Acerca De Una Versión Dinámica Del Problema De La Mochila

Silva Bruno*; Torres Luis M.**

**Escuela Politécnica Nacional, Facultad de Ciencias, Quito, Ecuador*
e-mail: bruno.silbas gmail.com
***Escuela Politécnica Nacional, Centro de Modelización Matemática ModeMat, Quito, Ecuador*
e-mail: luis.torres epn.edu.ec

Resumen: *El problema de lamochila (Knapsack Problem, KP) es un problema clásico de optimización combinatoria que ha sido ampliamente estudiado por más de un siglo (ver, por ejemplo [12] y las referencias allí citadas). Es uno de los problemas de programación lineal entera más simples; aparece como subproblema en otros problemas más complejos y tiene muchas aplicaciones prácticas como: el corte de material [6], la selección de inversiones de capital y portafolios financieros [10], la correcta administración de recursos de cómputo, del ancho de banda de una conexión, del espacio de almacenamiento en discos duros [14], etc. Variantes dinámicas de este problema han sido estudiadas por sus aplicaciones prácticas, aunque no en gran extensión y con pocos resultados obtenidos hasta el presente. La variante considerada en este artículo consiste en agregar una dimensión temporal (discreta) al problema clásico: a cada objeto se le asigna una duración, que indica el intervalo de tiempo que éste debe permanecer dentro de la mochila cada vez que es seleccionado. Se busca maximizar el valor total almacenado en la mochila dentro de un horizonte temporal T. Formulamos un modelo de programación lineal entera para este problema y presentamos un algoritmo de solución exacto basado en el esquema branch-and-bound. Adicionalmente, estudiamos su comportamiento y su desempeño computacional.*

Palabras clave: *problema de la mochila (knapsack problem), optimización dinámica, branch-and-bound, heurísticas primales.*

Abstract: *The Knapsack Problem (KP) is a classical combinatorial optimization problem that has been widely studied for more than a hundred years (see, for a example [12] and the references there in). It is one of the simplest linear integer programming problems and appears as a subproblem in other more complex problems. It has many practical applications in such diverse areas as cutting-stock [6]; investment selection in capital and financial portfolios [10]; the correct administration of a computer RAM memory, band-width of a connection, disk space [14], etc. Dynamic variants of this problem have been studied for their practical applications, although not in a wide extension and with few results reported up to the present. In this paper we consider a variant which consists in adding a (discrete) temporal dimensión to the classic problem: a duration is assigned to each object indicating the interval of time that it has to remain inside the knapsack whenever it is chosen. The objective is to maximize the total value stored in the knapsack within a certain temporal horizon T. We formulate an integer programming model for this problem and develop an exact solution algorithm based on the branch-and-bound scheme. Furthermore, we report results on its computational behavior and performance.*

Keywords: *knapsack problem, dynamic optimization, branch-and-bound, primal heuristics.*

1. INTRODUCCION

Dados una mochila con capacidad C y un conjunto de n objetos diferentes, con valores p_1, ...p_n y tamaños w_1, ..., wn, el *problema clásico de la mochila (Knapsack Problem - KP)* consiste en seleccionar un subconjunto de objetos (para almacenarlos en la mochila) cuya suma de tamaños no supere C y cuya suma de valores sea la mayor posible.

Introduciendo para cada objeto una variable binaria x_i, $1 \leq i \leq n$, que indique si el mismo debe ser incluido o no en la selección, el problema puede formularse como un programa de optimización lineal entera:

$$(KP) = \begin{cases} max \sum_{i=1}^{n} p_i x_i \\ s.t \sum_{i=1}^{n} w_i x_i \leq C, \\ x_i \in \{0,1\}, \forall_i \in \{1, \dots, n\} \end{cases}$$

El problema de la mochila es un problema clásico en la optimización combinatoria. Constituye uno de los problemas más simples de la programación lineal entera, aparece como subproblema en muchos otros problemas más complejos y tiene diversas aplicaciones prácticas. Diferentes técnicas de solución han sido abordadas durante las últimas décadas. En 1957, Dantzig [4] presentó un método eficiente para determinar la solución de la relajación continua del problema (CKP), y por lo tanto, una cota superior para el problema discreto. En 1967, Kolesar propuso el primer algoritmo tipo

branch-and-bound para KP. Durante los años 70's, los métodos tipo branch-andbound se desarrollaron más, gracias a lo cual fue posible resolver problemas con un gran número de variables. El algoritmo más conocido de este período se debe a Horowitz y Sahni [7]. En 1977Martello y Toth [11] propusieron la primera cota superior que mejora el valor de la relajación continua CKP. En la década de los 80's los resultados tienen que ver con la solución de problemas de gran tamaño. Balas y Zemel [2] presentaron un nuevo enfoque para resolver el problema clasificando, en muchos casos, solo un subconjunto pequeño de las variables. Desde esta época en adelante se empezaron a estudiar variantes de este problema tales como su versión acotada, no-acotada y el problema de la mochila de elección múltiple. Martello y Toth publicaron en 1990 una revisión exhaustiva de los diferentes resultados teóricos y métodos de solución existentes hasta ese momento [12].

Desde el enfoque de la teoría de complejidad computacional, el problema de la mochila pertenece a la clase de problemas NP-difíciles (ver detalles de la definición de esta clase, por ejemplo, en [5, p. 247]), para los cuales suele asumirse que no existen algoritmos polinomiales de solución (a menos que P=NP). Para demostrar esto, basta notar que KP es una generalización del problema SUBSET-SUM (que es uno de los problemas NP-completos estándares [8]): dados un conjunto finito $A = \{a_1, ..., a_n\}$ de números enteros, y un número $B \in \mathbf{Z}$, determinar si existe un subconjunto $A' \subseteq A$ tal que $\sum_{ai \in A'} ai = B$.

En efecto, puede verse que toda instancia de SUBSET-SUMpuede transformarse en una instancia de KP con *n* objetos, cuyos pesos y valores están dados por *wi = pi = ai*, y donde la capacidad de la mochila es igual a *B*. Por otra parte, puede considerarse a KP como uno de los problemas "más fáciles" dentro de la clase NP-difícil, pues existen para este problema esquemas de aproximación eficientes (PTAS [17] y FPTAS [9]).

El KP aparece en aplicaciones prácticas en procesos de toma de decisiones del mundo real, tales como la búsqueda de patrones de corte para materias primas que generen el menor desperdicio posible [6], la selección de inversiones de capital y portafolios financieros [10] y la optimización de recursos computacionales [14]. Se han estudiado algunas variantes dinámicas del problema de la mochila. Por ejemplo, Papastavrou [13] considera una situación en la cual los objetos arriban de acuerdo a un proceso de Poisson en el tiempo. Cada objeto tiene asociados una demanda de un recurso limitado y un valor. Cuando un objeto arriba, debe decidirse si el mismo es aceptado o rechazado. El objetivo es diseñar una estrategia óptima para maximizar el valor acumulado esperado, correspondiente a los objetos aceptados dentro de un horizonte de tiempo. En otro contexto, Bartlett et al. [3] consideran el problema de calendarizar la asignación de un recurso compartido para atender un conjunto de pedidos, cada uno de los cuales tiene un tiempo de inicio, un tiempo de finalización y un monto solicitado del recurso. Los autores denotan a la variante del problema como problema temporal de la mochila (TKP, temporal knapsack problem).

En la variante dinámica que consideramos en este artículo, los *n* objetos tienen asociados, además de valores y tamaños, duraciones $d_1, ..., d_n$. Esto significa que si el objeto *i* es agregado a la mochila en el tiempo *t*, el mismo permanecerá dentro de la misma hasta el tiempo $t + d_i - 1$.

El objetivo del problema es maximizar el valor total almacenado en la mochila dentro de un horizonte temporal. Presentamos un modelo de programación lineal entera para este problema, estudiamos cotas superiores, heurísticas primales y proponemos un algoritmo de solución basado en el esquema branch-and-bound. Finalmente, presentamos resultados comparativos del desempeño computacional de una implementación de nuestro algoritmo frente al solver SCIP [1] para problemas lineales enteros. Los resultados completos del presente trabajo han sido publicados en [16].

2. EL PROBLEMA KP-DUR

2.1 Modelo de programación entera

Como se señaló anteriormente, en esta variante del problema se tienen dados una mochila con capacidad *C* y un conjunto de *n* objetos los cuales tienen asociados valores *p*1, ..., *pn*, tamaños *w*1, ..., *wn* y duraciones *d*1, ..., *dn*. El objetivo es maximizar el valor total almacenado en la mochila dentro de un intervalo de tiempo, el mismo que consideraremos discretizado en períodos {1, 2, ..., *T*}. Si el objeto *i* ingresa en la mochila en el período $t \in \{1, ..., T - di - 1\}$, permanecerá en la misma hasta el período *t + di + 1*. Asumimos que todos los parámetros C, T, p_i, w_i y d_i son enteros, y que un objeto puede ingresar más de una vez en la mochila.

Definimos variables binarias x_{it}, $i \in \{1, ..., n\}$, $t \in \{1, ..., T\}$, que nos indican si el objeto i ingresa en la mochila en el período t; y variables binarias $z_{it,}$ $i \in \{1, ..., n\}$, $t \in \{1, ..., T\}$, que nos indican si el objeto *i* se encuentra presente en la mochila en el período *t*. De esta manera, el problema puede formularse como el siguiente programa de optimización entera (KP-DUR*):

$$
\begin{cases}
\max \sum_{i=1}^{n} \sum_{t}^{T} p_i x_{it} & (1) \\
s.t \sum_{i=1}^{n} w_i z_{it} \leq C, \forall t \in \{1, \dots, T\}, & (2) \\
x_{it} = 0, \forall i \in \{1, \dots, n\}, \forall t \in \{T - d_i + 2, \dots, T\} & (3) \\
x_{it} = 1 \Longrightarrow z_{i,t+1} = 1, \forall i \in \{1, \dots, n\}, \forall l \in \{0, \dots, d_i - 1\} & (4) \\
x_{it} = 1 \Longrightarrow x_{i,t+1} = 0, \forall i \in \{1, \dots, n\}, \forall l \in \{1, \dots, d_i - 1\} & (5) \\
z_{it} \leq \sum_{l=0}^{min\{d_i - 1, t-1\}} x_{i,t-l}, \forall i \in \{1, \dots, n\}, \forall t \in \{1, \dots, T\} & (6) \\
x_{it}, z_{it} \in \{0,1\}, \forall i \in \{1, \dots, n\}, \forall t \in \{1, \dots, T\}
\end{cases}
$$

La función objetivo (1) mide el valor total de los objetos que ingresan en la mochila dentro del horizonte de tiempo *{1, ..., T}.* Las restricciones (2) sirven para asegurar que en cada período la capacidad de la mochila sea respetada. La restricciones (3) impiden que un objeto ingrese en la mochila demasiado tarde, es decir, si su tiempo de permanencia previsto excede el horizonte de tiempo *T.* La restricciones (4) establecen que si un objeto i ingresa a la mochila, entonces este objeto permanece en la misma durante di unidades de tiempo. Las restricciones (5) aseguran que ningún objeto pueda volver a entrar en la mochila hasta que no haya salido de ella. Finalmente, las restricciones (6) expresan que un objeto no puede estar en la mochila en un período t sin haber ingresado a ésta en ninguno de los períodos anteriores *{t, t−1, ..., t−di+1}.* Esta formulación del problema es simple y directa, pero no es adecuada para el desarrollo de algoritmos de solución. Entre otras dificultades, no todas las familias de restricciones son lineales, pues en el caso de (4) y (5) se trata de implicaciones lógicas. Por este motivo, hemos considerado una formulación alternativa, la misma que se describe a continuación.

Nuevamente, definimos variables binarias x_{it}, $i \in \{1, ..., n\}$, $t \in \{1, ..., T\}$, que nos indican si el objeto *i* ingresa en la mochila en el período *t* y variables binarias z_{it}, $i \in \{1, ..., n\}$, $t \in \{1, ..., T\}$, que nos indican si el objeto *i* está en la mochila en el período *t.* Además, introducimos variables binarias y_{it}, $i \in \{1, ..., n\}$, $t \in \{1, ..., T\}$, que nos indican si el objeto *i sale* de la mochila en el período *t.*

El modelo (KP-DUR) se formula de la siguiente manera:

$$\begin{cases} \max \sum_{i=1}^{n}\sum_{t}^{T} p_i x_{it} & (7) \\ s.t \sum_{i=1}^{n} w_i z_{it} \le C, \forall t \in \{1, \dots, T\}, & (8) \\ y_{it+d_i} = x_{it}, \forall i \in \{1, \dots, n\}, \forall t \in \{1, \dots, T - d_i\}, & (9) \\ x_{it} = 0, \forall i \in \{1, \dots, n\}, \forall t \in \{T - d_i + 2, \dots, T\}, & (10) \\ y_{it} = 0, \forall i \in \{1, \dots, n\}, \forall t \in \{1, \dots, d_i\} & (11) \\ z_{i1} = x_{i1}, \forall i \in \{1, \dots, n\} & (12) \\ z_{it} = z_{i,t-1} + x_{it} - y_{it}, \forall i \in \{1, \dots, n\}, \forall t \in \{2, \dots, T\}, & (13) \\ x_{it}, y_{it}, z_{it} \in \{0,1\}, \forall i \in \{1, \dots, n\}, \forall t \in \{1, \dots, T\} & \end{cases}$$

La función objetivo (7) mide el valor total de los objetos ingresados en la mochila dentro del horizonte de tiempo *{1, ..., T}.* Las restricciones (8) sirven para asegurar que en cada período la capacidad de la mochila sea respetada, las restricciones (9) establecen que si el objeto i ingresa a la mochila, entonces este objeto debe salir de la misma luego de di unidades de tiempo. Las restricciones (10) impiden que un objeto ingrese en la mochila si su duración prevista excede el horizonte de tiempo *T.* De manera recíproca, las restricciones (11) impiden que un objeto salga de la mochila en los di períodos iniciales ya que esto no tiene sentido práctico. Las restricciones (12) y (13) especifican la relación entre las variables z_{it}, x_{it}, y_{it} y pueden verse como restricciones de balance: si el objeto *i* entra en la mochila en el período *t* entonces el valor de z_{it} aumenta (con relación al valor de $z_{i,t-1}$). Si el objeto sale de la mochila, el valor de z_{it} disminuye. En todos los demás casos, el valor de z_{it} permanece inalterado.

En [16] se demuestra que las formulaciones KP-DUR y KP-DUR* son equivalentes entre sí. En adelante describiremos el esquema general de la demostración. Para los detalles (y las demostraciones de los lemas intermedios), referimos al lector al documento de la tesis. Notaremos como F_{IP} al conjunto de todas las soluciones factibles de un problema de optimización IP. Para los dos problemas anteriores, tenemos los conjuntos $F_{KP\text{-}DUR*}$ y $F_{KP\text{-}DUR}$, cuyas propiedades estudiaremos a continuación. Se establecen primero los siguientes resultados auxiliares para KP-DUR:

LEMA 1 ([**16**]). *Sea* $(x, y, z) \in \mathcal{F}_{KP-DUR}$.
$\forall i \in \{1, \dots, n\}, t \in \{1, \dots, T - d_i + 1\}$ *si* $x_{it} = 1 \Rightarrow$
$z_{i,t+1} = 1, \forall l \in \{0, \dots, d_i - 1\}$.
$\forall i \in \{1, \dots, n\}, t \in \{1, \dots, T - d_i + 1\}$ *si* $x_{it} = 1 \Rightarrow$
$x_{i,t+1} = 0, \forall l \in \{0, \dots, d_i - 1\}$.

LEMA 2 ([**16**]). *Sea* $(x, y, z) \in \mathcal{F}_{KP-DUR}$. *Entonces*

$$z_{it} \le \sum_{l=0}^{min\{d_i-1, t-1\}} x_{i,t} - l, \forall i \in \{1, \dots, n\}, \forall t \in \{1, \dots, T\}$$

De los dos lemas anteriores podemos concluir el siguiente corolario.

COROLARIO 1. *Si* $(x, y, z) \in$, *entonces* $(x, z) \in \mathcal{F}_{KP-DUR*}$.
Además, notar que ambas soluciones nalcanzan el mismo valor en la función objetivo

Para el sentido inverso de la equivalencia, requerimos de los siguientes resultados.

LEMA 3 ([**16**]). *Sea* $(x, z) \in \mathcal{F}_{KP-DUR*}$. *Entonces*

$$z_{it} \le \sum_{l=0}^{min\{d_i-1, t-1\}} x_{i,t-l}, \forall i \in \{1, \dots, n\}, \forall t \in \{1, \dots, T\}$$

LEMA 4 ([**16**]). *Sea* $(x, z) \in \mathcal{F}_{KP-DUR*}$. *Definimos* $y \in \{0,1\}^{n \times T}$ *por medio de*:

$$y_{it} = \begin{cases} 0, & si\ t = 1, \\ z_{i,t-1} + x_{it} - z_{it} & si\ t \in \{2, \dots, T\} \end{cases}$$

Entonces $(x, y, z) \in \mathcal{F}_{KP-DUR}$. *Notar además que los valoresde las funciones objetivo para ambas soluciones coinciden*

En el Corolario 1 y el Lema 4 hemos demostrado que PDUR y KP-DUR* son dos formulaciones equivalentes del mismo problema. Se tiene además el siguiente resultado para KP-UR*, que emplearemos posteriormente.

LEMA 5 ([**16**]). *Sea* $(x,z) \in \mathcal{F}_{KP-DUR*}$. *Entonces*

$$\sum_{l=1}^{T} z_{it} = d_i \sum_{l=1}^{T} x_{it}, \forall i \in \{1,\dots,n\}.$$

Del Lema 4 se sigue además que:

COROLARIO 2. *Sea* $(x,y,z) \in \mathcal{F}_{KP-DUR.}$ *Entonces*

$$\sum_{l=1}^{T} z_{it} = d_i \sum_{l=1}^{T} x_{it}, \forall i \in \{1,\dots,n\}.$$

2.2 Cotas superiores e inferiores

Presentamos a continuación dos formulaciones del problema clásico de la mochila que pueden emplearse para obtener cotas superiores al valor óptimo de KP-DUR.

$$(KP-CT)\begin{cases} máx \sum_{i=1}^{n} p_i y_i \\ s.t \sum_{i=1}^{n} (w_i d_i) y_i \le CT, \\ 0 \le y_i \le \left\lfloor \frac{T}{d_i} \right\rfloor, y_i \in \mathbb{N}, \forall i \in \{1,\dots,n\} \end{cases}$$

Este problema tiene la forma de una de las generalizaciones del problema clásico de la mochila, específicamente de la versión acotada BKP.

La capacidad de la mochila es igual a *CT*, el tamaño del objeto *i* es $w_i d_i$ y la cantidad disponible del mismo es $\left\lfloor \frac{T}{d_i} \right\rfloor$. Hemos demostrado que KP-CT puede verse como una relajación de KP-DUR:

LEMA 6 ([**16**]). *Dada una función factible* $(x,y,z) \in \mathcal{F}_{KP-DUR,}$ *es posible definir una solución* $\dot{y} \in \mathcal{F}_{KP-CT,}$ *tal que los valores objetivos de ambas soluciones coincidan.*

Del teorema anterior concluimos que toda solución óptima de KP-CT es es cota superior para KP-DUR. El siguiente problema tiene la forma de la versión clásica del problema de la mochila (KP):

$$(KP-PD)\begin{cases} máx \sum_{i=1}^{n} \frac{p_i}{d_i} x_i \\ s.t \sum_{i=1}^{n} w_i x_i \le C, \\ x_i \in \{0,1\}, \forall i \in \{1,\dots,n\} \end{cases}$$

La relación entre KP-PD y KP-DUR está establecida en el siguiente resultado.

LEMA 7 ([**16**]). *Sea* $(x,y,z) \in \mathcal{F}_{KP-DUR.}$ *y sean* $\hat{x}^1,\dots,\hat{x}^T \in \{0,1\}^n$ *definidos por* $\hat{x}_i^t = z_{it}$, *para todo* $t \in \{1,\dots,T\}, i \in \{1,\dots,n\}$. *Entonces* $\hat{x}^t \in \mathcal{F}_{KP-PD,} \forall t \in \{1,\dots,T\}$.

En otras palabras, para cada período t ∈ {1, ..., T}, la solución de KP-DUR restringida a t es una solución factible para KP-PD. Por otra parte, sean $(\bar{x},\bar{y},\bar{z}) \in$ FKP-DUR una solución óptima para una instancia de KP-DUR y $x^* \in$ FKP−PD una solución óptima para la instancia correspondiente de KP-PD. Tenemos entonces que para el valor objetivo de estas soluciones se cumple,

$$\begin{aligned} \sum_{i=1}^{n}\sum_{t=1}^{T} p_i \bar{x}_{it} &= \sum_{i=1}^{n} p_i \sum_{t=1}^{T} \bar{x}_{it} \\ \sum_{i=1}^{n} p_i \left(\frac{\sum_{t=1}^{T} \bar{z}_{it}}{d_i} \right), &\text{por el corolario 2} \\ &= \sum_{i=1}^{n} \frac{p_i}{d_i} \sum_{t=1}^{T} \bar{z}_{it}, \\ &= \sum_{i=1}^{n} \frac{p_i}{d_i} \sum_{t=1}^{T} \hat{x}_i^t, \\ &\le \sum_{t=1}^{T}\sum_{i=1}^{n} \frac{p_i}{d_i} x_i^* = T \sum_{i=1}^{n} \frac{p_i}{d_i} x_i^* \end{aligned}$$

donde $\hat{x}^t$ son las soluciones factibles para KP-PD definidasen el Lema 7. Por lo tanto, hemos demostrado el siguiente resultado:

LEMA 8 ([**16**]). *Una solución óptima* x^* *de una instancia de* $KP-PD$ *permite definir la siguiente cota superior para la instancia de* $KP-DUR$:

$$\left\lfloor T \sum_{i=1}^{n} \frac{p_i}{d_i} x_i^* \right\rfloor$$

Además de la cota superior indicada en el Lema 8, una solución óptima x∗ para KP-PD puede emplearse para definir una solución factible para KP-DUR, al repetir x∗ de manera periódica dentro del horizonte de tiempo T, seleccionando cada objeto i para el cual $x_i^* = 1$, $\left\lfloor \frac{T}{d_i} \right\rfloor$veces consecutivas, conforme se indica a continuación: Heurística Primal KP-PDH

$$1. \forall i \in \{1,\dots,n\}, si\ x_i^* = 1, entonces\ fijar$$
$$Ti := \left\{ kd_i + 1 : 0 \le k < \left\lfloor \frac{T}{d_i} \right\rfloor \right\}$$

$$\hat{x}_{it} \coloneqq 1, \qquad \forall t \in T_i$$
$$\hat{y}_{it+d_i} \coloneqq 1, \qquad \forall t \in T_i/\{T - d_i + 1\}$$
$$\hat{z}_{it+l} \coloneqq 1, \qquad \forall t \in T_i, 0 \leq l \leq d_i$$

2. $Fijar\ \hat{x}_{it} \coloneqq \hat{y}_{it} = \hat{z}_{it} = 0\ en\ todos\ los\ demás\ casos.$

Es fácil comprobar que $(\hat{x}, \hat{y}, \hat{z})$ es una solución factible para KP-DUR. Utilizaremos esta heurística primal dentro del esquema tipo branch-and-bound para la solución de KP-DUR que describimos en adelante.

2.3 Algoritmo KPD-BB

Tenemos todos los componentes necesarios para formular un esquema tipo branch-and-bound para KP-DUR. Emplearemos la heurística KP-PDH para construir una solución factible.

Luego de análisis computacionales preliminares, decidimos emplear el modelo KP-PD (por sobre el modelo KPCT) para el cálculo de cotas superiores durante la exploración del árbol de branch-and-bound. Esta decisión se tomó al evaluar la calidad de las cotas obtenidas frente al tiempo de cálculo necesario sobre varias instancias de prueba.

Por otra parte, el modelo KP-PD puede ser extendido de manera natural para incorporar restricciones adicionales que aparecen a partir de las decisiones de ramificación, como veremos más adelante. Las decisiones de ramificación (branching) consisten en forzar o prohibir que un objeto *i* ingrese a la mochila en un período *t*. Representaremos estas decisiones mediante ternas de la forma (*i*, *t*, *k*) donde:

$$k = \begin{cases} 1, si\ el\ objeto\ i\ ingresa\ en\ la\ mochila\ en\ el \\ período\ t; \\ 0, caso\ contrario \end{cases}$$

Cada nodo del árbol de búsqueda tiene asociada una lista de ternas que representan las decisiones tomadas previamente. Al iniciar el procesamiento del nodo, la información de esta lista se emplea para calcular, para cada período $t \in \{1, ..., T\}$, el conjunto de objetos A^t que pueden ser seleccionados para ingresar en la mochila (a los que llamamos *objetos disponibles*), además de la capacidad residual de la misma $\hat{C}_t$. Se emplea luego una versión modificada del algoritmo de Horowitz-Sahni [7] para calcular una cota superior: se resuelven modelos KP-PD para *T* problemas clásicos de la mochila, cada uno sobre el conjunto A^t de objetos, y considerando una mochila con capacidad $\hat{C}_t$ para $t \in \{1, ..., T\}$. La suma de los valores óptimos sobre todos los períodos es una cota superior al valor óptimo de KP-DUR para el nodo actual. El algoritmo completo está descrito a continuación.

Sean:

$\hat{X} \in \{0,1\}^{n \times T}$ la matriz solución para el nodo actual;

$\hat{z} = \sum_{t=1}^{T} \sum_{i=1}^{n} \frac{p_i}{d_i} \hat{x}_{it}$ el valor de la solución asociada a $\hat{X}$

$X \in \{0,1\}^{n \times T}$ la matriz con la mejor solución factible encontrada hastael momento;

$z = \sum_{t=1}^{T} \sum_{i=1}^{n} \frac{p_i}{d_i} x_{it}$ el valor de la mejor solución factible encontrada hasta el momento;

L la lista de ternas que representan las decisiones de ramificación tomadas
hasta llegar al nodo actual;

Q la cola de nodos por procesar, cada uno representado por su lista de ternas asociada;

$\mathbf{A} \in \mathbf{Z}^{n \times T}$ la matriz de disponibilidad de objetos para el nodo actual;

$\hat{C}_t = C - \sum_{i:\, A_{it}=1} = w_i$ capacidad residual de la mochila en el período *t*, para el nodo actual. Es importante señalar que los elementos de las matrices solución ˆ**X***it*, **X***it* indican la presencia o no del objeto *i* dentro de la mochila en el período *t*. Es decir, corresponden realmente a las variables *zit* del modelo KP-DUR.

Producere: KPD-BB
input: *n*, C,T,(p),(w),(d);
output: z,(X);
begin

1. [inicializar]
   z : = 0;
   $\hat{z}$: = 0;
   X : = 0
   $\hat{X}$: = 0
   $\hat{C}_t$: = $C, \forall t \{1, \dots, T\}$

2. [calcular una solución factible empleando la huerística primal]
   encontrar una solución óptima $\bar{x}$ de la instancia asociada KP-PD

$$z \coloneqq \left[T \sum_{i=1}^{n} \left[\frac{T}{d_i} \right] p_i \bar{x}_i, \qquad X_{it} \coloneqq \begin{cases} \bar{x}_i \ si\ t \leq \left[\frac{T}{d_i} \right] d_i \\ 0\ caso\ contrario \end{cases} \right]$$

3. [incluir en L algunas decisiones de ramificaciones forzadas por el modelo]
   agregar a L las ternas $(i,t,0), \forall i \in \{1, \dots, n\}, \forall t \in \{T - d_i + 2, \dots, T\}$ forzadas por (10);
4. Q : = {L};
5. [lazo principal del esquema branch-and-bound]
   while $Q \neq \emptyset$ **do**
   retirar la primera lista *L* de *Q;*
6. [inicio del procesamiento de un nodo]
   obtener la matriz de disponibilidades *A* y las capacidades resicuales $\hat{C}_t$ a partir de las ternas de *L;*
7. [calcular la cota superior KP-PD]

for *t := 1 to T* **do**
begin

- **e**ncontrar una solución óptima $\bar{x}^t$ de KP-PD con los objetos disponibles en cada columna At de A, empleando la apacidad residual $\hat{C}_t$
- almacenar $\bar{x}^t$ en la columna t de la matriz solución para el nodo actual:

$$\hat{X}_{it} := \begin{cases} 1, & si\ A_{it} = 1 \text{ ó } \bar{x}_i^t = 1 \\ 0 & caso\ contrario \end{cases}$$

end;
end for;

$$\hat{z} := \sum_{t=1}^{T} \sum_{i=1}^{n} \frac{p_i}{d_i} \hat{X}_{it}$$

8. [podado del árbol de búsqueda]
if $\hat{z} \leq$ z **end while**;

9. [comprobar si la solución actual es factible]
if $\hat{X}$es factible para KP-DUR
begin
[actualizar la mejor solución obtenida hasta el momento]
z := $\hat{z}$;
$X_{it} := \hat{X}_{it}$, $\forall i \in \{1, ..., n\}$, $\forall t \in \{1, ..., T\}$;
end;
end while;
else
begin
[ramificación]
generar dos ternas nuevas $t_1 := (i, t, 1)$, $t_2 := (i, t, 0)$;
$L_1 := L \cup \{t_1\}$, $L_2 := L \cup \{t_2\}$;
agregar las dos nuevas listas L_1, L_2 al inicio de la cola Q;
end;
end while;
end.

El primer paso en el procesamiento de un nodo del árbol de búsqueda consiste en recuperar la lista asociada de ternas de ramificación y construir a partir de ésta una matriz de disponibilidades $A \in \{-1,-2, 0, 1\}^{n \times T}$ que identifica para cada objeto *i* en cada período *t* uno de cuatro posibles estados:

- *libre* (*Ait* = −1), si el objeto puede ser seleccionado para ingresar en la mochila;
- *posible* (*Ait* = −2), si el objeto no puede ingresar en la mochila en el período actual, pero puede estar presente en la misma debido a que podría ingresar en un período anterior;
- *presente* (*Ait* = 1), si el objeto forzosamente está presente en la mochila en el período actual, por alguna decisión de ramificación tomada previamente durante la exploración del árbol de búsqueda;
- *prohibido* (*Ait* = 0), si el objeto no puede estar en la mochila en el período actual.

El objeto *i* está disponible en el período *t* si $A_{it} \in \{-1,-2\}$.

El cambio de era protagonizado por el advenimiento de la Televisión Digital, evidencia las profundas transformacionesque esta nueva tecnología incorpora a laexperiencia del usuario, ya que además de entretenersecon los contenidos que se difunden por las estacionesde televisión, éste tiene la posibilidad de interactuarcon este medio de comunicación a través de diferentesaplicaciones y modalidades, revolucionando así, elmodelo tradicional de mirar televisión. Actualmente,los estándares de transmisión de TV Digital adoptadosa nivel mundial permiten el transporte ágil de grandesvolúmenes de información, lo que propicia la disponibilidadde excesiva oferta televisiva en los hogares.Esta sobreoferta de alternativas de entretenimiento originará un escenario en el que el usuario se enfrentea una amplísima cantidad de programación disponible,que a pesar de contar

A partir de esta matriz, se construyen T instancias de KPPD,cada una de las cuales emplea los objetos disponiblesen un período determinado. La capacidad de la mochilaen cada instancia corresponde a la capacidad residual parael período correspondiente, luego de descontar el pesode los objetos presentes (Ait = 1):

$$\hat{C}_t := C - \sum_{i:A_{it}=1} w_i$$

Cada instancia de KP-PD se resuelve hasta la optimalidad empleando una versión modificada del algoritmo deHorowitz-Sahni. La matriz $\hat{X} \in \{0, 1\}^{n \times T}$se construyeusando como columnas las soluciónes óptimas de las Tinstancias de KP-PD, además de la información de objetospresentes o prohibidos. Si el valor de la función objetivopara $\hat{X}$ es inferior al valor de la mejor solución factibleencontrada hasta el momento, la solución es desechada(poda del árbol).

Caso contrario, se verifica la factibilidad de $\hat{X}$ Si esta solución es factible, se actualiza la mejor solución X:= $\hat{X}$,z:= $\hat{z}$. Si $\hat{X}$ no es factible, una ramificación tiene lugar ydos nuevas listas son agregadas a la cola Q. Para la ramificaciónhemos investigado tres posibles técnicas:

- La primera técnica de ramificación busca en las matricesA y en la solución actual $\hat{X}$ el primer elementolibre que haya sido seleccionado por la cota superior:

for i := 1, ..., n **do**
for t := T, ..., 1 **do**
if $A_{it} = -1$ and$\widehat{X_{it}} = 1$, **then**

$$\begin{cases} t_1 := (i, t, 1), \\ t_2 := (i, t, 0). \end{cases}$$

break;

- La segunda técnica de ramificación busca simplemente el primer objeto libre:

for i := 1, ..., n **do**
  for t := T, ..., 1 **do**
    if $A_{it} = -1$, **then**

$$\begin{cases} t_1 := (i, t, 1), \\ t_2 := (i, t, 0). \end{cases}$$

break;

La tercera técnica de ramificación ordena las filas de las matrices A y $\hat{X}$ en sentido descendente de los valoresde los objetos, siendo i = 1 la fila que correspondeal objeto de mayor valor e i = n la fila asociadaal de menor valor (originalmente, las filas están ordenadas descendentemente por la densidad *p/wd* de los objetos). Luego se procede como en la primera técnica.

En los tres casos, el elemento (i, t) seleccionado es empleado para construir dos ternas t_1, t_2 que representan las decisiones de forzar la entrada del objeto i en la mochila en el período t, o de prohibir su ingreso. Con estas ternas, se definen las nuevas listas L1:= L ∪ {t1} yL2:= L ∪ {t2}, las mismas que se añaden a la cola Q de listas por procesar, y la iteración termina. El algoritmo termina cuando todas las listas de ternas pendientes en Q han sido procesadas, y devuelve como resultado el valor óptimo del problema.

Ejemplo 1. Veamos el funcionamiento general del esquema para la siguiente instancia de KP-DUR:
Se tiene una mochila con capacidad C = 5, y 4 objetos con las siguientes características:

$valores \quad (p_i):$	2	11	6	3
$pesos \quad (w_i):$	1	4	4	2
$duraciones\ (d_i):$	2	3	2	3

Se quiere maximizar el valor total almacenado en la mochila en un horizonte temporal T = 4, y los objetos están ordenados de acuerdo a sus densidades $\frac{p1}{w1d1} > \frac{p2}{w2d2} > \ldots > \frac{p4}{w4d4}$. Laprimera solución obtenida mediante la heurística primal vienedada por:

$X =$

-1	-2	1	1
-1	-1	-2	-2
-1	-1	-1	-2
-1	-1	-2	-2

donde X_{it} tiene el valor de 1 cuando el objeto i ∈ {1, ..., 4} está presente en la mochila en el período t ∈ {1, ..., 4}. En este caso, la solución consiste en ingresar a la mochila el primer objeto en los tiempos t = 1 y t = 3, e ingresar el segundo objeto en el tiempo t = 1. El valor de esta solución inicial es z = 2 + 2 +11 = 15. Se generan entonces las ternas forzadas por el modelo y se construye la lista inicial.

$$L := \{(1,4,0), (2,3,0), (2,4,0), (3,4,0), (4,3,0), (4,4,0)\}.$$

Luego entramos en el lazo principal del esquema branch andbound.A partir de L se construye la primera matriz de disponibilidades:

$A =$

-1	-1	-1	-2
-1	-1	-2	-2
-1	-1	-1	-2
-1	-1	-2	-2

Con la matriz A se construyen T instancias de KP-PD independientes, cada una con los objetos disponibles y la capacidad residual existente en cada período de tiempo. En este caso, para as 4 instancias todos los objetos están disponibles y la capacidad residual es la capacidad total de la mochila. Al resolver cada instancia empleando nuestra modificación del algoritmo de Horowitz-Sahni, la solución encontrada es:

$\hat{X} =$

1	1	1	1
1	1	1	1
0	0	0	0
0	0	0	0

con un valor de ˆz = 18, que es mayor que el valor de la mejor solución z. Se comprueba entonces si esta solución es factible o no, en nuestro caso es infactible. Se procede a crear dos ternas de ramificación, empleando para este ejemplo la primera técnica de ramificación, que selecciona i = 1, t = 3 y t1:= (1, 3, 1), t2:= (1, 3, 0). Con estas ternas se crean las nuevas listas L1 = L ∪ {t1}, L2 = L ∪ {t2} y se las agrega al inicio de la cola Q, con lo que termina la primera iteración del lazo principal. En la segunda iteración, se retira la primera listaL1 de la cola, la matriz de disponibilidades obtenida a partir de ésta es:

$A =$

1	1	1	1
1	1	1	0
0	0	0	0
0	0	0	0

Para esta nueva matriz de disponibilidades se crean las T instancias de KP-PD y se resuelve independientemente cada una de ellas. Las instancias para t ∈ {1, 2} tienen nuevamente los cuatro objetos y toda la capacidad de la mochila disponibles. Para t ∈ {3, 4}, están disponibles los objetos 2, 3,4 y la capacidad residual es $\widehat{Ct}$ = C − w1 = 4. Aun así, el resultado obtenidosigue siendo el mismo:

$\hat{X} =$

1	1	1	1
1	1	1	1
0	0	0	0
0	0	0	0

Nuevamente, tenemos que el valor $\hat{z}$ es mayor que el de la solución actual z. Además, la solución es infactible y volvemos a generar dos nuevas ternas de ramificación t3:= (1, 1, 1), t4:=(1, 1, 0), junto con las nuevas listas L3 = L1 ∪{t3}, L4 =L1 ∪ {t4}, que son agregadas a la cola Q. Siete iteraciones más adelante, se llega a la solución factible:

$\hat{X} =$

1	1	1	1
0	0	0	0
1	1	1	1
0	0	0	0

con un valor de ˆz = 16. Como este valor supera al de la mejor solución encontrada hasta el momento, se actualiza ésta última:

$\hat{X} =$

1	1	1	1
0	0	0	0
1	1	1	1
0	0	0	0

Luego de 11 iteraciones, termina la exploración de todas las listas pendientes en Q, y se constata que esta solución es la solución óptima de la instancia.

3. RESULTADOS COMPUTACIONALES

3.1 Instancias de Prueba

Para analizar el desempeño práctico del algoritmo, se utilizaron instancias construidas mediante la modificación de un algoritmo de generación de instancias para la versión clásica de KP elaborado por David Pisinger [15]. Se realizaron cambios al mismo para que construya instancias para KP-DUR agregando duraciones a los objetos y un horizonte temporal T. El algoritmo de generación utilizados parámetros: un entero n que indica el número de objetos y un entero R en función del cual se definen todas las demás variables del problema. Este algoritmo genera tres tipos de instancias:

1. No correlacionadas, en las cuales C = 3R, T = R, y los valores, pesos y duraciones son generados independientemente simulando distribuciones uniformes de probabilidad:

$$p_j \rightsquigarrow U_{[1,R]}, \quad \forall j \in \{1,\dots,n\},$$
$$w_j \rightsquigarrow U_{[1,R]}, \quad \forall j \in \{1,\dots,n\},$$
$$d_j \rightsquigarrow U_{[1,R]}, \quad \forall j \in \{1,\dots,n\}.$$

2. Débilmente correlacionadas, en las cuales C = 3R, T =⌈1.1R⌉, y los valores, pesos y duraciones son generados como se indica a continuación:

$$p_j \rightsquigarrow U_{[1,R]}, \quad \forall j \in \{1,\dots,n\},$$
$$w_j \rightsquigarrow U_{\left[p_j-\frac{R}{10},\, p_j+\frac{R}{10}\right]}, \quad \forall j \in \{1,\dots,n\},$$
$$d_j \rightsquigarrow U_{\left[p_j-\frac{R}{10},\, p_j+\frac{R}{10}\right]}, \quad \forall j \in \{1,\dots,n\}.$$

3. Fuertemente correlacionadas, en las cuales C = 3R, T =⌈2.1R⌉, y los valores, pesos y duraciones se generan por medio de:

$$p_j \rightsquigarrow U_{[1,R]}, \quad \forall j \in \{1,\dots,n\},$$
$$w_j \rightsquigarrow U_{\left[p_j+\frac{R}{10},\right]}, \quad \forall j \in \{1,\dots,n\},$$
$$d_j \rightsquigarrow U_{\left[2p_j+\frac{R}{10}\right]}, \quad \forall j \in \{1,\dots,n\}.$$

Con este generador se construyeron 27 instancias de prueba para KP-DUR, las cuales se detallan en la Tabla1.

			R		
	n	Tipo	10	20	50
Instancias	20	1	Instancia_1	Instancia_4	Instancia_7
		2	Instancia_2	Instancia_5	Instancia_8
		3	Instancia_3	Instancia_6	Instancia_9
	50	1	Instancia_10	Instancia_13	Instancia_16
		2	Instancia_11	Instancia_14	Instancia_17
		3	Instancia_12	Instancia_15	Instancia_18
	100	1	Instancia_19	Instancia_22	Instancia_25
		2	Instancia_20	Instancia_23	Instancia_26
		3	Instancia_21	Instancia_24	Instancia_27

Tabla 1. Instancias de prueba para KP-DUR.

3.2 Eficiencia computacional

El desempeño computacional de nuestro algoritmo KPDBB fue comparado sobre las instancias de prueba descritas anteriormente contra el desempeño de una implementación directa del modelo KP-DUR en el solver general para programas enteros SCIP, versión 2.1 [1]. Las pruebas computacionales se desarrollaron en una máquina con procesador AMD Athlon (tm) X2 Dual-Core QL-64 2.10 GHz con memoria RAM de 4.00 GB y un sistema operativo Windows 7 de 64 bits. El código fue compilado empleando el compilador GNU GCC versión 4.5.2.Discutiremos a continuación en más detalle algunas instancias relevantes. Un factor a considerar al analizar la calidad de una solución es la brecha de optimalidad (gap):

$$gap = \frac{cota\ superior - mejor\ solución\ alcanzada}{mejor\ solución\ alcanzada}$$

Reportamos aquí únicamente resultados obtenidos con KPD-BB configurado para usar la primera heurística de ramificación, pues para ésta se presentó el mejor comportamiento en todas las instancias. Resultados más detallados se describen en [16]. El comportamiento de los algoritmos para la instancia 14 está detallado en la Figura1. Esta instancia es del tipo débilmente correlacionada, con n = 50 y R = 20. Podemos notar que el valor dela cota superior KP-PD es de 596, encontrada en t = 0sy es muy cercano al valor de la cota obtenida por SCIP, cuyo valor es de 589 en t = 1s. La primera solución factible encontrada mediante la heurística primal tuvo un valor de 566 obtenida en el tiempo t = 0s mientras que la primera solución factible encontrada por SCIP en el tiempo t = 10s tiene un valor de 580. Dentro del período de 1 hora KPD-BB encontró una solución factible con un valor de 579, que es un 1% menor al valor 585 dela solución óptima encontrada por SCIP en t = 260s. El gap inicial de KPD-BB fue de 5%, y se redujo a 2% en un período de una hora.

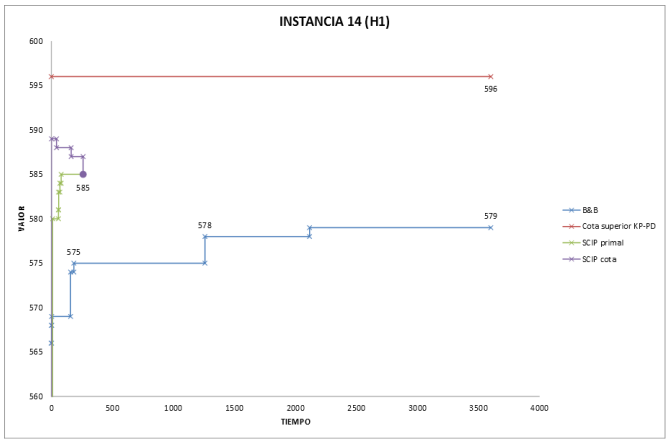

Figura 1. Comportamiento de los algoritmos para la instancia 14, heurística de ramificación 1.

En la Figura 2 se reportan los resultados obtenidos por los algoritmos SCIP y KPD-BB para la instancia 15. Esta instancia es del tipo fuertemente correlacionada con n = 50 y R = 20. Podemos observar que el valor de la cota superior KP-PD fue de 422 encontrada en t = 0s que es un 2% mayor a la obtenida por SCIP, con un valor de408 en t = 6s. La primera solución factible encontrada mediante la heurística primal tuvo un valor de 391obtenida en el tiempo t = 0s mientras que la primera solución factible encontrada por SCIP en el tiempo t = 8sEsta solución es un 1% menor que la solución óptima encontrada por el SCIP en t = 20s, de valor 403, que fue verificada como óptima en t = 1124s. Puede notarse que los primeros resultados obtenidos mediante la heurística prima y la cota superior de KPD-BB son competitivos con aquellos obtenidos por SCIP, tomando en cuenta su valor y su tiempo de cálculo tuvo un valor de 391. KPD-BB encontró una solución factible con un valor de 395 en el tiempo t = 23s la misma que no pudo ser mejorada en todo el período de cálculo.

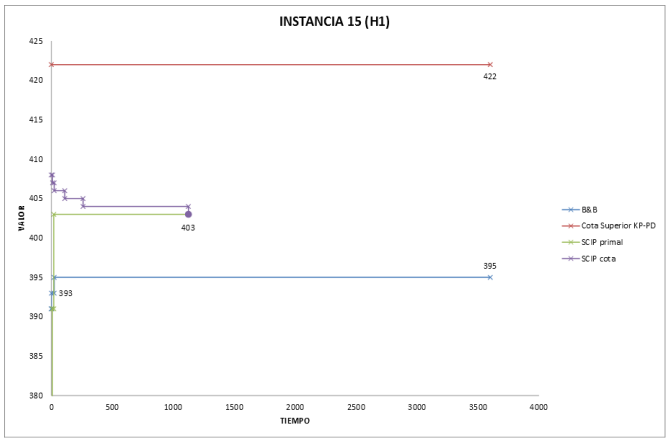

Figura 2. Comportamiento de los algoritmos para la instancia 15, heurística de ramificación 1.

Sin embargo, el esquema tiene problemas para "cubrir el último tramo" de la brecha de optimalidad (por lo general, menor al 5%). Esto puede deberse a la simetría de las soluciones factibles, la misma que fuerza a KPD-BB a desperdiciar tiempo de cálculo explorando soluciones de igual valor. Esto ocurre sobre todo en las instancias del tipo fuertemente correlacionadas. El gap inicial de KPD-BB fue de 7%, y se redujo a 6% en un período de una hora. En la Figura 3 se presentan los resultados para instancia 27, que es del tipo fuertemente correlacionada con n = 100 y R = 50. El valor de la cota superior KP-PD encontrada en t = 0s es de 1491, que es menor al de la última cota obtenida por SCIP, cuyo valor es de 1493 ent = 1411s. La primera cota superior obtenida por SCIP tuvo un valor de 1504 en t = 135s.

La primera solución factible encontrada mediante la heurística primal tiene un valor de 1400 obtenida en el tiempo t = 1s mientras que la primera solución factible encontrada por SCIP tiene un valor de 1400 en el tiempo t = 1411s. KPD-BB encontró una solución factible de mejor valor 1406 en t = 47s y ésta no pudo ser mejorada dentro del tiempo de cálculo permitido. SCIP encontró una solución factible más durante todo el tiempo de cálculo permitido cuyo valor es de 1416, obtenida en t = 2627s. El gap inicial de KPD-BB fue de 6,5%, y se redujo a 6% en un período de una hora. Se corrobora nuevamente que para instancias de gran tamaño, KPD-BB encuentra de forma eficiente soluciones factibles y cotas superiores de valores similares a aquellas obtenidas por SCIP en untiempo de cálculo considerablemente menor.

Figura 3. Comportamiento de los algoritmos para la instancia27, heurística de ramificación 1.

La simetría de soluciones sigue demostrando ser un problema para el esquema planteado ya que no permite explorar el árbol de decisiones de manera más eficiente.

4. CONCLUSIONES

Hemos presentado un modelo de programación entera para una versión dinámica del problema de la mochila, en la cual los objetos tienen asociados tiempos de permanencia en la mochila y se busca maximizar el valor en la misma dentro de un horizonte de tiempo. Hemos propuesto un algoritmo de solución exacta para este problema de optimización combinatoria, basado en el esquema de branch-and-bound. Finalmente, hemos estudiado el rendimiento computacional de nuestro algoritmo KPDBBal comparar su desempeño sobre instancias de prueba contra el desempeño del solver lineal entero general SCIP. Los experimentos computacionales de la sección precendente permiten concluir que el esquema KPD-BB encuentra rápidamente cotas superiores y soluciones primales de un valor aceptable, pero que generalmente no consigue cerrar la brecha de optimalidad dentro de un tiempo de cálculo razonable. Por el contrario, SCIP es más eficiente reduciendo la brecha de optimalidad pero requiere de un mayor tiempo de cálculo para encontrar la primera solución factible, sobre todo en instancias grandes (por ejemplo, en las instancias 22 a la 27). De esta manera, nuestro esquema de solución KPD-BB es particularmente adecuado para situaciones donde el tiempo de cálculos es un factor crítico a considerar. Por ejemplo, para aplicaciones en tiempo real.

REFERENCIAS

[1] T. Achterberg. Scip: Solving constraint integer programs. Mathematical Programming Computation, 1(1):1–41, 2009.http://mp.zib.de/index.php/MPC/artile/view/4.

[2] E. Balas and E. Zemel. An algorithm for large zerooneknapsack problems. Math.Oper. Res., 28(5):1130–1154, 1980.

[3] M. Bartlett, A. Frisch, Y. Hamadi, I.Miguel, S. Tarim,and C. Unsworth. The temporal knapsack problemand its solution. In R. Barták and M. Milano, editors,Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, volume 3524 of Lecture Notes in Computer Science, pages34–48. Springer Berlin / Heidelberg, 2005.

[4] G. Dantzig. Discrete variable extremum problems. Math. Oper. Res., 5(2):266–277, Apr. 1957.

[5] M. R. Garey and D. S. Johnson. Computers and Intractability:A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York, NY, USA, 1979.

[6] P. Gilmore and R. Gomory. A linear programmingapproach to the cutting stock problem. Math. Oper.Res., 9(6):849–859,Nov. 1961.

[7] E. Horowitz and S. Sahni. Computing partitionswith applications to the knapsack problem. J. ACM,21:277–292, April 1974.

[8] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, Complexity of Computer Computations, The IBM ResearchSymposia Series, pages 85–103. PlenumPress,New York, 1972.

[9] H. Kellerer and U. Pferschy. A new fully polynomialtime approximation scheme for the knapsackproblem. J. Comb. Optim., 3(1):59–71, 1999.

[10] H. Kellerer, U. Pferschy, and D. Pisinger. Knapsack Problems. Springer, Germany, 2004.

[11] S. Martello and P. Toth. An upper bound for thezero-one knapsack problemand a branch and bound algorithm. European Journal of Operational Research,1(3):169–175,May 1977.

[12] S. Martello and P. Toth. Knapsack Problems: Algorithms and Computer Implementations. Wiley, NewYork, NY, USA, 1990.

[13] J. D. Papastavrou, S. Rajagopalan, and A. J. Kleywegt. The dynamic and stochastic knapsack problem with deadlines. Management Science, 42:1706–1718, December 1996.

[14] R. Parra-Hernandez, D. Vanderster, and N. J. Dimopoulos. Resource management and knapsack formulations on the grid. In Proceedings of the 5thIEEE/ACM International Workshop on Grid Computing, GRID ’04, pages 94–101,Washington, DC, USA,2004. IEEE Computer Society.

[15] D. Pisinger. David pisinger's optimization codes. http://www.diku.dk/~pisinger/ odes.html.

[16] B. Silva. Algoritmos de solución para una versión dinámica del problema de la mochila. Tesis de grado en Ingeniería Matemática, Escuela Politécnica Nacional, Quito, Ecuador, 2014.

[17] V. V. Vazirani. Approximation algorithms. Springer, Berlin, 2001.