Algoritmo Genético para la Secuenciación de Tareas en Líneas de Flujo

Oscar Alejandro Hernández López

Resumen

La planificación de actividades constituye un factor determinante en la industria manufacturera y los servicios. Garantizar una adecuada calendarización de las mismas contribuye a elevar la eficiencia y el nivel de servicio de las empresas de cara a los clientes. En este artículo se resuelve un problema de secuenciación de tareas en líneas de flujo mediante un algoritmo genético. El mismo tiene como objetivo minimizar el tiempo total de producción de una serie de trabajos. Con los resultados computacionales obtenidos se concluye que el algoritmo se comporta similarmente para cualquiera de las instancias ejecutadas. Alcanza un 13.48 % de gap como promedio respecto a los óptimos de la literatura, aunque con una mayor dispersión en los resultados para las instancias de mayor tamaño.

Palabras clave: calendarización, secuenciación de tareas en líneas de flujo, Algoritmo Genético.

1. Introducción

Dentro de la gestión de la organización industrial el principal objetivo es optimizar el sistema en que se trabaja. Los problemas de calendarización (scheduling) están presentes en muchos casos de organización. Estos son un grupo de problemas de optimización combinatoria que juegan un papel crucial en diversas industrias tales como la manufacturera y los servicios. Se basan en la organización de tareas y/o asignación de recursos con el fin de optimizar una función objetivo que generalmente busca la minimización del tiempo. El Problema de Secuenciación de Tareas en Líneas de Flujo (PSTLF) es uno de los más populares dentro de los problemas de calendarización, los mismos se aplican en casi un cuarto de los sistemas de manufactura y líneas de ensamblaje [8, 17, 22]. En un PSTLF tradicional un trabajo no puede ser transferido a la siguiente máquina antes que su procesamiento sea finalizado [13, 14].

Los conceptos de las clases NP-duro y NP-completo [5] evidencian que muchas investigaciones se han concentrado en el diseño de algoritmos heurísticos. El problema de flow shop es conocido por ser NP-Hard [7], de ahí que se decida aplicar una metaheurística como alternativa de solución al mismo.

Este trabajo está estructurado de la siguiente manera. Primero, en la Sección 2 se muestra una breve revisión de la literatura, explicando en que consiste el problema de secuenciación de tareas en líneas de flujo, así como los principales enfoques heurísticos utilizados para resolverlo. En la sección 3 se presenta la modelación matemática del problema. En la sección 4 se muestra una descripción sobre las principales características de los algoritmos genéticos así como su funcionamiento; además de cómo es aplicado para resolver el problema en cuestión. En la sección 5 se muestran los resultados experimentales del trabajo y finalmente en la sección 6 se presentan las conclusiones de este proyecto.

2. Revisión de la Literatura

Entre los problemas de calendarización tratados en la literatura se encuentra el de Secuenciación de Tareas en Líneas de Flujo (Flow Shop) [12]. Este problema consiste en determinar una secuencia de n trabajos en un conjunto de m máquinas distribuidas en serie.

Esta es una configuración en las fábricas donde los productos comienzan a ser procesados en la máquina 1 y continúan su procesamiento hasta que son finalizados en la última máquina m.

Cada trabajo debe ser procesado secuencialmente en todas las máquinas. En esta secuencia cada trabajo j, $j = \{1, ..., n\}$ tiene un tiempo de procesamiento de p_{ij} unidades en cada máquina i, $i = \{1, ..., m\}$ [18].

Una variante en la literatura para este tipo de problemas es asumir que una vez que la secuencia de producción de los trabajos se determina en la primera máquina, esta se mantiene para el resto de las otras [18]. Es importante considerar el tiempo en que se concluye cada trabajo, denotado por C_j puesto que el objetivo para el Problema de Secuenciación de Tareas en Líneas de Flujo (PSTLF) es la minimización del máximo C_j, que sería la diferencia entre el inicio y finalización de la secuencia de trabajos o lo que se conoce como *makespan*.

Para resolver el PSTLF se han propuesto enfoques heurísticos, basados principalmente en procedimientos de mejora iterativa. Entre ellos se encuentra Osman and Potts [11], los cuales proponen un Recocido Simulado, así como Widmer and Hertz [21] y Taillard [16] una Búsqueda Tabú. Otro tipo de algoritmos también ha sido aplicado, como es el caso de los algoritmos genéticos (AG) [4, 6, 10, 15]. Para este trabajo se utiliza un algoritmo de este tipo para darle solución al PSTLF. En la siguiente sección se explica a detalle en qué consiste y cómo se aplicó en este proyecto.

3. Modelación matemática

Existen diferentes formulaciones matemáticas del problema Wagner [19], Manne [9], Fisher [3], Blazewicz et al. [2]. En este trabajo se ha adoptado el modelo propuesto por Adams et al. [1].

Sea V = $\{0, 1, ..., n\}$ el conjunto de operaciones o trabajos, donde 0 y n son consideradas como operaciones ficticias de inicio (la primera operación de todos los trabajos) y fin (última operación de todos los trabajos), respectivamente. Sea M el conjunto de m máquinas y A el conjunto de pares de operaciones ordenadas, restringidas por las relaciones de precedencia de cada trabajo. Para cada maquina m, E_m describe el conjunto de todos los pares de operaciones que no se pueden traslapar. Para cada operación i se fija un tiempo de procesamiento p_i y un inicio mas temprano posible de procesamiento de i, denotado por ti, siendo una variable a determinar durante la optimización.

El PSTLF puede ser modelado como:

$$mint_n$$

sujeto a:

$$t_j - t_i \geq p_i \forall (i,j) \in A, \tag{1}$$

$$t_j - t_i \geq p_i o t_i - t_j \geq p_j \forall (i,j) \in E_k, \forall m \in M, \tag{2}$$

$$t_i \geq 0 \forall i \in V, \tag{3}$$

La restricción 1 asegura que la secuencia de procesamiento de operaciones en cada trabajo se corresponda con el orden predeterminado. La restricción 2 demanda que exista solo un trabajo en cada máquina a la vez. La restricción 3 asegura el completamiento de todos los trabajos.

4. Algoritmo Genético

4.1. Codificación

En los algoritmos genéticos cada solución de un problema de optimzación es codificado como una cadena. La estructura se muestra en la figura 1. Cada miembro de la población se representa como un cadena binaria de longitud L que corresponde a la codificación del problema. Cada cadena se refiere al cromosoma.

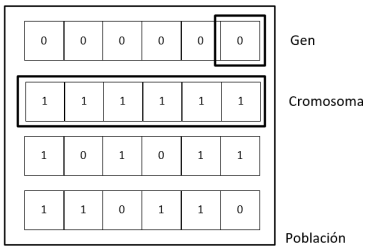


Figura 1: Estructura de un algoritmo genético

Para este problema no se codifica la solución como una cadena binaria sino como una cadena de enteros de N (número de trabajos) elementos. Por ejemplo, en la figura 2 se ilustra una secuencia para N=5. Cada gen es un número entero entre 0 y N, los cuales representan el trabajo a realizar. La secuencia en la que los trabajos son realizados está dada por el orden en que aparece cada gen en el cromosoma.

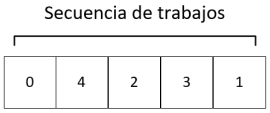


Figura 2: Cromosoma genérico

El primer paso para implementar cualquier AG es generar una población inicial. En la mayoria de los casos se genera una población inicial aleatoria, luego cada cadena es evaluada y le es asignada un valor de aptitud (*fitness*).
Normalmente las notaciones de función de evaluación y fitness se usan indistintamente, no obstante se hace necesario distinguir que la función de evaluación (objetivo) es una medida del rendimiento respecto a un conjunto de parámetros. La función de fitness transforma esa medida en una asignacion de oportunidades reproductivas.
Las oportunidades reproductivas son asignadas, de acuerdo a que aquellos cromosomas con mejores valores de la función objetivo son los que representan las mejores soluciones, le son dados más posibilidades de "reproducirse"que a aquellos con soluciones más pobres [20].
En la figura 3 se muestran los principales elementos y cómo funcionan los AG, donde i representa la inicialización, $f(x)$: función de evaluación, P: condición de parada, Se: selección, Cr: cruzamiento, Mu: mutación, Re: reemplazo y X^*: mejor solución.

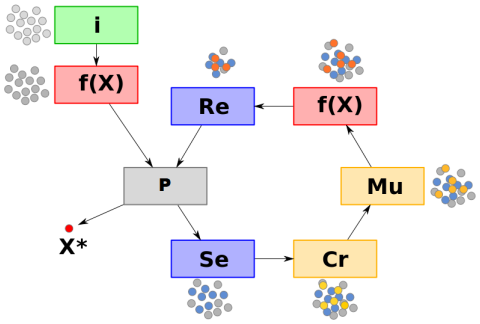


Figura 3: Funcionamiento de un algoritmo genético

El pseudocódigo que ilustra el funcionamiento de esta metaheurística se muestra en el Algoritmo 1.

4.2. Operadores genéticos

Un AG está compuesto por tres operadores, los cuales son selección, recombinación o cruzamiento y mutación. La ejecución de un AG se puede ver como un proceso de dos etapas. Comienza con la población actual (current), luego la selección es aplicada a dicha población para crear la población intermedia, luego la recombinación y la mutación son aplicadas a esta población intermedia para crear la próxima población [20].

4.2.1. Selección

Existen diferentes maneras de realizar la selección. Se podría ver la población como una ruleta donde cada individuo es representado por un espacio que corresponde proporcionalmente a su fitness. Al girar repetidamente la ruletra, los individous son elegidos usando muestras estocásticas con reemplazo para completar la población intermedia.
En los problemas de maximización el método usual de selección es la medida del fitness relativo dado por el ratio entre el valor de un cromosoma determinado entre la media de la población. Sin embargo en los problemas de minimización ocurre lo contrario, por lo que debemos

input : Instancia de trabajos a secuenciar
output: Una solución factible de la calendarización como una permutación de trabajos

1 ***Inicialización:*** *Generar aleatoriamente una población inicial* P_1 *de* N_{pop} *trabajos* (N_{pop} *soluciones)*;
2 ***Selección:*** *Seleccionar* N_{pop} *pares de soluciones de la población actual de acuerdo a la probabilidad de selección*;
3 ***Cruzamiento:*** *Aplicar cruzamiento a cada par seleccionado en el Paso 2 para generar* N_{pop} *soluciones*;
4 ***Mutación:*** *Aplicar mutación a cada una de las* N_{pop} *soluciones generadas*;
5 ***Actualización elitista:*** *Remover aleatoriamente una solución de la población actual y añadir la mejor solución de la población anterior a la actual*;
6 **if** *número de iteraciones es alcanzado* **then**
7 | PARAR;
8 **else**
9 | Regresar al paso 2
10 **end**

Algorithm 1: Algoritmo Genético para el PSTLF

modificar la condición para que los cromosomas con bajos valores sean los "buenos".
Según Reeves [15] para efectuar esta selección primero se ordenan los cromosomas. Luego propone que la selección de los padres se realice de acuerdo a la siguiente distribución de probabilidad:

$$\frac{2k}{M(M+1)}$$

donde k representa el $k-esimo$ cromosoma en orden ascendente de fitness (en este caso de scheduling, orden descendente de makespan). Esto implica que el valor de la mediana tiene una probabilidad de $\frac{1}{M}$ de ser seleccionado, mientras que el $M-esimo$ (fittest) tiene una probabilidad de $\frac{2}{(M+1)}$, aproximadamente el doble que la mediana.

4.2.2. Recombinación o Cruzamiento

La recombinación es una operación la cual genera una nueva cadena, denominada hijo, a partir de dos cadenas, las cuales se consideran como padres. En Murata et al. [10] se proponen 10 operadores de recombinación, los cuales fueron probados mediante simulaciones para el PSLF y se encontró que el operador de recombinación que mejor se comporta para las instancias generadas es la primera versión de cruzamiento. El mismo es el que se utliza para dar solución al problema de este artículo. Este cruzamiento se describe en la figura 4:

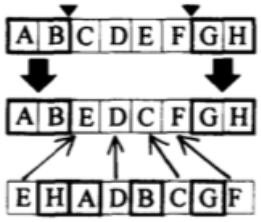


Figura 4: Recombinación del algoritmo genético

En la figura el primer y tercer cromosoma representan el primer y tercer padre respectivamente. Cada gen representa un trabajo y cada cromosoma la secuencia al realizarlos. El segundo cromosoma representa el hijo correspondiente al cruzamiento. En esta recombinación dos pares de dos genes son seleccionados al azar, por ejemplo el gen A y gen B conformarían el primer par, mientras que el gen G y el gen H conformarían el segundo par seleccionado en el primer padre. Estos genes son heredados al hijo, colocándolos en la misma posición. Los trabajos que quedan fuera de la selección de ambos pares son heredados al hijo en el orden en que aparecen ubicados en el segundo padre.

4.2.3. Mutación

La mutación es la operación de cambiar el orden de n trabajos en cada solución generada por el operador de cruzamiento. Esta operación de mutación puede ser vista como una transición de una solución actual a una solución de su vecindario en algoritmos de búsqueda local.

En Murata et al. [10] fueron simulados cuatro diferentes operadores de mutación para el PSLF. En el artículo se muestra que el operador con mejor desempeño es el que llaman *Shift change*. Este consiste en remover un trabajo de una posición y colocarlo en otra como se muestra en la figura 5. Las posiciones son seleccionadas aleatoriamente. Esta mutación es aplicada a cada hijo con una probabilidad $P_m = 1$.

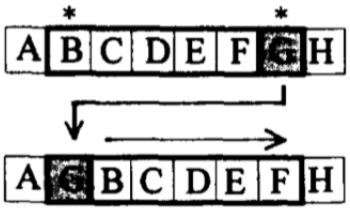


Figura 5: Operador de mutación Shift change

4.2.4. Estrategia elitista

Como se aaplican todas los operadores a N_{pop} pares de cadenas (padres) se obtienen N_{pop} numero de hijos. Como último paso, en la estrategia elitista se remueve aleatoriamente una cadena de la población actual y se añade la mejor cadena de la población anterior a la actual. Luego continúa el proceso con esta nueva población generada.

4.2.5. Terminación

Un número total de evaluaciones (generaciones) se usó como condición de parada. Para concluir el algoritmo se especificaron 10 000 evaluaciones.

5. Resultados Computacionales

Como referencia se tomaron las instancias proporcionadas por Vallada et al. [18], las mismas constan de 240 instancias grandes y 240 pequeñas. Las instancias pequeñas siguen las siguientes combinaciones de n trabajos y m máquinas: $n = \{10, 20, 30, 40, 50, 60\}$, $m = \{5, 10, 15, 20\}$. Diez instancias son generadas para cada combinación por lo que en total serían $6 \times 4 \times 10 = 240$ en total. Nótese que las instancias pequeñas llegan hasta 60 trabajos y 20 máquinas.

Para el caso de las instancias grandes también se tienen 240, con $n = \{100, 200, 300, 400, 500, 600, 700, 800\}$ trabajos y $m = \{20, 40, 60\}$ máquinas. También se generan diez instancias para cada combinación, por lo que se tienen $8 \times 3 \times 10 = 240$ en total .

Cada instancia se encuentra en un documento de texto siguiendo la misma estructura: la primera fila del archivo indica el número de trabajos seguido del número de máquinas. Luego, una matriz con los tiempos de procesamiento para cada trabajo en cada máquina es mostrada. Estos tiempos de procesamiento fueron generados siguiendo uan distribución uniforme entre 1 y 99 [18].

Cada instancia fue ejecutada una sola vez por el algoritmo. Como parámetros se establecieron una probabilidad de cruzamiento $P_c = 1$ y una probabilidad de mutación $P_m = 1$. Estos parámetros fueron escogidos en base a los resultados arrojados en Murata et al. [10]. Los otros parámetros que se fijaron fue una población de 100 individuos y 25 generaciones. Para justificar la selección de los mismos, primeramente decir que lo recomendado sería realizar un diseño de experimentos. Este no fue realizado en este trabajo, por lo que se prevee que estos dos parámetros no sean los que permitan explotar al máximo las potencialidades del AG. A su vez, se realizó una prueba con la instancias más retadoras y se fijó una población de 100 individuos. A partir de experimentaciones con el objetivo de investigar en qué punto convergía el algoritmo a su mejor solución, se encontró que

a partir de la generación 25 el AG no mejoraba la función objetivo. Por esta razón se concluyó que para este tamaño de población sólo 25 generaciones hacen falta para obtener su mejor resultado.
Los resultados que se obtienen al ejecutar el algoritmo con las instancias descritas anteriormente se muestran en la figura 6. Para las instancias más grandes se obtiene un gap promedio de 14.95 % mientras que para las instancias pequeñas el gap promedio es de 12.00 %, aunque estas últimas con mucha mayor dispersión.

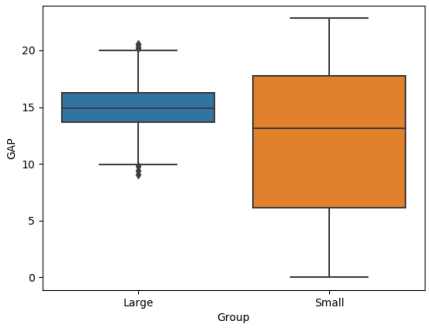


Figura 6: Diagrama de caja de gap vs. tamaño de instancias

Un análisis similar se realizó para cada grupo de instancias (grandes y pequeñas), por cada configuración trabajos-máquinas. Los resultados se muestran en la Figura 7. El eje nxm indica la configuración, donde n se refiere a la cantidad de trabajos y m a la cantidad de máquinas. Se puede concluir que las instancias con 10 trabajos son las que mejor se comportan en cuanto a valor de la función objetivo, al presentar un menor gap. También se puede concluir que para todas las instancias pequeñas la configuración con menor gap es la que tiene 5 máquinas, sin importar la cantidad de trabajos.
Los resultados de las instancias grandes se muestran en la figura 8. Aquí la principal conclusión a la que se puede arribar es que en el segundo subgrupo la configuración con la que se obtienen mejores resultados en cuanto a la función objetivo es con la cantidad de 20 máquinas.
Otro factor que se mide es el tiempo de ejecución para cada tamaño de instancia. El mismo se muestra en la figura 9. Para ello se utilizó un diagrama de dispersión. En el mismo se llega a la conclusión que los tiempos de ejecución para las instancais pequeñas se comportan de manera lineal, no siendo así para el caso de las grandes instancias.

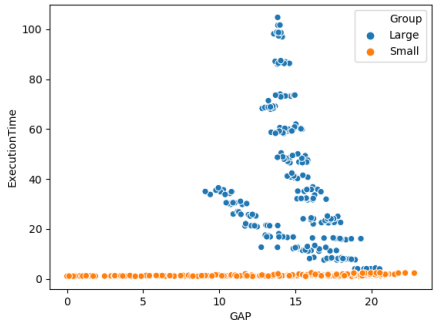
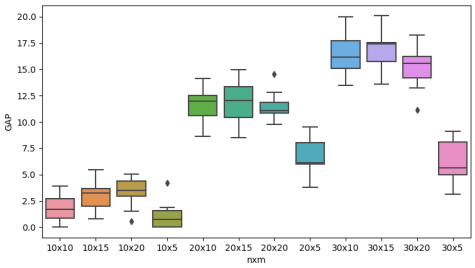


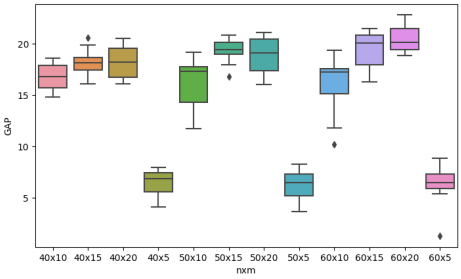
Figura 9: Diagrama de dispersión de gap vs. tiempo de ejecución

6. Conclusiones

En este trabajo se analizó el rendimiento de un algoritmo genético para el problema de secuenciación de tareas en líneas flujo. Para los parámetros fijados se obtuvo con las instancias probadas resultados de 13.48 % de gap como promedio respecto a los óptimos de la literatura. Como trabajo futuro queda realizar un diseño de experimentos para fijar los parámetros de tamaño de población y cantidad de generaciones para obtener mejores resultados. También se necesita ejecutar cada una de las 480 varias veces para de esta manera tener una mejor medida de la variablidad de los resultados particulares para cada instancia en las que se ejecuta el algoritmo.

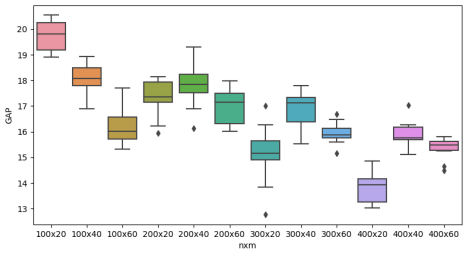


(a) Subgrupo 1 de Instancias pequeñas

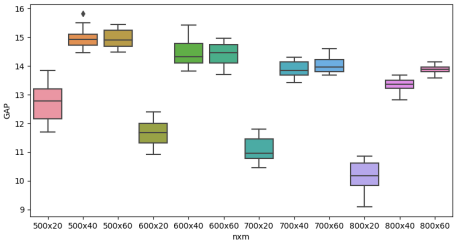


(b) Subgrupo 2 de Instancias pequeñas

Figura 7: Diagrama de caja de gap vs. configuración



(a) Subgrupo 1 de Instancias grandes



(b) Subgrupo 2 de Instancias grandes

Figura 8: Diagrama de caja de gap vs. configuración

Referencias

[1] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[2] Jacek Blazewicz, Moshe Dror, and Jan Weglarz. Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research*, 51(3):283–300, 1991.

[3] Marshall L Fisher. Optimal solution of scheduling problems using lagrange multipliers: Part i. *Operations Research*, 21(5): 1114–1127, 1973.

[4] BR Fox and MB McMahon. Genetic operators for sequencing problems. In *Foundations of genetic algorithms*, volume 1, pages 284–300. Elsevier, 1991.

[5] Juris Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90, 1982.

[6] Hisao Ishibuchi, Naohisa Yamamoto, Tadahiko Murata, and Hideo Tanaka. Genetic algorithms and neighborhood search algorithms for fuzzy flowshop scheduling problems. *Fuzzy Sets and systems*, 67(1): 81–100, 1994.

[7] AHG Rinnooy Kan. *Machine scheduling problems: classification, complexity and computations*. Springer Science & Business Media, 2012.

[8] Wen-Chiung Lee and Chin-Chia Wu. Some single-machine and m-machine flowshop scheduling problems with learning considerations. *Information Sciences*, 179(22): 3885–3892, 2009.

[9] Alan S Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.

[10] Tadahiko Murata, Hisao Ishibuchi, and Hideo Tanaka. Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering*, 30(4):1061–1071, 1996.

[11] Ibrahim H Osman and CN Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557, 1989.

[12] Michael Pinedo. *Scheduling*. Springer, 2012.

[13] Chris N Potts and Luk N Van Wassenhove. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, 43(5):395–406, 1992.

[14] CN Potts and KR Baker. Flow shop scheduling with lot streaming. *Operations research letters*, 8(6):297–303, 1989.

[15] Colin R Reeves. A genetic algorithm for flowshop sequencing. *Computers & operations research*, 22(1):5–13, 1995.

[16] Eric Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, 47(1):65–74, 1990.

[17] Reza Tavakkoli-Moghaddam, Alireza Rahimi-Vahed, and Ali Hossein Mirzaei. A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives: weighted mean completion time and weighted mean tardiness. *Information Sciences*, 177(22):5072–5090, 2007.

[18] Eva Vallada, Rubén Ruiz, and Jose M Framinan. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240(3):666–677, 2015.

[19] Harvey M Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[20] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[21] Marino Widmer and Alain Hertz. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41(2):186–193, 1989.

[22] Yunqiang Yin, Dehua Xu, Kaibiao Sun, and Hongxing Li. Some scheduling problems with general position-dependent and time-dependent learning effects. *Information Sciences*, 179(14):2416–2425, 2009.