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

Roberto Carrazana Pernía

La Laguna, 16 de abril de 2024

Figura 1: Midjourney

La Figura 1 muestra una imagen creada por Midjourney recreando un ejemplo de aplicación real del problema abordado en esta práctica.

Agradecimientos

Quiero expresar mi sincero agradecimiento al profesorado por su compromiso en la enseñanza, demostrado, entre otras cosas, en la creación constante de material docente y su búsqueda perseverante de nuevas maneras para infundir conocimiento. Esta práctica no es más que otro ejemplo de su trabajo para proporcionarnos a los estudiantes un desafío con el que desarrollar nuestras capacidades y dar otro paso en nuestra carrera académica y profesional como programadores e ingenieros.

Licencia

Resumen

La presente práctica aborda el problema "Parallel Machines Scheduling" incluyendo tiempos de setup, y se ha propuesto encontrar soluciones óptimas o cercanas a óptimas mediante algoritmos aproximados: Voraz, GRASP (Greedy Randomized Adaptive Search Procedure) y GVNS (General Variable Neighborhood Search). Además, se estudian los resultados obtenidos en busca de conclusiones sobre los beneficios de usar cada uno de ellos.

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

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Contexto

El problema del Parallel Machines Scheduling, también conocido como programación de tareas paralelas en máquinas, es un desafío clásico en la optimización combinatoria y tiene importantes aplicaciones en la planificación y programación de sistemas de producción, logística, manufactura y procesamiento de datos, entre otros campos. En el contexto de esta práctica, el objetivo general es encontrar soluciones de alta calidad en un tiempo razonable mediante algoritmos aproximados.

1.1.1. Objetivos

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

- Obj1: Implementar un algoritmo voraz que proporciona soluciones de alta calidad en un tiempo reducido.
- Obj2: Desarrollar un algoritmo GRASP que permita refinar soluciones obtenidas mediante un enfoque voraz.
- Obj3: Diseñar un algoritmo GVNS que pone en práctica lo aprendido y previamente implementado para el algoritmo GRASP pero usando un enfoque diferente a la hora de explorar el espacio de soluciones.
- Obj4: Realizar experimentos computacionales para evaluar el rendimiento y la eficacia de los algoritmos implementados.

1.2. Motivación

La resolución eficiente de este problema es crucial para mejorar la productividad y la eficiencia en una amplia gama de entornos de producción, y además propone un desafío atractivo para aquellas personas con afinidad por la programación y el diseño de algoritmos.

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

Podemos generar un array por 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 constrictivo 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

El algoritmo de búsqueda local comienza con una solución inicial S y repite iterativamente un proceso de mejora hasta que no se pueda encontrar una mejora adicional. En cada iteración, se realiza un movimiento a una solución vecina S y se evalúa si este movimiento disminuye el tiempo total de finalización (TCT) de las tareas. Si el movimiento mejora la solución, se actualiza la solución actual; de lo contrario, se deshace dicho movimiento. Este proceso continúa hasta que no se logra una mejora en el TCT.

Búsqueda Local

1: S = Solución a mejorar;
2: **repeat**
3: $S^* = S$;
4: Realizar un movimiento a S^*;
5: Si el movimiento disminuye el TCT, entonces $S = S^*$;
6: En caso contrario deshacer el movimiento;
7: **until** (no se consiga una mejora)
8: Devolver S;

Figura 3.2: Algoritmo general para la búsqueda local

3.3. GRASP

La fase constructiva del algoritmo GRASP comienza seleccionando un conjunto inicial de tareas con los menores valores de tiempo de setup y las asigna a las primeras posiciones de la solución. Luego, de manera iterativa, se elige aleatoriamente una tarea candidata sin asignar y se determina la mejor máquina-posición que minimiza el incremento en el tiempo total de finalización. Esta tarea se inserta en la solución y se repite el proceso hasta que todas las tareas hayan sido asignadas a alguna máquina. Este enfoque permite explorar una amplia variedad de soluciones iniciales y tiene la flexibilidad de adaptarse a diferentes instancias del problema. A continuación se presenta el pseudocódigo de esta fase constructiva.

Fase constructiva para el algoritmo GRASP

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**
5: Crear lista de candidatos con las n mejores tareas sin asignar;
5: Elegir aleatoriamente una de las tareas candidatas;
5: Obtener la 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 3.3: Fase constructiva (GRASP)

El algoritmo GRASP combina la construcción voraz con una búsqueda local para encontrar soluciones de alta calidad. En este caso, realiza un número predeterminado de iteraciones(en esta implementación serían 100) y, en cada iteración, se ejecuta la fase constructiva descrita anteriormente para obtener una solución inicial (S^*) a la que aplicar una búsqueda local y mejorarla. Si la solución mejorada es mejor que la solución actual, esta última se actualiza.

Algoritmo GRASP

1: S = Solución vacía;
2: i = 1;
3: **repeat**
2: $i++$;
4: S^* = Fase constructiva GRASP;
5: S^* = BúsquedaLocal(S^*);
6: Si $S^* < S$ entonces $S = S^*$;
7: **until** (i == 100)
8: Devolver S;

Figura 3.4: Algoritmo GRASP

3.4. GVNS

El algoritmo GVNS combina elementos de búsqueda local con perturbaciones aleatorias para explorar de manera eficiente el espacio de soluciones. Al igual que el GRASP, realiza un número fijo de iteraciones y en cada iteración se genera una solución inicial mediante el algoritmo GRASP (sin aplicar búsquedas locales). Luego, se aplica un proceso iterativo de perturbación y búsqueda local. La fase de 'Shaking' realiza un número determinado de perturbaciones en la solución inicial para diversificar la búsqueda. Después de cada perturbación, se aplica la búsqueda local de gran vecindario (VND) para mejorar la solución obtenida. Si la solución obtenida es mejor que la solución actual, se actualiza la solución actual y se reinicia el contador de perturbaciones. De lo contrario, se incrementa el contador de perturbaciones. Este proceso se repite hasta que se alcanza el número máximo de perturbaciones. Finalmente, se devuelve la mejor solución encontrada.

Algoritmo GVNS

1: S = Solución vacía;
2: i = 0;
3: **repeat**
5: S^* = Solución GRASP sin búsquedas locales;
6: $k = 0$; // Número máximo de perturbaciones
7: **repeat**
8: Shaking(S^*, k);
9: VND(S^*);
10: Si $S^* < S$ entonces $S = S^*$ y $k = 0$;
11: En caso contrario $k++$;
12: **until** (k == kMax)
13: $i++$;
14: **until** (i == 100)
15: Devolver S;

Figura 3.5: Algoritmo GVNS

Capítulo 4

Experimentos y resultados computacionales

4.1. Constructivo voraz

El algoritmo voraz planteado es capaz de proporcionar soluciones con una calidad aceptable teniendo en cuenta la diferencia de tiempo con respecto al resto de algoritmos estudiados en esta práctica. De igual manera, es importante destacar que en la mayoría de ocasiones sus soluciones son peores a las del resto.

Algoritmo voraz

Problema	n	m	TCT	CPU(ms)	
$I40j\_2m\_S1\_1$	40	2	1038	1	
$I50j\_2m\_S1\_1$	50	2	1443	2	
$I60j\_2m\_S1\_1$	60	2	1589	5	
$I40j\_4m\_S1\_1$	40	4	559	2	
$I50j\_4m\_S1\_1$	50	4	737	3	
$I60j\_4m\_S1\_1$	60	4	835	3	
$I40j\_6m\_S1\_1$	40	6	381	1	
$I50j\_6m\_S1\_1$	50	6	505	2	
$I60j\_6m\_S1\_1$	60	6	574	4	
...	...	...	...	...	...

Cuadro 4.1: Algoritmo voraz. Tabla de resultados

4.2. Multiarranque

Los resultados de ejecutar las distintas búsquedas locales indica que estas son sensibles a los valores ”m” y ”n” del problema. Esto puede verse en los TCT obtenidos, donde para menor número de máquinas la búsqueda de inserción intra-máquina es la mejor, mientras que a mayor número de máquinas lo es la inserción inter-máquina. Los movimientos de intercambio intra-máquina proporcionan los peores resultados independientemente de los parámetros del problema, incluso a pesar de su elevado porcentaje de mejora para algunos casos.

Multiarranque

Problema	Tipo de búsqueda	TCT	CPU(ms)	Mejora promedio (%)
$I40j\_2m\_S1\_1$	$InsertionIntra$	952	408	22.24
	$SwapIntra$	987	209	8.2
	$InsertionInter$	987	342	17.72
	$SwapInter$	980	322	16.02
...	...	...	...	...
$I60j\_2m\_S1\_1$	$InsertionIntra$	1515	1062	34.88
	$SwapIntra$	1539	667	8.88
	$InsertionInter$	1530	885	13.98
	$SwapInter$	1538	1016	11.36
...	...	...	...	...
$I50j\_4m\_S1\_1$	$InsertionIntra$	702	297	10.96
	$SwapIntra$	719	237	3.2
	$InsertionInter$	694	738	26.14
	$SwapInter$	709	628	24.98
...	...	...	...	...
$I50j\_6m\_S1\_1$	$InsertionIntra$	485	300	7.66
	$SwapIntra$	496	210	4
	$InsertionInter$	478	698	25.06
	$SwapInter$	484	671	25.16
...	...	...	...	...
$I40j\_6m\_S1\_1$	$InsertionIntra$	350	227	4.22
	$SwapIntra$	354	183	1.24
	$InsertionInter$	346	443	30.28
	$SwapInter$	347	367	21.14
...	...	...	...	...
$I60j\_6m\_S1\_1$	$InsertionIntra$	528	409	7.3
	$SwapIntra$	544	348	2.36
	$InsertionInter$	536	1077	22.78
	$SwapInter$	536	969	28.28
...	...	...	...	...

Cuadro 4.2: Multiarranque. Tabla de resultados

4.3. GRASP

Las ejecuciones del algoritmo no muestran un cambio significativo en el TCT obtenido para distintos tamaños de lista de candidatos. En cuanto al tiempo de ejecución se puede ver un ligero incremento para un mayor tamaño de LRC. Una lista de candidatos más grande puede habilitar una peor selección de tareas en la fase constructiva, sin embargo el estudio de los resultados no confirma esta teoría y probablemente se deba al trabajo realizado por la búsqueda local.

GRASP					
Problema	n	$\|LRC\|$	Ejecución	TCT	CPU(ms)
$I70j\_4m\_S1\_1$	70	2	1	889	580
$I70j\_4m\_S1\_1$	70	2	2	890	586
$I70j\_4m\_S1\_1$	70	2	3	890	607
$I70j\_4m\_S1\_1$	70	2	4	882	610
$I70j\_4m\_S1\_1$	70	2	5	889	583
...	...	...	...	...	...
$I70j\_4m\_S1\_1$	70	3	1	900	567
$I70j\_4m\_S1\_1$	70	3	2	881	603
$I70j\_4m\_S1\_1$	70	3	3	885	645
$I70j\_4m\_S1\_1$	70	3	4	883	630
$I70j\_4m\_S1\_1$	70	3	5	887	635
...	...	...	...	...	...
$I70j\_4m\_S1\_1$	70	6	1	889	749
$I70j\_4m\_S1\_1$	70	6	2	895	706
$I70j\_4m\_S1\_1$	70	6	3	894	680
$I70j\_4m\_S1\_1$	70	6	4	882	718
$I70j\_4m\_S1\_1$	70	6	5	869	674

Cuadro 4.3: GRASP. Tabla de resultados

4.4. GVNS

El algoritmo GVNS es consistente en cuanto a los TCT obtenidos para distintas ejecuciones del mismo problema, pero es extremadamente lento para ciertas combinaciones. Por ejemplo, en la segunda mitad de la tabla se muestran ejecuciones para dos problemas modificando el valor de ”k”, y se aprecia una diferencia significativa en el tiempo de ejecución al incrementar este parámetro de 2 a 3. Otro factor que puede estar teniendo un gran impacto negativo en el rendimiento es el número de tareas, ya que todos los problemas con 60 tareas presentan los mayores valores de CPU.

GVNS

Problema	m	k_{max}	Ejecución	TCT	CPU(ms)
$I60j\_2m\_S1\_1$	2	2	1	1512	4970
$I60j\_2m\_S1\_1$	2	2	2	1531	4400
$I60j\_2m\_S1\_1$	2	2	3	1535	4689
$I60j\_2m\_S1\_1$	2	3	1	1515	7961
$I60j\_2m\_S1\_1$	2	3	2	1521	7225
$I60j\_2m\_S1\_1$	2	3	3	1521	5224
...	...	...	...	...	...
$I60j\_4m\_S1\_1$	4	2	1	786	2836
$I60j\_4m\_S1\_1$	4	2	2	789	2495
$I60j\_4m\_S1\_1$	4	2	3	799	2576
$I60j\_4m\_S1\_1$	4	3	1	782	3604
$I60j\_4m\_S1\_1$	4	3	2	780	3504
$I60j\_4m\_S1\_1$	4	3	3	802	3321
...	...	...	...	...	...
$I40j\_6m\_S1\_1$	6	2	1	353	894
$I40j\_6m\_S1\_1$	6	2	2	345	786
$I40j\_6m\_S1\_1$	6	2	3	360	239
$I50j\_6m\_S1\_1$	6	2	1	472	415
$I50j\_6m\_S1\_1$	6	2	2	479	318
$I50j\_6m\_S1\_1$	6	2	3	484	338
...	...	...	...	...	...
$I40j\_6m\_S1\_1$	6	5	1	346	1683
$I40j\_6m\_S1\_1$	6	5	2	349	1481
$I40j\_6m\_S1\_1$	6	5	3	339	1987
$I50j\_6m\_S1\_1$	6	5	1	473	3595
$I50j\_6m\_S1\_1$	6	5	2	480	2786
$I50j\_6m\_S1\_1$	6	5	3	476	2622

Cuadro 4.4: GVNS. Tabla de resultados

4.5. Análisis comparativo entre algoritmos

El algoritmo más rápido es el voraz, proporcionando soluciones con una calidad aceptable en un instante de tiempo. Por otro lado, tanto el GRASP como el GVNS siempre dan mejores resultados en cuanto a la calidad de la solución. GRASP posee el mejor balance pues sus resultados están a la par con los que brinda GVNS y en menos tiempo la gran mayoría de veces.

Capítulo 5

Conclusiones y trabajo futuro

Las principales conclusiones obtenidas son las siguientes:

- Algoritmo Voraz: Su calidad es limitada, especialmente en instancias del problema más complejas o con mayores dimensiones. A pesar de ello, puede ser útil como punto de partida para otros enfoques más complejos pues su gasto de recursos es mínimo.
- GRASP: El algoritmo GRASP mostró una capacidad prometedora para encontrar soluciones de alta calidad en un tiempo razonable. Su naturaleza adaptable y su capacidad para combinar elementos de búsqueda local y aleatoriedad lo hacen adecuado para problemas difíciles donde se requiere un equilibrio entre exploración y explotación del espacio de búsqueda.
- GVNS: Es eficaz para mejorar las soluciones encontradas por GRASP mediante la aplicación de perturbaciones aleatorias y búsqueda local de gran vecindario. Se observó que GVNS produce soluciones de alta calidad, incluso en problemas de gran tamaño. Sin embargo, las pruebas denotan cierta ineficacia en la implementación y no parece ser una opción razonable frente a GRASP.

Como trabajo futuro, se pueden considerar las siguientes propuestas:

- Explorar la aplicación de otros enfoques de optimización, como algoritmos genéticos o búsqueda tabú, para abordar el problema del Parallel Machines Scheduling.
- Realizar experimentos computacionales más extensos y detallados para evaluar el rendimiento de los algoritmos en una variedad más amplia de instancias del problema.
- Investigar y desarrollar variantes y mejoras del algoritmo GVNS, como ajustes en los parámetros, nuevas estrategias de perturbación o métodos de búsqueda local más eficientes.

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.