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

Eric Bermúdez Hernández

La Laguna, 15 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.

Licencia

Resumen

Se ha abordado un desafiante problema de optimización de programación conocido como el "Problema de Programación de Máquinas Paralelas con Tiempos de Preparación Dependientes.º "Scheduling optimization problem", centrando su objetivo en la minimización del Tiempo Total de Completación o .ºtal Completion Time", por sus siglas TCT. Este enfoque integrado combina diversas técnicas avanzadas de optimización y metaheurísticas para desarrollar soluciones eficientes y efectivas. A través del uso del Procedimiento de Búsqueda Adaptativa Greedy Randomizada (GRASP), se generan soluciones iniciales robustas que luego son refinadas mediante la implementación de Búsqueda Local (LS), destacando un enfoque iterativo y adaptable para acercarse a la solución óptima.

Además, el trabajo integra la Búsqueda de Vecindario Variable General (VNS), ampliando la capacidad del sistema para explorar el espacio de soluciones más allá de los óptimos locales mediante la alteración sistemática de las estructuras de vecindario. Esta estrategia asegura una exploración más profunda y diversificada del espacio de búsqueda, facilitando el descubrimiento de soluciones de alta calidad que de otro modo podrían pasar desapercibidas.

La metodología implementada subraya una estrategia comprensiva que no solo se enfoca en la generación de soluciones iniciales mediante algoritmos codiciosos y aleatorizados sino que también enfatiza la importancia de la refinación de soluciones a través de técnicas de búsqueda local y la exploración estratégica del espacio de soluciones utilizando VNS. Esta combinación de técnicas subraya el esfuerzo por resolver el problema de optimización de programación de una manera que minimice el Tiempo Total de Completación, asegurando una asignación eficiente de tareas a las máquinas paralelas teniendo en cuenta los tiempos de preparación dependientes, un aspecto crítico para la optimización en entornos de producción y manufactura.

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

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 constructivo voraz . . . 4
   - 3.2. Búsquedas Locales . . . 5
   - 3.3. GRASP . . . 5
   - 3.4. GVNS . . . 6
4. **Experimentos y resultados computacionales** **7**
   - 4.1. Constructivo voraz . . . 7
   - 4.2. Multiarranque . . . 7
   - 4.3. GRASP . . . 9
     - 4.3.1. Algoritmo GRASP iterando 100 veces en la fase constructiva y con un límite de 50 intentos para las búsquedas locales . . . 9
     - 4.3.2. Algoritmo GRASP iterando 50 veces en la fase constructiva y con un límite de 5 intentos para las búsquedas locales . . . 10
     - 4.3.3. Conclusiones experimentación . . . 11
   - 4.4. GVNS . . . 12
   - 4.5. VNS . . . 12
   - 4.6. Análisis comparativo entre algoritmos . . . 12
5. **Conclusiones y trabajo futuro** **14**

Índice de Figuras

1. Midjourney 2

3.1. Algoritmo constructivo voraz 5

Índice de Tablas

Apartado 4.1 Constructivo voraz - página 7
Apartado 4.2 Multiarranque - páginas 7, 8
Apartado 4.3 GRASP con distintos parámetros y conclusiones - páginas 9, 10, 11
Apartado 4.4 GVNS - página 12
Apartado 4.5 VNS - página 12

Capítulo 1

Introducción

1.1. Contexto

Esta investigación aborda el desafío de optimizar el Tiempo Total de Completación en la programación de máquinas paralelas con tiempos de preparación dependientes del orden de las tareas, un problema clave en la gestión de operaciones industriales. El objetivo es desarrollar un marco de trabajo que reduzca los costes operativos y mejore la puntualidad en la entrega de productos, crucial en mercados competitivos. Se propone una solución que combina métodos de optimización avanzada, ofreciendo herramientas prácticas y adaptables para mejorar la eficiencia de la producción. Este proyecto busca contribuir tanto a la teoría de programación de operaciones como a la práctica industrial, promoviendo la excelencia operativa y la competitividad empresarial.

1.1.1. Objetivos

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

- Objetivo 1. Implementación algoritmo voraz y fase constructiva del algoritmo GRASP.
- Objetivo 2. Implementación del algoritmo GRASP con búsquedas locales.
- Objetivo 3. Práctica completa, incluyendo GVNS e informe en Latex.

1.2. Motivación

La optimización del Tiempo Total de Completación en la programación de máquinas paralelas es crucial para la industria manufacturera, pues mejora la productividad, reduce costos y agiliza la entrega de productos, aspectos esenciales en un mercado global competitivo. Este problema no solo presenta un desafío computacional y matemático por su complejidad, sino que también es vital para la eficiencia operativa y la sostenibilidad de las empresas. Abordarlo contribuye significativamente tanto a la gestión operacional como al avance académico en ingeniería industrial, estimulando la innovación y ofreciendo soluciones aplicables a diversos contextos industriales. La motivación detrás de resolver este problema radica en su potencial para mejorar la competitividad empresarial y enriquecer el conocimiento en la gestión de operaciones.

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

Breve descripción implementación

Para realizar la implementación, primero se creó una clase abstracta llamada “Algoritmo” de la que derivan los distintos algoritmos que resuelven el problema, siendo estos algoritmos el algoritmo voraz, GRASP y GVNS. Además de la clase abstracta “Algoritmo”, se implementaron dos clases abstractas llamadas “MovimientoEntre” y “MovimientoIntra”, de las que derivan las clases que se centran en implementar los algoritmos de búsqueda de soluciones vecinas. De la clase “MovimientoEntre” heredan las clases que realizan operaciones entre las máquinas que son “IntercambioEntre” e “InserciónEntre”, las cuales implementan las técnicas de búsquedas de soluciones locales que son intercambiar dos tareas de máquinas diferentes e insertar una tarea de una máquina en otra máquina diferente. Por otro lado, de la clase “MovimientoIntra” heredan las clases “IntercambioIntra” e “InserciónIntra” que implementan intercambios y reinserciones de tareas de la misma máquina respectivamente. En cuanto a las clases secundarias que ayudan para tener más organizado el código tenemos la clase “Problema” que a partir del fichero con la especificación del problema obtenemos los datos necesarios para la ejecución de los distintos algoritmos y también tenemos la clase “Solución” que contiene la solución obtenida de la ejecución de los algoritmos. Tanto la clase “Problema” como la clase “Solución” están contenidas en las clases “AlgoritmoVoraz”, “AlgoritmoGVNS” y “AlgoritmoGrasp”.

3.1. Algoritmo constructivo 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 se aplican para optimizar soluciones iniciales generadas por algoritmos constructivos voraces, mejorando el resultado mediante ajustes incrementales en la secuencia de tareas de las máquinas. Este método enfatiza en pequeñas modificaciones que, pese a su simplicidad, pueden llevar a una notable mejora en la minimización del Tiempo Total de Completación. Su relevancia radica en la capacidad de explorar eficientemente el espacio de soluciones cercanas, refinando la solución inicial hacia un óptimo más cercano. La combinación de un constructivo voraz, para obtener rápidamente una solución viable, y la búsqueda local, para perfeccionar esta solución, establece una estrategia eficaz para enfrentar complejos desafíos de programación de máquinas paralelas.

3.3. GRASP

El algoritmo GRASP (Greedy Randomized Adaptive Search Procedure) es una técnica de metaheurística que se utiliza para encontrar soluciones aproximadas a problemas de optimización complejos. Combina elementos de construcción greedy con la aleatorización para generar una solución inicial, seguido de una fase de búsqueda local que intenta mejorarla iterativamente. El equilibrio entre la aleatoriedad y el enfoque codicioso permite al GRASP explorar eficientemente el espacio de soluciones, evitando quedar atrapado en óptimos locales prematuramente.

La implementación que he realizado del algoritmo GRASP que se desprende del análisis de los archivos proporcionados, se sigue un enfoque estructurado que comienza con la creación de una solución inicial mediante un procedimiento constructivo que equilibra entre decisiones aleatorias y codiciosas. Esto implica seleccionar, de un conjunto de posibles candidatos, aquellos que tienen mayor probabilidad de contribuir a una solución óptima, basándose en una función de evaluación, pero introduciendo un factor aleatorio para permitir la diversidad en la solución inicial.

Una vez generada esta solución inicial, se aplica un proceso de búsqueda local mediante un algoritmo que se escoge entre Reinserción e intercambio de tareas entre máquinas y en una misma máquina, en cada algoritmo se examinan las soluciones vecinas a la actual con el objetivo de encontrar una mejora. Este proceso se repite, iterando entre la generación

de soluciones iniciales y la búsqueda local, hasta que se cumplen ciertos criterios de terminación, como puede ser una cantidad específica de iteraciones sin mejora. Por último, cabe destacar que no está optimizado el hecho de evaluar los movimientos, por lo que se calculan constantemente el TCT de las máquinas una vez se ha modificado por lo que hace que el método pese a encontrar una solución fáctible, lo hace en un tiempo que puede ser mucho más reducido con las modificaciones oportunas.

3.4. GVNS

El algoritmo General Variable Neighborhood Search (GVNS) es una estrategia de optimización que se basa en la premisa de que diferentes estructuras de vecindario pueden revelar diferentes óptimos locales. Por ello, alterna sistemáticamente entre varios vecindarios durante la búsqueda para escapar de óptimos locales y potencialmente encontrar soluciones globales o de mayor calidad. La implementación efectiva de GVNS requiere definir cuidadosamente estos vecindarios y cómo pasar de uno a otro, así como establecer un procedimiento de búsqueda local que refine las soluciones encontradas en cada vecindario.

En la implementación de este algoritmo, inicialmente, la implementación establece una solución base, generada por el método constructivo del algoritmo GRASP, que sirve como punto de partida para la exploración del espacio de soluciones.

La esencia de la aplicación GVNS en este contexto radica en cómo se ha definido y utilizado una variedad de vecindarios. Cada vecindario corresponde a un conjunto diferente de posibles modificaciones a la solución actual, como intercambios o reubicaciones de tareas entre máquinas, lo que permite explorar el espacio de soluciones desde diversas perspectivas. La transición entre estos vecindarios se gestiona de manera que, si una búsqueda en un vecindario particular no produce mejoras, el algoritmo amplía su búsqueda moviéndose a un vecindario más amplio o diferente.

Para cada vecindario seleccionado, se lleva a cabo una búsqueda local detallada, enfocándose en encontrar la mejor solución posible dentro de ese entorno específico. Este proceso no solo ayuda a mejorar la solución actual sino que también aumenta las posibilidades de salir de óptimos locales mediante el cambio estratégico a vecindarios alternativos.

Este enfoque iterativo y adaptativo continúa hasta que se cumplen ciertos criterios de terminación, que es un número específico de iteraciones sin mejora significativa. A través de este meticuloso proceso, la implementación del GVNS demuestra ser una herramienta poderosa para abordar el complejo problema de programación de máquinas paralelas, aprovechando la diversidad de los vecindarios para explorar efectivamente el espacio de soluciones y encontrar configuraciones óptimas o cercanas a óptimas. Por último, cabe destacar que no está optimizado el hecho de evaluar los movimientos, por lo que se calculan constantemente el TCT de las máquinas una vez se ha modificado por lo que hace que el método pese a encontrar una solución fáctible, lo hace en un tiempo que puede ser mucho más reducido con las modificaciones oportunas.

Capítulo 4

Experimentos y resultados computacionales

4.1. Constructivo voraz

Algoritmo voraz

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	15667	29 196 µs
I40j_4m_S1_1.txt	40	2	8477	13 312 µs
I40j_6m_S1_1.txt	40	3	6261	10 484 µs
I40j_8m_S1_1.txt	40	4	5265	11 216 µs
I50j_2m_S1_1.txt	50	5	28477	16 826 µs
I60j_2m_S1_1.txt	60	6	37774	29 589 µs
I70j_2m_S1_1.txt	70	7	47570	44 678 µs

4.2. Multiarranque

Multiarranque con Intercambio de tareas entre máquinas

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	14101	21 675 083 µs
I40j_4m_S1_1.txt	40	2	7749	16 263 095 µs
I40j_6m_S1_1.txt	40	3	5734	15 450 303 µs
I40j_8m_S1_1.txt	40	4	4569	16 151 176 µs
I50j_2m_S1_1.txt	50	5	25902	42 023 305 µs
I60j_2m_S1_1.txt	60	6	34784	72 417 131 µs
I70j_2m_S1_1.txt	70	7	45019	116 149 946 µs

Multiarranque con Intercambio de tareas en la misma máquina

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	14623	119 938 705 µs
I40j_4m_S1_1.txt	40	2	8214	51 379 472 µs
I40j_6m_S1_1.txt	40	3	5836	35 353 223 µs
I40j_8m_S1_1.txt	40	4	4829	30 144 475 µs
I50j_2m_S1_1.txt	50	5	26612	249 180 824 µs
I60j_2m_S1_1.txt	60	6	36130	463 980 654 µs
I70j_2m_S1_1.txt	70	7	46299	721 074 408 µs

Multiarranque con Reinserción de tareas entre máquinas

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	14971	18 610 073 µs
I40j_4m_S1_1.txt	40	2	8363	15 190 284 µs
I40j_6m_S1_1.txt	40	3	5965	14 881 684 µs
I40j_8m_S1_1.txt	40	4	4636	15 285 780 µs
I50j_2m_S1_1.txt	50	5	27264	36 247 968 µs
I60j_2m_S1_1.txt	60	6	37328	60 906 776 µs
I70j_2m_S1_1.txt	70	7	46886	98 513 630 µs

Multiarranque con Reinserción de tareas en la misma máquina

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	14571	19 662 915 µs
I40j_4m_S1_1.txt	40	2	8160	14 935 002 µs
I40j_6m_S1_1.txt	40	3	5972	15 018 311 µs
I40j_8m_S1_1.txt	40	4	4684	15 271 356 µs
I50j_2m_S1_1.txt	50	5	27332	36 924 446 µs
I60j_2m_S1_1.txt	60	6	36580	63 307 262 µs
I70j_2m_S1_1.txt	70	7	47037	99 448 526 µs

4.3. GRASP

4.3.1. Algoritmo GRASP iterando 100 veces en la fase constructiva y con un límite de 50 intentos para las búsquedas locales

GRASP con Reinserción de tareas entre máquinas

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	14246	971 163 µs
I40j_4m_S1_1.txt	40	2	7934	1 009 670 µs
I40j_6m_S1_1.txt	40	3	5886	1 097 917 µs
I40j_8m_S1_1.txt	40	4	4732	1 197 248 µs
I50j_2m_S1_1.txt	50	5	27815	1 958 123 µs
I60j_2m_S1_1.txt	60	6	35768	2 978 407 µs
I70j_2m_S1_1.txt	70	7	46963	4 691 796 µs

GRASP con Intercambio de tareas en la misma máquina

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	15720	1 366 748 µs
I40j_4m_S1_1.txt	40	2	8757	1 276 152 µs
I40j_6m_S1_1.txt	40	3	6336	1 251 469 µs
I40j_8m_S1_1.txt	40	4	5056	1 372 527 µs
I50j_2m_S1_1.txt	50	5	28057	2 769 502 µs
I60j_2m_S1_1.txt	60	6	36451	4 362 224 µs
I70j_2m_S1_1.txt	70	7	48132	6 861 085 µs

GRASP con Reinserción de tareas en la misma máquina

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	15581	1 000 824 µs
I40j_4m_S1_1.txt	40	2	8581	1 062 215 µs
I40j_6m_S1_1.txt	40	3	6096	1 158 819 µs
I40j_8m_S1_1.txt	40	4	4863	1 203 339 µs
I50j_2m_S1_1.txt	50	5	28493	1 783 737 µs
I60j_2m_S1_1.txt	60	6	38270	3 005 365 µs
I70j_2m_S1_1.txt	70	7	47905	4 746 222 µs

GRASP con Intercambio de tareas entre máquinas

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	14744	1 369 322 µs
I40j_4m_S1_1.txt	40	2	8274	1 183 447 µs
I40j_6m_S1_1.txt	40	3	6252	1 185 333 µs
I40j_8m_S1_1.txt	40	4	5049	1 237 050 µs
I50j_2m_S1_1.txt	50	5	25987	2 547 280 µs
I60j_2m_S1_1.txt	60	6	37293	4 447 438 µs
I70j_2m_S1_1.txt	70	7	45805	7 561 604 µs

4.3.2. Algoritmo GRASP iterando 50 veces en la fase constructiva y con un límite de 5 intentos para las búsquedas locales

GRASP con Reinserción de tareas entre máquinas

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	15378	502 721 µs
I40j_4m_S1_1.txt	40	2	8631	532 515 µs
I40j_6m_S1_1.txt	40	3	5933	622 515 µs
I40j_8m_S1_1.txt	40	4	4999	627 356 µs
I50j_2m_S1_1.txt	50	5	26463	955 798 µs
I60j_2m_S1_1.txt	60	6	38085	1 553 439 µs
I70j_2m_S1_1.txt	70	7	46574	2 403 521 µs

GRASP con Intercambio de tareas en la misma máquina

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	15554	603 821 µs
I40j_4m_S1_1.txt	40	2	8678	565 030 µs
I40j_6m_S1_1.txt	40	3	6262	584 149 µs
I40j_8m_S1_1.txt	40	4	5091	629 983 µs
I50j_2m_S1_1.txt	50	5	28523	1 053 083 µs
I60j_2m_S1_1.txt	60	6	37993	1 955 873 µs
I70j_2m_S1_1.txt	70	7	47806	2 714 861 µs

GRASP con Reinserción de tareas en la misma máquina

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	15284	520 450 µs
I40j_4m_S1_1.txt	40	2	8627	561 960 µs
I40j_6m_S1_1.txt	40	3	6235	592 649 µs
I40j_8m_S1_1.txt	40	4	5121	645 021 µs
I50j_2m_S1_1.txt	50	5	28155	969 894 µs
I60j_2m_S1_1.txt	60	6	37861	1 596 624 µs
I70j_2m_S1_1.txt	70	7	48070	2 489 539 µs

GRASP con Intercambio de tareas entre máquinas

Problema	n	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	40	1	15106	579 426 µs
I40j_4m_S1_1.txt	40	2	8193	587 103 µs
I40j_6m_S1_1.txt	40	3	6203	622 176 µs
I40j_8m_S1_1.txt	40	4	5012	686 472 µs
I50j_2m_S1_1.txt	50	5	26517	1 138 711 µs
I60j_2m_S1_1.txt	60	6	36532	2 043 970 µs
I70j_2m_S1_1.txt	70	7	45098	3 140 344 µs

4.3.3. Conclusiones experimentación

Viendo los resultados de las tablas, podemos sacar como conclusión que en todos los casos, debido al mayor número de iteraciones, el primer conjunto de resultados tarda el doble que el segundo conjunto. Además, en el primer conjunto de resultados, el TCT hallado es mejor que en el segundo cuando se ejecuta GRASP con Reinserción de tareas en la misma máquina y aunque no en todos los casos, también cuando se ejecuta con Reinserción de tareas entre máquinas. Por otro lado, cuando se ejecuta GRASP con Intercambio de tareas en la misma máquina el segundo conjunto obtiene mejores resultados y cuando se ejecuta GRASP con Intercambio de tareas entre máquinas, algunos resultados son mejores en el primer conjunto y otros en el segundo.

4.4. GVNS

GVNS

Problema	m	k_{max}	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	2	4	1	14329	206 050 054 µs
I40j_4m_S1_1.txt	3	4	2	7843	185 317 785 µs
I40j_6m_S1_1.txt	4	4	3	5729	112 608 845 µs
I40j_8m_S1_1.txt	6	4	4	4584	73 286 381 µs
I50j_2m_S1_1.txt	8	4	5	25868	449 007 101 µs
I60j_2m_S1_1.txt	2	4	6	35192	896 529 815 µs
I70j_2m_S1_1.txt	2	4	7	45160	1 484 465 273 µs

4.5. VNS

VNS

Problema	m	k_{max}	Ejecución	TCT	CPU
I40j_2m_S1_1.txt	2	4	1	14317	19 590 980 µs
I40j_4m_S1_1.txt	3	4	2	7633	15 117 314 µs
I40j_6m_S1_1.txt	4	4	3	5697	9 175 256 µs
I40j_8m_S1_1.txt	6	4	4	4401	8 057 911 µs
I50j_2m_S1_1.txt	8	4	5	25736	181 601 596 µs
I60j_2m_S1_1.txt	2	4	6	34233	56 526 788 µs
I70j_2m_S1_1.txt	2	4	7	45336	271 588 257 µs

4.6. Análisis comparativo entre algoritmos

Haciendo un análisis comparativo entre los algoritmos implementados para resolver el problema propuesto podemos apreciar que el Algoritmo Voraz destaca por su rapidez, con tiempos de ejecución considerablemente menores en comparación con las variantes de GRASP y GVNS, lo que lo convierte en una opción atractiva para obtener rápidas estimaciones o soluciones iniciales. Sin embargo, esta velocidad viene a expensas de la calidad del TCT, que generalmente es superior al de los métodos más complejos.

Los algoritmos GRASP, tanto con intercambio de tareas entre máquinas como en la misma máquina, y con reinserción, muestran una notable mejora en el TCT comparado con el Voraz, pero a coste de tiempos de ejecución dramáticamente mayores debido a la pobre optimización de su implementación. Esta relación entre calidad de solución y tiempo de ejecución se acentúa especialmente en las variantes de GRASP con modificaciones más complejas de las tareas, evidenciando cómo la intensificación de la búsqueda local puede mejorar el TCT a expensas del tiempo computacional.

Las variantes de GRASP iterando 100 veces en la fase constructiva y con límites en las búsquedas locales presentan una mejora incremental en el TCT con respecto a sus contrapartes estándar, pero esta mejora es marginal en relación al aumento significativo en los tiempos de ejecución. Este fenómeno subraya la importancia de equilibrar la profundidad de la búsqueda con la eficiencia computacional.

Por otro lado, VNS y GVNS ofrecen resultados mixtos. Mientras VNS logra un buen balance entre el TCT y los tiempos de ejecución, siendo competitivo especialmente en escenarios con un mayor número de tareas y máquinas, GVNS incurre en tiempos de ejecución prohibitivamente altos, aunque en algunos casos logre mejoras en el TCT. Este contraste pone de manifiesto el impacto significativo que tiene la elección de estrategias de búsqueda y la configuración de parámetros en la eficiencia de los algoritmos de optimización.

Capítulo 5

Conclusiones y trabajo futuro

Durante la ejecución de esta práctica, he logrado comprender una serie de conceptos que anteriormente me resultaban ambiguos o a los cuales no les otorgaba la relevancia debida. Ha quedado claro que la implementación de mis algoritmos dista de ser perfecta, principalmente por la redundancia en los cálculos realizados. Esta repetición hace que los algoritmos sean notablemente lentos, aunque logren encontrar soluciones aceptablemente buenas en un marco temporal razonable. A través de esta experiencia, he aprendido la importancia de realizar una labor de optimización meticulosa al implementar soluciones para problemas específicos. Más allá de la mera optimización de operaciones, se destaca la relevancia de emplear las estructuras de datos adecuadas, ya que estas pueden influir significativamente en el tiempo de ejecución de los algoritmos. En mi caso particular, aunque he utilizado predominantemente vectores, reconozco que la selección de otras estructuras de datos podría resultar en una mayor eficiencia y marcar una diferencia notable.

Adicionalmente, deseo resaltar mi encuentro con el problema del "Parallel Machine Scheduling Problem with Dependent Setup Times". Este descubrimiento me ha permitido apreciar la complejidad del mundo real, donde es necesario abordar desafíos cotidianos mediante la investigación y la experimentación. En esta práctica, he reflexionado sobre cómo solucionar este problema, además de intercambiar opiniones y debatir con mis compañeros de clase acerca de sus implementaciones de los distintos algoritmos, proponiendo conjuntamente diversas mejoras y soluciones. Este proceso colectivo simula un ambiente de investigación, aunque finalmente cada uno aplique los algoritmos según su criterio y estilo personal. En resumen, a pesar de lamentar no haber logrado una solución más eficiente, me siento satisfecho con los resultados obtenidos y considero que he superado mis expectativas dentro de mis capacidades.

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.