Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Informe de la Práctica 7 de DAA
Curso 2023-2024

Parallel Machine Scheduling Problem with Dependent Setup Times

Álvaro Fontenla León

La Laguna, 16 de abril de 2024

Licencia

Resumen

Resolución del problema de planificación de tareas en distintas máquinas de forma paralela con tiempos de preparación y ejecución dependientes utilizando los algoritmos voraz, GRASP y GVNS multiarranque.

Palabras clave: Scheduling optimization problem, Total Completion Time, GRASP, LS, VNS.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Contexto

En esta práctica nos encontramos ante un problema de planificación de tareas en distintas máquinas de forma paralela con tiempos de preparación y ejecución dependientes. El principal objetivo de esta práctica es minimizar el tiempo total de finalización de las tareas, maximizando la productividad.

1.1.1. Objetivos

Durante el desarrollo de esta práctica, se han cumplido los objetivos listados a continuación:

- Minimización del tiempo total de finalización (TCT).
- Desarrollo de diferentes algoritmos buscando una mayor optimalidad.
- Resolución de instancias de distintas características.

1.2. Motivación

La realización de esta práctica supone el primer paso en la resolución de problemas comunes en la vida real, fuera del ámbito educativo, lo cual es un desafío interesante y motivador.

Capítulo 2

Parallel Machine Scheduling Problem with Dependent Setup Times

2.1. Descripción

En esta práctica estudiaremos un problema de secuenciación de tareas en máquinas paralelas con tiempos de setup dependientes; Parallel Machine Scheduling Problem with Dependent Setup Times [1]. El objetivo del problema es asignar tareas a las máquinas y determinar el orden en el que deben ser procesadas en las máquinas de tal manera que la suma de los tiempos de finalización de todos los trabajos, es decir, el tiempo total de finalización (*TCT*), sea minimizado.

El tiempo de setup es el tiempo necesario para preparar los recursos necesarios (personas, máquinas) para realizar una tarea (operación, trabajo). En algunas situaciones, los tiempos de setup varían según la secuencia de trabajos realizados en una máquina; por ejemplo en las industrias química, farmacéutica y de procesamiento de metales, donde se deben realizar tareas de limpieza o reparación para preparar el equipo para realizar la siguiente tarea.

Existen varios criterios de desempeño para medir la calidad de una secuenciación de tareas dada. Los criterios más utilizados son la minimización del tiempo máximo de finalización (*makespan*) y la minimización del *TCT*. En particular, la minimización del *TCT* es un criterio que contribuye a la maximización del flujo de producción, la minimización de los inventarios en proceso y el uso equilibrado de los recursos.

El problema abordado en esta práctica tiene las siguientes características:

- Se dispone de *m* máquinas paralelas idénticas que están continuamente disponibles.
- Hay *n* tareas independientes que se programarán en las máquinas. Todas las tareas están disponibles en el momento cero.
- Cada máquina puede procesar una tarea a la vez sin preferencia y deben usarse todas las máquinas.
- Cualquier máquina puede procesar cualquiera de las tareas.
- Cada tarea tiene un tiempo de procesamiento asociado p_j.
- Hay tiempos de setup de la máquina s_{ij} para procesar la tarea j justo después de la tarea i, con $s_{ij} \neq s_{ji}$, en general. Hay un tiempo de setup s_{0j} para procesar la primera tarea en cada máquina.

- El objetivo es minimizar la suma de los tiempos de finalización de los trabajos, es decir, minimizar el TCT.

El problema consiste en asignar las *n* tareas a las *m* máquinas y determinar el orden en el que deben ser procesadas de tal manera que se minimice el TCT.

El problema se puede definir en un grafo completo $G = (V, A)$, donde $V = \{0, 1, 2, \cdots, n\}$ es el conjunto de nodos y A es el conjunto de arcos. El nodo 0 representa el estado inicial de las máquinas (trabajo ficticio) y los nodos del conjunto $I = \{1, 2, \cdots, n\}$ corresponden a las tareas. Para cada par de nodos $i, j \in V$, hay dos arcos $(i, j), (j, i) \in A$ que tienen asociados los tiempos de setup s_{ij}, s_{ji} según la dirección del arco. Cada nodo $j \in V$ tiene asociado un tiempo de procesamiento p_j con $p_0 = 0$. Usando los tiempos de setup s_{ij} y los tiempos de procesamiento p_j, asociamos a cada arco $(i, j) \in A$ un valor $t_{ij} = s_{ij} + p_j, (i \in V, j \in I)$.

Sea $P_r = \{0, [1_r], [2_r], \cdots, [k_r]\}$ una secuencia de $k_r + 1$ tareas en la máquina *r* con el trabajo ficticio 0 en el posición cero de P_r, donde $[i_r]$ significa el nodo (tarea) en la posición i_r en la secuencia *r*. Luego, el tiempo de finalización $C_{[i_r]}$ del trabajo en la posición i_r se calcula como $C_{[i_r]} = \sum_{j=1}^{i_r} t_{[j-1][j]}$. Tenga en cuenta que en el grafo *G* representa la longitud de la ruta desde el nodo 0 al nodo $[i_r]$.

Sumando los tiempos de finalización de los trabajos en P_r obtenemos la suma de las longitudes de las rutas desde el nodo 0 a cada nodo en P_r ($TCT(P_r)$) como:

$$TCT(P_r) = \sum_{i=1}^{k} C_{[i]} = kt_{[0][1]} + (k-1)t_{[1][2]} + \cdots + 2t_{[k-2][k-1]} + t_{[k-1][k]} \tag{2.1}$$

Usando lo anterior, el problema se puede formular como encontrar *m* rutas simples disjuntas en G que comienzan en el nodo raíz 0, que juntas cubren todos los nodos en *I* y minimizan la función objetivo.

$$z = \sum_{r=1}^{m} TCT(P_r) = \sum_{r=1}^{m} \sum_{i=1_r}^{k_r} (k_r - i + 1)t_{[i-1][i]} \tag{2.2}$$

Tenga en cuenta que el coeficiente $(k_r - i + 1)$ indica el número de nodos después del nodo en la posición $i_r - 1$.

2.1.1. Representación de las soluciones

La solución utilizada ha sido la propuesta en clase, la creación de un array de tareas para cada una de las máquinas, S={A_1, A_2,.., A_m}. En ellos se insertarán las tareas a ser procesadas en cada máquina en el orden establecido.

Capítulo 3

Algoritmos

3.1. Algoritmo consturctivo voraz

Un constructivo voraz

Un algoritmo constructivo voraz muy sencillo para este problema parte del subconjunto, S, formado por las m tareas con menores valores de t_{0j} asignadas a los m arrays que representan la secuenciación de tareas en las máquinas. A continuación, añade a este subconjunto, iterativamente, la tarea-máquina-posición que menor incremento produce en la función objetivo. El pseudocódigo de este algoritmo se muestra a continuación.

Algoritmo constructivo voraz

1: Seleccionar la m tareas $j_1, j_2, .., j_m$ con menores valores de t_{0j} para ser introducidas en las primeras posiciones de los arrays que forman la solucion S;
2: S = {A_1={j_1}, A_2={j_2},.., A_m={j_m}};
3: **repeat**
4: $S^* = S$;
5: Obtener la tarea-maquina-posicion que minimiza el incremento del TCT;
6: Insertarla en la posicion que corresponda y actualizar S^*;
7: **until** (todas las tareas han sido asignadas a alguna maquina)
9: Devolver S^*;

Figura 3.1: Algoritmo constructivo voraz

3.2. Búsquedas Locales

Las búsquedas locales son un apartado importante en la resolución de este problema ya que nos permitirá la generación de soluciones de mayor calidad, por lo tanto, se han de escoger métodos de búsqueda local de calidad. Teniendo esto en cuenta, se han desarrollado los siguientes métodos:

- Reinserción de tareas en la misma máquina.
- Reinserción de tareas entre máquinas.
- Intercambio de tareas en la misma máquina.
- Intercambio de tareas entre máquinas.

Estos cuatro métodos de búsqueda local nos permitirán realizar una búsqueda amplia sobre el espacio de soluciones haciendo uso de los algoritmos descritos más adelante.

3.3. GRASP

El algoritmo GRASP desarrollado se puede separar en dos partes: la fase constructiva y la fase de mejora.

3.3.1. Fase Constructiva

La fase constructiva de este algoritmo se ha desarrollado teniendo en cuenta la aleatoriedad necesaria y manteniendo un cierto nivel de optimalidad. Se ha utilizado el algoritmo voraz desarrollado en gran medida para esta fase. El único cambio realizado ha sido implementado en la línea 5 de 3.1.

En este caso, las tareas se ordenan según su TCT y se escoge de manera aleatoria entre los tres mejores candidatos, asegurando la aleatoriedad del algoritmo y manteniendo cierto nivel de optimalidad.

3.3.2. Fase de Mejora

Una vez construida una solución, se procede a su mejora, haciendo uso de las búsquedas locales mencionadas anteriormente.

Esta fase se ha implementado buscando la mayor optimalidad posible, ejecutando la fase de mejora múltiples veces, hasta que la mejor solución encontrada, no se pueda mejorar más.

Fase de mejora del algoritmo GRASP

1: Construir una solución parcial usando la fase constructiva del algoritmo;
2: **repeat**
3: S^* = BúsquedaLocal(S);
4: $if(S^* < S)\{$;
5: $S = S^*$;
6: $\}$;
7: **until** (la mejor solución no haya sido mejorada)
8: Devolver S;

Figura 3.2: Fase de mejora del algoritmo GRASP

3.4. GVNS

Este último algoritmo se ha desarrollado de manera que, al igual que el algoritmo GRASP, se puede separar en dos partes: la perturbación (shake) y el descenso (VND).

Algoritmo GVNS

1: $maxIterations, maxK, iterations = 0$;
2: $k = 0$;
3: **repeat**
4: S = GRASPConstruct();
5: $k = 1$;
6: **repeat**
7: $ShakeS$ = Shake(S, k);
8: $VNDS$ = VND(ShakeS);
9: if (VNDS <S) {
10: $S = VNDS$;
11: $k = 1$;
12: } else k++;
13: solutions.add(S);
14: $iterations$++;
15: **until** ($k = maxK$)
16: **until** ($iterations = maxIterations$)
17: sort($solutions$)
18: Devolver $solutions$[0];

Figura 3.3: Algoritmo GVNS

3.4.1. Perturbación

La perturbación juega un papel importante en este algoritmo ya que nos permite escapar de óptimos locales, por lo tanto, es crucial seleccionar un buen método de perturbación. El más adecuado sería la reinserción de tareas entre máquinas. Este método es el que más puede llegar a perturbar una solución.

3.4.2. VND

El descenso o búsqueda de mejora se ha desarrollado haciendo uso de las cuatro estructuras de entorno mencionadas en la búsqueda local. Se ha optado por el siguiente orden de ejecución de la búsqueda local sobre estas estructuras de entorno: reinserción entre máquinas, intercambio en la misma máquina, reinserción en la misma máquina e intercambio entre máquinas.

Algoritmo VND

1: $S^* = S$;
2: $k = 0$;
3: **repeat**
4: switch (k) {;
5: k = 0 ->S^{**} = ReinsertInter(S^*);
6: k = 1 ->S^{**} = SwapIntra(S^*);
7: k = 2 ->S^{**} = ReinsertIntra(S^*);
8: k = 3 ->S^{**} = SwapInter(S^*);
9: };
10: if ($S^{**} < S^*$) {
11: $S^* = S^{**}$;
12: $k = 0$;
13: } else k++;
14: **until** (k = 4)
15: Devolver S^*;

Figura 3.4: Algoritmo VND

Capítulo 4

Experimentos y resultados computacionales

4.1. Constructivo voraz

Algoritmo voraz				
Problema	m	Ejecución	TCT	$CPU(ms)$
$I40j\_2m\_S1\_1$	2	1	13739	1
$I40j\_2m\_S1\_1$	4	2	7572	1
$I40j\_2m\_S1\_1$	6	3	5552	0
$I40j\_2m\_S1\_1$	8	4	4558	0

Cuadro 4.1: Algoritmo voraz. Tabla de resultados

4.2. GRASP

GRASP				
Problema	m	Ejecución	TCT	$CPU(ms)$
$I40j\_2m\_S1\_1$	2	1	13667	1413
$I40j\_2m\_S1\_1$	4	2	7531	1163
$I40j\_2m\_S1\_1$	6	3	5416	578
$I40j\_2m\_S1\_1$	8	4	4367	661

Cuadro 4.2: GRASP. Tabla de resultados

4.3. GVNS

VNS				
Problema	m	Ejecución	TCT	$CPU(ms)$
$I40j\_2m\_S1\_1$	2	1	12791	140196
$I40j\_2m\_S1\_1$	4	2	6906	93454
$I40j\_2m\_S1\_1$	6	3	4971	59669
$I40j\_2m\_S1\_1$	8	4	3997	67330

Cuadro 4.3: GVNS. Tabla de resultados

4.4. Análisis compartativo entre algoritmos

Como podemos observar en las tablas de resultados, hay bastantes diferencias entre los algoritmos. La primera diferencia notable es el tiempo que consumen los algoritmos. Observamos que el algoritmo voraz tiene un consumo de tiempo ínfimo respecto a los otros dos algoritmos. En cuanto al consumo de tiempo del algoritmo GRASP, aunque es más lento que el voraz, es un tiempo más que aceptable en comparación al consumo de tiempo del algorimto GVNS, el cual es el algoritmo con mayor tiempo de ejecución de los tres utilizados, multiplicando por cien, el tiempo consumido por el algoritmo GRASP.

Por otro lado, los resultados arrojados por el algoritmo GVNS son los mejores, lo que justifica en gran medida su tiempo de ejecución, ya que mejora en un 25 % en promedio, la solución inicial. El segundo algoritmo con mejores resultados es el GRASP. Merece la pena indicar que, la búsqueda local con mejores resultados es el intercambio de tareas entre máquinas, lo que mejora la solución construida por el algoritmo alrededor de un 10 %, en comparación a las demás búsquedas locales. En último lugar, el algoritmo voraz, el cual, aunque su tiempo de ejecución sea el menor, es el algoritmo que arroja peores resultados.

Capítulo 5

Conclusiones y trabajo futuro

Una vez realizada la práctica y habiendo utilizado los nuevos algoritmos vistos en clase, me he dado cuenta de que el mundo de la resolución de problemas y la búsqueda de nuevos algoritmos puede ser bastante más complicado y fascinante de lo que originalmente me planteaba, ya que los algoritmos que había utilizado anteriormente no son comparables a la complejidad teórica de GRASP y VNS.

Por otro lado, como comentaba en la motivación, este problema nos acerca un poco más a los problemas que pueden ser planteados en la vida real, alejándonos de los problemas académicos que, son necesarios, pero, en mi opinión, este tipo de problemas son mucho más interesantes ya que nos plantean pensar un poco más allá de lo que estamos acostumbrados.

Bibliografía

[1] S. Báez, F. Angel-Bello, A. Alvarez, and B. Melián-Batista. A hybrid metaheuristic algorithm for a parallel machine scheduling problem with dependent setup times. *Computers and Industrial Engineering*, 131:295–305, 2019.