Búsqueda local para la programación de la producción en ambiente tipo *flow-shop*

Lina Reyes[1]

[1] Universidad de Los Andes
Bogotá, Colombia
l.reyesa@uniandes.edu.co

Resumen

Este documento presenta una heurística para resolver el problema de programación *flow-shop*. Los algoritmos de búsqueda local que se proponen son: swap e inserción. Para probar la eficiencia de los métodos propuesto se utilizaron diez instancias de gran porte disponibles en la literatura especializada. Los resultados de las búsquedas locales muestran que en promedio se presentó un GAP del 4,8% y 5,2%.

1 Introducción

Un problema de programación de la producción consiste en encontrar una secuencia de trabajos que serán ejecutados en un conjunto de máquinas, tal que minimice el *makespan* (C_{max}), que se define como el tiempo de finalización del último trabajo en dejar el sistema. Una versión de este problema es el *flow-shop*, puede describirse como un conjunto de *M* máquinas y un conjunto de *N* trabajos. Un trabajo consiste en operaciones que se realizan en cada maquina y los trabajos se procesan en serie en las maquinas, cada trabajo tiene la misma ruta de máquinas. No hay restricciones de precedencia entre las operaciones de los diferentes trabajos. El problema *flow-shop* se considera NP-HARD (Umam, Mustafid, & Suryono, 2022) por lo que suele resolverse mediante métodos de aproximación o heurísticos. Por lo tanto, las tareas más complejas deben resolverse con métodos heurísticos, los más comunes son: el recocido simulado, la búsqueda tabú, los algoritmos genéticos, la colonia de hormigas y algoritmos de búsqueda local para explorar soluciones en una vecindad, donde aquellas soluciones que pueden ser alcanzables a partir de una solución inicial, por medio de un movimiento que puede ser un intercambio entre elementos que conforman la solución inicial mejorando la función objetivo actual.

En este artículo propone el uso de dos técnicas heurísticas de búsqueda local para el problema de asignación de máquinas *flow-shop* para minimizar *makespan,* denotado (F ‖ Cmax), en las instancias propuestas por Taillard (1993). La solución de la que parten las búsquedas locales es el orden lexicográfico de 500 trabajos (*t1, t2, t3, …, t500*) y son procesados por 20 máquinas en el mismo orden. Los dos algoritmos de búsqueda local utilizados son: swap e inserción explicados al detalle en la Sección 3.

2 Descripción del problema

El problema de secuenciación de tareas en sistemas de producción lineal, *flow-shop*, ha sido un tema de gran importancia en la investigación de operaciones y se puede describir de la siguiente manera: Dado un conjunto *m* de máquinas, y un conjunto de *n* trabajos. Cada trabajo está compuesto de un conjunto de *m* operaciones que deben realizarse en diferentes máquinas. Todos los trabajos tienen el mismo orden de procesamiento al pasar por las máquinas. No hay restricciones de precedencia entre las operaciones de los distintos trabajos. Los trabajos no pueden interrumpirse y cada máquina solo puede procesar un trabajo a la vez. El problema consiste en encontrar la secuencias de trabajos en las máquinas que

minimicen el *makespan*, es decir, esta va a ser nuestra función objetivo.

A continuación, se presenta una formulación matemática para el problema de producción *flow-shop* con permutación propuesta por (Emmons & Vairaktarakis, 2012).

$$\min c_{mn} \qquad (1)$$

s, a.

$$c_{mn} \geq c_{m-1,n} + \sum_{j=1}^{|N|} p_{mj} x_{nj} \qquad \forall \qquad n \in N, m \in M \qquad (2.1)$$

$$c_{mn} \geq c_{m,n-1} + \sum_{j=1}^{|N|} p_{mj} x_{nj} \qquad \forall \qquad n \in N, m \in M \qquad (2.2)$$

$$\sum_{i=1}^{|N|} x_{ij} = 1 \qquad \forall \qquad j \in N \qquad (3.1)$$

$$\sum_{j=1}^{|N|} x_{ij} = 1 \qquad \forall \qquad i \in N \qquad (3.1)$$

$$x_{in} \in \{0,1\} \qquad \forall \qquad i, n \in N \qquad (4.1)$$

$$c_{mn} \in \mathbb{Z}_{+} \qquad \forall \qquad n \in N, m \in M \qquad (4.2)$$

La variable de decisión x_{in} indica que el trabajo $n \in N$ está en la posición $i \in N$ y la variable c_{mn} indica el tiempo de finalización en la máquina $m \in M$ de la posición del trabajo $i \in N$.

La función objetivo (1) minimiza el tiempo total de finalización (*makespan*) dado por el periodo en el que el último trabajo programado que termina en la última máquina. Las Ecuaciones (2.2) y (2.3) modelan el tiempo de finalización de los trabajos en las distintas máquinas. Esto, en función del tiempo de finalización de un trabajo dado la máquina anterior (2.2) y del trabajo anterior en la máquina actual (2.3). Las Ecuaciones (3.1) y (3.2) garantizan que cada trabajo sea asignado a una sola posición en la programación, y que cada posición reciba un trabajo. Por último, las Ecuaciones (4.1) y (4.2) modelan la naturaleza binaria y entera de las variables, respectivamente.

3 Metodología de solución

En esta sección se presentan los dos algoritmos heurísticos de búsqueda local para generar mejores soluciones al problema de *flow-shop* propuesto por (Taillard, 1993) así mismo varios componentes y se especifica el diseño de cada algoritmo. Uno de los principales componentes de la búsqueda local es la definición del conjunto de movimientos para crear una vecindad. Un movimiento cambia la ubicación de algunos trabajos en una permutación dada.

3.1. Algoritmo swap

La heurística de intercambio consiste en intercambiar dos trabajos sobre la solución inicial. Si este intercambio no mejora la solución actual que no se actualiza, se repite el proceso hasta actualizar una mejor solución. El Algoritmo 1 muestra el pseudo-código de esta heurística.

Algoritmo 1 - Swap

1. **Input:** Solución factible, *t*
2. **Parameters:** NúmInter número intercambios
3. **Output:** Mejor secuencia, *makespan*
4. Mejor secuencia = solución factible
5. Incumbent = *makespan* de la mejor solución
6. **Repeat** (NúmInter)
7. T_1 = aleatorio1 (1, 500)
8. T_2 = aleatorio2 (1, 500)
9. Swap T_1 y T_2
10. *Makespan* nuevo = *makespan* (nueva solución)
11. **if** Makespan ($nueva\ solución$) $<$ $Incumbente$ **then**
12. $mejor\ secuencia \leftarrow Nueva\ solución$
13. Incumbent = *Makespan* nuevo
14. Return incumbente, mejor secuencia

El algoritmo toma como entrada una solución factible y lo tiempos de procesamiento de cada uno de los trabajos. La tres primeras líneas inicializan la mejor solución como solución inicial. El algoritmo realizara un numero estipulado de intercambios entre trabajos (línea 6-14). A partir de una posición aleatoria dada, se evalúa un intercambio con otra posición aleatoria en la secuencia (línea 7-9). En la línea 10 se evaluará em *makespan* de la nueva secuencia y si este nuevo *makespan* es mejor al de la secuencia anterior se actualizan a la mejor solución y la secuencia de trabajos (línea 10-12). Finalmente, se obtiene la mejor solución encontrada y su objetivo.

3.2. Algoritmo Inserción

El movimiento de inserción consiste en retirar un trabajo de su posición original, para luego insertarlo en una nueva posición. El Algoritmo 2 pseudo-código muestra el pseudo-código de esta heurística

Algoritmo 2 - Inserción

1. **Input:** Solución factible, *t*
2. **Parameters:** NúmInter número intercambios
3. **Output:** Mejor secuencia, *makespan*
4. Mejor secuencia = solución factible
5. Incumbent = *makespan* de la mejor solución
6. **Repeat** (NúmInter)
7. T_1 = aleatorio1 (1, 500)
8. T_2 = aleatorio2 (1, 500)
9. insert T_1 en pos (T_2 + 1)
10. *Makespan* nuevo = *makespan* (nueva solución)
11. **if** $Makespan\ (nueva\ solución) < Incumbente$ **then**
12. $mejor\ secuencia \leftarrow Nueva\ colución$
13. Incumbent = *Makespan* nuevo
14. Return incumbente, mejor secuencia

El algoritmo toma como entrada una solución factible y lo tiempos de procesamiento de cada uno de los trabajos. La tres primeras líneas inicializan la mejor solución como solución inicial. El algoritmo

realizara un número estipulado de intercambios entre trabajos (línea 6-14). A partir de una posición aleatoria dada, se asigna una nueva posición aleatoria en la secuencia (línea 7-9). En la línea 10 se evaluará em *makespan* de la nueva secuencia y si este nuevo *makespan* es mejor al de la secuencia anterior se actualizan a la mejor solución y la secuencia de trabajos (línea 10-12). Finalmente, se obtiene la mejor solución encontrada y su objetivo.

4 Experimentos computacionales y análisis de resultados

Las instancias con las cuales se evaluó el desempeño de los algoritmos son las presentadas y utilizadas por (Talliard, 1993). El algoritmo propuesto se implementó en Python y se ejecutaron en una máquina Mac Book Air 10.1, chip M1, CPU de ocho núcleos con cuatro núcleos de rendimiento y cuatro de eficiencia. En los dos algoritmos las diez instancias se ejecutaron hasta la iteración número 10.000.
Los resultados obtenidos se resumen en la Tabla 1. *M* representa el número de máquinas y *n* los trabajos. Se toma como referencia la mejor solución encontrada por Talliard (1993), se calcula el porcentaje del GAP del algoritmo propuesto a través de la Ecuación 1.

$$GAP\ (\%) = \frac{(Valor\ obtenido - Valor\ de\ la\ mejor\ solución)}{Valor\ de\ la\ mejor\ solución} \quad \textbf{(1)}$$

Inst.	*m*	*n*	Mejor solución	SWAP		Inserción	
				Avg. Objective	*Gap*	Avg. Objective	*Gap*
ta111	20	500	26.040	27.405	5,2%	27.414	5,3%
ta112	20	500	26.520	27.917	5,3%	28.008	5,6%
ta113	20	500	26.371	27.524	4,4%	27.700	5,0%
ta114	20	500	26.456	27.568	4,2%	27.720	4,8%
ta115	20	500	26.334	27.595	4,8%	27.609	4,8%
ta116	20	500	26.477	27.681	4,5%	27.809	5,0%
ta117	20	500	26.389	27.517	4,3%	27.685	4,9%
ta118	20	500	26.560	27.759	4,5%	27.991	5,4%
ta119	20	500	26.005	27.554	6,0%	27.570	6,0%
ta120	20	500	26.457	27.637	4,5%	27.707	4,7%

Tabla 1. Resultados del makespan para los algoritmos SWAP y Incersión para las instancias (Talliard, 1993).

Con el objetico de probar la eficiencia de los dos métodos, se crearon un conjunto de instancias aleatorias con las siguientes características: se realizaron 30 corridas para cada uno y se tomó el promedio de los *makespan* hallados.

En cuanto al gap, en la Tabla 1, tuvieron un rendimiento similar, en todas las instancias. El gap máximo alcanzado fue de 6,0% y el más bajo en el algoritmo swap e inserción fue del 4,2% y 4,8% respectivamente.

El tiempo computacional para establecer el orden de los 500 trabajos en las 20 maquinas, en el algoritmo swap fue de 408 segundos, y en el algoritmo de inserción fue de 413 segundos.

Teniendo en cuenta esto, en la Figura 1 y 2 se muestra el comportamiento de la función objetivo a lo largo de las iteraciones de los algoritmos propuestos.

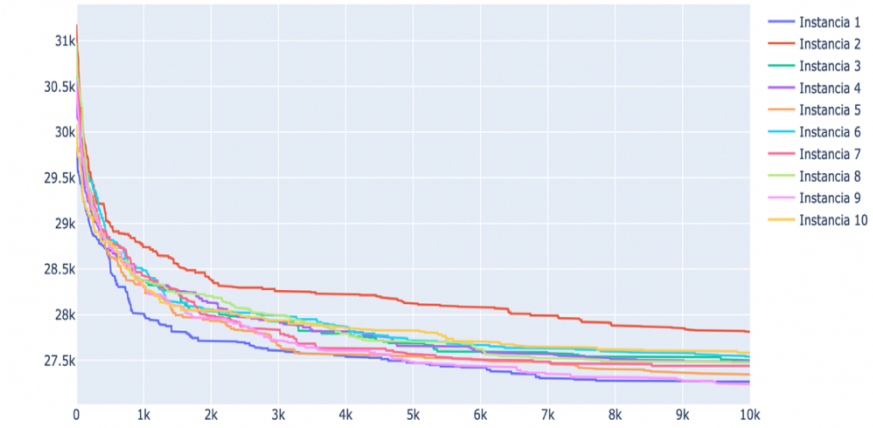

Figura 1. Comportamiento de la función objetivo para el algoritmo SWAP.

En la Figura 1 se evidencia para la búsqueda local swap, una disminución en promedio 8 % la función objetivo inicial para cada instancia. En términos de *makespan* disminuye el tiempo de procesamiento de los trabajos en un promedio de 27615. Si se realizan más de 10.000 iteraciones no hay una disminución significativa en la función objetivo, aumentando inversamente el tiempo computacional de la búsqueda local.

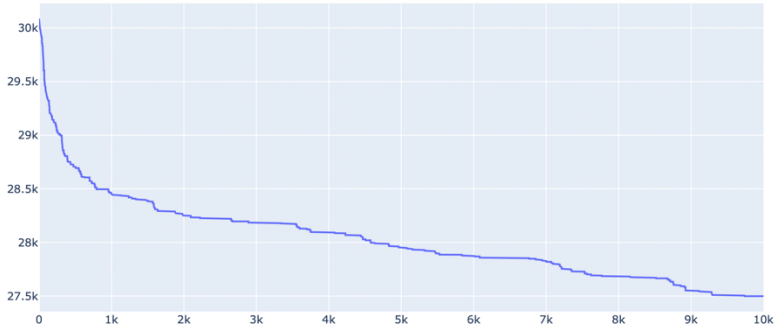

Figura 2. Comportamiento de la función objetivo para el algoritmo de inserción en la instancia tall1.

En la Figura 2 se evidencia para la búsqueda local de inserción, una disminución en promedio 8 % la función objetivo inicial para cada instancia. En términos de *makespan* disminuye el tiempo de procesamiento de los trabajos en un promedio de 27.721. Si se realizan más de 10.000 iteraciones no hay una disminución significativa en la función objetivo, aumentando inversamente el tiempo computacional de la búsqueda local.

5 Conclusiones y trabajos futuros

En este trabajo se proponen dos heurísticas de búsqueda local para el problema del *flow-shop*. Los algoritmos swap e inserción presentaron resultados similares con un promedio de gaps del 4,8% y 5,3% respectivamente y un tiempo computacional considerable para el tamaño del problema. Teniendo en cuenta que a medida que aumenta el número de trabajos el tiempo computacional va a aumentar.

Ambos algoritmos, generan intercambios de manera muy efectiva y con un número bajo de iteraciones logrando disminuir el *makespan*.

Para trabajos futuros, se pueden plantear mejores búsquedas locales que permitan generar través de una exploración más eficiente sin recurrir a la aleatoriedad, o controlarla mediante metaheurísticas que gerencien mejor los vecindarios para mejorar los GAP que se están presentando en este trabajo.

6 Referencias

Umam, M. S., Mustafid, M., & Suryono, S. (2022). A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem. *Journal of King Saud University - Computer and Information Sciences*, 7459-7467.

Emmons, H., & Vairaktarakis, G. (2012). *Flow shop scheduling: theoretical results, algorithms, and applications*(Vol. 182). Springer Science & Business Media.

Lee, C. Y. (1997). Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint. *Operations research letters*, 20(3), 129-139.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *european journal of operational research,* 64(2), 278-285.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *In Annals of Discrete Mathematics* (Vol. 5, Issue C, pp. 287–326). https://doi.org/10.1016/S0167-5060(08)70356-X