cuanto al coste temporal, este es equivalente al de la búsqueda local, ya que la construcción voraz inicial adaptativa tiene un coste temporal insignificante, parecido al del algoritmo voraz, al lado de la fase de mejora de la solución, equivalente al de la búsqueda local. Por ello, en lo que concierne al coste temporal global, este continua siendo alto, especialmente para tamaños de problema grandes.

Por último, con relación al valor de la función de evaluación de la solución obtenida, esta posee un valor bueno, donde si se observan algunas de las soluciones obtenidas usando $randomFactor = 2$, como la de la figura 5.10, se puede determinar que se hace un buen aprovechamiento de la superficie de apoyo.

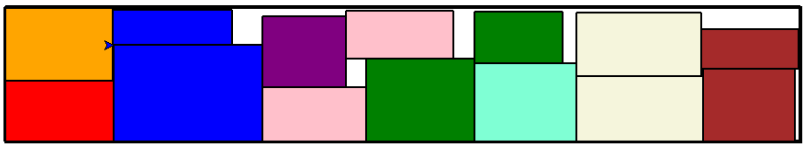


Figura 5.10: Ejemplo de llenado obtenido por GRASP

5.1.4. Recocido simulado

Para optimizar el valor de los parámetros que necesita este algoritmo, este estudio queda dividido en dos. En primer lugar se tratará de identificar cuáles son los valores de temperatura inicial y factor de enfriado que obtienen mejores resultados, y, posteriormente, se evaluará la forma como influye obtener la configuración inicial de forma informada.

Para agilizar la búsqueda, en primer lugar se prueba con una serie de valores para observar como se comportan. Al realizar esta práctica, se puede determinar que los valores de temperatura deben de ser bajos, ya que la función de evaluación esta definida en el intervalo $[0, 1]$. Asimismo, el cooling rate se comporta mejor con valores altos, donde mantiene la temperatura a lo largo de las iteraciones.

Por ejemplo, haciendo uso de valores como $t_0 = 0.2$ se tiene que, con $\Delta J = 0.1$ la tasa de aceptación es de 0.6%, resultando en una probabilidad de transicion baja al ser una configuración que empeora de forma importante la actual, mientras que para configuraciones que no la empeoran de forma importante, como con $\Delta J = 0.01$, donde de 60%, la probabilidad de aceptación es muy superior, lo que permite explotar el espacio de búsqueda en fases iniciales. En las fases finales, con la reducción de la temperatura,lo más probable es solo aceptar soluciones que mejoren la configuración o la empeoren de forma mínima.

Con lo anterior, se realiza una búsqueda en rejilla, que consiste en ejecución , para así obtener la constante de enfriado y la temperatura inicial que, de forma simultanea, se comportan de mejor manera. Los resultados se muestran en la figura 5.11.

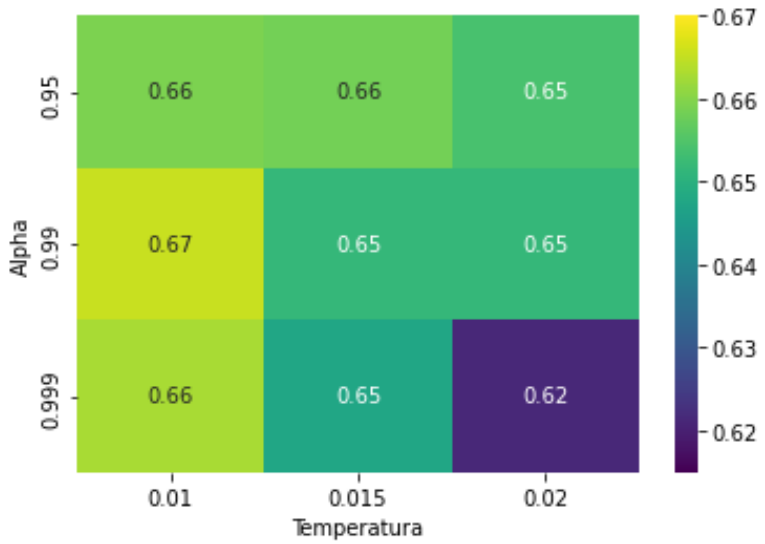


Figura 5.11: Función de evaluación en SA en función de alpha y t_0

En segundo lugar se estudia como influye el parámetro randomStart, de igual forma que se hizo para el algoritmo de búsqueda local, de nuevo usando la mismas heurísticas para la configuración obtenida de forma informada. Los resultados son los siguientes, representados en la figura 5.12.

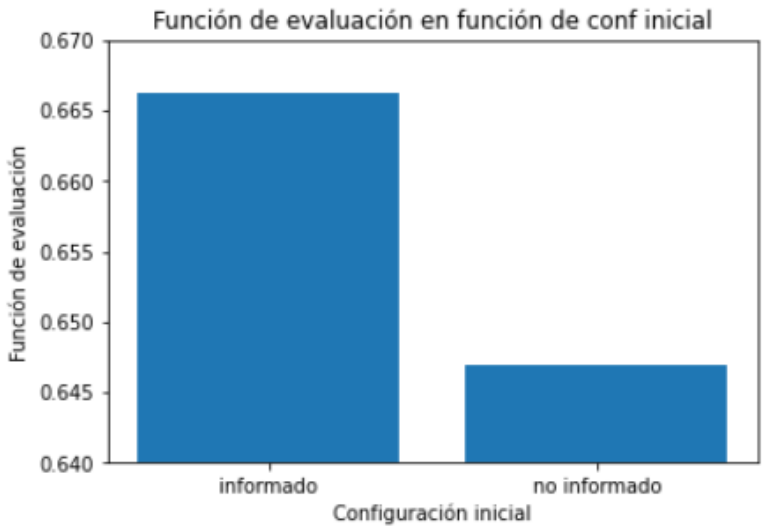


Figura 5.12: Función de evaluación en SA en función de la configuración de inicio

En estos se puede observar que el hecho de comenzar con la configuración obtenida por una buena heurística, de nuevo, hace que los resultados sean algo mejores.

Por último, cabe destacar que, a nivel global, con la combinación parámetros adecuada, el algoritmo se comporta de forma excelente para este problema, generando soluciones con funciones de evaluación altas y con un aprovechamiento de la superficie del camión bueno. Un ejemplo se muestra en la figura 5.13.

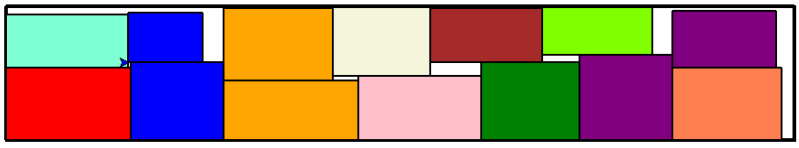


Figura 5.13: Llenado de un camión por el algoritmo SA

5.1.5. Algoritmo genético

Para estudiar el comportamiento de la implementación de este algoritmo, en primer lugar se estudiará cuáles son las portabilidades de mutación y cruce que lo optimizan los resultados. Tras esto, el estudio se centrará identificar el mejor operador de cruce y en evaluar el parámetro tamaño de población y su relación tanto con la función de evaluación como con el tiempo de ejecución.

Antes de realizar la búsqueda en red se han realizando pruebas de forma manual y se ha observado que, para obtener buenos resultados, es necesario hacer uso tanto de una probabilidad de mutación alta como de una probabilidad de cruce baja. Teniendo esto en cuenta se realiza una búsqueda en red, para ello probando con 0.95, 0.965 y 0.98 para la probabilidad de mutación y 0.1, 0.15 y 0.2 para la de cruce. Los resultados se muestran en la figura 5.14, que muestran que la mejor combinación es la obtenida con una probabilidad de mutación de 0.965 y una de cruce de 0.15.

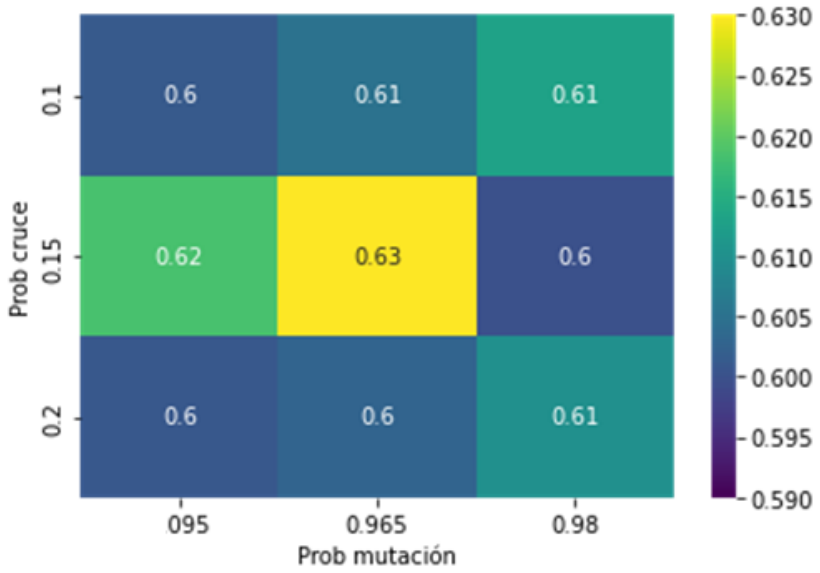


Figura 5.14: Función de evaluación atendiendo a pMut y pCruce

Tras esto se compara el rendimiento de los dos operadores de cruce definidos, cuya función de evaluación media se puede observar en la figura 5.15, y muestra que ambos se comportan de forma parecida, siendo OX1 ligeramente superior, aunque la diferencia es pequeña ya que se ve atenuada por la escasa probabilidad de cruce utilizada, al usar los parámetros que mejor se comportaban en el estudio anterior.

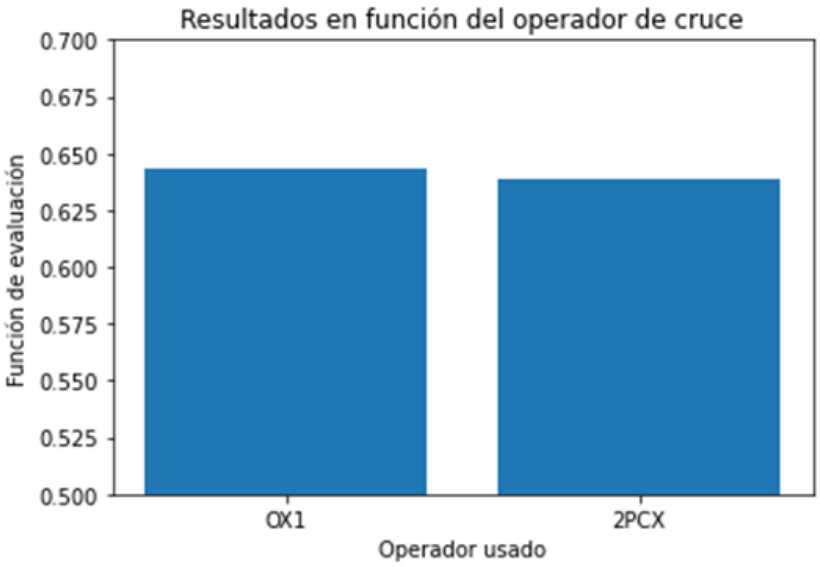
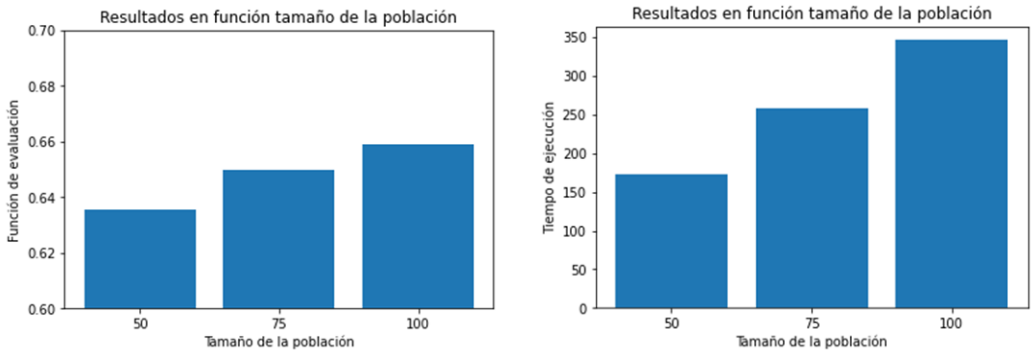


Figura 5.15: Función de evaluación por operador de cruce

En cuanto a los tamaños de la población y su impacto en el resultado, en la figura 5.16a se muestra la evolución la función de evaluación y en la figura 5.16b la diferencia temporal. En ellas destaca que, tanto la función de valor como el tiempo de ejecución, crecen de forma destacable con el aumento del tamaño del problema, por lo que este parámetro deberá ajustarse atendiendo al tiempo disponible para la ejecución.



(a) Función de evaluación **(b)** Tiempo

Figura 5.16: Resultados obtenidos por tamaño de población

En general, cabe destacar que este algoritmo no logra alcanzar los resultados obtenidos por las técnicas metaheurísticas basadas en trayectorias vistas anteriormente. Es preciso señalar que, aun siendo los resultados de la función de evaluación inferiores, su tiempo de ejecución no es demasiado lento ante cualquier tamaño de entrada, ya que este depende en mayor medida del tamaño de población y del criterio de parada.

5.2. Comparativas entre algoritmos

Por último, se realiza un estudio entre algoritmos usando para ello la combinación de parámetros que mejor se ha comportado para cada uno de ellos. En este caso, la comparativa se realiza entre los siguientes:

- Algoritmo voraz: hace uso de la heurística best fit para la selección de la posición y para la selección de ítem la que le otorga la misma importancia al cliente y la altura.
- Algoritmo de búsqueda local: se ejecuta una única iteración y la configuración inicial se obtiene de forma informada, usando las mismas heurísticas de selección y colocación que en el algoritmo voraz.
- GRASP: se hace uso randomFactor y número de reinicios igual a 3. En cuanto a las heurísticas, de nuevo se hacen uso las del voraz.
- Recocido simulado: se toma como temperatura inicial 0.01 y como constante de enfriado 0.99. En lo relativo a la formación de la configuración inicial, de nuevo, se hace de forma informada.
- Algoritmo genético: se usa como probabilidades de mutación y de cruce 0.965 y 0.15 respectivamente. Como operador de cruce se hace uso de OX1 y se toma un tamaño de población de 100.

Los resultados de la función de evaluación media a cualquier tamaño de entrada se muestran en la figura 5.17.

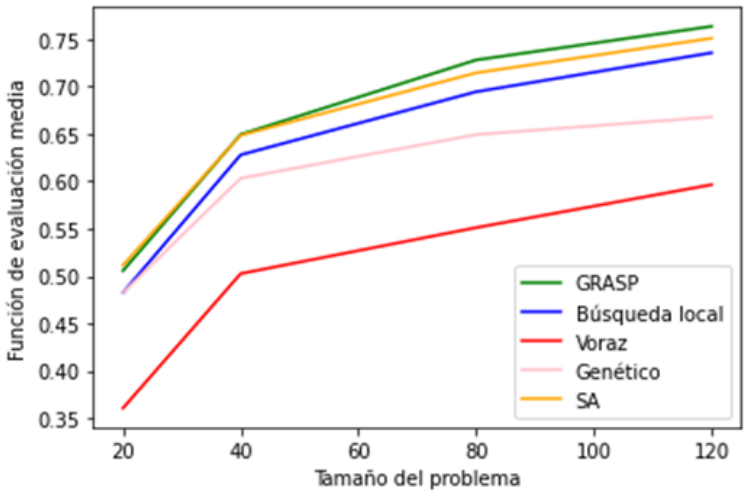


Figura 5.17: Comparativa de función de evaluación entre algoritmos

En esta se puede observar que el algoritmo que se comporta de mejor forma es el SA o recocido simulado, seguido por GRASP.

En cuanto a los tiempos de ejecución, los valores medios atendiendo al algoritmo y tamaño de problema se muestran en la figura 5.18.

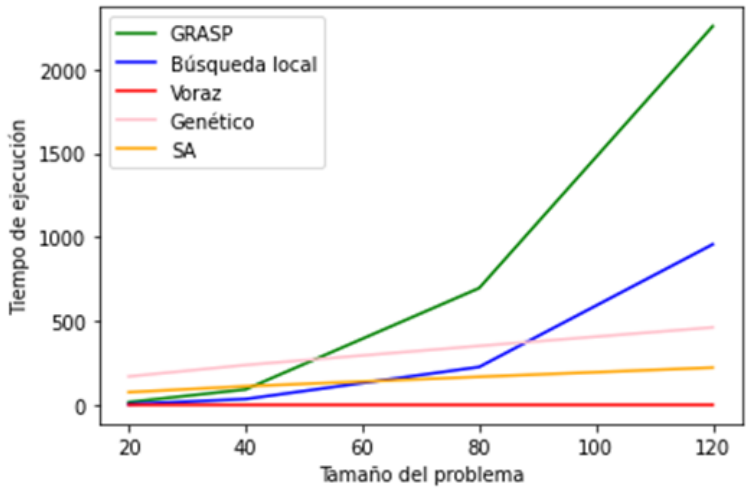


Figura 5.18: Comparativa de tiempos entre los distintos algoritmos

En esta, cabe destacar que el de mayor rapidez es el voraz. Para tamaños de problema pequeños todos tienen un tiempo de ejecución aceptable. En cambio, para tanto GRASP como el algoritmo de búsqueda local, con el crecimiento del tamaño del problema los tiempos de ejecución aumentan de forma importante, debido a la cantidad de configuraciones generadas por el operador de vecindad. Para el algoritmo voraz, el genético y recorrido simulado el tiempo no se dispara con el crecimiento del tamaño del problema, a diferencia de los dos anteriores.

Resumiendo, en este caso para el problema a tratar el algoritmo voraz obtiene soluciones en tiempo rápido pero lejanas al óptimo. El de búsqueda local, más lento por el coste de evaluar una gran cantidad de vecinos, genera resultados buenos, pero queda estancado en óptimos locales en lugar de buscar el óptimo global.

Es por ello que las técnicas metaheurísticas proporcionan un enfoque ideal para aproximarse a este óptimo global en tiempo mucho menor que el que tomaría un algoritmo exacto. En este caso para el problema en cuestión se comportan de muy buena manera, en especial las basadas en trayectorias. Como se ha visto, tanto SA como GRASP obtienen muy buenos resultados al hacer un buen balance entre explotación y exploración del espacio de búsqueda, siendo GRASP más lenta que SA cuando el tamaño del problema crece.

En este caso, cabe destacar que los metaheurísticos basados en poblaciones como el algoritmo genético no presentan resultados tan buenos como los basados en trayectorias, quizá por los operadores de cruce diseñados y por la forma del espacio de búsqueda.

6. Conclusiones

En este último capítulo se abordarán tanto las conclusiones del trabajo realizado como las propuestas para trabajo futuro y las competencias de la intensificación abordadas.

6.1. Conclusiones del trabajo realizado

A lo largo de este trabajo se ha abordado un problema real, como es la carga de camiones, con una gran cantidad de variables y restricciones presentes en el propio problema al aplicarse en la vida cotidiana.

Es importante destacar que los enfoques metaheurísticos implementados producen resultados de gran calidad, también ayudándose en este caso de heurísticas como la de posicionamiento, dependientes del problema, obteniendo resultados mejores que las alcanzables por un humano y siempre respetando cada una de las restricciones relativas al problema, pudiendo resolver una instancia de este en segundos, y por ello resultando en una forma eficiente de automatización.

Se puede determinar que una aplicación de los algoritmos implementados mejoraría el proceso manual para instancias propias de la vida real, ya que permitiría tanto reducir costes del transporte al reducir el número de camiones a cargar, como agilizar las entregas de los pedidos, al cargar un mayor volumen de media por camión, por lo que sería interesante su aplicación en empresas que trabajen con procesos reales de logística y transporte para el llenado de camiones.

Por último, cabe destacar que, tomando como referencia los resultados obtenidos en el estudio del capítulo anterior, la técnica metaheurística que se ha logrado adaptar de mejor manera al algoritmo es la del recocido simulado (SA), obteniendo resultados muy por encima de procedimientos voraces, o búsquedas locales, acercándose al óptimo, y a la vez produciendo resultados en una cota temporal aceptable, todo ello gracias al correcto ajuste de sus parámetros para realizar un balance entre exploración y explotación para el problema a tratar.

6.2. Trabajo futuro

Son varias las ideas que han surgido a lo largo del proyecto que podrían mejorar el trabajo realizado para la resolución del problema.

En primera instancia, sería interesante desarrollar una aplicación funcional que permita calcular las mejores disposiciones de ítems de una entrada data, en lugar de hacer uso de un cuaderno de Jupyter, no usable ni intuitiva para un usuario final.

Del mismo modo, habría sido interesante profundizar en el estudio de las heurísticas de selección y colocación, ya que las implementadas no terminan de amoldarse al problema en cuestión por la gran cantidad de restricciones. Una idea que no se ha incluido finalmente en la memoria para ello es la búsqueda local de la heurística de selección, ponderando por cada uno de los atributos una vez normalizados.

Además, por falta de espacio habría sido interesante poder implementar algún metaheurístico poblacional adicional, tal como el particle swarm optimization PSO o el ant colony optimization (ACO) y un algoritmo memético para así poder comparar de forma más minuciosa entre algoritmos.

Por último, otra de las ideas que no ha sido implementada, es la agilización de la búsqueda entre vecinos, que realentiza las técnicas metaheurísticas que hacen uso de una búsqueda local, y donde en otros problemas como en el del viajante de comercio se hacen uso de técnicas de clustering para evitar explorar vecinos parecidos que llevarían a la misma solución, así reduciendo su complejidad temporal.

6.3. Competencias adquiridas

[CM3] Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.

Para la implementación de una solución para este problema, que contiene una gran cantidad de restricciones, ha sido necesario determinar qué técnicas y estructuras de datos permitan obtener de la forma más eficiente posible la solución esperada. Además, he tenido que demostrar la razón por la cual los algoritmos exactos no son adecuados para este problema en cuestión, para el cual las técnicas heurísticas y metaheurísticas se comportan de forma más eficiente.

[CM4] Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

Para la resolución del problema se han hecho uso de una amplia gama de técnicas propias de los sistemas inteligentes, haciendo uso tanto de técnicas heurísticas como metaheurísticas que permitan obtener soluciones de calidad en tiempo eficiente.

[CM5] Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación, percepción y actuación en ambientes o entornos inteligentes.

Esta competencia ha sido adquirida al tratar con un problema tan complejo debido a las restricciones asociadas, donde en la literatura apenas hay presentes resoluciones a este donde se tengan en cuenta la totalidad de restricciones presentes, por lo que ha sido imprescindible razonar metodologías y técnicas, como la de las esquinas, que permitieran presentar el problema en cuestión de forma computable para poder llevar a cabo su resolución.

Referencia bibliográfica

[Ali et al., 2022] Ali, S., Ramos, A., Carravilla, M., and Oliveira, J. (2022). On-line three-dimensional packing problems: A review of off-line and on-line solution approaches. *Computers & Industrial Engineering*, 168:108122.

[Alonso Martínez et al., 2016] Alonso Martínez, M. T., Alvarez-Valdes, R., Iori, M., Parreño, F., and Tamarit, J. (2016). Mathematical models for multi container loading problems. *Omega*, 66.

[Alonso Martínez et al., 2020] Alonso Martínez, M. T., Alvarez-Valdes, R., and Parreño, F. (2020). A grasp algorithm for multi container loading problems with practical constraints. *4OR*, 18.

[Bertsimas and Tsitsiklis, 1993] Bertsimas, D. and Tsitsiklis, J. (1993). Simulated Annealing. *Statistical Science*, 8(1):10 – 15.

[Bortfeldt et al., 2003] Bortfeldt, A., Gehring, H., and Mack, D. (2003). A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29:641–662.

[Ceschia and Schaerf, 2010] Ceschia, S. and Schaerf, A. (2010). Local search for a multi-drop multi-container loading problem. *Journal of Heuristics*, 19.

[CETM, 2023] CETM (2023). Evolución del precio del gasóleo. Disponible en `https://www.cetm.es/evolucion-precios-gasoleo/`.

[Chahar et al., 2021] Chahar, V., Katoch, S., and Chauhan, S. (2021). A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, 80.

[Correcher et al., 2017] Correcher, J., Alonso Martínez, M. T., Parreño, F., and Alvarez-Valdes, R. (2017). Solving a large multicontainer loading problem in the car manufacturing industry. *Computers & Operations Research*, 82.

[Darmann and Döcker, 2019] Darmann, A. and Döcker, J. (2019). On simplified np-complete variants of not-all-equal 3-sat and 3-sat.

[de la Ossa, 2022] de la Ossa, L. (2022). Diseño de algoritmos tema 5.

[Divakaran, 2019] Divakaran, S. (2019). Fast approximation schemes for bin packing.

[documentacion python, 2023] documentacion python (2023). Disponible en https://docs.python.org/3/library/turtle.html.

[documentacion python listas, 2023] documentacion python listas (2023). Disponible en https://www.ics.uci.edu/~pattis/ICS-33/lectures/complexitypython.txt.

[Dto.software, 2021] Dto.software (2021). Apuntes ingeniería del software 2 año 2021 tema 3 pruebas del software, uclm.

[Europeo, 2018] Europeo, P. (2018). Cambio climático en europa: hechos y cifras. Disponible en https://www.europarl.europa.eu/news/es/headlines/society/20180703STO07123/cambio-climatico-en-europa-hechos-y-cifras.

[Fanslau and Bortfeldt, 2010] Fanslau, T. and Bortfeldt, A. (2010). A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, 22:222–235.

[Feo and Resende, 1995] Feo, T. and Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

[Fernández et al., 2010] Fernández, A., Gil, C., Márquez, A., Baños, R., Montoya, M., and Alcayde, A. (2010). A new memetic algorithm for the two-dimensional bin-packing problem with rotations. volume 79, pages 541–548.

[Flores María Julia, 2021a] Flores María Julia, Gámez José Antonio, G. I. (2021a). Tema 6 sistemas inteligentes.

[Flores María Julia, 2021b] Flores María Julia, Gámez José Antonio, G. I. (2021b). Tema 8 sistemas inteligentes: algoritmos genéticos.

[Gonçalves and Resende, 2012] Gonçalves, J. and Resende, M. (2012). A parallel multipopulation biased random-key genetic algorithm for a container loading problem. *Computers & OR*, 39:179–190.

[He et al., 2018] He, T., Wang, H., and Yoon, S. W. (2018). Comparison of four population-based meta-heuristic algorithms on pick-and-place optimization. *Procedia Manufacturing*, 17:944–951.

[Hilbig, 2012] Hilbig, B. (2012). Heuristics: The foundations of adaptive behavior. gerd gigerenzer, ralph hertwig, thorsten pachur (eds.). oxford university press (2011). 872 pp., hardcover, € 73.60, isbn: 978-0199744282. *Journal of Economic Psychology*, 33:223–225.

[Hopper and Turton, 2001] Hopper, E. and Turton, B. (2001). An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128:34–57.

[IPCC, 2023] IPCC (2023). Comunicado de prensa del ipcc: "la acción climática urgente puede garantizar un futuro habitable para todos". Disponible en `https://www.ipcc.ch/site/assets/uploads/2023/03/IPCC_AR6_SYR_PressRelease_es.pdf`.

[Jansen and Solis-Oba, 2006] Jansen, K. and Solis-Oba, R. (2006). An asymptotic approximation algorithm for 3d-strip packing. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 143–152.

[Kirke et al., 2013] Kirke, T., While, L., and Kendall, G. (2013). Multi-drop container loading using a multi-objective evolutionary algorithm. *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, pages 165–172.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

[laSexta, 2022] laSexta (2022). España, país de camiones sin camioneros: radiografía de un sector que vuelve a los paros. Disponible en `https://www.lasexta.com/programas/al-rojo-vivo/espana-pais-camiones-camioneros-radiografia-sector-que-vuelve-paros_20221108636a43a377e04b0001f3042f.html`.

[Mack et al., 2004] Mack, D., Bortfeldt, A., and Gehring, H. (2004). A parallel hybrid local search algorithm for the container loading problem. *International Transactions in Operational Research*, 11:511 – 533.

[Mann, 2017] Mann, Z. (2017). The top eight misconceptions about np-hardness. *Computer*, 50:72–79.

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

[Mellon, 2020] Mellon, C. (2020). Lecture 19: Np-completeness i. Disponible en `https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1103.pdf`.

[Millán Páramo et al., 2014] Millán Páramo, C., Begambre Carrillo, O., and Millán Romero, E. (2014). Propuesta y validación de un algoritmo simulated annealing modificado para la solución de problemas de optimización. *Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería*, 30(4):264–270.

[Minsterio de industria, 2023] Minsterio de industria, c. y. t. (2023). Logistica y transporte. Disponible en `https://www.investinspain.org/es/sectores/logistica-transporte`.

[MITMA, 2020] MITMA (2020). Infografía sobre emisiones co2 de los coches. Disponible en `https://esmovilidad.mitma.es/noticias/infografia-sobre-emisiones-co2-de-los-coches`.

[MITMA, 2022] MITMA (2022). Observatorio de costes del transporte de mercancias por carretera enero 2022. Disponible en https://www.mitma.gob.es/recursos_mfom/listado/recursos/observatoriocostesmercanciasenero2022v1.pdf.

[Miyazawa and Wakabayashi, 1997] Miyazawa, F. and Wakabayashi, Y. (1997). An algorithm for the three-dimensional packing problem with asymptotic performance analysis. *Algorithmica*, 18:122–144.

[Naseera, 2020] Naseera, S. (2020). P, np, np-hard & np-complete problems.

[Ndèye Fatma Ndiaye, 2014] Ndèye Fatma Ndiaye, Adnan Yassine, I. D. (2014). A branch-and-cut algorithm to solve the container storage problem.

[Nesmachnow, 2014] Nesmachnow, S. (2014). An overview of metaheuristics: Accurate and efficient methods for optimisation. *International Journal of Metaheuristics*, 3:320.

[NexoTrans, 2022] NexoTrans (2022). El 23% de las emisiones co2 son provocadas por camiones tradicionales de combustión. Disponible en https://www.nexotrans.com/noticia/105601/nexotrans/el-23-de-las-emisiones-co2-son-provocadas-por-camiones-tradicionales-de-combustion.html.

[Osman and Kelly, 1996] Osman, I. and Kelly, J. (1996). Meta-heuristics: An overview.

[OTLE, 2023] OTLE (2023). Transporte de mercancías (toneladas) por modo y ámbito (nacional e internacional). Disponible en https://apps.fomento.gob.es/bdotle/visorBDpop.aspx?i=519.

[Parreño et al., 2008] Parreño, F., Alvarez-Valdés, R., Tamarit, J., and Oliveira, J. (2008). A maximal-space algorithm for the container loading problem. *INFORMS Journal on Computing*, 20:412–422.

[Pisinger and Sigurd, 2005] Pisinger, D. and Sigurd, M. (2005). The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2:154–167.

[Ramos et al., 2017] Ramos, A., Silva, E., and Oliveira, J. (2017). A new load balance methodology for container loading problem in road transportation. *European Journal of Operational Research*, 266.

[Rao and Pawar, 2020] Rao, R. V. and Pawar, R. B. (2020). Self-adaptive multi-population rao algorithms for engineering design optimization. *Applied Artificial Intelligence*, 34(3):187–250.

[ROADEF, 2022a] ROADEF (2022a). Roadef challenge 2022. Disponible en https://www.roadef.org/challenge/2022/en/index.php.

[ROADEF, 2022b] ROADEF (2022b). Roadef challenge 2022 git repository. Disponible en https://github.com/renault-iaa/challenge-roadef-2022/.

[Shi and Zhang, 2018] Shi, J. and Zhang, Q. (2018). A new cooperative framework for parallel trajectory-based metaheuristics. *Applied Soft Computing*, 65:374–386.

[SINC, 2009] SINC (2009). La década del 2000 al 2009 ha sido la más cálida. Disponible en `https://www.agenciasinc.es/Noticias/La-decada-del-2000-al-2009-ha-sido-la-mas-calida`.

[UIUC, 2020] UIUC (2020). What is an algorithmic problem? Disponible en `https://courses.engr.illinois.edu/cs374/fa2020/lec_prerec/10/10_1_1_0.pdf`.