Programación Matemática y Software (2014) 6 (2): 57-64. ISSN: 2007-3283

Un algoritmo de búsqueda local iterada como solución al problema de la mochila

An algorithm of iterated local search as a solution to the knapsack problem

Yainier Labrada-Nueva,[1]* Juana Enríquez-Urbano,[1] Yadián García-Ojito[1]

[1] Departamento de Biometría, Centro de Identificación y Seguridad Digital (Cised), Universidad de las Ciencias Informáticas (UCI).
Carretera a San Antonio de los Baños km 2.5, Reparto Torrens, Boyeros. La Habana, Cuba. CP 19370
*correo-e: yainier@gmail.com

PALABRAS CLAVE:

optimización combinatoria, heurística, búsqueda local iterada, problema de la mochila

RESUMEN

El problema de la mochila se clasifica como un problema de optimización combinatoria y, desde el punto de vista computacional, entra en la categoría de los problemas NP-completos [1]. En este artículo se soluciona el problema de la mochila aplicando un algoritmo de búsqueda local iterada en el contexto de la distribución de productos almacenados en bodegas y a su distribución en tiendas. En los resultados obtenidos se observa la convergencia del algoritmo implementado.

KEYWORDS:

combinatorial optimization, metaheuristics, iterated local search, knapsack problem

ABSTRACT

The knapsack problem is classified as a combinatorial optimization problem and from the computational point of view falls into the category of NP-complete [1] problems. In this article the knapsack problem is solved applying a local search algorithm iterated in the context of the distribution of products stored in warehouses and distribution in stores. In the results of the implemented algorithm the convergence is observed.

Recibido: 6 de marzo de 2014 • **Aceptado:** 4 de junio de 2014 • **Publicado en línea:** 30 de junio de 2014

1 INTRODUCCIÓN

El problema de la mochila está clasificado en el tipo de optimización combinatoria que, desde el punto de vista computacional, se clasifica como un problema NP-completo [1], esto significa que hasta el momento no existe un algoritmo exacto que pueda resolverlo en un tiempo polinomial. Si existe una solución exacta, no se encuentra; la que se encuentra es una solución próxima a la mejor en un tiempo polinomial. Esta condición implica que para solucionar el problema que se está analizando se aplique un algoritmo heurístico (búsqueda local iterada). Los métodos heurísticos son reglas empíricas para encontrar soluciones a problemas basados en la experiencia y que no tienen pruebas de optimalidad.

El problema de la mochila modela una situación similar al llenado de una mochila: se tiene una mochila con una capacidad de soporte y esa capacidad es el peso que puede cargar. También se tiene un conjunto de elementos con un peso y un beneficio específico. Al introducir estos elementos en la mochila, siempre se verifica que el peso no sobrepase la capacidad que soporta la mochila y, a medida que se introducen más elementos, asimismo se verifica que la suma de los pesos de los elementos que están dentro de la mochila más el peso del elemento que se analiza en ese momento, no sobrepase el peso máximo que soporta la mochila.

Éste es un problema que se observa en la vida cotidiana; por ejemplo, al distribuir productos en bodegas donde se tienen que maximizar las ganancias, una forma es conocer el peso de cada producto y la capacidad de carga de cada caja donde se acomodarán estos productos para aprovechar mejor su espacio.

En ese caso las cajas representan la mochila con un peso de soporte máximo y cada uno de los productos, un elemento con un peso y un beneficio. Los productos se acomodan en las cajas en función de su capacidad y del peso de cada producto. En las cajas siempre se acomodan los productos que más beneficios tengan, siempre y cuando el peso del producto no sobrepase el máximo permitido.

2 ESTADO DEL ARTE

En la comunidad científica muchos autores han resuelto el problema de la mochila, varios han usado métodos heurísticos como la búsqueda local iterada, *branch and bound*, algoritmos genéticos, entre otros. En un estudio [2] le dan solución al problema de la mochila mediante un algoritmo GRASP (*Greedy Randomized Adaptive Search Procedure*) y una búsqueda local iterada; usan lo mejor que posee cada uno de estos algoritmos y realizan pruebas experimentales utilizando los *benchmark*. En otro trabajo [3] aplican un algoritmo genético para encontrar una respuesta satisfactoria y hacen pruebas experimentales a través de los *benchmark*.

3 BÚSQUEDA LOCAL ITERADA

Existen diferentes métodos con los que se puede generar una solución mucho mejor en comparación con los que actualmente se utilizan. Los métodos de optimización tienen un submétodo llamado búsqueda local iterada, el cual está basado en una heurística determinística. Para darle solución al problema que trata este trabajo, se aplica un algoritmo de búsqueda local iterada porque representa una técnica potente, robusta, eficiente y además es sencillo de implementar [4]. Según Glover y colaboradores [5], el método analizado es un método estocástico de búsqueda local que iterativamente aplica búsquedas locales a perturbaciones del punto actual (solución inicial) de una manera aleatoria en un espacio de soluciones óptimas. Este método trabaja en conjunto con la búsqueda local, y es mediante éste que se obtiene la primera solución con la cual se iniciará el recorrido por el espacio de soluciones hasta mejorar progresivamente.

Para comprender mejor el funcionamiento de la búsqueda local iterada, es necesario entender, en primer lugar, cómo funciona la búsqueda local:

1. Del espacio de soluciones, se obtiene una primera composición generada estocásticamente, la cual se evalúa mediante la función objetivo y se genera la solución inicial (S) [6].
2. A esta solución inicial se le aplica un cambio con alguna estructura de vecindad, se evalúa con la función objetivo y se obtiene una nueva solución (S').
3. Si S' es mejor que la actual, S será sustituida, de lo contrario permanecerá igual; es decir, si $S' \square S$ entonces $S \neg S'$.
4. Dentro de la búsqueda local se repiten los pasos 2 y 3 hasta que se cumpla con el criterio de paro.

El cambio que se aplique a S mediante alguna estructura de vecindad será el que determine el tipo de solución obtenida; si ésta fue mejor, será remplazada; de lo contrario, permanecerá hasta encontrar una que la sustituya.

Para salir del espacio de soluciones óptimas locales, es necesario hacer uso del algoritmo de búsqueda local iterada [5], el cual, una vez obtenido el valor óptimo local, es tomado como el mejor e iniciará la siguiente iteración. Con este nuevo valor, si en las siguientes iteraciones se encuentra uno mucho mejor que el actual, lo reemplaza; si no es así, se genera un nuevo cambio hasta encontrar uno mejor que el actual.

En forma de pseudocódigo, el algoritmo de búsqueda local iterada se muestra en la figura 1.

```
Hacer
    Generar solución inicial s
    Hacer
        s*=Estructura_de_Vecindad(s)
        Si(f(s*)>=f(s)) entonces
           s=s*
            Si(f(s*)>f(Sm-LS)) entonces
              Sm-LS=s*
            Fin-si
        Fin-si
    Hasta CP-LS
    Si(f(Sm-LS)>f(Sm-ILS)) entonces
        Sm-ILS = Sm-LS
    Fin-si
Mientras CP_ILS
```

Figura 1. Búsqueda local iterada [7]

Modelo matemático

El modelo matemático del problema de la mochila es representado como un problema de programación lineal entera como muestran las ecuaciones 1 a 6 [1]. Se define:

- C como la capacidad de la mochila.
- P_i como el beneficio unitario obtenido por ingresar el producto i en la mochila.
- W_i como el peso del producto i.
- N como la cantidad de productos.

Luego de definir las variables, se plantea como modelo matemático de la siguiente forma [1]:

Maximizar $$f = \sum_{i=1}^{n} pixi \quad (1)$$

Sujeto a: Restricciones MM

$$\sum_{i=1}^{n} wixi \leq C \quad (2)$$

$$c \geq 0 \quad (3)$$

$$p_i > 0 \quad (4)$$

$$w_i > 0 \quad (5)$$

$$x_i = \{0, 1\} \quad (6)$$

La ecuación 1 corresponde a la función objetivo que significa que se va a maximizar la suma de los beneficios; las demás ecuaciones son restricciones. La número 2 indica que la suma de todos los pesos de los elementos siempre tiene que ser menor o igual que la capacidad en peso que soporta la mochila. Las ecuaciones 3, 4 y 5 muestran que tanto la capacidad de la mochila como el peso y el beneficio de los elementos son números positivos. Por último, la variable x toma valores de 0 ó 1 cuando un elemento está o no dentro de la mochila; cuando el valor es cero, el elemento no está dentro de la mochila, en caso de que x tome valor de 1, significa que el elemento está dentro de la mochila.

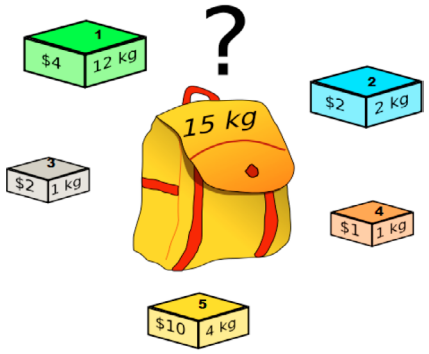

Figura 2. Representación del problema de la mochila

En la figura 2 se expone una idea del problema que se está analizando. Se tiene una mochila de 15 kg que se puede llenar con cajas de distinto peso y beneficio, se debe elegir cada una de las cajas en función de su peso y beneficio sin exceder los 15 kg de peso que soporta la mochila. Como se puede observar, cada caja representa un elemento: el elemento 1 tiene un beneficio de $4 y un peso de 12 kg; el elemento 2 tiene un beneficio de $2 y un peso de 2 kg; el elemento 3

tiene un beneficio de $2 y pesa 1 kg; el elemento 4 tiene un beneficio de $1 y un peso de 1 kg; el elemento 5 tiene un beneficio de $10 y un peso de 4 kg. Lo que se hace es llenar la mochila con los elementos que más beneficios posean, sin que los pesos de los elementos excedan la capacidad de la mochila.

4 DESCRIPCIÓN DEL ALGORITMO

En este algoritmo inicialmente se generan de forma aleatoria los elementos que serán parte de la solución. Como se explicó al principio del artículo, cada elemento posee un peso y un beneficio, ambos datos son enteros; éstos conforman los datos de entrada del algoritmo y se almacenarán en un vector que los represente. También existe otro vector que representa la solución del problema; ya que en este vector sólo estarán los elementos que cumplan con un peso menor a la capacidad máxima de soporte de la mochila; estos elementos se toman de forma aleatoria. El peso que soporta la mochila está representado por la tercera parte de la suma de todos los elementos que inicialmente se generaron de forma aleatoria.

También existe otro vector que representa las soluciones, ya que en él existe un dato en donde se almacena la solución actual; ese dato es el beneficio total, que representa la suma de todos los beneficios de los elementos dentro del arreglo que constituye la solución.

Luego de llegar a la primera solución, se comienza a buscar soluciones vecinas mediante una búsqueda local iterada. Ésta se realiza con un reemplazo de un elemento que está dentro del arreglo que representa la solución por otro elemento que está dentro del arreglo de donde se tomaron los elementos que conforman la solución; este reemplazo es aleatorio. Una vez hecho el reemplazo, se evalúa nuevamente la función objetivo para obtener una nueva solución. Esta solución se compara con la que se tenía anteriormente: si la nueva solución es mejor, se reemplaza; si no, la solución que prevalece es la que había anteriormente.

La figura 3 representa el diagrama de flujo del algoritmo para un mejor entendimiento del funcionamiento del mismo.

5 ESTRUCTURA DE VECINDAD

Para decidir la forma en la cual se debe determinar el espacio de soluciones del problema que se aborde,

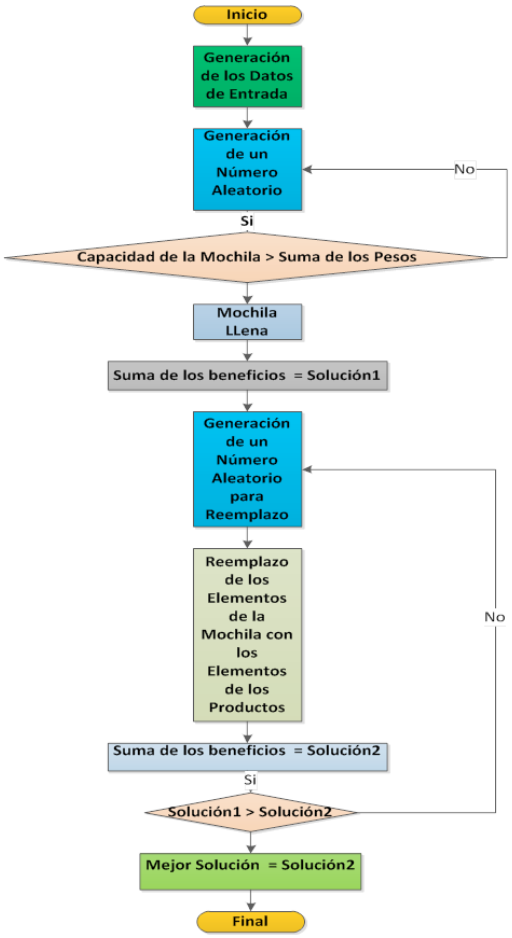

Figura 3. búsqueda local iterada

primero se debe conocer y entender el problema [8]. Es así como se define el espacio de soluciones que tendrá un vecindario, sin olvidar el objetivo: maximizar.

Se parte de una solución factible que cumple con las restricciones del modelo matemático, en este caso, los elementos que conforman esta solución factible son aquellos cuyo peso no sobrepasa el peso que soporta la mochila. La solución inicial también son todos aquellos elementos cuya suma de pesos en conjunto no sobrepasa el peso que soporta la mochila. Al contar con eso se tiene una solución factible que cumple con las restricciones del modelo matemático.

Para explicar lo que es una vecindad (figura 4), se toma la primera solución a la que se llega; ésta se representa como un punto al cual se asigna una s, todas las soluciones que estén cerca de ese punto serán consideradas el vecindario representado por $N(s)$, y S representa el espacio de soluciones S [6]. Gráficamente se observa así [7]:

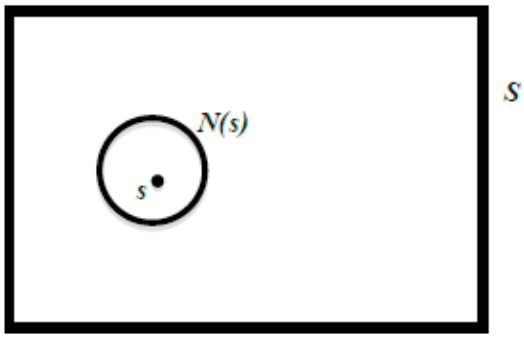

Figura 4. Un espacio de soluciones *S*, una solución potencial *s* y su vecindario *N(s)* indicado por el círculo interior alrededor de *s*

Matemáticamente se observa que $s \in S$; el vecindario de *s* se define como un conjunto de posibles soluciones cerca de *s*, *N(s)*, representado por la función. Si a partir de este punto se genera otro que mejore el primero, éste será representado por *s'*. De esta manera, hay que moverse paso a paso desde una solución inicial hacia una solución que mejore la anterior, que se evalúe en la función objetivo y que cumpla con todas las restricciones del modelo matemático.

$$f(S') >= f(S)$$

6 ESTRUCTURA DE VECINDAD CON POSICIÓN ALEATORIA

Para la estructura de búsqueda de vecindad con posición aleatoria, inicialmente se poseen dos vectores: en el primer vector están todos los elementos que representan los datos de entrada al algoritmo y se denomina vector de elementos; el segundo vector representa la solución inicial factible, que está formada por los elementos cuyos pesos no sobrepasaron el peso que soporta la mochila, a este vector se le denomina vector solución.

Una vez que se tengan ambos vectores, se genera un número aleatorio que esté dentro del rango de la longitud de los dos. Ese número aleatorio representa la misma posición en los dos vectores; luego se obtiene un elemento en cada uno de los vectores y se realiza el reemplazo. Cuando se realiza el reemplazo, se obtiene esta nueva solución vecina y con ésta se vuelve a evaluar la función objetivo del modelo matemático; luego se compara la solución obtenida con la que se tenía anteriormente y, si esta solución es mejor, se reemplaza por la anterior; si no, la anterior se mantiene. Siempre se guarda la mejor solución por cada iteración, la figura 5 es un ejemplo de cómo se hace este proceso.

7 RESULTADOS EXPERIMENTALES

Para la implementación de este algoritmo se utilizó el lenguaje de programación C++ en la plataforma Visual Studio .Net 2010, y como hardware se utilizó una *laptop* con procesador Core i3 con 4 Gigas de Memoria RAM. Luego de hacer 30 pruebas se presentan los resultados a partir del promedio de las pruebas hechas al algoritmo implementado. La figura 6 muestra la gráfica de los beneficios en función de la cantidad de iteraciones.

Aquí se puede apreciar que el beneficio crece en forma lineal hasta llegar a un valor determinado que es el máximo beneficio que se obtiene; luego de llegar a este valor, el beneficio se mantiene constante a medida que aumentan las iteraciones del algoritmo. Lo anterior significa que a partir de este valor el algoritmo comienza a converger. La forma de crecimiento del beneficio en este algoritmo puede aproximarse al comportamiento de una recta, cuya ecuación se puede escribir como $y=m*x + n$, donde $m = 700$ es la pendiente de esta recta, y *n* es el valor donde esta recta corta al eje de las *y*. En este caso, como la aproximación del crecimiento del beneficio no corta al eje de las *y*, entonces el valor de *n* es cero, por lo que la ecuación quedaría de la siguiente forma: $y=700x$; esto quiere decir que para cualquier valor de una iteración se puede obtener el valor del beneficio, siempre y cuando la iteración esté en un intervalo de 0 a 300, que aproximadamente es el intervalo en donde el beneficio crece. Después de 300 iteraciones aproximadamente, el beneficio comienza a ser constante, es decir el mismo; es entonces cuando se dice que se llega a un máximo local.

La figura 7 representa la gráfica de los beneficios en función del tiempo.

Aquí se puede apreciar que, a medida que transcurre el tiempo, el beneficio aumenta. Como se observa en la figura 7, desde que comenzó la ejecución del algoritmo hasta transcurridos aproximadamente 40 minutos, el valor del beneficio comienza a crecer y llega un momento en que comienza a ser constante. Luego se observa que nuevamente comienza a crecer; lo hace aproximadamente a partir de los 55 minutos y hasta los 90 minutos. En este intervalo el

Solución

Posición	0	1	2	3	4	5	6	7	8	9	n....	Total
Beneficio	9	5	6	7	8	6	8	7	3	2	n....	58

Elementos

Posición	0	1	2	3	4	5	6	7	8	9	n....
Beneficio	19	45	42	12	21	34	18	23	19	123	n....

Búsqueda Local
Solución

Posición	0	1	2	3	4	5	6	7	8	9	n....	Total
Beneficio	9	45	6	7	8	6	8	7	3	2	n....	101

Elementos

Posición	0	1	2	3	4	5	6	7	8	9	n....
Beneficio	19	5	42	12	21	34	18	23	19	123	n....

Figura 5. Estructura de vecindad con posición aleatoria

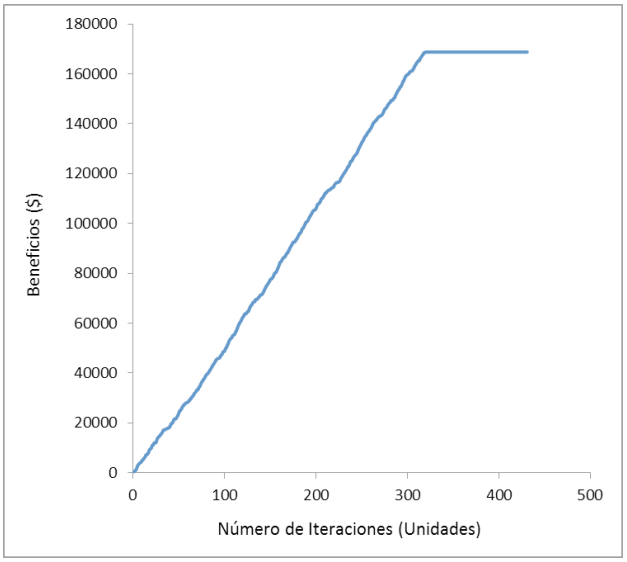

Figura 6. Beneficio en función de las iteraciones

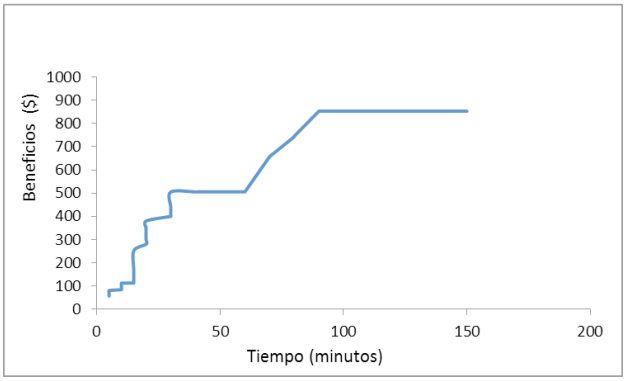

Figura 7. Beneficio en función del tiempo

crecimiento del beneficio puede ser aproximado a una línea recta con ecuación $y=mx+n$ con pendiente m; luego se observa que, aproximadamente a partir de los 90 minutos, el beneficio llega a su máximo valor, comienza a ser constante y entonces se manifiesta la convergencia del algoritmo porque llega al máximo local.

Tanto la figura 6 como la 7 muestran la convergencia del algoritmo. Cabe resaltar que estas gráficas muestran el promedio de las 30 pruebas experimentales que se realizaron al mismo. La primera gráfica representa el promedio de las 30 pruebas realizadas con un total de 1 000 elementos dentro de la mochila, es decir, 1 000 elementos dentro del vector que conforma la solución. La segunda gráfica de la figura 7 representa las 30 pruebas que se hicieron con 100 elementos dentro del vector que representa la solución.

La figura 8 representa el promedio de las 30 pruebas realizadas al algoritmo implementado. Estas pruebas se realizaron con 50 elementos dentro del vector que representa la solución. Precisamente lo que quiere decir es que, a medida que pasa el tiempo, el beneficio aumenta hasta llegar a un valor determinado, pero no necesariamente debe aumentar el peso, sino que éste puede aumentar o disminuir en función del elemento.

Esto se puede observar, por ejemplo, en tiendas que se dediquen a la compra y venta de productos. En ellas cada producto tiene un peso y un beneficio, y es posible que el producto tenga un alto beneficio pero el peso sea mínimo. Un ejemplo de esto puede ser una memoria flash comparada con una caja de bombones; en este caso, en función de la necesidad del cliente de tener una memoria flash y no una caja de bombones,

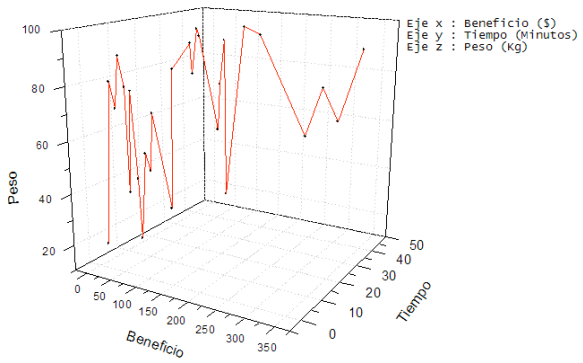

Figura 8. Beneficio en función del peso y del tiempo

la memoria tiene un beneficio grande y, sin embargo, no tiene un peso significativo, mientras que la caja de bombones tiene un peso mayor pero un beneficio menor.

8 CONCLUSIONES

Se puede apreciar que, a medida que transcurre el tiempo, los beneficios aumentan pero no necesariamente el peso. Este parámetro puede aumentar o disminuir aun al aumentar el beneficio, y esto es posible, puesto que el beneficio de cada elemento es independiente del peso que tiene. Esto permite que la solución al problema de la mochila pueda enfocarse en la distribución de productos en una bodega, cuando se tienen que maximizar los beneficios de los productos, dependiendo de su peso y del peso que soporten las cajas en donde se almacenarán.

REFERENCIAS

1. Papadimitriou, C. H., Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*. New York: Englewood Cliffs, 1998.
2. Soares-Vianna, D., Dianin-Vianna, M. F. Local search-based heuristics for the multiobjective multidimensional knapsack problem. *Producao*. 2013, 23 (3), 478-487.
3. Hristakeva, M., Shrestha, D. Solving the 0-1 Knapsack Problem with Genetic Algorithms. *Midwest Instruction and Computing Symposium*. 2004.
4. Lourenco, H. R., Martin, O. C., Stutzle, T. Iterated local search, en Glover, F., Kochenberger, G. (eds.). *Handbook of Metaheuristics*. Norwell: Kluwer Academic Publishers, 2002, 321-353.
5. Khebbache, S., Prins, C., Yalaoui, A. Iterated local search algorithm for the constrained two-dimensional non-guillotine cutting problem. *Journal of Industrial and Systems Engineering*. 2008, 2 (3), 164-179.
6. Michaelewicz, Z., Fogel, D. B. *How to Solve it: Modern Heuristics*. Berlín: Springer-Verlag, 2004.
7. Cruz-Chávez, M. A., Martínez-Oropeza, A., Serna-Barquera, S. A. Neighborhood hybrid structure for discrete optimization problems. En: Electronics, Robotics and Automotive Mechanics Conference, CERMA2010. México: IEEE-Computer Society, 2010, 108-113.
8. Joyanes, A. L., Zahonero, M. I. *Programación en C: Metodología, algoritmos y estructura de datos*. México: McGraw-Hill, 2000.

Acerca de los autores

Yainier Labrada-Nueva es Ingeniero en Ciencias Informáticas por la Universidad de las Ciencias Informáticas (UCI) en la Habana, Cuba. Actualmente trabaja en el Centro de Identificación y Seguridad Digital (Cised), perteneciente a la Facultad 1. Es profesor docente en el Departamento de Ciencias Básicas de la misma facultad. Ha impartido clases de Matemáticas, Física y Programación web como asignatura optativa.

Juana Enríquez-Urbano es Ingeniera Mecánica por el Instituto Tecnológico de Orizaba y Maestra en Ciencias en Ingeniería Mecánica con especialidad en Sistemas Térmicos por el Centro Nacional de Desarrollo Tecnológico (Cenidet).

Yadián García-Ojito es Ingeniero en Ciencias Informáticas por la Universidad de las Ciencias Informáticas (UCI) en la Habana, Cuba. Actualmente trabaja en el Departamento de Tecnología, perteneciente a la Facultad 1. Es profesor docente en el Departamento de Programación de la misma facultad. Ha impartido clases de Ingeniería de Software, Gestión de Software y Programación web.