Heurísticas para secuenciamiento de tareas en líneas de flujo

Roger Z. Ríos Mercado,* Jonathan F. Bard**

En muchos ambientes de manufactura, tales como aquellos provenientes de las industrias química, farmacéutica y de procesamiento de productos, es común que las instalaciones lleven a cabo diversos tipos de tareas. En este caso, el uso de un sistema para producir, por ejemplo, diferentes compuestos químicos puede requerir algún trabajo de limpieza entre el procesamiento de tareas, y el tiempo para preparar la máquina o estación que procesa las tareas para la siguiente tarea puede depender en gran medida de la tarea predecesora inmediata. Por lo tanto, un modelo que maneje esta propiedad "dependiente de la secuencia de tareas" se convierte en crucial al encarar el problema.

En este artículo planteamos el problema de encontrar una secuencia de *n* tareas en un ambiente de línea de flujo (flow shop) de *m* máquinas, con tiempos de preparación dependientes de la secuencia que minimice el tiempo de procesamiento de todas las tareas, también conocido como *makespan* C_{max}. Matemáticamente, el problema consiste en encontrar una permutación de las tareas que resulte en un tiempo mínimo de procesamiento de todas ellos, el cual es un problema típico de las ramas de secuenciamiento de tareas y optimización combinatoria, ambas disciplinas de la investigación de operaciones. Denotaremos este problema como SDSTFS por sus siglas en inglés (sequence-dependent setup time flow shop).

El SDSTFS es un problema de los llamados "difíciles", en el sentido de que cualquier algoritmo que intente obtener una solución óptima a instancias de tamaño moderado puede tomar una cantidad exponencial de tiempo de ejecución. La mayoría de los problemas en mundo real caen en esta categoría. Esto se convierte en un asunto crucial, ya que en la práctica las personas a cargo de la toma de decisiones requieren soluciones rápidas a sus problemas. Un camino típico es el de desarrollar heurísticas, es decir, algoritmos que corren relativamente rápido y, de manera inteligente, encuentran soluciones aproximadas al problema en cuestión. Al cambiar optimalidad por rapidez, el reto está en explotar efectivamente la estructura matemática del problema para poder encontrar soluciones factibles de alta calidad.

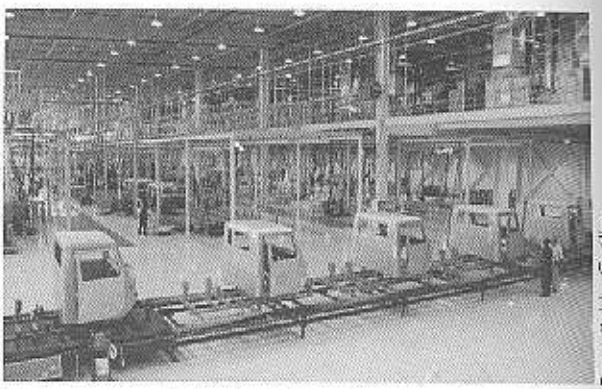

Foto: Rafaela Cuéllar

Línea de ensamblado de tractocamiones.

Nuestro trabajo incluye el desarrollo de dos heurísticas. Una de las heurísticas propuestas está basada en el trabajo de Nawaz et al.,[1] quienes han tenido éxito en problemas de secuenciamiento de líneas de flujo en general (sin considerar tiempos de preparación). Aquí extendemos su procedimiento

* Programa de Posgrado en Ingeniería de Sistemas, Universidad Autónoma de Nuevo León.
** Programa de Posgrado en Investigación de Operaciones e Ingeniería Industrial, Universidad de Texas en Austin.

para manejar los tiempos de preparación. El otro algoritmo propuesto es un procedimiento de búsqueda ávida aleatoria adaptativa (GRASP, por sus siglas en inglés), la cual es una técnica que ha sido aplicada muy exitosamente a varios tipos de problemas de optimización combinatoria. Los procedimientos propuestos son comparados con un algoritmo previamente desarrollado (SETUP) por Simons[2] para este problema. Además, desarrollamos una fase de postproceso (búsqueda local) y la adaptamos a cada heurística con el fin de mejorar la calidad de la solución. Básicamente, una búsqueda local consiste en analizar las soluciones factibles vecinas de una solución factible dada y "moverse" a una de éstas si hay una mejora en la función objetivo (tiempo total de procesamiento, en este caso). La aplicación iterativa de la búsqueda local hasta que ya no es posible mejorar la solución garantiza que se ha encontrado una solución óptima local (no necesariamente óptima global).

Los procedimientos son evaluados en una clase de instancias generadas aleatoriamente, donde los tiempos de preparación son una orden de magnitud menores que los tiempos de procesamiento (el cual es el escenario más comúnmente encontrado en la industria). Observamos que las heurísticas propuestas fueron superiores a la existente.

Antecedentes

Planteamiento del problema: El problema de línea de flujo considerado consiste en un conjunto de n tareas que deben ser procesadas a través de un conjunto de m máquinas cada una. Cada tarea tiene el mismo ruteo tecnológico a través de las máquinas. Es decir, cada tarea debe procesarse primero en la máquina 1, luego en la 2, y así hasta la máquina m. Asumimos que cada tarea está disponible inicialmente y no existe un tiempo límite de entrega. El tiempo de procesamiento de la tarea j en la máquina i se denota por p_{ij}. Se asume también que existe un tiempo de preparación que depende de la secuencia, es decir, para cada máquina i, hay un tiempo de preparación que se incurre antes de procesar cada tarea que depende de ambas, la tarea a ser procesada (k) y la tarea que la precede inmediatamente (j). Denotamos este tiempo de preparación como s_{ijk}. Las restricciones tecnológicas del sistema consisten en que una máquina no puede procesar más de una tarea a la vez. El objetivo es encontrar una secuencia (o permutación) de las tareas tal que minimice el tiempo en el cual la última tarea en la secuencia termina su procesamiento en la última máquina.

Trabajo existente: La investigación en el campo de secuenciamiento y programación de tareas (sequencing and scheduling) ha sido muy amplia. Un panorama excelente en el tema, incluyendo resultados de complejidad computacional, esquemas de optimización exacta y algoritmos de aproximación para diversos tipos de problemas de secuenciamiento puede encontrarse en el trabajo de Lawler et al.[3] Allahverdi et al.[4] presentan una completísima actualización de problemas de secuenciamiento que involucran tiempos de preparación específicamente.

En el caso particular del problema general de línea de flujo, varios estudios han mostrado (ver, por ej.[5]) que el procedimiento más efectivo es la heurística de Nawaz et al.[1] Aquí nosotros intentamos tomar ventaja de este resultado, extendiendo el procedimiento al caso que incluye tiempos de preparación. Este procedimiento se describe más adelante.

El caso especial del SDSTFS se convierte en un problema más difícil. Dada la dificultad del problema, uno tiene que hacer ciertas suposiciones sobre la naturaleza de los tiempos de preparación para tratar de simplificar el problema y desarrollar algoritmos especializados para el problema en cuestión.

La heurística más efectiva, previamente conocida para el SDSTFS, fue desarrollada por Simons.[2] Su algoritmo (SETUP) transforma una instancia del problema en una instancia del bien conocido Problema del Agente Viajero (TSP, por sus siglas en inglés) y luego trata de encontrar una secuencia factible mediante la solución del asociado TSP. Decimos entonces que SETUP es una heurística basada en el TSP. El TSP[6] es un problema clásico de optimización combinatoria, el cual se formula como sigue. Dado un conjunto de ciudades (o nodos) y de

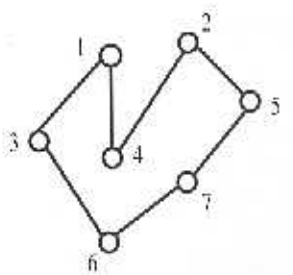

Fig. 1. Un tour en un TSP de siete ciudades.

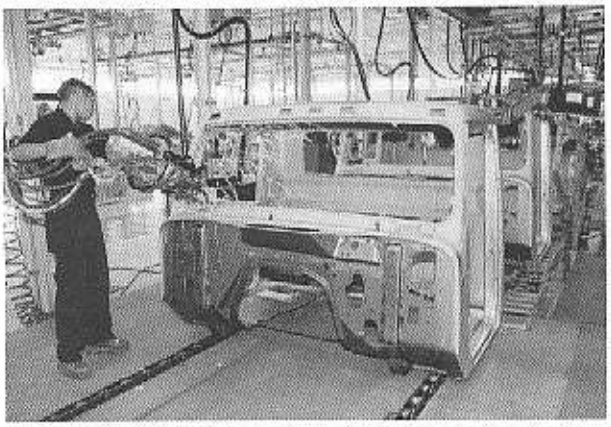

Foto: Pablo Cuéllar

Línea de ensamblado de tractocamiones.

una distancia (o costo) entre cada par de ciudades, el problema consiste en encontrar el camino (o tour) que recorra todas las ciudades una sola vez empleando una distancia mínima. La figura 1 ilustra el tour (1,4,2,5,7,6,3,1) en un grafo de siete ciudades. El TSP, aunque aparentemente sencillo de formular, es extremadamente difícil de resolver. El excelente texto de Lawler et al.[7] ilustra el amplio repertorio de técnicas de solución desarrolladas para este problema. La idea de Simons era precisamente beneficiarse de los métodos de solución del TSP para resolver el SDSTFS.

En términos de intentar resolver el SDSTFS en forma exacta existe otro tipo de trabajo.[8,9] Sin embargo, éste será expuesto en otra oportunidad ya que se trata de técnicas más sofisticadas, que emplean un tiempo de ejecución mayor y son en esencia de naturaleza muy distintas al enfoque que se presenta en este trabajo, que es básicamente en algoritmos de solución aproximada con tiempos de ejecución relativamente pequeños.

Heurísticas

Heurística NEHT-RB

Como se señaló en la sección 2, la heurística más efectiva para el problema general de línea de flujo (sin tiempos de preparación) es la de Nawaz et al.,[1] referido como el procedimiento NEH (Nawaz, Enscore y Ham). Taillard[10] desarrolló una mejor forma de implementar computacionalmente el NEH. En este trabajo, extendemos NEH para que sea capaz de manejar también tiempos de preparación. A nuestro procedimiento le llamamos NEHT-RB (NEH, modificado por Taillard y extendido por Ríos y Bard).

La idea del NEHT-RB de construir una secuencia factible es relativamente simple. En cada iteración del algoritmo, hay una secuencia parcial S. Se selecciona una tarea h de la lista de tareas P que aún no han sido programadas. La secuencia parcial S y la tarea h definen una función ávida única

$$\varphi(j) : \{0,1,\ldots,|S|\} \rightarrow \Re$$

donde $\varphi(j)$ es el tiempo de terminación de todas las tareas en la nueva secuencia S' resultante de insertar la tarea h justo después de la j-ésima tarea en S, $|S|$ denota la cardinalidad de S y $\Re$ es el conjunto de los números reales. Aquí, la posición $j=0$ significa una inserción al inicio de la secuencia. La tarea h es insertada en la posición

$$k = \text{argmin}_j\{\varphi(j)\}$$

esto es, en la posición en S que resulta en el mínimo valor del tiempo de terminación de las tareas en la secuencia parcial S'.

Procedimiento NEHT-RB

Entrada: Un conjunto $P = \{1, 2, ..., n\}$ de tareas a ser secuenciadas.

Salida:	Una secuencia factible S.
Paso 0:	Inicializar S = conjunto vacío
Paso 1:	mientras $\mid P \mid > 0$ efectuar
Paso 1-a:	Remover h, la primer tarea de P
Paso 1-b:	Calcular $\varphi(j)$ para cada posición $j = 0, 1, ..., \mid S \mid$
Paso 1-c:	Elegir $k = \text{argmin}_j\{\varphi(j)\}$
Paso 1-d:	Insertar h en la posición k en S
Paso 2:	Entregar S como salida
Paso 3:	Fin

Fig. 2. Pseudocódigo de NEHT-RB

La figura 2 muestra el pseudocódigo de NEHT-RB. Su función de complejidad computacional[11] es $O(mn^2)$, donde m y n son el número de máquinas y tareas, respectivamente. Una descripción más detallada de esta heurística puede encontrarse en[12].

Heurística GRASP

Un procedimiento de búsqueda ávida adaptativa aleatorizada (GRASP: Greedy Randomized Adaptive Search Procedure) es una metodología heurística enfocada típicamente hacia problemas de optimización combinatoria, la cual combina el po-

der de heurísticas ávidas, aleatorización y técnicas de búsqueda local. GRASP[13] ha sido aplicada exitosamente a una diversidad de problemas derivados de la optimización combinatoria.

GRASP consiste de dos fases: una fase de construcción, durante la cual se forma la solución factible y una fase de postproceso, en la cual se aplica la búsqueda local para mejorar la calidad de la solución. Durante la etapa de construcción, la solución factible es construida añadiendo un elemento a la vez. En cada iteración, la elección del siguiente elemento a ser añadido se logra ordenando primeramente el subconjunto de restantes elementos en una lista de elementos candidatos con respecto a una función ávida apropiada. Esta función evalúa el beneficio de seleccionar cada uno de los elementos. El componente probabilístico de GRASP se manifiesta al elegir a uno de los elementos de la lista de candidatos aleatoriamente, y no necesariamente el mejor. Una forma de limitar el tamaño de la lista de candidatos es por cardinalidad, donde sólo los mejores elementos λ son incluidos en la lista. Cada vez que la fase 1 de GRASP es ejecutada se obtiene una solución distinta (en este caso, una secuencia). Luego entonces, la estrategia es la de correr GRASP un número N de veces, usando diferentes semillas para el generador de números aleatorios, obteniendo así N posibles soluciones diferentes y tomando la mejor de ellas como la solución al problema.

Ahora bien, la elección del parámetro λ requiere conocimiento profundo de la estructura del problema. Tiene que existir un compromiso entre ser o no ser muy estricto. Si el criterio es muy selectivo, por ejemplo, serán muy pocos los candidatos disponibles. El caso extremo es cuando sólo un elemento es permitido en la lista restringida de candidatos, el cual corresponde a una postura puramente ávida. En este caso, GRASP reportará una sola solución perdiendo por completo la ventaja del factor aleatorio. La ventaja de ser restrictivo al formar la lista de candidatos es que no se compromete demasiado el valor de la función objetivo. Sin embargo, el riesgo de tener una baja variabilidad es que la solución óptima (y posiblemente otras muy buenas soluciones) pudieran no ser encontradas.

GRASP para el SDSTFS

La fase constructiva de GRASP es muy similar al algoritmo NEHT-RB. La diferencia entre éstos está en la forma en que eligen la posición de la secuencia parcial, donde la siguiente tarea será insertada. En GRASP, las posiciones disponibles para inserción son ordenadas por valores no decrecientes de $\varphi(j)$ y se forma la lista restringida de candidatos (LRC) con las mejores λ posiciones. La estrategia probabilística de GRASP selecciona una de estas posiciones al azar con igual probabilidad. La tarea h es entonces insertada en la posición elegida en S. La figura 3 muestra el pseudocódigo de la fase 1 del procedimiento. La fase 1 de GRASP corre en tiempo $O(mn^2)$. Una descripción más detallada puede encontrarse en [12]. Nótese que el caso extremo de GRASP cuando $\lambda = 1$ corresponde a una estrategia puramente ávida, es decir, GRASP se reduce a NEHT-RB.

Procedimiento GRASP-Fase-1

Entrada:	Un conjunto $P = \{1, 2, ..., n\}$ de tareas a ser secuenciadas y λ, el tamaño de la lista restringida de candidatos.
Salida:	Una secuencia factible S.
Paso 0:	Inicializar S = conjunto vacío
Paso 1:	mientras $\|P\| > 0$ efectuar
Paso 1-a:	Remover h, la primer tarea de P
Paso 1-b:	Calcular $\varphi(j)$ para cada posición $j = 0, 1, ..., \|S\|$
Paso 1-c:	Construir la LRC con las mejores λ posiciones
Paso 1-d:	Elegir aleatoriamente una posición k de LRC
Paso 1-e:	Insertar h en la posición k en S
Paso 2:	Entregar S como salida
Paso 3:	Fin

Fig.3. Pseudocódigo de fase constructiva de GRASP

Fase de postproceso

En general, una solución construida por un algoritmo de aproximación no es necesariamente una solución óptima local. Entonces, es conveniente aplicar una fase de postproceso donde se intenta fundamentalmente mejorar la actual secuencia S, mediante la inspección de su "vecindad" (el conjunto de secuencias que están cercanas a S), moviéndose a una secuencia S' con un mejor valor de la función objetivo (tiempo total de procesamiento). Esta nueva secuencia S' define a su vez una vecindad diferente y el proceso se repite iterativamente hasta que ya no sea posible encontrar una mejor solución. En este caso, decimos que hemos alcanzado un óptimo local (con respecto a la vecindad considerada). La búsqueda local depende de cómo se defina la vecindad de la solución factible dada.

En este trabajo aplicamos una vecindad llamada reinserción de cadenas de tareas, la cual consiste en soluciones vecinas que se obtienen de la actual al remover cadenas de dos o tres tareas y reinsertarlas en diferente posición en la secuencia.

Trabajo experimental

El propósito del experimento es el de evaluar el comportamiento de cada heurística mediante un análisis comparativo. Para medir la calidad de las soluciones obtenidas es necesario saber qué tan lejos o cerca se encuentra la solución de la solución óptima. Como no se conoce la solución óptima, usamos en su lugar una cota inferior, lo cual nos provee de una sobreestimación del error. Esto nos garantizo que el verdadero margen de error con respecto a una solución óptima (desconocida) es aún menor. La cota inferior es construida mediante un procedimiento expuesto detalladamente en la referencia[9]. A la diferencia relativa entre una solución factible y una cota inferior se le llama margen de optimalidad relativa y se calcula de la siguiente forma:

$$\frac{(\text{mejor solución factible} - \text{mejor cota inferior})}{\text{mejor cota inferior}} \times 100\%$$

Por ejemplo, si en un problema dado, el algoritmo encuentra una solución factible con valor 250, y la mejor (más alta) de las cotas inferiores conocidas es 200, entonces el margen de optimalidad es 25% ((250-200)/200 x 100%).

Los parámetros usados en GRASP son $\lambda = 2$ y N=50 iteraciones de fase 1. Es decir, se aplica la fase 1 N veces, generando N secuencias. La fase de postproceso se aplica posteriormente, a la mejor de estas N soluciones. Los procedimientos fueron codificados en lenguaje C++ y compilados con el compilador CC de Sun versión 2.0.1, usando la opción de compilación –O en una Sparc Station 10. Los tiempos de ejecución (CPU) fueron obtenidos con la función clock().

Para evaluar los procedimientos, se generaron aleatoriamente 20 instancias del SDSTFS para cada combinación

$$m \times n \in \{(2,4,6) \times (20,50,100)\}$$

con los tiempos de procesamiento p generados de acuerdo a una distribución de probabilidad uniforme en el intervalo [1,99] y los tiempos de preparación s_{ijk} generados en el intervalo [1,10]. Una evaluación más extensa de estas heurísticas puede encontrarse en la referencia [12]

Por cada combinación $m \times n$ efectuamos varias comparaciones:

(a) Resumen de estadísticas: Intentamos identificar características dominantes mediante la compilación de estadísticos como: número de veces que una heurística encuentra la mejor solución, porcentaje promedio de margen de optimalidad, tiempo de ejecución promedio y peor tiempo de ejecución.

(b) Prueba de Friedman: Esta es una prueba no paramétrica, análoga a la prueba clásica de análisis de varianza, en la cual aplicamos la hipótesis nula de que todas las heurísticas tienen la misma calidad contra la hipótesis alternativa de que al menos una de las heurísticas no tiene igual calidad a las restantes.

(c) Prueba de Wilcoxon: Si la prueba de Friedman es significativa, esto es, la hipótesis nula es rechazada, intentamos identificar la "mejor" heurística mediante la realización de una prueba de pares entre las heurísticas. Aplicamos la prueba de rangos con signo de Wilcoxon, una prueba estadística no paramétrica conocida, para comparar dos de las tres heurísticas. Para una mayor información sobre estas pruebas de Friedman y Wilcoxon, véase Conover.[14]

El resumen de estadísticas concernientes al valor de la función objetivo se muestra en la Tabla I. Las heurísticas se identifican por sus iniciales (N, S y G). Para cada celda, los números en la primera fila (Mejor) indican el número de veces que cada heurística encontró la mejor solución. Los números en la segunda fila (MO) muestran el margen promedio de optimalidad. Lo primero que notamos es que la diferencia entre los valores de la solución obtenida por los algoritmos es relativamente pequeña. Aunque GRASP parece dominar en la mayoría de los casos, su mejoría con respecto a NEHT-RB es marginal. Sin embargo, ambos procedimientos dominan a SETUP ampliamente, con la excepción de los casos de dos máquinas, donde puede observarse que SETUP mejora conforme aumenta el número de tareas.

Las estadísticas de tiempo de ejecución se pre-

Tabla 1. Comparación de heurísticas

m	Estadística	$n = 20$			$n = 50$			$n = 100$		
		S	N	G	S	N	G	S	N	G
2	Mejor	0	3	17	5	2	14	14	3	5
	MO promedio (%)	2.6	2.1	1.4	1.2	1.4	1.1	0.8	1.2	1.0
4	Mejor	0	2	18	1	3	16	1	1	18
	MO promedio (%)	9.1	7.0	6.1	4.3	3.7	3.2	3.5	3.2	2.7
6	Mejor	1	4	15	0	2	18	0	2	18
	MO promedio (%)	17.7	14.1	13.2	8.4	6.8	6.1	6.2	5.1	4.7

Tabla 2. Estadísticos de tiempo de ejecución

m	Estadística	$n = 20$			$n = 50$			$n = 100$		
		S	N	G	S	N	G	S	N	G
2	Promedio	0.1	0.1	2.5	1.9	1.2	18.4	12.6	8.9	114.3
	Peor	0.4	0.3	2.8	3.1	2.1	22.7	20.5	14.2	149.3
4	Promedio	0.4	0.20	4.1	4.2	2.6	33.3	23.6	18.9	219.0
	Peor	0.8	0.4	4.9	8.5	5.1	37.5	45.9	36.6	265.5
6	Promedio	0.5	0.3	6.0	7.0	3.6	49.5	43.1	30.2	328.8
	Peor	0.9	0.6	6.7	13.6	6.7	60.2	67.7	52.2	437.9

Tabla 3. Resultados de la prueba de Wilcoxon

	$n = 20$	$n = 50$	$n = 100$
2	GRASP ($p < 0.0004$)	GRASP ($p < 0.0444$)	SETUP ($p < 0.0149$)
4	GRASP ($p < 0.0011$)	GRASP ($p < 0.0011$)	GRASP ($p < 0.0006$)
6	GRASP ($p < 0.0005$)	GRASP ($p < 0.0011$)	GRASP ($p < 0.0004$)

sentan en la Tabla 2. Como puede apreciarse, NEHT-RB y SETUP toman, en promedio, aproximadamente la misma cantidad de tiempo. Ambos, a su vez, son como tres veces más rápidos que GRASP, para los parámetros utilizados. También puede observarse que SETUP tiene el mejor comportamiento que NEHT-RB, con respecto al peor tiempo de ejecución.

Para aquellas celdas, donde la prueba de Friedman fue significativa, se efectuó la prueba de Wilcoxon entre cada par de heurísticas. Estos resultados se tabulan en la Tabla 3, donde cada celda muestra la heurística que se encontró estadísticamente superior y el nivel de significancia p de la prueba. Como puede verse, todas las pruebas fueron significativas con nivel de confidencia $\alpha = 0.05$. El procedimiento SETUP fue mejor en el caso de 2 máquinas y 100 tareas, mientras que GRASP fue superior en todos los demás casos.

Conclusiones

En este trabajo hemos presentado dos heurísticas para el problema de minimización de tiempo máximo de ejecución, en líneas de flujo con tiempo de preparación dependientes de la secuencia. Ambos procedimientos fueron evaluados computacionalmente y comparados con la heurística de Simons, reportada como la más exitosa anteriormente para esta clase de problemas de secuenciamiento.

Para la evaluación de los algoritmos se utilizó una clase de instancias generadas aleatoriamente, donde los tiempos de preparación son pequeños con respecto a los tiempos de procesamiento de las

tareas, lo cual es una característica común en la industria.

Nuestro estudio computacional reveló que:

(a) La calidad de las heurísticas es bastante buena, lo cual se refleja en los pequeños valores de intervalos de optimalidad encontrados.

(b) La heurística de Simons (basada en el TSP) trabajó mejor en las instancias de 2 máquinas; sin embargo al crecer el número de máquinas, NEHT-RB y GRASP (basadas en inserción) tendieron a dominar. Este comportamiento proviene del hecho de que mientras menor sea el número de máquinas, mayor será la similitud con el TSP y, por ende, uno esperaría lógicamente que una heurística basada en el TSP produzca mejores resultados. Recordamos al lector que en SETUP, la "distancia" entre tareas se calcula como la suma de los tiempos de preparación entre tareas en todas las máquinas. En el caso extremo donde hay una sola máquina, el problema se reduce totalmente a una instancias del TSP. Al comenzar a considerar un número mayor de máquinas, esta suma de tiempos de preparación se vuelve menos representativa de la "distancia" entre las tareas.

(c) No se encontró una diferencia estadística significativa entre NEHT-RB y GRASP en términos del valor de la solución obtenida; sin embargo, la primera corre considerablemente más rápido.

El presente estudio provee al practicante de una mejor manera de encontrar secuencias de alta calidad, cuando éste encara escenarios de líneas de flujo donde los tiempos de preparación son pequeños comparados con los tiempos de procesamiento. Los procedimientos NEHT-RB y GRASP proveen rápidamente soluciones muy cercanas al óptimo para esta clase de instancias.

Agradecimientos

La investigación del primer autor fue apoyada por el Consejo Nacional de Ciencia y Tecnología y por una beca E. D. Farmer, otorgada por la Universidad de Texas, en Austin. El segundo autor fue apoyado por el Programa de Investigación Avanzada del Consejo Coordinador de Educación Superior de Texas. Igualmente, expresamos nuestro agradecimiento a dos árbitros anónimos, cuyas observaciones ayudaron a mejorar la exposición del artículo.

Resumen

En este trabajo presentamos dos heurísticas para el problema de minimizar el tiempo máximo de procesamiento de las tareas, en un ambiente de manufactura de línea de flujo con tiempos de preparación. Uno de los procedimientos propuestos es una extensión de un algoritmo que ha sido muy exitoso para el problema general de línea de flujo (sin tiempos de preparación). El otro es un procedimiento de búsqueda ávida aleatoria adaptativa, la cual es una técnica que ha resuelto exitosamente varios tipos de problemas de optimización combinatoria. Ambos procedimientos se evalúan y comparan computacionalmente con una heurística previamente desarrollada para este tipo de problema de secuenciamiento de tareas. Se observa que los procedimientos propuestos son superiores a la heurística existente, en relación a la calidad de la solución encontrada.

Palabras clave: Investigación de operaciones, sistemas de manufactura, secuenciamiento de líneas de flujo, tiempos de preparación, heurísticas.

Abstract

In this work we present two heuristics for the flow-shop machine scheduling problem with setup times and makespan minimization criteria. One of the proposed procedures is an extension of an algorithm that has been very successful for the general flow-shop scheduling problem. The other is a greedy randomized adaptive search procedure (GRASP) which is a technique that has successfully addressed many kinds of combinatorial optimization problems. Both procedures are compared to a previously developed algorithm. It is observed that the proposed procedures outperform the existing heuristic.

Key words: operations research, manufacturing systems, flow-shop scheduling, sequence-dependent setup times, heuristics, GRASP.

Referencias

1. M. Nawaz, E. E. Enscore Jr. e I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91-95, 1983.

2. J. V. Simons Jr. Heuristics in flow shop scheduling with sequence dependent setup times. Omega, 20(2):215-225, 1992.
3. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan y D. Shmoys. Sequencing and scheduling: Algorithms and complexity. En S. S. Graves, A. H. G. Rinnooy Kan y P. Zipkin, editores, *Handbook in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, 445-522. North Holland, New York, 1993.
4. A. Allahverdi, J.N.D. Gupta y T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2): 219-239, 1999
5. S. Turner y D. Booth. Comparison of heuristics for flow shop sequencing. *Omega*, 15(1):75-85, 1987.
6. J. L. González Velarde y R. Z. Ríos Mercado. Investigación de operaciones en acción: Aplicación del TSP en problemas de manufactura y logística. *Ingenierías*, 2(4):18-23, 1999.
7. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan y D. B. Shmoys, editores. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 1990.
8. R. Z. Ríos-Mercado y J. F. Bard. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers & Operations Research*, 25(5):351-366, 1998.
9. R. Z. Ríos-Mercado y J. F. Bard. A branch-and-bound algorithm for permutation flow shops with sequence dependent setup times. *IIE Transactions*, 31(8):721-731, 1999.
10. E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65-74, 1990.
11. M. R. Garey y D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
12. R. Z. Ríos-Mercado y J. F. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110(1):76-98, 1998.
13. A. Díaz, F. Glover, H. M. Ghaziri, J. L. González, M. Laguna, P. Moscato y F. T. Tseng. *Optimización Heurística y Redes Neuronales*. Editorial Paraninfo, Madrid, 1996
14. W. Conover. *Practical Nonparametric Statistics*. Wiley, New York, 1980.