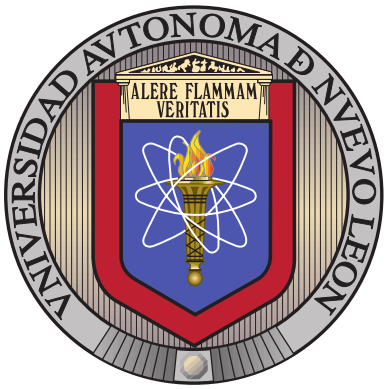
Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado



Secuenciación en máquinas paralelas no relacionadas con tiempos de preparación y tareas de mantenimiento preventivo

por

MC. Oliver Avalos Rosales

en opción al grado de

Doctor en Ingeniería

con especialidad en Ingeniería de Sistemas

San Nicolás de los Garza, Nuevo León septiembre 2014

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado

Secuenciación en máquinas paralelas no relacionadas con tiempos de preparación y tareas de mantenimiento preventivo

por

MC. Oliver Avalos Rosales

en opción al grado de

Doctor en Ingeniería

con especialidad en Ingeniería de Sistemas

San Nicolás de los Garza, Nuevo León septiembre 2014

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis «Secuenciación en máquinas paralelas no relacionadas con tiempos de preparación y tareas de mantenimiento preventivo», realizada por el alumno MC. Oliver Avalos Rosales, con número de matrícula 1506783, sea aceptada para su defensa como opción al grado de Doctor en Ingeniería con especialidad en Ingeniería de Sistemas.

El Comité de Tesis

Dra. Ada Álvarez Socarrás
Directora

Dr. Francisco Angel-Bello Acosta
Codirector

Carlos Andrés Romano
Revisor

Yasmín A. Ríos Solís
Revisora

Iris Abril Martínez Salazar
Revisora

Vo. Bo.

Dr. Simón Martínez Martínez
División de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, septiembre 2014

Dedicatoria

A mis padres, a mi familia y a mi esposa quienes me han acompañado en este camino.

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

A mis directores de tesis la Dra. Ada Álvarez y el Dr. Francisco Angel-Bello, por depositar su confianza en mi para el desarrollo de este proyecto, por apoyarme de manera incondicional durante todo este tiempo en lo acádemico y también en lo personal.

A los miembros de mi comité la Dra. Yasmín y la Dra. Iris por sus valioso análisis y aportaciones que me permitieron enriquecer este documento.

Al Dr. Carlos Andrés por formar parte de mi comité y por recibirme de manera tan amena durante mi estancia en Valencia en la cual me permitió colaborar con él, conocer su lado humano y a su agradable familia.

A mi esposa Yajaira por brindarme ese apoyo tan esencial que me hace avanzar a la velocidad adecuada.

Resumen

MC. Oliver Avalos Rosales.

Candidato para el grado de Doctor en Ingeniería
con especialidad en Ingeniería de Sistemas.

Universidad Autónoma de Nuevo León.
Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio:

Secuenciación en máquinas paralelas no relacionadas con tiempos de preparación y tareas de mantenimiento preventivo

Número de páginas: 82.

Objetivos y método de estudio: El objetivo principal de este trabajo de tesis es ofrecer soluciones con un buen valor del *makespan* a dos problemas de secuenciación con poco esfuerzo computacional. El primer problema de secuenciación consiste en secuenciar tareas en máquinas paralelas no relacionadas con tiempos de preparación dependientes de la máquina y de la secuencia tal que se minimize el *makespan* o tiempo de terminación de la última tarea que se procese. El segundo problema, además de lo anterior considera un esquema de tareas de mantenimiento preventivo de las máquinas, por lo que se tienen periodos de tiempo en los que las máquinas no estan disponibles para realizar el procesamiento de las tareas.

El método de estudio consiste en realizar un análisis de las técnicas de solución en problemas afines y considerar las características que deben ser tomadas en cuenta para desarrollar una metodología de solución eficiente. Como resultado de lo anterior analizamos cada problema en dos pasos. Primero, formulamos un modelo matemático para obtener soluciones óptimas, determinamos el alcance del modelo matemático en cuanto al tamaño de instancias que puede resolver, analizamos las soluciones de la relajación lineal del modelo, el tiempo de cómputo para obtenerlas dichas soluciones y derivamos un conjunto de desigualdades válidas para fortalecer el modelo. Segundo, proponemos algoritmos de solución para los casos en que el modelo es incapaz de obtener soluciones en un tiempo de cómputo razonable para la mayoría de las instancias. En dichos algoritmos se proponen dos nuevas estrategias, una para lidiar con la estructura mín máx de la función objetivo y una más para explotar las relaciones entre los tipos de decisión que surgen en cada problema.

CONTRIBUCIONES Y CONCLUSIONES: La contribución de este proyecto radica en dos aspectos: el primero, la modelación matemática de ambos problemas, la propuesta de una nueva linealización para el *makespan* y la aportación de desigualdades válidas que mejoran la evaluación de la función objetivo en los modelos matemáticos. El segundo aspecto es el diseño e implementación de algoritmos de solución eficientes basados en las estrategias que proponemos y que adaptamos de acuerdo a los requerimientos de cada problema.

En el primer problema se superó el desempeño de las formulaciones previamente publicadas en la literatura científica. Se resolvió a optimalidad instancias hasta seis veces más grandes en comparación con otras formulaciones, usando el mismo tiempo computacional, y se obtuvieron soluciones óptimas en instancias del mismo tamaño hasta cuatro órdenes de magnitud más rápido. Con el algoritmo propuesto se superaron los mejores resultados conocidos de la literatura para un *benchmark* de instancias y se mostró la conveniencia de usar las estrategias propuestas.

En el segundo problema fue la primera vez que se modeló matemáticamente.

Las desigualdades válidas lograron acelerar el proceso de solución usando un optimizador comercial. Sin embargo, solo se lograron resolver a optimalidad instancias de tamaño pequeño. Con el algoritmo se obtuvieron buenos resultados en comparación con los obtenidos con el modelo matemático en instancias pequeñas y medianas.

Firma de la directora: _____

Dra. Ada Álvarez Socarrás

Firma del codirector: _____

Dr. Francisco Angel-Bello Acosta

Capítulo 1

Introducción

La gran diversidad de problemas reales del ámbito científico, de producción de bienes y de distribución de recursos que surgen en la mayoría de las empresas han motivado un gran desarrollo de técnicas y metodologías de solución.

Uno de los problemas con mayor relevancia en optimización combinatoria es el problema del agente viajero, o TSP (Burkard et al., 1998). El TSP tiene una formulación muy sencilla, es extremadamente difícil de resolver desde el punto de vista computacional y conjuntamente con sus variaciones, extensiones y generalizaciones puede describir una gran cantidad de problemáticas en el ámbito de los procesos de producción y de distribución. Encontrar buenas soluciones en tiempos de cómputo razonables para éste tipo de problemas se ha convertido en un reto para un gran número de investigadores.

El TSP se puede formular de la siguiente manera: Un agente vive en una ciudad, debe visitar solamente una vez cada una de las n ciudades donde vende su mercancía y regresar a su ciudad de origen. El objetivo es determinar en qué orden se visitarán las ciudades de forma tal que la distancia recorrida sea la menor posible.

Una aplicación muy importante del TSP está asociada a la programación de tareas en una máquina o procesador cuando existe un tiempo de preparación de la máquina al cambiar de tipo de tarea que está realizando. Por lo general, el tiempo que se invierte para preparar la máquina es improductivo y como lo muestran las recopilaciones sobre los multiples estudios desde mediados de 1960 (Allahverdi et al.,

2008; Allahverdi y Soroush, 2008), es objetivo de las empresas que éste sea el menor posible. Esta problemática se puede modelar mediante un TSP, considerando a las tareas como a las ciudades a visitar y a los tiempos de preparación de la máquina como a las distancias entre las diferentes ciudades .

Este trabajo de tesis está enfocado a desarrollar metodologías de solución eficientes a problemas de secuenciación de tareas, donde hay tiempos de preparación que dependen del orden en el que se procesen las tareas. También se resalta la importancia de incluir un programa de mantenimiento preventivo de las máquinas cuando se lleva a cabo la planeación de la producción, destacando que hay periodos de tiempo en que las máquinas no están disponibles.

1.1 Descripción de la problemática

En muchos problemas de secuenciación se supone que las máquinas están siempre disponibles para su uso. En la práctica, estas se pueden detener por diversas razones: averías o mantenimiento. El mantenimiento y la polítca de mantenimiento juegan un rol crucial para alcanzar la efectividad de los sistemas operacionales a un costo mínimo. La forma tradicional de planear el mantenimiento involucra seleccionar políticas óptimas de estrategias de mantenimiento conocidas tales como mantenimiento preventivo, mantenimiento correctivo, etc. (Ruiz et al., 2007).

En secuenciación, el mantenimiento preventivo forma parte de un área de estudio conocida como *restricciones de disponibilidad*. Dicha área contempla que las máquinas pueden no estar disponibles para realizar su función en ciertos periodos de tiempo. Los problemas de secuenciación con restricciones de disponibilidad han atraído la atención de muchos investigadores y en años recientes ha incrementado el número de publicaciones al respecto (Ma et al., 2010).

La consideración conjunta de tiempos de preparación dependientes y un programa de mantenimiento preventivo aumenta considerablemente la complejidad de los

problemas de secuenciación, esto debido a que se conjugan las estructuras de problemas con disponibilidad de restricciones y problemas con la estructura del TSP, por lo que es natural tratar de resolver versiones más simples del problema para ganar experiencia. Por un lado, la mayoría de los trabajos que han abordado problemas de secuenciación con tiempos de preparación no han considerado tareas de mantenimiento preventivo. Por otro lado, la gran mayoría de quienes han considerado problemas de secuenciación con restricciones de disponibilidad no han incluido tiempos de preparación de la máquina, por lo que sus resultados no siempre son aplicables cuando sí se consideran. Los pocos trabajos que consideran ambas características no han realizado estudios sobre los ambientes de producción de máquinas paralelas, ni sobre un criterio distinto al *makespan*.

Por lo anterior, en el trabajo de tesis abordamos dos problemas. El primer problema consiste en determinar cómo procesar un conjunto de tareas sobre un conjunto de máquinas paralelas no relacionadas con tiempos de preparación dependientes de la secuencia y de la máquina, tal que se minimice el *makespan*. El segundo problema es una versión extendida del primero, porque además de todo lo anterior, las máquinas tienen definido un programa de mantenimiento preventivo periódico.

En términos generales, en ambos problemas se considera que se cuenta con un conjunto de tareas para ser procesadas cada una en solo una máquina, de un conjunto de máquinas. Cada vez que se va a procesar una nueva tarea, es necesario realizar un grupo de operaciones y adecuaciones a la máquina, las cuales dependen de la máquina, de la tarea que se estaba procesando y de la que va a entrar a procesamiento. El procesamiento de una tarea no puede ser interrumpido.

En el segundo problema, además, para cada máquina, cada cierto intervalo de tiempo fijo debe realizarse un mantenimiento. Todas las tareas de mantenimiento tienen duración conocida, la cual es la misma en una máquina, pero varía de una máquina a otra. También hay tiempos de preparación de la máquina, antes y después de un mantenimiento, que dependen de las tareas que se procesen antes o después del mismo.

1.2 Motivación

Diversas razones sugieren incluir programas de mantenimiento preventivo de las máquinas en los sistemas de producción: (1) la mayoría de los sistemas de producción pretenden ser confiables y de alto desempeño; (2) los sistemas de mantenimiento están diseñados para asegurar el buen estado y confiabilidad de las instalaciones de producción; (3) los costos de mantenimiento pueden formar un gran porcentaje del presupuesto de operación; y (4) una planeación adecuada del mantenimiento puede reducir costos y alargar la vida útil de las máquinas (Ruiz et al., 2007). Sin embargo, las políticas de mantenimiento influyen en la disponibilidad de las máquinas: aspecto poco considerado en problemas de secuenciación, específicamente, en los que involucran tiempos de preparación dependientes. Por lo que, se deben analizar los sistemas de producción considerando de manera simultánea dos aspectos: tiempos de preparación dependientes de la secuencia y restricciones de disponibilidad ocasionados por mantenimiento preventivo.

Dada la complejidad de los problemas que consideran solo uno de los dos aspectos, se han desarrollado algoritmos de aproximación, procedimientos heurísticos y procedimientos metaheurísticos para resolverlos; poco esfuerzo se ha realizado por aumentar el tamaño de instancias que podrían ser resueltas a optimalidad. Por otro lado, cuando se integran los problemas de secuenciación de tareas con costos de preparación dependientes y los de restricciones de disponibilidad se obtienen problemas fuertemente NP-duros (Ángel-Bello et al., 2011a). En la literatura científica pocos trabajos han abordado estos problemas, en particular el caso de máquinas paralelas no ha sido estudiado aún. De aquí, que en este proyecto se pretenda avanzar en el estado del arte con las formulaciones matemáticas del primer y segundo problema para la obtención de soluciones exactas, la derivación de desigualdades válidas para reforzar las formulaciones y el desarrollo de procedimientos metaheurísticos para encontrar soluciones de buena calidad en tiempos de cómputo razonables.

1.3 Objetivos

El objetivo central de este trabajo es avanzar en el estado del arte en problemas de secuenciación que consideran de manera conjunta tiempos de preparación dependientes y programas de mantenimiento de las máquinas, con el objetivo de minimizar el *makespan*.

La parte fundamental de la investigación es determinar las características particulares de ambos problemas que permitan explotar sus estructuras y propiedades para el desarrollo de modelos y técnicas de solución eficientes.

El objetivo general del proyecto de investigación es ofrecer para ambos problemas soluciones de buena calidad, con el menor esfuerzo computacional posible, para instancias de diversos tamaños. Para conseguirlo necesitamos cumplir los siguientes objetivos particulares.

- Estudiar el primer problema para analizar sus formulaciones matemáticas, conocer su alcance y determinar posibles mejoras. Así mismo, estudiar en la literatura científica las metodologías de solución que han mostrado mejor desempeño y las características que las favorecen.
- Desarrollar algoritmos específicos para dar solución al primer problema.
- Diseñar experimentos computacionales que permitan evaluar el desempeño y alcance de las reformulaciones propuestas y de la metodología de solución.

Con base en lo anterior:

- Desarrollar la modelación mátematica del segundo problema, así como llevar a cabo posibles mejoras de la misma mediante el análisis de las soluciones obtenidas de su relajación lineal y la aportación de desigualdades válidas.
- Desarrollar algoritmos específicos para dar solución al segundo problema.

- Diseñar experimentos computacionales que permitan evaluar el desempeño y alcance de la formulación y de la metodología de solución propuesta.

1.4 Contribución científica

La contribución de este proyecto radica en dos aspectos: el primero, la modelación matemática de ambos problemas, la propuesta de una nueva linealización para el *makespan* y la aportación de desigualdades válidas que mejoran la evaluación de la función objetivo. En este aspecto, en el primer problema se supera el desempeño de las formulaciones previamente publicadas en la literatura científica, mientras que en el segundo problema es la primera vez que se modela matemáticamente. El segundo aspecto es el diseño e implementación de algoritmos de solución, que si bien están basados en metaheurísticas conocidas, nosotros incluimos varias características que ayudan a mejorar considerablemente su desempeño. Por ejemplo, una estrategia para tomar ventaja de la relación entre los tipos de decisión que surgen en cada problema, y una estrategia para lidiar con la estrucura de la función objetivo. Cada estrategia es adaptada a los requerimientos de cada problema. En este aspecto, para el primer problema se superan los mejores resultados conocidos de la literatura para un *benchmark* de instancias, mientras que para el segundo muestra buenos resultados en comparación con los obtenidos con el modelo matemático.

Tanto las desigualdades válidas propuestas en las formulaciones matemáticas como las estrategias propuestas en los algoritmos metaheurísticos pueden aplicarse en otros problemas de secuenciación y/o con otras metaheurísticas.

1.5 Estructura del documento

En el capítulo 2 se presenta la notación para describir de manera compacta los problemas de secuenciación. Se mencionan brevemente algunas reglas de prioridad que han sido empleadas para resolver estos problemas. Se estudian trabajos de la

literatura que han abordado algunos de los aspectos del problema. Finalmente, se describen las áreas de oportunidad en las cuales se puede contribuir con el proyecto de investigación.

En el capítulo 3 se estudia el primer problema de secuenciación, el cual no considera mantenimiento en la planeación de la producción. Se muestran tres reformulaciones del problema y se proponen mejoras a éstas para incrementar el tamaño de las instancias que se resuelven a optimalidad. Se describe el algoritmo de solución y las estrategias que lo componen para obtener soluciones de buena calidad para este problema. Se realiza experimentación tanto con el modelo como con el algoritmo para determinar su desempeño y se describen las conclusiones respecto al primer problema.

En el capítulo 4 se estudia el segundo problema de secuenciación, el cual considera conjuntamente la planeación de la producción y un programa de mantenimiento preventivo. Se muestra el modelo matemático desarrollado y desigualdades válidas para mejorar el desempeño de la formulación. Se describen el algoritmo de solución y las estrategias que lo componen. Se realiza experimentación para evaluar el desempeño del algoritmo y se presentan las conclusiones del capítulo.

En el capítulo 5 se presentan las conclusiones generales del proyecto de investigación y las futuras líneas de investigación que se derivan de él.

Capítulo 2

Marco teórico

Los problemas de secuenciación aparecen en una gran variedad de contextos. Éstos siempre involucran realizar un número de actividades que emplean diversos recursos en determinados periodos de tiempo y los recursos son limitados. Las actividades a ser realizadas puedes ser llamados "trabajos", "proyectos" o "asignaciones" y son compuestas de partes elementales llamadas "tareas" u "operaciones" y "fechas". Cada actividad requiere cierta cantidad de recursos específicos, por un tiempo específico, llamado "tiempo de procesamiento". Los recursos también tienen partes elementales, llamadas "máquinas", "células", "transporte" y "fechas" (Pinedo, 2012; Morton, 1993).

Los problemas de secuenciación son comúnmente complicados debido a un gran número de restricciones que relacionan tareas con otras tareas, recursos con tareas o con otros recursos, o cualquiera de los dos con eventos externos al sistema. Por ejemplo, puede no ser posible usar dos recursos simultáneamente durante parte del día o sobre una misma actividad, un recurso puede no estar disponible durante intervalos específicos debido a mantenimiento, etc. Dado que estas complejas interrelaciones pueden hacer muy difíciles los problemas de secuenciación, es natural tratar de resolver versiones más simples del problema para ganar experiencia .

Encontrar un buen objetivo para minimizar o maximizar puede ser difícil para un problema de secuenciación por muchas razones. Primero, objetivos tales como satisfacción del cliente, calidad o rapidez son difíciles de cuantificar. Segundo, cada departamento usualmente trata con tres tipos de objetivos muy diferentes:

- maximizar el rendimiento del departamento,
- satisfacer los deseos del cliente en calidad y rapidez, y
- minimizar los costos actuales.

Por consiguiente, se han utilizado diversos enfoques:

- resolver problemas con un objetivo a la vez,
- resolver el problema multi-objetivo por curvas de intercambio entre objetivos, y
- combinar objetivos al asignar pesos a los deseos de los clientes y al rendimiento del departamento.

En este capítulo presentamos la notación estándar que se usa en problemas de secuenciación para referirse a ellos de forma concisa. Luego, presentamos algunas metodologías de solución empleadas en problemas de secuenciación, resumimos algunos de los problemas con una sola máquina y máquinas paralelas. Finalmente, señalamos las áreas de oportunidad para desarrollar el proyecto de investigacción y señalamos la opción que nosotros elegimos.

La clasificación presentada en la siguiente sección está basada en un esquema de clasificación definido en el trabajo de Graham et al. (1977) y que ha sido altamente usado en la literatura (Brucker, 2007; Ma et al., 2010; Lawler et al., 1993).

2.1 Clasificación de problemas de secuenciación

De manera general, supongamos que m máquinas $M_i (i = 1, \ldots, m)$ tienen que procesar n tareas $J_j (j = 1, \ldots, n)$. Un programa de secuenciación es, para cada tarea, la asignación de uno o más intervalos de tiempo para una o más máquinas. Los programas pueden ser representados por un diagrama de Gantt. Los diagramas

de Gantt pueden ser orientados a las máquinas u orientados a las tareas. El correspondiente problema de secuenciación, consistirá en encontrar un programa que satisfaga ciertas restricciones.

Las clases de problemas de secuenciación son clasificados en base a la notación estándar de tres campos $\alpha|\beta|\gamma$, donde α especifica el ambiente máquina, β expone las características de las tareas y γ denota el criterio de optimalidad. Esta notación fue primeramente introducida por Graham et al. (1977) y ha evolucionado con la consideración de nuevos aspectos.

Ambiente máquina El primer campo $\alpha \in \alpha_1\alpha_2\alpha_3$ caracterizado por tres parámetros describe el ambiente máquina.

Los posible valores de α_1 son $\alpha_1 \in \{\emptyset, P, Q, R, F, J, O, FF\}$. Si $\alpha_1 \in \{P, Q, R\}$, entonces tenemos el caso de máquinas paralelas, donde cada tarea debe ser procesada en alguna de las máquinas $M_1, \ldots, M_m$. Si $\alpha_1 = P$ entonces tenemos máquinas paralelas idénticas, tal que para los tiempos de procesamiento p_{ij} de la tarea J_j en M_i tenemos que $p_{ij} = p_j$ para todas las máquinas. Si $\alpha_1 = Q$ tenemos máquinas paralelas uniformes, es decir, $p_{ij} = p_j/s_i$ donde s_i es la velocidad de la máquina M_i. Finalmente, si $\alpha_1 = R$ tenemos máquinas paralelas no relacionadas, es decir, $p_{ij} = p_j/s_{ij}$ para una velocidad s_{ij} de M_i dependiente de la tarea. Si $\alpha_1 \in \{F, J, O, FF\}$, tenemos un modelo multi-operación, es decir, asociado con cada J_j hay un conjunto de operaciones $O_{j1}, \ldots, O_{jn_j}$. Si $\alpha_1 = J$ tenemos el caso *job shop* con relaciones de precedencia de la forma $O_{j1} \to O_{j2} \to O_{j3} \to \ldots \to O_{jn_j}$. Si $\alpha_1 = F$ indica el caso *flow shop*, $\alpha_1 = O$ el caso *open shop* y $\alpha_1 = FF$ el caso *flexible flow shop*.

El parámetro $\alpha_2 \in \{\emptyset, m\}$ indica un número variable o igual a m, respectivamente, de máquinas (para sistemas de máquinas en paralelo) o de etapas (para sistemas de máquinas en línea).

El parámetro $\alpha_3 \in \{\emptyset, h_{il}\}$ describe los intervalos de no disponibilidad (Ma et al., 2010). En esta notación h_{il} describe el número de intervalos de no disponibilidad

y las máquinas en la que aparecen, de tal forma que h_{il} describe un número arbitrario dichos intervalos en cada máquina. Si i es reemplazado por un entero positivo, significa que éstos son sólo en esa máquina; en otro caso éstos son permitidos en todas las máquinas. Si l es remplazado por un entero positivo, este denota el número de intervalos de no disponibilidad en la máquina correspondiente; en otro caso el número es arbitrario. Por ejemplo, h_{1l} representa el problema con un número arbitrario dichos intervalos en M_1, pero ninguno en las otras máquinas; h_{i1} representa un problema con uno solo de estos intervalos en cada máquina; h_{11} representa el problema con un solo intervalo en M_1. En particular, cuando hay solo una máquina, el subíndice i puede ser omitido.

En los dos problemas que abordamos en esta tesis empleamos el parámetro R, que se refiere a máquinas paralelas no relacionadas, y en el segundo problema usaremos el parámetro h_{il} para referirnos a múltiples intervalos de no disponibilidad.

Características de las tareas El segundo campo $\beta \in \{\beta_1, \ldots, \beta_5\}$ describe las características de las tareas o de los recursos. El parámetro $\beta_1 \in \{\emptyset, t - pmtn, pmtn\}$ indica la posibilidad de interrupción, donde $\emptyset$, $t - pmtn$, $pmtn$ denotan, respectivamente, sin interrupción, operaciones interrumpidas e interrupciones arbitrarias. El parámetro $\beta_2 \in \{\emptyset, r_j\}$ se refiere a que cualquier tarea podría procesarse al tiempo cero, o que hay un tiempo r_j a partir del cual puede procesarse una tarea j. El parámetro $\beta_3 \in \{\emptyset, d_j\}$ describe que no se asumen fechas de entrega o bien, que se imponen fechas de entrega sobre un conjunto de tareas. El parámetro $\beta_4 \in \{\emptyset, online\}$ refleja si el problema es *offline* ($\emptyset$), es decir, si toda la información para programar la producción es conocida, u *online*, si hay información que no se conoce. El parámetro $\beta_5 \in \{\emptyset, s_{jk}, s_{ijk}\}$, indica respectivamente que, no hay tiempos de preparación (o éstos son independientes de la secuencia), hay tiempos de preparación que dependen de la secuencia, o hay tiempos de preparación que dependen tanto de la máquina como de la secuencia (para múltiples máquinas). Una notación más extensa para describir tiempos de preparación se encuentra en el trabajo de Allahverdi et al. (2008).

En los dos problemas que abordamos en esta tesis empleamos el parámetro s_{ijk} para indicar que se tienen tiempos de preparación dependientes tanto de la máquina como de la secuencia.

Criterio de optimalidad El tercer campo γ representa la medida de desempeño a ser optimizada. Los criterios considerados en la literatura son medidas relacionadas a los tiempos de terminación de las tareas C_j's y las fechas de entrega d_j's, por ejemplo: C_{max}, el máximo tiempo de terminación de las tareas; C_{min}, el mínimo tiempo de terminación de las tareas; L_{max}, la máxima diferencia entre el tiempo de terminación de las tareas y su fecha de entrega; T_{max}, la máxima tardanza de las tareas, donde $T_j = \text{mín}\{0, C_j - d_j\}$ representa la tardanza de la tarea j; $\sum C_j$, el tiempo de terminación total, es decir, la suma de los tiempos de terminación de las tareas; $\sum w_j C_j$, la suma ponderada de los tiempos de terminación de las tareas; $\sum U_j$, el número de tareas tardías; entre otras.

Hay muchos criterios de desempeño para medir la calidad de un programa de producción. Uno de los más usados es la minimización del máximo tiempo de terminación de la secuencia, conocido como *makespan* (C_{max}). Esta es una medida de desempeño importante que da el tiempo total empleado en procesar todas las tareas en consideración (De y Morton, 1980). El *makespan* es relevante en situaciones en las que simultáneamente se recibe un grupo de tareas requeridas para terminarse tan pronto como sea posible (Allahverdi, 2000). Este tipo de situación es especialmente común en estaciones de servicios, centros de datos, y cómputo en la nube (por ejemplo *Amazon Elastic Compute Cloud*) (Tian et al., 2010). En los dos problemas que abordamos en esta tesis minimizamos C_{max}.

A continuación mencionamos brevemente algunas de las metodologías empleadas para resolver problemas de secuenciación.

Metodologías de solución Entre las metodologiás de solución podemos encontrar: reglas de prioridad, algoritmos de aproximación, heurísticas, metaheurísticas

y algortimos exactos.

Las *reglas de prioridad* son fáciles de implementar y sus complejidades computacionales son bajas, por lo que se usan para tratar de resolver muchos problemas. Las siguientes reglas son las más sobresalientes :

- Regla del tiempo de procesamiento más corto (SPT). Secuencia las tareas en orden no-decreciente de su tiempo de procesamiento p_j. El problema clásico de una máquina y el objetivo de minimizar el total completion time $(1||\sum C_i)$ puede resolverse aplicando la regla SPT .
- Regla del tiempo de procesamiento más corto ponderado (WSPT). Secuencia las tareas en orden no-decreciente de la proporción p_j/w_j.
- Regla del tiempo de procesamiento más largo (LPT). Secuencia las tareas en orden no-creciente de su tiempo de procesamiento p_j.
- Regla del tiempo de entrega más pronto (EDD). Secuencia las tareas en orden no-decreciente de su fecha de entrega d_j.
- Regla de Johnson para un problema de dos máquinas en ambiente *flow shop*: J_j precede a J_k si $\text{mín}(p_{j1}, p_{k2}) < \text{mín}(p_{j2}, p_{k1})$.

Un *algoritmo ρ-aproximación* es un algoritmo que crea una solución con el criterio que es a lo más $\rho \geq 1$ veces el valor óptimo. Para problemas *offline*, el error relativo del peor caso (R_H) de un algoritmo H es definido como

$$R_H = \text{máx}\{C_H(I)/C_{opt}(I) - 1 : \text{para todo } I\}$$

donde $C_H(I)$ es el desempeño del algoritmo H aplicado a alguna instancia I del problema y C_{opt} es el valor de una solución óptima. Algunos de los algoritmos de aproximación propuestos en algunos trabajos son modificaciones o generalizaciones de las reglas de prioridad .

2.2 Revisión de literatura

En algunos problemas de secuenciación hay un tiempo de preparación de la máquina para procesar una tarea, dado por limpieza o cambio de herramienta. Este tiempo puede ser independiente o dependiente de la última tarea procesada. La consideración de tiempos de preparación ha ganado gran interés de la comunidad científica, debido a que se obtuvieron grandes ahorros cuando se incluyeron explícitamente en varios ambientes industriales y de servicios del mundo real, en el trabajo de Allahverdi et al. (2008) se recopilan diversas aplicaciones.

Por otro lado, están los problemas que consideran restricciones de disponibilidad. En el caso determinista, con excepción del trabajo de Handlarski (1980) que evalúa dos políticas de mantenimiento, no es hasta la década de los 90's cuando se publican algunos resultados. Ma et al. (2010) realizan un estudio sobre estos problemas, presentan resultados de complejidad, algoritmos exactos y algoritmos de aproximación para ambientes de secuenciación de una máquina, máquinas paralelas, *flow shop*, *job shop* y *open shop* con diferentes criterios.

2.2.1 Tiempos de preparación

Allahverdi y Soroush (2008) resumen diferentes tipos de aplicaciones donde se han considerado tiempos/costos de preparación y los beneficios de incluirlos explícitamente en la programación de la producción. Además, proporcionan una definición formal de tiempos/costos de preparación y presentan investigaciones relevantes en el tema.

Se han publicado muchos trabajos que consideran tiempos de preparación. Allahverdi et al. (1999, 2008) realizan estudios exhaustivos sobre problemas de secuenciación con tiempos/costos de preparación de la máquina, dependientes e independientes de la secuencia, desde su aparición a mediados de los 60's hasta 1999 y de 1999 al 2006, respectivamente. Allahverdi et al. (2008) clasifican la literatura de acuerdo

a: (1) ambiente de produccción como secuenciación en una sola máquina, máquinas paralelas, en secuencia y otras; (2) procesamiento por lote o no-lote; (3) tiempos de preparación dependientes o independientes de la secuencia; y (4) disponibilidad de las tareas o de los lotes. Concluyen lo siguiente:

- Ha habido un gran salto de la investigación anual en este campo, de 190 trabajos en 25 años a 300 trabajos adicionales en los 6 años siguientes, hasta el 2006.
- De los problemas de una máquina, máquinas paralelas, *flow shop*, *open shop* y *job shop* se encontraron cerca de 80, 70, 100, 20 y 10 artículos, respectivamente.
- Para el caso de una sola máquina, tres cuartas partes de los artículos consideran procesamiento por lote. Para máquinas paralelas la mayoría considera procesamiento por no-lote. En problemas de *flow shop*, dos terceras partes considera procesamiento por no-lote.
- Los métodos de solución más comúnes son algoritmos de ramificación y acotamiento, formulaciones de programación matemática, algoritmos de programación dinámica, heurísticas y metaheurísticas.
- Entre las metaheurísticas, los algoritmos genéticos se usan en cerca de 35 artículos, mientras que búsqueda tabú en cerca de 18. Recocido simulado también es muy usado, pero un poco menos que búsqueda tabú. En algunos casos los mejores resultados los dan métodos que usan búsqueda local mientras que en otros metaheurísticas híbridas
- Los métodos de solución que han recibido menos atención son las heurísticas con garantía de desempeño, algoritmos *online*, colonia de hormigas (ACO) y algoritmos de enjambre de partículas.
- Algunas clases de problemas han recibido menos atención, por ejemplo, problemas con múltiples máquinas (m no fijo), problemas multiobjetivo, problemas

con múltiples familias (en procesamiento por lote), problemas con tamaño de lote no acotado y problemas estocásticos.

- La mayoría de los artículos estudian tiempos de preparación independientes porque tratar con tiempos dependientes es más difícil.

Dada la inmensa variedad de problemas de secuenciación, en la revisión literaria después del 2006 nos centramos en aquellos problemas donde se consideran: tiempos de preparación de la máquina dependientes de la secuencia, tareas sin interrupción, ambientes de una máquina y máquinas paralelas. Analizamos dichos ambientes debido a que los problemas en una sola máquina son de carácter fundamental por ser interpretados como bloques para problemas más complejos, mientras que los ambientes de máquinas paralelas son los estudiados en este proyecto de investigación.

Una máquina Para el problema de una máquina con tiempos de preparación dependientes de la secuencia y el objetivo del tiempo total de terminación $(1|s_{jk}|\sum T_j)$, Gupta y Smith (2006) presentan dos algoritmos, una heurística de búsqueda local basada en el espacio de solución y un procedimiento GRASP (*Greedy Randomized Adaptative Search Procedure*). Liao y Juan (2007) proponen un algoritmo ACO (*Ant Colony Optimization*) con características que incluyen un nuevo parámetro para la feromona inicial y ajuste del tiempo de aplicación de la búsqueda local, entre otras. En el caso más general con el objetivo de la suma poderada de los tiempos de terminación $(1|s_{jk}|\sum w_j T_j)$, Lin y Ying (2007) usan recocido simulado, algoritmos genéticos y búsqueda tabú para resolver el problema; Liao y Juan (2007); Anghinolfi y Paolucci (2008) proponen ACO's, el más reciente incluye un nuevo mecanismo de actualización de la feromona global y un mecanismo de aprendizaje independiente de la función objetivo; Anghinolfi y Paolucci (2009) proponen un algoritmo de enjambre de partículas, analizando los mecanismos de inteligencia y de búsqueda local incluidos; Tasgetiren et al. (2009) proponen el uso de un método de evolución diferencial discreta desarrollado por ellos, el cual emplea esquemas de inicialización de la población (basados en métodos constructivos) y búsqueda local. Kirlik y Oguz

(2012) proponen un procedimiento general de búsqueda por vecindades variables. Ellos aplican el mismo algoritmo al problema $1|s_{jk}|\sum T_j$. Concluyen que el procedimiento de solución es efectivo, eficiente y robusto, dado que en la mayoría de los casos se desempeña igual o mejor que los mejores algoritmos conocidos para ambos problemas. Luo y Hu (2013) proponen un híbrido entre GRASP y reencadenamiento de trayectorias, en este caso el desempeño no es satisfactorio. De los trabajos anteriores se observan algunas características diferentes entre las metodologías, pero la mayoría incluye búsquedas locales. Se destacan aquellos que usan búsqueda por vecindades variables, por lo que es uno de los aspectos que incorporaremos en nuestros algoritmos.

Para el problema $1|S_{ij}|m - objetivos$, Choobineh et al. (2006) proponen un algoritmo de búsqueda tabú multi-objetivo. El algoritmo produce un conjunto de soluciones de acuerdo a la ponderación de los objetivos. Ellos también formulan un modelo lineal entero mixto para obtener soluciones óptimas para el problema con tres objetivos.

Máquinas paralelas El problema de secuenciación en máquinas paralelas ha sido extensamente estudiado, una recopilación interesante sobre máquinas paralelas puede ser encontrada en el trabajo de Mokotoff (2001). Sin embargo, la mayoría de los trabajos estudian el caso de máquinas paralelas idénticas o uniformes, donde el tiempo de procesamiento de una tarea es el mismo independientemente de la máquina donde se procese o es proporcional a la rapidez de la máquina, respectivamente.

Menos esfuerzo se ha hecho para estudiar el caso donde el tiempo de procesamiento de cada tarea depende de la máquina en la cual es procesada, es decir, el caso cuando las máquinas son no relacionadas (Cheng et al., 2004). Esta es una situación común en muchas aplicaciones donde hay máquinas paralelas con diferentes capacidades.

Monkman et al. (2008) estudian un problema donde buscan minimizar el tiempo de preparación total (TST), representado por $P|S_{ij}|TST$. Ellos abordan un caso

de estudio en una fábrica de productos eléctricos con un alto volumen de ensamble a la orden. Sus resultados empíricos muestran una reducción en los tiempos de preparación del 20 %.

De los trabajos que abordan el caso en que las máquinas son no relacionadas, solo algunos estudian el problema considerando tiempos de preparación de la máquina dependientes de la secuencia. La mayoría asume que no hay costos por cambiar el tipo de tarea que se procesa o que dichos costos son independientes de la secuencia, sin embargo, esta situación puede no ser cierta en la práctica. En varios entornos industriales o de servicio del mundo real estos tiempos son dependientes de la secuencia, es decir, dependen no solo de la tarea que será procesada, sino también de la tarea que se termine de procesar (Lee y Pinedo, 1997).

El número de trabajos dirigidos a problemas de secuenciación en máquinas paralelas no relacionadas con tiempos de preparación dependientes de la secuencia y de la máquina es aún menor, a pesar que esta situación aparece, por ejemplo, en la industria textil, en la manufactura de tarjetas de circuitos impresos y en la industria química (Rabadi et al., 2006). Todo lo previamente mencionado nos motivó a enfocarnos en problemas de secuenciación con máquinas paralelas no relacionadas.

Respecto al problema con el objetivo de minimizar el *makespan* se desarrollaron recientemente algoritmos metaheurísticos. Helal et al. (2006) proponen un algoritmo de búsqueda tabú. Rabadi et al. (2006) sugieren una nueva metaheurística, MetaRaSP, la cual incorpora aleatoriedad con reglas de prioridad para construir soluciones factibles. Más recientemente, Arnaout et al. (2010) proponen un algoritmo de colonia de hormigas; Ying et al. (2012) desarrollan una metodología de recocido simulado que incorpora una estrategia de búsqueda restringida. Vallada y Ruiz (2011) presentan un algoritmo genético que emplea un nuevo operador de cruzamiento e incluye un procedimiento de búsqueda local rápida. Fleszar et al. (2012) proponen un algoritmo de búsqueda por vecindades variables híbrido con programación matemática. Hasta donde sabemos, solo un trabajo (Tran y Beck, 2012) ha desarrollado un método exacto para resolver éste problema y uno más (Rocha et al.,

2008) para minimizar el *makespan* más la tardanza ponderada. Tran y Beck (2012) proponen un método basado en descomposición de Benders, mientras que Rocha et al. (2008) presentan un algoritmo de ramificación y acotamiento.

Ninguno de los trabajos anteriores considera restricciones de disponibilidad, esto es, asumen que las máquinas están disponibles todo el tiempo, por lo que el ambiente de máquinas paralelas no relacionadas con tiempos de preparación dependientes de la máquina y de la secuencia ofreció un buen punto de partida para analizar el problema, desarrollar aspectos novedosos en los algoritmos de solución y comparar su desempeño con las metodologías de la literatura, para finalmente considerar estos aspectos al incorporar un programa de mantenimiento preventivo.

2.2.2 Restricciones de disponibilidad

Para el problema de una máquina y el objetivo de minimizar el total completion time si la máquina tiene un tiempo de liberación $(1, h_1||\sum C_i)$ la regla SPT es optimal si dicho intervalo está al principio del periodo de planeación. Con un solo de estos intervalos Lee y Liman (1992) demuestran que el problema $1, h_1||\sum C_i$ es NP-duro y acotan el error relativo a de la regla SPT a 2/7. Más tarde Qi et al. (1999) proponen un algoritmo de aproximación MSPT con un error relativo acotado de $R_{MSPT} \leq 3/17$. Breit (2007) presenta un algoritmo paramétrico $O(n \log n)$ en el que garantiza un error relativo menor que 0.1. Si consideramos ahora el problemo con objetivo del *makespan* $(1, h_1||C_{max})$ Lee (1996) muestra fácilmente que este problema es NP-duro. También muestra que si ordenamos las tareas mediante la regla LPT, y después asignamos cada tarea tan pronto como sea posible mientras la máquina esté disponible y luego asignamos las tareas restantes en un orden arbitrario cuando vuelve a estar disponible, entonces este procedimiento conduce a un error relativo de $R_{LPT} \leq 1/3$.

Para $1, h_l||C_{max}$, donde múltiples intervalos de no disponibilidad son permitidos, Lee (1996) demuestra que el problema es NP-duro en el sentido fuerte. Ji et

al. (2007) abordan el problema cuando dichos intervalos son actividades de mantenimiento periódicas, mientras que Chen (2008) las considera flexibles (casi-periódicas). El propone dos modelos de programación entera binaria y una heurística para instancias grandes.

Lee (1996) establece que el problema $P, h_{i1}||C_{max}$ es NP-duro siempre que haya al menos una máquina disponible. Además, muestra errores relativos en el peor de los casos de $m-1$ y $(m+1)/2-1$ para las reglas de prioridad *list scheduling* (LS) y LPT, respectivamente. Cuando cada máquina puede tener un intervalo de no disponibilidad $(P, h_{i1}||C_{max})$, Hwang y Chang (1998) analizan el desempeño de LPT y muestran que el comportamiento en el peor caso es gobernado por el número de máquinas no disponibles simultáneamente (*nds*). También, muestran que el error relativo de LPT es $R_{LPT} \leq 1$ si el número máximo de máquinas *nds* es acotado por $m/2$. Hwang et al. (2005) generalizan este resultado para un número arbitrario λ de máquinas *nds* con un error de $1 + \frac{1}{2}\lceil m/(m-\lambda)\rceil$.

Para el problema de secuenciación en dos máquinas paralelas idénticas con un solo periodo de no-disponibilidad en una de ellas $P2, h_{21}||C_{max}$, Liao et al. (2005) parten el problema en cuatro subproblemas y los resuelven a optimalidad basándose en el algoritmo optimal para dos máquinas TMO (*Two Machine Optimal*). Lin y Liao (2007) generalizan los resultados para un intervalo de no disponibilidad en cada máquinas $P2, h_{j1}||C_{max}$, asumen que los dos intervalos tienen la misma duración, dividen el problema en diferentes casos y desarrollan un algoritmo basado en búsqueda lexicográfica para encontrar soluciones óptimas para cada uno de ellos.

Para el problema $Rm, h_{il}||C_{max}$, donde cada máquina tiene múltiples intervalos de no disponibilidad, Suresh y Ghaudhuri (1996) proponen una heurística multi-paso. Ellos tratan dichos intervalos de una máquina como una tarea hecha solo en una máquina y trasladan el problema a $n+m$ tareas con m máquinas, entonces formulan un modelo lineal y lo resuelven con el algoritmo para el problema de asignación generalizada.

Nosotros observamos que las metodologías empleadas en los trabajos que consideran restricciones de disponibilidad abordan solo un periodo de disponibilidad y se basan principalmente en algoritmos de aproximación derivados a partir de las reglas de prioridad. Mientras que aquellos que incluyen tiempos de preparación dependientes de la secuencia suelen desarrollar procedimientos metaheurísticos. Por lo tanto, para el segundo problema que estudiaremos, el cual contempla múltiples periodos de disponibilidad e incluye tiempos de preparación dependientes, aplicaremos técnicas basadas en metaheurísticas pero tomaremos en consideración la regla de prioridad LPT que con tanta frecuencia se utiliza en los algoritmos de aproximación antes mencionados.

2.2.3 Tiempos de preparación dependientes y mantenimiento preventivo

En secuenciación, el mantenimiento preventivo es un caso particular de un área conocida como restricciones de disponibilidad. Considerar mantenimiento preventivo implica que puede ocurrir uno o más periodos de tiempo en el que la máquina no esté disponible. Han sido pocos los que han integrado secuenciación de la produccción y mantenimiento preventivo periódico considerando además tiempos de preparacion dependientes.

Para el problema $1, h_k|s_{ij}|C_{max}$, Chen (2009) estudia un caso real que surge en la industria textil y de manufactura. El asume que una tarea puede interrumrse debido a mantenimiento y que la tarea interrumpida puede ser retomada sin tiempo de preparación si éste ya fue realizado. El tampoco considera tiempo de preparación para realizar mantenimiento, lo que simplifica el problema en el sentido que la inclusión de mantenimiento no afecta el orden en el cual se procesarán las tareas. Ángel-Bello et al. (2011a,b) proponen, primero, una formulación matemática y desigualdades válidas para el problema; y segundo, una metodología de solución basada en GRASP y búsqueda tabú. Pacheco et al. (2012) proponen una reformulación del problema y utilizan una búsqueda tabú para resolverlo.

Naderi et al. (2009a) estudian el problema $J, h_{jk}|s_{ij}|C_{max}$, ellos comparan el desempeño de varios algoritmos heurísticos y metaheurísticos, donde destaca el desempeño de un algoritmo SPT-GA, híbrido de la regla SPT adaptada al problema y un algoritmo genético.

Naderi et al. (2009b) estudian el problema $FF, h_{jk}|s_{ij}|C_{max}$, donde cada etapa del FF está compuesta de máquinas paralelas idénticas. Ellos proponen un algoritmo basado en Búsqueda por Vecindades Variables (VNS) y comparan su desempeño contra otros algoritmos que ellos mismos adaptan al problema.

2.2.4 Otros estudios

Lee y Lin (2001) consideran tiempos de procesamiento deterministas e interrupciones dadas por una ley de probabilidad (estocásticas). Ellos consideran que la velocidad de la máquina disminuye linealmente con el tiempo y que retoma su velocidad original con una actividad de mantenimiento. Low et al. (2008) también consideran deterioro lineal, pero solo un periodo de mantenimiento fijo. Lee y Wu (2009, 2010); Wei y Wang (2010) también estudian situaciones donde el tiempo de procesamiento y tiempo de preparación varían con el tiempo, sin embargo, no consideran intervalos de no disponibilidad. Kuo y Yang (2007) estudian el problema de una sola máquina con tiempos de preparación dependientes del pasado y consideración de aprendizaje. Ellos asumen que el proceso de aprendizaje disminuye el tiempo de procesamiento en función del número de repeticiones. También consideran varias funciones objetivo y proponen algoritmos exponenciales para resolverlos. Yang y Chand (2008) estudian los problemas de aprendizaje y olvido sobre el problema de secuenciar familias de tareas en una sola máquina para minimizar $\sum C_j$, donde un tiempo de preparación ocurre cuando una máquina cambia de procesar una tarea de una familia a otra de otra familia. Ellos proponen una algoritmo de ramificación y un procedimiento heurístico.

2.3 Áreas de oportunidad

La gran mayoría de los estudios que consideran tiempos dependientes de la secuencia, suponen que las máquinas están siempre disponibles durante el periodo de planeación. Sin embargo, esto puede no ser cierto, dado que una máquina puede no estar disponible durante ciertos periodos de tiempo ya sea por fallas (caso estocástico) o por mantenimiento (caso determinista). Por otra parte, la mayoría de los problemas que consideran restricciones de disponibilidad consideran que no hay tiempos de preparación o que estos son independientes de la secuencia. Sin embargo, hay contextos en los que los tiempos de preparación dependientes de la secuencia deben ser considerados (Allahverdi y Soroush, 2008).

Dado que ambas suposiciones simplifican el análisis y/o reflejan ciertas aplicaciones particulares, esto afectaría la calidad de la solución de problemas de secuenciación que explícitamente consideren ambos aspectos. Los problemas de secuenciación que tratan los tiempos de preparación dependientes de la secuencia y restricciones de disponibilidad de las máquinas por mantenimiento preventivo, son de reciente aparición. Los únicos resultados actuales en esta área se desarrollaron para los casos de una sola máquina, *job shop* y *flexible flow shop*, considerando la función objetivo del *makespan*.

Hasta donde sabemos, para el caso de máquinas paralelas no existe en la literatura un trabajo que aborde un problema de secuenciación en el que se conjugan tiempos de preparación dependientes y mantenimiento preventivo. Se tiene la oportunidad de estudiar dicho ambiente de producción u otras funciones objetivo. En este proyecto se estudia el ambiente de máquinas paralelas no relacionadas, tiempos de preparación dependientes de la máquina y de la secuencia, número indefinido de tareas de mantenimiento en cada máquina y la función objetivo del *makespan*, es decir, el problema denotado por $R, h_{jk}|s_{ijk}|C_{max}$.

Capítulo 3

El problema $R|S_{ijk}|C_{max}$

En este capítulo estudiamos el problema de secuenciar n tareas en m máquinas paralelas no relacionadas con el objetivo de minimizar el *makespan*, considerando tiempos de preparación de la máquina dependientes de la máquina y de la secuencia ($R|S_{ijk}|C_{max}$). Por simplicidad nos referiremos a éste como primer problema.

3.1 Introducción

En el trabajo de Garey y Johnson (1979) se muestra que la minimización del *makespan* considerando dos máquinas paralelas idénticas es un problema NP-duro. Es claro que, un problema con máquinas paralelas no relacionadas y tiempos de preparación dependientes de la máquina y de la secuencia es también NP-duro. Esto y el hecho que la resecuenciación es comúnmente requerida, el uso de algoritmos exactos llega a ser costoso en tiempo computacional. Por ello, no es sorprendente que muchas metodologías que han sido desarrolladas estén basadas en metaheurísticas.

En este trabajo nosotros proponemos una nueva linealización del *makespan* que empleada en dos nuevas reformulaciones del problema permite resolver de manera exacta, instancias más grandes que las publicadas en la literatura y usando menor tiempo de cómputo. Para instancias aún más grandes, diseñamos e implementamos un algoritmo de solución basado en algoritmos multiarranque y búsqueda por vecindades variables, el cual supera los mejores resultados conocidos. Para nuestro algoritmo, definimos y proponemos el uso de movimientos compuestos que toman

ventaja de la relación entre los sub-problemas de asignación y secuenciación, lo que considerablemente mejora el desempeño del algoritmo.

3.2 Descripción del problema

Las siguientes suposiciones y notación son usadas para describir el problema:

- Hay un conjunto M de m máquinas paralelas no relacionadas.
- Las máquinas están continuamente disponibles y cada máquina procesa una tarea a la vez sin interrupción, es decir, una vez que el procesamiento de una tarea ha iniciado, este no puede ser interrumpido.
- Hay un conjunto N de n tareas para ser secuenciadas.
- Todas las tareas están disponibles para procesarse al tiempo cero. No hay restricciones de precedencia impuestas al orden de procesamiento de las tareas.
- Cada tarea j tiene asociado un tiempo de procesamiento p_{ij} que depende de la máquina i en la que se procese.
- Hay un tiempo de preparación s_{ijk} de la máquina i para procesar la tarea k justo después de procesar la tarea j.
- El objetivo es minimizar el *makespan* C_{max}. Usando el término *span* para denotar el tiempo de terminación de una máquina, el *makespan* denota el máximo *span* en la solución del problema.

La Figura 3.1 a) muestra que los tiempos de preparación son asimétricos, mientras que la Figura 3.1 b) muestra que los tiempos de procesamiento y tiempos de preparación son dependientes de la máquina.

Encontrar la solución a este problema significa determinar la asignación de tareas a máquinas y el orden en el cual cada máquina procesará las tareas asignadas a ella.

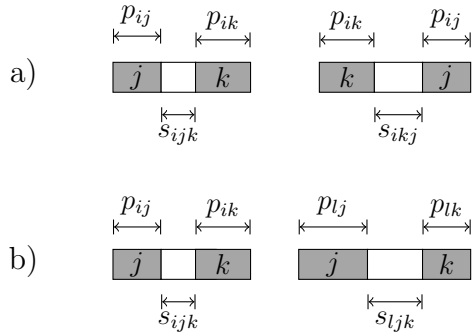


Figura 3.1: Representación gráfica de los tiempos de procesamiento y tiempos de preparación.

La Figura 3.2 muestra una representación gráfica de una solución al problema en cuestión con 17 tareas y 3 máquinas. En la figura, los bloques blancos representan tiempos de preparación y los bloques grises representan tiempos de procesamiento.

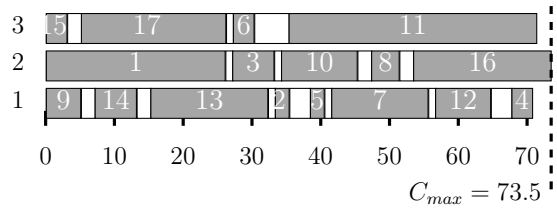


Figura 3.2: Representación gráfica de una solución.

Primero, proponemos dos reformulaciones enteras mixtas, Model1 y Model2, para formalizar el problema. Model1 es una modificación de las formulaciones presentadas por Rabadi et al. (2006) y Vallada y Ruiz (2011). Model2 es una generalización del modelo propuesto para el sub-problema de asignación usado en un método de descomposición de Benders (Tran y Beck, 2012). Posteriormente, se incluye una nueva linealización para el *makespan* para obtener algunas variantes de estos modelos.

Para describir el primer modelo (denominado Model1), introducimos las siguientes variables.

$$X_{ijk} = \begin{cases} 1, & \text{si la tarea } j \text{ es secuenciada justo antes de la tarea } k \text{ en} \\ & \text{la máquina } i, \\ 0, & \text{en otro caso.} \end{cases}$$

C_j : Tiempo de terminación de la tarea j.

C_{max} : *Makespan*

Denotaremos por N_0 el conjunto N más una tarea ficticia 0 y V denotará una constante muy grande. Las variables X_{i0k} y X_{ij0} son usadas para especificar cuáles tareas k y j serán la primer y última tareas procesadas en cada máquina. El tiempo de procesamiento al igual que los tiempos de preparación de la tarea ficticia serán considerados nulos ($p_{i0} = 0$, $s_{i0k} = 0$ y $s_{ij0} = 0$, $\forall i \in M$).

El denominado Model1 puede establecerse como

$$\text{Minimizar:} \quad C_{max}. \tag{3.1}$$

Sujeto a:

$$\sum_{i \in M} \sum_{\substack{j \in N_0 \\ j \neq k}} X_{ijk} = 1, \qquad \forall k \in N, \tag{3.2}$$

$$\sum_{i \in M} \sum_{\substack{k \in N_0 \\ j \neq k}} X_{ijk} = 1, \qquad \forall j \in N, \tag{3.3}$$

$$\sum_{\substack{k \in N_0 \\ k \neq j}} X_{ijk} - \sum_{\substack{h \in N_0 \\ h \neq j}} X_{ihj} = 0, \qquad \forall j \in N, \forall i \in M, \tag{3.4}$$

$$\sum_{k \in N} X_{i0k} \leq 1, \qquad \forall i \in M, \tag{3.5}$$

$$C_k - C_j + V(1 - X_{ijk}) \geq s_{ijk} + p_{ik}, \qquad \forall j \in N_0, \forall k \in N, j \neq k, \forall i \in M, \tag{3.6}$$

$$C_0 = 0, \tag{3.7}$$

$$C_j \leq C_{max}, \qquad \forall j \in N, \tag{3.8}$$

$$X_{ijk} \in \{0, 1\}, \qquad \forall j \in N_0, \forall k \in N_0, j \neq k, \forall i \in M,$$

$$C_j \geq 0, \qquad \forall j \in N. \qquad (3.9)$$

El objetivo (3.1) minimiza el *makespan*. Las restricciones (3.2) establecen que cada tarea tiene exactamente una predecesora, mientras las restricciones (3.3) establecen que cada tarea tiene exactamente un sucesora. Las restricciones (3.4) son las llamadas "restricciones de conservación de flujo". Éstas aseguran que si una tarea es secuenciado en una máquina, entonces su predecesora y sucesora deben ser procesadas en la misma máquina. Las restricciones (3.5) aseguran que a lo más una tarea es secuenciada como la primer tarea en esa máquina. Las restricciones (3.6) proporcionan un orden de procesamiento correcto, evitando ciclos. Básicamente establecen que, si $X_{ijk} = 1$, entonces el tiempo de terminación de la tarea k debe ser mayor que el tiempo de terminación de la tarea j . Si $X_{ijk} = 0$, las restricciones son redundantes. La restricción (3.7) fija a cero el tiempo de terminación de la tarea ficticia y en conjunto con las restricciones (3.6), garantizan que el tiempo de terminación de todas las tareas es positivo. Las restricciones (3.8) linealizan la función objetivo. Finalmente, las restricciones (3.9) definen la naturaleza de las variables.

Para la segunda formulación (denominado Model2), además de las variables empleadas en Model1, introducimos las siguientes variables:

$$Y_{ik} = \begin{cases} 1, & \text{si la tarea } k \text{ es asignada a la máquina } i \\ 0, & \text{en otro caso.} \end{cases}$$

Model2 puede ser obtenido de Model1 remplazando las restricciones (3.2), (3.3) y (3.4) por

$$\sum_{i \in M} Y_{ik} = 1, \qquad \forall k \in N, \qquad (3.10)$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ijk}, \qquad \forall k \in N, \forall i \in M, \qquad (3.11)$$

$$Y_{ij} = \sum_{k \in N_0, k \neq j} X_{ijk}, \qquad \forall j \in N, \forall i \in M, \qquad (3.12)$$

y agregando $Y_{ik} \geq 0$ a las restricciones (3.9). No es necesario definir las variable Y_{ik} como variable binarias, ya que esto se satisface de la definición de las variables X_{ijk} como binarias y la estructura de las restricciones.

Las restricciones (3.10) aseguran que cada tarea es asignada a exactamente una máquina. Las restricciones (3.11) establecen que cada tarea tiene exactamente un predecesor y ambos, tarea y predecesor, son asignados a la misma máquina. Las restricciones (3.12) garantizan que cada tarea tiene exactamente un sucesor y ambas son asignadas a la misma máquina.

Model1 tiene $n^2m + nm$ variables binarias, $n + 1$ variables continuas y $n^2m + nm+3n+m+1$ restricciones, mientras que Model2 tiene n^2m+nm variables binarias, $nm + n + 1$ variables continuas y $n^2m + 2nm + 2n + m + 1$ restricciones.

En ambas formulaciones el *makespan* $C_{max} = \text{máx}_{j \in N}\{C_j\}$ fue linealizado como el máximo de los tiempos de terminación de las tareas, es decir,

$$C_j \leq C_{max}, \qquad \forall j \in N.$$

Cuando resolvimos Model1 y Model2 usando un algoritmo de ramificación y acotamento (B&B), observamos que la cota inferior obtenida por la relajación lineal de cada modelo es muy débil. Esto nos motivó a proponer una representación diferente de la función objetivo, la cual fue obtenida del análisis de la estructura de las soluciones de la relajación lineal de cada modelo entero mixto.

3.2.1 Nueva linealización del *makespan*: Model1a, Model1b, Model2a, Model2b y Model3

Específicamente, cuando resolvimos la relajación lineal de cada modelo para diferentes instancias, observamos que los valores de los C_j son cercanos a cero. Con dichos valores de tales variables, también el valor de la función objetivo es cercano a cero.

Para obtener una nueva linealización del *makespan* usamos el concepto de *span*,

concepto que ya fue usado en diferentes metodologías de solución, pero al cual no se le ha dado importancia en el desarrollo de los modelos matemáticos. El *span* O_i de la máquina i es definido como el tiempo de terminación de la última tarea procesada en la máquina i. Usando variables X_{ijk} el *span* puede ser calculado como

$$O_i = \sum_{\substack{j \in N_0 \\ j \neq k}} \sum_{k \in N} (s_{ijk} + p_{ik}) X_{ijk}, \qquad \forall i \in M,$$

y entonces, el *makespan* puede linealizarse como

$$O_i \leq C_{max}, \qquad \forall i \in M,$$

o equivalentemente como

$$\sum_{\substack{j \in N_0 \\ j \neq k}} \sum_{k \in N} (s_{ijk} + p_{ik}) X_{ijk} \leq C_{max}, \qquad \forall i \in M. \tag{3.13}$$

Además, usando las variables Y_{ik} la linealización (3.13) puede ser reescrita como

$$\sum_{\substack{j \in N_0 \\ j \neq k}} \sum_{k \in N} s_{ijk} * X_{ijk} + \sum_{k \in N} p_{ik} * Y_{ik} \leq C_{max}, \qquad \forall i \in M. \tag{3.14}$$

No es difícil ver que cualquier solución factible satisface estas restricciones, lo cual significa que las restricciones (3.13) y (3.14) son desigualdades válidas para Model1 y Model2, respectivamente. Por otro lado el lado izquierdo de dichas restricciones calcula fácilmente el tiempo en que cada máquina procesa su última tarea (O_i), como la suma del tiempo de preparación para procesar cada tarea más el tiempo de procesamiento de dicha tarea. Además, estas no dependen del valor de la constante V y forzan a C_{max} a tomar un valor positivo si cualquiera de las variables X_{ijk} o Y_{ik} son positivas, lo cual siempre ocurre debido a las restricciones (3.2)-(3.4) o (3.10)-(3.12).

Para cada modelo (Model1 y Model2) se consideran dos alternativas:

a) Reemplazar en Model1 y Model2, las restricciones (3.8) por las nuevas restricciones (3.13) y (3.14), respectivamente, obteniendo Model1a y Model2a.

b) Agregar a Model1 y Model2, las nuevas restricciones (3.13) y (3.14), respectivamente, obteniendo Model1b y Model2b.

Note que Model1a tiene $n-m$ menos restricciones que Model1, Model1b tiene m restricciones más que Model1, Model2a tiene $n-m$ menos restricciones que Model2 y Model2b tiene m restricciones más que Model2.

La motivación de una tercera formulación radica en explorar el desempeño y factibilidad de aplicación de la nueva linealización con otro tipo de variables usadas en problemas de secuenciación. Por ello decidimos modelar el problema con variables que asignan una tarea a una posición en la secuencia, variables que introducimos a continuación.

$$X_{ijkl} = \begin{cases} 1, & \text{si una tarea } j \text{ es procesada en la posición } l \text{ y la tarea } k \text{ es procesada} \\ & \text{en la posición } l+1 \text{ en la máquina } i \\ 0, & \text{en otro caso.} \end{cases}$$

$$Y_{ijl} = \begin{cases} 1, & \text{si la tarea } j \text{ es asignada a la máquina } i \text{ en la posición } l \\ 0, & \text{en otro caso.} \end{cases}$$

$$\text{Minimizar:} \quad C_{max} \tag{3.15}$$

Sujeto a:

$$\sum_{i}\sum_{l} Y_{ij}^{l} = 1, \qquad \forall j \in N, \tag{3.16}$$

$$\sum_{j} Y_{ij}^{l} \leq 1, \qquad \forall i \in M, \forall l \in N, \tag{3.17}$$

$$\sum_{k \in N, k \neq j} X_{ijk}^{l} = Y_{ij}^{l} \qquad \forall j \in N, \forall l \in N^{-}, \forall i \in M, \tag{3.18}$$

$$\sum_{j \in N_0, j \neq k} X_{ijk}^{l} = Y_{ik}^{l+1} \qquad \forall k \in N, \forall l \in N^{-}, \forall i \in M, \tag{3.19}$$

$$\sum_{k \in N}\sum_{l \in N} s_{i0k} X_{i0kl} + \sum_{j \in N_0}\sum_{k \in N}\sum_{l \in N} s_{ijk} X_{ijkl}$$

$$+\sum_{k\in N}\sum_{l\in N} p_{ik}Y_{ikl} \leq C_{max}, \qquad \forall i \in M \quad (3.20)$$

$$Y_{ijk} \in \{0,1\}, \qquad \forall j \in N_0, \forall k \in N, j \neq k, \forall i \in M,$$

$$X_{ijkl} \geq 0, \qquad \forall j \in N, \forall k \in N, \forall l \in N, \forall i \in M, \quad (3.21)$$

Las restricciones (3.16) establecen que una tarea solo puede asignarse a alguna posición de alguna de las máquinas. Las restricciones (3.17) establece que en cada posición de cada máquina solo se puede asignar a lo más una tarea. Las restricciones (3.18) establecen que si una tarea j fue asignada en el nivel l de la máquina i entonces debe haber una tarea k que le sucede en la posición $l+1$ en la misma máquina. Las restricciones (3.19) establecen que si una tarea k es asignada en la posición $l+1$ entonces debe haber una tarea j que le precede en el nivel l de la misma máquina. Las restricciones (3.20) linealizan la función objetivo como el máximo de los *span*'s de las máquinas. Las restricciones (3.21) establecen las naturaleza de las variables.

En la sección de experimentos computacionales mostraremos que la nueva linealización del *makespan* acelera el proceso de solución de optimizador basado en ramificación y acotamiento. También se dará un análisis del comportamiento de las alternativas previas.

3.3 Algoritmo metaheurístico propuesto

Cuando se diseña un método de solución para problemas de secuenciación hay que considerar que usualmente es difícil en este contexto definir vecindarios que preserven la factibilidad, pero construir una solución puede ser un proceso más simple. Por otro lado, la diversificación debe ser una componente importante del método para no localizarse en pequeñas áreas del espacio de solución. Los métodos multiarranque proporcionan una estructura que aporta diversificación y aprovecha la facilidad de construir soluciones. A continua se describiran algunas ideas de los algoritmos multiarranque, así como un algoritmo genérico. Para más información sobre

estos algoritmos ver Martí et al. (2010).

Un método multiarranque se ejecuta múltiples veces desde puntos iniciales distintos en el espacio de solución. Los primeros fueron desarrollados para problemas de optimización no lineal sin restricciones y consistieron de la evaluación de la función objetivo en puntos generados aleatoriamente.

En optimización con metaheurísticas los algoritmos multiarranque se componen, generalmente de dos fases: 1) constructivo o generador de diversificación, 2) método de mejora. Un algoritmo multiarranque itera entre estas dos fases mientras guarda la mejor solución encontrada a lo largo del método de mejora. La Figura 3.3 muestra un pseudo-código para un algoritmo multiarranque genérico.

```
   Data: Instancia del problema.
1  x* ← ∅;
2  while criterio de parada no se satisfaga do
3  |   x ← Constructivo ();
4  |   x ← Mejora (x);
5  |   if x es mejor que x* then
6  |   |   x* ← x;
7  |   end
8  end
   Result: Solución x*.
```

Figura 3.3: Pseudo-código para un algoritmo multiarranque genérico.

La función del constructivo o del generador de diversificación es proporcionar soluciones iniciales para la fase de mejora, garantizando al mismo tiempo un cierto grado de diferencia entre ellas. Con el fin de aportar diversificación, la aleatoriedad o algunos tipos de memoria se incluyen comúnmente en los métodos que se utilizan para proporcionar soluciones iniciales. Otra forma de incluir la diversidad es generar nuevas soluciones al perturbar las soluciones generadas previamente.

En cuanto a los métodos de mejora, los procedimientos de búsqueda local son comúnmente simples y sin embargo poderosos. En los métodos de búsqueda local, la búsqueda de una nueva solución se lleva a cabo con respecto a una estructura de vecindad. Por lo general, los vecinos a una solución dada son soluciones factibles que

se pueden obtener a través de movimientos simples. La búsqueda comienza en una solución inicial factible, explora el vecindario y se mueve a una solución vecina que mejora el valor de la función objetivo, si dicha solución existe. Cuando se encuentra una nueva solución, se reanuda la búsqueda. Ésta termina cuando para la solución actual no existen soluciones vecinas que mejoren, es decir, cuando la búsqueda llega a un óptimo local.

A continuación, describimos nuestra implementación de los procedimientos constructivo y de mejora para el problema abordado.

3.3.1 Constructivo

La representación de la solución más comúnmente utilizada para el problema de programación de máquinas paralelas es un arreglo S_i de tareas para cada máquina i, que representa el orden de procesamiento de las tareas asignadas a esa máquina, es decir, $x = \{S_1, S_2, \ldots, S_m\}$, donde S_1 representa el orden de procesamiento de las tareas asignadas a la máquina uno, S_2 represeta lo mismo para la máquina dos, y así sucesivamente.

Un pseudo-código para este procedimiento constructivo se muestra en la Figura 3.4 y funciona de la siguiente manera: En primer lugar, las tareas se clasifican en orden no creciente de acuerdo al valor de su tiempo de procesamiento promedio sobre todas las máquinas, es decir, $\bar{p_j} = \sum_{i \in M} p_{ij}/m$, (línea 2) y luego se forma una lista de candidatos con las primeras s tareas (línea 3), de la cual se selecciona una tarea al azar (línea 5). Para la tarea seleccionada j se determina el mejor punto de inserción en cada secuencia S_i (líneas 6-10), es decir, en el punto de inserción donde el *span* de la máquina aumente menos. Denotemos por Δ_i el aumento de la duración O_i de la máquina i para el mejor punto de inserción de la tarea j en esa máquina. Entonces la tarea j se inserta en la máquina $i^* = \text{argmin } \{O_i + \Delta_i\}$ en el mejor punto de inserción en la máquina i^* (línea 11). Entonces la lista de candidatos se actualiza (línea 12) y el proceso se repite (regresamos a línea 5) hasta que todas las tareas se hayan asignado a las máquinas.

```
Data: Instancia del problema.
foreach i ∈ M do  S_i ← Secuencia vacía: O_i ← 0:
L ← Lista de tareas j ∈ N ordeada no crecientemente por p̄_j ;
LC ← Lista candidata con las primeras s tareas;
while LC ≠ ∅ do
    j ← Seleccionar aleatoriamente una tarea de LC;
    foreach i ∈ M do
        Encontrar la mejor posición q_i para insertar j en S_i;
        Guardar q_i y Δ_i;
    end
    i* ← arg mín_i{O_i + Δ_i}
    Insertar j en la secuencia S_i* en la posición q_i*
    Borrar j de L y actualizar LC;
end
Result: Solución x = (S_1, S_2, ..., S_m).
```

Figura 3.4: Pseudo-código para el procedimiento Constructivo().

3.3.2 Procedimiento de mejora

El procedimiento de mejora consiste en dos etapas. En ambas etapas se utilizan movimientos entre máquinas y dentro de las máquinas, pero se utilizan de forma diferente en cada etapa.

En la primera etapa, primeramente, aplicamos búsquedas locales basadas en los movimientos entre máquinas, entonces aplicamos a cada máquina movimientos dentro de la máquina basados en Or-opt (Or, 1976). En la segunda etapa, se emplean movimientos compuestos. Éstos los definimos como el uso de un solo movimiento entre máquinas y la mejora, mediante un Or-opt, a cada secuencia de las máquinas involucradas en el movimiento.

En ambas etapas se aplican ideas tomadas de la metaheuristica de búsqueda por vecindades variables VNS (Hansen et al., 2010) que se basa en los siguientes hechos: (i) un mínimo local con respecto a una estructura de vecindad no es necesariamente un mínimo local respecto a otras vecindades y ii) un mínimo global es un mínimo local con respecto a todas las estructuras de vecindad posibles.

Un pseudo-código genérico de VND (caso determinista del VNS) se muestra

en la Figura 3.5. En éste la solución inicial se mejora mediante la exploración de varias estructuras de vecindad $N_g(g = 1, \ldots, g_{\text{máx}})$, empezando por N_1 (línea 1). Se explora la vecindad N_g (línea 3) y si se encontró una mejora el algoritmo reinicia la exploración desde N_1 (línea 4), de lo contrario, se mueve a N_{g+1} (línea 5). Cuando g excede $g_{\text{máx}}$ el proceso se termina (línea 2).

Data: Solución inicial x.
1 $g \leftarrow 1$;
2 **while** $g \leq g_{\text{máx}}$ **do**
3 | $x' \leftarrow$ La mejor solucion de $N_g(x)$;
4 | **if** x' *es mejor que* x **then** $x \leftarrow x'$; $g \leftarrow 1$;
5 | **else** $g \leftarrow g + 1$;
6 **end**
Result: Solución x.

Figura 3.5: Pseudo-codigo para un VND genérico.

3.3.2.1 PROCEDIMIENTO DE MEJORA: ETAPA I

En esta etapa se incluyen dos vecindades para explorar las asignaciones de tareas a las máquinas y otro para mejorar la secuencia de tareas en cada máquina. Los vecindarios usados para tratar con el problema de asignación son:

- El clásico vecindario de inserción donde cada tarea se extrae de la máquina asignada actualmente y se inserta en todas las posiciones posibles de todas las demás máquinas.
- El vecindario de intercambio donde cada tarea de cada máquina se intercambia con cada tarea asignada a alguna otra máquina.

Ambos, el vecindario de inserción y el de intercambio, se han dividido en pequeños sub-vecindarios y una búsqueda VND ha sido implementada para cada tipo de movimiento. Esto debido por una parte a que los vecindarios de inserción e intercambio tienen un gran número de soluciones y por otra parte a que con la subdivisión se logra explorar en primer término la parte de la vecindad que disminuye el *makespan* y en segundo término la parte de vecindad que no disminuye el *makespan*,

pero que abre espacio en las máquinas para que en iteraciones posteriores se pueda lograr una disminución del *makespan*.

Específicamente, cada vecindario se ha dividido en sub-vecindarios, donde el primero en ser explorado es el que involucra a la máquina más ocupada. Es decir, denotemos por L_{maq} la lista de índices de las máquinas ordenados de una manera no creciente por su valor de *span* y sean $[i]$ y $[l]$ las máquinas que ocupan la posición i y la posición l en L_{maq}, respectivamente. Entonces

- $N^i_{ins}(x)$: sub-vecindario de inserción donde cada tarea se extrae de la máquina $[i]$ y se inserta en todas las posiciones posibles de todas las demás máquinas.
- $N^i_{int}(x)$: sub-vecindario de intercambio en el que cada tarea de la máquina $[i]$ se intercambia de posición con cada tarea asignada a cualquier otra máquina $[l]$, $i < l$.

El procedimiento VND que utiliza sub-vecindarios $N^i_{ins}(x)$ se denotará como *ins-VND*, mientras que el otro que utiliza sub-vecindarios $N^i_{int}(x)$ se denominará como *int*-VND.

La Figura 3.6 muestra un pseudo-código para el procedimiento de mejora: Etapa I. En primer lugar, se aplica *ins*-VND (línea 2). Cuando no se encuentran mejoras usando estas sub-vecindades, se aplica *int*-VND (línea 3). Cuando no se encuentran mejoras usando estos sub-vecindarios, un procedimiento de búsqueda local basado en movimientos del Or-opt se aplica a cada máquina tratando de disminuir el *span* de la máquina (línea 4). Los procedimientos en la líneas 2-4 se repiten hasta que dos procedimientos consecutivos no mejoran sus soluciones de entrada.

Los detalles sobre nuestra implementación de VND serán discutidos en la subsección 3.3.2.4.

Data: Solución $x = (S_1, S_2, \ldots, S_m)$.
1 **repeat**
2 | $x \leftarrow$ Encuentra óptimo local con *ins*-VND (x);
3 | $x \leftarrow$ Encuentra óptimo local con *int*-VND (x);
4 | $x \leftarrow$ **foreach** $i \in M$ **do** $S_i \leftarrow$ Or-opt (S_i);
5 **until** *hasta que dos procedimientos consecutivos no mejoran la solución*
Result: Solución $x = (S_1, S_2, \ldots, S_m)$.

Figura 3.6: Pseudo-código para el procedimiento de mejora: Etapa I (x).

3.3.2.2 PROCEDIMIENTO DE MEJORA: ETAPA II

La idea general es similar a la esbozada en la Figura 3.6 pero ahora usamos movimientos compuestos para definir los sub-vecindarios. Un movimiento de inserción (ó intercambio) compuesto consiste en un solo movimiento de inserción (ó intercambio), además de la optimización de la secuencia en ambas máquinas afectadas por la inserción (ó intercambio).

En este caso, para cada VND, los subvecindarios se dividen de la siguiente manera:

- $N^i_{com\_ins}(x)$: El sub-vecindario de *inserción compuesto* donde cada tarea de la máquina $[i]$ se extrae de dicha máquina y se inserta al final de cualquier otra máquina $[l]$, y ambas secuencias $S_{[i]}$ y $S_{[l]}$ se optimizan usando Or-opt.
- $N^i_{com\_int}(x)$: El sub-vecindario de *intercambio compuesto* donde cada tarea de la máquina $[i]$ es intercambiada con cada tarea asignada a otra máquina $[l]$, $i < l$, y las dos secuencias $S_{[i]}$ y $S_{[l]}$ se optimizan usando Or-opt.

El procedimiento VND que utiliza sub-vecindarios $N^i_{com\_ins}(x)$ lo llamaremos *com_ins*-VND, mientras que al otro que utiliza sub-vecindarios $N^i_{com\_int}(x)$ lo denotaremos como *com_int*-VND.

La Figura 3.7 muestra un pseudo-código para el procedimiento de mejora: Etapa II. En primer lugar, se aplica *com_ins*-VND (línea 2). Cuando no se encuentran

más mejoras, se aplica *com_int*-VND (línea 3). Los procedimientos de las líneas 3 y 4 se repiten hasta que uno de estos procedimientos no mejora su solución de entrada.

Data: Solución $x = (S_1, S_2, \ldots, S_m)$.
1 **repeat**
2 | $x \leftarrow$ Encuentra un óptimo local con *com_ins*-VND (x);
3 | $x \leftarrow$ Encuentra un óptimo local con *com_int*-VND (x);
4 **until** *intercambio compuesto no mejore la solución*
Result: Solución $x = (S_1, S_2, \ldots, S_m)$.

Figura 3.7: Pseudo-código para el procedimiento de mejora: Etapa II (x).

3.3.2.3 Estrategia para lidiar con las estructura minmax del *makespan*

El *makespan* es una de las funciones objetivo más estudiadas en problemas de secuenciación. La estructura **mín máx{}** de ésta y otras funciones han propiciado el uso de funciones auxiliares para mejorar el desempeño de los algoritmos. Ya que se ha comprobado que si la búsqueda se basa solamente en comparar soluciones respecto a la función mín máx, esta es indiferente a las soluciones que no afectan el valor de la función mín máx, pero que mejoran la eficiencia total del algoritmo (Fleszar et al., 2012; Bhatia et al., 1977).

En transporte, Bhatia et al. (1977) abordan el problema de minimización del tiempo máximo de transporte $T_{\text{máx}}()$. Ellos utilizan el costo de transporte $C()$ como función auxiliar. Su algoritmo se mueve de una solución x a otra solución y, si $T_{\text{máx}}(x) < T_{\text{máx}}(y)$, o $T_{\text{máx}}(x) = T_{\text{máx}}(y)$ y $C(y) < C(x)$. Con base a lo anterior ellos desarrollan un algoritmo de solución exacto, con lo cual demostraron que su problema es polinomial, por lo que consideramos que se le debe dar la relevancia debida al uso de funciones auxiliares en problemas de secuenciación con la función objetivo del *makespan*.

En secuenciación, para el problema abordado en este capítulo, Vallada y Ruiz (2011) proponen un criterio de aceptación de movimientos que involucra dos máquinas. Al analizarlo se puede observar que se satisface que el *makespan* se reduce

o permanece con su mismo valor, pero la suma de los tiempos de terminación de las dos máquinas disminuye. Es decir, si x, $C_i(x)$, $C_l(x)$, y, $C_i(y)$ y $C_l(y)$ son la solución, el tiempo de terminación de la máquina i y el tiempo de terminación de la máquina l, antes y después del movimiento, respectivamente. Entonces $C_{max}(y) < C_{max}(x)$ o $C_{max}(y) = C_{max}(x)$ y $C_i(y)+C_l(y) < C_i(x)+C_l(x)$. Para el mismo problema, Fleszar et al. (2012) incluyen movimientos que involucran cualquier número de máquinas. Consideran el *makespan* C_{max} como función objetivo y la suma de los tiempos de terminación de las máquinas $\sum_{i\in M} C_i$ como función auxiliar, cuando el *makespan* no cambia.

En nuestro algoritmo, a lo largo de la búsqueda se aceptan los movimientos en relación con la siguiente estrategia que se explica a continuación.

Considere la posibilidad de un movimiento que involucra las máquinas i y l. Denotemos por O_i y O_i^{mov} el *span* de la máquina i antes y después si el movimiento *mov* se ejecutara. Aceptamos el movimiento si se disminuye el *makespan* restringido a las máquinas involucradas, es decir,

$$\text{máx}\{O_i^{mov}, O_l^{mov}\} < \text{máx}\{O_i, O_l\}. \tag{3.22}$$

Entre varios criterios analizados, este criterio de aceptación mostró los mejores resultados de rendimiento en pruebas experimentales. Nótese que implícitamente se satisface que un movimiento aceptable no empeora el *makespan*. Además, puede generalizarse fácilmente a movimientos que involucran más de dos máquinas.

3.3.2.4 Detalles de la implementación

Un pseudo-código genérico para nuestros procedimientos VND se muestra en la Figura 3.8. En este pseudo-código la palabra *type* se refiere al tipo de movimiento que se utiliza en cada procedimiento VND.

Vamos a definir el valor de un movimiento *mov* como

$$valor(mov) = \text{máx}\{O_i, O_l\} - \text{máx}\{O_i^{mov}, O_l^{mov}\},$$

Data: Solución $x = (S_1, S_2, \ldots, S_m)$.
$L_{maq} \leftarrow$ Lista de máquinas ordenadas de manera no-creciente por *span*;
$i \leftarrow 1$,
while $i \leq m$ **do**
  $[i] \leftarrow$ La $i-$ésima máquina en L_{maq}
  $mejor\_mov \leftarrow$ El mejor movimiento en el sub-vecindario $N^i_{type}(x)$.
  if $valor(mejor\_mov) > 0$ **then**
    $X \leftarrow$ Aplicar $mejor\_mov$ a x,
    Actualizar L_{maq}, $\quad i \leftarrow 1$,
  end
  else $i \leftarrow i + 1$,
end
Result: Solución $x = (S_1, S_2, \ldots, S_m)$.

Figura 3.8: Pseudo-código para el tipo de procedimiento *type*-VND (x).

o de forma equivalente como

$$valor(mov) = \text{mín}\{ \quad \text{máx}\{O_i, O_l\} - O_i^{mov}, \\ \text{máx}\{O_i, O_l\} - O_l^{mov}\}.$$

Decimos que un movimiento es aceptable si su valor es positivo. Por otra parte, decimos que $mov2$ es mejor que $mov1$ si $valor(mov2) > valor(mov1)$.

Para no explorar toda la vecindad y reducir el número de cálculos, primero inicializamos la expresión $valor(mejor\_mov) = 0$, luego evaluamos a un vecino verificando la siguiente desigualdad

$$a = \text{máx}\{O_i, O_l\} - O_i^{mov} > valor(mejor\_mov).$$

Si no se cumple, descartamos el vecino sin más cálculos, pero si se cumple, verificamos la segunda desigualdad

$$b = \text{máx}\{O_i, O_l\} - O_l^{mov} > valor(mejor\_mov).$$

Si esta no se cumple, descartamos la solución vecina. De lo contrario, actualizamos el valor de $mejor\_mov$ como $valor(mejor\_mov) = \text{mín}\{a, b\}$ y exploramos un nuevo vecino.

Para reducir aún más las operaciones, re-evaluamos los spans de las máquinas i y l a través del cálculo de los cambios que se realizarían si el movimiento se ejecutara. Es decir,

$$O'_i = O_i + \delta_i$$

y

$$O'_l = O_l + \delta_l,$$

donde δ_i y δ_l, se refieren a qué tanto el *span* de las máquinas i y l cambiaría si se realizara el movimiento. Recordemos que se definieron los movimientos en relación con los vecindarios establecidos en cada etapa.

En resumen, nuestro algoritmo multi-arranque puede ser descrito a través del pseudo-código presentado en la Figura 3.9.

```
   Data: Instancia del problema.
1  x* ← ∅;
2  while Criterio de parada no se satisfaga do
3  |   x ← Constructivo ();
4  |   x ← Mejora: Etapa I (x);
5  |   x ← Mejora: Etapa II (x);
6  |   if x es mejor que x* then
7  |   |   x* ← x;
8  |   end
9  end
   Result: Solución x*.
```

Figura 3.9: Pseudo-código para el algoritmo multi-arraque propuesto.

3.4 RESULTADOS COMPUTACIONALES

En esta sección primeramente mostramos los resultados obtenidos cuando comparamos las 7 formulaciones propuestas. También describimos los experimentos llevados a cabo para evaluar el desempeño del algoritmo propuesto.

Los experimentos fueron realizados en una PC Pentium Dual Core con procesador de 2.00 GHz y 3 GB RAM, bajo el sistema operativo Ubuntu 12.04.

3.4.1 Instancias

Nosotros usamos tres tamaños de instancias: pequeñas, medianas y grandes. Las instancias pequeñas y grandes, creadas por Vallada y Ruiz (2011), están disponibles en http://soa.iti.es. Adicionalmente, generamos instancias medianas para conocer el alcance de los modelos propuestos en este capítulo.

En el conjunto de instancias pequeñas hay cuatro grupos de número de máquinas (2, 3, 4, 5) y cuatro grupos de número de tareas (6, 8, 10, 12). Los tiempos de preparación son uniformemente distribuidos en cuatro intervalos: 1-9, 1-49, 1-99 y 1-124. Los tiempos de procesamiento estan uniformemente distribuidos entre 1-99. Para las instancias grandes hay cinco grupos con número de máquinas (10, 15, 20, 25, 30) y cinco grupos con número de tareas (50, 100, 150, 200, 250). Hay 10 réplicas para cada posible combinación de números de máquinas, números de tareas y rangos de tiempos de preparación, obteniendo un total de 640 instances pequeñas y 1000 instancias grandes.

Nosotros generamos las instancias medianas en una forma similar. Consideramos las siguientes combinaciones de números de tareas y números de máquinas: $n = \{20, 30, 40, 50, 60\}$ y $m = \{2, 3, 4, 5\}$. Los tiempos de preparación fueron uniformemente distribuidos en tres rangos: 1-49, 1-99 y 1-124. Los tiempos de procesamiento fueron uniformemente distribuidos entre 1-99. Generamos 10 réplicas por cada combinación, dando un total de 600 instancias medianas.

3.4.2 Comparando las formulaciones

Todas las formulaciones fueron implementadas usando *Concert Technology* de CPLEX 12.2, al cual se le dio un tiempo límite de ejecución de 3600 segundos. Si el optimizador no fue capaz de alcanzar la solución óptima en dicho tiempo, entonces se reportó la mejor solución entera obtenida.

Todas las instancias fueron agrupadas de acuerdo al número de tareas y número de máquinas. Por lo cual, los resultados son promediados sobre todas las instancias

pertenecientes a cada grupo, es decir, 40 en grupos de instancias pequeñas y 30 en grupos de instancias medianas.

La tabla 3.1 muestra resultados comparando las formulaciones usando diferentes linealizaciones para el *makespan*, es decir, Model1, Model2, Model1a, Model2a, Model1b, Model2b y Model3. Las columnas 1 y 2 se refieren al tamaño de la instancia. Las entradas en las columnas 3 y 6 (Uns) exiben cuantas instancias de cada grupo alcanzaron el tiempo límite; las entradas en las columnas 4 y 7 (GAP) muestran el gap promedio, mientras que las columnas 5 y 8 (tiempo) exponen el tiempo promedio computacional consumido (en segundos), para el Model1 y Model2, respectivamente. Dado que Model1a, Model2a, Model1b, Model2b y Model3 resolvieron a optimalidad todas las instancias, solo se reportan en las columnas 9 a la 13, los tiempos computacionales consumidos por ellos para alcanzar soluciones óptimas.

Los GAP's presentados en la tabla 3.1 son el promedio de los gap's obtenidos del optimizador y son calculados como

$$gap = 100 * \frac{best\_obj\_int - best\_lower\_bound}{best\_obj\_int},$$

donde *best_obj_int* es el valor de la función objetivo de la mejor solución factible encontrada y el *best_low_bound* es la mejor cota inferior obtenida.

Tabla 3.1: Desempeño de las formulaciones en instancias pequeñas.

n	m	Model1			Model2			Model1a	Model2a	Model1b	Model2b	Model3
		Uns	GAP	tiempo	Uns	GAP	tiempo	tiempo	tiempo	tiempo	tiempo	tiempo
6	2	0	0	0.52	0	0	0.41	0.10	0.07	0.06	**0.05**	0.28
	3	0	0	0.28	0	0	0.32	0.21	0.16	0.10	**0.08**	0.48
	4	0	0	0.29	0	0	0.30	0.21	0.21	0.11	**0.09**	0.24
	5	0	0	0.31	0	0	0.28	0.19	0.23	0.09	**0.08**	0.21
8	2	0	0	13.47	0	0	10.45	0.24	0.12	0.16	**0.10**	1.08
	3	0	0	5.87	0	0	5.38	0.38	0.26	0.30	**0.17**	2.06
	4	0	0	1.71	0	0	2.05	0.51	0.39	0.29	**0.18**	1.62
	5	0	0	1.14	0	0	1.23	0.45	0.44	0.18	**0.17**	1.36
10	2	7	4.25	1499.81	3	1.36	1124.72	0.46	0.23	0.37	**0.22**	3.44
	3	0	0	167.00	0	0	133.34	0.77	0.39	0.48	**0.38**	6.71
	4	0	0	26.89	0	0	31.94	1.00	0.62	0.63	**0.44**	15.56
	5	0	0	7.8	0	0	11.1	0.87	0.81	0.46	**0.36**	8.9
12	2	38	49.45	3549.48	38	47.10	3532.12	0.84	**0.35**	1.09	0.36	15.72
	3	29	18.29	3094.16	28	17.87	2990.70	1.37	0.57	1.00	**0.52**	31.26
	4	3	1.83	796.61	4	1.12	799.23	2.68	1.01	2.61	**0.87**	101.82
	5	0	0	91.45	0	0	112.00	2.18	1.22	1.76	**0.71**	82.48

Note que los modelos que usan la linealización típica para el *makespan*, Model1 y Model2 no son capaces de resolver la mayoría de las instancias de 12 tar-

eas, mostrando además un tiempo computacional y gap grandes. Por otro lado, los modelos que incluyen la linealización propuesta resuelven a optimalidad todas las instancias, Model1a, Model1b, Model2a y Model2b consumiendo menos de tres segundos y Model3 menos de 102 segundos en promedio. En particular, el que muestra mejor desempeño es Model2b, el cual incluye variables de asignación y utiliza ambas linealizaciones para el *makespan*.

La tabla 3.2 muestra el desempeño del modelo Model2b en instances medianas. Como antes, las columnas 1 y 2 se refieren al tamaño de la instancia, las entradas en la columna 3 (Uns) muestra el número de instancias que no fueron resueltas a optimalidad por cada grupo, mientras que la columna 4 (GAP) muestra el gap promedio. La columna 5 reporta el tiempo computacional promedio consumido (en segundos).

Tabla 3.2: Desempeño del modelo Model2b en instancias medianas.

n	m	Model2b		
		Uns	GAP	tiempo
20	2	0	0	1.25
	3	0	0	3.25
	4	0	0	10.96
	5	0	0	43.3
30	2	0	0	4.19
	3	0	0	19.07
	4	0	0	165.15
	5	0	0	460.14
40	2	0	0	12.74
	3	0	0	79.4
	4	0	0	589.9
	5	3	0.2	1730.5
50	2	0	0	44.11
	3	0	0	332.87
	4	4	0.17	1925.3
	5	20	2.27	3187.95
60	2	0	0	111.7
	3	1	0.02	1171.5
	4	9	0.49	2765.61
	5	28	3.58	3580.46

Como se puede observar Model2b es capaz de resolver a optimalidad todas

las instancias de 20 y 30 tareas, 97.5 % de las instancias de 40 tareas, 80 % de las instancias de 50 tareas, 68.33 % de las instancias de 60 tareas.

3.4.3 Evaluando el desempeño del algoritmo propuesto

Realizamos dos experimentos para evaluar el desempeño del algoritmo propuesto: una comparación con soluciones óptimas y/o mejores soluciones obtenidas de Model2b para instancias medianas, y una comparación con soluciones de la literatura para instancias grandes.

El algoritmo fue implementado en el lenguaje C++. El criterio de parada del algoritmo consiste en permitir un tiempo de cómputo máximo. Después de calibración preliminar, nosotros lo establecimos como $200 * n$ milisegundos.

Como se mencionó anteriormente, en el procedimiento de mejora etapa I y etapa II los movimientos entre y dentro de las máquinas son usados de diferentes maneras. En la primera etapa, primero, se aplican movimientos entre máquinas, luego en cada máquina se aplica búsqueda local con movimientos dentro de la máquina. En la seguna etapa, se usarán movimientos compuestos. Cada movimiento compuesto usa un movimiento entre máquinas y aplica a las máquinas involucradas una búsqueda local para mejorar su secuencia. Por lo tanto, para evaluar el desempeño de cada etapa nosotros probamos tres variantes del algoritmo:

- MAI: sólo se aplica la primer etapa del procedimiento de mejora.
- MAII: sólo se aplica la segunda etapa del procedimiento de mejora.
- MAIII: se aplican ambas etapas del procedimieto de mejora.

Comparación con soluciones exactas para instancias medianas. No reportamos resultados concernientes a instancias pequeñas, ya que las tres versiones del algoritmo propuesto obtienen soluciones óptimas para todas las instancias, consumiendo menos de un milisegundo en promedio por instancia.

La tabla 3.3 muestra la comparación entre las tres versiones del algoritmo multiarranque con las soluciones óptimas y/o mejores obtenidas usando Model2b en instancias medianas. Las tres versiones se ejecutaron con el mismo tiempo límite. Las columnas 1 y 2 se refieren al tamaño de la instancia. Las columnas 3, 4, 5 y 6 reportan la desviación relativa para Model2b, MAI, MAII y MAIII, respectivamente, a la mejor solución encontrada.

Tabla 3.3: Desviación relativa entre Model2b y las tres versiones del algoritmo multiarranque en instancias medianas.

n	m	Desviación relativa			
		Model2b	MAI	MAII	MAIII
20	2	0	0	0	0
	3	0	0	**0.11**	0.12
	4	0	0.17	0.04	**0.02**
	5	0	0.11	**0**	**0**
30	2	0	0.56	**0.27**	0.32
	3	0	0.71	0.5	**0.4**
	4	0	1.06	0.89	**0.58**
	5	0	2.19	0.77	**0.39**
40	2	0	1.9	**1.27**	1.31
	3	0	2.58	**1.14**	1.45
	4	0	3.7	**1.62**	1.78
	5	0	4.3	2.5	**1.74**
50	2	0	3.25	1.9	**1.67**
	3	0	4.63	2.61	**2.46**
	4	0	5.51	3.16	**2.61**
	5	0	5.69	2.86	**2.56**
60	2	0	4.49	2.87	**2.65**
	3	0	5.86	2.98	**2.64**
	4	0	6.52	3.38	**3.27**
	5	0.12	6.74	3.05	**2.57**

De la tabla 3.3 nosotros observamos que las dos versiones que incluyen la etapa II (movimientos compuestos) obtienen mejores valores. Esto indica que es preferible incluir la etapa II en lugar de asignar todo el tiempo computacional permitido a la etapa I. Más aún, la variante que incluye ambas etapas alcanza mejores resultados cuando el tamaño de la instancia crece.

Comparación con resultados de la literatura para instancias grandes

Dado que para estas instancias no se conocen soluciones óptimas, presentamos una comparación entre soluciones obtenidas por las tres versiones de nuestro algoritmo y los mejores resultados conocidos de la literatura, especificamente el algoritmo genético (GA) propuesto por Vallada y Ruiz (2011). La búsqueda tabú de Helal et al. (2006) y el Meta-RaPS de Rabadi et al. (2006), no son comparados dado que el GA obtiene mejores resultados para estas instancias. En la tabla 3.4, las columnas 3, 4 y 5 muestran la desviación relativa promedio del algoritmo genético y las tres versiones del algoritmo multiarranque propuesto. El tiempo límite para el GA se muestra en la columna 7, mientras que los tiempos límites para MAI, MAII y MAIII, son mostrados en la columna 8 (MA's), para cada combinación de m y n.

Note que MAIII se desempeña mejor que el GA en 83.5 % de las instancias, tiene el mismo comportamiento en 6.2 % y solo en 10.3 % de las instancias obtiene peores resultados. Más aún, con las variantes probadas de nuestro algoritmo, encontramos 871 nuevas mejores soluciones para las 1000 instancias generadas por Vallada y Ruiz (2011).

Los resultados en la tabla 3.4 confirman que las variantes que incluyen ambas etapas alcanzan mejores resultados para instancias grandes. Nosotros observamos que a pesar que obtenemos buenos resultados usando solo la etapa II, estos son mejorados cuando la etapa II inicia de buenas soluciones proporcionadas por la etapa I.

3.5 Conclusiones del capítulo

En este capítulo discutimos el problema de secuenciación en máquinas paralelas no relacionadas con tiempos de preparación dependientes de la máquina y de la secuencia. Propusimos varias reformulaciones del problema para minimizar el *makespan*. La principal contribución de estas reformulaciones consistió en linealizar el *makespan* como el máximo de los tiempos de terminación de las máquinas. Esta

Tabla 3.4: Comparación entre las tres versiones del algoritmo Multi-arranque y resultados publicados sobre instancias grandes.

n	m	Desviación relativa				Tiempo (seg.)	
		GA	MAI	MAII	MAIII	GA	MA's
50	10	2.05	3.28	2.75	**1.75**	12.5	10
	15	4.71	1.77	1.47	**0.85**	18.8	10
	20	7.41	2.47	2.83	**1.26**	25	10
	25	8.63	2.20	2.43	**2.07**	31.3	10
	30	7.19	2.22	2.95	**1.40**	37.5	10
100	10	2.97	7.60	2.01	**1.43**	25	20
	15	4.59	6.58	1.69	**1.02**	37.5	20
	20	5.90	5.00	2.10	**1.08**	50	20
	25	8.94	4.04	1.60	**1.21**	62.5	20
	30	9.98	3.47	1.91	**0.98**	75	20
150	10	4.47	8.80	1.77	**0.73**	37.5	30
	15	6.21	9.80	1.92	**1.04**	56.3	30
	20	7.64	7.80	1.87	**0.99**	75	30
	25	10.56	6.86	1.87	**0.98**	93.8	30
	30	10.62	4.28	1.64	**0.30**	112.5	30
200	10	2.77	5.62	4.27	**0.99**	50	40
	15	7.59	9.57	3.12	**0.70**	75	40
	20	10.9	10.28	1.88	**0.97**	100	40
	25	13.3	8.98	2.27	**1.21**	125	40
	30	13.82	8.17	2.55	**0.89**	150	40
250	10	**1.22**	3.99	22.81	2.91	62.5	50
	15	6.62	7.59	14.16	**0.81**	93.8	50
	20	10.5	10.01	5.58	**0.52**	125	50
	25	15.52	11.07	2.35	**0.89**	156.3	50
	30	15.44	8.83	2.69	**0.94**	187.5	50
Promedio		7.98	6.41	3.7	**1.12**	75.01	30

linalización proporcionó cotas duales mejoradas las cuales aceleraron el proceso de solución al usar un optimizador comercial de ramificación y acotamiento.

Los experimentos computacionales indicaron la superioridad de los modelos que incluyen la linealización propuesta para el *makespan*, dichos modelos resolvieron a optimalidad todas las instancias pequeñas. Más aún el modelo que incluye variables de asignación, así como la linealización típica y la propuesta para el *makespan* obtuvo los mejores resultados. Éste resolvió a optimalidad instancias hasta seis veces más grandes que las otras formulaciones usando el mismo tiempo computacional y

obtuvo soluciones óptimas en instancias del mismo tamaño hasta cuatro órdenes de magnitud más rápido. Para el conjunto de instancias medianas, variando de 20 a 60 tareas, Model2b pudo resolver la mayoría de las instancias. El gap máximo para las instancias no resueltas fue de 3.58 %. Por lo tanto, este modelo podría ayudar a extender el tamaño de instancias que pueden ser resueltas por métodos exactos desarrollados para este problema. Adicionalmente, la forma de linealizar el *makespan* puede ser usada en otros problemas de secuenciación que consideran la misma función objetivo.

Nosotros también propusimos un algoritmo multiarranque que incluye dos etapas en su fase de mejora. Ambas etapas incluyen movimientos entre y dentro de las máquinas, pero son usados en diferentes formas en cada etapa. El algoritmo toma ventaja de la rapidez de búsqueda en la primer etapa y la interdependencia entre los subproblemas de asignación y secuenciación en la segunda etapa. La subdivisión de los vecindarios y el criterio de aceptación propuestos permiten lidiar con la estructura de la función objetivo y reducir el tiempo computacional para explorar cada vecindario.

En experimentos computacionales, MAIII mostró pequeñas desviaciones respecto a soluciones óptimas en un conjunto de 600 instancias medianas y su superioridad sobre los mejores resultados conocidos de la literatura en un conjunto de 1000 instancias grandes. Los resultados reflejaron el buen desempeño de los movimientos compuestos y la conveniencia de usar ambas etapas del procedimiento de mejora.

Capítulo 4

El problema $R, h_{ik}|S_{ijk}|C_{max}$

En este capítulo se aborda el problema de secuenciación de tareas en máquinas paralelas no relacionadas con tiempos de preparación de la máquina dependientes de la máquina y de la secuencia, y un programa de mantenimiento periódico para cada máquina minimizando el *makespan*. También nos referiremos a éste como el segundo problema.

4.1 Introducción

Las compañías generalmente utilizan diferentes tipos de máquinas para procesar sus productos y cada máquina pierde confiabilidad en el sentido que se degrada con la edad, hasta que llega el momento que falla (Blischke y Murthy, 2011). Las operaciones de mantenimiento pueden ser clasificadas en dos grandes grupos: mantenimiento correctivo (MC) y mantenimiento preventivo (MP). El MC toma lugar cuando las fallas ya ocurrieron. El MP se da para mantener el sistema con un nivel de operación deseable. Una variedad de modelos se han desarrollado para determinar periodos óptimos de MP (McCall, 1965; Pierskalla y Voelker, 1976). Por lo tanto pueden definirse diversas políticas con la intención de determinar cuándo es necesario realizar operaciones de mantenimiento preventivo en las máquinas. Definimos a continuación las siguientes tres políticas:

Política I mantenimiento preventivo en intervalos de tiempos predefinidos fijos.

Política II modelo de periodo óptimo para el mantenimiento preventivo maximizando la disponibilidad de las máquinas.

Política III realizar el mantenimiento de manera que se satisfaga un umbral de confiabilidad mínima para un periodo de producción dado.

La política usada más ampliamente en la industria es la política I. En ésta las operaciones son planeadas con anterioridad en tiempos fijos sin considerar modelos probabilísticos para el tiempo de falla, y se hace el mejor uso de las paradas programadas después de periodos de producción cíclicos: semanal, mensual o incluso anual.

En este capítulo estudiamos el problema de integrar secuenciación de la producción con mantenimiento preventivo en un ambiente de producción de máquinas paralelas no relacionadas donde existen tiempos de preparación dependientes y el objetivo a minimizar es el *makespan*. En este problema, usaremos la Política I de mantenimiento, es decir, donde las actividades de mantenimiento se desarrollarán en cada máquina al final de intervalos de tiempo predefinidos fijo, independientemente de la cantidad de tiempo que la máquina estuvo en operación en ese intervalo.

4.2 Descripción del problema

Las siguientes suposiciones son usadas para describir el problema:

- Hay un conjunto M de m máquinas paralelas no relacionadas.
- Las máquinas están siempre disponibles. Cada máquina debe procesar una tarea a la vez sin interrupción, es decir, una vez que inicia el procesamiento de una tarea, esta no puede ser interrumpida.
- Hay un conjunto N de n tareas para ser secuenciadas.
- Todos las tareas están disponibles al tiempo cero. No existen restricciones de precedencia sobre el orden de procesamiento de las tareas.
- Cada tarea j tiene asociada un tiempo de procesamiento p_{ij} que depende de la máquina i en la que sea procesada.

- Hay un tiempo de preparación s_{ijk} de la máquina i, requerido para procesar la tarea k justo después de procesar la tarea j.
- Entre dos actividades de mantenimiento consecutivas en la máquina i existe un intervalo de tiempo T_i.
- Para poder comenzar a procesar una tarea específica en la máquina i, se debe de disponer de tiempo suficiente hasta la siguiente actividad de mantenimiento, para preparar dicha máquina y procesar la tarea.
- Cada actividad de mantenimiento en la máquina i, denotada con índice 0, consume una cantidad de tiempo fija p_{i0} y requiere de un tiempo de preparación s_{ij0} que depende de la última tarea j que se procesó en la máquina.
- Cuando se termina una actividad de mantenimiento a la máquina i, existe un tiempo de preparación s_{i0k} para cada tarea k.
- El objetivo es minimizar el *makespan* C_{max}.

La Figura 4.1 muestra una representación gráfica de una solución al problema en cuestión, para un problema de 17 tareas y tres máquinas. Los cuadros grises representan los tiempos de preparación y los espacios en blanco representan tiempos de ocio antes de cada tarea de mantenimiento.

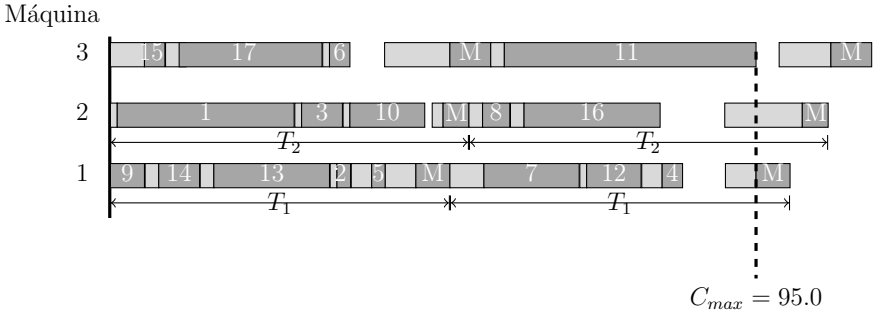


Figura 4.1: Representación gráfica de una solución.

Primero, proponemos dos formulaciones enteras mixtas, Model1-M y Model2-M, para formalizar el problema. Ambos modelos incluyen la linealización del

makespan presentada en el capítulo 3. Después, proponemos desigualdades válidas que mejoran la representación del *span* de cada máquina.

Definimos un bloque como el intervalo de tiempo entre dos actividades de mantenimiento consecutivas. Programar un bloque significará por consiguiente asignar y secuenciar las tareas de dicho bloque.

Note que dado que los intervalos de tiempo entre mantenimientos son fijos, para el cálculo de la función objetivo basta considerar el número de bloques sin contar el último y el tiempo consumido por las tareas secuenciadas en el último bloque. Por ello es necesario distinguir el último bloque de los demás. Denotemos por H el conjunto compuesto por $\{1, 2\}$ que nos permitirá separar el último bloque de una máquina (asociado al índice 1) de los demás bloques de la misma máquina (asociados al índice 2).

Utilizaremos las siguientes variables en el modelo:

$$x^1_{ijk} = \begin{cases} 1, & \text{si se procesa la tarea } k \text{ justo después de procesar la tarea } j \text{ en} \\ & \text{el último bloque de la máquina } i, \\ 0, & \text{en otro caso.} \end{cases}$$

$$x^2_{ijk} = \begin{cases} 1, & \text{si se procesa la tarea } k \text{ justo después de procesar la tarea } j \text{ en} \\ & \text{la máquina } i \text{ en cualquier bloque excluyendo el último,} \\ 0, & \text{en otro caso.} \end{cases}$$

b_i el número de bloques en la máquina i.

u_{ij} el tiempo transcurrido entre la terminación de la tarea j y la tarea de mantenimiento previa, si j se procesa en la máquina i.

Denotaremos por N_0 el conjunto N más una tarea de mantenimiento con índice 0 y V denotará una constante muy grande. Las variables x_{i0k} y x_{ij0} son usadas para especificar cuáles tareas k y j serán la primera y última procesadas en cada máquina.

El Model1-M puede establecerse como

$$Minimizar \quad C_{max} \tag{4.1}$$

$$Sujeto \quad a:$$

$$\sum_{h\in H}\sum_{i\in M}\sum_{\substack{j\in N_0\\ j\neq k}} x^h_{ijk} = 1, \qquad \forall k \in N, \tag{4.2}$$

$$\sum_{h\in H}\sum_{i\in M}\sum_{\substack{k\in N_0\\ j\neq k}} x^h_{ijk} = 1, \qquad \forall j \in N, \tag{4.3}$$

$$\sum_{\substack{k\in N_0\\ k\neq j}} x^h_{ikj} - \sum_{\substack{k\in N_0\\ k\neq j}} x^h_{ijk} = 0, \qquad \forall j \in N_0, \forall h \in H, \forall i \in M \tag{4.4}$$

$$\sum_{k\in N} x^1_{i0k} \leq 1, \qquad \forall i \in M, \tag{4.5}$$

$$\sum_{k\in N} x^2_{i0k} = (b_i - 1), \qquad \forall i \in M, \tag{4.6}$$

$$u_{ij} - u_{ik} + (T_i + s_{ijk} + p_{ik}) * x^h_{ijk} \leq T_i, \qquad \forall j \in N_0, \forall k \in N, \forall i \in M, \tag{4.7}$$

$$u_{ij} \leq T_i - (s_{ij0} + p_{i0})x^h_{ij0}, \qquad \forall j \in N, \forall h \in H, \forall i \in M \tag{4.8}$$

$$u_{i0} = 0, \qquad \forall i \in M, \tag{4.9}$$

$$T_i(b_i - 1) + \sum_{\substack{j\in N_0\\ j\neq k}}\sum_{k\in N}(s_{ijk} + p_{ik})x^1_{ijk} \leq C_{max}, \qquad \forall i \in M, \tag{4.10}$$

$$x^1_{ijk}, x^2_{ijk}\{0, 1\} \qquad \forall j \in N_0, \forall k \in N_0, \forall i \in M, \tag{4.11}$$

Las restricciones (4.2) establecen que cada tarea tiene solo una tarea que le precede. Las restricciones (4.3) establecen que cada tarea tiene una tarea que le sucede en el orden. Las restricciones (4.4) establecen que si una tarea es secuenciada en un cierto bloque de una máquina, la tarea que le sucede también sea secuenciada en el mismo bloque. Las restricciones (4.5) dicen que el último bloque de cada máquina inicia con a lo más una tarea. Las restricciones (4.6) contabilizan el número de bloques en cada máquina exceptuando el último. Las restricciones (4.7) establecen un orden entre las tareas asignadas a un mismo bloque. Las restricciones (4.8) establecen cotas superiores para el tiempo transcurrido desde la terminación del último

mantenimiento y el tiempo de terminación de cada tarea. Las restricciones (4.9) inicializan las variables u_{ij} asociadas a las tareas de mantenimiento. Las restricciones (4.10) linealizan la función objetivo como el máximo de los tiempos de terminación de las máquinas. Las restricciones (4.11) establecen la naturaleza de las variables de decisión.

Para la segunda formulación (denominado Model2-M), además de las variables empleadas en Model1-M, introducimos las siguientes variables:

$$y_{ik}^1 = \begin{cases} 1, & \text{si una tarea } k \text{ se asigna al último bloque de la máquina } i, \\ 0, & \text{en otro caso.} \end{cases}$$

$$y_{ik}^2 = \begin{cases} 1, & \text{si una tarea } k \text{ se asigna a la máquina } i, \text{ pero no en el último bloque,} \\ 0, & \text{en otro caso.} \end{cases}$$

Model2-M puede ser obtenido de Model1-M reemplazando las restricciones (4.2)-(4.4) por:

$$\sum_{h \in H} \sum_{i \in M} y_{ik}^h = 1, \qquad \forall j \in N, \tag{4.12}$$

$$\sum_{\substack{j \in N_0 \\ j \neq k}} x_{ijk}^h = y_{ik}^h, \qquad \forall k \in N, \forall h \in H, \forall i \in M, \tag{4.13}$$

$$\sum_{\substack{k \in N_0 \\ k \neq j}} x_{ijk}^h = y_{ij}^h, \qquad \forall j \in N, \forall h \in H, \forall i \in M. \tag{4.14}$$

Las restricciones (4.12) aseguran que cada tarea es asignada a exactamente un bloque de una máquina. Las restricciones (4.13) establecen que cada tarea tiene exactamente un predecesor y ambos, tarea y predecesor, son asignados al mismo bloque de la misma máquina. Las restricciones (4.14) garantizan que cada tarea tiene exactamente un sucesor y ambas son asignadas al mismo bloque.

Model1-M tiene $2n^2m + 2nm$ variables binarias, $nm + 2m$ variables continuas y $n^2m + 5nm + 2n + 6m$ restricciones, mientras que Model2-M tiene $2n^2m + 2nm$ variables binarias, $2nm+3m$ variables continuas y $n^2m+7nm+n+5m$ restricciones.

Note que el *makespan* está linealizado en términos de los *span* de las máquinas. A pesar de ello, al resolver la relajación lineal de ambos modelos observamos lo siguiente: las variables x^2_{i0k} son cero para cada posible valor de i y de k, lo cual implica que b_i sea cero. De la misma manera, las variables x^1_{ijk} toman el valor cero para cada posible valor de i, j y k. Las únicas variables que toman valores positivos son x^2_{ijk}, para cada posible valor de i, j, $j \neq 0$ y $k \in N$. De lo anterior se tiene que, a pesar de que todas las tareas son asignadas o secuenciadas en los primeros bloques de las máquinas, el valor de la función objetivo en la relajación lineal es cero ($C_{max} = 0$).

Tomando en cuenta lo observado, decidimos acotar inferiormente el valor de la variable b_i y superiormente el nivel de ocupación del último bloque de cada máquina. Las primeras cotas incluyen todas las variables que antes tenían valores positivos. En ambas cotas se suma el tiempo de procesamiento y tiempo de preparación de las tareas secuenciadas total o parcialmente. Las cotas se agregan como desigualdades válidas al modelo.

$$\sum_{\substack{j\in N_0\\ j\neq k}} \sum_{k\in N_0} (s_{ijk} + s_{ik}) x^2_{ijk} \leq T_i(b_i - 1), \qquad \forall i \in M, \tag{4.15}$$

$$\sum_{\substack{j\in N_0\\ j\neq k}} \sum_{k\in N_0} (s_{ijk} + s_{ik}) x^1_{ijk} \leq T_i, \qquad \forall i \in M. \tag{4.16}$$

Además, usando las variables de asignación y^l_{ik}, las desigualdades válidas (4.15) y (4.16) pueden ser reescritas como

$$\sum_{\substack{j\in N_0\\ j\neq k}} \sum_{k\in N_0} s_{ijk} * x^2_{ijk} + \sum_{k\in N_0} p_{ik} * y^2_{ik} \leq T_i(b_i - 1), \qquad \forall i \in M, \tag{4.17}$$

$$\sum_{\substack{j\in N_0\\ j\neq k}}\sum_{k\in N_0} s_{ijk} * x^1_{ijk} + \sum_{k\in N_0} p_{ik} * y^1_{ik} \leq T_i, \qquad \forall i \in M. \tag{4.18}$$

Finalmente, Model1-M queda conformado por las restricciones (4.1)-(4.11), (4.15)-(4.16), mientras que el Model2-M queda conformado por (4.1), (4.5)-(4.14), (4.17)-(4.18).

Dado que en el problema del capítulo anterior los mejores resultados se obtuvieron usando el modelo que emplea ambos tipos de variables, en este capítulo realizaremos experimentación solo con Model2-M. Por otro lado, como se mostrará en la sección de experimentos, usando dicho modelo solo se obtuvieron soluciones óptimas para instancias pequeñas, por lo que fue necesario desarrollar la metodología de solución que describimos a continuación.

4.3 Algoritmo metaheurístico propuesto

Al igual que en el capítulo anterior, nosotros proponemos una algoritmo multiarranque, dado que estos métodos proporcionan una estructura que aporta diversificación y aprovechan la "facilidad"de construir soluciones.

A continuación, describimos nuestra implementación de los procedimientos constructivo y de mejora para el problema abordado.

4.3.1 Procedimiento constructivo

La representación de la solución que utilizaremos para el problema de programación de máquinas paralelas con mantenimiento preventivo es un arreglo S^l_i para cada grupo l de tareas entre dos mantenimientos consecutivos en la máquina i, que representa el orden de procesamiento de las tareas asignadas a ese bloque en esa máquina, es decir, $x = \{S^1_1, \ldots, S^{b_1}_1; S^1_2, \ldots, S^{b_2}_2; \ldots; S^1_m, \ldots, S^{b_m}_m\}$, donde S^1_1 representa el orden de procesamiento de las tareas asignadas a la máquina uno, antes del primer mantenimiento, S^2_1 representa el orden de procesamiento de las tareas asignadas a la máquina uno, entre el primer y el segundo mantenimiento, y así sucesivamente.

Para este problema el nivel de ocupación de la máquina es evaluado como

$$O_i = (b_i - 1) * T_i + \text{mín}\{o_{i1}, o_{i2}, \ldots, o_{ib_i}\}, \tag{4.19}$$

donde definimos o_{il} como el tiempo efectivo de un bloque l, el cual es el tiempo transcurrido desde la última tarea de mantenimiento hasta el tiempo de terminación de la última tarea del bloque l en la máquina i y b_i denota el número de bloques usados en la máquina i.

La idea general es, dada una solución parcial, primero, se trata de insertar las tareas donde no se incremente el tiempo de terminacion de ninguna máquina, es decir, en las máquinas que tengan más de un bloque, se insertaran en bloques distintos de los últimos siempre y cuando sea factible. Segundo, si tal inserción no fue posible, se insertará donde el *span* de las máquinas aumente menos, es decir, en el último bloque de una máquina (si la máquina tiene al menos un bloque y es factible la inserción) o en un bloque nuevo.

El procedimiento para construir una solución inicial, mostrado en el pseudo-código de la Figura 4.2, funciona de la siguiente manera: En primer lugar, las tareas se enlistan en orden no-creciente en función de su tiempo de procesamiento promedio, es decir, $\hat{p_j} = \sum_{i \in M} p_{ij}/m$, y luego se forma una lista de candidatos con las primeras s tareas de la que se selecciona una al azar. Para la tarea seleccionada j se determina para cada máquina (que tenga más de un bloque) el mejor punto de inserción en un bloque distinto del último. En tal caso, el *span* de la máquina no cambia y el valor de la inserción se mide por el tiempo de ocio del bloque δ_{ij}. Entonces la tarea j se inserta en el bloque l^* en la máquina i^*, tal que $\{i^*, l^*\} = \arg\text{mín}\{\delta_{il}\}$. Cuando tal inserción no existe, se determina para cada máquina que tenga al menos un bloque, el mejor punto de inserción en el último bloque. Si la máquina no tiene ningún bloque o la inserción no es factible se inserta la tarea en un bloque nuevo, es decir, donde el *span* de la máquina aumente menos. Denotemos por Δ_i el aumento de la duración O_i de la máquina i para el mejor punto de inserción de la tarea j en esa máquina. Entonces la tarea j se inserta en la máquina $i^* = \arg\text{mín}\{O_i + \Delta_i\}$ en el mejor punto de inserción del último bloque en la máquina i^*. Luego, la lista de candidatos

se actualiza y el proceso se repite hasta que todas las tareas han sido asignadas.

Data: *Entrada.txt* instancia del problema.
Result: Solución $X = (S_1, S_2, ..., S_m)$.
foreach $k \in M$ **do** $b_k \leftarrow$ Secuencia de bloques vacía: $S_{b_k} \leftarrow$ Secuencia de tareas vacía:
$L_{tareas} \leftarrow$ Lista de los tareas ordenadas decrecientemente por el valor $\hat{p}_{ij}$;
$LC \leftarrow$ Lista candidata de las primeras s tareas en L_{tareas};
while $LC \neq \emptyset$ **do**
    $j \leftarrow$ Elegir aleatoriamente una tarea de LC;
    foreach $k \in M$ **do**
        Encontrar la mejor posición q_b^k para insertar j en S_b^k, $b \neq n_b$;
        Guardar q_{b_k} y el nuevo o_{b_k};
    end
    $k^* \leftarrow \arg\min_k T_k - o_{b_k}$
    if $(T_k * -o_{b_k}*) > 0$ **then**
        Insertar j en la secuencia S_{k^*} en la posición q_{k^*} Remover j de L_{tareas} y actualizar LC;
    else
        foreach $k \in M$ **do**
            Encontrar la mejor posición q_b^k para insertar j en S_b^k del último bloque;
            Guardar q_{b_k} y el nuevo o_{b_k};
        end
        $k^* \leftarrow \arg\min_k O_k$
    end
end

Figura 4.2: Pseudo-código para el procedimiento Constructivo().

4.3.2 Procedimiento de mejora propuesto

En el problema estudiado en este capítulo nosotros consideramos tres tipos de decisiones:

i) determinar la asignación de tareas a máquinas,

ii) determinar, para las tareas asignadas a una máquina, a qué bloques de esa máquina deben ser asignadas, y por último,

iii) determinar la secuencia de las tareas de cada bloque de cada máquina.

El procedimiento de mejora generaliza las mejores estrategias del capítulo 3, es decir, se propone aplicar primero movimientos para cada tipo de decisión y luego aplicar movimientos que relacionan diferentes tipos de decisiones.

El procedimiento de mejora consiste en dos etapas. En las dos etapas se utilizan movimientos entre bloques de distintas máquinas (tipo1), entre bloques de la misma máquina (tipo2) y movimientos dentro de los bloques (tipo3), pero se utilizan de forma diferente en cada etapa.

4.3.2.1 ESTRATEGIA PARA LIDIAR CON LA ESTRUCTURA DEL *makespan*

A lo largo de la búsqueda, los movimientos serán aceptados en relación con el criterio que se explica a continuación.

Si el movimiento es entre bloques de diferentes máquinas, considere la posibilidad de un movimiento que implica un bloque de la máquina i y un bloque de la máquina i'. Denotemos al *span* de la máquina i y al de la máquina i' antes de que se ejecute el movimiento por O_i y $O_i's$, respectivamente. Y después de que éste se ejecute por O_i^{mov} y $O_{i'}^{mov}$, respectivamente. En este caso aceptamos el movimiento si se disminuye el *makespan* restringido a las máquinas involucradas, es decir,

$$\text{máx}\{O_i^{mov}, O_{i'}^{mov}\} < \text{máx}\{O_i, O_{i'}\}. \tag{4.20}$$

Si el movimiento es entre bloques de una misma máquina i, denotemos o_{il}, $o_{il'}$, o_{il}^{mov} y $o_{il'}^{mov}$ el tiempo efectivo del bloque l, y el bloque l' antes de que se ejecute el movimiento y después de que éste se ejecute, respectivamente. En este caso aceptamos el movimiento si disminuye el mínimo de los tiempos efectivos de los bloques.

$$\text{mín}\{o_{il}^{mov}, o_{il'}^{mov}\} < \text{mín}\{o_{il}, o_{il'}\}. \tag{4.21}$$

Si el movimiento es dentro de un mismo bloque, aceptamos el movimiento si disminuye el tiempo de efectivo del bloque.

$$o_{il}^{mov} < o_{il}. \tag{4.22}$$

Con cualquiera de las tres ecuaciones anteriores se satisface implicitamente que el *makespan* nunca empeorará. Por lo que los movimientos son aceptables si:

- el *makespan* disminuye, o
- el *makespan* se mantiene con el mismo valor, pero se satisface el criterio que corresponda con el tipo de movimiento que se realice.

4.3.2.2 Procedimiento de mejora: Etapa I

Definimos a continuación tres diferentes procedimientos de búsqueda, uno por cada tipo de movimiento.

Primero definimos los k-intercambios entre dos secuencias arbitrarias de tareas. Éstos son la unión de tres movimientos. El primero representa la extracción de cada tarea de la primer secuencia para insertarla en cada posición de la segunda secuencia, el segundo, representa el intercambio de cada tarea de la primer secuencia con cada tarea de la segunda secuencia, y el tercero, representa la extracción de cada tarea de la segunda secuencia para insertarla en cada posible posición de la primer secuencia.

Procedimiento de búsqueda con movimientos tipo 1 Los movimientos entre-máquinas representan los movimiento de tipo 1 y consisten en probar los k-intercambios de cada bloque de cada máquina con cada bloque de las demás máquinas.

El vecindario de movimientos entre máquinas se divide en pequeños sub-vecindarios. Implementamos una búsqueda VND basada en dichos subvecindarios. Esto debido a que con la subdivisión se logra explorar en primer término la parte de la vecindad que disminuye el *makespan* y en segundo término la parte de vecindad que no disminuye el *makespan*, pero que abre espacio en las máquinas para que en iteraciones posteriores se pueda lograr una disminución del *makespan*.

Específicamente, el vecindario se ha dividido en sub-vecindarios, donde el primero en ser explorado es el que involucra a la máquina más ocupada. Es decir,

denotemos por L_{maq} la lista de índices de las máquinas ordenados de una manera no creciente por su valor de *span* y sean $[i]$ y $[l]$ las máquinas que ocupan la posición i y la posición l en L_{maq}, respectivamente. Entonces

- $N^i(x)$: sub-vecindario de movimientos entre máquinas donde se exploran los k-intercambios entre cada bloque de la máquina $[i]$ con cada bloque de las demás máquinas $[l]$, con $i < l$.

En este procedimiento de búsqueda se emplea la ecuación (4.20) y al igual que en el capítulo anterior vamos a definir el valor de un movimiento mov como

$$valor(mov) = \text{máx}\{O_i, O_i'\} - \text{máx}\{O_i^{mov}, O_{i'}^{mov}\},$$

por lo que decimos que un movimiento es aceptable si su valor es positivo y decimos que $mov2$ es mejor que $mov1$ si $valor(mov2) > valor(mov1)$.

El pseudo-código de este procedimiento se muestra en la Figura 4.3.

```
Data: Solución x = (S_1, S_2, ..., S_m).
1  L_maq ← Lista de máquinas ordenadas de manera no-creciente de acuerdo al
   span;
2  i ← 1,
3  while i ≤ m do
4  |  [i] ← La i-ésima máquina en L_maq
5  |  mejor_mov ← El mejor movimiento en el sub-vecindario N^i(x).
6  |  if valor(mejor_mov) > 0 then
7  |  |  x ← Aplicar mejor_mov a x,
8  |  |  Actualizar L_maq,    i ← 1,
9  |  end
10 |  else  i ← i + 1,
11 end
Result: Solución x = (S_1, S_2, ..., S_m).
```

Figura 4.3: Pseudo-código para el procedimiento Búsqueda1().

PROCEDIMIENTO DE BÚSQUEDA CON MOVIMIENTOS TIPO 2 Los movimientos entre-bloques representan los movimientos tipo 2 y consisten en probar los k-intercambios de cada bloque de una máquina dada con los demás bloques de la misma máquina.

Definimos un procedimiento de búsqueda donde solo es necesario verificar la ecuación (4.21), ya que el objetivo es minimizar el mmínimo tiempo efectivo entre cada par de bloques en una misma máquina. Al usar este objetivo se reduce el tiempo de terminación de la máquina o inclusive puede reducirse el número de bloques. El pseudo-código de este procedimiento se muestra en la Figura 4.4.

```
Data: El conjunto S_k de bloques de la máquina k.
Result: S_k.
repeat
    foreach S_k^{l1} ∈ S_k do
        foreach S_k^{l2} ∈ S_k, l2 > l1 do
            Calcular el costo de los k-intercambios entre los bloques l1 y l2
        end
    end
    if hubo mejora then
        Realizar movimiento.
    end
until no haya una mejora
```

Figura 4.4: Pseudo-código para el procedimiento Búsqueda2().

PROCEDIMIENTO DE BÚSQUEDA CON MOVIMIENTOS TIPO 3 La reinserción representa los movimientos tipo 3 y consisten en probar, para un bloque dado, reinserciones de cadenas de tareas consecutivas sobre cada posición posible del mismo bloque. Estos son los movimientos que usa el Or-Opt.

Con los movimientos tipo 3 definimos una búsqueda local que generaliza los movimientos del Or-opt, la cual iterativamente relocaliza cadenas de tamaño $1, 2, \ldots, nb$ donde el objetivo es minimizar el tiempo efectivo en ese bloque. Dicho de otra forma, la intención es tratar de encontrar la mejor secuencia de las tareas de cada bloque. Al optimizar la secuencia de cada bloque solo se verifica la ecuación (4.22). El pseudo-código de este procedimiento se muestra en la Figura 4.5.

Una vez definidos los movimientos y separado el proceso de búsqueda de acuerdo al tipo de movimiento el proceso de mejora de la etapa uno puede resumirse de la siguiente manera.

```
Data: S_k^l una secuencia del bloque l de la máquina k
Result: S_k^l
repeat
    for nb = 1, 2, 3 do
        foreach subsecuencia de "nb" tareas do
            foreach posición restante en la secuencia do
                Calcular el costo de reinsertar en la secuencia la subsecuencia de
                "nb" tareas a dicha posición;
                if se encontró mejora then guardar movimiento;
            end
        end
    end
    if se encontró mejora then
        Realizar movimiento;
    end
until no haya una mejora
```

Figura 4.5: Pseudo-código para el procedimiento Búsqueda3().

En el procedimiento de mejora: Etapa I, primeramente, aplicamos una búsqueda local basada en los movimientos entre bloques de distintas máquinas, luego aplicamos a cada máquina una búsqueda local basada en movimientos entre bloques de la misma máquina, y por último aplicamos a cada máquina una búsqueda local basada en movimientos dentro de cada bloque. Este proceso se repite hasta que dos procedimientos consecutivos no mejoren la solución. El pseudo-código general del proceso se muestra en la Figura 4.6.

```
Data: Solución x = (S_1, S_2, ..., S_m).
repeat
    x ← Búsqueda1 (x);
    x ← foreach i ∈ M do S_i ← Búsqueda2 (S_i);
    x ← foreach i ∈ M do foreach l ∈ B_i do S_i^l ← Búsqueda3 (S_i^l);
until hasta que dos procedimientos consecutivos no mejoran la solución
Result: Solución x = (S_1, S_2, ..., S_m).
```

Figura 4.6: Pseudo-código para el procedimiento de mejora: Etapa I (x).

4.3.2.3 PROCEDIMIENTO DE MEJORA: ETAPA II

Para tratar con las relaciones entre los diferentes tipos de decisión definimos dos tipos de movimiento compuestos: el primero, tipo1Compuesto, lo definimos como

el uso de un movimiento tipo 1 seguido de la Búsqueda3() aplicada a cada bloque involucrado en el movimiento; y el segundo, tipo2Compuesto, lo definimos como el uso de un movimiento tipo 2 seguido de la Búsqueda3() aplicada a cada bloque involucrado en el movimiento.

En la etapa II, primeramente se emplean movimientos tipo1Compuesto, y luego aplicamos búsqueda local basada en movimientos tipo2Compuesto. Repetimos el proceso hasta que que no sea posible mejorar la solución con ninguno de los movimientos compuestos. Definimos el procedimiento Búsqueda1Compuesta() de manera análoga a Búsqueda1() pero con la diferencia que en lugar de cada movimiento tipo1 se usa el tipo1Compuesto. De la misma manera definimos el procedimiento Búsqueda2Compuesta() de manera análoga a Búsqueda2() pero con la diferencia que en lugar de cada movimiento tipo2 se usa el tipo2Compuesto. El pseudo-código del procedimiento se muestra en la Figura 4.7.

Data: Solución $x = (S_1, S_2, \ldots, S_m)$.
1 **repeat**
2 | $x \leftarrow$ Búsqueda1Compuesta();
3 | $x \leftarrow$ **foreach** $i \in M$ **do** $S_i \leftarrow$ Búsqueda2Compuesta();
4 **until** *hasta que un procedimiento no mejore la solución*
Result: Solución $x = (S_1, S_2, \ldots, S_m)$.

Figura 4.7: Pseudo-código para el procedimiento de mejora: Etapa II (x).

4.4 EXPERIMENTACIÓN

En esta sección primero mostramos los resultados obtenidos por la segunda formulación con desigualdades válidas Model2-M y determinamos cuál es su alcance. También describimos los experimentos llevados a cabo para evaluar el desempeño del algoritmo metaheurístico propuesto.

Usamos instancias pequeñas y medianas, las cuales fueron generadas para conocer el alcance de los modelos propuestos en este capítulo.

Las instancias pequeñas las generamos considerando las siguientes combina-

ciones de números de tareas y números de máquinas: $n = \{6, 8, 10, 12\}$ y $m = \{2, 3, 4, 5\}$. Los tiempos de preparación, incluyendo los de mantenimiento, fueron uniformemente distribuidos en cuatro rangos: 1-9, 1-49, 1-99 y 1-124. Los tiempos de procesamiento, incluyendo los de mantenimiento, fueron uniformemente distribuidos entre 1-99. Generamos 10 réplicas por cada combinación, dando un total de 640 instancias medianas.

Las instancias medianas las generamos considerando las siguientes combinaciones de números de tareas y números de máquinas: $n = \{15, 20, 25, 30\}$ y $m = \{2, 4, 6, 8\}$. Los tiempos de preparación, incluyendo los de mantenimiento, fueron uniformemente distribuidos en tres rangos: 1-49, 1-99 y 1-124. Los tiempos de procesamiento, incluyendo los de mantenimiento, fueron uniformemente distribuidos entre 1-99. Generamos 5 réplicas por cada combinación, dando un total de 240 instancias medianas.

Los experimentos fueron realizados en una PC Pentium Dual Core con procesador de 2.00 GHz y 3 GB RAM, bajo el sistema opetarivo Ubuntu 12.04. La formulación fue implementada usando *Concert Technology* de CPLEX 12.2, al cual se le dio un tiempo límite de ejecución de 3600 segundos. Si el optimizador no fue capaz de alcanzar la solución óptima en dicho tiempo, entonces se reportó la mejor solución entera obtenida. Por otro lado el algoritmo multiarranque (MA) se le dio un tiempo de ejecución de $200n$ milisegundos.

Todas las instancias fueron agrupadas de acuerdo al número de tareas y número de máquinas. Por lo cual, los resultados son promediados sobre todas las instancias pertenecientes a cada grupo, es decir, 40 en grupos de instancias pequeñas y 15 en grupos de instancias medianas.

En las tablas de resultados el gap es calculado como $(C_{max} - LB)/C_{max} * 100$, donde LB es la cota inferior obtenida por Model2-M, y C_{max} es la solución factible obtenida por un algoritmo dado (Model2-M o MA), mientras que gap2 es calculado como $(C_{max} - C'_{max})/C_{max} * 100$, donde C_{max} es la solución obtenida por MA, y C'_{max}

Tabla 4.1: Comparación en instancias pequeñas.

		Model2-M			MA		
n	m	Uns	gap	tiempo	gap	tiempo_b	tiempo_l
6	2	0	0	0.49	0	0	1.21
	3	0	0	0.45	0	0	1.21
	4	0	0	0.42	0	0	1.21
	5	0	0	0.47	0.3	0	1.21
8	2	0	0	2.6	0	0.01	1.6
	3	0	0	4.07	0	0	1.6
	4	0	0	0.85	0	0	1.6
	5	0	0	0.73	0	0.02	1.6
10	2	0	0	16.05	0	0.01	2
	3	0	0	34.28	0	0.01	2
	4	0	0	4.93	0	0.01	2
	5	0	0	1.35	0	0.01	2
12	2	6	1.26	959.41	1.26	0.1	2.41
	3	11	2.38	1446.87	1.76	0.08	2.41
	4	3	1.55	357.21	1.05	0.09	2.41
	5	0	0	3.16	0	0.04	2.41

es la solución factible obtenida por Model2-M.

La comparación entre los resultados obtenidos de Model2-M y el algoritmo multiarranque (MA) en instancias pequeñas y medianas se muestran respectivamente en la Tabla 4.1 y la Tabla 4.2. En ambas tablas, las columnas 1 y 2 se refieren al tamaño de la instancia. Las columnas 3, 4 y 5 reportan el número de instancias que no fueron resueltas a optimalidad (Uns), la desviación relativa y el tiempo promedio para Model2-M, respectivamente. Las columnas 6, 7 y 8 reportan la desviación relativa (gap), el tiempo promedio para obtener la mejor solución encontrada (tiempo_b) y el tiempo límite del algoritmo (tiempo_l), respectivamente. Adicionalmente, la tabla 4.2 incluye la columna 9 la cual reporta la desviación relativa (gap2) de las soluciones obtenidas por MA respecto a las soluciones factibles obtenidas de Model2-M.

Note que el modelo logra resolver todas las instancias de 6, 8 y 10 tareas, resuelve 34 de las 40 instancias de 12 tareas con 2 máquinas, 29 de las 40 instancias de 12 tareas con 3 máquinas, 37 de las 40 instancias de 12 tareas con 4 máquinas y todas las instancias de 12 tareas con 5 máquinas. En total Model2-M logra resolver 620 de las 640 instancias pequeñas. También se observa que el algoritmo obtiene bajos

Tabla 4.2: Comparación en instancias medianas.

		Model2M			MA			
n	m	Uns	gap	tiempo	gap	tiempo_b	tiempo_l	gap2
15	2	8	12.9	2286.95	12.9	0.2	3	0
	4	3	5.59	723.11	5.59	0.25	3	0
	6	0	0	3.22	0	0.27	3	0
	8	0	0	3.93	0	0.18	3	0
20	2	15	28.48	3600.44	27.83	0.85	4.01	-1.05
	4	9	18.03	2229.55	19.23	1.75	4.01	1.16
	6	0	0	62.45	0.32	0.72	4.01	0.32
	8	0	0	87.13	0.36	0.73	4	0.36
25	2	15	30.39	3600.5	32.69	2.7	5.03	2.57
	4	11	25.35	2893.84	27.99	2.14	5.04	2.18
	6	0	0	838.52	0.46	1.36	5.03	0.46
	8	2	0.23	1261.84	1.36	2.02	5.01	1.13
30	2	15	34.54	3600.89	33.51	2.7	6.06	-1.63
	4	15	31.35	3601.65	30.67	2.89	6.07	-1
	6	7	8.09	2530.19	10.72	2.04	6.08	2.7
	8	13	8.34	3394.5	8.26	2.75	6.03	-0.26

valores de desviación relativa respecto a las cotas inferiores obtenidas por Model2-M, destacando que obtiene 638 iguales o mejores soluciones y solo dos peores soluciones que Model2-M.

Note que en la Tabla 4.2 el algoritmo propuesto muestra una baja desviación de las cotas inferiores obtenidas por el modelo en aquellos conjuntos de instancias en que Model2-M encuentra la mayoria de las soluciones óptimas. En general, para todas las combinaciones de tareas-máquinas, el algoritmo propuesto reporta gap's muy similares a los gap's reportados para Model2-M y se reporta poca desviación (gap2) de MA respecto a Model2-M. Además, en las pruebas experimentales observamos que en cada instancias no resuelta a optimalidad la convergencia de las cotas inferiores hacia la solución óptima fue muy lenta. Las observaciones anteriores sugieren que a pesar de que se tengan gap's reportados por Model2-M y el algoritmo propuesto MA de hasta 34.54 % y 33.51 %, respectivamente, las soluciones factibles obtenidas por ambos no están tan alejadas de las soluciones óptimas.

4.5 Conclusiones del capítulo

En este capítulo propusimos dos formulaciones para un problema de secuenciación en máquinas paralelas no relacionadas con tiempos de preparación dependientes de la máquina y de la secuencia, y tareas de mantenimiento preventivo. Pruebas preliminares mostraron que usando las desigualdades válidas en los modelos, se mejoran las cotas duales obtenidas por la relajacion lineal, lo cual acelera el proceso de solución del optimizador cuando se usa un algoritmo comercial de ramificacion y acotamiento. También propusimos un procedimiento de solución que hibridiza procedimientos multiarranque y de búsqueda por vecindades variables. El procedimiento de solución toma ventaja de la interdependencia entre los diferentes tipos de decisión para obtener soluciones de alta calidad. En experimentos computacionales en instancias pequeñas, el algoritmo propuesto mostró poca desviación de las cotas inferiores obtenidas de Model2-M, destacando que el modelo obtuvo 620 soluciones óptimas de las 640 instancias. En instancias medianas con el tiempo límite que se le impuso el algoritmo obtuvo soluciones de calidad similar a las soluciones factibles obtenidas por el modelo con un tiempo limite de 3600 segundos. Diversas observaciones en la tabla 4.2 y en pruebas experimentales sugieren que las soluciones obtenidas por el algoritmo no estan tan alejadas de las soluciones óptimas como parecieran indicar los altos valores de gap. Esta conjetura podria ser verificada si se contara con mejores cotas inferiores del problema.

Capítulo 5
Conclusiones generales

Es bien conocido que se deben incluir programas de mantenimiento preventivo de las máquinas en los sistemas de producción. Sin embargo, las políticas de mantenimiento influyen en la disponibilidad de las máquinas, por lo que este aspecto ha sido poco considerado en problemas de secuenciación, específicamente, en los que involucran tiempos de preparación dependientes. La consideración conjunta de tiempos de preparación dependientes y un programa de mantenimiento preventivo aumenta significativamente la complejidad de los problemas de secuenciación, por lo que es natural tratar de resolver versiones más simples del problema para ganar experiencia. Los pocos trabajos que consideran ambas características no han realizado estudios sobre los ambientes de producción de máquinas paralelas, ni sobre un criterio distinto al *makespan*.

Por lo anterior, en el trabajo de tesis abordamos dos problemas. El problema $R|S_{ijk}|C_{max}$ consiste en determinar cómo procesar un conjunto de tareas sobre un conjunto de máquinas paralelas no relacionadas con tiempos de preparación dependientes de la secuencia y de la máquina, tal que se minimice el *makespan*. El problema $R, h_{ik}|S_{ijk}|C_{max}$ es una versión extendida del primero, porque además de todo lo anterior, las máquinas tienen definido un programa de mantenimiento preventivo periódico.

En el caso del primer problema, lo reformulamos matemáticamente y propusimos una nueva linealización para la función objetivo del *makespan*. Mostramos que las formulaciones propuestas que emplean la nueva linealización tienen un desempeño sobresaliente respecto a aquellas que emplean solo la linealización clásica.

Diseñamos e implementamos un procedimiento de solución basado en técnicas metaheurísticas. Propusimos una estrategia para lidiar con la estructura de la función objetivo. Así como una estrategia para tomar ventaja de la relación entre los tipos de decisión: asignacion de tareas a máquinas y secuenciación de tareas asignadas a cada máquina. Ésta última comprende una primera etapa que emplea movimientos clásicos para este tipo de problemas y una segunda etapa, en la que se definen movimientos compuestos.

Estudiamos y formulamos matemáticamente el problema $R, h_{ik}|S_{ijk}|C_{max}$ por primera vez. Reforzamos la formulación agregando desigualdades válidas. Desarrollamos un procedimiento de solución, donde generalizamos las estrategias definidas para el primer problema, según los requerimientos del segundo problema.

Como futuras líneas de investigación estamos considerando las siguientes posibilidades:

- Estudiar el problema $R, h_{ik}|S_{ijk}|C_{max}$ cambiando la política de mantenimiento por algunas de las políticas II o III señaladas en la sección 4.1. Con dichas políticas el mantenimiento es casi periódico, no hay tiempos de ocio y el tiempo entre dos mantenimientos depende de la cantidad de tiempo que la máquina haya estado procesando tareas. En este caso el modelo matemático requiere cambios menores, sin embargo el algoritmo propuesto si podría requerir de cambios mayores ya que cambia la forma de evaluación de la funcion objetivo y por ende algunas de las funciones auxiliares o criterios de aceptación propuestos podrían cambiar.

- Explorar el beneficio del uso de funciones auxiliares para otros problemas de secuenciación con la misma función objetivo. Por ejemplo, sabemos que un caso particular del problema $R, h_{ik}|S_{ijk}|C_{max}$ es el problema $1, h_{ik}|S_{jk}|C_{max}$, el cual ya ha sido estudiado previamente en la literatura, por lo que se podría aplicar nuestro algoritmo propuesto en la sección 4.3 para dicho problema y comparar su desempeño con los algoritmos que fueron desarrollados específicamente para

ese problema.

- Explorar el diseño o definición de funciones auxiliares en otros problemas de secuenciación con otras funciones mín máx. En secuenciación también hay otras funciones mín máx como la minimización de la tardanza máxima, por lo que la idea básica a partir de la cual se diseñaron los criterios de aceptación de los capítulos 3 y 4 puede ser de utilidad en el desarrollo de algoritmos de solución a problemas con dicha función.

- Emplear los movimientos compuestos para otros problemas de secuenciación con diferentes funciones objetivo. Los movimientos compuestos se definieron para atender las relaciones existentes entre diferentes tipos de decisión que surgen en un problema dado, por lo que pueden ser utilizados incluso si cambiamos la función objetivo, por ejemplo si consideraramos el problema $R|S_{ijk}|\sum_j C_j$.

- Aplicar los movimientos compuestos en problemas de ruteo de vehículos. Dado el gran parecido que hay en estructura entre algunos problemas de secuenciación con algunos problemas de ruteo de vehículos, puede haber gran beneficio de emplear dichos movimientos en las metodologías de solución que se desarrollen para dichos problemas.

Bibliografía

Allahverdi, A. (2000), Minimizing mean flowtime in a two-machine flowshop with sequence-independent setup times. *Computers & Operations Research*, **27** (2), págs. 111–127.

Allahverdi, A. y H. Soroush (2008), The significance of reducing setup times/setup costs. *European Journal of Operational Research*, **187**(3), págs. 978–984.

Allahverdi, A., J. N. Gupta y T. Aldowaisan (1999), A review of scheduling research involving setup considerations. *Omega*, **27**(2), págs. 219–239.

Allahverdi, A., C. Ng, T. E. Cheng y M. Y. Kovalyov (2008), A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, **187**(3), págs. 985–1032.

Ángel-Bello, F., A. Álvarez, J. Pacheco y I. Martínez (2011a), A single machine scheduling problem with availability constraints and sequence-dependent setup costs. *Applied Mathematical Modelling*, **35**(4), págs. 2041–2050.

Ángel-Bello, F., A. Álvarez, J. Pacheco y I. Martínez (2011b), A heuristic approach for a scheduling problem with periodic maintenance and sequence-dependent setup times. *Computers & Mathematics with Applications*, **61**(4), págs. 797–808.

Anghinolfi, D. y M. Paolucci (2008), A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. *International Journal of Operations Research*, **5**(1), págs. 1–17.

ANGHINOLFI, D. y M. PAOLUCCI (2009), A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, **193**(1), págs. 73–85.

ARNAOUT, J., G. RABADI y R. MUSA (2010), A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, **21**(6), págs. 693–701.

BHATIA, H., K. SWARUP y M. PURI (1977), A procedure for time minimization transportation problem. *Indian Journal of Pure and Applied Mathematics*, **8**(8), págs. 920–929.

BLISCHKE, W. R. y D. P. MURTHY (2011). *Reliability: modeling, prediction, and optimization*, volumen 767. John Wiley & Sons.

BREIT, J. (2007), Improved approximation for non-preemptive single machine flow-time scheduling with an availability constraint. *European Journal of Operational Research*, **183**(2), págs. 516–524.

BRUCKER, P. (2007). *Scheduling algorithms*. Springer.

BURKARD, R. E., V. G. DEINEKO, R. VAN DAL, J. A. VAN DER VEEN y G. J. WOEGINGER (1998), Well-solvable special cases of the traveling salesman problem: a survey. *SIAM review*, **40**(3), págs. 496–546.

CHEN, J.-S. (2008), Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. *European Journal of Operational Research*, **190**(1), págs. 90–102.

CHEN, W. (2009), Scheduling with dependent setups and maintenance in a textile company. *Computers & Industrial Engineering*, **57**(3), págs. 867–873.

CHENG, T., Q. DING y B. LIN (2004), A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, **152**(1), págs. 1–13.

CHOOBINEH, F. F., E. MOHEBBI y H. KHOO (2006), A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, **175**(1), págs. 318–337.

DE, P. y T. MORTON (1980), Scheduling to minimize makespan on unequal parallel processors. *Decision Sciences*, **11**(4), págs. 586–602.

FLESZAR, K., C. CHARALAMBOUS y K. HINDI (2012), A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, **23**(5), págs. 1949–1958.

GAREY, R. y D. JOHNSON (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman. URL `http://books.google.com.mx/books?id=fjxGAQAAIAAJ`.

GRAHAM, R. L., E. L. LAWLER, J. K. LENSTRA y A. RINNOOY KAN (1977), Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics. v5*, págs. 287–326.

GUPTA, S. R. y J. S. SMITH (2006), Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, **175**(2), págs. 722–739.

HANDLARSKI, J. (1980), Mathematical analysis of preventive maintenance schemes. *Journal of the Operational Research Society*, págs. 227–237.

HANSEN, P., N. MLADENOVIĆ y J. A. M. PÉREZ (2010), Variable neighbourhood search: methods and applications. *Annals of Operations Research*, **175**(1), págs. 367–407.

HELAL, M., G. RABADI y A. AL-SALEM (2006), A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, **3**(3), págs. 182–192.

HWANG, H.-C. y S. Y. CHANG (1998), Parallel machines scheduling with machine shutdowns. *Computers & Mathematics with Applications*, **36**(3), págs. 21–31.

HWANG, H.-C., K. LEE y S. Y. CHANG (2005), The effect of machine availability on the worst-case performance of lpt. *Discrete Applied Mathematics*, **148**(1), págs. 49–61.

JI, M., Y. HE y T. E. CHENG (2007), Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & operations research*, **34**(6), págs. 1764–1770.

KIRLIK, G. y C. OGUZ (2012), A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. *Computers & Operations Research*, **39**(7), págs. 1506–1520.

KUO, W.-H. y D.-L. YANG (2007), Single machine scheduling with past-sequence-dependent setup times and learning effects. *Information Processing Letters*, **102** (1), págs. 22–26.

LAWLER, E. L., J. K. LENSTRA, A. H. RINNOOY KAN y D. B. SHMOYS (1993), Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, **4**, págs. 445–522.

LEE, C.-Y. (1996), Machine scheduling with an availability constraint. *Journal of global optimization*, **9**(3-4), págs. 395–416.

LEE, C.-Y. y S. D. LIMAN (1992), Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, **29**(4), págs. 375–382.

LEE, C.-Y. y C.-S. LIN (2001), Single-machine scheduling with maintenance and repair rate-modifying activities. *European Journal of Operational Research*, **135** (3), págs. 493–513.

LEE, W.-C. y C.-C. WU (2009), A note on single-machine group scheduling problems with position-based learning effect. *Applied Mathematical Modelling*, **33**(4), págs. 2159–2163.

LEE, W.-C. y C.-C. WU (2010), A note on optimal policies for two group scheduling problems with deteriorating setup and processing times. *Computers Industrial Engineering*, **58**(4), págs. 646–650.

LEE, Y. y M. PINEDO (1997), Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, **100**(3), págs. 464–474.

LIAO, C.-J. y H.-C. JUAN (2007), An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research*, **34**(7), págs. 1899–1909.

LIAO, C.-J., D.-L. SHYUR y C.-H. LIN (2005), Makespan minimization for two parallel machines with an availability constraint. *European Journal of Operational Research*, **160**(2), págs. 445–456.

LIN, C.-H. y C.-J. LIAO (2007), Makespan minimization for two parallel machines with an unavailable period on each machine. *The International Journal of Advanced Manufacturing Technology*, **33**(9-10), págs. 1024–1030.

LIN, S.-W. y K.-C. YING (2007), Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. *The International Journal of Advanced Manufacturing Technology*, **34**(11-12), págs. 1183–1190.

LOW, C., C.-J. HSU y C.-T. SU (2008), Minimizing the makespan with an availability constraint on a single machine under simple linear deterioration. *Computers & Mathematics with Applications*, **56**(1), págs. 257–265.

LUO, J. y Y. HU. A new grasp and path relinking for single machine scheduling

with sequence dependent setups. En *Control and Automation (ICCA), 2013 10th IEEE International Conference on*, págs. 490–495. IEEE (2013).

MA, Y., C. CHU y C. ZUO (2010), A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, **58**(2), págs. 199–211.

MARTÍ, R., J. M. MORENO-VEGA y A. DUARTE. Advanced multi-start methods. En *Handbook of Metaheuristics*, págs. 265–281. Springer (2010).

MCCALL, J. J. (1965), Maintenance policies for stochastically failing equipment: a survey. *Management science*, **11**(5), págs. 493–524.

MOKOTOFF, E. (2001), Parallel machine scheduling problems: a survey. *Asia-Pacific Journal of Operational Research*, **18**(2), págs. 193–242.

MONKMAN, S. K., D. J. MORRICE y J. F. BARD (2008), A production scheduling heuristic for an electronics manufacturer with sequence-dependent setup costs. *European journal of operational research*, **187**(3), págs. 1100–1114.

MORTON, T. (1993). *Heuristic scheduling systems: with applications to production systems and project management*, volumen 3. Wiley. com.

NADERI, B., M. ZANDIEH y S. F. GHOMI (2009a), Scheduling sequence-dependent setup time job shops with preventive maintenance. *The International Journal of Advanced Manufacturing Technology*, **43**(1-2), págs. 170–181.

NADERI, B., M. ZANDIEH y S. F. GHOMI (2009b), A study on integrating sequence dependent setup time flexible flow lines and preventive maintenance scheduling. *Journal of intelligent manufacturing*, **20**(6), págs. 683–694.

OR, I. (1976). *Traveling Salesman-type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking*. Northwestern University.

PACHECO, J., F. ÁNGEL-BELLO y A. ÁLVAREZ (2012), A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. *Journal of Scheduling*, págs. 1–13.

PIERSKALLA, W. P. y J. A. VOELKER (1976), A survey of maintenance models: the control and surveillance of deteriorating systems. *Naval Research Logistics Quarterly*, **23**(3), págs. 353–388.

PINEDO, M. L. (2012). *Scheduling: theory, algorithms, and systems*. Springer.

QI, X., T. CHEN y F. TU (1999), Scheduling the maintenance on a single machine. *Journal of the Operational Research Society*, págs. 1071–1078.

RABADI, G., R. MORAGA y A. AL-SALEM (2006), Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, **17**(1), págs. 85–97.

ROCHA, P., M. RAVETTI, G. MATEUS y P. PARDALOS (2008), Exact algorithms for a acheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, **35**(4), págs. 1250–1264.

RUIZ, R., J. CARLOS GARCÍA-DÍAZ y C. MAROTO (2007), Considering scheduling and preventive maintenance in the flowshop sequencing problem. *Computers & Operations Research*, **34**(11), págs. 3314–3330.

SURESH, V. y D. GHAUDHURI (1996), Scheduling of unrelated parallel machines when machine availability is specified. *Production Planning & Control*, **7**(4), págs. 393–400.

TASGETIREN, M. F., Q.-K. PAN y Y.-C. LIANG (2009), A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. *Computers & Operations Research*, **36**(6), págs. 1900–1915.

TIAN, K., Y. JIANG, X. SHEN y W. MAO. Makespan minimization for job co-scheduling on chip multiprocessors. Reporte técnico WM-CS-2010-08, Department of Computer Science, College of William & Mary (2010).

TRAN, T. y J. BECK. Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. En *20th European Conference on Artificial Intelligence*, págs. 774–780 (2012).

VALLADA, E. y R. RUIZ (2011), A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, **211**, págs. 612–622.

WEI, C.-M. y J.-B. WANG (2010), Single machine quadratic penalty function scheduling with deteriorating jobs and group technology. *Applied Mathematical Modelling*, **34**(11), págs. 3642–3647.

YANG, W.-H. y S. CHAND (2008), Learning and forgetting effects on a group scheduling problem. *European Journal of Operational Research*, **187**(3), págs. 1033–1044.

YING, K., Z. LEE y S. LIN (2012), Makespan minimization for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, **23**(5), págs. 1795–1803.

Ficha autobiográfica

MC. Oliver Avalos Rosales

Candidato para el grado de Doctor en Ingeniería
con especialidad en Ingeniería de Sistemas

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Tesis:

Secuenciación en máquinas paralelas no relacionadas con tiempos de preparación y tareas de mantenimiento preventivo

Nací el treinta de septiembre de 1984, en la ciudad de Morelia, Michoacán, siendo el segundo hijo de Reginaldo Avalos y Emilia Rosales. Asistí a la escuela primaria "Nicolás Bravo" y la secundaria federal "15 de Mayo", ambas en mi localidad. En 1999 ingresé a la preparatoria "Isaac Arriaga" en la cual despertó mi interés de ingresar a la Facultad de Ciencias Físico-Matemáticas, donde estudié la licenciatura. Durante la licenciatura mi vida se tornó hacia el camino de la aplicación de las matemáticas y la programación, por lo que ingresé en el 2009 a estudiar la maestría en el Posgrado en Ingeniería de Sistemas (PISIS) en la Universidad Autónoma de Nuevo León (UANL), la cual realicé bajo la asesoría y dirección de la Dra. Elisa Schaeffer. Al culminar los estudios de maestria decidí realizar el doctorado en colaboración con la Dra. Ada Álvarez y el Dr. Francisco Angel-Bello con quienes he desarrollado el proyecto de tesis que se presenta en este escrito.