Búsqueda local para solucionar el problema del Flow Shop Scheduling (FSSP)

Luis E. Tarazona[1]

[1] Universidad de los Andes
Carrera 1 Este # 19 A – 40, Bogotá D.C., Colombia
le.tarazona@uniandes.edu.co

Resumen

En este documento se considera dos heurísticas de búsqueda local para resolver el FSSP cuyo objetivo es buscar la secuencia en la que se deben organizar los trabajos en un determinado conjunto de máquinas para minimizar el tiempo de terminación de todos estos (Cmax). Se discutirá el rendimiento del algoritmo considerando el tiempo computacional de solución y se comparará los resultados obtenidos de costo respecto a los BKS de las instancias utilizadas y los obtenidos en otras investigaciones.

1 Introducción

Un sistema Flow Shop es aquel sistema que tiene como característica principal que la secuencia de operaciones que se realizan en un conjunto de máquinas es el mismo para todos los trabajos que se realizan en este. Ejemplos de este sistema pueden ser el ensamble de electrodomésticos el envasado de bebidas, la producción de alimentos envasados, etc. En un sistema que tiene una producción de alto volumen y baja variedad. Otras características que tiene este sistema es que requiere maquinaria con propósito específico, operadores menos capacitados y siempre produce considerando tener inventario. El problema radica cuando se requiera realizar la programación de una cantidad considerable de trabajos, la toma de decisión se ve afectada porque si la cantidad de trabajos es muy grande es más complicado realizar una programación adecuada en un tiempo prudente y definitivamente realizarla manualmente será prácticamente imposible de hacer. Es por eso que este problema es NP-Hard ya que no puede ser resuelto en tiempos computacionales razonables.

Este problema de programación en este tipo de sistemas de producción, ha sido estudiado desde 1950, con la heurística propuesta por Johnson (1954), que solucionaba el problema con 2 máquinas. (Taillard, 1989) propuso una serie de instancias para hacer un estudio de este problema donde obtuvo soluciones óptimas. Estas han servido para desarrollar diversos estudios que logran solucionar este problema con M cantidad de máquinas y N cantidad de trabajos en tiempos computacionales razonables, como el desarrollado por (Faith, Yun-chia, Mehmet, & Gunes, 2006) y (Guangchen, Liang, Xinyu, Peigen, & Fatih, 2020) que realiza una metaheurística PSO para resolver este problema. (Khurshid, y otros, 2021) desarrollaron una metaheurística híbrida con recocido simulado para poder mejorar la programación de este tipo de sistemas.

Es por eso que el presente trabajo presenta dos soluciones heurística con búsqueda local de intercambio de dos posiciones (SWAP) y búsqueda local con inserción (INSERT) para solucionar el FSSP, en el siguiente apartado se muestra la descripción del problema FSSP, en el apartado tres se muestra el proceso de ambas heurísticas planteadas para este trabajo, en el apartado cuarto se muestra los resultados obtenidos y comparados con los resultados (BKS) de (Taillard, 1989) y los resultados de (Khurshid, y otros, 2021) ya que usaron las instancias propuestas por el primer autor en mención. Finalmente se muestra las conclusiones del trabajo.

2 Descripción del problema

El problema consiste en realizar la organización para el procesamiento de un número de trabajos (ji) en un determinado número de máquinas (mi), considerando dos supuestos muy importantes, la disponibilidad de máquina, que refiere a que no se puede realizar un trabajo si la máquina no está disponible, y la precedencia de operaciones, que refiere a que no se puede realizar una operación sin antes haber terminado una operación predecesora.

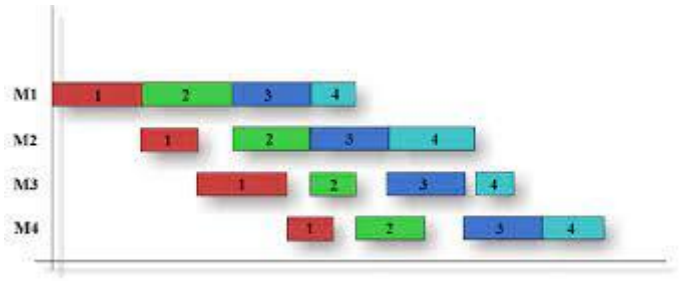

Figura 1: Diagrama de Gantt del sistema flow shop

Como se puede ver en la Figura 1, se cumple claramente los dos supuestos más importantes de este problema, en esta Figura se contempla una programación de 4 trabajos en 4 máquinas, si se considera un número mayor de trabajos el problema se convierte más difícil de desarrollar. (Pinedo, 2016) representó este problema con la siguiente formulación:

Parámetros, índices y conjuntos

n: número de trabajos
m: número de máquinas
j: índice de trabajos, $j = 1 \dots n$
i: índice de máquinas, $i = 1 \dots m$
J: Conjunto de trabajos
M: Conjunto de máquinas
p_{ji}: Tiempo de procesamiento del trabajo j en la máquina i

$$M_{ji} = \begin{cases} 1 \; si \; el \; trabajo \; j \; se \; puede \; procesar \; en \; la \; máquina \; i \\ 0 \; si \; es \; lo \; contrario \end{cases}$$

Variables

$$x_{jk} = \begin{cases} 1 \; si \; el \; trabajo \; j \; es \; el \; k - ésimo \; en \; la \; secuencia \\ 0 \; si \; es \; lo \; contrario \end{cases}$$

$$z_j = \begin{cases} 1 \; si \; el \; trabajo \; j \; es \; programado \\ 0 \; si \; es \; lo \; contrario \end{cases}$$

Función objetivo

$$minimizar \; Cmax \qquad (1)$$

Restricciones

$$\sum_{j=1}^{n} x_{jk} = 1 \quad para\ k\ \in J \tag{2}$$

$$\sum_{k=1}^{n} x_{jk} = 1 \quad para\ j\ \in J \tag{3}$$

$$I_{ik} + \sum_{j=1}^{n} x_{j,k+1}(p_{ji} + s_{ji}) * M_{ji} + W_{i,k+1} - W_{ik} - \sum_{j=1}^{n} x_{jk}(p_{j,i+1} + s_{j,i+1}) * M_{j,i+1} \tag{4}$$

$$\sum_{k=1}^{n}\sum_{i=1}^{m} x_{jk}(p_{ji} + s_{ji}) * M_{ji} \leq T_{max} \tag{5}$$

$$x_{jk}\ \in \{0,1\}\ \ j,k \in J \tag{6}$$

$$z_j\ \in \{0,1\}\ \ j \in J \tag{7}$$

La ecuación (1) muestra la función objetivo, la cual es minimizar el tiempo total de procesamiento. La ecuación (2) garantiza que cada posición solo tenga un trabajo asociado, la ecuación (3) indica que cada trabajo se encuentre en una posición. La ecuación (4) garantiza define la diferencia entre el tiempo cuando el trabajo en la posición k+1 inicia en la máquina i+1 y el tiempo en el que el k termina en la máquina i. la ecuación (5) define el tiempo máximo de Cmax tiene que ser menor o igual al tiempo máximo de programación. Las ecuaciones (6) y (7) indica que las variables son binarias.

3 Procedimiento heurístico

Para realizar la solución de este problema se planteó dos heurísticas de búsqueda local, la primera fue el intercambio de posición (SWAP) y la segunda la inserción de una posición entre dos posiciones (INSERT), para ello se desarrolló el siguiente procedimiento verbal antes de desarrollar el algoritmo de forma computacional considerando las 10 instancias de 500 trabajos y 20 máquinas propuestas por (Taillard, 1989):

SOLUCIÓN INICIAL

1) Considerando una secuencia inicial de 0 a 500 se calcula la función objetivo inicial.

SWAP

1) Definir el número de intercambios
2) De la secuencia inicial, se le genera un vecino, el cual es intercambia la posición i a la i+1 y viceversa.
3) Se calcula la función objetivo para este vecino.
4) Si la nueva función objetivo es menor a la función objetivo inicial, aceptar vecino y actualizar como nueva secuencia, sino repetir desde paso 1 hasta llegar al número de intercambios definidos.

INSERT

1) Definir el número de cambios
2) De la secuencia inicial, se le genera un vecino, el cual la posición i se pondrá entre las posiciones h y j.
3) Se calcula la función objetivo para este vecino.
4) Si la nueva función objetivo es menor a la función objetivo inicial, aceptar vecino y actualizar como nueva secuencia, si no, repetir desde paso 1 hasta llegar al número de cambios definidos.

Se desarrolló los siguientes pseudocódigos para ambos algoritmos de búsqueda local:

SWAP

1. Input → (secuencia, trabajos, máquinas, tiempos de procesamiento)
2. FO inicial = Cmax
3. Secuencia inicial = Secuencia
4. Repeticiones = Repeticiones
5. Determinar vecino (intercambio i - j)
6. i = i +1
7. i+1 = i
8. FO nueva = Cmax nuevo
9. Si Cmax < Cmax nuevo
10. Cmax nuevo = Cmax
11. i = i
12. i+1 = i+1
13. Secuencia nueva = secuencia
14. Sino:
15. Cmax = Cmax nuevo
16. secuencia = Secuencia nueva
17. Repetir desde paso 5 hasta repeticiones = repeticiones
18. Output → (Cmax_nuevo, secuencia_nueva)

INSERT

1. Input → (secuencia, trabajos, máquinas, tiempos de procesamiento)
2. FO inicial = Cmax
3. Secuencia inicial = Secuencia
4. Cambios = cambios
5. Determinar vecino (i en h y j)
6. Determinar Secuencia nueva
7. FO nueva = Cmax nuevo
8. Si Cmax < Cmax nuevo
9. Cmax nuevo = Cmax
10. Secuencia nueva = secuencia
11. Sino:
12. Cmax = Cmax nuevo
13. secuencia = Secuencia nueva
14. Repetir desde paso 5 hasta intercambios = intercambios
15. Output → (Cmax_nuevo, secuencia_nueva)

4 Resultados

Los resultados fueron obtenidos utilizando un computador portátil de sistema operativo WINDOWS 11 de procesador CORE i7 8va generación de 16 GB de RAM. El número de repeticiones para detener el SWAP fue de 6, ya que realizaba todos los intercambios posibles, y el número de intercambios para el INSERT fue de 300000.

En la Tabla 1 se muestra los resultados obtenidos para la aplicación de las heurísticas. Se observa para

el SWAP un tiempo computacional mínimo de 2496 segundos equivalente a 41 minutos de búsqueda y un tiempo computacional máximo de 7361 segundos, equivalente a 122 minutos de búsqueda. Además, se aprecia que los valores de GAP ($Resultados - BKS/BKS$) presentan porcentajes por encima del 10% lo que se interpreta como resultados bastante alejados del mejor resultado (BKS) en un tiempo computacional razonable, lo que demuestra que la búsqueda local por intercambio de posiciones tiene la desventaja que, a mayor cantidad de trabajos, el tiempo computacional es mucho mayor. Pasa lo contrario con la aplicación del INSERT, el cual tuvo un tiempo computacional mínimo de 1853 segundos equivalente a 30 minutos de búsqueda y un tiempo computacional máximo de 2623 segundos equivalente a 44 minutos de búsqueda. Se aprecia, además, que todos los valores de GAP estuvieron por debajo del 3% lo que demuestra que se obtuvo resultados factibles muy cercanos al mejor resultado (BKS) para cada instancia en tiempos computacionales razonables.

	SWAP		**INSERT**				
Instancias	**TC (seg)**	**Cmax**	**TC (seg)**	**Cmax**	**BKS**	**GAP SWAP**	**GAP INSERT**
1	3160,36	29198	2264,33	26640	26040	12,13%	2,30%
2	5140,27	29608	1853,03	26904	26520	11,64%	1,45%
3	7361,40	29352	2252,51	26810	26371	11,30%	1,66%
4	3692,97	29404	1830,32	26750	26456	11,14%	1,11%
5	2496,59	29380	1808,47	26550	26334	11,57%	0,82%
6	3121,08	29644	2585,55	26800	26477	11,96%	1,22%
7	3776,18	29327	2212,14	26619	26389	11,13%	0,87%
8	3167,43	29661	2531,30	26961	26560	11,68%	1,51%
9	3524,72	29255	2623,83	26375	26005	12,50%	1,42%
10	4521,22	29357	2345,69	26902	26457	10,96%	1,68%

Tabla 1: Resultados obtenidos y GAP

En la Tabla 2 se muestra los GAP obtenidos para las pruebas realizadas por (Khurshid, y otros, 2021), se evidencia claramente que la heurística INSERT propuesta en esta investigación tiene mejores resultados que el algoritmo $\mathbf{PSO_{ENT}}$ ya que los GAP de este algoritmo son mayores a 2% mientras que los de la presente investigación solamente la instancia 1 presenta un GAP por encima de 2%. Respecto a los algoritmos $\mathbf{NEGA_{VNS}}$ y $\mathbf{HES_{SA}}$ estos presentan mejores resultados de GAP, sin embargo, no se está muy distante de estos, por lo que se podría mejorar los resultados con mayor tiempo de búsqueda.

Instancias	**$NEGA_{VNS}$**	**GAP $NEGA_{VNS}$**	**HES_{SA}**	**GAP HES_{SA}**	**PSO_{ENT}**	**GAP PSO_{ENT}**
Inst_1	26228	0,72%	26187	0,56%	26737	2,68%
Inst_2	26688	0,63%	26799	1,05%	27497	3,68%
Inst_3	26522	0,57%	26496	0,47%	27277	3,44%
Inst_4	26586	0,49%	26612	0,59%	27080	2,36%
Inst_5	26541	0,79%	26514	0,68%	26915	2,21%
Inst_6	26582	0,40%	26661	0,69%	27203	2,74%
Inst_7	26660	1,03%	26529	0,53%	27057	2,53%
Inst_8	26711	0,57%	26750	0,72%	27270	2,67%
Inst_9	26148	0,55%	26223	0,84%	26622	2,37%
Inst_10	26611	0,58%	26619	0,61%	27164	2,67%

Tabla 2: Comparación con GAP de otros autores

5 Conclusiones

Se desarrolló dos algoritmos heurísticos de búsqueda local (SWAP e INSERT) para solucionar el problema de programación de sistemas de producción Flow shop. Se determinó que la heurística INSERT presenta mejores resultados comparados al SWAP, siendo su búsqueda más precisa y obteniendo resultados con un GAP para las instancias menor a 2% (a excepción de la instancia 1).

Se recomienda para futuras investigaciones realizar un estudio con un tiempo de búsqueda mayor para conseguir mejores resultados y probar una metaheurística de búsqueda que pueda optimizar el tiempo.

Referencias

Faith, T., Yun-chia, L., Mehmet, S., & Gunes, G. (2006). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. Elsevier, 1930-1947.

Guangchen, W., Liang, G., Xinyu, L., Peigen, L., & Fatih, T. (2020). Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm. Swarm and Evolutionary Computation,.

Khurshid, B., Maqsood, S., Omair, M., Sarkar, B., Ahmad, I., & Muhammad, K. (2021). An improved evolution strategy hibrydization with simulated annealing for permutation flow shop scheduling problems. IEEE.

Pinedo, M. (2016). Scheduling: Theory, Algorithms and Systems. Springer International Publishing.

Taillard, E. (1989). Benchmarks for basic scheduling problems.