Figueroa Mata, Geovanni; Carrera Retana, Ernesto.
Un problema tipo *bin-packing*
Tecnología en Marcha, *Vol. 24, N.° 2, Abril-Junio 2011, P. 34-43*

Un problema tipo *bin-packing*

Fecha de recepción: 28/08/2010
Fecha de aceptación: 27/09/2010

Geovanni Figueroa Mata[1]
Ernesto Carrera Retana[2]

Palabras clave

Optimización combinatoria, heurísticas, algoritmos probabilísticos, programación entera.

Resumen

Se presentan dos algoritmos heurísticos para resolver un problema de tipo bin-packing en el cual se quiere guardar objetos de n tipos diferentes (en donde la demanda del objeto de tipo i-ésimo está dada por d_i) en m tipos de recipientes con capacidades no necesariamente iguales; todos los recipientes de tipo j =1,2,...,m deben tener la misma distribución de los objetos. El objetivo es minimizar el costo de los recipientes por utilizar, donde el costo de cada tipo de recipiente está dado por w_j (j =1,2,...,m). Para cada recipiente de tipo j =1,2,...,m se deben encontrar el número de repeticiones x_j del recipiente, así como el número de copias a_{ij} del objeto i-ésimo que se deben almacenar en el recipiente j-ésimo (j =1,2,...,m).

Key words

Combinatorial optimization, heuristics, probabilistic algorithms, integer programming.

Abstract

We show two heuristic algorithms to solve a bin-packing problem type, in which we want to store objects of n different kinds (where the requirement for the i the object is given by d_i) into m different types of bins with possibly non-equal capacities; all the j-type bins (j =1,2,...,m) must have the same distribution for the objects. The objective is to minimize the cost of the used bins, where the cost of each one is given by w_j (j =1,2,...,m). For each j -type bin, the number of repetitions x_j of the bin and the number of copies a_{ij} of the i the object in that bin must be found.

Introducción

El problema que se explora se enmarca dentro de la optimización combinatoria,

1. Profesor e investigador del Instituto Tecnológico de Costa Rica, Escuela de Matemática. Correo electrónico: *gfigueroa@itcr.ac.cr.*
2. Profesor e investigador del Instituto Tecnológico de Costa Rica, Escuela de Matemática. Correo electrónico: *lecarrera@itcr.ac.cr.*

la cual es una rama de la matemática que estudia problemas de optimización donde la región factible es discreta. De manera sorprendente, este hecho vuelve complejo un problema que en el caso continuo puede ser resuelto con relativa facilidad.

Debido a la dificultad de encontrar soluciones óptimas para este tipo de problemas, se utilizan técnicas heurísticas. Este tipo de técnicas tiene como fin encontrar una *buena solución*, es decir, una solución que aunque no sea óptima, sea aceptable. Recientemente ha tomado gran fuerza el uso de técnicas metaheurísticas, algunas de las cuales están inspiradas en características de la evolución biológica: reproducción, mutación, recombinación y selección. Entre estas se encuentran los algoritmos genéticos, colonias de hormigas, búsqueda tabú y recocido simulado [1].

Por lo general, los problemas de optimización combinatoria tienen un espacio factible enorme, incluso para instancias pequeñas del problema, lo cual dificulta la posibilidad de hallar una solución óptima en un tiempo computacional razonable. Esta es la razón que justifica el uso de técnicas heurísticas y metaheurísticas, las cuales pueden producir buenas soluciones en un tiempo razonable.

Por lo general, los problemas de optimización combinatoria tienen un espacio factible enorme, incluso para instancias pequeñas del problema, lo cual dificulta la posibilidad de hallar una solución óptima en un tiempo computacional razonable. Esta es la razón que justifica el uso de técnicas heurísticas y metaheurísticas, las cuales pueden producir buenas soluciones en un tiempo razonable.

El problema

En el proyecto de investigación *Algoritmos evolutivos, optimización de Lipschitz y optimización combinatoria* [3] se desarrollaron varias heurísticas para encontrar posibles soluciones al siguiente problema:

Dado el número de tipos de objetos n, la demanda para cada tipo de objeto $d = (d_1,d_2,...d_n) \in \mathbb{N}^n$, el número de tipos de recipientes m, con capacidades $c = (c_1,c_2,...c_m) \in \mathbb{N}^m$ y costos positivos $w = (w_1,w_2,...w_m) \in \mathbb{R}^m$, determinar $= (x_1,x_2,...x_m) \in \mathbb{N}^m$ y $A = [a_{ij}]$ tales que:

$$\text{Se minimice: } w \cdot x = \sum_{i=1}^{m} w_j x_j$$

$$\text{sujeto a: } \sum_{i=1}^{n} a_{ij} x_j \text{ para } i = 1,2,...,n$$

$$\sum_{i=1}^{n} a_{ij} = c_j \text{ para } j = 1,2,...,m$$

$$a_{ij} \in \{0,1,...,c_j\}$$

A manera de ejemplo, considere el siguiente problema: en una tienda de abarrotes reciben mercadería nueva, pero se dan cuenta de que tenían almacenados 97 jugos de uva, 76 de manzana y 68 de naranja, y podrían caducar si no se venden pronto. Entonces se sugiere ponerlos en oferta en paquetes surtidos de 4 y 6 sabores, de manera que tanto los paquetes de 4, como los paquetes de 6 sabores sean iguales (es decir, que tengan los mismos sabores y la misma cantidad de cada sabor). Los costos de empacar cada paquete son de ¢50 y ¢72, respectivamente.

¿Cómo se deben empacar los jugos, de manera que se pongan en oferta todos los que estaban almacenados y se minimice el costo del empaque?

Observe que se tienen tres tipos diferentes de jugos (objetos, n=3) y dos tipos diferentes de empaques (recipientes, m=2). Para modelar el problema, suponga que:

- x es el número de empaques necesarios con capacidad para 4 jugos (repeticiones del recipiente 1).
- y es el número de empaques necesarios con capacidad para 6 jugos (repeticiones del recipiente 2).
- a_{ij} es el número de jugos de tipo i (uva, manzana o naranja) que se deben poner en el recipiente de tipo j (número de copias del objeto i-ésimo en el recipiente j-ésimo).
- $d = (d_1,d_2,d_3) = (97,76,68)$ los requerimientos de cada tipo de jugo.
- $c = (c_1,c_2,c_3)=(4,6)$ las capacidades de cada empaque.

- $w = (w_1, w_2) = (50, 72)$ el costo de cada empaque (¢50 y ¢72).

Con esta notación se puede representar una instancia del problema por medio de la siguiente tabla:

	x	y	
d_1	a_{11}	a_{12}	r_1
d_2	a_{21}	a_{22}	r_2
d_3	a_{31}	a_{32}	r_3

Se han incluido los términos, r_1 r_2 y r_3, los cuales representar el exceso o sobrante respecto a la cantidad deseada para cada uno de los objetos.

Por otro lado, como el número de jugos por paquete debe ser 4 o 6, deben satisfacerse las siguientes condiciones:

$$\sum_{i=1}^{1} a_{i1} = 4 \qquad \sum_{i=1}^{1} a_{12} \qquad (1)$$

Además, para agotar las existencias se debe cumplir que:

$$a_{11}x + a_{12}y \geq d_1 = 97$$
$$a_{21}x + a_{22}y \geq d_2 = 76 \quad (2)$$
$$a_{31}x + a_{32}y \geq d_3 = 68$$

Y se quiere minimizar $50x + 72y$.

Observe que el problema es complejo, pues no solo se debe minimizar el costo, sino que además se debe buscar la forma de distribuir los jugos en los paquetes. Esto hace que sea imposible aplicar directamente alguna técnica de programación lineal o entera, pues a *priori* no se conoce la distribución que se debe asignar a cada tipo de empaque.

Por ejemplo, una solución como la siguiente:

	30	23	
97	2	2	9
76	1	2	0
68	1	2	8

afirma que necesitamos 30 paquetes del tipo 1 (con capacidad para 4 jugos) y 23 paquetes del tipo 2 (con capacidad para 6 jugos), con un costo total de *50 * 30 + 72 * 23 = 3156*. Además, en cada paquete de 4 se deben colocar dos jugos de uva, uno de manzana y uno de naranja; en cada paquete de 6 se deben colocar dos jugos de cada tipo. Observe que esta solución satisface las condiciones (1) y (2). Por otro lado, como deben ponerse en oferta todos los jugos existentes, pueden requerirse jugos extra; por ejemplo, en este caso se requieren 9 jugos de uva y 8 de naranja.

Espacio factible

Para tener una idea de la complejidad del problema, se explorará el espacio factible. Para esto, se calcularán todas las posibles matrices de distribución de jugos que pueden presentarse:

$$A^k = \begin{pmatrix} a^k_{11}\, a^k_{32} \\ a^k_{21}\, a^k_{32} \\ a^k_{31}\, a^k_{32} \end{pmatrix}$$

Observe que como $a_{11} + a_{21} + a_{31} = 4$, donde cada a_{11} es un entero entre 0 y 4, se tienen 15 posibles tripletas con suma 4, como se muestra en el cuadro 1. Aquí se indican los valores de los a_{11} presentes en la tripleta, los restantes son cero; por ejemplo, hay 6 tripletas que contienen los valores 3 y 1.

Cuadro 1. Tripletas con suma 4.

a_{i1}	*Total*
4	*3*
3,1	*6*
2,2	*3*
1,1,2	*3*

Para la segunda columna, las posibles tripletas con suma 6 se muestran en el cuadro 2, en total son 28. Así, se tienen *15 * 28 = 420* matrices de distribución, algunas de las cuales producen una solución óptima, es decir, minimizan los costos.

Cuadro 2. Tripletas con suma 6.

a_{i1}	Total
6	3
5,1	6
4,2	6
3,3	3
4,1,1	3
3,2,1	6
2,2,2	1

En la figura 1 se muestran las soluciones óptimas obtenidas al resolver la relajación lineal del problema:

$$\begin{cases} \text{Mín } 50x + 72y \\ A \cdot (x,y)T \geq d^T \\ a_{ij} \in \{0,1,...,c_j\} \\ x,y \in \mathbb{N} \end{cases}$$

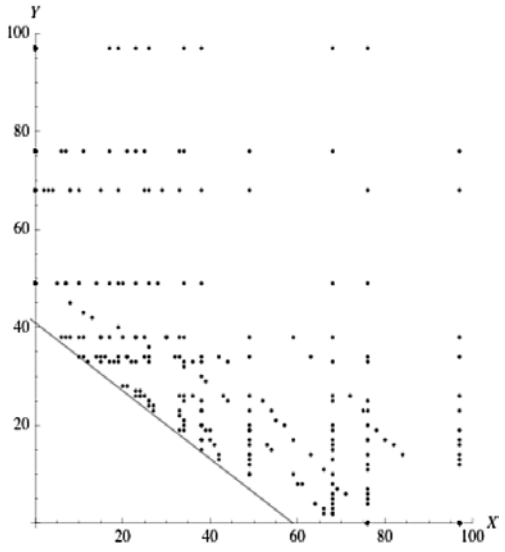

Figura 1. Soluciones óptimas relajadas.

Por medio de programación lineal, para cada matriz de distribución A y para el vector de demandas $d = (d_1, d_2, d_3) = (97, 76, 68)$. Observe que detrás de cada uno de los puntos de la gráfica de la figura 1 está una o varias matrices de distribución, las cuales tienen a este punto como solución óptima. La recta $50x + 72y$ que se muestra en la Figura 1 corresponde a la mejor solución que se obtuvo por este método.

En general, para determinar el número posible de matrices A^k, se determina el número de distribuciones para cada columna, tal que $a_{1j} + a_{2j} + ... + a_{nj} = c_j$ donde c_j es la capacidad del recipiente respectivo. El número de distribuciones para el recipiente en particular está dado por las combinaciones:

$$\binom{n+c_j-1}{n-1} = \frac{(n+c_j-1)}{(n-1)c_j}$$

y en consecuencia el número total de matrices está dado por:

$$\prod_{i=1}^{m} \binom{n+c_j-1}{c_j-1}$$

Observe que la cantidad de posibles matrices A^k aumenta drásticamente con respecto a c_j, n y sobre todo con respecto a m; además, no toda matriz A^k es una solución factible, por ejemplo, aquellas matrices tales que $a_{i1} = a_{i2} = ... = a_m = 0$ para algún $i \in \{1,2,...,n\}$ es decir, donde no se asignaron copias de la demanda i-ésima en ninguno de los recipientes.

Búsqueda aleatoria

Note que para resolver este problema por medio de relajación lineal se requiere resolver alrededor de 420 problemas de programación lineal para obtener una solución que podría ser óptima; sin embargo, en problemas más grandes el

número de problemas de programación lineal por resolver es mucho mayor y el tiempo computacional requerido podría ser prohibitivo.

Por esta razón se buscan otras alternativas, una de ellas es la búsqueda aleatoria. Esta técnica es relativamente simple: consiste en generar aleatoriamente una cantidad *razonable* de soluciones factibles a un problema de optimización dado y de entre estas elegir la mejor. Dichas soluciones factibles normalmente son generadas por medio de una distribución de probabilidades, usualmente uniforme. De esta manera, no se explora todo el espacio factible, sino una muestra. Claro está, cuanto mayor sea la porción explorada, mayor probabilidad de hallar la solución óptima.

En este caso, la idea es generar de manera aleatoria matrices de distribución y resolver por medio de programación lineal el problema asociado. Para esto se desarrolló el algortimo BUSQALEATORIA, junto con las siguientes funciones:

- **mdist***(m,n,**c**):* esta función genera, por medio de una ruleta sesgada construida a partir del vector ***d***, una matriz de distribuciones. El objetivo de esto es almacenar primero los objetos cuya cantidad es mayor. Aquí, ***c*** es el vector de capacidades de los paquetes ($c_1 = 4$ *y* $c_2 = 6$).
- **pl**(***w,A,d***): esta función resuelve por medio de programación lineal la relajación lineal del problema:

$$\begin{cases} \text{Minimizar: } w \cdot x^T \\ \text{Sujeto a:} \\ A \cdot x^T w \geq d^T \\ a_{1j} \in \{0,0,...,c,\} \\ x^T \in \mathbb{N} \end{cases}$$

Si la solución obtenida por medio de la relajación lineal es entera, entonces es la solución óptima del problema entero. Si la solución obtenida por medio de la relajación lineal no es entera, se redondea hacia arriba (para asegurar que se satisface el problema), aunque es posible que ninguno de los redondeos proporcione la solución óptima del problema entero.

El algoritmo BUSQALEATORIA encuentra un solución al problema, no necesariamente óptima. Tiene como entradas: el número *n* de tipos de objetos, el número *m* de tipos de recipientes, el vector *d* de demandas, el vector *c* de capacidades, el vector *w* de costos de empaque y el número *k* de soluciones que serán generadas aleatoriamente. La salida del algoritmo es la tupla (x^+,A^-) donde x^+ es la solución óptima local encontrada y *A* la matriz de distribución de los objetos.

Al ejecutar el algoritmo con los datos del ejemplo y $k = 50$, se obtuvo la siguiente solución:

$$A^k = \begin{bmatrix} 3 & 2 \\ 1 & 2 \\ 0 & 2 \end{bmatrix}$$

$$x^+ = (10,34)$$

Es decir, se debe hacer un total de 44 paquetes distribuidos de la siguiente forma: 10 paquetes de capacidad 4

Algoritmo: BUSQALEATORIA
(n, m, d, c, w, k)

1 $A^+ \leftarrow$ **mdist**(m,n,c)
2 $x^+ \leftarrow$ **pl**(w,A,d)
3 $mejor \leftarrow w \cdot x$
4 **para** $i \leftarrow 2$ **hasta** k:
5 $\hat{A} \leftarrow$ **mdist**(m,n,c)
6 $\hat{x} \leftarrow$ **pl**(w, $\hat{A}$,d)
7 $mejor_1 \leftarrow w \cdot \hat{x}$
8 **si** $mejor_1 < mejor$:
9 $mejor \leftarrow mejor_1$
10 $A^+ \leftarrow \hat{A}$
11 $x \leftarrow \hat{x}$
12 **regresa** (x^+, A^-)

conteniendo cada uno tres jugos de uva y uno de manzana, además, 34 paquetes de capacidad 6 conteniendo cada uno dos jugos de uva, dos de manzana y dos de naranja. Todo con un costo total de empaque de $50 \cdot 10 + 72 \cdot 34 = 2948$ colones. En este caso, la solución corresponde a la solución óptima.

Algoritmo heurístico

El algoritmo BUSQALEATORIA presentado en la sección anterior es no determinístico, en el sentido de que al ejecutarlo sobre la misma instancia del problema puede producir soluciones distintas. El algoritmo que se presenta a continuación tiene la característica de que para la misma instancia del problema siempre produce la misma solución, por lo que es un determinístico. Este algoritmo es de tipo heurístico y al igual que el anterior la solución que encuentra no es necesariamente óptima.

El algoritmo parte de la suposición (no siempre cierta) de que es necesario que las demandas se vayan satisfaciendo al momento de ir construyendo la distribución para cada uno de los recipientes. En el ejemplo que presentamos, lo que se quiere es que algunas de las demandas se agote en el primer recipiente, y entonces se buscan todas las distribuciones posibles para el resto de las demandas; se determina cuánto falta por satisfacer para cada una y luego se llena a partir del siguiente recipiente, aplicando la misma idea de manera recursiva.

Al final, se va a llegar al caso en el que se tiene que llenar el último recipiente, que sería la *base* de la recursión, la cual puede resolverse de forma óptima (como se demostró en [2]), mediante el algoritmo que se muestra a continuación.

Utilización de un solo tipo de recipiente

Suponga que se cuenta con un único tipo de recipiente, las demandas de n tipos distintos de objetos $d = (d_1,d_2,...d_c)$ y la capacidad c del recipiente tal que $n \leq c$. Se debe determinar la matriz de distribuciones $A = (a_{11} = a_{21},..., a_{c1})^T$, donde a_{i1}representa el número de copias del i-ésimo tipo de objetos almacenados en el único tipo de recipiente y el número de repeticiones x tal que:

se minimice: x (3)

sujeto a:

$$\sum_{i=1}^{x} a_{i1} = a_{11} + a_{21} + ... + a_{n\,1} = c \quad (4)$$

$i = 1,2,...n$ (5)

$x \in \mathbb{N}$ (6)

$a_{i1} \in \{0,1,...,n\}\ i \in \{1,2,...n\}$ (7)

En este caso, no interesa el costo del recipiente, ya que es único. Minimizar el número de recipientes x por utilizar es lo mismo que minimizar $w \cdot x$ para valores positivos de w.

Suponga que los jugos $d = (97,76,68)^T$ se quisieran poner en oferta solamente en paquetes de 6, todos iguales. Puesto que se necesita tener al menos un jugo de cada sabor en el recipiente, una primera solución posible es:

i	Demandas	Copias	Repeticiones requeridas
1	97	1	97
2	76	1	76
3	68	1	68

De esta manera, el mínimo número de paquetes que permite agotar *todas* las existencias es 97, ya que con menos paquetes la existencia del primer jugo no sería agotada. Como todavía se pueden colocar más jugos en el paquete, se podría agregar un jugo más del primer tipo:

i	Demandas	Copias	Repeticiones requeridas
1	97	2	49
2	76	1	76
3	68	1	68

Así, ya no son necesarios 97 paquetes (puesto que al haber 2 jugos del tipo 1, para agotar las existencias se requieren tan solo 49 paquetes), sino 76, necesarios para agotar las existencias del segundo tipo de jugo. De esta manera, se repite el procedimiento hasta que el recipiente esté lleno. En el cuadro 3 se presenta una matriz $M^{nx\,(c-n+1)} = [m_{ij}]$ que contiene el ejemplo completo.

$$\begin{bmatrix} (1,97^{+}) & (2,49) & (2,49) & (2,49^{+}) \\ (1,76) & (1,76^{+}) & (2,38) & (2,38) \\ (1,68) & (1,68) & (1.68^{+}) & (2,34) \end{bmatrix}$$

Cuadro 3. Distribución óptima en un solo tipo de recipiente con capacidad 6.

La solución para este ejemplo, con $d = (97,76,68)^T$ y $c = 6$ es $x^+ = 49$ y $A = (2,2,2,)^T$, como se muestra en la última columna del cuadro 3.

En cada columna se muestra una solución factible, cada una construida a partir de la solución anterior, con excepción de la primera columna, que corresponde a colocar un jugo de cada tipo. El primer término de la tupla corresponde al número de jugos y el segundo término al número de paquetes requeridos para utilizar todos los jugos del tipo respectivo. La demanda que requiere mayor número de paquetes se marca con un asterisco, y en el siguiente paso se agrega un jugo de dicho tipo.

La idea para resolver el problema es que el mínimo número requerido de repeticiones del recipiente corresponde al techo de la mayor proporción entre la demanda requerida del i-ésimo objeto d_1 y el número de copias que tiene el recipiente del i-ésimo objeto a_{i1}, es decir, $\max \{[d_j / a_{j1}\ ] / i = 1,2,...,n\}$. Por ello, el primer paso es colocar una copia de todos los objetos en el recipiente (ello es posible gracias a que $c \geq n$, luego se escoge el objeto que define el mínimo número de repeticiones del recipiente para satisfacer todas las demandas y se agrega una copia de dicho objeto en nuestro recipiente, repitiendo el procedimiento hasta que esté lleno.

El algoritmo UNRECIPIENTE resuelve el problema de manera óptima para el caso de un solo tipo de recipiente. Toma como entrada el número n de tipos distintos de objetos, la capacidad del recipiente c y las demandas de los objetos $d = (d_1,d_2,...d_c)^T$, tal que $c \geq n$. La salida del algoritmo es la tupla (x^+, A^-), donde x^+ es la solución óptima para el problema y A^- determina el número de copias de cada objeto en el recipiente.

Algoritmo: UNRECIPIENTE (n, c, **d**)

1 $a_{i1} \leftarrow 1$ **para** $i = 1,2,...,n$

2 **para** $c^1 \leftarrow n + 1$ **hasta** c

3 encuentra j tal que

$\{[d_j / a_{j1}] = \max \{[d_j / a_{j1}\ ] / i = 1,2,...,n\}$

4 $a_{j1} \leftarrow a_{j1} + 1$

5 $x^+ \leftarrow \max \{[d_i / a_{i1}\ ] / i = 1,2,...,n\}$

6 **regresa** (x^+, A^-)

5.2. Caso general m ≥ 2

Continuando con el ejemplo y aplicando la suposición descrita al inicio de esta sección, suponga que se quiere agotar el segundo tipo de jugo en el primer paquete, y que se quiere agotar de manera que el número de paquetes sea exacto para ello. Entonces se podrían colocar 1, 2, 3 o 4 jugos en el paquete y se requerirían 76, 38, 26 o 19 paquetes respectivamente. Entonces se descartan los extremos (19 y 76) y se va a resolver el problema construyendo 26 paquetes y 38 paquetes con capacidad 4, y se toma la mejor de ambas las opciones.

Aquí solamente se desarrollará la idea con 38. Entonces se tiene:

	38		
97			
76	2	0	0
68			

Como sobran dos espacios, se prueba con todas las posibles distribuciones para el primer paquete

	38		
97	2		-21
76	2	0	0
68	0		-68

	38		
97	1		-59
76	2	0	0
68	1		-30

	38		
97	0		-97
76	2	0	0
68	2		12

Los valores negativos en la última columna representan los jugos que faltan de colocar en el segundo paquete, así que se aplica el algoritmo para el caso de un recipiente, y se encuentra la distribución óptima para el segundo paquete:

	38	17	
97	2	2	13
76	2	0	0
68	0	4	0

	38	15	
97	1	4	1
76	2	0	0
68	1	2	0

	38	17	
97	0	6	5
76	2	0	0
68	2	0	12

en cuyo caso se escoge la segunda de las soluciones.

Ahora se va a generalizar el procedimiento anterior. Para poder satisfacer de manera exacta alguna de las demandas, por ejemplo, la del objeto *i*-ésimo, entonces si el número de repeticiones es x, y dado que se está construyendo la distribución para el primer recipiente, entonces debe cumplirse que $0 \le a_{i1}x - d_i < a_{i1}$, donde $a_{i1} \in \{1,2,...,c_1\}$. Así:

$$d_i \le a_{i1}x < a_{i1} + d_i$$

$$\Rightarrow \quad \frac{d_i}{a_{i12}} \le x \le 1 + \frac{d_i}{a_{i1}}$$

$$\Rightarrow \quad x = \left[\frac{d_i}{a_{i12}} \right]$$

donde [p] es el menor entero mayor o igual a p, dado que el valor de x debe ser entero.

Por lo tanto, se determinan todos los posibles valores de x para todas las demandas, y en cada una de ellas, para todos los valores de a_{i1} =1,2,...,c_1. Tal procedimiento genera un vector x de tamaño $c_1 \cdot n$. Este se ordena de manera creciente y se quita una porción de los extremos (los valores más pequeños y más grandes de x), que en general no dan como resultado una buena solución.

Recuerde que cada uno de los valores x se obtuvo mediante $[d_i / a_{i1}]$, por lo que

Algoritmo: HEUR RECURSIVO (n, m, **d**, c, w)

1 ***si*** $m = 1$

2 *regresa UN RECIPIENTE* $(n, c_1, \mathbf{d})$

3 $mejor \leftarrow m \cdot \max\{w\} \cdot \max\{d\}$

4 $\overset{*}{x} \leftarrow [d_i / q]$ ***para*** $i = 1,2,...,n$, $q = 1,2,...,c_1$

5 *ordenar* $\overset{*}{x}$ *y quitar una porción de los extremos*

6 ***para cada*** $x \in \overset{*}{x}$

7 ***para cada*** *vector* $(a_{11}, a_{21}, ..., a_{n1})$ *con* a_{i1} fijo

8 $n^1 \leftarrow$ número de demandas que faltan por satisfacer

9 $d^1 \leftarrow$ demandas que faltan por satisfacer

10 $c^1 \leftarrow (c_2, c_3, ..., c_m)$

11 $w \leftarrow (w_2, w_3, ..., w_m)$

12 $(x^1, A^1) \leftarrow$ *HEUR RECURSIVO* $(n', m-1, d', c', w')$

13 $mejor \leftarrow w_1 \cdot x + w^1 \cdot x^1 \cdot \max\{d\}$

14 **si** $mejor^1 < mejor$:

15 $mejor \leftarrow mejor^1$ $mejor = mejor^1$

16 $\overset{*}{x} \leftarrow (x, x^1_1, x^1_2, ..., x^1_{x-1})$

17 $A^+ \leftarrow (a_{11}, a_{21}, ...a_{c1})^T A^1$

18 **regresa** (x^+, A^-)

para cada uno de dichos valores se fija el valor a_{i1} respectivo y se calculan todas las posibles distribuciones para la columna $a_{11}, a_{21}, ..., a_{n1}$; entonces se calculan las demandas que han quedado sin satisfacer y se resuelve el problema de manera recursiva disminuyendo en 1 el número de recipientes. Al final se escoge la mejor de las soluciones.

Para el ejemplo de los jugos, la solución obtenida con este algoritmo es la siguiente:

	11	34	
97	0	3	5
76	4	1	2
68	0	2	0

Con un costo de $50 \cdot 11 + 72 \cdot 34 = 2998$ colones.

Conclusiones

Los algoritmos presentados fueron desarrollados como parte del proyecto de investigación: *Algoritmos evolutivos, optimización de Lipschitz y optimización combinatoria* [3]. Estos fueron ejecutados en instancias más grandes del problema y en los casos en que se pudo verificar, los resultadosco incidieron con la solución óptima y fueron obtenidos en un tiempo computacional *razonable*. Además, como parte del proyecto se desarrolló un algoritmo genético y se estudiaron algunas variaciones al problema [4]. Actualmente se trabaja en el diseño de un algoritmo paralelo que resuelva el problema.

Bibliografía

[1] Blum, Christian & Roli, Andrea (2003). *Metaheuristic in combinatorial optimization: overview and conceptual comparison*. ACM computing Survey, 35:(3), pp 268-308.

[2] Carrera R., Luis Ernesto (2006). *Algoritmo para encontrar el óptimo a una simplificación de un problema combinatorio presente en la industria editorial*. Tesis para optar por el grado de MSc. en Matemática Computacional, CINVESTAV, México, D.F.

[3] Figueroa M., Geovanni & Carrera R., Luis (2010). *Algoritmos evolutivos, optimización de Lipschitz y optimización combinatoria: informe final.* Vicerrectoría de Investigación, Instituto Tecnólogico de Costa Rica.

[4] Figueroa M., Geovanni & Carrera R., Luis (2010). *Estudio de una variante del problema "bin-packing"*. Programas y resúmenes del XVII Simposio Internacional de Métodos Matemáticos aplicados a las Ciencias. Universidad de Costa Rica. pp. 85--86.

[5] Hromokovic, Juraj (2005). *Design and Analysis of Randomized Algorithms*. Springer Verlag, Be rlin.

[6] Kreher, Donald & Stinson, Douglas (1999). *Combinatorial Algorithms: generation, enumeration and search*. CRC Press, New York.

[7] Papadimitriou, Christos & Steiglitz, Kenneth (1998). *Combinatorial Optimization: algorithms and Complexity*. Dover Plublications, New York. 1998.