Algoritmo de recocido simulado aplicado al problema de secuenciamiento regular

Simulated annealing algorithm applied to a flow-shop problem

Jhon Jairo Santa Chávez[1], César Augusto Peñuela Meneses[2], Mauricio Granada Echeverry[3]

[1] (c), M.Sc. en Ingenierías, M.Sc. en Instrumentación Física, Universidad Tecnológica de Pereira, jjsanta@utp.edu.co
[2] Ph. D., Universidad Tecnológica de Pereira, capenuela@unilibrepereira.edu.co
[3] Ph. D., Universidad Tecnológica de Pereira, magra@utp.edu.co

Fecha de recepción: 12/12/2013 Fecha de aceptación del artículo: 25/04/2014

Resumen

En este trabajo se realizó una descripción detallada del algoritmo de Recocido Simulado como metodología de solución a problemas de optimización combinatorial. Este se enfocó especialmente a la solución del problema de secuenciamiento de tareas dentro de una planta de producción con recursos limitados. Se presentan resultados de simulaciones realizadas sobre un problema de la literatura especializada, mostrando la amplia versatilidad, eficiencia, y facilidad de implementación del algoritmo propuesto.

Palabras clave

Secuenciamiento de tareas, Optimización combinatorial, Recocido simulado.

Abstract

In this paper a detailed description of the simulated annealing algorithm is presented. This strategy was employed to solve a well know flow-shop problem, which behavior is featured by the combinatorial explosion and non linear modeling. Therefore, the strategy could be focused to analyze the efficiency of the production in facilities with limited resources. Simulations results performed in a problem in the literature are presented. By this mean, the wide versatility, efficiency, and the easy implementation of the proposed algorithm is verified.

Keywords

Combinatorial optimization, Flow-shop problem, Simulated annealing.

1. Introducción

La permanencia de una empresa dentro de mercados competitivos exige una planificación eficiente de las actividades productivas, de forma que se aproveche al máximo los recursos disponibles y se reduzcan los costos a niveles apropiados sin afectar la calidad de los bienes manufacturados. Desde este punto de vista, la eficiencia en el proceso productivo puede ser definida como el porcentaje de tiempo en que los recursos son usados para la generación del producto final. En este aspecto se reconoce la existencia de actividades que, si bien son necesarias para el funcionamiento de la planta, generan periodos en los que la materia prima no es procesada, como, por ejemplo, el arranque de maquinaria, reconfiguración de maquinaria por cambio de productos, mantenimiento, tiempos muertos e interrupciones por roturas. Por otro lado, existen periodos de tiempo en los cuales parte de la maquinaria (y operarios) permanece inactiva a la espera del finalizado de un proceso previo, lo cual redunda en el aumento de costos por mano de obra inactiva y retrasos en los tiempos de entrega. Así, es evidente la necesidad de generar un programa en el que todas las actividades necesarias para la

generación del producto final sean ejecutadas en el menor tiempo posible. De manera más formal, el problema secuenciamiento de tareas consiste en la generación de un programa de eventos para la ejecución de n tareas las cuales deben ser procesadas en un conjunto de m máquinas ubicadas en línea.

Debido a la importancia dentro el planeamiento operativo de las empresas, este problema se ha convertido en un área de estudio clásico en la literatura, y se ha tratado de resolver a partir de diversas metodologías de solución, entre las cuales se destacan aquellas que implementan metaheurísticas, debido a su eficiencia, facilidad de programación y calidad de resultados [1].

En este trabajo, se propuso la implementación de un algoritmo denominado como Recocido Simulado (RS), el cual fue propuesto en 1982 por tres investigadores de IBM Society [2], y cuya principal característica es que puede escapar de óptimos locales. Un trabajo similar, desarrollado en forma contemporánea e independiente, fue publicado por [3]. El algoritmo de RS puede interpretarse como un proceso iterativo similar al algoritmo de Metrópolis [4], en el cual se evalúa y se decrementa de forma controlada un parámetro de control conocido como Temperatura. El algoritmo de Metrópolis simula el proceso físico basado en la temperatura a la cual los sólidos obtienen estados de baja energía o de equilibrio térmico, conocido como recocido de sólidos. Durante este proceso, la energía libre del sólido es minimizada.

En la optimización combinatoria se desarrolla un proceso similar al de recocido de sólidos. Este proceso puede ser formulado como una metodología que encuentra, entre muchas soluciones, aquella que tenga mínimo costo. Así, se establecen las siguientes correspondencias:

a) Una alternativa de solución en un Problema de Optimización Combinatoria (POC) es equivalente al estado energético de un sistema real.
b) Una alternativa vecina de un POC corresponde a una pequeña distorsión al estado energético actual del sistema real.
c) El valor de la función objetivo en un POC es equivalente a la energía del estado actual del sistema.
d) Encontrar una buena solución en un POC es equivalente a encontrar estados de baja energía en el sistema real.

Desde el punto de vista práctico, la técnica de RS generalmente consigue soluciones de buena calidad. Se considera un método general que puede ser aplicado a muchos tipos de problemas y es de fácil implementación. Los problemas abordados con esta técnica que mejores resultados presentan son aquellos cuya codificación permite una evaluación rápida y directa de la función objetivo después de cada perturbación. Esto evita alcanzar tiempos computacionales prohibitivos.

2. Metodología

2.1 Planteamiento del problema

El problema consiste en determinar una secuencia de tareas entre las $n!$ secuencias posibles pasando por todas las máquinas (Programación con Permutaciones) procurando optimizar una determinada medida de desempeño. En este trabajo se midió la duración total de programación. Esto se refiere al intervalo de tiempo transcurrido desde la ejecución de la primera tarea en la primera máquina hasta la ejecución de la última tarea en la última máquina. De igual manera a como se define en [1], las consideraciones usuales de un problema de secuenciamiento de tareas son:

a) Cada máquina está disponible continuamente y sin interrupciones.
b) Cada máquina puede procesar una tarea por vez.
c) Cada tarea sólo puede ser procesada por una máquina cada vez.
d) Los tiempos de procesamiento de las tareas en las diferentes máquinas son conocidos y fijos.
e) Las tareas tienen la misma opción de ser programadas.

f) Los tiempos de preparación de las operaciones en las distintas máquinas están incluidos en los tiempos de procesamiento.
g) Las operaciones en las máquinas, una vez iniciadas, no deben ser interrumpidas.

Las diferentes formas de programación de tareas en el conjunto de máquinas disponibles se denominan técnicamente "*secuencias de permutación*". En estas secuencias, el tiempo de procesamiento de la tarea *j* en la máquina *i* se denota por p_{ij}, y el tiempo total de procesamiento o de ejecución se denomina $Cmax$ (*Makespan*).

2.2 Modelado matemático

El objetivo es encontrar la secuencia de los trabajos a programar la minimiza el *Makespan,* que corresponde al tiempo en el cual el último trabajo es finalizado en la máquina m. El problema investigado usualmente es denotado por $n/m/P/Cmax$ y definido de la siguiente forma: Si se tienen los tiempos de procesamiento $p(i,j)$ para el trabajo *i,* en la máquina *j,* y la secuencia de trabajo $[J_1,J_2,...,J_n]$ donde se calcula el tiempo completo de procesamiento o *Makespan* $C(J_t, j)$. Cada tarea debe pasar por todas las máquinas, siendo que todas las tareas tienen el mismo orden de procesamiento. Es decir, la secuencia de tareas en cada máquina es la misma. Por otro lado, cada máquina m_i puede procesar sólo una tarea n_j a la vez.

Por ejemplo, para un problema simple de 4 tareas y 3 máquinas, los tiempos de procesamiento P y la secuencia π tienen la forma mostrada en la Figura 1. Además, los máximos tiempos de procesamiento en cada máquina son definidos en el vector r.

Para encontrar el valor de la función objetivo, dada una secuencia π de tareas, se utiliza una

P =	T_1	T_2	T_3	T_4
M_1	4	3	5	7
M_2	7	7	5	9
M_3	3	3	4	1

Π =				
	4	3	1	2

r =				
	25	40	20	21

Figura 1. Datos de entrada del problema.

matriz auxiliar $Cmxn$ en la cual se almacenan los diferentes tiempos de procesamiento en forma secuencial. En la construcción de esta matriz se siguen los siguientes pasos:

a) Se obtiene el tiempo de procesamiento de la primera tarea en la primera máquina y se almacena en la matriz C.

$$C_{1,1} = P_{1,\pi 1} \qquad (1)$$

b) Usando la ecuación (2) se calcula el tiempo de procesamiento de todas las tareas en la primera máquina de la línea de producción, con $j\epsilon[2,3,..,n]$

$$C_{1,j} = C_{1,j-1} + P_{1,\pi j} \qquad (2)$$

c) Usando la ecuación (3) se calcula el tiempo de procesamiento de la primera tarea en cada una de las máquinas.

$$C_{i,1} = C_{i-1.1} + P_{i,\pi 1} \qquad i\epsilon\{2,3,..,m\} \qquad (3)$$

d) Usando la ecuación (4), con $j\epsilon\{2,3,...,n\}$, e $i\epsilon\{2,3,...,m\}$, y con la información recopilada hasta el momento en la matriz C, se calcula el resto de sus elementos.

$$C_{i,j} = max\{C_{i,j-1}; C_{i-1,j}\} + P_{i,\pi j} \qquad (4)$$

También, es común incorporar restricciones en los tiempos de procesamiento de cada tarea, a fin de cumplir con tiempos de entrega. En este caso se cuenta con la información de tiempos máximos de entrega dados en un vector r de tamaño n. Una forma de evaluar la calidad de una alternativa de solución, teniendo en cuanta estas restricciones, es penalizando la función objetivo.

El problema de Flow-shop que se describe puede ser representado gráficamente como se muestra en la Figura 2, donde se asume una secuencia $\pi = [4, 3, 1, 2]$. Las barras debajo de cada máquina representan, proporcionalmente, los tiempos de procesamiento definidos en *P*.

En la Figura 3 se muestra una representación gráfica de la información contenida en la matriz *C,* en donde es fácil verificar cualquier violación de los tiempos máximos de entrega.

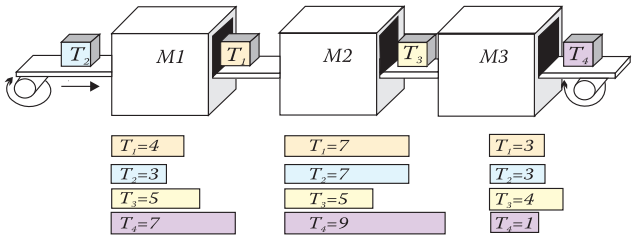


Figura 2. Flujo en línea en un taller con $m = 3$ y $n = 4$.

2.3 Metodología de solución

La idea original que dio lugar a la metaheurística denominada *Recocido Simulado* (RS), se encuentra en el *Algoritmo de Metrópolis*. Este algoritmo se basa en la técnica de Monte Carlo para generar una secuencia de estados del sólido en su proceso de enfriamiento. Si se tiene un estado actual del sólido i con energía E_i, entonces el subsiguiente estado j con energía E_j es generado aplicando un mecanismo de perturbación consistente en provocar una pequeña distorsión en el estado actual. Si la diferencia de energía entre estados, $(E_j\text{-}E_i)$, es menor o igual a cero, el estado j es aceptado como el estado actual. Si la diferencia de energía es mayor a cero, el estado j tiene una cierta probabilidad, Φji, de ser aceptado. Dicho valor es determinado por el factor de Boltzmann, tal como se expresa en la ecuación (5).

$$\Phi ji = e^{-\left(\frac{Ej-Ei}{KB \cdot T}\right)} \quad (5)$$

donde T denota la temperatura, y KB es una constante física conocida como constante de Boltzmann. El algoritmo de RS es concebido para problemas de minimización y, por lo tanto, siempre se considera que existe un mejoramiento del estado actual cuando $E_j\text{-}E_i \leq 0$. Sin embargo, en términos generales, un mejoramiento existe cuando se presenta un decremento en la energía del sistema.

En optimización, un movimiento (paso de una alternativa a otra ubicada en su vecindario) será aceptado si éste mejora la función objetivo; o en caso contrario, si su probabilidad de aceptación es mayor que un número aleatorio uniformemente distribuido. Este tipo de estrategias permiten que el algoritmo escape de óptimos locales. Por otro lado, y con el fin de avanzar en la convergencia a una solución, en la medida en que evoluciona el método de optimización se disminuye la temperatura y se incrementa la longitud de la cadena. Con esto disminuye la probabilidad de aceptar soluciones de mala calidad.

La codificación del problema define el espacio de búsqueda y por consiguiente la estructura de vecindad. Este aspecto juega un papel fundamental en la eficiencia del algoritmo debido al concepto de proximidad entre estados energéticos. En un sistema real un estado energético es una transformación continua del estado energético anterior, y por lo tanto, una pequeña perturbación produce un pequeño cambio energético. En un problema de optimización combinatoria, una inadecuada definición de un vecino podría, eventualmente, producir grandes diferencias con respecto a la función objetivo, lo cual es indeseable para el algoritmo de RS.

Por esta razón, se ha demostrado que el algoritmo RS tiene gran eficiencia en algunos problemas específicos, mientras que en otros es fácilmente superado por una heurística simple. Algunos autores, como en [6]-[8], muestran que RS podría ser

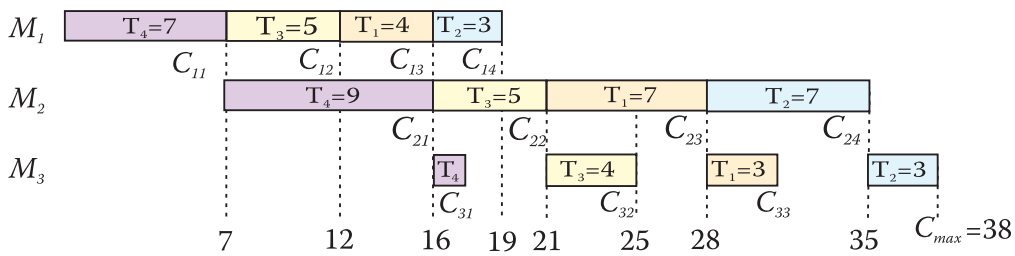


Figura 3: Diagrama de Ganttcon $m = 3, n = 4$ y $\pi = [4, 3, 1, 2]$.

más efectivo en problemas que forman conjuntos ultra métricos, tales como: diseño de capas de micro circuitos electrónicos, problemas con propiedades fractales, entre otros.

Los aspectos fundamentales que se deben definir en el algoritmo de RS, tienen que ver con lo que se denomina el programa de enfriamiento, el cual abarca los siguientes puntos: Temperatura inicial, longitud de la Cadena, ley de enfriamiento, y temperatura final.

2.3.1 Temperatura Inicial (T_0):

Es calculada únicamente al inicio del proceso junto con la longitud de la primera cadena. Su valor depende principalmente del problema tratado y de la probabilidad de aceptación τ_0. En un problema combinatorio, la temperatura debe tener las mismas dimensiones de la función objetivo del problema. Por ejemplo, para el problema del agente viajero, la temperatura T corresponde a una distancia. Valores muy elevados, hacen que el proceso tenga un mayor esfuerzo computacional y valores bajos pueden hacer que el algoritmo quede atrapado en soluciones de baja calidad al inicio del proceso. Aunque existen muchas propuestas, el valor de T_0 puede ser calculado usando el siguiente algoritmo:

i. A partir de una alternativa inicial con una función objetivo E_1, generar k perturbaciones aleatorias (vecinos) y evaluar la función objetivo E_j de cada una. Esto puede verse como una exploración parcial del vecindario, en donde el tamaño k del mismo no es un aspecto crítico. Seguidamente se obtiene el valor promedio de la función objetivo usando la ecuación (6).

$$\Delta E = \frac{\sum_{j=1}^{k} Ej - E1}{k} \qquad (6)$$

ii. Escoger una tasa de aceptación de vecinos τ_0 basada en el porcentaje de alternativas que degraden (desmejoren) la calidad de la alternativa inicial. Valores bajos de τ_0 (por ejemplo τ_0 = 20%) indican que se está asumiendo una alternativa inicial de buena calidad ya que sólo serán aceptados unos pocos vecinos que degraden la alternativa inicial. Es recomendable, con niveles altos de temperatura, usar valores altos de τ_0, por ejemplo $\tau_0 = 50\%$. La ecuación (7) puede ser usada para determinar el valor del parámetro T_0.

$$\tau_0 = e^{-\frac{\Delta E}{T0}} \qquad (7)$$

Otra propuesta para encontrar el valor de la temperatura inicial T_0 consiste en generar las mismas k perturbaciones y contar el número m_1 de veces que se encuentra un vecino de mejor calidad y el número m_2 de veces que se encuentra uno de peor calidad. De esta manera se deduce el valor T_0 de la siguiente expresión:

$$T_0 = \frac{\overrightarrow{\Delta E^+}}{ln\left(\frac{m_2}{m_2\tau_0 - m_1(1-\tau_0)}\right)} \qquad (8)$$

Donde $\overrightarrow{\Delta E^+}$ es el valor promedio considerando únicamente las transiciones que degradaron la función objetivo.

2.3.2 Longitud de la cadena de Markov

Debe permitir al proceso alcanzar el cuasi equilibrio en cada nivel de temperatura. Más que un parámetro, la longitud de la cadena define cuando se debe realizar un cambio de estado de energía. Una manera simple de seleccionar la longitud de la cadena es hacerlo de acuerdo con el tamaño del problema. Así, por ejemplo, un cambio de estado puede tomar lugar cuando una de las siguientes dos condiciones se cumple:

- $12 \cdot N$ perturbaciones son aceptadas.
- $\rho = 100 \cdot N$ perturbaciones son realizadas.

Los valores 12 y 100 pueden ser ajustados de forma diferente, dependiendo del problema que se esté abordando. Cuando se cumple una de las dos condiciones anteriores se dice que el sistema alcanzó el equilibrio termodinámico. El parámetro N indica el número de variables del problema. Para una mayor longitud en la cadena de Markov, significa una mayor exploración del vecindario. Por ejemplo, para el problema del agente viajero con 230 ciudades, $N = 230$ y para el problema de asignación

generalizada con 100 tareas y 5 agentes, $N = 500$. A medida que la temperatura disminuye, menos alternativas vecinas (perturbaciones) son aceptadas, y por lo tanto la exploración del vecindario en busca de una mejor alternativa se intensifica hasta revisar, en el peor caso, $100N$ alternativas vecinas. En otras palabras, esto significa que a medida que la temperatura disminuye la longitud de la cadena de Markov aumenta implícitamente.

El aumento de la cadena de Markov también puede hacerse de forma explícita, haciendo que la segunda condición de equilibrio termodinámico aumente un porcentaje determinado, en cada iteración o cambio de estado k. Se puede usar la siguiente ley geométrica para activar un cambio de estado:

- $12\,N$ perturbaciones aceptadas.
- $\rho_{k+1} = \beta.\rho_k$ perturbaciones realizadas, donde $\rho>1$ y $\rho_0 = 100N$.

2.3.3 Ley de Enfriamiento

El enfriamiento del sistema puede hacerse de muchas formas. La más aceptada, por su simplicidad, es la ley de decremento geométrico, donde la temperatura de un nuevo estado está dada por:

$$T_{k+1} = \alpha \cdot T_k \quad (9)$$

donde α es una constante en el intervalo [0.5-0.9].

2.3.4 Temperatura final

Este parámetro define el criterio de parada del algoritmo. Puede ser usado el criterio de activar la finalización del algoritmo cuando se alcanzan más de n estados de temperatura sin ninguna aceptación de vecinos. Generalmente n es un valor entero entre 3 y 5. También se puede utilizar un número determinado de cambios de estado o iteraciones.

Los valores de muchos de estos parámetros, en ausencia de resultados teóricos, son escogidos a través de ajustes empíricos. Para cierto tipo de problemas, escoger un adecuado ajuste de estos parámetros es una tarea complicada por la gran sensibilidad que éstos producen en el desempeño del algoritmo. Este aspecto es una de las principales desventajas que presenta la metodología de RS [9-10].

El mecanismo de aceptación de alternativas se introduce a través de la regla de aceptación de Metrópolis: Si se tienen dos soluciones, π_i (alternativa actual) y π_j (alternativa modificada), con valores de la función objetivo $f(\pi_i)\,y\,f(\pi_j)$, respectivamente, entonces el criterio de aceptación determina si P_j es aceptado como alternativa actual. Para ello, se aplica la siguiente probabilidad de aceptación P_j:

$$p_j = \begin{cases} 1 & si\ f(\pi_j) - f(\pi_i) \le 0 \\ e^{-\left(\frac{f(\pi_j)-f(\pi_i)}{T_k}\right)} & de\ lo\ contrario \end{cases} \quad (10)$$

En la ecuación (10), es utilizada una versión levemente modificada del factor de Boltzmann, donde, para altas temperaturas este factor es cercano al valor uno, y, por lo tanto, la mayoría de vecinos son aceptados, transformando el algoritmo en una especie de exploración aleatoria en el espacio de solución. Por otro lado, a bajas temperaturas este factor es cercano a cero, y, en consecuencia, la mayoría de movimientos que degradan la función objetivo son rechazados. En este último caso, el algoritmo se comporta como un clásico mejoramiento iterativo, mejor conocido como *"Estrategia Golosa"*. En una temperatura intermedia, el algoritmo tiene la capacidad de salir de óptimos locales, ya que permite degradamientos controlados de la función objetivo.

El algoritmo de RS tiene como características principales la ventaja de adaptarse fácilmente a una gran diversidad de problemas y ser una metaheurística cuya convergencia es garantizada bajo ciertas condiciones. Sin embargo, el establecimiento de estas condiciones de convergencia no ha sido aceptado por unanimidad en la comunidad científica. Algunos estudios basados en cadenas de Markov, como el propuesto por [11], demuestran el comportamiento asintótico en la convergencia del método. Esta demostración se basa en cumplimiento de dos propiedades: la reversibilidad y la conectividad. La propiedad de reversibilidad establece que para cualquier cambio permitido en el sistema debe permitirse también una oposición al cambio.

La propiedad de conectividad establece que cualquier estado del sistema puede ser alcanzado comenzando desde cualquier otro estado, en un número finito de cambios elementales. Estas dos propiedades permiten justificar el decremento de la temperatura en estados, de lo cual depende la velocidad de convergencia del algoritmo, y garantiza que una solución de buena calidad puede ser encontrada en un tiempo polinomial para ciertos problemas de tipo NP-hard [11]

Sin embargo, otros autores enfocan sus estudios en el comportamiento asintótico del método [12-13]. El principal resultado de este trabajo es que se demuestra que el algoritmo de RS converge a un óptimo global con una probabilidad igual a la unidad, siempre que, en un tiempo $t \to \infty$, la temperatura $T(t)$ no se decremente más rápidamente que la expresión $C=log(t)$, donde C es una constante que depende del problema. Este resultado significa, que hasta ahora no existe un método lo suficientemente generalizado y unánime para ser aplicado a cualquier tipo de problema.

Algunas verificaciones importantes, que muestran un adecuado ajuste de los parámetros involucrados en la técnica de RS, son las siguientes:

- La generación de números reales aleatorios en el intervalo (0,1) debe ser caracterizada por una distribución uniforme.
- La calidad de los resultados no debe variar significativamente cuando el algoritmo es ejecutado varias veces, con diferentes semillas y con diferentes configuraciones iniciales.

En la Figura 4 se presenta el diagrama de flujo del algoritmo descrito.

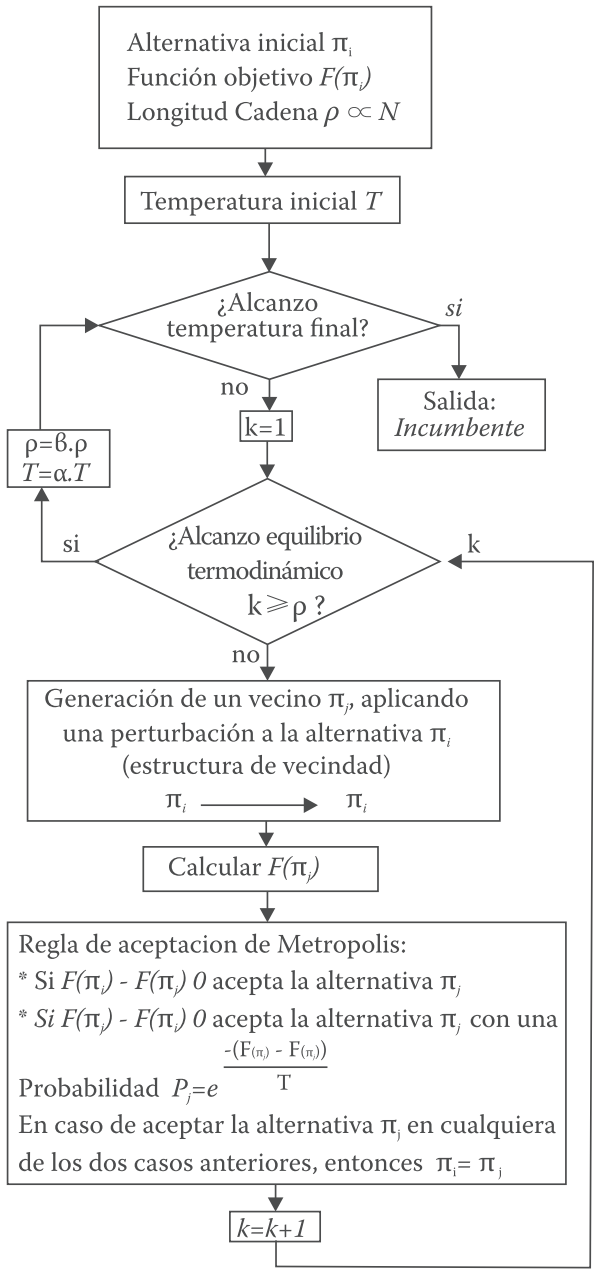


Figura 4. Diagrama de flujo del algoritmo propuesto.

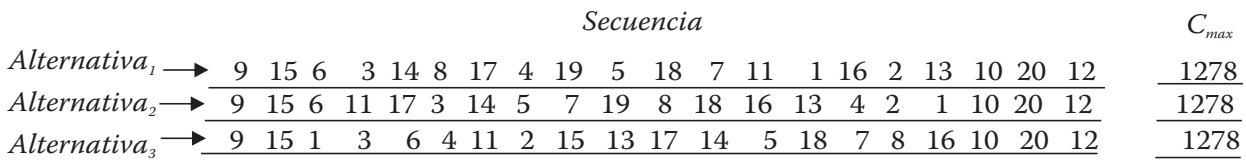
3. Resultados y análisis

El algoritmo propuesto se utilizó para resolver un problema con 20 tareas ($n = 20$) y 5 máquinas ($m = 5$). Los tiempos de procesamiento de cada tarea son dados en la Figura 5. Estos datos corresponden a un caso de prueba reconocido, que puede ser encontrado en el siguiente sitio de Internet: http://mistic.heig-vd.ch/taillard/

El algoritmo propuesto se ejecutó 30 veces y la mejor respuesta obtenida igualó a la reportada en la literatura en 3 oportunidades. En todas las veces, las respuestas fueron cercanas al óptimo reportado lo que muestra un adecuado ajuste de los parámetros de control, para este caso particular. Las mejores soluciones encontradas se muestran en la Figura 6. Como se puede apreciar, existen diferentes alternativas con el mismo valor óptimo.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}	n_{13}	n_{14}	n_{15}	n_{16}	n_{17}	n_{18}	n_{19}	n_{20}
m_1	54	83	15	71	77	36	53	38	27	87	76	91	14	29	12	77	32	87	68	94
m_2	79	3	11	99	56	70	99	60	5	56	3	61	73	75	47	14	21	86	5	77
m_3	16	89	49	15	89	45	60	23	57	64	7	1	63	41	63	47	26	75	77	40
m_4	66	58	31	68	78	91	13	59	49	85	85	9	39	41	56	40	54	77	51	31
m_5	58	56	20	85	53	35	53	41	69	13	86	72	8	49	47	87	58	18	68	28

Figura 5. Tiempos de Procesamiento.



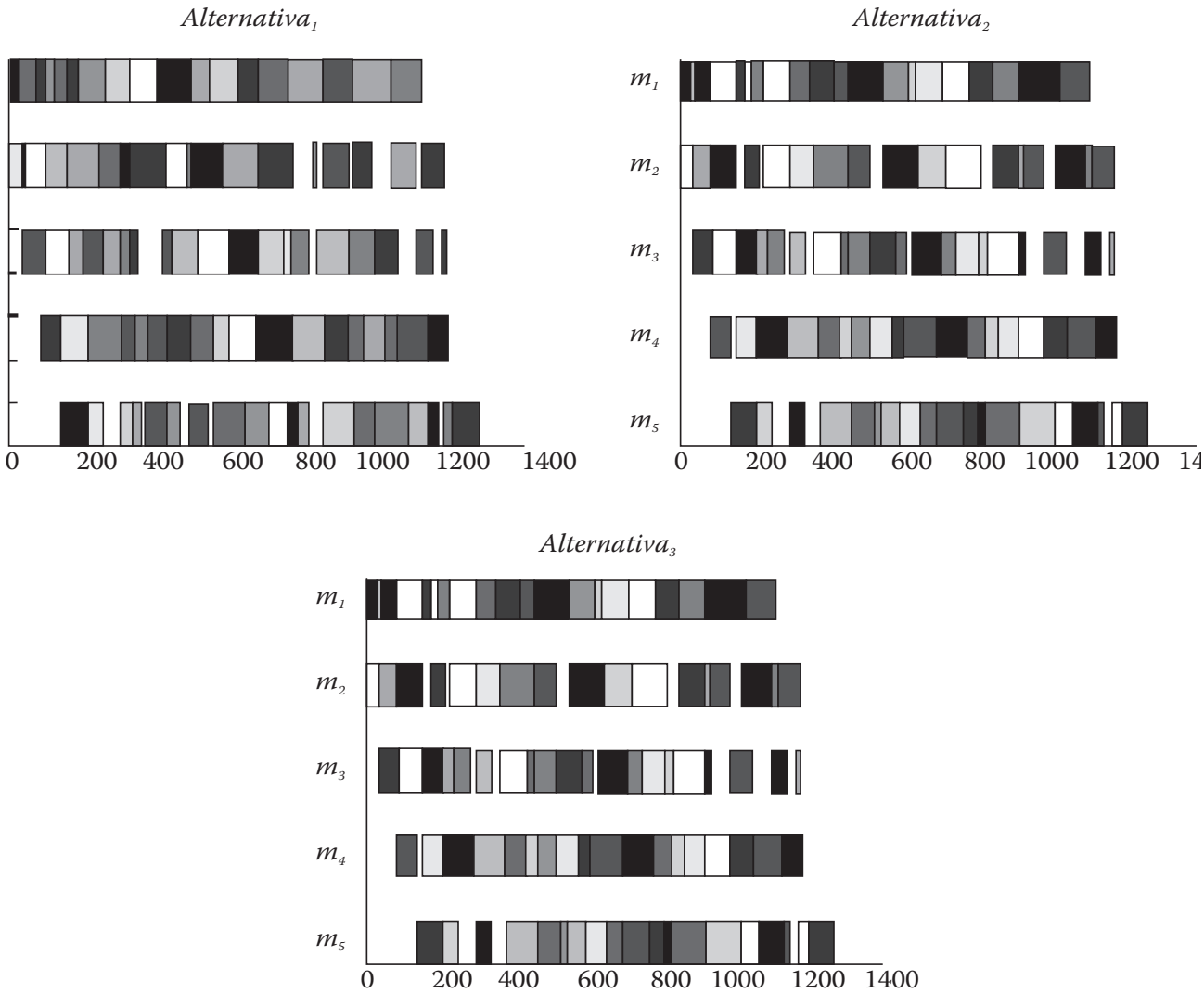


Figura 6. Mejores alternativas encontradas en 30 ejecuciones.

4. Conclusiones

En este trabajo se generó una descripción detallada del algoritmo de Recocido Simulado, con lo cual se pretende mostrar los beneficios de esta estrategia cuando es aplicada a problemas de optimización combinatorial NP-Hard.

En particular, se resolvió el problema de secuenciamiento de tareas en línea (flow-shop) para un problema de 20 tareas y 5 máquinas. Las respuestas obtenidas alcanzaron, en todas las veces, respuestas cercanas al óptimo reportado.

Referencias

1. Toro Ocampo, E. M., Restrepo G., J.S. and Granada E., M.: *Algoritmo Genético Modificado Aplicado al Problema de Secuenciamiento de Tareas en Sistemas de Producción Lineal – Flow Shop. Scientia et Technica,* año XII, No 30, mayo 2006, pp. 285-290.
2. Kirkpatrick, S. and Toulouse, G.: 1985, *Configuration space analysis of travelling salesman problems*, J. Physique, 46 pp. 1277–1292.
3. Cerny, V.: 1985, *Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm*, J. of Optimization Theory and Applications, 45(1) pp. 41–51.
4. Metropolis, N., R. A. R. M. T. A. and Teller, E.: 1953, *Equation of state calculations by fast computing machines*, J. Chem. Phys., 21 pp. 1087–1090.
5. Yong, Z. and Sannomiya, N.: 2000, *A method for solving large-scale flowshop problems by reducing search space of genetic algorithms,* IEEE International Conference on Systems, vol. 3 pp. 1776–1781.
6. Kirkpatrick, S., G. C. and Vecchi, M.: 1983, *Optimization by simulated annealing*, Science, 220(4598) pp. 671–680.
7. Rammal, R., T. G. and Virasoro, M.: 1986, *Ultrametricity for physicists,* Reviews of Modern Physics, 58(3) pp. 765–788.
8. Solla, S., S. G. and White, S.: 1986, "Configuration space analysis for optimization problems", In Bienenstock, E., Fogelman Soulie, F., and Weisbuch, G., editors, *Disordered Systems and Biological Organization,* New York, Springer-Verlag. pp. 283–292.
9. Dréo, J., P. A. S. P. and Taillard, E.: 2003, *Metaheuristics for Hard Optimization*. Springer.
10. Reeves Colin. *A generis Algorithm for FlowshopSequencing*. Pergamon. Computers Ops Res. Vol 22No 1, pp 5-13, 1995 Great Britain.
11. Aarts, E. H. L. and Van Laarhoven, P. J. M.: 1985, *Statistical cooling: a general approach to combinatorial optimization problems*, Philips J. of Research, 40 pp. 193–226.
12. Hajek, B.: 1988, *Cooling schedules for optimal annealing*, Math. Oper. Res., 13 pp. 311–329.
13. Hajek, B. and Sasaki, G.: 1989, *Simulated annealing to cool or not*, Systems and Control Letters, 12 pp. 443–447.