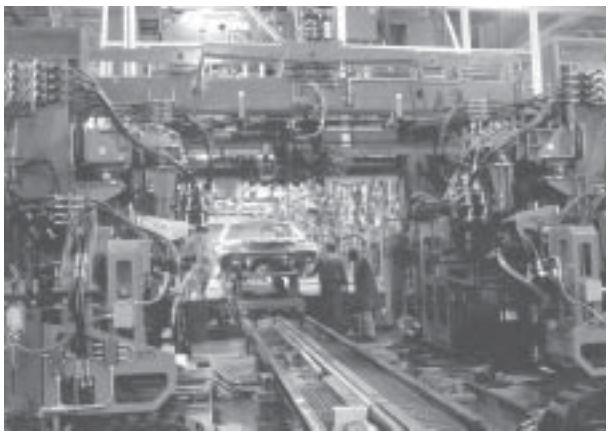
Secuenciando óptimamente líneas de flujo en sistemas de manufactura□

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 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.

Propiedades de dependencia de secuencia son factores relevantes en otros campos. Por ejemplo, la programación de un aeroplano, llegando o saliendo del área en la terminal, puede modelarse como un problema de secuenciamiento de tareas en una máquina. Debido a que las separaciones de tiempo entre aviones sucesivos pertenecientes a diferentes categorías tienen que ser cambiados de acuerdo a su respectiva posición, los tiempos de preparación dependientes de la secuencia deben ser tomados en cuenta para una descripción más real del problema.[1]

En este artículo planteamos el problema de encontrar una secuencia de *n* tareas, en un ambiente de línea de flujo (flow shop) de *m* máquina, 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 ellas, problema típico de la rama de optimización combinatoria. Denotaremos este problema como SDSTFS por sus siglas en inglés (sequence-dependent setup time flow shop).



Este problema, como muchos otros en el campo de secuenciamiento de sistemas de manufactura, es sumamente difícil de resolver, ya que está clasificado técnicamente como NP-difícil. Sin entrar en detalles técnicos, se dice que un problema es NP-difícil cuando se demuestra que cualquier algoritmo de solución tiene un tiempo de ejecución que aumenta, en el peor de los casos, exponencialmente con el tamaño del problema. Véase Garey y Johnson[2] para un tratado más amplio del tema de complejidad.

□ El presente artículo está basado en la investigación «Secuenciando óptimamente líneas de flujo en sistemas de manufactura», galardonada con el Premio de Investigación UANL 1999, en la categoría de Ciencias Exactas, otorgado en sesión solemne del Consejo Universitario de la UANL, en septiembre de 2000.

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

dad computacional. El que un problema esté catalogado como NP-difícil no significa que no pueda resolverse, sino que uno debe de proponer algoritmos de solución que exploten favorablemente la estructura matemática del problema, para que sean capaces de resolver la mayoría de las instancias del problema, en tiempos de ejecución relativamente pequeños. Ese es el reto.

El objetivo de este artículo es presentar una técnica de optimización exacta para el SDSTFS, basado en la metodología de ramificación y acotamiento, la cual enumera inteligentemente, de forma implícita, todas las posibles soluciones. Nuestro trabajo incluye la derivación y desarrollo de los componentes esenciales de esta metodología incluyendo procedimientos de acotamiento inferior, acotamiento superior y eliminación por dominio.

Es importante destacar la marcada importancia que tiene el desarrollo de cotas inferiores en problemas de optimización. Una buena cota, al emplearse dentro de un método enumerativo de optimización exacta, como ramificación y acotamiento, puede producir soluciones óptimas o cercanas al óptimo en tiempos relativamente pequeños, ya que ésta ayuda a eliminar un número muy grande de soluciones factibles. En contraste, una cota inferior de pobre calidad tiene prácticamente un impacto nulo, ocasionando que el algoritmo de optimización termine por examinar un número exponencial de soluciones factibles antes de llegar al óptimo, lo cual implica, en consecuencia, un tiempo de ejecución relativamente grande. El arte de saber derivar y generar cotas inferiores de buena calidad es una de las áreas de investigación de mayor importancia en el campo de optimización.

Como se demuestra en la evaluación numérica, el algoritmo propuesto representa un avance significativo y contundente al estado del arte, ya que éste es capaz de resolver instancias del problema de tamaño mucho mayor que las instancias que habían sido resueltas antes con otros métodos. Una de las razones primordiales para el éxito de nuestro algoritmo es que fuimos capaces de desarrollar una cota inferior que supera notablemente a las cotas inferiores desarrolladas con anterioridad por otros investigadores.

Trabajo relacionado

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, se encuentran 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.

Hasta donde sabemos, no se han desarrollado a la fecha métodos efectivos para resolver óptimamente el SDSTFS. Algunos esfuerzos por resolver el problema han sido realizados por Srikar y Ghosh,[5] y por Stafford y Tseng[6] en términos de resolver formulaciones de programas enteros mixtos. Srikar y Ghosh introdujeron una formulación que requiere sólo la mitad de variables enteras que las requeridas por la formulación tradicional. Usaron este modelo y el optimizador de programas enteros mixtos de SCICONIC/VM (basado en ramificación y acotamiento) para resolver varias instancias del SDSTFS. La instancia más grande que pudieron resolver fue una de 6 tareas y 6 máquinas en aproximadamente 22 minutos de tiempo de ejecución en una computadora Prime 550.

Posteriormente, Stafford y Tseng corrigieron un error en la formulación Srikar-Ghosh, y, usando el optimizador LINDO, lograron resolver una instancia de 5 máquinas y 7 tareas en cerca de 6 horas de tiempo de ejecución en una computadora personal (PC).

En,[7-8] desarrollamos un esquema de optimización de ramificación y corte con éxito limitado. Aun cuando encontramos que el algoritmo ahí propuesto arrojaba mejores resultados que los publicados antes, fuimos todavía incapaces de resolver (o proveer un buen juicio de la calidad de las soluciones en términos del intervalo de optimalidad) instancias de tamaño moderado. La de mayor tamaño que pudimos resolver fue una de 6 máquinas y 8 tareas en 60 minutos de tiempo de ejecución en una Sun Sparcstation 10. Otros trabajos se han enfocado en heurísticas (algoritmos de soluciones aproximadas),[9,10,11] y variaciones del SDSTFS. Véase[4] para un extenso panorama.

Formulación del problema

En un ambiente de línea de flujo, se tiene un conjunto de *n* tareas que deben ser secuenciadas en un conjunto de *m* máquinas, donde cada tarea tiene el

mismo orden de ruteo a través de las máquinas, es decir, cada una de las tareas debe ser procesada primero en la máquina 1, luego en la 2, y así sucesivamente hasta la máquina *m.* Se asume también que la secuencia de tareas en cada máquina es la misma y que cada máquina puede procesar una sola tarea a la vez. Dichas secuencias se denominan técnicamente *secuencias de permutación*. Las tareas están todas disponibles al inicio del proceso (tiempo cero) y no tienen tiempo límite de terminación o entrega. El tiempo de procesamiento de la tarea *j* en la máquina *i* se denota por p_{ij}. Asumimos también que hay un tiempo de preparación de la máquina y que éste depende del orden de la secuencia, de tal modo que para cada máquina *i* hay un tiempo de preparación que debe preceder al inicio del procesamiento de una tarea dada que depende de ambos, la tarea a ser procesada (*k*) y la tarea que la precede (*j*). Este tiempo de preparación es denotado por s_{ijk}. Nuestro objetivo es encontrar una secuencia de tareas que minimice el tiempo de terminación de todas las tareas (también conocido como *makespan*), o equivalentemente, el tiempo de terminación de la última tarea en la secuencia en la última máquina. La función que desea minimizarse se le conoce como *función objetivo*. En la literatura

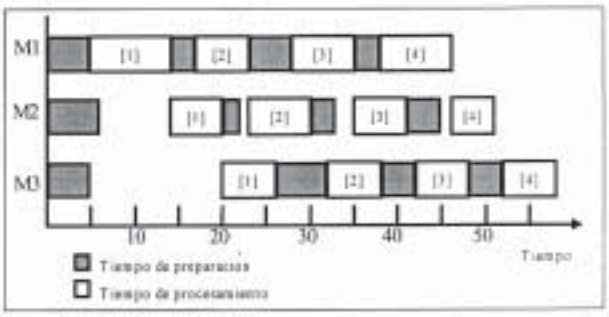


Fig. 1. Ejemplo de una secuencia de 4 tareas en una línea de flujo de 3 máquinas

de secuenciamiento, este problema se denota por SDSTFS (sequence-dependent setup time flow shop). Para un tratado más amplio de problemas de secuenciamiento véase el excelente texto de Pinedo.[2] La figura 1 ilustra un ejemplo de 4 tareas y 3 máquinas, donde el makespan, para la secuencia mostrada es 58.

Metodología de solución

El conjunto de soluciones factibles del SDSTFS se representa, desde un punto de vista combinatorio, como X = { conjunto de todas las posibles secuencias de *n* tareas }. Este conjunto es finito por lo que una solución óptima puede obtenerse por un método que enumere todas las posibles soluciones en X y reporte la que tenga el valor mínimo de la función objetivo. Este tipo de metodología es llamada enumeración. Sin embargo, una enumeración completa de todos los elementos de X es difícilmente práctica, porque el número de soluciones a ser consideradas es típicamente enorme. En este caso, para *n* tareas, el número total de posibles secuencias es *n*!, donde , es un número que crece exponencialmente. Por lo tanto, cualquier método que se jacte de ser efectivo debe ser capaz de detectar soluciones factibles dominadas por otras (es decir, que su valor en el objetivo sea peor), de tal modo que pueden ser excluidas de consideración explícita. Mientras más inteligente sea el algoritmo en detectar este tipo de relaciones, mayor será el número de soluciones factibles que podrán ser eliminadas y, por ende, menor será el tiempo de ejecución del algoritmo.

Una de las técnicas enumerativas de mayor popularidad es la de ramificación y acotamiento. Básicamente la técnica de ramificación y acotamiento consiste en un procedimiento iterativo de partición del dominio en dominios o subproblemas de menor tamaño. Al evaluar un subproblema dado (generado previamente por el proceso de partición o ramificación), en lugar de resolverlo óptimamente (lo cual requeriría de un esfuerzo computacional relativamente grande), se procede a encontrar una cota inferior del valor de la función objetivo del subproblema. Pueden suceder dos cosas: Si esta cota resulta ser mayor o igual que el valor de la mejor solución factible encontrada hasta el momento, entonces significa que en dicho subproblema no puede encontrarse una solución mejor que la que se tiene. Por consiguiente se procede a descartarlo (descartándose de paso todos sus posibles subproblemas descendientes). Ahora bien, si la cota es menor que el valor de la mejor solución factible encontrada hasta ese momento, entonces se procede a ramificarlo en subproblemas (que tendrán a éste como generador), que después serán considerados. Una vez hecho esto, el algoritmo selecciona otro subproblema de la lista que no han sido explorados. El algoritmo

termina con una solución óptima del problema, o bien, al llegar a un tiempo límite máximo permitido, en cuyo caso se reporta la mejor solución factible encontrada (no necesariamente óptima).

Un esquema de ramificación y acotamiento para problemas de minimización posee los siguientes componentes generales:

- Una *regla de ramificación*, la cual define particiones del conjunto de soluciones factibles en subconjuntos.
- Una *regla de acotamiento inferior*, la cual provee una cota inferior en valor de cada solución en el subconjunto generado por la regla de ramificación.
- Una *estrategia de búsqueda*, que selecciona uno de los subconjuntos previamente generados por la regla de ramificación y que aún no ha sido explorado.

Otros componentes adicionales como *regla de eliminación por dominio* y *procedimiento de acotamiento superior* pueden también estar presentes y, si se saben explotar adecuadamente, pueden conducir a mejoras sustanciales en el rendimiento global del algoritmo.

Los fundamentos de esta técnica, aplicada a resolver problemas de optimización, pueden encontrarse en el trabajo de Ibaraki.[13,14] Los detalles de nuestra contribución en el desarrollo de los componentes del algoritmo arriba mencionados para el SDSTFS se hallan en[15].

Trabajo experimental

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 SparcStation 10. Los tiempos de ejecución (CPU) fueron obtenidos con la función clock). Para evaluar los procedimientos, se generaron aleatoriamente instancias del SDSTFS, con los tiempos de procesamiento p_{ij} generados de acuerdo a una distribución de probabilidad discreta uniforme en el intervalo [20,100] y los tiempos de preparación s_{ijk} generados en el intervalo [20,40], lo cual es representativo de instancias reales, como ha sido documentado en[16].

En el experimento se pretende evaluar el rendimiento general del algoritmo completo de ramificación y acotamiento con todos sus componentes efectivamente integrados. Para una evaluación detallada de los componentes individuales véase[15]. Los parámetros de detención del algoritmo se prefijan a 30 minutos como límite de tiempo de ejecución e intervalo de optimalidad relativa a 1%.

El intervalo relativo de optimalidad nos indica qué tan lejos está la solución encontrada del óptimo global. En este caso como no se conoce el óptimo global, el intervalo se calcula con respecto a una cota inferior, lo cual desde luego sobreestima el error. Mientras mayor sea este intervalo, mayor es el error en la solución, naturalmente. El intervalo de optimalidad se calcula como

Tabla I. Evaluación del algoritmo

Tamaño *m* x *n*	Intervalo de optimalidad (%)			Tiempo (seg)			Instancias resueltas (%)
	Mejor	Promedio	Peor	Mejor	Promedio	Peor	
2x10	0.3	0.9	1.0	1	235	560	100
4x10	0.8	0.9	1.0	2	68	222	100
6x10	0.9	1.0	1.0	29	265	450	100
2x15	0.0	1.0	2.6	3	725	1800	70
4x15	0.9	2.2	4.5	7	1074	1800	50
6x15	1.0	2.9	4.5	38	1624	1800	10
2x20	0.5	1.0	1.6	7	1298	1800	70
4x20	2.4	4.2	5.1	1800	1800	1800	0
6x20	1.5	5.0	8.1	1800	1800	1800	0

Tabla II. Evaluación del algoritmo en instancias de 100 tareas

Tamaño *m* x *n*	Intervalo de optimalidad inicial (%)			Intervalo de optimalidad final (%)			Instancias resueltas (%)
	Mejor	Promedio	Peor	Mejor	Promedio	Peor	
2 x 100	1.2	3.4	8.4	0.6	1.4	2.1	30
4 x 100	3.3	5.1	6.5	2.3	4.2	5.7	0
6 x 100	5.0	7.6	9.4	4.3	6.0	7.2	0

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 intervalo de optimalidad es 25% ((250-200)/200 x 100%).

La Tabla 1 muestra un resumen de las estadísticas que se calcularon en base a 10 instancias para cada combinación de tamaño mostrada. Como puede apreciarse, todas las instancias de 10 tareas fueron resueltas óptimamente en un tiempo promedio de menos de 5 minutos, lo cual se convierte en una mejora impactante al compararse con el trabajo previamente publicado. Como recordamos, el tamaño de la instancia más grande que había sido resuelta anteriormente era un problema de 6 máquinas y 8 tareas. De hecho, nuestro algoritmo fue capaz de resolver óptimamente el 43% de todas las instancias de 15 tareas, y el 23% de todas las instancias de 20 tareas.

Finalmente, la Tabla II muestra el desempeño del algoritmo al aplicarse en instancias de 100 tareas. El 30% de las instancias de 2 máquinas fueron resueltas, mientras que el 70% terminaron con un intervalo de optimalidad de 1.3% o mejor, es decir, prácticamente resueltas. En general, el intervalo de optimalidad promedio desde el inicio hasta la terminación del algoritmo mejoró en un 2.0%, 0.9% y 1.6% para las instancias de 2, 4 y 6 máquinas, respectivamente.

Conclusiones

Los resultados computacionales demostraron convincentemente la efectividad del algoritmo propuesto, resultando en una mejora impactante y significativa sobre el trabajo previamente publicado. El algoritmo fue capaz de resolver (con margen de optimalidad menor al 1%) el 100%, 43% y 23% de todas las instancias probadas de 10, 15 y 20 tareas, respectivamente. Además, en el caso de instancias de 100 tareas, nuestro algoritmo produjo soluciones con intervalos de optimalidad promedio de 1.4%, 4.2% y 6.0% al aplicarse a instancias de 2, 4 y 6 máquinas.

Nuestras recomendaciones sobre trabajo a futuro se enfocan a considerar otro tipo de variaciones del problema estudiado, incorporando elementos que en ocasiones también aparecen en ciertos escenarios de manufactura. Por ejemplo, el introducir restricciones de fecha límite para la entrega de tareas restringe notablemente el conjunto de secuencias factibles. La introducción de estas fechas límite nos permite además considerar otro tipo de funciones objetivo como, por ejemplo, la minimización del número de tareas que terminan tarde, o, si existe alguna penalización por entregar tareas tarde, la minimización de la penalización total. Otra de las variaciones del problema yace en el campo probabilístico o estocástico. Con frecuencia, los parámetros del sistema están sujetos a variaciones mayores, y por tanto no puede considerarse que se conocen con certeza, tal y como se estudió en este trabajo. En consecuencia, los parámetros son en realidad variables aleatorias y como tal deben de modelarse y analizarse con métodos probabilísticos. Estas variaciones aquí mencionadas no han sido estudiadas aún y representarían, desde luego, una contribución importante en el campo de la programación y secuenciamiento de tareas en sistemas de manufactura.

Agradecimientos

La investigación del primer autor fue apoyada por el Consejo Nacional de Ciencia y Tecnología y por las becas E. D. Farmer y David Bruton, otorgadas 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.

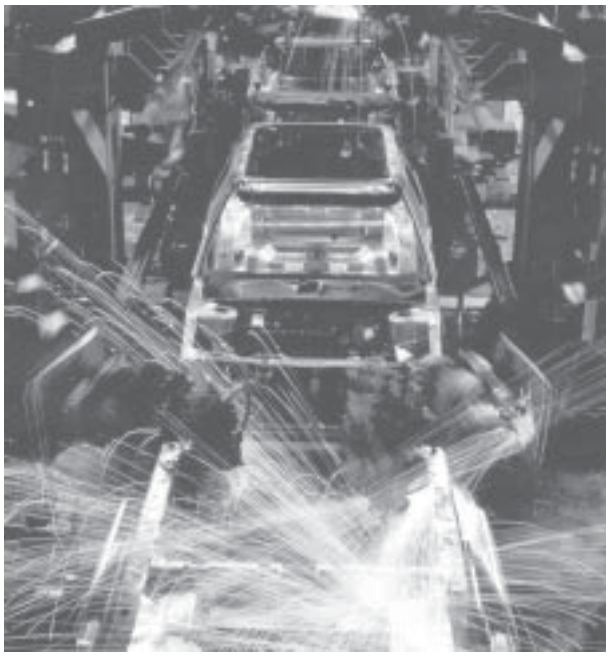
Resumen

En este trabajo se presenta un esquema enumerativo basado en la técnica de ramificación y acotamiento para resolver uno de los problemas de mayor importancia en el campo de secuenciamiento de tareas en ambientes de manufactura. El problema considerado es la minimización del tiempo de procesamiento de todas las tareas en un sistema de línea de flujo, incluyendo tiempos de preparación de máquinas dependientes de la secuencia de tareas. Este problema es clasificado técnicamente como sumamente difícil de resolver. El trabajo incluye el desarrollo e implementación de procedimientos de acotamiento superior e inferior, criterio de eliminación por dominio y estrategias inteligentes de enumeración parcial. La integración de todos estos componentes dentro de un marco de ramificación y acotamiento resulta en un algoritmo sumamente efectivo, el cual es evaluado computacionalmente en un amplio rango de instancias del problema. Los resultados demuestran la amplia superioridad del algoritmo propuesto sobre trabajo previamente desarrollado por otros investigadores.

Palabras clave: Investigación de operaciones, sistemas de manufactura, secuenciamiento de líneas de flujo, tiempos de preparación, ramificación y acotamiento.

Abstract

In this work we present an enumerative scheme based on branch -and-bound- for solving one of the most important problems in machine scheduling. The problem addressed is the flow shop scheduling problem with sequence-dependent setup times and makespan minimization criteria. This problem is classified as hard to solve. Our work includes the development of lower and upper bound schemes, dominance elimination criteria, and an intelligent partial enumeration strategy. All these components were fully integrated within a branch-and-bound frame, resulting in a very effective algorithm, which is evaluated in a wide range of problem instances. The numerical evaluation shows the overwhelming superiority of the proposed algorithm over existing approaches.



Keywords: operations research, manufacturing systems, flowshop scheduling, sequence-dependent setup times, branch and bound.

Referencias

1. R. D. Dear. The dynamic scheduling of aircraft in the near terminal area. FTL Report R76-9, Massachussetts Institute of Technology, Septiembre 1976.
2. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
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 and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega,* 27(2): 219-239, 1999.
5. B. N. Srikar and S. Ghosh. A MILP model for the *n*-job, *m*-stage flowshop with sequence dependent set-up times. *International Journal of Production Research*, 24(6):1459-1474, 1986.
6. E. F. Stafford and F. T. Tseng. On the Srikar-

Ghosh MILP model for the SDST flowshop problem. *International Journal of Production Research*, 28(10):1817-1830, 1990.

7. R. Z. Ríos-Mercado and J. F. Bard. The flowshop scheduling polyhedron with setup times. Reporte Técnico ORP96-07, Programa de Posgrado en Investigación de Operaciones, Universidad de Texas, Austin, EUA, Julio 1996.
8. R. Z. Ríos-Mercado and 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. J. V. Simons Jr. Heuristics in flow shop scheduling with sequence dependent setup times. *Omega*, 20(2):215-225, 1992.
10. R. Z. Ríos-Mercado and J. F. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110(1):76-98, 1998.
11. R. Z. Ríos-Mercado and J. F. Bard. An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times. *Journal of Heuristics*, 5(1):57-74, 1999.
12. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1995.
13. T. Ibaraki. Enumerative approaches to combinatorial optimization: Part I. *Annals of Operations Research*, 10(1-4):1-340, 1987.
14. T. Ibaraki. Enumerative approaches to combinatorial optimization: Part II. *Annals of Operations Research*, 11(1-4):341-602, 1987.
15. R. Z. Ríos-Mercado and J. F. Bard. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. *IIE Transactions*, 31(8):721-731, 1999.
16. J. N. D. Gupta and W. P. Darrow. The two-machine sequence dependent flowshop scheduling problem. *European Journal of Operational Research*, 24(3):439-446, 1986.