REVISTA Universidad EAFIT
Vol. 39. No. 131. 2003. pp. 76-86

Algoritmo
de búsqueda aleatoria para la programación de la producción en un taller de fabricación

Mario César Vélez Gallego
Master en Ingeniería Industrial de la Universidad de los Andes.Profesor del Departamento de Ingeniería de Producción de la Universidad EAFIT.
marvelez@eafit.edu.co

Carlos Alberto Castro Zuluaga
Master en Ingeniería Industrial de la Universidad de los Andes . Profesor del Departamento de Ingeniería de Producción de la Universidad EAFIT.
ccastro@eafit.edu.co

Jairo Maya Toro
Master en Ingeniería de Controles de la Universidad de *Bochum*, Alemania. Profesor del Departamento de Ingeniería de Producción de la Universidad EAFIT.
jmaya@eafit.edu.co

Recepción: 20 de enero de 2003 I Aceptación: 23 de abril de 2003

Resumen

El problema de la programación de producción en talleres de fabricación o configuraciones *job-shop* fue uno de los casos de programación más complejos que se derivaron de múltiples trabajos de investigación que surgieron después de que en 1954, S. M. Johnson publicara una solución al problema de minimizar el tiempo de ejecución de *n* trabajos en dos máquinas. Sin embargo, al igual que muchas otras situaciones, de este tipo, el problema de configuraciones *Job - Shop* tampoco se ha podido resolver a pesar de los adelantos tecnológicos, principalmente porque la programación puede arrojar una gran cantidad de posibles soluciones y es difícil hacer una evaluación que permita obtener la solución más óptima. Este artículo presenta una posible solución a este problema, con base en la técnica de búsqueda aleatoria en la región factible, en la cual se generan soluciones aleatorias, y se guarda la mejor de las soluciones obtenidas. Esta técnica, aunque no garantiza la obtención de la solución más óptima, permite obtener buenas soluciones en intervalos de tiempo muy cortos.

Palabras Claves

Programación de Producción/
Búsqueda Aleatoria Pura/
Secuenciación

Abstract

Since 1954, when Mr. S. M. Johnson published a solution that minimized the time for completion of *n* jobs in two machines, research in production scheduling has been enormous. During the last 40 years, there has been a lot of improvement in the search for answers for more complex problems than the one solved by Johnson. However, despite technological advances, many of these problems are still waiting for a solution. This article exposes a solution to the problem of job shop scheduling based on the random search method, in which solutions are randomly generated and then the best obtained is saved. Although this technique doesn't guarantee optimal solutions, it has proved to find very good solutions in short periods of time.

Key Words

Manufacturing Scheduling/ Random Search/ Job-Shop / Scheduling

1. Introducción

La programación de producción ha sido uno de los temas de investigación más recurrentes en el área de la administración de operaciones durante la última mitad del siglo XX. Debido a la cada vez más fuerte competencia en los mercados, para muchas empresas se ha hecho necesario ampliar su portafolio de productos, buscando de esta manera satisfacer a más clientes potenciales. Este cambio aparentemente simple trae consigo grandes problemas en la planeación y el control de los sistemas productivos: Un número amplio de referencias implica mayores niveles de inventario o mayores exigencias en cuanto a capacidad instalada, lo que a su vez implica mayores costos operacionales. Si a estos problemas se le suma una pobre programación de la producción, éstos se pueden ver multiplicados.

Debido a estas razones y a la gran complejidad que implica encontrar una solución al problema de la programación de la producción, éste ha sido el tema de muchos proyectos de investigación durante los últimos cuarenta años. En especial el tema de la programación de producción en talleres de fabricación o configuraciones del tipo *job-shop* es el que atrae más la atención de los investigadores debido a que es especialmente complejo de resolver.

2. El problema de la programación de producción en talleres de fabricación

Un taller de fabricación o *job-shop* es una configuración productiva que permite fabricar una gran cantidad de productos diferentes. Este tipo de sistema se caracteriza por tener máquinas de propósito general, y porque cada trabajo tiene una ruta de fabricación única, que no necesariamente se repite entre trabajos. Un taller de maquinado constituye un buen ejemplo de un *job-shop*: Sus máquinas -tornos, fresadoras, taladros, rectificadoras, etc.- son de propósito general, ya en ellas se pueden procesar una casi infinita gama de productos diferentes. En estos talleres cada producto puede tener una ruta diferente dentro del sistema; es decir, alguna orden de producción irá primero al torno para luego pasar a la fresadora y por último a la rectificadora, mientras que otra orden puede iniciar su proceso en la fresadora, seguir a la rectificadora y terminar en el torno. Explicaciones más detalladas sobre las características de un sistema *job-shop* pueden encontrarse en Miltemburg (1996) y Sipper (1997).

Jain, y Meeran (1999) describen el problema de la programación de producción en ambientes *job-shop*

de la siguiente manera: Un conjunto de *n* trabajos debe ser procesado en *m* máquinas. Cada trabajo debe ser procesado en las máquinas en un orden establecido, que no puede ser violado bajo ninguna circunstancia. Esta última condición se conoce como la restricción de precedencia, y consiste en que, por ejemplo, un producto no puede ser empacado sin haber sido pintado antes. En este caso la operación de pintura precede a la operación de empaque. Este problema se conoce como el problema de programación de producción *job-shop* de *n x m*, donde *n* es el número de trabajos y *m* el número de máquinas.

Considérese una situación en la cual cuatro trabajos (1, 2, 3, 4) van a ser programados en cuatro máquinas (A, B, C, D) de acuerdo con la secuencia y los tiempos dados en la tabla 1.

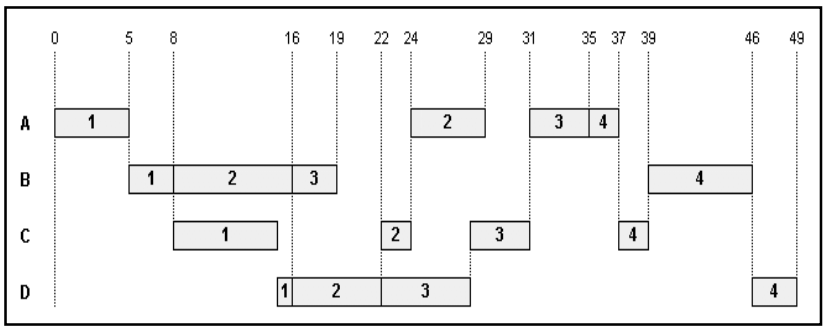
El siguiente es un problema de programación de *job-shop* de 4×4; es decir, cuatro trabajos para ser programados en cuatro máquinas. En la figura 1 se puede apreciar una solución al problema expresada en forma de diagrama de Gantt.

Nótese que en el trabajo 1, la operación en la máquina A precede a la operación en la máquina B, y ésta a su vez precede la operación en la máquina C. Estas son las relaciones de precedencia mencionadas anteriormente. Una de las características que hace que la programación de producción en ambientes *job-shop* sea tan compleja de resolver es precisamente que estas restricciones de precedencia cambian de un trabajo a otro, como se puede apreciar en la tabla 1.

Tabla 1. Ejemplo de un problema de programación de producción en un ambiente *job-shop*

Trabajo	Operación (Máquina, tiempo)			
	1	2	3	4
1	(A, 5)	(B, 3)	(C, 7)	(D, 1)
2	(B, B)	(D, 6)	(C, 2)	(A, 5)
3	(B, 3)	(D, 6)	(C, 4)	(A, 4)
4	(A, 2)	(C, 2)	(B, 7)	(D, 3)

Figura 1. Diagrama de Gantt de una solución al problema de programación de producción



Los supuestos básicos de los que parten la mayor parte de las soluciones propuestas son los siguientes (Sipper, 1997):

- Los tiempos de operación son conocidos con certidumbre.
- Los tiempos de cambio de referencia son conocidos e independientes del orden de procesamiento.
- Todos los trabajos que se van a procesar están disponibles en tiempo cero.
- No existen restricciones de precedencia entre trabajos. Sólo se consideran restricciones de precedencia entre las operaciones de un mismo trabajo.
- Una vez que el trabajo está montado en una máquina, éste no puede ser interrumpido.
- Aunque en algunos casos, estos supuestos no se ajustan mucho a la realidad, tratar de relajarlos implicaría una complejidad aun mayor en el ya bastante complejo problema.

Adicionalmente existen en la realidad condiciones frecuentes tales como:

- Estaciones de trabajo: múltiples máquinas donde es posible realizar una o varias de las operaciones de un mismo trabajo.
- Operaciones que se realizan fuera del taller (recubrimientos superficiales, tratamientos térmicos etc.)
- Restricción de recursos: Las herramientas y/o dispositivos y operarios son recursos limitados que suelen ser compartidos por máquinas de una misma estación o estaciones diferentes.
- Lotes de transporte: Cuando un trabajo está conformado por K piezas idénticas, la cantidad de piezas que se producen después de cada Setup en cada máquina la denominaremos Lote, y la cantidad de piezas que se transportan entre máquinas la denominaremos Lote de transporte, el cual no necesariamente es igual al Lote, por el contrario, el desempeño del sistema aumenta cuando el tamaño del lote de transporte se aproxima a la unidad.

Para la solución de este problema en particular se han planteado diversos objetivos, que van desde disminuir el inventario en proceso hasta mejorar el servicio al cliente entendido como cumplimiento de las fechas de entrega. Sipper (1997) enumera los objetivos más comunes y que han recibido la mayor atención por parte de los investigadores:

- Minimizar el flujo promedio $\overline{F}$, o el tiempo medio que un trabajo permanece en el taller.
- Minimizar el tiempo requerido para terminar todos los trabajos. Esta medida de desempeño se conoce como el *makespan* o C_{max}. En la figura 1 se puede apreciar que la solución propuesta presenta un *makespan* de 49 unidades de tiempo.
- Minimizar el número de trabajos retrasados.
- Minimizar la tardanza promedio $\overline{T}$, o el tiempo promedio que un trabajo se retrasa con respecto a la fecha de entrega pactada con el cliente.

De los anteriores, el objetivo más usado en la literatura es el de minimizar el *makespan* o C_{max}. Este objetivo es equivalente a minimizar los tiempos muertos, o a maximizar la utilización de las máquinas, y ésta es tal vez la razón por la cual ha sido adoptado con mayor frecuencia por los investigadores.

La complejidad del problema de programación de producción en *job-shop* radica en la cantidad abrumadora de posibles soluciones. Para un problema de *n x m,* esta cantidad está dada por $(n!)^m$. En estas condiciones un problema aparentemente sencillo de *20 x 10,* en el que se requiera programar 20 trabajos en 10 máquinas tiene 7.2652×10^{138} posibles soluciones. Si una computadora tardara un microsegundo (10^{-6} segundos) en evaluar cada solución, la edad estimada del universo no bastaría para evaluar todas las posibles soluciones al problema.

En la década de 1950, cuando se planteó por primera vez este problema, se hicieron algunos avances importantes, como la obtención de soluciones óptimas para los problemas *2xm* y *nx2,* resuelto por Akers (1956), y Jackson (1956) respectivamente. Estos algoritmos desarrollados se

conocen como Métodos Eficientes. Desde entonces, y a pesar de los grandes progresos alcanzados durante los últimos años, no existen en la actualidad Métodos Eficientes para encontrar soluciones aceptables a problemas con $m \geq 3$ y $n \geq 3$ (Jain y Meeran, 1999). Las formulaciones matemáticas han sido otro procedimiento propuesto para resolver el problema, utilizando técnicas de programación matemática como la programación lineal entera mixta (Manne,1960; Van Hulle, 1991), aunque estas técnicas han sido cuestionadas por Giffler y Thompson (1960) y French(1982). Otros procedimientos de optimización desarrollados para resolver el problema de programación de producción, han sido las técnicas Branch and Bound (Lageweg *et al*, 1977; Barker y McMahon, 1985), los Métodos de Aproximación (Baker, 1974; French, 1982; Morton y Pentico, 1993), las Heurísticas basadas en el cuello de botella (Adams *et al*, 1988), las técnicas de Inteligencia Artifical (IA), tales como las Redes Neuronales (Castro, 1998; Zhang y Huang, 1995) y últimamente los Métodos de Búsqueda Locales y Meta-heurísticos, entre los cuales se encuentra la Simulación por Recocido (Kirkpatrik *et al*, 1983; Cerny, 1985), Algoritmos Genéticos (Holland, 1975; Goldberg, 1989) y Búsqueda Tabú (Glover, 1989, 1990).

1. La búsqueda aleatoria pura como técnica para resolver problemas complejos

La búsqueda aleatoria pura consiste en generar aleatoriamente un número grande de soluciones a un problema de optimización, normalmente generadas por medio de la distribución uniforme, y seleccionar la mejor de ellas. De acuerdo con Shi (2000), la búsqueda aleatoria pura casi siempre converge a la solución óptima y es aplicable a casi la totalidad de los problemas de optimización debido a los pocos supuestos necesarios para su uso.

La principal desventaja que presenta el método radica en que puede ser lento para encontrar una solución óptima o muy cercana al óptimo, aunque este problema se ve mitigado por la gran evolución que los sistemas cómputo tienen en la actualidad.

En problemas de gran complejidad, y para los cuales no existen procedimientos que permitan resolverlos de manera satisfactoria, como es el caso del problema de programación de producción en *job-shop*, la búsqueda aleatoria pura se presenta como una alternativa, ya que permite abordar casi cualquier tipo de problema de optimización, con la ventaja adicional de que no es necesario hacer prácticamente ningún supuesto; uno de los principales problemas de los modelos de optimización tradicionales.

El algoritmo de la búsqueda aleatoria pura puede expresarse de la siguiente forma:

Sea f(e) una función a minimizar, donde f es una función de la solución e.

Sea E el conjunto de posibles soluciones.

Generar una solución inicial $e_0 \in E$
$h_0 \leftarrow F(e_0)$
$e_{optimo} \leftarrow e_0$ y *Hacer* $h_{optimo} \leftarrow h_0$
$k \leftarrow 1$
Mientras que $k \leq N$
    Generar una solución aleatoria e_k
    Calcular h_k $f(e_k)$
    Si $h_k < h_{optimo}$, entonces
        $e_{optimo} \leftarrow e_k$
        $h_{optimo} \leftarrow h_k$
    Fin Si
    $k \leftarrow k + 1$
Fin Mientras

El método para generar soluciones aleatorias puede variar considerablemente para cada problema. Sin embargo, por lo general se usan números pseudo-aleatorios con distribución uniforme para obtener estas soluciones, ya que de esta manera se garantiza que las soluciones evaluadas provienen de diferentes zonas dentro del espacio solución, y que no se está dando preferencia a ninguna en particular.

2. Solución propuesta: algoritmo para la programación de la producción en talleres de fabricación

Debido a que los problemas de programación de producción en ambientes *job-shop* son muy complejos, no existen en la actualidad algoritmos que permitan encontrar programas óptimos en un periodo razonable de tiempo. Por esta razón los esfuerzos de los investigadores se han concentrado en desarrollar métodos que encuentren buenas soluciones en periodos razonables de tiempo (Hopp, 1996). Según Hopp, un programa de producción óptimo sólo es válido en un modelo matemático. En la práctica lo que se necesita es una solución factible y buena.

Aun si se pudiera encontrar una solución óptima, ésta estaría sustentada en supuestos como el carácter determinístico de los tiempos de operación y de cambio de referencia. Debido a que en la realidad estos tiempos son aleatorios, y en algunos casos con altos niveles de variación, la búsqueda de una solución óptima determinística en un problema con variables aleatorias es, por decirlo de alguna manera, una ilusión. En la práctica, lo que se necesita en un taller de fabricación es una buena solución a un problema imposible, y es bajo esta consideración que los métodos de solución como la búsqueda aleatoria pura adquieren relevancia.

De acuerdo con la experiencia recogida por los autores, lo que se hace en la práctica es que la persona encargada de la programación, con base en su experiencia y conocimiento del proceso, encuentra una solución factible al problema, muchas veces sin ningún tipo de ayuda computacional. Si partimos de la base de que una computadora puede procesar miles de veces más soluciones que una persona, entonces es muy probable que con la ayuda de una técnica como la búsqueda aleatoria, se encuentren soluciones mejores a las encontradas de forma manual por las personas del taller.

El algoritmo propuesto en este trabajo usa la búsqueda aleatoria pura con una variante importante: Este algoritmo sólo genera soluciones aleatorias factibles; es decir, soluciones que cumplan con las restricciones de precedencia descritas anteriormente. Por ejemplo, si se considera el problema planteado en la tabla 1, en una primera etapa del algoritmo, sólo la primera operación de cada trabajo podría ser programada. Si aleatoriamente se seleccionara la primera operación del trabajo 3 para ser programada; entonces la nueva tarea a programar puede ser la primera operación de los trabajos 1, 2 y 4, y la segunda operación del trabajo 3. Bajo esta lógica, la búsqueda aleatoria sólo recorre soluciones factibles.

3. Resultados obtenidos

Para evaluar los beneficios potenciales de la solución propuesta se tomó un problema de programación de 6 trabajos en 4 máquinas, y se le pidió a un grupo de seis personas conocedoras del problema que lo resolvieran. El problema propuesto en la tabla 2.

A cada persona se le pidió que encontrara un programa de producción que minimice el tiempo de terminación de todos los trabajos, o el makespan. También se midió el tiempo que utilizó cada una de ellas en encontrar una solución al problema. Las soluciones y los tiempos se ilustran en la tabla 3.

Los resultados obtenidos para el mismo problema usando el algoritmo propuesto se presentan en la tabla 4.

El algoritmo de búsqueda aleatoria muestra resultados significativamente mejores a los que puede obtener una persona sin ninguna herramienta de apoyo. La diferencia más importante radica en el tiempo de búsqueda, que en promedio es de 17.7 segundos; mientras que una persona que conoce el problema y ha trabajado en el tema tarda en promedio 19.66 minutos en obtener una solución generalmente inferior a la encontrada por el algoritmo. Mientras que el algoritmo propuesto encontró soluciones de hasta 52 unidades de tiempo en el makespan, la mejor solución encontrada por las personas fue de 54 unidades de tiempo.

El algoritmo propuesto es el siguiente:

Sea M el número de máquinas disponibles y N el número de tareas a programar
Sea $S_{(i,k)}$ *la máquina en la cual se procesa la* k *-esima operación de la tarea i*
Sea $T_{(i,j)}$ *el tiempo que tarda la tarea i en la máquina j*
Sea X_j *el tiempo en que termina la última tarea programada en la máquina j*
Sea Y_i *el tiempo en que termina la última operación programada de la tarea i*
Sea U_i *la cantidad de operaciones programadas de la tarea i*
Sea F_j *el conjunto de tareas candidatas a ser programadas en la máquina j*
Sea C_j *la cantidad de tareas candidatas a programar en la máquina j*
Sea m un número aleatorio discreto con distribución uniforme en el intervalo* $[1, M]$
Sea t un número aleatorio discreto con distribución uniforme en el intervalo* $[1, C_m\bullet]$
Sea $Makespan_l$ *el makespan de la l-esima solución encontrada*
Sea Makespan el mínimo Makespan encontrado*
Sea I el número de iteraciones
Sean i, c, k, l contadores

$Makespan^* \leftarrow \infty$
Para $l = 1$ *hasta I*
  $Makespan_l \leftarrow 0$
  Para $c = 1$ *hasta* $M \times N$
    Hacer $F_j \leftarrow \varphi \quad \forall j$
    Hacer $U_i = 0 \quad \forall i$
    Hacer $C_j \leftarrow 0 \quad \forall j$
    Para $i = 1$ *hasta N*
      $F_{S_{(i,U_i+1)}} \leftarrow i$
      $C_{S_{(i,U_i+1)}} \leftarrow C_{S_{(i,U_i+1)}} + 1$
    Siguiente i
    Generar $m^* \mid C_m^* \neq 0$
    *Generar t**
    Programar la tarea t en la máquina m*, desde Max* $\{X_m^*, Y_t^*\}$ *hasta Max* $\{X_m^*, Y_t^*\} + T_{(t^*, m^*)}$
    $Makespan_l \leftarrow Max\{Max\{X_m^*, Y_t^*\} + T_{(t^*, m^*)}, Makespan_l\}$
    $X_m^* \leftarrow Max\{X_m^*, Y_t^*\} + T_{(t^*, m^*)}$
    $Y_m^* \leftarrow Max\{X_m^*, Y_t^*\} + T_{(t^*, m^*)}$
    $U_t^* \leftarrow U_t\bullet + 1$

  Siguiente c
  $Makespan^* \leftarrow Min\{Makespan_l, Makespan^*\}$
Siguiente l

Tabla 2. Problema propuesto

Trabajo	Operación (Máquina, tiempo)			
	1	2	3	4
1	(A,6)	(B,8)	(C,13)	(D,5)
2	(A,4)	(B,1)	(C,4)	(D,3)
3	(D,3)	(B,8)	(A,6)	(C,4)
4	(B,5)	(A,10)	(C,15)	(D,4)
5	(A,3)	(B,4)	(D,6)	(C,4)
6	(C,4)	(A,2)	(B,4)	(D,5)

Tabla 3. Resultados del experimento

Persona	Makespan	Tiempo (min)
1	54	20
2	59	22
3	57	19
4	66	15
5	58	27
6	54	15
Promedio	58	19.66
Desviación estándar	4.43	4.55

Tabla 4. Resultados con el algoritmo propuesto

Corrida	Makespan	Tiempo (min)
1	54	0.30215
2	53	0.2938
3	54	0.2929
4	52	0.2983
5	53	0.2928
6	55	0.2938
Promedio	53.5	0.2956
Desviación estándar	1.048	0.003789

4. Análisis comparativo

En esta sección se compara el algoritmo de búsqueda aleatoria pura propuesto, el cual se hace parte del software de Planeación, Programación y Control de la Producción: Arquímedes (2002), proyecto desarrollado con el auspicio de la Universidad EAFIT por Departamento de Ingeniería de Producción de la Universidad EAFIT, con el método de Shifting Bottleneck propuesto por Pinedo (1999), utilizando el software de dominio público Lekin diseñado en Stern School of Business, de la Universidad de New York en un proyecto dirigido por el Profesor Michael L. Pinedo y el Profesor Asociado Xiuli Chao. El análisis comparativo se hizo para problemas de *nX4* y problemas con una matriz cuadrada *nXm*, hallando el porcentaje de error entre los resultados obtenidos para el *makespan* entre los dos métodos. El porcentaje de error se calculó de la siguiente manera:

$$\%\ Error = \frac{C_{max}Arquimedes - C_{max}Lekin}{C_{max}Lekin}$$

En el gráfico 1 se muestran los resultados obtenidos del porcentaje de error para problemas de 6 a 40 trabajos en 4 máquinas. Como se puede observar, el método propuestos por Pinedo obtuvo para uno de los problemas resueltos, un valor del 9% mejor en comparación con el algoritmo propuesto; en 5 problemas un resultado mejor en un 4,5% en promedio, y en los restantes problemas los errores encontrados fueron menores al 1,2%, llegando Arquímedes a superarlo en algunas oportunidades.

Se puede ver entonces que los problemas pequeños planteados los márgenes de error son relativamente bajos.

El gráfico 2 muestra el porcentaje de error para problemas *nXm*, en el cual el número de máquinas y de trabajos se incrementa desde 5 hasta 10, generando un alto número de combinaciones posibles con respecto al problema *nX4* explicado anteriormente.

Como se puede observar el error máximo que se presenta para los diferentes problemas es del 20%. Sin embargo es importante resaltar que el algoritmo de búsqueda aleatoria desarrollado para Arquímedes además de cumplir con las restricciones de precedencia descritas anteriormente, relaja los siguientes supuestos de los enunciaron en el numeral 2 del presente artículo:

- Los tiempos de cambio de referencia pueden ser dependientes del orden de procesamiento.
- Todos los trabajos que se van a procesar pueden no estar disponibles en tiempo cero.
- Se aceptan estaciones de trabajo (Una operación puede ser programa en varias máquinas).

- Pueden existir operaciones que se realizan fuera del taller.
- Se pueden definir restricción de recursos.
- Se puede definir Lotes de transporte

Gráfico 1. Porcentaje de Error problemas *nX4*

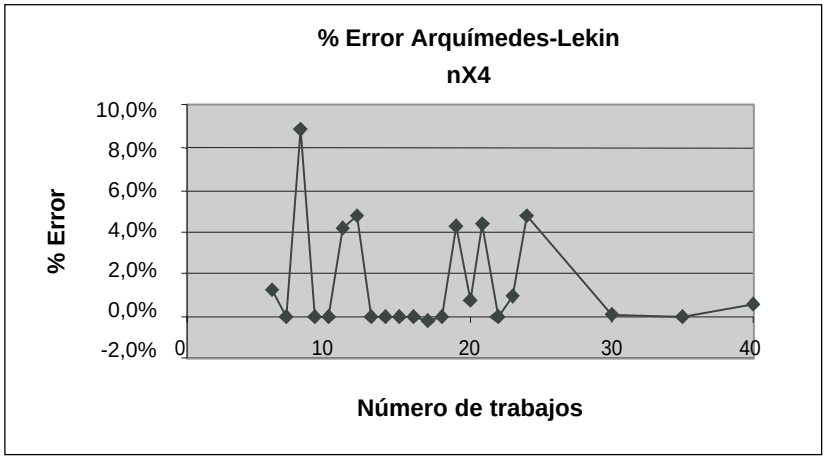
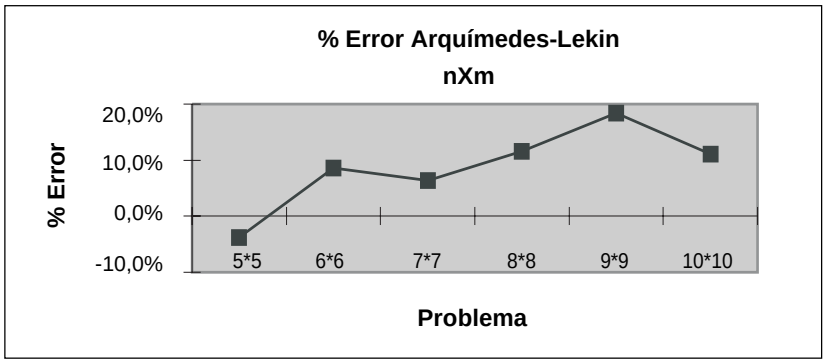


Gráfico 2. Porcentaje de Error problema *nXm*



La relajación de las restricciones antes descritas hacen que Arquímedes sea aplicable a situaciones industriales reales, obteniendo soluciones que, aunque no son óptimas, se presentan como una buena aproximación a la realidad de la programación de la producción de talleres de fabricación. Adicionalmente es importante resaltar que a medida que el problema es más real, es decir se aumentan el número de restricciones, se disminuye la cantidad de soluciones factibles, es decir, se aumenta la eficiencia del algoritmo de búsqueda aleatoria en la región factible.

Conclusiones

- El método de búsqueda aleatoria pura propuesto para resolver el problema de la programación de producción en talleres de fabricación probó ser muy efectivo en al solución de este tipo de problemas. El algoritmo obtuvo soluciones iguales o mejores a las obtenidas por personas sin ayuda computacional, en un periodo de tiempo considerablemente menor.
- Si se compara el algoritmo propuesto con una persona acostumbrada a programar la producción de forma manual, y sin ninguna ayuda computacional; el primero obtiene en promedio, soluciones con un makespan 8.4% mejor. Estos mejores resultados se obtienen 65 veces más rápido con el algoritmo propuesto.
- Comparando el algoritmo propuesto con un heurístico desarrollado para aplicaciones industriales, se puede observar que error porcentual promedio desfavorable para nuestro algoritmo es de 1,6% para problemas pequeños y del 8,7% para problemas grandes, manteniendo en el algoritmo de búsqueda aleatoria la relajación de casi la totalidad de las restricciones impuestas para este tipo de problemas, convirtiendo entonces la búsqueda aleatoria en una buena opción para resolver los problemas reales de programación de producción.
- En problemas de optimización altamente complejos, los métodos no convencionales de optimización, aunque no garantizan la obtención de una solución óptima, se convierten en una alternativa muy importante, ya que siempre será preferible una solución buena en unos pocos minutos que una solución óptima en varios cientos de días.
- A medida que la capacidad de los computadores crece, se incrementa la posibilidad de explorar áreas cada vez mayores en el espacio solución, lo que a su vez garantiza que se encuentren mejores soluciones. Esto hará posible que en el futuro cercano se resuelvan muchos problemas en el área de la administración de operaciones, que hasta ahora no han podido ser resueltos debido a la gran complejidad que encierran.

Bibliografía

Adams, J., Balas, E. and Zawack, D. "The Shifting Bottleneck Procedure for Job-Shop Scheduling". En: *Management Science*. March, 34(3), pp. 391-401.

Akers, S.B. (1956). "A Graphical Approach to Production Scheduling Problems". En: *Operations Research*. Vol. 4. pp. 244-245.

Baker, K. R. (1974). Introduction to Sequencing and Scheduling. John Wiley & Sons. New York.

Barker, J.R. and McMahon, G.B. (1985). "Scheduling the General Job-Shop". En: *Management Science*. Vol. 31, pp. 594-598.

Castro, C.A. (1998). "Aplicación de las Redes Neuronales de Hoppfield al problema del Job-Shop Scheduling". Tesis de Maestría. Universidad de los Andes. Bogotá

Cerny, V. (1985). "Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm". En: *Journal of Optimization Theory and Applications*. Vol. 45, pp. 41-51.

French, S. (1982). "Sequencing and Scheduling - An Introduction to the Mathematics of the Job-Shop. Ellis Horwood, John Wiley & Sons. New York.

Giffer, B. and Thompson G.L. (1960). "Algorithms for Solving Production Scheduling Problems, En: *Operations Research*. Vol. 8, pp. 487-503.

Goldberg, D. E. (1989). "Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley, Reading, Massachusetts.

Glover, F. (1989). "Tabu Search-Part 1". En: *ORSA Journal on Computing*. Vol.1, pp. 190-206.

Glover, F. (1989). "Tabu Search-Part 2". En: *ORSA Journal on Computing*. Vol.2, pp. 4-32.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. The University of Michigan Press. Ann Arbor.

Hopp W. y Spearman, M. (1996). Factory Physics. McGraw Hill.

Jain, A. y Meeran, S. (1998). "A state of the art review of job shop scheduling techniques". En: Technical Report. Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland.

Jackson, J.R. (1956). "An Extension of Johnson's Result on Job Lot Scheduling". En: *Naval Research Logistics Quarterly*. Vol. 3, pp.201-203.

Johnson, S. M. (1954). "Optimal two and three stage production schedules with setup times included". En: *Naval Research Logistics Quarterly*. Vol. 1. pp. 61-68.

Kirkpatrick, S., Gelatt, C. D. Jr. and Vecchi. M. P. (1983). "Optimization by Simulating Annealing". En: *Science*. 220(4598), 13 May, pp. 671-680.

Lagewegm B. J., Lenstra, K. and Rinnooy Kan, A. H. G. (1977). "Job-Shop Scheduling by Implicit Enumeration". En: *Management Science*. Vol. 2, pp. 441-450

Manne, A.S. (1960). "On the Job-Shop Scheduling Problem". En: *Operations Research*. Vol. 8 pp. 219-223

Miltemburg, J. (1996). Estrategias de Fabricación. Productivity Press. Madrid.

Morton, T.E. and Pentico, D.W. (1993). Heuristic Scheduling Systems. Wiley Series in Engineering and Techology Management. Wiley. New York.

Pinedo, M. and Chao, X. (1999). Operations Scheduling with Applications in Manufacturing and Services. McGraw Hill. Quebec.

Pinedo, M. and Chao, X. (1999). Software LEKIN.
<http://www.stern.nyu.edu/~mpinedo>

Shi, L. (2000). "Nested Partitions for Global Optimization". En: *Operations Research*. Vol. 48. No 3.

Sipper, D. y Bulfin, R. (1997). Production Planning, Control and Integration. Mc Graw Hill.

Van Hulle, M. M. (1991). "A Goal Programming Network for Mixed Integer Linear Programming. A Case Study for the Job-Shop Problem". En: *International Journal of Neural Networks*. Vol. 2. pp. 201-209

Zhang, H.C. and Huang, S.H. (1995). "Applications of Neural Networks in Manufacturing: A state-of-the-art Survey. En: International Journal of Production Research, 33(3), pp. 705-728.