1 Introducción: Investigación operativa

Scheduling (programación de tareas). Consiste en asignar recursos a actividades en el tiempo. Matematicamente estos problemas estan calificados como los más dificiles

1.1 Flow Shop Scheduling Problem (FSSP)

Es un problema de lineas de producción. Los trabajos J_i deben ser procesados en las máquinas M_i con tiempos fijos p_{ji} y son independientes para cada trabajo. Además asumimos que los tiempos de trabajo ya han sido optimizados.

		Máquinas					
		M_1	M_2	M_3	M_4	M_5	M_6
Trabajos	J_1	3	6	3	3	4	3
	J_2	4	3	5	3	5	2
	J_3	6	5	2	2	2	4
	J_4	4	5	2	2	5	5
	J_5	2	2	5	6	3	5
	J_6	2	3	5	5	3	3

Figura 1: Instancia a analizar

Rápidamente, sin analizar el tiempo que puede demorar encontrar la solución optima; podemos pensar en diseñar un algoritmo de fuerza bruta, es sencillo para nosotros pero imposible para la máquina.

```
1: function BRUTE_FORCE( )
2:     Set π_b = first permutation
3:     for all permutations π ∈ Π do
4:         Set π = next permutation
5:         if C_max(π) < C_max(π_b) then
6:             update π_b = π
7:     return π_b
```

Figura 2: Algoritmo de fuerza bruta

1.2 ¿Qué es una Heurística?

Procedimiento simple diseñado de manera inteligente para crear una solución o para buscar mejores soluciones que satisfagan a cierto problema de optimización.

- Idea, criterio, método, o regla que ayuda a decidir cuál alternativa es mejor
- Idea basada en la intuición o en el sentido común
- Idea que utiliza la estructura o contexto del problema

La primera Heurística en la historia fue nombrada *Método Monte Carlo* y se basaba en escoger una muestra del total y observar cuántos cumplen con el propósito.

1.2.1 Búsqueda aleatoria para el FSSP

Selecciona una muestra aleatoria extraída del espacio de soluciones para encontrar un resultado númerico (media esperada, mejor, peor)

```
1: function RANDOM_SEARCH( )
2:     Set π_b = random permutation
3:     for k iterations do        // k es el tamaño de muestra
4:         Set π = new random permutation
5:         if C_max(π) < C_max(π_b) then
6:             update π_b = π
7:     return π_b
```

Figura 3: Búsqueda aleatoria

Este enfoque es muy general y puede ayudar a resolver otro tipo de problemas. Sin embargo, no es el mejor para resolver el problema propuesto:

- La muestra puede ser demasiado pequeña y no representativa
- La media de la muestra $\approx$ la media de la población
- El mejor de la muestra no es el mejor de la población

2 Métodos heurísticos básicos

2.1 Heurísticas constructivas

Construyen una solución desde cero, añadiendo uno a uno los componentes a la solución parcial, hasta que la solución esté completa. La pregunta importante para diseñar esta Heurística es ¿Cuál elemento debería añadir y cómo?

2.1.1 Heurística constructiva de Nawaz, Enscore y Ham

Propuesta en 1983 para resolver el FSSP y se divide en dos partes importantes.

Primera etapa: Cuál es el elemento que debo insertar Determina un orden de inserción. En este caso, se ordena del más grande al más pequeño según su tiempo total de procesamiento. $\pi_0 = (J_4, J_5, J_1, J_2, J_3, J_6)$

Segunda etapa: Cómo o dónde lo debo de insertar Insertar estos trabajos, uno a uno, en la mejor posición, comenzando con $\pi = (J_4)$

```
1: function NEH_Heuristic( )
2:     π_o = Priority_Order( )    // ordenados por el tiempo total
3:     π = (π_o(1))
4:     for k ∈ [2, n] do
5:         k' = Best_Insertion_Position(π, π_o(k))
6:         π = (π(1),...,π(k'−1), π_o(k), π(k'),...,π(k−1))
7:     return π
```

Figura 4: Heurística NEH

- Si ocurre empates al realizar la segunda etapa, se ha probado que existe una técnia para solucionar este problema, por el momento solo cogemos el orden del primer resultado.
- La calidad de la solución se mide con el desvío relativo

$$DR = \frac{C_{\max}(\pi) - C_{\max}(\pi^*)}{C_{\max}(\pi^*)}$$

2.1.2 Costo de la Heurística constructiva

Tenemos que analizar los costos en las dos partes que componen el algoritmo.

- Ordenar cuesta $O(nlogn)$
- NEH evalúá $\frac{n(n+1)}{2-1}$ programas parciales, y esto es $O(n^2)$. Recordemos que cuando insertamos el primer trabajo, no comparamos con nada; luego cuando insertamos el segundo trabajo, lo podemos insertar antes o después del primer trabajo y asi sucesivamente.
- Finalmente, evaluar cada programa parcial significa evaluar los J_i trabajos en las M_j máquinas. Por lo tanto el costo es $O(nm)$

Entonces, la Heurística constructiva NEH original tiene un tiempo de ejecución de $O(n^3m)$.

Sin embargo Taillard calculó la mejor posición de inserción en cuatro matrices de $O(nm)$. Con esta aceleración, NEH inserta los n trabajos en un tiempo $O(n^2m)$

2.1.3 Aceleración de Taillard

Taillard hace dos análisis para poder aplicar esta aceleración. Supongamos que tenemos ya programado 4 trabajos en 4 máquinas. Entonces, ¿dónde puedo agregar un quinto trabajo?

Los tiempos de finalización más tempranos Imaginemos que empujamos todos los trabajos hacía el inicio, lo más posible que se pueda. Es decir, el procesamiento del trabajo pasa a la siguiente justo cuando culmina su procesamiento en la primera máquina.

Conclusión: Si insertamos el quinto trabajo en cualuquir lugar, nos daremos cuenta que los tiempos de finalización en cada máquina de los trabajos procesados antes del nuevo trabajo insertado **no cambian**

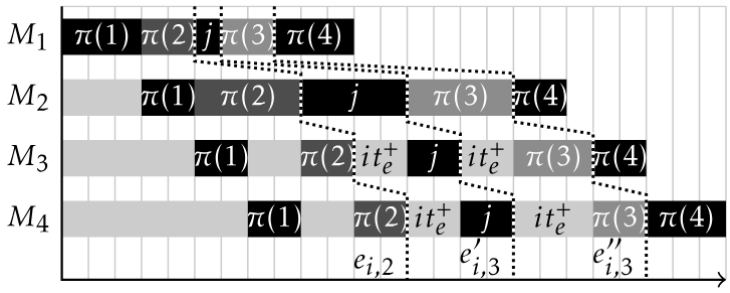


Figura 5: Tiempos de finalización más tempranos

Los tiempos de iniciación más tardios Imaginemos que empujamos los trabajos hacia el final, lo más posible que se pueda. Recordemos que los tiempos de procesamiento de cada trabajo en cada máquina varía, por lo tanto en cada máquina pueden quedar espacios sin procesamiento. Dichos espacios pueden ser ocupados por los trabajos que ya se procesaron en la máquina.

Conclusión: Si insertamos el quinto trabajo en cualquier lugar, nos daremos cuenta que los tiempos de iniciación tardía en cada máquina de los trabajos procesados después del nuevo trabajo insertado **no cambian**

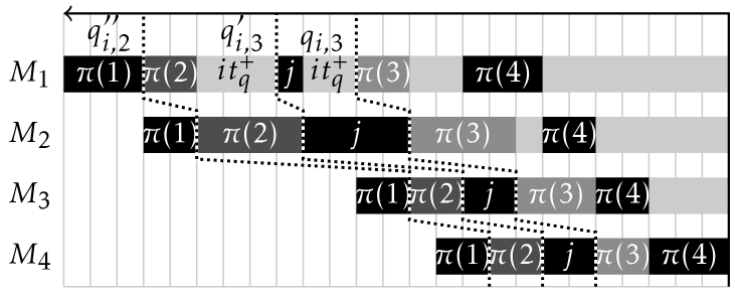


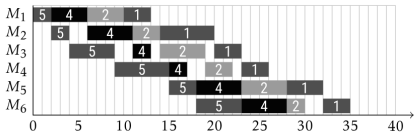
Figura 6: Tiempos de iniciación más tardíos

Taillard vió que podía usarlos para calcular el *Makespan*. Lo que propuso es realizar los siguientes cálculos para todos los trabajos en cada máquina.

1. Calcular los **tiempos de finalización más tempranos**

$$\pi_0 = (J_4, J_5, J_1, J_2, J_3, J_6) \qquad \pi = (J_5, J_4, J_2, J_1) \qquad Next\ work : J_3$$

$$Calcule\ e_{k,i} = \max(e_{k,i-1},\ e_{k-1,i}) + p_{\pi(k),i} \qquad para\ i \in [m], k \in [|\pi|] \qquad con\ e_{0,i} = e_{k,0} = 0$$



$e_{k,i}$	i						
	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	2	4	9	15	18	23
k 2	0	6	11	13	17	23	28
3	0	10	14	19	22	28	30
4	0	13	20	23	26	32	35

Figura 7: Matriz de los tiempos de finalización más tempranos

Recuerda: Los tiempos de finalización más tempranos $e_{k,i}$ antes de la posición de inserción no cambian

2. calcular los tiempos de finalización con el nuevo trabajo insertado en cada posible posición

$$\pi_0 = (J_4, J_5, J_1, J_2, J_3, J_6) \qquad \pi = (J_5, J_4, J_2, J_1) \qquad Nextwork : J_3$$

$$Calcule\ e'_{k,i} = \max(e'_{k,i-1},\ e_{k-1,i}) + p_{j,i} \qquad para\ i \in [m], k \in [|\pi|+1] \qquad con\ e'_{k,0} = 0$$

$e_{k,i}$	i						
	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	2	4	9	15	18	23
k 2	0	6	11	13	17	23	28
3	0	10	14	19	22	28	30
4	0	13	20	23	26	32	35

$e'_{k,i}$	i						
	0	1	2	3	4	5	6
1	0	6	11	13	15	17	21
2	0	8	13	15	17	20	27
k 3	0	12	17	19	21	25	32
4	0	16	21	23	25	30	34
5	0	19	25	27	29	34	39

Figura 8: Matriz de los tiempos de finalización más tempranos más J_3

Los tiempos de fin tempranos $e'_{k,i}$ se calculan con los tiempos $e_{k-1,i}$ y con

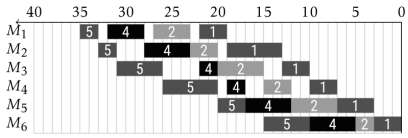
$p_{j,i}$	1	2	3	4	5	6
J_3	6	5	2	2	2	4

Figura 9: Tiempo de proceso del trabajo J_3

3. Calcular los **tiempo de iniciación más tardios**

$$\pi_0 = (J_4, J_5, J_1, J_2, J_3, J_6) \qquad \pi = (J_5, J_4, J_2, J_1) \qquad Nextwork : J_3$$

$$Calcule\ q_{k,i} = \max(q_{k,i+1},\ q_{k+1,i}) + p_{\pi(k),i} \qquad para\ i \in [m], k \in [|\pi|] \qquad con\ q_{|\pi|+1,0} = q_{k,m+1} = 0$$



$q_{k,i}$	i						
	1	2	3	4	5	6	7
1	35	33	31	26	20	15	0
2	32	28	22	19	17	10	0
k 3	27	23	20	15	12	5	0
4	22	19	13	10	7	3	0
5	0	0	0	0	0	0	0

Figura 10: Matriz de tiempos de iniciación más tardios

Recuerda: Los tiempo de inicio tardíos $q_{k,i}$ después de la posición de inserción no cambian

4. Teniendo las dos matrices de tiempos de finalización más tempranos y los tiempos de iniciación más tardios, calculamos la suma matricial (celda a celda) de estos. Luego calculamos el máximo valor posible de cada fila (comparamos sus columnas correspodientes), el valor resultante nos indica el *Makespan* de insertar el nuevo trabajo en dicha posición

$$\pi_0 = (J_4, J_5, J_1, J_2, J_3, J_6) \qquad \pi = (J_5, J_4, J_2, J_1) \qquad Nextwork: J_3$$

$$Calcule\ MC_k = \max_{i \in [m]}(e'_{k,i} + q_{k,i}) \qquad para\ k \in [|\pi| + 1]$$

$e'_{k,i}+q_{k,i}$	i					
	1	2	3	4	5	6
1	41	44	44	41	37	36
2	40	41	37	36	37	37
k 3	39	40	39	36	37	37
4	38	40	36	35	37	37
5	19	25	27	29	34	39

k	MC_k
1	44
2	41
3	40
4	40
5	39

Figura 11: Suma de las matrices y el mayor valor de cada fila

Seleccionamos la posición de inserción con el minimo *Makespan*

Podemos verlo de forma gráfica para poder captar la idea de Taillard

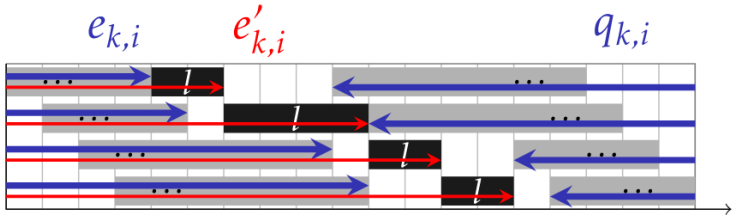


Figura 12: Resumen de los cálculos realizados

Estos cáculos evaluan n posiciones de inserción en un tiempo $O(nm)$. Con esta acelearción, NEH inserta los n trabajos en un tiempo $O(n^2m)$

Cuánto cueta la Heurística constructiva NEH Con esta aceleración, NEH inserta los n trabajos en un tiempo $0(n^2m)$

```
1: function NEH_Heuristic( )
2:     π_o = Priority_Order( )                          Ordenar cuesta O(n log n)
3:     π = (π_o(1))
4:     for k ∈ [2, n] do
5:         k' = Best_Insertion_Position(π, π_o(k))      Inserta O(n) trabajos
6:         π = (π(1),...,π(k'−1), π_o(k), π(k'),...,π(k−1))
7:     return π
```

```
1: function Best_Insertion_Position(π, j)
2:     Calculate values e_{i,k}, q_{i,k}, e'_{i,k}, and MC_k     Cada valor cuesta O(nm)
3:     K = {k' | MC_{k'} = mín_{k∈[n]}{MC_k}}
4:     k' = Tiebreaker(K)                                        Use la primer posición
5:     return k'
```

Figura 13: Complejidad temporal

2.2 Heurísticas de búsqueda local

Comienzan desde una solución inicial (puede ser aleatoria), intentan reemplazar la solución actual por una mejor solución vecina, repiten este paso hasta que no hayan mejores soluciones vecinas. La pregunta que que ayuda a diseñar correctamente la Heurística es ¿Qué cambio podría mejorar esta solución?

2.2.1 Que es la vecindad

Es un subcconjunto del conjunto toal y le asigna a S dicho conjunto

3 Metaheurísticas iterativas

3.1 Búsqueda local iterativa (ILS)

3.2 Algoritmo iterativo goloso (IG)

4 Preguntas y observaciones

- Acerca de la evaluación de la **Heurística constructiva**, supuestamente el resultado tenemos que compararlo con el rendimiento óptimo, ¿cómo podemos saber el óptimo si estamos en busca de ello?
- Acerca de la evaluación de la **Aceleración de Taillard**
    1. Cuando calculamos los tiempo de finalización más tempranos, ¿se realiza para cada posición posible de inserción? Si es asi, en el ejemplo se observa los resultados cuando se ha insertado al comienzo de todos los trabajos ¿Por qué cuando el nuevo trabajo lo inserto en la primera posición, ya se calcula los tiempos de finalización si lo inserto en las otras posiciónes? Una posible respuesta es que los tiempos de finalización más tempranos **antes de la posición de inserción no cambian**

2. Por qué se calcula el máximo tiempo de cada máquina y dicho valor será donde se inserte el nuevo trabajo

- Acerca de la **Busqueda local**

1. Cuando yo saco el trabajo y lo inserto calculando el mejor *Makespan* (usando la acelearción de Taillard) ¿estoy realizando teoricamente la inserción del minimo local de la vecindad actual (es decir, el orden de los trabajos con la inserción hecha)?
2. Por qué al realizar todas las inserciones posibles, tengo como resultado solo un óptimo local y no un óptimo global

5 Referencias

- Dr. Alexander Benavides