Revista EIA, ISSN 1794-1237 Número 8, p. 99-115. Diciembre 2007
Escuela de Ingeniería de Antioquia, Medellín (Colombia)

METAHEURÍSTICOS: UNA ALTERNATIVA PARA LA SOLUCIÓN DE PROBLEMAS COMBINATORIOS EN ADMINISTRACIÓN DE OPERACIONES

Mario César Vélez*
José Alejandro Montoya**

RESUMEN

La escasa difusión que se les ha dado a las nuevas técnicas de solución de problemas complejos en las áreas de administración de operaciones por parte de universidades y publicaciones no académicas tiene como consecuencia directa que las empresas pierdan oportunidades para operar con más eficiencia y a menores costos. Este artículo pretende divulgar las ideas fundamentales detrás de una de las técnicas de solución de problemas combinatorios de más desarrollo en los últimos años: los metaheurísticos. Para ilustrar estas ideas se presenta un ejemplo de un problema combinatorio clásico en el área del secuenciamiento de operaciones y se propone un algoritmo de solución que hace uso de algunas de estas técnicas.

PALABRAS CLAVE: metaheurísticos; optimización; administración de operaciones.

ABSTRACT

The scarce diffusion given to the newest techniques for solving complex operations management problems has as a direct consequence that companies lose opportunities to operate at lower costs and higher efficiency. The objective of this article is to introduce and explain the fundamental ideas behind metaheuristics, a solution technique for combinatorial problems that has received the most attention from the academic community in the last few years. In order to illustrate these ideas, an example of a classical combinatorial

* Ingeniero de Producción, Universidad EAFIT; Master en Ingeniería Industrial, Universidad de los Andes Master in Industrial and Systems Engineering, Georgia Institute of Technology; Profesor Asistente, Universidad EAFIT. marvelez@eafit.edu.co

** Estudiante de Ingeniería de Producción, Universidad EAFIT. jmonto36@eafit.edu.co

Artículo recibido 24-VII-2007. Aprobado 18-XI-2007
Discusión abierta hasta junio de 2008

problem in the sequencing of operations area is presented, and a solution algorithm making use of some of these techniques is proposed.

KEY WORDS: Metaheuristics; optimization; operations management.

INTRODUCCIÓN

Resolver un problema de optimización es encontrar la mejor solución posible a un problema formulado en lenguaje matemático, donde el criterio que evalúa la calidad de una solución es cuantitativo, generalmente asociado a un costo y denominado *función objetivo*. En un problema combinatorio de optimización se desea encontrar un orden específico sobre un conjunto de elementos discretos (Aarts y Lenstra, 2003; Sait y Youssef, 1999). Para ilustrar esta definición puede considerarse el problema de encontrar la ruta que debe seguir un viajero para visitar un número determinado de ciudades, de manera que la distancia recorrida sea mínima. En este problema una solución es una posible ruta (orden), y la solución óptima es la ruta que minimiza distancia recorrida (un orden específico).

La definición formal del problema general de optimización combinatoria puede verse en Blum y Roli (2003) y en Sait y Youssef (1999). Problemas que se ajustan a esta definición aparecen en campos tan diversos como en el diseño de nuevas moléculas, de redes de telecomunicaciones y de nuevas aleaciones, en el posicionamiento de satélites, en la planeación de redes de transmisión de energía y en el desarrollo de circuitos impresos (Grötschel y Lovasz, 1995; Hoffman, 2000). La administración de operaciones no es la excepción, presentándose a diario problemas de este tipo. A continuación se enumeran los más representativos, según Kolen y Lenstra (1995):

- *El problema del agente viajero o TSP.*[1] Consiste en encontrar la secuencia en que un viajero debe visitar *n* ciudades, de manera que la distancia recorrida sea mínima. Encontrar el recorrido más corto en el que un taladro automático puede hacer un número determinado de perforaciones o determinar la ruta más corta en la que un operario de un almacén debe recorrer las estanterías y recoger un pedido de múltiples productos son algunos ejemplos. Una recopilación de problemas con esta estructura puede examinarse en Hoffman y Padberg (2000).
- *Diseño de rutas o VRP*.[2] Consiste en determinar el menor número de vehículos necesarios para atender un conjunto de clientes en una zona geográfica y la ruta que debe seguir cada uno para visitar a todos los clientes en un intervalo de tiempo y al menor costo. Este problema aparece en empresas de mensajería o envío de paquetes, donde se deben distribuir los paquetes recibidos; o en empresas que atienden directamente a sus clientes con una flota de vehículos propios.
- *Diseño de redes*. Otro problema común en administración de operaciones consiste en determinar la ubicación óptima de uno o varios centros de distribución, o el sitio donde se debe localizar una planta de manera que se minimice el costo de operación.
- *Secuenciamiento*. Para Pinedo (2001), el problema consiste en asignar recursos limitados a tareas en el tiempo. Aunque en esta definición las tareas son el objeto del secuenciamiento, es común encontrar que en la literatura se hable del secuenciamiento de trabajos en máquinas, donde el término "trabajo" se refiere a un conjunto de tareas que deben ejecutarse en un orden determinado, y el

[1] Del inglés *Traveling Salesman Problem*.

[2] Del inglés *Vehicle Routing Problem*.

término "máquina" se refiere al recurso limitado. En el problema del secuenciamiento hay que encontrar los tiempos en que se deben iniciar cada una de las tareas de manera que se cumpla con cierto criterio de optimización, el más común de los cuales consiste en encontrar el menor tiempo total de fabricación. Dentro del gran número de publicaciones en este campo, Pinedo (2001) es tal vez la mejor fuente de referencia.

Grötschel (1991) y Kolen y Lenstra (1995) presentan una amplia colección de ejemplos reales de problemas combinatorios en administración de operaciones.

1. PROBLEMAS FÁCILES Y PROBLEMAS DIFÍCILES

Ante un problema de optimización, la primera pregunta que se debe responder es si es fácil o difícil de resolver. Aunque parece una pregunta simple, sólo a partir de la década de los setenta los investigadores abordaron este tema, introduciendo un nuevo campo de investigación: la complejidad computacional, la cual, entre otros usos, determina si un problema es fácil o no de acuerdo con los algoritmos conocidos para resolverlo.

Un algoritmo es un conjunto ordenado y finito de operaciones que permiten solucionar un problema. Para la teoría de la complejidad computacional, la capacidad de un algoritmo para resolver un problema la determina el número de operaciones aritméticas necesarias para su ejecución. Un problema es fácil si existe un algoritmo que lo resuelve en tiempo polinomial; es decir, si el número de operaciones necesarias para que el algoritmo resuelva el problema es una función polinomial del tamaño del problema. Si esta función no es polinomial, se dice que el algoritmo es no polinomial y el problema se considera difícil.

Aunque esta clasificación parezca sólo de interés teórico, en la práctica resulta de gran importancia. Saber si un problema se puede resolver en tiempo polinomial equivale a saber si es posible encontrar la solución óptima en unos cuantos segundos o minutos o si, por el contrario, son necesarios años o incluso siglos para hacerlo. En el último caso surge la necesidad de desarrollar estrategias para encontrar soluciones buenas a un costo computacional razonable. Ahuja, Magnati y Orlin (1993), Bertsimas y Tsitsiklis (1997), T'kindt y Billaut (2005) y Tovey (2002) presentan un tratamiento detallado sobre el tema.

De otro lado, mientras los problemas se clasifican en fáciles o difíciles, los algoritmos se clasifican en exactos o completos y de aproximación o heurísticos. Los algoritmos exactos son aquellos en los que existe la garantía de que encontrarán la solución óptima; mientras que para los algoritmos de aproximación sólo se puede afirmar que encontrarán una solución aceptable, no necesariamente óptima (Blum y Roli, 2003; Sait y Youssef, 1999).

Como los algoritmos exactos más eficientes conocidos hoy para resolver problemas combinatorios requieren un número exponencial de operaciones (Aarts y Lenstra, 2003), podemos afirmar que estos problemas son difíciles. Estos algoritmos, entre los que se encuentran la programación dinámica y el algoritmo de ramificación y acotamiento, implican un costo computacional en ocasiones tan elevado que hace que deban descartarse como alternativa de solución.

2. LOS METAHEURÍSTICOS Y LA OPTIMIZACIÓN COMBINATORIA

Debido a que la mayoría de los problemas de optimización combinatoria se clasifican como difíciles (Aarts y Lenstra, 2003), la investigación se ha concentrado en desarrollar algoritmos de aproximación (Blum y Roli, 2003; Martí, 2003). Dentro de este área, el término metaheurístico lo introdujo Glover (1986) al definir una clase de algoritmos de aproximación que combinan heurísticos tradicionales con estrategias eficientes de exploración del espacio de búsqueda (Blum y Roli, 2003). Osman y Kelly (1996) proponen la siguiente definición:

Los metaheurísticos son métodos aproximados diseñados para resolver problemas de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Los metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos.

La mayor ventaja de los metaheurísticos frente a otros métodos está en su gran flexibilidad, lo que permite usarlos para abordar una amplia gama de problemas (Voß y Woodruff, 2006). Esta característica ha hecho que en los últimos años hayan cobrado una relevancia que se ve reflejada en el surgimiento de revistas especializadas como el *Journal of Heuristics* y el *INFORMS Journal on Computing*, y en la publicación de un importante número de libros, entre los que se destacan los de Aarts y Lenstra, 2003; Blum y Roli, 2003; Glover y Kochenberger, 2003; Martí, 2003; Melián, Moreno y Vega, 2003; Osman y Kelly, 1996, y Sait y Youssef, 1999.

Entre los metaheurísticos más exitosos se encuentran el recocido simulado (*simulated annealing*), la búsqueda tabú (*tabu search*), los algoritmos genéticos (*genetic algorithms*) y las redes neuronales artificiales (*artificial neural networks*). Otras ideas recientes incluyen la optimización por colonias de hormigas (*ant collony optimization*), la búsqueda local iterativa (*iterated local search*) y la computación evolutiva (*evolutionary computing*), entre otras. Para revisar las ideas más importantes, así como una clasificación de los tipos de metaheurísticos y una descripción de las características deseables en ellos, se puede recurrir a Hoffman (2000); una recopilación completa y reciente sobre los más importantes metaheurísticos la exponen Glover y Kochenberger (2003).

En el presente trabajo se exponen y se ilustran con un ejemplo las ideas que dieron origen a los tres metaheurístcos más utilizados en investigación de operaciones (Sait y Youssef, 1999): el recocido simulado, los algoritmos genéticos y la búsqueda tabú.

A continuación se indica la notación común usada en los algoritmos:

Sea Ω el espacio de búsqueda o conjunto de todas las posibles soluciones

Sea $x_i \in \Omega$ un elemento del conjunto de posibles soluciones

Sea $N(x_i) \subseteq \Omega$ el conjunto de soluciones vecinas de x_i.

Sea f: $\Omega \rightarrow \Re$ la función objetivo

Sea $x_0 \in \Omega$ una solución inicial arbitraria

Con relación a la definición de N(x), se dice que dos soluciones son vecinas si son cercanas entre sí (Aarts y Lenstra, 2003), donde el criterio de cercanía está restringido a la forma como se representen las soluciones del problema; y aun para un mismo problema puede existir una variedad de posibles vecindarios.

3. ALGORITMO DE RECOCIDO SIMULADO

El recocido simulado es uno de los metaheurísticos más utilizados en optimización combinatoria (Sait y Youssef, 1999) y uno de los que presenta resultados más prometedores en los campos donde ha sido usado (Aarts y Lenstra, 2003).

En la teoría sobre tratamiento térmico de metales, el recocido consiste en someter un metal a alta temperatura por un periodo largo, para luego enfriarlo lenta y uniformemente (Callister, 2005), con el fin de producir una estructura con bajo nivel de esfuerzos en la cual los átomos adopten una configuración con mínima energía remanente (Sverdlin y Ness, 1997). Desde la óptica de la optimización, el estado final del metal constituye la solución óptima al objetivo de minimizar la energía remanente. De la última observación surge la propuesta de Kirkpatrick, Gelatt y Vecchi (1983), que se fundamenta en la analogía entre el recocido y la búsqueda de la solución óptima en un problema de optimización.

En la idea básica se parte de una solución inicial y se selecciona al azar una solución vecina. Si la solución vecina es mejor, se adopta como la nueva solución. Si no lo es, se acepta como la nueva solución con una probabilidad que decrece a medida que el algoritmo avanza. La idea de aceptar con cierta probabilidad soluciones de menor calidad le permite al algoritmo salir de óptimos locales, como se ilustra en la figura 1.

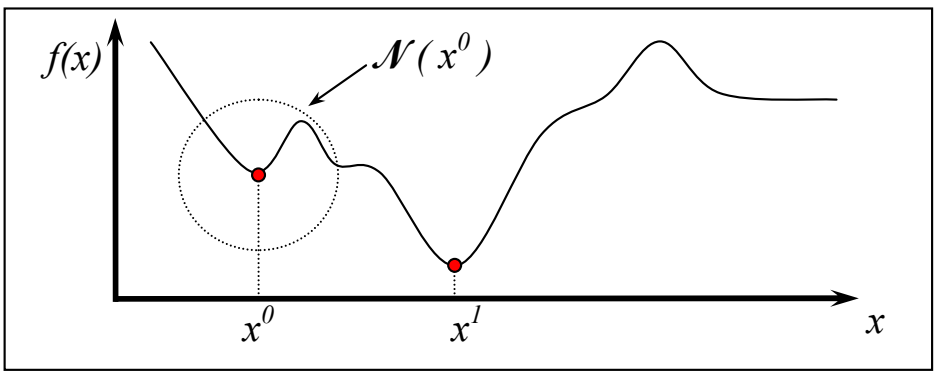

Figura 1. Óptimo local y óptimo global

La figura 1 ilustra un problema en el que se desea encontrar el valor de *x* que minimiza la función *f(x)*, donde N(*x*) representa el vecindario de *x*. Si durante la búsqueda, el método de optimización llega a x^0 y el método no acepta soluciones de inferior calidad en N(x^0), el método queda atrapado en el óptimo local x^0. Estas situaciones son las que metaheurísticos como el recocido simulado tratan de evitar al aceptar, eventualmente y por diversos mecanismos, soluciones de menor calidad, para mejorar la probabilidad de llegar a x^1, el óptimo global. Se encuentran definiciones rigurosas de óptimo local, óptimo global y vecindario en Aarts y Lenstra (2003) y en Sait y Youssef (1999).

El último aspecto importante es el esquema de enfriamiento. Siguiendo con la analogía entre el proceso físico de recocido y el metaheurístico, la probabilidad con la cual se aceptan soluciones de menor calidad para permitirle al metaheurístico escapar de posibles óptimos locales es función de la temperatura. Así como en el recocido, donde a medida que la temperatura desciende es más difícil que un átomo se mueva a una posición de menor energía, en el recocido simulado la probabilidad de aceptar soluciones de menor calidad desciende a medida que el algoritmo avanza. El algoritmo es el siguiente:

Sean T_i , $T_f \in (0,1)$ las temperaturas inicial y final respectivamente

Sea T la temperatura

Sea n el número de iteraciones por nivel

Sea $\varepsilon \in (0,1)$ la tasa de enfriamiento

Hacer $\mathrm{h}_0 \leftarrow f(x_0)$, $\mathrm{h}_{\text{optimo}} \leftarrow h_0$, $\mathrm{x}_{\text{optimo}} \leftarrow x_0$
Hacer $T \leftarrow T_i$
Mientras $T \geq T_f$
  Hacer $k \leftarrow 1$
  Mientras $k \leq n$
    Seleccionar al azar $x_v \in N(x_{\text{optimo}})$
    Si $f(x_v)$ es una mejor solución que $f(x_{optimo})$
      Hacer $\mathrm{x}_{\text{optimo}} \leftarrow x_v$
    Sino
      Generar un número aleatorio r uniforme en el intervalo (0,1).
      Si $r \leq T$
        Hacer $\mathrm{x}_{\text{optimo}} \leftarrow x_v$
      Fin
    Fin
    Hacer $k \leftarrow k+1$
  Fin
  Hacer $T \leftarrow \varepsilon \times T$
Fin

4. ALGORITMOS GENÉTICOS

Los algoritmos genéticos presentan una propuesta eficaz motivada en la observación de que la evolución natural ha sido extraordinariamente exitosa en desarrollar especies complejas y bien adaptadas por medio de un mecanismo simple (Metaxiotis y Psarras, 2004). La técnica emula la evolución natural para explorar con eficiencia el espacio de búsqueda con el supuesto de que unos individuos con ciertas características son aptos para sobrevivir y transmiten estas características a su descendencia (Sait y Youssef, 1999).

Los algoritmos genéticos operan sobre una población o conjunto de soluciones representadas como cadenas binarias o cromosomas. Durante la ejecución, el algoritmo cruza los individuos de mayor aptitud para renovar la población y elimina los de menor aptitud. Al final, el cromosoma de mayor aptitud es la solución al problema (Metaxiotis y Psarras, 2004). Algunos conceptos básicos son (Davis, 1991):

- *Cromosoma*. Cadena binaria que representa un individuo o solución, donde cada elemento en la cadena se conoce como gen (figura 2a).

- *Población*. Conjunto finito de cromosomas (figura 2b).
- *Aptitud*. Criterio que evalúa la calidad de un cromosoma. A mayor aptitud, mejor la solución y mayor la probabilidad de que sobreviva y transmita sus características a su descendencia.

Las operaciones más usadas en este proceso de búsqueda son (Davis, 1991):

- *Cruce*. Operación por medio de la cual se producen nuevos descendientes a partir de dos cromosomas padre seleccionados al azar (figura 3a).
- *Mutación*. En esta operación se seleccionan al azar y se cambian uno o más genes en el cromosoma; ocurre con probabilidades muy bajas (figura 3b).

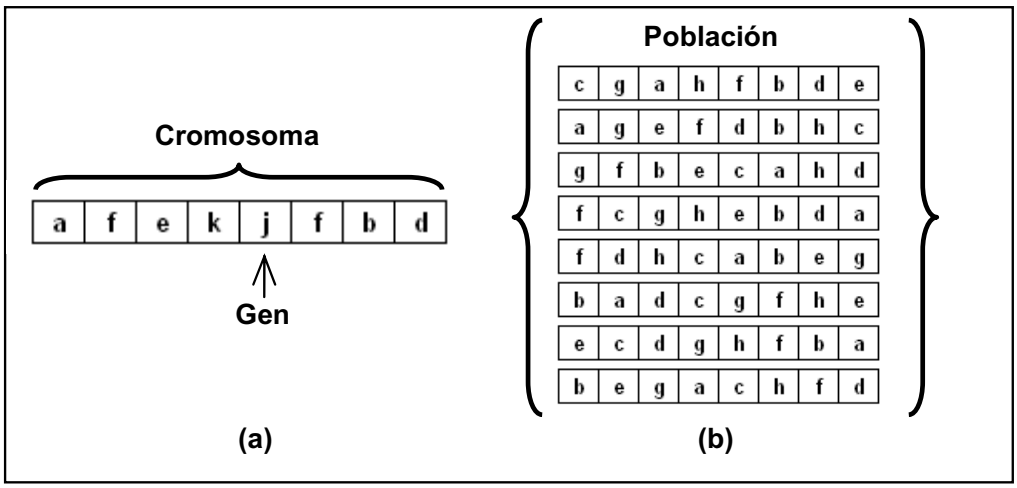

Figura 2. Representación de la población en un algoritmo genético

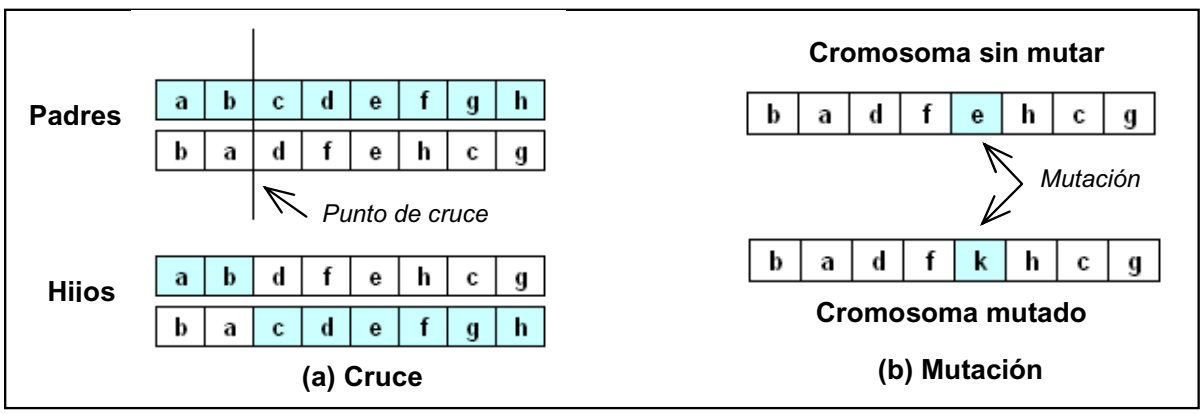

Figura 3. Operaciones básicas en algoritmos genéticos

La idea básica de un algoritmo genético, tomada de Sait y Youssef (1999), es:

Sea $P = \{x^1, x^2, ...\} \subseteq \Omega$ la población

Mientras no se cumpla el criterio de parada:

- Seleccionar como padres $\{x^i, x^j\} \in P$ con probabilidad proporcional a $a(x^i)$ y $a(x^j)$
- Cruzar los padres $\{x^i, x^j\}$ para generar el conjunto O de k hijos $O = \{x_1^{i,j}, x_2^{i,j}, ..., x_k^{i,j}\}$
  - Para cada hijo generado:
    - Generar mutación con probabilidad P_m
    - Hacer que este hijo ingrese a P con probabilidad proporcional a su aptitud
    - Si el hijo ingresa a P, entonces
      - Seleccionar al azar, con probabilidad inversamente proporcional a la aptitud, un elemento que salga de P para ser reemplazado por el nuevo elemento.
    - Fin
  - Fin

Fin

Luego de la aparición de los algoritmos genéticos en trabajos como el de Bremerman (Bremerman, Rogson y Salaff, 1966) y el de Holland (1975) han surgido iniciativas que continúan explorando la idea de imitar el proceso de evolución natural. Estas propuestas reciben el nombre de algoritmos evolutivos y constituyen un campo de investigación más amplio. Unas explicaciones más detalladas sobre el tema, así como posibles implementaciones de los algoritmos genéticos, se pueden apreciar en Aarts y Lenstra (2003) y en Davis (1991). Por su parte, en Chaudhry y Luo (2005) y en Aytug, Khouja y Vergara (2003) pueden encontrarse recopilaciones sobre el uso de algoritmos genéticos en problemas de administración de operaciones.

5. BÚSQUEDA TABÚ

La búsqueda tabú, introducida por Glover (1986), es el más reciente de los tres metaheurísticos descritos en este trabajo, y su contribución más importante consiste en que mientras los otros metaheurísticos guardan información sobre la mejor solución encontrada, la búsqueda tabú mantiene información almacenada sobre las últimas soluciones visitadas con el fin de usarla para guiar la búsqueda y evitar que el algoritmo se mueva a soluciones visitadas recientemente, por lo que se dice que la búsqueda tabú tiene memoria (Aarts y Lenstra, 2003). En este sentido se dice que hay cierto aprendizaje y que la búsqueda es inteligente (Martí, 2003).

En cada iteración, el algoritmo explora el vecindario de la mejor solución encontrada. Se adopta como mejor solución la mejor solución en el vecindario, aun si ésta no es mejor que la solución actual. Para que el algoritmo no se quede atrapado en un ciclo, se almacena la información relativa a las soluciones recientemente visitadas en una lista llamada *lista tabú* (Sait y Youssef, 1999).

La lista tabú no es una enumeración de soluciones, ya que almacenar soluciones completas, aun en pequeñas cantidades, y comparar cada solución con las que hay en la lista es muy costoso computacionalmente. Para acopiar información sobre el pasado reciente, la búsqueda tabú

almacena los últimos movimientos, donde un *movimiento* es una operación por medio de la cual se alcanza una solución vecina a partir de la solución actual. Por ejemplo, si una solución se puede representar con un vector, como se ilustra en la figura 4, un movimiento puede consistir en intercambiar la posición de dos elementos en el vector. Nótese que para almacenar el movimiento sólo es necesario almacenar dos valores (Ahuja, Magnati y Orlin, 1993; Aytug, Khouja y Vergara, 2003) y que esta información por sí sola no representa una solución (figura 4a).

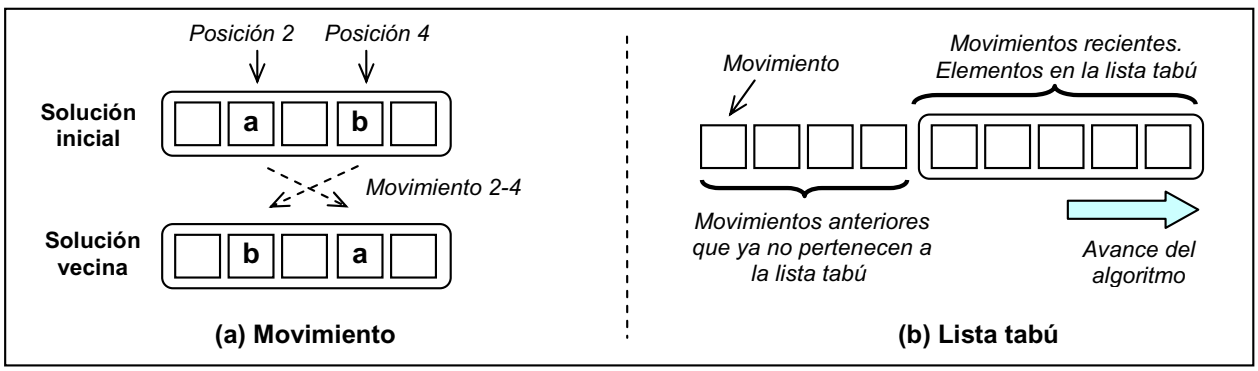

Figura 4. Estructuras básicas en la búsqueda tabú

La lista tabú es una lista en la cual se almacenan los movimientos recientes por un lapso denominado *tiempo de permanencia*, como se muestra en la figura 4b. El algoritmo previene los ciclos al prohibir que se ejecute un movimiento en la lista; aunque esta prohibición puede anularse eventualmente si se cumple una condición, denominada *criterio de aspiración*, que suele consistir en admitir un movimiento en la lista tabú si la solución resultante es la mejor hasta el momento. Otros criterios de aspiración pueden estudiarse en Sait y Youssef (1999).

Además de la lista tabú, que determina la memoria de corto plazo, se consideran otros dos tipos de memoria conocidas como memorias de mediano y largo plazo. El mecanismo de memoria de mediano plazo consiste en favorecer la búsqueda en regiones donde se han encontrado buenas soluciones. Este proceso se conoce como de intensificación y se lleva a cabo dándoles alta prioridad a soluciones similares a la actual y penalizando las que se alejen de ella. Se dice que la intensificación es un mecanismo de memoria de mediano plazo porque normalmente el proceso se da por terminado si durante un número fijo de iteraciones no se logra mejorar la función objetivo. Una vez terminado el proceso de intensificación puede resultar conveniente explorar regiones nuevas en el espacio de búsqueda. El mecanismo de memoria de largo plazo cumple con este objetivo de diversificación por favorecer soluciones con características diferentes a la solución actual. Ambos procesos, la intensificación y la diversificación, se incorporan en la función objetivo con pesos o ponderaciones que se modifican durante el desarrollo del algoritmo, de manera que ambas fases se alternen durante la búsqueda (Aarts y Lenstra, 2003). La idea básica tomada de (Sait y Youssef, 1999) es la siguiente:

Sea $x^* \in \Omega$ la mejor solución encontrada
Sea N_A el nivel de aspiración

Hacer $x^* \leftarrow x_0$
Mientras no se cumpla el criterio de parada:
  Seleccionar al azar el conjunto $V \subseteq N(x^*)$ de vecinos de x^*
  Sea $x' \in V$ la mejor solución en V
  Si el movimiento que generó x' no está en la lista Tabú
    Aceptar el movimiento haciendo $x^* \leftarrow x'$
    Actualizar la lista Tabú incorporando a la lista el movimiento que generó x'
    Actualizar N_A
  Sino
    Si se cumple el criterio de aspiración
      Aceptar el movimiento haciendo $x^* \leftarrow x'$
      Actualizar la lista Tabú incorporando a la lista el movimiento que generó x'
    Fin
  Fin
Fin

6. SECUENCIAMIENTO DE OPERACIONES

Esta sección describe un problema de optimización combinatoria en administración de operaciones: el secuenciamiento de tareas en un sistema de producción *flow-shop* (FSSP[3]) con el fin de usarlo para ilustrar algunos de los metaheurísticos de más amplio desarrollo.

Un sistema de producción de flujo en línea o *flow-shop* consiste en un grupo de máquinas en serie en las cuales es necesario procesar un conjunto de trabajos, donde las operaciones que componen cada trabajo se ejecutan en el mismo orden (Pinedo, 2001). En este contexto, el FSSP está definido por $[p_{ij}]_{nxm}$, que representa el tiempo de procesamiento correspondiente al trabajo *i* en la máquina *j*. Se supone que las máquinas sólo pueden procesar un trabajo a la vez y que ninguna operación puede interrumpirse una vez iniciada. Pinedo (2001) y Askin y Standridge (1993) ofrecen una revisión detallada del FSSP.

Debido a que el FSSP es un problema difícil si el número de máquinas es mayor o igual a tres (Garey, Johnson y Sethi, 1976), ha recibido gran atención desde la década de los cincuenta cuando Johnson (1954) publicó el primer estudio sobre el tema. En la mayoría de los trabajos publicados se propone como objetivo minimizar el tiempo total necesario para la terminación de los trabajos o *makespan* (Ruiz, Maroto y Alcaraz, 2005), aunque es tan amplia la literatura que existe sobre este problema que es posible encontrar textos para prácticamente todas las medidas de desempeño conocidas. Pinedo (2001) y T'kindt y Billaut (2005) ofrecen una revisión detallada de problemas de secuenciamiento.

[3] Del inglés *Flow-Shop Scheduling Problem*.

7. BÚSQUEDA DE SECUENCIAS DE PERMUTACIÓN EN EL FSSP

La complejidad del FSSP es la razón por la cual gran parte de la investigación se ha concentrado en la búsqueda de *secuencias de permutación* o secuencias en las que todos los trabajos se ejecutan en el mismo orden en cada máquina (Hejazi y Saghafian, 2005). Esta restricción adicional impuesta al problema permite reducir de forma sustancial el espacio de búsqueda. A continuación se presenta un ejemplo que permitirá demostrar como los metaheuristicos descritos pueden encontrar la secuencia de permutación de menor tiempo de fabricación en el FSSP. Para el ejemplo se resolverá un problema en el que deben procesarse 10 trabajos en un sistema *flow-shop* con 10 máquinas. Los tiempos de procesamiento del trabajo *i* en la máquina *j* están dados en la tabla 1.

Tabla 1. Tiempos de procesamiento

Trabajo	Máquina									
	1	2	3	4	5	6	7	8	9	10
1	1	3	4	5	4	9	5	2	1	7
2	1	6	1	2	10	5	5	10	10	2
3	10	6	9	8	2	2	5	5	7	10
4	2	7	9	9	10	10	6	1	2	5
5	6	8	7	2	2	6	7	6	2	8
6	4	10	5	6	7	9	2	6	2	3
7	10	10	7	7	1	4	6	9	10	1
8	8	1	8	2	6	10	4	3	6	5
9	6	9	5	4	9	8	8	2	1	3
10	7	3	8	7	1	1	1	7	7	6

Por último, para describir las aplicaciones de cada metaheuristico en la solución del ejemplo anterior son necesarias algunas definiciones.

Representación. El primer paso en el diseño de un metaheurístico es determinar la representación de las soluciones. Para la búsqueda de secuencias de permutación en el FSSP, una posible representación consiste en definir un vector de *n* posiciones, donde la i-ésima posición representa el índice del trabajo que se va a procesar en el i-ésimo turno de la secuencia. Así, en un problema de cuatro trabajos, una posible secuencia sería [2, 1, 4, 3], donde el 4 en la tercera posición indica que el trabajo 4 es el tercero en ser procesado.

Vecindario. Sea $N(S) \subseteq \Omega$ el vecindario de la secuencia *S*. Para el FSSP se va a usar un *vecindario de intercambios simples* (Aarts y Lenstra, 2003). Por facilidad sólo se consideran dos tipos de vecindarios de intercambio simple: el vecindario de traslado y el de intercambio. En Aarts y Lenstra (2003) se pueden consultar vecindarios más complejos. La figura 5a ilustra una operación de traslado, y la figura 5b, un intercambio.

Los algoritmos descritos en esta sección fueron programados en Matlab 7.0 y ejecutados en una máquina con procesador Intel Core Duo 1.86 GHz de 2 GB de memoria RAM.

Algoritmo de recocido simulado para el FSSP. El algoritmo que se describe en seguida es una versión simplificada del presentado por Pinedo (2001).

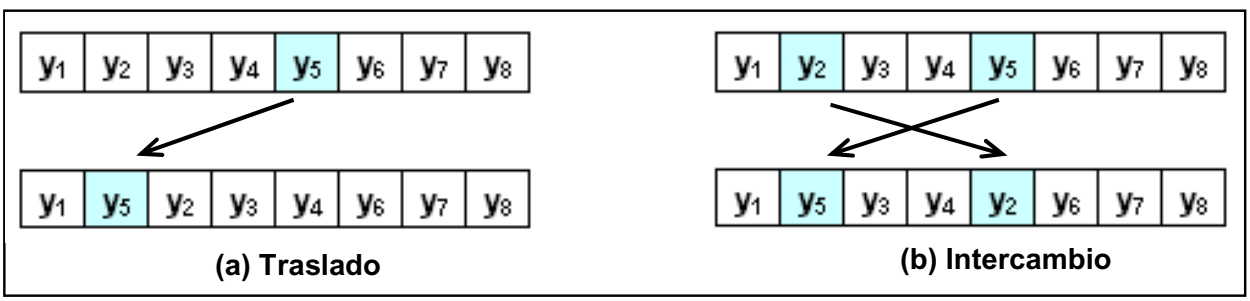

Figura 5. Vecindarios de traslado y de intercambio

Hacer $T_i = 1$, $T_f = 0.01$, $n = 200$, $\varepsilon = 0.95$
Generar x_0, una solución (permutación) inicial generada al azar
Hacer $h_0 = f(x_0)$
Hacer $T = T_0$
Mientras $T > T_f$
  $k = 1$
  Mientras $k \leq n$
    Generar $q \sim U(0,1)$
    Si $q \leq 0.5$
      Generar x ,una solución en el vecindario de intercambio de x_0
    Si no
      Generar x, una solución en el vecindario de traslado de x_0
    Fin
    Calcular $h = f(x)$
    Si $h < h_0$
      Hacer $x_0 \leftarrow x$
    Si no
      Generar $r \sim U(0,1)$
      Si $r < T$
        Hacer $x_0 \leftarrow x$
      Fin
    Fin
    $k = k + 1$
  Fin
  $T = \varepsilon \times T$
Fin
Imprimir h_0 y $f(h_0)$, la mejor solución encontrada y el makespan correspondiente

Al implementar este algoritmo se obtuvo un makespan de 117 unidades de tiempo correspondiente a la secuencia S = [8, 5, 4, 2, 9, 7, 10, 1, 6, 3].

Algoritmo genético para el FSSP. Este algoritmo es una versión simplificada del propuesto por Colin R. Reeves (1995). Antes de describirlo se presentan algunos elementos necesarios para su mejor comprensión.

- *Probabilidad de selección*. La probabilidad con la que se seleccionan los padres es proporcional

a su calidad. Para el FSSP esta probabilidad crece a medida que el *makespan* es menor, de manera que si se ordenan las M secuencias de la población de forma descendente, de acuerdo con el *makespan*, la probabilidad de seleccionar la k-ésima secuencia está dada por $p(k) = 2k[M(M+1)]^{-1}$.

- *Cruce*. El cruce de los padres P_1 y P_2 para crear los hijos H_1 y H_2 se lleva a cabo de la siguiente forma: se seleccionan al azar dos posiciones en la secuencia para formar un segmento de cruce y se intercambian estos segmentos entre los padres, como se ilustra en la figura 6a. Después del intercambio, si las nuevas secuencias contiene valores repetidos, como en el ejemplo de la figura 6b, se eliminan los valores repetidos que se encuentran por fuera del segmento de cruce y en su lugar se asignan los valores así: los valores eliminados en H_1 se reemplazan con los valores correspondientes al segmento de cruce de P_1 y los valores eliminados en H_2 se reemplazan con los valores correspondientes al segmento de cruce de P_2, en el mismo orden (figura 6c).
- *Mutación*. La mutación de una secuencia consiste en cambiarla por una secuencia vecina, utilizando el vecindario de intercambio descrito.

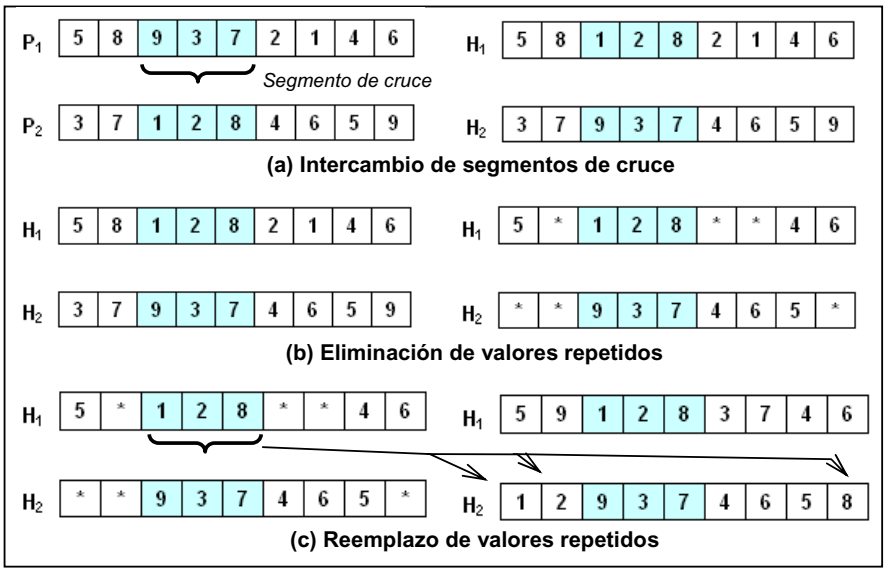

Figura 6. Operación de cruce

El algoritmo es el siguiente:

Inicialización

Sea $\beta = 0.5$ la probabilidad inicial de mutación, y sea $\theta = 0.1$ la tasa de decrecimiento de β

Sea $P = \{\phi\}$, $M = 600$ y $Q(0) = 0$

Para $k = 1$ hasta M

Generar al azar P_k, el k-ésimo individuo de la población

Hacer $P \leftarrow P \cup \{P_k\}$

Calcular $q(k) = 2k[M(M+1)]^{-1}$, la función de masa de probabilidad para la selección de padres

Calcular $Q(k) = Q(k-1) + q(k)$, la función de distribución acumulada correspondiente a $q(k)$

Fin

Algoritmo genético

Para $w = 1$ hasta 100

Ordenar los individuos en P de manera que $f(P_1) \geq f(P_2) \geq \cdots \geq f(P_M)$

Generar $r \sim U(0,1)$ y hacer $k = 1$

Mientras $Q(k) \leq r$

$k = k + 1$

Fin

$Padre_1 = P_k$, $Padre_2 = Padre_1$

Mientras $Padre_2 = Padre_1$

Generar $r \sim U(0,1)$ y hacer $k = 1$

Mientras $Q(k) \leq r$

$k = k + 1$

Fin

$Padre_2 = P_k$

Fin

Cruzar $Padre_1$ y $Padre_2$ para crear dos hijos: $Hijo_1$ e $Hijo_2$

Para $i = 1$ hasta 2

Generar $r \sim U(0,1)$

Si $r \leq \beta$

Mutar $Hijo_i$

Fin

Generar $r \sim U(0,1)$

Si $r \leq f(P_M)/f(Hijo_i)$

Hacer $P \leftarrow P \cup \{Hijo_i\}$

Generar $r \sim U(0,1)$ y hacer $k = 1$

Mientras $Q(k) \geq r$

$k = k + 1$

Fin

$P \leftarrow P \setminus \{P_k\}$

Fin

$\beta = \theta \times \beta$

Fin

Imprimir P_M y $f(P_M)$: la mejor solución encontrada y el makespan correspondiente

Como resultado de la aplicación de este algoritmo se obtuvo un makespan de 121 unidades de tiempo correspondiente a la secuencia S = [1, 5, 8, 2, 10, 4, 3, 7, 6, 9].

Búsqueda Tabú para el FSSP. El algoritmo está basado en el propuesto por Pinedo (2001).

Sea $N_A = f(x_0)$ el nivel de aspiración
Hacer $k = 1$, $M = 10$, $Perm = 10$
Hacer $TabuList(k) = \{0,0\}$
Para $i = 1$ hasta 5000
  Para $k = 1$ hasta M
    Generar x_k ,una solución vecina de x_0, intercambiando las posiciones m_1^k y m_2^k
  Fin
  Sea $t = Arg\min\{f(x_k)\}$
  Si la pareja m_1^t, m_2^t es un elemento de $TabuList$
    Si $f(x_t) < N_A$
      Hacer $x_0 \leftarrow x_t$ y $N_A = f(x_t)$
    Fin
  Si no
    Hacer $x_0 \leftarrow x_t$ y $N_A = f(x_t)$
    Hacer $TabuList(k) = \{m_1^t, m_2^t\}$
    $k = k + 1$
    Si $k = Perm$
      $k = 1$
    Fin
  Fin
Fin
Imprimir x_0 y $f(x_0)$: la mejor solución encontrada y el makespan correspondiente

Después de su implementación el algoritmo arrojó un makespan de 114 unidades de tiempo correspondiente a la secuencia S = [1, 2, 8, 4, 10, 5, 3, 7, 6, 9].

8. CONCLUSIONES

- Los problemas reales que enfrentan a diario los profesionales encargados de las áreas de administración de operaciones y logística son por lo general de una gran complejidad, ante todo debido a que la cantidad de soluciones posibles o el espacio de búsqueda es muy grande, y explorarlo exhaustivamente para encontrar la solución óptima suele ser imposible con la tecnología actual. Este hecho ha impulsado el desarrollo de técnicas no convencionales de optimización que, como los metaheurísticos, permiten encontrar soluciones

aceptables en un tiempo de cómputo razonable con la incorporación de ideas innovadoras, de ordinario copiadas de la naturaleza.

- Los metaheurísticos, más que algoritmos rígidos, constituyen ideas generales que permiten un margen de maniobra muy amplio a la hora de ser programados. Es esta gran versatilidad la que los hace muy atractivos, ya que es posible adaptarlos a casi cualquier problema de optimización combinatoria.
- Los metaheurísticos combinan de forma diferente dos conceptos clave para el desarrollo de un buen algoritmo de búsqueda: la intensificación y la diversificación. La intensificación consiste en explorar a fondo una región del espacio de búsqueda donde se han encontrado buenas soluciones. Esta intensificación se lleva a cabo, por lo general, perturbando la mejor solución encontrada hasta el momento para construir soluciones vecinas o cercanas en el espacio de búsqueda. La diversificación, por el contrario, consiste en probar en zonas inexploradas del espacio de búsqueda para evitar que el algoritmo quede atrapado en óptimos locales. La idea de aceptar eventualmente soluciones de inferior calidad en el recocido simulado o la idea de las mutaciones en los algoritmos genéticos son ejemplos de diversificación. Del equilibrio entre estas dos estrategias depende en buena medida la calidad del algoritmo final.
- Además de definir la representación de las soluciones, en cada metaheurístico es necesario tomar un número importante de decisiones que determinan su desempeño. El esquema de enfriamiento en el recocido simulado, el tamaño de la población y la forma como se efectúan los cruces y las mutaciones en los algoritmos genéticos, la representación de los movimientos o el criterio de aspiración en la búsqueda tabú son algunos ejemplos. Es tarea del programador seleccionar con cuidado estos criterios, ya que de su selección depende la calidad de las soluciones que se encuentren.
- De los resultados obtenidos al programar los metaheurísticos expuestos para resolver un ejemplo del FSSP no debe concluirse, en ninguna circunstancia, que uno de los metaheurísticos sea mejor que otro. Dada la naturaleza estocástica de estos algoritmos, es necesario un análisis estadístico detallado para concluir al respecto.

BIBLIOGRAFÍA

Aarts, E. and Lenstra, J. K. (2003). Local search in combinatorial optimization. Eindhoven y Atlanta: Princeton University Press, p. 1-137.

Ahuja, R. K.; Magnati, T. L. and Orlin, J. B. (1993). Network flows. Englewood Cliffs, NJ: Prentice Hall.

Askin, R. G. and Standridge, C. R. (1993). Modeling and analysis of manufacturing systems. New York: John Wiley and Sons.

Aytug, H.; Khouja, M. and Vergara, F. E. (2003). Use of genetic algorithms to solve production and operations management problems: A review. International Journal of Production Research, vol. 41, No. 17, p. 3955-4009.

Bertsimas, D. and Tsitsiklis, J. N. (1997). Introduction to linear optimization. Belmont, Massachusetts: Athenea Scientific, p. 359-392.

Blum, C. and Roli A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, vol. 35, No. 3, p. 268-308.

Bremerman, H. J.; Rogson, M. and Salaff, S. (1966). Evolution and optimization, natural automata and useful simulations. Washington, D. C: Spartan Books, p. 3-41.

Callister, W. (2005). Fundamentals of materials science and engineering: An integrated approach, 2 ed. New York: John Wiley and Sons.

Chaudhry, S. S. and Luo, W. (2005). Application of genetic algorithms in production and operations management: A review. International Journal of Production Research, vol. 43, No. 19, p. 4083-4101.

Davis, L. (1991). Handbook of genetic algorithms. New York: Van Nostrand Reinhold.

Garey, M.; Johnson, D. S. and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research, vol. 1, No. 2, p. 117-129.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, vol. 13, No. 5, p. 533-349.

Glover, F. and Kochenberger, G. A. (2003). Handbook of metaheuristics. Berlin: Springer.

Grötschel, M. (1991). Discrete mathematics in manufacturing. Proceedings of the Second International Conference on Industrial and Applied Mathematics. Washington: Society for Industrial and Applied Mathematics, p.119-145.

Grötschel, M. and Lovasz, L. (1995). Handbook of combinatorics, vol. II, cap. 28. Cambridge, Massachusetts: The MIT Press y North-Holland, p. 1541-1598.

Hejazi, S. R. and Saghafian S. (2005). Flowshop scheduling problems with makespan criterion: A review. International Journal of Production Research, vol. 43, No. 14/15, p. 2895-2929.

Holland, J. H. (1975). Adaptation in natural and artificial systems. Ann Arbor: The University of Michigan Press.

Hoffman, K. L. (2000). Combinatorial optimization: Current successes and directions for the future. Journal of Computational and Applied Mathematics, vol. 124, No. 1-2, p. 341-360.

Hoffman, K. and Padberg M. (2000). Traveling salesman problem. [on line] http://iris.gmu.edu/~khoffman/papers/trav_salesman.html (Diciembre de 2006).

Johnson, S. M. (1954). Optimal two and three-stage production schedules with setup times included. Naval Research Logistics Quarterly, vol. 1, No 61, p. 61-68.

Kirkpatrick, S.; Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing. Science, vol. 220, No. 4598, p. 671-680.

Kolen, A. W. J. and Lenstra K. (1995). Handbook of combinatorics, vol. II, cap. 35. Amsterdam: The MIT Press y North-Holland, p. 1875-1910.

Martí, R. (2003). Procedimientos metaheurísticos en optimización combinatoria. Matematiques, vol. 1, No 1, p. 3-62.

Melián, B.; Moreno, J. A. and Vega, J. M. (2003). Metaheuristics: A global view. Revista Iberoamericana de Inteligencia Artificial, vol. 2, No. 19, p. 7-28.

Metaxiotis, K. and Psarras J. (2004). The contribution of neural networks and genetic algorithms to business decision support: Academic myth or practical solution? Management Decision, vol. 42, No. 1/2, p. 229-242.

Osman, I. H. and Kelly J. P. (eds.). (1996). Meta-Heuristics: theory and applications. Boston: Kluwer Academic, p. 1-21.

Pinedo, M. (2001). Scheduling theory, algorithms and systems, 2 ed. Englewood Cliffs, NJ: Prentice Hall.

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. Computers and Operations Research, vol. 22, No. 1, p. 5-13.

Ruiz, R.; Maroto, C. and Alcaraz, J. (2005). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics, European Journal of Operational Research, vol. 165, No. 1, p. 34-54.

Sait, S. M. and Youssef H. (1999). Iterative computer algorithms with applications in engineering: Solving combinatorial optimization problems. Piscataway: IEEE Computer Society Press, p. 1-248.

Sverdlin, A. V. and Ness A. R. (1997). Fundamental concepts in steel heat treatment. Steel Heat Treatment Handbook. New York: Marcel Dekker, p. 1-44.

T'kindt, V. and Billaut J. C. (2005). Multicriteria scheduling: Theory, models and algorithms, 2 ed. Berlin: Springer.

Tovey, C. A. (2002). Tutorial on computational complexity. Interfaces, vol. 32, No. 3, p. 30-61.

Voß, S. and Woodruff, D. L. (2006). Introduction to computational optimization models for production planning in a supply chain, 2 ed. Berlin: Springer.