Aplicación de CMSA al problema de empaquetado con desbordamiento

Isaac Lozano-Osorio[1][0000−0002−2608−8464] y Christian Blum[2][0000−0002−1736−3559]

[1] GRAFO Research, Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Madrid, España.
isaac.lozano@urjc.es

[2] Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, España.
christian.blum@iiia.csic.es

Resumen En este trabajo se presenta una primera aplicación del algoritmo *Construct, Merge, Solve & Adapt* (CMSA) al problema de empaquetado con desbordamiento (OBPP, del inglés *Overflowing Bin Packing Problem*), un problema de optimización combinatoria recientemente introducido en el que el objetivo es minimizar la suma total de la infrautilización y sobreutilización de un conjunto de contenedores con capacidades heterogéneas. Para la resolución de las subinstancias generadas por CMSA se emplea el modelo de flujo en red (arcflow) propuesto por Martinovic y Strasdat, que constituye el estado del arte en resolución exacta del problema. El ajuste de parámetros se realiza mediante la herramienta `irace` sobre un subconjunto de instancias representativas extraídas de los conjuntos de referencia más competitivos. Los resultados computacionales obtenidos sobre un amplio conjunto de benchmarks muestran que CMSA alcanza la solución óptima en la totalidad de las instancias de los conjuntos Crainic, Dragone y Hemmelmayr, mientras que los conjuntos Belov, `MS1` y `MS2`, caracterizados por su mayor tamaño y dificultad, suponen un reto significativo con desviaciones considerables respecto a los valores óptimos conocidos. Estos resultados sugieren que el modelo de flujo, aunque teóricamente robusto, puede resultar excesivamente complejo para la resolución eficiente de subinstancias dentro de CMSA, lo que motiva como trabajo futuro su sustitución por una formulación basada en cobertura de conjuntos, así como el diseño de operadores de búsqueda local específicos para el problema.

Keywords: CMSA · Optimización Combinatoria · OBPP.

1. Introducción

La asignación eficiente de recursos escasos constituye uno de los problemas fundamentales en investigación operativa y optimización combinatoria. Entre los problemas más estudiados en este ámbito se encuentran el problema de empaquetado en contenedores (BPP) [6] y sus variantes, que han dado lugar a una rica familia de problemas tanto teóricamente desafiantes como de relevancia práctica.

En este trabajo nos centramos en el problema de empaquetado con desbordamiento (OBPP, del inglés *Overflowing Bin Packing Problem*), introducido por Cerrone et al. [4] y recientemente estudiado en profundidad por Martinovic y Strasdat [11,12].

En el OBPP se dispone de un conjunto de ítems de tamaños conocidos y un conjunto de contenedores de capacidades conocidas, con el objetivo de minimizar la suma total de la subutilización y sobreutilización de cada contenedor. A diferencia del BPP clásico, los contenedores pueden llenarse por encima o por debajo de su capacidad, y todos ellos, incluso los vacíos, contribuyen a la función objetivo. Este problema presenta conexiones notables con problemas de planificación justo a tiempo [9], consolidación de servidores [13] y gestión de recursos en entornos de computación en la nube [1], lo que lo convierte en un problema de interés tanto teórico como aplicado.

Desde el punto de vista de su resolución exacta, Martinovic y Strasdat [11] propusieron una formulación basada en flujo en red (*arcflow*) que supera teóricamente y computacionalmente al modelo de asignación previo, resolviendo a optimalidad la totalidad de las instancias de referencia disponibles. Sin embargo, el carácter exacto de dicho enfoque impone limitaciones naturales en instancias de gran tamaño, lo que motiva el desarrollo de métodos heurísticos capaces de obtener soluciones de alta calidad en tiempos de cómputo razonables.

En este trabajo proponemos la aplicación del algoritmo CMSA (*Construct, Merge, Solve & Adapt*) [3] al OBPP, utilizando el modelo *arcflow* como formulación para la resolución de subinstancias. CMSA es un método metaheurístico que combina la construcción probabilística de soluciones con la resolución exacta de subproblemas reducidos, lo que lo convierte en un enfoque especialmente adecuado para problemas de empaquetado. Los resultados obtenidos se comparan frente a las soluciones exactas de referencia sobre un amplio conjunto de instancias de benchmark, mostrando que CMSA es capaz de alcanzar el óptimo en la mayoría de los conjuntos considerados, si bien los conjuntos más difíciles, como Belov, `MS1` y `MS2`, suponen un reto significativo que abre interesantes líneas de trabajo futuro.

El resto del artículo se organiza como sigue. La Sección 2 describe formalmente el OBPP. La Sección 3 presenta el algoritmo CMSA y su adaptación al problema. La Sección 4 describe el conjunto de instancias utilizados junto a la experimentación y resultados obtenidos. Finalmente, la Sección 5 presenta las conclusiones y el trabajo futuro.

2. Descripción formal

Formalmente, dado un conjunto de tipos de ítems con tamaños c_i y cantidades d_i (número de copias del tipo de ítem i que deben distribuirse entre los contenedores de modo que $\sum_k x_{ik} = d_i$), y un conjunto de contenedores con capacidades C_k, el objetivo es encontrar una asignación que minimice la suma de desviaciones absolutas $\sum_k |C_k - \ell_k|$, donde $\ell_k = \sum_i c_i \cdot x_{ik}$ es la carga total del contenedor k, siendo $x_{ik} \in \mathbb{Z}_+$ el número de ítems de tipo i asignados al

contenedor k. A diferencia del BPP clásico, los contenedores pueden llenarse por encima o por debajo de su capacidad, y todos ellos, incluso los vacíos, contribuyen a la función objetivo con un coste igual a $|C_k - \ell_k|$, es decir, la desviación absoluta entre su capacidad y su carga real. En particular, un contenedor vacío incurre en el coste máximo posible, igual a su capacidad completa C_k.

La Figura 3 ilustra este concepto sobre una instancia con tres tipos de ítems (c_1=5 en rojo con d_1=3 copias, c_2=2 en verde con d_2=3 copias, c_3=3 en azul con d_3=2 copias) y tres contenedores (C_1=10, C_2=12, C_3=6), cuya capacidad se indica mediante una línea horizontal.

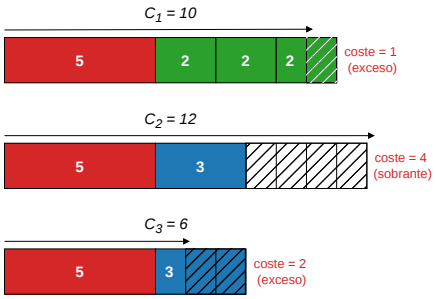


Figura 1.
(a) Solución factible (coste = 7).

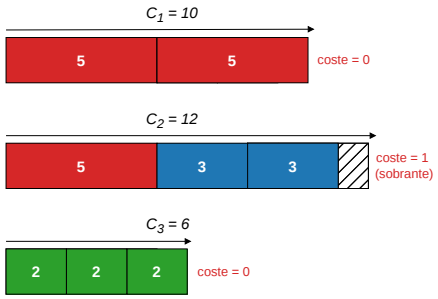


Figura 2.
(b) Solución óptima (coste = 1).

Figura 3. Comparación entre una solución factible arbitraria y la solución óptima para una instancia OBPP con tres tipos de ítems y tres contenedores. Las zonas rayadas indican la desviación de cada contenedor respecto a su capacidad.

En la solución factible de la Figura 3(a), el contenedor C_1 recibe un ítem de tamaño 5 y los tres ítems de tamaño 2, superando su capacidad en una unidad; C_2 recibe un ítem de tamaño 5 y un ítem de tamaño 3, quedando 4 unidades libres; y C_3 recibe un ítem de tamaño 5 y un ítem de tamaño 3, superando su capacidad en 2 unidades. La desviación total acumulada es $1 + 4 + 2 = 7$. La solución óptima, mostrada en la Figura 3(b), reduce esta desviación a 1: los contenedores C_1 y C_3 quedan perfectamente equilibrados y únicamente C_2 presenta una unidad sobrante.

Cabe destacar que Martinovic y Strasdat [12] proponen una técnica de preprocesamiento que permite reducir el tamaño de las instancias antes de aplicar cualquier método de resolución. En particular, si existe un par (i, k) tal que el tamaño de un ítem de tipo i coincide exactamente con la capacidad del contenedor k, es decir, $c_i = C_k$, entonces existe siempre una solución óptima en la que dicho contenedor contiene únicamente una copia de ese ítem, obteniendo coste cero. Por tanto, tanto el contenedor k como una unidad del tipo de ítem i pueden eliminarse de la instancia sin alterar el valor óptimo. Esta reducción se aplica de forma iterativa hasta que no existan más pares de este tipo, reduciendo potencialmente el número de ítems y contenedores a considerar.

El modelo de asignación lineal para el OBPP se formula introduciendo variables $x_{ik} \in \mathbb{Z}_+$ que representan el número de ítems de tipo i asignados al contenedor k, y variables auxiliares $s_k \geq 0$ que capturan la desviación absoluta de cada contenedor respecto a su capacidad:

$$\text{mín} \sum_{k \in K} s_k \tag{1}$$

$$\text{s.t.} \quad \sum_{k \in K} x_{ik} = d_i, \quad i \in I, \tag{2}$$

$$\sum_{i \in I} c_i \cdot x_{ik} + s_k \geq C_k, \quad k \in K, \tag{3}$$

$$\sum_{i \in I} c_i \cdot x_{ik} - s_k \leq C_k, \quad k \in K, \tag{4}$$

$$x_{ik} \in \mathbb{Z}_+, \quad i \in I,\ k \in K. \tag{5}$$

La restricción (2) garantiza que todos los ítems son asignados, mientras que las restricciones (3) y (4) linearizan el valor absoluto $s_k = |C_k - \sum_{i \in I} c_i \cdot x_{ik}|$, imponiendo que s_k sea mayor o igual que la desviación en ambos sentidos (por exceso y por sobrante). Dado que el objetivo minimiza s_k, en cualquier solución óptima se alcanzará la igualdad.

Como alternativa al modelo de asignación, Martinovic y Strasdat [12] introducen una formulación basada en flujo en red, denominada modelo *arcflow*. Para ello, reformulan ligeramente el problema agrupando los contenedores en tipos $k \in \bar{K}$, donde cada tipo agrupa aquellos contenedores con la misma capacidad y D_k denota el número de copias disponibles de cada tipo. La idea central consiste en construir un grafo dirigido $G = (V, A)$ en el que cada camino desde el nodo origen hasta un nodo sumidero representa un patrón de llenado válido para un contenedor. Los arcos se dividen en dos tipos: los arcos ítem modelan la colocación de ítems individuales, mientras que los arcos contenedor conectan cada nodo del grafo con un nodo sumidero t_k asociado a cada tipo de contenedor k, con un coste igual a $|C_k - v|$, es decir, la desviación entre la capacidad del contenedor y la carga acumulada en el camino hasta dicho nodo. De este modo, minimizar el coste total del flujo en la red equivale a minimizar la desviación acumulada de todos los contenedores. La red resultante tiene un número de nodos proporcional a $L + t$ y un número de arcos proporcional a $L \cdot (m + t)$, siendo L la capacidad máxima de los contenedores, m el número de tipos de ítems y t el número de tipos de contenedor, lo que la convierte en una formulación compacta.

Desde el punto de vista de su complejidad, el OBPP es un problema $\mathcal{NP}$-difícil, al ser una generalización del BPP clásico. Además, presenta dos características que lo hacen especialmente desafiante: por un lado, el espacio de soluciones puede exhibir un alto grado de simetría cuando las capacidades de los contenedores son homogéneas, ya que permutar los índices de los contenedores produce múltiples soluciones factibles con el mismo valor objetivo; por otro lado, la relajación lineal del modelo de asignación proporciona una cota inferior

arbitrariamente débil, lo que dificulta la poda del espacio de búsqueda en los métodos exactos basados en programación entera.

3. Propuesta algoritmica

El algoritmo CMSA [3] es un metaheurístico que combina la construcción probabilística de soluciones con la resolución exacta de subproblemas reducidos.

La Figura 4 ilustra el flujo completo de la propuesta multiarranque del algoritmo CMSA propuesto para el OBPP. En este contexto, un componente de solución es un arco de la red de flujo del modelo *arcflow* (ya sea un arco de ítem o un arco de contenedor), de modo que una solución completa queda representada por el conjunto de arcos que forman los caminos de todos los contenedores. El subconjunto activo C', sobre el que se resuelve la subinstancia en la fase de resolución exacta, se inicializa vacío, y el algoritmo ejecuta un bucle principal compuesto por cuatro fases: construcción probabilística de soluciones, incorporación a C' de los componentes presentes en las soluciones construidas, resolución de la subinstancia restringida mediante el modelo matemático, y adaptación de C' incrementando la edad de los componentes y eliminando los obsoletos. Este bucle se repite hasta que se agota el tiempo de la iteración o se obtiene una solución con coste cero, lo que indica que se ha alcanzado el valor óptimo. Al finalizar el tiempo máximo de iteración, se comprueba si se dispone de tiempo adicional; en ese caso, se reinicia C' y el proceso comienza de nuevo. En caso contrario, se devuelve la mejor solución encontrada durante el experimento.

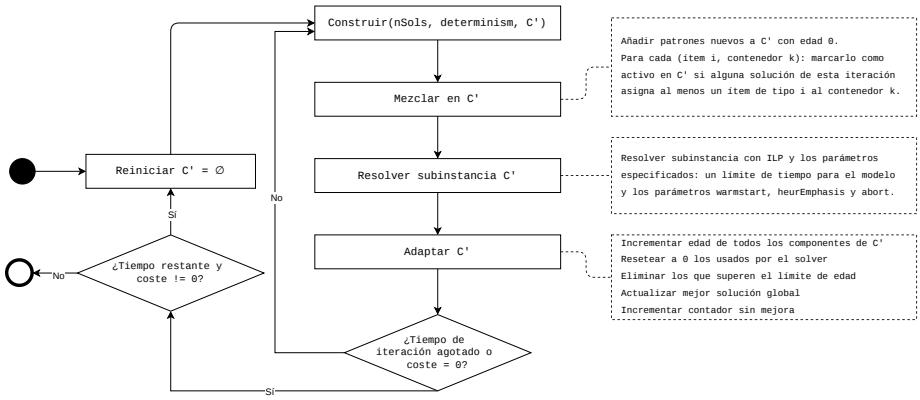


Figura 4. Diagrama de flujo del algoritmo CMSA aplicado al OBPP.

Los componentes de solución se definen como pares (i, k), donde i es un tipo de ítem y k es un contenedor. Un componente (i, k) está activo en C' si por lo menos un ejemplar del tipo de ítem i fue asignado al contenedor k en alguna de las soluciones construidas recientemente (con independencia del número de copias

asignadas). A cada componente activo se le asocia una edad, que se incrementa en cada iteración en la que el componente no es utilizado por el solver exacto. Cuando la edad supera el límite `age`, el componente es eliminado de C'.

En la fase de construcción, se generan `nSols` soluciones mediante un constructivo voraz aleatorizado por contenedor (*Bin-Completion Greedy*). Para cada contenedor (en orden decreciente de capacidad), se añaden ítems uno a uno mientras exista alguno que reduzca estrictamente la desviación actual $|C_k - \ell_k|$, seleccionando en cada paso el ítem que mayor reducción produce. Los ítems no colocados en esta primera fase se asignan mediante un voraz de respaldo que, para cada ítem restante (en orden decreciente de tamaño), lo coloca en el contenedor que minimiza la desviación resultante $|C_k - (\ell_k + c_i)|$. La complejidad del constructivo es $\mathcal{O}(N \cdot m \cdot n)$, donde $N = \sum_i d_i$ es el número total de ítems. Para introducir diversidad, con probabilidad $1 -$ `determinism` el ítem seleccionado se sustituye por uno elegido uniformemente al azar entre todos los candidatos que mejoran estrictamente la desviación del contenedor.

En la fase de resolución, se construye una subinstancia restringida a los componentes activos en C' y se resuelve mediante el modelo *arcflow* (Modelo III) de Martinovic y Strasdat [12], utilizando Gurobi como solver exacto con un límite de tiempo de `gurobiSeconds` segundos. La restricción de C' se aplica a nivel de grupo de capacidad: para cada capacidad C_k, solo se incluyen en la red los tipos de ítems activos en algún contenedor de esa capacidad, y los arcos de contenedor (v, C_k) solo se añaden si el nodo v es alcanzable usando exclusivamente dichos tipos. Los ítems cuyo tipo no está activo en ningún contenedor, o cuya demanda no puede cubrirse completamente con la red restringida, se asignan mediante una heurística voraz tras la resolución del modelo. El solver puede interrumpirse anticipadamente si se encuentra una solución que mejora la mejor solución conocida, controlado por el parámetro `abort`.

La Figura 5 muestra la red de flujo de la subinstancia C' para el ejemplo de la Figura 1. Cada nodo $v \in V$ representa un nivel de carga acumulada en un contenedor; el nodo 0 es el estado inicial (contenedor vacío) y su posición horizontal es proporcional a v, de modo que la longitud de cada arista refleja el tamaño del ítem. Una arista de ítem $(u \to u + c_i)$ indica la colocación de un ítem del tipo i que eleva la carga de u a $u + c_i$; se dibujan en rojo (t_0, $c = 5$), verde (t_1, $c = 2$) y azul (t_2, $c = 3$), con capacidad de flujo d_i sobre cada arco. Los tres nodos sumidero t_6, t_{10} y t_{12} representan los contenedores C_3=6, C_1=10 y C_2=12; una arista de contenedor $(v \to t_{\text{cap}})$ con coste $|C_k - v|$ consume una unidad de flujo cuando un contenedor de esa capacidad alcanza la carga v.

La solución constructiva asignó a C_1 los tipos t_0, t_1 y a C_2, C_3 los tipos t_0, t_2, definiendo C'. Para que una arista de contenedor $(v \to t_{\text{cap}})$ sea válida en la subinstancia, debe existir un camino desde el nodo 0 hasta v usando exclusivamente los tipos permitidos para esa capacidad. El nodo 8 solo se alcanza mediante $0 \xrightarrow{t_0} 5 \xrightarrow{t_2} 8$; como t_2 no aparece en C_1, dicho tipo no está permitido para t_{10} y la arista $(8 \to t_{10})$ se descarta (gris). El mismo razonamiento excluye el nodo 3 de t_{10}, y los nodos 2, 4 y 7 (alcanzables solo mediante t_1, ausente en C_2 y C_3) de los sumideros t_{12} y t_6.

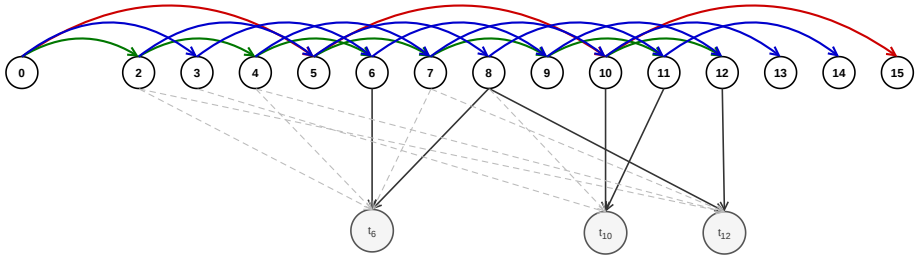


Figura 5. Red de flujo restringida a C' para la instancia de ejemplo. Las aristas de ítem están coloreadas por tipo (t_0 rojo, t_1 verde, t_2 azul); las aristas de contenedor discontinuas en gris son descartadas por la restricción de C'.

El modelo de flujo asigna una variable entera de flujo a cada arista: $\xi_{uv} \geq 0$ para aristas de ítem y $\xi_{v,t_k} \geq 0$ para aristas de contenedor. La conservación de flujo en cada nodo $v \neq 0$ (flujo entrante igual a flujo saliente) impone que las variables formen caminos completos de 0 hasta los sumideros, sin flujo "suelto" en nodos intermedios. Las restricciones de demanda $\sum \xi_{uv} + y_i = d_i$ garantizan que todos los ítems queden asignados, y las de uso de contenedor $\sum_v \xi_{v,t_k} = D_k$ exigen que cada sumidero reciba exactamente D_k unidades de flujo. El objetivo minimiza la desviación total $\sum_{(v,t_k)} |C_k - v| \cdot \xi_{v,t_k}$ más una penalización $\sum_i c_i y_i$ por los ítems no cubiertos por la red restringida. Gurobi resuelve el programa lineal entero (ILP) resultante; la asignación x_{ik} se recupera trazando hacia atrás el camino de flujo de cada contenedor desde su nodo de carga hasta 0, donde cada arista (u, v) recorrida identifica un ítem asignado a ese contenedor.

Tras la resolución, los componentes de C' se actualizan incrementando la edad de todos los pares activos y reseteando a cero la edad de los pares utilizados en la solución obtenida por el solver. Los pares cuya edad supera el límite `age` son eliminados de C', reduciendo el tamaño de la subinstancia en iteraciones sucesivas.

4. Resultados

Esta sección presenta los resultados experimentales obtenidos por el algoritmo CMSA propuesto para el OBPP. Para evaluar su rendimiento, se compara con la formulación más eficiente de la literatura, concretamente el modelo *arc-flow*. Con el objetivo de garantizar una comparación justa y reproducible, todos los métodos se ejecutaron bajo las mismas condiciones: ejecución monohilo, límite de tiempo de 600 segundos por instancia y un servidor equipado con un procesador AMD EPYC 9655P limitado a 32 núcleos y 32 GB de RAM. La implementación del algoritmo se realizó en Java 25 sobre Oracle GraalVM, empleando Gurobi 11.0.3 como solver exacto para la resolución del modelo exacto y de las subinstancias.

Los conjuntos de instancias utilizados en la comparación provienen de varias fuentes. Los conjuntos `MS1` y `MS2`, propuestos por Martinovic y Strasdat [12],

contienen 500 instancias cada uno, construidas a partir de pares (m, p) donde $m \in 10, 20, \ldots, 100$ es el número de tipos de ítem, $p \in 3, 5, 10, 15, 20$ el número de ítems individuales por contenedor en la solución óptima y cada tipo tiene entre 1 y 10 copias. En `MS1` la solución óptima es siempre cero, mientras que en `MS2` las capacidades se perturban ligeramente y el óptimo puede ser estrictamente positivo, convirtiéndola en el conjunto más exigente de los dos.

Los conjuntos `Belov_1`-`Belov_4` [2], `Crainic_1`-`Crainic_3` [5] y `Hemmelmayr_1`-`Hemmelmayr_2` [8] provienen de instancias del problema de empaquetado de tamaño variable, adaptadas al OBPP por [12]. Los conjuntos Belov son los de mayor escala, con un promedio de unos 99 tipos de ítem, alrededor de 51 copias por tipo y cerca de 5000 ítems repartidos en aproximadamente 2000 contenedores, lo que los convierte en uno de los más difíciles de resolver. Los conjuntos Crainic presentan escalas más moderadas: `Crainic_1` tiene unos 51 tipos con aproximadamente 3 copias de media y 175 ítems en total distribuidos en 72 contenedores de un único tipo de capacidad; `Crainic_2` y `Crainic_3` tienen entre 14 y 20 tipos pero con muchas más copias por tipo (23 y 32 de media respectivamente), acumulando unos 450 ítems en pocos contenedores. Los conjuntos Hemmelmayr difieren notablemente entre sí: `Hemmelmayr_1` presenta cerca de 186 tipos con unas 4 copias de media y casi 600 contenedores, mientras que `Hemmelmayr_2` es considerablemente más pequeño, con 62 tipos, unas 3 copias de media y unos 70 contenedores.

Los conjuntos Dragone, también denominados F1, F2 y F3 en la literatura [4], fueron los primeros propuestos específicamente para el OBPP, con 200 instancias cada uno. En promedio tienen unos 55 tipos de ítem con entre 3 y 4 copias por tipo, resultando en torno a 185 ítems individuales por instancia. F1 y F2 tienen solución óptima cero por construcción, mientras que F3 presenta capacidades casi homogéneas (con apenas 1 o 2 tipos de contenedor distintos) sin garantía de optimalidad nula, lo que la convierte en el subconjunto más desafiante de los tres.

4.1. Ajuste de parámetros

El algoritmo CMSA requiere la configuración de un conjunto de parámetros para obtener su mejor rendimiento. El ajuste de dichos parámetros se realizó mediante la herramienta `irace` [10] sobre un subconjunto de 39 instancias seleccionadas por su mayor dificultad, pertenecientes principalmente a los conjuntos Belov, `MS1` y `MS2`. Los parámetros considerados, junto con sus dominios de búsqueda, son los siguientes: el número de soluciones construidas en cada iteración (`nSols` $\in [2, 1000]$), la edad máxima de los componentes de solución (`age` $\in [1, 10]$), el grado de determinismo en la construcción (`determinism` $\in [0, 1]$), el uso de arranque en caliente en Gurobi (`warmstart` $\in \{true, false\}$), la interrupción anticipada del solver (`abort` $\in \{true, false\}$), el énfasis en soluciones heurísticas (`heurEmphasis` $\in \{true, false\}$) y el tiempo máximo asignado a Gurobi por subinstancia en segundos (`gurobiSeconds` $\in [1, 300]$). La mejor configuración del algoritmo CMSA encontrada mediante `irace` utiliza 850 soluciones por iteración (`nSols`=850), un número máximo de reinicios de 5 (`nRestarts`=5), de forma que

el tiempo total disponible se distribuye equitativamente entre ellos, una edad máxima de 4 (`age`=4) y un nivel de determinismo de 0.8 (`determinism`=0.8), valor elevado que indica un comportamiento predominantemente determinista en la construcción de soluciones. El `abort` anticipado se desactiva, mientras que el énfasis heurístico de Gurobi se activa (`heurEmphasis`=true), siendo el tiempo máximo asignado al solver por subproblema de 187 segundos (`gurobiSeconds`=187).

Para comprender en mayor profundidad el impacto individual de cada parámetro del algoritmo CMSA, se realizó un análisis de ablación siguiendo la metodología propuesta por Fawcett y Hoos [7]. Este tipo de análisis tiene como motivación principal determinar qué parámetros son responsables de las diferencias de rendimiento entre dos configuraciones algorítmicas: una configuración de partida y una configuración objetivo, generalmente la mejor encontrada por el proceso de configuración automática. Para ello, el análisis genera, en cada paso, todas las configuraciones que difieren en exactamente un parámetro respecto a la configuración actual, las evalúa sobre un conjunto de instancias y selecciona aquella que mayor mejora produce. Este proceso se repite hasta que la configuración de partida se ha transformado completamente en la configuración objetivo, revelando así la secuencia de cambios más influyentes y la contribución marginal de cada parámetro.

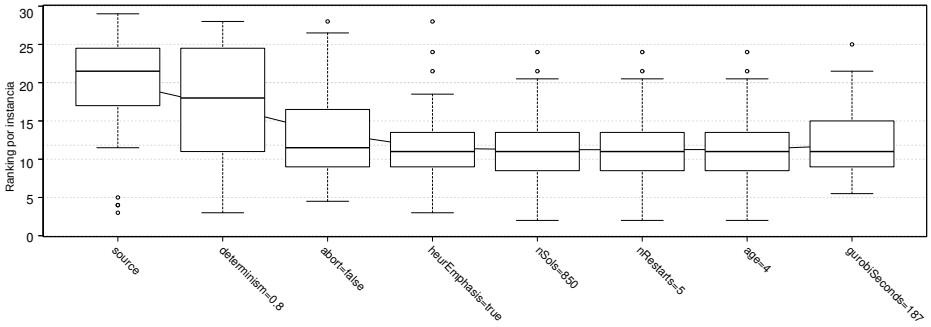


Figura 6. Análisis de ablación del CMSA para el OBPP. Cada caja muestra el ranking por instancia tras fijar el parámetro indicado, partiendo de source hasta la configuración objetivo de irace. Menor ranking indica mejor rendimiento.

En nuestro caso, se tomó como configuración fuente los valores por defecto del algoritmo y como objetivo la configuración élite identificada por irace. La Figura 6 muestra la evolución del ranking medio por instancia a lo largo del proceso. El cambio más significativo se produce al fijar `determinism`=0.8, lo que indica que el nivel de determinismo en la construcción de soluciones es el parámetro más influyente. A continuación, `abort`=false aporta una mejora adicional relevante. Los parámetros restantes (`heurEmphasis`=true, `nSols`=850, `nRestarts`=5, `age`=4 y `gurobiSeconds`=187) contribuyen de forma más moderada pero acumulativa, hasta alcanzar el rango final de la configuración objetivo.

4.2. Resultados computacionales

En esta sección se presentan los resultados obtenidos por CMSA sobre los conjuntos de instancias descritos anteriormente, comparados con los resultados exactos (ILP) de referencia. Para cada conjunto se reportan el valor medio de la función objetivo obtenida por CMSA sobre los óptimos conocidos (FO), el número de coincidencias con la solución exacta (Óptimos), el tiempo medio en encontrar la mejor solución en segundos (Tiempo (s)) y la desviación media absoluta respecto al óptimo conocido (Desv). La Tabla 1 muestra los resultados obtenidos de nuestra propuesta.

Cuadro 1. Resultados computacionales de CMSA comparados con el ILP.

		ILP			CMSA			
Conjunto	Instancias	FO	Óptimos	Tiempo (s)	FO	Óptimos	Tiempo (s)	Desv
`MS1`	500	**0,00**	**439**	**85,06**	18,94	359	229,24	18,94
`MS2`	500	**2,31**	**395**	**86,14**	17,68	306	337,20	15,37
`Belov_1`	50	**43759,88**	**50**	**149,64**	43759,96	46	599,48	0,08
`Belov_2`	50	**36828,18**	**50**	**136,82**	64328,54	45	601,80	27500,36
`Belov_3`	50	**27845,38**	**50**	**120,71**	38893,14	46	605,36	11047,76
`Belov_4`	50	**23833,60**	**50**	**110,28**	71013,24	42	600,78	47179,64
`Crainic_1`	300	**129,81**	**300**	0,14	**129,81**	**300**	**0,00**	**0,00**
`Crainic_2`	60	**102,30**	**60**	0,26	**102,30**	**60**	**0,00**	**0,00**
`Crainic_3`	480	**59,09**	**480**	0,05	**59,09**	**480**	**0,00**	**0,00**
`Dragone_F1`	240	**0,00**	**240**	1,54	**0,00**	**240**	**0,00**	**0,00**
`Dragone_F2`	240	**0,00**	**240**	2,91	**0,00**	**240**	**0,00**	**0,00**
`Dragone_F3`	240	**730,62**	**240**	0,75	**730,63**	**240**	**0,00**	**0,00**
`Hemmelmayr_1`	150	**3048,30**	**150**	0,96	**3048,30**	**150**	**0,00**	**0,00**
`Hemmelmayr_2`	50	**64,52**	**50**	0,26	**64,52**	**50**	**0,00**	**0,00**
Total	**2960**	**2621,16**	**2794**	**35,34**	4160,45	2604	137,82	1539,29

En la Tabla 1 se observa un comportamiento claramente diferenciado entre conjuntos. Por un lado, CMSA alcanza la solución exacta en la totalidad de las instancias de Crainic, Dragone y Hemmelmayr, confirmando que estos conjuntos no suponen un reto significativo para el algoritmo. Por otro lado, los conjuntos Belov, `MS1` y `MS2` se revelan como los más competitivos, tal y como mostraba la literatura. En los conjuntos Belov, CMSA obtiene 179 de los 200 óptimos (46, 45, 46 y 42 instancias en `Belov_1` a `Belov_4` respectivamente), y las desviaciones crecen de forma considerable con el índice del conjunto, llegando a una desviación absoluta media de 47179,64 en `Belov_4`. Esto refleja la elevada dificultad de estas instancias, con cerca de 5000 ítems y 2000 contenedores, que supone un reto abierto para métodos heurísticos. En `MS1` y `MS2` el comportamiento es intermedio: se obtienen 359 y 306 coincidencias respectivamente, pero con desviaciones absolutas medias de 18,94 y 15,37 respectivamente. En conjunto, CMSA alcanza la solución exacta en 2604 de los 2794 óptimos conocidos de las 2960 instancias consideradas, con un tiempo medio de ejecución de 137,82 segundos.

5. Conclusiones y trabajo futuro

En este trabajo se ha presentado una primera aplicación del algoritmo CMSA al problema de empaquetado con desbordamiento (OBPP), comparando los resultados obtenidos frente a las soluciones exactas ILP de referencia. Los resultados muestran que CMSA es capaz de alcanzar la solución óptima en la totalidad de las instancias de los conjuntos Crainic, Dragone y Hemmelmayr, mientras que los conjuntos Belov, `MS1` y `MS2` suponen un reto significativo, con desviaciones considerables respecto al óptimo conocido. Una de las principales limitaciones identificadas reside en el modelo utilizado para resolver las subinstancias durante la fase de resolución de CMSA. En concreto, el modelo *arcflow*, si bien teóricamente superior en términos de cota LP, resulta excesivamente complejo para ser resuelto de forma eficiente en el contexto de subinstancias reducidas, donde la rapidez de resolución es crítica para el rendimiento global del algoritmo. En este sentido, una línea de trabajo futuro especialmente prometedora es la sustitución del modelo arcflow por una formulación basada en cobertura de conjuntos, siguiendo la aproximación descrita en [3], donde se demuestra que este tipo de modelos puede comportarse de forma significativamente más eficiente dentro de CMSA al reducir la complejidad de las subinstancias y favorecer una resolución más rápida por parte del modelo. Adicionalmente, el diseño de operadores de búsqueda local específicos para el OBPP y la exploración de estrategias de construcción de soluciones más adaptadas a instancias de gran tamaño, como las del conjunto Belov, constituyen direcciones de investigación de gran interés para mejorar la calidad de las soluciones obtenidas.

Agradecimientos Este trabajo ha sido posible gracias al apoyo del Ministerio de Economía y Competitividad (subvención con referencia PID2024-156045NB-I00) y la Comunidad Autónoma de Madrid (subvención con referencia TEC-2024/COM-404).

Declaración de intereses Los autores no tienen ningún conflicto de intereses que declarar que sea relevante para el contenido de este artículo.

Referencias

1. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems **28**(5), 755–768 (2012). `https://doi.org/10.1016/j.future.2011.04.017`, special Section: Energy efficiency in large-scale distributed systems
2. Belov, G., Scheithauer, G.: A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. European Journal of Operational Research **141**(2), 274–294 (Sep 2002). `https://doi.org/10.1016/s0377-2217(02)00125-x`
3. Blum, C.: Construct, Merge, Solve & Adapt: A Hybrid Metaheuristic for Combinatorial Optimization. Springer Nature Switzerland (2024). `https://doi.org/10.1007/978-3-031-60103-3`

4. Cerrone, C., Dragone, R., Sciomachen, A.: The balanced bin packing problem. Conference presentation at the International Conference on Optimization and Decision Science (ODS 2023), Ischia, Italy (September 2023)
5. Crainic, T.G., Perboli, G., Rei, W., Tadei, R.: Efficient lower bounds and heuristics for the variable cost and size bin packing problem. Computers & Operations Research **38**(11), 1474–1482 (Nov 2011). https://doi.org/10.1016/j.cor.2011.01.001
6. Delorme, M., Iori, M., Martello, S.: Bin packing and cutting stock problems: Mathematical models and exact algorithms. European Journal of Operational Research **255**(1), 1–20 (Nov 2016). https://doi.org/10.1016/j.ejor.2016.04.030
7. Fawcett, C., Hoos, H.H.: Analysing differences between algorithm configurations through ablation. Journal of Heuristics **22**(4), 431–458 (Jan 2015). https://doi.org/10.1007/s10732-014-9275-9
8. Hemmelmayr, V., Schmid, V., Blum, C.: Variable neighbourhood search for the variable sized bin packing problem. Computers & Operations Research **39**(5), 1097–1108 (May 2012). https://doi.org/10.1016/j.cor.2011.07.003
9. Kubiak, W.: Minimizing variation of production rates in just-in-time systems: A survey. European Journal of Operational Research **66**(3), 259–271 (1993). https://doi.org/10.1016/0377-2217(93)90215-9
10. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016). https://doi.org/10.1016/j.orp.2016.09.002
11. Martinovic, J., Strasdat, N.: The overflowing bin packing problem: theoretical results and compact ILP formulations. Mathematical Methods of Operations Research **102**(2-3), 395–433 (Dec 2025). https://doi.org/10.1007/s00186-025-00913-3
12. Martinovic, J., Strasdat, N.: Correction: The overflowing bin packing problem: theoretical results and compact ILP formulations. Mathematical Methods of Operations Research (Apr 2026). https://doi.org/10.1007/s00186-026-00920-y
13. Speitkamp, B., Bichler, M.: A mathematical programming approach for server consolidation problems in virtualized data centers. IEEE Transactions on Services Computing **3**(4), 266–278 (2010). https://doi.org/10.1109/TSC.2010.25