Traballo Fin de Mestrado

O problema de empaquetamento "The bin packing problem"

Diego Frade Amil

Mestrado en Técnicas Estatísticas

Curso 2018-2019

Proposta de Traballo Fin de Mestrado

Título en galego: O problema de empaquetamento “The bin packing problem”

Título en español: El problema de empaquetamiento “The bin packing problem”

English title: The bin packing problem

Modalidade: Modalidade A

Autor/a: Diego Frade Amil, Universidade de Santiago de Compostela

Director/a: María Luísa Carpente Rodríguez, Universidade da Coruña

Breve resumo do traballo:

O problema de optimización combinatoria de empaquetamento consiste en tratar de meter o número máximo de obxectos nunha serie de contedores tratando de non exceder un volume dado (custo, peso...). As solucións a este problema teñen un gran espectro de aplicacións: transporte aéreo, marítimo e terrestre, problemas de corte industrial, etc. Neste traballo preténdese facer unha revisión da literatura máis relevante no tema. Tratarase de facer unha clasificación deste tipo de problemas, facendo fincapé nos modelos exactos e noutras técnicas de solución aproximadas.

Recomendacións: Bibliografía recomendada:

Lodi, A., Martello, S., Monaci, M. (2002). Two-dimensional packing problems: A survey. European journal of operational research, 141(2), 241- 252.

Martello, S., Pisinger, D., Vigo, D. (2000). The three-dimensional bin packing problem. Operations Research, 48(2), 256-267.

Agradecementos

A Luisa Carpente Rodríguez, directora deste TFM, pola súa paciencia, a súa axuda e os seus consellos para sacar adiante con éxito este traballo.

Ao profesor Silvano Martello, pola súa amable atención e axuda coas dúbidas relativas ao MTP.

A Luis Alberto Silva Escamilla, pola súa amabilidade ao compartir o material relacionado co MBS.

A todas aquelas persoas que me brindaron o seu ánimo e apoio durante os meus estudos de grao e mestrado, en especial meus pais Isabel e Joaquín; e a todas as que se cruzaron no meu camiño e contribuíron a converterme na persoa que son e que fixeron posible que chegase a onde cheguei.

Simplemente, GRAZAS.

Índice xeral

Prefacio

Empaquetar obxectos nun ou en varios recipientes ten múltiples aplicacións na nosa vida cotiá. Algúns casos resultan evidentes, como colocar roupa e zapatos nunha maleta de viaxe ou cargar contedores nun barco de mercadorías. Pola contra, outras situacións como asignar unha serie de traballos ou cálculos a un certo número de máquinas ou computadores non parecen tan obvias, mais chega con considerar os traballos como obxectos e as máquinas como contedores para ver a correspondencia cos chamados *problemas de empaquetamento* (*packing problems* en lingua inglesa), nome que reciben ditas situacións.

Ao longo dos anos, numerosos autores estudaron un gran abano de problemas deste tipo, cuxas principais diferenzas radican na forma dos obxectos e recipientes ou na dimensión dos mesmos. Tamén se producen distintas variantes de problemas de empaquetamento mediante a adición de determinadas restricións: no número de recipientes, na orde en que deben empaquetarse os obxectos, na posición destes dentro dos recipientes... No presente traballo abordamos un dos problemas de empaquetamento máis populares, que na literatura especializada se denomina *bin packing problem* (BPP) e cuxa posible tradución podería ser *problema do empaquetamento en recipientes ou contedores*, denominación que serve para distinguilo da categoría xenérica de problemas de empaquetamento. Precisamente, o termo *empaquetamento* acompañaranos ao longo deste TFM e faremos uso del constantemente, de xeito indistinto para indicar a asignación de obxectos nun recipiente concreto como para a propia solución final do problema.

A estrutura xeral do traballo é a seguinte. No Capítulo 1 facemos unha breve introdución ao BPP, situándoo dentro dos problemas de empaquetamento e indicando as técnicas de resolución existentes e os recursos dispoñibles en Internet. No Capítulo 2 entramos de cheo no BPP nunha soa dimensión, no que nos deteremos especialmente. Revisaremos variantes e casos particulares e, de xeito pormenorizado, algúns métodos para resolvelo, así como ferramentas existentes para o seu tratamento. O Capítulo 3 dedicarémolo a examinar o BPP en máis dunha dimensión, explicando brevemente as principais características e métodos de resolución en cada caso. E finalizamos este TFM cunhas breves conclusións e futuras liñas de investigación no Capítulo 4.

Capítulo 1

Introdución ao problema

Os *problemas de empaquetamento* (*packing problems*) son unha clase de problemas de optimización combinatoria nos que xeralmente se trata de empaquetar un conxunto de obxectos nun único contedor ou en varios e cuxo obxectivo é, por exemplo, minimizar o espazo non ocupado por tales obxectos ou o número de contedores empregados (Schweer 2010). Este problema correspóndese con innumerables situacións da vida cotiá (colocar roupa nun armario, libros nunha biblioteca...), cuestións industriais (carga de contedores nun barco, corte de pezas...) ou diferentes liñas de investigación na xeometría sobre o empaquetado de figuras planas dentro doutras: círculos en círculos, círculos en cadrados, cadrados en cadrados e un longo etcétera. Con isto último están relacionados os chamados problemas de cobertura, que fan referencia a cubrir o interior dunha figura coa maior cantidade posible doutras de menor tamaño e idénticas entre si. Un exemplo é empaquetar círculos nun cadrado. O obxectivo é, dado un conxunto de círculos, determinar o radio máximo que permita empaquetalos no cadrado sen que se solapen ou, equivalentemente, maximizar a distancia mínima entre o centro de cada par de círculos (Nurmela e Östergård 1997). Na Figura 1.1[1] pódense ver dous exemplos de problemas de empaquetamento: carga de contedores e empaquetado de círculos nun cadrado. Convén destacar que no exemplo da carga de contedores nun buque estariamos realmente ante dous problemas de empaquetamento que forman un só: o enchido de cada contedor e a disposición destes dentro ou sobre a cuberta do barco.

Figura 1.1: Exemplos de problemas de empaquetamento: buque de contedores (esquerda) e círculos nun cadrado (dereita).

[1] As imaxes que aparecen nesta Figura foron tomadas de Wikimedia Commons:

- Keith Skipper (2015) *CSCL Globe on her maiden voyage arriving at Felixstowe* (baixo licencia CC BY-SA 2.0).
- Parcly Taxel (2018) *The densest packing of fifteen equal circles in a square* (baixo licencia FAL/LAL).

Moi relacionados cos problemas de empaquetamento atopamos os chamados *problemas de corte* ou *cutting problems* na literatura especializada, sobre algún dos cales falaremos brevemente máis adiante. A abundancia de bibliografía sobre estes dous tipos de problemas deu lugar á aparición de numerosas denominacións para, ás veces, os mesmos problemas. Co fin de poñer orde entre tanto barullo de nomes, creáronse as *tipoloxías de problemas de corte e empaquetamento*, que organizan estes tipos de problemas en categorías homoxéneas atendendo a unha serie de criterios. A primeira en levarse a cabo foi a de Dyckhoff (1990). Nesta, establécense 9 criterios para a clasificación dos distintos problemas: dimensión, medida da cantidade de obxectos e contedores (dicreta ou continua), forma, variedade e dispoñibilidade de obxectos e contedores, restricións xeométricas, restricións de asignación, obxectivo (minimizar ou maximizar) e estado da información e variabilidade (datos coñecidos ou aleatorios). Tendo en conta estes criterios, Dickhoff propón 96 tipos de problemas identificando as seguintes 4 características:

- Dimensión: unha (1), dúas (2), tres (3) ou N dimensións, N$>$ 3 (N).
- Tipo de asignación: todos os contedores e unha selección de obxectos (B) ou unha selección de contedores e todos os obxectos (V).
- Variedade de contedores: un só (O), todos iguais (I) ou con diferentes medidas (D).
- Variedade de obxectos: poucos con diferentes formas e medidas (F), moitos con moitas formas e medidas diferentes (M), moitos con relativamente poucas formas e medidas diferentes (R) e obxectos con formas congruentes (idénticas) (C).

A publicación de Dyckhoff (1990) marcou inicialmente un fito na investigacións dos problemas C&P (siglas do inglés *cutting and packing*), permitindo unha boa estruturación destes problemas. Pero non acadou a relevancia internacional que cabía esperar, en parte pola utilización dun código de abreviaturas pouco comprensibles para o ámbito investigador maioritariamente anglófono (de feito, as abreviaturas dos tipos de asignación proveñen de senllas palabras en alemán). Ademais, ante a crecente produción e especialización bibliográfica, esta tipoloxía acabou por ser insuficiente para abarcar todo o desenvolvemento acadado nos problemas C&P. Partindo das bases desta tipoloxía e tratando de corrixir as súas eivas, xorde a publicación de Wäscher et al. (2007). Nesta obra modifícanse as características definitorias dos problemas e créanse novas categorías de problemas combinando ditas características con algúns dos 9 criterios vistos (básicos, intermedios e refinados).

No presente traballo trataremos un tipo particular de problema de empaquetamento, incluído dentro da categoría de problemas C&P básicos que nos presenta Wäscher et al. (2007): o *bin packing problem* (BPP) ou *problema de empaquetamento en recipientes*. Dado un conxunto de *elementos* de varios tamaños, a finalidade de dito problema é acomodar os citados elementos en *recipientes* sen exceder a capacidade destes e usando o menor número posible (Escamilla et al. 2017). Como acabamos de ver nesta primeira definición informal, contamos con dúas palabras clave ao tratar este problema: elementos ou *items*, que tamén denominaremos obxectos ao longo do traballo; e recipientes (*bins*) ou contedores. As características dos recipientes determinan as categorías intermedias do BPP que propón Wäscher et al. (2007) (véxase a Figura 1.2):

- *Single bin-size bin packing problem* (SBSBPP): Todos os recipientes teñen as mesmas medidas. Tal característica fai que esta sexa a categoría comunmente máis asociada co propio BPP, considerando as restantes como variantes do problema xeral. En función da dimensión, atopamos as seguintes versións, que ademais son as máis coñecidas:
  - Unidimensional (1D-BPP): Os tamaños dos elementos (tamaño, peso, requisito de capacidade, lonxitude...) e dos recipientes (capacidade) veñen dados por unha soa variable.
  - 2 dimensións (2D-BPP): Os tamaños correspóndense con dúas variables (altura e anchura).
  - 3 dimensións (3D-BPP): Os tamaños están definidos por tres variables (altura, anchura e profundidade).

Existe tamén unha variante en 4 dimensións na que se ten en conta unha variable adicional, ademais das tres citadas para o 3D-BPP, que pode estar relacionada co peso, co valor ou proveito do obxecto (Yang 2017) ou co tempo (Dyckhoff 1990).

- *Multiple bin-size bin packing problem* (MBSBPP): Existe un número reducido de clases de recipientes en función das súas medidas e, polo tanto, a cantidade de recipientes por clase tende a ser grande.
- *Residual bin packing* (RBPP): A variedade de clases de medidas dos recipientes é moito maior ca no MBSBPP, o que implica que o número de recipientes en cada unha delas sexa menor.

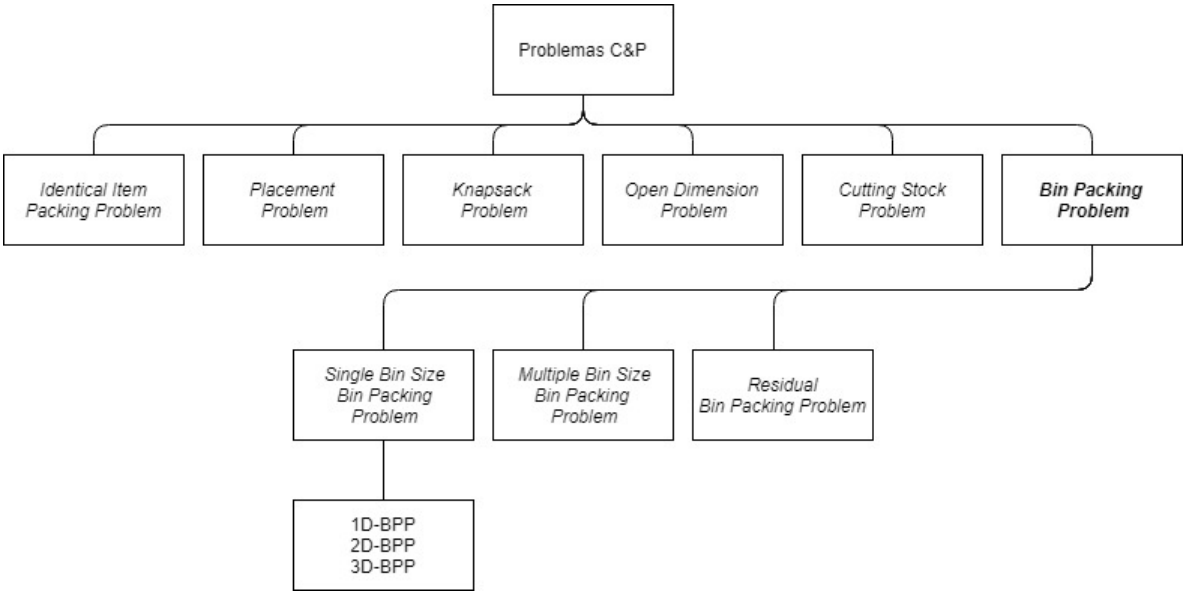

Figura 1.2: Clasificación do BPP dentro dos problemas C&P (Wäscher et al. 2007).

O BPP é un tema de investigación no que afondaron numerosos autores, situación que deu lugar a unha gran colección de material científico. Tal é o alcance destas contribucións que a obra de Coffman et al. (2004) está completamente dedicada á recompilación de bibliografía sobre o BPP, superando o medio milleiro de artigos, libros e teses que tratan sobre este problema, as súas características, propiedades, métodos de resolución, casos relacionados e outros moitos aspectos. Dentro desa colección bibliográfica convén destacar, pola súa prolífica produción, a autores como Silvano Martello, Edward G. Coffman Jr., Andrea Lodi, Yossi Azar, Alberto Caprara, Leah Epstein, M. R. Garey ou Janos Csirik, entre outros.

1.1. Técnicas de resolución

Xa comentamos ao comezo desta introdución ao BPP que este era un problema de optimización e, como tal, conta con certos métodos ou algoritmos de resolución que poden clasificarse atendendo á súa precisión:

- Exactos: Devolven sempre a solución óptima do problema. Dentro desta clase podemos destacar os *modelos de programación matemática*. Estes constan dunha función, chamada *función obxectivo*, que debe ser maximizada ou minimizada tendo en conta unha serie de *restricións* que limitan as posibles solucións. No caso do BPP, estes modelos adoitan ser de *programación linear*, que

se caracterizan por ter funcións de tipo linear como restricións (Martello e Toth 1990, Jin et al. 2003, Lodi et al. 2004). Ademais, existen algoritmos exactos para resolver o BPP nas súas distintas variantes (Martello e Toth 1990, Scholl et al. 1997, Korf 2002).

- Aproximados: Dan lugar a solucións situadas nunha determinada porcentaxe do óptimo. Así, dise que un algoritmo é λ-aproximado cando devolve unha solución x tal que:

$$\begin{cases} OPT \leq x \leq \lambda \cdot OPT & \text{se } \lambda > 1, \\ \lambda \cdot OPT \leq x \leq OPT & \text{se } \lambda < 1, \end{cases}$$

  onde OPT denota a solución óptima. Para a resolución do BPP, existen certas estratexias na asignación dos elementos que determinan unha boa colección de algoritmos aproximados (Scholl et al. 1997, Lodi et al. 2002).

- Heurísticos: Producen solucións sen garantía algunha de optimalidade, pero adoitan ter un tempo de execución curto. Unha das heurísticas máis destacables para a resolución do BPP (unidimensional) é o MBS (Gupta e Ho 1999), pero existe un número moi elevado de métodos deste tipo (Fleszar e Hindi 2002, Lodi et al. 2002b, Maarouf et al. 2008).

Idealmente, os métodos preferibles son os exactos, pero podemos atoparnos ante problemas nos que estes algoritmos sexan moi lentos. Entón, deberiamos botar man dos algoritmos aproximados ou, se estes tamén son demasiado lentos, dos heurísticos.

1.2. Recursos para o *bin packing problem*

Dispoñibles en Internet, podemos atopar numerosas ferramentas para o tratamento do BPP. Nesta sección realizamos unha breve introdución dunha pequena recompilación destes recursos, pero afondaremos en cada un deles no apartado correspondente (véxase a Táboa 1.1). Ademais, varias destas ferramentas serán empregadas co fin de ilustrar os diferentes contidos do presente traballo.

A primeira ferramenta, e a que máis utilizaremos, é o paquete `gbp` do software estatístico R. Os comandos e funcións que alberga permiten a resolución do BPP dende unha ata catro dimensións, así como a súa representación gráfica ou visualización das solucións. Por outra banda, na biblioteca BPPLIB (Delorme et al. 2018) (dispoñible en `http://or.dei.unibo.it/library/bpplib`) podemos atopar unha boa colección de códigos, conxuntos de datos (ou instancias) e aplicacións para a visualización do 1D-BPP. As restantes ferramentas organizámolas en tres grandes grupos:

- Códigos:
  - *Geeks for geeks* (`https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/`): códigos para o 1D-BPP.
  - Martin Broadhurst (`http://www.martinbroadhurst.com/bin-packing.html`): códigos para o 1D-BPP.
  - Google (`https://developers.google.com/optimization/bin/knapsack`): códigos para o 1D-BPP.
- Visualizadores:
  - `http://www.binpacking.4fan.cz`: Visualizador para o 1D-BPP e o 2D-BPP.
  - *Computer Science Field Guide* (`https://csfieldguide.org.nz/en/interactives/bin-packing/`): Visualizador para o 1D-BPP.
  - TwoBinGame (`https://gianlucacosta.info/TwoBinPack/`): Visualizador para o 2D-BPP.

  - *3D Bin Packing* (`http://www.3dbinbox.com/App/Views/index.html`): Visualizador para o 3D-BPP.

- Instancias:
  - *Roll Project* (`http://www.rollproject.org/one-dimensional-bin-packing-problems/`): Instancias para o 1D-BPP.

Recursos BPP			
	Códigos	Visualización	Instancias
`gbp`		×	
BPPLIB	×	×	×
Geek for geeks	×		
Martin Broadhurst	×		
Google	×		
"`4fan`"		×	
TwoBinGame		×	
CS Field Guide		×	
Roll Project			×

Táboa 1.1: Algúns recursos para o BPP dispoñibles en Internet.

Capítulo 2

Bin packing problem nunha dimensión (1D-BPP)

Este capítulo centrarase nas características máis notables do problema nunha soa dimensión, que tamén recibe o nome de "BPP clásico" e se codifica como 1/V/I/M segundo a tipoloxía de Dyckhoff (1990). Con afán de simplificar a lectura destas páxinas, referirémonos habitualmente a este problema coa notación 1D-BPP indicada anteriormente, onde 1D nos remite ao carácter unidimensional do problema, notación empregada tamén nalgunha das referencias bibliográficas consultadas (Escamilla et al. 2017). Abordaremos os recursos dispoñibles en Internet para tratar o 1D-BPP, dende códigos de distintos métodos de resolución ata aplicacións para visualizar os resultados. Ademais, introduciremos unha serie de conxuntos de datos que nos servirán para ilustrar o funcionamento dalgúns métodos de resolución que tamén explicaremos neste capítulo.

2.1. Definición e características

O 1D-BPP pode verse como o problema de empaquetar un conxunto de obxectos en recipientes cunha determinada capacidade sen que se supere esta e tratando de utilizar a menor cantidade posible de recipientes. Para familiarizar máis ao lector con este problema, comezaremos ilustrándoo con exemplos de situacións da vida cotiá:

Exemplo 1 (**ferretería**). *Un comercio de ferretería recibe pedidos de tubos de ferro por parte de varios clientes. Estes tubos deben satisfacer as seguintes medidas en metros: 4,3,3,2,2,2,1,1,1 e 1. O encargado da tenda ten á venda pezas enteiras de 5 metros de lonxitude, polo que deberá realizar cortes nestas para satisfacer as demandas. Pero o comerciante non quere desperdiciar moito ou ningún material, así que deberá buscar a mellor forma de cortar as pezas de 5 metros de xeito que obteña as seccións de tubo encargadas usando a menor cantidade de material posible (é dicir, de pezas enteiras).*

Exemplo 2 (**montacargas**). *Unha empresa de mantemento e limpeza de edificios conta cun montacargas para subir os produtos que precisa nas tarefas para as que é contratada a unha determinada altura ou andar dos inmobles. Nunha das encargas, necesita subir diversos vultos con distintos pesos, pero o montacargas soamente é capaz de levantar un determinado peso máximo. O encargado ten que subir todos estes produtos a un andar concreto do edificio de xeito que se realicen o menor número de ascensos/descensos do montacargas, pois este é un aparello bastante lento e cada operación require de gran cantidade de tempo. Polo tanto, deberá poñer os vultos no montacargas da mellor maneira posible respectando que o peso total non sexa superior ao máximo que pode levantar o aparello.*

Exemplo 3 (**combustible**). *Unha empresa comercializadora de combustibles recibe na súa planta de distribución distintos derivados de petróleo que son transportados en camións cisterna con capacidades*

variables. Na planta dispón de grandes depósitos para o almacenamento do combustible, todos eles do mesmo tamaño. A política da empresa é considerar o contido das cisternas como unha unidade indivisible, é dicir, que as cisternas se baleiren completamente nun só depósito, evitando situacións nas que un mesmo camión teña que repartir o seu contido entre dous depósitos ou máis. Debido ao custe de limpeza e tratamento destes, a dirección da empresa precisa almacenar todo o contido das cisternas usando o menor número de depósitos posible, sen exceder (como é de supoñer) a capacidade de ditos depósitos.

A característica común que comparten tanto os elementos coma os recipientes é ser descritos por unha única variable (lonxitude, peso ou volume), que denominaremos sen perda de xeneralidade *tamaño* no caso dos obxectos e *capacidade* no caso dos recipientes. Dito isto, cal é e como calculamos a solución, preferiblemente óptima, destes e outros problemas similares? A esta cuestión trataremos de dar resposta no que resta de capítulo. Neste punto, podemos dar entón unha definición formal do problema:

Definición 1 (1D-BPP). *Dados n obxectos ou elementos con tamaños t_j, $j \in \{1, \dots, n\}$, o problema 1D-BPP consiste en empaquetar todos os obxectos en recipientes ou contedores cunha mesma capacidade c, tratando de usar o menor número de recipientes posible.*

Exemplo 4. *Trasladando isto aos datos do Exemplo 1, temos entón:*

$$n = 10, \qquad c = 5, \qquad t_j \in \{4, 3, 3, 2, 2, 2, 1, 1, 1, 1\}.$$

Nalgunhas referencias bibliográficas (Martello e Toth 1990) indícase tamén unha cantidade inicial ou cota superior de contedores dispoñibles á hora de definir o problema, non sendo habitualmente o número óptimo de recipientes. Pola contra, para outros autores (por exemplo, Gupta e Ho 1999) esta cantidade non ten por que estar limitada superiormente, xa que o interese deste problema radica, ao fin e ao cabo, en minimizar o número de recipientes utilizados, de aí que se pase por alto dita cantidade. Por outra banda, Scholl et al. (1997) considera que poden distinguirse dúas versións deste problema:

- Minimizar o número de recipientes para unha capacidade individual c dada.
- Minimizar a capacidade de cada recipiente para unha cantidade de recipientes m dada.

A segunda versión é equivalente ao problema de programar n traballos diferentes que teñen tempos de operación t_j en m procesadores idénticos en paralelo co obxectivo de minimizar a capacidade de execución ou vida útil, tamén coñecida como *makespan*, que vén a ser o tempo de execución total. Este problema recibe o nome de *multiprocessor scheduling problem* (Alvim e Ribeiro 2004), pero tamén se coñece como *dual bin packing problem* (DBP), denominación que fai alusión ao seu carácter dual respecto do BPP clásico (1D-BPP)(Alvim et al. 2004).

2.1.1. Variantes e problemas relacionados

Neste traballo soamente repararemos na primeira das versións indicadas por Scholl et al. (1997), é dicir, tratar de empaquetar todos os obxectos en recipientes de igual capacidade, sendo o obxectivo minimizar o número de recipientes necesarios e suxeito á restrición que impón a capacidade de cada un deles. Mais convén destacar que existe diversa literatura que trata diferentes modificacións na definición orixinal do 1D-BPP ou outros problemas relacionados con el:

- O 1D-BPP garda unha estreita relación con outros problemas de corte e empaquetamento como o *cutting stock problem* (CSP), o *knapsack problem* (KP) e o *subset-sum problem* (SSP), sobre os cales afondaremos máis adiante (Delorme et al. 2016, Martello e Toth 1990). No que se refire a outros problemas de optimización, Valério de Carvalho (2002) propón unha definición do 1D-BPP como un caso particular do *problema de ruta de vehículos* (ou *vehicle routing problem*, VRP). O obxectivo neste problema é achar a ruta ou rutas óptimas que, xeralmente, minimicen unha determinada función (custo, distancia, tempo...) atendendo a determinadas restricións (como

pasar unha soa vez por cada lugar, por exemplo) e volvendo ao rematar a ruta ao punto de saída. Visto deste modo, os elementos e os recipientes do 1D-BPP corresponderíanse cos clientes e os vehículos do VRP, respectivamente. Como o VRP se define nun grafo, nos seus nodos estarían os clientes e os vehículos desprazaríanse sobre as arestas. O valor da carga do vehículo que demanda o cliente corresponderíase co tamaño do elemento e a capacidade dos vehículos viría a ser a capacidade dos recipientes.

- Podemos ter variantes do BPP clásico modificando as características dos tamaños dos elementos. Por exemplo, cando estes forman unha *secuencia divisible*, é dicir, a secuencia de tamaños

$$t_1 > t_2 > ... > t_i > t_{i+1} > ... > t_n$$

verifica que para todo j, t_{j+1} divide de xeito exacto a t_j. O par formado pola lista de tamaños e a capacidade c pode denominarse *debilmente divisible* se a lista é unha secuencia divisible, ou *fortemente divisible* cando ademais o elemento t_1 (o de maior tamaño) divide de xeito exacto á capacidade c. A secuencia divisible de tamaños aparece con frecuencia en certas aplicacións, como o almacenamento de memoria en dispositivos informáticos, onde ditos tamaños adoitan ser potencias de 2 (Coffman et al. 1987). Por outra banda, se os elementos do problema teñen tamaños variables, entón estamos ante unha versión chamada *open-end BPP* (1D-OEBPP, ou simplemente OEBPP), cuxa aplicación resulta útil cando se realizan pagamentos e cambios de divisa (Mohamed et al. 2016).

- Existen variantes do 1D-BPP en función da cantidade de elementos que hai para empaquetar. Se a cada recipiente se poden asignar como moito k recipientes, daquela estamos ante o *k-item BPP* (Wäscher et al. 2007). Se se permite a entrada e saída de elementos, tense o 1D-BPP dinámico, que se aplica habitualmente a dispositivos de almacenamento (memorias USB, discos duros externos...), por exemplo (Coffman et al. 1987).

- Tamén podemos atopar variacións da función obxectivo. Unha variante é o problema de maximizar o número de elementos empaquetados: dados m recipientes con capacidade c cada un deles, o obxectivo é empaquetar tantos elementos nos recipientes dados como sexa posible (Coffman et al. 1987). En cambio, podemos ter un obxectivo que sexa maximizar o número de recipientes usados para empaquetar todos os elementos ou, equivalentemente, minimizar o número de elementos en cada recipiente, suxeito en calquera caso a unha das seguintes dúas restricións:

  1. Unha vez se pecha un recipiente, non se pode empaquetar nel ningún elemento sen asignar.
  2. Non se pode inicializar un novo recipiente se o obxecto co que se estea nese momento ten cabida no recipiente que xa está sendo usado.

  Este problema, con calquera das restricións, denomínase *maximum resource BPP*. Esta variante do 1D-BPP resulta interesante para unha das partes involucradas nel cando o recurso se debe comprar. Por exemplo, cando se contrata unha empresa para transportar elementos en camión (o recurso) dun lugar a outro. Ao cliente interésalle que o número de viaxes que realice sexa o menor posible (que se correspondería co propio 1D-BPP onde os recipientes son as viaxes realizadas polo camión), mentres que á empresa lle interesa que o número de viaxes sexa o maior posible (Boyar et al. 2006).

2.1.2. Complexidade computacional

Antes de tratar a complexidade do 1D-BPP, repasaremos as definicións das distintas clases de complexidade:

- P: Un problema pertence a esta clase se existe un algoritmo que resolve calquera exemplo de dito problema en tempo polinomial.

- NP: Abrangue os problemas para os que existe un algoritmo que verifique unha solución dun exemplo de dito problema en tempo polinomial.
- *NP-completo*: Calquera problema NP se pode reducir a un desta clase en tempo polinomial. Son os máis difíciles da clase NP.
- *NP-duro* ou *NP-difícil*: Calquera problema da clase NP se pode reducir a un desta clase en tempo polinomial, diferenciándose dos NP-completos en que un problema NP-duro non tem por que ser da clase NP.

O problema de atopar o empaquetamento óptimo no 1D-BPP é NP-duro ou NP-difícil. A estratexia que se segue en Martello e Toth (1990) para ver que o 1D-BPP é NP-duro é probar que a súa "versión de recoñecemento" R(1D-BPP) é NP-completo. A partir diso, pódese enunciar o seguinte resultado (Martello e Toth 1990):

Teorema 1. *O problema 1D-BPP é NP-duro en sentido forte.*

2.2. Modelos de programación linear para diferentes variantes do problema

Nesta sección presentamos os modelos de programación linear que permiten a resolución exacta do 1D-BPP, así como doutras variantes ou casos particulares do citado problema.

2.2.1. Modelo básico

O 1D-BPP pode formularse como problema de programación linear establecendo previamente os seguintes parámetros e variables. En primeiro lugar, dada unha cota superior m no número de recipientes, establécense variables binarias y_i ($i \in \{1, \dots, m\}$) que tomarán o valor 1 se o recipiente i é usado e 0 en caso contrario. Ademais, dado o número n de obxectos que hai que empaquetar, establécense tamén as variables binarias x_{ij} ($i \in \{1, \dots, m\}$, $j \in \{1, \dots, n\}$), que tomarán o valor 1 se o obxecto j é asignado ao recipiente i e 0 en caso contrario. Daquela, a forma final do problema será (Martello e Toth 1990):

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{i=1}^{m} y_i \\
\text{suxeito a} \quad & \sum_{j=1}^{n} t_j x_{ij} \leq c y_i, \quad i \in \{1, \dots, m\}, \\
& \sum_{i=1}^{m} x_{ij} = 1, \qquad j \in \{1, \dots, n\}, \\
& y_i \in \{0, 1\}, \qquad i \in \{1, \dots, m\}, \\
& x_{ij} \in \{0, 1\}, \qquad i \in \{1, \dots, m\}, \;\; j \in \{1, \dots, n\}.
\end{aligned}
\tag{2.1}
$$

Como xa se comentou, o obxectivo é minimizar o número de recipientes a usar, que se expresará como a suma das y_i: $\sum_{i=1}^{m} y_i$. Suporase que os pesos ou tamaños t_j son enteiros positivos e asúmese sen perda de xeneralidade que a capacidade c é tamén un enteiro positivo e que $t_j \leq c \; \forall j \in \{1, \dots, n\}$. A primeira e principal restrición do problema é que a suma dos tamaños dos obxectos t_j non supere a capacidade c do recipiente i ao que foron asignados:

$$\sum_{j=1}^{n} t_j x_{ij} \leq c y_i.$$

A segunda das restricións do problema impón que cada elemento soamente poida estar nun único recipiente, o que traducido a linguaxe matemática é:

$$\sum_{i=1}^{m} x_{ij} = 1.$$

Exemplo 5. *Botando man da linguaxe AMPL, o solver Gurobi (do que falaremos máis adiante) e dos datos do Exemplo 1, aos que engadimos unha cota superior $m = 6$, ilustramos a resolución do 1D-BPP mediante a formulación arriba proposta. O código empregado para programar este problema atópase no correspondente apéndice deste traballo e a solución podemos vela na Táboa 2.1. Á vista dos resultados, todas as pezas foron cortadas (é dicir, todos os obxectos empaquetados) e resulta sinxelo comprobar que non quedou material desperdiciado (non se deixou espazo desocupado).*

1D-BPP Exemplo 1										
Nº óptimo de tubos	4									
Pezas	1	2	3	4	5	6	7	8	9	10
Lonxitudes	4	3	3	2	2	2	1	1	1	1
Tubo asignado	4	2	3	1	3	1	4	1	2	2

Táboa 2.1: Solución do 1D-BPP para a instancia do Exemplo 1.

2.2.2. Variantes e casos particulares

Unha variante do 1D-BPP procede do ámbito da industria papeleira, onde os termos recipiente e empaquetar se adoitan substituír por *rolos* e *cortar*, respectivamente. Neste caso, o obxectivo é similar ao do 1D-BPP, pero hai un cambio: non teremos unha cantidade de elementos a empaquetar, senón que agora temos varios tipos ou patróns de obxectos, de cada un dos cales hai que realizar un certo número de copias segundo sexan requeridas. Así, se temos u tipos de elementos, cada un co seu tamaño ou peso t_j e unha demanda d_j ($j \in \{1, \dots, u\}$), e un número ilimitado de rolos cunha capacidade c, o obxectivo está en producir d_j copias de cada tipo j de elementos usando o mínimo número de rolos de xeito que non se exceda a capacidade destes. O problema así presentado denomínase *cutting stock problem* (CSP). Tomando unha cota superior m no numero de rolos e as variables binarias y_i definidas para o 1D-BPP e establecendo ξ_{ij} como o número de elementos do tipo j cortados no rolo i ($i \in \{1, \dots, m\}$, $j \in \{1, \dots, u\}$), podemos formular o CSP como un problema de programación linear enteira do xeito seguinte (Delorme et al. 2016):

$$\begin{aligned}
\text{minimizar} \quad & \sum_{i=1}^{m} y_i \\
\text{suxeito a} \quad & \sum_{j=1}^{u} t_j \xi_{ij} \leq c y_i, \qquad i \in \{1, \dots, m\}, \\
& \sum_{i=1}^{m} \xi_{ij} = d_j, \qquad j \in \{1, \dots, u\}, \\
& y_i \in \{0, 1\}, \qquad i \in \{1, \dots, m\}, \\
& \xi_{ij} \in \mathbb{Z}^+ \cup \{0\}, \qquad i \in \{1, \dots, m\}, \;\; j \in \{1, \dots, u\}.
\end{aligned} \tag{2.2}$$

Podemos ver a relación entre 1D-BPP e CSP como segue: o 1D-BPP é un caso especial do CSP se consideramos que as demandas d_j toman o valor 1 para todo j e, reciprocamente, CSP pode verse como un 1D-BPP no cal existen d_j copias de cada elemento do tipo j.

Exemplo 6. *Para ilustrar a relación entre o 1D-BPP e o CSP, botamos man de novo dos datos do Exemplo 1. Resulta sinxelo ver que estes datos se adecúan bastante ben ao CSP, pois chega con decatarse de que contamos con $u = 4$ tipos de pezas de tamaños $t_j \in \{1, 2, 3, 4\}$ e temos tamén as demandas d_j, $j = 1, \ldots, 4$.*

Resolveremos o problema con dous conxuntos de demandas, o primeiro con $d_j = 1$ para todo j e o segundo con d_j tomando o valor do número pezas de cada tipo que son encargadas. Á vista da Táboa 2.2, tomando as demandas con valor 1, estamos ante un 1D-BPP con soamente 4 elementos, mentres que tomando o outro conxunto de demandas, temos a mesma solución (salvo intercambio de recipientes/tubos) que se obtivo do 1D-BPP (véxase a Táboa 2.1), o que era de esperar como xa adiantamos antes. □

CSP Exemplo 1				
Tipos	"4m"	"3m"	"2m"	"1m"
Lonxitudes	4	3	2	1
Demanda	1	1	1	1
Nº óptimo de tubos			2	
Tubo asignado	1	2	2	1
Demanda	1	2	3	4
Nº óptimo de tubos			4	
Tubo(s) asignado(s)	2	3 e 4	1, 1 e 4	1, 2, 3 e 3

Táboa 2.2: Solución do CSP para a instancia do Exemplo 1.

Volvamos ao 1D-BPP e imaxinemos agora que soamente contamos cun recipiente para realizar o empaquetamento, situación que imposibilita que poidan ser escollidos todos os n obxectos. Neste caso, teremos que facer unha selección destes (un subconxunto do total) atendendo á utilidade, proveito ou valor que teña ou se lle asigne cada un deles, sendo o obxectivo disto que o valor total sexa o máximo posible. Este é o chamado *problema da mochila binario* (en inglés, $0-1$ *knapsack problem* ou *binary knapsack problem* (KP)), Neste caso, dado o número n de elementos que potencialmente se poden escoller, establécense as variables binarias x_j ($j \in \{1, \ldots, n\}$) que tomarán o valor 1 se o elemento j resulta seleccionado e 0 en caso contrario. Así, a formulación completa do problema KS é a seguinte (Martello e Toth 1990):

$$\begin{aligned} \text{maximizar} \quad & \sum_{j=1}^{n} p_j x_j \\ \text{suxeito a} \quad & \sum_{j=1}^{n} t_j x_j \leq c, \\ & x_j \in \{0,1\}, \qquad j \in \{1,\dots,n\}. \end{aligned} \tag{2.3}$$

A capacidade do recipiente (neste caso, a mochila) denotámola tamén por c, unha medida que non pode excederse, sendo a principal restrición do problema. Polo tanto, a suma dos tamaños ou pesos t_j dos obxectos será menor ou igual a dita cantidade: $\sum_{j=1}^{n} t_j x_j \leq c$. Ademais, dado que cada obxecto ten unha utilidade ou proveito p_j, a escolla busca que o valor total do contido da mochila sexa o maior posible, polo que se maximizarán a suma dos proveitos dos elementos escollidos: $\sum_{j=1}^{n} p_j x_j$. Suporemos tamén sen perda de xeneralidade que as cantidades t_j, p_j e c toman valores enteiros non negativos, $\sum_{j=1}^{n} t_j > c$ e $t_j < c \; \forall j \in \{1,\dots,n\}$.

Se a utilidade ou proveito de cada elemento se fai coincidir co tamaño do propio elemento, isto é, $p_j = t_j$, temos un caso particular do KS no que tratamos de seleccionar os obxectos cuxa suma de pesos non exceda a capacidade da mochila e de xeito que dita suma sexa o maior posible. Entón, este novo problema podemos velo do seguinte xeito: dado un conxunto de n números $\{t_1,\dots,t_n\}$ e outro número c, existirá un subconxunto deses números cuxa suma non exceda ou sexa exactamente igual a c?. Este é o chamado *problema da suma de subconxuntos* (en inglés, *subset-sum problem* (SSP)), cuxa formulación se obtén de xeito inmediato a partir da do KS soamente con tomar $p_j = t_j$ e tendo en conta as correspondentes consideracións (Martello e Toth 1990):

$$\begin{aligned} \text{maximizar} \quad & \sum_{j=1}^{n} t_j x_j \\ \text{suxeito a} \quad & \sum_{j=1}^{n} t_j x_j \leq c, \\ & x_j \in \{0,1\}, \qquad j \in \{1,\dots,n\}. \end{aligned} \tag{2.4}$$

O SSP é un problema co que nos atopamos en multitude de ocasións. Por exemplo, se pensamos en calquera sistema monetario, formado por moedas e billetes con distintos timbres, decatámonos que o pagamento dun importe xusto (ou a devolución dun cambio, no seu caso) non é mais que buscar unha combinación de moedas e billetes cuxos valores sumen o importe que corresponda.

Agora ben, se temos a posibilidade de usar máis dun subconxunto (equivalentemente, máis dunha mochila) e repetir o proceso de escolla para cada un deles (é dicir, que a suma non exceda unha cantidade dada) co fin de todos os números (obxectos) estean asignados a un subconxunto (mochila), estaremos de novo no problema do que partimos: o 1D-BPP. Daquela, o 1D-BPP pode interpretarse como un SSP múltiple, no que todos os contedores teñen a mesma capacidade, hai que seleccionar todos os obxectos e o obxectivo é minimizar o número de recipientes. Deste xeito, o 1D-BPP pode incluírse no ámbito dos problemas da mochila (*knapsack problems*).

Exemplo 7. *Para ilustrar os problemas KS e SSP, consideraremos de novo o Exemplo 1 e asignando para o primeiro a todos os obxectos o mesmo proveito $p_j = 1$, $j = 1,\dots,10$, e para o segundo simplemente tomaremos os seus tamaños, obtendo as solucións que se mostran na Táboa 2.3.*

KS/SSP Exemplo 1										
Pezas	1	2	3	4	5	6	7	8	9	10
Lonxitudes	4	3	3	2	2	2	1	1	1	1
Demanda KS	1	1	1	1	1	1	1	1	1	1
Selección KS	-	-	-	-	-	-	×	×	×	×
Valor óptimo KS						4				
Demanda SSP	4	3	3	2	2	2	1	1	1	1
Selección SSP	-	-	-	-	-	×	×	×	-	×
Valor óptimo SSP						5				

Táboa 2.3: Solucións do KS e SSP do Exemplo 1.

2.3. Recursos para o 1D-BPP

Nesta sección presentamos de maneira máis pormenorizada os datos que empregaremos para ilustrar os distintos métodos de resolución do 1D-BPP que expliquemos e outros recursos para este problema.

2.3.1. Instancias

Os conxuntos de datos que se usan para describir o 1D-BPP adoitaremos denominalos *instancias* ao longo deste traballo. Están formadas polo número de elementos que se desexan empaquetar n, a capacidade dos recipientes c e os tamaños ou pesos t_j.

Para ilustrar os métodos que expliquemos, introduciremos aquí unha serie de conxuntos de instancias conforme á nomenclatura habitual da bibliografía consultada. O primeiro conxunto de datos que empregaremos é de creación propia e correspóndense co Exemplo 1 exposto anteriormente, que consta dos seguintes valores:

$$n = 10, \qquad c = 5, \qquad t_j \in \{4, 3, 3, 2, 2, 2, 1, 1, 1, 1\}.$$

Esta instancia foi creada a propósito para que o empaquetamento óptimo non deixase espazo libre en ningún recipiente, sendo a suma dos t_j múltiplo da capacidade c.

En Gupta e Ho (1999) ilústrase a efectividade do método heurístico MBS (que veremos no correspondente apartado) con cinco "problemas difíciles" que os autores recolleron de referencias bibliográficas anteriores. Destes cinco conxunto de datos, escollemos tres instancias, que denominamos GuHo, e recollemos detalladamente na Táboa 2.4 e representadas na Figura 2.3. Decidimos incluílas para empregalas cando a nosa instancia de referencia (Exemplo 1) non permita mostrar algunha característica concreta dos métodos que expliquemos.

Os restantes conxuntos de datos son tomados da biblioteca BPPLIB (da que falaremos a continuación):

- Tomaremos unha instancia de cada colección de Falkenauer: "Falkenauer_u120_00" das uniformes e "Falkenauer_t60_00" dos tripletes.

Instancia	n	c	t_j
GuHo1	6	100	60,50,30,20,20,20
GuHo4	15	17	17,9,7,6,5,5,4,4,4,4,4,4,4,4,4
GuHo5	10	61	44,24,24,22,21,17,8,8,6,6

Táboa 2.4: Instancias de Gupta e Ho (1999).

- Das coleccióńs Scholl, tomamos a instancia "N4C1W1_H" do primeiro grupo e"N4W3B1R5" do segundo.
- Das instancias aleatorias tamén tomamos un par: "BPP_100_120_0.2_0.8_8" e "BPP_200_300_0.2_0.8_0", en cuxos nomes xa van explícitos n, c e a proporción dos valores mínimo e máximo dos tamaños.

Escollemos instancias das coleccióńs de Falkenauer e Scholl por ser utilizadas nalgunhas das referencias bibliográficas consultadas para a realización deste traballo. Para todas as instancias seleccionadas, recollemos os seus principais datos na Táboa 2.5.

Nome	**Orixe**	n	c	t_j
Exemplo 1	propia	10	5	1 a 4
GuHo1	Gupta e Ho	6	100	20 a 60
GuHo4	Gupta e Ho	15	17	4 a 17
GuHo5	Gupta e Ho	10	61	6 a 44
Falkenauer_u120_00	Falkenauer (uniforme)	120	150	20 a 98
Falkenauer_t60_00	Falkenauer (tripletes)	60	1000	251 a 495
N4C1W1_H	Scholl et al.	500	100	1 a 100
N4W3B1R5	Scholl et al.	500	1000	114 a 168
BPP_100_120_0.2_0.8_8	aleatoria	100	120	24 a 96
BPP_200_300_0.2_0.8_0	aleatoria	200	300	61 a 240

Táboa 2.5: Descrición das instancias.

Como xa adiantamos, a primeira ferramenta da que falaremos é a biblioteca BPPLIB (Delorme et al. 2018), dispoñible na web `http://or.dei.unibo.it/library/bpplib`. Trátase dunha biblioteca dedicada ao 1D-BPP e ao CSP, na que se dispón de referencias bibliográficas, códigos (ou ligazóns a eles) para a resolución exacta destes problemas, instancias e unha aplicación para visualizar (e resolver) o 1D-BPP. Nela atopamos diversas coleccióńs de instancias, clasificadas en catro categorías:

- **Instancias da literatura**: Trátase de conxuntos de datos propostos por varios autores nos seus artigos de investigación e empregados en numerosos traballos posteriores:
  - **Falkenauer**: Datos elaborados por Emanuel Falkenauer en *A hybrid grouping genetic algorithm for bin packing* (1996). Ditos datos constitúense en dúas coleccións de 80 instancias cada unha: Falkenauer U, cos tamaños dos elementos uniformemente distribuídos, $n \in \{120, 250, 500, 1000\}$ e $c = 150$; e Falkenauer T, instancias máis complexas onde se inclúen os chamados *tripletes*, é dicir, grupos de tres elementos (un grande e dous pequenos) que teñen que ser asignados no mesmo recipiente en calquera solución óptima, sendo $n \in \{60, 120, 249, 501\}$ e $c = 1000$.
  - **Scholl**: Instancias usadas por Scholl, Klein e Jürgens para probar o método BISON (Scholl et al. 1997), do que falaremos no seu momento. Estes autores botaron man de tres grupos de instancias con tamaños de obxectos distribuídos uniformemente: 720 instancias fáciles con $n \in \{50, 100, 200, 500\}$ e $c \in \{100, 120, 150\}$, 480 de dificultade media con $n \in \{50, 100, 200, 500\}$ e $c = 1000$ e 10 instancias difíciles, que destacan por ter capacidades moi grandes ($n = 200, c = 100000$).
  - **Wäscher**: Son 17 instancias moi difíciles con $n \in [57, 239]$ e $c = 10000$, descritas por G. Wäscher e T. Gau no artigo *Heuristics for the integer one-dimensional cutting stock problem: a computational study* (1996).
  - **Schwerin**: Dous conxuntos de 100 instancias fáciles con $n = 100, 120$ e $c = 1000$, expostas en *The bin-packing problem: a problem generator and some numerical experiments with FFD packing and MTP* (1997) de P. Schwerin and G. Wäscher.
  - **Schoenfield**: 28 instancias difíciles con $n = 160, 180, 200$ e $c = 1000$, propostas por J.E. Schoenfield en *Fast, exact solution of open bin packing problems without linear programming* (2002).
- **Instancias xeradas aleatoriamente**. Estas foron creadas para avaliar o comportamento de métodos de resolución exactos (Delorme et al 2016). En concreto, temos a posibilidade de elixir entre 3840 conxuntos de datos, con diferentes valores para n (50, 100, 200, 300, 400, 500, 750 e 1000) e c (50, 75, 100, 120, 125, 150, 200, 300, 400, 500, 750 e 1000). Os valores dos t_j de cada instancia foron xerados tendo en conta o correspondente valor de c, pois están limitados por valores mínimo e máximo que dependen de dita cantidade: $0.1c$ ou $0.2c$ e $0.7c$ e $0.8c$, respectivamente.
- **Instancias difíciles**. Son dous conxuntos de instancias con valores de n lixeiramente superiores a 200, 400, 600,800 e 1000 e capacidades $c \in \{2500, 10000, 20000, 40000, 80000\}$.
- **Instancias GI**. Son 240 instancias xeradas aleatoriamente con valores altos de c (500000 e 1500000) e foron propostas por Gschwind e Irnich (as súas iniciais deron lugar ao nome) en *Dual inequalities for stabilized column generation revisited* (2016).

Outras coleccións de instancias podemos atopalas na web *Roll Project* (`http://www.rollproject.org/one-dimensional-bin-packing-problems/`). En concreto, trátase de dous conxuntos (*Problem Set A* e *Problem Set B*), con 15830 e 3968 instancias, respectivamente. 1370 destas instancias foron tomadas dos conxuntos xa mecionados de Falkenauer e Scholl co fin de establecer uns parámetros (tabulados na propia web) para a construción das demais. Unha característica destes datos é o feito de que os elementos están clasificados en catro clases atendendo ao seu tamaño: pequenos ($t_j \leq c/4$), medianos ($c/4 < t_j \leq c/3$), grandes ($c/3 < t_j \leq c/2$) e enormes ($t_j > c/2$). Ambos conxuntos de instancias atópanse comprimidos en ficheiros para lectura en linguaxe SQL.

2.3.2. Códigos

No que respecta a códigos que resolven o 1D-BPP, en BPPLIB atopamos os seguintes:

- **MTP**. Inclúese o código programado en linguaxe FORTRAN deste método de ramificación e acotación recollido en Martello e Toth (1990), do que falaremos en profundidade no seguinte capítulo.
- **BISON**. Algoritmo baseado no anterior, programado en Pascal e explicado en Scholl et al. (1997). Indícase que pode ser obtido dos propios autores vía correo electrónico.
- **CVRPSEP**. Proporciónase unha ligazón ao código C deste método exposto por J. Lysgaard en *CVRPSEP: A package of separation routines for the capacitated vehicle routing problem* (2003). Este algoritmo foi obtido a partir do MTP.
- **SCIP-BP**. Inclúese a ligazón a este algoritmo *branch-and-price*, un tipo de métodos xurdido da combinación de ramificación e acotación coa técnica de xeración por columnas. SCIP-BP está baseado na regra de ramificación explicada por D.M. Ryan e B.A. Foster no artigo *An integer programming approach to scheduling* (1981).
- **DPFLOW**. Código C++ deste modelo formulado para ser resolto como un problema de programación linear enteira, recollido por H. Cambazard e B. O'Sullivan en *Propagating the bin packing constraint using linear programming* (2010).

Na web *Geeks for geeks* (`https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/`) aparece un apartado dedicado aos algoritmos aproximados de resolución do 1D-BPP: NF, FF, FFD e BF, que se verán máis adiante. Para os catro, ofrécesenos o código en linguaxe C++ e, a maiores, en Java para o NF. Ademais, é posible editar e executar os códigos dende a propia páxina web.

Na web de Martin Broadhurst (`http://www.martinbroadhurst.com/bin-packing.html`) atopamos unha pequena introdución ao 1D-BPP, así como a implementación de sete algoritmos aproximados en linguaxe C: NF, FF, FFD, BF, BFD, WF e WFD, dos que falaremos no correspondente apartado do seguinte capítulo.

Entre as ferramentas do buscador Google, atopamos unha sección dentro da área de optimización dedicada ao problema da mochila, que vimos era un caso particular do 1D-BPP. Alí temos á nosa disposición códigos nas linguaxes Python e Java de dito problema (dispoñible en `https://developers.google.com/optimization/bin/knapsack`).

2.3.3. Visualización do problema

Na web `http://www.binpacking.4fan.cz` podemos visualizar paso a paso a execución dalgúns algoritmos aproximados. Para visualizar o BPP unidimensional podemos escoller entre un conxunto de datos propio (indicando n, c e t_j) ou aleatorio (onde podemos indicar tamaños mínimo e máximo dos elementos, capacidade do recipiente e número de elementos) e tamén indicar que algoritmo queremos, a escoller entre versión *on-line* (NF, FF ou BF) ou *off-line* (NFD, FFD ou BFD), conceptos que xa explicaremos máis adiante. Durante a visualización, tamén pode regularse a velocidade de asginación de elementos. Na Figura 2.1 móstrase a execución da asignación de elementos dun 1D-BPP con datos aleatorios, un total de $n = 200$ elementos con tamaños t_j variando entre 10 e 100 unidades que se van colocando en recipientes de capacidade $c = 300$, usando o algoritmo *off-line* FFD.

Dende a biblioteca BPPLIB podemos descargar unha aplicación (dispoñible para Windows e Linux) para resolver de xeito interactivo o 1D-BPP con datos aleatorios ou introducidos polo usuario: **BppGame**. Na web do seu creador (`http://gianlucacosta.info/BppGame/`) pode accederse a máis información sobre a aplicación. O funcionamento é sinxelo: tras introducir os datos do problema ou elixir un aleatorio, o usuario pode "xogar", é dicir, pode desprazar os elementos da parte dereita e colocalos na posición que corresponda da parte esquerda para realizar os empaquementos que considere.

Na Figura 2.2 móstrase a visualización dun exemplo aleatorio do 1D-BPP. Esta aplicación serviranos ao longo do traballo para ilustrar instancias con poucos elementos e empaquetamentos que usen poucos recipientes, coma aquelas instancias con menos elementos que empregaremos para ilustrar os métodos (véxase a Figura 2.3[1]).

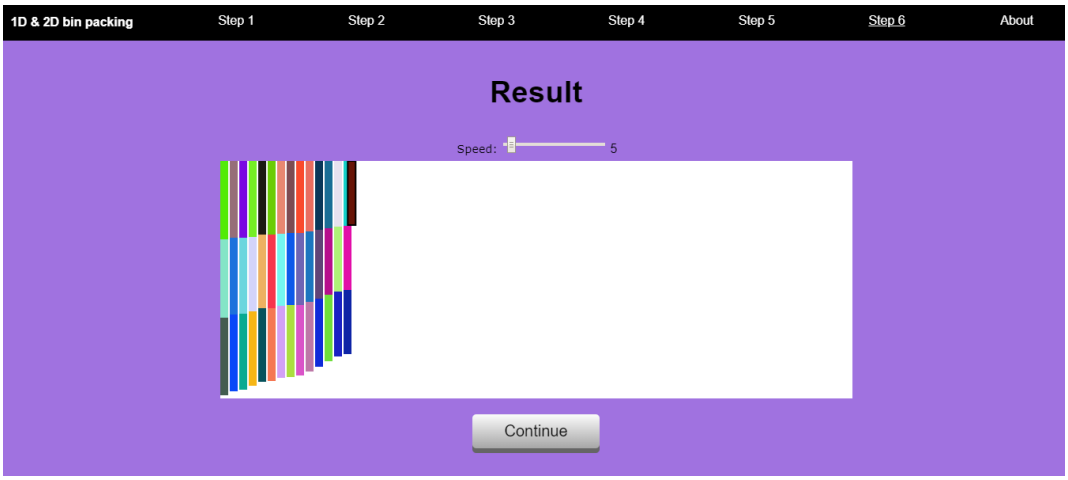

Figura 2.1: Visualización (en execución) da asignación de elementos a recipientes na web `http://www.binpacking.4fan.cz`.

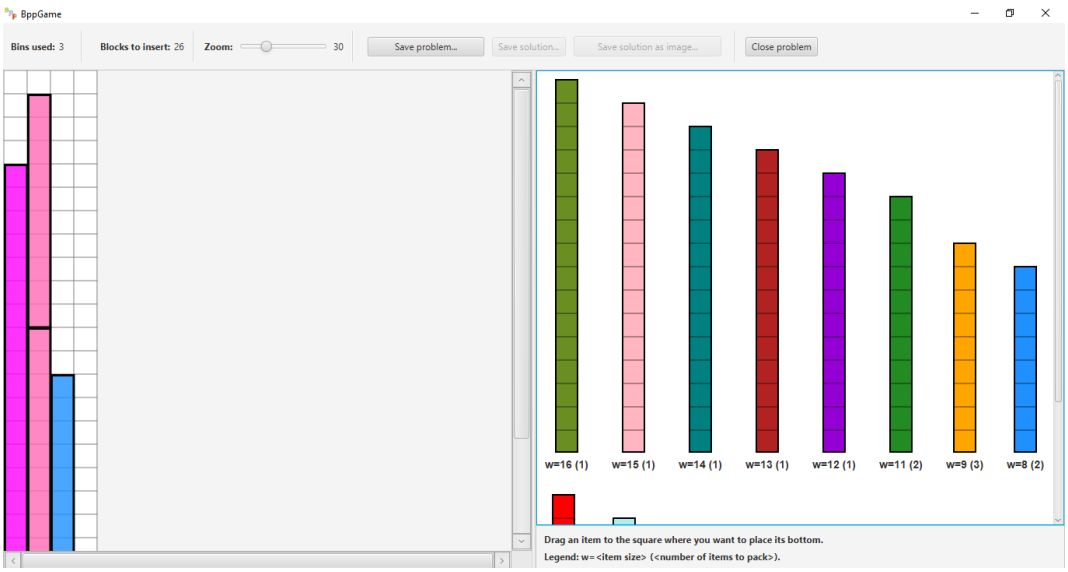

Figura 2.2: Visualización dun exemplo aleatorio en BppGame.

[1]Represéntanse as instancias do Exemplo 1 (arriba esquerda), GuHo1 (arriba dereita), GuHo4 (abaixo esquerda) e GuHo5 (abaixo dereita) en BppGame. Ao pé de cada elemento indícanse o tamaño e a cantidade de copias (elementos do mesmo tamaño) de cada un (entre parénteses). Os tamaños de GuHo1 aparecen divididos por 10 para unha representación máis manipulable e empregáronse distintas medidas de *zoom* para dar cabida nun espazo similar a todas as instancias.

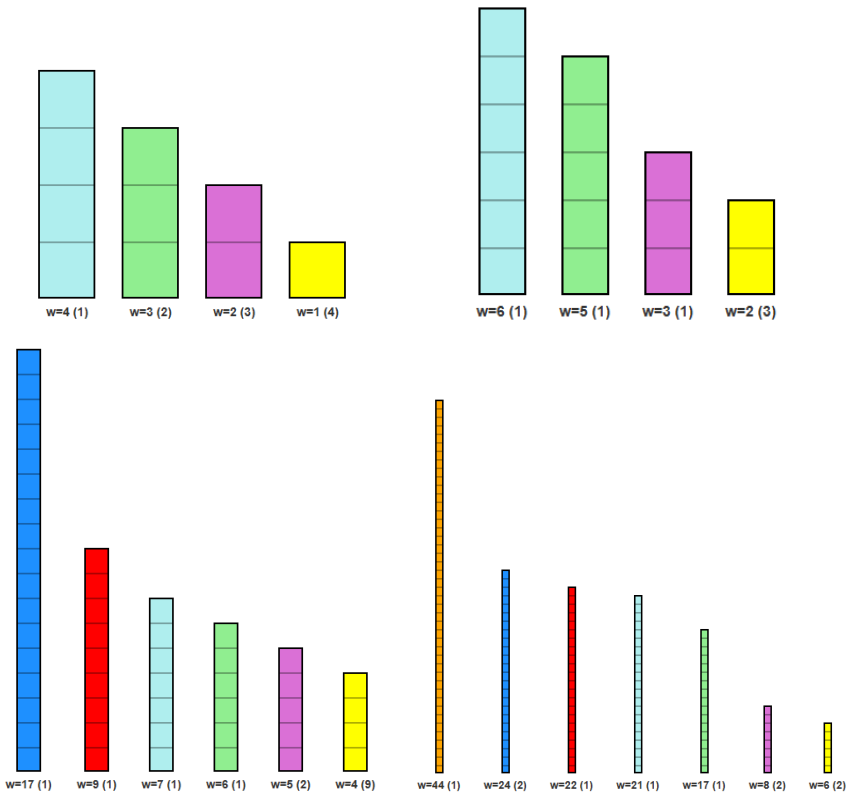

Figura 2.3: Representación das instancias do Exemplo 1 e as GuHo.

Na web *Computer Science Field Guide* (`https://csfieldguide.org.nz/en/interactives/bin-packing/`) atopamos outro visualizador para o 1D-BPP (véxase a Figura 2.4). Ao igual que nas aplicacións xa vistas, nesta tamén podemos xerar problemas aleatorios ou introducir os nosos propios datos. O funcionamento volve ser sinxelo: débense arrastrar os segmentos azuis (elementos) ata os grises (recipientes), podendo engadir tantos destes últimos como consideremos necesarios.

Figura 2.4: Visualización dun exemplo aleatorio en *Computer Science Field Guide*.

2.3.4. Resultados experimentais

Unha vez presentadas as instancias que empregaremos e demais recursos dispoñibles para o 1D-BPP, imos realizar a continuación unha primeira proba coas mesmas, resolvendo o 1D-BPP como problema de programación linear enteira coa formulación (2.1). Como xa fixemos cando describimos o problema e tratamos os casos particulares deste (CSP, KS e SSP), botaremos man da linguaxe de programación AMPL para implementar o código que despois enfrontamos ao *solver* Gurobi, un motor de optimización computacional que permite a resolución de problemas de programación linear, linear enteira mixta, cuadrática, cuadrática enteira mixta... Dada a cantidade de variables e restricións que se crean polo gran número de datos dalgunhas instancias, usaremos a aplicación cliente–servidor NEOS Server, que nos permite acceso a unha libraría de máis de 60 solvers (entre eles o citado Gurobi), para resolver o 1D-BPP con cada unha das instancias. Na Táboa 2.6 recóllense os resultados obtidos. Polo motivo antes citado da gran cantidade de datos, soamente indicamos o número óptimo de recipientes para cada instancia e non a disposición dos elementos nos recipientes. Ademais, engadimos o tempo de computación que calculamos en base aos tempos de inicio e final que se nos proporcionan en NEOS Server. Este tempo de cálculo está moi emparellado co número de elementos dos que conste a correspondente instancia, tendendo a ser alto se dita cantidade tamén o é, aínda que non é o único aspecto que pode enlentecer o cómputo. Observamos que no caso do Exemplo 1 e das instancia extraídas de Gupta e Ho (1999), os cálculos son case inmediatos ao ter un número reducido de datos. O caso contrario atopámolo na do primeiro conxunto de Scholl, na que se superaron as tres horas e media de tempo de cálculo. Esta tardanza no cómputo pode deberse ao feito de que os tamaños dos elementos máis grandes desta instancia coinciden ou distan poucas unidades do valor da capacidade do recipiente. A isto hai que unir a existencia de tamaños repetidos, o que axudaría a facer máis lento o proceso ao ter que explorarse as "mesmas" combinacións de elementos en varias ocasións, sen que necesariamente fosen as óptimas.

Instancia	**Nº óptimo**	**Tempo de cálculo**
Exemplo 1	4	5 s
GuHo1	2	5 s
GuHo4	5	10 s
GuHo5	3	15 s
Falkenauer_u120_00	48	40 s
Falkenauer_t60_00	20	2 min 30 s
N4C1W1_H	251	3 h 40 min 33 s
N4W3B1R5	71	6 min 26 s
BPP_100_120_0.2_0.8_8	53	25 s
BPP_200_300_0.2_0.8_0	115	1 min 41 s

Táboa 2.6: Número óptimo de recipientes para a formulación (2.1) do 1D-BPP.

2.4. Outros métodos de resolución do 1D-BPP

Atendendo á clasificación de métodos de resolución feita no Capítulo 1, expoñemos a continuación unha pequena mostra de métodos empregados para resolver o 1D-BPP.

2.4.1. Métodos exactos

Ademais dos modelos de programación matemática, existen determinados algoritmos que permiten obter unha solución exacta para o 1D-BPP. Martello e Toth (1990) e Scholl et al. (1997) coinciden en afirmar que, en contraste con métodos aproximados ou heurísticos, hai poucas referencias a algoritmos exactos. Precisamente nestas dúas obras preséntansenos senllos métodos que devolven a solución exacta, clasificados dentro dos *algoritmos de ramificación e acotación (ou poda)* (*branch and bound*, B&B). Esta técnica enumera todas as posibles solucións do problema que se estea a considerar, almacenando as solucións parciais (chamadas *subproblemas*) nunha estrutura en forma de *árbore* (Morrison et al. 2016). Os *nodos* sen explorar da árbore xeran *fillos* mediante a partición do espazo de solucións en pequenos subespazos (é dicir, ramificación ou *branching*) e empréganse certas regras para descartar (ou podar) subespazos que non sexan óptimos (acotación ou *bounding*). Unha vez explorada toda a árbore, o algoritmo devolve a mellor solución atopada. Este método conta con tres compoñentes que afectan de xeito significativo á súa execución:

- Estratexia de busca: Determina orde en que se exploran os subproblemas, o que pode repercutir no tempo de computación e na memoria necesaria. Existen diversas estratexias: *depth-first search* (DFS) ou busca en profundidade, na que se considera a listaxe de subproblemas sen explorar coma un montón e vaise eliminando o subproblema situado no posición máis alta para explorar o seguinte, incorporando os posibles fillos xerados nos postos máis elevados do montón e repetindo o proceso; *breadth-first search* (BrFS) ou busca en anchura, que explora todos os subproblemas situados á mesma distancia da raíz da árbore antes de pasar aos fillos destes; *best-first search*...

- Estratexia de ramificación: Determina como se realiza a partición do espazo de solucións en novos subproblemas, é dicir, como se xeran os fillos dun subproblema. Existen dous tipos: estratexias *binarias*, que dividen cada subproblema noutros dous mutuamente excluíntes; e *non binarias*, que seleccionan un subproblema dun conxunto con máis de dúas opcións.

- Regras de poda ou descarte: Son normas para evitar explorar subespazos non óptimos, entre as que podemos destacar as *cotas inferiores* no valor da función obxectivo en cada subproblema, usadas para descartar subproblemas nos que non se mellore a solución que se teña nese momento; e as *relacións ou regras de dominancia*, que permiten descartar subproblemas que se atopan *dominados* por outro (é dicir, aqueles para os cales a solución que proporcionan se ve mellorada pola obtida noutro).

Unha primeira e sinxela cota inferior no número de recipientes para o 1D-BPP pode calcularse a partir da formulación do problema como modelo de programación linear (Ecuación (2.1)). Soamente hai que relaxar de xeito continuo as variables y_i e x_{ij} (Martello e Toth 1990):

$$\begin{aligned} &0 \leq y_i \leq 1, && i \in \{1, \dots, m\}, \\ &0 \leq x_{ij} \leq 1, && i \in \{1, \dots, m\}, j \in \{1, \dots, n\}. \end{aligned} \tag{2.5}$$

Esta relaxación implica que os elementos se poidan dividir entre dous ou máis recipientes. Isto podemos velo claramente no noso Exemplo 3 (empresa de combustibles) onde, ao eliminar a restrición de baleirar completamente o contido de cada cisterna nun depósito, se pode distribuír o combustible entre varios depósitos, tarefa sinxela ao tratarse dunha substancia líquida. A resolución da Ecuación (2.5) vén dada por $x_{ii} = 1$, $x_{ij} = 0$ con $j \neq i$ e $y_i = t_i/c$, obténdose como solución óptima $\frac{1}{c}\sum_{i=1}^{n} t_i$, é dicir, o cociente entre a suma dos tamaños dos obxectos e a capacidade dos recipientes. Dado que o

número de contedores ten que ser un enteiro, tomaremos como primeira cota inferior o menor enteiro inmediatamente posterior a esa cantidade, é dicir:

$$LB_1 = \left\lceil \frac{1}{c} \sum_{j=1}^{n} t_j \right\rceil. \tag{2.6}$$

Exemplo 8. *Calculando esta cota coa instancia do Exemplo 1, temos o seguinte:*

$$LB_1 = \left\lceil \frac{1}{5} \sum_{j=1}^{10} t_j \right\rceil = \left\lceil \frac{1}{5}(4+3+3+2+2+2+1+1+1+1) \right\rceil = \left\lceil \frac{20}{5} \right\rceil = 4.$$

Polo tanto, podemos afirmar que serán precisos un mínimo de 4 recipientes para empaquetar os 10 elementos dos que consta este problema. Na Táboa 2.7 recóllense as cotas LB_1 para as distintas instancias que usamos como datos, que acompañados dos datos de número de elementos n e capacidade dos recipientes c.

Instancia	n	c	LB_1
Exemplo 1	10	5	4
GuHo1	6	100	2
GuHo4	15	17	5
GuHo5	10	61	3
Falkenauer_u120_00	120	150	48
Falkenauer_t60_00	60	1000	20
N4C1W1_H	500	100	249
N4W3B1R5	500	1000	71
BPP_100_120_0.2_0.8_8	100	120	51
BPP_200_300_0.2_0.8_0	200	300	105

Táboa 2.7: Cota LB_1 para as instancias.

O principal método de resolución exacta para o 1D-BPP que presentamos a continuación é o MTP de Martello e Toth. Tamén comentaremos algunhas características doutro par de algoritmos: o BISON de Scholl, Klein e Jürgens e do *bin-completion* de Korf (2002). Ademais, incluímos aquí a aplicación da función `gbp::bpp_solver()` do paquete `gbp` de R, que tamén devolve a solución exacta deste problema.

MTP

Este método, desenvolvido en Martello e Toth (1990), é un algoritmo de ramificación e acotación que está baseado na estratexia que se segue no algoritmo FFD. Aínda que se explicarán máis adiante tanto esta coma outras estratexias, podemos adiantar que consiste en empaquetar cada elemento no primeiro recipiente con suficiente espazo dispoñible para albergalo e é necesario que os obxectos estean ordenados con arranxo á disposición decrecente dos seus tamaños t_j. Ademais, conforme os recipientes van sendo inicializados, este algoritmo procede a numeralos. A continuación, explicamos brevemente as compoñentes do MTP, seguindo a exposición que se fai en Scholl et al. (1997): o proceso de redución MTRP, as cotas inferiores LB_2 e LB_3 e o proceso de ramificación e acotación.

Proceso de redución MTRP Antes de examinar este procedemento, imos dar unha definición formal do concepto de empaquetamento (Scholl et al. 1997):

Definición 2 (**Empaquetamento**). *Un conxunto P de obxectos dise que é un empaquetamento se $\sum_{j\in P} t_j \leq c$.*

Co fin de reducir os datos do problema e descartar solucións non óptimas, defínese ademais o seguinte criterio de dominancia (Scholl et al. 1997):

Definición 3 (**Dominancia**). *Un empaquetamento P_1 dise que domina a outro empaquetamento P_2 se existe unha partición de P_2 en subconxuntos $S_1, \dots, S_q$ e un subconxunto $\{j_1, \dots, j_q\}$ de P_1 tales que $w_{j_h} \geq \sum_{k\in S_h} w_k$ para $h = 1, \dots, q$. Ademais, diremos que empaquetamentos distintos son equivalentes entre si cando conteñen os mesmos tamaños.*

É posible construír un empaquetamento en particular de tal forma que domine a todos os demais. A definición anterior implica que calquera solución que conteña ao empaquetamento dominado P_2 pode transformarse facilmente noutra solución co mesmo número de recipientes que conteña ao empaquetamento dominante P_1 intercambiando cada subconxunto S_h polo correspondente obxecto j_h.

Exemplo 9. *Para ilustrar estas dúas definicións, botamos man de novo da instancia do Exemplo 1, cuxos datos eran $n = 10$, $t_j \in \{4, 3, 3, 2, 2, 2, 1, 1, 1, 1\}$ e $c = 10$. Exemplos de posibles empaquetamentos son:*

$$P_1 = \{1, 7\}, P_2 = \{2, 7\}, P_3 = \{2, 4\}, P_4 = \{2, 7, 8\}, P_5 = \{4, 7, 8, 9\},$$

xa que en ningún dos conxuntos se excede a capacidade dos recipientes:

$$\sum_{j\in P_1} t_j = \sum_{j\in P_3} t_j = \sum_{j\in P_4} t_j = \sum_{j\in P_5} t_j = 5 = c, \sum_{j\in P_2} t_j = 4 < c.$$

En canto á dominancia, vemos facilmente que o empaquetamento P_1 domina a P_2, P_4 e P_5 pois existen subconxuntos nestes que poden ser substituídos polo elemento 1 ($t_1 = 4$). Por outra banda, P_5 é dominado por P_1, P_3 e P_4. Por exemplo, o elemento 2 ($t_2 = 3$) en P_3 pode ser substituto do conxunto $\{7, 8, 9\}$ de elementos en P_5.

Esta regra de dominancia é usada no MTRP do seguinte xeito: se un empaquetamento P que contén ao obxecto j domina a todos os outros que tamén o conteñen, entón os datos do problema vense reducidos fixando o empaquetamento P (é dicir, asignando todos os obxectos de P a un recipiente). Dado que examinar todas as posibles relacións de dominancia require moito tempo de computación, MTRP considera soamente empaquetamentos que conteñan tres elementos como moito. Ademais, o procedemento restrínxese a casos sinxelos de dominancia que poden ser contrastados de xeito eficiente. Se nun destes contrates se identifica un empaquetamento que conteña ao obxecto j e que domine a todos os demais posibles que o conteñan, fíxase dito empaquetamento e redúcense os datos do problema. O tempo de computación deste proceso é $O(n^2)$.

Este procedemento de redución ten unha dobre aplicación. Por unha banda, permite reducir o tamaño dos datos orixinais do problema, chegando a atoparse nalgúns casos unha solución óptima só coa súa aplicación. Por outro lado, úsase para calcular a cota inferior LB_3, da que falaremos máis adiante.

Cota inferior LB_2 O cálculo desta cota comeza coa partición do conxunto de índices dos obxectos (que denotaremos por J) en tres subconxuntos:

$$J_1 = \{j \in J | t_j > c - a\}, J_2 = \{j \in J | c - a \geq t_j > c/2\} \text{ e } J_3 = \{j \in J | c/2 \geq t_j \geq a\},$$

onde a é un número enteiro no intervalo pechado $[0, c/2]$. Á vista da definición dos subconxuntos, os elementos de J_1 e J_2 non poden combinarse con outros deses mesmos subconxuntos, pois ocupan máis da metade da capacidade dos recipientes, polo que serán precisos $|J_1|+|J_2|$ recipientes para empaquetar os elementos dos dous primeiros conxuntos. O espazo dispoñible que deixan os elementos do segundo conxunto pode calcularse como $|J_2|c - \sum_{j \in J_2} t_j$. Por outra banda, debido á restrición de capacidade, os elementos de J_3 non poden asignarse a recipientes nos que xa haxa un elemento de J_1, polo que soamente se poden combinar cos de J_2 ou entre si. No mellor dos casos, o espazo restante nos $|J_2|$ recipientes será completado cos elementos de J_3 e o espazo necesario para os elementos sen empaquetar será $\sum_{j \in J_3} t_j - \left(|J_2|c - \sum_{j \in J_2} t_j\right)$. Polo tanto, a cantidade mencionada $|J_1| + |J_2|$ de recipientes verase incrementada en

$$\left\lceil \frac{\sum_{j \in J_3} t_j - \left(|J_2|c - \sum_{j \in J_2} t_j\right)}{c} \right\rceil, \tag{2.7}$$

unha expresión que nos recorda á fórmula da cota LB_1 (Ecuación (2.6)). Polo tanto, unha cota inferior $L(a)$ á hora de empaquetar os elementos destes subconxuntos é:

$$L(a) = |J_1| + |J_2| + \text{máx}\left\{0, \left\lceil \frac{\sum_{j \in J_3} t_j - \left(|J_2|c - \sum_{j \in J_2} t_j\right)}{c} \right\rceil\right\}.$$

Convén mencionar o porqué do máximo no terceiro sumando. A razón é que, se os elementos de J_3 se asignan na súa totalidade aos recipientes que xa conteñen elementos de J_2, non temos garantido que o espazo restante nestes recipientes sexa nulo, é dicir, que a Ecuación (2.7) sexa 0. Se isto non fose así, a citada expresión sería negativa e estariamos ante unha situación de diminución no número $|J_1| + |J_2|$ inicial de recipientes, o que daría pé a unha contradición. Polo tanto, asignando nese caso o valor 0 ao terceiro sumando, o número de obxectos en J_1 e J_2 constituiría unha cota inferior no número óptimo de recipientes. Dito isto, tomarase como cota inferior LB_2 o maior $L(a)$ para todo $a \in [0, c/2]$:

$$LB_2 = \text{máx}\Big\{L(a) | a \in [0, c/2]\Big\}.$$

É sinxelo ver que LB_2 é sempre maior ou igual ca LB_1. Se consideramos calquera instancia con $a = 0$, temos:

$$\begin{aligned} L(0) =& 0 + |J_2| + \text{máx}\left\{0, \left\lceil \frac{\sum_{j \in J} t_j - |J_2|c}{c} \right\rceil\right\} = \\ =& |J_2| + \text{máx}\left\{0, \left\lceil \frac{\sum_{j \in J} t_j}{c} \right\rceil - \left\lceil \frac{|J_2|c}{c} \right\rceil\right\} = \\ =& |J_2| + \text{máx}\{0, LB_1 - |J_2|\} = \text{máx}\{|J_2|, LB_1\}, \end{aligned}$$

de onde sacamos que $LB_1 \leq L(0) \leq LB_2$. Tamén convén sinalar que o cálculo desta cota ten sentido unicamente se hai elementos cuxos tamaños excedan a cantidade $c/2$. En canto ao tempo de computación, neste caso requírese $O(n)$.

Exemplo 10. *Imos calcular a cota LB_2 para os datos do Exemplo 1. Como datos de partida, temos $c = 5$, $t_j \in \{4, 3, 3, 2, 2, 2, 1, 1, 1, 1\}$ e $J = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Dado que a capacidade é $c = 5$, daquela $c/2 = 2{,}5$ e $a \in \{0, 1, 2\}$.*

- $a = 0$*: Os conxuntos son* $J_1 = \emptyset$, $J_2 = \{1, 2, 3\}$ *e* $J_3 = \{4, 5, 6, 7, 8, 9, 10\}$. *A partir deles, podemos calcular:*

$$|J_1| = 0, |J_2| = 3, \sum_{t_j \in J_2} t_j = 4 + 3 + 3 = 10, \sum_{t_j \in J_3} t_j = 2 + 2 + 2 + 1 + 1 + 1 + 1 = 10.$$

Daquela:

$$L(0) = 0 + 3 + \text{máx}\left\{0, \left\lceil \frac{10 - (3 \times 5 - 10)}{5} \right\rceil \right\} = 4.$$

- $a = 1$*: Os conxuntos resultantes son os mesmos que no caso anterior, polo que:*

$$L(1) = L(0) = 4.$$

- $a = 2$*: Os conxuntos son:* $J_1 = \{1\}$, $J_2 = \{2, 3\}$ *e* $J_3 = \{4, 5, 6\}$. *Entón temos:*

$$|J_1| = 1, |J_2| = 2, \sum_{t_j \in J_2} t_j = 3 + 3 = 6, \sum_{t_j \in J_3} t_j = 2 + 2 + 2 = 6.$$

Daquela:

$$L(2) = 1 + 2 + \text{máx}\left\{0, \left\lceil \frac{6 - (2 \times 5 - 6)}{5} \right\rceil \right\} = 4.$$

Polo tanto, a cota LB_2 *é:*

$$LB_2 = \text{máx}\{L(0), L(1), L(2)\} = 4.$$

Na Táboa 2.8 podemos ver unha comparativa entre esta cota, a LB_1 *e o número de elementos* n. *Ditas cotas están calculadas a partir dos datos das instancias seleccionadas. O cálculo de* LB_2 *levámolo a cabo usando o código do software R que se atopa no correspondente apéndice. Realizámolo para todas as instancias, a pesares de que nalgún caso os tamaños dos elementos non exceden a cantidade* $c/2$. *Observamos que se manteñen en valores iguais ou algo maiores a* LB_1.

Cota inferior LB_3 Na construción desta nova cota inferior empréganse o procedemento MTRP e a cota inferior LB_2. Pártese dunha instancia I do problema orixinal, á cal aplicamos o procedemento MTRP, polo que conseguimos fixar p_1 empaquetamentos, reducindo I á instancia "residual" I_1. Aplicando LB_2 a I_1, obtemos o límite $L'_1 = p_1 + LB_2(I_1)$. O proceso continúa de xeito reiterado: a instancia I_k vese relaxada eliminando o elemento máis pequeno e, tras aplicar MTRP, fíxanse p_{k+1} empaquetamentos, quedando unha instancia residual I_{k+1} e un novo límite inferior $L'_k = p_1 + \cdots + p_k + LB_2(P_k)$; e finaliza cando a instancia residual se baleira tras K iteracións. Polo tanto, a cota inferior LB_3 será o máximo de todas as cotas L'_k construídas:

$$LB_3 = \text{máx}\left\{L'_1, \ldots, L'_K\right\}.$$

Dado que en cada iteración se elimina (polo menos) un elemento da instancia residual e como MTRP tiña un tempo $O(n^2)$, o cálculo de LB_3 tomará $O(n^3)$.

Exemplo 11. *Como o cálculo desta cota precisa, entre outros, da execución do método MTRP, ilustraremos ambos procedementos coa axuda da instancia Exemplo 1 e seguindo os seudocódigos que se proporcionan en Martello e Toth (1990). Partimos dos datos xa coñecidos:*

$$n = 10, \qquad c = 5, \qquad t_j \in \{4, 3, 3, 2, 2, 2, 1, 1, 1, 1\}.$$

Instancia	n	LB_1	LB_2
Exemplo 1	10	4	4
GuHo1	6	2	2
GuHo4	15	5	5
GuHo5	10	3	3
Falkenauer_u120_00	120	48	48
Falkenauer_t60_00	60	20	20
N4C1W1_H	500	249	251
N4W3B1R5	500	71	71
BPP_100_120_0.2_0.8_8	100	51	53
BPP_200_300_0.2_0.8_0	200	105	114

Táboa 2.8: Comparación das cotas LB_1 e LB_2 para as instancias.

Nunha primeira execución do MTRP, obtemos como saída o número de empaquetamentos fixados e a disposición dos elementos nos recipientes:

$$z = 3, \qquad b_j = \{1, 2, 3, 2, 3, 0, 1, 0, 0, 0\},$$

é dicir, que logramos realizar tres empaquetamentos doutros tantos pares de elementos: nun primeiro recipiente os elementos 1 e 7, no segundo os elementos 2 e 4 e no terceiro os elementos 3 e 5. Permanecen sen asignar os elementos 6, 8, 9 e 10. A instancia residual I_1 vén definida por

$$n = 4, \quad c = 5, \quad t_j \in \{2, 1, 1, 1\}.$$

Neste punto, aplícase a cota LB_2 a I_1 e resulta $LB_2 = 1$. Daquela, o cálculo de L'_1 redúcese a sumar o número de empaquetamentos fixados co valor da cota inferior para a instancia residual:

$$L'_1 = p_1 + LB_2(I_1) = 3 + 1 = 4.$$

Como aínda quedan elementos sen asignar, elimínase o de menor tamaño da instancia e procédese a executar de novo o MTRP para os datos

$$n = 3, \quad c = 5, \quad t_j \in \{2, 1, 1\}.$$

Obtense desta volta un único empaquetamento ($p_2 = 1$) para os tres elementos dos que constaba a instancia, polo que en termo globais temos

$$z = 4, \qquad b_j = \{1, 2, 3, 2, 3, 4, 1, 4, 4, -\}.$$

Non hai instancia residual á que aplicar LB_2 e, polo tanto, temos $L'_2 = p1 + p2 = 4$, terminando o proceso. Finalmente, a cota LB_3 é:

$$LB_3 = \text{máx}\{L'_1, L'_2\} = 4.$$

Convén engadir que, asignando o elemento 10 (o suprimido) ao recipiente 4, obtemos o empaquetamento óptimo para a instancia orixinal.

Proceso de ramificación e acotación O MTP aplica unha estratexia de busca en profundidade (DFS) cun esquema de ramificación baseado na estratexia do algoritmo FFD. Para iso, é necesario que os elementos estean dispostos en orde decrecente dos seus tamaños. En cada nodo da árbore, asígnase o primeiro elemento "libre" (é dicir, o de maior tamaño) ao recipiente xa inicializado con maior espazo dispoñible ou, se isto non é posible, a un novo. O proceso segue calculando as cotas inferiores LB_2 e LB_3 e aplicando o método de redución MTRP. Para estes cálculos, o subproblema xerado en cada nodo reláxase como segue: para calquera recipiente inicializado créase un novo elemento cuxo tamaño é igual ao tamaño total dos elementos contidos nel e aplícanse as cotas e a redución a este elemento e aos que están por asignar. Procédese a descartar un nodo cando unha das súas cotas inferiores é maior ou igual que a cota superior UB, que se corresponde co valor da función obxectivo da solución actual. Se non é posible descartalo, aplícanse os algoritmos FFD, BFD e WFD (que se explicarán máis adiante) para mellorar UB e tratar de descartar o nodo. Todos aqueles nodos non descartados son ramificados seguindo a estratexia xa mencionada. Ademais, úsase de xeito adicional un criterio de dominancia para restrinxir o número de subproblemas xerados, evitando desta maneira crear empaquetamentos nos que se cambie o elemento máis pequeno por outro menor, xa que o segundo empaquetamento estaría dominado polo orixinal.

Aplicación a datos Como xa comentamos no seu momento, o código do MTP está á disposición dos interesados na biblioteca BPPLIB. Programado en linguaxe FORTRAN, consta dunha serie de subrutinas que realizan distintas tarefas dentro da execución do procedemento principal e require como parámetros de entrada o número n de elementos, os tamaños t_j e a capacidade dos recipientes c. Devolve como resultado o número óptimo de recipientes e a asignación de elementos a estes. Na Táboa 2.9 recóllense os resultados de aplicar o MTP ao noso conxunto de instancias. Tamén se inclúen as cotas LB_1 e LB_2 que calculamos con anterioridade. Observamos que en sete instancias o número óptimo de recipientes coincide con ditas cotas, habendo lixeiras variacións nas demais.

Instancia	n	LB_1	LB_2	**Óptimo**
Exemplo 1	10	4	4	4
GuHo1	6	2	2	2
GuHo4	15	5	5	5
GuHo5	10	3	3	3
Falkenauer_u120_00	120	48	48	48
Falkenauer_t60_00	60	20	20	20
N4C1W1_H	500	249	251	251
N4W3B1R5	500	71	71	71
BPP_100_120_0.2_0.8_8	100	51	53	53
BPP_200_300_0.2_0.8_0	200	105	114	115

Táboa 2.9: Resultado do MTP aplicado ás instancias e comparación coas cotas inferiores LB_1 e LB_2.

Aplicado o MTP ás catro instancias con menor número de datos, obtemos as seguintes disposicións de elementos:

- Exemplo 1 ($c = 5$): Os elementos 1 ($t_1 = 4$) e 7 ($t_7 = 1$) no primeiro recipiente, 2 ($t_2 = 3$) e 4 ($t_4 = 2$) no segundo, 3 ($t_3 = 3$) e 5 ($t_5 = 2$) no terceiro e os catro restantes ($t_6 = 2$, $t_8 = t_9 = t_{10} = 1$) no cuarto.
- GuHo1 ($c = 100$): Os elementos 2 ($t_2 = 50$), 3 ($t_3 = 30$) e 4 ($t_4 = 20$) no primeiro recipiente e os tres restantes ($t_1 = 60$, $t_5 = t_6 = 20$) no segundo.
- GuHo4 ($c = 17$): O elemento 1 ($t_1 = 17$) no primeiro recipiente; 2 ($t_2 = 9$), 10 e 13 ($t_{10} = t_{13} = 4$) no segundo; 3 ($t_3 = 7$), 4 ($t_4 = 6$) e 7 ($t_7 = 4$) no terceiro; 5 ($t_5 = 5$), 8, 11 e 14 ($t_8 = t_{11} = t_{14} = 4$) no cuarto e os catro restantes ($t_6 = 5$, $t_9 = t_{12} = t_{15} = 4$) no quinto.
- GuHo5 ($c = 61$): Os elementos 1 ($t_1 = 44$) e 6 ($t_6 = 17$) no primeiro recipiente; 3 ($t_3 = 24$), 4 ($t_4 = 22$), 8 ($t_8 = 8$) e 10 ($t_{10} = 6$) no segundo e os catro restantes ($t_2 = 24, t_5 = 21, t_7 = 8, t_9 = 6$) no terceiro.

Na Figura 2.5 móstranse os óptimos e as asignacións das instancias Falkenauer e aleatorias. Para a lectura das disposicións de elementos, comézase de esquerda a dereita e de arriba a abaixo, indicando cada posición (elemento) o recipiente no que foi emprazado. Non incluímos os resultados das instancias Scholl debido ao gran número de elementos ($n = 500$).

Bin packing solution procedure (BISON)

O BISON é outra ferramenta para a resolución exacta do 1D-BPP, desenvolvida en Scholl et al. (1997) e cuxo nome é o acrónimo de *BIn packing SOlutioN procedure*. Toma como base o método MTP de Martello e Toth, empregando algunhas das súas compoñentes (explicadas anteriormente). A estas, engádense novas funcionalidades como, por exemplo, as indicadas brevemente a continuación:

- **Cotas interiores adicionais**: Calcúlanse tres novas cotas inferiores para o número de recipientes. A cota LB_4 constrúese de xeito moi similar á LB_2 do método MTP. A cota LB_5 elabórase a partir da segunda versión do 1D-BPP proposta precisamente por Scholl et al. (1997), que recordemos era minimizar a capacidade c de cada recipiente para unha cantidade m de recipientes dada. E por último, a cota LB_6 parte dunha base semellante a LB_2 e LB_4, pero o seu proceso de cálculo dá lugar a unha casuística determinada pola cantidade de recipientes empregados, como de completos están ditos recipientes e a cantidade de elementos asignados e por asignar.
- **Novas regras de dominancia**: Utilízanse tres novas regras de dominancia relativas a empaquetamentos máximos (aqueles nos que non se pode engadir un novo elemento sen infrinxir a restrición de capacidade), empaquetamentos en xeral e solucións parciais.

Bin completion

En Korf (2002) exponse un novo algoritmo exacto con certas semellanzas co MTP de Martello e Toth, pero que o autor presenta como aparentemente máis veloz ca este. Denominado *bin completion* "terminación, remate ou conclusión do recipiente", utiliza de maneira máis efectiva as regras de dominancia do MTP e, ao igual ca este e o BISON, é un algoritmo de ramificación e acotación. A diferenza co MTP é que non considera cada elemento e decide en que recipiente debe colocarse, senón que considera o conxunto dominante de elementos que poden completar cada recipiente.

O esquema xeral do *bin completion* é o seguinte. En primeiro lugar, calcúlase unha solución inicial co método BFD (que xa se verá). A continuación calcúlase a cota inferior LB_2 do MTP, que en Korf (2002) tamén se denomina *estimated wasted space* ou "espazo ocupado estimado". Se esta cota coincide coa solución calculada inicialmente, o algoritmo devolverá ese valor como o óptimo. En caso contrario,

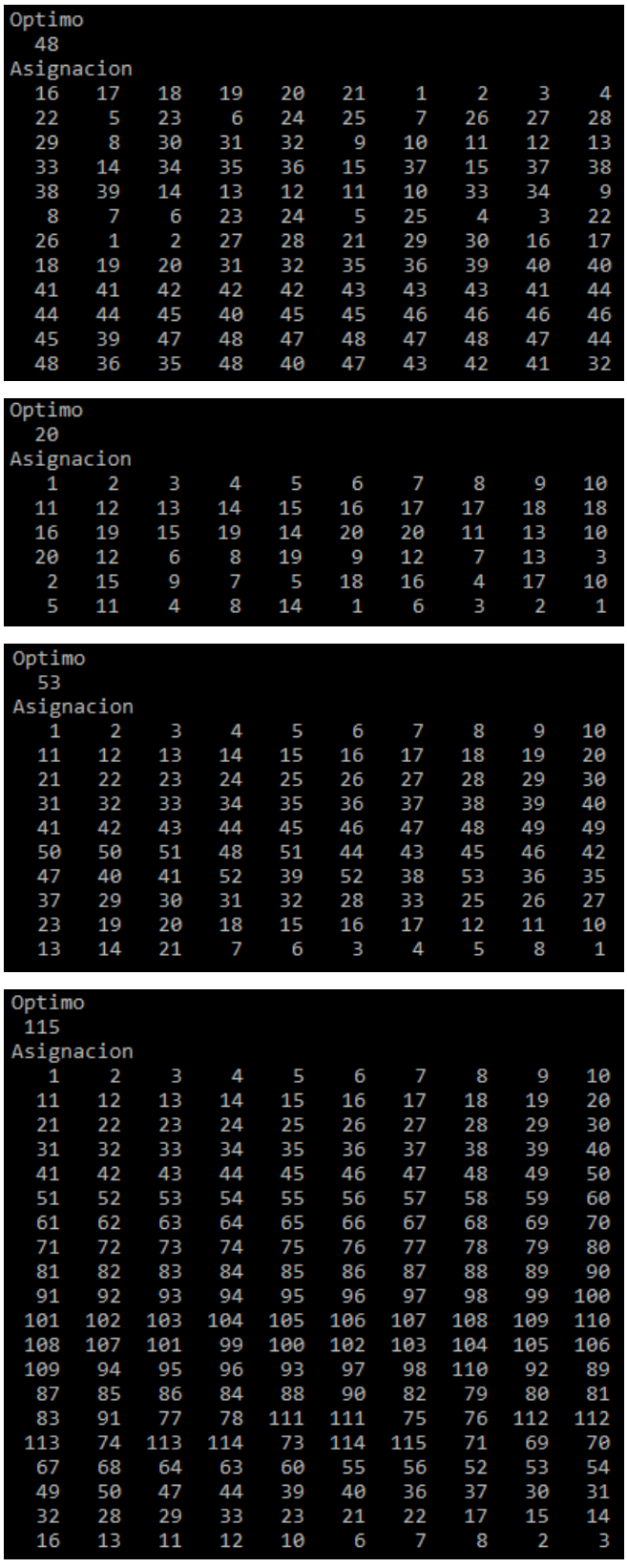

Figura 2.5: Óptimos e asignacións obtidos co MTP para as instancias Falkenauer e aleatorias.

comézase coa mellor solución obtida ata o momento e trata de mellorarse mediante ramificación e acotación.

Ao igual que nos algoritmos xa vistos, considéranse os elementos en orde decrecente do seu tamaño. Tomado un elemento, este método xera todos os conxuntos dominantes de elementos que completan o recipiente no que se atopa o citado obxecto. Se non existe ningún destes conxuntos ou se hai unicamente un, complétase o recipiente dese xeito e pásase ao seguinte, que se inicia co elemento non asignado de maior tamaño. Se hai máis dun posible conxunto dominante para completar o recipiente, ordénanse en orde decrecente da suma dos tamaños dos elementos que os compoñen, tomando o de maior suma e deixando os demais para posibles ramificacións.

Como cota inferior dunha solución parcial, este método toma a suma dos tamaños de todos os elementos máis o espazo dispoñible ou libre nos recipientes xa completados, cantidade que é dividida pola capacidade dos recipientes e redondeada ao menor enteiro inmediatamente superior. De xeito equivalente, poden sumarse o número de recipientes xa usados cos tamaños dos elementos sen asignar, dividindo pola capacidade e redondeando do mesmo xeito ca antes, un resultado que segundo o autor é máis efectivo ca a cota LB_2 do MTP, pois agora tense en consideración os espazo empregado nos recipientes completos.

A maior parte do tempo de computación deste algoritmo emprégase no calculo dos conxuntos dominantes para completar os recipientes, xerando unha colección de conxuntos factible e contrastando despois a dominancia. Este proceso comeza tomando un elemento j e buscando aquel de maior tamaño cuxa suma co t_j non exceda a capacidade c. A continuación, búscanse todos os posibles pares de elementos que non estean dominados de xeito individual polo último elemento engadido (é dicir, cuxa suma de tamaños sexa maior que o tamaño deste último). Unha vez calculados estes pares, búscanse tripletes de elementos dominantes e despois conxuntos de 4 ou máis elementos que verifiquen a condición de dominancia. A busca de conxuntos factibles de dous ou máis elementos e determinar cal domina aos demais dá lugar a un novo 1D-BPP, que xeralmente se pode resolver por forza bruta debido ao seu pequeno tamaño.

Exemplo 12. *Ilustraremos este método coa instancia GuHo4, que tomaba os seguintes valores:*

$$n = 15, \qquad c = 17, \qquad t_j \in \{17, 9, 7, 6, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4\}.$$

O primeiro recipiente estará ocupado polo elemento 1, pois o seu tamaño coincide coa capacidade do recipiente. Restan, polo tanto, 14 elementos con tamaños $t_j \in \{9, 7, 6, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4\}$. *Tomado o elemento de maior tamaño* ($t_j = 9$), *os tamaños dos elementos restantes que poderían acompañar ao primeiro son 7, 6, 5, 4 e 4+4. Calquera dos elementos de tamaño 4 é dominado polas demais combinacións de tamaños. Dado que non hai outras dominancias, tomamos a suma 4+4 (o maior tamaño), co que completamos o segundo recipiente e os demais conxuntos reservaríanse para posibles ramificacións do problema. Daquela, quedan por asignar 11 elementos con tamaños* $t_j \in \{7, 6, 5, 5, 4, 4, 4, 4, 4, 4, 4\}$. *Tomando agora o elemento de tamaño 7, pode estar acompañado por conxuntos de tamaño 6, 5, 4, 6+4, 5+5, 5+4 ou 4+4. Os únicos dominantes son os de tamaños 6+4 e 5+5. Como o tamaño total é o mesmo en ambos casos, continuaremos desenvolvendo o problema con calquera deles, por exemplo o conxunto 6+4. Completamos, por tanto, o terceiro recipiente e quedan 8 elementos con tamaños* $t_j \in \{5, 5, 4, 4, 4, 4, 4, 4, 4, 4\}$. *Neste novo subproblema, tomamos o primeiro dos elementos de tamaño 5, para o cal temos os conxuntos posibles 5, 4, 5+4, 4+4 e 4+4+4. Destes, os dominantes son 5+4 e 4+4+4, ante o cal tomamos o de maior suma* $(4 + 4 + 4)$, *polo que completamos con el o cuarto recipiente. Por último, temos 4 elementos con tamaños* $t_j \in \{5, 4, 4, 4\}$ *que completan o quinto recipiente. A solución que nos devolvería o bin completion sería un empaquetamento con 5 recipientes coa asignación que aparece recollida na Táboa 2.10. Se no seu momento escollésemos a combinación* $5 + 5$, *a solución que obteriamos sería de 6 recipientes, que claramente non é a óptima.*

***bin completion* GuHo4**					
Recipiente	1	2	3	4	5
Elementos	1	2,7,8	3,4,9	5,10,11,12	6,13,14,15
Tamaños	17	9,4,4	7,6,4	5,4,4,4	5,4,4,4

Táboa 2.10: Asignación obtida mediante *bin completion* para GuHo4.

Paquete gbp de R

O software estatístico R permítenos a resolución exacta do BPP dende unha a catro dimensións a través do paquete `gbp`. Na documentación deste (Yang 2017) introdúcese o problema facendo imaxinar ao lector que é o xerente dunha tenda e ten que xestionar unha serie de pedidos dos seus clientes. Os datos recóllense en dúas táboas:

- `it`. Nesta táboa especifícanse os pedidos, que quedan identificados cunha etiqueta (`oid`), e en cada un deles inclúense un ou máis tipos de produtos (tamén identificados (`sku`)) con catro medidas específicas: lonxitude `l`, profundidade `d`, altura `h` e peso `w`.
- `bn`. Dado que os pedidos serán empaquetados en caixas ou recipientes (non necesariamente iguais), as características destes irán especificadas noutra táboa de datos cun identificador (`id`) e as correspondentes medidas (lonxitude, profundidade, altura e peso).

O obxectivo é empaquetar cada pedido no menor número de recipientes e nos recipientes de menor tamaño posible. As mencionadas táboas serven como argumentos de entrada para a función `gbp::bpp_solver()`, que nos devolve a solución ao problema noutra táboa de datos `sn$it`. Esta consta de 12 columnas que, a maiores das da táboa `it`, recollen os seguintes datos: os identificadores de elementos que van empaquetados no mesmo recipiente (`tid`), outros identificadores combinando pedido e recipiente (`otid`), o tipo de recipiente no que vai empaquetado cada elemento (`bid`) e as coordenadas `x,y,z` do vértice de referencia do elemento. Para ver os empaquetamentos de cada recipiente, temos a función `bpp_viewer()`.

Como xa indicamos, esta función permite solucionar o problema BPP dende unha ata catro dimensións, polo que para a súa aplicación ao 1D-BPP, caso que nos ocupa neste momento, debemos tomar unha serie de consideracións:

- Non consideramos múltiples pedidos, senón un só con n produtos ou elementos.
- As capacidades dos nosos recipientes non varían, serán sempre constantes.
- Traballaremos nunha soa dimensión, polo que os tamaños dos elementos e a capacidade dos recipientes serán especificadas nunha das tres primeiras medidas (lonxitude, profundidade ou altura). Por comodidade, tomaremos a primeira delas, lonxitude, que faremos corresponder cos tamaños e a capacidade.
- Tras diversas comprobacións experimentais, comprobamos que non se deben tomar como nulas as medidas non empregadas, pois provócanse erros na execución de R. Asignando a ditas medidas valores moito máis pequenos ca os tamaños t_j ou a capacidade c (moi próximos a 0), resólvese o problema.
- Mención á parte ten o peso (de elementos e recipientes). A experimentación con distintos exemplos lévanos a afirmar que, dependendo da instancia, podemos asignar a esta variable o tamaño dos elementos ou a capacidade dos recipientes, respectivamente, ou ben valores máis pequenos, incluso

próximos a 0. Neste segundo caso, debemos ter coidado de non asignar os mesmos valores para elementos e para recipientes, senón que os valores que tomemos para os recipientes deben ser maiores, para evitar que se produza unha restrición adicional. Podemos velo coma unha especie de contador dos elementos que podemos almacenar en cada recipiente. Isto debe terse en conta para unha boa execución da función.

Tendo en conta estas consideracións, recuperamos a instancia do Exemplo 1 para ilustrar o método de R. Os datos das táboas `it`, `bn` e `sn$it` recollémolos nas Táboas 2.11, 2.12 e 2.13, respectivamente.

oid	sku	l	d	h	w
1.00	1	4.00	0.01	0.01	4.00
1.00	2	3.00	0.01	0.01	3.00
1.00	3	3.00	0.01	0.01	3.00
1.00	4	2.00	0.01	0.01	2.00
1.00	5	2.00	0.01	0.01	2.00
1.00	6	2.00	0.01	0.01	2.00
1.00	7	1.00	0.01	0.01	1.00
1.00	8	1.00	0.01	0.01	1.00
1.00	9	1.00	0.01	0.01	1.00
1.00	10	1.00	0.01	0.01	1.00

Táboa 2.11: Datos dos elementos do Exemplo 1 (táboa `it`).

id	l	d	h	w
b1	5.00	0.01	0.01	5.00

Táboa 2.12: Datos dos recipientes do Exemplo 1 (táboa `bn`).

Na columna `tid` da Táboa 2.13 podemos ver o número do recipiente en que se empaquetou cada elemento, polo que buscando o máximo atoparemos a solución ao problema. Neste caso, o número óptimo é 4, como xa vimos con anterioridade. Os empaquetamentos resultantes móstranse na Figura 2.6. Debido a que algunhas das instancias teñen un número alto de elementos, non tabulamos os empaquetamentos obtidos e soamente recollemos os óptimos que devolve a función, tal e como se mostra na Táboa 2.14.

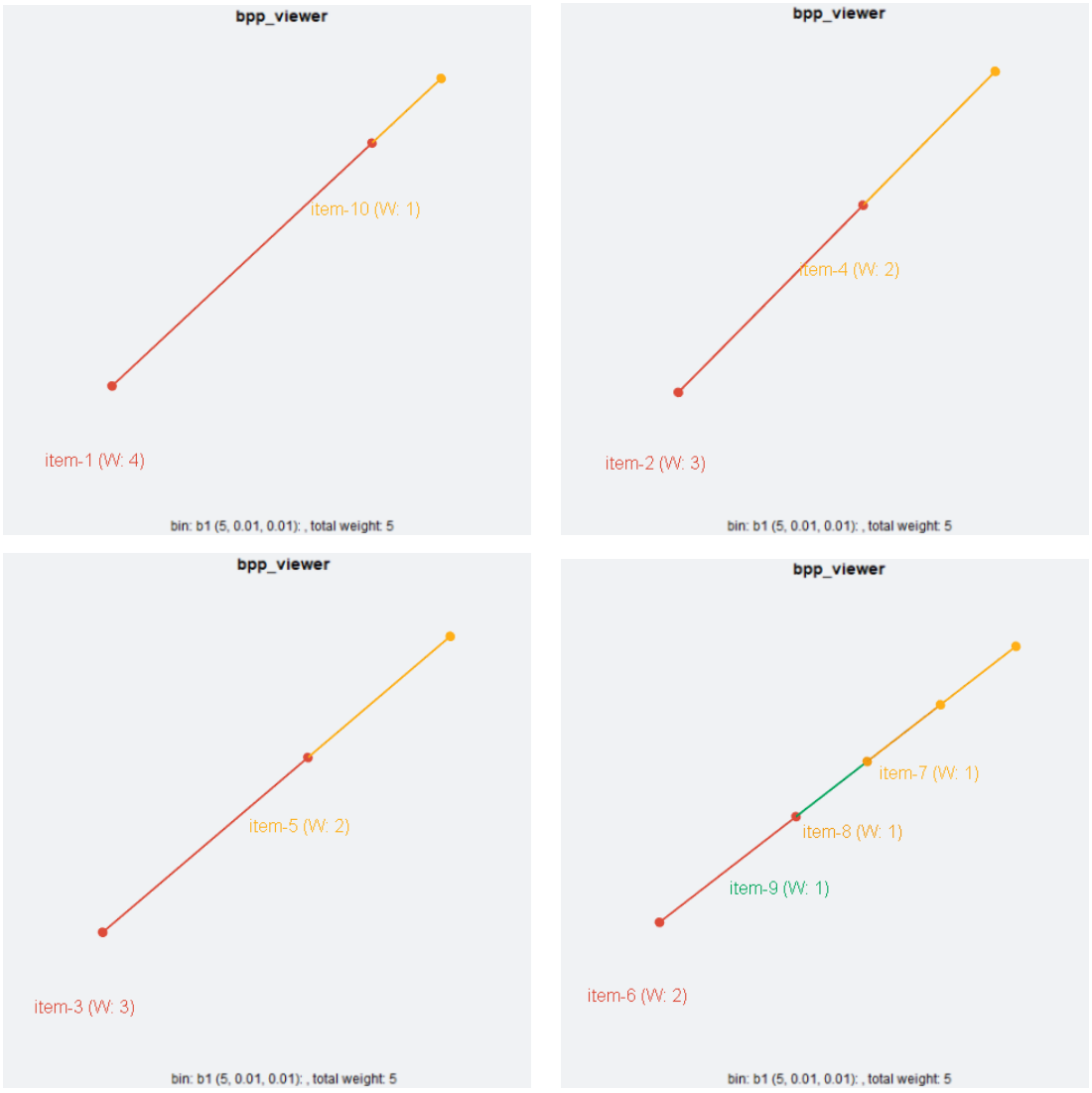

Figura 2.6: Representación dos empaquetamentos obtidos con `gbp::solver_bpp` para o Exemplo 1.

oid	tid	otid	bid	sku	x	y	z	l	d	h	w
1.00	1.00	1X1	b1	1	0.00	0.00	0.00	4.00	0.01	0.01	4.00
1.00	1.00	1X1	b1	10	4.00	0.00	0.00	1.00	0.01	0.01	1.00
1.00	2.00	1X2	b1	2	0.00	0.00	0.00	3.00	0.01	0.01	3.00
1.00	3.00	1X3	b1	3	0.00	0.00	0.00	3.00	0.01	0.01	3.00
1.00	2.00	1X2	b1	4	3.00	0.00	0.00	2.00	0.01	0.01	2.00
1.00	3.00	1X3	b1	5	3.00	0.00	0.00	2.00	0.01	0.01	2.00
1.00	4.00	1X4	b1	6	0.00	0.00	0.00	2.00	0.01	0.01	2.00
1.00	4.00	1X4	b1	7	4.00	0.00	0.00	1.00	0.01	0.01	1.00
1.00	4.00	1X4	b1	8	3.00	0.00	0.00	1.00	0.01	0.01	1.00
1.00	4.00	1X4	b1	9	2.00	0.00	0.00	1.00	0.01	0.01	1.00

Táboa 2.13: Asignación para o Exemplo 1 (táboa `sn$it`) realizada coa función `gbp::solver_bpp`.

Instancia	**Nº óptimo**
Exemplo 1	4
GuHo1	2
GuHo4	5
GuHo5	3
Falkenauer_u120_00	48
Falkenauer_t60_00	20
N4C1W1_H	251
N4W3B1R5	71
BPP_100_120_0.2_0.8_8	53
BPP_200_300_0.2_0.8_0	115

Táboa 2.14: Número óptimo de recipientes para as distintas instancias calculado coa función `gbp::solver_bpp`.

O paquete `gbp` proporciónanos unha segunda opción a través da función `gbp1d_solver_dpp()`, que precisa como argumentos de entrada un vector `p` de beneficios ou proveitos, outro vector `t` cos tamaños dos elementos e unha constante enteira `c` como capacidade do recipiente. O que nos devolve é a solución ao problema de mochila binario (KS), do que xa falamos anteriormente. A relación deste co SSP e co 1D-BPP que xa vimos permítenos dicir que a aplicación reiterada desta función (cos cambios ou modificacións pertinentes) nos serviría para solucionar o 1D-BPP.

2.4.2. Métodos aproximados

Para analizar o rendemento dos algoritmos aproximados, introduciremos a *análise do peor caso*, consistente en atopar unha cota superior no número óptimo de recipientes. Así, para unha instancia I e un algoritmo aproximado A para o 1D-BPP, denotamos por $A(I)$ o número de recipientes usados ao aplicar A a I, $OPT(I)$ é o número óptimo de recipientes para empaquetar os elementos de I e sexa $R_A(I) = A(I)/OPT(I)$. Definimos entón os seguintes conceptos (Martello e Toth 1990, Coffman et al. 1996):

Definición 4 (**Razón absoluta de rendemento do peor caso**). *A razón absoluta de rendemento do peor caso R_A é o menor número real r tal que $R_A(I) \leq r$ para calquera instancia I:*

$$R_A = \text{ínf}\{r \in \mathbb{R} : R_A(I) \leq r \ \ \forall I\}.$$

Definición 5 (**Razón asintótica de rendemento do peor caso**). *A razón asintótica de rendemento do peor caso R_A^∞ é o menor número real r tal que, para un enteiro positivo k, $R_A(I) \leq r$ para calquera instancia I e de xeito que $OPT(I) \geq k$:*

$$R_A^\infty = \text{ínf}\{r \in \mathbb{R} : R_A(I) \leq r \ \ \forall I \ \ e \ \ OPT(I) \geq k\}.$$

Dentro dos algoritmos aproximados podemos atopar dúas clases básicas en relación coa orde dos elementos que se teñen que empaquetar. En primeiro lugar, temos os algoritmos *on-line*, que toman os obxectos na orde en que veñen dados e empaquétanos nun recipiente de acordo coa estratexia de selección elixida, polo que se poden aplicar cando os datos do problema son dinámicos. Estes algoritmos serven para modelar situacións nas que os elementos son obxectos físicos e non existen un espazo intermedio para o seu almacenamento previo a ser empaquetados (Coffman et al. 1996). Por outra banda, os algoritmos *off-line* requiren coñecer *a priori* tanto o número de elementos como os seus tamaños e establecen unha orde determinada de acordo con algún criterio establecido previamente. Normalmente, estes adoitan ordenar de xeito decrecente os obxectos segundo o seu tamaño t_j, é dicir,

$$t_1 \geq t_2 \geq ... \geq t_n,$$

e posteriormente asignalos a recipientes. Esta segunda clase de algoritmos precisa, daquela, que os datos do problema sexan estáticos. A continuación, expoñemos as principais estratexias de selección para métodos de ambas clases (Scholl et al. 1997):

- *Next-Fit*: Consiste en asignar o primeiro elemento ao primeiro recipiente e cada un dos demais elementos é asignado ao mesmo recipiente ca o seu predecesor (é dicir, o contedor que estamos completando neste momento) se se axusta ou cabe nel. Se non é así, dáse por pechado o recipiente e iníciase un novo onde se coloca o elemento co que se estea. É dicir, se o elemento excede o espazo dispoñible, colócase no seguinte (*next*) recipiente.

  O algoritmo *on-line* desta estratexia denomínase tamén *Next-Fit* (NF), cuxo tempo de computación é $O(n)$. Para calquera instancia I, tense que $NF(I) \leq 2 \cdot OPT(I) - 1$, dándose a igualdade en instancias con óptimos altos. Dado que hai un termo constante na expresión da cota, a razón absoluta do peor caso non nos servirá para determinar o comportamento deste algoritmo (Martello e Toth 1990). Así, usaremos a ratio asintótica, cuxo valor é $R_{NF}^\infty = 2$ (Coffman et al. 1996).

A versión *off-line* recibe o nome de *Next-Fit Decreasing* (NFD), facendo fincapé esta denominación (ao igual que os outros casos que veremos) na estratexia de ordenación de xeito decrecente dos tamaños dos elementos. O tempo de computación deste algoritmo é $O(n \log n)$ debido á citada ordenación dos elementos e a súa razón asintótica é $R^{\infty}_{NFD} = 1{,}69103\ldots$.

- *First-Fit*: Asígnase cada obxecto no primeiro (*first*) recipiente xa inicializado con suficiente espazo dispoñible. Se dito elemento non cabe en ningún dos xa iniciados, empaquétase nun novo.

  O algoritmo *on-line First-Fit* (FF) ten un tempo de computación $O(n \log n)$. Para calquera instancia I temos a seguinte cota:

  $$FF(I) \leq \left\lceil \frac{17}{10} OPT(I) \right\rceil,$$

  polo que a razón asintótica será neste caso $R^{\infty}_{FF} = 17/10$ (Coffman et al. 1996). A versión *off-line* denomínase *First-Fit Decreasing* (FFD) e ten o mesmo tempo de computación que FF. En canto á cota, tense que

  $$FFD(I) \leq \frac{11}{9} OPT(I) + 4,$$

  para calquera instancia I. Polo tanto, $R^{\infty}_{FFD} = 11/9 = 1{,}222\ldots$ Tanto FF como FFD devolven solucións cando menos tan boas como as que proporcionan NF e NFD, respectivamente.

- *Best-Fit*: Este método asigna o obxecto ao recipiente no que o axuste é mellor (*best*), é dicir, ao recipiente que ten o menor espazo por completar pero suficiente para acoller ao obxecto, primando en caso de empate o recipiente iniciado antes. Iníciase un novo recipiente en caso de que o obxecto non caiba en ningún dos previamente abertos.

  A versión *on-line* denomínase *Best-Fit* (BF) e ten a mesma razón asintótica ca o FF: $R^{\infty}_{BF} = 17/10$. O algoritmo *Best-Fit Decreasing* (BFD) conta coa mesma cota e a mesma razón asintótica ca o FFD: $BFD(I) \leq \frac{11}{9} OPT(I) + 4$ para calquera instancia I e $R^{\infty}_{BFD} = 11/9 = 1{,}222\ldots$ Tanto BF como BFD teñen un tempo de computación $O(n \log n)$.

- *Worst-Fit*: En contraste coa estratexia anterior, neste caso colócase o obxecto no recipiente onde o axuste é peor (*worst*), é dicir, no que conte con maior espazo libre por completar, iniciando un novo se é o caso.

  Worst-Fit Tamén se denomina o algoritmo *on-line* resultante desta estratexia e a súa razón asintótica resulta ser a mesma que a do NF. As versións *on-line* (WF) e *off-line* (*Worst-Fit Decreasing*, WFD) teñen o mesmo tempo de computación ca os correspondentes algoritmos das estratexias *First-Fit* e *Best-Fit*.

De xeito máis xeral, ao falar de algoritmos *on-line*, podemos dicir que un algoritmo *Any-Fit* (AF) é todo aquel que non inicializa un novo recipiente a non ser que o elemento que debe ser empaquetado non teña cabida en ningún dos recipientes xa parcialmente completados. E podemos engadir tamén que un algoritmo *Almost Any-Fit* (AAF) é todo aquel que non empaqueta un elemento no recipiente parcialmente completo iniciado en primeiro lugar a non ser que haxa máis como el ou sexa o único con suficiente espazo dispoñible. Un exemplo de algoritmo deste segundo tipo é o *Almost Worst-Fit*, que empaqueta o elemento no recipiente parcialmente completo iniciado en segundo lugar se non hai outro no que se axuste, en cuxo caso se asigna a tal recipiente. Ademais, temos os seguintes resultados: se A é un algoritmo AF, entón $R^{\infty}_{FF} \leq R^{\infty}_{A} \leq R^{\infty}_{NF}$; e se é un algoritmo AAF, entón $R^{\infty}_{A} = R^{\infty}_{FF}$ (Coffman et al. 1996).

Existen diversas extensións dos métodos antes citados, entre as cales destacamos as seguintes variantes dos métodos FFD e BFD (Coffman et al. 1996):

- *Modified First-Fit Decreasing* (MFFD). Trátase dunha variante do FFD que difire deste na maneira de empaquetar os elementos con tamaños no intervalo $\left(\frac{1}{6}c, \frac{1}{3}c\right]$. Na asignación destes elementos, considera recipientes que soamente conteñen un elemento cuxo tamaño supere a metade da capacidade. Para cada un deste recipientes, o algoritmo comproba se os dous elementos sen asignar de menor tamaño do citado intervalo se poden colocar en dito recipiente. Se é así, asígnase o menor destes elementos ao recipiente xunto co de maior tamaño posible dos que quedan por colocar (non necesariamente o segundo menor). Se isto non é posible, continúase o empaquetamento como no FFD. Isto produce unha notable mellora con respecto ao FFD, aínda que o tempo de computación é o mesmo: $O(n \log n)$. A razón asintótica do peor caso para o MFFD é $R^{\infty}_{MFFD} = \frac{71}{60} = 1{,}8333\ldots$
- *Best-Two-Fit* (B2F). Este algoritmo funciona igual que o FFD ata que o recipiente está "completo", é dicir, ata que o espazo dispoñible neste é menor que o tamaño do elemento máis pequeno sen asignar. É entón cando B2F trata de cambiar o elemento de menor tamaño que contén o recipiente por dous elementos sen asignar de tal xeito que se minimice o espazo residual. O seu tempo de computación é $O(n^2)$ e a súa razón asintótica é $R^{\infty}_{B2F} = \frac{5}{4} = 1{,}25$, que non é mellor que a do FFD. Agora, se consideramos o algoritmo que execute a B2F e a FFD e tome a mellor das solucións calculadas, o resultado será máis eficiente. De feito, a razón asintótica desta combinación (que denotaremos por CBF) verifica:
$$1{,}164\ldots = \frac{227}{195} \leq R^{\infty}_{CBF} \leq \frac{6}{5} = 1{,}2,$$
sendo mellor cás razóns do FFD e do B2F.
- *Most-k-Fit* (MF_K), $k \geq 1$. Son un conxunto de algoritmos que empaquetan recipiente por recipiente. En cada un, colócase o elemento de maior tamaño aínda sen asignar. Logo, asígnase o elemento de menor tamaño non asignado que caiba, xunto ao cal se van engadindo outros k ou menos elementos que tamén caiban, deixando o menor espazo posible. Cando dito espazo é insuficiente para o menor elemento sen empaquetar, pásase ao seguinte recipiente. O tempo de computación deste método é $O(n^k \log n)$, que o fai pouco práctico para valores de $k \geq 2$. O algoritmo MF_2 obtén mellores resultados que FFD, pero ningún MF_K parece superar ao B2F.
- *Best-Fit Randomized* (BFR). É unha variante do BFD que tamén aplica a estratexia de selección *Best-Fit* á lista de elementos que, no canto de estar ordenados, foron permutados de xeito aleatorio. Isto diferenza este método dos vistos ata agora, xa que non produce como solución unha única asignación de elementos, senón unha distribución aleatoria de empaquetamentos. Así, para medir o seu comportamento, úsase o número esperado de recipientes $\mathbb{E}(BFR(I))$ para calquera instancia I, e así poder definir a correspondente razón asintótica R^{∞}_{BFR}, que verifica:
$$1{,}08 \leq R^{\infty}_{BFR} \leq 1{,}5.$$

Para finalizar a exposición destes algoritmos, engadimos un par de observacións. A pesar de que nesta sección nos centramos na razón asintótica do peor caso, tamén a razón absoluta R_A resulta interesante, en particular se se traballa con instancias cun número pequeno de elementos. Neste caso, tense que se o algoritmo A é FF ou BF, entón $R_A \leq 1{,}75$; en cambio, se é FFD ou BFD, entón $R_A = 1{,}5$. No primeiro caso, a razón está próxima á asintótica $R^{\infty}_{FF} = R^{\infty}_{BF} = 1{,}7$; pero no segundo hai bastante diferencia con $R^{\infty}_{FFD} = R^{\infty}_{BFD} = 11/9$ (Coffman et al. 1996). Por outra banda, en Gupta e Ho (1999) indícase que os algoritmos FFD e BFD teñen unha gran debilidade: en problemas nos que a solución óptima precisa que todos ou case todos os recipientes estean totalmente cheos (chamados "problemas difíciles"), a execución destes métodos empeora, xa que moitas veces as solucións que devolven requiren máis recipientes dos que proporciona a solución óptima. Esta é unha das razóns que expoñen os autores para motivar a súa heurística MBS, que veremos máis adiante.

Aplicación a datos

Neste apartado levaremos a cabo a aplicación dos principais métodos aproximados ás instancias coas que vimos traballando. Para iso, botamos man do software R para programar os métodos NF e FF e as súas correspondentes versións *off-line*, códigos incluídos no correspondente apéndice. Ademais, empregamos o código en linguaxe C que proporciona a web *Geeks for geeks* para o caso do BF (BFD) e adaptámolo para o caso do WF (WFD) (versión que se pode ver no correspondente apéndice). Dado que os tamaños dos elementos dos nosos datos xa están dispostos en orde decrecente, os resultados proporcionados polos algoritmos *on-line* e *off-line* van ser os mesmos, xa que unha das diferenzas é precisamente a orde de tamaños dos obxectos. Por exemplo, se os tamaños dos elementos do Exemplo 1 estivesen ordenados da seguinte forma:

$$\{1, 1, 4, 3, 1, 2, 2, 3, 1, 2\},$$

é sinxelo ver que os algoritmos *on-line* devolverían como número de recipientes a usar 4 (FF e BF), 5 (WF) e 6 (NF). Así pois, centrarémonos soamente nos algoritmos *off-line* das catro principais estratexias vistas: NFD, FFD, BFD e WFD. Ilustraremos paso a paso que ocorre se os aplicamos á instancia do Exemplo 1, onde a capacidade dos recipientes é $c = 5$:

- NFD. O elemento 1 ($t_1 = 4$) colócase no primeiro recipiente. Dado que o espazo dispoñible (unha unidade) non é suficiente para o elemento 2 ($t_2 = 3$), péchase dito recipiente e colócase o segundo elemento nun segundo recipiente. De igual modo, como o espazo restante (2 unidades) neste recipiente non é dabondo para o elemento 3 ($t_3 = 3$), péchase e colócase o elemento no terceiro recipiente, onde tamén procedemos a colocar o elemento 4 ($t_4 = 2$). Totalmente cheo o terceiro recipiente, iniciamos o cuarto colocando o elemento 5 ($t_5 = 2$), que estará acompañado polos elementos 6 e 7 ($t_6 = 2, t_7 = 1$) e que completan o recipiente. Iniciamos o quinto recipiente, onde se colocan os restantes elementos 8, 9 e 10 ($t_8 = t_9 = t_{10} = 1$), quedando dúas unidade de espazo sen utilizar. Polo tanto, o algoritmo devólvenos 5 recipientes como solución ao problema (véxase a Figura 2.7, esquerda).

- FFD e BFD. Aínda que estes algoritmos seguen estratexias diferentes, a solución para esta instancia é a mesma, polo que soamente indicaremos unha vez o proceso. Igual que antes, o elemento 1 colócase no primeiro recipiente. O elemento 2 colocase nun segundo recipiente ao non ter cabida no primeiro. O elemento 3 colocase nun terceiro recipiente por non caber nos dous primeiros. O elemento 4 tampouco cabe no primeiro recipiente e vai parar ao segundo recipiente, que é o primeiro no que cabe ou no que mellor cabe (podía asignarse ao terceiro recipiente, con mesmo espazo residual, pero en caso de empate prima o recipiente inicializado en primeiro lugar). O elemento 5 non cabe no primeiro recipiente e vai parar ao terceiro. O elemento 6 non cabe tampouco no primeiro e, como os demais están completos, iníciase un cuarto recipiente para contelo. O elemento 7 cabe no primeiro (e é no que mellor o fai) e os restantes tres elementos asígnanse ao cuarto recipiente, logrando completar a súa capacidade. Ambos algoritmos devolven como solución 4 recipientes (véxase a Figura 2.7, dereita).

- WFD. O proceso de asignación deste algoritmo é o mesmo que o do FFD e do BFD ata o elemento 6. Neste punto, temos os recipientes segundo e terceiro completos e o primeiro e o cuarto con 1 e 3 unidades de espazo dispoñible, respectivamente. O elemento 7 asígnase ao que ten maior capacidade residual, é dicir, o cuarto. O mesmo acontece co elemento 8. No caso do elemento 9, é asignado ao primeiro recipiente (pois tanto este como o cuarto teñen soamente unha unidade de espazo dispoñible e o empate favorece ao primeiro). E por último, o elemento 10 colócase no cuarto recipiente, completando a súa capacidade. Este algoritmo devolve como solución 4 recipientes, se ben a disposición dos elementos difire das obtidas con FFD e BFD, tal e como acabamos de explicar (véxase a Figura 2.7, dereita).

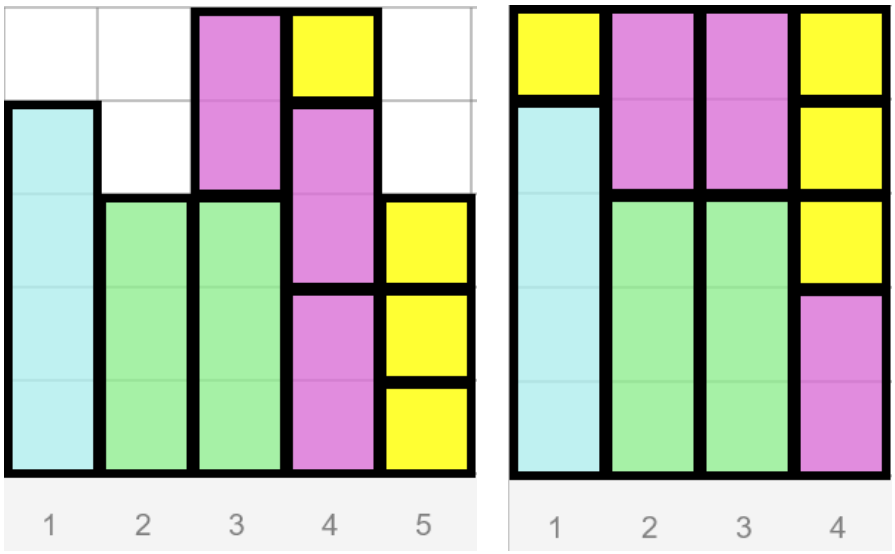

Figura 2.7: Representación en BppGame dos empaquetamentos obtidos con NFD (esquerda) e FFD/BFD/WFD (dereita) para o Exemplo 1.

Na Táboa 2.15 recóllense as solucións dos mencionados algoritmos aplicados á instancia do Exemplo 1. A pesares de que o algoritmo WFD teña unha estratexia aparentemente pouco eficiente, nalgunhas ocasións pode dar unha solución mellor ca FFD ou BFD, que en principio devolven unha solución máis próxima ao óptimo. Tal é o caso da instancia GuHo1, na que FFD e BFD precisan 3 recipientes, mentres que WFD bota man soamente de 2 (véxanse a Táboa 2.16 e a Figura 2.8). A clave está na asignación do elemento 3 ($t_3 = 30$). FFD e BFD colócano no primeiro recipiente, xa que é o primeiro no que cabe e o que menor espazo residual ten (40 unidades), mentres que no segundo se atopa só o elemento 2 ($t_2 = 50$) ocupando a metade da capacidade do recipiente. Esta asignación imposibilita que o primeiro recipiente se encha por completo, así como tampouco se completará o segundo, sendo necesario iniciar un terceiro para dar cabida ao último elemento. Pola contra, WFD coloca o mencionado elemento neste segundo recipiente (é o que máis espazo dispoñible ten) e isto serve para que os restantes elementos completen enteiramente ambos recipientes, sen necesidade dun terceiro contedor.

	Elemento (tamaño)										
Algoritmo	1(4)	2(3)	3(3)	4(2)	5(2)	6(2)	7(1)	8(1)	9(1)	10(1)	**Recipientes**
NFD	1	2	3	3	4	4	4	5	5	5	5
FFD	1	2	3	2	3	4	1	4	4	4	4
BFD	1	2	3	2	3	4	1	4	4	4	4
WFD	1	2	3	2	3	4	4	4	1	4	4

Táboa 2.15: Disposición dos elementos do Exemplo 1 segundo os distintos algoritmos.

	Elemento (tamaño)						
Algoritmo	1(60)	2(50)	3(30)	4(20)	5(20)	6(20)	**Recipientes**
NFD	1	2	2	2	3	3	3
FFD	1	2	1	2	2	3	3
BFD	1	2	1	2	2	3	3
WFD	1	2	2	1	1	2	2

Táboa 2.16: Disposición dos elementos da instancia GuHo1 segundo os distintos algoritmos.

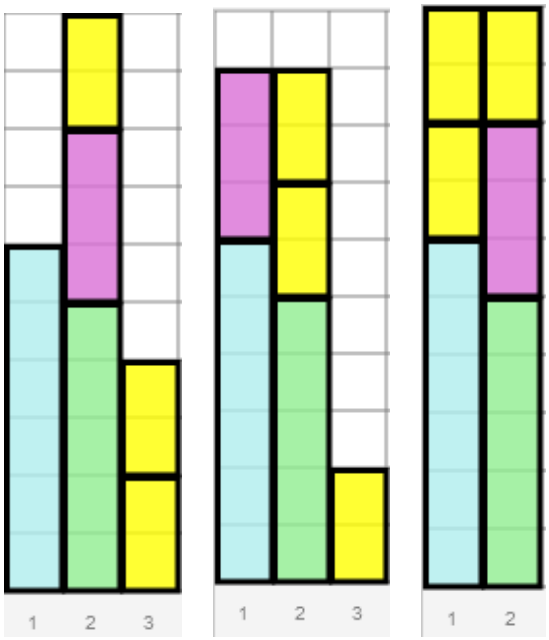

Figura 2.8: Representación en BppGame dos empaquetamentos obtidos con NFD (esquerda), FFD/BFD (centro) e WFD (dereita) para a instancia GuHo1. Os tamaños e a capacidade aparecen divididos por 10 para unha representación máis manipulable.

Ante o impracticable labor de tabular a disposición dos empaquetamentos para parte das nosas instancias debido á cantidade de elementos, unicamente indicamos na Táboa 2.17 o número de recipientes que son usados en cada algoritmo, acompañados polos óptimos e as correspondentes razóns $R_A(I)$, que nos permiten comparar coa asintótica R_A^∞. Como cabía esperar, NFD proporciona resultados como mínimo tan bos coma os dos anteriores, pero obténdose tamén diferenzas significativas nalgúns casos. Como os métodos FFD e BFD devolven a mesma solución en todos os casos, incluímolos ambos nunha mesma columna. Observamos que nas as instancias GuHo1 e GuHo4 a razón $R_{FFD/BFD}(I)$ supera a correspondente asintótica. Poderiamos afirmar que o enteiro k para o que se verifica a definición da razón asintótica debería ser maior que as solucións obtidas con estes algoritmos pero, ao contar con tan poucas instancias e de tamaños tan variados, non temos o suficiente apoio para manter tal hipótese. Como se tratan de instancias con poucos elementos, podemos tomar a absoluta $R_{FFD/BFD}$ que, como

vimos con anterioridade toma o valor 1,5, que efectivamente limita os valores que devolve en cada caso $R_{FFD/BFD}(I)$. Por outra banda, WFD devolve solucións moi semellantes a FFD e BFD, mellorando no caso xa mencionado da instancia GuHo1 e só empeorando minimamente na uniforme de Falkenauer e na primeira de Scholl. Deste último algoritmo non incluímos o cálculo das $R_A(I)$, pois non atopamos na bibliografía consultada a correspondente razón asintótica R^{∞}_{WFD} para realizar a comparativa.

Instancia	**Óptimo**	NFD	$R_{NFD}(I)$	FFD/BFD	$R_{FFD/BFD}(I)$	WFD
Exemplo 1	4	5	1.25	4	1	4
GuHo1	2	3	1.5	3	1.5	2
GuHo4	5	6	1.2	6	1.2	6
GuHo5	3	4	1.333	4	1.333	4
Falkenauer_u120_00	48	67	1.3958	49	1.0208	50
Falkenauer_t60_00	20	25	1.25	23	1.15	23
N4C1W1_H	251	321	1.2789	251	1	252
N4W3B1R5	71	76	1.0704	74	1.0423	74
BPP_100_120_0.2_0.8_8	53	70	1.3208	53	1	53
BPP_200_300_0.2_0.8_0	115	147	1.2783	115	1	115
R^{∞}_A			1.691		1.222	

Táboa 2.17: Número de recipientes segundo algoritmos *off-line* e comparación co óptimo e coa razón asintótica.

2.4.3. Métodos heurísticos

Nesta sección abordaremos algunhas das técnicas heurísticas desenvolvidas para a resolución do 1D-BPP. Comezaremos describindo o MBS (Gupta e Ho 1999), baseado en minimizar o espazo total dispoñible ou sen usar dos recipientes. Falaremos, ademais, de varios métodos elaborados a partir deste aplicando lixeiras variacións. Tamén explicaremos brevemente as características doutros métodos heurísticos como VNS (Fleszar e Hindi 2002), HI_BP (Alvim et al. 2004) e WABP (Loh et al. 2008).

Minimum bin slack (MBS)

O método heurístico que imos tratar a continuación atópase explicado en Gupta e Ho (1999). A motivación que levou aos autores a desenvolver este algoritmo foi a debilidade dos métodos FFD e BFD, citada como observación na sección dedicada aos métodos aproximados. Estes autores parten da idea de que minimizar o número de recipientes necesarios é equivalente a minimizar o total do espazo desocupado (residual ou dispoñible) (*slack* en inglés) nos recipientes. Así, o algoritmo MBS (siglas de *minimum bin slack*) é un método centrado ou enfocado no recipiente, é dicir, que trata de atopar o conxunto de obxectos que se axusta o mellor posible á capacidade de cada recipiente,

a diferenza do que realizan os citados FFD e BFD, que dan prioridade á velocidade de colocación dos obxectos nos recipientes e proporcionan solucións que requiren un maior número de contedores (Escamilla et al. 2017). Este algoritmo bota man do concepto de busca lexicográfica co fin de encher cada recipiente optimizando o espazo deste e, unha vez completo, pasar ao seguinte recipiente sempre co obxectivo de reducir o espazo desocupado neles. A orde dos tamaños dos obxectos tamén xoga un papel importante, xa que esta terá que ser decrecente para o correcto funcionamento do algoritmo. Outra clave deste método é o concepto de *backtracking*. Esta estratexia consiste en probar distintas combinacións de elementos ata dar cunha posible solución que, se non verifica as restricións esixidas, se desbota e se exploran outras. Isto úsase para colocar e eliminar obxectos do recipiente (Escamilla et al. 2017). Baseándose nos conceptos e características mencionadas, constrúese o algoritmo L, que minimiza o espazo desocupado dunha soa caixa. O seu uso reiterado constitúe o algoritmo MBS.

Para describir os pasos que se seguen no algoritmo L, utilizaremos a notación empregada na bibliografía consultada (Gupta e Ho 1999, Escamilla et al. 2017). Sexan, pois, $\pi = (\pi(1), \dots, \pi(j))$ os j elementos empaquetados no recipiente i-ésimo tales que $t_{\pi(k)} \geq t_{\pi(k+1)}$ $1 \leq k \leq j-1$. Sexa tamén $P_\pi = \sum_{k=1}^{j} t_{\pi(k)}$ a suma dos tamaños dos elementos no recipiente i (tamén chamada carga). Ademais, créase o conxunto $\sigma = (\sigma(1), \dots, \sigma(s))$ correspondente aos elementos sen asignar ordenados por tamaños decrecentes. O algoritmo L toma como parámetros de entrada os tamaños dos elementos, a capacidade c, o índice do recipiente i, o conxunto de elementos non asignados e $j = 1$ (asígnase o primeiro elemento -o de maior tamaño- ao recipiente). Os pasos a seguir son (véxase a Figura 2.9):

1. Se a carga actual do recipiente (é dicir, a suma dos tamaños dos obxectos asignados e do último tomado) coincide coa súa capacidade ($P_\pi = c$), o algoritmo termina devolvendo como solución a disposición dos obxectos no contedor (que denotaremos por S_i) e o espazo non ocupado neste $c - P_\pi$. En caso contrario, continúase ao paso 2.

2. Búscase o índice do elemento tomado no conxunto de non asignados ($q/\sigma(q) = \pi(j)$). Se o tamaño do recipiente non supera o espazo dispoñible ($P_\pi < c$), asígnase o obxecto actual ao recipiente ($S_i = \pi$), actualízase o índice dos asignados ($j = j+1$) e vaise ao paso 4. En cambio, se o tamaño supera o espazo dispoñible, vaise tamén ao paso 4 pero sen asignar dito obxecto.

3. Mentres haxa obxectos non asignados ($q < s$), tómase o posterior ao último co que se probou ($\pi(j) = \sigma(q+1)$) e vólvese ao paso 2. En caso contrario, pasamos ao paso 5.

4. En caso de que a única posibilidade sexa un empaquetamento cun só obxecto ($j = 1$), o algoritmo finaliza devolvendo a disposición do recipiente S_i e o espazo non completado neste $c - P_\pi$. Noutro caso (é dicir, se non se encheu completamente o recipiente ou se excedía a súa capacidade tras probar con cada un dos restantes obxectos non asignados), retírase o último elemento colocado no contedor ($j = j - 1$), regresando tanto este como os seguintes ao conxunto de non asignados ($q/\sigma(q) = \pi(j)$) e repítese o paso 3.

O MBS repite o algoritmo L ata que todos os obxectos son asignados a recipientes. Toma como datos de entrada os tamaños dos elementos, c, $i = 1$ e $s = n$. Os pasos que segue son os seguintes (véxase a Figura 2.9):

1. Úsase o algoritmo L para determinar a asignación de obxectos S_i ao recipiente i. Actualízase o conxunto dos elementos non asignados σ e vaise ao paso 2.

2. Se dito conxunto non está baleiro ($\sigma = \emptyset$), tómase un novo recipiente ($i = i + 1$) e vólvese ao paso 1. Se o conxunto está baleiro, o algoritmo detense, devolvendo como solución a asignación de obxectos $(S_1, \dots, S_i)$ aos recipientes $1, \dots, i$ e o espazo non ocupado total ($i \cdot c - \sum_{j=1}^{n} t_j$).

O algoritmo comeza tomando os obxectos de maior tamaño debido a que normalmente é máis difícil o seu empaquetado en recipientes (Fleszar e Hindi 2002). Este método xera unha solución óptima se

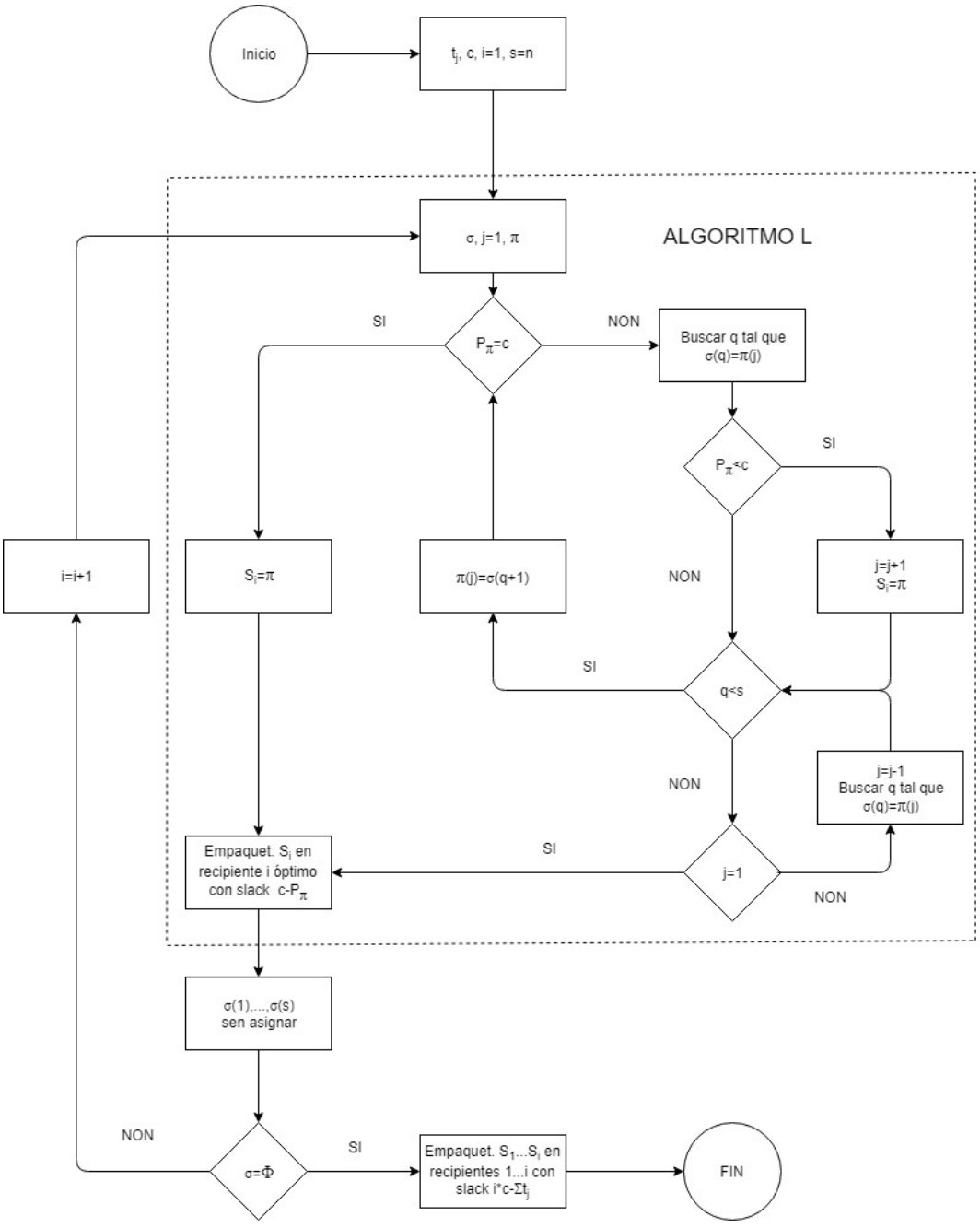

Figura 2.9: Diagrama de fluxo do algoritmo MBS.

a capacidade dos recipientes c é cando menos a metade do requisito de capacidade total dos obxectos, é dicir

$$2c \geq \sum_{j=1}^{n} t_j.$$

Dito doutro xeito, o algoritmo MBS atopa unha solución óptima para instancias nas que soamente son necesarios dous recipientes para empaquetar todos os elementos. Ademais, Gupta e Ho (1999) mostran que este algoritmo é mellor que os xa citados FFD e BFD en termos da calidade da solución. Ao expor estes tres algoritmos aos mesmos cinco problemas, con distintos números de obxectos, requisitos de tamaño e capacidade dos recipientes, estes autores observaron que as solucións obtidas co MBS melloraban nunha unidade as que devolvían tanto FFD como BFD. A conclusión á que chegaron estes autores é que o devandito algoritmo é de grande utilidade na resolución de dous tipos de problemas 1D-BPP: aqueles onde a suma dos tamaños ou requisitos de capacidade dos obxectos é menor ou igual que o dobre da capacidade dos recipientes, garantindo solución óptima como xa dixemos anteriormente; e problemas para os cales unha solución óptima precise que a maioría dos recipientes ou todos eles estean exactamente completos.

Segundo Escamilla et al. (2017), os conxuntos de datos ou instancias empregados en Gupta e Ho (1999) non teñen recoñecemento na literatura especializada, polo que non se pode ter a certeza de que o MBS funcione igual de ben con outras instancias. Para iso, en Escamilla et al. (2017) aplican este método a datos creados especificamente para o 1D-BPP. En particular, catro conxuntos de datos con 10 instancias cada un, tres deles descritos na biblioteca BPPLIB: os dous primeiros conxuntos de Scholl e o de Schoenfield. Comprobaron que o algoritmo MBS dá bos resultados aos comparalos con FFD e BFD. Incluíron tamén o cálculo do tempo de computación, que era maior en xeral para o MBS. Con todo, atopáronse con certas variacións nos resultados deste método respecto dos outros, algo que ditos autores acusaron a certas características dos datos, como a existencia de valores de tamaños repetidos, o número de díxitos destes tamaños ou a premisa sinalada en Gupta e Ho (1999) de que o MBS atopa a solución óptima se a suma dos tamaños dos elementos é como máximo igual ao dobre da capacidade dos recipientes.

Aplicación a datos En primeiro lugar, ilustraremos os pasos do algoritmo MBS co noso Exemplo 1. Recordemos que os datos desta instancia eran:

$$n = 10, \qquad c = 5, \qquad t_j \in \{4, 3, 3, 2, 2, 2, 1, 1, 1, 1\}.$$

O proceso é o seguinte:

- $k = 1$. No primeiro recipiente asígnase o primeiro elemento da instancia ($t_1 = 4$). Este obxecto deixa 1 unidade de espazo residual no recipiente, insuficiente para poder asignar calquera dos elementos 2 a 6 ($t_j \in \{2, 3\}$). Dito espazo dispoñible é totalmente ocupado polo elemento 7 ($t_7 = 1$), polo que se dá por finalizado o empaquetamento do primeiro recipiente ao non quedar espazo por ocupar e actualízanse os datos: temos agora 8 elementos sen asignar, cuxos tamaños son $t_j \in \{3, 3, 2, 2, 2, 1, 1, 1\}$.

- $k = 2$. Asignamos ao segundo recipiente o elemento de maior tamaño ($t_1 = 3$), que deixa dúas unidades de espazo dispoñible. Neste espazo non cabe o elemento 2 ($t_2 = 3$), pero si o fai perfectamente o elemento 3 ($t_3 = 2$), co que se completa o segundo recipiente. Actualizamos: 6 elementos sen asignar con tamaños $t_j \in \{3, 2, 2, 1, 1, 1\}$.

- $k = 3$. Este terceiro recipiente complétase sen deixar espazo residual cos elementos 1 e 2 ($t_1 = 3$, $t_2 = 2$) e restan 4 elementos por asignar con tamaños $t_j \in \{2, 1, 1, 1\}$.

- $k = 4$. Os elementos sen asignar serven para encher por completo o cuarto recipiente.

Polo tanto, para o Exemplo 1, MBS devolve como solución un empaquetamento que emprega 4 recipientes sen que haxa espazo residual algún. O empaquetamento resultante é o mesmo que o obtido cos métodos aproximados FFD, BFD e WFD que representamos na Figura 2.7. Con todo, na aplicación do MBS ao Exemplo 1 non tivo lugar o reemprazamento de ningún elemento que dese unha mellor combinación (*backtracking*). Para ilustrar esta situación, tomamos a instancia GuHo4 (sacada de Gupta e Ho (1999), como xa dixemos), cuxos valores eran:

$$n = 15, \qquad c = 17, \qquad t_j \in \{17, 9, 7, 6, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4\}.$$

O proceso é como segue:

- $k = 1$. Colocamos o elemento 1 no primeiro recipiente e, como o seu tamaño coincide coa capacidade do recipiente, dámolo por completado. Actualizamos os datos: 14 elementos sen asignar con tamaños $t_j \in \{9, 7, 6, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4\}$.

- $k = 2$. Colocamos o elemento 1 ($t_1 = 9$) no segundo recipiente, que deixa un espazo residual de 8 unidades. En dito espazo ten cabida o elemento 2 ($t_2 = 7$), deixando así 1 unidade de espazo libre. Dado que ningún dos restantes elementos se pode asignar a este recipiente, procedemos a retirar o elemento 2 e probamos co seguinte ($t_3 = 6$), quedando agora 2 unidades de espazo residual. Do mesmo xeito, ningún elemento sen asignar pode colocarse en dito espazo se exceder a capacidade do recipiente, polo que retiramos o elemento 3 e substituímolo polo seguinte ($t_4 = 5$). Agora dispoñemos de 3 unidades de espazo, insuficientes para calquera dos restantes elementos. Como o elemento 5 ten o mesmo tamaño ca este, descartámolo tamén e tomamos como substituto o elemento 6 ($t_6 = 4$). Agora, o espazo dispoñible é de 4 unidades, que se completan co elemento 7 ($t_7 = 4$). Actualizamos datos: 11 elementos sen asignar con tamaños $t_j \in \{7, 6, 5, 5, 4, 4, 4, 4, 4, 4, 4\}$.

- $k = 3$. Colocamos o elemento 1 ($t_1 = 7$) neste recipiente e deixa 10 unidades de espazo libre. A continuación colócase o elemento 2 ($t_2 = 6$) e soamente restan 4 unidades de espazo, insuficientes para os elementos 3 e 4 ($t_3 = t_4 = 5$), pero que son completadas polo elemento 5 ($t_5 = 4$). Actualizamos: 8 elementos con tamaños $t_j \in \{5, 5, 4, 4, 4, 4, 4, 4\}$.

- $k = 4$. Asignamos o primeiro elemento ($t_1 = 5$) ao novo recipiente, quedando 12 unidades de espazo por completar. Colocamos entón os elementos 2 e 3 ($t_2 = 5, t_3 = 4$), co que os espazo dispoñible se reduce a 3 unidades, insuficientes para calquera dos demais elementos. Dado que os estes son do mesmo tamaño que o 3 (último en asignarse), non paga a pena proceder a substituír este por calquera dos demais pois o resultado será o mesmo. Daquela, retiramos o elemento anterior (o 2) e asignamos de novo o elemento 3, co que agora o espazo restante é de 8 unidades, que se completan cos elementos 4 e 5. Actualizamos: 4 elementos con tamaños $t_j \in \{5, 4, 4, 4\}$.

- $k = 5$. Asignamos o primeiro elemento ao recipiente e os restantes catro enchen por completo o contedor, polo que se remata o proceso de asignación.

Polo tanto, a solución que devolve MBS é un empaquetamento con cinco recipientes completamente cheos, tal e como podemos ver na Figura 2.10.

Para as demais instancias, botamos man do código do MBS implementado por Luis Alberto Silva Escamilla, que tivo a xentileza de compartilo para realizar este traballo e a quen agradecemos a súa colaboración. Dito código foi programado en linguaxe C++ e consta de 4 ficheiros: *Util*, que se encarga de ler o ficheiro de texto plano cos datos do problema (n, c, t_j); *Instancia*, que extrae os valores n e c para asignalos a senllas variables e os tamaños t_j para un vector; *HeurísticaMBS*, que contén a implementación dos algoritmos L e MBS anteriormente descritos, así como outras funcións menores; e *MBSMain*, ficheiro principal que executa todas as funcións dos anteriores. Este programa devolve por pantalla a disposición dos elementos en cada recipiente, o número de recipientes necesarios e o tempo de execución do algoritmo. Na Táboa 2.18 recollemos a cantidade de contedores empregados para

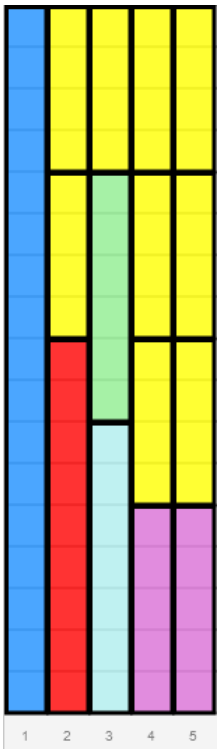

Figura 2.10: Representación en BppGame do empaquetamento resultante da instancia GuHo4 empregando o MBS.

cada instancia, que acompañamos dos datos de capacidade ou espazo total (suma das capacidades dos recipientes), espazo necesario ou ocupado (suma dos tamaños dos elementos) e espazo residual total ou *slack*, este último calculado como diferenza dos dous anteriores.

Variantes do MBS

A partir do algoritmo MBS pódense construír outros métodos heurísticos para a resolución do 1D-BPP, que difiren do algoritmo orixinal en pequenos detalles e funcionalidades. Presentamos a continuación catro métodos recollidos en Fleszar e Hindi (2002), que poden proporcionar resultados distintos dos obtidos co MBS a pesares de estar baseados nel.

MBS' Esta é unha versión lixeiramente modificada do MBS. Dita modificación con respecto ao orixinal consiste en elixir e fixar permanentemente un obxecto antes do empaquetamento dun recipiente concreto, algo que se pode facer pois todos obxectos teñen que ser empaquetados nalgún recipiente. Unha escolla axeitada é o obxecto de maior tamaño, pois é o que menor espazo deixa por encher no recipiente e acurta o tempo de computación. O resto do proceso non varía respecto ao algoritmo orixinal.

Semella claro que MBS' dará lugar a diferentes empaquetamentos e diferentes solucións das que se obteñen co MBS. Nalgúns casos poden ser peores, ao obterse empaquetamentos nos que os primeiros recipientes non están cheos completamente (debido ao ter fixado como semente un obxecto de gran tamaño). Mais tamén pode xerar resultados mellores que o MBS, xa que este tende a usar primeiro os obxectos pequenos e posteriormente os de maior tamaño para os cales non se chegaran a completar os recipientes nos que están contidos.

Instancia	MBS	Espazo total	Espazo necesario	*Slack*
Exemplo 1	4	20	20	0
GuHo1	2	200	200	0
GuHo4	5	85	85	0
GuHo5	3	183	180	3
Falkenauer_u120_00	49	7350	7078	272
Falkenauer_t60_00	21	21000	20000	1000
N4C1W1_H	251	25100	24847	253
N4W3B1R5	71	71000	70628	372
BPP_100_120_0.2_0.8_8	53	6360	6061	299
BPP_200_300_0.2_0.8_0	115	34500	31477	3023

Táboa 2.18: Número de recipientes segundo o algoritmo MBS e comparativa dos espazos total, necesario e residual (*slack*).

Exemplo 13. *Fleszar e Hindi (2002) ilustran a situación de mellora respecto ao MBS co seguinte exemplo: consideremos un conxunto a de* 3 *obxectos de idéntico tamaño* $t_1 = t_2 = t_3 = t_a = 5$ *e outro conxunto b con outros* 3 *obxectos tamén co mesmo tamaño* $t_4 = t_5 = t_6 = t_b = 3$ *que deben ser empaquetados en recipientes de capacidade* $c = 9$*. Aplicando MBS, o primeiro recipiente será completado cos obxectos* 4, 5 *e* 6 *(a suma dos tamaños coincide con* c*) e, como os restantes obxectos son dun tamaño maior á metade da capacidade* c*, serán necesarios* 3 *recipientes para empaquetalos (que ademais non estarán completamente cheos), polo que precisaremos un total de 4 recipientes neste empaquetamento e quedan* $3 \cdot (c - t_a) = 12$ *unidades de espazo desocupado. Pola contra, MBS' permite fixar un obxecto no primeiro empaquetamento, polo que poderemos asignar un obxecto de tamaño* 5 *e outro de tamaño* 3 *en cada recipiente, sendo soamente necesarios* 3 *contedores e restando* $3 \cdot (c - (t_a + t_b)) = 3$ *unidades de capacidade por ocupar.*

***Relaxed* MBS'** Tanto o "MBS' relaxado" coma os restantes métodos son xa variantes do MBS'. Neste caso, trátase dunha modificación na que se acepta que algúns empaquetamentos deixen certo espazo desocupado, sen buscar unha alternativa mellor. Isto pode facerse cambiando o criterio de parada no algoritmo L, pasando de establecer o espazo non utilizado ou *slack* en 0 a facelo por debaixo doutro valor permitido. Ademais, este método admite os chamados empaquetamentos máximos, aqueles que non poden ser incrementados engadindo algún obxecto non asignado sen violar a restrición de capacidade.

***Perturbation* MBS'** Esta modificación do MBS', que se denota en Loh et al. (2008) como PMBS', parte dunha solución inicial, non necesariamente obtida mediante o MBS', e constrúe outra nova perturbándoa. A perturbación ten lugar do seguinte xeito: escóllese e fíxase un obxecto dun recipiente cunha cantidade relativamente grande de espazo dispoñible e realízase un novo empaquetamento con este obxecto tendo en conta todos os demais, usando o procedemento xa descrito anteriormente para un

só recipiente (algoritmo L). Os obxectos deste novo empaquetamento asígnanse a un novo recipiente, que é engadido á solución do problema, mentres que se retira desta todo aquel recipiente que quedase baleiro durante o proceso. Ademais, débense definir coidadosamente a selección do obxecto e a ordenación do conxunto de elementos. Na escolla do elemento tense en conta unha probabilidade baseada na proporción de espazo dispoñible nun recipiente concreto respecto do espazo dispoñible total. Como consecuencia, non se escollen e fixan obxectos de recipientes completamente cheos e a probabilidade de escolla aumenta coa cantidade de espazo dispoñible no recipiente correspondente. Na orde do conxunto de elementos tense en conta que o obxecto elixido e fixado se coloca en primeiro lugar e o resto en orde decrecente do espazo non ocupado dos seus respectivos recipientes. Estas escollas dan prioridade a usar obxectos de recipientes menos cheos e tenden a respectar os completos ou case completos da solución actual. O carácter aleatorio destes dous procesos evita os ciclos e aumenta a diversidade da busca. Unha vez que ten lugar cada paso, é dicir, despois de que se cree un novo empaquetamento e incluído nun novo recipiente, a nova solución compárase coa actual: se require menos recipientes, adóptase esta. Todo isto detense cando o algoritmo non é quen de mellorar a solución actual tras un certo número de pasos ou cando se alcanza a cota inferior no número de recipientes.

***Sampling* MBS'** O terceiro método derivado do MBS' invoca este algoritmo varias veces, cambiando a orde dos obxectos cada vez e tomando a mellor solución. O proceso remata cando se alcanza a cota inferior no número de recipientes ou o algoritmo non é quen de mellorar a solución tras un certo número de pasos. Neste método, a orde no conxunto de elementos está baseada na orde decrecente dos tamaños destes, cunha probabilidade de seleccionar un determinado obxecto en cada etapa baseada na proporción entre o tamaño deste e a suma total de tamaños. Tras completarse a ordenación, bota a andar o algoritmo MBS'.

Outros métodos heurísticos

A continuación, explicaremos brevemente algunhas características doutros métodos heurísticos:

- *Variable neighbourhood search* (VNS) (Fleszar e Hindi 2002). O esquema do método de *busca de veciñanza variable* é analizar *veciñanzas* cada vez máis distantes da solución actual e "saltar" a unha nova se e soamente se se produce unha mellora no resultado. Ten como base os *movementos* (*moves* en inglés), definidos como o traspaso dun obxecto dun recipiente a outro ou o intercambio de dous obxectos entre os seus respectivos recipientes contedores. Os movementos permiten definir o concepto de veciñanza: definimos a k-ésima veciñanza da solución x ($N_k(x)$) como o conxunto de solucións obtidas a partir de x realizando sucesivamente k movementos de distintos obxectos. Durante cada etapa do proceso de execución do VNS, que toma como solución inicial x unha calculada mediante MBS', repítense os seguintes tres pasos:

  - *Shaking* ou abaneo: A partir da k-ésima veciñanza de x, xérase aleatoriamente un punto x'.
  - Busca local: Aplícase un método de busca local con x' como solución inicial para obter o óptimo local x''. A determinación do óptimo local realízase aplicando o algoritmo de máximo descenso (*steepest descent algorithm*) ou do gradiente. Ademais, a elección da función obxectivo é de gran importancia. Para o 1D-BPP, minimizar o número de recipientes necesarios resulta pouco útil xa que existen moitas configuracións de empaquetamento posibles que poden usar o mesmo número de recipientes. Tendo en conta que unha boa solución é aquela na que se teñan recipientes completos ou case, o axeitado parece tomar unha función obxectivo na que se maximice a carga dos recipientes, polo que unha posible función obxectivo é:
$$\text{máx}\, f(x) = \sum_{\alpha=1}^{m} l(\alpha)^2,$$
onde m denota o número de recipientes na solución inicial x, α o índice destes, $l(\alpha)$ a suma dos tamaños dos obxectos asignados a α (é dicir, a carga do recipiente α). Desta forma,

ademais de maximizar a carga dos recipientes, búscase precisamente reducir o número destes. Pódese ver que o valor desta función obxectivo non varía se se engade ou retira un recipiente baleiro. O cálculo da variación da función obxectivo ten en conta o tipo do movemento. Así, no caso dun traspaso do obxecto j do recipiente α ao recipiente β calcúlase como

$$\Delta f = (l(\alpha) - t_j)^2 + (l(\beta) + t_j)^2 - l(\alpha)^2 - l(\beta)^2.$$

De xeito análogo, no caso dun intercambio dos obxectos j_1 e j_2 entre os seus respectivos recipientes α e β tense

$$\Delta f = (l(\alpha) - t_{j_1} + t_{j_2})^2 + (l(\beta) + t_{j_1} - t_{j_2})^2 - l(\alpha)^2 - l(\beta)^2.$$

- Moverse ou non: Se o óptimo local é mellor que a solución actual, tómase este como nova solución (é dicir, $x = x''$) e reiníciase o proceso con $k = 1$. En caso contrario, increméntase a veciñanza ($k = k + 1$). Soamente se consideran como válidos aqueles movementos nos que non se excede a capacidade c dos recipientes. O algoritmo de descenso enumera todos aqueles movementos nos que se mellora a función obxectivo e, finalmente, realízase o que maximiza a mellora da función obxectivo, se isto é posible. Ao comparar un novo óptimo local x'' co actual x, primeiro compáranse os números de recipientes que se obteñen con eles, $m(x'')$ e $m(x)$. Se $m(x'') < m(x)$, entón tómase como solución $x = x''$. En caso de que coincidan o número de recipientes, avalíase a función obxectivo no óptimo local. Se $f(x'') > f(x)$, entón tómase tamén a solución $x = x''$.

- HI_BP (Alvim et al. 2004). Entre as características deste método destacan o uso das regras de dominancia do MTP e o cálculo dunha cota inferior LB que se emprega para construír unha solución factible do DBP (*dual bin packing problem*, que recordemos consiste en minimizar a capacidade de cada recipiente para unha cantidade de recipientes m dada) tomando $m = LB$. A partir desta solución inicial, trata de calcularse unha factible para o 1D-BPP usando esa mesma cantidade de recipientes.

- *Weight annealing* (WA) (Loh et al. 2008). Este método, que podemos traducir como *recocido de peso*, permite que se poida escapar dun óptimo local "malo" realizando cambios no problema mediante a asignación de pesos en diferentes partes da solución. As principais características do WA aplicado ao 1D-BPP (que denotaremos por WABP) son:

  - Función obxectivo: Loh et al (2008) toman a función obxectivo dada por Fleszar e Hindi (2002) e vista anteriormente:

$$\text{maximizar} \quad f(x) = \sum_{\alpha=1}^{m} l(\alpha)^2,$$

    onde m denota o número inicial de recipientes dispoñibles en x, α o índice destes e $l(\alpha)$ a suma dos tamaños dos obxectos asignados a α (é dicir, a carga do recipiente α).

  - Asignación de pesos: Unha das claves deste método é a distorsión dos tamaños dos obxectos. Estes cambios conséguense asignando diferentes pesos aos recipientes e aos obxectos que conteñen tendo en conta o ben empaquetados os recipientes. Para cada recipiente α, asígnase o peso

$$w_\alpha^T = (1 + Kr_\alpha)^T, \tag{2.8}$$

    onde K é unha constante, T é un parámetro de temperatura e $r_\alpha = (c - l(\alpha))/c$. O parámetro de escala K controla a distorsión do tamaño de cada obxecto, que é proporcional ao espazo libre no correspondente recipiente, mentres que r_α representa a proporción de espazo sen encher ou capacidade residual fronte ao total. Cando nos atopamos nun máximo local, este segundo parámetro pode tomar valores altos en recipientes que non están "ben

empaquetados". Para saír desta situación, auméntanse os pesos dos obxectos dos recipientes peor empacados. Incrementa así a posibilidade de intercambio entre un obxecto grande do recipiente no que se está traballando e outro pequeno procedente doutro recipiente, xa que o obxectivo é maximizar o número de recipientes completamente cheos.

O WABP toma unha solución inicial xerada polo algoritmo FFD, xa visto. Co fin de mellorar a solución, calcúlanse os pesos para os recipientes e aplícanse aos elementos do correspondente recipiente. A continuación, procédese a executar e avaliar se son factibles distintos movementos entre os obxectos. Á semellanza do método de Fleszar e Hindi (2002), pódense intercambiar (*swaping*) obxectos entre todos os posibles pares de recipientes. Loh et al. (2008) usan catro tipos de intercambios que denotan por Swap(1,0), Swap(1,1), Swap(1,2) e Swap(2,2). Convén recordar que os dous primeiros tipos son os xa descritos por Fleszar e Hindi (2002) e vistos no apartado correspondente ao algoritmo VNS. Así, no Swap(1,0) considérase o movemento dun obxecto j desde o recipiente α ao recipiente β. O cambio na función obxectivo é o seguinte:

$$\Delta f_{(1,0)} = (l(\alpha) - t_j)^2 + (l(\beta) + t_j)^2 - l(\alpha)^2 - l(\beta)^2.$$

De xeito semellante, no Swap(1,1) ten lugar o intercambio dos obxectos j_1 e j_2 entre os recipientes α e β que os conteñen respectivamente. A variación na función obxectivo é:

$$\Delta f_{(1,1)} = (l(\alpha) - t_{j_1} + t_{j_2})^2 + (l(\beta) + t_{j_1} - t_{j_2})^2 - l(\alpha)^2 - l(\beta)^2.$$

No Swap(1,2) intercámbiase o obxecto j_1 do recipiente α cos obxectos j_2 e j_3 do recipiente β, resultando modificada a función obxectivo en:

$$\Delta f_{(1,2)} = (l(\alpha) - t_{j_1} + t_{j_2} + t_{j_3})^2 + (l(\beta) + t_{j_1} - t_{j_2} - t_{j_3})^2 - l(\alpha)^2 - l(\beta)^2.$$

Por último, no Swap(2,2) intercámbianse os obxectos j_1 e j_2 do recipiente α cos obxectos j_3 e j_4 do recipiente β. A función obxectivo vese modificada do seguinte xeito:

$$\Delta f_{(2,2)} = (l(\alpha) - t_{j_1} - t_{j_2} + t_{j_3} + t_{j_4})^2 + (l(\beta) + t_{j_1} + t_{j_2} - t_{j_3} - t_{j_4})^2 - l(\alpha)^2 - l(\beta)^2.$$

O proceso de mellora, e o algoritmo en sí, remata cando se alcanza un número de iteración previamente indicado polo usuario.

2.4.4. Conclusións

Para finalizar este capítulo, recompilamos as conclusións que se extraen nas distintas referencias consultadas sobre a comparación entre os métodos que detallamos con anterioridade. Así, o BISON supera con claridade ao MTP nas probas realizadas en Scholl et al. (1997). Sometidos ambos métodos aos mesmos conxuntos de instancias e ás mesmas limitacións de tempo, o BISON é capaz de resolver un maior número de problemas que o MTP. Os autores "culpan" desa efectividade ás compoñentes adicionais que tiña o BISON respecto do seu antecesor. De igual xeito, o *bin completion* supera ao MTP. Neste caso faise patente a diferenza entre ambos métodos conforme medra o número de elementos da instancia, aumentando de xeito considerable o tempo de execución do MTP, mentres que o *bin completion* alcanza o óptimo relativamente rápido (Korf 2002).

No seu momento xa comentamos que en Gupta e Ho (1999) se afirma que o MBS devolve mellores resultados que os algoritmos FFD e BFD, baseándose na aplicación dos tres métodos as mesmas cinco instancias. Porén, estas non teñen o suficiente recoñecemento na literatura, polo que a aplicación a conxuntos de instancias creadas especialmente para o 1D-BPP permite afirmar que os resultados do MBS son, salvo pequenas variacións, moi semellantes aos dos citados algoritmos aproximados (Escamilla et al. 2017).

A comparativa entre o MBS e a súa variante MBS' devolve resultados moi similares para ambos métodos, mais globalmente o MBS' alcanza mellores solucións e faino empregando un menor tempo

de execución. Todas as demais variantes (do MBS') tamén resultan moi eficientes en termos de tempo de computación. O algoritmo VNS devolve resultados moi similares ás heurísticas derivadas do MBS (Fleszar e Hindi 2002). Para obter as mellores solucións posibles, estes autores combinan o PMBS' co VNS (que denotamos como PMBS'+VNS) do sequinte xeito: primeiro executan o PMBS' e, se non se alcanza o número óptimo de recipientes, úsase a solución que proporcione como solución inicial para o VNS. Esta combinación dá lugar a un método máis efectivo que calquera outra combinación das heurísticas tratadas no citado artigo. Ademais, compáranse o MBS' con algoritmos aproximados (FFD, BFD, WFD...) e este método resulta ser superior en canto a calidade da solución. Así mesmo, comparando o PBMS'+VNS co método exacto BISON con iguais limitacións de tempo, compróbase que o primeiro algoritmo resulta ser bastante competitivo.

Alvim et al. (2004) realizan comparativas entre a súa heurística HI_BP e métodos xa existentes como o PMBS'+VNS e o BISON. En ambos casos, prodúcense mellores resultados atendendo á calidade da solución (é dicir, como de ben se aproxima ao óptimo ou se o alcanza) e ao tempo de computación, respectivamente. E por último, Loh et al. (2008) comparan o seu algoritmo WABP cos tres métodos anteriores. De novo, os resultados favorecen ao WABP, que mellora ás heurísticas precedentes en calidade da solución e ao método exacto en tempo de computación.

Estas conclusións dan a entender que a construción de novos métodos de resolución do 1D-BPP, baseados ou non noutros existentes previamente, leva aparellada unha progresiva mellora nos resultados, tanto aproximación ou consecución do número óptimo de recipientes necesarios para o empaquetamento coma maior eficiencia e menor tempo de computación dos algoritmos.

No noso caso, dado o número reducido de instancias que empregamos para ilustrar algúns dos métodos que fomos explicando, reduciremos as conclusións a comentar os resultados en termos de calidade da solución. O número de recipientes que nos devolve cada método para cada instancia aparece exposto na Táboa 2.19. Os métodos exactos, que resumimos nunha soa columna, proporcionan as solucións óptimas, ás que se aproximan moi ben os resultados devoltos pola heurística MBS. Valores semellantes ou apenas unhas unidades peores obtéñense con tres dos catro métodos aproximados utilizados (FFD, BFD e WFD). Ademais, podemos ver como MBS devolve mellores solucións ca FFD e BFD para as instancias GuHo, feito que xa se comentou con anterioridade. O cuarto dos métodos aproximados (NFD) devolve, con gran diferenza nalgúns casos, as peores solucións da colección de métodos usados para as instancias seleccionadas, algo que cabía esperar debido á súa estratexia de asignación de elementos.

Instancia	**Exactos**	**NFD**	**FFD**	**BFD**	**WFD**	**MBS**
Exemplo 1	4	5	4	4	4	4
GuHo1	2	3	3	3	2	2
GuHo4	5	6	6	6	6	5
GuHo5	3	4	4	4	4	3
Falkenauer_u120_00	48	67	49	49	50	49
Falkenauer_t60_00	20	25	23	23	23	21
N4C1W1_H	251	321	251	251	252	251
N4W3B1R5	71	76	74	74	74	71
BPP_100_120_0.2_0.8_8	53	70	53	53	53	53
BPP_200_300_0.2_0.8_0	115	147	115	115	115	115

Táboa 2.19: Resultados dos distintos métodos.

Capítulo 3

Introdución ao *bin packing problem* en varias dimensións

A pesar de que o 1D-BPP sexa considerado o "BPP clásico", o certo é que para o público en xeral o concepto de *empaquetamento* parece facerse máis claro cando estamos ante entidades de dúas ou tres dimensións. Así pois, no presente capítulo faremos unha pequena introdución ao BPP en máis dunha dimensión. Falaremos de BPP bidimensional cando a capacidade dos recipientes e o tamaño dos elementos a empaquetar se converten en alturas e anchuras. Noutras palabras, o problema conta neste caso con dúas variables tanto para contedores como obxectos, tratándose pois dun empaquetamento rectangular. De xeito análogo ao caso unidimensional, referirémonos a este problema como 2D-BPP. Aumentando unha dimensión, atopámonos ante o 3D-BPP, no que os elementos e os recipientes son descritos en función da súa altura, anchura e profundidade. Aínda que os exemplos máis comúns do BPP multidimensional son os xa citados, existe unha terceira posibilidade, que podemos denominar 4D-BPP, onde aos elementos e recipientes do caso tridimensional se lles engade unha cuarta variable, xeralmente relacionada co peso das compoñentes do problema. Por último, como xeneralización do 1D-BPP a múltiples dimensións, tamén falaremos brevemente do *vector bin packing*. Neste problema, tanto os elementos como os recipientes pasan a ser vectores d-dimensionais, $d \geq 1$. Ao longo deste capítulo explicaremos brevemente as características destes problemas e dalgúns métodos para a súa resolución, acompañando nalgúns casos dos recursos dispoñibles en Internet e exemplos ilustrativos.

3.1. BPP bidimensional (2D-BPP)

3.1.1. Definición e características

O obxectivo do BPP en dúas dimensións é empaquetar obxectos nun número mínimo de recipientes respectando a capacidade destes, como xa vimos para o 1D-BPP. Mais agora debemos ter en conta que obxectos e recipientes contan con senllas variables de anchura e altura. Así, podemos dar a seguinte definición formal do problema (Lodi et al. 2002):

Definición 6 (**2D-BPP**). *Dado un conxunto de n elementos rectangulares con alturas h_j e anchuras w_j $j \in J = \{1, \ldots, n\}$, o 2D-BPP consiste en empaquetar todos os elementos en recipientes de idénticas altura H e anchura W usando o menor número de recipientes posible.*

Neste problema, os obxectos teñen que empaquetarse co seu borde ancho en paralelo ao borde ancho dos recipientes. Tamén asumimos, sen perda de xeneralidade, que todos os datos de altura e anchura son enteiros positivos que verifican que $w_j \leq W$ e $h_j \leq H$ $(j \in J)$. En canto á complexidade computacional, o 2D-BPP é NP-duro en sentido forte como xeneralización do 1D-BPP, cuxa complexidade xa comentamos no seu momento.

Exemplos deste problema atopámolos no ámbito do almacenamento comercial, onde a mercadoría se ten que colocar en estantes ou baldas, ou na confección das páxinas de xornais e revistas, nas cales se teñen que colocar os artigos e os anuncios. Algúns problemas relacionados ou variantes do 2D-BPP son os seguintes:

- Un problema semellante ao 2D-BPP dáse, por exemplo, nas industrias madeireiras ou téxtiles, onde se recortan elementos rectangulares a partir de pezas longas de materias. Este problema denomínase *strip packing problem* (SPP ou 2D-SPP) (problema de empaquetamento en tira ou banda) e, formalmente, defínese como segue: dados n elementos rectangulares con alturas h_j e anchuras w_j e un único recipiente de anchura W e altura infinita (tira ou *strip*), o obxectivo é colocar todos os elementos da tira de xeito que se minimice a altura da tira que se debe usar (Lodi et al. 2002).

- Outro caso interesante de empaquetamento rectangular é aquel no que cada elemento j ten asociado un beneficio ou proveito $p_j > 0$, polo que o obxectivo do problema é seleccionar o subconxunto de elementos que maximice o beneficio total. A este problema adoita chamárselle *cutting stock* ou *cutting knapsack* (Lodi et al. 2002).

- Un caso similar a estes problemas de empaquetamento atopámolo no mundialmente coñecido videoxogo *Tetris*. Nel temos unha banda de anchura fixa e debemos colocar os elementos, de formas dispares e van caendo dende a parte superior da mesma (situación *on-line*), utilizando o menor espazo posible. Algo que diferenza o *Tetris* dos citados 2D-BPP e SPP é que o completar unha fila implica a súa desaparición, algo que na vida real (onde se aplican os citados problemas) non é viable. Como é sabido, neste xogo os obxectos non poden moverse cara arriba, non hai colisións entre eles, cada elemento detense se e soamente queda totalmente suxeito pola súa parte inferior e debe estar colocado antes de que comece a descender o seguinte elemento. Ademais, existe a posibilidade de rotar o elemento para encaixalo na posición desexada (Schweer 2010). Estas son características do *Tetris* que non necesariamente teñen por que verificar en xeral o 2D-BPP e o SPP.

- Existe tamén outra variante do 2D-BPP chamada *cylindrical bin packing problem*, na que os recipientes son rectángulos e os elementos son círculos. Un exemplo real de aplicación deste problema dáse cando se deben cargar tubos en contedores (Wäscher et al. 2007).

A cota inferior máis evidente para o 2D-BPP garda similitude coa do caso unidimensional e obtense relaxando o problema de xeito que cada elemento poida ser dividido en unidades cadradas. Así, obtense a chamada *cota xeométrica*:

$$LB_x = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil.$$

Dun xeito similar á cota LB_2 desenvolvida no MTP, tamén se obtén unha cota inferior para o 2D-BPP que domina á xeométrica (Lodi et al. 2002). Así, dado un número enteiro $a \in [1, \frac{1}{2}W]$, podemos dividir o conxunto J de elementos no seguintes tres subconxuntos:

$$J_1 = \left\{ j \in J : w_j > W - a \right\}, \quad J_2 = \left\{ j \in J : W - a \geq w_j > \frac{1}{2}W \right\}, \quad J_3 = \left\{ j \in J : \frac{1}{2}W \geq w_j \geq a \right\}$$

Como os elementos de J_1 e J_1 non poden empaquetarse de dous en dous nun mesmo recipiente, constrúese unha cota LB^W para a instancia composta polos elementos deses conxuntos do mesmo xeito ca no caso unidimensional, considerando como tamaños dos elementos as alturas h_j ($j \in J_1 \cup J_2$) e como capacidade dos recipientes a altura H. Ademais, os elementos de J_3 non se poden combinar cos de J_1, polo que se terá a cota:

$$LB^W(a) = LB^W + \max\left\{0, \frac{1}{WH}\left\lceil \left\{ \sum_{j \in J_2 \cup J_3} w_j h_j - \left(H \cdot LB^W - \sum_{j \in J_1} h_j \right) W \right\} \right\rceil \right\}$$

Intercambiando anchuras e alturas, temos a cota LB^H e podemos calcular unha cota inferior global

$$LB_2 = \text{máx}\left\{ \max_{1\leq a\leq 1/2W}\left\{LB^W(a)\right\}, \max_{1\leq a\leq 1/2H}\left\{LB^H(a)\right\}\right\},$$

é dicir, tómase o máximo que producen as respectivas cotas en anchura e altura.

3.1.2. Métodos de resolución

A maioría dos métodos que trataremos a continuación empaquetan os obxectos en filas formando niveis (*levels*) (Lodi et al. 2002). O primeiro nivel constitúeo o fondo do recipiente e os obxectos son colocados coa súa base nel. Cada un dos seguintes niveis vén determinado pola liña horizontal que se traza pola cima do elemento de maior altura do nivel antecedente. Sen perda de xeneralidade, podemos asumir que:

- En cada nivel, o obxecto máis alto é o situado máis á esquerda.
- O nivel máis alto de cada recipiente é o nivel inicial.
- Os obxectos son enumerados e ordenados en orde non crecente das súas alturas h_j.

Métodos exactos

Atendendo ás consideracións anteriores, establécese un modelo de programación lineal para o 2D-BPP (véxase Lodi et al. 2004). Este modelo asume que hai dispoñibles n niveis potenciais, cada un deles asociado co elemento i que os inicializa. De xeito similar, asumimos que hai dispoñibles n recipientes potenciais, asociados ao nivel k que os inicializa. Ademais, empréganse catro conxuntos de variables binarias, dous referidos ao empaquetamento de elementos en niveis:

$$y_i = \begin{cases} 1 & \text{se o elemento } i \text{ inicializa o nivel } i \\ 0 & \text{noutro caso} \end{cases} \qquad i = 1, \ldots, n.$$

$$x_{ij} = \begin{cases} 1 & \text{se o elemento } j \text{ se empaqueta no nivel } i \\ 0 & \text{noutro caso} \end{cases} \qquad i = 1, \ldots, n-1; \quad j > i.$$

e os dous restantes relacionados co empaquetamento de niveis en recipientes:

$$q_k = \begin{cases} 1 & \text{se o nivel } k \text{ inicializa o recipiente } k \\ 0 & \text{noutro caso} \end{cases} \qquad k = 1, \ldots, n.$$

$$z_{ki} = \begin{cases} 1 & \text{se o nivel } i \text{ se empaqueta no recipiente } k \\ 0 & \text{noutro caso} \end{cases} \qquad k = 1, \ldots, n-1; \quad i > k.$$

O modelo é o seguinte (Lodi et al. 2004):

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{k=1}^{n} q_k \\
\text{suxeito a} \quad & \sum_{i=1}^{j-1} x_{ij} + y_j = 1, \quad j \in \{1, \dots, n\}, \\
& \sum_{j=i+1}^{n} w_j x_{ij} \leq (W - w_i) y_i, \qquad i \in \{1, \dots, n-1\}, \\
& \sum_{k=1}^{i-1} z_{ki} + q_i = 1, \quad i \in \{1, \dots, n\}, \\
& \sum_{i=k+1}^{n} h_i z_{ki} \leq (H - h_k) q_k, \qquad k \in \{1, \dots, n-1\}, \\
& y_i \in \{0,1\}, \qquad i \in \{1, \dots, n\}, \\
& x_{ij} \in \{0,1\}, \qquad i \in \{1, \dots, n-1\}, \ j > i, \\
& q_k \in \{0,1\}, \qquad k \in \{1, \dots, n\}, \\
& z_{ki} \in \{0,1\}, \qquad k \in \{1, \dots, n-1\}, \ i > k.
\end{aligned}
$$

A función obxectivo minimiza o número de recipientes a usar. En canto ás restricións, a primeira impón que cada elemento se empaquete soamente unha vez, a segunda é a restrición de anchura para cada nivel, a terceira indica que cada nivel só se pode asignar unha vez e a derradeira é a restrición de altura para cada recipiente.

Métodos aproximados

Dada a similitude entre o SPP e o 2D-BPP, introduciremos en primeiro lugar varios algoritmos para a resolución do SPP, que sentan a base dos que permiten resolver o 2D-BPP (Lodi et al. 2002). Todos estes métodos requiren que os elementos estean colocados en orden non crecente da súa altura e os empaquetamentos realizaranse por niveis, sendo nivel inicial a base da tira e aliñándose os elementos á esquerda. O primeiro algoritmo é o *Next-Fit Decreasing Height* (NFDH), que empaqueta cada obxecto no nivel no que está a traballar se é posible. En caso contrario, dito nivel dáse por concluído e iníciase un novo, que se crea tomando a liña que proporciona a cima do elemento máis alto do nivel anterior. Pola súa banda, o método FFDH (*First-Fit Decreasing Height*) empaqueta cada obxecto no primeiro nivel (contando dende o inicial) en que caiba e créase un novo nivel como no caso do NFDH. Por último, o algoritmo BFDH (*Best-Fit Decreasing Heigth*) coloca cada elemento naquel nivel xa iniciado no que mellor se axuste, é dicir, onde o espazo dispoñible sexa menor pero suficiente, comezando un novo nivel se tal axuste non é posible. Existen outros métodos para o SPP que non empaquetan os elementos por niveis. Tal é o caso do algoritmo BL (*Bottom-Left*), que require que os elementos estean ordenados en orde non crecente de anchuras e empaqueta cada elemento na posición máis baixa posible, tendo en conta o aliñamento á esquerda.

Como xa dixemos, os métodos para resolver o 2D-BPP botan man dalgún dos anteriores. O algoritmo HFF (*Hybrid First-Fit*) consta de dúas fases: na primeira execútase FFDH e obtense un empaquetamento en tira (*strip packing*), que se pode converter nunha instancia do 1D-BPP se tomamos un elemento por nivel, con tamaño igual á altura do nivel e a capacidade do recipiente igual a H; na segunda fase, aplícase o algoritmo aproximado FFD para problemas unidimensionais e conséguese a solución para o problema de partida. De xeito análogo, o HNF (*Hybrid Next-Fit*) usará o NFDH na primeira fase para crear un problema unidimensional, ao que se lle aplicará o NFD na segunda fase. Usando o BFDH e o BFD nas correspondentes fases, obtense o algoritmo HBF (*Hybrid Best-Fit*), coñecido tamén como FBS (*Finite Best-Strip*).

Existen máis métodos aproximados á parte dos mencionados anteriormente, como por exemplo (Lodi et al. 2002):

- FC (*Floor-Ceiling*, "chan-teito"): Método de dúas fases que, ademais de empaquetar os obxectos de esquerda a dereita coa súa base no chan do nivel, tamén empaqueta os elementos de dereita a esquerda coa cima destes tocando o teito do nivel, que se corresponde coa liña que determina a tapa superior do elemento de maior altura do nivel.
- KP (*Knapsack Packing*, "empaquetamento da mochila"): Algoritmo de dúas fases que inicia cada nivel co elemento máis alto sen empaquetar e complétao botando man da solución do correspondente problema da mochila, que maximiza a área empaquetada. A segunda fase deste método, como a do anterior, pode resolverse mediante o BFD ou un algoritmo exacto para o 1D-BPP.
- FFF (*Finite First-Fit*): É unha variante dunha soa fase do HFF que empaqueta cada elemento no nivel máis baixo do primeiro recipiente en que caiba, mentras que en caso contrario se inicia un novo nivel no recipiente no que mellor se axuste ou se inicia un novo recipiente.
- FBL (*Finite Bottom-Left*): Tomando os elementos en orde non crecente das súas anchuras, examínanse todos os recipientes inicializados para tomar aquel no que se pode empaquetar na posición máis baixa e aliñada á esquerda posible, iniciando un novo contedor se o anterior non é factible.
- AD (*Alternative Directions*): Non empaqueta os obxectos en niveis, senón que o fai en bandas non horizontais, alternando de esquerda a dereita e de dereita a esquerda, de aí o seu nome.

3.1.3. Recursos e exemplos

Como xa explicamos para o 1D-BPP, un dos visualizadores alí empregados (a web `http://www.binpacking.4fan.cz/`) permite tamén ilustrar o caso bidimensional. Os cambios dun problema ao outro radican na adición dunha variable máis (altura) e no tipo de asignación (en altura ou en anchura). A Figura 3.1 mostra a visualización dun exemplo aleatorio con 500 elementos cuxas dimensións varían entre 10 e 30 unidades e cun recipiente (ou tira) con altura 300 unidades e anchura 800.

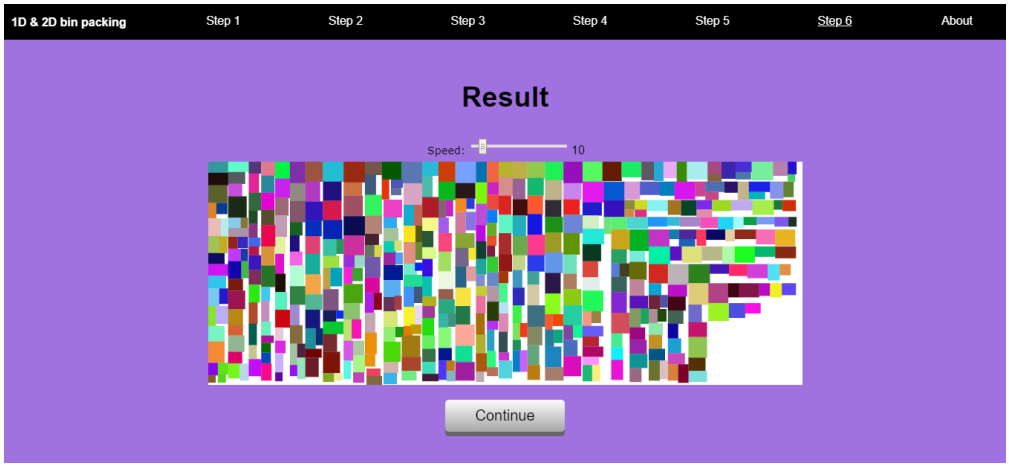

Figura 3.1: Visualización dun exemplo aleatorio de 2D-BPP en `http://www.binpacking.4fan.cz`.

Na web `https://gianlucacosta.info/TwoBinPack/` atopamos o TwoBinGame, unha aplicación para visualizar o 2D-BPP. É unha ferramenta do mesmo creador do BppGame (para o caso unidimensional) e está dispoñible para Windows e Linux. O funcionamento é similar ao da aplicación para o 1D-BPP: tras elixir o ficheiro cos datos do problema ou un exemplo de proba, o usuario pode "xogar", é dicir, pode desprazar os elementos da parte inferior e colocalos na posición que corresponda da parte superior para realizar os empaquementos que considere. Nesta aplicación, a diferenza do BppGame, cóntase cun límite de tempo. Na Figura 3.2 móstrase un exemplo do seu funcionamento.

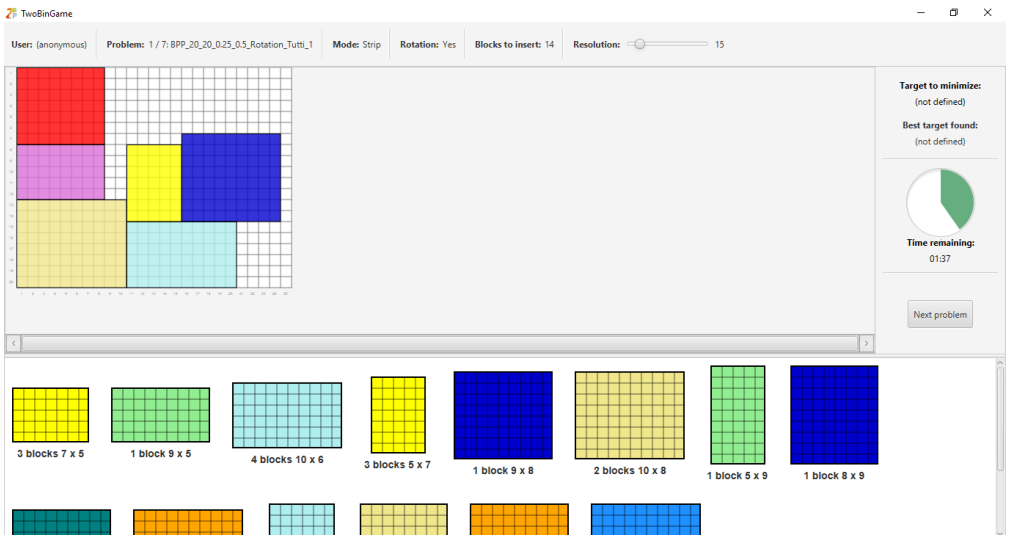

Figura 3.2: Visualización dun problema no TwoBinGame.

Tamén coma no caso unidimensional, o paquete `gbp` de R permítenos resolver o BPP en dúas dimensións. Para iso, a diferenza do caso unidimensional, indicaremos nas variable `l` e `h` as correspondentes anchura/lonxitude e altura. No caso da variable `w` (peso) empregarémola coma unha especie de contador dos elementos asignados a cada recipiente (Yang 2017). Para ilustrar o 2D-BPP imos recuperar dúas instancias que usamos no caso unidimensional: o Exemplo 1 e GuHo4. No caso da instancia GuHo4, empregarémola considerando en ambas dimensións (altura e anchura) os datos que nos proporcionaba dita instancia, é dicir, tanto os recipientes como os elementos serán cadrados. Como o 2D-BPP non ten por que aplicarse unicamente a elementos e recipientes cadrados, os datos da variable anchura para a instancia do Exemplo 1 non coincidirán na súa totalidade cos da variable altura. Polo tanto, temos estas novas instancias, ás que nos referiremos coa mesma denominación acompañada do adxectivo "bidimensional":

$$n = 10, \qquad (H, W)' = (5, 5)', \qquad (h_j, w_j)' \in \begin{pmatrix} 4 & 3 & 3 & 2 & 2 & 2 & 1 & 1 & 1 & 1 \\ 3 & 3 & 2{,}5 & 1{,}75 & 1{,}5 & 1{,}25 & 1 & 1 & 0{,}75 & 0{,}5 \end{pmatrix}$$

$$n = 15, \qquad (H, W)' = (17, 17)', \qquad (h_j, w_j)' \in \begin{pmatrix} 17 & 9 & 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 17 & 9 & 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}.$$

Na Figura 3.3 móstrase a asignación óptima obtida ao aplicar as funcións de `gbp` ao Exemplo 1 bidimensional. Como xa ocorrera para o 1D-BPP, os tres primeiros elementos colocáronse en recipientes diferentes. Xunto ao de maiores dimensións foron asignados outros seis elementos de menores medidas,

no segundo recipiente atópanse o elemento 2 e o 10 e o elemento 3 fica só no terceiro e último contedor necesario para realizar este empaquetamento. Na Figura 3.4 móstrase o empaquetamento óptimo no caso de GuHo4 bidimensional. Vemos que o primeiro elemento ocupa totalmente un recipiente, como era de esperar. No segundo recipiente, atópanse os elementos 2, 3, 4, 6, 7, 8, 9 e 15, sendo preciso un terceiro recipiente para os restantes elementos (5, 10, 11, 12, 13 e 14). Á vista das Figuras, comprobamos que os empaquetamentos non foron realizados en niveis, como fan os métodos que explicamos con anterioridade. Ao facelo en niveis, téndese a desperdiciar espazo nos recipientes, que nalgúns casos serviría para empaquetar máis elementos e minimizar o número de contedores necesarios.

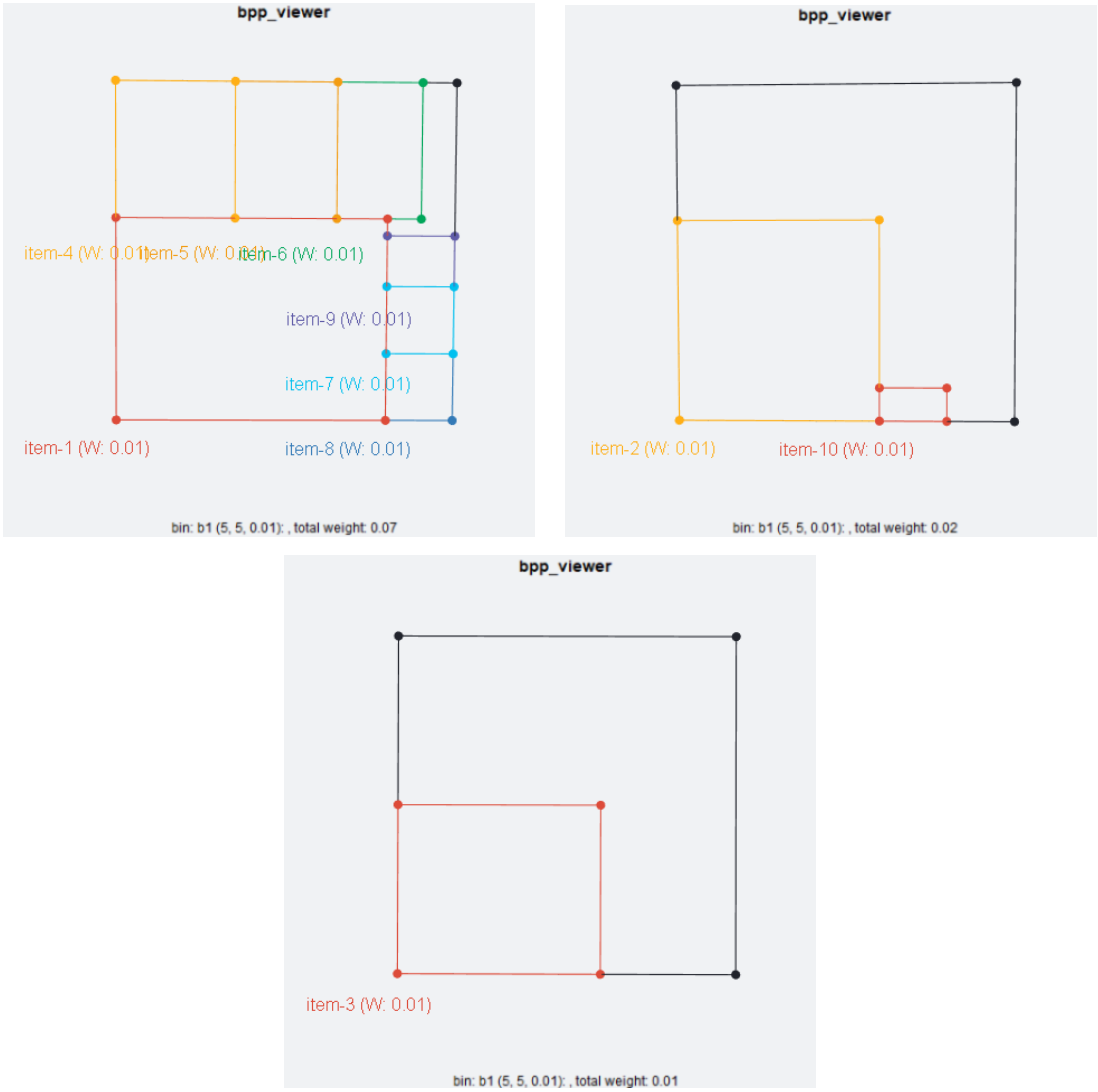

Figura 3.3: Representación dos empaquetamentos obtidos con `gbp` para o Exemplo 1 bidimensional.

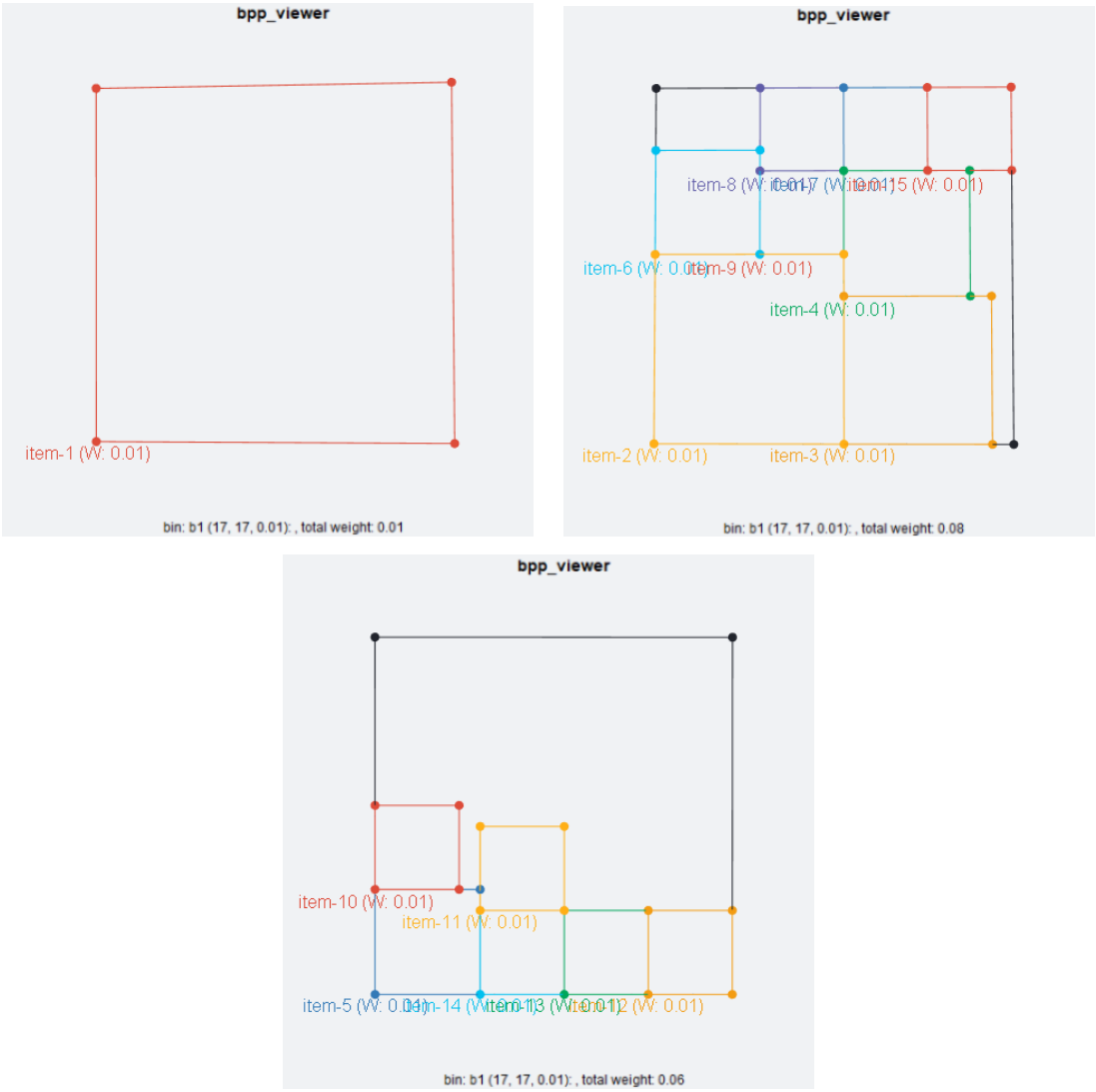

Figura 3.4: Representación dos empaquetamentos obtidos con `gbp` para GuHo4 bidimensional.

3.2. BPP tridimensional (3D-BPP)

3.2.1. Definición e características

Empaquetar obxectos tridimensionais no mínimo número de contedores é o obxectivo do 3D-BPP. A maiores do caso bidimensional, aquí temos a variable profundidade (*depth*), que en numerosas referencias bibliográficas tamén se denomina lonxitude. Ao igual que fixemos nos casos dunha e dúas dimensións, procedemos a dar unha definición formal do 3D-BPP (Martello et al. 2000):

Definición 7 (**3D-BPP**). *Dado un conxunto de n de elementos ou caixas con alturas h_j, anchuras w_j e profundidades d_j $j \in J = \{1, \dots, n\}$, o 3D-BPP consiste en empaquetar todos os elementos en recipientes de idénticas altura H, anchura W e profundidade D usando o menor número de recipientes posible.*

Ademais, asumiremos que as caixas non poden rotarse, é dicir, deben empaquetarse coas caras en paralelo ás do recipiente ao que se asignen. Sen perda de xeneralidade, os valores das variables serán enteiros positivos satisfacendo $h_j \leq H$, $w_h \leq W$ e $d_j \leq D$ ($j \in J$). Ademais, como xeneralización do 1D-BPP, o 3D-BPP tamén é un problema NP-duro en sentido forte.

Problemas relacionados con este ou variantes do mesmo son os seguintes:

- *Strip packing problem* (SPP ou 3D-SPP): Xa vimos a súa relación en dúas dimensións. Para o SPP en 3D, o obxectivo é empaquetar todos os elementos nunha tira de anchura W, profundidade D e altura infinita de xeito que dita altura sexa mínima (Lodi et al. 2002b).
- Carga de mochila: Cada caixa ten asociado un beneficio ou proveito e o obxectivo é escoller o conxunto de caixas que caiban nun só contedor e cuxo proveito sexa máximo. Notemos que esta é a versión en tres dimensións do KP. Se o proveito se corresponde co seu volume, convértese nun problema de minimizar o espazo desocupado ou libre. Este caso particular pode verse como a versión 3D do SSP (*subset-sum problem* ou problema de suma de subconxuntos) (Martello et al. 2000).
- Carga de contedor: Todas as caixas deben ser empaquetadas nun só contedor, que ten unha lonxitude/profundidade infinita. O obxectivo é atopar o empaquetamento que minimice dita lonxitude (Martello et al. 2000).
- Unha variante do 3D-BPP é o chamado *cube packing problem*, no cal tanto elementos como recipientes son cubos (Wäscher et al. 2007).

A cota inferior máis obvia para este problema (ao igual que xa vimos para dimensións menores) procede da relaxación de considerar cada caixa dividida en unidades cúbicas. Isto dá pé a:

$$LB = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j d_j}{HWD} \right\rceil.$$

Outras cotas inferiores que dominan a esta poden obterse dun xeito similar ao caso unidimensional (Martello et al. 2000).

3.2.2. Métodos de resolución

Métodos exactos

Ao igual que en dimensións inferiores, comezamos a exposición de métodos de resolución do 3D-BPP analizando o correspondente modelo de programación matemática. Para iso, botaremos man da formulación desenvolvida en Jin et al. (2003). Dita formulación está condicionada ao que os autores denominan "restricións prácticas", sendo dúas as que máis peso teñen: rotación dos elementos só sobre a dirección da altura, pois é moi habitual na vida real que os obxectos non poidan colocarse sobre un dos seus laterais; e recipientes de medidas heteroxéneas, o que permite unha xeneralización do 3D-BPP máis aló do caso en que todos os recipientes son idénticos (que é o que vimos tratando no presente traballo). Para formular o problema, introdúcense unha serie de parámetros e variables:

- Como xa vimos para o BPP noutras dimensións, os parámetros n e m indican o número elementos e de potenciais recipientes a ser usados, respectivamente.
- w_j, h_j e d_j son enteiros positivos que indican a anchura, altura e profundidade do elemento j, $j \in \{1, \ldots, n\}$.
- W_i, H_i e D_i son enteiros positivos que indican a anchura, altura e profundidade do recipiente i, $i \in \{1, \ldots, m\}$. Sen perda de xeneralidade, a anchura situarase no eixo Y e a profundidade no eixo Y do sistema de coordenadas. No caso de ter recipientes de iguais dimensións, daquela teriamos $W_i = W$, $H_i = H$ e $D_i = D$ para calquera recipiente i.

- x_j, y_j e z_j son as coordenadas de referencia do elemento j nun recipiente.
- dx_j, dy_j, wx_j e wy_j son variables binarias que indican a dirección do elemento j nun recipiente. Por exemplo,

$$dx_j = \begin{cases} 1 & \text{se a profundidade do elemento } j \text{ é paralela ao eixo X} \\ 0 & \text{noutro caso} \end{cases}.$$

  Tense que $dy_j = 1 - dx_j, wx_j = 1 - dx_j$ e $wy_j = dx_j$.
- $a_{jk}, b_{jk}, c_{jk}, d_{jk}, e_{jk}$ e f_{jk} son variables binarias que indican a posición relativa do elemento j respecto do elemento k no mesmo recipiente: lado esquerdo, lado dereito, detrás, diante, enriba e debaixo, respectivamente.
- As variables binarias X_{ji} e Y_i defínense como segue ($j \in \{1, \dots, n\}, i \in \{1, \dots, m\}$):

$$X_{ji} = \begin{cases} 1 & \text{se o elemento } j \text{ se asigna ao recipiente } i \\ 0 & \text{noutro caso} \end{cases} \qquad Y_i = \begin{cases} 1 & \text{se se usa o recipiente } i \\ 0 & \text{noutro caso} \end{cases}$$

- E por último, M é un enteiro moi grande.

Polo tanto, o modelo de programación proposto por Jin et al. (2003) para o 3D-BPP é o seguinte ($j \in \{1, \dots, n\}, i \in \{1, \dots, m\}$):

$$\begin{aligned}
\text{minimizar} \quad & \sum_{i=1}^{m} Y_i \\
\text{suxeito a} \quad & x_j + d_j dx_j + w_j(1 - dx_j) \leq x_k + (1 - a_{jk})M, && \forall j < k, \\
& x_k + d_k dx_k + w_k(1 - dx_k) \leq x_j + (1 - b_{jk})M, && \forall j < k, \\
& y_j + w_j dx_j + d_j(1 - dx_j) \leq y_k + (1 - c_{jk})M, && \forall j < k, \\
& y_k + w_k dx_k + d_k(1 - dx_k) \leq y_j + (1 - d_{jk})M, && \forall j < k, \\
& z_j + h_j \leq z_k + (1 - e_{jk})M, && \forall j < k, \\
& z_k + h_k \leq z_j + (1 - f_{jk})M, && \forall j < k, \\
& a_{jk} + b_{jk} + c_{jk} + d_{jk} + e_{jk} + e_{jk} + f_{jk} \geq X_{ji} + X_{ki} - 1, && \forall j, i < k, \\
& \sum_{i=1}^{m} X_{ji} = 1, && \forall j, \\
& x_j + d_j dx_j + w_j wx_j \leq D_i + (1 - X_{ji})M, && \forall j, i, \\
& y_j + w_j wy_j + d_j dy_j \leq W_i + (1 - X_{ji})M, && \forall j, i, \\
& z_j + h_j \leq H_i + (1 - X_{ji})M, && \forall j, i, \\
& \sum_{j=1}^{n} X_{ji} = 1, && \forall j, \\
& dx_j, dy_j, wx_j, wy_j, a_{jk}, b_{jk}, c_{jk}, d_{jk}, e_{jk}, f_{jk}, X_{ji}, Y_i \in \{0, 1\}, \\
& x_j, y_j, z_j \in \mathbb{Z}.
\end{aligned}$$

De novo, a función obxectivo é minimizar o número de recipientes a usar. En canto ás restricións, as seis primeiras garanten que os elementos non se solapen, a sétima indica que a revisión por solapamento ten lugar unicamente se os dous elementos están no mesmo recipiente, a oitava asegura que cada elemento quede empaquetado nun único recipiente, as tres seguintes son as condicións para o empaquetamento do elemento j no recipiente i e a derradeira igualdade indica que un recipiente é usado cando un elemento é empaquetado nel.

Métodos heurísticos

En Lodi et al. (2002b) exponse un método heurístico para a resolución do 3D-BPP. Trátase do *Height first-Area second* (HA) (*Altura primeiro-Área segundo*), que empaqueta os elementos en *capas* (*layers*). A parte inferior ou *chan* da primeira capa coincide coa base do recipiente e os elementos son empaquetados coa súa base nela. O chan de cada nova capa vén determinada pola altura do elemento máis alto da capa inferior. Isto recórdanos ao empaquetamento en niveis do 2D-BPP. Pero a consecución dun empaquetamento efectivo nunha capa trae consigo dous problemas, que xeralmente poden dar pé a conflitos. Por unha parte, se se quere un bo enchido en vertical, tratarase de empaquetar a mesma capa elementos cunha altura similar. Por outro lado, se o que queremos é un bo enchido en horizontal, teremos que resolver un 2D-BPP coas bases dos elementos. O que fai a heurística HA é tomar a mellor das solucións destes dous problemas.

En Sweep (2003) explícase unha estratexia de empaquetamento coñecida como *enfoche nivel-rebanda* (*level-slice approach*). Consiste en dividir os recipientes tanto horizontal (ao longo da altura do recipiente) como verticalmente (ao longo da profundidade do recipiente). As porcións horizontais resultantes denomínanse *niveis* e as verticais, *rebandas* e poden verse como os andares dun edificio e as rebandas dun molete de pan, respectivamente. A intersección dun nivel cunha rebanda é o que se chama *nivel-rebanda*. Estas intersección constitúen subdivisións do recipiente orixinal e poden interpretarse como recipientes unidimensionais, pois tanto a altura como a profundidade están fixadas dalgún modo e queda como única variable a anchura. Tendo en conta isto, aplícanse algoritmos para o 1D-BPP xurdido ao considerar as anchuras dos obxectos, coa restrición de que a suma das anchuras non supere a anchura do recipiente. As alturas e profundidades das nivel-rebandas vanse axustando ás dos obxectos máis alto e/ou máis profundos que se asignen nelas. Cando non é posible empaquetar ningún elemento na nivel-rebanda actual, créase unha nova. O proceso de creación é como segue:

- Se o nivel actual ten suficiente profundidade, créase unha nova nivel-rebanda coa mesma altura que a anterior pero cuxa profundidade comeza a contarse a partir de onde finaliza a anterior. Se se asigna un obxecto cuxa altura supere a da nivel-rebanda anterior, axústase a altura de todo o nivel a esta nova medida.

- Se non hai suficiente profundidade no nivel actual, iníciase unha nova nivel-rebanda á altura determinada polo obxecto máis alto do nivel previo (é dicir, a altura do propio nivel).

- O proceso continúa ata que non hai espazo no recipiente para crear novas nivel-rebandas.

Como xa comentamos, os subproblemas que se resolven son 1D-BPP, polo que se poden aplicar os algoritmos destinados a resolver este tipo de problemas. Porén, o enfoque nivel-rebanda enfróntase a dous grandes inconvenientes: obxectos con excesiva profundidade, o pode provocar un desperdicio de espazo en cada rebanda; e obxectos con excesiva altura, que dá pé a niveis con moito espazo malgastado. Isto pode resolverse, ou cando menos reducirse, ordenando os obxectos primeiro por altura e despois por profundidade.

Tendo en conta a división en rebandas dos recipientes, Maarouf et al. (2008) desenvolven a heurística *Peak Filling Slice Push* (PFSP), que consta de dous pasos:

- O recipiente divídese en rebandas, que son enchidas coa técnica *peak filling* ou recheo máximo. Isto consiste en crear "subrebandas" cada vez que se coloca un elemento na rebanda para repetir o proceso nela e completar a rebanda en altura. Cando non se poden crear máis subrebandas, o

método vai completando as demais mentras sexa posible. O algoritmo trata de colocar os elementos de maiores dimensións na base das rebandas, reservando os máis pequenos para completar as subrebandas posteriormente.

- Unha vez enchida cada rebanda, empúrrase contra as anteriores co fin de comprimir os obxectos sen que se solapen e minimizar, na medida do posible, o espazo sen usar. Este é o chamado *push method* ou método de empuxe.

3.2.3. Recursos e exemplos

Para a visualización do 3D-BPP contamos cunha aplicación na web *3D Bin Packing* (`http://www.3dbinbox.com/App/Views/index.html`), cuxas interfaces podemos ver na Figura 3.5. En primeiro lugar, podemos optar por introducir os nosos propios datos (*blank optimization*) ou seleccionar o exemplo que se proporciona (*example data optimization*) (imaxe superior). Nesta primeira interface tamén podemos consultar e descargar o código empregado, programado en linguaxe C. En ambos casos, a nosa selección lévanos a unha nova interface onde podemos introducir ou ver os parámetros do problema, en función da escolla realizada (imaxe central). Podemos escoller empaquetar en palés ou en contedores de varias medidas. Nunha última pantalla, podemos ver a representación gráfica do empaquetamento en cada contedor, que podemos rotar e, ademais, podemos ver numericamente a disposición dos elementos nos recipientes en varias táboas e observar a distribución do espazo (ocupado e baleiro) dos recipientes en gráficos de sectores (imaxe inferior).

Ao igual que para o 1D-BPP e o 2D-BPP, o paquete `gbp` de R permítenos resolver o BPP en tres dimensións. Neste caso, indicaremos nas variables `l`, `h` e `d` as correspondentes anchura/lonxitude, altura e profundidade, deixando a variable `w` (peso) coma unha especie de contador dos elementos asignados a cada recipiente (Yang 2017). Ao igual que fixemos na caso bidimensional, aproveitamos as instancias Exemplo 1 e GuHo4 coas correspondentes modificacións para ilustrar o 3D-BPP. Así, ditas instancias "tridimensionais" son, respectivamente,

$$n = 10, \qquad (H, W, D)' = \begin{pmatrix} 5 \\ 5 \\ 4 \end{pmatrix}, \qquad (h_j, w_j, d_j)' \in \begin{pmatrix} 4 & 3 & 3 & 2 & 2 & 2 & 1 & 1 & 1 & 1 \\ 3 & 3 & 2{,}5 & 1{,}75 & 1{,}5 & 1{,}25 & 1 & 1 & 0{,}75 & 0{,}5 \\ 3 & 2 & 2{,}5 & 1{,}75 & 1{,}25 & 1 & 1 & 0{,}5 & 0{,}5 & 0{,}5 \end{pmatrix}$$

$$n = 15, (H, W, D)' = \begin{pmatrix} 17 \\ 17 \\ 17 \end{pmatrix}, (h_j, w_j, d_j)' \in \begin{pmatrix} 17 & 9 & 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 17 & 9 & 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 17 & 9 & 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}.$$

Obsérvese que instancia GuHo4 tridimensional dá lugar a un *cube packing problem*, xa que tanto elementos como recipientes son cubos. Aplicando os comandos do paquete `gbp` de R (Yang 2017), obtemos os empaquetamentos óptimos para ambas instancias, como mostran as Figuras 3.6 e 3.7, respectivamente. No caso do Exemplo 1 tridimensional, son necesarios dous recipientes para empaquetar os dez elementos: nove están no primeiro recipiente e o elemento 3 asignouse só ao segundo recipiente. De igual modo, para GuHo4 tridimensional tamén se precisan dous para os 15 elementos cúbicos: un ocupado completamente polo primeiro elemento e outro para os 14 restantes.

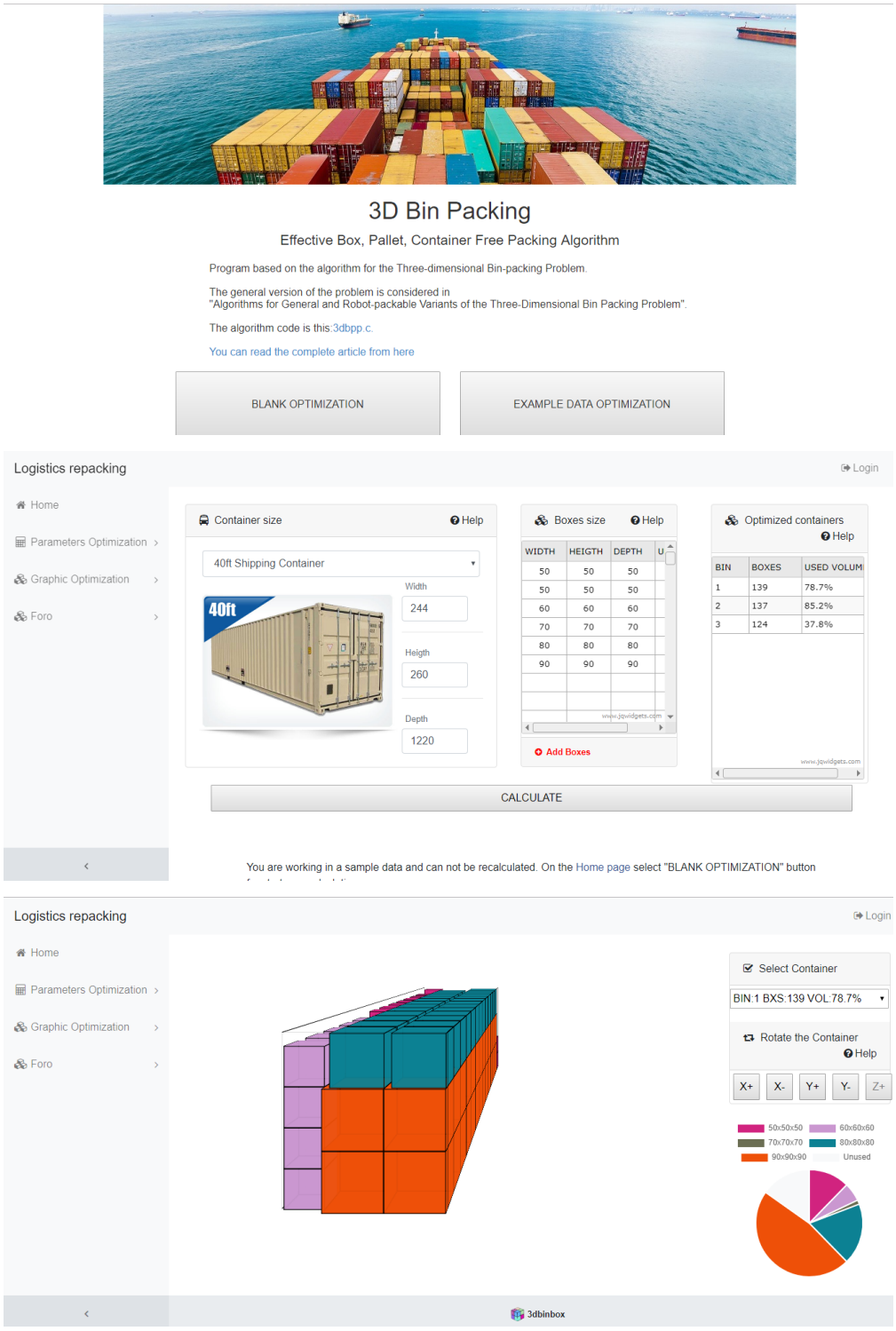

Figura 3.5: Visualización do exemplo en *3D Bin Packing*.

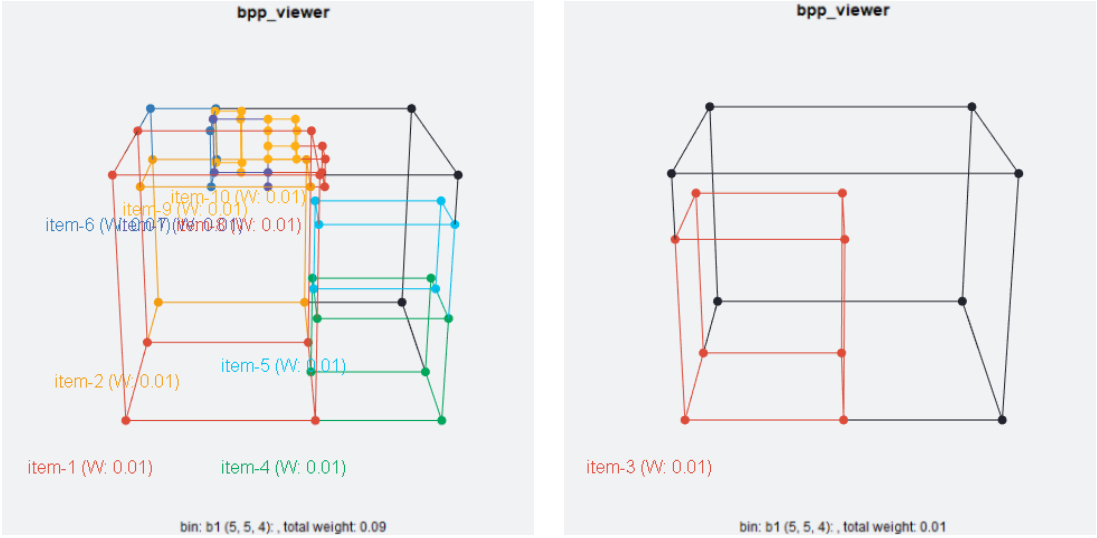

Figura 3.6: Representación dos empaquetamentos obtidos con `gbp` para o Exemplo 1 tridimensional.

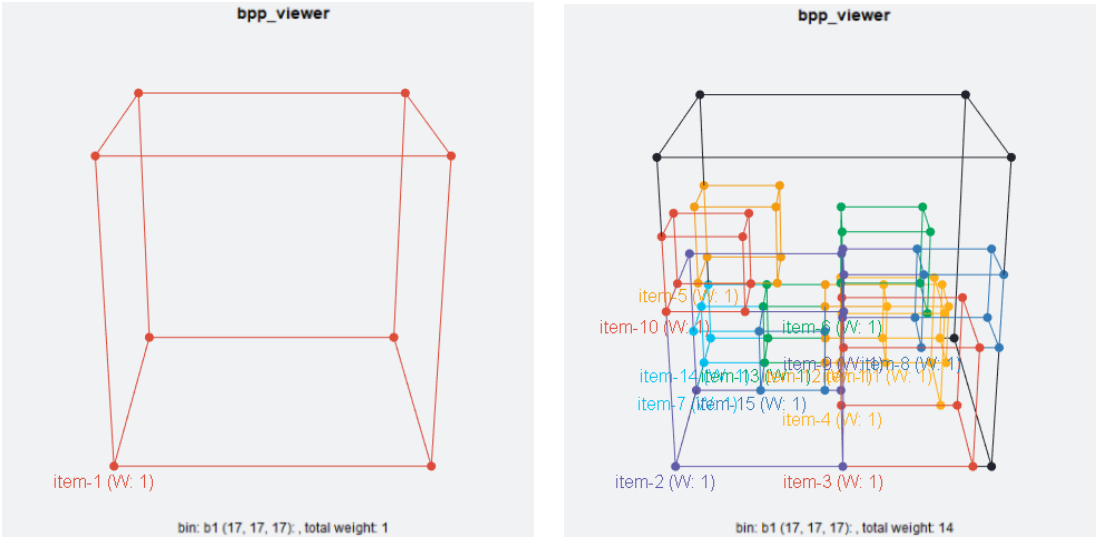

Figura 3.7: Representación dos empaquetamentos obtidos con `gbp` para GuHo4 tridimensional.

3.3. BPP en 4 dimensións (4D-BPP)

Este problema vén a ser unha variante do 3D-BPP onde se engade unha cuarta variable ao problema orixinal. Pode estar relacionada co tempo, coma no caso de caixas que teñan que empaquetarse en contedores en certos períodos de tempo sen interrupcións ou como a cocción de pan nun forno (Dyckhoff 1990). Tamén pode tratarse dunha variable que indique un certo valor ou proveito que ten cada obxecto e que debe maximizarse ao empaquetalos nos recipientes, ou ben ser restricións de peso (Yang 2017). Neste último caso, cada obxecto tridimensional conta agora cunha nova variable que indica a cantidade de materia que o conforma. Pola súa banda, os recipientes, que no 3D-BPP limitaban o espazo ao ancho, ao alto e en profundidade, engaden unha limitación no peso máximo que poderían soportar. Así pois, podemos tratar de definir de xeito formal o problema, ao estilo do feito en dimensións menores:

Definición 8 (**4D-BPP**). *Dado un conxunto de n de elementos ou caixas con alturas h_j, anchuras w_j, profundidades d_j e pesos ou valores p_j $j \in J = \{1, \dots, n\}$, o 4D-BPP consiste en empaquetar todos os elementos en recipientes con idénticos parámetros de altura H, anchura W, profundidade D e peso ou valor máximo permitido P usando o menor número de recipientes posible.*

3.3.1. Recursos e exemplos

Como ocorría nas dimensións inferiores, o paquete `gbp` tamén nos permite a resolución do BPP en catro dimensións. Neste caso empregaremos as catro variables `l`, `h`, `d` e `w` os correspondentes datos de anchura/lonxitude, altura, profundidade e peso (ou valor) (Yang 2017). Para ilustrar esta variante do BPP, botamos man de novo das instancias Exemplo 1 e GuHo4 tridimensionais, ás que engadimos as variables p_j e P que indicarán neste caso o peso dos elementos o peso máximo que soportan os recipientes, respectivamente. Para ditos pesos, usamos como datos a suma das demais medidas. As novas instancias de catro dimensións serán:

$$n = 10, \quad \begin{pmatrix} H \\ W \\ D \\ P \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 4 \\ 14 \end{pmatrix}, \quad \begin{pmatrix} h_j \\ w_j \\ d_j \\ p_j \end{pmatrix} \in \begin{pmatrix} 4 & 3 & 3 & 2 & 2 & 2 & 1 & 1 & 1 & 1 \\ 3 & 3 & 2{,}5 & 1{,}75 & 1{,}5 & 1{,}25 & 1 & 1 & 0{,}75 & 0{,}5 \\ 3 & 2 & 2{,}5 & 1{,}75 & 1{,}25 & 1 & 1 & 0{,}5 & 0{,}5 & 0{,}5 \\ 10 & 8 & 8 & 5{,}5 & 4{,}75 & 4{,}25 & 3 & 2{,}5 & 2{,}25 & 2 \end{pmatrix}$$

$$n = 15, \quad \begin{pmatrix} H \\ W \\ D \\ P \end{pmatrix} = \begin{pmatrix} 17 \\ 17 \\ 17 \\ 51 \end{pmatrix},$$

$$\begin{pmatrix} h_j \\ w_j \\ d_j \\ p_j \end{pmatrix} \in \begin{pmatrix} 17 & 9 & 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 17 & 9 & 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 17 & 9 & 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 51 & 27 & 21 & 18 & 15 & 15 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \end{pmatrix}.$$

Na Figura 3.8 móstrase o resultado do empaquetamento para o Exemplo 1 4-dimensional, que emprega un total de 4 recipientes, mentres que na Figura 3.9 temos a asignación para GuHo4 en 4 dimensións, onde o número óptimo é 6 recipientes. Observamos que en ambos casos todos os recipientes dispoñen dunha gran cantidade de espazo sen utilizar debido á restrición no peso, alcanzándose a cantidade máxima nun só caso (recipiente 1 de GuHo4). Esta limitación no peso tamén contribúe a que o número de elementos por recipiente sexa reducido, a diferenza dos respectivos casos tridimensionais, nos que un só recipiente albergaba a práctica totalidade dos recipientes.

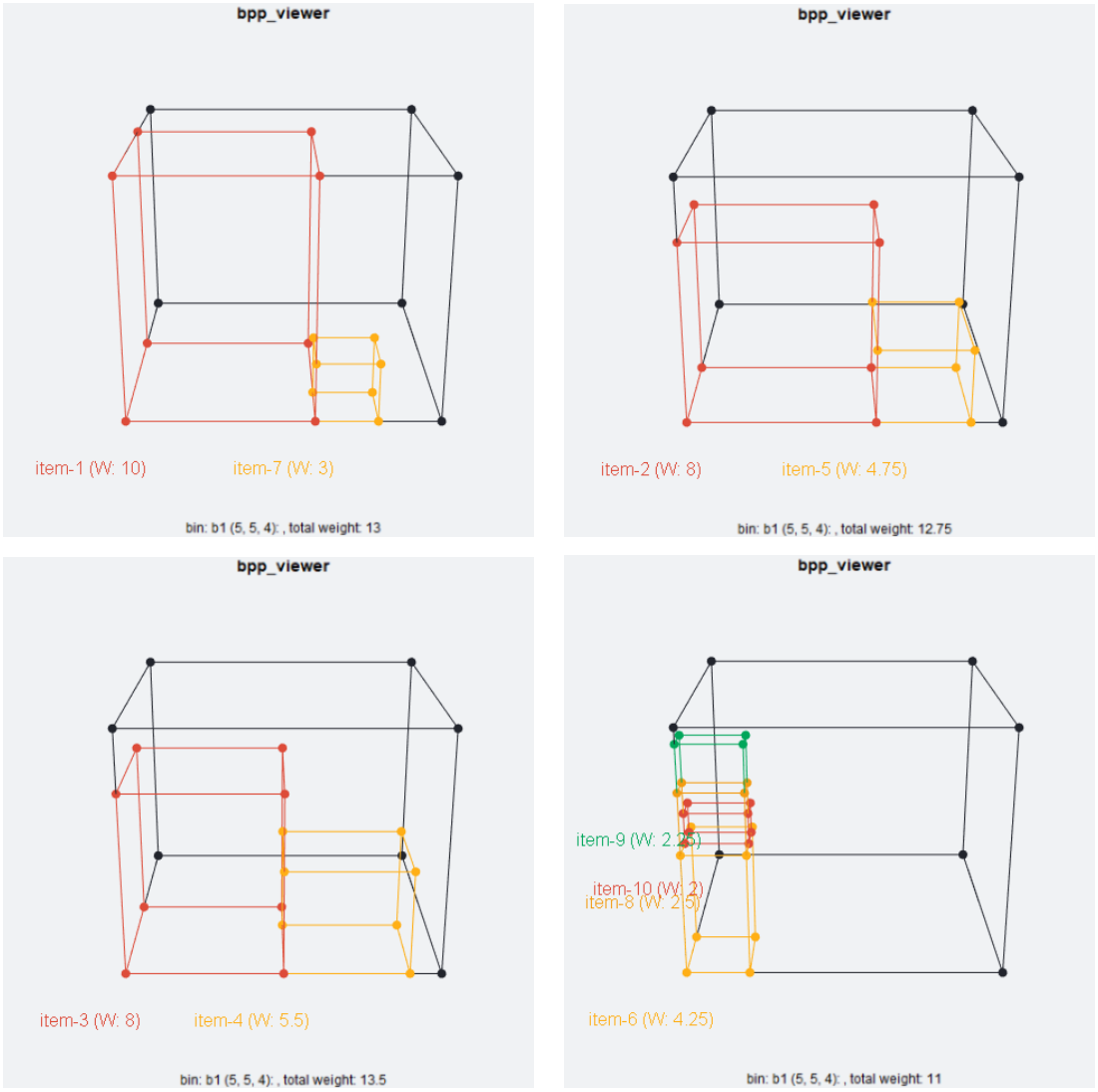

Figura 3.8: Representación dos empaquetamentos obtidos con `gbp` para o Exemplo 1 4-dimensional.

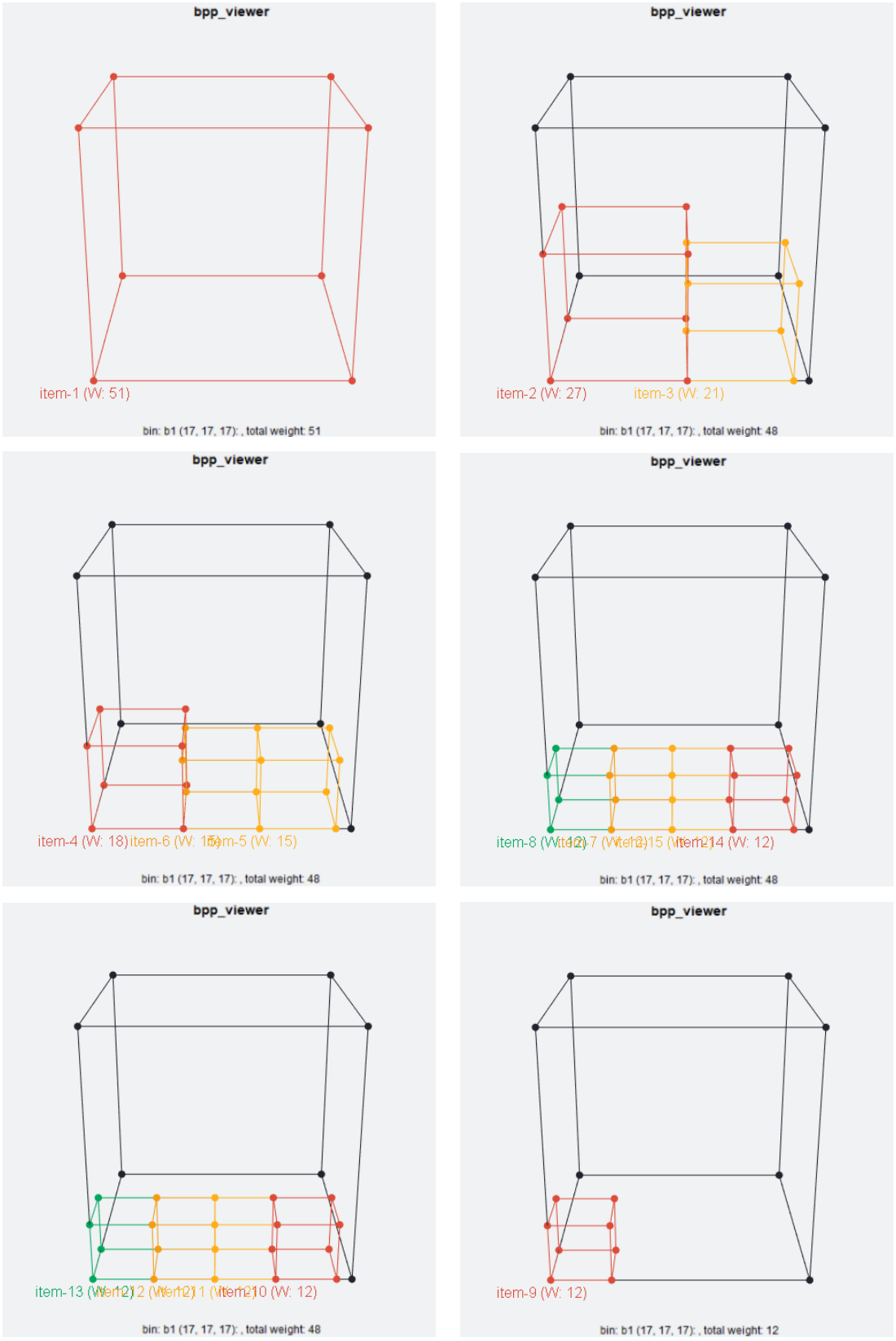

Figura 3.9: Representación dos empaquetamentos obtidos con `gbp` para GuHo4 4-dimensional.

3.4. *Vector bin packing* (VBP)

O *vector bin packing* (VBP) é unha xeneralización do BPP a múltiples dimensións, onde pasamos a considerar os elementos e os recipientes como vectores. A definición formal deste problema é a seguinte (Csirik et al. 1990, Johnson 2016)):

Definición 9. *Dado un enteiro $d \geq 1$ e un conxunto $\{x_1, \dots, x_n\}$ de elementos que son vectores d-dimensionais da forma $(x_{j1}, \dots, x_{jd}), j \in \{1, \dots, n\}$ con $x_{ji} \in [0, 1], i \in \{1, \dots, d\}$, daquela o VBP consiste en asignar os elementos ao mínimo número de recipientes d-dimensionais con capacidade unitaria en cada dimensión, de tal xeito que o vector formado pola suma dos elementos de cada recipiente non exceda ao vector unitario d-dimensional.*

Se tomamos $d = 1$, atopariámonos no caso 1D-BPP. Cómpre dicir que ao longo do presente traballo non esiximos que os elementos do BPP nas súas distintas variantes tivesen tamaños no intervalo $[0, 1]$ nin recipientes con medidas unitarias, mais parece que é o habitual no VBP, segundo a bibliografía consultada (Csirik et al. 1990, Johnson 2016).

Algúns dos métodos habituais de resolución do VBP proveñen de xeneralizar o algoritmo FFD (descrito no seu momento para o 1D-BPP) a múltiples dimensións. Csirik et al. (1990) considera o algoritmo *generalized first-fit decreasing* (GFFD), consistente en dous pasos:

1. Ordénanse os elementos como segue:

$$x_{1,\text{máx}} \geq \cdots \geq x_{n,\text{máx}},$$

   onde $x_{j,\text{máx}} = \text{máx}_{i=1}^{d} x_{j,i}$, $j \in \{1, \dots, n\}$, é dicir, que se ordenan os elementos en orde non crecente do máximo das súas compoñentes.

2. Aos elementos así ordenados, aplícaselles a estratexia FF (*first-fit*), coma se fose o caso 1D-BPP.

A este algoritmo tamén se lle denomina FFDmax. Seguindo con este criterio de definir "ordes non crecentes", xorden outros algoritmos (Jonhson 2016):

- FFDsum: os elementos son ordenados en orde non crecente da suma das súas coordenadas $\sum_{i=1}^{d} x_{j,i}, j \in \{1, \dots, n\}$.
- FFDprod: os elementos son ordenados en orde non crecente do produto das súas coordenadas $\prod_{i=1}^{d} x_{j,i}, j \in \{1, \dots, n\}$.
- FFDlex: os elementos son ordenados seguindo a orde lexicográfica: x_j precede a $x_{j'}$ se $x_{ji} = x_{j'i}, i \in$ ou se existe $i^* \leq d$ tal que $x_{ji} = x_{j'i}, i \in \{1, \dots, i^* - 1\}$ e $x_{ji^*} < x_{j'i^*}$.

Podemos definir de xeito análogo os algoritmos BFDmax, BFDsum, BFDprod e BFDlex, xeneralizacións do *best-fit decreasing* (BFD), substituíndo a estratexia FF pola BF á hora de ordenar os elementos.

Capítulo 4

Conclusións

No presente traballo vimos de tratar o problema de empaquetamento en recipientes ou *bin packing problem* (BPP). Fixemos moito fincapé na versión máis coñecida do mesmo, a unidimensional (1D-BPP), tamén chamado BPP clásico. Explicamos diversos métodos para a súa resolución, aplicando a unha pequena colección de instancias algúns deles, ben fose tras obter o seu código e revisar as súas características como tras programalos por completo ou modificar pequenos detalles. Introducimos ademais o problema en máis dunha dimensión, describindo brevemente algúns algoritmos e ilustrando cada caso co correspondente exemplo. Para o BPP en cada dimensión, tamén fixemos unha revisión dos recursos dispoñibles en Internet para o seu uso e aprendizaxe (códigos, instancias e visualizadores).

Polo que puidemos comprobar, o 1D-BPP é, con diferenza, a versión do BPP que conta con máis bibliografía e recursos, algo que non se repite do mesmo modo para dimensións maiores. Ademais, a pesares de ser un problema que encaixa en numerosas situacións da vida cotiá, non parece ser tan coñecido coma outros problema de empaquetamento (p.e. o problema da mochila). Por iso, as futuras liñas de traballo neste senso poderían estar orientadas do seguinte xeito:

- Afondar en moitos aspectos brevemente mencionados ou que quedaron por incluír neste traballo como, por exemplo, a relación do BPP con outros problemas de optimización (rutas de vehículos...).
- Ampliar o material, tanto bibliográfico como didáctico, do BPP en varias dimensións.
- Crear novas aplicacións para dispositivos móbiles sobre este problema, que permita o seu coñecemento e estudo a un amplo abano de público, dun xeito similar ao éxito que tivo (e segue tendo) o *Tetris*.
- Crear un novo paquete de R que recolla algúns dos métodos de resolución aproximada e heurísticos, á semellanza do `gbp` para a solución exacta.
- Con todo o precedente, ampliar o contido deste TFM para convertelo nunha gran revisión bibliográfica sobre o BPP.

Apéndice A

Códigos en AMPL

O habitual ao traballar con AMPL é construír tres ficheiros con extensións .mod (modelo do problema), .dat (datos que se usarán) e .run (comandos de resolución e de visualización de resultados).

Formulación 1D-BPP e variantes

###Ficheiro .mod para o 1D-BPP###

```
param n>=0, integer;
param m>=0, integer;
param c >=0, integer;
set elementos := 1..n;
set recipientes := 1..m;
param t {j in elementos};
var y {i in recipientes} binary;
var x {(i,j) in recipientes, elementos} binary;
#Formulación do problema
minimize z: sum{i in recipientes} y[i];
subject to restriccion1 {i in recipientes}: sum{j in elementos} t[j]*x[i,j]<=c*y[i];
subject to restrición2 {j in elementos}: sum{i in recipientes} x[i,j]=1;
```

###Ficheiro .dat para o 1D-BPP###

```
param n:=10;
param m:=6;
param c:=5;
param t:=
1 4
2 3
3 3
4 2
5 2
6 2
7 1
8 1
9 1
10 1;
```

###Ficheiro .run para o 1D-BPP###

```
solve;
#Número óptimo de recipientes
display z;
#Recipientes usados
display y;
#Tamaños dos elementos
display t;
#Disposición dos elementos nos recipientes
display x;
```

Para os problemas CSP, KS e SSP, soamente expoñemos os ficheiros do modelo, xa que os arquivos de datos e de comandos son moi similares aos do 1D-BPP:

###Ficheiro .mod para o CSP###

```
param u>=0, integer;
param m>=0, integer;
param c >=0, integer;
set tipos := 1..u;
set recipientes := 1..m;
param t {j in tipos};
param d {j in tipos};
var y {i in recipientes} binary;
var xi {(i,j) in {recipientes, tipos}}>=0 integer;
#formulación do problema
minimize z: sum{i in recipientes} y[i];
subject to restriccion1 {i in recipientes}: sum{j in tipos} t[j]*xi[i,j]<=c*y[i];
subject to restrición2 {j in tipos}: sum{i in recipientes} xi[i,j]=d[j];
```

###Ficheiro .mod para o KS###

```
param n>=0, integer;
param c>=0, integer;
set elementos := 1..n;
param t {j in elementos};
param p {j in elementos};
var x {j in elementos} binary;
#Formulación do problema
maximize z: sum{j in elementos} p[j]*x[j];
subject to restriccion: sum{j in elementos} t[j]*x[j]<=c;
```

###Ficheiro .mod para o SSP###

```
param n>=0, integer;
param c>=0, integer;
set elementos := 1..n;
param t {j in elementos};
var x {j in elementos} binary;
#Formulación do problema
maximize z: sum{j in elementos} t[j]*x[j];
subject to restriccion: sum{j in elementos} t[j]*x[j]<=c;
```

Apéndice B

Códigos en R

Cota LB_2

Para o cálculo desta cota, construímos unha función que toma como parámetros de entrada a capacidade c e o vector de tamaños t_j.

```
LB2=function(c,t_j){
a=seq(0,c/2,1)
L=numeric(length(a))
for(i in 1:length(a)){
J1=which(t_j[]>c-a[i])
J2=which(c/2<t_j[]&t_j[]<=c-a[i])
J3=which(c/2>=t_j[]&t_j[]>=a[i])
aux=sum(t_j[J3])-(length(J2)*c-sum(t_j[J2]))
L[i]=length(J1)+length(J2)+max(0,ceiling(aux/c))
}
LB=max(L)
return(LB)
}
```

Algoritmos *fit*

Coa axuda do software estatístico R, programamos os algoritmos NF e FF e as respectivas versións *decreasing* (é dicir, cos tamaños dos obxectos ordenados de xeito decrecente):

```
## Algoritmo Next-Fit (Decreasing)
NF=function(c,t_j){
t_j=sort(t_j,decreasing=T) #Para o NFD ou en caso de que non estean xa ordenados os tamaños
l_alpha=0  #inicializamos a carga
m=1  #inicializamos o número de recipientes
b=numeric(length(t_j))  #vector de recipientes
for(j in 1:length(t_j)){
l_alpha=l_alpha+t_j[j]  #engadimos un novo obxecto
if(l_alpha<=c){  #comprobamos se a carga supera a capacidade
b[j]=m  #asignamos o obxecto j ao recipiente m
}
else{  #senón
```

```
m=m+1; #inicializamos un novo recipiente
l_alpha=t_j[j]; #a carga será o tamaño do obxecto j
b[j]=m; #asignamos j ao novo recipiente
}
}
sol=matrix(c(t_j,b),nrow=2,ncol=length(t_j),byrow=T)
rownames(sol)=c("t_j","b")
return(sol)
}


##################################
## Algoritmo First-Fit (Decreasing)
FF=function(c,t_j){
t_j=sort(t_j,decreasing=T) #Para o FFD ou en caso de que os tamaños non estean ordenados
pesos=t_j  #Duplicamos o vector de tamaños
m=length(t_j)  #Número inicial de recipientes dispoñibles
b=numeric(length(t_j))  #vector indicador de recipientes
l_b=numeric(m) #vector de cargas de recipientes
for(j in 1:length(t_j)){
for(i in 1:m){
if(t_j[j]!=0){  #non consideramos os obxectos
#xa asignados
l_b[i]=l_b[i]+t_j[j];  #engadimos un novo obxecto
if(l_b[i]<=c){  #comprobamos que a carga non
#supera a capacidade
b[j]=i;  #asignamos o obxecto j ao
#recipiente i
t_j[j]=0  #"borramos" o tamaño do
#obxecto j xa asignado para
#non telo en conta posteriormente
}
else{ #senón
l_b[i]=l_b[i]-t_j[j]; #retirámolo
}
}
}
}
sol=matrix(c(pesos,b),nrow=2,ncol=length(t_j),byrow=T)
rownames(sol)=c("t_j","b")
return(sol)
}
```

Apéndice C

Códigos en C

Algoritmo *worst-fit decreasing*

Modificando algúns detalles do código en linguaxe C do algoritmo *best-fit decreasing* dispoñible na web *Geek for geeks*, creamos a versión para a estratexia *worst-fit*.

```
int worstFitDec(int weight[], int n, int c)
{
// Ordénanse os elementos en orde decrecente, se non o están
sort(weight, weight+n, greater<int>());

// Iníciase contador de recipientes
int res = 0;

// Créase un vector para o espazo restante nos recipientes
// Como moito, n recipientes
int bin_rem[n];

// Asignación de recipientes
for (int i=0; i<n; i++)
{
// Búscase o recipiente no que caiba o recipiente
int j;

// Iníciansе indicador de máximo espazo dispoñible e
// índice de recipiente
int max = 1, bi = 0;

for (j=0; j<res; j++)
{
if (bin_rem[j] >= weight[i] && bin_rem[j] + weight[i] > max)
{
bi = j;
max = bin_rem[j] + weight[i];
}
}

// Se non hai un recipiente no que colocar o elemento
```

```
// iníciase un novo
if (max==1)
{
bin_rem[res] = c - weight[i];
res++;
}
else // Asígnase ao mellor recipiente
bin_rem[bi] -= weight[i];
}
return res;
}
```

Bibliografía

[1] Alvim ACF, Ribeiro CC (2004) A hybrid bin-packing heuristic to multiprocessor scheduling. En: Ribeiro CC, Martins SL (ed) Experimental and Efficient Algorithms. WEA 2004. Lecture Notes in Computer Science. Springer, Berlin, pp 1-13.

[2] Alvim ACF, Ribeiro CC, Glover F, Aloise DJ (2004) A hybrid improvement heuristic for the one-dimensional bin packing problem. Journal of Heuristics 10:205-229.

[3] Boyar J, Epstein L, Favrholdt LM, Kohrt JS, Larsen KS, Pedersen MM, Wøhlk S (2006) The maximum resource bin packing problem. Theoretical Computer Science 362:127-139.

[4] Coffman Jr EG, Garey MR, Johnson DS (1987) Bin packing with divisible item sizes. Journal of Complexity 3:406-428.

[5] Coffman Jr EG, Garey MR, Johnson DS (1996) Bin packing approximation algorithms: A survey. Approximation Algorithms for NP-Hard Problems, 46-93.

[6] Coffman Jr EG, Csirik J, Johnson DS, Woeginger GJ (2004) An introduction to bin packing. https://www.inf.u-szeged.hu/∼csirik/ed5ut.pdf. Accedido 10 de xaneiro de 2019.

[7] Csirik J, Frenk JBG, Labbé M, Zhang S (1990) On the multidimensional vector bin packing. Acta Cybernetica 9:361-369.

[8] Delorme M, Iore M, Martello S (2016) Bin packing and cutting stock problems: mathematical models and exact algorithms. European Journal of Operational Research 255:1-20.

[9] Delorme M, Iore M, Martello S (2018) BPPLIB: A library for bin packing and cutting ctock problems. Optimization Letters 12:235-250.

[10] Dyckhoff H (1990) A typology of cutting and packing problems. European Journal of Operational Research 44:145-159.

[11] Escamilla LAS, Zacatelco HC, de la Rosa Flores R (2017) Implementación del algoritmo MBS para resolver instancias de bin packing. Research in Computing Science 134:45-53.

[12] Falkenauer E (1996) A hybrid grouping genetic algorithm for bin packing. Journal of heuristics 2:5-30.

[13] Fleszar K, Hindi KS (2002) New heuristics for one-dimensional bin-packing. Computers & Operations Research 29:821-839.

[14] Gupta JND, Ho JC (1999) A new heuristic algorithm for the one-dimensional bin-packing problem. Production Planning & Control 10:598-603.

[15] Jin Z, Ito T, Ohno K (2003) The three-dimensional bin packing problem and its practical algorithm. JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing 46:60-66.

[16] Jonhson DS (2016) Vector Bin Packing. En: Kao MY (ed) Encyclopedia of Algorithms. Springer, Nova York.

[17] Korf RE (2008, xullo) A new algorithm for optimal bin packing. Aaai/Iaai 731-736.

[18] Lodi A, Martello S, Monaci M (2002) Two-dimensional packing problems: A survey. European Journal of Operational Research 141:241-252.

[19] Lodi A, Martello S, Vigo D (2002) Heuristic algorithms for the three-dimensional bin packing problem. European Journal of Operational Research 141:410-420.

[20] Lodi A, Martello S, Vigo D (2004) Models and Bounds for Two-Dimensional Level Packing Problems. Journal of Combinatorial Optimization 8:363-379.

[21] Loh KH, Golden B, Wasil E (2008) Solving the one-dimensional bin packing problem with a weight annealing heuristic. Computers & Operations Research 35:2283-2291.

[22] Maarouf WF, Barbar AM, Owayjan MJ (2008) A new heuristic algorithm for the 3D bin packing problem. Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering 342-345.

[23] Martello S, Pisinger D, Vigo D (2000) The three-dimensional bin packing problem, Operations Research 48:256-267.

[24] Martello S, Toth P (1990) Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons, Chichester.

[25] Mohamed M, Mohamed T, Billal R (2016) Modeling and solving the open-end bin packing problem. International Journal of Advanced Computer Science and Applications 7:399-404.

[26] Morrison DR, Jacobson SH, Sauppe JJ, Sewell EC (2016) Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optimization, 19:79-102.

[27] Nurmela KJ, Östergård PR (1997) Packing up to 50 equal circles in a square. Discrete & Computational Geometry 18:111-120.

[28] Scholl A, Klein R, Jürgens C (1997) BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers & Operations Research 24:627-645.

[29] Schweer N (2010) Algorithms for packing problems. Cuvillier.

[30] Sweep S (2003) Three Dimensional Bin-Packing Issues and Solutions. University of Minnesota.

[31] Valério de Carvalho JM (2002) LP models for bin packing and cutting stock problems. European Journal of Operational Research 141:253-273

[32] Wäscher G, Haußner H, Schumann H (2007) An improved typology of cutting and packing problems. European Journal of Operational Research 183:1109-1130.

[33] Yang G (2017) gbp: A Bin Packing Problem Solver. R package version 0.1.0.4 https://cran.r-project.org/web/packages/gbp/index.html. Accedido 1 de febreiro de 2019.