Heurísticas de búsqueda local para solucionar problemas de *Flow-shop*

Daniel Sebastián Giraldo Herrera[1]

[1] Universidad de Los Andes
Bogotá, Colombia
ds.giraldoh@uniandes.edu.co

Resumen

Este articulo propone dos heurísticas de búsqueda local para resolver el problema de secuenciamiento de máquinas *Flow-Shop Scheduling Problem* (FSSP). En este problema se basa en determinar cuál será el orden más adecuado para llevar a cabo una cantidad de tareas secuenciadas de forma que pasen en el mismo orden por todas las máquinas, con el fin de minimizar el *makespan* (tiempo de finalización de todos los trabajos).

1 Introducción

El *Flow-Shop Scheduling Problem* (FSSP) es clasificado como un problema de optimización NP-Hard, este problema es de orden $O(n!)$, Su programación está definida como una secuencia de trabajos que deben ser programados en la misma secuencia de máquinas y cada trabajo tiene un tiempo de ejecución diferente dependiendo de la máquina en la que se está trabajando (Umam, Mustafid, & Suryono, 2022).

El problema consiste en minimizar el tiempo de ejecución de todas las actividades desde el inicio de ejecución del primer trabajo y la finalización del último trabajo en la última máquina, este tiempo de ejecución de tareas es llamado *makespan*. Para este problema se plantean los siguientes supuestos (Taillard, 1990):

1. El orden de procesamiento de los trabajos en las máquinas es la misma para todos los trabajos; el problema es encontrar la secuencia de trabajos que minimiza el *makespan*.
2. Cada máquina puede procesar solo un trabajo a la vez.
3. Cualquier operación de un trabajo solo puede realizarse en una máquina de manera simultánea.
4. Para todos los trabajos, la operación puede iniciar en la siguiente máquina si fue completado en la etapa anterior y no hay un trabajo en curso en la máquina siguiente.
5. El tiempo de procesamiento de desplazamiento entre máquinas sucesivas es despreciable.
6. Los tiempos de alistamiento de los trabajos son incluidos en el tiempo de procesamiento y no dependen de la secuencia

Varias heurísticas han sido propuestas para solucionar el problema de *flow-shop*, entre ellos el algoritmo de Gupta, Johnson, RA, Palmer, CDS (Campbell, Dudek y Smith), algoritmos de proceso rápido, el NEH (algoritmo de Nawaz, Enscore y Ham). La complejidad computacional de estos algoritmos esta defindа como $O(nlog(n) + nm)$, en el mejor de los casos y otros como el de NEH tienen una complejidad de $O(n^2m)$.

En este artículo plantean dos algoritmos de búsqueda local con el fin de dar solución al grupo de instancias planteado por Taillard donde se debe minimizar el *makespan* para 500 trabajos y 20 máquinas (Taillard, 1993).

2 Modelo matemático

El problema de *flow-shop* es definido como un número n de trabajos que deben ser programados en un grupo de m máquinas, todos tienen el mismo orden de procesamiento. Cada tarea $i, i \in \{1,2,3,\dots,n\}$ requiere un tiempo de procesamiento $d_{i,j}$ para cada maquina $j, j \in \{1,2,3,\dots,m\}$. Los tiempos de alistamiento están presupuestados dentro del tiempo de procesamiento $d_{i,j}$. El objetivo de programar el orden de secuencia de procesamiento tiene como objetivo minimizar el tiempo de procesamiento de todas las tareas, mejor conocido en la literatura como *makespan* (C_{max}) (Emmons & Vairaktarakis, 2012).

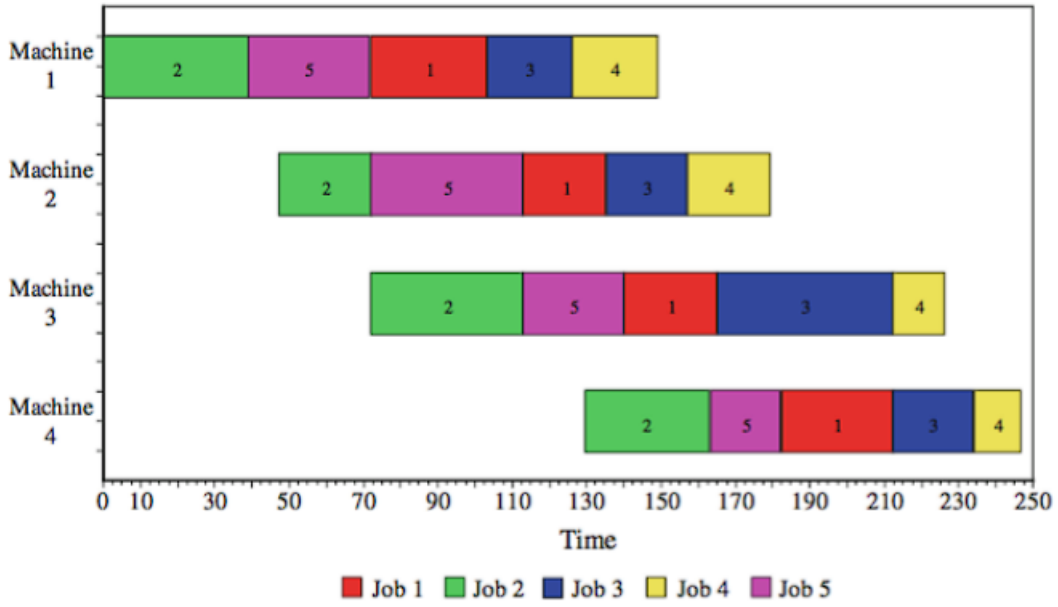

Ilustración 1. Ejemplo de diagrama de Gantt de un *flow-shop*

La formulación matemática para la programación del *flow-shop* propuesto por (Emmons & Vairaktarakis, 2012)

$$\min c_{mn} \tag{1}$$

S. A.

$$c_{mn} \geq c_{m-1,n} + \sum_{j=1}^{|N|} p_{mj} x_{nj} \quad \forall \quad n \in N, m \in M \tag{2}$$

$$c_{mn} \geq c_{m,n-1} + \sum_{j=1}^{|N|} p_{mj} x_{nj} \quad \forall \quad n \in N, m \in M \tag{3}$$

$$\sum_{i=1}^{|N|} x_{ij} = 1 \quad \forall \quad j \in N \tag{4}$$

$$\sum_{j=1}^{|N|} x_{ij} = 1 \quad \forall \quad i \in N \tag{5}$$

$$x_{in} \in \{0,1\} \quad \forall \quad i,n \in N \tag{6}$$

$$c_{mn} \in \mathbb{Z}_+ \quad \forall \quad n \in N, m \in M \tag{7}$$

La Ecuación (1) es la función objetivo, que busca minimizar el tiempo total de fabricación de todas las tareas llamado *makespan* a través de todas las máquinas. La Restricción (2) calcula el tiempo de terminación de fabricación en la maquina anterior, la Restricción (3) determina el tiempo de terminación del trabajo anterior en la máquina actual. Las Restricciones (4) y (5) se aseguran de que una tarea solo pueda ser ejecutada en una máquina y que cada máquina solo pueda procesar una tarea. Finalmente, las Restricciones (6) y (7) muestran la naturaleza de las variables, binaria y entera respectivamente.

3 Metodología

Para probar la eficiencia de los algoritmos de búsqueda local planteada se utilizan las instancias de *flow-shop* de Taillard (1993) para un tamaño de 500 trabajos y 20 máquinas. La solución inicial para cada heurística es la secuencia de tareas en orden lexicográfico [t1, t2, t3, … , t500] y el orden de las máquinas también es el orden lexicográfico [M1, M2, M3, … , M20].

3.1 Búsqueda local Cross order

El primer algoritmo de búsqueda local es llamado *cross order* para el secuenciamiento de trabajos en *flow-shop*. Esta heurística, elige dos trabajos de manera aleatoria, los trabajos que existen en medio de estos dos trabajos cambiarán de orden, haciendo que su secuenciamiento sea el orden invertido de cómo está actualmente, para este procedimiento se actualizará el *makespan* bajo el criterio de mejor mejora, guardando únicamente la mejor función objetivo de todos los intercambios en un vecindario de búsqueda de 1.000 iteraciones.

Búsqueda local cross order

1. $Secuencia = solución\ inicial$
2. $C_{max} = Makespan\ (Secuencia), mejora \leftarrow true$
3. $\boldsymbol{For}\ i\ in\ \#iteraciones\ \boldsymbol{do}$
4. $\quad \boldsymbol{t_1 = rand(pos(secuencia))}$
5. $\quad \boldsymbol{t_2 = rand(pos(secuencia))}$
6. $\quad$ **Sec1** = Secuencia de tareas desde la primera tarea hasta $\boldsymbol{t_1}$
7. $\quad$ **Sec2** = Secuencia de tareas invertida de orden desde $\boldsymbol{t_1 + 1}$ hasta $\boldsymbol{t_2 - 1}$
8. $\quad$ **Sec3** = Secuencia de tareas desde $\boldsymbol{t_2}$ hasta la última tarea
9. $\quad NuevaSecuencia = Sec1 + Sec2 + Sec3$
10. $\quad NuevoC_{max} = Makespan(NuevaSecuencia)$
11. $\quad\quad \boldsymbol{If}\ NuevoC_{max} < C_{max}\ \boldsymbol{then}$
12. $\quad\quad\quad Secuencia = NuevaSecuencia$
13. $\quad\quad\quad C_{max} = NuevoC_{\max}$
14. $\quad\quad \boldsymbol{end\ if}$
15. $\boldsymbol{end\ for}$
16. $\boldsymbol{return}\ \ secuencia, C_{max}$

El algoritmo presentado anterior inicializa con una solución factible inicial, la cual es el orden de las tareas, el cálculo de *makespan*de dicha solución, El funcionamiento de la búsqueda local (líneas 3 a 15) del *cross order* trabaja para un número determinado de iteraciones, en este caso 1000 iteraciones, se elegirán dos posiciones al azar del vector de soluciones (líneas 4 y 5) dividiendo el vector de la secuencia en tres partes. La primera sección del vector, el cual hace referencia a la secuencia de tareas con la que se inicia, hasta la primera tarea elegida al azar (línea 6), en la sección dos del vector se guarda en orden

invertido desde la tarea elegida al azar $\boldsymbol{t_1 + 1}$ y la tarea anterior a la segunda tarea elegida al azar $\boldsymbol{t_1 - 1}$ (línea 7). La ultima parte de la secuencia es guardada en la sección tres en su orden normal (línea 8). Luego se unen estas generando la nueva secuencia de tareas (línea 9) y calculando el *makespan* de la nueva solución. Si este nuevo orden es mejor que la actual es guardada en la incumbente (línea 13) de lo contrario se sigue iterando hasta terminar el número de intentos. Finalmente se retorna la mejor secuencia de tareas encontrada y la función objetivo de las tareas.

Para ejemplificar el funcionamiento de la búsqueda local *cross order,* la Ilustración 2 presenta el vector de secuencia de tareas inicial, donde se eligen al azar dos posiciones, en este caso la posición tres y ocho del vector, y dividiendo el vector en tres secciones.

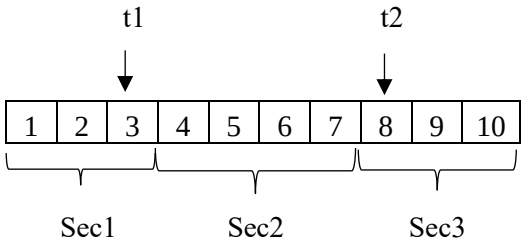

Ilustración 2. Vector de secuencia inicial de tareas.

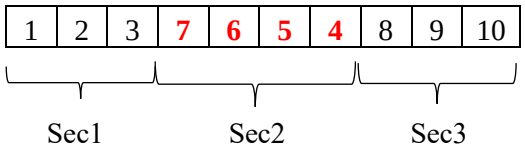

Ilustración 3. Vector de secuencia nueva de tareas.

La Ilustración 3 presenta el nuevo orden de la secuencia de tareas, donde la sección dos del vector se cambió de sentido, generando una nueva secuencia.

3.2 Búsqueda local swap

El segundo algoritmo de búsqueda local propuesta es el *swap*, el cual realiza el intercambio de dos tareas en orden secuencial, explorando el espacio de búsqueda y actualizando el orden de secuencia de tareas al momento de encontrar un *makespan* con tiempo de ejecución menor al de la configuración actual.

Búsqueda local swap

1. $Secuencia = solución\ inicial$
2. $C_{max} = Makespan\ (Secuencia), mejora \leftarrow true$
3. $\boldsymbol{While}\ mejora\ \boldsymbol{do}$
4. $\quad mejora \leftarrow false$
5. $\quad \boldsymbol{for}\ i\ in\ Secuencia\ \textbf{do}$
6. $\quad\quad \boldsymbol{for}\ j\ in\ Secuencia\ \textbf{do}$
7. $\quad\quad\quad NuevaSecuencia = Secuencia$
8. $\quad\quad\quad \boldsymbol{Swap}\ (NuevaSecuencia) = tarea[i], tarea[j] \leftarrow tarea[j], tarea[i]$
9. $\quad\quad\quad NuevoC_{max} = Makespan(NuevaSecuencia)$
10. $\quad\quad\quad \boldsymbol{if}\ NuevoC_{max} < C_{max}\ \boldsymbol{then}$
11. $\quad\quad\quad\quad Secuencia = NuevaSecuencia$
12. $\quad\quad\quad\quad C_{max} = NuevoC_{\max}$
13. $\quad\quad\quad\quad mejora = true$
14. $\quad\quad\quad \boldsymbol{end\ if}$
15. $\quad\quad \boldsymbol{end\ for}$

16. ***if*** $mejora = true$
17. ***break***
18. ***end if***
19. ***end for***
20. ***end while***
21. ***return*** $secuencia, C_{max}$

El segundo algoritmo de búsqueda local propuesto es el *swap*, el cual realiza un intercambio en el orden de tareas de dos de ellas intercambiándolas, y calculando el nuevo *makespan*. El algoritmo comienza con una orden de secuencia de tareas factible y el valor de la función objetivo de dicho orden (líneas 1 y 2). El ciclo de repeticiones se hace hasta que no se encuentre una mejor solución (líneas 3 a 20). Para cada trabajo en la secuencia se realizarla el intercambio con los trabajos siguientes y se calculará en cada iteración el *makespan* de esa secuencia (líneas 5 a 9), si hay dicho intercambio entre tareas es menor al *makespan* de la solución actual este se actualizará y se volverá a hacer el intercambio de tareas desde la primera posición del vector. Finalmente se retorna la mejor secuencia de tareas encontrada y la función objetivo de las tareas.

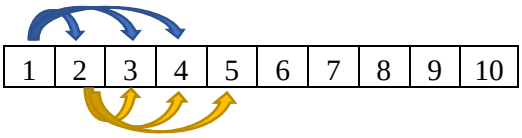

Ilustración 4. *Swap* entre tareas en una solución factible

La Ilustración 4 muestra cómo se comporta la búsqueda local, esta búsqueda es bastante exhaustiva y genera muy buenos resultados, aunque presenta un alto costo computacional y tiempos de ejecución largos. La complejidad algorítmica de esta búsqueda local es $O(n^3)$, la cual podría disminuirse si se eligieran al azar las posiciones a intercambiar, sacrificando así calidad en las secuencias de tareas encontradas debido a la aleatoriedad de los intercambios.

4 Análisis de resultados

Para determinar el desempeño de las búsquedas locales planteadas, y verificar que puedan generar buenos resultados para problemas reales, se ponen a prueba deiz de las instancias propuestas por Taillard (1993), constituidas por 20 maquinas y 500 trabajos. Las especificaciones técnicas de la máquina en la que se corrieron dichas instancias son: Dell G15, CPU Ryzen 7, 5800 Hz, 8Gb RAM. La comparación se realizo con las mejores soluciones reportadas (*BKS*) que provienen de distintas investigaciones, ya que es un problema muy estudiado aún, la mayoría de ellas reportadas por Taillard (1993), y otras encontradas más recientemente por Companys Pascual, Mateo Doll y Aleman (2005).

Tabla 1: Resultados obtenidos por los algoritmos de búsqueda local

Instancia	BKS	Cross order	Swap	Cross order gap (%)	Swap gap (%)
Tai500_20_1	26040	28241	26918	8.45	3.37
Tai500_20_2	26520	28661	27047	8.07	1.99
Tai500_20_3	26371	28565	27042	8.32	2.54
Tai500_20_4	26456	28283	26999	6.91	2.05
Tai500_20_5	26334	28343	26952	7.63	2.35
Tai500_20_6	26469	28735	26894	8.56	1.61
Tai500_20_7	26389	28517	26929	8.06	2.05

Tai500_20_8	26560	28946	27129	8.98	2.14
Tai500_20_9	26005	28237	27156	8.58	4.43
Tai500_20_10	26457	29009	26918	9.65	1.74
			Promedio	8.32	2.43

Los algoritmos fueron probados en las diez instancias, donde todos corrieron exitosamente y generaron soluciones de buena calidad, donde el desempeño de la heurística de *Swap* tuvo el mejor desempeño en todas las instancias teniendo un promedio de 2.43% de Gap respecto a las mejores soluciones encontradas en la literatura, donde el Gap más alejado fue el de la instancia nueve con un 4.43%, y el mejor fue de la instancia diez con un 1,74% de gap. La heurística de *cross order* tuvo un gap promedio de 8.32% respecto a las mejores soluciones, cabe recalcar que esta heurística de búsqueda local es aleatoria y su desempeño puede variar considerablemente, para este ejercicio se fijó una semilla para eliminar tanta variabilidad. Por lo que se deben realizar más experimentos con el fin de determinar adecuadamente el gap promedio de más corridas e instancias.

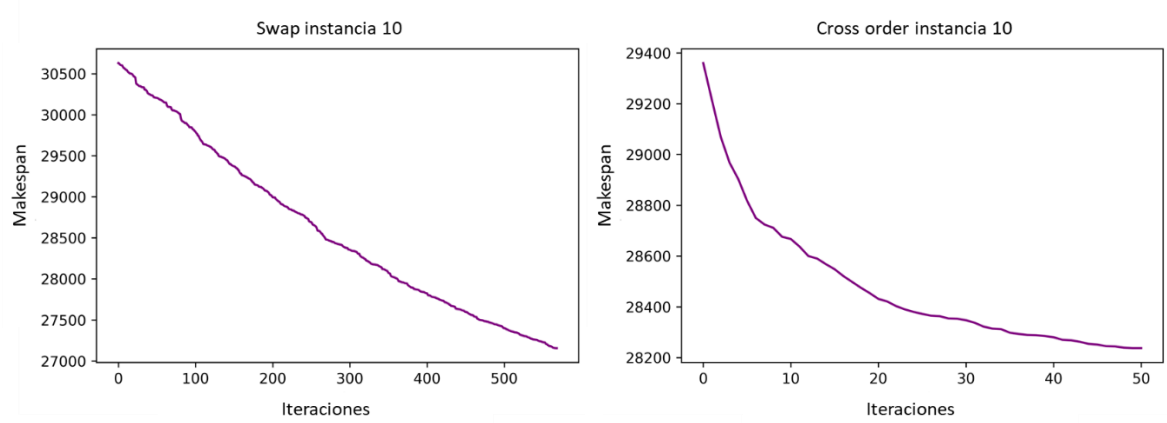

Ilustración 5. Desempeño heurísticas de búsqueda local para la instancia 10.

En la Ilustración 5 se puede apreciar el comportamiento de las búsquedas locales a través de las iteraciones, para la búsqueda local de swap que tiene mejores soluciones se puede apreciar que realiza más de 500 iteraciones, su vecindario de exploración es más grande por lo tanto puede explotar mejores soluciones. La búsqueda local de cross order por el contrario tiene menos del 10% de iteraciones realizadas que el swap, cabe aclarar que el *swap* funciona bajo el criterio de primera mejora mientras que el *cross order* funciona bajo el criterio de mejor mejora.

5 Conclusiones

Teniendo en cuenta que la motivación principal de este artículo es generar búsquedas locales en diferentes tamaños de vecindarios y diferentes criterios para actualizar la función objetivo para el problema de secuenciamiento de tareas *flow-shop*, las heurísticas planteadas generan soluciones cercanas a las mejores encontradas en la literatura.

Como trabajo futuro se podría plantear una metaheurística para disminuir los tiempos de computacionales, actualmente el tiempo computacional empleado para estas búsquedas locales es bastante excesivo y no es competitivo, gestionar mejor la búsqueda local como el *swap* si tener que recorrer de forma ordenada cada tarea e intercambiarla y recalcular todo genera un sobre esfuerzo para la maquina que puede ser mitigado utilizando un tabú search o VNS, con el fin de eliminar los intercambios que se saben que ya no generan buenas soluciones.

6 Referencias

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

Framinan , J. M., Leisten, R., & García, R. R. (2014). *Manufacturing scheduling systems.* Springer.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, 65-74.

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

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.