Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Posgrado en Optimización

Estudio del problema de programación de la producción en un ambiente multi-propósito flexible con división de lotes

Proyecto de Investigación

Alumno:
Miguel Ángel Fernández Romero

Asesores:
Dr. Eric Alfredo Rincón García
Dr. Antonin Ponsich

Ciudad de México, México

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Agradezco al *CONACyT* por la beca otorgada para realizar mis estudios de maestría.

Agradezco a mis padres, hermanos y hermanas por el apoyo incondicional que me han brindado a lo largo de mi vida.

Agradezco a Liliana por la compañía que me ha brindado, por motivarme a dar un paso más y por compartir grandes momentos conmigo.

Agradezco a mis asesores y maestros el Dr. Antonin Ponsich, el Dr. Eric Alfredo Rincón García y el Dr. Roman Anselmo Mora Gutiérrez por sus consejos y el gran apoyo que me han dado a lo largo de todo el tiempo de conocerlos.

Agradezco a mis amigos del posgrado, Edwin, Gilberto, David, Lizbeth, Naim y Alejandro por la amistad otorgada.

Resumen

El problema de programación de tareas conocido como programación de la producción en un ambiente multi-propósito flexible con división de lotes es una variante del problema de tipo programación de la producción, en la cual los lotes pueden dividirse en sublotes de diferentes tamaños y asignarse a diferentes máquinas, de tal forma que se disminuyen los tiempos muertos y el tiempo total de procesamiento. El objetivo consiste en minimizar la amplitud de proceso, es decir, la fecha de terminación de la última operación en la última máquina.

Debido a la complejidad computacional de este problema, normalmente se recurre a técnicas heurísticas para poder resolverlo. En este trabajo, se propone un algoritmo que combina estrategias de Búsqueda Tabú, con vecindades y técnicas de división de lotes basados en la ruta crítica de cada solución generada. Para determinar la eficiencia del algoritmo propuesto, se adaptaron las instancias edata, rdata y vdata de Hurink [1]. Debido a que este problema casi no se ha reportado en la literatura, no fue posible encontrar soluciones, que sirvieran como punto de comparación, para las instancias antes mencionadas. Por lo tanto, se emplearon dos estrategias para poder evaluar el desempeño del algoritmo propuesto. Primero, se resolvieron las instancias propuestas hasta donde fue posible, con el solver Gurobi. Segundo, se emplearon los mejores resultados reportados, sin división de lotes, para este mismo conjunto de instancias. Los experimentos realizados muestran que el algoritmo propuesto es capaz de generar buenas soluciones en tiempos de cómputo aceptables. Por otro lado, se evidencian los beneficios de la estrategia combinando flexibilidad y división de lotes, introducida en este trabajo.

(Palabras clave: Programación de tareas, División de lotes, Heurísticas Búsqueda Tabú, Gurobi)

Abstract

The flexible job shop scheduling problem with lot streaming or lot splitting is a variant of the job shop scheduling problem, in which a job can be divided into sublots of different sizes and assigned to different machines, in such a way that processing times can be reduced. In this version the objective is to minimize the makespan.

Due to the computational complexity, heuristic techniques are usually used to solve this type of problem. In this thesis, we propose an algorithm that combines tabu search strategies, with specific neighborhoods and lot splitting techniques based on the critical path of each generated schedule. To determine the efficiency of the proposed algorithm, the Hurink's [1] instances edata, rdata, vdata were adapted to the lot splitting policy.

Since this problem has hardly been reported in the literature, it was not possible to find solutions to compare with, for the aforementioned instances. Therefore, two strategies were used to evaluate the performance of the proposed algorithm. First, the instances were solved as far as possible, with the Gurobi solver. Second, the best reported solutions, without lot streaming, were used as a reference for this set of instances. The experiments showed that the proposed algorithm is able to generate good solutions in an reasonable computing time. Besides, these results provide clear evidence regarding the benefits of strategy proposed in this work, combining flexibilty and lot streaming.

(Keywords: Flexible job shop, lot streaming, Job shop scheduling problem, Tabu Search, Gurobi)

Capítulo 1

Introducción

El problema de programación de tareas es un problema al cual todas las empresas de manufactura y de servicios se enfrentan en su día a día. Por ejemplo, una empresa que recibe de sus clientes varias órdenes de trabajo, donde cada orden requiere de una secuencia de operaciones para obtener el producto final de acuerdo a las especificaciones del cliente. En general, los problemas de programación de tareas pueden ser descritos como *asignar recursos, en el tiempo, a un conjunto de trabajos a realizar*[2]. Es importante mencionar que el término *trabajo* se puede referir, según el contexto de estudio o la aplicación considerada (producción, servicios, horarios), a un tipo de producto tangible o intangible como pueden ser una clase de actividades o de tareas. Por lo tanto, en el resto de esta tesis, se ocupará el término trabajo por considerar que de esta forma queda abierto a una gama más amplia de aplicaciones.

La programación de tareas dentro de una empresa es de gran importancia, ya que una buena programación mejora significativamente la productividad de la empresa y genera una buena imagen ante el cliente. Al contrario, una mala programación implica retrasos de los pedidos del cliente, lo cual puede provocar pérdidas importantes en el futuro. Esta clase de problemas también tiene gran importancia en otras áreas de aplicación como son: el tránsito aéreo al despegue y aterrizaje, en un grupo de clientes esperando recibir un servicio, un conjunto de programas a ser ejecutados en un centro de cómputo, o médicos y enfermeras cuidando pacientes en el hospital.

Por lo anterior, el problema ha sido de gran interés para empresas y para varios investigadores, generando el desarrollo de múltiples variantes representativas de sistemas reales, en las que se introducen restricciones adicionales, como: la fecha de entrega de los pedidos, tiempo de preparación, mantenimiento en las máquinas, costo de recursos, pedidos divididos, fechas de liberación de pedidos, entre otros. Asimismo, diversos objetivos pueden ser

formulados, por ejemplo: minimizar el tiempo final de producción, el costo total de la producción, tardanza o precocidad, costo de almacenaje, etc.

Frente a la variedad de modelos posibles, se suele establecer una clasificación de los problemas de acuerdo a la configuración de las líneas de producción. En algunas ocasiones, las órdenes de trabajo tienen que pasar por todas las máquinas, aunque no necesariamente siguen la misma secuencia de operaciones. Bajo estas condiciones, se trata de un problema al cual, durante el resto de esta tesis, se le llamará programación de la producción en un ambiente multi-propósito, mismo que será denotado como JSSP por sus siglas en inglés (Job Shop Scheduling Problem). En estos casos, se considera que cada operación sólo puede ser procesada en una máquina ya definida. Para una descripción completa y explicaciones detalladas sobre esta configuración, se refiere el lector a [2]. En [3], los autores demuestran que los problemas de programación JSSP son problemas que pertenecen a la clase NP-difícil.

Un caso diferente ocurre cuando una o varias operaciones pueden ser realizadas por diferentes máquinas, implicando una etapa previa de selección de la máquina en la que será procesada cada operación. Cuando un problema cumple con este conjunto de características, se dice que se trata de una versión flexible del JSSP, a la cual se le denotará como FJSSP, por sus siglas en inglés (Flexible Job Shop Scheduling Problem). Este funcionamiento confiere una mayor flexibilidad al sistema de producción al poder disminuir los tiempos muertos en las máquinas, y al reducir el tiempo de terminación de los trabajos. Si dos máquinas pueden procesar el mismo conjunto de operaciones y si el tiempo de procesamiento de todas las operaciones es igual en ambas máquinas, se dice que son máquinas similares o idénticas. Sin embargo, no en todos los casos se tienen máquinas idénticas, implicando que una operación puede tener tiempos diferentes de procesamiento según la máquina a la que haya sido asignada.

A su vez, varias empresas permiten dividir las órdenes de trabajo o lotes en sublotes. A esta estrategia se le denota como LS por sus siglas en inglés (Lot Streaming), y permite un aumento en el uso de las máquinas, ya que dos o más operaciones de un mismo trabajo dividido en sublotes pueden ser procesados simultáneamente en diferentes máquinas Este funcionamiento permite terminar el trabajo de manera anticipada. El lote puede ser dividido en sublotes del mismo tamaño o en sublotes de diferentes tamaños, que comparten las mismas características que el lote original, particularmente la misma secuencia de operaciones. LS se puede aplicar tanto a problemas tipo JSSP como FJSSP. En este último caso (denotado como FJSSP-LS), las operaciones pueden ser realizadas por diferentes máquinas y es necesario asignar a cada operación una máquina con la capacidad de procesarla, por lo general, los sublotes no siguen la misma secuencia en las máquinas. Además,

los sublotes son una división arbitraria de los lotes originales en fracciones cuyos número y tamaños debe determinar la estrategia de resolución. Al contrario, la cota superior sobre el número de sublotes forma parte de los datos de la instancia. Nótese que los sublotes no necesariamente seguirán la misma secuencia en las máquinas. Para estos casos, se conoce la demanda del trabajo y un número máximo de sublotes en los cuales se puede dividir. Cabe mencionar que esta clase de problemas, debido a su complejidad, casi no está estudiada y existe muy poca literatura especializada reportada al respecto.

Por lo anterior, las modalidades FJSSP, JSSP-LS y FJSSP-LS agregan elementos al problema JSSP, como son: libertades en la selección de las máquinas y en la división de lotes. Así, generar una solución del FJSSP-LS implica determinar una asignación válida de operaciones a máquinas, una división de lotes y resolver el JSSP resultante. Por lo cual estos problemas pueden verse como generalizaciones del JSSP.

Sin importar si se trata de una configuración de tipo JSSP, FJSSP, FJSSP-LS, es posible formular una variedad de objetivos, típicamente relacionados con los tiempos de terminación de las operaciones o bien con el retraso o la tardanza con respecto a fechas de entrega asociadas a cada una de las tareas. Sin embargo, normalmente se busca minimizar la amplitud de proceso (tiempo total de producción), conocido en inglés como makespan, ya sea por compromiso con el cliente, cuestiones económicas o valor agregado a su producto.

Diversos modelos matemáticos se han planteado para el JSSP con minimización de la amplitud de proceso. Entre los más conocidos se encuentran el modelo disyuntivo o de precedencias de Manne [4], el modelo basado en posiciones de Wagner [5] y finalmente el modelo de tiempos indexados de Bowman [6].

En [3] se demuestra que los problemas de tipo JSSP pertenecen a la clase NP-difícil. A pesar de que, en la literatura especializada, no existe una prueba formal de que las variantes FJSSP, JSSP-LS y FJSSP-LS son de tipo NP-difícil, varios autores [7, 8, 9, 10], han observado que el uso de paquetes computacionales especializados como CPLEX y Gurobi se ve restringido a la resolución de instancias pequeñas [11], motivando el uso de técnicas metaheurísticas para el tratamiento de instancias grandes o medianas.

Entre los métodos heurísticos que se mencionan con mayor frecuencia en la literatura se encuentran Algoritmos Genéticos (GA) [12], Optimización por Colonia de Hormigas (ACO) [13], Búsqueda en Vecindades Variables [14] y particularmente Búsqueda Tabú (TS). Se debe destacar que entre la literatura revisada, diferentes versiones de TS son las que han producido los mejores resultados sobre el JSSP, particularmente minimizando la amplitud

de proceso.
La primera implementación de TS sobre el JSSP fue introducida por Taillard [15], cuya principal aportación fue el uso de una estructura de vecindad, introducida anteriormente en [16], con frecuencia denotada como N1 [17]. Posteriormente, distintos esquemas de vecindades fueron propuestas e integradas en TS que también incluía una memoria de largo plazo para reinicializar la búsqueda a partir de soluciones élite [18] o una técnica de Reencadenamiento de Trayectorias [19] (PR por sus siglas en inglés, Path Relinking). Estas diferentes versiones de TS produjeron los mejores resultados sobre el JSSP, en problemas cuyo objetivo era minimizar la amplitud de proceso. Para los problemas FJSSP y JSSP-LS, se han introducido diferentes técnicas pero cabe mencionar que las mejores soluciones se han obtenido con las propuestas que integran una TS como mecanismo adicional de mejora local en las soluciones.

Por lo tanto, en el presente proyecto se propone desarrollar un modelo para el problema FJSSP-LS y diseñar e implementar una técnica de resolución basada en TS, que será posteriormente probada sobre una serie de instancias generadas en el marco del proyecto.

1.1. Objetivos

Se sabe que el JSSP es un problema NP-difícil, sin embargo el estudio de técnicas exactas, el desarrollo de software y el aumento en el poder de cómputo, hace posible el encontrar soluciones óptimas a problemas cada vez más grandes, en este caso se trabaja el FJSSP-LS, una parte interesante es conocer la capacidad de resolución de los equipos actuales y también es necesario generar soluciones de buena calidad en un tiempo razonable. Por lo que se establecen los siguientes objetivos.

1.1.1. Objetivo General

Proponer un modelo de Programación Matemática para el problema de programación JSSP flexible y con división de lotes (FJSSP-LS) y aplicar dos técnicas de optimización, un método exacto y una técnica metaheurística, para comparar su desempeño sobre un conjunto de instancias

1.1.2. Objetivos Particulares

1. Determinar las características particulares del FJSSP-LS para identificar los elementos constitutivos del modelo a producir.

2. Formular un modelo de Programación Matemática para el FJSSP-LS.

3. Adaptar un banco de instancias seleccionado de la literatura especializada al modelo generado en el objetivo anterior.

4. Aplicar un paquete de Programación Matemática al juego de instancias creado en el paso anterior.

5. Diseñar una técnica metaheurística adaptada para el tratamiento del FJSSP-LS.

6. Desarrollar un programa que genere soluciones aleatorias para el FJSSP-LS, y que mejore la calidad de las mismas aplicando la técnica diseñada en el inciso anterior.

7. Realizar el ajuste de parámetros para la metaheurística creada.

8. Efectuar experimentos computacionales con el programa desarrollado en los pasos anteriores sobre las instancias propuestas y comparar su desempeño con las soluciones óptimas o cotas producidas por el paquete de Programación Matemática.

El resto del presente documento se organiza de la siguiente manera. El capítulo 2 presenta el estado del arte sobre los problemas relacionados con programación de tipo JSSP y sus variantes, así como sobre Búsqueda Tabú. La definición del modo operativo y del modelo de Programación Mixta se incluye en el capítulo 3. En el capítulo 4, se describe el algoritmo basado en Búsqueda Tabú desarrollado para resolver el FJSSP-LS. Los experimentos computacionales y resultados obtenidos se presentan y discuten en el capítulo 5. Finalmente, las conclusiones y perspectivas para trabajos futuros se proponen en el capítulo 6.

Capítulo 2

Estado del Arte

2.1. Programación de la producción

El JSSP fue formalizado por Muth y Thompson en su libro (Industrial Scheduling) [20]. Se puede formular de la siguiente manera: dada una instancia del JSSP $n \times m$, en la cual n trabajos se deben procesar exactamente una vez en cada una de las m máquinas, el conjunto de n trabajos puede definirse como $J = \{J_1, ..., J_n\}$, mientras que el conjunto de m máquinas es $M = \{M_1, ..., M_m\}$. Cada trabajo se enruta a través de las m máquinas en un orden predefinido, que implica restricciones de precedencia. El procesamiento de un trabajo en una máquina se llama una operación y el procesamiento del trabajo i en la máquina j se denota por O_{ij}. Por lo que el conjunto de operaciones se puede definir como $\mathbf{O} = \{O_{ij} | i \in \{1, ..., n\}, j \in \{1, ..., m\}\}$. Una vez que se inicia su procesamiento, una operación no se puede interrumpir y no se permite la concurrencia es decir, el procesamiento O_{ij} no puede comenzar si el procesamiento O_{ij-1} no ha sido completado.

Cada operación de $O_{ij} \in \mathbf{O}$ en la máquina j del trabajo i debe tener un tiempo de procesamiento entero $p_{ij} \in \mathbb{N}$, lo que también se conoce como las limitaciones de procesamiento de la máquina.

Balas [21] representa el JSSP a través de una gráfica disyuntiva, en donde J representa el conjunto de trabajos y M el conjunto de máquinas. En la gráfica $G = (V, A, E)$, se tiene que $V = \{0, 1, 2, ...q\}$ es el conjunto de nodos que representan todas las operaciones, donde 0 y q representan el inicio y fin de las operaciones respectivamente. A es el conjunto de arcos conjuntivos (dirigidos), que conectan las operaciones consecutivas del mismo trabajo y E es el conjunto de arcos disyuntivos con operaciones de conexión para ser procesadas por la misma máquina $E = \cup_{k=1}^{m} E_k$, con E_k el subconjunto de arcos disyuntivos correspondientes a la máquina k. Zhang [22] lo muestra con el

siguiente ejemplo, utilizando tres trabajos y tres máquinas, los datos se muestran en la Tabla 2.1 y la gráfica obtenido se muestra en la Figura 2.1, donde las operaciones de j_1, j_2 y j_3 son {1,2,3}, {4,5,6} y {7,8,9} respectivamente.

Trabajo	(máquina, tiempo de procesamiento)		
j1	(1,3)	(2,2)	(3,5)
j2	(1,3)	(3,5)	(2,1)
j3	(2,2)	(1,5)	(3,3)

Tabla 2.1: Ejemplo de tres trabajos en tres máquinas

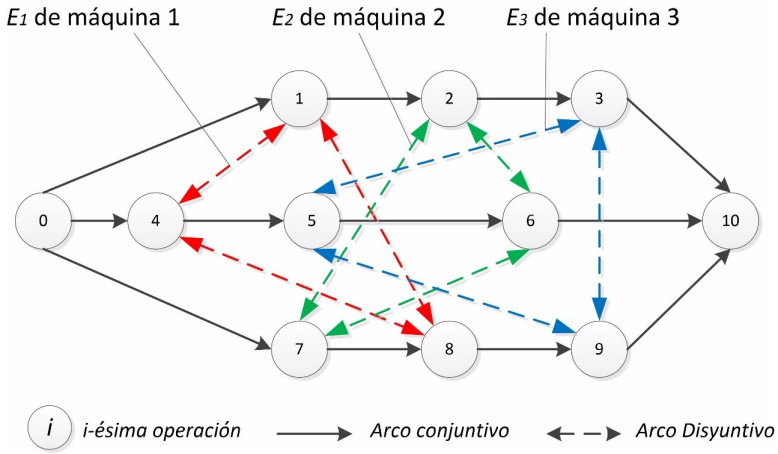


Figura 2.1: Gráfica disyuntiva de un problema con n=3, m=3 y q=10

En el ejemplo presentado, la ruta crítica (la ruta más larga de principio a fin en la gráfica dirigida del JSSP, cuya longitud representa la amplitud de proceso C_{max}) incluye las operaciones {0, 4, 1, 8, 9, 3, 10} y su longitud es 19. También es posible descomponer el camino crítico en un número de bloques. Un bloque es una secuencia máxima de operaciones críticas adyacentes procesadas consecutivamente en la misma máquina. Aquí, la ruta crítica se divide en dos bloques, B1 = {4, 1, 8} y B2 = {9, 3}.

De acuerdo a los formalismos presentados anteriormente, se han introducido diferentes modelos de programación lineal entera mixta:

- Modelo disyuntivo: este enfoque se basa en la variable binaria de precedencia z_{ijhgk}, introducido por Manne en 1960 [4]. Denota la secuencia

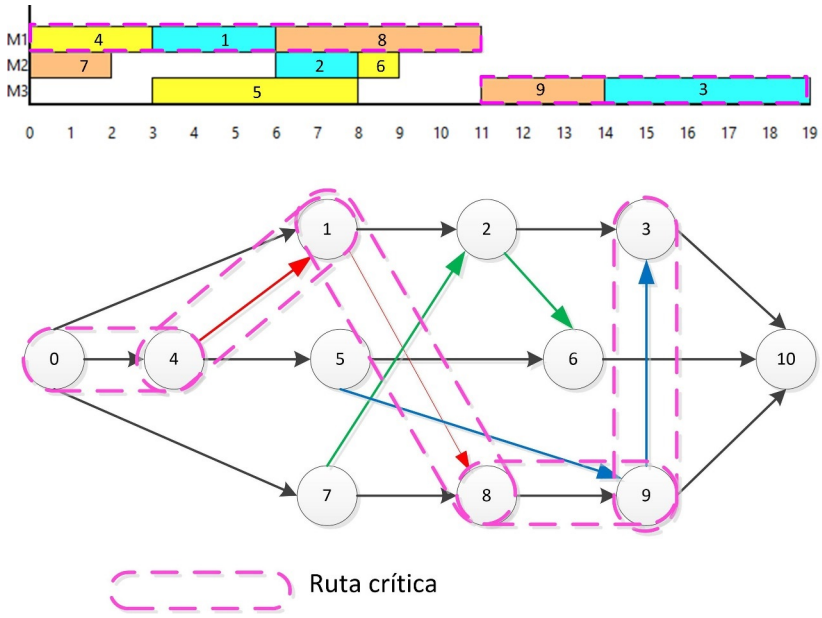


Figura 2.2: Ruta crítica y gráfico de Gantt del ejemplo con n=3 y m=3

de operaciones asignadas en la misma máquina. $z_{ijhgk} = 1$ si la operación O_{ij} precede a la operación O_{hg} en la máquina k, de lo contrario $z_{ijhgk} = 0$. La operación O_{ij} no necesariamente se coloca inmediatamente antes de la operación O_{hg} cuando $z_{ijhgk} = 1$. Este tipo de variables sólo se tiene que definir para $i < h$.

- Modelo basado en posiciones: para este caso, cada máquina tiene un conjunto de variables binarias, que especifican la posición en la que cada trabajo es procesado dentro de la máquina, generando así una secuencia de procesamiento. Estas variables son $x_{ijlk} = 1$ si la operación O_{ij} está asignada a la posición l en la máquina k; de lo contrario, $x_{ijkl} = 0$ [5].

- Modelo basado en tiempos indexados: bajo este enfoque, se asignan operaciones a los periodos unitarios de tiempo. Se emplea la variable binaria w_{ijuk} que es igual a 1 si la operación O_{ij} se procesa en el periodo u en la máquina k y es igual a cero en caso contrario. En el modelo de tiempos indexados, el horizonte de programación se considera discreto [6].

Por otro lado, el problema ha sido abordado con técnicas metaheurísticas y métodos exactos, entre las técnicas metaheurísticas destacan: Algoritmos Genéticos [23, 24, 25], Búsqueda Tabú [15, 22, 26, 27], Sistema Basados en

Agentes [28, 29], Optimización por Colonia de Hormigas [13, 30, 31, 32, 33], Redes Neuronales (NN) [34, 35] Optimización por Enjambre de Partículas (PSO) [36, 37], Búsqueda en Vecindades Variables (VNS) [14, 38], Colonia de Abejas Artificiales (ABC) [39], entre otras.

La gran variedad de trabajos realizados en este sentido permitió identificar que las técnicas de búsqueda local son las más eficientes para resolver el JSSP. Esta eficiencia está condicionada, sin embargo, por el uso de una vecindad adecuadamente diseñada. Dentro de la literatura revisada, una de las primeras vecindades propuestas consistía en generar todas las posibles inversiones de operaciones, consecutivas o no. Pero esta estructura de vecindad es ineficiente ya que implica un gran número de vecinos, de los cuales una mayoría se caracteriza por tener el mismo costo que la solución original. Posteriormente, en [16] Van Laarhoven menciona que un componente clave de una solución factible es la ruta crítica. En dicho trabajo, se incluyen algunas características importantes de la ruta crítica:

- Dada una solución factible, el intercambio de dos operaciones críticas adyacentes no puede producir una solución no factible.
- La permutación de las operaciones no críticas no puede mejorar la función objetivo e incluso puede crear una solución no factible.

De esta forma, Van Laarhoven introduce una estructura de vecindad embebida en una estrategia de Recocido Simulado, denotada como N1 en [17] y que fue posteriormente aprovechada en muchos otros trabajos de investigación. Esta vecindad se construye generando todos los posibles intercambios entre cualquier par de las operaciones críticas adyacentes en la misma máquina. Esta vecindad tiene un tamaño mucho más reducido que la vecindad simple descrita anteriormente e implica necesariamente variaciones en la función de costo.

Posteriormente, se propusieron algunas vecindades que se basan en movimientos de operaciones en la ruta crítica. Dentro de los más conocidos se encuentran la vecindad N4 propuesto por Dell'Amico y Trubian [40], Nowicki y Smutnicki exploran a través de N5 [18] y Balas y Vazacopoulos con N6 [41]. La vecindad N5 es particularmente famoso por haber sido integrado en la TS propuesta en [18]. N5 implica la inversión de un arco en una posición extrema de un bloque crítico (inicial y final) y es sustancialmente más pequeña que las otras vecindades. Por otro lado, las vecindades N4 y N6 invierten más de un arco disyuntivo a la vez, permitiendo explorar una vecindad considerablemente mayor.

Zhang [22] propone una estructura de vecindad extendida, en la que se define un movimiento al seleccionar la primera o última operación del bloque

crítico y al colocarla dentro del bloque [22]; o, de manera inversa, al seleccionar una operación interna de un bloque crítico y al colocarla al principio o al final del bloque.

Por otro lado, entre la diversidad de técnicas propuestas para resolver el JSSP destaca el Algoritmo Genético de clave aleatoria sesgado (BRKGA) presentado por Gonçalves y Resende [26] que fue capaz de obtener nuevas cotas en 57 instancias clásicas. Peng *et al.* [27] diseñaron una técnica de TS con PR. TS / PR obtiene resultados competitivos, demostrando su eficacia en términos de calidad de la solución y su eficiencia computacional. En particular, TS / PR es capaz de mejorar las cotas superiores para 49 de los 205 casos probados, y resuelve un ejemplo desafiante (SWV15) que había permanecido sin resolver durante más de 20 años.

Más recientemente, en [42], el autor trabajó con el objetivo de obtener una programación tal que el impacto en el rendimiento total sea menor en caso de interrupciones. Para ello, usó dos algoritmos basados en técnicas exactas y dos algoritmos basados en métodos heurísticos. Las estrategias exactas no fueron capaces de encontrar la solución óptima en instancias grandes, por lo cual en estos casos reportaron los resultados obtenidos por las técnicas heurísticas.

Wen y Christopher [43] realizan una comparación empírica de los siguientes modelos: modelo disyuntivo de Manne, modelo basado en posiciones de Wagner y modelo de tiempos indexados de Bowman, a través de los softwares CPLEX, Gurobi y SCIP. Los modelos y paquetes computacionales fueron aplicados a diferentes conjuntos de instancias, cada uno contiene 10 instancias, cuyos tamaños son 3×3, 4×3, 5×3, 3×6, 3×8, 3×10, 5×5, 8×8, 10×10, 15×15, 20×15, 20×20. La implementación del modelo de Manne fue capaz de resolver un mayor número de instancias en un tiempo preestablecido.

Entre las instancias comúnmente trabajadas sobre el JSSP se encuentran las siguientes:

- Fisher y Thompson [44], 3 instancias: FT06, FT10, y FT20.
- Lawrence [45], 40 instancias: LA01–40.
- Adams *et al.* [46], 5 instancias: ABZ5-9.
- Applegate y Cook [47], 10 instancias: ORB01–10.
- Yamada y Nakano [48], 4 instancias: YN1–4.
- Storer *et al.* [49], 10 instancias: SWV01–10.
- Taillard [15], 50 instancias: TA01-50.

- Demirkol *et al.* [50], 80 instancias DMU01-80.

2.2. Programación de la producción en un ambiente multi-propósito flexible

El JSSP clásico requiere la secuenciación de operaciones en máquinas determinadas, mientras que en el FJSSP la asignación de una operación no es fija de antemano y puede así ser procesada en una máquina seleccionada de un conjunto de máquinas con las características adecuadas. En el FJSSP, no sólo debe tenerse en cuenta la secuencia de las operaciones en cada máquina, sino también la asignación de operaciones a las máquinas adecuadas (enrutamiento). Por lo tanto, el FJSSP es más complejo que el JSSP y requiere un modelo nuevo o bien una adaptación de los modelos ya propuestos para JSSP. La programación de trabajos en el FJSSP se puede clasificar en los dos subproblemas siguientes [11]:

- Enrutamiento: consiste en seleccionar una máquina adecuada entre las disponibles para procesar cada operación.
- Programación: las operaciones asignadas se ordenan en todas las máquinas seleccionadas para obtener un calendario factible que minimice un objetivo predefinido.

Kacem *et al.* [51] clasifican los FJSSP en dos subproblemas basándose en la flexibilidad:

1. FJSSP total: cada operación puede ser procesada en cualquiera de las m máquinas de las estaciones de trabajo.
2. FJSSP parcial: algunas operaciones sólo son realizables en algunas de las m máquinas disponibles en las estaciones de trabajo.

Algunas medidas de desempeño para el FJSSP suelen ser la amplitud de proceso, la carga de trabajo en las máquinas, minimización de retrasos totales y minimización de costos de producción. Brucker y Schlie [52] fueron los primeros en abordar el FJSSP, que ha sido trabajado a través de diferentes técnicas metaheurísticas como son: Optimización por Colonia de Hormigas [33, 13, 12], Colonia Artificial de Abejas [39], Sistema Inmune Artificial [53, 54], GRASP (Greedy Randomized Adaptive Search Procedure) [55, 56] Búsqueda en Vecindades Variables [57, 14], PSO [58, 59, 60], Recocido Simulado [61, 62, 63], Algoritmos Genéticos [64, 65] y diversos métodos híbridos.

Al igual que para el JSSP, se han propuesto diferentes modelos de Programación Matemática y se han desarrollado estudios para comparar su eficiencia relativa. Las medidas que por lo general se toman son la amplitud de proceso, el tiempo computacional requerido, así como el número de variables y restricciones generadas por el modelo.

En [7], se realiza una comparación a través de cuatro modelos y propone un nuevo modelo de tiempos indexados, todos con el objetivo de reducir la amplitud de proceso y el esfuerzo computacional. Los modelos fueron probados con las instancias pequeñas de Fattahi [62] con hasta 4 trabajos, 4 operaciones y 5 máquinas (SFJS1-SFJS10) y de tamaño medio-grande con hasta 12 trabajos, 4 operaciones y 8 máquinas (MFJS1-MFJS10). Los diferentes modelos fueron resueltos con CPLEX, con un tiempo computacional límite de 3600 segundos. Los modelos sobre los cuales se efectuaron las pruebas son:

1. Modelo basado en variables de posición de secuencia. Este tipo de variables es propuesto por primera vez por Wagner para formular el JSSP. En cuanto al FJSSP, fue utilizado por Fattahi *et al.* [66] para formular el FJSSP.

2. Modelo basado en variables de precedencia: enfoque introducido por Manne, de los cuales hay tres tipos de enfoque para el FJSSP presentados a continuación:

   - El modelo propuesto por Özgüven *et al.* [67] para formular el FJSSP con y sin flexibilidad del plan de proceso.
   - El modelo presentado por Gao *et al.* [68] se construye con los tiempos de finalización de cada operación y una variable de precedencia, la cual define la precedencia de trabajos en una misma máquina.
   - Kim y Egbelu [69] formulan su modelo con variables de inicio de la operación en la máquina asignada y variables de precedencia.

3. Modelo de tiempos indexados propuesto en [7]: se asignan operaciones a los periodos de tiempo de la máquina con capacidad de procesamiento, la variable w_{ijku} que es igual a 1, si O_{ij} es procesado por la máquina k durante el periodo u, 0 en otro caso.

En su análisis, los autores de [7] determinan que el modelo basado en variables de precedencia propuesto por Özgüven presenta mayores ventajas,

ya que con él fue posible resolver un mayor número de instancias en un tiempo determinado. En segundo y tercer lugares aparecen los modelos de [69] y [68] respectivamente. Estos últimos modelos también basados en variables de precedencia, en términos de número de variables y restricciones son aproximadamente similares al de [67], obteniendo ventaja sobre los modelos de [66] y de [7].

Por otro lado, el FJSSP ha sido trabajado con diferentes técnicas metaheurísticas como son: Optimización por Colonia de Hormigas [33, 13, 12], Colonia Artificial de Abejas [39], Sistema Inmune Artificial [53, 54], GRASP [55, 56] Búsqueda en Vecindades Variables [57, 14], PSO [58, 59, 60], Recocido Simulado [61, 62, 63], Algoritmos Genéticos [64, 65] y diversos híbridos.

Una metaheurística que ha sido ampliamente trabajada para generar soluciones para el FJSSP es TS. En [11], se mencionan algunos trabajos en los cuales se utilizaron variantes de dicha técnica para minimizar la amplitud de proceso. En Brandimarte [70], se descompone el FJSSP en subproblemas, uno para la asignación de trabajos a máquinas y el segundo para la secuenciación, y se propone una técnica jerárquica basada en TS aplicada para minimizar la tardanza total ponderada, además de la amplitud de proceso. Esta estrategia se beneficia de los flujos de información entre las fases de enrutamiento y secuenciación, acelerando la convergencia. Por desgracia, fácilmente se queda atrapado en óptimos locales. Por otro lado, Hurink *et al.* [1] y Mastrolilli y Gambardella [71] desarrollaron métodos basados en TS, que comparados con el de [70] obtienen soluciones de mejor calidad gracias al diseño de vecindades eficientes. Ennigrou y Ghédira [72] proponen hibridizar TS con dos sistemas multi-agentes, incluyendo agentes de trabajo, agentes de recursos y un agente de interfaz. Se obtienen soluciones de calidad debido a estrategias específicas de diversificación. Vilcot y Billau [73] abordan el FJSSP para minimizar dos criterios, la amplitud de proceso y el retraso máximo. Introducen dos técnicas de TS multi-objetivo para encontrar un conjunto de soluciones no-dominadas. Jia *et al.* [74] utilizan una estrategia de TS con Reencadenamiento de Trayectorias (Path Relinking) para tratar un FJSSP Multi-objetivo (MOFJSSP), en el cual los objetivos a tratar son: minimización de la amplitud de proceso y la tardanza total ponderada, minimizar la carga de trabajo en las máquinas y la carga de trabajo aplicada al FJSSP.

Entre las instancias comúnmente trabajadas sobre el FJSSP se encuentran las siguientes:

- Kacem (2002), 4 instancias.
- Brandimarte´s (1993), 10 instancias: MK01 – MK10.
- Dauzère-Pérès and Paull (1997), 18 instancias.

- Barnes and Chambers (1994), 21 instancias.
- Hurink's (1994), 129 instancias adaptadas del JSSP (FT, LA).
- Fattahi (2007), 20 instancias (SFJS, MFJS).

2.3. División de lotes

La configuración de la división de lotes, también conocida como Lot Streaming (LS) o Lot Splitting, es un concepto en el que un lote de producción se divide en sub-lotes más pequeños para que su procesamiento en las estaciones sucesivas se pueda realizar en varias máquinas simultáneamente (producción en paralelo), permitiendo un progreso acelerado. De esta forma el problema implica determinar el número y tamaño de los sublotes que se emplearán. La división de lotes se ha aplicado en FSSP, JSSP y sistemas de ensamble [75]. Por ser de especial interés para este trabajo, a continuación se presenta una breve descripción de algunos artículos en los cuales se reporta la aplicación de LS al JSSP y al FJSSP.

Para JSSP-LS, se han encontrado pocos trabajos reportados en la literatura. Uno de ellos es el trabajo de Dauzère y Lasserre [76], que proponen una heurística iterativa para resolver el JSSP - LS al adoptar la heurística del "desplazamiento de cuello de botella" (*Shifting Bottleneck*).

Chinyao *et al.* [8] desarrollan un modelo para el JSSP-LS, en el cual buscan minimizar la amplitud de proceso y reducir el costo total de producción. Para el costo total, se considera el costo de manejo de materiales, costo de instalación y el costo de inventario. Una característica interesante de este trabajo es que se incluyen tiempos de preparación de las máquinas dependientes de la secuencia de precesamiento. Las pruebas que realizaron, fueron hechas sobre instancias pequeñas, 6 máquinas y 6 trabajos, a partir de las cuales obtienen las siguientes conclusiones:

1. Una asignación por lotes con sublotes de igual tamaño proporciona un mejor beneficio de tiempo (una menor amplitud de proceso) que la asignación de lotes con diferentes tamaños. Por otra parte, no es adecuado tener una gran diferencia entre los tamaños de sublotes.

2. La división proporciona un beneficio en la amplitud de proceso con respecto al caso sin división (un lote completo). Este beneficio se ve incrementado cuando aumenta el número máximo de sublotes permitidos. Sin embargo, la reducción de la amplitud de proceso se vuelve menor a medida que el número de sublotes aumenta.

3. El costo total está integrado en tres aspectos: el costo de instalación, el costo de manejo de materiales y el costo de inventario. Hay un incremento en el costo total con el número de sublotes en un entorno de producción de 3×3, pero con dos sublotes de igual tamaño se obtiene un menor costo total en entornos de producción de 4×4, 5×5 y 6×6

.

Buscher y Chen [9] resuelven el JSSP - LS considerando sublotes consistentes, es decir sublotes cuyo tamaño es constante durante toda la secuencia de procesamiento. Para ello, usa una técnica de tres fases: predeterminación de tamaños de sublotes, la determinación de los horarios en base a TS y la variación de tamaños de los sublotes. Utilizando LS o división de lotes, los autores logran reducir la amplitud de proceso en alrededor del 20 % para el 90 % de las instancias tratadas usando 2 sublotes.

Udo [77] introduce su modelo para el JSSP - LS , considerando tiempos de preparación de las máquinas. Se demuestra que aunque los tiempos de preparación implican una influencia negativa en la reducción de la amplitud de proceso, LS sigue siendo eficiente. Este modelo permite obtener mejores resultados en términos del tiempo y de la amplitud de proceso, para instancias en las que el modelo de Chinyao [8] no logra encontrar soluciones óptimas.

Defersha y Chen [10] formulan un modelo para el FJSSP - LS y desarrollan para resolverlo el Parallel Genetic Algorithm (PGA). El modelo considera varias cuestiones tales como: tiempos de preparación dependientes de la secuencia, la fecha de disponibilidad de la máquina y el retraso. Presentan ejemplos numéricos para demostrar las características del modelo. Saber [78] (2014) realiza su estudio basado en el trabajo de Defersha y Chen diseñando un híbrido entre GA y programación lineal (LP).

Novas [79] propone un modelo MILP para el FJSSP-LS, el cual tiene como medidas de desempeño la amplitud de proceso, el tiempo de finalización total de los sublotes y la tardanza de todos los lotes. El modelo MILP resultante fue implementado en CPLEX 12.5 (ILOG Optimization Studio). Debido a la ausencia de instancias benchmark, se adaptaron las instancias de Fattahi *et al.* [66], asignándoles una demanda y un número máximo de sublotes. Cabe mensionar que el trabajo de Novas [79] constituye una base fundamental para el desarrollo de esta tesis.

2.4. Técnicas metaheurísticas de resolución

Debido a que el JSSP es un problema NP-difícil, intentar resolver problemas de gran tamaño a través de algoritmos exactos (Programación lineal,

Programación Dinámica, Retroceso o Ramificación y Acotamiento) se vuelve ineficiente, al consumir demasiado tiempo y requerir una gran cantidad de recursos computacionales. Efectivamente, a medida que el tamaño del problema crece, el tiempo que se necesita para resolverlo por medio de técnicas exactas aumenta exponencialmente, por ello se llega a recurrir a técnicas metaheuríticas que son *estrategias inteligentes generales para diseñar o mejorar procedimientos heurísticos para la resolución de problemas con un alto rendimiento* [80]. Entre sus ventajas destaca la capacidad de dar una buena solución en un tiempo razonable. Sin embargo los métodos metaheurísticos tienen la desventaja de no poder garantizar la optimalidad del resultado proporcionado.

Una metaheurística empieza con una solución o una población de soluciones, que pueden generarse con diferentes estrategias, incluso de forma aleatoria y a través de operaciones de variación, produce soluciones nuevas. Cada metaheurística también cuenta con un mecanismo de reemplazo, que actualiza las soluciones en términos de su calidad respectiva, o de otros criterios. El proceso de búsqueda termina cuando se cumple un criterio de terminación, el cual puede ser un número máximo de iteraciones o cuando se comprueba la convergencia a un resultado deseado. Un primer tipo de metaheurísticas es conocido como metaheurísticas basadas en trayectorias (o de Búsqueda Local), debido a que trabajan con una única solución, cuyo camino en el espacio de búsqueda forma una trayectoria de la solución inicial a la final. Ejemplos de este tipo de metaheurísticas son: Búsqueda en Vecindades Variables, Recocido Simulado y TS.

Otra clase de metaheurísticas son las técnicas poblacionales, las cuales mejoran principalmente la calidad de las soluciones a través del intercambio de información entre ellas, algunas metaheurísticas poblacionales son Algoritmos Genéticos y optimización por enjambre de partículas (PSO). Una ventaja de las metaheurísticas poblacionales es que son buenas en el proceso de exploración del espacio de búsqueda. Sin embargo, el uso de poblaciones en un número grande de iteraciones genera un mayor número de llamadas a la función objetivo, por lo tanto, un mayor costo computacional que las metaheurísticas basadas en trayectorias.

Dentro de la gran variedad de metaheurísticas poblacionales y de búsqueda local que han sido aplicadas en los problemas de JSSP y sus variantes, destaca TS, motivo por el cual fue seleccionada para la realización de esta tesis.

2.5. Búsqueda Tabú

El término Búsqueda Tabú fue introducido por Fred Glover en el mismo artículo que introdujo el término metaheurística [80]. Los principios fundamentales de la búsqueda fueron elaborados en una serie de artículos de finales de los años 80 a principios de los 90, que fueron luego unificados en el libro "Tabu Search" [81]. TS es una metaheurística que guía un procedimiento heurístico de trayectorias en la búsqueda de optimalidad global, gracias a una memoria adaptativa que prohíbe temporalmente volver a soluciones ya visitadas, lo que permite la aparición de ciclos en la trayectoria de búsqueda y la exploración en nuevas regiones del espacio de búsqueda.

El procedimiento canónico de TS es simple. Se inicia con una solución factible almacenada como la solución actual y la mejor solución encontrada. Se producen entonces soluciones vecinas o candidatas, de acuerdo a una estructura de vecindad adaptada al problema tratado. Éstas son evaluadas por una función objetivo y el candidato con el mejor costo se selecciona como una nueva solución (paradigma del mejor vecino). Este mecanismo se llama un movimiento y el movimiento inverso se añade a una lista tabú para evitar ciclos en la trayectoria de búsqueda. Se actualiza la lista tabú y si la solución nueva es mejor que la mejor solución encontrada, se almacena como nueva mejor solución. Además, el concepto de aspiración permite romper el estatus tabú de algunos movimientos, cuando mejoran la solución actual.

Entre las implementaciones más importantes de TS en el JSSP, ya sea como técnica principal o como parte del procedimiento en la búsqueda local, se encuentran: la primera implementación de TS al JSSP, propuesta por Taillard [15], así como las estrategias $TSAB$[18] e $i-TSAB$[19] propuestas por Nowicki y Smutnicki. $i-TSAB$ es una extensión de $TSAB$ en la cual se presenta la estructura de vecindad $N5$, el cual consiste en el intercambio de operaciones extremas en el bloque crítico con sus operaciones adyacentes, también cuenta con una memoria de largo plazo, donde es guardada la información de las mejores soluciones y re-intensificación en torno a soluciones de alta calidad previamente encontradas, posteriormente con $i-TSAB$ intensifican la búsqueda a través de PR, entre las soluciones de alta calidad.

Posteriormente Zhang [22] implementa una nueva estructura de vecindad y la estrategia TS/SA [82], que llegó a superar a varias técnicas contra las que fue comparado. En esta técnica híbrida, el recocido simulado (SA) se utiliza para encontrar las soluciones de élite dentro del espacio de búsqueda, para que TS pueda re-intensificar la búsqueda entre las soluciones prometedoras.

Gonçalves y Resende [26] presentan el Algoritmo genético de claves aleatorias BRKGA, que implementa una versión extendida del método gráfico de Akers [83] y posteriormente integran a TS con la vecindad propuesta por

Nowicki y Smutnicki [18]. Esta técnica logra llegar al óptimo en todas las instancias FT y ORB, y en las instancias LA falla únicamente en LA29, quedando a una unidad del óptimo. Sin embargo logra superar a varias técnicas y establecer nuevos límites en 57 instancias (42 de DMU, 9 de TA, 1 de YN y 5 de SWV).

Bo Peng *et al.* [27] desarrollan una técnica de TS/PR que utiliza el TS de Cheng [84] y la vecindad propuesta por Zhang [22]. El TS/PR llega a ser una estrategia competitiva y establece 49 nuevas cotas en las instancias trabajadas, entre las cuales se encuentra el SWV15 que había permanecido sin resolver durante más de 20 años.

El uso de TS sobre problemas de programación de trabajos ha sido muy amplio, algunas otras implementaciones adicionales a las ya mencionadas anteriormente son la de Busher [9] para el JSSP-LS, o las hibridaciones propuestas en [85, 86, 87] para resolver el FJSSP.

Capítulo 3

Programación flexible con división de lotes

El trabajo de esta tesis se enfoca en el FJSSP-LS, un problema que ha sido escasamente reportado en la literatura a pesar de estar presente en la industria. Este problema hereda las características originales de JSSP y FJSSP, se considera que hay un conjunto de trabajos[1] J, un conjunto de máquinas K, cada trabajo es asociado con una demanda d_j, $j = 1, ..., n$, y puede ser dividido hasta en $MaxS_j$ $(j = 1, ..., n)$ sublotes. Cada operación requerida por cada sublote puede ser procesada en una máquina seleccionada del conjunto de máquinas $K_{i,j}$,de las máquinas capaces de procesar la $i-ésima$ operación del trabajo j. El tiempo de procesamiento de cada operación es proporcional al tamaño del sublote y la máquina seleccionada.
Esta tesis se enfoca a los trabajos con las siguientes características:

- Ningún trabajo tiene prioridad sobre otro.
- Los sublotes son consistentes, los sublotes de cada lote pueden tener diferentes tamaños, pero lo conservan durante cada operación en toda la ruta de procesamiento.
- Los sublotes son discretos, los componen partes o piezas discretas.
- Se desprecian los tiempos de preparación.
- Las máquinas no requieren mantenimiento durante el plan de trabajo.
- Ninguna máquina podrá procesar dos operaciones al mismo tiempo.

[1]Se debe recordar que el término trabajo se puede referir a un tipo de productos o una clase de actividades o tareas, de acuerdo a la aplicación considerada.

- Una vez que una máquina empieza a procesar un sublote, el procesamiento no puede ser interrumpido.

El objetivo a optimizar es encontrar un plan de trabajo o programa donde todos los trabajos hayan sido concluidos en el menor tiempo posible.

3.1. Instancias

Debido a que el FJSSP-LS es un problema relativamente nuevo, no existen instancias reportadas en la literatura, por lo cual se tomaron como referencia las instancias de Hurink [1]. Se debe destacar que estas instancias son el resultado de tres adaptaciones a instancias del JSSP, que a su vez fueron adaptaciones de las instancias propuestas por Lawrence [45], Fisher y Thompson [44].

Para generar instancias acordes a las características del problema, a las instancias de Hurik se les agregó una demanda y el número máximo de sublotes en los que puede dividirse cada lote. De esta forma, cada instancia propuesta contiene:

- $|J|$: Número total de trabajos.
- $|K|$: Número total de máquinas.
- d_j: Demanda del $j-ésimo$ trabajo.
- $MaxS_j$: Máximo de sublotes en el que se puede dividir el $j-ésimo$ trabajo.
- $|I_j|$: Número de operaciones demandadas por el lote j.
- $|K_{i,j}|$: Número de máquinas capaces de procesar la $i-ésima$ operación del trabajo j.
- $t_{i,j,k}$: Tiempo de procesamiento del lote j en la operación i en la máquina k.

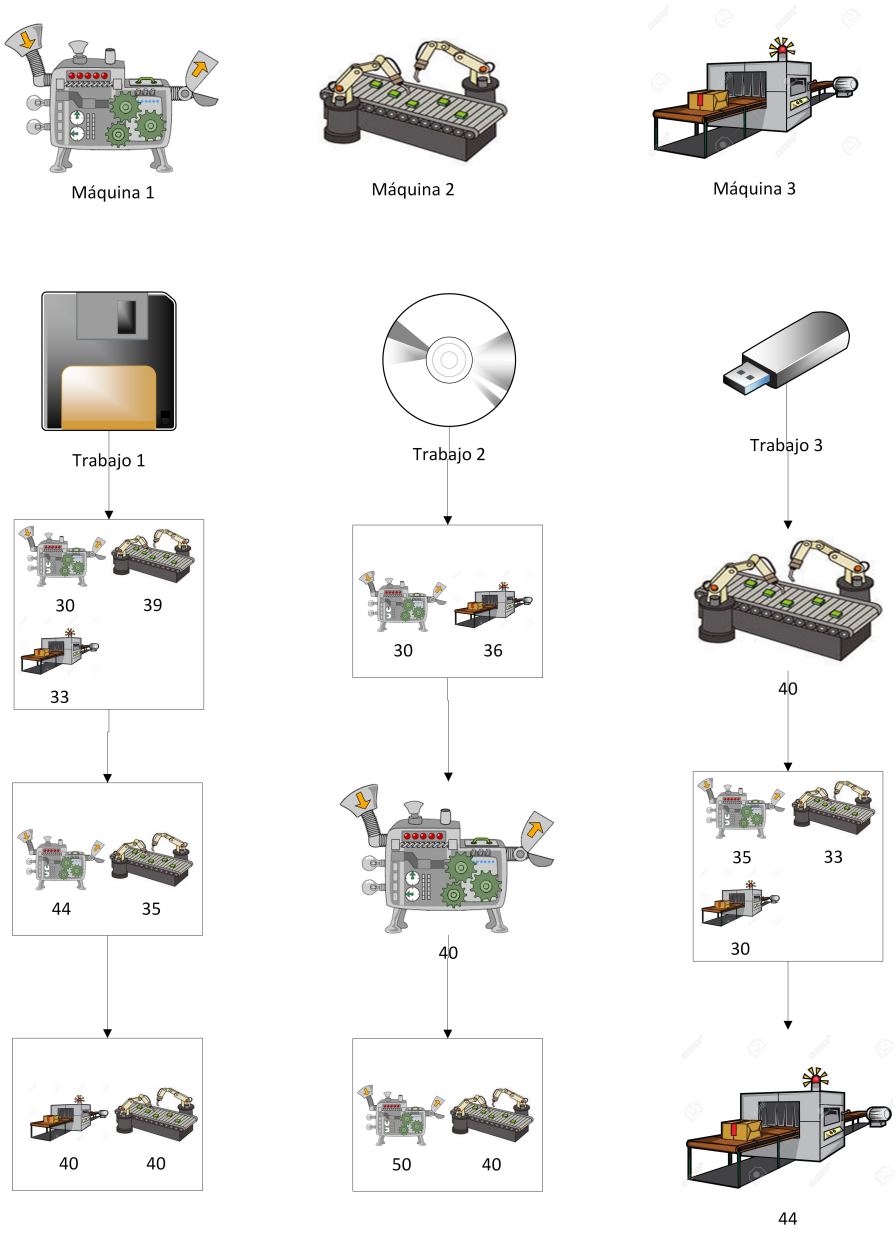
Para ejemplificar, consideremos una instancia con **3** trabajos y **3** máquinas, donde la demanda de los trabajos es **200, 450 y 400** piezas y el número máximo de sublotes son **2, 3 y 2** para cada trabajo respectivamente. Esta información es contenida en la primera linea de la instancia, mostrando primero el número de trabajos y número de máquinas, posteriormente demanda y máximo de sublotes de cada trabajo:

3	3	200	2	450	3	400	2

Tabla 3.1: Trabajos, máquinas, demandas y número máximo de sublotes

El resto de la instancia se mantiene igual a las instancias propuestas por Hurink. La secuencia de operaciones es dada como se muestra en la Figura 3.1. La segunda linea dentro de la instancia representa al trabajo 1, la tercera al trabajo 2 y así sucesivamente para los n trabajos.

Figura 3.1: Ejemplo de FJSSP



Para el ejemplo mostrado en la Figura 3.1, se tiene que el trabajo 1 está compuesto por **3** operaciones, la primera operación puede ejecutarse en **3** máquinas, las cuales son la máquina **1** con un tiempo de procesamiento de **30**, la máquina **2** con un tiempo de procesamiento de **39** o la máquina

3 con **33** unidades de tiempo. La segunda operación puede ejecutarse en **2**, la máquina **1** con un tiempo de procesamiento de **44**, o la máquina **2** con **35**. Para la última operación se cuenta con **2** máquinas, la máquina **2** o la máquina **3**, ambas con un tiempo de procesamiento de **40**.

3	3	1	30	2	39	3	33	2	1	44	2	35	2	2	40	3	40

Tabla 3.2: Información del trabajo 1

Resumiendo, para los productos 2 y 3 de la instancia de la Figura 3.1 queda como sigue:

3	3	200	2	450	3	400	2										
3	3	1	30	2	39	3	33	2	1	44	2	35	2	2	40	3	40
3	2	1	30	3	36	1	1	40	2	1	50	2	40				
3	1	2	40	3	1	35	2	33	3	30	1	3	44				

3.2. Modelo FJSSP-LS

En esta sección se presenta el modelo propuesto para el FJSSP-LS, mismo que se basa en los modelos de Manne, Özgüven y Novas (ver anexos A, B, C, presentados como referencia para el lector), con el objetivo de minimizar únicamente la amplitud de proceso. A diferencia del modelo de Novas, no se considera que los sublotes mantengan un orden de procesamiento y los sublotes son independientes entre sí, por lo que dos sublotes de un mismo trabajo y de una misma operación pueden ser procesados en máquinas diferentes. Para generar el modelo se requieren de las siguientes variables:

O_{ij}	Denota la operación i del trabajo j
Conjuntos/Indices	
J/j	Lotes o trabajos
I/i	Operaciones
K/k	Máquinas
I_j	Operaciones demandadas por el lote j
$K_{i,j}$	Máquinas con capacidad de procesar la operación i sobre el lote j
S_j	Conjunto de posibles sublotes para el trabajo j. Su cardinalidad, denotada como $MaxS_j$, representa el número máximo de sublotes en que puede dividirse j.
Parámetros	
$pt_{k,i,j}$	Tiempo de procesamiento en la máquina k de la operación i de una unidad del producto j
M	Número entero suficientemente grande
Variables	
$Cmax$	Amplitud de proceso
$W_{k,s,i,j}$	Asume el valor 1 si el sublote s en O_{ij} se realiza en la máquina k, 0 en otro caso.
$U_{s,i,j}$	Tamaño del sublote s sobre el que se ejecuta O_{ij}
C_j	Tiempo de finalización del trabajo j
$S_{s,i,j,k}$	Tiempo de inicio de O_{ij} del sublote s en la máquina k
$C_{s,i,j,k}$	Tiempo de finalización de O_{ij} del sublote s en la máquina k
$Z_{sijs'i'j'k}$	Asigna 1 si O_{ij} del sublote s precede a $O_{i'j'}$ del sublote s' en la máquina k

En el modelo la función objetivo a optimizar es la minimización de la

amplitud de proceso.

$$Min\ Cmax \tag{3.1}$$

Para garantizar que la suma total de todos los sublotes sea mayor o igual a la demanda total del trabajo se emplea la siguiente desigualdad:

$$\sum_{s \in S_j} U_{sij} \geq d_j; \qquad \forall i \in I_j, \forall j \in J \tag{3.2}$$

Como se mencionó anteriormente, el tamaño de cada sublote debe permanecer igual durante todas las operaciones, lo cual se logra con la siguiente ecuación.

$$U_{sij} = U_{si'j}; \forall s \in S_j, \forall j \in J, \forall i, i' \in I_j \tag{3.3}$$

En la restricción 3.4 se asigna una operación a una única máquina del conjunto de máquinas capaces.

$$\sum_{k \in K_{ij}} W_{ksij} = 1; \qquad \forall i \in I_j, \forall s \in S_j, \forall j \in J \tag{3.4}$$

Las restricciones 3.5 y 3.6 son complementarias y establecen el tiempo de finalización de una operación. Si $W_{ksij} = 0$, se obliga que los tiempos de inicio y terminación del sublote s en la máquina k sean igual a 0. Al contrario, si $W_{ksij} = 1$, la fecha de terminación C_{sijk} del procesamiento del sublote s es posterior a la fecha de inicio mas el tiempo de procesamiento.

$$S_{sijk} + Cs_{sijk} \leq W_{ksij} * M; \tag{3.5}$$

$$\forall s \in S_j, \forall i \in I_j, \forall j \in J, \forall k \in K_{i,j}$$

$$C_{sijk} \geq S_{sijk} + pt_{kij} * U_{sij} - (1 - W_{ksij}) * M; \tag{3.6}$$

$$\forall s \in S_j, \forall i \in I_j, \forall j \in J, \forall k \in K_{i,j}$$

La restricción 3.7 garantiza que la operación de un sublote no inicie hasta que la anterior haya terminado.

$$\sum_{k \in K_{ij}} S_{sijk} \geq \sum_{k \in K_{ij}} C_{si-1jk}; \forall i = 2, ..., I_j, \forall s \in S_j, \forall j \tag{3.7}$$

En 3.8 se establece el tiempo de terminación de cada trabajo.

$$C_j \geq \sum_{k \in K_{ij}} C_{sijk}; \forall j \in J, i = I_j, \forall s \in S_j \tag{3.8}$$

La restricción 3.9 establece que la amplitud de proceso representa el tiempo empleado para completar los trabajos se establece la restricción.

$$Cmax \geq C_j; \qquad \forall j \tag{3.9}$$

Las restricciones 3.10 y 3.11, denotan la secuencia de operaciones asignadas en la misma máquina. $Z_{sijs'i'j'k}$ es igual a 1 si el sublote s en O_{ij} precede a el sublote s' en $O_{i'j'}$ en la máquina k, 0 de lo contrario. O_{ij} no se coloca necesariamente inmediatamente antes de la operación $O_{i'j'}$ cuando $Z_{sijs'i'j'k} = 1$.

$$S_{sijk} \geq C_{s'i'j'k} - Z_{sijs'i'j'k} * M; \tag{3.10}$$

$$\forall s \in S_j, \forall j \leq j', \forall i, i' \in I_j, \forall k \in K_{ij} \cap K_{i'j'}$$

$$S_{s'i'j'k} \geq C_{sijk} - (1 - Z_{sijs'i'j'k}) * M; \tag{3.11}$$

$$\forall s \in S_j, \forall j \leq j', \forall i, i' \in I_j, \forall k \in K_{ij} \cap K_{i'j'}$$

Las restricciones de no negatividad para los tiempos de inicio y final de todas las operacione se establecen en la desigualdad 3.12.

$$S_{sijk} \geq 0, C_{sijk} \geq 0; \qquad \forall s, i, j, k \tag{3.12}$$

Para garantizar la no negatividad de los tiempos de finalización de los trabajos se establece la siguiente restricción 3.13.

$$C_j \geq 0; \qquad \forall i, j \tag{3.13}$$

En la restricción 3.14 se establece W_{sijk} como una variable binaria, que es igual a 1 cuando la operación i del trabajo j, del sublote s es procesada en la máquina k, 0 de lo contrario.

$$W_{sijk} \in \{0, 1\}; \qquad \forall s, i, j, k \tag{3.14}$$

La restricción 3.15 establece $Z_{sijs'i'j'k}$ como una variable binaria, que es igual a 1 si O_{ij} del sublote s precede a $O_{i'j'}$ del sublote s' en la máquina k, 0 de lo contrario.

$$Z_{sijs'i'j'k} \in \{0,1\}; \qquad \forall i \leq i', \forall j, j', \forall k \in K_{ij} \cap K_{i'j'} \tag{3.15}$$

La restricción 3.16 establece que el tamaño de cada sublote debe ser mayor o igual a cero.

$$U_{sij} \geq 0; \qquad \forall s \in S_j, \forall i \in I_j, \forall j \in J \tag{3.16}$$

Finalmente el modelo queda de la siguiente manera:

$$Min\ Cmax \tag{3.17}$$

$$\sum_{s \in S_j} U_{sij} \geq d_j; \qquad \forall i \in I_j, \forall j \in J \tag{3.18}$$

$$U_{sij} = U_{si'j}; \forall s \in S_j, \forall j \in J, \forall i, i' \in I_j \tag{3.19}$$

$$\sum_{k \in K_{ij}} W_{ksij} = 1; \qquad \forall i \in I_j, \forall s \in S_j, \forall j \in J \tag{3.20}$$

$$S_{sijk} + Cs_{sijk} \leq W_{ksij} * M; \tag{3.21}$$
$$\forall s \in S_j, \forall i \in I_j, \forall j \in J, \forall k \in K_{i,j}$$

$$C_{sijk} \geq S_{sijk} + pt_{kij} * U_{sij} - (1 - W_{ksij}) * M; \tag{3.22}$$
$$\forall s \in S_j, \forall i \in I_j, \forall j \in J, \forall k \in K_{i,j}$$

$$\sum_{k \in K_{ij}} S_{sijk} \geq \sum_{k \in K_{ij}} C_{si-1jk}; \forall i = 2, ..., I_j, \forall s \in S_j, \forall j \tag{3.23}$$

$$C_j \geq \sum_{k \in K_{ij}} C_{sijk}; \forall j \in J, i = I_j, \forall s \in S_j \tag{3.24}$$

$$Cmax \geq C_j; \qquad \forall j \tag{3.25}$$

$$S_{sijk} \geq C_{s'i'j'k} - Z_{sijs'i'j'k} * M; \tag{3.26}$$
$$\forall s \in S_j, \forall j \leq j', \forall i, i' \in I_j, \forall k \in K_{ij} \cap K_{i'j'}$$

$$S_{s'i'j'k} \geq C_{sijk} - (1 - Z_{sijs'i'j'k}) * M; \tag{3.27}$$
$$\forall s \in S_j, \forall j \leq j', \forall i, i' \in I_j, \forall k \in K_{ij} \cap K_{i'j'}$$

$$S_{sijk} \geq 0, C_{sijk} \geq 0; \qquad \forall s, i, j, k \tag{3.28}$$

$$C_j \geq 0; \qquad \forall i, j \tag{3.29}$$

$$W_{sijk} \in \{0, 1\}; \qquad \forall s, i, j, k \tag{3.30}$$

$$Z_{sijs'i'j'k} \in \{0, 1\}; \qquad \forall i \leq i', \forall j, j', \forall k \in K_{ij} \cap K_{i'j'} \tag{3.31}$$

$$U_{sij} \geq 0; \qquad \forall s \in S_j, \forall i \in I_j, \forall j \in J \tag{3.32}$$

Capítulo 4

Búsqueda Tabú

Como ya se mencionó anteriormente, TS fue introducida por Fred Glover, quien introdujo el término metaheurística en el mismo trabajo[80]. Esta técnica se encuentra dentro de la categoría de búsqueda local. Entre sus características principales se encuentra el uso de memoria que prohíbe temporalmente volver a soluciones ya visitadas, lo que permite la exploración de nuevas regiones en el espacio de búsqueda.

TS comienza generando una solución desde la cual inicia el proceso de búsqueda. La solución inicial puede ser generada de manera aleatoria o a través de alguna técnica heurística que ofrezca soluciones de calidad. Posteriormente, se realiza la exploración de la vecindad, lo cual implica modificar la solución actual para generar soluciones vecinas. La mejor solución vecina reemplaza a la solución actual aunque implique un deterioro en la función objetivo. Para evitar la formación de ciclos en la trayectoria de búsqueda, se prohíbe volver a la solución de la cual parte. Esta estrategia permite no quedar atrapado en óptimos locales y favorecer la exploración de nuevas zonas en el espacio de soluciones, las cuales, aunque sean peores, pueden guiar hacia el óptimo global del problema. El éxito de este proceso depende de la estructura de vecindad utilizada. Algunas de las vecindades que han dado mejores resultados se mencionaron en la sección 2.1.

La memoria es un aspecto clave en la TS. Participa en la generación de vecinos y ayuda a guiar la búsqueda a través del espacio de soluciones. Una estructura de memoria en TS es la lista tabú, ésta conserva los movimientos prohibidos durante un número limitado de iteraciones, con el fin de evitar permanecer en óptimos locales y así visitar nuevas zonas. Además, una longitud de la lista tabú igual a n garantiza evitar ciclos de longitud menor o igual a n en la trayectoria de búsqueda. Sin embargo, se puede permitir una solución generada con algún movimiento prohibido, si esa solución resulta tener un costo menor que el mejor valor encontrado hasta el momento, a ésto se le conoce como criterio de aspiración.

El tamaño de la lista tabú indica el tiempo que un movimiento permanece en la lista tabú. Se inicia con una lista vacía y se van guardando movimientos en cada iteración de la búsqueda, hasta que alcanza su capacidad máxima. Posteriormente, cuando se añade un nuevo elemento a la lista, el elemento más antiguo deja la lista tabú. Otro tipo de memoria diferente a la lista tabú se basa en una intensificación en regiones prometedoras (memoria de largo plazo), partiendo desde una solución ya visitada anteriormente y continuando la búsqueda en una dirección diferente a las que ya se hayan tomado.

Entre las aplicaciones más destacadas de TS hacia los problemas de programación, se encuentra el trabajo realizado por E. Nowicki y C. Smutnicki [18], en el cual hace uso de estos tipos de memoria.

4.1. Implementación de Búsqueda Tabú

El algoritmo basado en TS diseñado en este trabajo inicia con la creación aleatoria de una solución. Todas las soluciones están representadas por tres cadenas. La primera indica el tamaño de cada posible sublote, la segunda proporciona la secuencia en que se procesarán las operaciones de cada trabajo y la tercera señala la máquina en la cual debe ser procesada. Una vez que se tiene la primera solución se inicia el proceso de búsqueda y mejora, para lo cual se emplean tres tipos de vecindades y tres listas tabú, una por cada una de las cadenas que forman una solución. En cada iteración, se elige una vecindad de forma aleatoria y se generan y evalúan todos los vecinos de la solución actual. La mejor solución vecina, cuya aceptación no esté prohibida por la lista tabú correspondiente a la vecindad empleada, reemplaza a la solución actual. A su vez, por el criterio de aspiración, se puede seleccionar una solución tabú si esta tiene un costo menor que la mejor solución encontrada hasta el momento. Las principales etapas del algoritmo propuesto se resumen en el pseudocódigo 1.

Algoritmo 1: Búsqueda Tabú

```
1  Generar solución inicial
2  Mientras No se cumpla criterio de terminación
3  Hacer:
4  Seleccionar una vecindad
5  Generar los elementos de la vecindad seleccionada
6  Evaluar a todos los vecinos generados
7  si Se satisface criterio de aspiración entonces
8  |  Guardar el mejor vecino como mejor solución
9  fin
10 en otro caso
11 |  Seleccionar el mejor vecino no tabú como solución actual
12 fin
13 Actualizar lista tabú
14 Termina algoritmo
```

4.2. Codificación y Decodificación

Para tratar el problema, se utiliza una codificación basada en el método de Gao *et al.*[68]. La codificación contiene tres cadenas, una cadena adicional a las referidas por Gao. La primera cadena, PS, denota la cantidad de piezas atribuidas a cada sublote, la segunda cadena, SO, hace referencia a la secuencia de operaciones, y la tercera cadena, MS, a las máquinas seleccionadas de

la estación de trabajo de cada operación.

4.2.1. Cadena PS, Piezas atribuidas en cada sublote

La cadena PS, contiene n secciones, una por cada trabajo, la longitud de cada sección es igual al número de sublotes $MaxS_j$.

Para ejemplificar el uso de estas cadenas, se considera el ejemplo de la Figura 3.1 donde la demanda es 200, 450 y 400 unidades para los trabajos 1, 2 y 3 respectivamente. Si, por ejemplo, para cada trabajo se considerara que los lotes deben ser del mismo tamaño (condición que no se impondrá en el resto del documento), además se establece que los trabajos 1 y 3 sean divididos en a lo más 2 sublotes, mientras que al trabajo 2 se le permiten hasta 3 sublotes, se obtiene una cadena PS la cual estará dividida en tres secciones, siendo el tamaño de cada sección 2, 3 y 2 respectivamente. La codificación de esta cadena es mostrada en la tabla 4.1.

100	100	150	150	150	200	200

Tabla 4.1: Cadena PS

Los sublotes serán identificados de acuerdo a la posición que ocupan dentro de la cadena PS, por ejemplo, el tercer sublote del trabajo 2, será identificado como el sublote 5.

4.2.2. Cadena SO, Secuencia de operaciones

La cadena SO está formada por los números de identificación de los sublotes. Por lo tanto, el número de identificación s de cada sublote del trabajo j aparecerá tantas veces como número de operaciones, $|I_j|$, requiera el trabajo j. La cadena será leída de izquierda a derecha, la $i-ésima$ aparición del número del sublote s hace referencia a la $i-ésima$ operación de ese sublote. Siguiendo con el ejemplo de la Figura 3.1, el trabajo 1 tiene 3 operaciones, por lo que los sublotes 1 y 2 que componen este trabajo tienen 3 operaciones. De la misma manera los sublotes 3, 4 y 5 tendrán el mismo número de operaciones que el trabajo 2 y los sublotes 6 y 7 que el trabajo 3. En la tabla 4.2 se presenta una posible cadena SO para este ejemplo

7	3	3	2	5	6	1	2	1	6	5	7	4	4	1	7	2	5	3	4	6

Tabla 4.2: Cadena SO

Como se mencionó anteriormente, las cadenas SO se deben leer de izquierda a derecha. y donde cada aparición de un sublote hace referencia a una operación de dicho sublote, por ejemplo, de acuerdo a la cadena mostrada en la Tabla 4.2, el sublote **7** es el primer sublote que será programado. Como ésta es la primera vez que aparece el número **7**, se entiende que se trata de la primera operación de dicho sublote. Posteriormente se observa será programado el sublote **3**, en su primera operación. Después se observa que aparece nuevamente el sublote **3**, por lo que será programada su segunda operación. Su tercera y última operación será programada cuando vuelva a aparecer, en este caso hasta la posición 19.

4.2.3. Cadena MS Máquinas asignadas

La cadena MS, denota la máquina asignada para procesar las operaciones de cada sublote. La cadena MS está dividida en varias secciones, una para cada sublote. Además, la longitud de cada sección está dada por el número total de máquinas requeridas para completar las operaciones del sublote correspondiente. MS coincide con la longitud de la cadena SO, $\sum_{j=1}^{N} MaxS_j \times |I_j|$, pero en este caso la cadena se usa para indicar la máquina asignada para cada operación.

Continuando con el ejemplo mostrado en la Figura 3.1, se tienen 7 secciones, una sección por cada sublote. La longitud de cada sección es igual al número de operaciones que necesitan procesarse, por lo que para este ejemplo, como todos los sublotes requieren 3 operaciones, la longitud de cada sección es de tamaño 3. La primera sección corresponde a las operaciones del sublote 1, la segunda a las operaciones del trabajo 2 y así sucesivamente.

En la Tabla 4.3, se presenta una solución generada de forma aleatoria. Como el primer y segundo sublote pertenecen al trabajo 1, ambos tienen la misma secuencia de operaciones y las mismas estaciones de trabajo $(K_{1,1}, K_{2,1}, K_{3,1})$. El sublote 1, en su primera operación, puede ser realizado en el conjunto de máquinas $K_{1,1} = \{1, 2, 3\}$, por lo que es seleccionada una de ellas de manera aleatoria, en este caso la máquina **1**. La segunda operación del sublote 1 puede ser procesada en $K_{2,1} = \{1, 2\}$ y la última operación en el conjunto de máquinas $K_{3,1} = \{2, 3\}$, se seleccionan **1** y **3** respectivamente. Para el sublote 2, las máquinas seleccionadas para su procesamiento son **2,1,3**. De la misma manera, los sublostes 3, 4 y 5 que pertenecen al trabajo 2, comparten el mismo conjunto de máquinas $(K_{1,2}, K_{2,2}, K_{3,2})$.

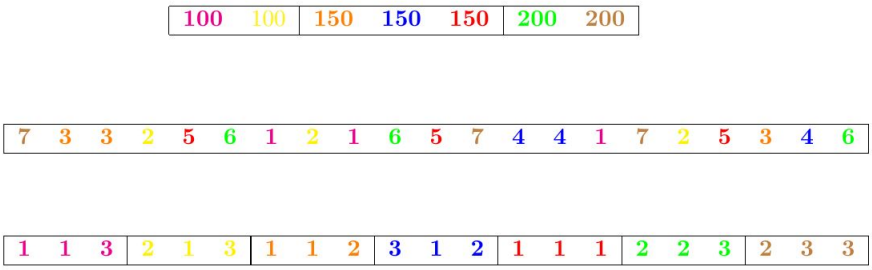
1	1	3	2	1	3	1	1	2	3	1	2	1	1	1	2	2	3	2	3	3

Tabla 4.3: Cadena MS

4.2.4. Decodificación

Dada la solución generada anteriormente, se obtuvieron las cadenas PS, SO y MS, resumidas en la Figura 4.1. Esta representación ahora debe ser decodificada para poder ser interpretada como un programa, es decir una determinación de las fechas de inicio y terminación de cada operación en cada máquina. El procedimiento utilizado se ilustra a continuación para la misma solución.

Figura 4.1: Cadenas utilizadas



La lectura de las cadenas es de izquierda a derecha. Se comienza con el primer elemento de la cadena SO, el cual hace referencia al sublote **7** en su primera operación. La asignación de máquinas para este sublote se encuentra en la séptima sección de la cadena MS. El primer elemento corresponde a la máquina asignada para la primera operación, por lo que se procesará en la máquina **2**. El tiempo de procesamiento se toma de manera proporcional al lote original, en este caso el tiempo de procesamiento será de **20** unidades, como la máquina se encuentra desocupada el tiempo de inicio es 0 y el tiempo de terminación 20.

La siguiente operación leída de SO corresponde al sublote **3**, la asignación de máquinas de este sublote se encuentra en la tercera sección de la cadena MS y la máquina asignada para la primera operación es la máquina **1**. El tiempo de procesamiento para esta operación es de **10** unidades. Siendo ésta la primera operación a procesar de la máquina 1 y al ser la primera operación de un sublote, puede empezar en el tiempo 0 y el tiempo de terminación para la operación queda en 10. La tercera operación también corresponde al sublote **3** e igualmente será procesada por la máquina **1**. Tiene un tiempo de procesamiento de **14** unidades, por lo que el tiempo de inicio es 10 y el tiempo de finalización será 24. Continuando de esta manera para las siguientes 9 operaciones se obtiene el diagrama de Gantt mostrado en la Figura 4.2.

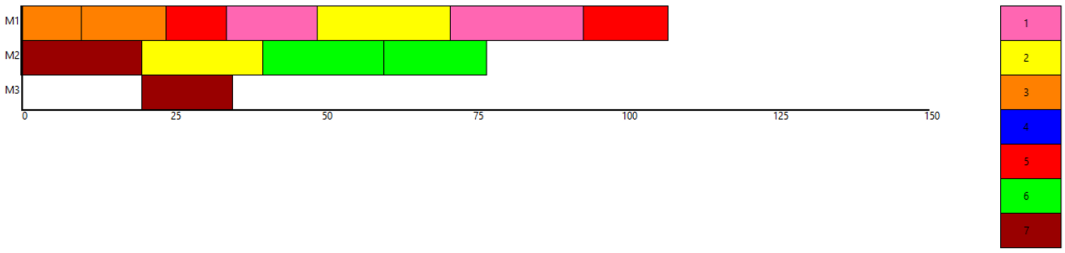


Figura 4.2: Construcción de una solución parcial

La operación 13 corresponde al sublote **4** y será procesada en la máquina **3**, con un tiempo de procesamiento de **12** unidades. Se observa que existe un tiempo muerto de 20 unidades en la máquina 3. Como el tiempo muerto es mayor que el tiempo de procesamiento, se puede aprovechar y programar el sublote 4 antes del sublote 7 como se muestra en la Figura 4.3. Este procedimiento garantiza la generación de programas activos, i.e., en los cuales no se puede adelantar el procesamiento de alguna tarea sin retrasar el inicio del procesamiento de al menos otra. Está demostrado que los programas óptimos pertenecen a la clase de los programas activos [2] y la estrategia de decodificación propuesta en el presente trabajo justamente permite garantizar que sólo se produzcan programas activos.

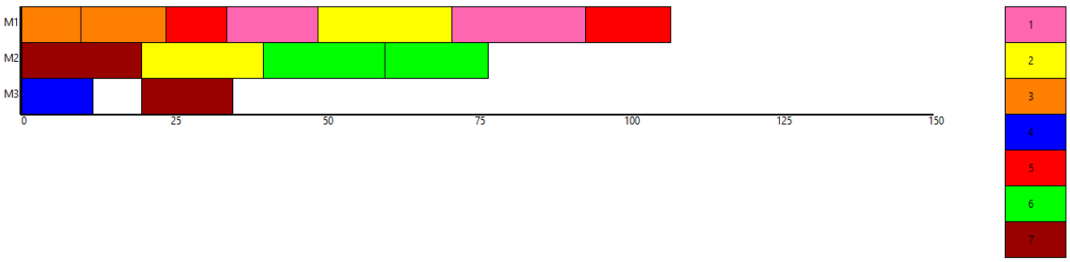


Figura 4.3: Aprovechamiento de tiempos muertos

Finalmente, al terminar de procesar todas las operaciones de esta solución se obtiene un plan de producción con una amplitud de proceso de 138 unidades mostrado en la Figura 4.4.

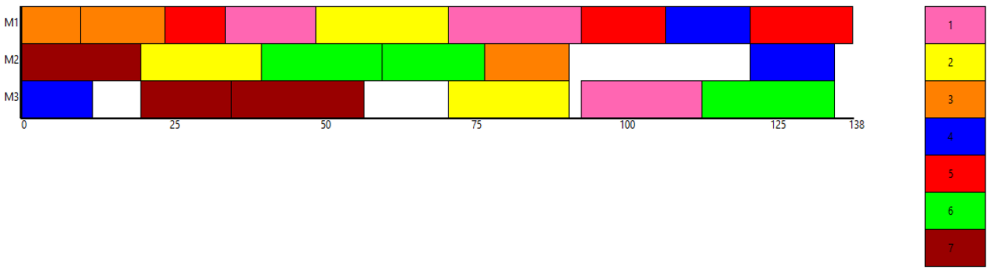


Figura 4.4: Solución inicial

El proceso de decodificación descrito para la solución anterior se refleja en los Algoritmos 2 y 3, integrados en TS. En el primero, se realiza sencillamente la lectura de las tres cadenas de datos para determinar cuál es la siguiente operación a programar O_s, así como el sublote asociado $s \in S$, el trabajo $j \in J$ del que deriva, y la máquina $k \in K$ sobre la cual será programada. Después se llama al Algoritmo 3 para determinar las fechas de inicio y terminación de la operación O_s, a través del conocimiento de *(i)* la fecha de terminación de la operación anterior del mismo sublote s, tf_{O_s-1s}, y de *(ii)* la fecha de liberación de la máquina k, td_k. Además, el algoritmo detecta posibles tiempos muertos en los que es posible colocar la operación O_s, garantizando así la generación de un programa activo.

Algoritmo 2: Decodificación

1 J:= Conjunto de trabajos.
2 S:= Conjunto de sublotes.
3 K:= Conjunto de máquinas.
4 O_s:= Siguiente operación a programar para el sublote s.
5 k:= Máquina seleccionada para procesar O_s.
6 td_k:= Fecha de disponibilidad de la máquina k, *i.e.* fecha de terminación de la operación programada en última posición en la máquina k.
7 $tp_{O_s,s,k}$:= tiempo de procesamiento de la operación O_s del sublote s en la máquina k.
8 **Para** $s := 1$ *hasta* $|S|$ **hacer**
9 | $O_s = 1$
10 **fin**
11 **Para** $k := 1$ *hasta* $|K|$ **hacer**
12 | $td_k = 0$
13 **fin**
14 **Para** $l := 1$ *hasta* $|SO|$ **hacer**
15 | Leer en la posición l de SO el índice s del siguiente sublote a programar, identificar el trabajo $j \in J$ asociado.
16 | Asignar el sublote s a la máquina k leída en la posición O_l de la sección s de la cadena MS.
17 | Leer en la posición s de la cadena PS el tamaño del sublote s, para determinar el tiempo de procesamiento de la operación O_s en la máquina k, $tp_{O_s,s,k}$.
18 | Establecer tiempos de inicio y fin del sublote s (Ver algoritmo 3).
19 | $O_s = O_s + 1$
20 **fin**
21 **Termina algoritmo**

Algoritmo 3: Tiempos de inicio y fin

```
1  O_s:= operación a programar del sublote s.
2  k:= Máquina seleccionada para procesar O_s.
3  td_k:= Fecha de disponibilidad de la máquina k.
4  ti_{O_s,s}:= tiempo de inicio de la O_s-ésima operación del sublote s.
5  tf_{O_s,s}:= tiempo de terminación de la O_s-ésima operación del sublote s.
6  tp_{O_s,s,k}:= tiempo de procesamiento de la operación O_s del sublote s en
   la máquina k.
7  si td_k = 0 & O_s = 1 \\ la máquina k aún no tiene ninguna operación
8                        \\ programada y la operación O_s es la primera
9                        \\ a programar del sublote s
10 entonces
11 |  ti_{O_s,s} = 0
12 fin
13 en otro caso
14 |  si td_k = 0 entonces
15 |  |  ti_{O_s,s} = tf_{O_s-1,s}
16 |  fin
17 |  en otro caso
18 |  |  si td_k > tf_{O_s-1,s} entonces
19 |  |  |  si existen tiempos muertos después de tf_{O_s-1,s} en la
   |  |  |  máquina s entonces
20 |  |  |  |  si O_s puede ser procesada en alguno de los tiempos
   |  |  |  |  muertos entonces
21 |  |  |  |  |  Programar O_s en la fecha más temprana posible en
   |  |  |  |  |  los tiempos muertos que tiene la máquina k
22 |  |  |  |  fin
23 |  |  |  |  en otro caso
24 |  |  |  |  |  ti_{O_s,s} = td_k
25 |  |  |  |  fin
26 |  |  |  fin
27 |  |  |  en otro caso
28 |  |  |  |  ti_{O_s,s} = td_k
29 |  |  |  fin
30 |  |  fin
31 |  |  en otro caso
32 |  |  |  ti_{O_s,s} = tf_{O_s-1,s}
33 |  |  fin
34 |  fin
35 fin
36 tf_{O_s,s} = ti_{O_s,s} + tp_{O_s,s,k}
37 td_k = tf_{O_s,s}
38 Termina algoritmo
```

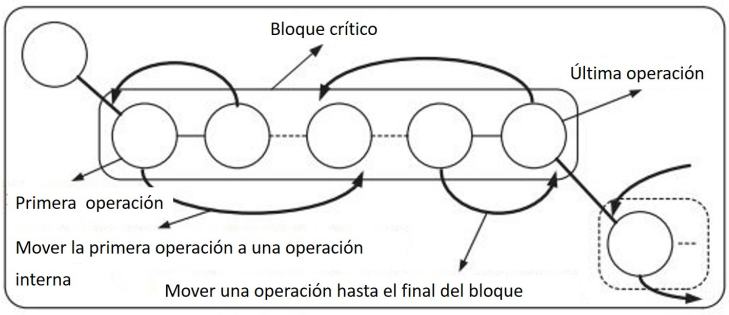
4.3. Vecindades

Para promover una adecuada exploración del espacio de soluciones, se propone el uso de tres vecindades. La primera es semejante a la propuesto por Zhang *et al.* [22], para mover la secuencia de operaciones. La segunda vecindad servirá para cambiar la asignación de las máquinas. Finalmente, la última vecindad permitirá modificar el tamaño de los sublotes.

4.3.1. Vecindad SO, secuencia de operaciones

La vecindad propuesta en [22] se muestra en la Figura 4.5. Para la aplicación de esta vecindad es necesario determinar primero la ruta critica y los bloques críticos que la componen. La vecindad está formado por todas las soluciones que se pueden generar mediante el cambio de las operaciones extremas dentro de un bloque crítico. Por lo tanto, se toma la operación inicial (final) y se coloca dentro del bloque crítico, recorriendo las demás operaciones. Este procedimiento está plasmado en Algoritmo 4.

Figura 4.5: Vecindad de secuencias



Algoritmo 4: Vecindad SO

```
1  Sa := Solución actual.
2  R := Ruta crítica de Sa.
3  b := Número de bloques críticos en R.
4  l_i := Número de operaciones en el i -ésimo bloque.
5  Para i := 1 a b hacer
6      si l_i > 1 entonces
7          si l_i = 2 entonces
8              Hacer una copia de Sa.
9              Intercambiar las operaciones dentro del bloque i.
10             Guardar la solución candidata como vecino.
11         fin
12         en otro caso
13             Para j := 1 hasta l_i − 2 hacer
14                 Copia de Sa.
15                 Insertar la operación inicial entre las operaciones j + 1 y
                   j + 2 del bloque i.
16                 Guardar la solución candidata como vecino.
17             fin
18             Para J :=l_i − 2 hasta 1 hacer
19                 Copia de Sa.
20                 Insertar la operación final entre las operaciones j y j + 1
                   del bloque i.
21                 Guardar la solución candidata como vecino.
22             fin
23         fin
24     fin
25 fin
26 Termina algoritmo
```

4.3.2. Vecindad AM, asignación de máquinas

Esta vecindad consiste en seleccionar las operaciones críticas cuya estación de trabajo tiene más de una máquina. Los vecinos serán generados probando cada una de estas operaciones en cada posible máquina de sus estaciones de trabajo. Este procedimiento está presentado en el Algoritmo 5.

Algoritmo 5: Vecindad AM

```
1  Sa := Solución actual.
2  R := Ruta crítica de Sa.
3  b := Número de bloques críticos en R.
4  l_p := Número de operaciones en el p-ésimo bloque.
5  K_ij:= Conjunto de máquinas capaces de procesar la operación O_ij.
6  k := Máquina de procesamiento actual.
7  Para p := 1 hasta b hacer
8      Para q :=1 hasta l_p hacer
9          si |K_ij| > 1 entonces
10             Seleccionar la operación siguiente q del bloque p, llamada
               O_ij.
11             Identificar K_ij.
12             Eliminar k de K_ij.
13             Para l:=1 hasta |K_ij| − l hacer
14                 Copia de Sa.
15                 Seleccionar una máquina k' de K_ij y asignarla al
                   procesamiento de O_ij.
16                 Eliminar k' de K_ij.
17                 Guardar la solución candidata como vecino.
18             fin
19         fin
20     fin
21 fin
22 Termina algoritmo
```

4.3.3. Vecindad DL, división de lotes

El cambio de tamaño se aplica a cada sublote $s \in S_j$ de cada trabajo $j \in J$. Consiste en reducir el tamaño del sublote considerado en cierta cantidad E_j y transferirla a otro sublote del mismo trabajo j. Esta cantidad, estimada en número de piezas o partes, se calcula como un porcentaje α del tamaño total de piezas dentro de la demanda del trabajo correspondiente. Cabe mencionar que se probó con valores de α variando entre 1 % y 20 %, incluso mediante estrategias dinámicas (α disminuyendo linealmente en el transcurso de la búsqueda), sin notar mucha sensibilidad en el desempeño del algoritmo. Por lo tanto, se utilizó arbitrariamente un valor de 5 %.

Así, cada vecino es generado sustrayendo la cantidad α de piezas deducidas al sublote considerado y sumándola a otro sublote del mismo trabajo, generando todas las combinaciones posibles. Nótese que existe el caso particular cuando el tamaño de algún sublote es menor que algún límite inferior, entonces se fija igual a 0 y el sublote "receptor" absorbe el excedente.

Algoritmo 6: Vecindad DL

```
1  Sa := Solución actual.
2  J:= Conjunto de trabajos.
3  MaxS_j: Máximo de sublotes del j−ésimo trabajo.
4  E_j := Cantidad de piezas a mover del trabajo j.
5  U_{s,j} := tamaño a procesar del sublote s derivado del trabajo j.
6  Para j := 1 hasta |J| hacer
7      si MaxS_j > 1 entonces
8          Para i :=1 hasta MaxS_j − 1 hacer
9              Para k := i+1 a MaxS_j hacer
10                 Copia de Sa.
11                 U_{i,j} = U_{i,j} − E_j
12                 U_{k,j} = U_{k,j} + E_j
13                 Guardar la solución candidata como vecino.
14             fin
15         fin
16         Para i := MaxS_j hasta 2 hacer
17             Para k := MaxS_j − 1 hasta 1 hacer
18                 Copia de Sa.
19                 U_{i,j} = U_{i,j} − E_j
20                 U_{k,j} = U_{k,j} + E_j
21                 Guardar la solución candidata como vecino.
22             fin
23         fin
24     fin
25 fin
26 Termina algoritmo
```

Capítulo 5

Resultados

Se muestran los experimentos computacionales realizados sobre el modelo matemático para el FJSSP-LS generado en el capítulo 3. Estos experimentos tienen dos objetivos, el primero consiste en determinar la relevancia del modelo propuesto, evaluada en términos de los beneficios que permite alcanzar con respecto de otras configuraciones ya reportadas en la literatura. Particularmente, se compara la amplitud del proceso obtenida para el FJSSP-LS con la del FJSSP sin división de lotes.

Por otro lado, la evaluación de los resultados obtenidos con el modelo propuesto deben ser una garantía de la eficacia y eficiencia. En otras palabras, es preciso evaluar el desempeño de la técnica de optimización desarrollada en el marco de este proyecto, i.e. el algoritmo basado en TS descrito en el capítulo 4. Para ello, todas las instancias creadas en este trabajo se resolvieron con un solver comercial de Programación Matemática, Gurobi [88], que demostró un excelente desempeño para resolver una amplia gama de problemas de programación entera mixta. La meta de esta estrategia es permitir una comparación de las soluciones obtenidas con TS con los óptimos (o las mejores soluciones factibles) identificados por Gurobi, aunque, como se verá a continuación, esto sólo será posible para las instancias más sencillas.

Así pues, este capítulo presenta una serie de experimentos que, en un primer tiempo, buscan validar el comportamiento de Gurobi sobre modelos más sencillos (JSSP y FJSSP, experimentos 1 y 2). En el experimento 2, también se aplica la heurística de TS, para evaluar su desempeño y compararlo con soluciones de Gurobi y del estado del arte. El experimento 3 consiste en aplicar Gurobi sobre una versión simplificada del FJSSP-LS, con la meta de evidenciar tanto los beneficios de la política de división de lotes como la eficacia comparada de la técnica de optimización. Finalmente, los experimentos computacionales definitivos, realizados con Gurobi y TS sobre las instancias de FJSSP-LS definidas en la sección 3.1, se describen en los experimentos

4, 5 y 6. Antes que nada, sin embargo, se empieza definiendo los aspectos prácticos de la implementación de los dos métodos de optimización utilizados en este trabajo.

5.1. Metodología experimental

Gurobi es un motor de optimización computacional, el cual es capaz de considerar automáticamente millones de posibles soluciones y resolver modelos matemáticos que cuentan con un gran número de variables y restricciones. Es un software comercial que fue diseñado para aprovechar las arquitecturas modernas y procesadores multi-núcleo. La implementación de un modelo en Gurobi se realiza de manera sencilla gracias a un lenguaje de representación amigable. Los modelos generados en el marco de este proyecto fueron creados a través de python en un archivo `.lp` y posteriormente fueron llamados de manera individual desde Gurobi a través de la ventana de comandos de Windows. Para cada instancia, se limitó la duración de las ejecuciones a un valor máximo de una hora.

En cuanto a TS, dado que esta técnica de optimización es estocástica, es necesario realizar varias ejecuciones por instancia para poder estudiar la robustez del algoritmo propuesto. En este estudio, se efectuaron 10 corridas por instancia. Además, la longitud de cada lista tabú (una por vecindad) se fijó de forma separada de las otras. Usando una estrategia de fuerza bruta, se determinó que los valores más adecuados eran de 13 para el tamaño de lista tabú asociada a la vecindad SO, 17 para el tamaño de lista tabú asociada a la vecindad AM y 17 para el tamaño de lista tabú asociada a la vecindad DL. Finalmente, TS utiliza dos criterios de paro. En primer lugar, se limita el tiempo de ejecución a una hora. Además, se estima que se obtiene convergencia cuando se realizan 10,000 (resp. 30,000) iteraciones consecutivas sin mejora para las instancias LA01-20 (resp. LA21-40).

5.2. Experimento 1: JSSP

Las primeras pruebas se realizaron con el banco de 40 instancias de Lawrence (LA), estas instancias son del tipo JSSP. El modelo utilizado es el mismo que el presentado para el FJSSP-LS, al cual se le quitaron los aspectos relacionados con flexibilidad y división de lotes, o sea que se reduce al modelo presentado en el Apéndice A de Manne [4]. Los resultados obtenidos por Gurobi se muestran en la Tabla 5.1. En la primera columna, se indica el nombre de la instancia, la segunda columna contiene el valor de la amplitud de proceso óptima encontrada por Gurobi, o bien la mejor solución factible (cota superior sobre el árbol de búsqueda) en caso de que se haya obtenido convergencia en una corrida de una hora. La cota inferior correspondiente se reporta en la tercera columna (si es igual a la cota superior, entonces la solución encontrada por Gurobi es óptima). Finalmente, en la cuarta columna, se incluye el tiempo de ejecución empleado por Gurobi.

Una primera observación es que Gurobi logra determinar la solución óptima para las instancias LA01-05, LA16-20 y LA24-25 (12 instancias de las 40), en un tiempo menor a una hora. Se puede notar que, para las demás instancias, el tiempo de cómputo alcanza los 3,600 segundos y sólo se identifica una cota superior. Se puede notar que, en la mayoría de estos casos en los que no se alcanzó la garantía de optimalidad, la cota inferior queda muy lejos de la mejor solución factible encontrada, lo que no da una información muy relevante sobre la distancia al óptimo. Por ello, se comparan las soluciones de Gurobi con los resultados reportados en otras fuentes.

Este análisis se presenta en la tabla 5.2 donde los resultados de TS se comparan con los obtenidos con TS/PR [27], técnicas de búsqueda de equilibrio global (GES) en [89], TS en [22], TS/SA en [82], HGA en [25], BRKGA en [26]. La primera columna indica el nombre de la instancia, la segunda contiene el mejor valor reportado en la literatura, de la columna 3 a la 6 incluye el costo reportado por los métodos antes citados. Finalmente, en la última columna, se incluye el valor encontrado por Gurobi. Excluyendo las 12 instancias (LA01-05, LA16-20 y LA24-25) en las que Gurobi encuentra el óptimo, en otras 17 instancias (LA06-15, LA22-23, LA30-LA33 y LA36) es capaz de encontrar la mejor solución reportada en la literatura, aunque no da una garantía de optimalidad. En cuanto a las 11 últimas instancias (en las que Gurobi no encuentra la mejor solución conocida), Gurobi queda en promedio a 2.20 % del mejor resultado reportado (la distancia relativa varía entre 0.28 % y 8.16 %), lo que parece muy razonable, aunque demuestra que un método exacto, para varias instancias, se ve superado por la complejidad del problema JSSP estándar.

Instancia	Mejor valor de Amplitud de proceso encontrado (*Indica óptimo)	Cota inferior	Tiempo (s)
LA01	666*	666	2.14
LA02	655*	655	5.13
LA03	597*	597	3.64
LA04	590*	590	0.83
LA05	593*	593	29.48
LA06	926	613	3600.05
LA07	890	627	3600.05
LA08	863	629	3600.03
LA09	951	696	3600.05
LA10	958	690	3600.03
LA11	1222	620	3600.12
LA12	1,039	559	3600.03
LA13	1,150	571	3600.12
LA14	1,292	575	3600.11
LA15	1207	568	3600.03
LA16	945*	945	8.42
LA17	784*	784	2.33
LA18	848*	848	0.91
LA19	842*	842	2.81
LA20	902*	902	2.13
LA21	1071	886	3600.06
LA22	927	852	3600.03
LA23	1032	775	3600.05
LA24	935*	935	1647.57
LA25	977*	977	1246.76
LA26	1,242	870	3600.08
LA27	1,288	819	3600.11
LA28	1,244	848	3600.09
LA29	1,246	855	3600.06
LA30	1,355	864	3600.16
LA31	1,784	825	3600.06
LA32	1,850	870	3600.05
LA33	1,719	791	3600.08
LA34	1,735	751	3600.12
LA35	1,900	795	3600.05
LA36	1,268	1,233	3600.02
LA37	1,401	1,215	3600.03
LA38	1,207	1,085	3600.03
LA39	1,251	1,139	3600.03
LA40	1,234	1,091	3600.08

Tabla 5.1: Resultados obtenidos por Gurobi sobre las instancias LA

Instancia	Best	TS/PR	BRKGA	GES	TS/SA	TS	HGA	Gurobi
LA01	**666**	**666**	**666**	**666**	–	–	**666**	**666**
LA02	**655**	**655**	**655**	**655**	–	–	**655**	**655**
LA03	**597**	**597**	**597**	**597**	–	–	**597**	**597**
LA04	**590**	**590**	**590**	**590**	–	–	**590**	**590**
LA05	**593**	**593**	**593**	**593**	–	–	**593**	**593**
LA06	**926**	**926**	**926**	**926**	–	–	**926**	**926**
LA07	**890**	**890**	**890**	**890**	–	–	**890**	**890**
LA08	**863**	**863**	**863**	**863**	–	–	**863**	**863**
LA09	**951**	**951**	**951**	**951**	–	–	**951**	**951**
LA10	**958**	**958**	**958**	**958**	–	–	**958**	**958**
LA11	**1222**	**1222**	**1222**	**1222**	–	–	**1222**	**1222**
LA12	**1039**	**1039**	**1039**	**1039**	–	–	**1039**	**1039**
LA13	**1150**	**1150**	**1150**	**1150**	–	–	**1150**	**1150**
LA14	**1292**	**1292**	**1292**	**1292**	–	–	**1292**	**1292**
LA15	**1207**	**1207**	**1207**	**1207**	–	–	**1207**	**1207**
LA16	**945**	**945**	**945**	**945**	–	–	**945**	**945**
LA17	**784**	**784**	**784**	**784**	–	–	**784**	**784**
LA18	**848**	**848**	**848**	**848**	–	–	**848**	**848**
LA19	**842**	**842**	**842**	**842**	**842**	**842**	844	**842**
LA20	**902**	**902**	**902**	**902**	–	–	907	**902**
LA21	**1046**	**1046**	**1046**	**1046**	**1046**	**1046**	**1046**	1071
LA22	**927**	**927**	**927**	**927**	–	–	935	**927**
LA23	**1032**	**1032**	**1032**	**1032**	–	–	**1032**	**1032**
LA24	**935**	**935**	**935**	**935**	**935**	**935**	953	**935**
LA25	**977**	**977**	**977**	**977**	**977**	**977**	981	**977**
LA26	**1218**	**1218**	**1218**	**1218**	–	–	**1218**	1242
LA27	**1235**	**1235**	**1235**	**1235**	**1235**	**1235**	1236	1288
LA28	**1216**	**1216**	**1216**	**1216**	–	–	**1216**	1244
LA29	**1152**	1153	1153	**1152**	1153	1156	1160	1246
LA30	**1355**	**1355**	**1355**	**1355**	–	–	**1355**	**1355**
LA31	**1784**	**1784**	**1784**	**1784**	–	–	**1784**	**1784**
LA32	**1850**	**1850**	**1850**	**1850**	–	–	**1850**	**1850**
LA33	**1719**	**1719**	**1719**	**1719**	–	–	**1719**	**1719**
LA34	**1721**	**1721**	**1721**	**1721**	–	–	**1721**	1735
LA35	**1888**	**1888**	**1888**	**1888**	–	–	**1888**	1900
LA36	**1268**	**1268**	**1268**	**1268**	**1268**	**1268**	1287	**1268**
LA37	**1397**	**1397**	**1397**	**1397**	**1397**	**1397**	1407	1401
LA38	**1196**	**1196**	**1196**	**1196**	**1196**	**1196**	**1196**	1207
LA39	**1233**	**1233**	**1233**	**1233**	**1233**	**1233**	**1233**	1251
LA40	**1222**	**1222**	**1222**	**1222**	1224	1224	1229	1234

Tabla 5.2: Comparación de resultados de instancias LA en JSSP

5.3. Experimento 2: FJSSP

Aunque tanto el algoritmo TS desarrollado en este trabajo como la implementación del modelo en Gurobi representan propuestas para resolver el FJSSP-LS, se consideró que se podía evaluar su desempeño al emplearlos para generar soluciones en las instancias de Hurink [1]. Se debe destacar que estas instancias fueron adaptadas del JSSP para el FJSSP, empleando tres estrategias, que consisten en aumentar cada vez más el número de máquinas que pueden procesar una operación. De esta forma, se construyen los conjuntos de instancias edata (en promedio 1.2 máquinas por operación, baja flexibilidad), rdata (en promedio 2 máquinas por operación, flexibilidad mediana) y vdata (en promedio 4 máquinas por operación, alta flexibilidad). Los resultados obtenidos son comparados contra los de TSN1 y TSN2 de Hurink *et al.* [1], IATS de Dauzère-Pérès y Lasserre [76], TS de Mastrolilli y Gambardella [71], HA de Li y Gao [90]. Por último, es importante mencionar que para estas instancias, el algoritmo fue modificado para que no aplicara división de lotes.

Los resultados obtenidos por Gurobi en las instancias del banco edata son mostrados en la Tabla 5.3, que incluye el nombre de la instancia, la mejor solución factible (cota superior) encontrada, la cota inferior producida por Gurobi y el tiempo de ejecución. En 15 de las 40 instancias, Gurobi garantiza la optimalidad de la solución reportada en el tiempo establecido. Para las demás instancias, la diferencia con la cota inferior es muy variable, dando pocas indicaciones sobre la calidad de la cota superior identificada.

En la Tabla 5.4, se muestran los resultados reportados para distintas técnicas heurísticas sobre estas instancias. Las últimas dos columnas incluyen los resultados obtenidos por Gurobi y la implementación de TS propuesta para esta tesis. En el caso de TS, el valor indicado es el de la mejor solución obtenida de las 10 ejecuciones realizadas. Se observa que Gurobi fue capaz de encontrar el óptimo en 24 de las 40 instancias probadas. Por otro lado, TS encuentra el óptimo, o la mejor solución reportada, en 19 instancias y en el peor de los casos se queda a 2.36 % de la mejor solución conocida, como ocurre en LA29. Estos resultados muestran la eficiencia de TS para alcanzar soluciones de buena calidad en este banco de instancias.

Instancia	edata	cota	tiempo(s)
LA01	609*	609	7.78
LA02	655*	655	8.53
LA03	550*	550	6.5
LA04	568*	568	3.28
LA05	503*	503	4.89
LA06	833	600	3600.06
LA07	762	558	3600.08
LA08	845	614	3600.06
LA09	878	695	3600.03
LA10	866	866	3012.95
LA11	1,107	589	3600.1
LA12	960	473	3600.06
LA13	1,053	577	3600.06
LA14	1,123	571	3600.11
LA15	1,111	571	3600.03
LA16	892*	892	1.67
LA17	707*	707	1.09
LA18	842*	842	5.34
LA19	796*	796	4.55
LA20	857*	857	1.75
LA21	1027	875	3600.02
LA22	880*	880	940.49
LA23	957	808	3600.06
LA24	909	862	3600.05
LA25	936*	936	883.34
LA26	1,172	821	3600.14
LA27	1,253	828	3600.08
LA28	1,206	816	3600.09
LA29	1,196	804	3600.11
LA30	1,262	879	3600.12
LA31	1,674	784	3600.03
LA32	1,818	889	3600.09
LA33	1,701	762	3600.11
LA34	1,694	735	3600.12
LA35	1,831	793	3600.11
LA36	1,160*	1,160	793.9
LA37	1,397	1,182	3600.05
LA38	1,149	1,050	3600.06
LA39	1,184*	1,184	655.97
LA40	1,173	1,042	3600.19

Tabla 5.3: Resultados presentados por Gurobi sobre las instancias edata en el FJSSP

Instancia	TSN1	TSN2	IATS	TS	HA	Gurobi	TS
LA01	611	618	**609**	**609**	**609**	**609**	**609**
LA02	**655**	656	**655**	**655**	**655**	**655**	**655**
LA03	573	566	554	**550**	**550**	**550**	563
LA04	578	578	**568**	**568**	**568**	**568**	**568**
LA05	**503**	**503**	**503**	**503**	**503**	**503**	**503**
LA06	**833**	**833**	**833**	**833**	**833**	**833**	**833**
LA07	765	778	765	**762**	**762**	**762**	768
LA08	**845**	**845**	**845**	**845**	**845**	**845**	**845**
LA09	**878**	**878**	**878**	**878**	**878**	**878**	**878**
LA10	**866**	**866**	**866**	**866**	**866**	**866**	**866**
LA11	1106	1106	**1103**	**1103**	**1103**	1107	1106
LA12	**960**	**960**	**960**	**960**	**960**	**960**	**960**
LA13	**1053**	**1053**	**1053**	**1053**	**1053**	**1053**	**1053**
LA14	1151	**1123**	**1123**	**1123**	**1123**	**1123**	**1123**
LA15	**1111**	1121	**1111**	**1111**	**1111**	**1111**	**1111**
LA16	924	961	915	**892**	**892**	**892**	**892**
LA17	757	757	**707**	**707**	**707**	**707**	708
LA18	864	864	843	**842**	**842**	**842**	843
LA19	850	813	**796**	**796**	**796**	**796**	**796**
LA20	919	919	864	**857**	**857**	**857**	**857**
LA21	1066	1085	1046	1017	**1014**	1027	1032
LA22	919	905	890	882	**880**	**880**	885
LA23	980	980	953	**950**	**950**	957	961
LA24	952	952	918	**909**	**909**	**909**	918
LA25	970	969	955	941	941	**936**	946
LA26	1169	1149	1138	1125	**1123**	1172	1146
LA27	1230	1236	1215	1186	**1184**	1253	1201
LA28	1204	1197	1169	1149	**1147**	1206	1156
LA29	1210	1205	1157	1118	**1115**	1196	1141
LA30	1253	1286	1225	**1204**	**1204**	1262	1228
LA31	1596	1593	1556	**1539**	1541	1674	1553
LA32	1769	1757	**1698**	**1698**	**1698**	1818	**1698**
LA33	1575	1575	**1547**	**1547**	**1547**	1701	**1547**
LA34	1627	1636	1623	**1599**	**1599**	1694	1619
LA35	**1736**	**1736**	**1736**	**1736**	**1736**	1831	**1736**
LA36	1247	1235	1171	1162	**1160**	**1160**	1169
LA37	1453	1456	1418	**1397**	**1397**	**1397**	**1397**
LA38	1185	1185	1172	1144	**1143**	1149	1146
LA39	1226	1226	1207	**1184**	**1184**	**1184**	1194
LA40	1214	1236	1150	1150	**1146**	1173	1158

Tabla 5.4: Resultados presentados por diferentes técnicas sobre las instancias edata

Los resultados obtenidos por Gurobi en las instancias del banco rdata son mostrados en la Tabla 5.5, que incluye el nombre de la instancia, la mejor solución factible (cota superior) encontrada, la cota inferior producida por Gurobi y el tiempo de ejecución. Se debe destacar, que estas instancias son más difíciles que las incluidas en el banco edata, ya que aumenta el número de máquinas capaces de procesar algunas de las operaciones. En parte, esta dificultad se ve reflejada en el deterioro del desempeño de Gurobi, que solo en 5 de las 40 instancias garantiza la optimalidad de la solución reportada en el tiempo establecido, LA16 a LA20.

En la Tabla 5.6, se muestran los resultados reportados para distintas técnicas heurísticas sobre estas instancias. Las últimas dos columnas incluyen los resultados obtenidos por Gurobi y la implementación de TS propuesta para esta tesis. En esta tabla destaca el resultado obtenido por Gurobi en la instancia LA02, ya que alcanza el mejor valor reportado en la literatura, sin dar garantía de optimalidad. Para el resto de las instancias, la eficiencia de Gurobi empeora conforme aumenta el número de trabajos considerados en las instancias. De hecho, las soluciones de peor calidad, en comparación con las reportadas por otros algoritmos, se encuentran de la instancia LA26 a LA35, que presentan el mayor número de trabajos en este banco de instancias.

En el caso de TS, el valor indicado es el de la mejor solución obtenida de las 10 ejecuciones realizadas. Se observa que TS encuentra el óptimo sólo en la instancia LA17. Sin embargo, las soluciones obtenidas siguen estando cerca de la mejor amplitud de proceso reportada. En el peor de los casos TS se queda a 7.64 % de la mejor solución conocida, como ocurre en LA37. Estos resultados muestran un deterioro en el desempeño de la eficiencia de TS, que se puede atribuir a la dificultad de las instancias. Sin embargo, el algoritmo continúa generando soluciones de buena calidad cercanas a los mejores costos conocidos.

Instancia	rdata	cota	gap %	tiempo
LA01	572	453	3600.06	
LA02	530	473	3600.01	
LA03	478	378	3600.08	
LA04	503	403	3600.06	
LA05	458	380	3600.03	
LA06	801	413	3600.11	
LA07	755	411	3600.11	
LA08	767	389	3600.16	
LA09	858	467	3600.23	
LA10	805	443	3600.06	
LA11	1087	516	3600.14	
LA12	947	433	3600.12	
LA13	1045	447	3600.14	
LA14	1076	443	3600.06	
LA15	1097	514	3600.08	
LA16	717	717	56.98	
LA17	646	646	7.22	
LA18	666	666	26.06	
LA19	700	700	113.89	
LA20	756	756	14.08	
LA21	879	717	3600.06	
LA22	804	673	3600.05	
LA23	899	663	3600.11	
LA24	854	704	3600.14	
LA25	827	736	3600.03	
LA26	1169	717	3600.31	
LA27	1202	686	3600.11	
LA28	1176	756	3600.97	
LA29	1108	723	3600.16	
LA30	1255	799	3600.06	
LA31	1951	737	3601.12	
LA32	1946	779	3600.19	
LA33	1657	723	3600.34	
LA34	1820	702	3600.14	
LA35	1918	733	3600.51	
LA36	1044	1016	3600.03	
LA37	1167	1020	3600.16	
LA38	977	952	3600.05	
LA39	1070	985	3600.05	
LA40	995	955	3600.03	

Tabla 5.5: Resultados presentados por Gurobi sobre las instancias rdata

Instancia	TSN1	TSN2	IATS	TS	PBM2h	PBS2h	HA	Gurobi	TS
LA01	577	577	574	571	572	574	**570**	572	575
LA02	535	535	532	**530**	**530**	532	**530**	**530**	537
LA03	481	486	479	478	478	482	**477**	478	484
LA04	509	506	504	**502**	**502**	509	**502**	503	509
LA05	460	458	458	**457**	458	462	**457**	458	459
LA06	801	803	800	**799**	**799**	801	**799**	801	801
LA07	752	752	750	750	750	751	**749**	755	753
LA08	767	768	767	**765**	**765**	767	**765**	767	767
LA09	859	857	854	**853**	**853**	856	**853**	858	857
LA10	806	805	805	**804**	805	807	**804**	805	808
LA11	1073	1073	1072	**1071**	**1071**	1072	**1071**	1087	1073
LA12	937	937	**936**	**936**	**936**	937	**936**	947	937
LA13	1039	1039	**1038**	**1038**	**1038**	1039	**1038**	1045	1039
LA14	1071	1071	**1070**	**1070**	**1070**	1071	**1070**	1076	1071
LA15	1093	1093	**1090**	**1090**	**1090**	**1090**	**1090**	1097	1091
LA16	**717**	**717**	**717**	**717**	N/A	N/A	**717**	**717**	742
LA17	**646**	**646**	**646**	**646**	N/A	N/A	**646**	**646**	**646**
LA18	674	673	669	**666**	N/A	N/A	**666**	**666**	687
LA19	725	709	703	**700**	N/A	N/A	**700**	**700**	736
LA20	**756**	**756**	**756**	**756**	N/A	N/A	**756**	**756**	758
LA21	861	861	846	**835**	N/A	N/A	**835**	879	897
LA22	790	795	772	**760**	N/A	N/A	**760**	804	818
LA23	884	887	853	842	N/A	N/A	**840**	899	893
LA24	825	830	820	808	N/A	N/A	**806**	854	860
LA25	823	821	802	791	N/A	N/A	**789**	827	840
LA26	1086	1087	1070	**1061**	N/A	N/A	**1061**	1169	1097
LA27	1109	1115	1100	1091	N/A	N/A	**1089**	1202	1134
LA28	1097	1090	1085	1080	N/A	N/A	**1079**	1176	1113
LA29	1016	1017	1004	998	N/A	N/A	**997**	1108	1030
LA30	1105	1108	1089	**1078**	N/A	N/A	**1078**	1255	1118
LA31	1532	1533	1528	**1521**	N/A	N/A	**1521**	1951	1539
LA32	1668	1668	1660	**1659**	N/A	N/A	**1659**	1946	1676
LA33	1511	1507	1501	**1499**	N/A	N/A	**1499**	1657	1517
LA34	1542	1543	1539	**1536**	N/A	N/A	**1536**	1820	1548
LA35	1559	1559	1555	**1550**	N/A	N/A	**1550**	1918	1570
LA36	1054	1071	1030	1030	N/A	N/A	**1028**	1044	1098
LA37	1122	1132	1082	1077	N/A	N/A	**1074**	1167	1156
LA38	1004	1001	989	962	N/A	N/A	**960**	977	1029
LA39	1041	1068	**1024**	**1024**	N/A	N/A	**1024**	1070	1080
LA40	1009	1009	980	**970**	N/A	N/A	**970**	995	1039

Tabla 5.6: Comparativa de resultados obtenidos por Gurobi y TS sobre las instancias rdata en el FJSSP

Por último, se presentan las tablas con los resultados obtenidos por Gurobi en las instancias del banco vdata. En la Tabla 5.8, se incluye el nombre de la instancia, la mejor solución factible (cota superior) encontrada, la cota inferior producida por Gurobi y el tiempo de ejecución. Se debe destacar, que estas instancias son las más difíciles que las incluyen en esta serie de experimentos, debido a que se aumenta el número de máquinas capaces de procesar algunas de las operaciones. De hecho, todas las operaciones puede ser procesadas en más de una máquina. Al igual que en el banco de instancias anterior, Gurobi sólo garantiza optimalidad en 5 de las 40 instancias, de LA16 a LA20.

En la Tabla 5.8, se muestran los resultados reportados para distintas técnicas heurísticas sobre estas instancias. Las últimas dos columnas incluyen los resultados obtenidos por Gurobi y la implementación de TS propuesta para esta tesis. Nuevamente destaca el resultado obtenido por Gurobi en la instancia LA04, ya que alcanza el mejor valor reportado en la literatura, sin dar garantía de optimalidad. Para el resto de las instancias, la eficiencia de Gurobi empeora conforme aumenta el número de trabajos considerados en las instancias. Se puede observar que de la instancia LA01 a LA20 Gurobi genera soluciones de buena calidad, incluso óptimas. Sin embargo, su desempeño se ve drásticamente afectado en las instancias más complicadas, como son de LA26 a LA40.

En el caso de TS, el valor indicado es el de la mejor solución obtenida de las 10 ejecuciones realizadas. Se observa que TS encuentra el óptimo en 4 instancias. Además, las soluciones obtenidas siguen estando cerca de la mejor amplitud de proceso reportada. En el peor de los casos TS se queda a 7.38 % de la mejor solución conocida. Más aún, se debe destacar que TS obtiene mejores soluciones que Gurobi en 32 instancias. De esta forma, se comprueba que el algoritmo mantiene un buen desempeño incluso en las instancias más difíciles como son de LA31 a LA35.

Instancia	vdata	cota	gap %	tiempo
LA01	572	413	27.80	3600.11
LA02	531	394	25.80	3600.08
LA03	478	349	26.99	3600.05
LA04	502	377	24.90	3600.06
LA05	459	380	17.21	3600.03
LA06	803	441	45.08	3600.05
LA07	755	376	50.20	3600.12
LA08	771	369	52.14	3600.05
LA09	855	382	55.32	3600.14
LA10	807		45.11	3600.12
LA11	1,081	413	61.79	3600.14
LA12	945	408	56.83	3600.22
LA13	1,045	382	63.45	3600.25
LA14	1,081	443	59.02	3600.17
LA15	1,105	378	65.79	3600.12
LA16	717	717	-	116.02
LA17	646	646	-	61.36
LA18	663	663	-	104.49
LA19	617	617	-	452.07
LA20	756	756	-	116.05
LA21	885	717	18.98	3600.22
LA22	800	619	22.63	3600.17
LA23	915	640	30.05	3600.17
LA24	858	704	17.95	3600.27
LA25	816	723	11.40	3600.16
LA26	1,240	717	42.18	3600.24
LA27	1,407	686	51.24	3600.13
LA28	1,288	756	41.30	3600.11
LA29	1,183	723	38.88	3600.13
LA30	1,351	726	46.26	3600.17
LA31	4,213	717	82.98	3600.06
LA32	4,336	756	82.56	3600.06
LA33	4,697	723	84.61	3600.05
LA34	4,071	656	83.89	3600.06
LA35	4,233	647	84.72	3600.06
LA36	1,152	948	17.71	3600.27
LA37	1,515	986	34.92	3607.46
LA38	1,218	943	22.58	3600.11
LA39	1,322	923	30.26	3600.08
LA40	1,235	955	22.67	3600.2

Tabla 5.7: Resultados presentados por Gurobi sobre las instancias vdata

Instancia	TSN1	TSN2	IATS	TS	HA	Gurobi	TS
LA01	573	575	572	**570**	**570**	572	573
LA02	531	530	**529**	**529**	**529**	531	535
LA03	482	481	479	**477**	**477**	478	482
LA04	504	503	503	**502**	**502**	**502**	508
LA05	464	461	460	**457**	**457**	459	465
LA06	802	**799**	800	**799**	**799**	803	802
LA07	751	752	750	**749**	**749**	755	753
LA08	766	766	766	**765**	**765**	771	768
LA09	854	854	**853**	**853**	**853**	855	855
LA10	805	805	805	**804**	**804**	807	806
LA11	1073	1073	**1071**	**1071**	**1071**	1081	1073
LA12	940	940	**936**	**936**	**936**	945	937
LA13	1040	1041	**1038**	**1038**	**1038**	1045	1040
LA14	1071	1080	**1070**	**1070**	**1070**	1081	1071
LA15	1091	1091	**1089**	**1089**	**1089**	1105	1091
LA16	**717**	**717**	**717**	**717**	**717**	**717**	**717**
LA17	**646**	**646**	**646**	**646**	**646**	**646**	**646**
LA18	**663**	**663**	**663**	**663**	**663**	**663**	**663**
LA19	**617**	**617**	**617**	**617**	**617**	**617**	660
LA20	**756**	**756**	**756**	**756**	**756**	**756**	**756**
LA21	826	825	814	806	**804**	885	859
LA22	745	744	744	739	**738**	800	786
LA23	826	829	818	815	**813**	915	869
LA24	796	796	784	**777**	**777**	858	823
LA25	770	769	757	756	**754**	816	801
LA26	1058	1058	1056	1054	**1053**	1240	1073
LA27	1088	1088	1087	**1085**	**1085**	1407	1109
LA28	1073	1073	1072	**1070**	**1070**	1288	1095
LA29	995	996	997	**994**	**994**	1183	1018
LA30	1071	1070	1071	**1069**	**1069**	1351	1092
LA31	1521	1521	1521	**1520**	**1520**	4213	1527
LA32	**1658**	1659	1659	**1658**	**1658**	4336	1671
LA33	1498	1499	1498	**1497**	**1497**	4697	1506
LA34	1536	1538	1537	**1535**	**1535**	4071	1546
LA35	1553	1551	1551	**1549**	**1549**	4233	1561
LA36	**948**	**948**	**948**	**948**	**948**	1152	994
LA37	**986**	**986**	**986**	**986**	**986**	1515	1052
LA38	**943**	**943**	**943**	**943**	**943**	1218	964
LA39	**922**	**922**	**922**	**922**	**922**	1322	990
LA40	**955**	**955**	**955**	**955**	**955**	1235	978

Tabla 5.8: Comparativa de resultados obtenidos por Gurobi y TS sobre las instancias vdata en el FJSSP

5.4. Experimento 3: FJSSP-LS

Los experimentos 1 y 2 fueron diseñados para evaluar la eficacia (en términos de la calidad de las soluciones encontradas) y la eficiencia (en términos de tiempo de cómputo) de las dos técnicas de optimización utilizadas en este estudio. Se efectuaron sobre problemas (JSSP y FJSSP) más sencillos que el abordado en esta tesis (FJSSP-LS) para poder comparar su desempeño respetivo con los de técnicas del estado del arte. Se observó que Gurobi, aunque identifica soluciones factibles cercanas a las propuestas en la literatura, no es siempre capaz de garantizar la solución óptima. Esta tendencia va aumentando conforme crece el tamaño de las instancia, no sólo en términos de número de trabajos y máquinas, sino también según el grado de flexibilidad (las instancias vdata, por permitir más máquinas por operación, son más difíciles de resolver que las rdata o edata). Por otro lado, el comportamiento de TS es robusto ya que sus resultados nunca quedan muy lejos de las mejores soluciones conocidas.

Es de esperar que un efecto similar se obtenga al considerar el modelo para FJSSP-LS: aunque introducir la división de lotes al FJSSP debería permitir mejorar el valor de la amplitud del proceso (aprovechando mejor los tiempos muertos), el crecimiento ocasionado en el tamaño del espacio de búsqueda (el número de variables binarias aumenta drásticamente) hace más complicada la tarea del método de resolución, siendo más difícil encontrar soluciones de calidad. El objetivo de este experimento es aumentar paulatinamente el grado de dificultad del problema para observar las tendencias sobre las soluciones obtenidas.

Estas corridas preliminares fueron realizadas utilizando Gurobi, sólo sobre una instancia, LA16 en edata, que pareció ser sencilla ya que Gurobi identifica la solución óptima para todos los grados posibles de flexibilidad (edata, rdata y vdata). El modelo para FJSSP-LS fue sin embargo modificado, para permitir un crecimiento gradual de la complejidad del problema, generando dos versiones del problema simplificado: *(i)* de los 10 trabajos a programar, sólo dos de ellos pueden ser divididos (se probaron todas las combinaciones posibles) y para cada uno de ellos, un máximo de dos sublotes podrá ser formado; *(ii)* además de las condiciones anteriores, se impone que el tamaño de los dos sublotes sea idéntico.

Los resultados para las condiciones *(i)* se presentan en la Tabla 5.9. La primera columna indica cuales fueron los sublotes modificados (01_02 significa que el trabajo 1 y 2 tienen un máximo 2 sublotes y los otros trabajos no pueden ser divididos), la segunda el mejor resultado obtenido por Gurobi, la tercera muestra la cota inferior producida en los casos que en que no se garantiza la optimalidad en el tiempo establecido y en la última columna se

incluye el tiempo de ejecución de Gurobi. Los resultados incluidos en la tabla muestran que Gurobi siempre encuentra una solución de mejor calidad que cuando no hay división de lotes (el óptima para LA16 en edata del FJSSP es 892). Sin embargo, Gurobi ya no es capaz de garantizar la optimalidad para todas las instancias, como ocurre en las instancias 02_03, 02_04, 03_06, 03_07, entre otras (15 en total). En estos casos, el crecimiento del espacio de búsqueda impidió la convergencia en un tiempo razonable, demostrando el efecto sobre la complejidad inducido por la estrategia de división de lotes. Como se verá en los experimentos siguientes, Gurobi no es capaz de garantizar la optimalidad para ninguna instancia del FJSSP-LS, confirmando el efecto mencionado.

LA16	Mejor resultado	Cota	tiempo(s)	LA16	Mejor resultado	Cota	tiempo(s)
01_02	852		505	04_05	811	810	86,400
01_03	877		58	04_06	838		4,545
01_04	843		23,476	04_07	830	812	86,400
01_05	817		215	04_08	843		265
01_06	863		127	04_09	833	828	86,400
01_07	866		23,470	04_10	797	741	86,400
01_08	878		116	05_06	826		2,799
01_09	878		240	05_07	816		12,205
01_10	816		2,018	05_08	820		245
02_03	864	850	86,400	05_09	820	819	86,400
02_04	833	827	86,400	05_10	794	773	86,400
02_05	816		430	06_07	866	842	86,400
02_06	847		4,621	06_08	875		51
02_07	851		435	06_09	862		43,149
02_08	861		25	06_10	823	816	86,400
02_09	857		41,162	07_08	875		135
02_10	805		9,011	07_09	866		4,715
03_04	827		329	07_10	820	803	86,400
03_05	804		990	08_09	881		6
03_06	858	846	86,400	08_10	830		575
03_07	860	855	86,400	09_10	814	813	86,400
03_08	880		87				
03_09	856	844	86,400				
03_10	812						

Tabla 5.9: Mejores resultados reportados por Gurobi en la instancia LA16, modificando dos trabajos en 2 sublotes

Para las siguientes pruebas se estableció una condición más restrictiva, *(ii)* los dos sublotes formados, deben ser del mismo tamaño. Los resultados obtenidos por Gurobi se encuentran en la Tabla 5.10. Al comparar los resultados de los dos últimos experimentos, se observa que si los dos sublotes son del mismo tamaño, la eficiencia de Gurobi mejora significativamente. Es importante destacar que Gurobi fue capaz de garantizar optimalidad con todas las instancias, usando un tiempo computacional menor: la reducción del espacio de búsqueda debida a una condición más estricta permite la convergencia.

Por otro lado, para la mayoría de instancias, las soluciones óptimas encontradas en esta parte se caracterizan por una amplitud del proceso mayor a las determinadas en el caso *(i)*: en estos casos, obligar el tamaño de los sublotes a ser iguales no permite alcanzar soluciones tan buenas como cuando estos tamaños se dejaban libres (la región factible es mayor). En algunas otras instancias (por ejemplo, 02_03 y 05_10), la solución óptima obtenida con la condición *(ii)* es mejor (amplitud del proceso menor) que la del experimento *(ii)*. Esto se debe a que el problema con la condición *(i)* es más difícil de resolver (de hecho, para las dos instancias mencionadas, no se había probado la optimalidad), que impide encontrar soluciones de calidad. En estos casos, la complejidad del problema y la dificultad experimentada para converger explican que no se obtuvieran mejores soluciones.

LA16	Mejor resultado	tiempo(s)	LA16	Mejor resultado	tiempo(s)
01_02	857	35	04_05	813	88
01_03	877	30	04_06	843	113
01_04	846	133	04_07	836	65
01_05	827	14	04_08	850	38
01_06	865	14	04_09	837	30
01_07	868	29	04_10	799	1,737
01_08	881	6	05_06	828	8
01_09	881	6	05_07	821	13
01_10	824	2	05_08	823	6
02_03	861	14	05_09	829	25
02_04	833	46	05_10	786	143
02_05	830	7	06_07	868	150
02_06	855	17	06_08	875	7
02_07	855	15	06_09	875	24
02_08	861	39	06_10	828	160
02_09	858	23	07_08	884	24
02_10	812	99	07_09	876	31
03_04	827	36	07_10	828	289
03_05	804	23	08_09	887	42
03_06	859	4	08_10	831	135
03_07	860	34	09_10	814	76
03_08	880	13			
03_09	863	7			
03_10	822	36			

Tabla 5.10: Mejores resultados reportados por Gurobi en la instancia LA16, modificando dos trabajos en dos sublotes del mismo tamaño

5.5. Experimento 4: FJSSP-LS edata

Las siguientes pruebas se hicieron modificando las demandas y el número máximo de sublotes en las instancias edata, rdata y vdata. Las modificaciones fueron generadas de manera aleatoria, para la demanda el número de unidades a procesar se selecciona de forma aleatoria del siguiente conjunto $\{1000, 1100, 1200, ..., 2000\}$ y el número máximo de sublotes permitidos está entre 2 y 6. La demanda y el número máximo de sublotes es el mismo para los 3 juegos de instancias y se encuentran en el Apéndice D.

En la Tabla 5.11 se muestra la siguiente información: la columna uno es el nombre de la instancia, la segunda columna muestra la dimensión del problema (trabajos $\times$ máquinas), de la columna tres a la cinco son datos reportados por Gurobi, que incluyen la mejor solución factible obtenida, la cota inferior producida y el tiempo de ejecución. Por último, en las columnas seis y siete se incluyen el mejor resultado encontrado por TS y el tiempo promedio empleado para encontrarlo, respectivamente.

En el juego de instancias edata, Gurobi ya no es capaz de garantizar optimalidad y empieza a reportar soluciones con un costo mayor que TS. Este comportamiento se muestra en casos como son LA01 a LA05, donde la diferencia ente los costos reportados por ambas técnicas es pequeña. Sin embargo, a medida que la dimensión del problema aumenta la diferencia entre Gurobi y TS también aumenta, como se observa en las instancias que van desde LA21 hasta LA40. Finalmente, para las instancias LA31 a LA35, Gurobi no es capaz de dar una solución en el tiempo de cómputo establecido. Se debe destacar que Gurobi fue limitado a un tiempo de ejecución de una hora en todas las instancias, mientras que en las primeras 20 instancias TS emplea menos de una hora y en las veinte instancias restantes, el tiempo de ejecución es cercano a una hora. Ahora bien, TS también fue limitada a ejecutarse durante una hora, pero fue diseñada para reportar el tiempo que requirió para encontrar la mejor solución visitada, por lo tanto, si el tiempo reportado se encontraba cercano a una hora, se puede inferir que aún podía seguir mejorando la calidad de las soluciones si se le permitía ejecutarse por más tiempo.

		Gurobi			Tabú	
Instancia	$n \times m$	best	bound	Tiempo(s)	best	Tiempo promedio(s)
LA01	10×5	615.895	216.462	3600.08	609.079	34.58
LA02	10×5	653.867	243.872	3600.22	636.372	180.24
LA03	10×5	571.481	95.285	3600.09	514.170	541.94
LA04	10×5	554.733	194.276	3600.12	525.111	181.82
LA05	10×5	509.021	135.775	3600.14	503.079	86.12
LA06	15×5	890.306	216.394	3600.25	833.228	133.27
LA07	15×5	828.118	180.822	3600.16	751.306	1111.14
LA08	15×5	894.468	228.460	3600.27	815.119	817.05
LA09	15×5	929.498	238.714	3600.27	853.402	787.96
LA10	15×5	937.090	151.437	3600.17	866.647	319.87
LA11	20×5	1,157.307	156.348	3600.13	1,090.550	2626.00
LA12	20×5	1,079.362	186.997	3600.08	949.382	1919.07
LA13	20×5	1,132.042	200.322	3600.4	1,053.367	816.42
LA14	20×5	1,195.248	196.609	3600.11	1,111.091	1582.12
LA15	20×5	1,350.506	139.048	3600.21	1,111.857	2670.51
LA16	10×10	796.315	374.413	3600.35	660.073	2769.11
LA17	10×10	643.203	201.438	3600.18	566.013	1323.50
LA18	10×10	707.608	338.684	3600.15	625.385	1495.88
LA19	10×10	739.530	322.993	3600.05	663.000	760.45
LA20	10×10	779.141	316.475	3600.23	669.753	902.76
LA21	15×10	1,163.112	314.828	3600.06	894.379	2903.60
LA22	15×10	1,069.151	278.957	3600.19	795.521	3266.17
LA23	15×10	1,173.406	332.189	3600.24	939.342	2829.80
LA24	15×10	984.349	349.133	3600.43	860.464	3120.42
LA25	15×10	1,091.300	308.696	3600.1	810.541	3202.75
LA26	20×10	1,431.745	334.973	3600.23	1,123.161	2634.47
LA27	20×10	1,473.577	338.470	3600.19	1,172.772	2825.09
LA28	20×10	1,383.823	316.000	3600.85	1,132.670	3161.52
LA29	20×10	1,682.738	262.590	3600.26	1,169.868	3170.34
LA30	20×10	1,649.383	347.085	3600.25	1,219.269	3357.94
LA31	30×10	-	298.794	3600.1	1,627.306	2867.45
LA32	30×10	-	252.158	3600.72	1,801.890	3281.05
LA33	30×10	-	308.996	3600.86	1,665.925	3414.45
LA34	30×10	-	233.500	3600.07	1,762.539	3428.84
LA35	30×10	-	289.500	3600.07	1,827.242	3438.85
LA36	15×10	1,374.646	385.500	3600.09	978.728	3460.44
LA37	15×10	1,579.237	416.545	3600.27	1,120.007	3473.20
LA38	15×10	1,297.249	500.403	3600.17	966.641	3409.80
LA39	15×10	1,427.272	455.719	3600.09	1,008.595	3397.86
LA40	15×10	1,314.234	367.064	3600.31	970.234	3487.17

Tabla 5.11: edata: Comparativa del FJSSP-LS usando Gurobi y TS

Para proporcionar indicaciones sobre la robustez de TS, se presentan en la Tabla 5.12 la mejor solución obtenida por TS sobre las 10 ejecuciones, el valor promedio, la desviación estándar y la peor solución obtenida. Se puede observar que la diferencia entre mejor valor obtenido y valor promedio siempre es pequeña, en promedio igual a 2.21 % (la distancia relativa máxima es igual a 8.33 %). En todos los casos, la robustez del comportamiento de TS se ve confirmada por los bajos valores de la desviación estándar, en promedio igual a 2.11 % del valor promedio de la amplitud del proceso.

Instancia	Mejor sol.	Promedio	Desv. Est.	Peor sol.	Tiempo promedio(s)
LA01	609.08	611.01	6.11	628.41	34.58
LA02	636.37	636.55	0.08	636.61	195.53
LA03	514.17	514.33	0.16	514.54	544.96
LA04	525.11	525.11	0.00	525.11	236.31
LA05	503.08	503.08	0.00	503.08	57.90
LA06	833.23	834.82	3.46	843.03	154.09
LA07	751.31	752.49	1.02	754.81	1,175.59
LA08	815.12	815.51	0.51	816.57	867.26
LA09	853.40	854.04	0.31	854.38	811.73
LA10	866.65	866.65	0.00	866.65	319.87
LA11	1,090.55	1,107.31	14.29	1,140.46	2,626.00
LA12	949.38	954.72	10.81	989.53	1,919.07
LA13	1,053.37	1,054.03	2.37	1,061.92	816.42
LA14	1,111.09	1,119.04	19.05	1,178.52	1,582.12
LA15	1,111.86	1,145.16	72.32	1,360.72	2,670.51
LA16	660.07	674.29	13.26	697.75	2,769.11
LA17	566.01	570.98	7.68	585.01	1,323.50
LA18	625.38	632.02	7.06	643.08	1,495.88
LA19	663.00	663.00	0.00	663.00	760.45
LA20	669.75	674.54	5.99	688.86	902.76
LA21	894.38	918.52	17.24	944.45	2,903.60
LA22	795.52	838.52	29.34	877.64	3,266.17
LA23	939.34	965.21	42.10	1,086.90	2,829.80
LA24	860.46	869.52	6.91	881.25	3,120.42
LA25	810.54	821.75	6.49	831.86	3,088.73
LA26	1,123.16	1,216.73	123.33	1,508.74	2,634.47
LA27	1,172.77	1,197.97	23.25	1,238.27	2,825.09
LA28	1,132.67	1,162.96	18.34	1,192.86	3,161.52
LA29	1,169.87	1,252.31	152.87	1,709.57	3,170.34
LA30	1,219.27	1,247.74	14.95	1,269.73	3,357.94
LA31	1,627.31	1,759.05	128.30	2,054.71	2,867.45
LA32	1,801.89	1,857.81	28.20	1,890.78	3,281.05
LA33	1,665.92	1,717.62	30.00	1,761.93	3,414.45
LA34	1,762.54	1,825.63	37.50	1,880.71	3,428.84
LA35	1,827.24	1,920.05	63.71	2,039.21	3,438.85
LA36	978.73	1,010.62	20.97	1,040.77	3,460.44
LA37	1,120.01	1,171.35	41.50	1,236.67	3,473.20
LA38	966.64	1,015.96	27.38	1,045.59	3,409.80
LA39	1,008.59	1,031.93	17.17	1,059.32	3,397.86
LA40	970.23	1,000.57	23.00	1,037.89	3,487.17

Tabla 5.12: edata: Estadísticas para TS

Para determinar si el uso de una división de lotes en realidad contribuye a disminuir la amplitud del proceso, se compararon los mejores resultados reportados para FJSSP en el banco de instancias edata con los resultados obtenidos por el algoritmo diseñado en esta tesis. En la Tabla 5.13, se presentan las instancias empleadas así como sus tamaños. En la tercera columna se muestra el mejor resultado reportado en la literatura para el FJSSP, la cuarta es el resultado obtenido por TS para el FJSSP-LS. La quinta columna refleja el aumento o la disminución de la amplitud de proceso, cuando los números son negativos indican reducciones en la amplitud del proceso al aplicar división de lotes. Se observa que hubo reducción en 26 casos de las 40 instancias probadas.

Para facilitar la comparación de los resultados reportados para FJSSP y FJSSP-LS, la Tabla 5.13 se dividió en dos tablas, 5.14 y 5.15. En la primera se incluyen a las instancias donde se ha obtenido una disminución en la amplitud de proceso, mientras que en la segunda se presentan las instancias en las cuales la amplitud de proceso obtenida por TS es mayor a la reportada para FJSSP. En la Tabla 5.14, se puede observar que existen instancias en las cuales se obtienen disminuciones muy pequeñas, como es el caso de la instancia LA27 en la cual se produce una mejora de menos del 1 %, y otras en las cuales hay disminuciones importantes, como en los casos LA16 y LA18 que mejoran más del 25 %. Por otro lado, en la Tabla 5.15, se observa que las primeras siete instancias tienen un aumento de amplitud de proceso menor al 0.1 %. Sin embargo, la calidad de las soluciones obtenidas por TS disminuye en las últimas siete instancias, llegando a obtener un deterioro de hasta 10.23 %. No obstante, se observó que para estas instancias, la mejor solución reportada se encontraba cerca del tiempo límite establecido para los dos algoritmos, una hora. Por lo cual, se decidió realizar algunas pruebas en las cuales TS podía emplear hasta 10 horas en este conjunto de instancias. La amplitud de proceso obtenida por TS en estas corridas se presenta en la Tabla 5.16. La ganancia obtenida es importante, ya que se obtiene una mejora en LA29 y LA30 de aproximadamente el 5 % y en el peor de los casos la pérdida, en comparación con FJSSP, es de 1.5 %.

Instancia	$n \times m$	FJSSP	FJSSP-LS	Diferencia	Tiempo promedio
LA01	10×5	609	609.079	0.079	34.58
LA02	10×5	655	636.372	-18.628	180.24
LA03	10×5	550	514.170	-35.830	541.94
LA04	10×5	568	525.111	-42.889	181.82
LA05	10×5	503	503.079	0.079	86.12
LA06	15×5	833	833.228	0.228	133.27
LA07	15×5	762	751.306	-10.694	1111.14
LA08	15×5	845	815.119	-29.881	817.05
LA09	15×5	878	853.402	-24.598	787.96
LA10	15×5	866	866.647	0.647	319.87
LA11	20×5	1,103	1,090.550	-12.450	2626.00
LA12	20×5	960	949.382	-10.618	1919.07
LA13	20×5	1,053	1,053.367	0.367	816.42
LA14	20×5	1,123	1,111.091	-11.909	1582.12
LA15	20×5	1,111	1,111.857	0.857	2670.51
LA16	10×10	892	660.073	-231.927	2769.11
LA17	10×10	707	566.013	-140.987	1323.50
LA18	10×10	842	625.385	-216.615	1495.88
LA19	10×10	796	663.000	-133.000	760.45
LA20	10×10	857	669.753	-187.247	902.76
LA21	15×10	1,014	894.379	-119.621	2903.60
LA22	15×10	880	795.521	-84.479	3266.17
LA23	15×10	950	939.342	-10.658	2829.80
LA24	15×10	909	860.464	-48.536	3120.42
LA25	15×10	941	810.541	-130.459	3202.75
LA26	20×10	1,123	1,123.161	0.161	2634.47
LA27	20×10	1,184	1,172.772	-11.228	2825.09
LA28	20×10	1,147	1,132.670	-14.330	3161.52
LA29	20×10	1,115	1,169.868	54.868	3170.34
LA30	20×10	1,204	1,219.269	15.269	3357.94
LA31	30×10	1,541	1,627.306	86.306	2867.45
LA32	30×10	1,698	1,801.890	103.890	3281.05
LA33	30×10	1,547	1,665.925	118.925	3414.45
LA34	30×10	1,599	1,762.539	163.539	3428.84
LA35	30×10	1,736	1,827.242	91.242	3438.85
LA36	15×10	1,160	978.728	-181.272	3460.44
LA37	15×10	1,397	1,120.007	-276.993	3473.20
LA38	15×10	1,143	966.641	-176.359	3409.80
LA39	15×10	1,184	1,008.595	-175.405	3397.86
LA40	15×10	1,146	970.234	-175.766	3487.17

Tabla 5.13: Comparación entre FJSSP y FJSSP-LS usando TS en la instancia edata

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA02	655	636.37	2.84
LA03	550	514.17	6.51
LA04	568	525.11	7.55
LA07	762	751.31	1.40
LA08	845	815.12	3.54
LA09	878	853.40	2.80
LA11	1,103	1,090.55	1.13
LA12	960	949.38	1.11
LA14	1,123	1,111.09	1.06
LA16	892	660.07	26.00
LA17	707	566.01	19.94
LA18	842	625.38	25.73
LA19	796	663.00	16.71
LA20	857	669.75	21.85
LA21	1,014	894.38	11.80
LA22	880	795.52	9.60
LA23	950	939.34	1.12
LA24	909	860.46	5.34
LA25	941	810.54	13.86
LA27	1,184	1,172.77	0.95
LA28	1,147	1,132.67	1.25
LA36	1,160	978.73	15.63
LA37	1,397	1,120.01	19.83
LA38	1,143	966.64	15.43
LA39	1,184	1,008.59	14.81
LA40	1,146	970.23	15.34

Tabla 5.14: edata: Porcentaje de mejora

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA01	609	609.079	0.01
LA05	503	503.079	0.02
LA06	833	833.228	0.03
LA10	866	866.647	0.07
LA13	1,053	1,053.367	0.03
LA15	1,111	1,111.857	0.08
LA26	1,123	1,123.161	0.01
LA29	1,115	1,169.868	4.92
LA30	1,204	1,219.269	1.27
LA31	1,539	1,627.306	5.74
LA32	1,698	1,801.890	6.12
LA33	1,547	1,665.925	7.69
LA34	1,599	1,762.539	10.23
LA35	1,736	1,827.242	5.26

Tabla 5.15: edata: Porcentaje de pérdidas

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA29	1,115	1,049.06	-5.914
LA30	1,204	1,143.85	-4.996
LA31	1,539	1,540.96	0.127
LA32	1,698	1,705.70	0.453
LA33	1,547	1,547.74	0.048
LA34	1,599	1,622.26	1.455
LA35	1,736	1,737.02	0.059

Tabla 5.16: edata: instancias LA29 a LA35 tras diez horas de ejecución

5.6. Experimento 5: FJSSP-LS rdata

Considerando que Gurobi proporcionó soluciones muy inferiores a las de TS en el banco edata, y que no es capaz de identificar soluciones óptimas, sólo se reportan las soluciones obtenidas por TS sobre los bancos rdata y vdata. En la Tabla 5.17 se presentan la mejor solución obtenida por TS sobre las 10 ejecuciones, el valor promedio, la desviación estándar y la peor solución obtenida. De la misma manera que para el banco edata, la diferencia entre el mejor valor obtenido y el valor promedio siempre es pequeña, en promedio igual a 1.57 % (la distancia relativa máxima es igual a 5.21 %). En todos los casos, la robustez del comportamiento de TS se ve confirmada por los bajos valores de la desviación estándar, en promedio igual a 1.12 % del valor promedio de la amplitud del proceso.

Por otro lado, dado el desempeño que tuvo TS en el banco de instancias edata, para determinar si el uso de una división de lotes en realidad contribuye a disminuir la amplitud del proceso en instancias de mayor flexibilidad en la asignación de operaciones, se procedió a realizar las pruebas sobre los dos bancos de instancias restantes, rdata y vdata. En esta sección se presentan los resultados obtenidos en el banco rdata. En la Tabla 5.18, se muestran los mejores resultados reportados para FJSSP junto con los resultados obtenidos por TS, se observa que hubo una reducción en 22 casos de las 40 instancias probadas.

Para facilitar la comparación de los resultados reportados para FJSSP y FJSSP-LS, la Tabla 5.18 se dividió en dos partes. En la primera de ellas, la Tabla 5.19, se puede observar que existe un número mayor de instancias en las cuales se obtienen mejoras muy pequeñas, en comparación con el banco de instancias edata. Como se aprecia en la Tabla 5.19, las mejoras van desde el 0 %, como en la instancia LA11, hasta más del 25 %, como en los casos LA17 y LA20. Las otras instancias, en las cuales se obtuvieron soluciones de menor calidad que las disponibles para FJSSP, se incluyen en la Tabla 5.21. En las primeras siete instancias hay un aumento de amplitud de proceso menor al 0.1 %. Sin embargo, la calidad de las soluciones obtenidas por TS disminuye en las últimas siete instancias, llegando a obtener un deterioro de hasta el 7.05 %. Por lo cual, nuevamente se determinó realizar algunas pruebas en las cuales TS podía emplear hasta 10 horas en este conjunto de instancias. La amplitud de proceso obtenida por TS en estas corridas se presenta en la Tabla 5.22. La ganancia obtenida para la instancia LA30 es de aproximadamente el 0.66 % y en el peor de los casos la pérdida, en comparación con FJSSP, es de 0.37 % para estas últimas corridas.

Instancia	Mejor sol.	Promedio	Desv. Est.	Peor sol.	Tiempo promedio(s)
LA01	570.10	570.30	0.09	570.44	53.38
LA02	529.21	529.30	0.05	529.36	185.69
LA03	476.92	477.06	0.07	477.15	366.82
LA04	501.80	501.89	0.06	501.96	418.88
LA05	456.80	456.89	0.05	456.95	231.49
LA06	799.15	799.29	0.09	799.40	320.42
LA07	749.27	749.37	0.07	749.44	603.57
LA08	765.31	765.45	0.06	765.49	617.41
LA09	853.25	853.33	0.06	853.42	385.58
LA10	804.65	804.72	0.03	804.75	1,024.23
LA11	1,070.96	1,071.37	0.24	1,071.85	2,072.69
LA12	935.64	935.75	0.06	935.82	1,490.88
LA13	1,037.63	1,037.75	0.08	1,037.84	1,077.44
LA14	1,069.60	1,069.67	0.05	1,069.73	1,064.11
LA15	1,090.28	1,091.67	1.76	1,096.29	1,617.09
LA16	559.51	567.96	6.24	580.16	2,693.99
LA17	482.76	487.31	3.26	492.32	2,412.55
LA18	531.90	535.94	2.33	539.12	1,596.44
LA19	554.89	560.67	3.29	565.66	2,687.94
LA20	566.75	572.37	4.77	581.91	1,263.78
LA21	814.45	822.93	8.79	838.89	3,047.66
LA22	773.04	792.75	11.70	812.06	3,346.72
LA23	828.79	848.67	14.13	870.80	3,399.06
LA24	795.30	818.65	39.99	929.84	3,178.61
LA25	756.65	760.02	3.05	767.57	2,670.13
LA26	1,068.80	1,087.66	12.68	1,107.36	3,462.09
LA27	1,105.92	1,126.03	14.89	1,155.41	3,286.69
LA28	1,098.70	1,122.18	12.39	1,141.44	3,391.01
LA29	1,033.31	1,069.88	35.16	1,129.42	3,292.81
LA30	1,096.27	1,137.78	23.14	1,167.80	3,460.60
LA31	1,578.96	1,659.36	50.27	1,732.65	3,500.25
LA32	1,754.78	1,805.96	39.15	1,877.77	3,484.64
LA33	1,604.71	1,642.51	29.77	1,701.99	3,468.46
LA34	1,618.31	1,661.19	37.79	1,734.33	3,225.79
LA35	1,597.53	1,640.00	20.55	1,663.18	3,441.03
LA36	889.59	932.69	27.54	978.70	3,356.84
LA37	965.43	989.45	15.13	1,017.98	3,495.02
LA38	852.80	875.56	17.30	900.46	3,215.91
LA39	867.25	900.71	25.13	950.07	3,235.47
LA40	838.47	882.19	22.75	913.13	3,428.46

Tabla 5.17: rdata: Estadisticas

Instancia	$n \times m$	FJSSP	FJSSP-LS	Diferencia	Tiempo promedio
LA01	10×5	570	570.101	0.101	61.411
LA02	10×5	530	529.212	-0.788	184.706
LA03	10×5	477	476.923	-0.077	285.711
LA04	10×5	502	501.798	-0.202	246.575
LA05	10×5	457	456.797	-0.203	145.624
LA06	15×5	799	799.147	0.147	408.877
LA07	15×5	749	749.273	0.273	614.229
LA08	15×5	765	765.313	0.313	672.233
LA09	15×5	853	853.248	0.248	440.160
LA10	15×5	804	804.653	0.653	1,192.237
LA11	20×5	1,071	1,070.963	-0.037	2,126.102
LA12	20×5	936	935.640	-0.360	1,571.087
LA13	20×5	1,038	1,037.632	-0.368	1,227.026
LA14	20×5	1,070	1,069.596	-0.404	1,229.469
LA15	20×5	1,090	1,090.284	0.284	1,617.090
LA16	10×10	717	559.507	-157.493	2,693.990
LA17	10×10	646	482.764	-163.236	2,412.552
LA18	10×10	666	531.898	-134.102	1,596.436
LA19	10×10	700	554.892	-145.108	2,687.938
LA20	10×10	756	566.747	-189.253	1,263.777
LA21	15×10	835	814.448	-20.552	3,047.660
LA22	15×10	760	773.041	13.041	3,346.721
LA23	15×10	840	828.789	-11.211	3,399.055
LA24	15×10	806	795.298	-10.702	3,178.612
LA25	15×10	789	756.652	-32.348	2,670.129
LA26	20×10	1,061	1,068.798	7.798	3,462.092
LA27	20×10	1,089	1,105.916	16.916	3,286.688
LA28	20×10	1,079	1,098.701	19.701	3,391.006
LA29	20×10	997	1,033.307	36.307	3,292.806
LA30	20×10	1,078	1,096.272	18.272	3,460.602
LA31	30×10	1,521	1,578.958	57.958	3,500.251
LA32	30×10	1,659	1,754.781	95.781	3,484.645
LA33	30×10	1,499	1,604.712	105.712	3,468.460
LA34	30×10	1,536	1,618.307	82.307	3,225.791
LA35	30×10	1,550	1,597.531	47.531	3,441.029
LA36	15×10	1,028	889.593	-138.407	3,356.844
LA37	15×10	1,074	965.431	-108.569	3,495.017
LA38	15×10	960	852.805	-107.195	3,215.911
LA39	15×10	1,024	867.249	-156.751	3,235.472
LA40	15×10	970	838.474	-131.526	3,428.463

Tabla 5.18: rdata: Comparativa FJSSP y FJSSP-LS usando TS

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA02	530	529.212	0.15
LA03	477	476.923	0.02
LA04	502	501.798	0.04
LA05	457	456.797	0.04
LA11	1,071	1,070.963	0.00
LA12	936	935.640	0.04
LA13	1,038	1,037.632	0.04
LA14	1,070	1,069.596	0.04
LA16	717	559.507	21.97
LA17	646	482.764	25.27
LA18	666	531.898	20.14
LA19	700	554.892	20.73
LA20	756	566.747	25.03
LA21	835	814.448	2.46
LA23	840	828.789	1.33
LA24	806	795.298	1.33
LA25	789	756.652	4.10
LA36	1,028	889.593	13.46
LA37	1,074	965.431	10.11
LA38	960	852.805	11.17
LA39	1,024	867.249	15.31
LA40	970	838.474	13.56

Tabla 5.19: rdata: Porcentaje de mejora

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA01	570	570.101	0.02
LA06	799	799.147	0.02
LA07	749	749.273	0.04
LA08	765	765.313	0.04
LA09	853	853.248	0.03
LA10	804	804.653	0.08
LA15	1,090	1,090.284	0.03
LA22	760	773.041	1.72
LA26	1,061	1,068.798	0.74
LA27	1,089	1,105.916	1.55
LA28	1,079	1,098.701	1.83
LA29	997	1,033.307	3.64
LA30	1,078	1,096.272	1.70
LA31	1,521	1,578.958	3.81
LA32	1,659	1,754.781	5.77
LA33	1,499	1,604.712	7.05
LA34	1,536	1,618.307	5.36
LA35	1,550	1,597.531	3.07

Tabla 5.20: rdata: Porcentaje de pérdida

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA01	570	570.101	0.02
LA06	799	799.147	0.02
LA07	749	749.273	0.04
LA08	765	765.313	0.04
LA09	853	853.248	0.03
LA10	804	804.653	0.08
LA15	1,090	1,090.284	0.03
LA22	760	773.041	1.72
LA26	1,061	1,068.798	0.74
LA27	1,089	1,105.916	1.55
LA28	1,079	1,098.701	1.83
LA29	997	1,033.307	3.64
LA30	1,078	1,096.272	1.70
LA31	1,521	1,578.958	3.81
LA32	1,659	1,754.781	5.77
LA33	1,499	1,604.712	7.05
LA34	1,536	1,618.307	5.36
LA35	1,550	1,597.531	3.07

Tabla 5.21: rdata: Porcentaje de pérdida

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA29	997	996.85	-0.015
LA30	1,078	1,070.89	-0.660
LA31	1,521	1,524.40	0.223
LA32	1,659	1,663.94	0.298
LA33	1,499	1,504.64	0.376
LA34	1,536	1,618.31	0.181
LA35	1,550	1,597.53	0.093

Tabla 5.22: rdata: instancias LA29 a LA35 tras diez horas de ejecución

5.7. Experimento 6: FJSSP-LS vdata

Las corridas finales fueron hechas con el banco vdata. Como en el experimento anterior, sólo se presentan los resultados obtenidos con TS. En la Tabla 5.23 se presenta la mejor solución obtenida por TS sobre las 10 ejecuciones, el valor promedio, la desviación estándar y la peor solución obtenida. De la misma manera que para los bancos anteriores, la diferencia entre mejor valor obtenido y valor promedio siempre es pequeña, en promedio igual a 0.61 % (la distancia relativa máxima es igual a 2.43 %). En todos los casos, la robustez del comportamiento de TS se ve confirmada por los bajos valores de la desviación estándar, en promedio igual a 0.44 % del valor promedio de la amplitud del proceso.

Nuevamente se compararon los mejores resultados reportados para FJSSP con los resultados obtenidos por TS. En la Tabla 5.24, se observa que hubo una reducción en 17 casos de las 40 instancias probadas, de las cuales diez se pueden considerar como importantes.

En la Tabla 5.25, al igual que en rdata, existe un número mayor de instancias en las cuales se obtienen disminuciones muy pequeñas, en comparación con el banco de instancias edata. Nuevamente, se divide la tabla en dos partes para facilitar el análisis de los resultados. En la Tabla 5.25, se presentan las instancias en las cuales se logra disminuir la amplitud del proceso. En este caso, se producen mejoras muy pequeñas, como es el caso de las primeras instancias en la tabla, en las cuales la amplitud de proceso se reduce en menos del 0.07 % y otras en las cuales hay disminuciones importantes, como en los casos LA17 y LA20, que mejoran más del 27 %. Por otro lado, en la Tabla 5.26 se incluyen las instancias en las cuales TS obtuvo una amplitud de proceso mayor que la reportada para FJSSP. Se observa que la pérdida es menor al 2 %. Al igual que en los bancos edata y rdata, la calidad de las soluciones obtenidas por TS disminuye en las últimas siete instancias. Por lo cual, se decidió realizar algunas pruebas en las cuales TS podía emplear hasta 10 horas en este conjunto de instancias. La amplitud de proceso obtenida por TS en estas corridas se presenta en la Tabla 5.27. Para estas últimas instancias no hubo una reducción en la amplitud de proceso. Sin embargo, para este banco, ninguna de las instancias llega a presentar un aumento en la amplitud de proceso superior al 1 %.

Instancia	Mejor sol.	Promedio	Desv. Est.	Peor sol.	Tiempo promedio(s)
LA01	570.16	570.36	0.20	570.81	118.16
LA02	529.22	530.14	2.56	537.42	183.65
LA03	476.97	477.34	0.50	478.69	298.00
LA04	501.72	501.87	0.14	502.14	144.94
LA05	456.85	457.05	0.14	457.21	168.77
LA06	799.24	799.37	0.12	799.55	713.17
LA07	749.37	750.74	3.14	759.34	307.50
LA08	765.28	765.80	1.25	769.35	354.97
LA09	853.27	853.43	0.08	853.57	299.64
LA10	804.58	804.73	0.06	804.82	843.31
LA11	1,071.23	1,072.39	1.09	1,074.29	2,274.10
LA12	935.62	935.87	0.22	936.29	991.96
LA13	1,037.68	1,037.93	0.33	1,038.74	1,376.97
LA14	1,069.60	1,071.09	3.53	1,080.94	955.58
LA15	1,089.94	1,090.75	1.19	1,093.75	1,447.33
LA16	539.87	541.55	1.52	543.97	2,451.09
LA17	470.23	471.75	0.90	473.66	1,678.00
LA18	522.73	524.65	1.23	526.37	930.93
LA19	538.36	539.54	1.06	541.41	1,675.74
LA20	549.39	551.28	1.39	553.11	922.37
LA21	803.00	805.63	2.22	811.05	2,300.20
LA22	738.62	744.96	4.74	754.61	2,881.58
LA23	817.16	823.41	5.24	834.53	3,211.46
LA24	777.59	784.20	11.61	816.28	2,949.62
LA25	754.70	756.23	1.09	758.19	2,200.91
LA26	1,057.69	1,060.71	2.07	1,064.36	3,387.61
LA27	1,090.86	1,094.66	3.00	1,098.41	3,208.04
LA28	1,074.33	1,079.12	3.15	1,084.76	3,320.77
LA29	1,001.77	1,018.61	9.98	1,031.42	3,423.14
LA30	1,075.71	1,084.65	5.14	1,090.63	3,016.80
LA31	1,531.41	1,562.34	13.54	1,576.53	3,447.55
LA32	1,679.50	1,693.82	8.26	1,705.70	3,314.82
LA33	1,524.48	1,537.58	9.52	1,547.33	3,471.78
LA34	1,558.85	1,568.95	4.68	1,575.73	3,513.38
LA35	1,571.03	1,588.22	7.95	1,596.88	3,357.04
LA36	826.28	838.82	10.56	856.23	3,540.44
LA37	881.14	898.04	7.99	908.33	3,535.21
LA38	778.63	794.94	9.76	804.96	3,252.70
LA39	794.21	813.55	11.09	828.08	3,172.43
LA40	805.87	820.68	13.22	836.86	3,386.64

Tabla 5.23: vdata: Estadisticas

Instancia	$n \times m$	FJSSP	FJSSP-LS	Diferencia	Tiempo promedio
LA01	10×5	570	570.165	0.165	118.162
LA02	10×5	529	529.223	0.223	183.651
LA03	10×5	477	476.966	-0.034	297.999
LA04	10×5	502	501.721	-0.279	144.936
LA05	10×5	457	456.846	-0.154	168.770
LA06	15×5	799	799.240	0.240	713.165
LA07	15×5	749	749.372	0.372	307.496
LA08	15×5	765	765.284	0.284	354.968
LA09	15×5	853	853.266	0.266	299.644
LA10	15×5	804	804.576	0.576	843.314
LA11	20×5	1,071	1,071.234	0.234	2,274.104
LA12	20×5	936	935.625	-0.375	991.958
LA13	20×5	1,038	1,037.683	-0.317	1,376.975
LA14	20×5	1,070	1,069.597	-0.403	955.578
LA15	20×5	1,089	1,089.937	0.937	1,447.332
LA16	10×10	717	539.867	-177.133	2,451.092
LA17	10×10	646	470.228	-175.772	1,678.005
LA18	10×10	663	522.731	-140.269	930.931
LA19	10×10	617	538.365	-78.635	1,675.737
LA20	10×10	756	549.390	-206.610	922.372
LA21	15×10	804	802.998	-1.002	2,300.197
LA22	15×10	738	738.617	0.617	2,881.583
LA23	15×10	813	817.157	4.157	3,211.455
LA24	15×10	777	777.589	0.589	2,949.621
LA25	15×10	754	754.697	0.697	2,200.906
LA26	20×10	1,053	1,057.685	4.685	3,405.972
LA27	20×10	1,085	1,090.863	5.863	3,215.578
LA28	20×10	1,070	1,074.326	4.326	3,269.271
LA29	20×10	994	1,001.774	7.774	3,402.404
LA30	20×10	1,069	1,075.705	6.705	3,208.004
LA31	30×10	1,520	1,531.406	11.406	3,476.862
LA32	30×10	1,658	1,679.496	21.496	3,376.070
LA33	30×10	1,497	1,524.484	27.484	3,455.715
LA34	30×10	1,535	1,558.849	23.849	3,480.766
LA35	30×10	1,549	1,571.026	22.026	3,364.211
LA36	15×10	948	826.283	-121.717	3,475.359
LA37	15×10	986	881.144	-104.856	3,497.715
LA38	15×10	943	778.626	-164.374	3,291.581
LA39	15×10	922	794.206	-127.794	3,278.251
LA40	15×10	955	805.869	-149.131	3,342.038

Tabla 5.24: vdata: Comparativa FJSSP y FJSSP-LS usando TS

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA03	477	476.966	0.01
LA04	502	501.721	0.06
LA05	457	456.846	0.03
LA12	936	935.625	0.04
LA13	1,038	1,037.683	0.03
LA14	1,070	1,069.597	0.04
LA16	717	539.867	24.70
LA17	646	470.228	27.21
LA18	663	522.731	21.16
LA19	617	538.365	12.74
LA20	756	549.390	27.33
LA21	804	802.998	0.12
LA36	948	826.283	12.84
LA37	986	881.144	10.63
LA38	943	778.626	17.43
LA39	922	794.206	13.86
LA40	955	805.869	15.62

Tabla 5.25: vdata: Porcentaje de mejora

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA01	570	570.165	0.03
LA02	529	529.223	0.04
LA06	799	799.240	0.03
LA07	749	749.372	0.05
LA08	765	765.284	0.04
LA09	853	853.266	0.03
LA10	804	804.576	0.07
LA11	1,071	1,071.234	0.02
LA15	1,089	1,089.937	0.09
LA22	738	738.617	0.08
LA23	813	817.157	0.51
LA24	777	777.589	0.08
LA25	754	754.697	0.09
LA26	1,053	1,057.685	0.44
LA27	1,085	1,090.863	0.54
LA28	1,070	1,074.326	0.40
LA29	994	1,001.774	0.78
LA30	1,069	1,075.705	0.63
LA31	1,520	1,531.406	0.75
LA32	1,658	1,679.496	1.30
LA33	1,497	1,524.484	1.84
LA34	1,535	1,558.849	1.55
LA35	1,549	1,571.026	1.42

Tabla 5.26: vdata: Porcentaje de pérdida

Instancia	FJSSP	FJSSP-LS	DIFERENCIA %
LA29	994	995.347961	0.14
LA30	1,069	1,070.66	0.15
LA31	1,520	1,520.93	0.06
LA32	1,658	1,658.81	0.05
LA33	1,497	1,499.26	0.15
LA34	1,535	1,536.21	0.08
LA35	1,549	1,550.77	0.11

Tabla 5.27: vdata: instancias LA29 a LA35 tras diez horas de ejecución

Capítulo 6

Conclusiones y trabajos futuros

En la literatura, se puede encontrar una gran cantidad de trabajos que reportan resultados relacionados con JSSP. Sin embargo, al buscar información sobre algunas de sus variantes, en este caso mediante la flexibilidad y la división de lotes, se descubre que existen pocas publicaciones relacionadas con estos temas, a pesar de que existen varias aplicaciones prácticas en la industria. Por lo anterior, se consideró importante abordar el problema del FJSSP-LS, que se caracteriza por un procesamiento flexible y la división de los trabajos en sublotes y permite tratarlos de manera independiente para minimizar los tiempos muertos y disminuir la amplitud de proceso.

Primero, se buscó la forma de generar un modelo de optimización para dicho problema, así que se realizó una revisión sobre los diferentes modelos existentes para el JSSP, FJSSP y FJSSP-LS. Se encontró que para estos problemas, los modelos disyuntivos tenían un mejor desempeño, ya que permiten resolver un mayor número de instancias en un tiempo establecido [43], [7]. Por lo tanto, se desarrolló finalmente un modelo de optimización disyuntivo que permite minimizar la amplitud de proceso mediante la asignación apropiada de trabajos a las máquinas disponibles y mediante la división de los trabajos en sublotes de tamaño adecuado. Se debe destacar, que hasta donde se tiene conocimiento, este modelo no ha sido reportado en la literatura, por lo cual constituye una contribución importante al estado del arte.

Debido a que el FJSSP-LS es un problema relativamente nuevo, no existen instancias reportadas en la literatura, por lo cual se tomaron como referencia las instancias de Hurink [1] y se modificaron para que algunos trabajos tuvieran la posibilidad de ser divididos en sublotes.

Posteriormente, el modelo fue implementado en el solver Gurobi y, para validar su correcto planteamiento, se empleó para resolver las instancias de Hurink. En unas pruebas preliminares, el modelo fue adaptado para no permitir la división de lotes. Como los resultados obtenidos coincidieron con los

reportados en la literatura, se procedió a resolver la instancias creadas para el FJSSP-LS. Al utilizar Gurobi para resolver estas instancias con el modelo propuesto, se observó que tiene dificultades para encontrar una solución de calidad, cuando tiene que determinar el tamaño de los sublotes. De hecho, fue interesante notar que al añadir como restricción que los sublotes fueran del mismo tamaño, Gurobi reducía su tiempo de ejecución considerablemente y encontraba las soluciones óptimas, demostrando la reducción de la dificultad del problema con esta hipótesis simplificadora. Otro punto importante a observar es que la división de ciertos trabajos tiene un mayor impacto sobre la amplitud de proceso, de esta manera se podría enfocar en los trabajos de mayor impacto para obtener soluciones de calidad en un tiempo mucho menor.

Por otro lado, en la revisión del estado del arte sobre problemas de programación de tareas, se encontró que para instancias de tamaño mediano o grande se ha propuesto el uso de diferentes técnicas heurísticas. Sin embargo, Búsqueda Tabú destacaba considerablemente, tanto por ser una de las estrategias más reportada, como por los buenos resultados obtenidos sobre JSSP, FJSSP y JSSP-LS. Por lo anterior, se consideró que TS constituía una técnica prometedora para el FJSSP-LS, así que se desarrolló un algoritmo basado en TS, el cual utiliza tres tipos de vecindades. Primeramente se utilizó una vecindad probada en la literatura del JSSP para la asignación de secuencias, el cual se centra en el movimiento de operaciones críticas. La segunda vecindad consiste en seleccionar las operaciones críticas y generar los vecinos programando cada operación en cada máquina posible de su estación de trabajo. Finalmente, el tercero mueve un porcentaje de piezas entre sublotes del mismo trabajo. El algoritmo obtenido se utilizó para generar soluciones de las instancias propuestas, limitando su tiempo de ejecución a una hora.

Lamentablemente, no existen resultados reportados para este problema, por lo cual no fue posible realizar comparaciones que permitieran evaluar el desempeño del algoritmo propuesto. Así que se optó por comparar la calidad de las soluciones obtenidas contra los mejores valores reportados en la literatura para el FJSSP. Como resultado de estas comparaciones, el algoritmo propuesto tuvo un buen desempeño, ya que fue capaz de obtener mejores resultados que Gurobi. En algunas instancias se observaron disminuciones importantes en la amplitud de proceso, en comparación con las reportadas para el FJSSP, y aún en las instancias más difíciles no presentó diferencias importantes. Los últimos experimentos computacionales consistieron en resolver las instancias de FJSSP-LS con Gurobi y TS. De manera muy clara, TS determina, para todas las instancias, soluciones de mejor calidad que las cotas superiores producidas por Gurobi, incapaz de garantizar la optimalidad de las soluciones encontradas. Por otro lado, los tiempos de cómputo repor-

tados son también muy a favor de TS. Además, se observó que en algunas instancias, TS podía mejorar la calidad de sus soluciones si se le permitía ejecutarse por más tiempo.

Otro punto importante es que se esperaría que, al dividir los lotes, se podrían disminuir los tiempos muertos y así reducir la amplitud de proceso. Sin embargo, de acuerdo a los resultados obtenidos, la división de lotes no necesariamente resulta benéfica en este sentido. De hecho, se notó que en el banco vdata, el que presenta una mayor flexibilidad en las operaciones, fue en el que menos instancias se mejoraron. Mientras que el banco donde más instancias fueron mejoradas fue el edata, el que presenta una menor flexibilidad de los tres bancos empleados.

Esta trabajo a su vez, se presenta como una alternativa a explorar para empresas que pueden permitirse la división de lotes y aún no lo están haciendo, o que lo hacen de manera arbitraria y buscan mejorar su estrategia. Asimismo, constituye un tema con una amplia área de desarrollo para aquellos interesados en investigación relacionada con problemas de programación de tareas, métodos heurísticos o técnicas exactas. Entre las diferentes perspectivas para trabajo futuro, se encuentran el análisis de tiempos muertos, uso de nuevas vecindades, diferentes codificaciones y técnicas heurísticas. Como extensión a este problema se puede considerar el agregar tiempos de preparación en las máquinas y/o fechas de entrega para los trabajos, por mencionar algunos.

Cabe mencionar que los primeros avances del presente proyecto de investigación fueron presentados en un congreso nacional [91] y en un congreso internacional [92]. Además, se proyecta que los últimos resultados obtenidos y unos estudios adicionales, aún en desarrollo, constituirán el objeto de al menos otra publicación.

Apéndice A

Modelo de Manne

Como ya fue mencionado, en el trabajo reportado por Wen y Christopher se establece que el modelo de precedencias de Manne [4], es el que presenta un mejor desempeño, ya que permite resolver un mayor número de instancias en la reducción de la amplitud de trabajo, dado un tiempo establecido, mediante el uso de CPLEX, Gurobi y SCIP [43].

En esta sección se presenta el modelo disyuntivo de Manne para el JSSP propuesto en 1960 [4], el cual busca minimizar la amplitud de proceso. Las variables a utilizar se listan a continuación.

O_{ij} Denota la operación i del trabajo j

Conjuntos/Indices

J/j Lotes o trabajos.
I/i Operaciones.
K/k Máquinas.
I_j Operaciones demandadas por el lote j.

Parámetros

$p_{i,j}$ Tiempo de procesamiento de O_{ij}.
M Número entero suficientemente grande.

Variables

$Cmax$ Amplitud de proceso.
$S_{i,j}$ Tiempo de inicio de O_{ij}.
$Z_{jj'k}$ Variable binaria que asigna 1 si el trabajo j precede a j' en la máquina k.

En el modelo original de Manne se busca minimizar la amplitud de proceso

por lo cual se usa la siguiente función objetivo A.1.

$$Min\ Cmax \tag{A.1}$$

Para garantizar que la amplitud de proceso representa el tiempo empleado para completar todos los trabajos se establece la siguiente familia de restricciones.

$$Cmax \geq S_{ij} + p_{ij}; \qquad \forall j, i = I_j \tag{A.2}$$

Para garantizar que la operación de un trabajo no inicie hasta que la anterior haya terminado se utiliza la siguiente familia de restricciones.

$$S_{ij} \geq S_{i-1j} + p_{i-1j}; \qquad \forall j \in J, \forall i \geq 2 \tag{A.3}$$

Las Ecuaciones A.4 y A.5 son complementarias, denotan la secuencia de trabajos asignados en la misma máquina. $Z_{jj'k}$ Es igual a uno si el trabajo j precede a el trabajo j' en la máquina k, cero de lo contrario. El trabajo j no se coloca necesariamente inmediatamente antes del trabajo j' cuando $Z_{jj'k} = 1$.

$$S_{jk} \geq S_{j'k} + p_{j'k} - (Z_{jj'k}) * M; \qquad \forall j, j' \in J, \forall j < j', \forall k \in K \tag{A.4}$$

$$S_{j'k} \geq S_{jk} + p_{jk} - (1 - Z_{jj'k}) * M; \qquad \forall j, j' \in J, \forall j < j', \forall k \in K \tag{A.5}$$

La Ecuación A.6 establece que los tiempos de inicio de todas las operaciones debe ser mayor a cero y A.7 establece $Z_{jj'k}$ como una variable binaria.

$$S_{ij} \geq 0; \qquad \forall i, j \tag{A.6}$$

$$Z_{jj'k} \in \{0, 1\}; \qquad \forall j, j' \in J, \forall k \in K \tag{A.7}$$

Apéndice B

Modelo de Özgüven

Özgüven presenta un modelo disyuntivo en el 2010 para FJSSP, el cual busca minimizar la amplitud de proceso. Posteriormente Yunus implementa diversos modelos en CPLEX siendo la implementación del modelo de Özgüven el que logra alcanzar el óptimo en más instancias, en un tiempo establecido. Las variables a utilizar se presentan a continuación.

O_{ij}	Denota la operación i del trabajo$j$.
Variables	
$Cmax$	Amplitud de proceso.
$V_{k,i,j}$	Variable binaria que asume el valor 1 cuando la operación i del lote j es ejecutada en la máquina k y 0 en otro caso.
$S_{i,j,k}$	Tiempo de inicio de O_{ij} en la máquina k.
$C_{i,j,k}$	Tiempo de finalización de O_{ij} en la máquina k.
$Z_{iji'j'k}$	Variable binaria que asigna 1 si O_{ij} precede a $O_{i'j'}$ en la máquina k.
Conjuntos/Indices	
J/j	Lotes (denominados jobs o trabajos).
I/i	Operaciones.
K/k	Máquinas.
I_j	Operaciones demandadas por el lote j.
$K_{i,j}$	Máquinas con capacidad de procesar O_{ij}.
Parámetros	
$p_{i,j,k}$	Tiempo de procesamiento de O_{ij} en la máquina k.
M	Número entero suficientemente grande.

En el modelo de Özgüven la función objetivo busca minimizar la amplitud de proceso.

$$Min\ Cmax \tag{B.1}$$

La familia de Ecuaciones B.2 establece que los tiempos de inicio y final de todas las operaciones deben ser mayor o igual a cero.

$$S_{ijk} \geq 0, C_{ijk} \geq 0; \qquad \forall i, j, k \tag{B.2}$$

La familia de Ecuaciones B.3 establece que los tiempos de finalización de todos los trabajos debe ser mayor o igual a cero.

$$C_j \geq 0; \qquad \forall j \tag{B.3}$$

La Ecuación B.4 establece V_{ijk} como una variable binaria, que es igual a uno cuando O_{ij} es procesada en la máquina k, 0 en otro caso.

$$V_{ijk} \in \{0, 1\}; \qquad \forall i, j, k \tag{B.4}$$

La Ecuación B.5 establece $Z_{iji'j'k}$ como una variable binaria, que es igual a 1 si O_{ij} precede a $O_{i'j'}$ en la máquina k, 0 en otro caso. La operación ij no necesariamente se coloca de manera inmediata antes del trabajo $i'j'$ cuando $Z_{iji'j'k} = 1$.

$$Z_{iji'j'k} \in \{0, 1\}; \qquad \forall i \leq i', \forall j, j', \forall k \in K_{ij} \cap K_{i'j'} \tag{B.5}$$

La Ecuación B.6 establece que cada operación de cada trabajo solo debe ser ejecutada en una única máquina del conjunto de máquinas capaces de procesarla.

$$\sum_{k \in K_{ij}} V_{ijk} = 1; \qquad \forall i, j \tag{B.6}$$

La familia de Ecuaciones B.7 y B.8 son complementarias y sirven para establecer el tiempo de finalización de O_{ij} en la máquina k.

$$S_{ijk} + C_{ijk} \leq V_{ijk} * M; \qquad \forall i, j, \forall k \in K_{ij} \tag{B.7}$$

$$C_{ijk} \geq S_{ijk} + p_{ijk} - (1 - V_{ijk}) * M; \forall i, j, \forall k \in K_{ij} \tag{B.8}$$

La Ecuación B.9 garantiza que la operación de un trabajo no inicie hasta que la anterior haya terminado.

$$\sum_{k \in K_{ij}} S_{ijk} \geq \sum_{k \in K_{ij}} C_{i-1jk}; \qquad \forall j, \forall i = 2, ..., I_j \tag{B.9}$$

La Ecuación B.10 establece un tiempo de terminación de cada trabajo.

$$C_j \geq \sum_{k \in K_{ij}} C_{ijk}; \qquad \forall j, i = I_j \tag{B.10}$$

Para garantizar que la amplitud de proceso representa el tiempo empleado para completar los trabajos se establece la Ecuación B.11.

$$Cmax \geq C_j; \qquad \forall j \tag{B.11}$$

Las Ecuaciones B.12 y B.13 son complementarias, y denotan la secuencia de operaciones asignada en la misma máquina. $Z_{iji'j'k}$ es igual a uno si O_{ij} precede a $O_{i'j'}$, O_{ij} no se coloca necesariamente inmediatamente antes de $O_{i'j'}$ cuando $Z_{iji'j'k} = 1$.

$$S_{i'j'k} \geq C_{ijk} - (1 - Z_{iji'j'k}) * M; \forall j \leq j', \forall i, i', \forall k \in K_{ij} \cap K_{i'j'} \tag{B.12}$$

$$S_{ijk} \geq C_{i'j'k} - Z_{iji'j'k} * M; \forall j \leq j', \forall i, i', \forall k \in K_{ij} \cap K_{i'j'} \tag{B.13}$$

Apéndice C

Modelo de Novas

O_{ij} Denota la operación i del trabajo$j$

Conjuntos/Indices

J/j	Lotes (denominados jobs o trabajos)
I/i	Operaciones
K/k	Máquinas
I_j	Operaciones demandadas por el lote j
$K_{i,j}$	Máquinas con capacidad de procesar la operación i sobre el lote j
S_j	Posibles sublotes para el lote j. Su cardinalidad representa el número máximo de sublotes en que puede dividirse j.

Parámetros

$pt_{k,i,j}$	Tiempo de procesamiento de una unidad del lote j en la operación i en la máquina k
d_j	Cantidad de unidades o partes a elaborar demandadas por j
dd_j	Fecha de entrega o "duedate" del lote j
M	Número entero suficientemente grande

Variables

$Cmax$	Amplitud de proceso.
T_j	Tardanza del lote j respecto a su fecha de entrega.
$W_{k,s,i,j}$	Variable binaria que asume el valor 1 cuando la operación i del lote j es ejecutada mediante el sublote$s$en la máquina k, y 0 de otra manera.
$X_{k,i,j}$	Variable binaria que asume el valor 1 cuando la operación i del lote j es ejecutada en la máquina k, independientemente del sublote sobre el que se ejecuta i; y 0 de otra manera.
$U_{s,i,j}$	Tamaño del sublote s sobre el que se ejecuta la operación i del lote j.
$Cs_{s,i,j}$	Tiempo de finalización de la operación i sobre el sublote s del lote j.
$Ss_{s,i,j}$	Tiempo de inicio de la operación i sobre el sublote s del lote j.
$C_{i,j}$	Tiempo de finalización de la operación i sobre todo el lote j.
$S_{i,j}$	Tiempo de inicio de la operación i sobre todo el lote j.

La expresión C.1, se asigna una operación i de un dado lote j, a una única máquina k entre aquellas que cuentan con capacidad para ejecutar la actividad i de dicho j. Por medio de la expresión C.2, se asignan los sublotes s de cada lote j, en cada operación i, a una única máquina k. La expresión C.3 es utilizada para vincular las variables empleadas en C.1 y C.2, asegurando que si una operación i está asignada a una máquina k, entonces todos los sublotes s del lote j están asignados a la misma máquina k.

$$\sum_{k \in K_{ij}} X_{kij} = 1; \qquad \forall i \in I_j, \forall j \in J \tag{C.1}$$

$$\sum_{k \in K_{ij}} W_{ksij} = 1; \qquad \forall i \in I_j, \forall s \in S_j, \forall j \in J \tag{C.2}$$

$$W_{ksij} = X_{kij}; \qquad \forall j \in J, \forall i \in I_j, \forall s \in S_j, \forall k \in K_{i,j} \tag{C.3}$$

El siguiente par de desigualdades tienen por objeto establecer los tiempos de procesamiento de cada sublote s del lote j cuando sobre éste se ejecuta la operación i en la máquina k.

$$Cs_{sij} \geq Ss_{sij} + pt_{kij} * U_{sij} - (1 - W_{ksij}) * M; \tag{C.4}$$

$$\forall s \in S_j, \forall i \in I_j, \forall k \in K_{i,j}, \forall j \in J$$

$$Ss_{sij} \leq Cs_{sij} + W_{ksij} * M; \qquad \forall s \in S_j, \forall i \in I_j, \forall k \in K_{i,j}, \forall j \in J \quad \text{(C.5)}$$

La expresión C.6 asegura la precedencia entre operaciones sucesivas, i e i', demandadas por un mismo sublote s del lote j.

$$Ss_{si'j} \geq Cs_{sij}; \qquad \forall s \in S_j, \forall i, i' \in I_j | i < last(I_j) \wedge i' = next(i), \forall j \in J \quad \text{(C.6)}$$

También se hace necesario asegurar la precedencia entre sublotes consecutivos, s y s', pertenecientes al conjunto de sublotes de j y sobre los que se ejecuta una misma operación i, lo cual queda establecido por la expresión C.7.

$$Ss_{s'ij} \geq Cs_{sij}; \qquad \forall s, s' \in S_j | s < last(S_j) \wedge s' = next(s), \forall i \in I_j, \forall j \in J \quad \text{(C.7)}$$

Las expresiones C.8 y C.9 tienen por fin sincronizar el inicio y fin de cada operación i de j, con el inicio de la operación i sobre el primer sublote s de j y el fin de la operación i sobre el último sublote s de j, respectivamente.

$$S_{ij} = Ss_{sij}; \forall j \in J, \forall i \in I_j, \forall s \in S_j | s = first(S_j) \quad \text{(C.8)}$$

$$C_{ij} = Cs_{sij}; \forall j \in J, \forall i \in I_j, \forall s \in S_j | s = last(S_j) \quad \text{(C.9)}$$

las operaciones de los trabajos, sin tener en cuenta los sublotes, ya que no hay sublotes de diferentes lotes intercalados en una misma secuencia de operaciones.

La sucesión de las operaciones sobre un dado equipo k, se logra mediante las expresiones lógicas C.10 y C.11.El uso de las mismas asegura que no exista superposición entre las distintas actividades de máquinado asignadas a un mismo equipo.

$$X_{kij} = 1 \wedge X_{kij'} = 1 \rightarrow S_{ij'} \geq C_{ij} \vee S_{ij} \geq C_{ij'}; \quad \text{(C.10)}$$

$$\forall i \in (I_j \cap I_{j'}), \forall k \in (K_{ij} \cap K_{ij'}), \forall j, j' \in J, j \neq j'$$

$$X_{kij} = 1 \wedge X_{ki'j} = 1 \rightarrow S_{i'j} \geq C_{ij}; \quad \text{(C.11)}$$

$$\forall j \in J, \forall k \in (K_{ij} \cap K_{i'j}), \forall i, i' \in I_j | i < last(I_j) \wedge i' = next(i)$$

Las expresiones C.12 y C.13 permiten definir el tamaño de los sublotes para cada lote.

$$U_{sij} \geq 0; \qquad \forall s \in S_j, \forall i \in I_j, \forall j \in J \tag{C.12}$$

$$\sum_{s \in S_j} U_{sij} \geq d_j; \qquad \forall i \in I_j, \forall j \in J \tag{C.13}$$

La siguiente expresión establece sublotes consistentes; es decir, cada sublote s de j, conserva su tamaño a medida que atraviesa todas las operaciones que requiera.

$$U_{sij} = U_{si'j}; \forall s \in S_j, \forall j \in J, \forall i, i' \in I_j | i < last(I_j) \wedge i' = next(i) \tag{C.14}$$

Se proponen un par de expresiones auxiliares, C.15 y C.16, que permiten al modelo ordenar los sublotes de cada lote de acuerdo a su tamaño, de manera creciente o decreciente, respectivamente.

$$U_{sij} \leq U_{s'ij}; \forall i \in I_j, \forall j \in J, \forall s, s' \in S_j | s < last(S_j) \wedge s' = next(s) \tag{C.15}$$

$$U_{sij} \geq U_{s'ij}; \forall i \in I_j, \forall j \in J, \forall s, s' \in S_j | s < last(S_j) \wedge s' = next(s) \tag{C.16}$$

Como función objetivo, la propuesta utiliza alternativamente tres medidas de desempeño. Por un lado, al emplear como función la minimización del valor de la variable Cmax C.17, se busca optimizar el tiempo de finalización de todas las operaciones de todos los lotes. En este caso, el modelo debe considerar la restricción C.18. Por otra parte, se plantea minimizar la función objetivo expresada en C.19. La misma se define como la suma de los tiempos de finalización de la última operación sobre todos los sublotes, de todos los lotes ("Sublots Total Completion Time", STCT). Por último, si se considera la minimización de la tardanza de los lotes, la medida de performance a optimizar es la expresión C.20, mientras que las desigualdades C.21 y C.22 deben incluirse en el modelo.

$$minCmax \tag{C.17}$$

$$Cmax \geq C_{sij}; \forall s \in S_j, \forall i \in I_j, \forall j \in J \tag{C.18}$$

$$min \sum_{\forall s \in S_j} \sum_{\forall i \in I_j | i = last(I_j)} \sum_{\forall j \in J} Cs_{sij} \tag{C.19}$$

$$min \sum_{\forall j \in J} T_j \tag{C.20}$$

$$T_j \geq C_{ij} - dd_j; \forall j \in J, \forall i \in I_j | i = last(I_j) \tag{C.21}$$

$$T_j \geq; \forall j \in J \tag{C.22}$$

Apéndice D

Instancias

Instancia	Demanda/ máximo de sublotes
LA01	1800 6 1300 6 1500 2 1800 2 1000 2 2000 2 2000 3 1300 2 1300 2 1400 4
LA02	1700 4 1100 2 1100 6 1300 2 1400 6 1900 5 1100 4 1900 5 1900 5 1600 6
LA03	1800 6 1900 6 1600 4 1900 6 1500 5 1900 5 1700 4 1500 5 1000 4 1600 3
LA04	1300 2 1300 4 1900 4 1200 3 1700 4 1100 3 1600 3 1200 5 1700 4 1700 4
LA05	2000 4 2000 4 1800 3 1000 5 1500 4 1100 3 1400 3 1500 4 1400 4 1500 6

Tabla D.1: Instancias de 10 trabajos y 5 máquinas

Instancia	Demanda/ máximo de sublotes
LA06	1400 4 1400 6 1000 6 1500 2 1200 5 1700 2 1100 6 1900 5 1700 5 1900 6 1100 4 1400 4 1700 4 1300 3 1200 2
LA07	1800 2 1200 4 1300 5 1100 3 2000 3 1400 5 2000 2 1200 5 1100 4 1800 3 1000 2 1600 5 1000 3 1400 4 1900 5
LA08	1600 5 1200 2 1700 4 1100 2 1100 5 1500 2 1700 2 1300 6 1200 5 1200 4 1200 2 1100 2 1200 6 2000 2 1200 6
LA09	1800 2 2000 2 2000 5 1000 6 1900 6 1700 6 1800 3 1300 2 1300 5 1300 5 1300 3 1700 2 1900 6 1200 2 1100 5
LA10	1500 5 2000 5 2000 3 1800 5 1600 5 1100 3 1800 3 2000 5 1000 4 1800 5 1600 3 1100 6 1600 5 1000 6 1400 4

Tabla D.2: Instancias de 15 trabajos y 5 máquinas

Instancia	Demanda/ máximo de sublotes
LA11	1200 3 2000 5 1900 6 1500 6 1500 3 1100 2 1400 3 1900 3 1900 6 1300 3 1800 4 1700 4 1000 5 1400 3 1900 6 1100 2 1900 4 1000 2 1700 6 1900 4
LA12	1700 6 2000 4 1700 5 1700 4 1900 2 1200 4 1600 4 1300 4 1400 4 1300 6 1600 2 1600 5 1200 4 1900 3 1700 3 1000 6 1800 4 1000 2 1800 3 1900 2
LA13	1200 6 1700 6 2000 5 1500 3 1200 5 1500 4 1300 2 1000 3 1300 3 1900 3 1200 2 1400 5 1600 4 1400 3 1900 4 1900 4 2000 5 1200 3 1700 4 1600 2
LA14	1800 4 1600 4 1700 2 1300 2 1300 3 1400 2 1600 6 1900 2 1300 2 1900 6 1000 6 1400 2 2000 4 1800 2 1500 6 1900 4 1900 5 1500 2 1600 4 1900 5
LA15	1100 6 1000 3 1300 4 1400 2 1300 6 1200 4 1000 3 1600 3 1900 3 1700 6 1700 2 1900 4 2000 6 1000 5 1900 4 1200 5 1500 5 1500 6 1700 5 1800 4

Tabla D.3: Instancias de 20 trabajos y 5 máquinas

Instancia	Demanda/ máximo de sublotes
LA16	1300 6 1100 6 1300 3 1300 2 1800 2 1700 6 1900 3 1900 6 2000 5 1500 4
LA17	1600 4 1900 5 1600 4 1900 4 1800 2 1200 6 1500 3 1900 3 1100 2 1500 6
LA18	1100 5 2000 5 1800 3 1500 4 1700 2 1400 6 1200 2 1000 3 1500 3 1900 3
LA19	1100 4 1600 2 1800 3 1100 4 1300 4 1500 5 1300 4 1800 4 1900 4 1100 5
LA20	2000 4 1600 4 2000 3 2000 2 1300 2 1100 2 1200 3 1200 3 1500 3 1000 5

Tabla D.4: Instancias de 10 trabajos y 10 máquinas

Instancia	Demanda/ máximo de sublotes
LA21	1300 6 2000 3 1500 2 1800 4 1900 3 1300 2 1300 4 1700 4 2000 2 1000 4 1300 3 1100 3 1500 2 1200 4 1300 4
LA22	2000 2 1500 6 1400 3 1500 6 1600 5 1400 5 1900 6 2000 5 1100 6 1100 6 1400 5 1900 5 1400 6 1200 3 1200 4
LA23	2000 5 1500 3 1100 3 1200 6 1300 5 1700 3 1700 4 1200 5 1800 6 1600 6 1200 2 1900 6 2000 6 1200 3 1400 2
LA24	1500 6 1800 4 1400 3 1100 6 1000 2 1700 4 1800 2 1000 3 1200 5 1600 6 2000 3 1500 6 1300 4 1200 4 1700 5
LA25	1100 2 1300 5 1800 4 1700 5 1700 2 1600 3 1900 6 1400 3 1600 2 1100 2 2000 2 1000 3 1900 3 1400 4 1300 3

Tabla D.5: Instancias de 15 trabajos y 10 máquinas

Instancia	Demanda/ máximo de sublotes
LA26	1100 3 1100 5 1900 3 2000 6 1100 5 1300 5 1100 3 1900 2 1800 2 1000 5 1700 6 1300 5 1500 5 1000 2 1200 3 1800 2 1500 4 1400 2 1500 2 1600 4
LA27	1500 2 1200 2 1800 4 1800 4 1000 2 1700 2 1200 2 1400 2 1700 4 1500 5 1400 6 1700 5 1300 6 1600 5 1600 6 1000 3 1200 3 1600 2 1700 4 2000 5
LA28	1900 5 1500 5 1200 2 1900 6 1900 4 1300 5 1200 3 1000 2 1700 4 1500 4 1900 4 2000 6 1400 4 1900 5 1900 4 1800 4 1400 2 2000 6 1400 4 1300 2
LA29	1100 6 1100 5 1000 2 1600 4 1500 5 1600 6 1800 5 1200 5 2000 2 1000 6 1100 5 1300 6 1000 3 1400 4 1200 4 2000 6 1400 4 1100 6 1700 4 1000 5
LA30	1300 3 1000 5 1200 4 1200 6 1300 3 1600 2 1600 3 1200 3 1100 6 1300 2 1800 2 1500 3 1000 6 1800 2 1100 6 1700 6 1100 2 1300 4 1300 5 1300 3

Tabla D.6: Instancias de 20 trabajos y 10 máquinas

Instancia	Demanda/ máximo de sublotes
LA31	1300 2 1700 6 2000 4 1200 3 1900 5 1900 2 1700 4 1800 4 1500 6 2000 4 1300 4 1100 6 1100 2 1000 2 1800 6 1300 2 1400 4 1900 6 2000 3 1300 5 1800 2 1100 2 1400 2 1800 4 1900 5 1400 4 1700 5 1800 4 1500 6 1100 3
LA32	1600 6 1800 5 2000 2 1200 3 1400 4 1200 3 1200 6 1700 4 2000 3 1900 4 1100 4 1600 5 1800 4 1800 4 1200 6 1800 3 1600 3 1100 2 1000 5 1200 5 1100 3 1200 6 1200 3 2000 6 1300 6 1100 3 1500 3 1600 3 1200 5 1900 3
LA33	1700 5 1100 4 1600 4 1000 3 1100 5 1300 3 1600 2 1600 5 1400 3 1700 6 1800 4 1000 6 1800 3 2000 3 1700 6 1300 6 1700 2 2000 2 1700 5 2000 5 1100 3 1400 2 1100 3 1700 4 1800 4 1200 3 1600 6 1800 2 1900 4 1200 3
LA34	1100 5 1800 5 1300 6 1100 6 1100 3 1600 5 1900 5 1600 5 1900 3 1900 5 2000 5 1100 3 1400 2 1400 4 1400 2 1700 4 1600 4 1700 6 1900 4 1000 6 1600 4 1600 2 1700 3 1600 5 1200 3 1800 5 2000 6 1400 4 1000 6 1100 4
LA35	1200 4 1500 2 1100 6 2000 6 1100 5 1700 2 2000 2 1100 2 1900 4 1200 5 1600 3 1800 6 1400 6 2000 2 1300 4 2000 3 1700 3 1300 3 1600 5 1500 2 1500 4 1500 3 1000 4 1700 3 1800 3 1000 3 1400 2 1900 3 1300 2 1900 3

Tabla D.7: Instancias de 30 trabajos y 10 máquinas

Instancia	Demanda/ máximo de sublotes
LA36	1600 5 1900 6 2000 4 1500 4 1500 6 1200 5 1700 5 1800 3 1900 2 1500 2 1600 5 1600 3 1000 6 1900 4 1200 6
LA37	1400 6 1300 5 1600 5 1800 4 1100 4 1100 6 1500 5 1600 5 1000 4 1100 4 1000 4 1000 6 1100 5 2000 2 1600 3
LA38	1700 5 1600 5 1700 4 1200 4 1600 2 1900 4 1200 3 1400 3 2000 6 1800 4 1800 5 1900 2 1900 5 1300 5 1500 2
LA39	1600 4 1200 4 2000 2 2000 5 1900 3 1500 2 1700 5 1200 4 1800 5 1900 3 1800 3 1700 3 1700 6 1800 2 1400 3
LA40	1400 6 1900 4 1200 5 1100 3 1000 5 1000 5 1200 4 1100 2 1900 3 2000 6 1300 3 1800 6 1300 4 1500 4 1300 6

Tabla D.8: Instancias de 15 trabajos y 15 máquinas

Bibliografía

[1] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *OR Spektrum*, vol. 15, no. 4, pp. 205–215, dec 1994.

[2] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. SPRINGER VERLAG GMBH, 2008. [Online]. Available: http://www.ebook.de/de/product/7294937/michael_l_pinedo_scheduling_theory_algorithms_and_systems.html

[3] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, may 1976.

[4] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, apr 1960.

[5] H. M. Wagner, "An integer linear-programming model for machine scheduling," *Naval Research Logistics Quarterly*, vol. 6, no. 2, pp. 131–140, jun 1959.

[6] E. H. Bowman, "The schedule-sequencing problem," *Operations Research*, vol. 7, no. 5, pp. 621–624, oct 1959.

[7] Y. Demir and S. K. İşleyen, "Evaluation of mathematical models for flexible job-shop scheduling problems," *Applied Mathematical Modelling*, vol. 37, no. 3, pp. 977–988, feb 2013.

[8] C. Low, C.-M. Hsu, and K.-I. Huang, "Benefits of lot splitting in job-shop scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 24, no. 9-10, pp. 773–780, nov 2004.

[9] U. Buscher and L. Shen, "An integrated tabu search algorithm for the lot streaming problem in job shops," *European Journal of Operational Research*, vol. 199, no. 2, pp. 385–399, dec 2009.

[10] F. M. Defersha and M. Chen, "Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time," *International Journal of Production Research*, vol. 50, no. 8, pp. 2331–2352, apr 2012.

[11] I. A. Chaudhry and A. A. Khan, "A research survey: review of flexible job shop scheduling techniques," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, aug 2015.

[12] R.-H. Huang, C.-L. Yang, and W.-C. Cheng, "Flexible job shop scheduling with due window—a two-pheromone ant colony approach," *International Journal of Production Economics*, vol. 141, no. 2, pp. 685–697, feb 2013.

[13] L.-N. Xing, Y.-W. Chen, and K.-W. Yang, "Double layer ACO algorithm for the multi-objective FJSSP," *New Generation Computing*, vol. 26, no. 4, pp. 313–327, aug 2008.

[14] M. Yazdani, M. Amiri, and M. Zandieh, "Flexible job-shop scheduling with parallel variable neighborhood search algorithm," *Expert Systems with Applications*, vol. 37, no. 1, pp. 678–687, jan 2010.

[15] É. D. Taillard, "Parallel taboo search techniques for the job shop scheduling problem," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 108–117, may 1994.

[16] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Operations Research*, vol. 40, no. 1, pp. 113–125, feb 1992.

[17] J. Błażewicz, W. Domschke, and E. Pesch, "The job shop scheduling problem: Conventional and new solution techniques," *European Journal of Operational Research*, vol. 93, no. 1, pp. 1–33, aug 1996.

[18] E. Nowicki and C. Smutnicki, "A fast tabo search algorithm for the job shop problem," *Management Science*, vol. 42, no. 6, pp. 797–813, jun 1996.

[19] E. Nowicki and C. Smutnicki, Eds., *An Advanced Tabu Search Algorithm for the Job Shop Problem*, vol. 8, no. 2. Springer Nature, apr 2005.

[20] P. Víctor and Z. Lillo, "Estado del arte del job shop scheduling problem," *Disponible on-line: http://www. alumnos. inf. utfsm. cl/˜ vpena/ramos/ili295/ia-jobshop. pdf*, 2006.

[21] E. Balas, "Machine sequencing via disjunctive graphs: An implicit enumeration algorithm," *Operations Research*, vol. 17, no. 6, pp. 941–957, dec 1969.

[22] C. Zhang, P. Li, Z. Guan, and Y. Rao, "A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem," *Computers & Operations Research*, vol. 34, no. 11, pp. 3229–3242, nov 2007.

[23] J. Huang, G. A. Süer, and S. B. R. Urs, "Genetic algorithm for rotary machine scheduling with dependent processing times," *Journal of Intelligent Manufacturing*, vol. 23, no. 5, pp. 1931–1948, mar 2011.

[24] D. ming Lei, "Minimizing makespan for scheduling stochastic job shop with random breakdown," *Applied Mathematics and Computation*, vol. 218, no. 24, pp. 11 851–11 858, aug 2012.

[25] R. Q. dao-er ji and Y. Wang, "A new hybrid genetic algorithm for job shop scheduling problem," *Computers & Operations Research*, vol. 39, no. 10, pp. 2291–2299, oct 2012.

[26] J. F. Gonçalves and M. G. C. Resende, "An extended akers graphical method with a biased random-key genetic algorithm for job-shop scheduling," *International Transactions in Operational Research*, vol. 21, no. 2, pp. 215–246, oct 2013.

[27] B. Peng, Z. Lü, and T. Cheng, "A tabu search/path relinking algorithm to solve the job shop scheduling problem," *Computers & Operations Research*, vol. 53, pp. 154–164, jan 2015.

[28] Y. Cao, Y. Yang, H. Wang, and L. Yang, "Intelligent job shop scheduling based on MAS and integrated routing wasp algorithm and scheduling wasp algorithm," *Journal of Software*, vol. 4, no. 5, jul 2009.

[29] M. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robotics and Autonomous Systems*, vol. 33, no. 2-3, pp. 169–178, nov 2000.

[30] S. P and S. Sumathi, "Solving fuzzy based job shop scheduling problems using ga and aco," *Citeseer*, 2010.

[31] K.-L. Huang and C.-J. Liao, "Ant colony optimization combined with taboo search for the job shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 4, pp. 1030–1046, apr 2008.

[32] J. Heinonen and F. Pettersson, "Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem," *Applied Mathematics and Computation*, vol. 187, no. 2, pp. 989–998, apr 2007.

[33] A. Rossi and G. Dini, "Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method," *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 5, pp. 503–516, oct 2007.

[34] G. R. Weckman, C. V. Ganduri, and D. A. Koonce, "A neural network job-shop scheduler," *Journal of Intelligent Manufacturing*, vol. 19, no. 2, pp. 191–201, jan 2008.

[35] S. Feng, L. Li, L. Cen, and J. Huang, "Using MLP networks to design a production scheduling system," *Computers & Operations Research*, vol. 30, no. 6, pp. 821–832, may 2003.

[36] T.-L. Lin, S.-J. Horng, T.-W. Kao, Y.-H. Chen, R.-S. Run, R.-J. Chen, J.-L. Lai, and I.-H. Kuo, "An efficient job-shop scheduling algorithm based on particle swarm optimization," *Expert Systems with Applications*, vol. 37, no. 3, pp. 2629–2636, mar 2010.

[37] H. Ge, W. Du, and F. Qian, "A hybrid algorithm based on particle swarm optimization and simulated annealing for job shop scheduling," in *Third International Conference on Natural Computation (ICNC 2007)*. IEEE, 2007.

[38] V. Roshanaei, B. Naderi, F. Jolai, and M. Khalili, "A variable neighborhood search for job shop scheduling with set-up times to minimize makespan," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 654–661, jun 2009.

[39] L. Wang, G. Zhou, Y. Xu, S. Wang, and M. Liu, "An effective artificial bee colony algorithm for the flexible job-shop scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 60, no. 1-4, pp. 303–315, sep 2011.

[40] M. Dell'Amico and M. Trubian, "Applying tabu search to the job-shop scheduling problem," *Annals of Operations Research*, vol. 41, no. 3, pp. 231–252, sep 1993.

[41] E. Balas and A. Vazacopoulos, "Guided local search with shifting bottleneck for job shop scheduling," *Management Science*, vol. 44, no. 2, pp. 262–275, feb 1998.

[42] A. Jamili, "Robust job shop scheduling problem: Mathematical models, exact and heuristic algorithms," *Expert Systems with Applications*, vol. 55, pp. 341–350, aug 2016.

[43] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers & Operations Research*, vol. 73, pp. 165–173, sep 2016.

[44] . T.-G. L. Fisher, H., "Probabilistic learning combinations of local job-shop scheduling rules," *Industrial scheduling*, 1963.

[45] S. Lawrence, "Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement)," *Graduate School of Industrial Administration*, 1984.

[46] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, vol. 34, no. 3, pp. 391–401, mar 1988.

[47] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 149–156, may 1991.

[48] T. Yamada and R. Nakano, "Genetic algorithm applicable to large-scale job-shop problems," *In PPSN*, 1992.

[49] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management Science*, vol. 38, no. 10, pp. 1495–1509, oct 1992.

[50] E. Demirkol, S. Mehta, and R. Uzsoy, "A computational study of shifting bottleneck procedures for shop scheduling problems," *Journal of Heuristics*, vol. 3, no. 2, pp. 111–137, 1997.

[51] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 1, pp. 1–13, feb 2002.

[52] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, dec 1990.

[53] E. Hart, P. Ross, and J. Nelson, "Producing robust schedules via an artificial immune system," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*. IEEE, 1998.

[54] A. M.-. K. J. Akhshabi, M., "Solving flexible job-shop scheduling problem using clonal selection algorithm." *Indian Journal of Science and Technology*, 2011.

[55] M. G. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures: Advances, hybridizations, and applications," in *Handbook of Metaheuristics*. Springer US, 2010, pp. 283–319.

[56] M. Rajkumar, P. Asokan, and V. Vamsikrishna, "A GRASP algorithm for flexible job-shop scheduling with maintenance constraints," *International Journal of Production Research*, vol. 48, no. 22, pp. 6821–6836, jan 2010.

[57] M. Amiri, M. Zandieh, M. Yazdani, and A. Bagheri, "A variable neighbourhood search algorithm for the flexible job-shop scheduling problem," *International Journal of Production Research*, vol. 48, no. 19, pp. 5671–5689, oct 2010.

[58] B. M.-. B. P. Boukef, H., "Flexible job-shop scheduling problems resolution inspired from particle swarm optimization," *Studies in Informatics and Control*, 2008.

[59] A. M.-. K. J. Akhshabi, M., "A particle swarm optimization algorithm for solving flexible job-shop scheduling problem." *Journal of Basic and Applied Scientific Research*, 2011.

[60] C. B. F. . K. M. Mekni, S., "Tribes-pso approach applied to the flexible job shop scheduling problem." *Journal of Modelling & Simulation of Systems*, 2012.

[61] T. Loukil, J. Teghem, and P. Fortemps, "A multi-objective production scheduling case study solved by simulated annealing," *European Journal of Operational Research*, vol. 179, no. 3, pp. 709–722, jun 2007.

[62] P. Fattahi, F. Jolai, and J. Arkat, "Flexible job shop scheduling with overlapping in operations," *Applied Mathematical Modelling*, vol. 33, no. 7, pp. 3076–3087, jul 2009.

[63] M. Zandieh, M. Yazdani, M. Gholami, and M. Mousakhani, "A simulated annealing algorithm for flexible job-shop scheduling problem," *Journal of Applied Sciences*, vol. 9, no. 4, pp. 662–670, apr 2009.

[64] M. Jensen, "Generating robust and flexible job shop schedules using genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 275–288, jun 2003.

[65] T. Kis, "Job-shop scheduling with processing alternatives," *European Journal of Operational Research*, vol. 151, no. 2, pp. 307–332, dec 2003.

[66] P. Fattahi, M. S. Mehrabad, and F. Jolai, "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 18, no. 3, pp. 331–342, jul 2007.

[67] C. Özgüven, L. Özbakır, and Y. Yavuz, "Mathematical models for job-shop scheduling problems with routing and process plan flexibility," *Applied Mathematical Modelling*, vol. 34, no. 6, pp. 1539–1548, jun 2010.

[68] J. Gao, M. Gen, and L. Sun, "Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm," *Journal of Intelligent Manufacturing*, vol. 17, no. 4, pp. 493–507, aug 2006.

[69] K.-H. Kim and P. Egbelu, "Scheduling in a production environment with multiple process plans per job," *International Journal of Production Research*, vol. 37, no. 12, pp. 2725–2753, aug 1999.

[70] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, no. 3, pp. 157–183, sep 1993.

[71] L. M. Gambardella and M. Mastrolilli., "Effective neighborhood functions for the flexible job shop problem." *Journal of scheduling 3.3 3-20.*, 1996.

[72] M. Ennigrou and K. Ghédira., "New local diversification techniques for flexible job shop scheduling problem with a multi-agent approach." *Autonomous Agents and Multi-Agent Systems 17.2 270-287.*, 2008.

[73] G. Vilcot and J.-C. Billaut., "A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem." *International Journal of Production Research 49.23 6963-6980.*, 2011.

[74] S. Jia and Z.-H. Hu, "Path-relinking tabu search for the multi-objective flexible job shop scheduling problem," *Computers & Operations Research*, vol. 47, pp. 11–26, jul 2014.

[75] M. Cheng, N. Mukherjee, and S. Sarin, "A review of lot streaming," *International Journal of Production Research*, vol. 51, no. 23-24, pp. 7023–7046, jun 2013.

[76] S. Dauzère-Pérès and J.-B. Lasserre, "Lot streaming in job-shop scheduling," *Operations Research*, vol. 45, no. 4, pp. 584–595, aug 1997.

[77] U. Buscher and L. She, "An integer programming formulation for the lot streaming problem in a job shop environment with setups," *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2011.

[78] S. Bayat Movahed, "Linear programming assisted genetic algorithm for solving a comprehensive job shop lot streaming problem," The University of Guelph, Tech. Rep., 2014.

[79] J. M. Novas, "Modelo milp para la programación de la producción en ambientes job-shop flexibles con división de lotes." *Iberoamerican Journal of Industrial Engineering*, 2017.

[80] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, jan 1986.

[81] F. Glover and M. Laguna, *Tabu Search*. Springer US, 1997.

[82] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan, "A very fast TS/SA algorithm for the job shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 1, pp. 282–294, jan 2008.

[83] S. B. Akers, "Letter to the editor—a graphical approach to production scheduling problems," *Operations Research*, vol. 4, no. 2, pp. 244–245, apr 1956.

[84] T. C. E. Cheng, B. Peng, and Z. Lü, "A hybrid evolutionary algorithm to solve the job shop scheduling problem," *Annals of Operations Research*, vol. 242, no. 2, pp. 223–237, feb 2013.

[85] J.-Q. Li, Q.-K. Pan, P. N. Suganthan, and T. J. Chua, "A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible

job shop scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 52, no. 5-8, pp. 683–697, jun 2010.

[86] Q. Zhang, H. Manier, and M.-A. Manier, "A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times," *Computers & Operations Research*, vol. 39, no. 7, pp. 1713–1723, jul 2012.

[87] J. J. Palacios, M. A. González, C. R. Vela, I. González-Rodríguez, and J. Puente, "Genetic tabu search for the fuzzy flexible job shop problem," *Computers & Operations Research*, vol. 54, pp. 74–89, feb 2015.

[88] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com

[89] P. M. Pardalos and O. V. Shylo, "An algorithm for the job shop scheduling problem based on global equilibrium search techniques," *Computational Management Science*, vol. 3, no. 4, pp. 331–348, 2006.

[90] X. Li and L. Gao, "An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem," *International Journal of Production Economics*, vol. 174, pp. 93–110, 2016.

[91] M. A. F. Romero, A. Ponsich, and E. A. R. García, "Resolución del problema flexible job shop con división de lotes." in *VI Congreso de la Sociedad Mexicana de Investigación de Operaciones*, 2017.

[92] M. A. F. Romero, A. Ponsich, E. A. R. García, and R. A. M. Gutiérrez, "A tabu search algorithm for the flexible job shop scheduling problems with lot streaming," in *Numerical and Evolutionary Optimization*, 2017.