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

Thomas Edward Bradley

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

Maria Belen Melian Batista

Licencia

Resumen

El siguiente informe viene a redactar los conceptos, pseudocoódigos y resultados obtenidos al evaluar distribuciones optimas de tareas entre varias maquinas. El número tanto de tareas como máquinas es variable y cada tarea tiene asociado un tiempo de ejecución y varios tiempos de transición a dicha tarea partiendo de cualquier otro.

Palabras clave: Scheduling optimization problem, Total Completion Time, Machine, Task, Greedy, GRASP, LS, VNS, GVNS.

Índice general

1. **Introducción** **1**
   - 1.1. Contexto . . . 1
     - 1.1.1. Objetivos . . . 1
   - 1.2. Motivación . . . 1
2. **Parallel Machine Scheduling Problem with Dependent Setup Times** **2**
   - 2.1. Descripción . . . 2
     - 2.1.1. Representación de las soluciones . . . 3
3. **Algoritmos** **4**
   - 3.1. Algoritmo consturictivo voraz . . . 4
   - 3.2. Búsquedas Locales . . . 4
   - 3.3. GRASP . . . 5
   - 3.4. GVNS . . . 5
4. **Experimentos y resultados computacionales** **6**
   - 4.1. Constructivo voraz . . . 6
   - 4.2. GRASP . . . 6
   - 4.3. GVNS . . . 7
5. **Conclusiones y trabajo futuro** **9**

Índice de Figuras

1. Midjourney 2

3.1. Algoritmo constructivo voraz 4
3.2. Algoritmo constructivo grasp 5

Índice de Tablas

4.1. Algoritmo voraz. Tabla de resultados 6
4.2. GRASP. Tabla de resultados 7
4.3. GVNS. Tabla de resultados 8

Capítulo 1

Introducción

1.1. Contexto

Se deseo crear un programa que pueda resolver un problema PMS a traves de varios algoritmos. Devolviendo una tabla para cada uno de estos para permitir una posterior evaluación y comparación de los resultados obtenidos.

1.1.1. Objetivos

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

- Almacenamiento de un problema PMS y posterior calculo de la matriz t
- Desarrollo de un algoritmo Voraz para la resolución del problema
- Muestreo de resultados mediante tablas ordenadas, conteniendo la ejecuión para un directorio dado
- Facil implementación de cualquier otra tabla que se desee evaluar
- Desarrollo de un algoritmo GRASP para la resolución del problema
- Desarrollo de un algoritmo GVNS para la resolución del problema
- Las 4 busquedas locales empleadas en GRASP y GVNS
- Una pequeña optimización en las busquedas locales, empleando una evaluación del TCT a nivel de maquina en lugar de a nivel global (no se implemento la manera más optima en este caso)

1.2. Motivación

Trabajar en un proyecto de mayor escala para mejorar las competencias informaticas en un entorno más cercano a la programación en la vida real.

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 consturictivo 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

Se han implementado 4 busquedas locales, las cuales se detallaran a continuación:

- **Inserción Interna** - Trata de insertar cada tarea de una maquinas en todas las posiciones restantes de la misma
- **Inserción Externa** -Trata de insertar cada tarea de las maquinas en todas las posiciones posibles en otras maquinas
- **Intercambio Interno** - Trata de intercambiar todos las tareas de una maquian entre si

- **Intercambio Externo** - Trata de intercambiar todas las tareas de las maquinas por todas las tareas de otras

3.3. GRASP

Para el algoritmo GRASP empleado, tras experimentar con varios tanmaños de instancias (veces que corremos la fase constructiva) opte por 10 ya que era un buen balance entre buenos resultados y una ejecución rápida.

Algoritmo constructivo voraz

Procedure GRASP
Begin
Preprocesamiento
Repeat
Fase Constructiva(Solución); Usando para ello una lista de n candidatos
PostProcesamiento(Solución); Miramos optimos locales para obtener una nueva solución
Actualizar(Solución, MejorSolución); Si es mejor que la mejor encontrada, se guarda
Until (Se halla llegado a las iteraciones deseadas);
End.

Figura 3.2: Algoritmo constructivo grasp

3.4. GVNS

En cuanto a las instancias (veces que corremos la fase constructiva) opte por 100, ya que era un buen número para obtener resultados y no tardaba una cantidad de tiempo excesivo (por no optimizar las busquedas locales tanto como podria, llevar acabo las 1000 iteraciones simplemente llevaba demasiado tiempo).

Capítulo 4

Experimentos y resultados computacionales

4.1. Constructivo voraz

Algoritmo voraz					
Problema	n	m	Ejecución	TCT	$CPU(ms)$
I40j$_2$m$_S$1$_1$.txt	40	2	1	13821	1515
I40j$_4$m$_S$1$_1$.txt	40	4	2	7353	2598
I40j$_6$m$_S$1$_1$.txt	40	6	3	5487	3494
I40j$_8$m$_S$1$_1$.txt	40	8	4	4447	5419

Cuadro 4.1: Algoritmo voraz. Tabla de resultados

4.2. GRASP

En el caso a continuación se emplearon todas las busquedas locales, ejecutadas todas una tras otra sobre la S inicial para obtener el mejor resultado de estas. Se muestran ejecuciones con distintos valores de |LRC|.

Se puede apreciar una mejora respecto a la calidad de la solución comparando estos con los obtenidos por el Voraz, aunque esto no siempre se garantiza y en muchos casos se mantiene muy cerca de este.

GRASP

Problema	n	m	|LRC|	Ejecución	TCT	$CPU(ms)$
I40j$_2$m$_S$l$_1$.txt	40	2	2	1	13179	297391
I40j$_4$m$_S$l$_1$.txt	40	4	2	2	7243	339764
I40j$_6$m$_S$l$_1$.txt	40	6	2	3	5281	308509
I40j$_8$m$_S$l$_1$.txt	40	8	2	4	4294	326351
I40j$_2$m$_S$l$_1$.txt	40	2	3	1	13604	424285
I40j$_4$m$_S$l$_1$.txt	40	4	3	2	7242	377835
I40j$_6$m$_S$l$_1$.txt	40	6	3	3	5313	371937
I40j$_8$m$_S$l$_1$.txt	40	8	3	4	4339	376899
I40j$_2$m$_S$l$_1$.txt	40	2	4	1	13364	483051
I40j$_4$m$_S$l$_1$.txt	40	4	4	2	7357	486922
I40j$_6$m$_S$l$_1$.txt	40	6	4	3	5278	428305
I40j$_8$m$_S$l$_1$.txt	40	8	4	4	4351	437327

Cuadro 4.2: GRASP. Tabla de resultados

4.3. GVNS

Como se puede apreciar, los resultados aqui ya empiezan a mejorar significativamente sobre los encontrados en el Voraz o el GRASP. Aunque esto viene a un coste mucho más elevado, tardandose para estos ejemplos relativamente simples entre 10 y 35 segundos (mientras que todos los casos del grasp se mantenian por debajo de 1)

VNS

Problema	n	m	|LCR|	k_{max}	Ejecución	TCT	$CPU(ms)$
I40j$_2$m$_S$1$_1$.*txt*	40	2	2	3	1	12869	9825768
I40j$_4$m$_S$1$_1$.*txt*	40	4	2	3	2	6994	10200866
I40j$_6$m$_S$1$_1$.*txt*	40	6	2	3	3	5123	11083254
I40j$_8$m$_S$1$_1$.*txt*	40	8	2	3	4	4147	11883723
I40j$_2$m$_S$1$_1$.*txt*	40	2	2	5	1	12826	15920742
I40j$_4$m$_S$1$_1$.*txt*	40	4	2	5	2	6958	18411219
I40j$_6$m$_S$1$_1$.*txt*	40	6	2	5	3	5088	19760014
I40j$_8$m$_S$1$_1$.*txt*	40	8	2	5	4	4135	22593976
I40j$_2$m$_S$1$_1$.*txt*	40	2	2	7	1	12854	22894786
I40j$_4$m$_S$1$_1$.*txt*	40	4	2	7	2	6990	25807515
I40j$_6$m$_S$1$_1$.*txt*	40	6	2	7	3	5093	29497841
I40j$_8$m$_S$1$_1$.*txt*	40	8	2	7	4	4142	31742390
I40j$_2$m$_S$1$_1$.*txt*	40	2	3	3	1	12959	9965562
I40j$_4$m$_S$1$_1$.*txt*	40	4	3	3	2	7064	11173606
I40j$_6$m$_S$1$_1$.*txt*	40	6	3	3	3	5116	12891138
I40j$_8$m$_S$1$_1$.*txt*	40	8	3	3	4	4149	12334810
I40j$_2$m$_S$1$_1$.*txt*	40	2	3	5	1	12858	17201037
I40j$_4$m$_S$1$_1$.*txt*	40	4	3	5	2	6977	18433506
I40j$_6$m$_S$1$_1$.*txt*	40	6	3	5	3	5106	21373970
I40j$_8$m$_S$1$_1$.*txt*	40	8	3	5	4	4157	20138231
I40j$_2$m$_S$1$_1$.*txt*	40	2	3	7	1	12867	24518749
I40j$_4$m$_S$1$_1$.*txt*	40	4	3	7	2	7006	26315639
I40j$_6$m$_S$1$_1$.*txt*	40	6	3	7	3	5090	29011823
I40j$_8$m$_S$1$_1$.*txt*	40	8	3	7	4	4141	34848217
I40j$_2$m$_S$1$_1$.*txt*	40	2	4	3	1	12816	10678527
I40j$_4$m$_S$1$_1$.*txt*	40	4	4	3	2	7046	11020342
I40j$_6$m$_S$1$_1$.*txt*	40	6	4	3	3	5118	12554915
I40j$_8$m$_S$1$_1$.*txt*	40	8	4	3	4	4165	13431885
I40j$_2$m$_S$1$_1$.*txt*	40	2	4	5	1	12791	17434455
I40j$_4$m$_S$1$_1$.*txt*	40	4	4	5	2	7012	20439771
I40j$_6$m$_S$1$_1$.*txt*	40	6	4	5	3	5101	22921831
I40j$_8$m$_S$1$_1$.*txt*	40	8	4	5	4	4134	23622101
I40j$_2$m$_S$1$_1$.*txt*	40	2	4	7	1	12898	23335267
I40j$_4$m$_S$1$_1$.*txt*	40	4	4	7	2	6983	25236654
I40j$_6$m$_S$1$_1$.*txt*	40	6	4	7	3	5108	30638487
I40j$_8$m$_S$1$_1$.*txt*	40	8	4	7	4	4142	32187295

Cuadro 4.3: GVNS. Tabla de resultados

Capítulo 5

Conclusiones y trabajo futuro

Una de las cosas que más se puede apreciar tras llevar a cabo el proyecto es la importancia que se le debe dar a la discusión entre optimización y tiempo de ejecución. Hasta que punto estamos disponibles a optimizar nuestra solución y cuando el tiempo empleado sobrepasa los beneficios obtenidos.

EL GNVS es un ejemplo muy bueno de esto ya que es obvio desde el principio que supera los algoritmos previo a ello, pero encontrar ese punto medio, ese balance, de calidad de solución frente a tiempo de cálculo es una cuestión importante.

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.